

SELENIUM



LEARN IN 1 DAY

Learn Selenium in 1 Day

By Krishna Rungta

GURU⁹⁹

Copyright 2018 - All Rights Reserved – Krishna Rungta

ALL RIGHTS RESERVED. No part of this publication may be reproduced or transmitted in any form whatsoever, electronic, or mechanical, including photocopying, recording, or by any informational storage or retrieval system without express written, dated and signed permission from the author.

Table Of Content

[**Chapter 1: Introduction to Selenium**](#)

[**Chapter 2: Install Selenium IDE and FireBug**](#)

[**Chapter 3: Introduction to Selenium IDE**](#)

[**Chapter 4: Creating your First Selenium IDE script**](#)

[**Chapter 5: How to use Locators in Selenium IDE**](#)

[**Chapter 6: How to enhance a script using Selenium IDE**](#)

[**Chapter 7: Store Variables, Echo, Alert, PopUp handling in Selenium IDE**](#)

[**Chapter 8: Introduction to WebDriver & Comparison with Selenium RC**](#)

[**Chapter 9: Guide to install Selenium WebDriver**](#)

[**Chapter 10: Creating your First Script in Webdriver**](#)

[**Chapter 11: Accessing Forms in Webdriver**](#)

[**Chapter 12: How to Select Option from DropDown using Selenium Webdriver**](#)

[Chapter 13: Accessing Links & Tables using Selenium Webdriver](#)

[Chapter 14: Keyboard & Mouse Event using Action Class in Selenium Webdriver](#)

[Chapter 15: How to Upload & Download a File using Selenium Webdriver](#)

[Chapter 16: XPath in Selenium: Complete Guide](#)

[Chapter 17: How TestNG makes Selenium tests easier](#)

[Chapter 18: Handling Date Time Picker using Selenium](#)

[Chapter 19: Alert & Popup handling in Selenium](#)

[Chapter 20: Handling Dynamic Web Tables Using Selenium WebDriver](#)

[Chapter 21: Using Contains, Sibling, Ancestor to Find Element in Selenium](#)

[Chapter 22: Implicit & Explicit Waits in Selenium](#)

[Chapter 23: Parameterization using XML and DataProviders: Selenium](#)

[Chapter 24: All About Excel in Selenium: POI & JXL](#)

[Chapter 25: Page Object Model \(POM\) & Page Factory in](#)

Selenium: Ultimate Guide

Chapter 26: Introduction to Selenium Grid

Chapter 27: Maven & Jenkins with Selenium: Complete Tutorial

Chapter 28: Creating Keyword & Hybrid Frameworks with Selenium

Chapter 29: Database Testing using Selenium: Step by Step Guide

Chapter 30: Handling Iframes in Selenium

Chapter 31: Cross Browser Testing using Selenium

Chapter 32: PDF , Emails and Screenshot of Test Reports in Selenium

Chapter 33: How to Take Screenshot in Selenium WebDriver

Chapter 34: Sessions, Parallel run and Dependency in Selenium

Chapter 35: Tutorial on Log4j and LogExpert with Selenium

Chapter 36: Selenium with HTMLUnit Driver & PhantomJS

Chapter 37: Using Robot API with Selenium

Chapter 38: How to use AutoIT with Selenium

Chapter 39: Desired Capabilities in Selenium

Chapter 40: SSL Certificate Error Handling in Selenium

Chapter 41: Handling Ajax call in Selenium Webdriver

Chapter 42: Listeners and their use in Selenium WebDriver

Chapter 43: Execute JavaScript based code using Selenium Webdriver

Chapter 44: Using Selenium with Python

Chapter 45: How to use intelliJ & Selenium Webdriver

Chapter 46: Test Case Priority in TestNG

Chapter 47: TestNG: Execute multiple test suites

Chapter 48: Introduction to TestNG Groups

Chapter 49: Verify Tooltip Using Selenium WebDriver

Chapter 50: Flash Testing with Selenium

Chapter 51: How to Find Broken links using Selenium Webdriver

Chapter 52: Selenium Core Extensions

Chapter 53: Using Apache Ant with Selenium

Chapter 54: Using Selenium with Github

Chapter 55: Handling Cookies in Selenium WebDriver

Chapter 56: Using SoapUI with Selenium

Chapter 57: XSLT Report in Selenium

Chapter 58: Firefox Profile - Selenium WebDriver

Chapter 59: Breakpoints and Startpoints in Selenium

Chapter 60: Top 100 Selenium Interview Questions & Answers

Chapter 61: Using Cucumber with Selenium

Chapter 62: Drag and Drop action in Selenium

Chapter 63: Selenium C# Webdriver Tutorial for Beginners

Chapter 64: Creating Object Repository in Selenium WebDriver

Chapter 65: Scroll UP or Down a page in Selenium Webdriver

Chapter 66: File Upload using Sikuli in Selenium Webdriver

Chapter 67: Gecko (Marionette) Driver Selenium: Download, Install, Use with Firefox

Chapter 68: Find Element and Find Elements in Selenium

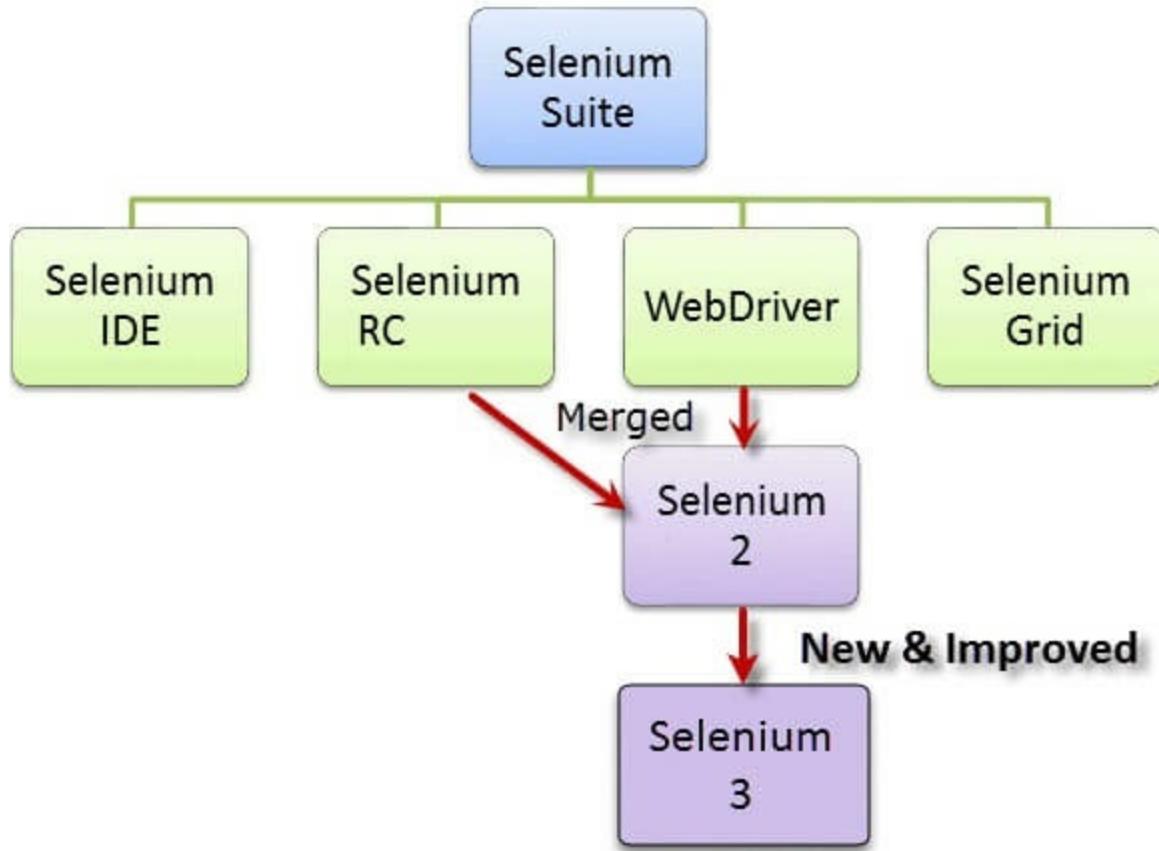
Chapter 1: Introduction to Selenium

What is Selenium?

Selenium is a free (open source) automated testing suite for web applications across different browsers and platforms. It is quite similar to HP Quick Test Pro (QTP now UFT) only that Selenium focuses on automating web-based applications. Testing done using Selenium tool is usually referred as Selenium Testing.

Selenium is not just a single tool but a suite of software's, each catering to different testing needs of an organization. **It has four components.**

- Selenium Integrated Development Environment (IDE)
- Selenium Remote Control (RC)
- WebDriver
- Selenium Grid



At the moment, Selenium RC and WebDriver are merged into a single framework to form **Selenium 2**. Selenium 1, by the way, refers to Selenium RC.

Who developed Selenium?

Since Selenium is a collection of different tools, it had different developers as well. Below are the key persons who made notable contributions to the Selenium Project

Primarily, Selenium was **created by Jason Huggins in 2004**. An engineer at ThoughtWorks, he was working on a web application that required frequent testing. Having

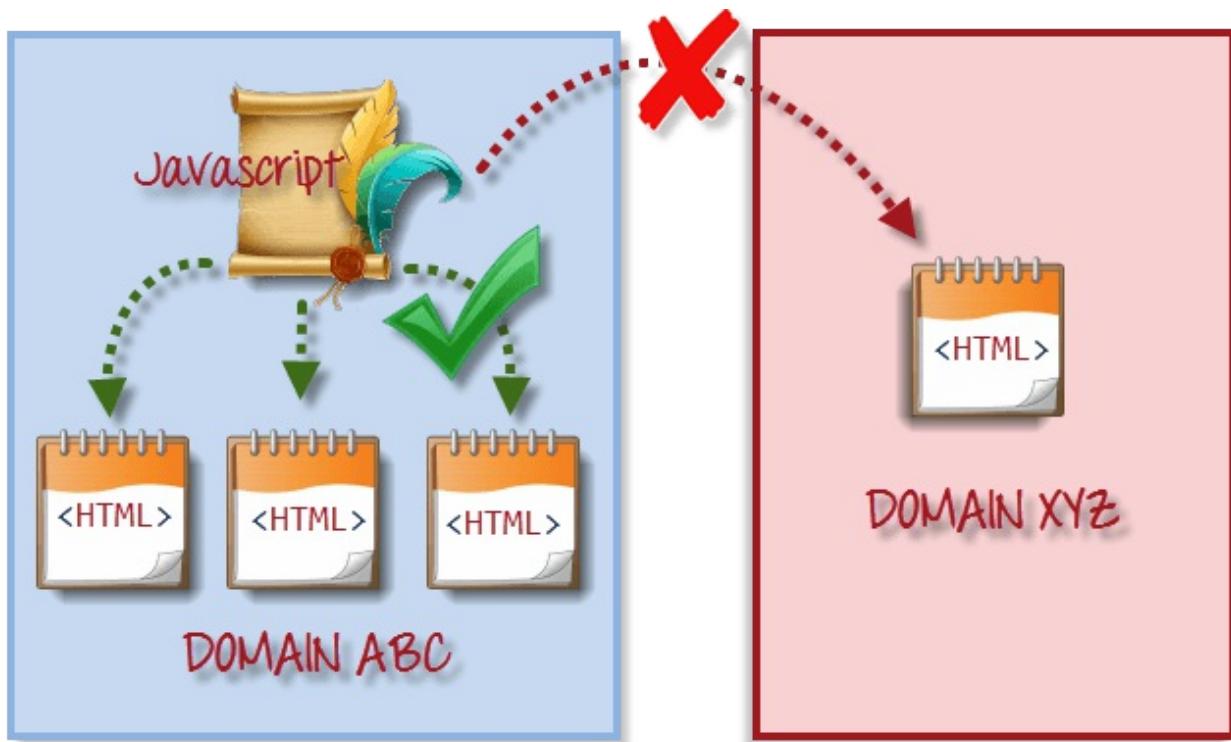


realized that the repetitive Manual Testing of their application was becoming more and more inefficient, he created a JavaScript program that would automatically control the browser's actions. He named this program as the "**JavaScriptTestRunner**".

Seeing potential in this idea to help automate other web applications, he made JavaScriptRunner open-source which was later re-named as **Selenium Core**.

The Same Origin Policy Issue

Same Origin policy prohibits JavaScript code from accessing elements from a domain that is different from where it was launched. Example, the HTML code in www.google.com uses a JavaScript program "randomScript.js". The same origin policy will only allow randomScript.js to access pages within google.com such as google.com/mail, google.com/login, or google.com/signup. However, it cannot access pages from different sites such as yahoo.com/search or guru99.com because they belong to different domains.



under same origin policy, a Javascript program can only access pages on the same domain where it belongs. It cannot access pages from different domains

This is the reason why prior to Selenium RC, testers needed to install local copies of both Selenium Core (a JavaScript program) and the web server containing the web application being tested so they would belong to the same domain

Birth of Selenium Remote Control (Selenium RC)



Paul Hammant

Unfortunately; testers using Selenium Core had to install the whole application under test and the web server on their own local computers because of the restrictions imposed by the **same origin policy**. So another ThoughtWork's engineer, **Paul Hammant**, decided to

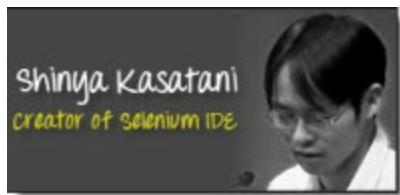
create a server that will act as an HTTP proxy to "trick" the browser into believing that Selenium Core and the web application being tested come from the same domain. This system became known as the **Selenium Remote Control** or **Selenium 1**.

Birth of Selenium Grid



Selenium Grid was developed by **Patrick Lightbody** to address the need of minimizing test execution times as much as possible. He initially called the system "**Hosted QA**." It was capable of capturing browser screenshots during significant stages, and also of **sending out Selenium commands to different machines simultaneously**.

Birth of Selenium IDE



Shinya Kasatani of Japan created **Selenium IDE**, a Firefox extension that can automate the browser through a record-and-playback feature. He came up with this idea to further increase the speed in creating test cases. He donated Selenium IDE to the Selenium Project in **2006**.

Birth of WebDriver



Simon Stewart created WebDriver circa **2006** when browsers and web applications were becoming more powerful and more restrictive with JavaScript programs like Selenium Core. **It was the first cross-platform testing framework that could control the browser from the OS level.**

Birth of Selenium 2

In **2008**, the whole Selenium Team decided to merge WebDriver and Selenium RC to form a more powerful tool called **Selenium 2**, with **WebDriver being the core**. Currently, Selenium RC is still being developed but only in maintenance mode. Most of the Selenium Project's efforts are now focused on Selenium 2.

So, Why the Name Selenium?

It came from a joke which Jason cracked one time to his team. Another automated testing framework was popular during Selenium's development, and it was by the company called **Mercury Interactive** (yes, the company who originally made QTP before it was acquired by HP). Since Selenium is a well-known antidote for Mercury

poisoning, Jason suggested that name. His teammates took it, and so that is how we got to call this framework up to the present.



Brief Introduction Selenium IDE

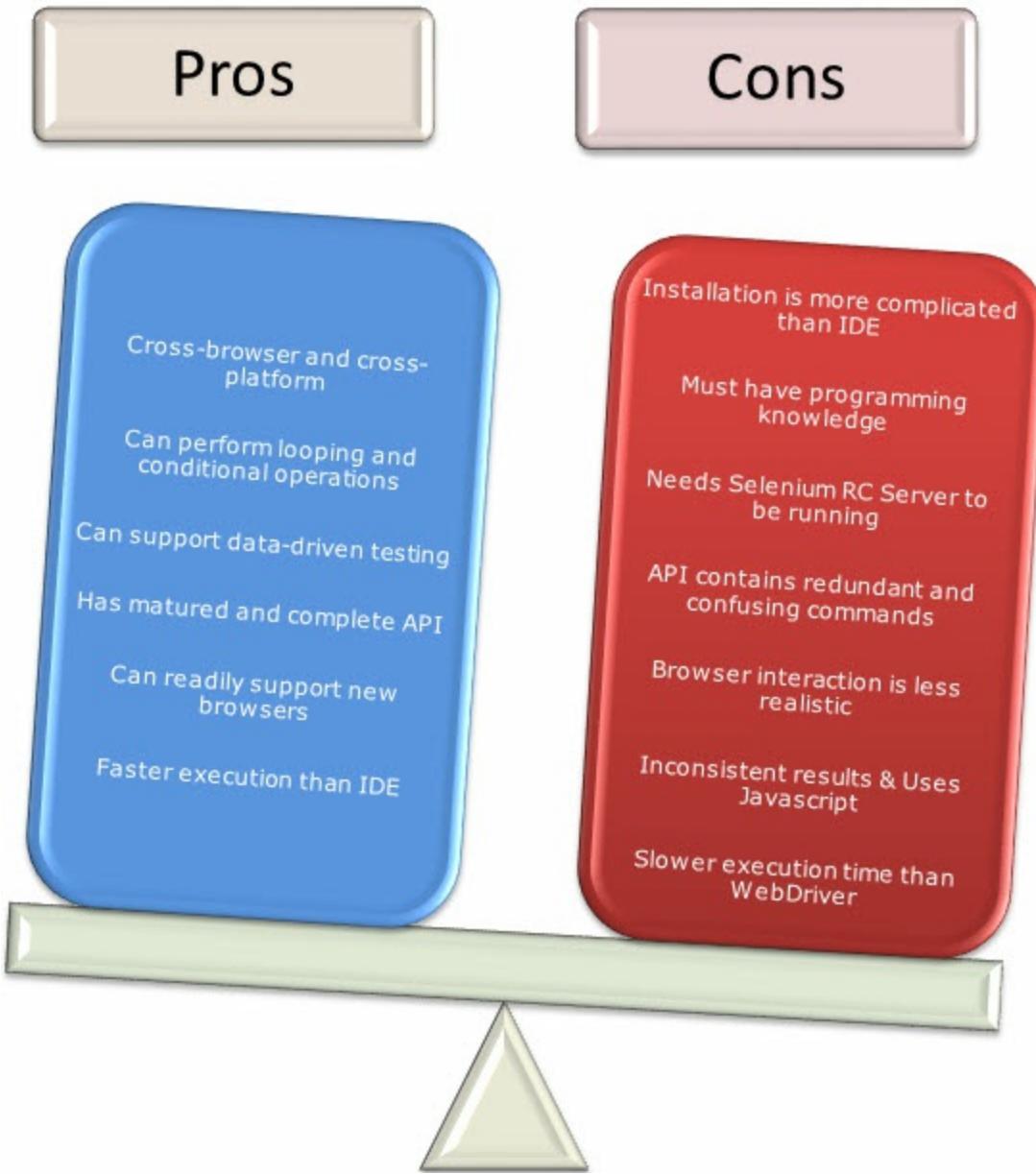
Selenium Integrated Development Environment (IDE) is the **simplest framework** in the Selenium suite and is **the easiest one to learn**. It is a **Firefox plugin** that you can install as easily as you can with other plugins. However, because of its simplicity, Selenium IDE should only be used as a **prototyping tool**. If you want to create more advanced test cases, you will need to use either Selenium RC or WebDriver.

<u>PROS</u>	<u>CONS</u>
<ul style="list-style-type: none">Very easy to use and install.No programming experience is required, though knowledge of HTML and DOM are needed.Can export tests to formats usable in Selenium RC and WebDriver.Has built-in help and test results reporting module.Provides support for extensions.	<ul style="list-style-type: none">Available only in Firefox.Designed only to create prototypes of tests.No support for iteration and conditional operations.Test execution is slow compared to that of Selenium RC and WebDriver.

Brief Introduction Selenium Remote Control (Selenium RC)

Selenium RC was the **flagship testing framework** of the whole Selenium project for a long time. This is the first automated web testing tool that **allowed users to use a programming language they prefer**. As of version 2.25.0, RC can support the following programming languages:

- Java
- C#
- PHP
- Python
- Perl
- Ruby



Brief Introduction WebDriver

The WebDriver proves itself to be **better than both Selenium IDE and Selenium RC** in many aspects. It implements a more modern and stable approach in automating the browser's actions. WebDriver, unlike Selenium RC, does not rely on JavaScript for Automation. **It controls the browser by directly communicating with it.**

The supported languages are the same as those in Selenium RC.

- Java
- C#
- PHP
- Python
- Perl
- Ruby

Pros	Cons
<p>Simpler installation than Selenium RC</p> <p>Communicates directly to the browser</p> <p>Browser interaction is more realistic</p> <p>No need for a separate component such as the RC Server</p> <p>Faster execution time than IDE and RC</p>	<p>Installation is more complicated than Selenium IDE</p> <p>Requires programming knowledge</p> <p>Cannot readily support new browsers</p> <p>Has no built-in mechanism for logging runtime messages and generating test results</p>

Selenium Grid

Selenium Grid is a tool **used together with Selenium RC to run parallel tests** across different machines and different browsers all at the same time. Parallel execution means running multiple tests at once.

Features:

- Enables **simultaneous running of tests in multiple browsers and environments.**
- **Saves time** enormously.
- Utilizes the **hub-and-nodes** concept. The hub acts as a central source of Selenium commands to each node connected to it.

Note on Browser and Environment Support

Because of their architectural differences, Selenium IDE, Selenium RC, and WebDriver support different sets of browsers and operating environments.

	Selenium IDE	WebDriver
Browser Support	Mozilla Firefox	Internet Explorer versions 6 to 11, both 32 and 64-bit Microsoft Edge version 12.10240 & above (partial support some functionalities under development) Firefox 3.0 and above Google Chrome 12.0. and above Opera 11.5 and above Android - 2.3 and above for phones and tablets (devices & emulators) iOS 3+ for phones (devices & emulators) and 3.2+ for tablets (devices & emulators) HtmlUnit 2.9 and above
Operating System	Windows, Mac OS X, Linux	All operating systems where the browsers above can run.

Note: Selenium WebDriver is termed as the successor of Selenium RC which has been deprecated & officially announced by SeleniumHQ.

How to Choose the Right Selenium Tool for Your Need

Tool	Why Choose?
Selenium IDE	<ul style="list-style-type: none"> • To learn about concepts on automated testing and Selenium, including: • Selenese commands such as type, open, clickAndWait, assert, verify, etc. • Locators such as id, name, xpath, css selector, etc. • Executing customized JavaScript code using runScript • Exporting test cases in various formats. • To create tests with little or no prior knowledge in programming. • To create simple test cases and test suites that you can export later to RC or WebDriver. • To test a web application against Firefox only.
Selenium RC	<ul style="list-style-type: none"> • To design a test using a more expressive language than Selenese • To run your test against different browsers (except HtmlUnit) on different operating systems. • To deploy your tests across multiple environments using Selenium Grid. • To test your application against a new browser that supports JavaScript. • To test web applications with complex AJAX-based scenarios.
WebDriver	<ul style="list-style-type: none"> • To use a certain programming language in designing your test case. • To test applications that are rich in AJAX-based functionalities. • To execute tests on the HtmlUnit browser. • To create customized test results.
Selenium Grid	<ul style="list-style-type: none"> • To run your Selenium RC scripts in multiple browsers and operating systems simultaneously. • To run a huge test suite, that needs to complete in the soonest

time possible.

A Comparison between Selenium and QTP(now UFT)

Quick Test Professional(QTP) is a proprietary automated testing tool previously owned by the company **Mercury Interactive** before it was **acquired by Hewlett-Packard in 2006**. The Selenium Tool Suite has many advantages over QTP as detailed below -

Advantages of Selenium over QTP

Selenium	QTP
Open source, free to use, and free of Commercial charge.	
Highly extensible	Limited add-ons
Can run tests across different browsers	Can only run tests in Firefox, Internet Explorer and Chrome
Supports various operating systems	Can only be used in Windows
Supports mobile devices	QTP Supports Mobile app test automation (iOS & Android) using HP solution called - HP Mobile Center
Can execute tests while the browser is minimized	Needs to have the application under test to be visible on the desktop
Can execute tests in parallel .	Can only execute in parallel but using Quality Center which is again a paid product.

Advantages of QTP over Selenium

QTP	Selenium
Can test both web and desktop applications	Can only test web applications
Comes with a built-in object repository	Has no built-in object repository
Automates faster than Selenium because it is a fully featured IDE.	Automates at a slower rate because it does not have a native IDE and only third party IDE can be used for development
Data-driven testing is easier to perform because it has built-in global and local data tables .	Data-driven testing is more cumbersome since you have to rely on the programming language's capabilities for setting values for your test data
Can access controls within the browser (such as the Favorites bar, Address bar, Back and Forward buttons, etc.)	Cannot access elements outside of the web application under test
Provides professional customer support	No official user support is being offered.
Has native capability to export test data into external formats	Has no native capability to export runtime data onto external formats
Parameterization Support is built	Parameterization can be done via programming but is difficult to implement.
Test Reports are generated automatically	No native support to generate test /bug reports.

Though clearly, QTP has more advanced capabilities, Selenium

outweighs QTP in three main areas:

- **Cost**(because Selenium is completely free)
- **Flexibility**(because of a number of programming languages, browsers, and platforms it can support)
- **Parallel testing**(something that QTP is capable of but only with use of Quality Center)

Summary

- The entire Selenium Tool Suite is comprised of four components:
 - **Selenium IDE**, a Firefox add-on that you can only use in creating relatively simple test cases and test suites.
 - **Selenium Remote Control**, also known as **Selenium 1**, which is the first Selenium tool that allowed users to use programming languages in creating complex tests.
 - **WebDriver**, the newer breakthrough that allows your test scripts to communicate directly to the browser, thereby controlling it from the OS level.
 - **Selenium Grid** is also a tool that is used with Selenium RC to execute parallel tests across different browsers and operating systems.
- Selenium RC and WebDriver was merged to form **Selenium 2**.
- Selenium is more advantageous than QTP in terms of **costs and flexibility**. It also allows you to **run tests in parallel**, unlike in QTP where you are only allowed to run tests sequentially.

Chapter 2: Install Selenium IDE and FireBug

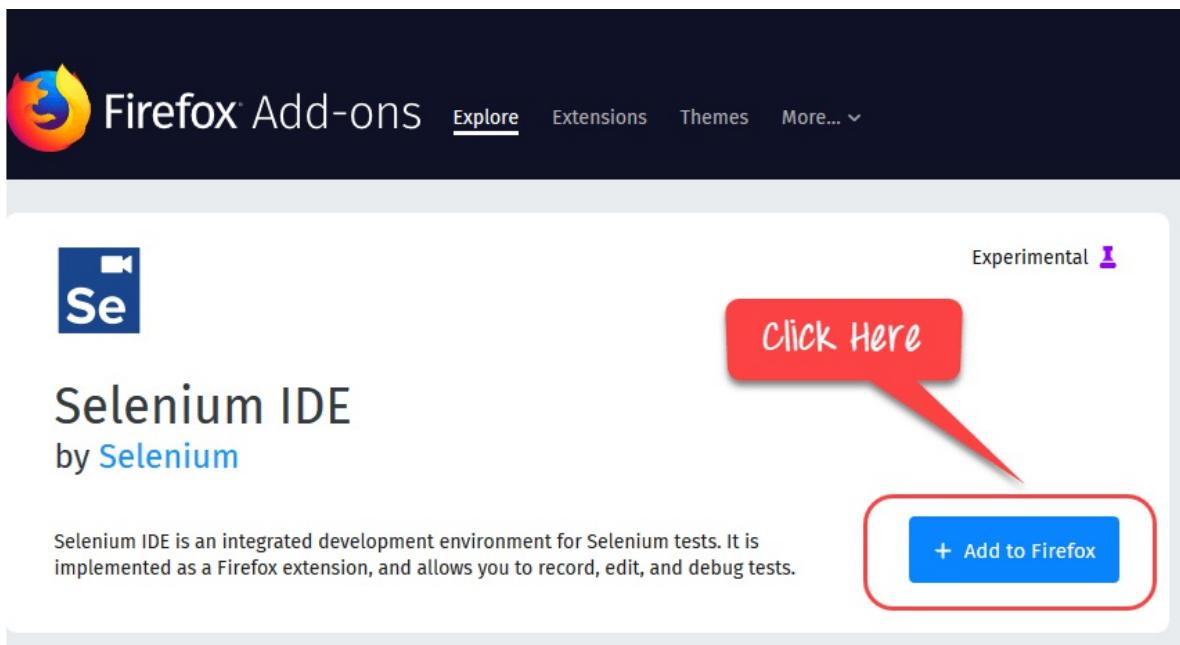
Installation of Selenium IDE

What you need

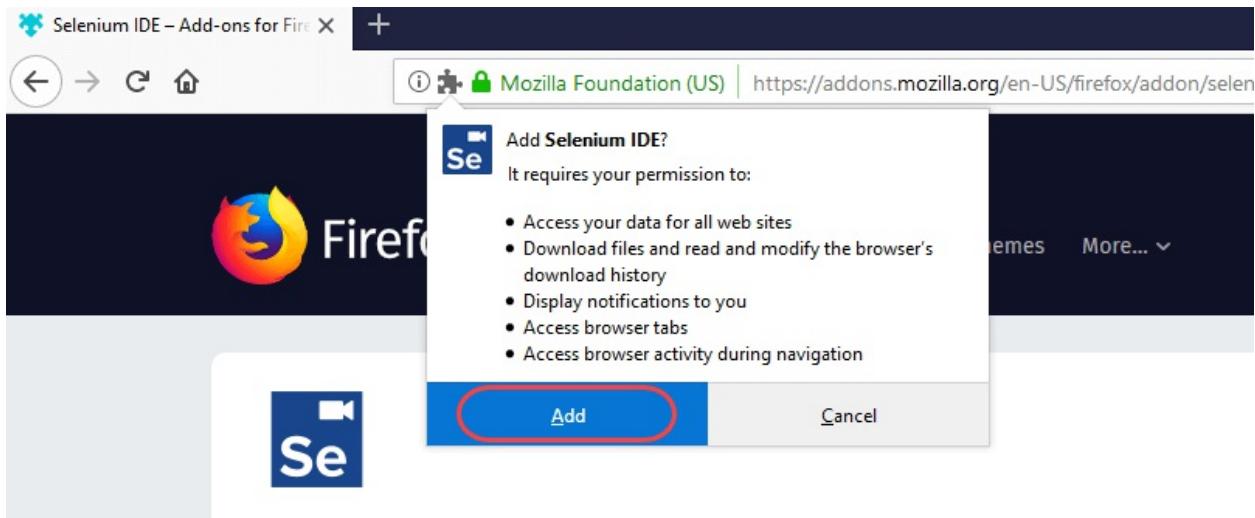
- Mozilla Firefox
- Active Internet Connection

If you do not have Mozilla Firefox yet, you can download it from <http://www.mozilla.org/en-US/firefox/new>.

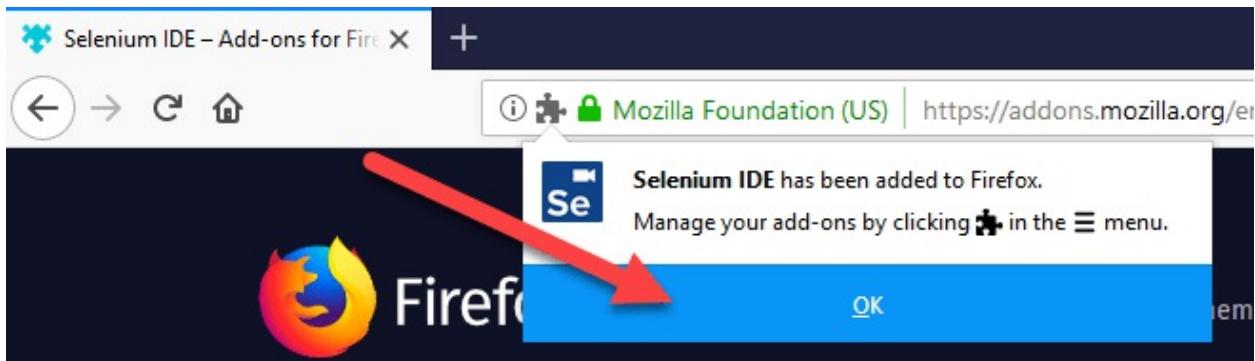
Steps 1) Launch Firefox and navigate to <https://addons.mozilla.org/en-US/firefox/addon/selenium-ide/>. Click on Add to Firefox



Steps 2) Wait until Firefox completes the download and then click "Add."



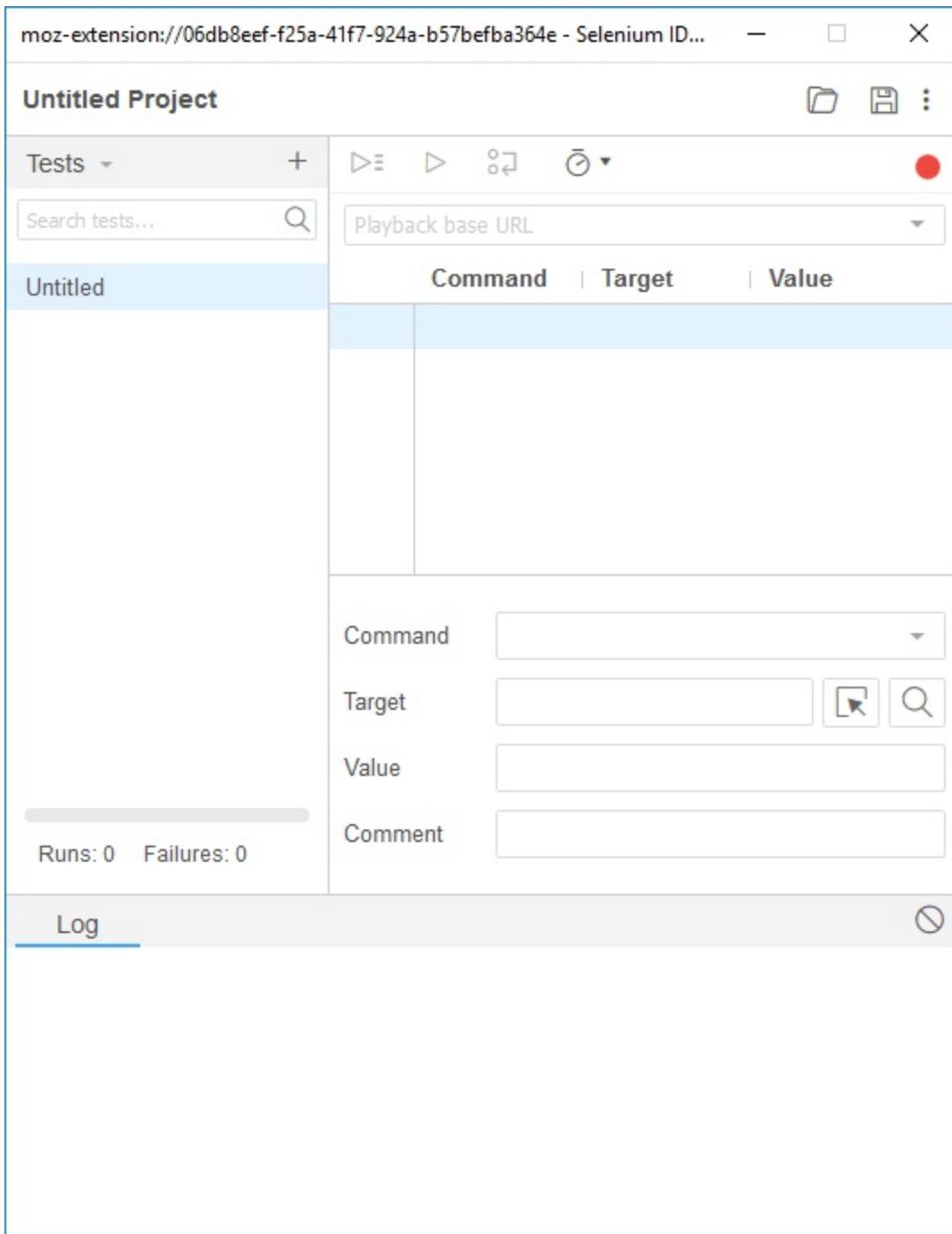
Steps 3) Once install is complete, you will get a confirmation message. Click "OK"



Steps 4) Click on the Selenium IDE icon



Selenium IDE will open

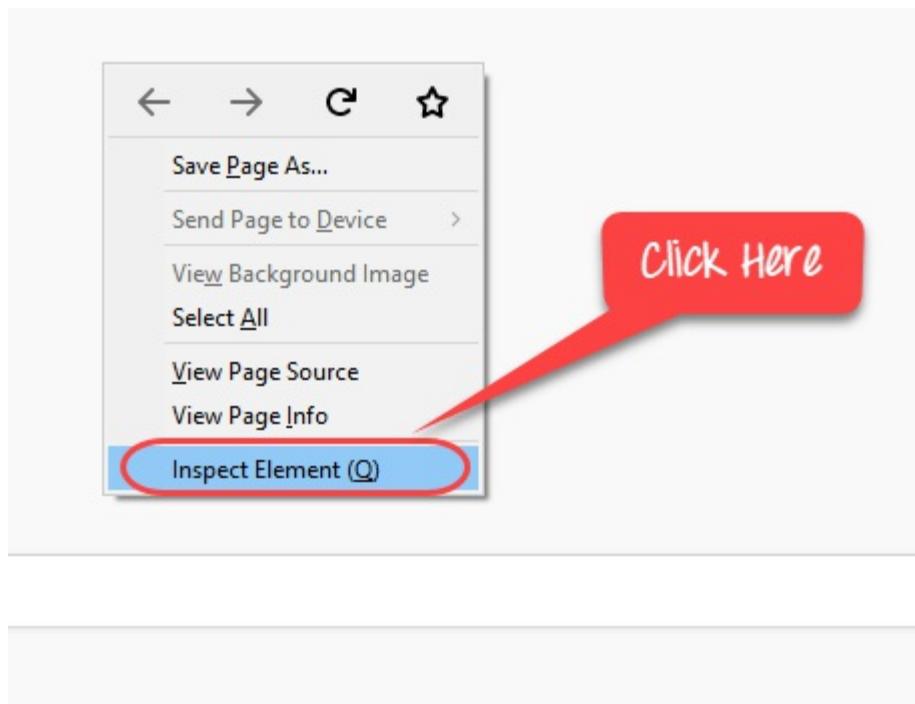


Firefox DevTools in Firefox

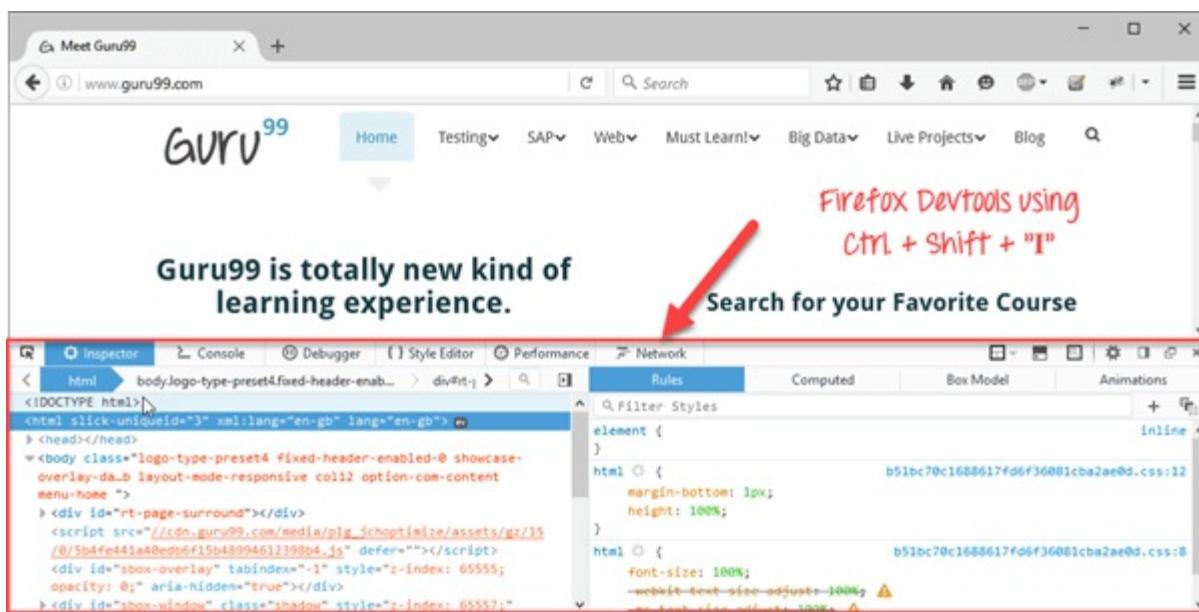
Firefox DevTools is a Firefox feature that we will use to **inspect the HTML elements** of the web application under test. It will provide us

the name of the element that our Selenese command would act upon.

Step 1) Right click anywhere on the page and select Inspect Element. You can also use shortcut Cntrl + Shift + I



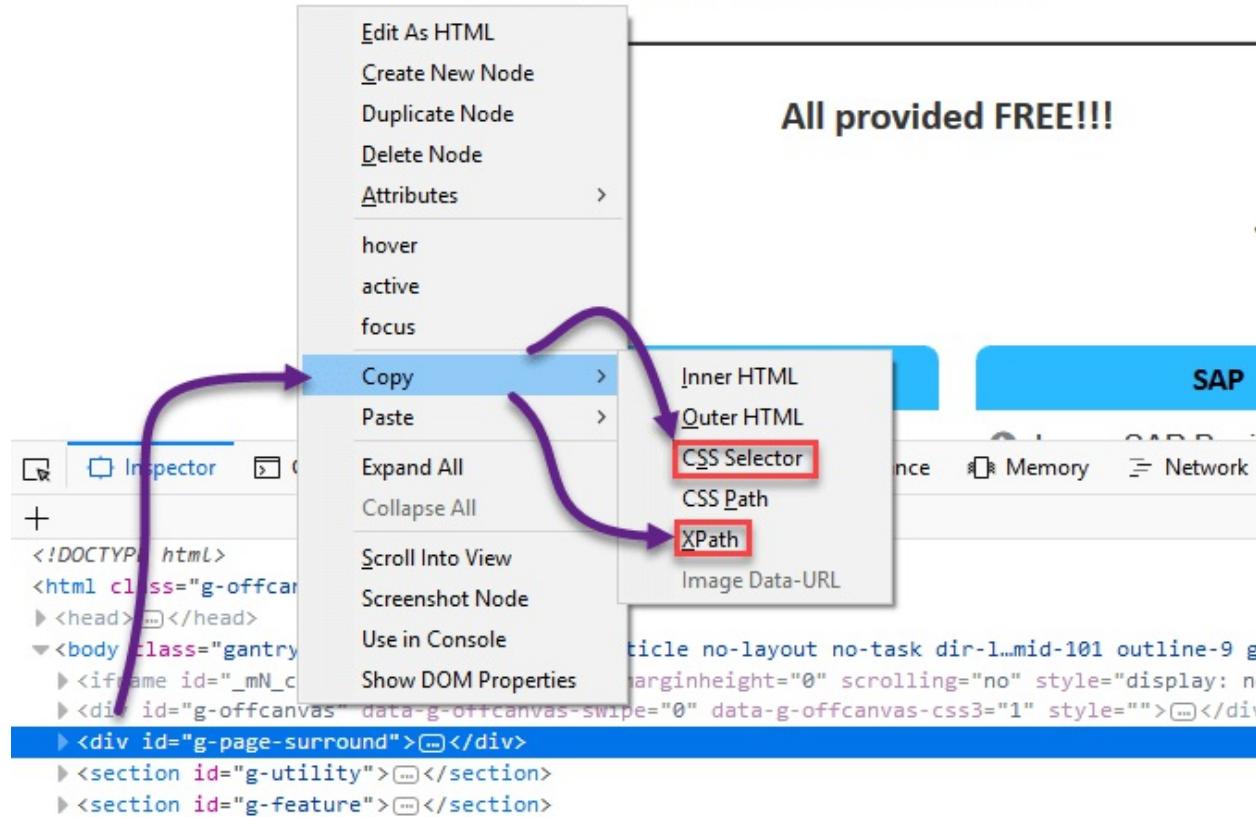
Step 2) You will see the Interface



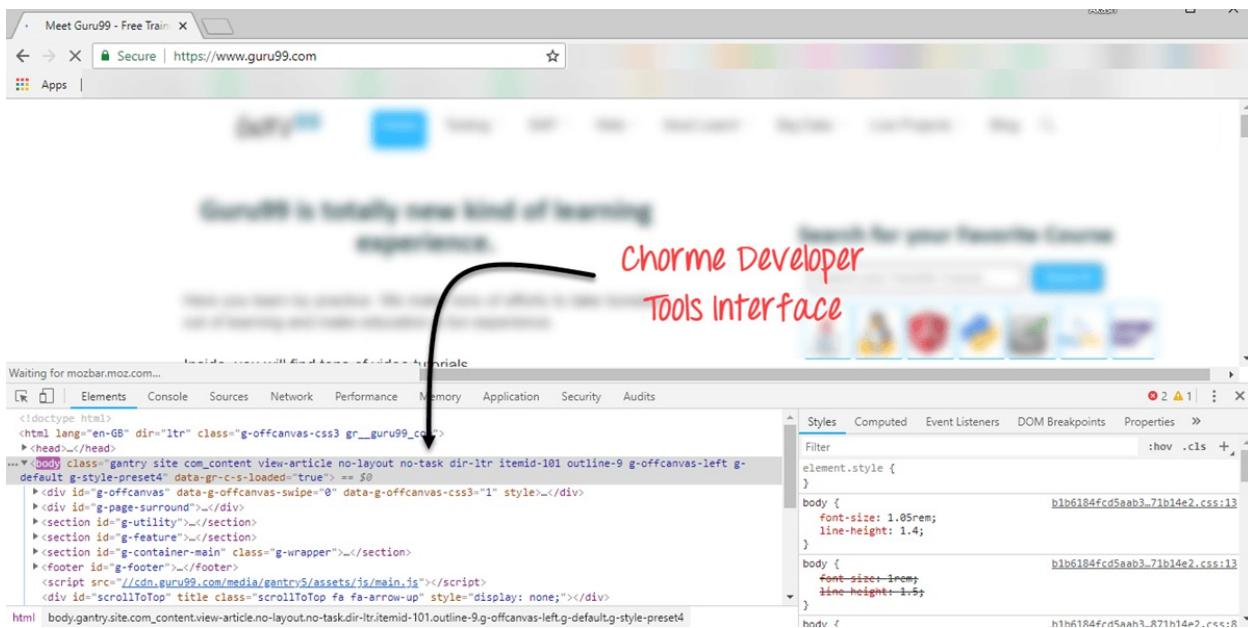
Step 3) You can right click on an element and chose CSS or XPath.
This is useful in object identification

Here you learn by practice. We make tons of efforts to take you out of learning and make education a fun experience.

Inside, you will find tons of video tutorials



Note: Likewise, you can also use Developer Tools in Chrome to identify object properties



Selenium IDE was deprecated, and the development had stopped. Only recently the project has been resurrected. The new Selenium lacks many features compared to the deprecated IDE. Features are being added but at a slow pace. To explore all the features of Selenium IDE, we recommend you use the old version. To use the old version of IDE

Step 1) Use Firefox 54 Portable Version check here

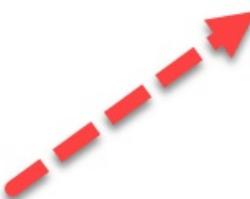
Step 2) Visit Selenium IDE version <https://addons.mozilla.org/en-US/firefox/addon/selenium-ide/> and install

Version 2.9.1.1-signed
Released March 15, 2015 · 796.5 KB
Works with Firefox 17.0 - 56.*

Selenium IDE v2.9.1
Merged all the official language exporters into the main addon and eliminated having a multi-xpi.

Source code released under [Custom License](#)

+ Add to Firefox



The following features may not be available in latest IDE version. We

will keep updating the tutorials as the new version is updated.

Plugins

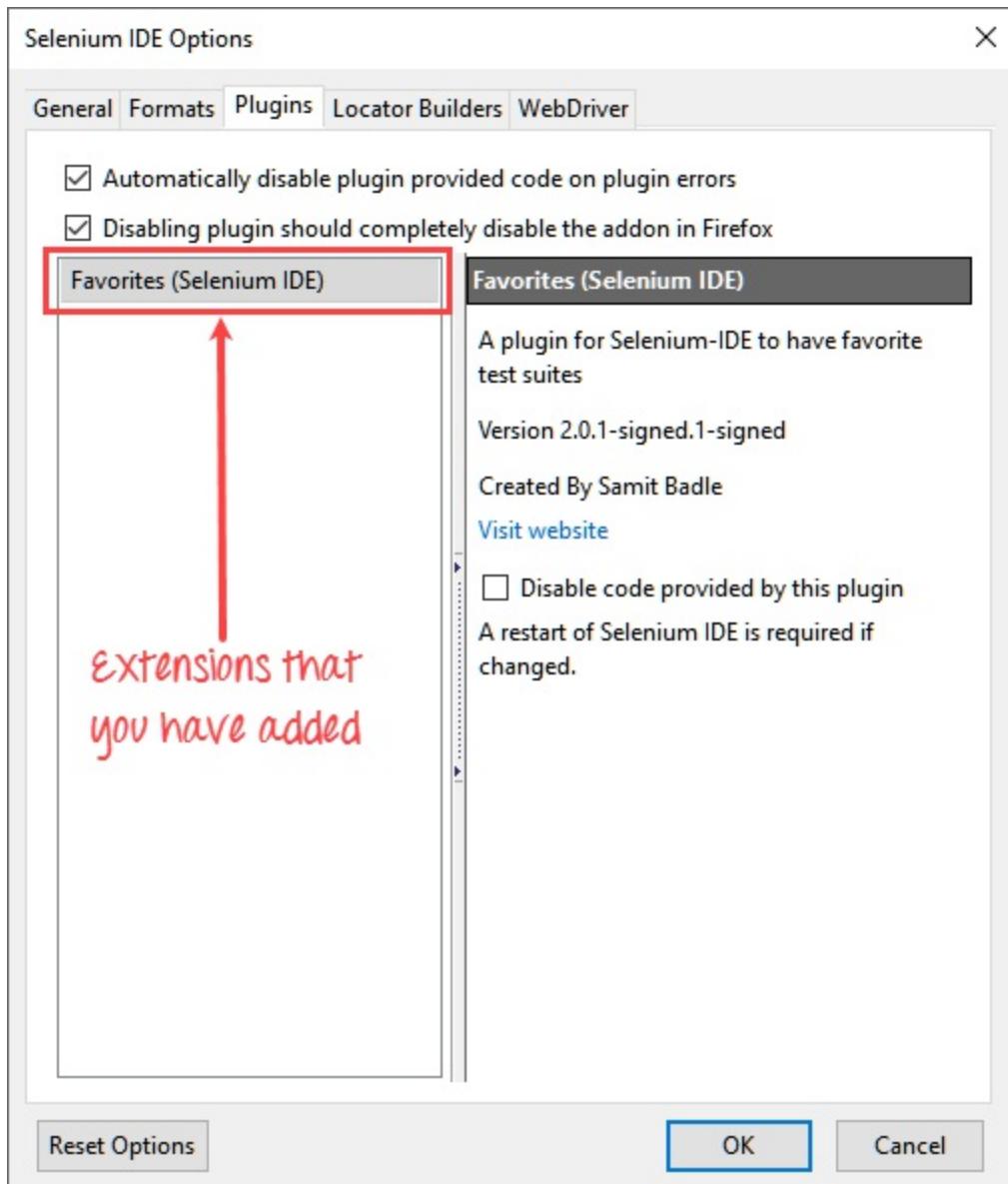
Selenium IDE can support additional Firefox add-ons or plugins created by other users. You can visit here for a list of Selenium add-ons available to date. Install them just as you do with other Firefox add-ons.

By default, Selenium IDE comes bundled with 4 plugins:

1. Selenium IDE: C# Formatters
2. Selenium IDE: Java Formatters
3. Selenium IDE: Python Formatters
4. Selenium IDE: Ruby Formatters

These four plugins are required by Selenium IDE to convert Selenese into different formats.

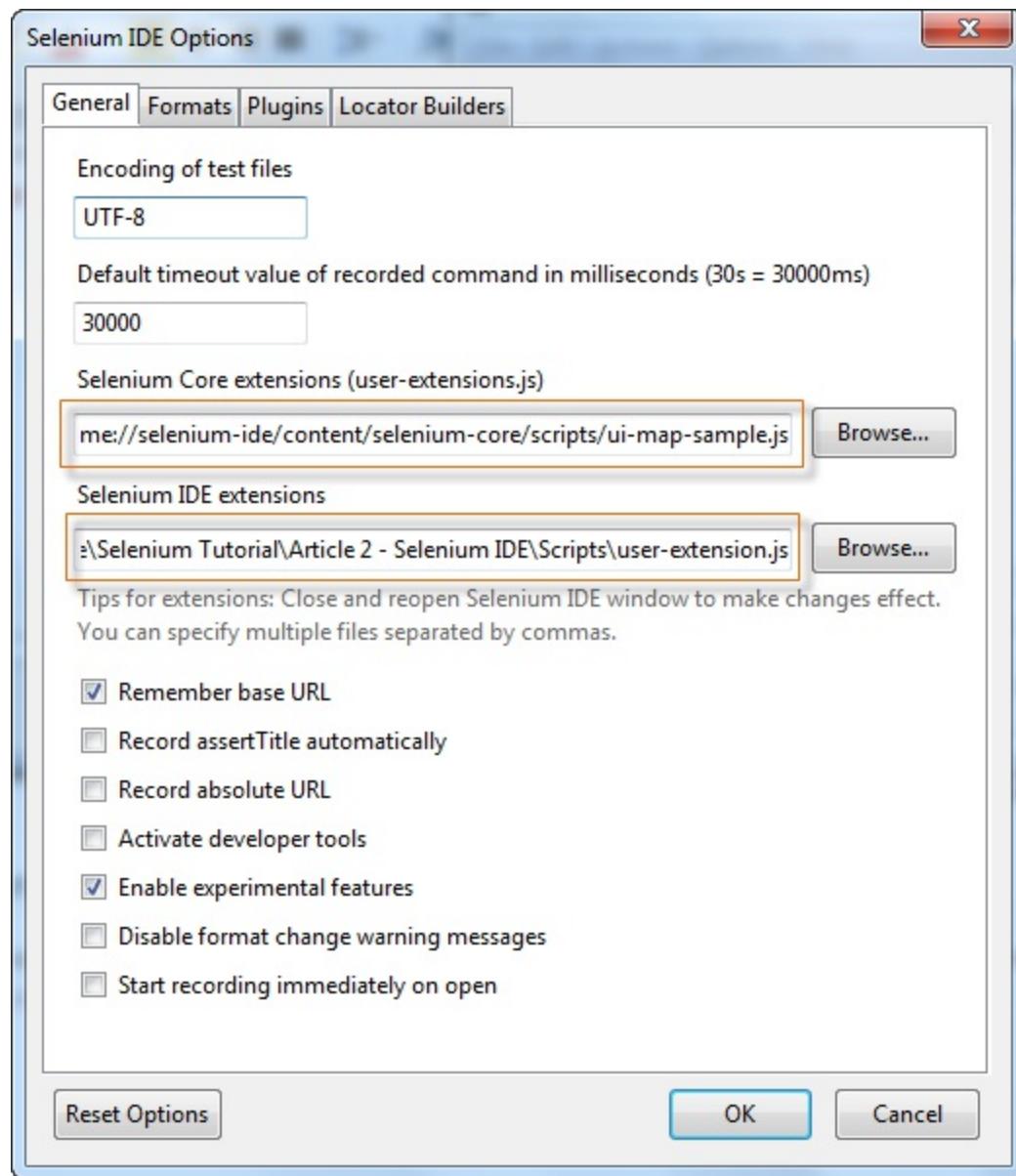
The Plugins tab shows a list of all your installed add-ons, together with the version number and name of the creator of each.



User Extensions

Selenium IDE can support user extensions to provide advanced capabilities. User extensions are in the form of JavaScript files. You install them by specifying their absolute path in either of these two fields in the Options dialog box.

- Selenium Core extensions (user-extensions.js)
- Selenium IDE extensions



You will be able to find tons of user extensions here.

Chapter 3: Introduction to Selenium IDE

Selenium IDE (Integrated Development Environment) is the simplest tool in the Selenium Suite. It is a Firefox add-on that creates tests very quickly through its record-and-playback functionality. This feature is similar to that of QTP. It is effortless to install and easy to learn.

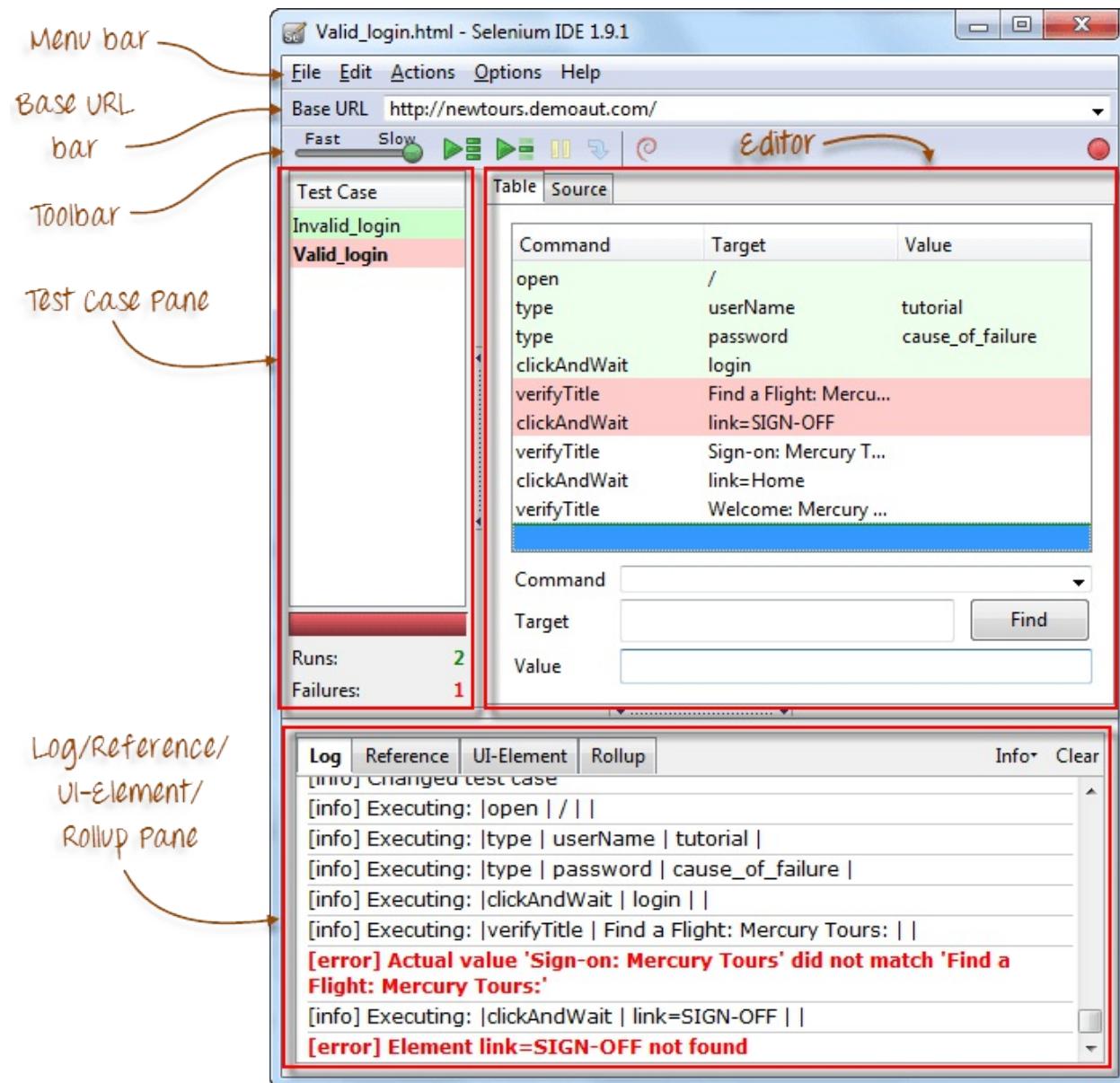
Because of its simplicity, Selenium IDE should only be used as a prototyping tool, not an overall solution for developing and maintaining complex test suites.

Though you will be able to use Selenium IDE without prior knowledge in programming, you should at least be familiar with HTML, JavaScript, and the DOM (Document Object Model) to utilize this tool to its full potential. Knowledge of JavaScript will be required when we get to the section about the Selenese command "**runScript**."

Selenium IDE supports autocomplete mode when creating tests. This feature serves two purposes:

- It helps the tester to enter commands more quickly.
- It restricts the user from entering invalid commands.

Features of Selenium IDE



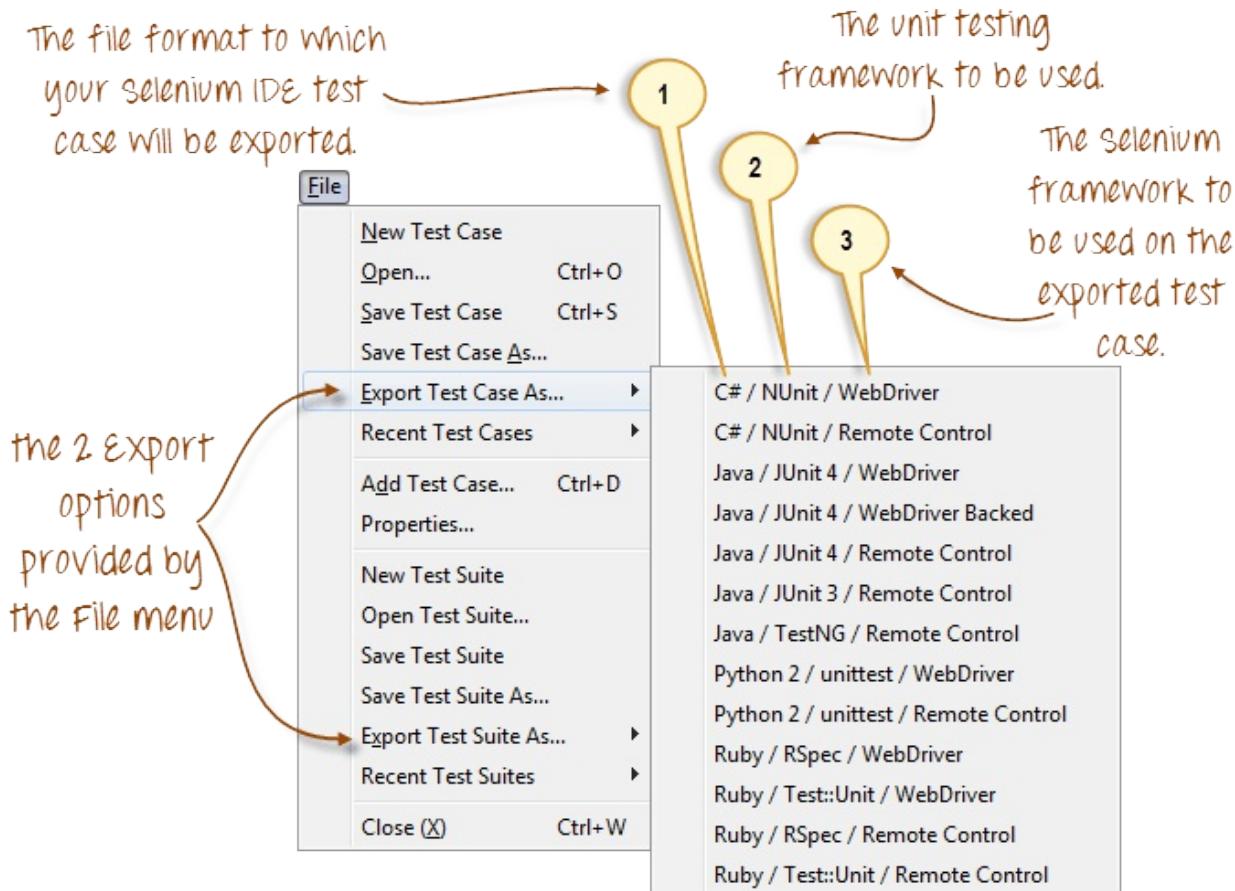
Menu Bar

It is located at the **top most portion** of the IDE. The most commonly used menus are the File, Edit, and Options menus.

File menu

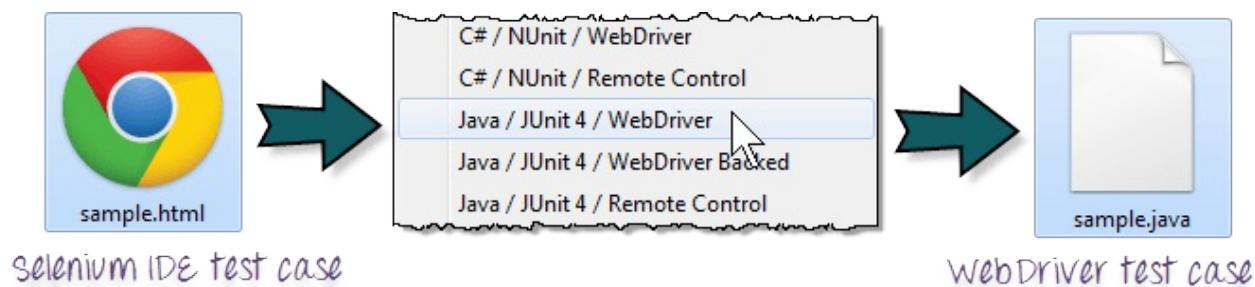
- It contains options to create, open, save and close tests.

- Tests are **saved in HTML format**.
- The most useful option is "**Export**" because **it allows you to turn your Selenium IDE test cases into file formats that can run on Selenium Remote Control and WebDriver**
 - "**Export Test Case As...**" will export only the currently opened test case.
 - "**Export Test Suite As...**" will export all the test cases in the currently opened test suite.



- As of **Selenium IDE v1.9.1**, test cases can be exported only to the following formats:
- .cs (C# source code)
- .java (Java source code)
- .py (Python source code)

- .rb (Ruby source code)

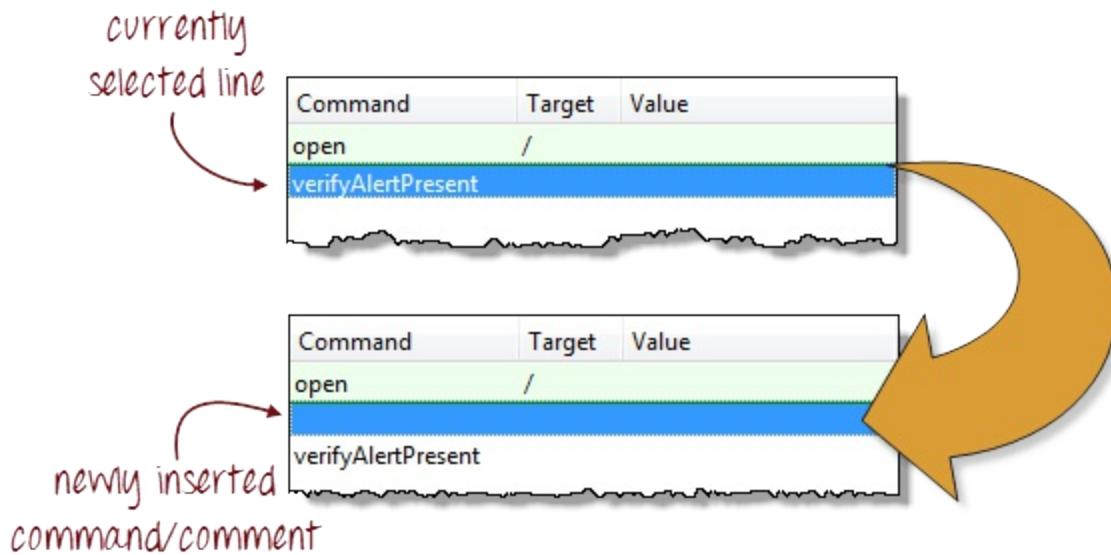


Edit Menu

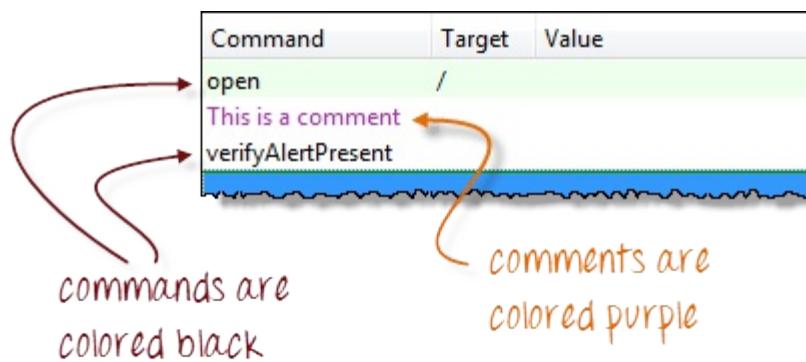
- It contains usual options like Undo, Redo, Cut, Copy, Paste, Delete, and Select All.
- The two most important options are the "**Insert New Command**" and "**Insert New Comment**".



- The newly inserted command or comment **will be placed on top of the currently selected line**.



- **Commands** are colored **black**.
- **Comments** are colored **purple**.

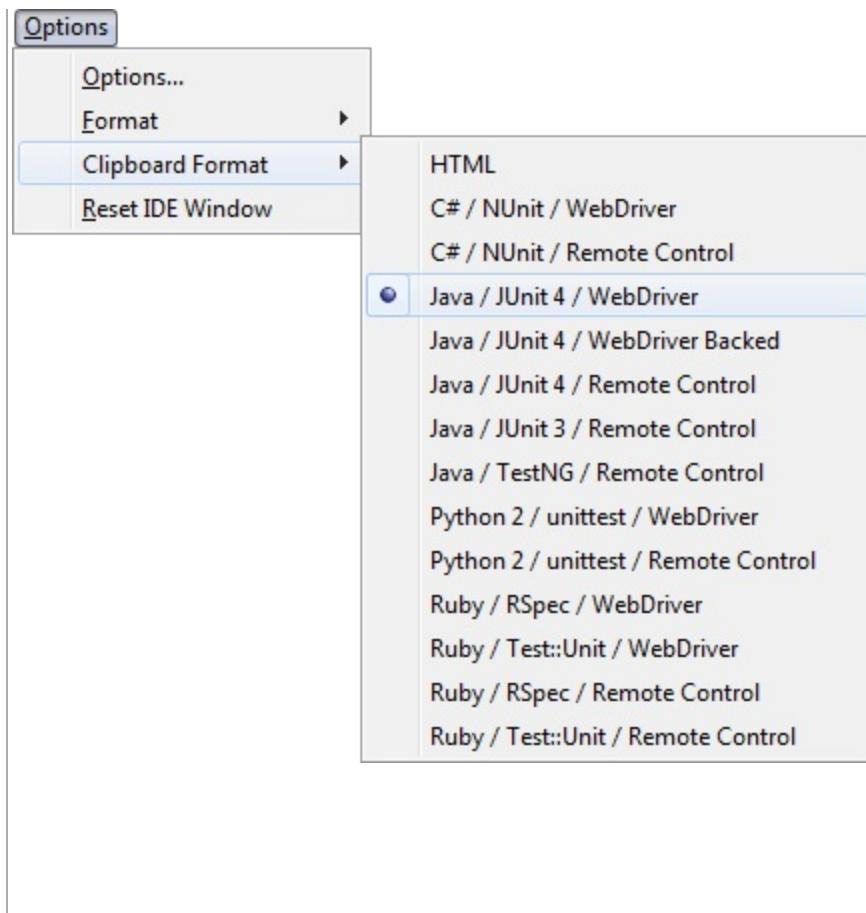


Options menu

It provides the **interface for configuring various settings** of Selenium IDE.

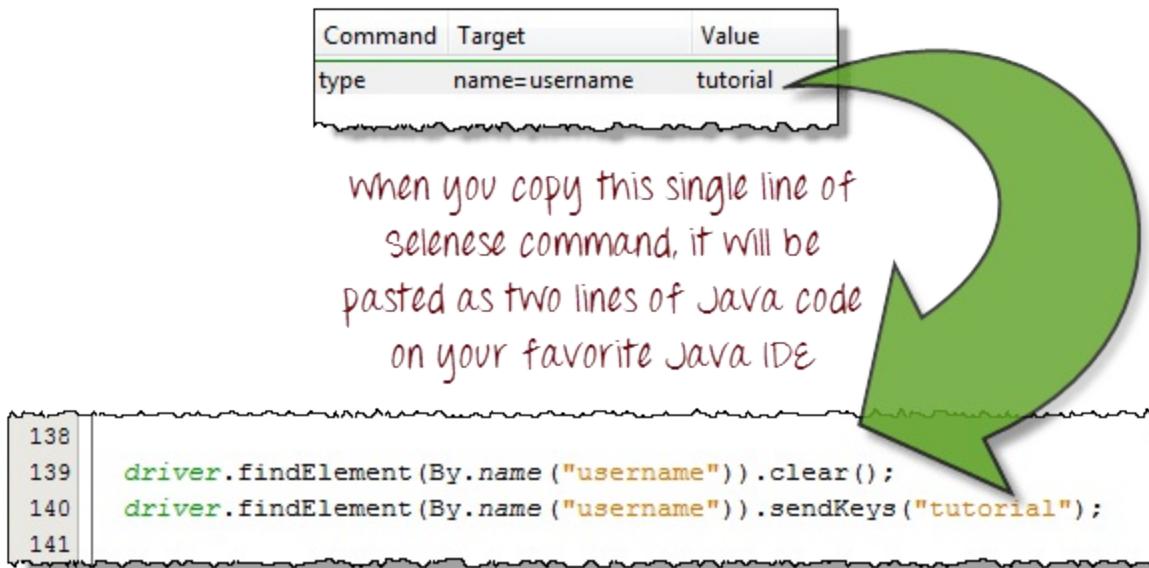
We shall concentrate on the **Options** and **Clipboard Format** options.

Clipboard Format



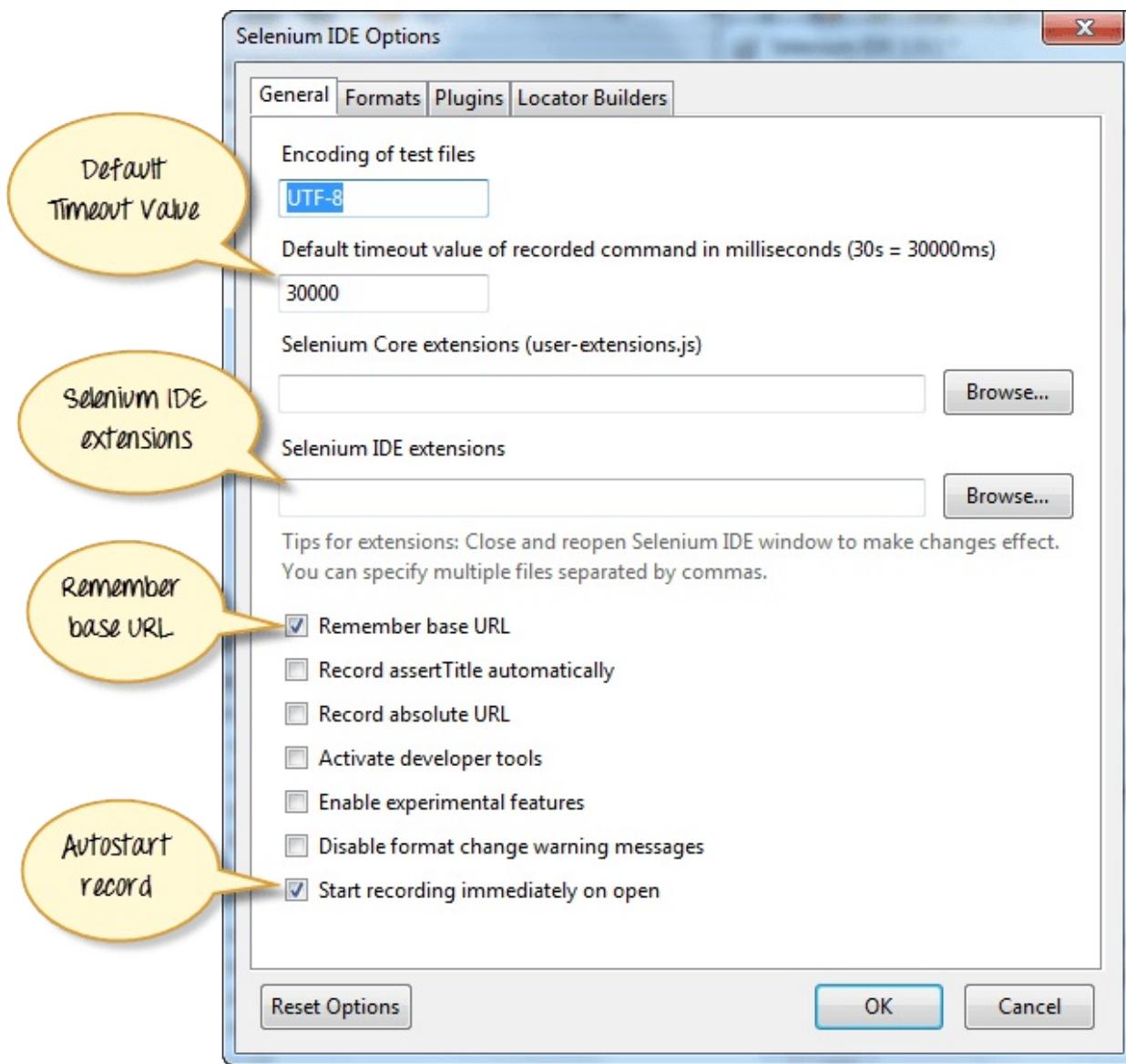
- **The Clipboard Format allows you to copy a Selenese command from the editor and paste it as a code snippet.**
- The format of the code follows the option you selected here in Clipboard Format's list.
- **HTML is the default selection.**

For example, when you choose **Java/JUnit 4/WebDriver** as your clipboard format, every Selenese command you copy from Selenium IDE's editor will be pasted as **Java code**. See the illustration below.



Selenium IDE Options dialog box

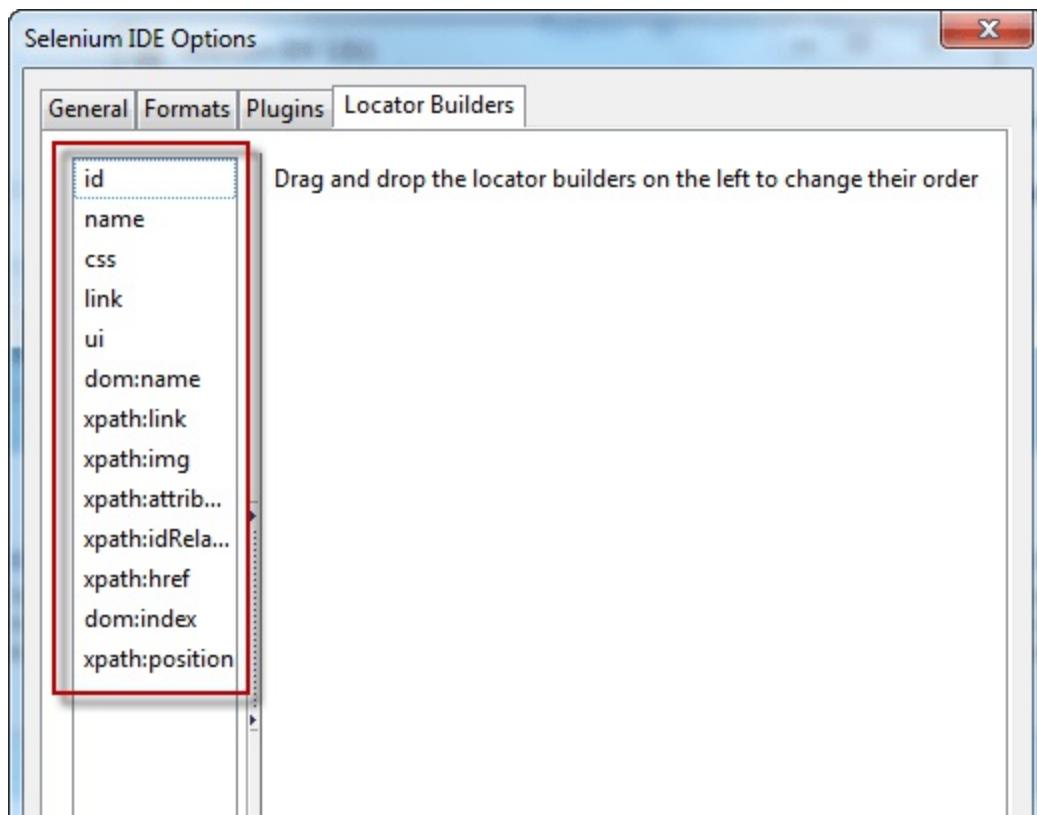
You can launch the Selenium IDE Options dialog box by clicking Options > Options... on the menu bar. Though there are many settings available, we will concentrate on the few important ones.



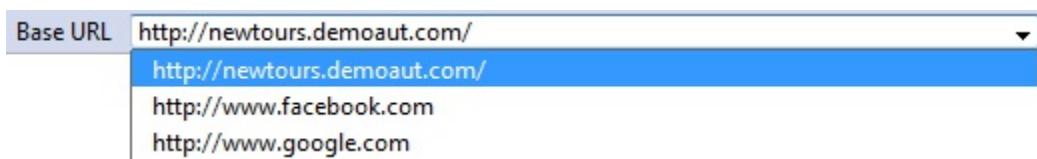
- **Default Timeout Value.** This refers to the time that Selenium has to wait for a certain element to appear or become accessible before it generates an error. **Default timeout value is 30000ms.**
- **Selenium IDE extensions.** This is where you specify the extensions you want to use to extend Selenium IDE's capabilities. You can visit <http://addons.mozilla.org/en-US/firefox/> and use "Selenium" as a keyword to search for the specific extensions.
- **Remember base URL.** Keep this checked if you want Selenium

IDE to remember the Base URL every time you launch it. If you uncheck this, Selenium IDE will always launch with a blank value for the Base URL.

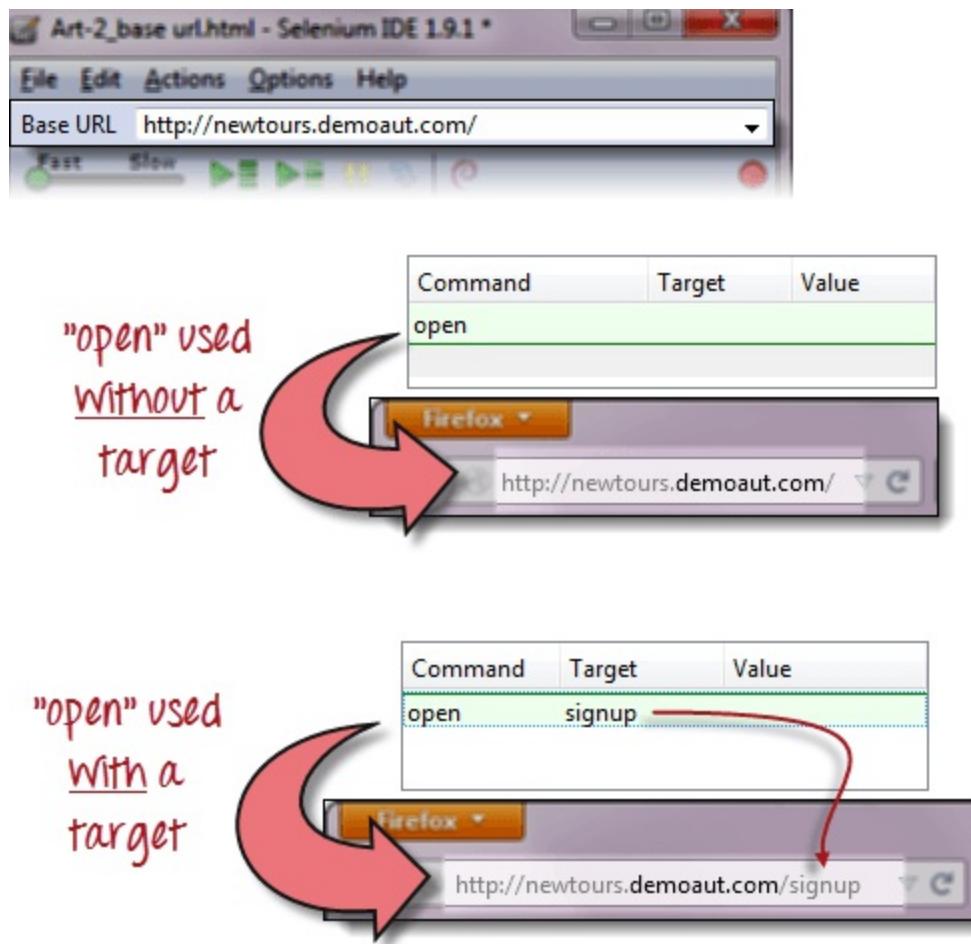
- **Autostart record.** If you check this, Selenium IDE will immediately record your browser actions upon startup.
- **Locator builders.** This is where you specify the order by which locators are generated while recording. **Locators are ways to tell Selenium IDE which UI element should a Selenese command act upon.** In the setup below, when you click on an element with an ID attribute, that element's ID will be used as the locator since "id" is the first one in the list. If that element does not have an ID attribute, Selenium will next look for the "name" attribute since it is second in the list. The list goes on and on until an appropriate one is found.



Base URL Bar



- It has a **dropdown menu that remembers all previous values** for easy access.
- The Selenese command "**open**" will take you to the URL that you specified in the **Base URL**.
- In this tutorial series, we will be using <http://newtours.demoaut.com> as our Base URL. It is the site for Mercury Tours, a web application maintained by HP for web Testing purposes. We shall be using this application because it contains a complete set of elements that we need for the succeeding topics.
- **The Base URL is very useful in accessing relative URLs.** Suppose that your Base URL is set to <http://newtours.demoaut.com>. When you execute the command "open" with the target value "signup," Selenium IDE will direct the browser to the sign-up page. See the illustration below.



Toolbar

	Playback Speed. This controls the speed of your Test Script Execution.
	Record. This starts/ends your recording session. Each browser action is entered as a Selenese command in the Editor.
	Play entire test suite. This will sequentially play all the test cases listed in the Test Case Pane.
	Play current test case. This will play only the currently selected test case in the Test Case Pane.
	Pause/Resume. This will pause or resume your playback.
	Step. This button will allow you to step into each command in your test script.
	Apply rollup rules. This is an advanced functionality. It allows you to group Selenese commands together and execute them as a single

|action.

Test Case Pane

The screenshot shows the Test Case pane in Selenium IDE. It lists two test cases: "Invalid_login" and "Valid_login". "Invalid_login" is highlighted in green, indicating it passed. "Valid_login" is highlighted in red, indicating it failed. Below the list, a summary bar shows "Runs: 2" and "Failures: 1". Handwritten annotations explain the color coding: "green means this test case Passed" points to the green-highlighted "Invalid_login"; "red means this test case Failed" points to the red-highlighted "Valid_login"; "color represents status of the whole test suite" points to the overall status bar; "# of test cases run" points to "Runs: 2"; and "# of failed test cases" points to "Failures: 1".

- In Selenium IDE, you can open **more than one test case at a time**.
- The **test case pane** shows you the **list of currently opened test cases**.
- When you open a test suite, the test case pane will **automatically list all the test cases** contained in it.
- The test case written in **bold font** is the **currently selected test case**
- After playback, **each test case is color-coded** to represent if it passed or failed.
 - Green color means "Passed."
 - Red color means "Failed."
- At the bottom portion is a summary of the number of test cases that were run and failed.

Editor

You can think of the editor as **the place where all the action happens**. It is available in two views: Table and Source.

Table View

- Most of the time, you will work on Selenium IDE using the **Table View**.
- This is **where you create and modify Selenese commands**.
- After playback, each step is color-coded.

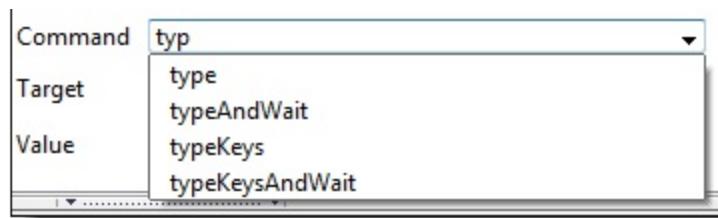
Command	Target	Value
open	/	
type	userName	tutorial
type	password	tutorial
clickAndWait	login	
verifyTitle	Find a Flight: Mercu...	
clickAndWait	link=SIGN-OFF	
verifyTitle	Sign-on: Mercury T...	
clickAndWait	link=Home	
verifyTitle	Welcome: Mercury ...	

Command

Target

Value

- To create steps, type the name of the command in the "Command" text box.
- **It displays a dropdown list of commands** that match with the entry that you are currently typing.
- Target is any parameter (like username, password) for a command and Value is the input value (like tom, 123pass) for those Targets.



Source View

- It displays the steps in HTML (default) format.
- It also allows you to edit your script just like in the Table View.

The Source View tab of the Selenium IDE shows an HTML script. The code includes several `<tr>` and `</tr>` tags, each containing multiple `<td>` tags. The content within the `<td>` tags includes various Selenium commands and their parameters, such as `verifyTitle`, `clickAndWait`, and `link=Home`. The entire script is enclosed in `<tbody></tbody>`, `</body>`, and `</html>`.

```

<tr>
    <td>verifyTitle</td>
    <td>Sign-on: Mercury Tours </td>
    <td></td>
</tr>
<tr>
    <td>clickAndWait</td>
    <td>link=Home</td>
    <td></td>
</tr>
<tr>
    <td>verifyTitle</td>
    <td>Welcome: Mercury Tours</td>
    <td></td>
</tr>
</tbody></table>
</body>
</html>

```

Log Pane

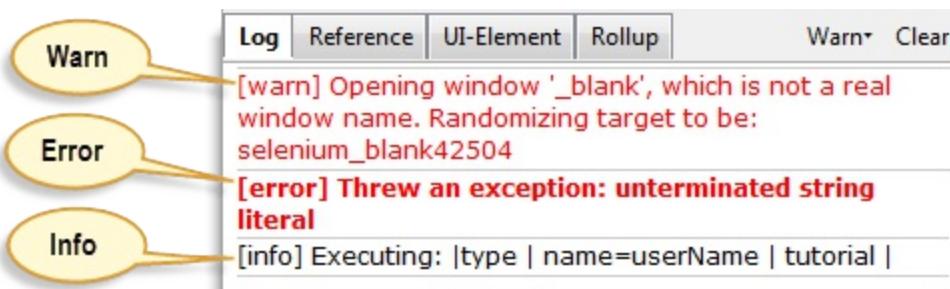
The Log Pane displays runtime messages during execution. It provides real-time updates as to what Selenium IDE is doing.

Logs are categorized into four types:

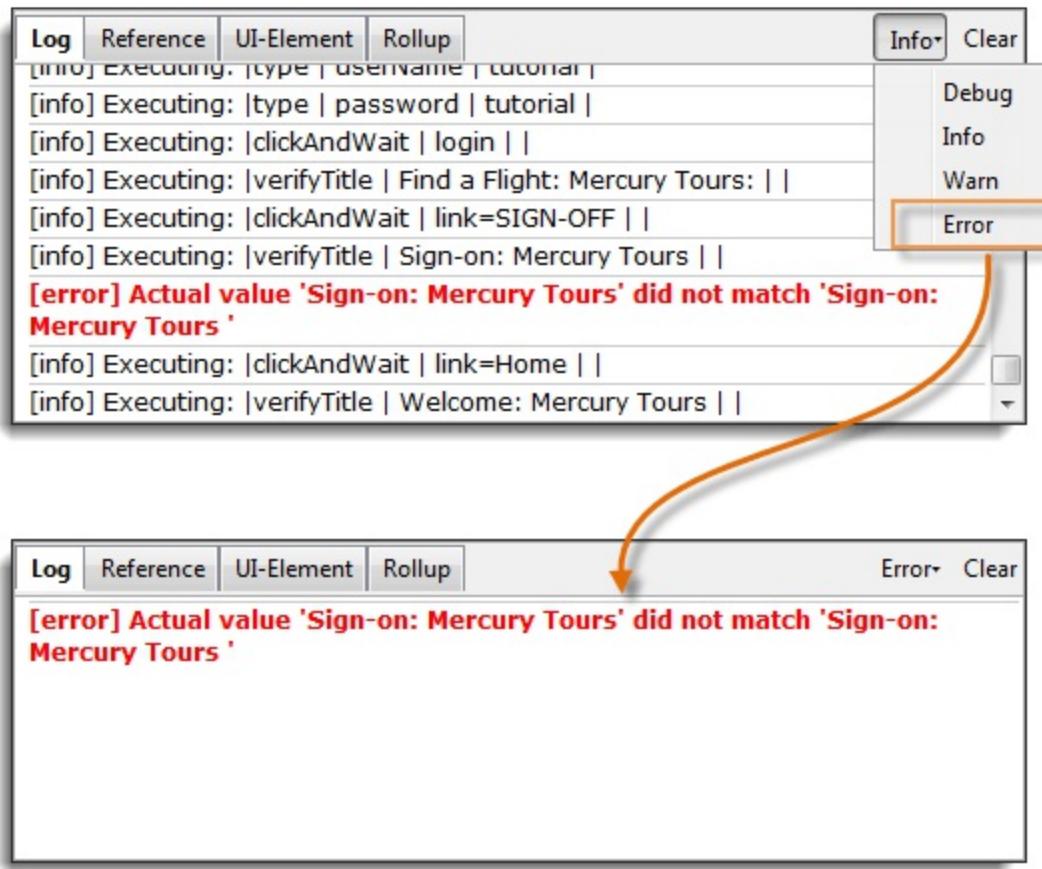
- Debug - By default, Debug messages are not displayed in the log

panel. They show up only when you filter them. They provide technical information about what Selenium IDE is doing behind the scenes. It may display messages such as a specific module has done loading, a certain function is called, or an external JavaScript file was loaded as an extension.

- Info - It says which command Selenium IDE is currently executing.
- Warn - These are warning messages that are encountered in special situations.
- Error - These are error messages generated when Selenium IDE fails to execute a command, or if a condition specified by "verify" or "assert" command is not met.

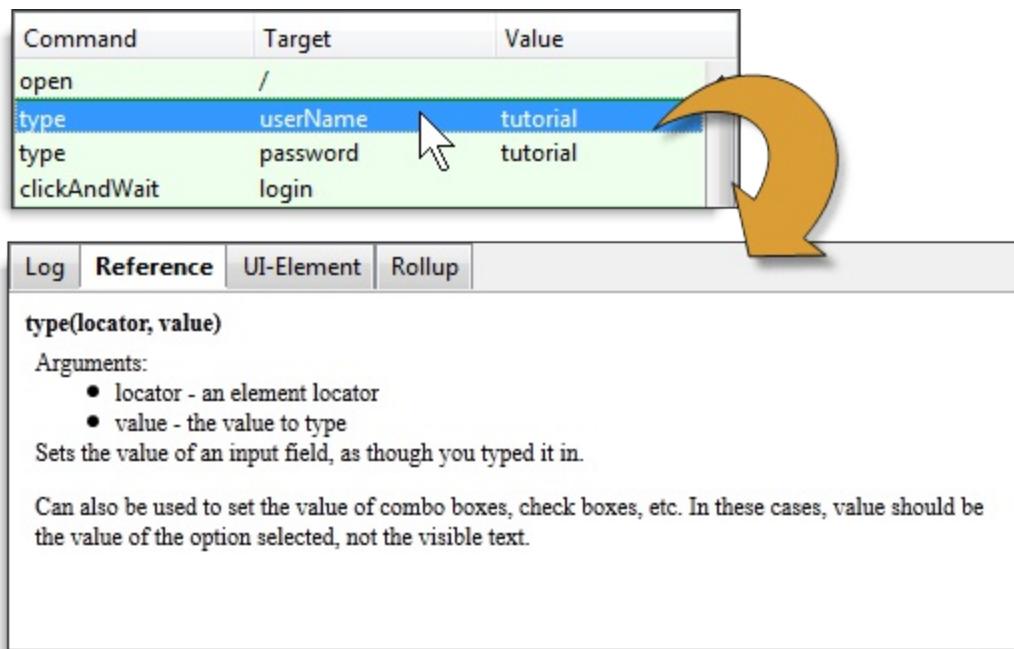


Logs can be filtered by type. For example, if you choose to select the "Error" option from the dropdown list, the Log Pane will show error messages only.



Reference Pane

The Reference Pane shows a concise description of the currently selected Selenese command in the Editor. It also shows the description about the locator and value to be used on that command.



A screenshot of the Selenium IDE interface. At the top, there's a table with three columns: Command, Target, and Value. The table contains the following rows:

Command	Target	Value
open	/	
type	userName	tutorial
type	password	tutorial
clickAndWait	login	

An orange arrow points from the bottom right towards the 'type' command in the table.

Below the table is a navigation bar with tabs: Log, Reference, UI-Element, and Rollup. The 'Reference' tab is selected. To its right is a detailed description of the 'type(locator, value)' command:

type(locator, value)

Arguments:

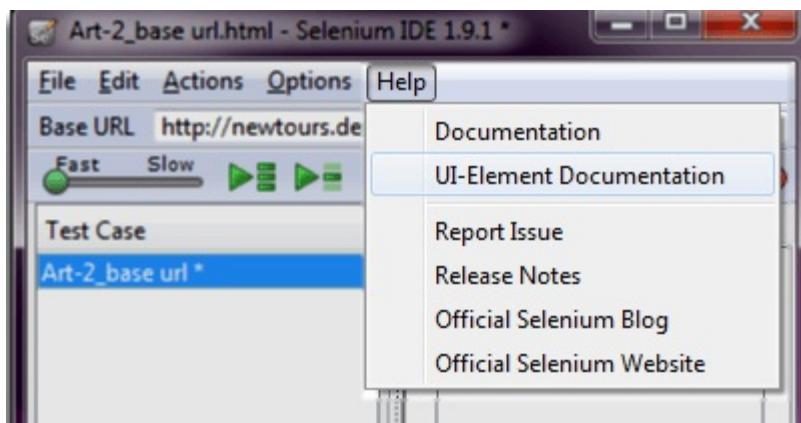
- locator - an element locator
- value - the value to type

Sets the value of an input field, as though you typed it in.

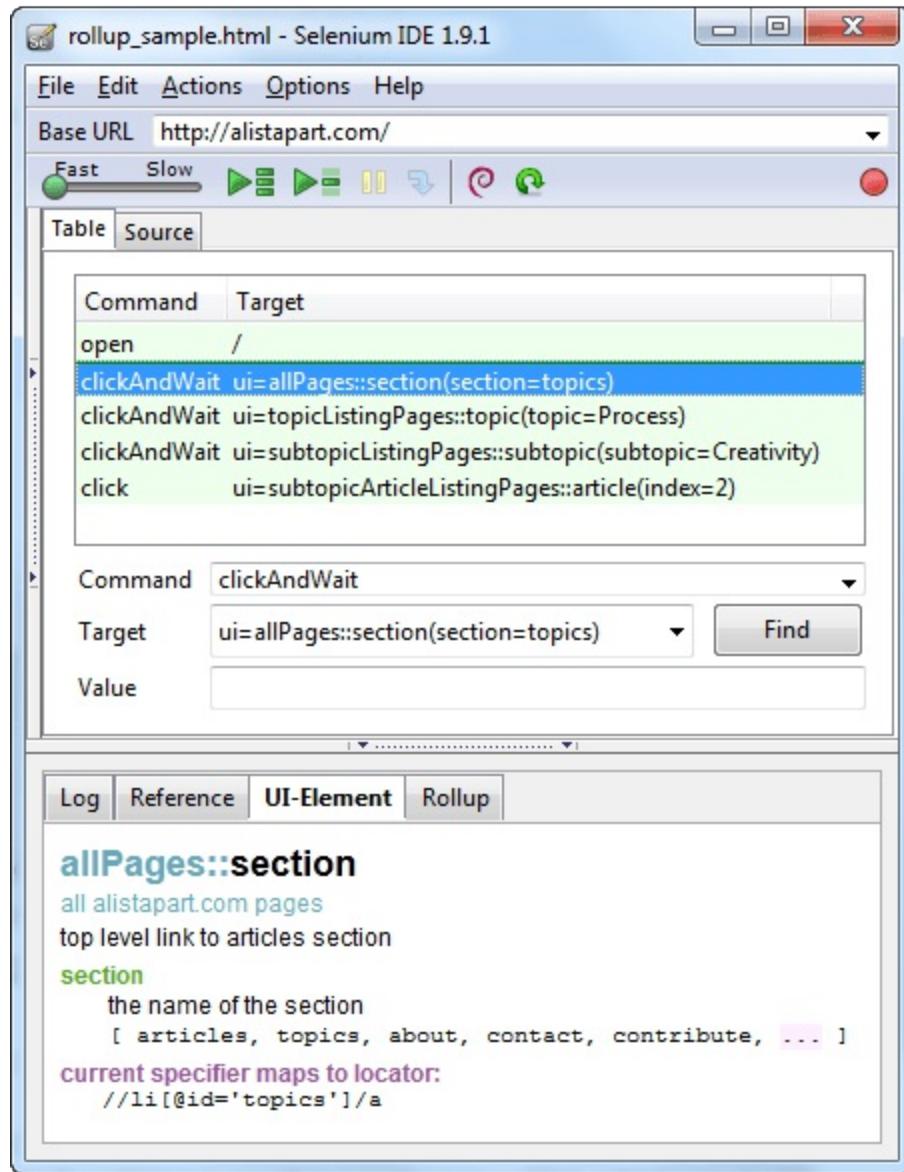
Can also be used to set the value of combo boxes, check boxes, etc. In these cases, value should be the value of the option selected, not the visible text.

UI-Element Pane

The UI-Element is for advanced Selenium users. **It uses JavaScript Object Notation (JSON) to define element mappings.** The documentation and resources are found in the "UI Element Documentation" option under the Help menu of Selenium IDE.



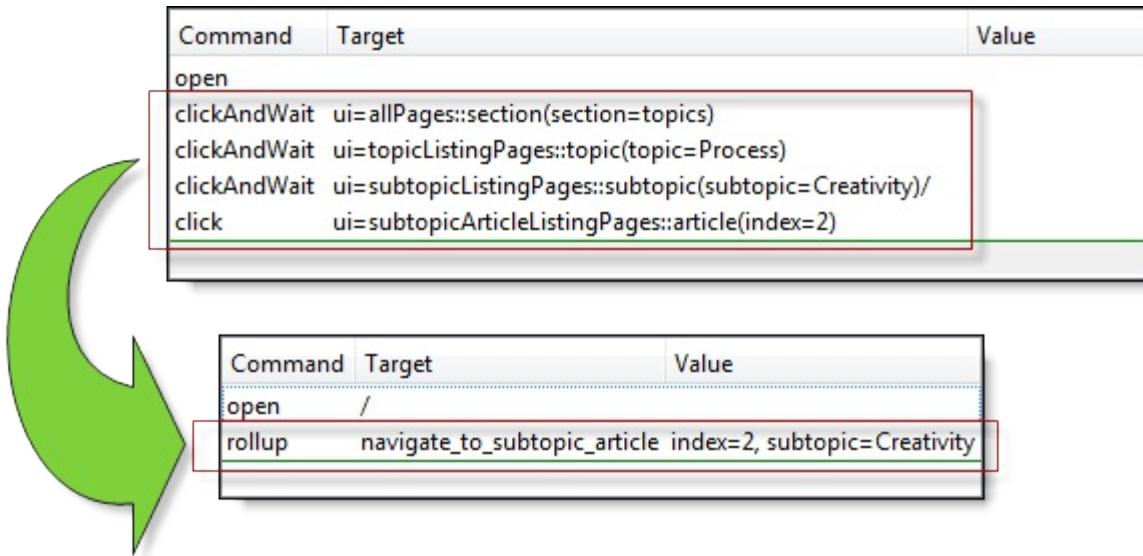
An example of a UI-element screen is shown below.



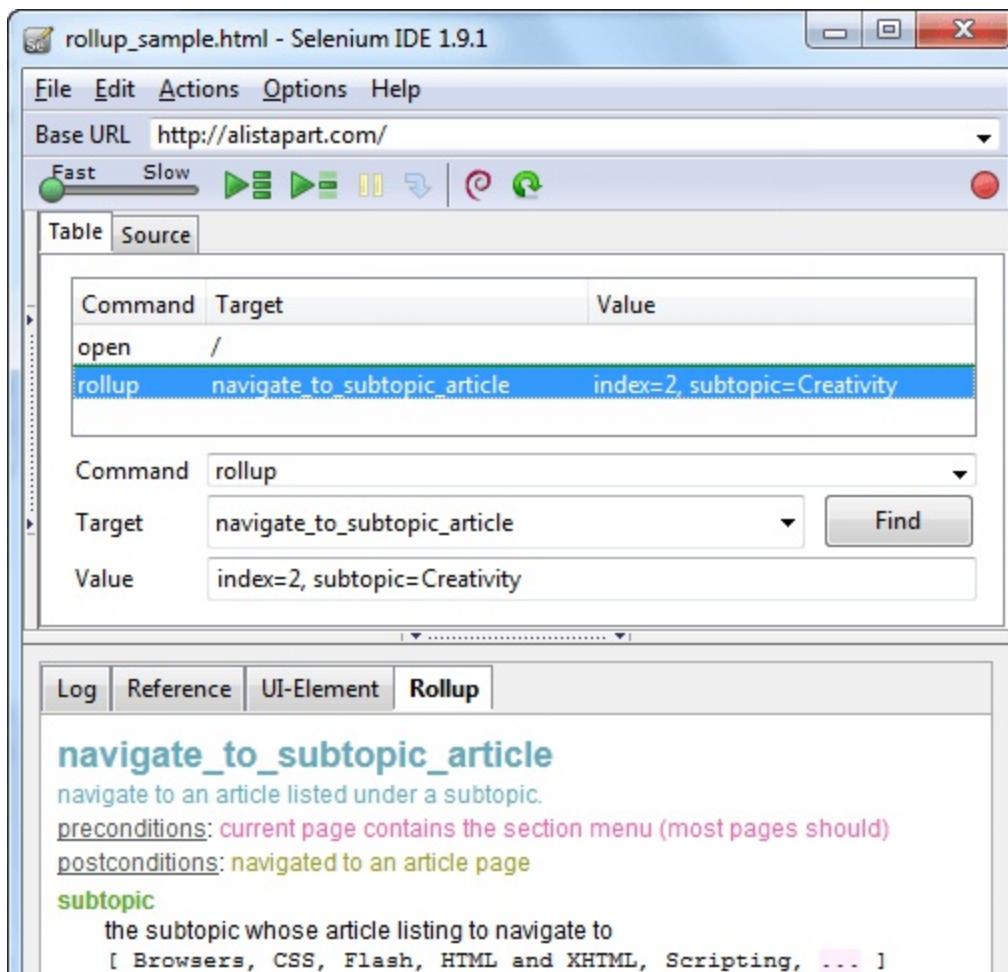
Rollup Pane

Rollup allows you to execute a group of commands in one step. A group of commands is simply called as a "rollup." It employs heavy use of JavaScript and UI-Element concepts to formulate a collection of commands that is similar to a "function" in programming languages.

Rollups are reusable; meaning, they can be used multiple times within the test case. Since rollups are groups of commands condensed into one, they contribute a lot in shortening your test script.



An example of how the contents of the rollup tab look like is shown below.



Summary

- Selenium IDE (Integrated Development Environment) **is the simplest tool** in the Selenium Suite.
- It must only be used as a **prototyping tool**.
- **Knowledge of JavaScript and HTML is required for intermediate topics** such as executing the "runScript" and "rollup" commands. A **rollup** is a collection of commands that you can reuse to shorten your test scripts significantly. **Locators** are identifiers that tell Selenium IDE how to access an element.
- **Firebug** (or any similar add-on) is used to obtain locator values.
- The **menu bar** is used in creating, modifying, and exporting test

cases into formats useable by Selenium RC and WebDriver.

- The **default format for Selenese commands is HTML**.
- The "**Options**" menu provides access to various configurations for Selenium IDE.
- The **Base URL** is useful in accessing **relative URLs**.
- The **Test Case Pane** shows the list of currently opened test cases and a concise summary of test runs.
- The **Editor** provides the **interface for your test scripts**.
- The **Table View** shows your script **in tabular format** with "Command", "Target", and "Value" as the columns.
- The **Source View** shows your script **in HTML format**.
- The **Log** and **Reference** tabs give feedback and other useful information when executing tests.
- The **UI-Element and Rollup** tabs are **for advanced Selenium IDE users only**. They both require considerable effort in coding JavaScript.
- **UI-Element** allows you to **conveniently map UI elements** using JavaScript Object Notation (JSON).

The following table summarizes the release history for the Selenium IDE.

Major version	Release date
1.0.10	06-Dec-10
1.5.0	15-Dec-11
1.8.1	01-Jun-12
2.1.0	30-Jun-13
2.2.0	06-Jul-13
2.3.0	09-Aug-13
2.5.0	02-Jan-14
2.8.0	29-Sep-14
2.9.0	09-Mar-15

2.9.1

- to be released

Chapter 4: Creating your First Selenium IDE script

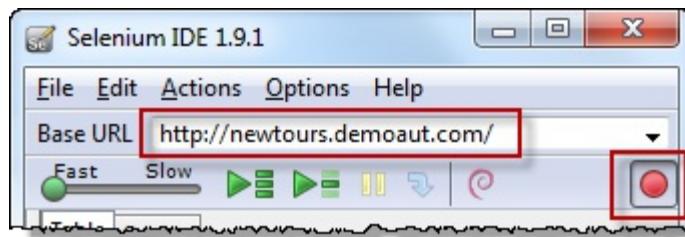
We will use the Mercury Tours website as our web application under test. It is an online flight reservation system that contains all the elements we need for this tutorial. Its URL is <http://demo.guru99.com/test/newtours/>, and this will be our Base URL.

Create a Script by Recording

Let us now create our first test script in Selenium IDE using the most common method - by recording. Afterward, we shall execute our script using the playback feature.

Step 1

- Launch Firefox and Selenium IDE.
- Type the value for our Base URL:
<http://demo.guru99.com/test/newtours/>.
- Toggle the Record button on (if it is not yet toggled on by default).



Step 2

In Firefox, navigate to <http://demo.guru99.com/test/newtours/>. Firefox should take you to the page similar to the one shown below.


MERCURY TOURS


[Use Java Version](#)


[CLICK HERE](#)

TIP #93

Always carry a travel first aid kit with bandages, antacids, aspirin, bee sting wipes, and other basic necessities.

one cool summer 

[SIGN-ON](#) [REGISTER](#) [SUPPORT](#) [CONTACT](#)

Dec 4, 2012

[Find A Flight](#)

Registered users can **sign-in here** to find the lowest fare on participating airlines.

User Name:

Password:

[Destinations](#)

 Find detailed information about [your destination](#).

[Vacations](#)

 Read about our [featured vacation destinations](#).

[Register](#)

 [Register here](#) to join Mercury Tours!

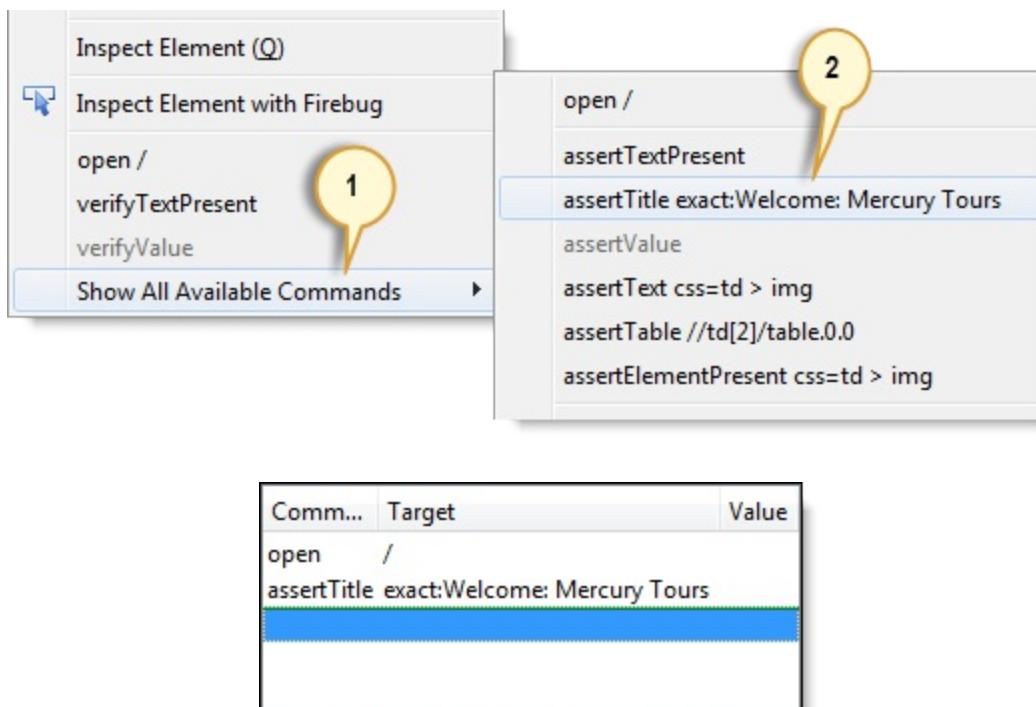
[Links](#)

[Business Travel @ About.com](#)
[Salon Travel](#)

© 2005, Mercury Interactive (v. 011003-1.01-058)

Step 3

- Right-click on any blank space within the page, like on the Mercury Tours logo on the upper left corner. This will bring up the Selenium IDE context menu. Note: Do not click on any hyperlinked objects or images
- Select the "Show Available Commands" option.
- Then, select "assertTitle exact: Welcome: Mercury Tours." This is a command that makes sure that the page title is correct.



After clicking on the assertTitle context menu option, your Selenium IDE Editor pane should now contain the following commands

Step 4

- In the "User Name" text box of Mercury Tours, type an invalid username, "invalidUNN".
- In the "Password" text box, type an invalid password, "invalidPWD".

Comm...	Target	Value
open	/	
assertTitle	exact:Welcome: Mercury Tours	
type	name=userName	invalidUN
type	name=password	invalidPW

Your Editor should now look like this

Step 5

- Click on the "Sign-In" button. Firefox should take you to this page.

SIGN-ON

Welcome back to Mercury Tours! Enter your user information to access the member-only areas of this site. If you don't have a log-in, please fill out the [registration form](#).

User Name:

Password:

SUBMIT

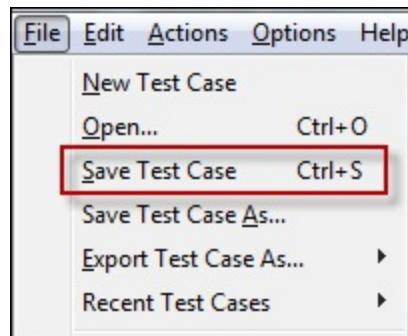
Step 6

Toggle the record button off to stop recording. Your script should now look like the one shown below.

Command	Target	Value
open		
assertTitle	exact:Welcome: Mercury Tours	
type	name=userName	invalidUN
type	name=password	invalidPW
clickAndWait	name=login	

Step 7

Now that we are done with our test script, we shall save it in a test case. In the File menu, select "Save Test Case". Alternatively, you can simply press Ctrl+S.



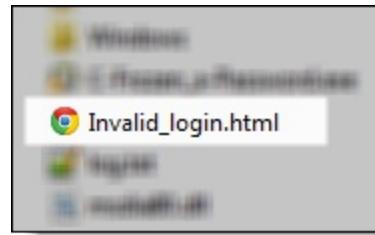
Step 8

- Choose your desired location, and then name the Test Case as "Invalid_login".
- Click the "Save" button.



Step 9.

Notice that the file was saved as HTML.



Step 10.

Go back to Selenium IDE and click the Playback button to execute the whole script. Selenium IDE should be able to replicate everything flawlessly.

The screenshot shows the Selenium IDE interface with a test case named "Invalid_login". The "Table" tab is selected. The command table contains the following entries:

Command	Target	Value
open		
assertTitle	exact:Welcome: Mercury Tours	
type	name=userName	invalidUN
type	name=password	invalidPW
clickAndWait	name=login	

Handwritten annotations in purple ink are present: "All green" with a red arrow pointing to the test case name; "The test Passed!!!" with a red arrow pointing to the "Runs: 1" and "Failures: 0" status bar at the bottom.

Introduction to Selenium Commands - Selenese

- Selenese commands can have up to a maximum of two parameters: target and value.
- Parameters are not required all the time. It depends on how many the command will need.

3 Types of Commands

Actions	<p>These are commands that directly interact with page elements.</p> <p>Example: the "click" command is an action because you directly interact with the element you are clicking at.</p> <p>The "type" command is also an action because you are putting values into a text box, and the text box shows them to you in return. There is a two-way interaction between you and the text box.</p>
Accessors	<p>They are commands that allow you to store values to a variable.</p> <p>Example: the "storeTitle" command is an accessor because it only "reads" the page title and saves it in a variable. It does not interact with any element on the page.</p>
Assertions	<p>They are commands that verify if a certain condition is met.</p> <h3>3 Types of Assertions</h3> <ul style="list-style-type: none"> • Assert. When an "assert" command fails, the test is stopped immediately. • Verify. When a "verify" command fails, Selenium IDE logs this failure and continues with the test execution. • WaitFor. Before proceeding to the next command, "waitFor" commands will first wait for a certain condition to become true. <ul style="list-style-type: none"> ◦ If the condition becomes true within the waiting period, the step passes. ◦ If the condition does not become true, the step fails. Failure is logged, and test execution proceeds to the next command. ◦ By default, the timeout value is set to 30 seconds. You can change this in the Selenium IDE Options dialog under the General tab.

Assert vs. Verify

ASSERT

test execution
was halted in
this part

no further logs were
displayed after this error
message, meaning that
execution indeed stopped

Command	Target	Value
open		
assertTitle	Welcome: Venus Tours	
type	name=userName	invalidUN
type	name=password	invalidPW
clickAndW...	name=login	

Command:

Target: Find

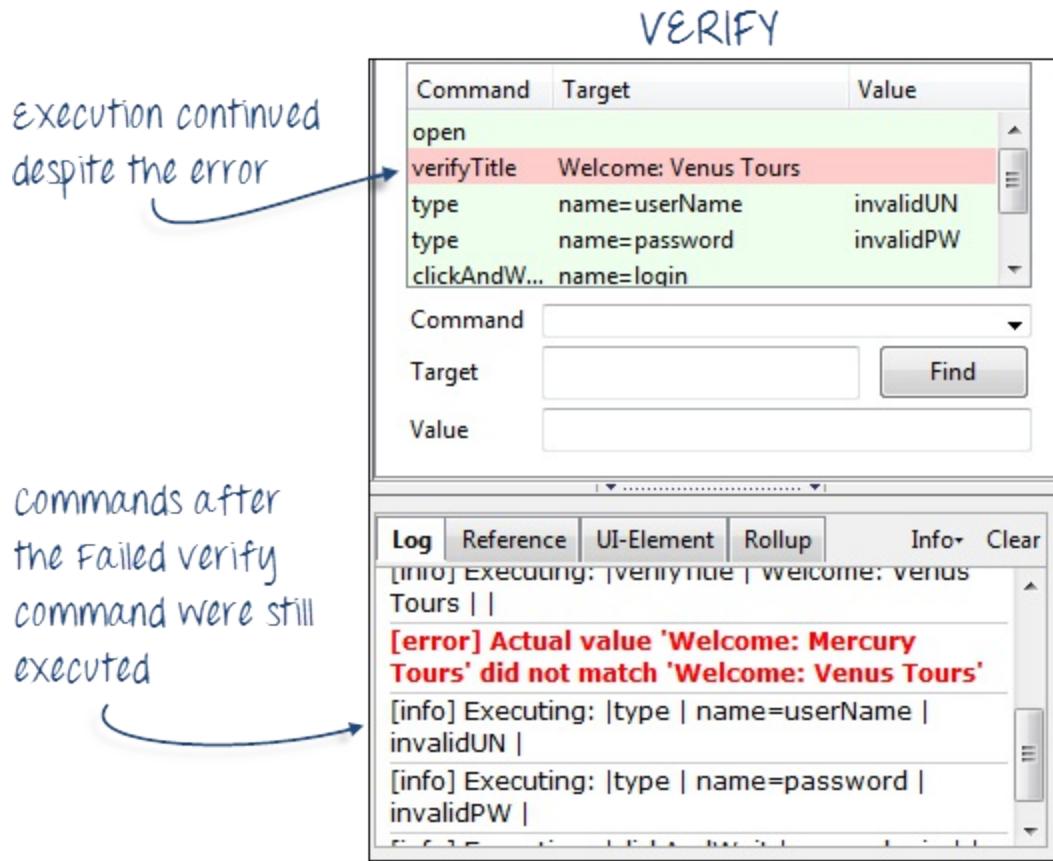
Value:

Log Reference UI-Element Rollup Info Clear

[info] Executing: |open |||

[info] Executing: |assertTitle | Welcome: Venus Tours ||

**[error] Actual value 'Welcome: Mercury Tours'
did not match 'Welcome: Venus Tours'**



Common Commands

Command	Number of Parameters	Description
open	0 - 2	Opens a page using a URL.
click/clickAndWait	1	Clicks on a specified element.
type/typeKeys	2	Types a sequence of characters.
verifyTitle/assertTitle	1	Compares the actual page title with an expected value.

verifyTextPresent	1	Checks if a certain text is found within the page.
verifyElementPresent	1	Checks the presence of a certain element.
verifyTable	2	Compares the contents of a table with expected values.
waitForPageToLoad	1	Pauses execution until the page is loaded completely.
waitForElementPresent	1	Pauses execution until the specified element becomes present.

Create a Script Manually with Firebug

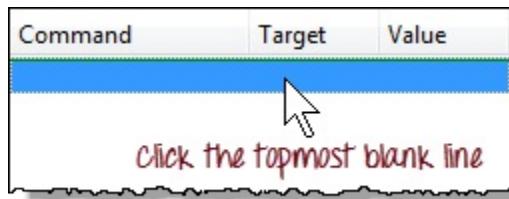
Now, we shall recreate the same test case manually, by typing in the commands. This time, we will need to use Firebug.

Step 1

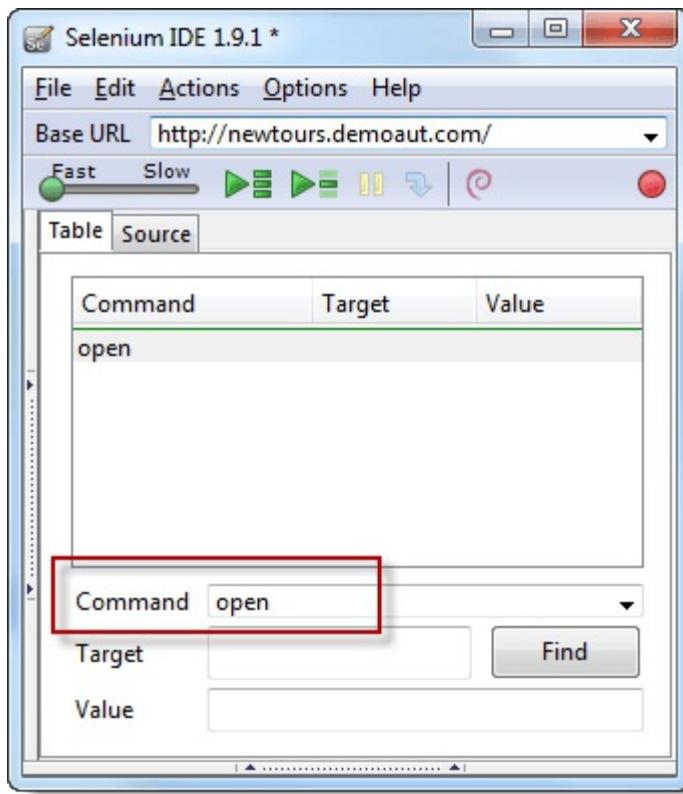
- Open Firefox and Selenium IDE.
- Type the base URL (<http://demo.guru99.com/test/newtours/>).
- The record button should be OFF.



Step 2: Click on the topmost blank line in the Editor.

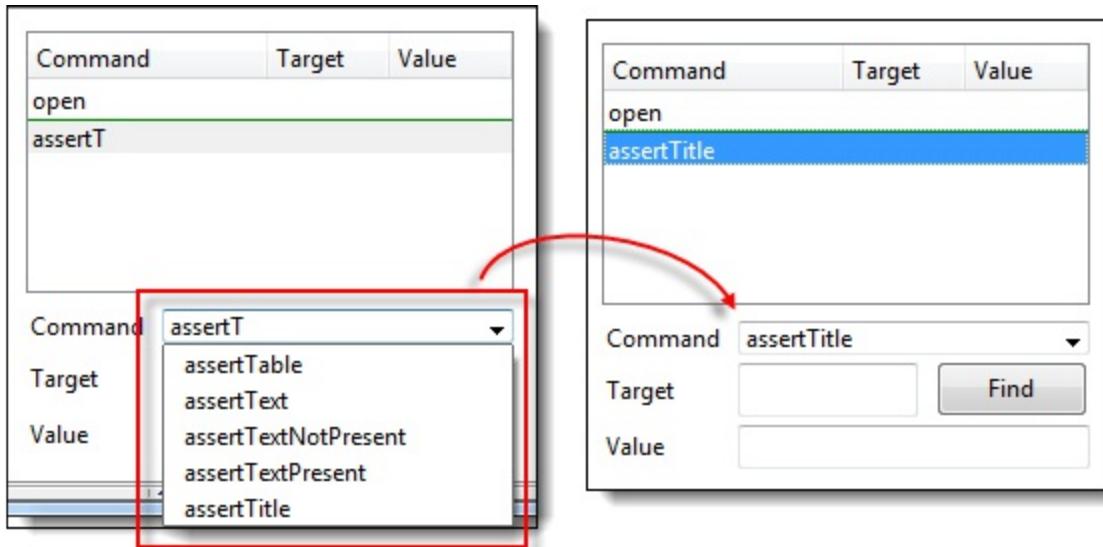


Type "open" in the Command text box and press Enter.



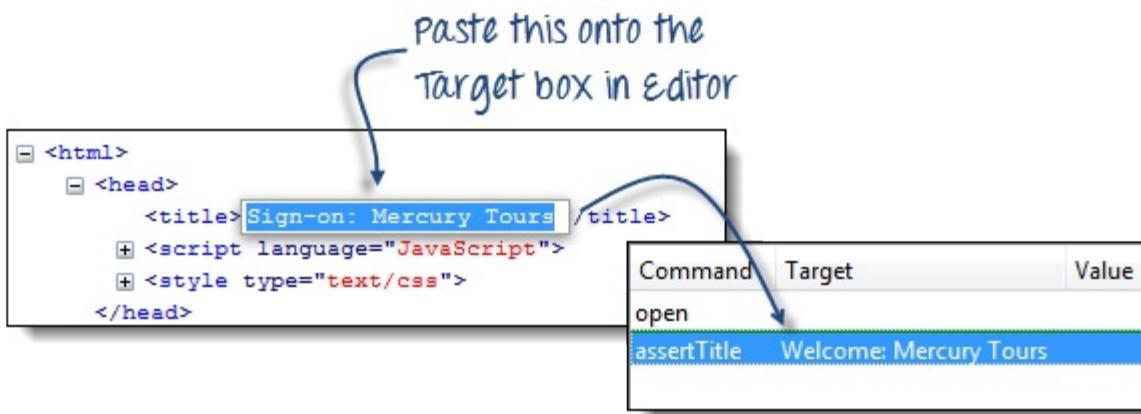
Step 3

- Navigate Firefox to our base URL and activate Firebug
- In the Selenium IDE Editor pane, select the second line (the line below the "open" command) and create the second command by typing "assertTitle" on the Command box.
- Feel free to use the autocomplete feature.



Step 4

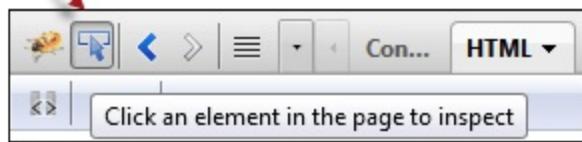
- In Firebug, expand the <head> tag to display the <title> tag.
- Click on the value of the <title> tag (which is "Welcome: Mercury Tours") and paste it onto the Target field in the Editor.



Step 5

- To create the third command, click on the third blank line in the Editor and key-in "type" on the Command text box.
- In Firebug, click on the "Inspect" button.

This is the "Inspect" button.



Click on the User Name text box. Notice that Firebug automatically shows you the HTML code for that element.

A screenshot of the Firebug interface. On the left, there's a preview of a travel website for Aruba. On the right, there's a login form with fields for 'User Name' and 'Password'. A red circle highlights the 'User Name' input field. A red arrow points from this field to the Firebug HTML panel below. The HTML panel shows the DOM structure and the specific `<input type="text" size="10" name="username">` line is highlighted in blue, indicating it's selected. A tooltip on the right says: 'This element has no style rules. You can [create a rule](#) for it.'

Step 6

Notice that the User Name text box does not have an ID, but it has a NAME attribute. We shall, therefore, use its NAME as the locator. Copy the NAME value and paste it onto the Target field in Selenium IDE.

A screenshot of the Selenium IDE interface. The 'Target' field contains the copied HTML code: `tr < tbody < table < td < tr < tbody < table < td < tr < tbody < table < div + <td align="right"> - <td width="112"> <input type="text" size="10" name='username'> </td> </tr>`. A red arrow points from the 'User Name' field in the Firebug screenshot above to this 'Target' field. Handwritten text at the bottom of the screenshot says: 'Paste this onto the "Target" field in Selenium IDE'.

Still in the Target text box, prefix "userName" with "name=", indicating that Selenium IDE should target an element whose NAME attribute is "userName."

Command	Target	Value
open		
assertTitle	exact:Welcome: Mercury Tours	
type	name=userName	

Type "invalidUN" in the Value text box of Selenium IDE. Your test script should now look like the image below. We are done with the third command. Note: Instead of invalidUN, you may enter any other text string. But Selenium IDE is case sensitive, and you type values/attributes exactly like in the application.

Command	Target	Value
open		
assertTitle	Welcome: Mercury Tours	
type	name=userName	invalidUN

Step 7

- To create the fourth command, key-in "type" on the Command text box.
- Again, use Firebug's "Inspect" button to get the locator for the "Password" text box.

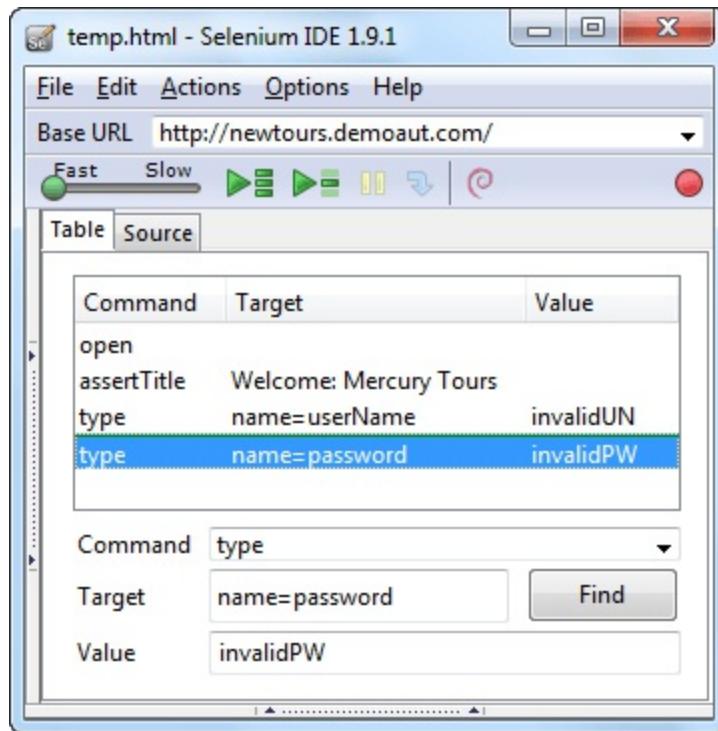
The Password text box only has the NAME attribute so we will use it as our locator

```

<tr>
  <td align="right">
    <td width="112">
      <input type="password" size="10" name='password'>
    </td>
  </tr>
  <tr>

```

- Paste the NAME attribute ("password") onto the Target field and prefix it with "name="
- Type "invalidPW" in the Value field in Selenium IDE. Your test script should now look like the image below.



Step 8

- For the fifth command, type "clickAndWait" on the Command text box in Selenium IDE.
- Use Firebug's "Inspect" button to get the locator for the "Sign In" button.

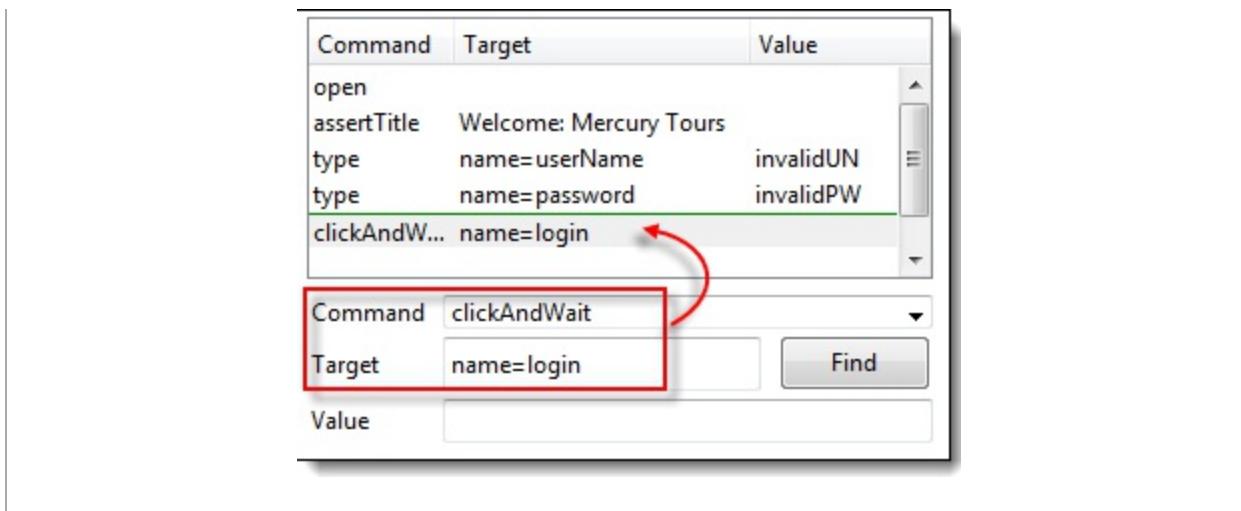
```

<td width="112">
    <div align="center">
        <input width="58" type="image" height="17" border="0"
            alt="Sign-In" src="/images
            /btn_signin.gif" value="Login" name='login'>
    </div>
</td>

```

Again, the only available locator is the NAME attribute so this will be the one that we shall use.

- Paste the value of the NAME attribute ("login") onto the Target text box and prefix it with "name=".
- Your test script should now look like the image below.



Step 9: Save the test case in the same way as we did in the previous section.

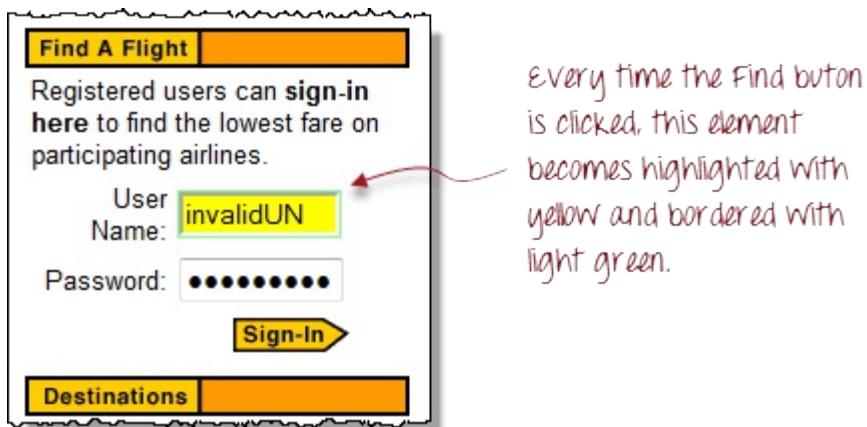
Using the Find Button

The Find button in Selenium IDE is used to verify if what we had put in the Target text box is indeed the correct UI element.

Let us use the Invalid_login test case that we created in the previous sections. Click on any command with a Target entry, say, the third command.

Command	Target	Value
open		
assertTitle	Welcome: Mercury Tours	
type	name=userName	invalidUN
type	name=password	invalidPW
clickAndW...	name=login	
Command	type	
Target	name=userName	
Value	invalidUN	

Click on the Find button. Notice that the User Name text box within the Mercury Tours page becomes highlighted for a second.



This indicates that Selenium IDE was able to detect and access the expected element correctly. If the Find button highlighted a different element or no element at all, then there must be something wrong with your script.

Execute Command

This allows you to execute any single command without running the whole test case. Just click on the line you wish to execute and then either click on "Actions > Execute this command"

from the menu bar or simply press "X" on your keyboard.

Step 1. Make sure that your browser is on the Mercury Tours homepage. Click on the command you wish to execute. In this example, click on the "type | userName | invalidUN" line.

Command	Target	Value
open	/	
type	userName	invalidUN
type	password	invalidPW
clickAndWait	login	
verifyTitle	Sign-on: Mercury Tours	

Step 2. Press "X" on your keyboard.

Step 3. Observe that the text box for username becomes populated with the text "invalidUN"



Executing commands this way is highly dependent on the page that Firefox is currently displaying. This means that if you try the example above with the Google homepage displayed instead of Mercury Tours', then your step will fail because there is no text box with a "userName" attribute within Google's homepage.

Start point

A start point is an indicator that tells Selenium IDE which lines the execution will start. Its shortcut key is "S".



Command	Target	Value
open	/	
type	userName	invalidUN
type	password	invalidPW
clickAndWait	login	
verifyTitle	Sign-on: Mercury Tours	
clickAndWait	link=Home	
verifyTitle	Welcome: Mercury Tours	

In the example above, playback will start on the third line (type | password | invalidPW). **You can only have one start point in a single test script.**

The start point is similar to Execute Command in such that they are dependent on the currently displayed page. The start point will fail if you are on the wrong page.

Breakpoints

Breakpoints are indicators that tell Selenium IDE where to automatically pause the test. **The shortcut key is "B".**



Command	Target	Value
open	/	
type	userName	invalidUN
type	password	invalidPW
clickAndWait	login	
verifyTitle	Sign-on: Mercury Tours	
clickAndWait	link=Home	
verifyTitle	Welcome: Mercury Tours	

The yellow highlight means that the current step is pending. This

proves that Selenium IDE has paused execution on that step. **You can have multiple breakpoints in one test case.**

Step

It allows you to execute succeeding commands one at a time after pausing the test case. Let us use the scenario in the previous section "Breakpoints."

Command	Target	Value
open	/	
type	userName	invalidUN
type	password	invalidPW
clickAndWait	login	
verifyTitle	Sign-on: Mercury Tours	
clickAndWait	link=Home	
verifyTitle	Welcome: Mercury Tours	

Command	Target	Value
open	/	
type	userName	invalidUN
type	password	invalidPW
clickAndWait	login	
verifyTitle	Sign-on: Mercury Tours	
clickAndWait	link=Home	
verifyTitle	Welcome: Mercury Tours	

Before clicking "Step."

The test case pauses at the line "clickAndWait | login".

After clicking "Step."

The "clickAndWait | login" line is run and pauses to the next command (verifyTitle | Sign-on: Mercury Tours).

Notice that the next line is paused even though there is no breakpoint there. This is the main purpose of the Step feature - it executes the succeeding commands one at a time to give you more time to inspect the outcome after each step.

Important Things to Note When Using Other Formats in Source View



Selenium IDE works well only with HTML - other formats are still in experimental mode. It is **NOT advisable** to create or edit tests using other formats in Source View because there is still a lot of work needed to make it stable. Below are the known bugs as of version 1.9.1.

- You will not be able to perform playback nor switch back to Table View unless you revert to HTML.
- The only way to add commands safely on the source code is by recording them.
- When you modify the source code manually, all of it will be lost when you switch to another format.
- Though you can save your test case while in Source View, Selenium IDE will not be able to open it.

The recommended way to convert Selenese tests is to use the "Export Test Case As..." option under the File menu, and not through the Source View.

Summary

- Test scripts can be created either by recording or typing the commands and parameters manually.
- When creating scripts manually, Firebug is used to get the locator.
- The Find button is used to check that the command is able to access the correct element.
- Table View displays a test script in tabular form while Source View displays it in HTML format.
- Changing the Source View to a non-HTML format is still experimental.
- Do not use the Source View in creating tests in other formats. Use the Export features instead.
- Parameters are not required all the time. It depends upon the command.
- There are three types of commands:
 - Actions - directly interacts with page elements
 - Accessors - "reads" an element property and stores it in a variable
 - Assertions - compares an actual value with an expected one
- Assertions have three types:
 - Assert - upon failure, succeeding steps are no longer executed
 - Verify - upon failure, succeeding steps are still executed.
 - WaitFor - passes if the specified condition becomes true within the timeout period; otherwise, it will fail
- The most common commands are:
 - open
 - click/clickAndWait
 - type/typeKeys
 - verifyTitle/assertTitle

- verifyTextPresent
- verifyElementPresent
- verifyTable
- waitForPageToLoad
- waitForElementPresent

Chapter 5: How to use Locators in Selenium IDE

What are Locators?

Locator is a command that tells Selenium IDE which GUI elements (say Text Box, Buttons, Check Boxes etc) its needs to operate on.

Identification of correct GUI elements is a prerequisite to creating an automation script. But accurate identification of GUI elements is more difficult than it sounds. Sometimes, you end up working with incorrect GUI elements or no elements at all! Hence, Selenium provides a number of Locators to precisely locate a GUI element

The different types of Locators in Selenium IDE

- ID
- Name
- Link Text
- CSS Selector
 - Tag and ID
 - Tag and class
 - Tag and attribute
 - Tag, class, and attribute
 - Inner text
- DOM (Document Object Model)
 - getElementById
 - getElementsByName
 - dom:name

- dom: index
- XPath

There are commands that do not need a locator (such as the "open" command). However, most of them do need Locators.

The choice of locator depends largely on your Application Under Test. In this tutorial, we will toggle between Facebook, new tours.demoaut on the basis of locators that these applications support. Likewise in your Testing project, you will select any of the above-listed locators based on your application support.

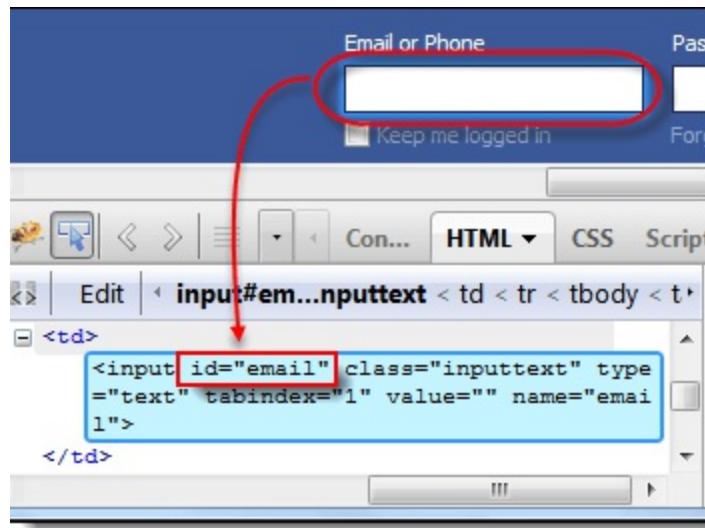
Locating by ID

This is the most common way of locating elements since ID's are supposed to be unique for each element.

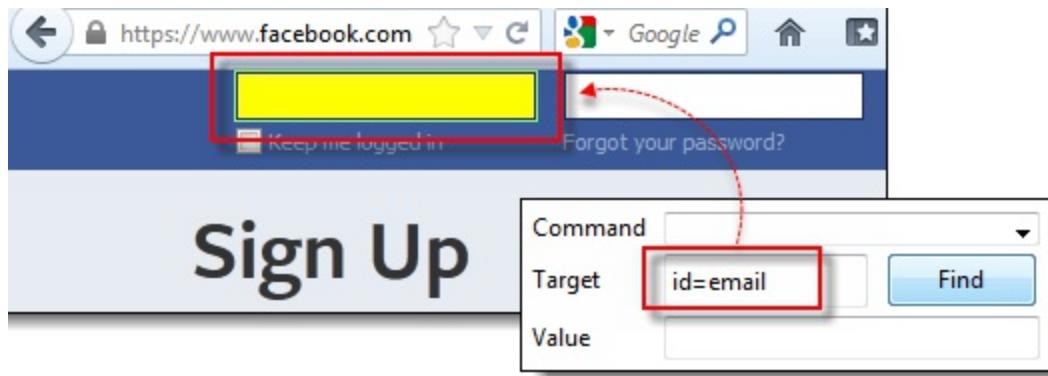
Target Format: `id=id of the element`

For this example, we will use Facebook as our test app because Mercury Tours do not use ID attributes.

Step 1. Since this tutorial was created, Facebook has changed their Login Page Design. Use this demo page <http://demo.guru99.com/test/facebook.html> for testing. Inspect the "Email or Phone" text box using Firebug and take note of its ID. In this case, the ID is "email."



Step 2. Launch Selenium IDE and enter "id=email" in the Target box. Click the Find button and notice that the "Email or Phone" text box becomes highlighted with yellow and bordered with green, meaning, Selenium IDE was able to locate that element correctly.



Locating by Name

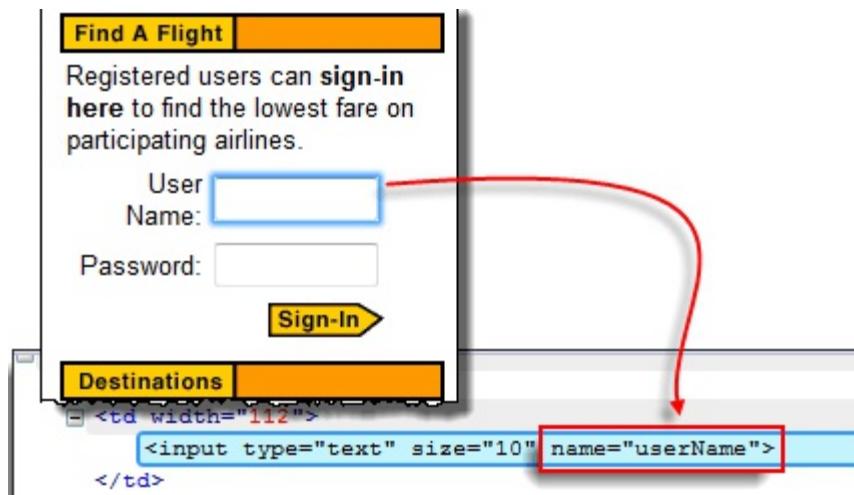
Locating elements by name are very similar to locating by ID, except that we use the "**name=**" prefix instead.

Target Format: `name=name of the element`

In the following demonstration, we will now use Mercury Tours

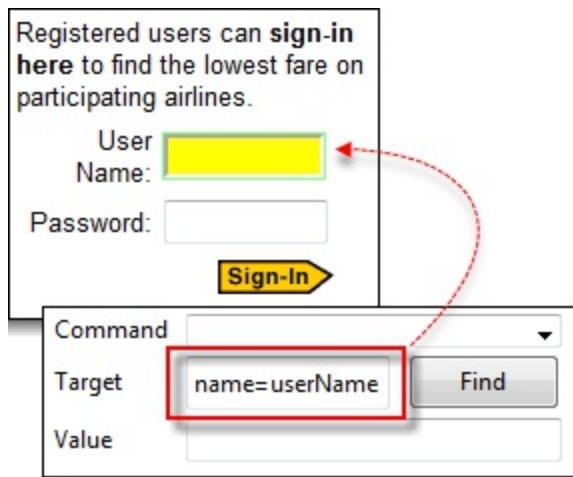
because all significant elements have names.

Step 1. Navigate to <http://demo.guru99.com/test/newtours/> and use Firebug to inspect the "User Name" text box. Take note of its name attribute.



Here, we see that the element's name is "userName".

Step 2. In Selenium IDE, enter "name=userName" in the Target box and click the Find button. Selenium IDE should be able to locate the User Name text box by highlighting it.



Locating by Name using Filters

Filters can be used when multiple elements have the same name.

Filters are additional attributes used to distinguish elements with the same name.

Target Format: name=*name_of_the_element* filter=*value_of_filter*

Let's see an example -

Step 1. Log on to Mercury Tours using "tutorial" as the username and password. It should take you to the Flight Finder page shown below.

Step 2. Using Firebug, notice that the Round Trip and One Way radio buttons have the same name "tripType." However, they have different VALUE attributes so we can use each of them as our filter.

Firebug

Flight Details

Type: Round Trip One Way

Passengers: 1

Departing From: Acapulco

Mercury Tours Flight Finder page

Step 3.

- We are going to access the One Way radio button first. Click the first line on the Editor.
- In the Command box of Selenium IDE, enter the command "click".
- In the Target box, enter "name=tripType value=oneway". The "value=oneway" portion is our filter.

Command	Target	Value
click	name=tripType value=oneway	

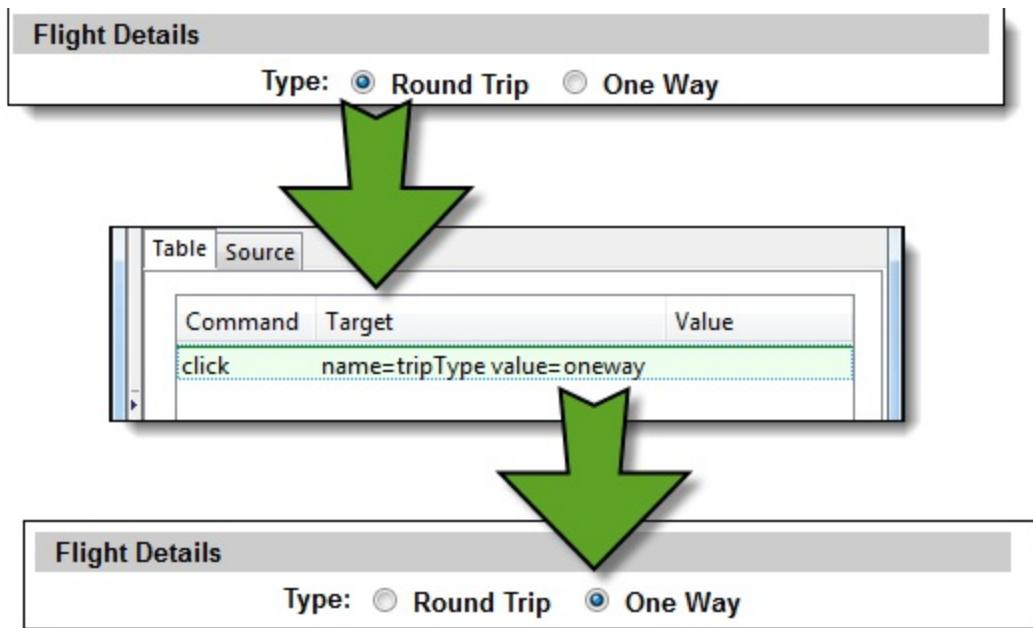
Command	click
Target	name=tripType value=oneway
Value	

Step 4. Click the Find button and notice that Selenium IDE is able to

highlight the One Way radio button with green - meaning that we are able to access the element successfully using its VALUE attribute.



Step 5. Press the "X" key in your keyboard to execute this click command. Notice that the One Way radio button became selected.



You can do the exact same thing with the Round Trip radio button, this time, using "name=tripType value=roundtrip" as your target.

Locating by Link Text

This type of locator applies only to hyperlink texts. We access the link by prefixing our target with "link=" and then followed by the hyperlink text.

Target Format: link=*link_text*

In this example, we shall access the "REGISTER" link found on the Mercury Tours homepage.

Step 1.

- First, make sure that you are logged off from Mercury Tours.
- Go to Mercury Tours homepage.

Step 2.

- Using Firebug, inspect the "REGISTER" link. The link text is found between and tags.
- In this case, our link text is "REGISTER". Copy the link text.



Step 3. Copy the link text in Firebug and paste it onto Selenium IDE's Target box. Prefix it with "link=".

Command	Target	Value
	link=REGISTER	

Command	Target	Value
	link=REGISTER	
		<input type="button" value="Find"/>

Step 4. Click on the Find button and notice that Selenium IDE was able to highlight the REGISTER link correctly.



Step 5. To verify further, enter "clickAndWait" in the Command box and execute it. Selenium IDE should be able to click on that REGISTER link successfully and take you to the Registration page shown below.

A screenshot of a web page titled 'REGISTER'. The page has a yellow header bar with the word 'REGISTER' and a search icon. Below the header, there is a message: 'To create your account, we'll need some basic information about you. This information will be used to send reservation confirmation emails, mail tickets when needed and contact you if your travel arrangements change. Please fill in the form completely.' There is a section titled 'Contact Information' with fields for 'First Name', 'Last Name', and 'Phone'. The 'First Name' field is currently empty.

Locating by CSS Selector

CSS Selectors are string patterns used to identify an element based on a combination of HTML tag, id, class, and attributes. Locating by CSS Selector is more complicated than the previous methods, but it is the most common locating strategy of advanced Selenium users because it can access even those elements that have no ID or name.

CSS Selectors have many formats, but we will only focus on the most common ones.

- Tag and ID
- Tag and class
- Tag and attribute
- Tag, class, and attribute
- Inner text

When using this strategy, we always prefix the Target box with "css=" as will be shown in the following examples.

Locating by CSS Selector - Tag and ID

Again, we will use Facebook's Email text box in this example. As you can remember, it has an ID of "email," and we have already accessed it in the "Locating by ID" section. This time, we will use a CSS Selector with ID in accessing that very same element.

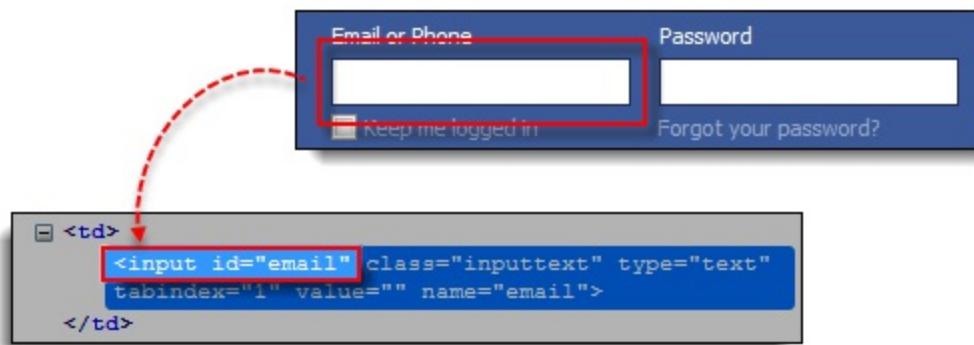
Syntax	Description
css= <i>tag#id</i>	<ul style="list-style-type: none"> • <i>tag</i> = the HTML tag of the element being accessed

- # = the hash sign. This should always be present when using a CSS Selector with ID
- id = the ID of the element being accessed

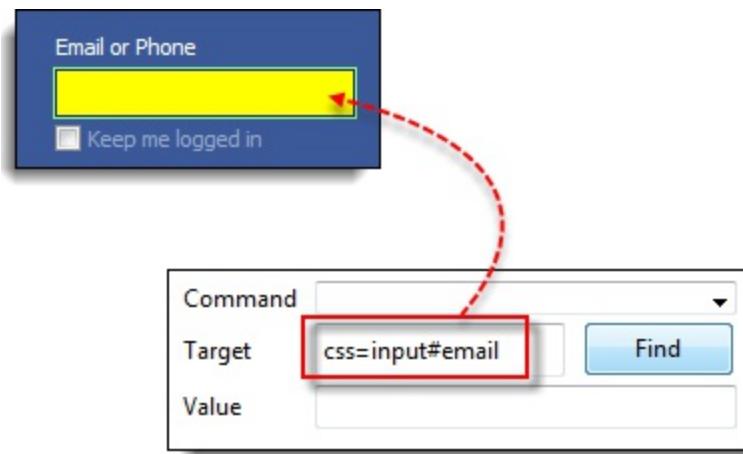
Keep in mind that the ID is always preceded by a hash sign (#).

Step 1. Navigate to www.facebook.com. Using Firebug, examine the "Email or Phone" text box.

At this point, take note that the HTML tag is "input" and its ID is "email". So our syntax will be "css=input#email".



Step 2. Enter "css=input#email" into the Target box of Selenium IDE and click the Find button. Selenium IDE should be able to highlight that element.



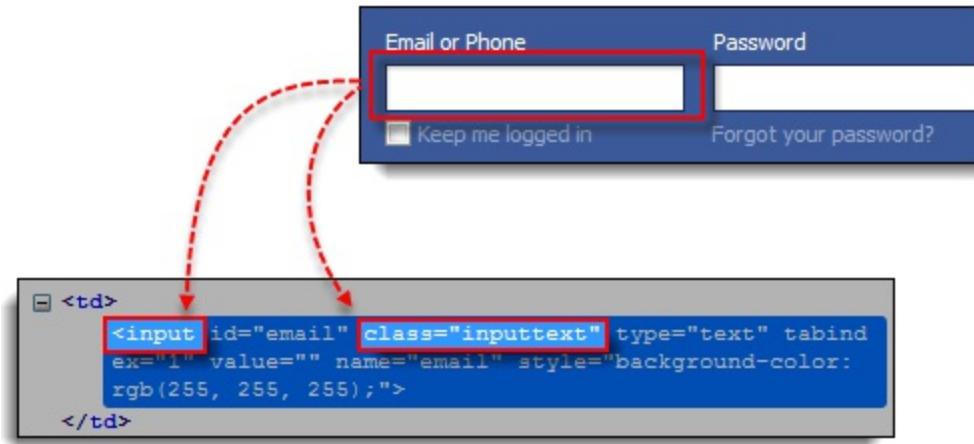
Locating by CSS Selector - Tag and Class

Locating by CSS Selector using an HTML tag and a class name is similar to using a tag and ID, but in this case, a dot (.) is used instead of a hash sign.

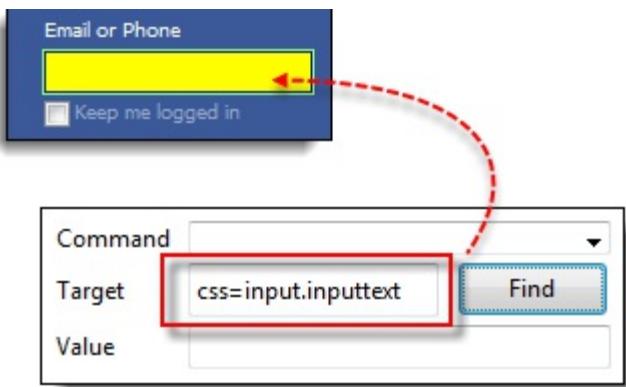
Syntax	Description
<code>css=tag.class</code>	<ul style="list-style-type: none"> • tag = the HTML tag of the element being accessed • . = the dot sign. This should always be present when using a CSS Selector with class • class = the class of the element being accessed

Step 1. Go to the demo page

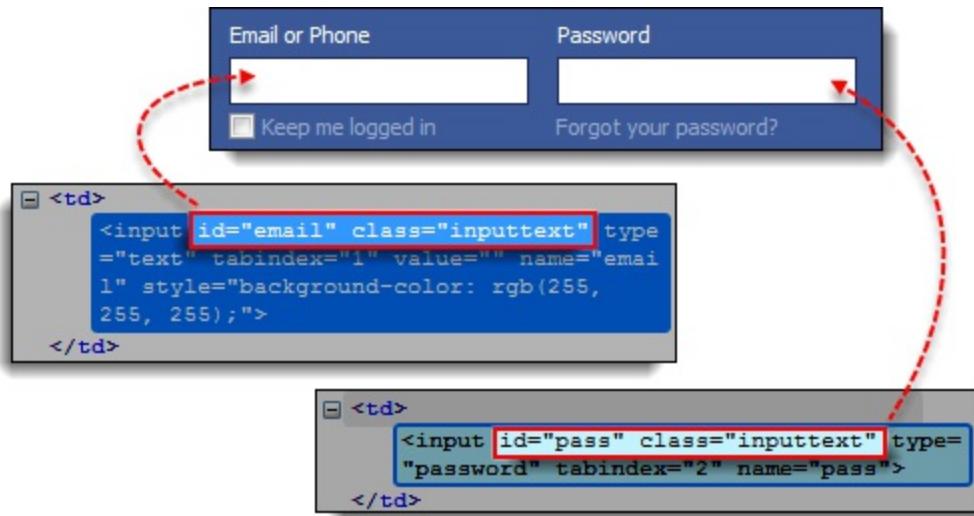
<http://demo.guru99.com/test/facebook.html> and use Firebug to inspect the "Email or Phone" text box. Notice that its HTML tag is "input" and its class is "inputtext."



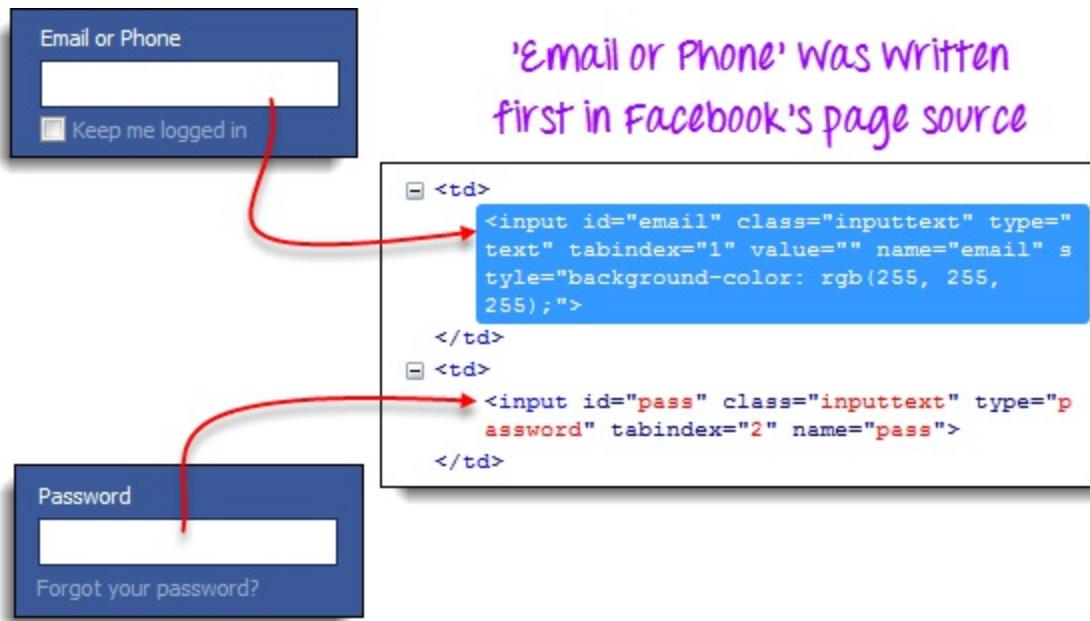
Step 2. In Selenium IDE, enter "css=input.inputtext" in the Target box and click Find. Selenium IDE should be able to recognize the Email or Phone text box.



Take note that when multiple elements have the same HTML tag and name, only the first element in source code will be recognized. Using Firebug, inspect the Password text box in Facebook and notice that it has the same name as the Email or Phone text box.



The reason why only the Email or Phone text box was highlighted in the previous illustration is that it comes first in Facebook's page source.



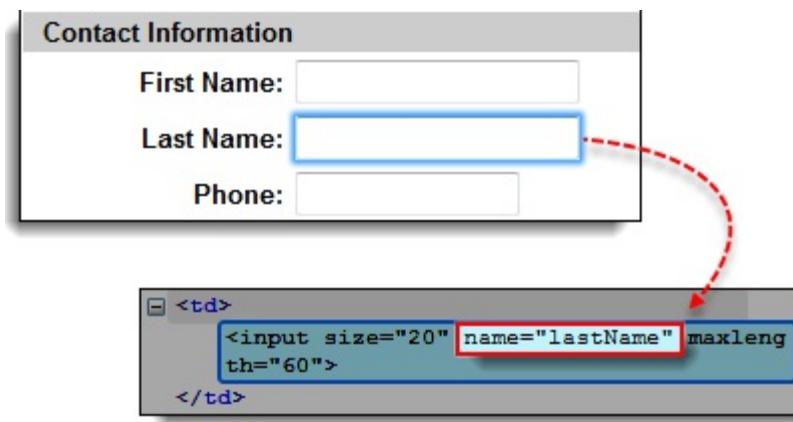
Locating by CSS Selector - Tag and Attribute

This strategy uses the HTML tag and a specific attribute of the element

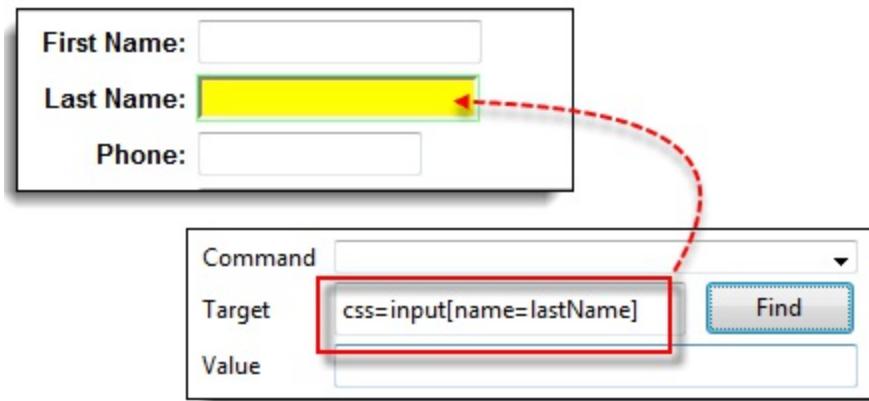
to be accessed.

Syntax	Description
css=tag[attribute=value]	<ul style="list-style-type: none"> • tag = the HTML tag of the element being accessed • [and] = square brackets within which a specific attribute and its corresponding value will be placed • attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID. • value = the corresponding value of the chosen attribute.

Step 1. Navigate to Mercury Tours' Registration page (<http://demo.guru99.com/test/newtours/register.php>) and inspect the "Last Name" text box. Take note of its HTML tag ("input" in this case) and its name ("lastName").



Step 2. In Selenium IDE, enter "css=input[name=lastName]" in the Target box and click Find. Selenium IDE should be able to access the Last Name box successfully.



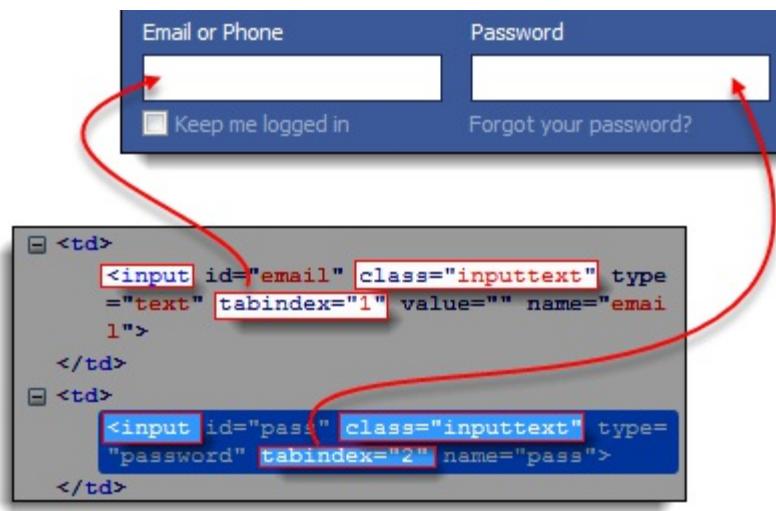
When multiple elements have the same HTML tag and attribute, only the first one will be recognized. This behavior is similar to locating elements using CSS selectors with the same tag and class.

Locating by CSS Selector - tag, class, and attribute

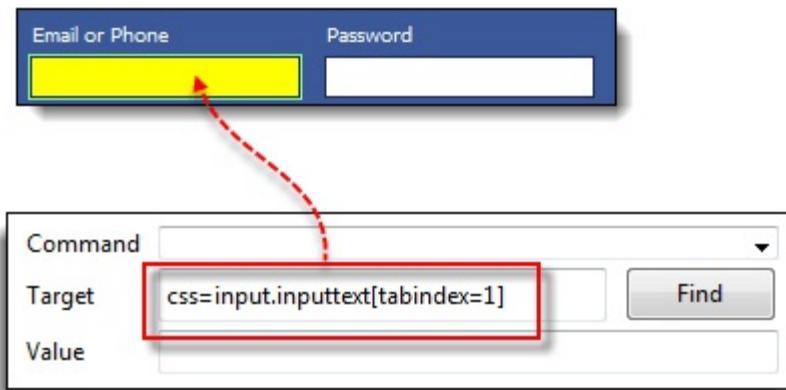
Syntax	Description
<code>css=tag.class[attribute=value]</code>	<ul style="list-style-type: none"> • tag = the HTML tag of the element being accessed • . = the dot sign. This should always be present when using a CSS Selector with class • class = the class of the element being accessed • [and] = square brackets within which a specific attribute and its corresponding value will be placed • attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID. • value = the corresponding value of the chosen attribute.

Step 1. Go to the demo page

<http://demo.guru99.com/test/facebook.html> and use Firebug to inspect the 'Email or Phone' and 'Password' input boxes. Take note of their HTML tag, class, and attributes. For this example, we will select their 'tabindex' attributes.

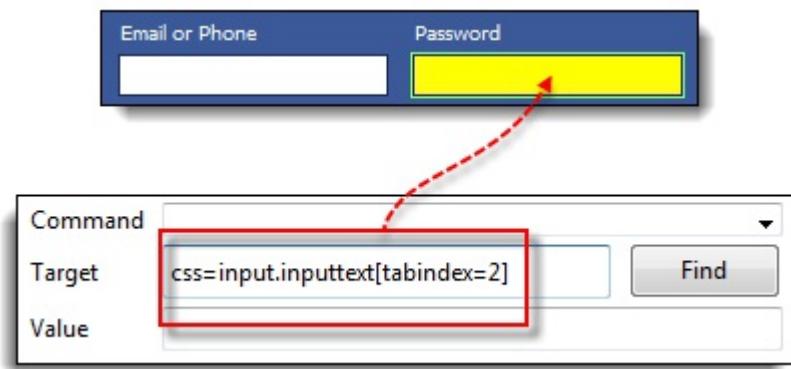


Step 2. We will access the 'Email or Phone' text box first. Thus, we will use a tabindex value of 1. Enter "css=input.inputtext[tabindex=1]" in Selenium IDE's Target box and click Find. The 'Email or Phone' input box should be highlighted.



Step 3. To access the Password input box, simply replace the value of the tabindex attribute. Enter "css=input.inputtext[tabindex=2]" in the

Target box and click on the Find button. Selenium IDE must be able to identify the Password text box successfully.

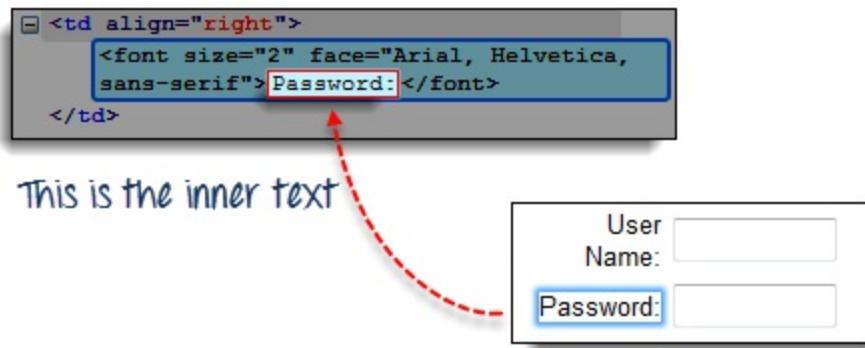


Locating by CSS Selector - inner text

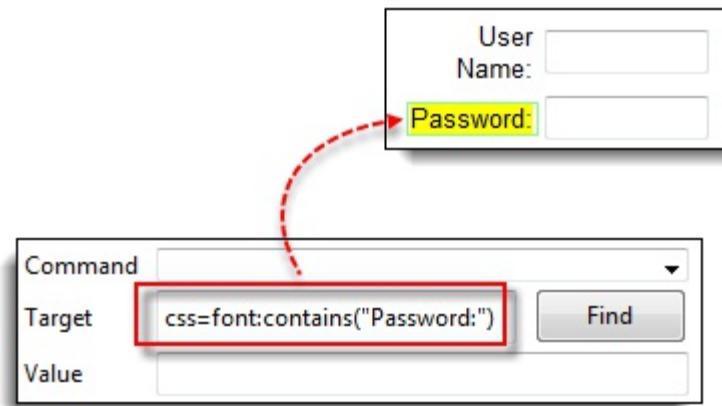
As you may have noticed, HTML labels are seldom given id, name, or class attributes. So, how do we access them? The answer is through the use of their inner texts. **Inner texts are the actual string patterns that the HTML label shows on the page.**

Syntax	Description
<code>css=tag:contains("inner text")</code>	<ul style="list-style-type: none"> • tag = the HTML tag of the element being accessed • inner text = the inner text of the element

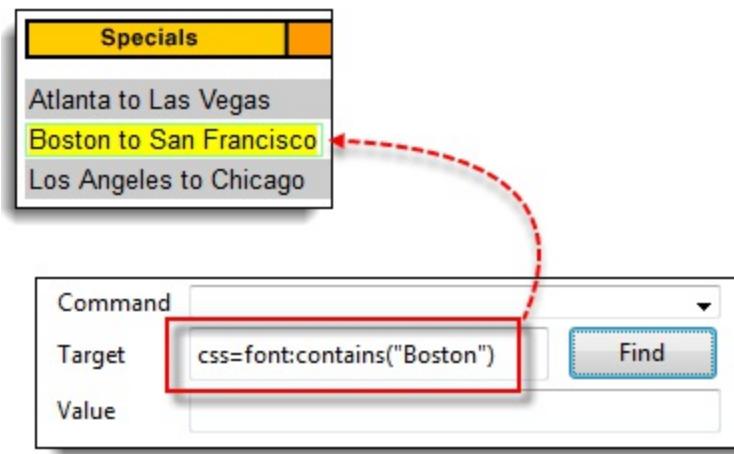
Step 1. Navigate to Mercury Tours' homepage (<http://demo.guru99.com/test/newtours/>) and use Firebug to investigate the "Password" label. Take note of its HTML tag (which is "font" in this case) and notice that it has no class, id, or name attributes.



Step 2. Type **css=font:contains("Password:")** into Selenium IDE's Target box and click Find. Selenium IDE should be able to access the Password label as shown in the image below.



Step 3. This time, replace the inner text with "Boston" so that your Target will now become "css=font:contains("Boston")". Click Find. You should notice that the "Boston to San Francisco" label becomes highlighted. This shows you that Selenium IDE can access a long label even if you just indicated the first word of its inner text.



Locating by DOM (Document Object Model)

The Document Object Model (DOM), in simple terms, is the way by which HTML elements are structured. Selenium IDE is able to use the DOM in accessing page elements. If we use this method, our Target box will always start with "dom=document..."; however, the "dom=" prefix is normally removed because Selenium IDE is able to automatically interpret anything that starts with the keyword "document" to be a path within the DOM anyway.

There are four basic ways to locate an element through DOM:

- getElementById
- getElementsByName
- dom:name (applies only to elements within a named form)
- dom:index

Locating by DOM - getElementById

Let us focus on the first method - using the getElementById method. The syntax would be:

Syntax	Description
document.getElementById("id of the element")	id of the element = this is the value of the ID attribute of the element to be accessed. This value should always be enclosed in a pair of parentheses ("").

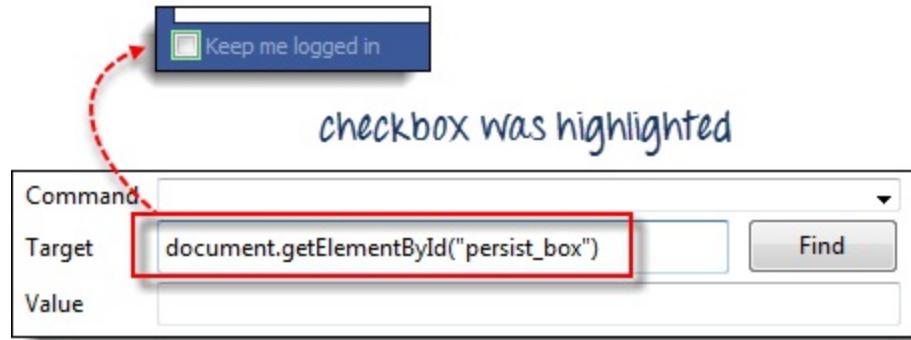
Step 1. Use this demo page

http://demo.guru99.com/test/facebook.html Navigate to it and use Firebug to inspect the "Keep me logged in" check box. Take note of its ID.



We can see that the ID we should use is "persist_box".

Step 2. Open Selenium IDE and in the Target box, enter "document.getElementById("persist_box")" and click Find. Selenium IDE should be able to locate the "Keep me logged in" check box. Though it cannot highlight the interior of the check box, Selenium IDE can still surround the element with a bright green border as shown below.



Locating by DOM - getElementsByName

The getElementById method can access only one element at a time, and that is the element with the ID that you specified. The getElementsByName method is different. It collects an array of elements that have the name that you specified. You access the individual elements using an index which starts at 0.

A photograph of a person's hand holding a single cherry. A dashed line extends from the word "getElementById" in red text above the hand to the word "id = \"userName\"". Another dashed line extends from the word "getElementsByName" below the hand to the word "UserName" in the code below.

getElementById

- It will get only one element for you.
- That element bears the ID that you specified inside the parentheses of getElementById().

getElementsByName



Syntax	Description
<code>document.getElementsByName("name") [index]</code>	<ul style="list-style-type: none"> • name = name of the element as defined by its 'name' attribute • index = an integer that indicates which element within getElementsByName's array will be used.

Step 1. Navigate to Mercury Tours' Homepage and login using "tutorial" as the username and password. Firefox should take you to the Flight Finder screen.

Step 2. Using Firebug, inspect the three radio buttons at the bottom portion of the page (Economy class, Business class, and First class radio buttons). Notice that they all have the same name which is "servClass".

The screenshot shows a 'Preferences' dialog box. At the top, there is a 'Service Class:' section with three radio buttons: 'Economy class' (selected), 'Business class', and 'First class'. Below this is a dropdown menu labeled 'Airline:' with the option 'No Preference'. Red arrows point from the 'Economy class' radio button and the 'No Preference' dropdown to the corresponding code elements in the Selenium IDE's DOM tree below.

```

<td>
  <font size="2">
    <input type="radio" value="Coach" name="servClass" checked="">
    <font face="Arial, Helvetica, sans-serif">
      Economy class
    <br>
    <input type="radio" value="Business" name="servClass">
    Business class
    <br>
    <input type="radio" value="First" name="servClass">
    First class
  </font>
</td>

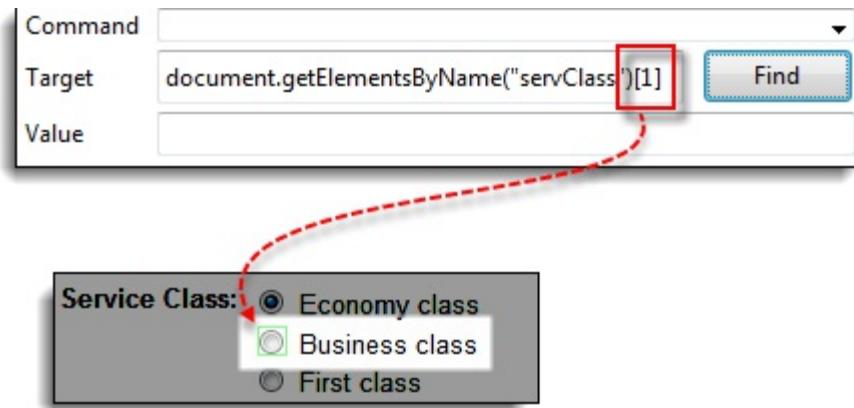
```

Step 3. Let us access the "Economy class" radio button first. Of all these three radio buttons, this element comes first, so it has an index of 0. In Selenium IDE, type "document.getElementsByName("servClass")[0]" and click the Find button. Selenium IDE should be able to identify the Economy class radio button correctly.

The screenshot shows the Selenium IDE interface with the 'Command' dropdown set to 'document.getElementsByName("servClass")[0]'. A red box highlights the 'Target' field. An arrow points from this field to the 'Economy class' radio button in the 'Service Class:' section of the preferences dialog at the bottom.

Step 4. Change the index number to 1 so that your Target will now

become `document.getElementsByName("servClass")[1]`. Click the Find button, and Selenium IDE should be able to highlight the "Business class" radio button, as shown below.

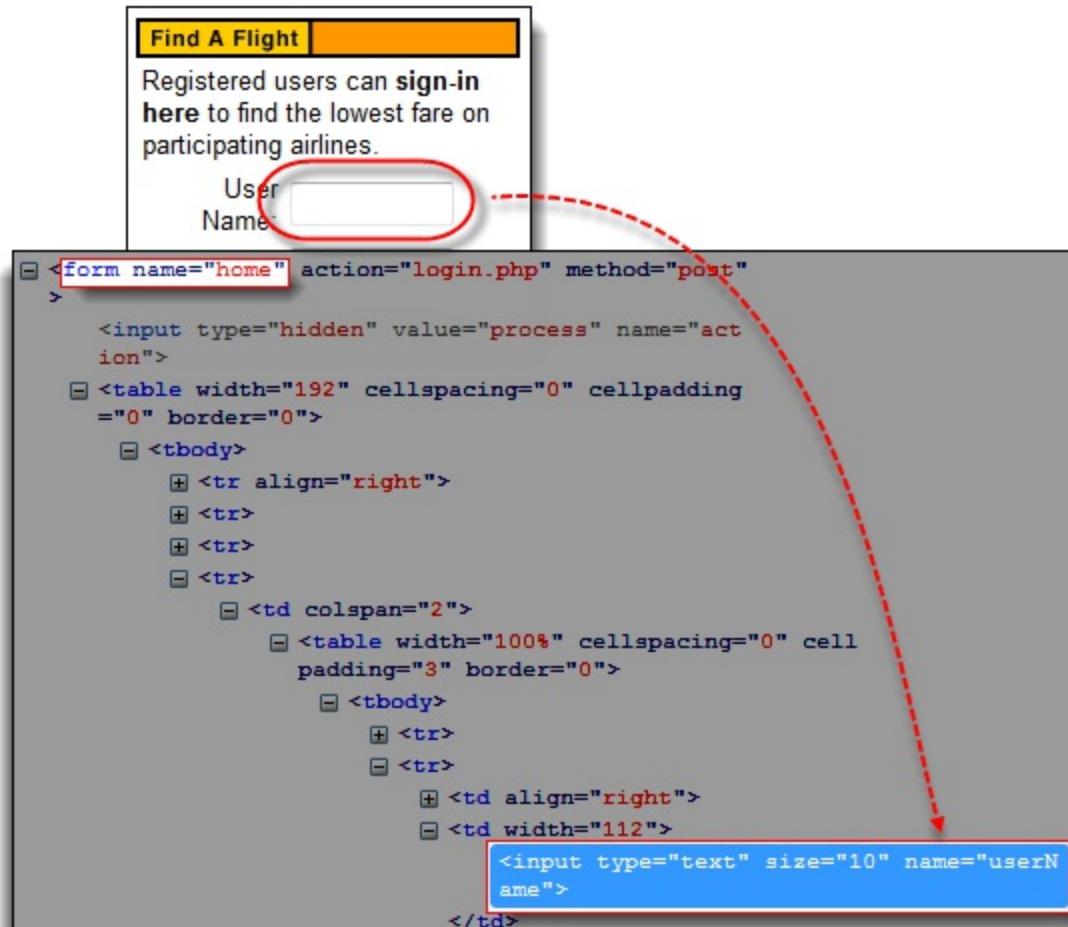


Locating by DOM - dom:name

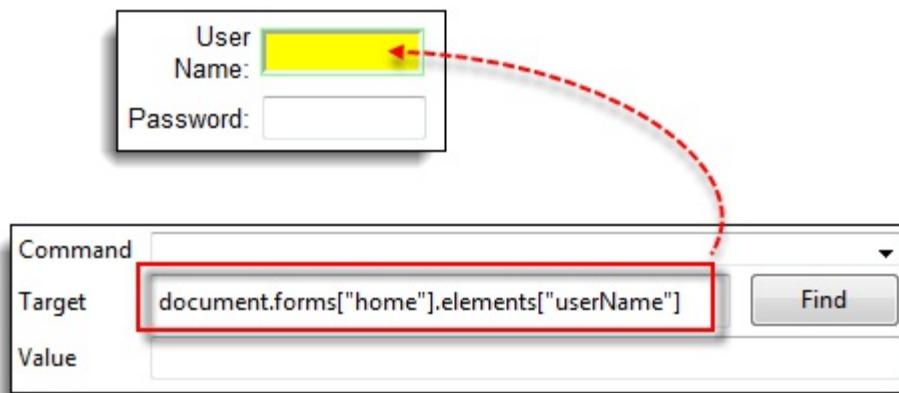
As mentioned earlier, this method will only apply if the element you are accessing is contained within a named form.

Syntax	Description
<code>document.forms["name of the form"].elements["name of the element"]</code>	<ul style="list-style-type: none"> • name of the form = the value of the name attribute of the form tag that contains the element you want to access • name of the element = the value of the name attribute of the element you wish to access

Step 1. Navigate to Mercury Tours homepage (<http://demo.guru99.com/test/newtours/>) and use Firebug to inspect the User Name text box. Notice that it is contained in a form named "home."



Step 2. In Selenium IDE, type "document.forms["home"].elements["userName"]" and click the Find button. Selenium IDE must be able to access the element successfully.



Locating by DOM - dom:index

This method applies even when the element is not within a named form because it uses the form's index and not its name.

Syntax	Description
<code>document.forms[<i>index of the form</i>].elements[<i>index of the element</i>]</code>	<ul style="list-style-type: none">• index of the form = the index number (starting at 0) of the form with respect to the whole page• index of the element = the index number (starting at 0) of the element with respect to the whole form that contains it

We shall access the "Phone" text box within Mercury Tours Registration page. The form in that page has no name and ID attribute, so this will make a good example.

Step 1. Navigate to Mercury Tours Registration page and inspect the Phone text box. Notice that the form containing it has no ID and name attributes.

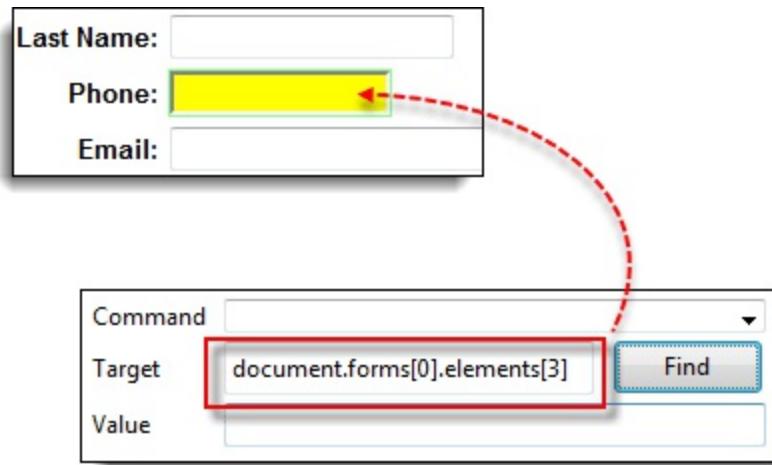
No name and id attributes. This is the only form in the page, so its index will be 0

```

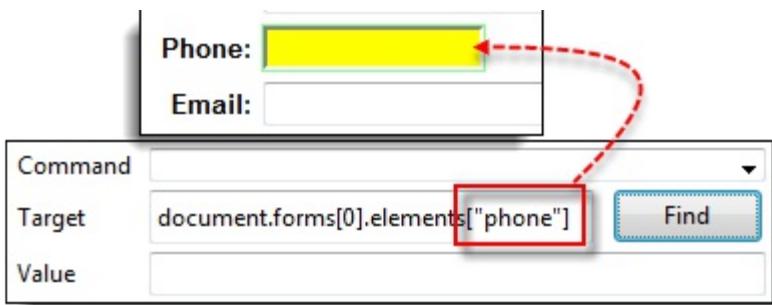
<form action="mercurycreate_account.php" method="post">
    <input type="hidden" value="process" name="mercury" style="background-color: rgb(255, 255, 255);"/>
    <table width="100%" cellpadding="2" border="0">
        <tbody>
            <tr bgcolor="#CCCCCC">
                <td align="right"></td>
                <td>
                    <input size="20" name="firstName" maxlength="60">
                </td>
            </tr>
            <tr>
                <td align="right"></td>
                <td>
                    <input size="20" name="lastName" maxlength="60">
                </td>
            </tr>
            <tr>
                <td align="right"></td>
                <td>
                    <input size="15" name="phone" maxlength="20" style="background-color: rgb(255, 255, 255);">
                </td>
            </tr>
        </tbody>
    </table>

```

Step 2. Enter "document.forms[0].elements[3]" in Selenium IDE's Target box and click the Find button. Selenium IDE should be able to access the Phone text box correctly.



Step 3. Alternatively, you can use the element's name instead of its index and obtain the same result. Enter "document.forms[0].elements['phone']" in Selenium IDE's Target box. The Phone text box should still become highlighted.



Locating by XPath

XPath is the language used when locating XML (Extensible Markup Language) nodes. Since HTML can be thought of as an implementation of XML, we can also use XPath in locating HTML elements.

Advantage: It can access almost any element, even those without class, name, or id attributes.

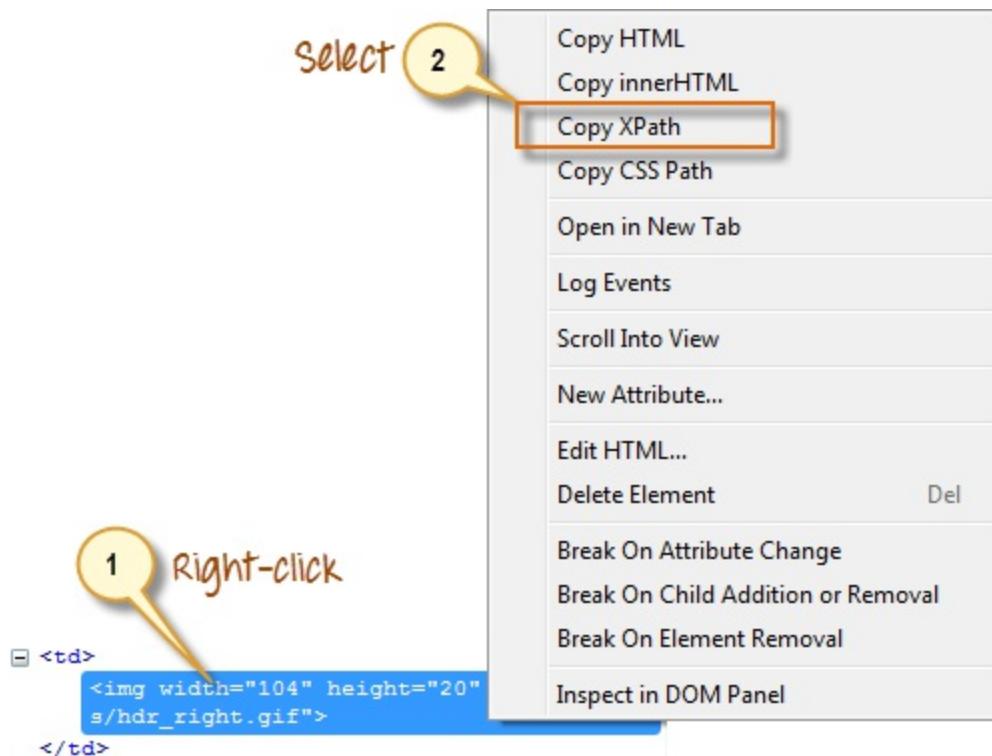
Disadvantage: It is the most complicated method of identifying elements because of too many different rules and considerations.

Fortunately, Firebug can automatically generate XPath locators. In the following example, we will access an image that cannot possibly be accessed through the methods we discussed earlier.

Step 1. Navigate to Mercury Tours Homepage and use Firebug to inspect the orange rectangle to the right of the yellow "Links" box. Refer to the image below.



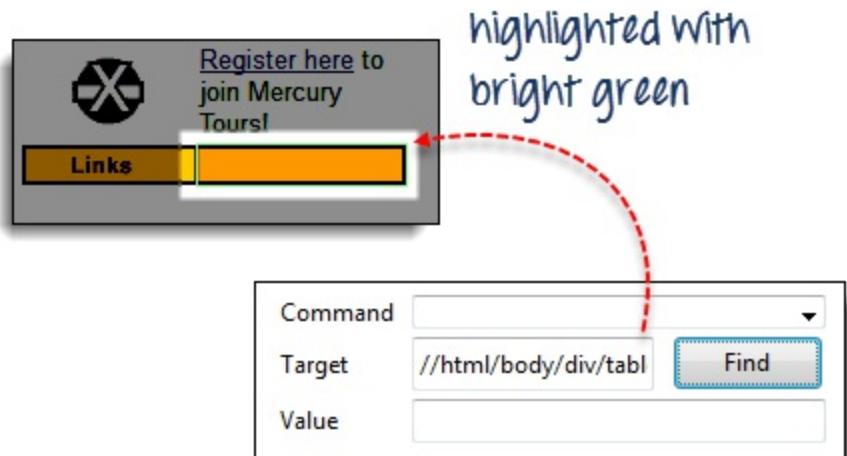
Step 2. Right click on the element's HTML code and then select the "Copy XPath" option.



Step 3. In Selenium IDE, type one forward slash "/" in the Target box then paste the XPath that we copied in the previous step. **The entry in your Target box should now begin with two forward slashes "://".**



Step 4. Click on the Find button. Selenium IDE should be able to highlight the orange box as shown below.



Summary

Syntax for Locator Usage

Method	Target Syntax	Example
By ID	<code>id=id_of_the_element</code>	<code>id=email</code>
By Name	<code>name=name_of_the_element</code>	<code>name=userName</code>
By Name Using Filters	<code>name=name_of_the_element name=tripType value=oneway filter=value_of_filter</code>	
By Link Text	<code>link=link_text</code>	<code>link=REGISTER</code>
Tag and ID	<code>css=tag#id</code>	<code>css=input#email</code>
Tag and Class	<code>css=tag.class</code>	<code>css=input.inputtext</code>
Tag and Attribute	<code>css=tag[attribute=value]</code>	<code>css=input[name=lastName]</code>
Tag, Class, and Attribute	<code>css=tag.class[attribute=value]</code>	<code>css=input.inputtext[tabindex=1]</code>

Chapter 6: How to enhance a script using Selenium IDE

In this tutorial, we look at commands that will make your automation script more intelligent and complete.

Verify Presence of an Element

We can use following two commands to verify the presence of an element:

- **verifyElementPresent** - returns TRUE if the specified element was FOUND in the page; FALSE if otherwise
- **verifyElementNotPresent** - returns TRUE if the specified element was NOT FOUND anywhere in the page; FALSE if it is present.

The test script below verifies that the UserName text box is present within the Mercury Tours homepage while the First Name text box is not. The First Name text box is actually an element present in the Registration page of Mercury Tours, not in the homepage.
Verify Presence of a Certain Text

This step failed, thereby confirming that the element with name=firstname is not indeed present in the page

Command	Target	Value
verifyElementPresent	name=userName	
verifyElementNotPresent	name=firstname	
verifyElementPresent	name=firstname	

Verify Presence of a Certain Text

- **verifyTextPresent** - returns TRUE if the specified text string was FOUND somewhere in the page; FALSE if otherwise
- **verifyTextNotPresent** - returns TRUE if the specified text string was NOT FOUND anywhere in the page; FALSE if it was found

Remember that these commands are case-sensitive.

Command	Target	Value
verifyTextPresent	Atlanta to Las Vegas	
verifyTextNotPresent	atlanta to Las Vegas	
verifyTextPresent	atlanta to Las Vegas	

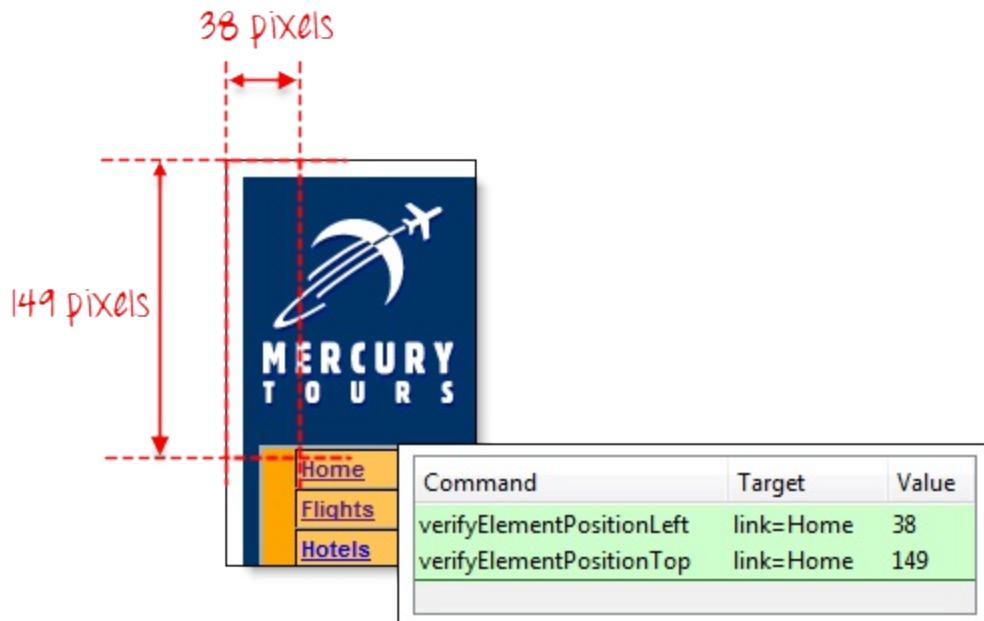
In the scenario above, "Atlanta to Las Vegas" was treated differently from "atlanta to Las Vegas" because the letter "A" of "Atlanta" was in uppercase on the first one while lowercase on the other. When the verifyTextPresent command was used on each of them, one passed

while the other failed.

Verify Specific Position of an Element

Selenium IDE indicates the position of an element by measuring (in pixels) how far it is from the left or top edge of the browser window.

- **verifyElementPositionLeft** - verifies if the specified number of pixels match the distance of the element from the left edge of the page. This will return FALSE if the value specified does not match the distance from the left edge.
- **verifyElementPositionTop** - verifies if the specified number of pixels match the distance of the element from the top edge of the page. This will return FALSE if the value specified does not match the distance from the top edge.



Wait commands

andWait commands

These are commands that will wait for a new page to load before moving onto the next command.

Examples are

- clickAndWait
- typeAndWait
- selectAndWait

PASSED

Command	Target	Value
open	/	
type	name=userName	tutorial
type	name=password	tutorial
clickAndWait	name=login	
assertTitle	Find a Flight: Mercury Tours:	

This PASSED because "clickAndwait" was used. Selenium IDE first waited for a new page to load before executing the "assertTitle" command.

FAILED

Command	Target	Value
open	/	
type	name=userName	tutorial
type	name=password	tutorial
click	name=login	
assertTitle	Find a Flight: Mercury Tours:	

This 2nd test case FAILED because "click" was used. Selenium IDE executed the "assertTitle" command without waiting for a new page to load.

waitFor commands

These are commands that wait for a specified condition to become true before proceeding to the next command (irrespective of loading of a new page). These commands are more appropriate to be used on AJAX-based dynamic websites that change values and elements without reloading the whole page. Examples include:

- waitForTitle
- waitForTextPresent
- waitForAlert

Consider the Facebook scenario below.



Note: The box appeared without reloading the page.

We can use a combination of "click" and "waitForTextPresent" to verify the presence of the text "Providing your birthday."

Command	Target	Value
open		
click	link=Why do I need to provide my birthday?	
waitForTextPresent	Providing your birthday	
verifyTextPresent	Providing your birthday	

PASSED

We cannot use clickAndWait because no page was loaded upon clicking on the "Why do I need to provide my birthday?" link. If we do, the test will fail

since clickAndWait was used even though no page was loaded, Selenium IDE basically waited for nothing

Command	Target	Value
open		
clickAndWait	link=Why do I need to provide my birthday?	
waitForTextPresent	Providing your birthday	
verifyTextPresent	Providing your birthday	

[info] Executing: clickAndWait link=Why do I need to provide my birthday?
[error] Timed out after 20000ms
[info] Executing: waitForTextPresent Providing your birthday

Logs

Summary

- The three most commonly used commands in verifying page elements are:
 - verifyElementPresent/ verifyElementNotPresent
 - verifyTextPresent/ verifyTextNotPresent
 - verifyElementPositionLeft/ verifyElementPositionTop

- Wait commands are classified into two:
 - andWait commands - used when a page is expected to be loaded
 - waitFor commands - used when no new page is expected to be loaded

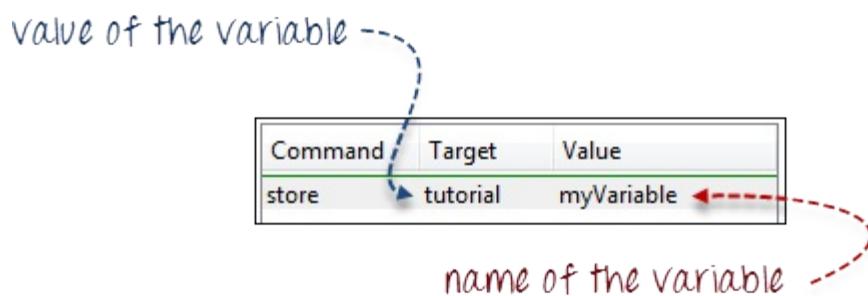
Chapter 7: Store Variables, Echo, Alert, PopUp handling in Selenium IDE

In this tutorial, we will learn, Store commands, Echo commands, Alerts and Popup handling.

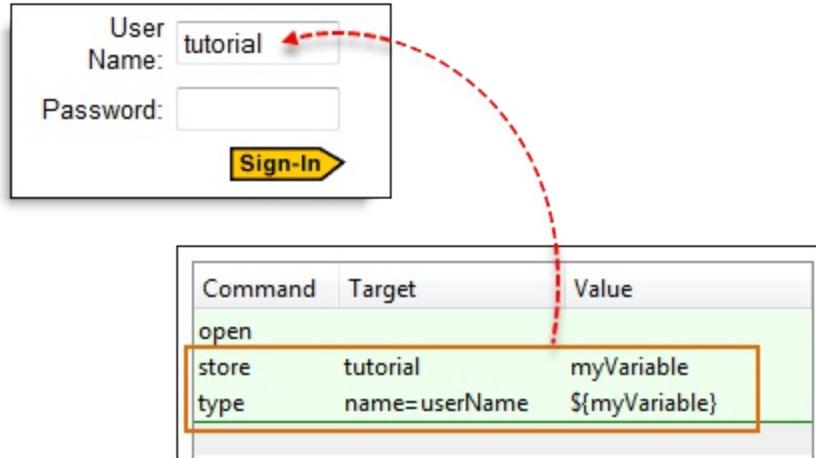
Storing Variables and the Echo command

Store

To store variables in Selenium IDE, we use the "store" command. The illustration below stores the value "tutorial" to a variable named "myVariable."

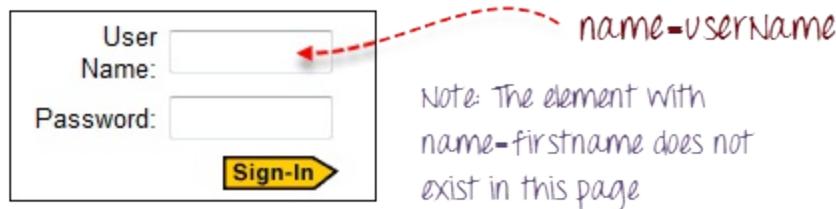


To access the variable, simply enclose it in a \${...} symbol. For example, to enter the value of "myVariable" onto the "userName" textbox of Mercury Tours, enter \${myVariable} in the Value field.



StoreElementPresent

This command stores either "true" or "false" depending on the presence of the specified element. The script below stores the Boolean value "true" to "var1" and "false" to "var2". To verify, we will use the "echo" command to display the values of var1 and var2. The Base URL for the illustration below was set to Mercury Tours homepage.



Command	Target	Value
open		
storeElementPresent	name=userName	var1
storeElementPresent	name=firstname	var2
echo	<code> \${var1}</code>	[info] Executing: echo \${var1}
echo	<code> \${var2}</code>	[info] echo: true [info] Executing: echo \${var2} [info] echo: false

StoreText

This command is used to store the inner text of an element onto a variable. The illustration below stores the inner text of the tag in Facebook onto a variable named 'textVar.'

Base URL = <http://www.facebook.com/>

Sign Up

It's free and always will be.

```
<div class="6_74">
  <h1 class="mbs _3mb _6n _6s _6v " style="font-size: 36px">Sign Up</h1>
  <h2 class="mbl _3m7 _6o _6s _mf ">It's free and always will be.</h2>
```

Since it is the only element in the page, it is safe to use 'css=h1' as our target. The image below shows that Selenium IDE was able to save the string "Sign Up" in the 'textVar' variable by printing its value correctly.

The screenshot shows the Selenium IDE interface. At the top is a table with three columns: Command, Target, and Value. The table contains the following rows:

Command	Target	Value
open		
storeText	css=h1	textVar
echo		<code> \${textVar}</code>

A red dashed arrow points from the bottom of the table to the Log tab in the Selenium IDE toolbar. Below the toolbar is a log window with the following entries:

- [info] Executing: |open |||
- [info] Executing: |storeText | css=h1 | textVar |
- [info] Executing: |echo | \${textVar} |||
- [info] echo: Sign Up

The last entry, "echo: Sign Up", is highlighted with a red box.

Alerts, Popup, and Multiple Windows

Alerts are probably the simplest form of pop-up windows. The most common Selenium IDE commands used in handling alerts are the following:

assertAlert	retrieves the message of the alert and asserts it to a string value that you specified
assertNotAlert	
assertAlertPresent	asserts if an Alert is present or not
assertAlertNotPresent	
storeAlert	retrieves the alert message and stores it in a variable that you will specify
storeAlertPresent	returns TRUE if an alert is present; FALSE if otherwise
verifyAlert	retrieves the message of the alert and verifies if it is equal to the string value that you specified
verifyNotAlert	
verifyAlertPresent	verifies if an Alert is present or not
verifyAlertNotPresent	

Remember these two things when working with alerts:

- Selenium IDE will automatically click on the OK button of the alert window, and so you will not be able to see the actual alert.
- Selenium IDE will not be able to handle alerts that are within the page's onload() function. It will only be able to handle alerts that are generated after the page has completely loaded.

In this example, we will use the storeAlert command to show that even though Selenium IDE did not show the actual alert, it was still able to retrieve its message.

Step 1. In Selenium IDE, set the Base URL to <http://jsbin.com>. & the full url is: <http://jsbin.com/usidix>

Step 2. Create the script as shown below.

Command	Target	Value
open	/usidix/1	
click	css=input[value="Go!"]	
storeAlert	alertMessage	
echo	<code> \${alertMessage}</code>	

When this button is clicked,
an alert will pop up

Step 3. Execute the script and do not expect that you will be able to see the actual alert.

This is from the info logs

```
[info] Executing: |open | /usidix/1 ||  
[info] Executing: |click | css=input[value="Go!"] ||  
[info] Executing: |storeAlert | alertMessage ||  
[info] Executing: |echo | ${alertMessage} ||  
[info] echo: This is an alert box.
```



This is the actual alert box when you try to click the "Go!" button manually. Notice that even though Selenium IDE prevented this alert from showing, it was still able to correctly get the alert message

Confirmations

Confirmations are popups that give you an OK and a CANCEL button, as opposed to alerts which give you only the OK button. The commands you can use in handling confirmations are similar to those in handling alerts.

- assertConfirmation/assertNotConfirmation
- assertConfirmationPresent/assertConfirmationNotPresent

- storeConfirmation
- storeConfirmationPresent
- verifyConfirmation/verifyNotConfirmation
- verifyConfirmationPresent/verifyConfirmationNotPresent

However, these are the additional commands that you need to use to instruct Selenium which option to choose, whether the OK or the CANCEL button.

- chooseOkOnNextConfirmation/chooseOkOnNextConfirmationAndText
- chooseCancelOnNextConfirmation

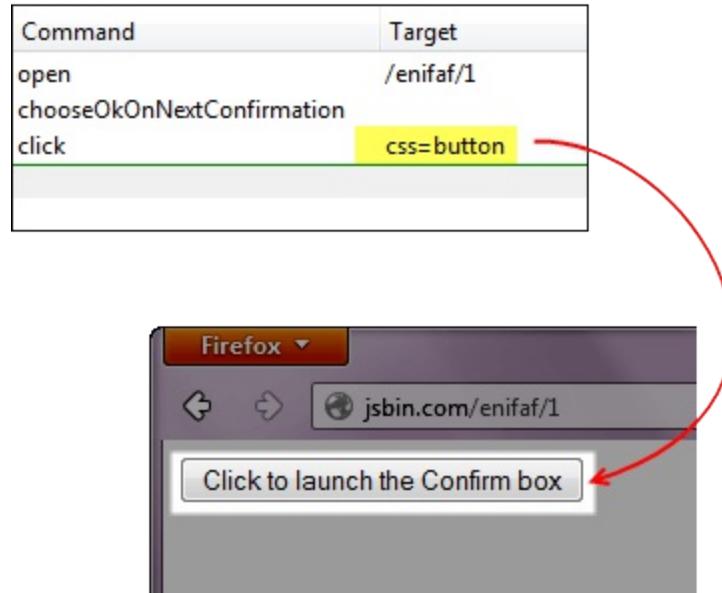
You should use these commands before a command that triggers the confirmation box so that Selenium IDE will know beforehand which option to choose. Again, you will not be able to see the actual confirmation box during script execution.

Let us test a webpage that has a button that was coded to show whether the user had pressed the OK or the CANCEL button.

Step 1. In Selenium IDE, set the Base URL to <http://jsbin.com>

& the full url is: <http://jsbin.com/enifaf>

Step 2. Create the script as shown below. This time, we will press the OK button first.



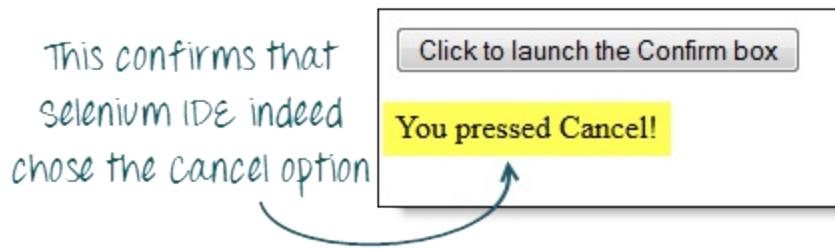
This is the button that will trigger the confirmation dialog box

Step 3. Execute the script and notice that you do not see the actual confirmation, but the webpage was able to indicate which button Selenium IDE had pressed.



This confirms that selenium IDE indeed chose the OK option in the confirm dialog

Step 4. Replace the "chooseOkOnNextConfirmation" command with "chooseCancelOnNextConfirmation" and execute the script again.



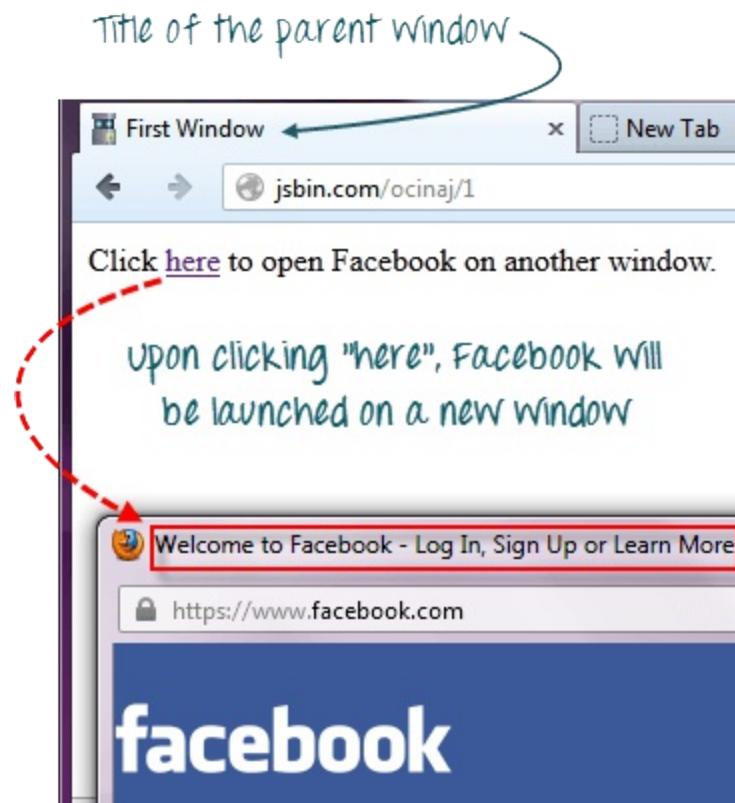
Multiple Windows

If you happen to click on a link that launches a separate window, **you must first instruct Selenium IDE to select that window first before you could access the elements within it.** To do this, you will **use the window's title as its locator.**

We use the selectWindow command in switching between windows.

We will use a link <http://jsbin.com/ocinaj/1> whose title is "First Window." The "here" hyperlink found on that page will open Facebook in a new window, after which we shall instruct Selenium IDE to do the following:

- Transfer control from the parent window to the newly launched Facebook window using the "selectWindow" command and its title as the locator
- Verify the title of the new window
- Select back the original window using the "selectWindow" command and "null" as its target.
- Verify the title of the currently selected window



Step 1. Set the Base URL to <http://jsbin.com>.

Step 2. Create the script as shown below.

Command	Target
open	/ocinaj/1
click	link=here
pause	5000
selectWindow	Welcome to Facebook - Log In, Sign Up or Learn More
verifyTitle	Welcome to Facebook - Log In, Sign Up or Learn More
selectWindow	null
verifyTitle	First Window

We need the "pause" command to wait for the newly launched window to load before we could access its title.

Step 3. Execute the script. Notice that the Test Case passed, meaning that we were able to switch between windows and verify their titles

successfully.

Command	Target
open	/ocinaj/1
click	link=here
pause	5000
selectWindow	Welcome to Facebook - Log In, Sign Up or Learn More
verifyTitle	Welcome to Facebook - Log In, Sign Up or Learn More
selectWindow	null
verifyTitle	First Window

Always remember that setting selectWindow's target to "null" will automatically select the parent window (in this case, the window where the element "link=here" is found)

Note: Facebook has changed the title since the creation of Tutorials. Please modify the code accordingly

Summary

- The "store" command (and all its variants) are used to store variables in Selenium IDE
- The "echo" command is used to print a string value or a variable
- Variables are enclosed within a \${...} when being printed or used on elements
- Selenium IDE automatically presses the OK button when handling alerts
- When handling confirmation dialogs, you may instruct Selenium IDE which option to use:
 - chooseOkOnNextConfirmation/chooseOkOnNextConfirmatio
 - chooseCancelOnNextConfirmation
- Window titles are used as locators when switching between

browser windows.

- When using the "selectWindow" command, setting the Target to "null" will automatically direct Selenium IDE to select the parent window.

Chapter 8: Introduction to WebDriver & Comparison with Selenium RC

Now that you have learned to create simple tests in Selenium IDE, we shall now create more powerful scripts using an advanced tool called **WebDriver**.

What is WebDriver?

WebDriver is a web automation framework that allows you to **execute your tests against different browsers**, not just Firefox (unlike Selenium IDE).



WebDriver also enables you to **use a programming language** in creating your test scripts (not possible in Selenium IDE).

- You can now use **conditional operations** like if-then-else or switch-case
- You can also perform **looping** like do-while.

Following programming languages are supported by WebDriver

- Java
- .Net
- PHP
- Python

- Perl
- Ruby

You do not have to know all of them. You just need to be knowledgeable in one. However, in this tutorial, we will be using Java with Eclipse as our IDE.

Difference between Selenium RC and Webdriver

Before the advent of WebDriver in 2006, there was another, **automation tool called Selenium Remote Control**. Both WebDriver and Selenium RC have following features:

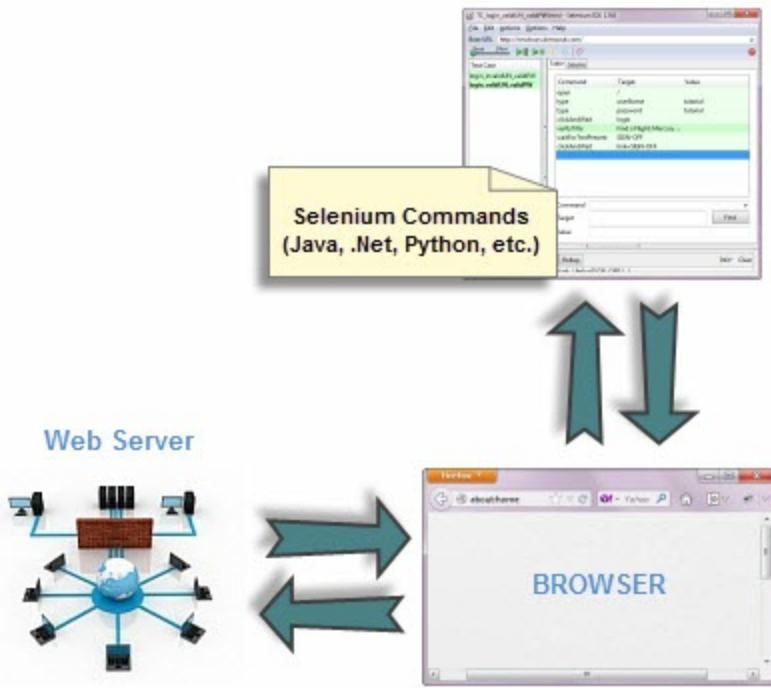
- They both allow you to **use a programming language** in designing your test scripts.
- They both allow you to **run your tests against different browsers**.

So how do they differ? Let us discuss the answers.

1. Architecture

WebDriver's architecture is simpler than Selenium RC's.

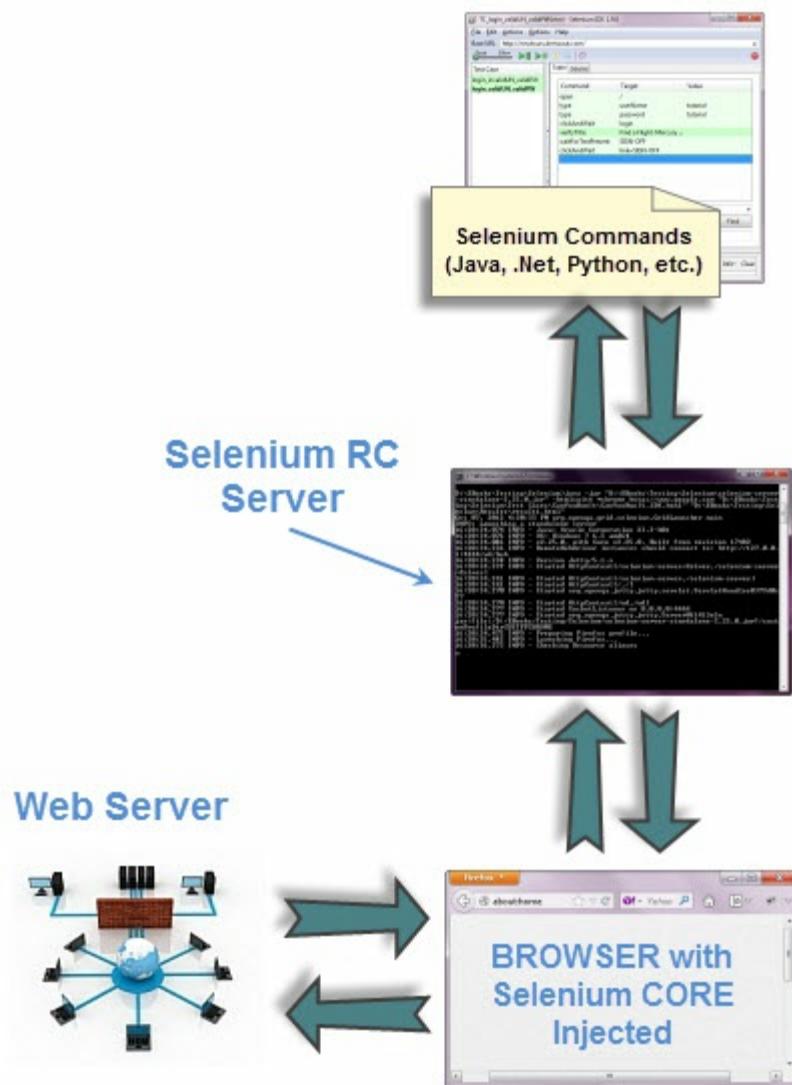
- It controls the browser from the OS level
- All you need are your programming language's IDE (which contains your Selenium commands) and a browser.



Selenium RC's architecture is way more complicated.

- You first need to launch a **separate application called Selenium Remote Control (RC) Server** before you can start testing
- The Selenium RC Server **acts as a "middleman" between your Selenium commands and your browser**
- When you begin testing, Selenium RC Server "injects" a **Javascript program called Selenium Core** into the browser.
- Once injected, Selenium Core will start receiving instructions relayed by the RC Server from your test program.
- When the instructions are received, **Selenium Core will execute them as Javascript commands.**
- The browser will obey the instructions of Selenium Core and will relay its response to the RC Server.
- The RC Server will receive the response of the browser and then display the results to you.

- RC Server will fetch the next instruction from your test script to repeat the whole cycle.



2. Speed



WebDriver is faster than Selenium RC since it speaks directly to the browser uses the browser's own engine to control it.



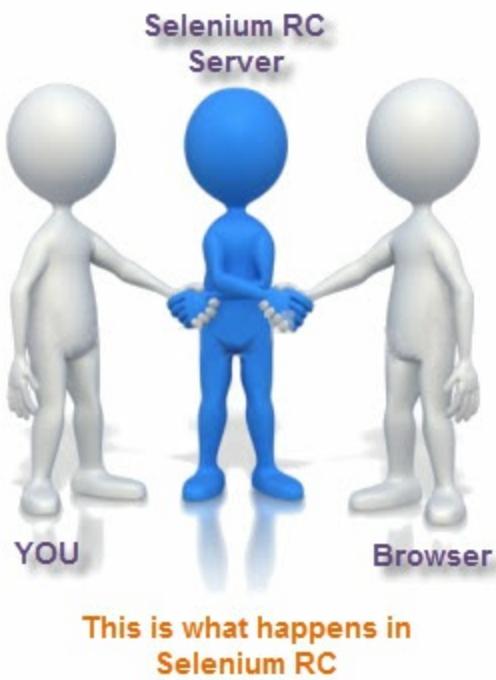
Selenium RC is slower since it uses a Javascript program called Selenium Core. This Selenium Core is the one that directly controls the browser, not you.

3. Real-life Interaction



WebDriver interacts with page elements in a more realistic

way. For example, if you have a disabled text box on a page you were testing, WebDriver really cannot enter any value in it just as how a real person cannot.



Selenium Core, just like other JavaScript codes, can access disabled elements. In the past, Selenium testers complain that Selenium Core was able to enter values to a disabled text box in their tests.

Differences in API

4. API



Selenium RC's API is more matured but contains redundancies and often confusing commands. For example, most of the time, testers are confused whether to use type or typeKeys; or whether to use click, mouseDown, or mouseDownAt. Worse, **different browsers interpret each of these commands in different ways too!**

WebDriver's API is simpler than Selenium RC's. It does not contain redundant and confusing commands.

5. Browser Support

WebDriver can drive an invisible browser
called HtmlUnit



WebDriver can support the headless HtmlUnit browser

HtmlUnit is termed as "headless" because it is an invisible browser - it is GUI-less.

It is a very fast browser because no time is spent in waiting for page elements to load. This accelerates your test execution cycles.

Since it is invisible to the user, it can only be controlled through automated means.

Selenium RC cannot support the headless HtmlUnit browser. It needs a real, visible browser to operate on.

Limitations of WebDriver

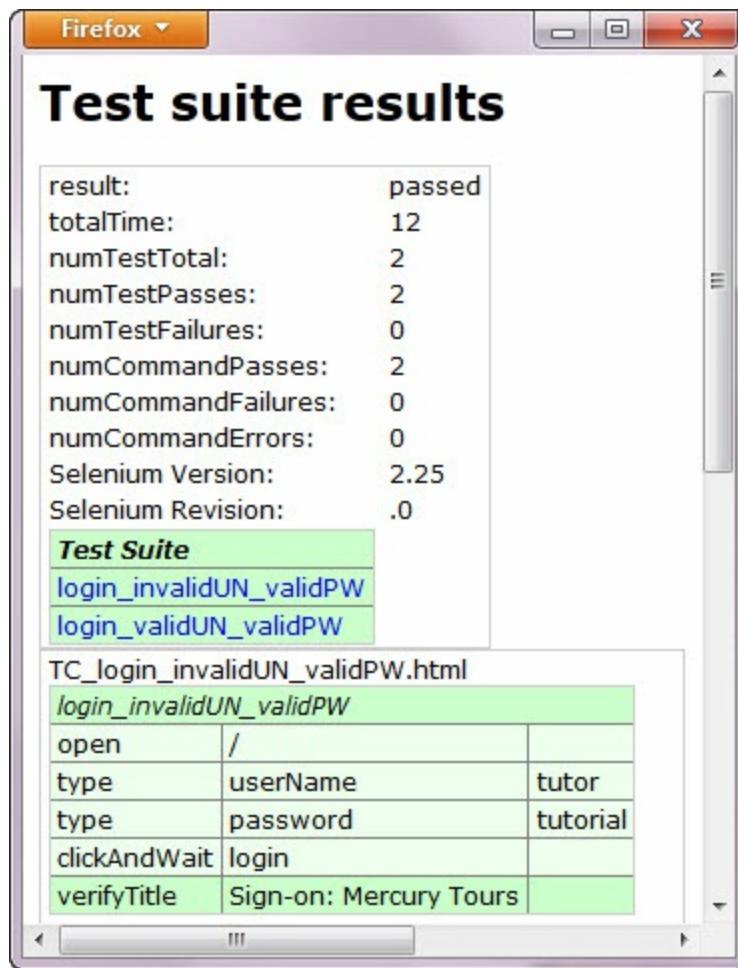
WebDriver Cannot Readily Support New Browsers

Remember that WebDriver operates on the OS level. Also, remember that different browsers communicate with the OS in different ways. If a new browser comes out, it may have a different process of communicating with the OS as compared to other browsers. So, **you have to give the WebDriver team quite some time to figure that new process out** before they can implement it on the next WebDriver release.

However, it is up to the WebDriver's team of developers to decide if they should support the new browser or not.

Selenium RC Has Built-In Test Result Generator

Selenium RC automatically generates an HTML file of test results. The format of the report was pre-set by RC itself. Take a look at an example of this report below.



WebDriver has no built-in command that automatically generates a Test Results File. You would have to rely on your IDE's output window, or design the report yourself using the capabilities of your programming language and store it as text, HTML, etc.

Summary

- WebDriver is a tool for testing web applications **across different browsers** using different programming languages.
- You are now able to make powerful tests because WebDriver **allows you to use a programming language** of your choice

in designing your tests.

- WebDriver is **faster than Selenium RC** because of its simpler architecture.
- WebDriver **directly talks to the browser** while Selenium RC needs the help of the RC Server in order to do so.
- WebDriver's API is more **concise** than Selenium RC's.
- WebDriver **can support HtmlUnit** while Selenium RC cannot.
- The only drawbacks of WebDriver are:
 - It **cannot readily support new browsers**, but Selenium RC can.
 - It **does not have a built-in command** for automatic generation of test results.

Chapter 9: Guide to install Selenium WebDriver

In this tutorial, we will install Webdriver (Java only) and Configure Eclipse

Step 1 - Install Java on your computer

Download and install the **Java Software Development Kit (JDK)** here.



Next –

1 Click this radio button

Java SE Development Kit 8u121
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.86 MB	jdk-8u121-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.83 MB	jdk-8u121-linux-arm64-vfp-hflt.tar.gz
Linux x86	162.41 MB	jdk-8u121-linux-i586.rpm
Linux x86	177.13 MB	jdk-8u121-linux-i586.tar.gz
Linux x64	159.96 MB	jdk-8u121-linux-x64.rpm
Linux x64	174.76 MB	jdk-8u121-linux-x64.tar.gz
Mac OS X	223.21 MB	jdk-8u121-macosx-x64.dmg
Solaris SPARC 64-bit	139.64 MB	jdk-8u121-solaris-sparcv9.tar.gz
Solaris SPARC 64-bit	99.07 MB	jdk-8u121-solaris-sparcv9.tar.Z
Solaris x64	140.42 MB	jdk-8u121-solaris-x64.tar.Z
Solaris x64	96.9 MB	jdk-8u121-solaris-x64.tar.gz
Windows x86	189.36 MB	jdk-8u121-windows-i586.exe
Windows x64	195.51 MB	jdk-8u121-windows-x64.exe

2 Choose the JDK that corresponds to your OS

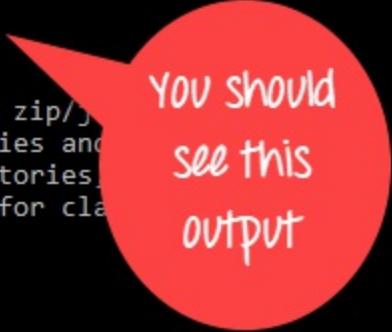
This JDK version comes bundled with Java Runtime Environment (JRE), so you do not need to download and install the JRE separately.

Once installation is complete, open command prompt and type “java”. If you see the following screen you are good to move to the next step

ct: Command Prompt

```
C:\Users\Krishna Rungta>java...
Usage: java [-options] class [args...]
            (to execute a class)
    or  java [-options] -jar jarfile [args...]
            (to execute a jar file)
where options include:
    -d32          use a 32-bit data model if available
    -d64          use a 64-bit data model if available
    -server       to select the "server" VM
                  The default VM is server.

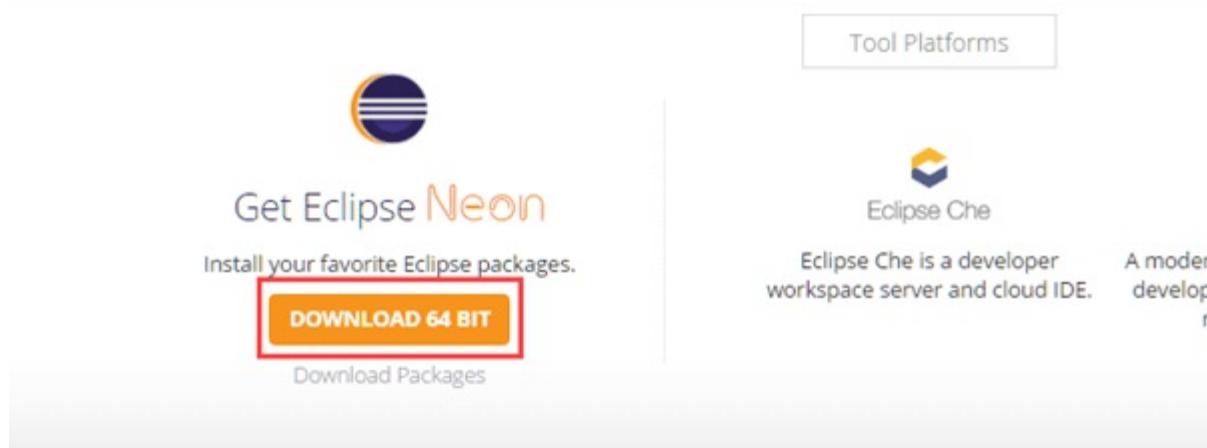
    -cp <class search path of directories and zip/jar files>
    -classpath <class search path of directories and ZIP archives to search for classes>
                  A ; separated list of directories
    -D<name>=<value>
                  set a system property
    -verbose:[class|gc|jni]
                  enable verbose output
    -version      print product version and exit
    -version:<value>
                  Warning: this feature is deprecated and will be removed
                  in a future release
```



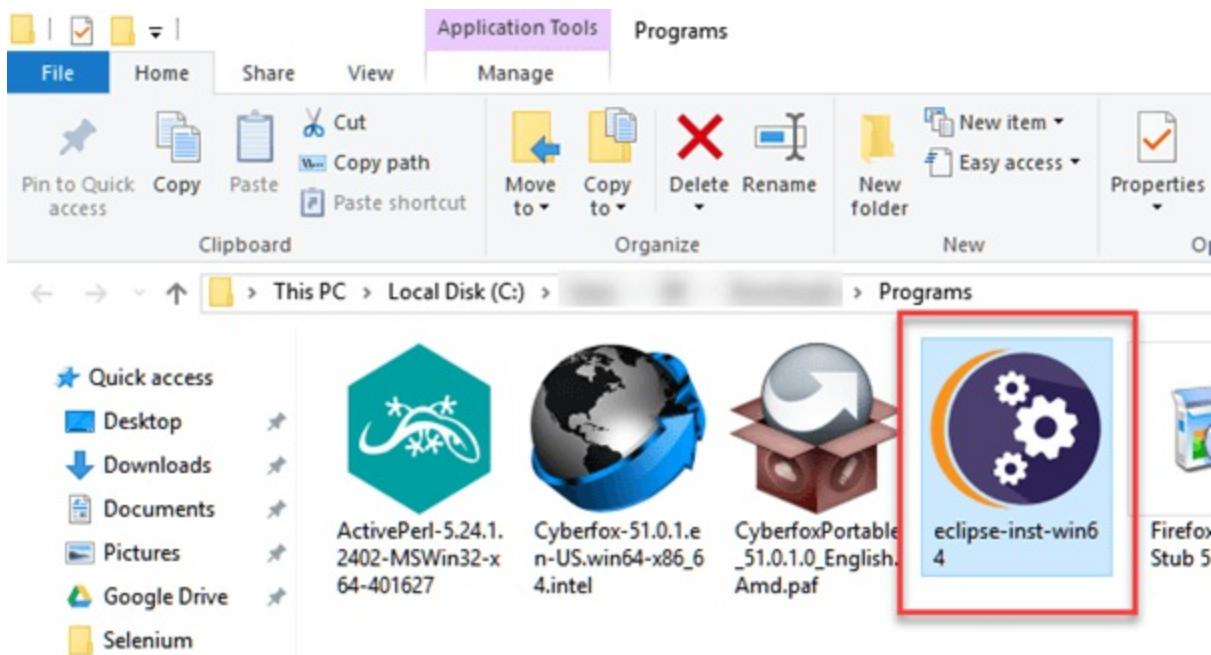
You should see this output

Step 2 - Install Eclipse IDE

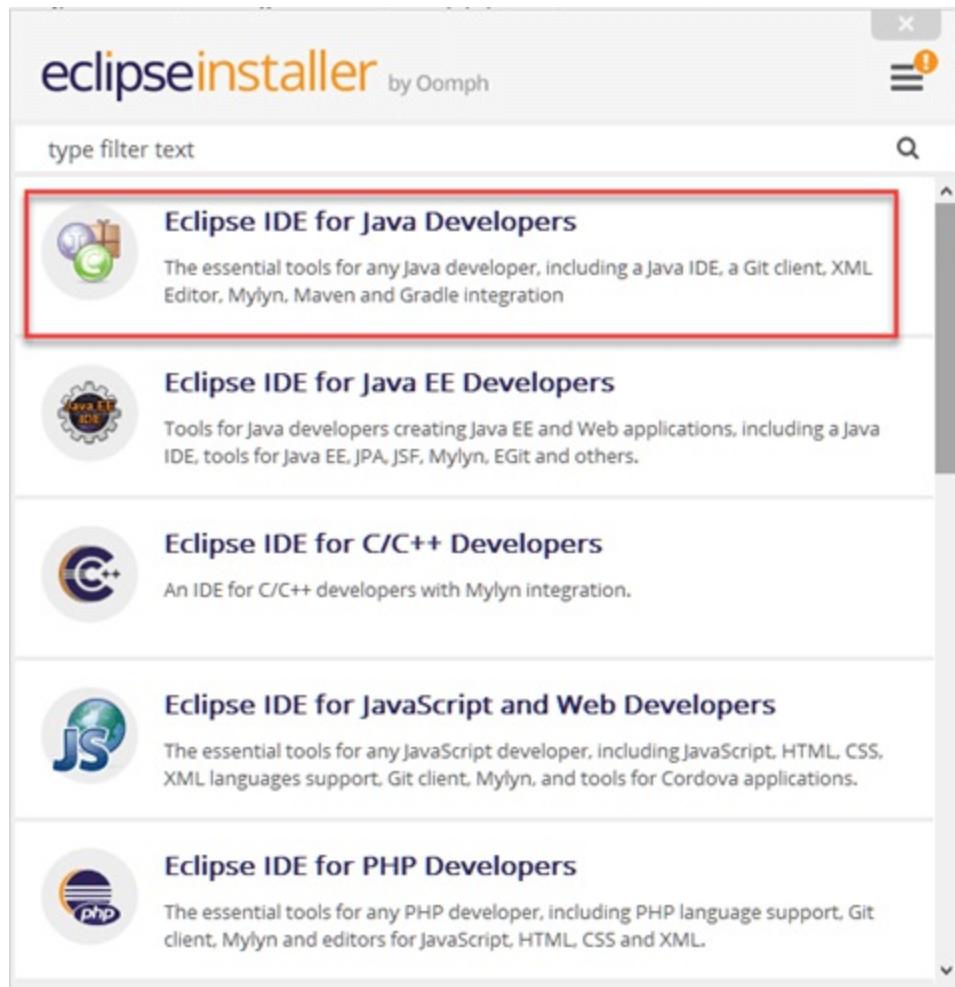
Download "**Eclipse IDE for Java Developers**" here. Be sure to choose correctly between Windows 32 Bit and 64 Bit versions.



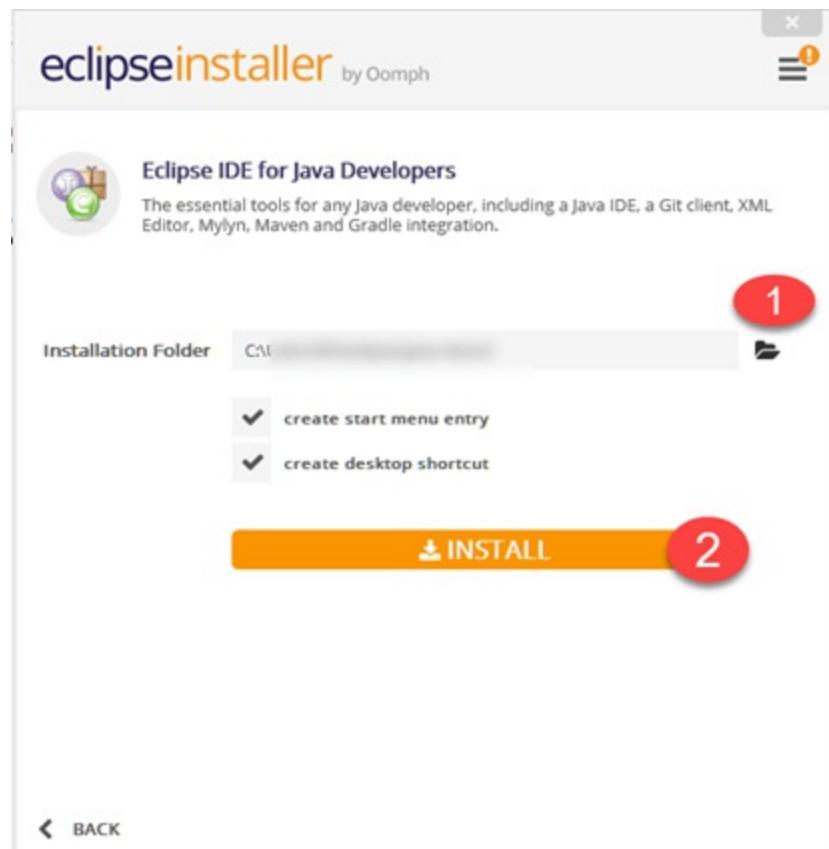
You should be able to download an exe file named "eclipse-inst-win64"



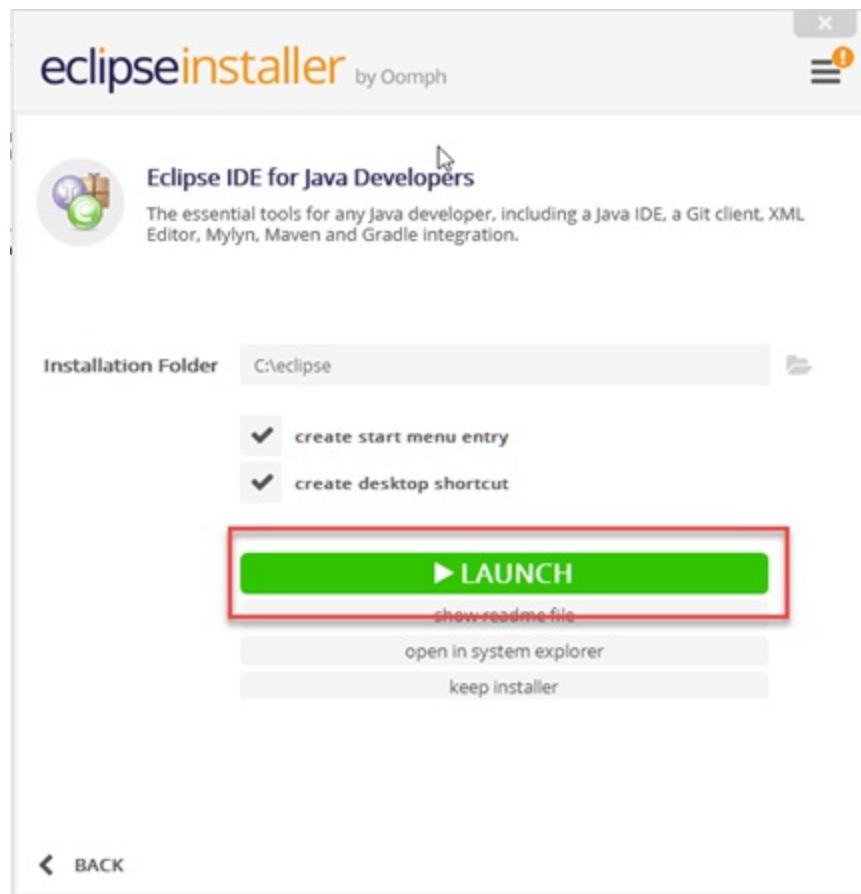
Double-click on file to Install the Eclipse. A new window will open.
Click Eclipse IDE for Java Developers.



After that, a new window will open which click button marked 1 and change path to "C:\eclipse". Post that Click on Install button marked 2



After successful completion of the installation procedure, a window will appear. On that window click on Launch



This will start eclipse neon IDE for you.

Step 3 - Download the Selenium Java Client Driver

You can download the **Selenium Java Client Driver** here. You will find client drivers for other languages there, but only choose the one for Java.

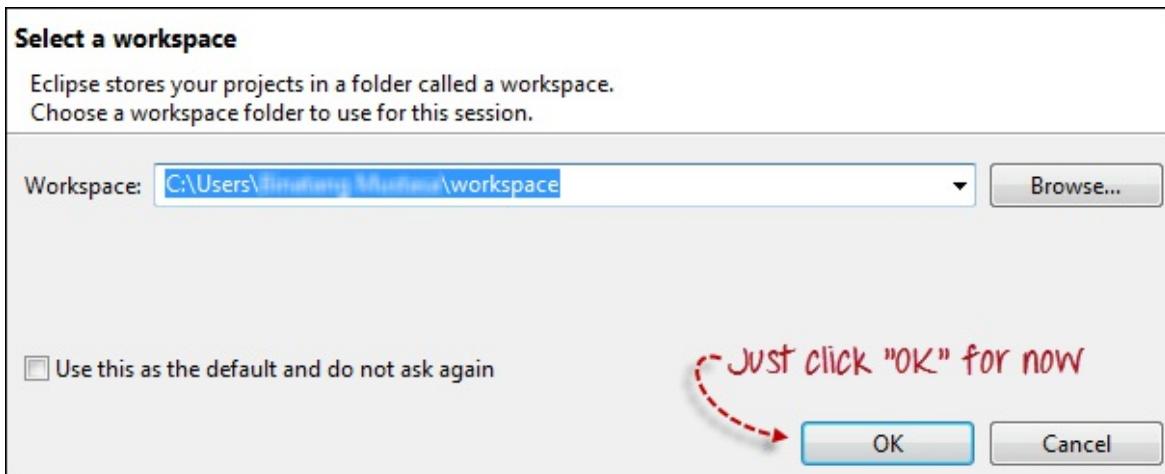
this is the download for Java Client Driver

Language	Client Version	Release Date	Download	Change log	Javadoc
Java	3.0.1	2016-10-18	Download	Change log	Javadoc
C#	3.0.0	2016-10-13	Download	Change log	API docs
Ruby	3.0.0	2016-10-13	Download	Change log	API docs
Python	3.0.2	2016-11-29	Download	Change log	API docs
Javascript (Node)	3.0.0-beta-2	2016-08-07	Download	Change log	API docs

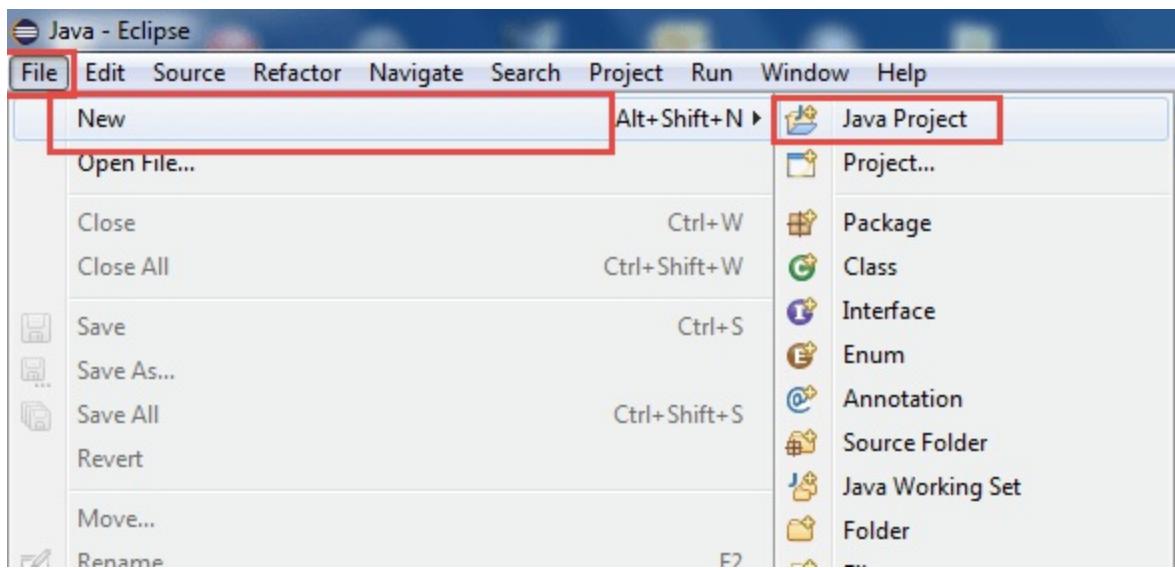
This download comes as a ZIP file named "selenium-2.25.0.zip". For simplicity, extract the contents of this ZIP file on your C drive so that you would have the directory "C:\selenium-2.25.0\". This directory contains all the JAR files that we would later import on Eclipse.

Step 4 - Configure Eclipse IDE with WebDriver

1. 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.
2. When asked to select for a workspace, just accept the default location.

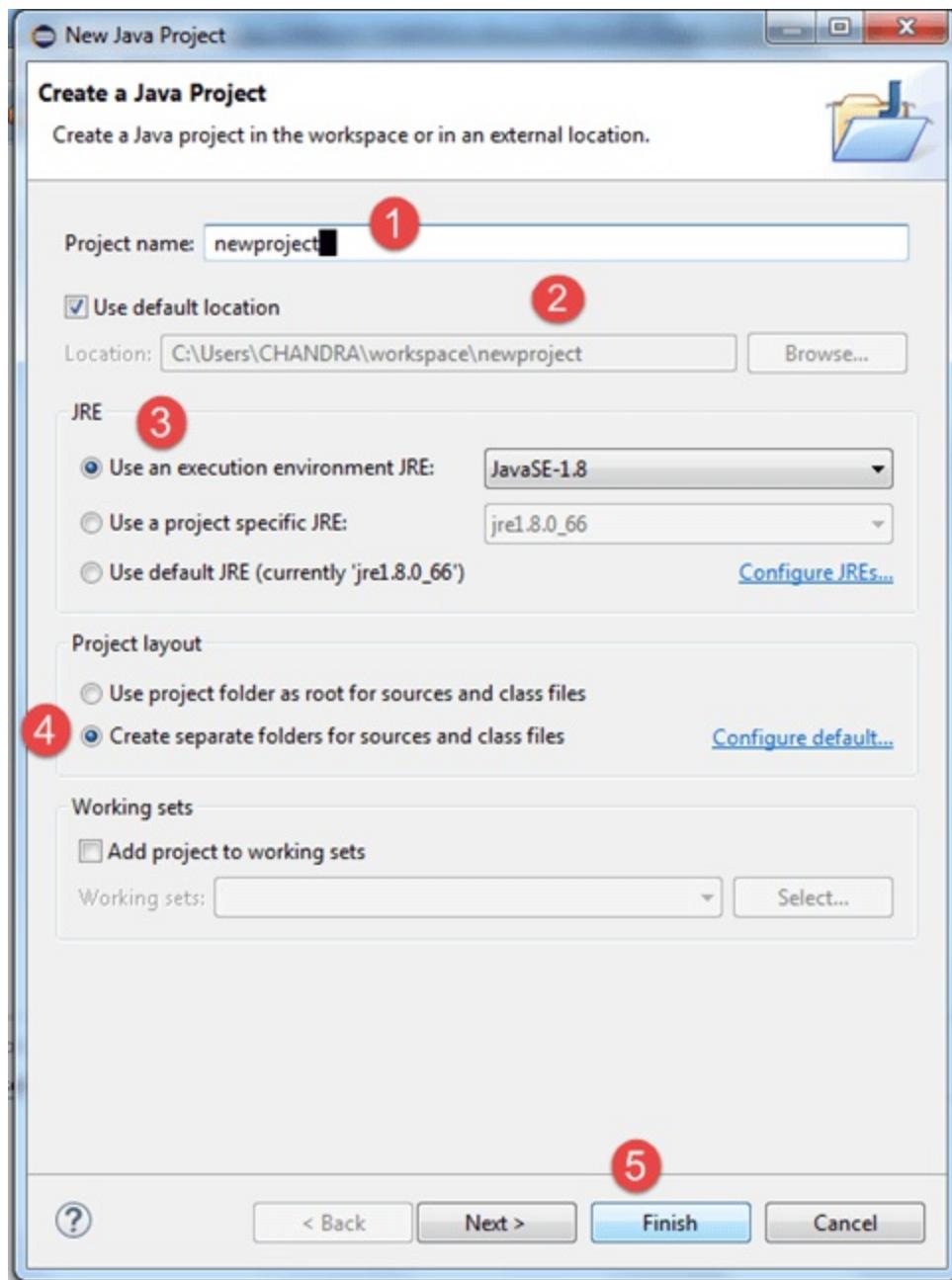


3. Create a new project through File > New > Java Project. Name the project as "newproject".



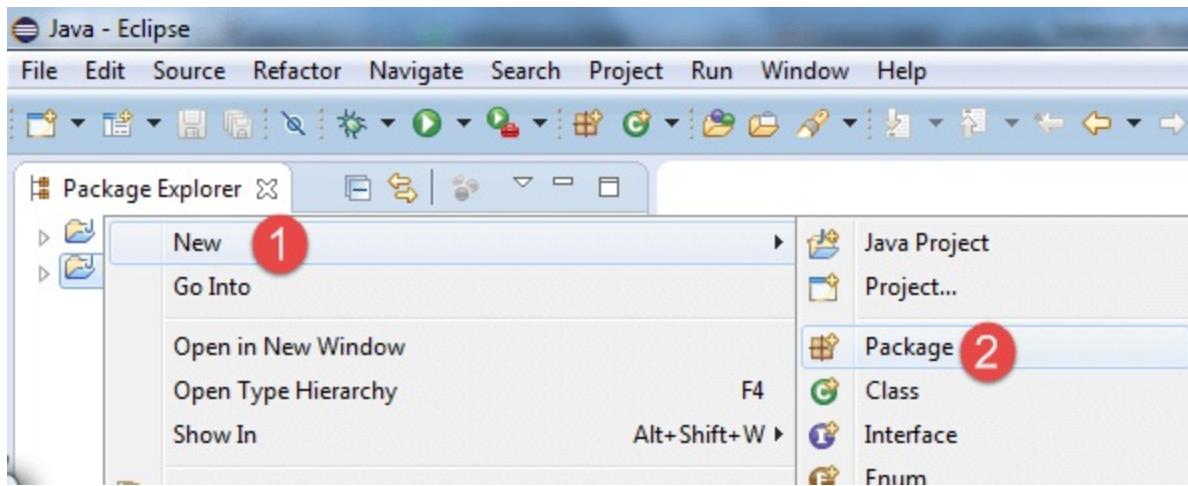
A new pop-up window will open enter details as follow

1. Project Name
2. Location to save project
3. Select an execution JRE
4. Select layout project option
5. Click on Finish button



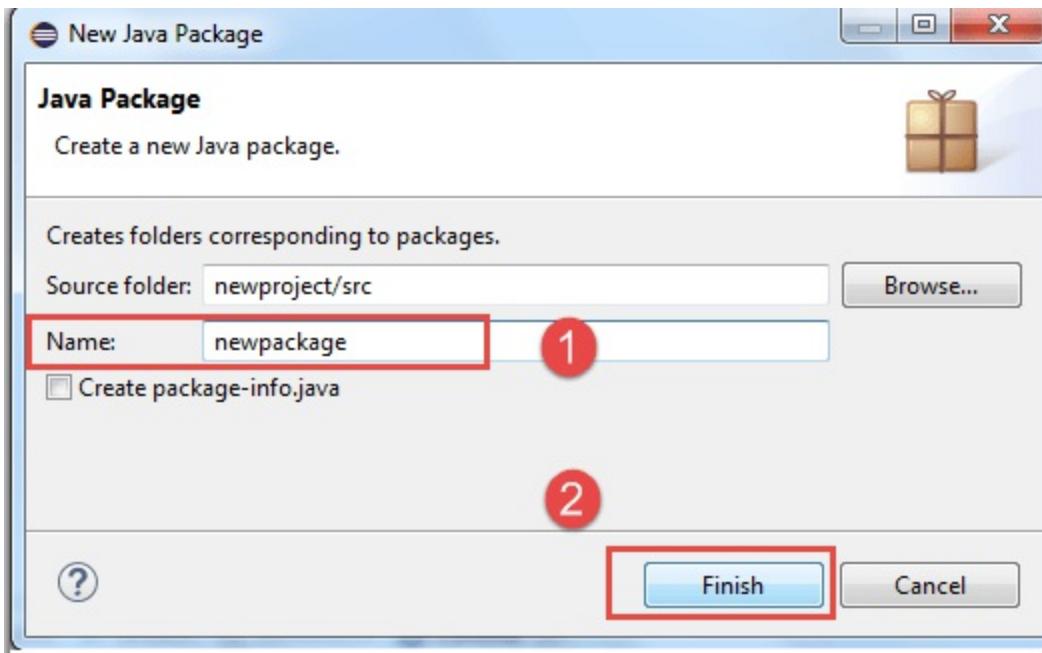
4. In this step,

1. Right-click on the newly created project and
2. Select New > Package, and name that package as "newpackage".

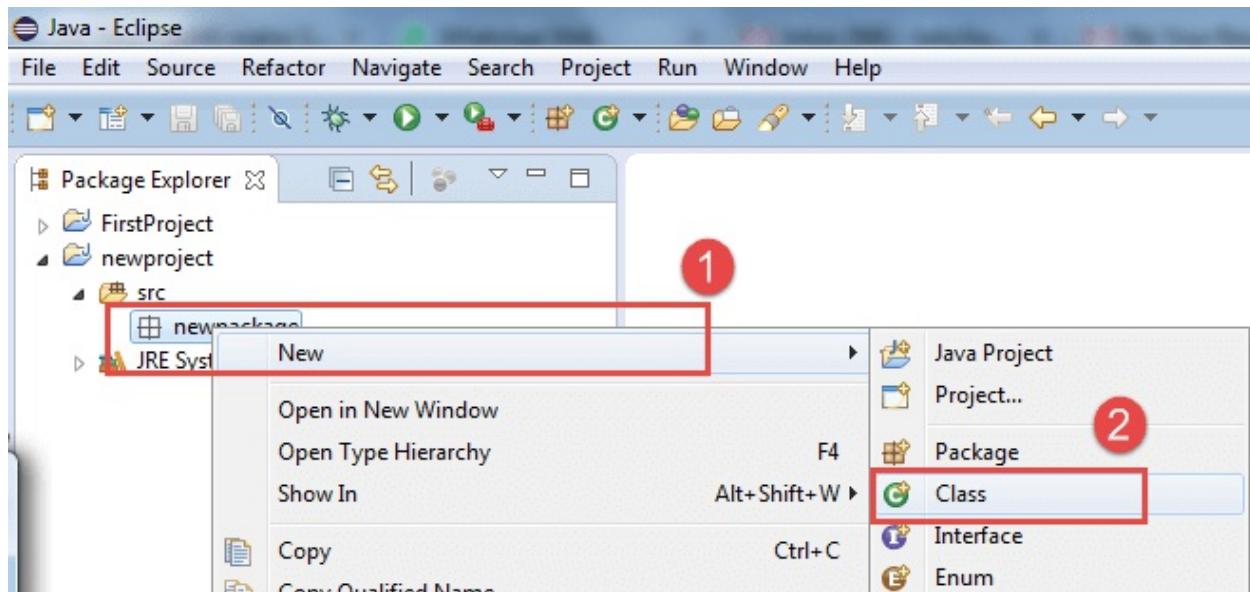


A pop-up window will open to name the package,

1. Enter the name of the package
2. Click on Finish button

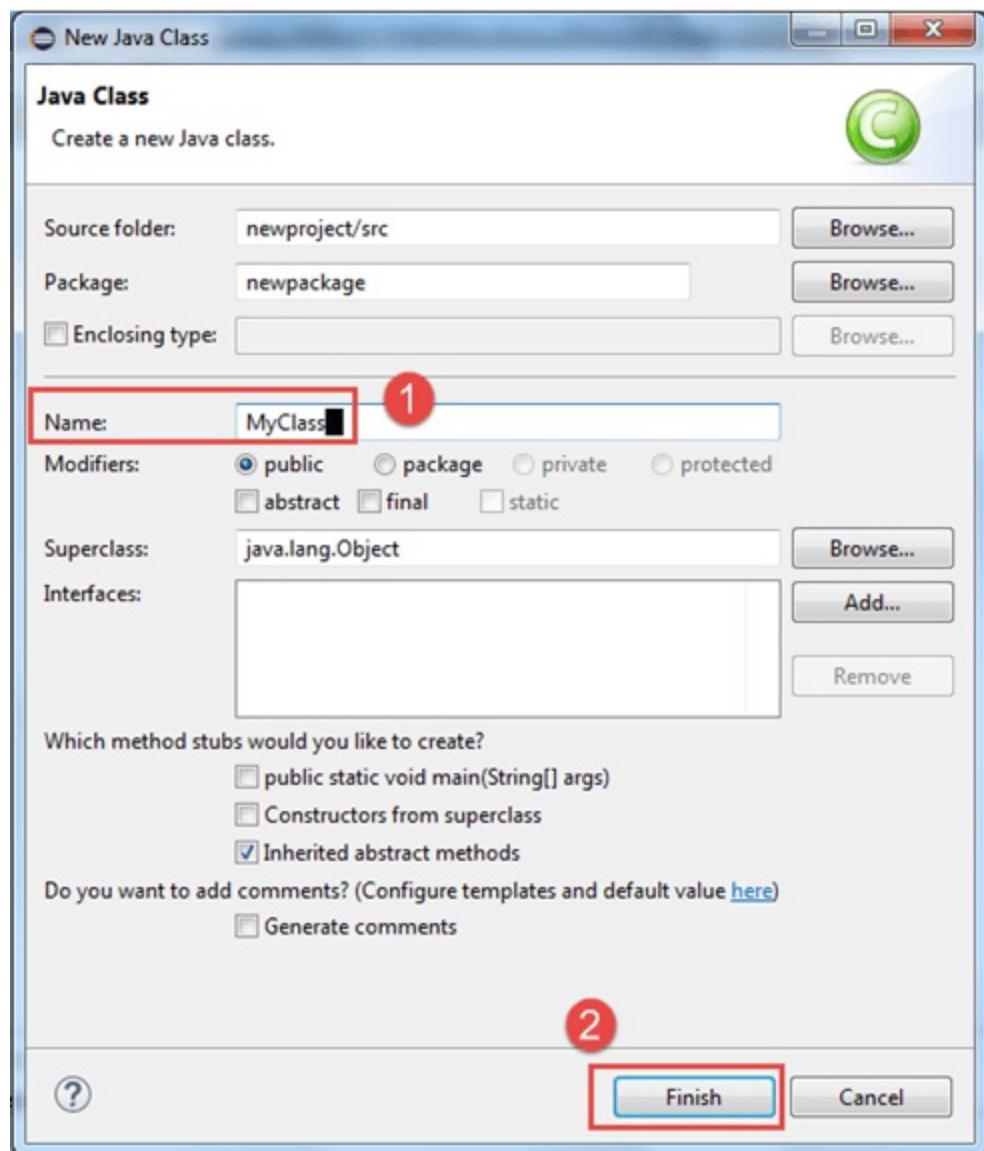


5. Create a new Java class under newpackage by right-clicking on it and then selecting- New > Class, and then name it as "MyClass". Your Eclipse IDE should look like the image below.

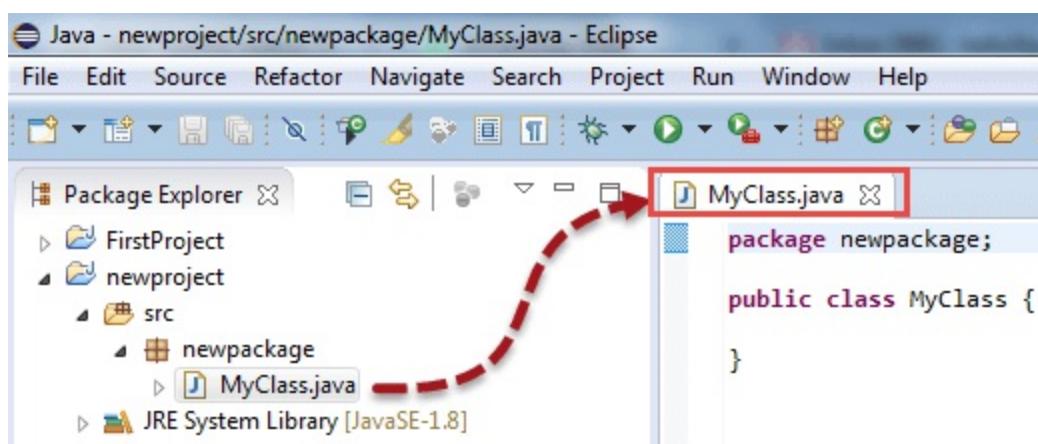


When you click on Class, a pop-up window will open, enter details as

1. Name of the class
2. Click on Finish button



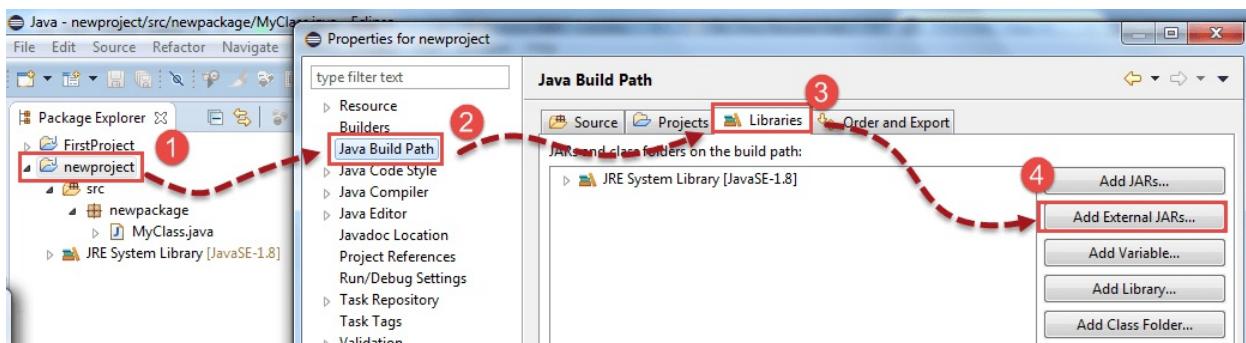
This is how it looks like after creating class.



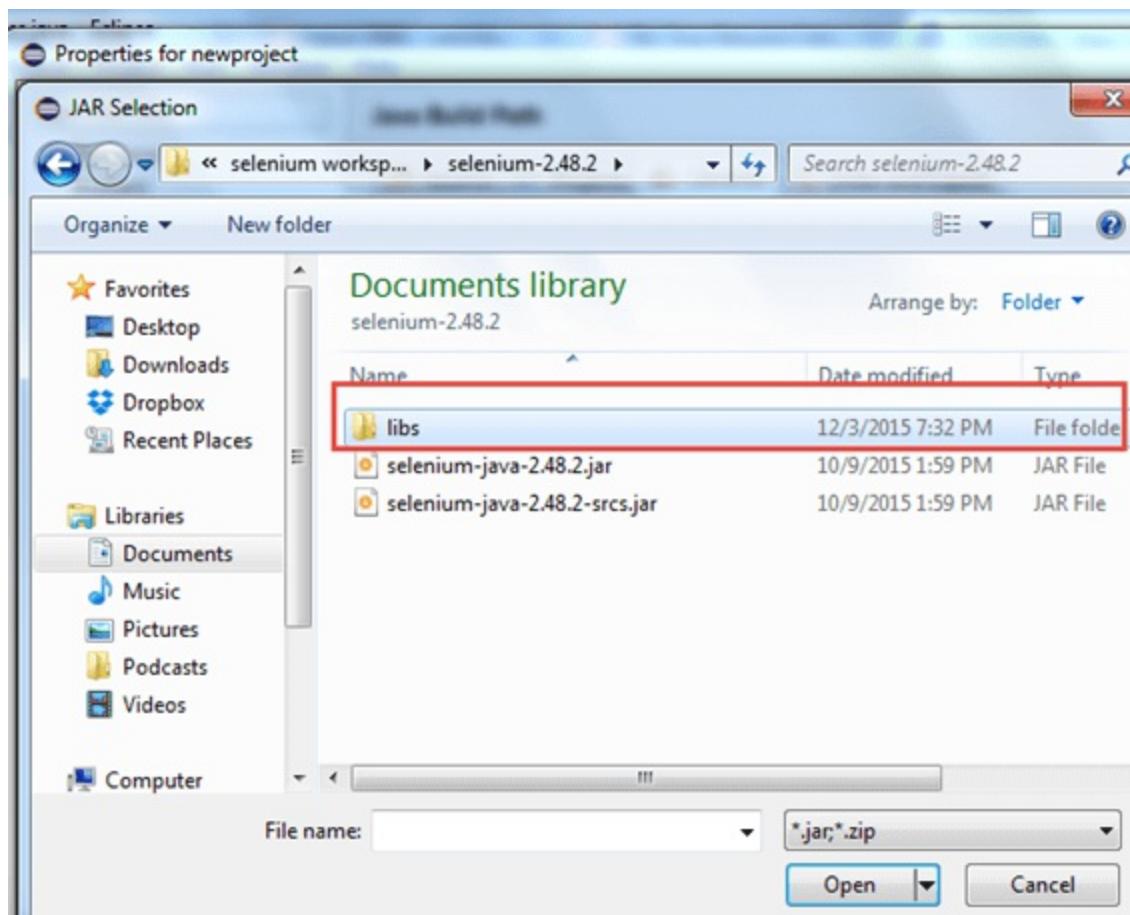
Now selenium WebDriver's into Java Build Path

In this step,

1. Right-click on "newproject" and select **Properties**.
2. On the Properties dialog, click on "Java Build Path".
3. Click on the **Libraries** tab, and then
4. Click on "Add External JARs.."

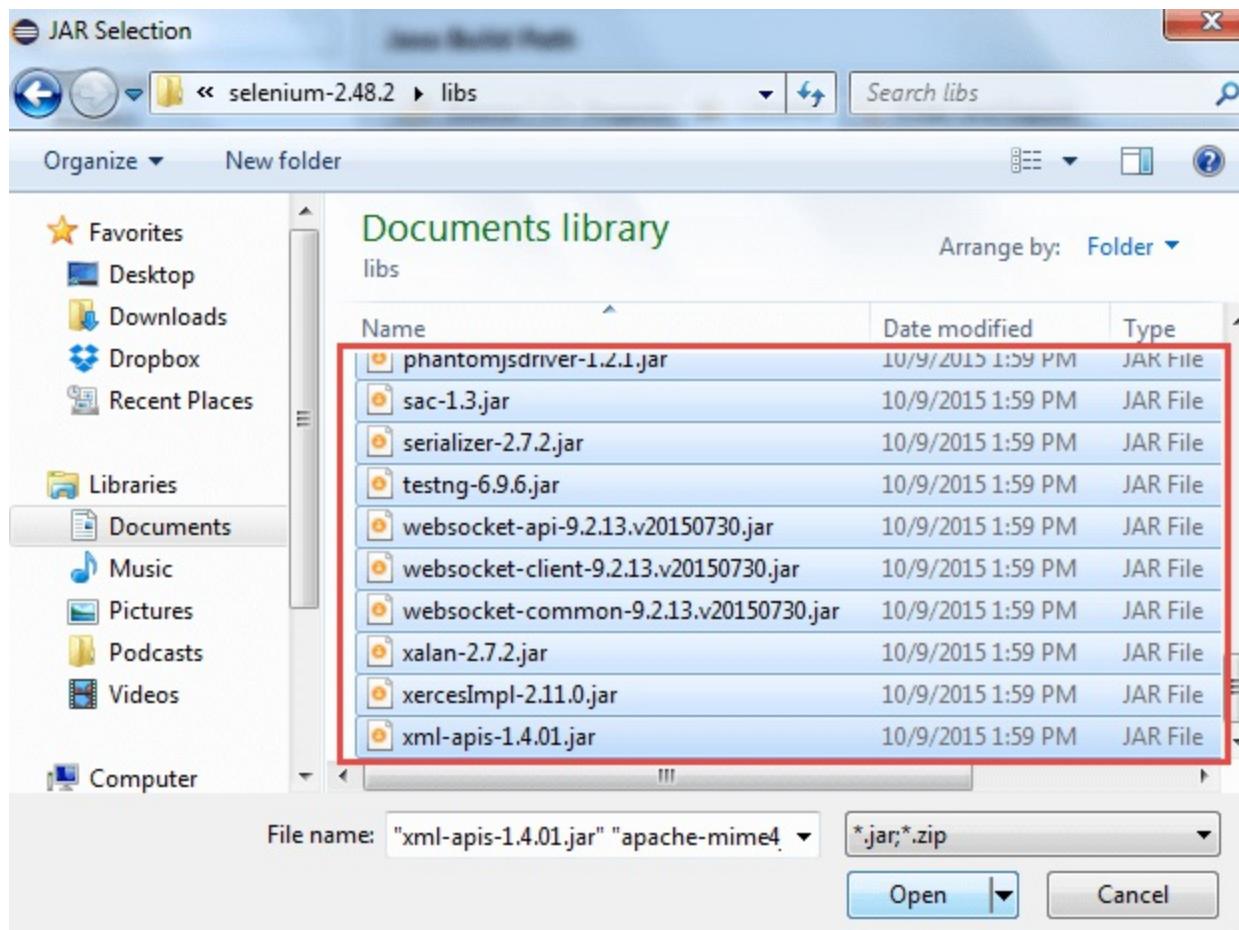


When you click on "Add External JARs.." It will open a pop-up window. Select the JAR files you want to add.

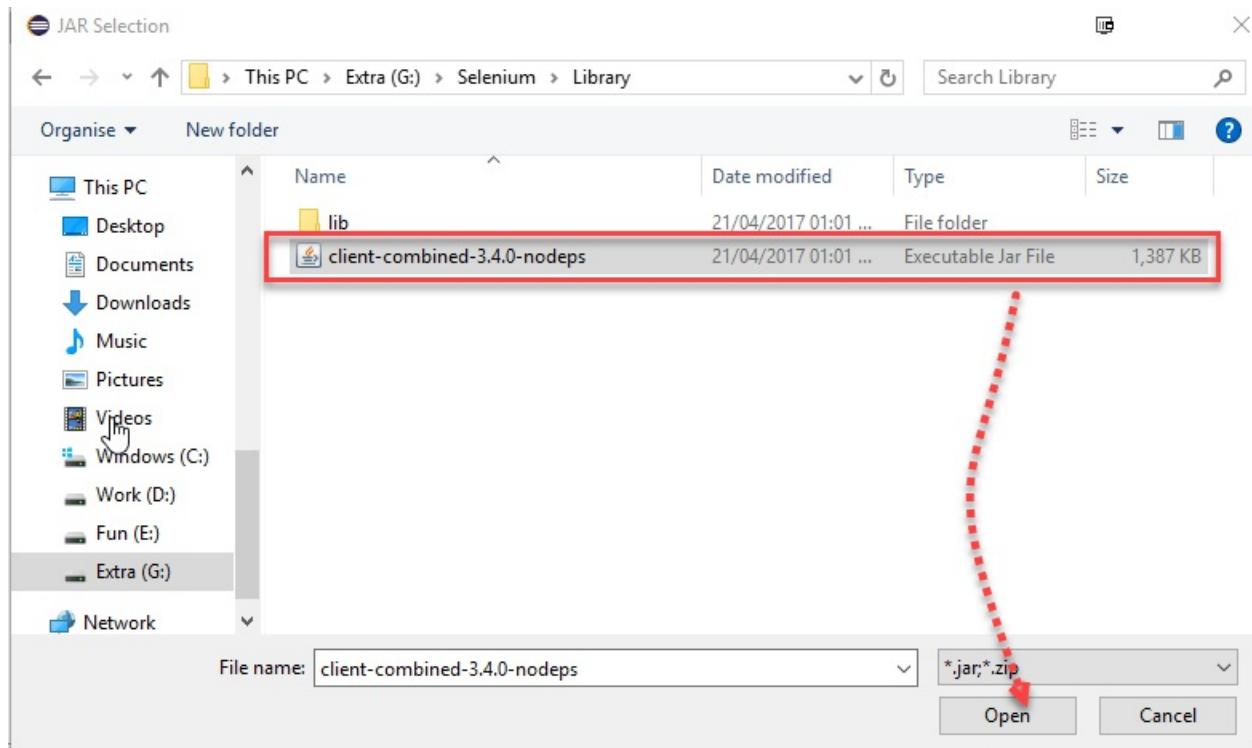


After selecting jar files, click on OK button.

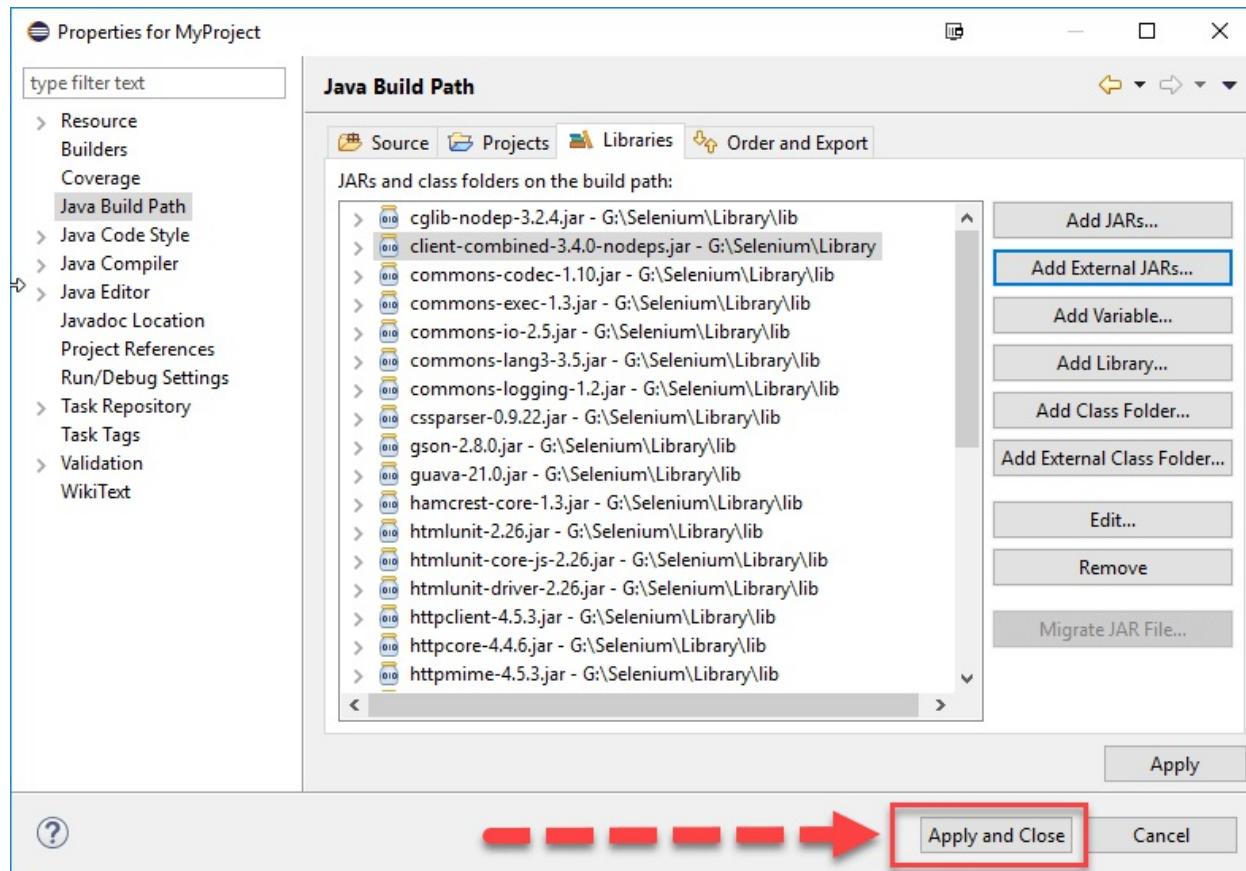
Select all files inside the lib folder.



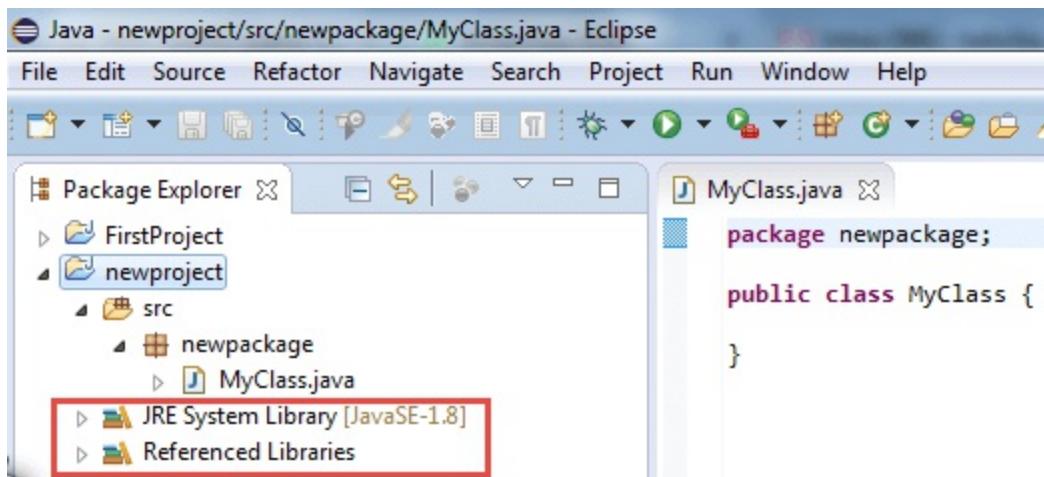
Select files outside lib folder



Once done, click "Apply and Close" button



6. Add all the JAR files inside and outside the "libs" folder. Your Properties dialog should now look similar to the image below.



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

Different Drivers

HTMLUnit and Firefox are two browsers that WebDriver can directly automate - meaning that no other separate component is needed to install or run while the test is being executed. For other browsers, a separate program is needed. That program is called as the **Driver Server**.

A driver server is different for each browser. For example, Internet Explorer has its own driver server which you cannot use on other browsers. Below is the list of driver servers and the corresponding browsers that use them.

You can download these drivers here

Browser	Name of Driver Server	Remarks
HTMLUnit	HtmlUnitDriver	WebDriver can drive HTMLUnit using HtmlUnitDriver as driver server
Firefox	Mozilla GeckoDriver	WebDriver can drive Firefox without the need of a driver server Starting Firefox 45 & above one needs to use gecko driver created by Mozilla for automation
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.

Summary

Aside from a browser, you will need the following to start using WebDriver

- **Java Development Kit (JDK).**
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- **Eclipse IDE** - <http://www.eclipse.org/downloads/>
- **Java Client Driver** - <http://seleniumhq.org/download/>

When starting a WebDriver project in Eclipse, do not forget to import the Java Client Driver files onto your project. These files will constitute your Selenium Library.

With new version of Selenium, there is no browser that you can automate without the use of a Driver Server.

Chapter 10: Creating your First Script in Webdriver

Using the Java class "myclass" that we created in the previous tutorial, let us try to create a WebDriver script that would:

1. fetch Mercury Tours' homepage
2. verify its title
3. print out the result of the comparison
4. close it before ending the entire program.

WebDriver Code

Below is the actual WebDriver code for the logic presented by the scenario above

Note: Starting Firefox 35, you need to use gecko driver created by Mozilla to use Web Driver. Selenium 3.0, gecko and firefox has compatibility issues and setting them correctly could become an uphill task. If the code does not work, downgrade to Firefox version 47 or below.
Alternatively, you can run your scripts on Chrome. Selenium works out of the box for Chrome. You just need to change 3 lines of code to make your script work with Chrome or Firefox

```
package newproject;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
//comment the above line and uncomment below line to use Chrome
```

```
//import org.openqa.selenium.chrome.ChromeDriver;
public class PG1 {

    public static void main(String[] args) {
        // declaration and instantiation of objects/variables

System.setProperty("webdriver.firefox.marionette", "C:\\geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        //comment the above 2 lines and uncomment below
2 lines to use Chrome

//System.setProperty("webdriver.chrome.driver", "G:\\chromedriver.exe");
        //WebDriver driver = new ChromeDriver();

        String baseUrl =
"http://demo.guru99.com/test/newtours/";
        String expectedTitle = "Welcome: Mercury Tours";
        String actualTitle = "";

        // launch Fire fox and direct it to the Base URL
        driver.get(baseUrl);

        // get the actual value of the title
        actualTitle = driver.getTitle();

        /*
         * compare the actual title of the page with the
expected one and print
         * the result as "Passed" or "Failed"
         */
        if (actualTitle.contentEquals(expectedTitle)){
            System.out.println("Test Passed!");
        } else {
            System.out.println("Test Failed");
        }

        //close Fire fox
        driver.close();

    }
}
```

{

Explaining the code

Importing Packages

To get started, you need to import following two packages:

1. **org.openqa.selenium.***- contains the WebDriver class needed to instantiate a new browser loaded with a specific driver
2. **org.openqa.selenium.firefox.FirefoxDriver** - contains the FirefoxDriver class needed to instantiate a Firefox-specific driver onto the browser instantiated by the WebDriver class

If your test needs more complicated actions such as accessing another class, taking browser screenshots, or manipulating external files, definitely you will need to import more packages.

Instantiating objects and variables

Normally, this is how a driver object is instantiated.

```
WebDriver driver = new FirefoxDriver();
```

A FirefoxDriver class with no parameters means that the default Firefox profile will be launched by our Java program. The default Firefox profile is similar to launching Firefox in safe mode (no extensions are loaded).

For convenience, we saved the Base URL and the expected title as variables.

Launching a Browser Session

WebDriver's **get()** method is used to launch a new browser session and directs it to the URL that you specify as its parameter.

```
driver.get(baseUrl);
```

Get the Actual Page Title

The WebDriver class has the **getTitle()** method that is always used to obtain the page title of the currently loaded page.

```
actualTitle = driver.getTitle();
```

Compare the Expected and Actual Values

This portion of the code simply uses a basic Java if-else structure to compare the actual title with the expected one.

```
if (actualTitle.contentEquals(expectedTitle)) {  
    System.out.println("Test Passed!");  
} else {  
    System.out.println("Test Failed!");  
}
```

Terminating a Browser Session

The "close()" method is used to close the browser window.

```
driver.close();
```

Terminating the Entire Program

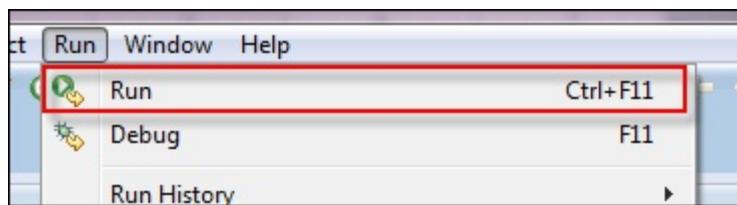
If you use this command without closing all browser windows first, your whole Java program will end while leaving the browser window open.

```
System.exit(0);
```

Running the Test

There are two ways to execute code in Eclipse IDE.

1. On Eclipse's menu bar, click **Run > Run**.
2. Press **Ctrl+F11** to run the entire code.



If you did everything correctly, Eclipse would output "Test Passed!"



Locating GUI Elements

Locating elements in WebDriver is done by using the "**findElement(By.locator())**" method. The "locator" part of the code is same as any of the locators previously discussed in the Selenium IDE chapters of these tutorials. Infact, it is recommended that you locate GUI elements using IDE and once successfully

identified export the code to webdriver.

Here is a sample code that locates an element by its id. Facebook is used as the Base URL.

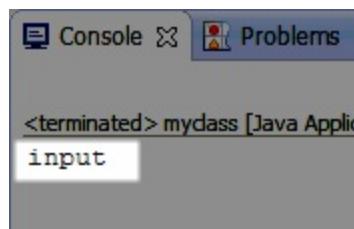
```
package newproject;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class PG2 {
    public static void main(String[] args) {

        System.setProperty("webdriver.firefox.marionette", "C:\\geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        String baseUrl = "http://www.facebook.com";
        String tagName = "";

        driver.get(baseUrl);
        tagName =
        driver.findElement(By.id("email")).getTagName();
        System.out.println(tagName);
        driver.close();
        System.exit(0);
    }
}
```

We used the **getTagName()** method to extract the tag name of that particular element whose id is "email". When run, this code should be able to correctly identify the tag name "input" and will print it out on Eclipse's Console window.



Summary for locating elements

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

Note on Using findElement(By.cssSelector())

By.cssSelector() does not support the "contains" feature.

Consider the Selenium IDE code below -

In Selenium IDE, this step passed

Command	Target	Value
open		
storeText	css=font:contains("Password:")	var
echo		<code> \${var}</code>

Log	Reference	UI-Element	Rollup	
[info] Executing: open				
[info] Executing: storeText				
css=font:contains("Password:") var				
[info] Executing: echo \${var}				
[info] echo: Password:				

The inner text was successfully printed

In Selenium IDE above, the entire test passed. However in the WebDriver script below, the same test generated an error because WebDriver does not support the "contains" keyword when used in the By.cssSelector() method.

```

1 package mypackage;
2
3④ import org.openqa.selenium.*;
4 import org.openqa.selenium.firefox.FirefoxDriver;
5
6 public class myclass {
7
8④     public static void main(String[] args) {
9         WebDriver driver = new FirefoxDriver();
10        String baseUrl = "http://newtours.demoaut.com";
11        String var = "";
12
13        driver.get(baseUrl);
14        var = driver.findElement(By.cssSelector("font:contains('Password:')")).getText();
15        System.out.println(var);
16
17        driver.close();
18        System.exit(0);
19    }
20 }
21

```

this line caused an error
because the 'contains'
keyword is not supported
by By.cssSelector() in
WebDriver

Eclipse IDE reports that error was caused by line 14,
the line where By.cssSelector() was used

at mypackage.myclass.main(myclass.java:14)
Caused by: org.openqa.selenium.remote.ErrorHandler\$UnknownServerException: An invalid or illegal string was specified

Common Commands

Instantiating Web Elements

Instead of using the long "driver.findElement(By.locator())" syntax every time you will access a particular element, we can instantiate a WebElement object for it. The WebElement class is contained in the "org.openqa.selenium.*" package.

```
WebElement myElement = driver.findElement(By.id("username"));
myElement.sendKeys("tutorial");
```

Clicking on an Element

Clicking is perhaps the most common way of interacting with web

elements. **The click() method is used to simulate the clicking of any element.** The following example shows how click() was used to click on Mercury Tours' "Sign-In" button.

```
driver.findElement(By.name("login")).click();
```

Following things must be noted when using the click() method.

- **It does not take any parameter/argument.**
- The method **automatically waits for a new page to load** if applicable.
- The element to be clicked-on, **must be visible** (height and width must not be equal to zero).

Get Commands

Get commands fetch various important information about the page/element. Here are some important "get" commands you must be familiar with.

get() <i>Sample usage:</i>	<ul style="list-style-type: none"> • It automatically opens a new browser window and fetches the page that you specify inside its parentheses. • It is the counterpart of Selenium IDE's "open" command. • The parameter must be a String object.
getTitle() <i>Sample usage:</i>	<ul style="list-style-type: none"> • Needs no parameters • Fetches the title of the current page • Leading and trailing white spaces are trimmed • Returns a null string if the page has no title
getPageSource() <i>Sample usage:</i>	<ul style="list-style-type: none"> • Needs no parameters • Returns the source code of the page as a String value
getCurrentUrl()	

<i>Sample usage:</i>	<ul style="list-style-type: none"> • Needs no parameters • Fetches the string representing the current URL that the browser is looking at
getText() <i>Sample usage:</i>	<ul style="list-style-type: none"> • Fetches the inner text of the element that you specify

Navigate commands

These commands allow you to refresh, go-into and switch back and forth between different web pages.

navigate().to() <i>Sample usage:</i>	<ul style="list-style-type: none"> • It automatically opens a new browser window and fetches the page that you specify inside its parentheses. • It does exactly the same thing as the get() method.
navigate().refresh() <i>Sample usage:</i>	<ul style="list-style-type: none"> • Needs no parameters. • It refreshes the current page.
navigate().back() <i>Sample usage:</i>	<ul style="list-style-type: none"> • Needs no parameters • Takes you back by one page on the browser's history.
navigate().forward() <i>Sample usage:</i>	<ul style="list-style-type: none"> • Needs no parameters • Takes you forward by one page on the browser's history.

Closing and Quitting Browser Windows

close() Sample usage:

- Needs no parameters
- **It closes only the browser window that WebDriver is currently controlling.**

quit() Sample usage:

- Needs no parameters
- **It closes all windows that WebDriver has opened.**



close()

- Will only close a
single window



quit()

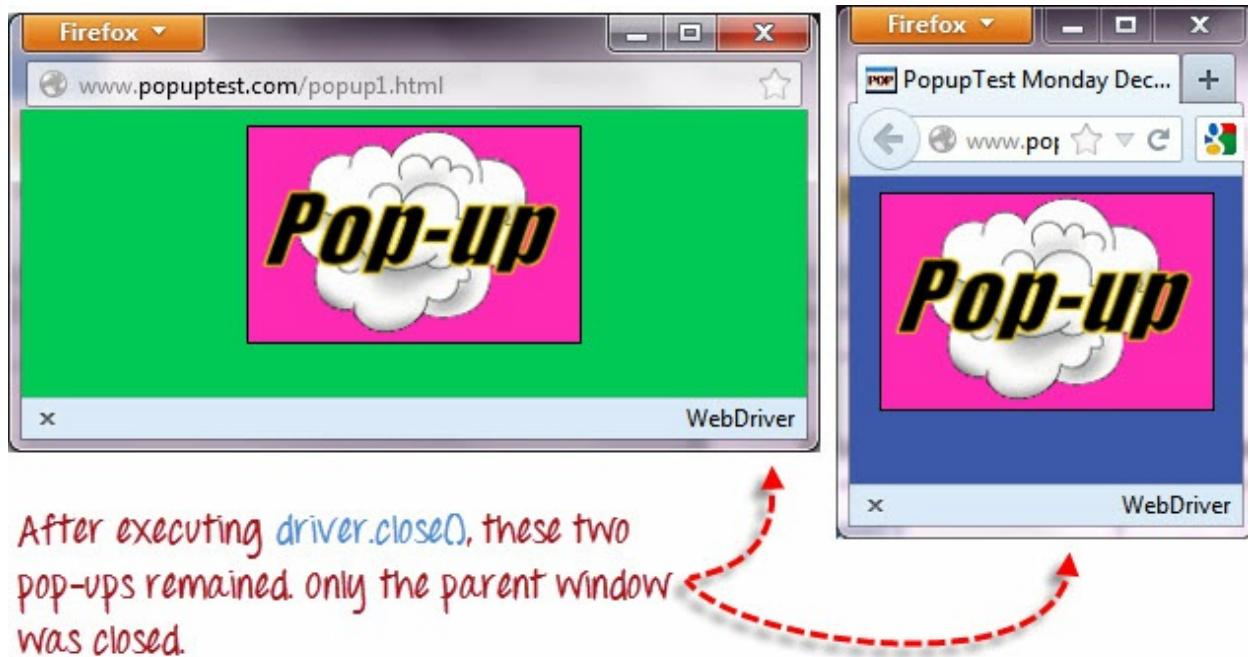
- Will close all
windows

To clearly illustrate the difference between close() and quit(), try to execute the code below. It uses a webpage that automatically pops up a window upon page load and opens up another after exiting.

Using close()

```
public static void main(String[] args) {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://www.popuptest.com/popuptest2.html");
    driver.close();
}
```

Notice that only the parent browser window was closed and not the two pop-up windows.



But if you use `quit()`, all windows will be closed - not just the parent one. Try running the code below and you will notice that the two pop-ups above will automatically be closed as well.

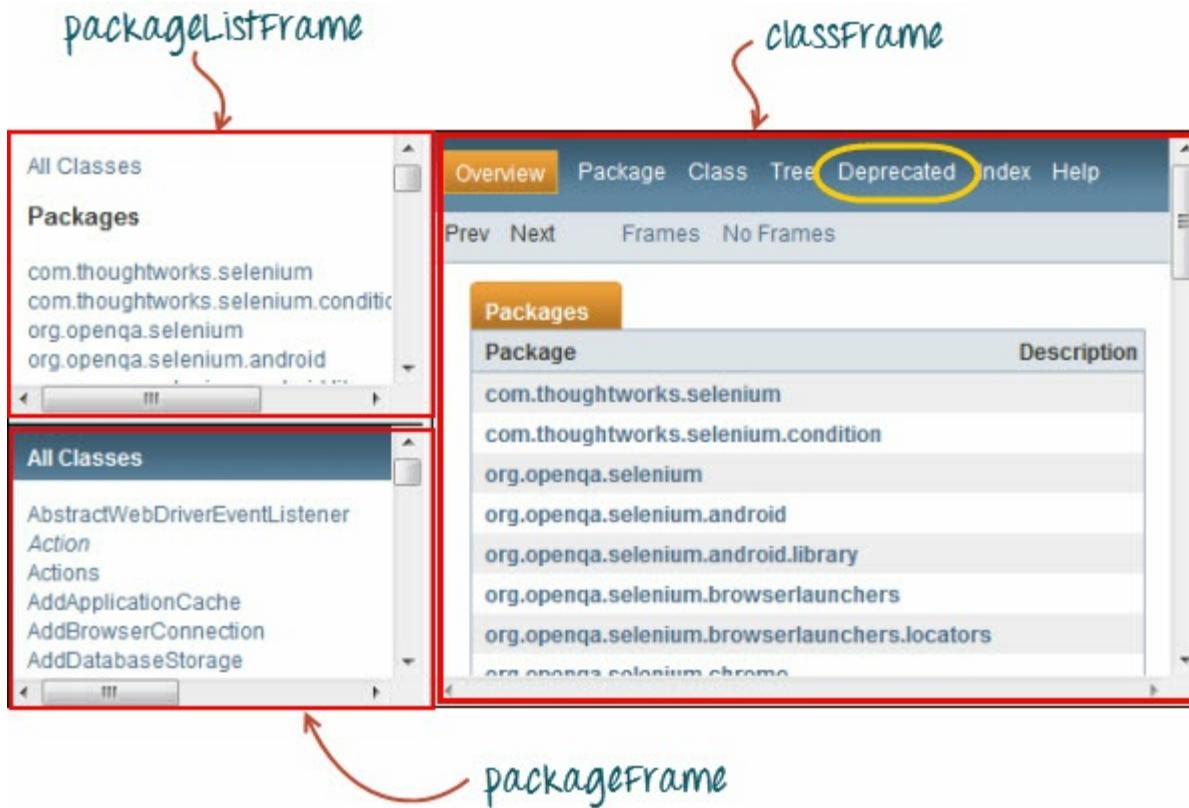
```
package newproject;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class PG3 {
    public static void main(String[] args) {

        System.setProperty("webdriver.firefox.marionette", "C:\\geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        driver.get("http://www.popuptest.com/popuptest2.html");
        driver.quit(); // using QUIT all windows will close
    }
}
```

Switching Between Frames

To access GUI elements in a Frame, we should first direct WebDriver to focus on the frame or pop-up window first before we can access elements within them. Let us take, for example, the web page <http://demo.guru99.com/selenium/deprecated.html>



This page has 3 frames whose "name" attributes are indicated above. We wish to access the "Deprecated" link encircled above in yellow. In order to do that, we must first instruct WebDriver to switch to the "classFrame" frame using the "**switchTo().frame()**" method. We will use the name attribute of the frame as the parameter for the "frame()" part.

```
package newproject;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class PG4 {
```

```
public static void main(String[] args) {  
  
System.setProperty("webdriver.firefox.marionette", "C:\\geckodriver.exe");  
    WebDriver driver = new FirefoxDriver();  
  
driver.get("http://demo.guru99.com/selenium/deprecated.html");  
    driver.switchTo().frame("classFrame");  
  
driver.findElement(By.linkText("Deprecated")).click();  
    driver.close();  
}  
}
```

After executing this code, you will see that the "classFrame" frame is taken to the "Deprecated API" page, meaning that our code was successfully able to access the "Deprecated" link.

Switching Between Pop-up Windows

WebDriver allows pop-up windows like alerts to be displayed, unlike in Selenium IDE. To access the elements within the alert (such as the message it contains), we must use the "**switchTo().alert()**" method. In the code below, we will use this method to access the alert box and then retrieve its message using the "**getText()**" method, and then automatically close the alert box using the "**switchTo().alert().accept()**" method.

First, head over to <http://jsbin.com/usidix/1> and manually click the "Go!" button there and see for yourself the message text.

Click the button below to launch an alert.



Lets see the WebDriver code to do this-

```
package mypackage;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class myclass {

    public static void main(String[] args) {

        System.setProperty("webdriver.firefox.marionette", "C:\\geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        String alertMessage = "";

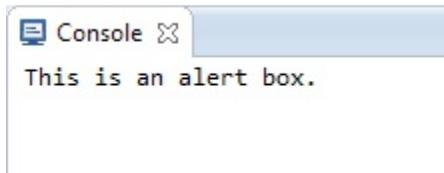
        driver.get("http://jsbin.com/usidix/1");

        driver.findElement(By.cssSelector("input[value=\"Go!\"]")).click();
        alertMessage = driver.switchTo().alert().getText();
        driver.switchTo().alert().accept();

        System.out.println(alertMessage);
        driver.quit();

    }
}
```

On the Eclipse console, notice that the printed alert message is:



Waits

There are two kinds of waits.

1. Implicit wait - used to set the default waiting time throughout the program
2. 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.

```
import java.util.concurrent.TimeUnit;
```

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

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

this means that you are setting 10 seconds as your default wait time. You can change "10" and "SECONDS" to any number and time unit you want.

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:

```
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
```

Step 2

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

WebDriver instance that
will use the Explicit wait

```
WebDriver driver = new FirefoxDriver();
WebDriverWait myWaitVar = new WebDriverWait(driver, 10);
```

number of seconds to wait

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.

```
myWaitVar.until(ExpectedConditions.visibilityOfElementLocated(By.id("username")));
driver.findElement(By.id("username")).sendKeys("tutorial");
```

Conditions

Following methods are used in conditional and looping operations --

- **isEnabled()** is used when you want to verify whether a certain element is enabled or not before executing a command.

for convenience, we saved the element with id="username" as an instance of the WebElement class. The WebElement class is contained in the package `org.openqa.selenium.*`

```
WebElement txtbox_username = driver.findElement(By.id("username"));
if(txtbox_username.isEnabled()){
    txtbox_username.sendKeys("tutorial");
}
```

- **isDisplayed()** is used when you want to verify whether a certain element is displayed or not before executing a command.

```
do{
    //do something here
}while (driver.findElement(By.id("username")).isDisplayed());
```

- **isSelected()** is used when you want to verify whether a certain **check box, radio button, or option in a drop-down box** is selected. It does not work on other elements.

```
//"one-way" and "two-way" are radio buttons
if (driver.findElement(By.id("one-way")).isSelected()) {
    driver.findElement(By.id("two-way")).click();
}
```

Using ExpectedConditions

The ExpectedConditions class offers a wider set of conditions that you can use in conjunction with WebDriverWait's until() method.

Below are some of the most common ExpectedConditions methods.

- **alertIsPresent()** - waits until an alert box is displayed.

```
if (myWaitVar.until(ExpectedConditions.alertIsPresent()) != null) {
    System.out.println("alert is present!");
}
```

- **elementToBeClickable()** - Waits until an element is visible and, at the same time, enabled. The sample code below will wait until the element with id="username" to become visible and enabled first before assigning that element as a WebElement variable named "txtUserName".

```
WebElement txtUserName = myWaitVar.until(ExpectedConditions
    .elementToBeClickable(By.id("username")));
```

- **frameToBeAvailableAndSwitchToIt()** - Waits until the given frame is already available, and then automatically switches to it.

This will automatically switch to the "viewIFRAME" frame once it becomes available.

```
myWaitVar.until(ExpectedConditions
    .frameToBeAvailableAndSwitchToIt("viewIFRAME"));
```

Catching Exceptions

When using isEnabled(), isDisplayed(), and isSelected(), WebDriver assumes that the element already exists on the page. Otherwise, it will throw a **NoSuchElementException**. To avoid this, we should use a try-catch block so that the program will not be interrupted.

```
WebElement txtbox_username =
driver.findElement(By.id("username"));
try{
    if(txtbox_username.isEnabled()){
        txtbox_username.sendKeys("tutorial");
    }
}

catch(NoSuchElementException nsee){
    System.out.println(nsee.toString());
}
```

If you use explicit waits, the type of exception that you should catch is the "TimeoutException".

```
WebDriverWait myWaitVar = new WebDriverWait(driver, 3);
try {
    myWaitVar.until(ExpectedConditions.visibilityOfElementLocated(By
        .id("username")));
    driver.findElement(By.id("username")).sendKeys("tutorial");
} catch (TimeoutException toe) {
    System.out.println(toe.toString());
}
```

Summary

- To start using the WebDriver API, you must import at least these two packages.
 - **org.openqa.selenium.***

- **org.openqa.selenium.firefox.FirefoxDriver**
- The **get()** method is the equivalent of Selenium IDE's "open" command.
- Locating elements in WebDriver is done by using the **findElement()** method.
- The following are the available options for locating elements in WebDriver:
 - By.**className**
 - By.**cssSelector**
 - By.**id**
 - By.**linkText**
 - By.**name**
 - By.**partialLinkText**
 - By.**tagName**
 - By.**xpath**
- The By.cssSelector() **does not** support the "**contains**" feature.
- You can instantiate an element using the **WebElement** class.
- Clicking on an element is done by using the **click()** method.
- WebDriver provides these useful **get commands**:
 - get()
 - getTitle()
 - getPageSource()
 - getCurrentUrl()
 - getText()
- WebDriver provides these useful **navigation commands**
 - navigate().forward()
 - navigate().back()
 - navigate().to()
 - navigate().refresh()
- The close() and quit() methods are used to close browser

windows. **Close()** is used to close a single window; while **quit()** is used to close all windows associated to the parent window that the WebDriver object was controlling.

- The **switchTo().frame()** and **switchTo().alert()** methods are used to direct WebDriver's focus onto a frame or alert, respectively.
- **Implicit waits** are used to set the waiting time throughout the program, while **explicit waits** are used only on specific portions.
- You can use the **isEnabled()**, **isDisplayed()**, **isSelected()**, and a combination of **WebDriverWait** and **ExpectedConditions** methods when verifying the state of an element. However, they do not verify if the element exists.
- When **isEnabled()**, **isDisplayed()**, or **isSelected()** was called while the element was not existing, WebDriver will throw a **NoSuchElementException**.
- When **WebDriverWait** and **ExpectedConditions** methods were called while the element was not existing, WebDriver would throw a **TimeoutException**.

Note:

`driver.get()` : It's used to go to the particular website , But it doesn't maintain the browser History and cookies so , we can't use forward and backward button , if we click on that , page will not get schedule

`driver.navigate()` : it's used to go to the particular website , but it maintains the browser history and cookies, so we can use forward and backward button to navigate between the pages during the coding of Testcase

Chapter 11: Accessing Forms in Webdriver

Forms are the fundamental web elements to receive information from the website visitors. Web forms have different GUI elements like Text boxes, Password fields, Checkboxes, Radio buttons, dropdowns, file inputs, etc.

We will see how to access these different form elements using Selenium Web Driver with Java. **Selenium encapsulates every form element as an object of WebElement**. It provides API to find the elements and take action on them like entering text into text boxes, clicking the buttons, etc. We will see the methods that are available to access each form element.

Introduction to WebElement, **findElement()**, **findElements()**

Selenium Web Driver encapsulates a simple form element as an object of **WebElement**.

There are various techniques by which the WebDriver identifies the form elements based on the different properties of the Web elements like ID, Name, Class, XPath, Tagname, CSS Selectors, link Text, etc.

Web Driver provides the following two methods to find the elements.

- **findElement()** – finds a single web element and returns as a

WebElement object.

- **findElements()** – returns a list of WebElement objects matching the locator criteria.

Let's see the code snippets to get a single element – Text Field in a web page as an object of WebElement using findElement() method. We shall cover the findElements() method of finding multiple elements in subsequent tutorials.

Step 1: We need to import this package to create objects of Web Elements

```
import org.openqa.selenium.WebElement;
```

Step 2: We need to call the findElement() method available on the WebDriver class and get an object of WebElement.

Refer below to see how it is done.

Input Box

Input boxes refer to either of these two types:

1. **Text Fields**- text boxes that accept typed values and show them as they are.
2. **Password Fields**- text boxes that accept typed values but mask them as a series of special characters (commonly dots and asterisks) to avoid sensitive values to be displayed.



Locators

The method `findElement()` takes one parameter which is a locator to the element. Different locators like `By.id()`, `By.name()`, `By.xpath()`, `By.CSSSelector()` etc. locate the elements in the page using their properties like ` id, name or path, etc.

You can use plugins like Fire path to get help with getting the id, xpath, etc. of the elements.

Using the example site <http://demo.guru99.com/test/login.html> given below is the code to locate the "Email address" text field using the id locator and the "Password" field using the name locator.

```
// Get the WebElement corresponding to the Email Address(TextField)
WebElement email = driver.findElement(By.id("email")); ①

// Retrieve the WebElement corresponding to the Password Field
WebElement password = driver.findElement(By.name("passwd")); ②
```

- 1) email text field is located by ID
- 2) Password field is located by name

1. Email text field is located by Id
2. Password field is located by name

Entering Values in Input Boxes

To enter text into the Text Fields and Password Fields, sendKeys() is the method available on the WebElement.

Using the same example of <http://demo.guru99.com/test/login.html> site, here is how we find the Text field and Password fields and enter values into them.

The screenshot shows a login form with two input fields: "Email address" and "Password". The "Email address" field contains "abcd@gmail.com" and has a green checkmark icon. The "Password" field contains a masked password "*****". A red arrow points from step 3 to the "Email address" field, and another red arrow points from step 4 to the "Password" field. To the right, a callout box contains the following numbered steps:

- 1) Find the "Email Address" Text Field using id locator
- 2) Find the "Password" Field using name locator
- 3) Enter text into the "Email Address" using the sendKeys() method
- 4) Enter password into the "Password" using sendKeys()

```

// Get the WebElement corresponding to the Email Address(TextField)
WebElement email = driver.findElement(By.id("email")); 1

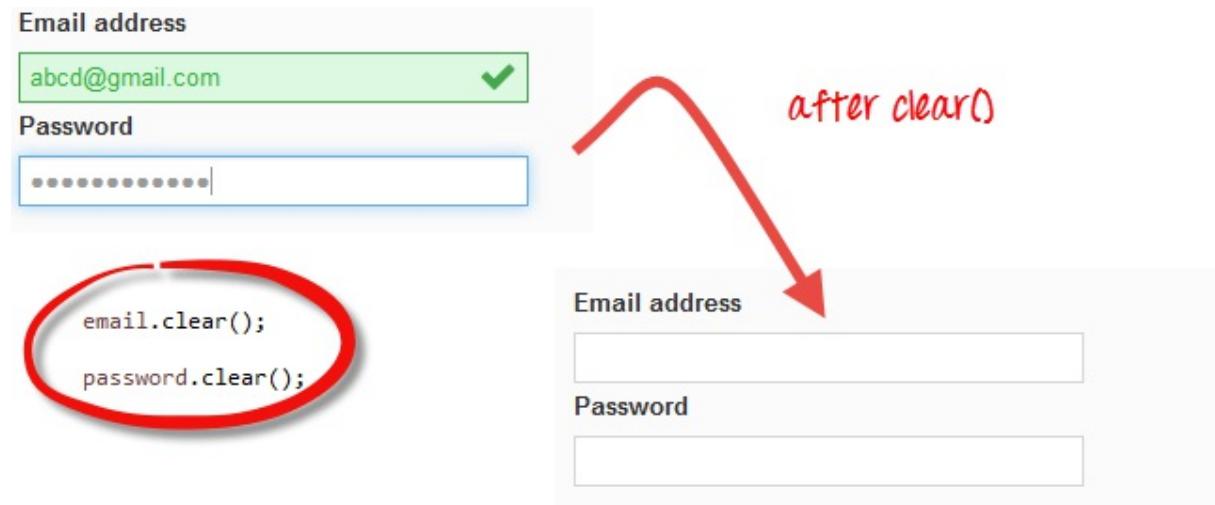
// Retrieve the WebElement corresponding to the Password Field
WebElement password = driver.findElement(By.name("passwd")); 2

email.sendKeys("abcd@gmail.com"); 3
password.sendKeys("abcdefghijklm"); 4
  
```

1. Find the "Email Address" Text field using the id locator.
2. Find the "Password" field using the name locator
3. Enter text into the "Email Address" using the sendKeys() method.
4. Enter a password into the "Password" field using the sendKeys() method.

Deleting Values in Input Boxes

The **clear()** method is used to delete the text in an input box. **This method does not need a parameter**. The code snippet below will clear out the text from the Email or Password fields



Buttons

The buttons can be accessed using the click() method.

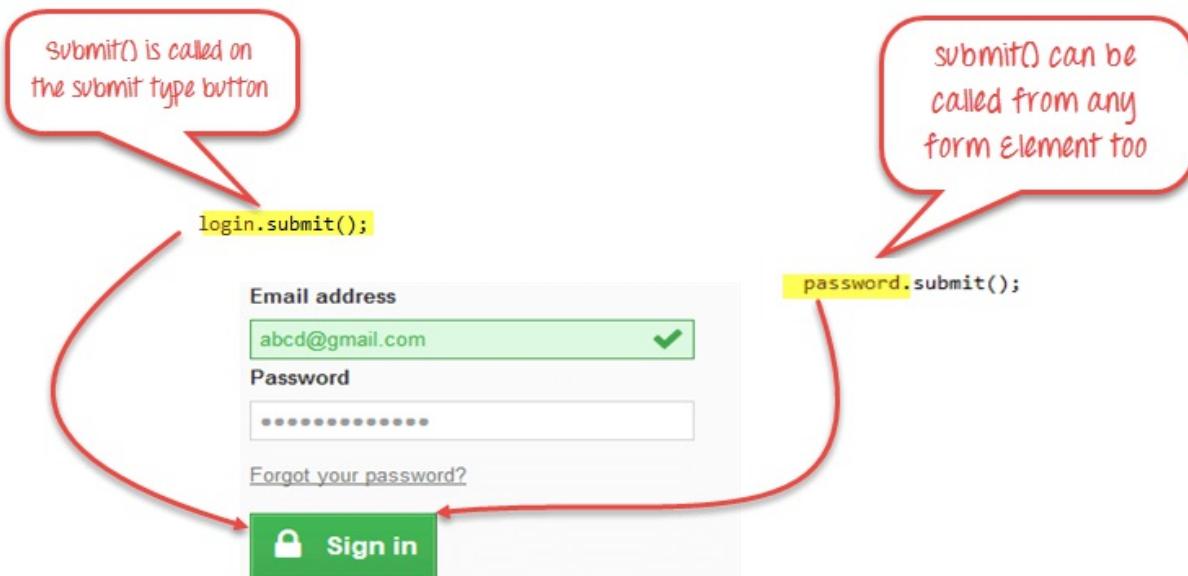
In the example above

1. Find the button to Sign in
2. Click on the "Sign-in" Button in the login page of the site to login to the site.



Submit Buttons

Submit buttons are used to submit the entire form to the server. We can either use the click () method on the web element like a normal button as we have done above or use the submit () method on any web element in the form or on the submit button itself.

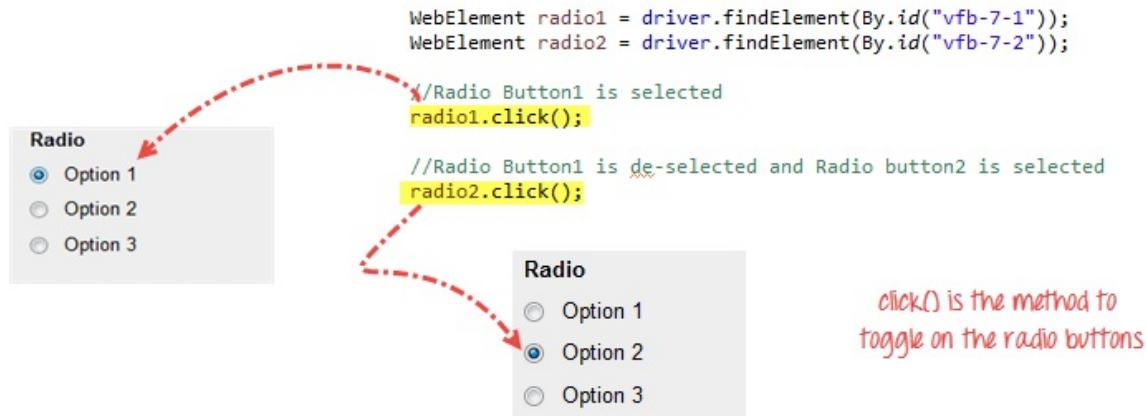


When submit() is used, WebDriver will look up the DOM to know which form the element belongs to, and then trigger its submit function.

Radio Button

Radio Buttons too can be toggled on by using the click() method.

Using <http://demo.guru99.com/test/radio.html> for practise, see that radio1.click() toggles on the "Option1" radio button. radio2.click() toggles on the "Option2" radio button leaving the "Option1" unselected.



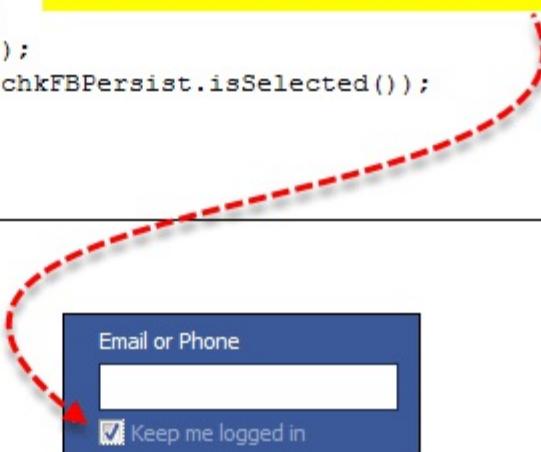
Check Box

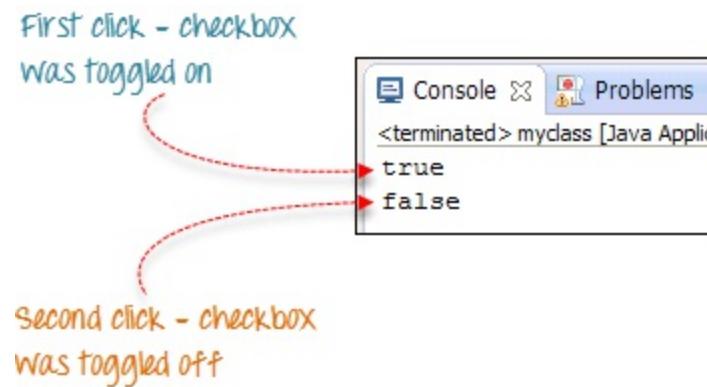
Toggling a check box on/off is also done using the **click()** method.

The code below will click on Facebook's "Keep me logged in" check box twice and then output the result as TRUE when it is toggled on, and FALSE if it is toggled off.

```
public static void main(String[] args) {
    WebDriver driver = new FirefoxDriver();
    String baseURL = "http://www.facebook.com";

    driver.get(baseURL);
    WebElement chkFBPersist = driver.findElement(By.id("persist_box"));
    for(int i=0; i<2; i++){
        chkFBPersist.click();
        System.out.println(chkFBPersist.isSelected());
    }
    driver.quit();
}
```





`isSelected()` method is used to know whether the Checkbox is toggled on or off.

Here is another example: <http://demo.guru99.com/test/radio.html>

```
@Test
public void tryCheckbox(){

    WebElement option1 = driver.findElement(By.id("vfb-6-0")); ①

    //This will Toggle On the Check box
    option1.click(); ②

    //Check whether the Check box is toggled on
    if(option1.isSelected()); ③
        System.out.println("Checkbox is Toggled On");

    }else{
        System.out.println("Checkbox is Toggled Off");
    }

    // This should Toggle Off the Check box
    option1.click(); ④

    // Lets see whether its Toggled Off
    if(!option1.isSelected()){
        System.out.println("Checkbox is now Toggled Off !!");
    }
}
```

1. Locate the checkbox element by its ID

2. `click()` toggles on the checkbox

3. `isSelected()` gives the toggle status of the checkbox

4. `click()` again on the checkbox turns the toggle off

Complete Code

Here is the complete working code

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.*;

public class Form {
    public static void main(String[] args) {

        // declaration and instantiation of objects/variables
        System.setProperty("webdriver.chrome.driver", "G:\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();

        String baseUrl =
        "http://demo.guru99.com/test/login.html";
        driver.get(baseUrl);

        // Get the WebElement corresponding to the Email
        Address(TextField)
        WebElement email = driver.findElement(By.id("email"));

        // Get the WebElement corresponding to the Password
        Field
        WebElement password =
        driver.findElement(By.name("passwd"));

        email.sendKeys("abcd@gmail.com");
        password.sendKeys("abcdefghijkl");
        System.out.println("Text Field Set");

        // Deleting values in the text box
        email.clear();
        password.clear();
        System.out.println("Text Field Cleared");

        // Find the submit button
        WebElement login =
        driver.findElement(By.id("SubmitLogin"));

        // Using click method to submit form
        email.sendKeys("abcd@gmail.com");
        password.sendKeys("abcdefghijkl");
```

```
login.click();
System.out.println("Login Done with Click");

//using submit method to submit the form. Submit used on
password field
driver.get(baseUrl);

driver.findElement(By.id("email")).sendKeys("abcd@gmail.com");

driver.findElement(By.name("passwd")).sendKeys("abcdefghijkl");
driver.findElement(By.id("SubmitLogin")).submit();
System.out.println("Login Done with Submit");

driver.get("http://demo.guru99.com/test/radio.html");
WebElement radio1 = driver.findElement(By.id("vfb-7-
1"));
WebElement radio2 = driver.findElement(By.id("vfb-7-
2"));

//Radio Button1 is selected
radio1.click();
System.out.println("Radio Button Option 1 Selected");

//Radio Button1 is de-selected and Radio Button2 is
selected
radio2.click();
System.out.println("Radio Button Option 2 Selected");

// Selecting CheckBox
WebElement option1 = driver.findElement(By.id("vfb-6-
0"));

// This will Toggle the Check box
option1.click();

// Check whether the Check box is toggled on
if (option1.isSelected()) {
    System.out.println("Checkbox is Toggled On");

} else {
    System.out.println("Checkbox is Toggled Off");
}
```

```
//Selecting Checkbox and using isSelected Method
driver.get("http://demo.guru99.com/test/facebook.html");
WebElement chkFBPersist =
driver.findElement(By.id("persist_box"));
for (int i=0; i<2; i++) {
    chkFBPersist.click ();
    System.out.println("Facebook Persists Checkbox
Status is - "+chkFBPersist.isSelected());
}
//driver.close();

}
```

Troubleshooting

If you encounter NoSuchElementException() while finding elements, it means that the element is not found in the page at the point the Web driver accessed the page.

1. Check your locator again using Firepath or Inspect Element in Chrome.
2. Check whether the value you used in the code is different from the one for the element in Firepath now.
3. Some properties are dynamic for few elements. In case, you find that the value is different and is changing dynamically, consider using By.xpath() or By.cssSelector() which are more reliable but complex ways.
4. Sometimes, it could be a wait issue too i.e., the Web driver executed your code even before the page loaded completely, etc.
5. Add a wait before findElement() using implicit or explicit waits.

Summary

- 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.
Submit Button	submit()	

- WebDriver allows selection of more than one option in a multiple SELECT element.
- 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.

Chapter 12: How to Select Option from DropDown using Selenium Webdriver

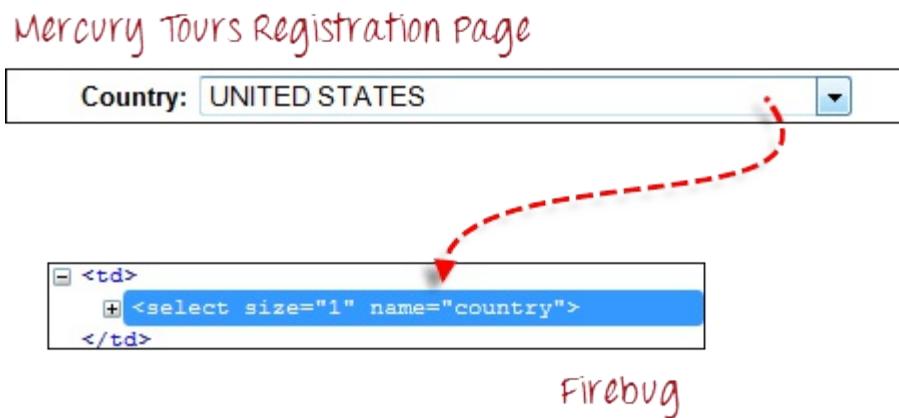
In this tutorial, we will learn how to handle Drop Down and Multiple Select Operations.

Drop-Down Box

Before we can control drop-down boxes, we must do following two things:

1. Import the package **org.openqa.selenium.support.ui.Select**
2. Instantiate the drop-down box as a "Select" object in WebDriver

As an example, go to Mercury Tours' Registration page (<http://demo.guru99.com/test/newtours/register.php>) and notice the "Country" drop-down box there.



Step 1

Import the "Select" package.

```
import org.openqa.selenium.support.ui.Select;
```

Step 2

Declare the drop-down element as an instance of the Select class. In the example below, we named this instance as "drpCountry".

```
Select drpCountry = new Select(driver.findElement(By.name("country")));
```

Step 3

We can now start controlling "drpCountry" by using any of the available Select methods. The sample code below will select the option "ANTARCTICA."

```
drpCountry.selectByVisibleText("ANTARCTICA");
```

Selecting Items in a Multiple SELECT elements

We can also use the **selectByVisibleText()** method in selecting multiple options in a multi SELECT element. As an example, we will take <http://jsbin.com/osebed/2> as the base URL. It contains a drop-down box that allows multiple selections at a time.

Page source

```
<select id="fruits" multiple="">
    <option value="banana">Banana</option>
    <option value="apple">Apple</option>
    <option value="orange">Orange</option>
    <option value="grape">Grape</option>
</select>
```



HTML page

The code below will select the first two options using the `selectByVisibleText()` method.

```
public static void main(String[] args) {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://jsbin.com/osebed/2");
    Select fruits = new Select(driver.findElement(By.id("fruits")));
    fruits.selectByVisibleText("Banana");
    fruits.selectByIndex(1);
}
```



Select Methods

The following are the most common methods used on drop-down elements.

Method	Description
selectByVisibleText() and deselectByVisibleText()	<ul style="list-style-type: none"> • Selects/deselects the option

Example:

```
drpCountry.selectByVisibleText("ANTARCTICA");
```

that displays the text matching the parameter.

- **Parameter:** The exactly displayed text of a particular option

selectByValue() and deselectByValue()

Example:

```
drpCountry.selectByValue("234");
```

- Selects/deselects the option whose "value" attribute matches the specified parameter.
- **Parameter:** value of the "value" attribute
- Remember that not all drop-down options have the same text and "value", like in the example below.

```
<option value="10">ANGUILLA </option>
<option value="234">ANTARCTICA </option>
<option value="1">ANTIGUA AND BARBUDA </option>
```

selectByIndex() and deselectByIndex()

Example:

```
drpCountry.selectByIndex(0);
```

- Selects/deselects the option at the given index.
- **Parameter:** the index of the option to be selected.

isMultiple()

Example:

```
if (drpCountry.isMultiple()) {
    //do something here
}
```

- Returns TRUE if the drop-down element allows multiple selections at a time; FALSE if otherwise.
- **No parameters needed**

deselectAll()

Example:

```
drpCountry.deselectAll();
```

- Clears all selected entries. This is only valid when the drop-down element supports multiple selections.
- **No parameters needed**

Here is the complete code

```

package newpackage;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;
import org.openqa.selenium.By;

public class accessDropDown {
    public static void main(String[] args) {

        System.setProperty("webdriver.firefox.marionette", "C:\\geckodriver.exe");
        String baseURL =
        "http://demo.guru99.com/test/newtours/register.php";
        WebDriver driver = new FirefoxDriver();
        driver.get(baseURL);

        Select drpCountry = new
        Select(driver.findElement(By.name("country")));
        drpCountry.selectByVisibleText("ANTARCTICA");

        //Selecting Items in a Multiple SELECT elements
        driver.get("http://jsbin.com/osebed/2");
        Select fruits = new
        Select(driver.findElement(By.id("fruits")));
        fruits.selectByVisibleText("Banana");
        fruits.selectByIndex(1);
    }
}

```

Summary

Element	Command	Description
Drop-Down Box	<i>selectByVisibleText()</i> / <i>deselectByVisibleText()</i>	selects/deselects an option by its displayed text
	<i>selectByValue()</i> / <i>deselectByValue()</i>	selects/deselects an option by the value of its "value" attribute
	<i>selectByIndex()</i>	selects/deselects an option by its index

<i>deselectByIndex()</i>	
<i>isMultiple()</i>	returns TRUE if the drop-down element allows multiple selection at a time; FALSE if otherwise
<i>deselectAll()</i>	deselects all previously selected options

To control drop-down boxes, you must first import the org.openqa.selenium.support.ui.Select package and then create a Select instance.

Chapter 13: Accessing Links & Tables using Selenium Webdriver

In this tutorial, we will learn the available methods to find and access the Links & Tables using Webdriver. Also, we will discuss some of the common problems faced while accessing Links and will further discuss on how to resolve them.

Part 1) Accessing Links

Links Matching a Criterion

Links can be accessed using an exact or partial match of their link text. The examples below provide scenarios where multiple matches would exist and would explain how WebDriver would deal with them.

Exact Match

Accessing links using their exact link text is done through the `By.linkText()` method. However, if there are two links that have the very same link text, this method will only access the first one. Consider the HTML code below

```
<html>
  <head>
    <title>Sample</title>
  </head>
  <body>
    <a href="http://www.google.com">click here</a>
    <br>
    <a href="http://www.fb.com">click here</a>
  </body>
</html>
```



When you try to run the WebDriver code below, you will be accessing

the first "click here" link

```
public static void main(String[] args) {
    String baseUrl = "file:///D:/newhtml.html";
    WebDriver driver = new FirefoxDriver();

    driver.get(baseUrl);
    driver.findElement(By.linkText("click here")).click();
    System.out.println("Title of page is: " + driver.getTitle());
    driver.quit();
}
```

Code:

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

public class MyClass {

    public static void main(String[] args) {
        String baseUrl =
"http://demo.guru99.com/test/link.html";

        System.setProperty("webdriver.chrome.driver", "G:\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();

        driver.get(baseUrl);
        driver.findElement(By.linkText("click here")).click();
        System.out.println("title of page is: " +
driver.getTitle());
        driver.quit();
    }

}
```

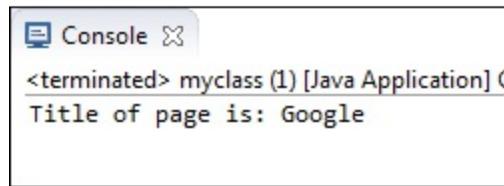
Here is how it works-

```
driver.findElement(By.linkText("Create a new account")).click();
```

findElement() is used to find Links in the page

click() is the method to access links

As a result, you will automatically be taken to Google.



Console X
<terminated> myclass (1) [Java Application] C
Title of page is: Google

Partial Match

Accessing links using a portion of their link text is done using the **By.partialLinkText()** method. If you specify a partial link text that has multiple matches, only the first match will be accessed. Consider the HTML code below.

```
<html>
  <head>
    <title>Partial Match</title>
  </head>
  <body>
    <a href="http://www.google.com">go here</a>
    <br>
    <a href="http://www.fb.com">click here</a>
  </body>
</html>
```



When you execute the WebDriver code below, you will still be taken to Google.

```
public static void main(String[] args) {
    String baseUrl = "file:///D:/partial_match.html";
    WebDriver driver = new FirefoxDriver();

    driver.get(baseUrl);
    driver.findElement(By.partialLinkText("here")).click();
    System.out.println("Title of page is: " + driver.getTitle());
    driver.quit();
}
```

Code:

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

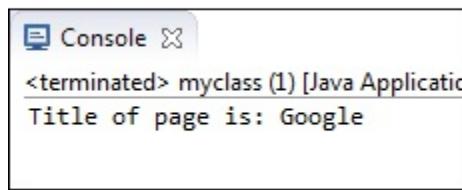
public class P1 {

    public static void main(String[] args) {
        String baseUrl =
"http://demo.guru99.com/test/link.html";

        System.setProperty("webdriver.chrome.driver", "G:\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();

        driver.get(baseUrl);
        driver.findElement(By.partialLinkText("here")).click();
        System.out.println("Title of page is: " +
driver.getTitle());
        driver.quit();
    }
}
```

```
    }  
}
```



A screenshot of a Java application's console window. The title bar says "Console". The text area contains the following output:
<terminated> myclass (1) [Java Application]
Title of page is: Google

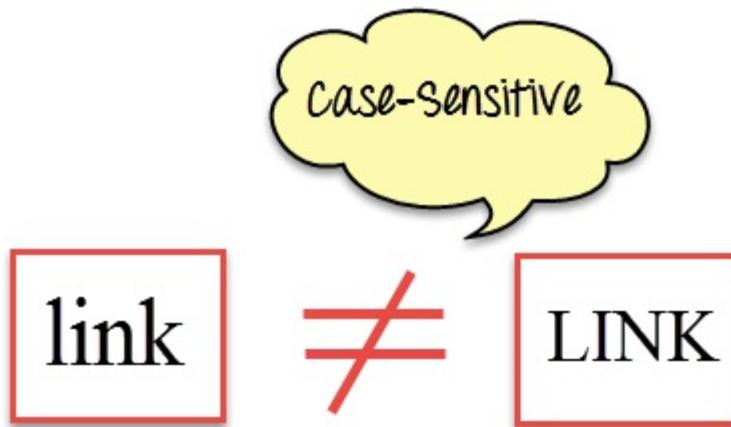
How to get the non-First Link

So, how to get around the above problem? In cases where there are multiple links with the same link text, and we want to access the links other than the first one, how do we go about it?

In such cases, generally, different locators viz... By.xpath(), By.cssSelector() or By.tagName() are used.

Most commonly used is By.xpath(). It is the most reliable one but it looks complex and non-readable too.

Case-sensitivity



The parameters for **By.linkText()** and **By.partialLinkText()** are both case-sensitive, meaning that capitalization matters. For example, in Mercury Tours' homepage, there are two links that contain the text "egis" - one is the "REGISTER" link found at the top menu, and the other is the "Register here" link found at the lower right portion of the page.

The link at the top menu



The link at the lower right portion of the page



Though both links contain the character sequence "egis," one is the "By.partialLinkText()" method will access these two links separately depending on the capitalization of the characters. See the sample code below.

```
public static void main(String[] args) {
    String baseUrl = "http://newtours.demoaut.com/";
    WebDriver driver = new FirefoxDriver();

    driver.get(baseUrl);

    String theLinkText = driver.findElement(By
        .partialLinkText("egis"))
        .getText();
    System.out.println(theLinkText);
    theLinkText = driver.findElement(By
        .partialLinkText("EGIS"))
        .getText();
    System.out.println(theLinkText);

    driver.quit();
}
```

Console X
<terminated> myclass (1)
Register here
REGISTER

Code

```
public static void main(String[] args) {  
    String baseUrl =  
"http://demo.guru99.com/test/newtours/";  
  
System.setProperty("webdriver.chrome.driver", "G:\\chromedriver.e  
xe");  
    WebDriver driver = new ChromeDriver();  
  
    driver.get(baseUrl);  
    String theLinkText = driver.findElement(By  
        .partialLinkText("egis"))  
        .getText();  
    System.out.println(theLinkText);  
    theLinkText = driver.findElement(By  
        .partialLinkText("EGIS"))  
        .getText();  
    System.out.println(theLinkText);  
  
    driver.quit();  
}  
}
```

All Links

One of the common procedures in web Testing is to test if all the links present within the page are working. This can be conveniently done using a combination of the **Java for-each loop**, **findElements()** & **By.tagName("a")** method.

The **findElements()** method, returns a list of Web Elements with tag a. Using a for-each loop, each element is accessed.

```

List<WebElement> allLinks = driver.findElements(By.tagName("a"));

for(WebElement link:allLinks){
    // filter the links with the required text
    if(link.getText().equals("Create a new account")){
        System.out.println(" Link : "+ link.getText());
    }
}

```

The WebDriver code below checks each link from the Mercury Tours homepage to determine those that are working and those that are still under construction.

```

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import java.util.List;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;

public class P1 {

    public static void main(String[] args) {
        String baseUrl =
"http://demo.guru99.com/test/newtours/";

System.setProperty("webdriver.chrome.driver", "G:\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();

        String underConsTitle = "Under Construction: Mercury
Tours";

driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

        driver.get(baseUrl);
        List<WebElement> linkElements =
driver.findElements(By.tagName("a"));
    }
}

```

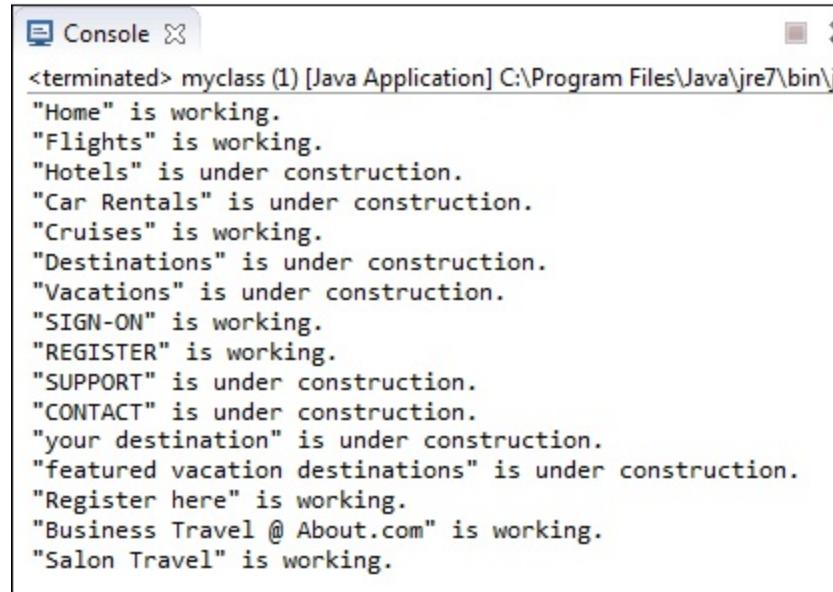
```
String[] linkTexts = new String[linkElements.size()];
int i = 0;

//extract the link texts of each link element
for (WebElement e : linkElements) {
    linkTexts[i] = e.getText();
    i++;
}

//test each link
for (String t : linkTexts) {

    driver.findElement(By.linkText(t)).click();
    if (driver.getTitle().equals(underConstTitle)) {
        System.out.println("'" + t + "'"
            + " is under construction.");
    } else {
        System.out.println("'" + t + "'"
            + " is working.");
    }
    driver.navigate().back();
}
driver.quit();
}
```

The output should be similar to the one indicated below.



The screenshot shows a Java application window titled "Console". The output window contains a list of travel-related links, each followed by a status message: "is working." or "is under construction." The links listed are: "Home", "Flights", "Hotels", "Car Rentals", "Cruises", "Destinations", "Vacations", "SIGN-ON", "REGISTER", "SUPPORT", "CONTACT", "your destination", "featured vacation destinations", "Register here", "Business Travel @ About.com", and "Salon Travel".

```
<terminated> MyClass (1) [Java Application] C:\Program Files\Java\jre7\bin\j
"Home" is working.
"Flights" is working.
"Hotels" is under construction.
"Car Rentals" is under construction.
"Cruises" is working.
"Destinations" is under construction.
"Vacations" is under construction.
"SIGN-ON" is working.
"REGISTER" is working.
"SUPPORT" is under construction.
"CONTACT" is under construction.
"your destination" is under construction.
"featured vacation destinations" is under construction.
"Register here" is working.
"Business Travel @ About.com" is working.
"Salon Travel" is working.
```

Links Outside and Inside a Block

The latest HTML5 standard allows the `<a>` tags to be placed inside and outside of block-level tags like `<div>`, `<p>`, or `<h3>`. The `"By.linkText()"` and `"By.partialLinkText()` methods can access a link located outside and inside these block-level elements. Consider the HTML code below.

```
<body>
  <p>
    <a href="http://www.google.com">Inside a block-level tag.</a>
  </p>

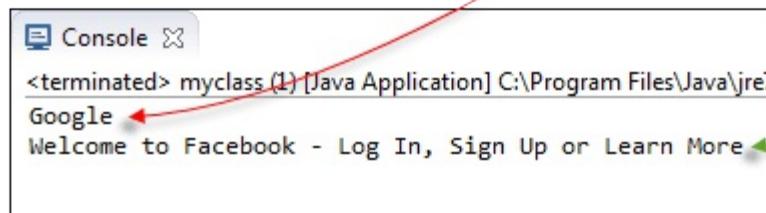
  <br>
  <a href="http://www.fb.com">
    <div>
      <span>Outside a block-level tag.</span>
    </div>
  </a>
</body>
```



The WebDriver code below accesses both of these links using By.partialLinkText() method.

```
public static void main(String[] args) {
    String baseUrl = "file:///D:/Links%20Outside%20and%20Inside%20a%20Block.html";
    WebDriver driver = new FirefoxDriver();

    driver.get(baseUrl);
    driver.findElement(By.partialLinkText("Inside")).click();
    System.out.println(driver.getTitle());
    driver.navigate().back();
    driver.findElement(By.partialLinkText("Outside")).click();
    System.out.println(driver.getTitle());
    driver.quit();
}
```



Code:

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

public class MyClass {
```

```
public static void main(String[] args) {
    String baseUrl =
"http://demo.guru99.com/test/block.html";

System.setProperty("webdriver.chrome.driver", "G:\\chromedriver.exe");
    WebDriver driver = new ChromeDriver();

    driver.get(baseUrl);

driver.findElement(By.partialLinkText("Inside")).click();
    System.out.println(driver.getTitle());
    driver.navigate().back();

driver.findElement(By.partialLinkText("Outside")).click();
    System.out.println(driver.getTitle());
    driver.quit();
}
}
```

The output above confirms that both links were accessed successfully because their respective page titles were retrieved correctly.

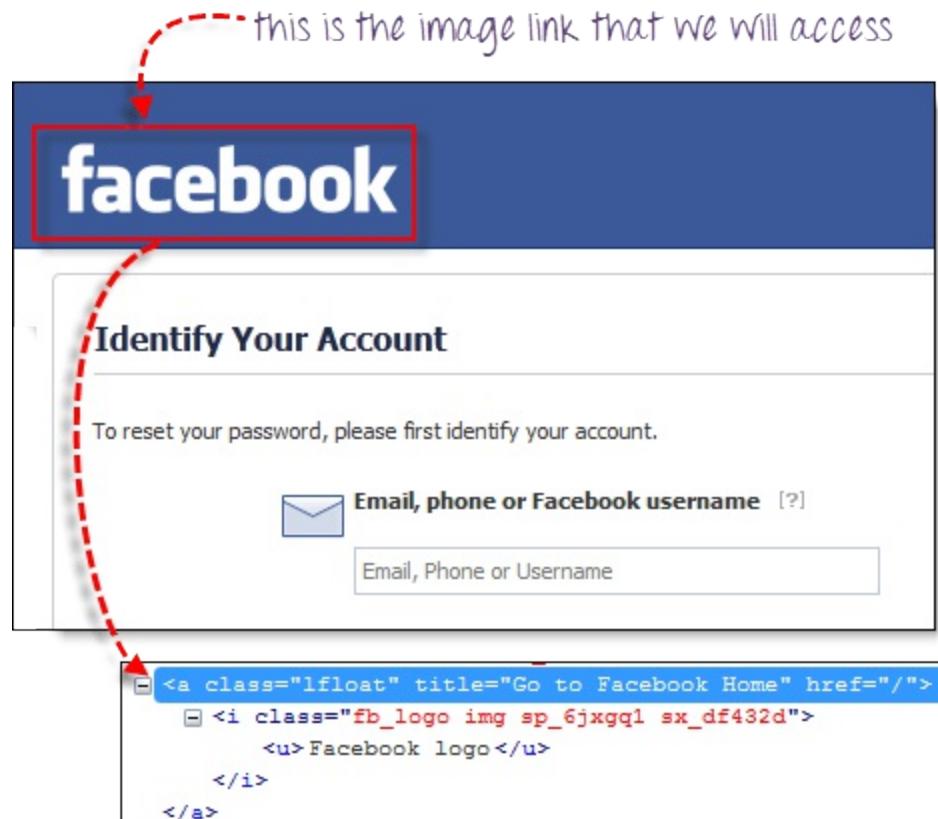
Accessing Image Links

Image links are the links in web pages represented by an image which when clicked navigates to a different window or page.

Since they are images, we cannot use the By.linkText() and By.partialLinkText() methods because image links basically have no link texts at all.

In this case, we should resort to using either By.cssSelector or By.xpath. The first method is more preferred because of its simplicity.

In the example below, we will access the "Facebook" logo on the upper left portion of Facebook's Password Recovery page.



We will use By.cssSelector and the element's "title" attribute to access the image link. And then we will verify if we are taken to Facebook's homepage.

```
package newproject;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class MyClass {

    public static void main(String[] args) {
        String baseUrl =
"https://www.facebook.com/login/identify?ctx=recover";

        System.setProperty("webdriver.chrome.driver", "G:\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();
```

```
driver.get(baseUrl);
//click on the "Facebook" logo on the upper left portion

driver.findElement(By.cssSelector("a[title=\"Go to Facebook
home\"]")).click();

//verify that we are now back on
Facebook's homepage
if (driver.getTitle().equals("Facebook -
log in or sign up")) {
    System.out.println("We are back at Facebook's
homepage");
} else {
    System.out.println("We are NOT in Facebook's
homepage");
}
driver.close();

}
```

Result



The screenshot shows a Java console window titled 'Console'. The title bar also includes the text '<terminated> myclass (1) [Java Application] C:\Program'. The main area of the console displays the message 'We are back at Facebook's homepage'.

Part 2) Reading a Table

There are times when we need to access elements (usually texts) that are within HTML tables. However, it is very seldom for a web designer to provide an id or name attribute to a certain cell in the table. Therefore, we cannot use the usual methods such as "By.id()", "By.name()", or "By.cssSelector()". In this case, the most reliable option is to access them using the "By.xpath()" method.

XPath Syntax

Consider the HTML code below.

```
<html>
    <head>
        <title>Sample</title>
    </head>
    <body>
        <table border="1">
            <tbody>
                <tr>
                    <td>first cell</td>
                    <td>second cell</td>
                </tr>
                <tr>
                    <td>third cell</td>
                    <td>fourth cell</td>
                </tr>
            </tbody>
        </table>
    </body>
</html>
```

We will use XPath to get the inner text of the cell containing the text "fourth cell."



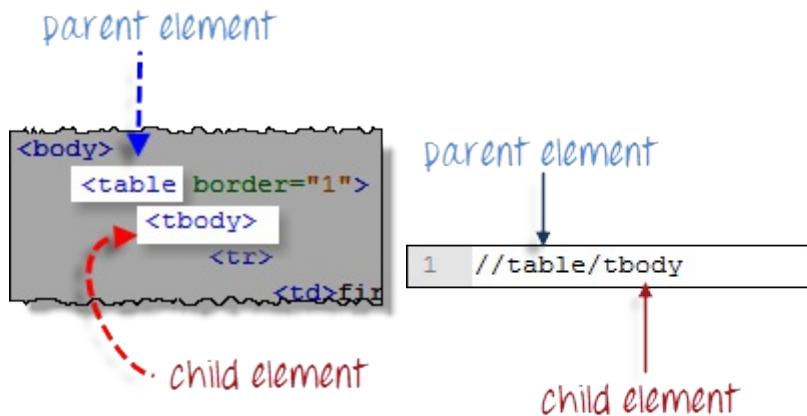
Step 1 - Set the Parent Element (table)

XPath locators in WebDriver always start with a double forward slash "://" and then followed by the parent element. Since we are dealing with tables, the parent element should always be the `<table>` tag. The first portion of our XPath locator should, therefore, start with "`//table`".

```
//table
```

Step 2 - Add the child elements

The element immediately under `<table>` is `<tbody>` so we can say that `<tbody>` is the "child" of `<table>`. And also, `<table>` is the "parent" of `<tbody>`. All child elements in XPath are placed to the right of their parent element, separated with one forward slash "/" like the code shown below.



Step 3 - Add Predicates

The `<tbody>` element contains two `<tr>` tags. We can now say that these two `<tr>` tags are "children" of `<tbody>`. Consequently, we can say that `<tbody>` is the parent of both the `<tr>` elements.

Another thing we can conclude is that the two `<tr>` elements are siblings. **Siblings refer to child elements having the same parent.**

To get to the `<td>` we wish to access (the one with the text "fourth cell"), we must first access the **second** `<tr>` and not the first. If we simply write `//table/tbody/tr`, then we will be accessing the first `<tr>` tag.

So, how do we access the second `<tr>` then? The answer to this is to use **Predicates**.

Predicates are numbers or HTML attributes enclosed in a pair of square brackets "[]" that distinguish a child element from its siblings. Since the `<tr>` we need to access is the second one, we shall use "[2]" as the predicate.

1 //table/tbody/tr[2]

The [2] predicate denotes that we are accessing the 2nd <tr> of the parent <tbody>

If we won't use any predicate, XPath will access the first sibling. Therefore, we can access the first <tr> using either of these XPath codes.

//table/tbody/tr

this will automatically access the first <tr> because no predicate was used

//table/tbody/tr[1]

this will access the first <tr> because the predicate [1] explicitly says it

Step 4 - Add the Succeeding Child Elements Using the Appropriate Predicates

The next element we need to access is the second <td>. Applying the principles we have learned from steps 2 and 3, we will finalize our XPath code to be like the one shown below.

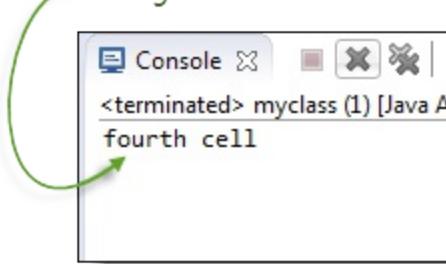
//table/tbody/tr[2]/td[2]

Now that we have the correct XPath locator, we can already access the cell that we wanted to and obtain its inner text using the code below. It

assumes that you have saved the HTML code above as "newhtml.html" within your C Drive.

```
public static void main(String[] args) {  
    String baseUrl = "file:///C:/newhtml.html";  
    WebDriver driver = new FirefoxDriver();  
  
    driver.get(baseUrl);  
    String innerText = driver.findElement(  
        By.xpath("//table/tbody/tr[2]/td[2]")).getText();  
    System.out.println(innerText);  
    driver.quit();  
}
```

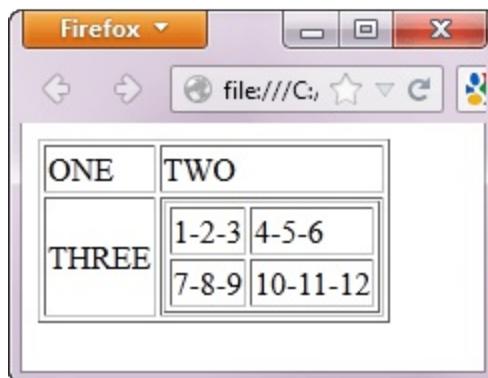
the inner text was
successfully retrieved
using XPath



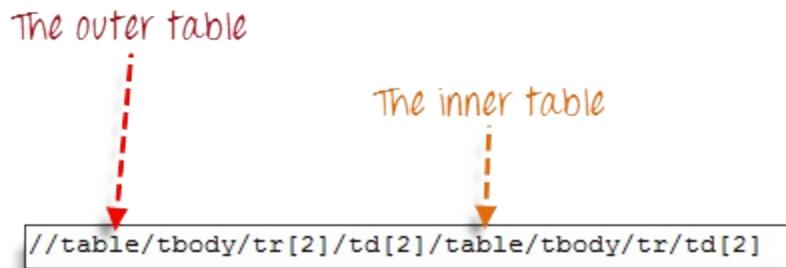
Accessing Nested Tables

The same principles discussed above applies to nested tables. **Nested tables are tables located within another table**. An example is shown below.

```
<html>
  <head>
    <title>Sample</title>
  </head>
  <body>
    <!--outer table-->
    <table border="1">
      <tbody>
        <tr>
          <td>ONE</td>
          <td>TWO</td>
        </tr>
        <tr>
          <td>THREE</td>
          <td>
            <!--inner table-->
            <table border="1">
              <tbody>
                <tr>
                  <td>1-2-3</td>
                  <td>4-5-6</td>
                </tr>
                <tr>
                  <td>7-8-9</td>
                  <td>10-11-12</td>
                </tr>
              </tbody>
            </table>
          </td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```



To access the cell with the text "4-5-6" using the "//parent/child" and predicate concepts from the previous section, we should be able to come up with the XPath code below.

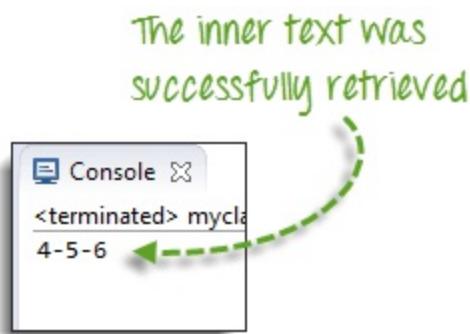


The WebDriver code below should be able to retrieve the inner text of the cell which we are accessing.

```
public static void main(String[] args) {
    String baseUrl = "file:///C:/newhtml.html";
    WebDriver driver = new FirefoxDriver();

    driver.get(baseUrl);
    String innerText = driver.findElement(By
        .xpath("//table/tbody/tr[2]/td[2]/table/tbody/tr/td[2]"))
        .getText();
    System.out.println(innerText);
    driver.quit();
}
```

The output below confirms that the inner table was successfully accessed.



Using Attributes as Predicates

If the element is written deep within the HTML code such that the number to use for the predicate is very difficult to determine, we can

use that element's unique attribute instead.

In the example below, the "New York to Chicago" cell is located deep into Mercury Tours homepage's HTML code.

Specials	
Atlanta to Las Vegas	\$398
Boston to San Francisco	\$513
Los Angeles to Chicago	\$168
New York to Chicago	\$198
Phoenix to San Francisco	\$213

```

<body>
  <div>
    <table height="100%" cellspacing="0" cellpadding="0" border="0">
      <tbody>
        <tr>
          <td valign="top" bgcolor="#003366">
            <td valign="top">
              <table cellspacing="0" cellpadding="0" border="0">
                <tbody>
                  <tr>
                  <tr>
                  <tr>
                  <tr>
                  <td>
                    <table cellspacing="0" cellpadding="0" border="0">
                      <tbody>
                        <tr>
                          <td width="14"> </td>
                        <td>
                          <table width="492" cellspacing="0" cellpadding="0" border="0">
                            <tbody>
                              <tr> </tr>
                            <tr>
                              <td width="273" valign="top">
                                <p>
                                  <table width="100%" cellspacing="0" cellpadding="0" border="0">
                                    <tbody>
                                      <tr>
                                      <tr>
                                      <tr valign="top">
                                        <td height="101">
                                          <table width="270" cellspacing="0" cellpadding="0" border="0" style="width: 270px; height: 101px; border-collapse: collapse; border: none; margin: 0; padding: 0; position: relative; background-color: #cccccc;">
                                            <tbody>
                                              <tr bgcolor="#cccccc">
                                                <td width="80%">
                                                  <font size="2">Vegas</font>
                                                </td>
                                              <td width="20%">
                                                </td>
                                              </tr>
                                            </tbody>
                                          </table>
                                        </td>
                                      </tr>
                                    </tbody>
                                  </table>
                                </p>
                              </td>
                            </tr>
                          </tbody>
                        </table>
                      </td>
                    </td>
                  </tr>
                </tbody>
              </table>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
  
```

this is the `<table>` that holds the New York to Chicago cell. Notice that it is buried deep into the HTML code, and the number to use as Predicate is difficult to determine.



In this case, we can use the table's unique attribute (`width="270"`) as

the predicate. **Attributes are used as predicates by prefixing them with the @ symbol.** In the example above, the "New York to Chicago" cell is located in the first <td> of the fourth <tr>, and so our XPath should be as shown below.

```
//table[@width="270"]/tbody/tr[4]/td
```

Remember that when we put the XPath code in Java, we should use the escape character backward slash "\\" for the double quotation marks on both sides of "270" so that the string argument of By.xpath() will not be terminated prematurely.

```
By.xpath("//table[@width=\"270\"]/tbody/tr[4]/td"))
```

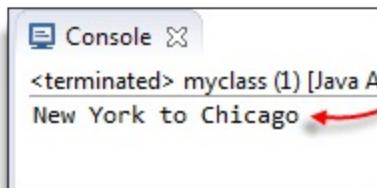
use the escape characters here

We are now ready to access that cell using the code below.

```
public static void main(String[] args) {
    String baseUrl = "http://newtours.demoaut.com/";
    WebDriver driver = new FirefoxDriver();

    driver.get(baseUrl);
    String innerText = driver.findElement(By
        .xpath("//table[@width=\"270\"]/tbody/tr[4]/td"))
        .getText();
    System.out.println(innerText);
    driver.quit();
}
```

the inner text was
successfully retrieved.



```
Console X
<terminated> myclass (1) [Java A]
New York to Chicago
```

A screenshot of a Java console window titled "Console X". It shows the output of a terminated process named "myclass (1) [Java A]". The output text "New York to Chicago" is displayed in black font. A red arrow points from the handwritten note above to the word "Chicago" in the console output.

Shortcut: Use Firebug/Chrome Inspect Element for Accessing Tables in Selenium

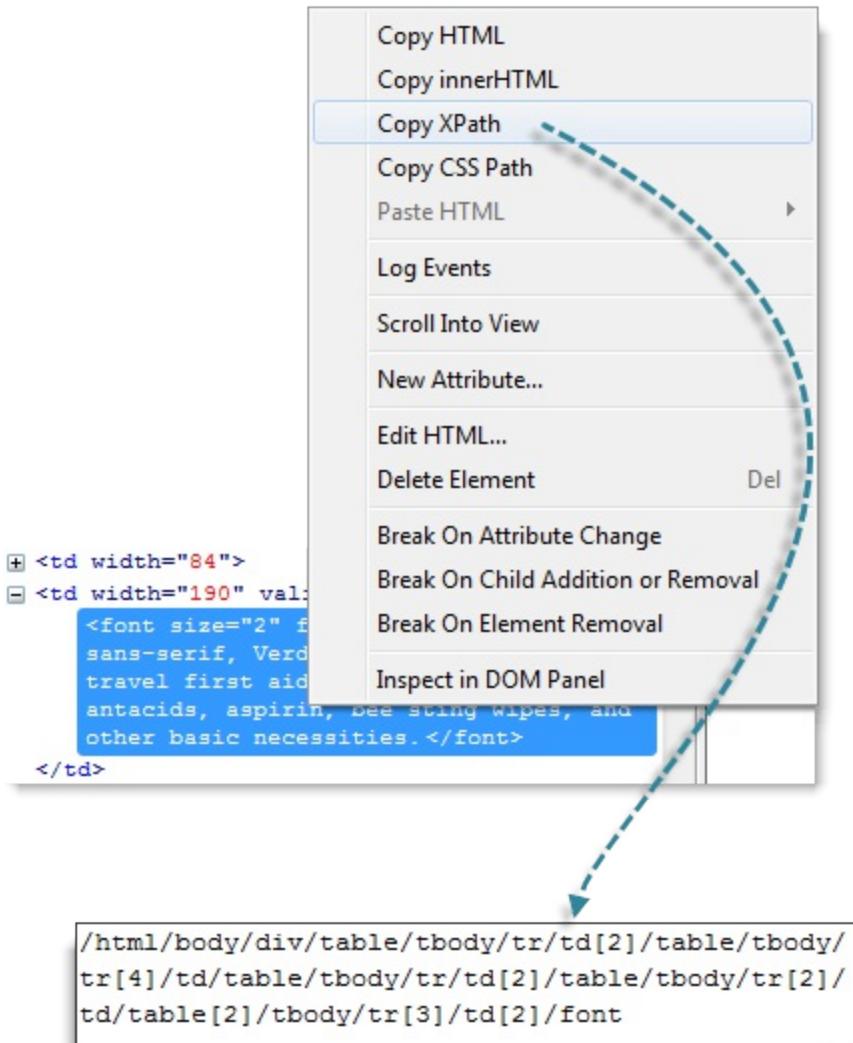
If the number or attribute of an element is extremely difficult or impossible to obtain, the quickest way to generate the XPath code is thru Firebug.

Consider the example below from Mercury Tours homepage.



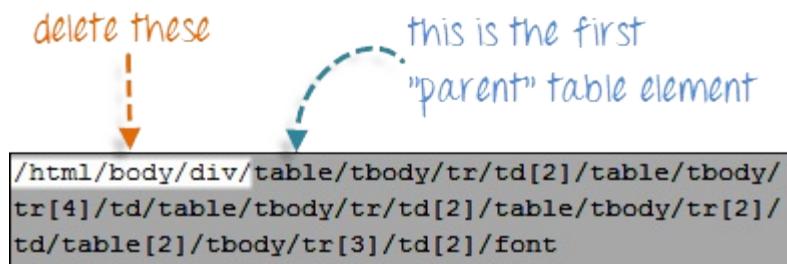
Step 1

Use Firebug to obtain the XPath code.



Step 2

Look for the first "table" parent element and delete everything to the left of it.



Step 3

Prefix the remaining portion of the code with double forward slash "://" and copy it over to your WebDriver code.

The remaining portion of the code, trimmed
and prefixed with "://"

```
//table/tbody/tr/td[2]/table/tbody/tr[4]/td/table/
tbody/tr/td[2]/table/tbody/tr[2]/td/table[2]/tbody
/tr[3]/td[2]/font
```

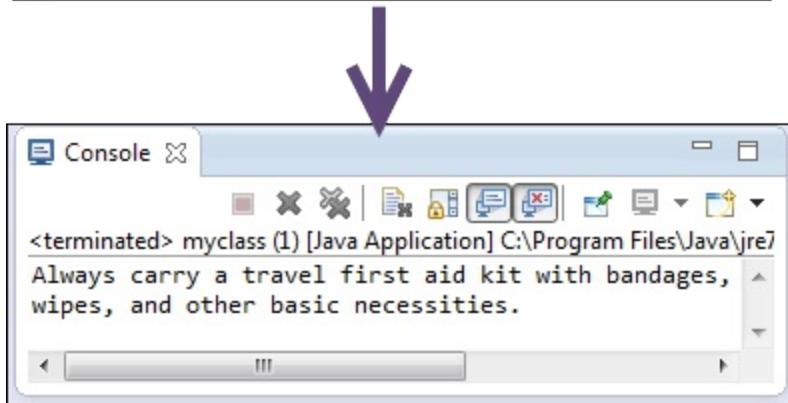


```
By.xpath("//table/tbody/tr/td[2]"
+ "/table/tbody/tr[4]/td"
+ "/table/tbody/tr/td[2]"
+ "/table/tbody/tr[2]/td"
+ "/table[2]/tbody/tr[3]/td[2]/font"))
```

When pasted onto the By.xpath() method

The WebDriver code below will be able to successfully retrieve the inner text of the element we are accessing.

```
public static void main(String[] args) {  
    String baseUrl = "http://newtours.demoaut.com/";  
    WebDriver driver = new FirefoxDriver();  
  
    driver.get(baseUrl);  
    String innerText = driver.findElement(  
        By.xpath("//table/tbody/tr/td[2]"  
            + "/table/tbody/tr[4]/td"  
            + "/table/tbody/tr/td[2]"  
            + "/table/tbody/tr[2]/td"  
            + "/table[2]/tbody/tr[3]/td[2]/font"))  
        .getText();  
    System.out.println(innerText);  
    driver.quit();  
}
```



Summary

- Links are accessed using the click() method.
- Apart from the locators available for any WebElement, Links also have link text based locators:
 - By.linkText() – locates the links based on the exact match of the link's text provided as a parameter.
 - By.partialLinkText() – locates links based on the partial text match of the link's text.
- Both the above locators are case Sensitive.
- If there are multiple matches, By.linkText() and By.partialLinkText() will only select the first match. In such cases where multiple links with the same link text are present, other locators based on xpath, CSS are used.

- `findElements()` & `By.tagName("a")` method finds all the elements in the page matching the locator criteria
- Links can be accessed by the `By.linkText()` and `By.partialLinkText()` whether they are inside or outside block-level elements.
- Accessing image links are done using `By.cssSelector()` and `By.xpath()` methods.
- `By.xpath()` is commonly used to access table elements.

Chapter 14: Keyboard & Mouse Event using Action Class in Selenium Webdriver

In this tutorial, we will learn handling Keyboard and Mouse Event in Selenium Webdriver

Handling Keyboard & Mouse Events

Handling special keyboard and mouse events are done using the **Advanced User Interactions API**. It contains the **Actions** and the **Action** classes that are needed when executing these events. The following are the most commonly used keyboard and mouse events provided by the Actions class.

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(sou rce, 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(s ource, x-offset, y-o ffset)	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 down.
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 sent

In the following example, we shall use the moveToElement() method to mouse-over on one Mercury Tours' table rows. See the example below.



The cell shown above is a portion of a <TR> element. If not hovered, its color is #FFC455 (orange). After hovering, the cell's color becomes transparent. It becomes the same color as the blue background of the whole orange table.

Step 1: Import the **Actions** and **Action** classes.

```
import org.openqa.selenium.interactions.Action;
import org.openqa.selenium.interactions.Actions;
```

Step 2: Instantiate a new Actions object.

```
Actions builder = new Actions(driver);
```

Step 3: Instantiate an Action using the Actions object in step 2.

```
Action mouseOverHome = builder
    .moveToElement(link_Home)
    .build();
```

In this case, we are going to use the `moveToElement()` method because we are simply going to mouse-over the "Home" link. The `build()` method is always the final method used so that all the listed actions will be compiled into a single step.

Step 4: Use the `perform()` method when executing the Action object we designed in Step 3.

```
mouseOverHome.perform();
```

Below is the whole WebDriver code to check the background color of the `<TR>` element before and after the mouse-over.

```
package newproject;

import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Action;
import org.openqa.selenium.interactions.Actions;

public class PG7 {

    public static void main(String[] args) {
        String baseUrl =
"http://demo.guru99.com/test/newtours/";

System.setProperty("webdriver.firefox.marionette", "C:\\geckodriver.exe");
        WebDriver driver = new FirefoxDriver();

        driver.get(baseUrl);
        WebElement link_Home =
driver.findElement(By.linkText("Home"));
        WebElement td_Home = driver
            .findElement(By
```

```

        .xpath("//html/body/div"
+ "/table/tbody/tr/td"
+ "/table/tbody/tr/td"
+ "/table/tbody/tr/td"
+ "/table/tbody/tr"));

Actions builder = new Actions(driver);
Action mouseOverHome = builder
    .moveToElement(link_Home)
    .build();

String bgColor =
td_Home.getCssValue("background-color");
System.out.println("Before hover: " + bgColor);
mouseOverHome.perform();
bgColor = td_Home.getCssValue("background-
color");
System.out.println("After hover: " + bgColor);
driver.close();
}
}

```

The output below clearly states that the background color became transparent after the mouse-over.

```

Console [x]
<terminated> AlertDemo [Java Application] C:\P
Before hover: rgba(255, 165, 0, 1)
After hover: rgba(0, 0, 0, 0)

```

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.

```

public static void main(String[] args) {
    String baseUrl = "http://www.facebook.com/";
    WebDriver driver = new FirefoxDriver();

    driver.get(baseUrl);
    WebElement txtUsername = driver.findElement(By.id("email"));

    Actions builder = new Actions(driver);
    Action seriesOfActions = builder
        .moveToElement(txtUsername)
        .click()
        .keyDown(txtUsername, Keys.SHIFT)
        .sendKeys(txtUsername, "hello")
        .keyUp(txtUsername, Keys.SHIFT)
        .doubleClick(txtUsername)
        .contextClick()
        .build();

    seriesOfActions.perform();
}

```

this will type "hello" in uppercase

this will highlight the text "HELLO"

this will bring up the context menu



Summary

- Handling special keyboard and mouse events are done using the AdvancedUserInteractions API.
- Frequently used Keyword and Mouse Events are doubleClick(), keyUp, dragAndDropBy, contextClick & sendKeys.

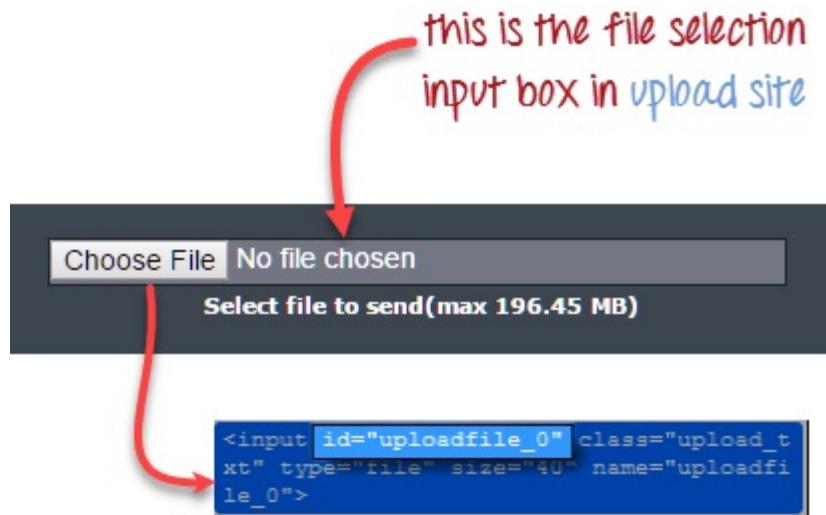
Chapter 15: How to Upload & Download a File using Selenium Webdriver

In this tutorial, we will learn How to deal with file uploads and downloads.

Uploading Files

For this section, we will use <http://demo.guru99.com/test/upload/> as our test application. This site easily allows any visitor to upload files without requiring them to sign up.

Uploading files in WebDriver is done by simply using the sendKeys() method on the file-select input field to enter the path to the file to be uploaded.



Let's say we wish to upload the file "C:\newhtml.html". Our WebDriver code should be like the one shown below.

```
package newproject;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
public class PG9 {
    public static void main(String[] args) {

System.setProperty("webdriver.firefox.marionette", "C:\\geckodriver.exe");
    String baseUrl = "http://demo.guru99.com/test/upload/";
    WebDriver driver = new FirefoxDriver();

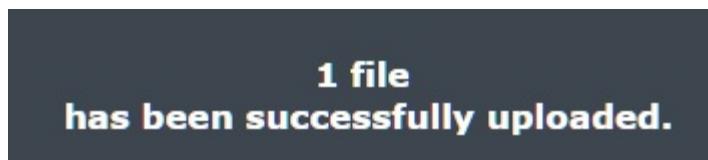
    driver.get(baseUrl);
    WebElement uploadElement =
driver.findElement(By.id("uploadfile_0"));

    // enter the file path onto the file-selection input
field
    uploadElement.sendKeys("C:\\\\newhtml.html");

    // check the "I accept the terms of service" check box
driver.findElement(By.id("terms")).click();

    // click the "UploadFile" button
driver.findElement(By.name("send")).click();
    }
}
```

After running this script, you should be able to upload the file successfully and you should get a message similar to this.



Remember following two things when uploading files in WebDriver

1. There is no need to simulate the clicking of the "Browse" button. WebDriver automatically enters the file path onto the file-selection text box of the <input type="file"> element
2. When setting the file path in your Java IDE, use the proper escape character for the back-slash.

```
// enter the file path onto the file-selection input field  
uploadElement.sendKeys("C:\\\\newhtml.html");
```

When used within a string, each back-slash in your file path is represented by a double back-slash

Downloading Files

WebDriver has no capability to access the Download dialog boxes presented by browsers when you click on a download link or button. However, we can bypass these dialog boxes using a separate program called "wget".

What is Wget?

Wget is a small and easy-to-use command-line program used to automate downloads. Basically, we will access Wget from our WebDriver script to perform the download process.

Setting up Wget

Step 1: In your C Drive, create a new folder and name it as "Wget".

Download wget.exe from here and Place it in the Wget folder you created from the step above.

Windows binaries of [GNU Wget](#)

A command-line utility for retrieving files using HTTP, HTTPS and FTP protocols.

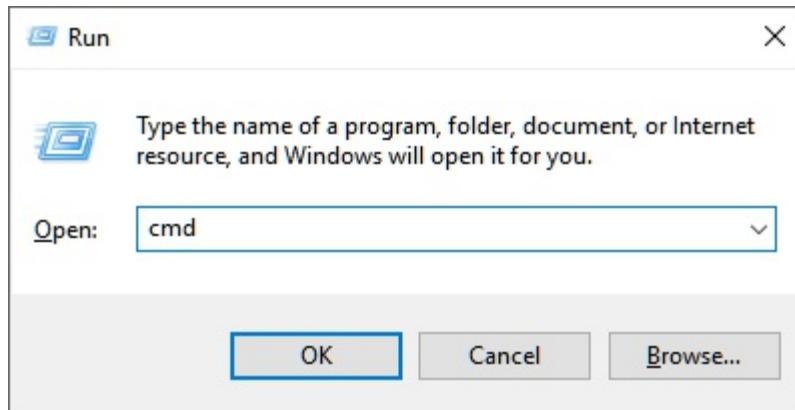
Warning: some antivirus tools recognise wget-1.18-win32.zip as [potentially dangerous](#). The file that triggers the warning is wget.exe.debug, which contains debugging symbols for wget.exe, and isn't even executable. If your AV is giving you trouble, and you don't need the documentation or debug symbols, you can download wget.exe directly, or switch to a less broken security product.

All of the binaries are compiled statically, meaning that wget.exe doesn't require any other files to work.

Version	32-bit binary	64-bit binary	Notes
1.18	wget-1.18-win32.zip		OpenSSL 1.0.2h, ZLib 1.2.8, taskbar progressbar , Windows certificate store support , manual
1.17.1	wget-1.17.1-win32.zip		OpenSSL 1.0.2e, ZLib 1.2.8, taskbar progressbar , Windows certificate store support , man page
1.17	wget-1.17-win32.zip	wget-1.17-win64.zip	OpenSSL 1.0.2d, ZLib 1.2.8, taskbar progressbar , man page
1.16.3	wget-1.16.3-win32.zip	wget-1.16.3-win64.zip	OpenSSL 1.0.2a, ZLib 1.2.8, taskbar progressbar , man page
1.16.2	wget-1.16.2-win32.zip	wget-1.16.2-win64.zip	OpenSSL 1.0.2, ZLib 1.2.8, taskbar progressbar , man page
1.16.1	wget-1.16.1-win32.zip	wget-1.16.1-win64.zip	OpenSSL 1.0.1j, ZLib 1.2.8, taskbar progressbar , man page
1.16	wget-1.16-win32.zip	wget-1.16-win64.zip	OpenSSL 1.0.1f, ZLib 1.2.8
1.15	wget-1.15-win32.zip	-	OpenSSL 1.0.0f, ZLib 1.2.5
1.13.4	wget-1.13.4-win32-static.zip	-	OpenSSL 1.0.0d, ZLib 1.2.5
1.13	wget-1.13-win32-static.zip	-	

[Source code](#)

Step 2: Open Run by pressing windows key + "R" ; type in "cmd & click ok



Type in the command "cd /" to move to the root directory

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\[REDACTED] >cd /
C:\>
```

Step 3: Type in the command to check whether the given setup is working

```
cmd /c C:\\Wget\\wget.exe -P C: --no-check-certificate
http://demo.guru99.com/selenium/msgr11us.exe
```

```
C:\>cmd /c C:\\Wget\\wget.exe -P C: --no-check-certificate http://demo.guru99.com/selenium/msgr11us.exe
--2017-02-21 18:16:36-- http://demo.guru99.com/selenium/msgr11us.exe
Resolving demo.guru99.com (demo.guru99.com)... 72.52.250.40
Connecting to demo.guru99.com (demo.guru99.com)|72.52.250.40|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 439704 (429K) [application/x-msdownload]
C:/msgr11us.exe: Permission denied

cannot write to 'C:/msgr11us.exe' (Bad file descriptor).
C:\>
```

There seems to be an issue writing into C drive.

Step 4: You need to debug the wget errors in command line before you execute the code using Selenium Webdriver. These errors will persist in Eclipse and the error messages will not be as informative. Best to first get wget working using command line. If it works in command line it will definitely work in Eclipse.

In our example, as show in step 3, there is a problem writing into C drive. Let's change the download location to D drive and check results.

```
cmd /c C:\\Wget\\wget.exe -P D: --no-check-certificate
http://demo.guru99.com/selenium/msgr11us.exe
```

```
C:\>cmd /c C:\\Wget\\wget.exe -P D: --no-check-certificate http://demo.guru99.com/selenium/msgr11us.exe
--2017-02-21 18:09:56-- http://demo.guru99.com/selenium/msgr11us.exe
Resolving demo.guru99.com (demo.guru99.com)... 72.52.250.40
Connecting to demo.guru99.com (demo.guru99.com)|72.52.250.40|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 439704 (429K) [application/x-msdownload]
Saving to: 'D:/msgr11us.exe'

msgr11us.exe    [Download Progress] 100%[=====] 429.40K   229KB/s   in 1.9s
2017-02-21 18:09:58 (229 KB/s) - [D:/msgr11us.exe] saved [439704/439704]
```

Annotations in the screenshot:

- Download Destination: A red box highlights the path "D:/msgr11us.exe".
- Download Source: An orange box highlights the URL "http://demo.guru99.com/selenium/msgr11us.exe".
- Download Progress: A yellow box highlights the progress bar at the bottom of the terminal output.
- Wget run Successful: A black box highlights the message "2017-02-21 18:09:58 (229 KB/s) - [D:/msgr11us.exe] saved [439704/439704]".

Messenger was downloaded successfully.

Before you proceed further don't forget to delete the downloaded file

Using WebDriver and Wget

In the following example, we will use WebDriver and wget to download a popular chat software called Yahoo Messenger. Our base URL shall be <http://demo.guru99.com/test/yahoo.html>.



Step 1

Import the "java.io.IOException" package because we will have to catch an IOException later in Step 4.

```
import java.io.IOException;
```

Step 2

Use getAttribute() to obtain the "href" value of the download link and save it as a String variable. In this case, we named the variable as "sourceLocation".

```
WebElement downloadButton = driver.findElement(By  
    .id("messenger-download"));  
String sourceLocation = downloadButton.getAttribute("href");
```

Step 3

Set-up the syntax for wget using the following command.

```
String wget_command = "cmd /c wget -P c: " + sourceLocation;
```

whatever you put after -P will be your download destination. In this case, the downloaded file will be placed on the C drive

Step 4

Initiate the download process by calling wget from our WebDriver code.

this line will call wget and will supply the command we specified in step 3

```
try {
    Process exec = Runtime.getRuntime().exec(wget_command);
    int exitVal = exec.waitFor();
    System.out.println("Exit value: " + exitVal);
} catch (InterruptedException | IOException ex) {
    System.out.println(ex.toString());
}
```

the waitFor() method is used to wait until wget completes the download

To sum it all up, your WebDriver code could look like the one shown below.

```
package newproject;
import java.io.IOException;

import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
public class PG8 {
    public static void main(String[] args) {

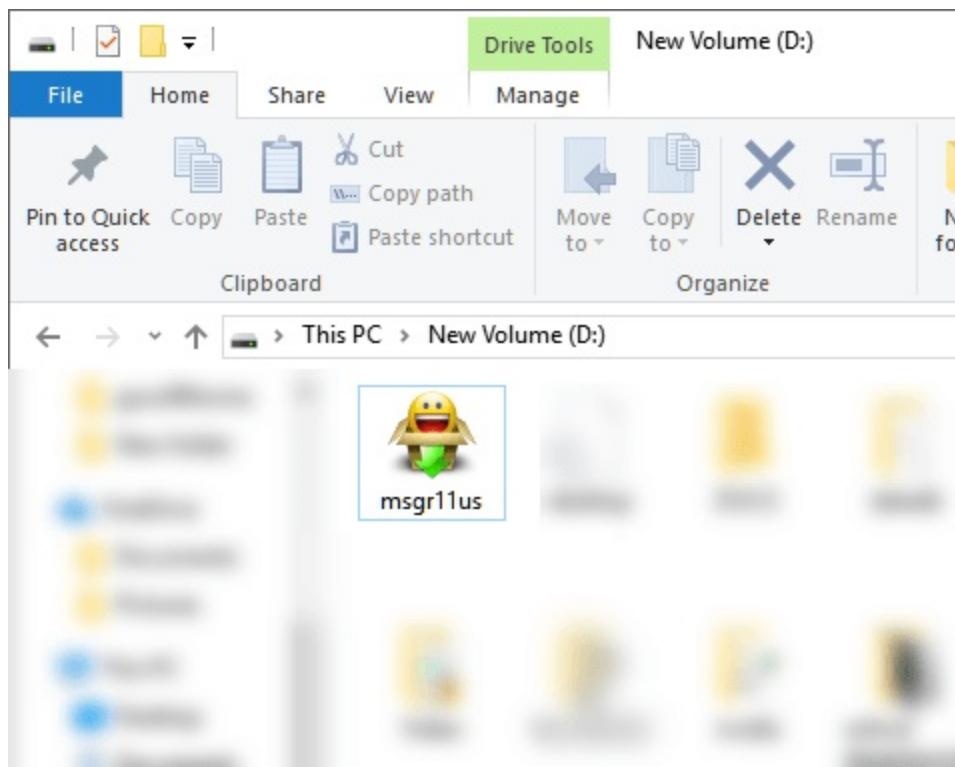
System.setProperty("webdriver.firefox.marionette", "C:\\geckodriver.exe");
        String baseUrl =
"http://demo.guru99.com/test/yahoo.html";
        WebDriver driver = new FirefoxDriver();

        driver.get(baseUrl);
        WebElement downloadButton = driver.findElement(By
.id("messenger-download")));
        String sourceLocation =
downloadButton.getAttribute("href");
        String wget_command = "cmd /c C:\\\\Wget\\\\wget.exe -P D: -
-no-check-certificate " + sourceLocation;

        try {
Process exec = Runtime.getRuntime().exec(wget_command);
        int exitVal = exec.waitFor();
        System.out.println("Exit value: " + exitVal);
        } catch (InterruptedException | IOException ex) {
        System.out.println(ex.toString());
        }
        driver.close();
    }

}
```

After executing this code, check your D drive and verify that the Yahoo Messenger installer was successfully downloaded there.



Summary

- Uploading files in WebDriver is done by simply using the `sendKeys()` method on the file-select input field to enter the path to the file to be uploaded.
- WebDriver cannot automate downloading of files on its own.
- The easiest way to download files using WebDriver is to use Wget.

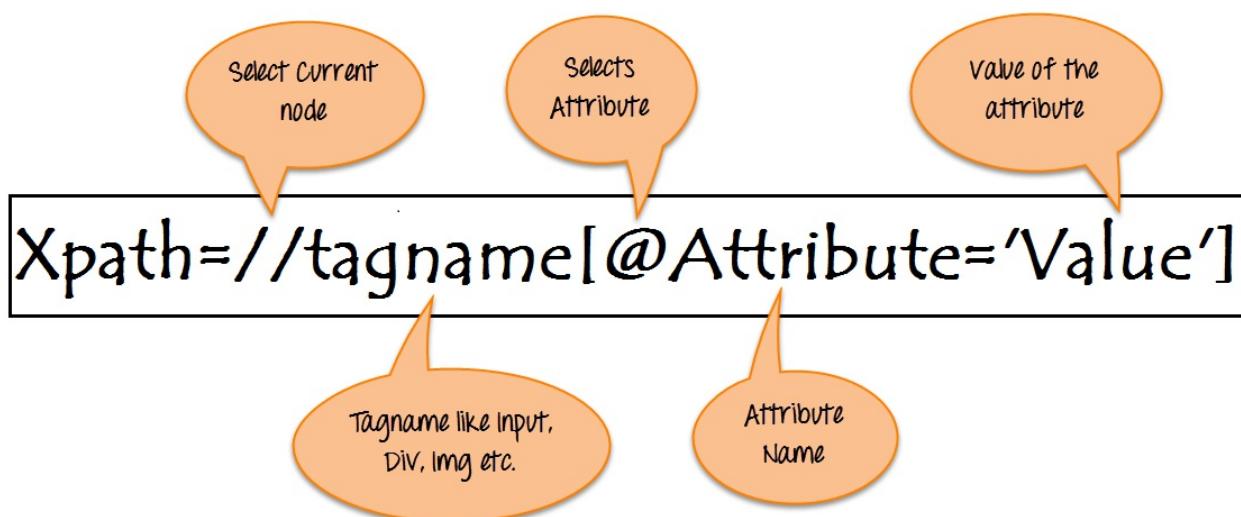
Chapter 16: XPath in Selenium: Complete Guide

In Selenium automation, if the elements are not found by the general locators like id, class, name, etc. then XPath is used to find an element on the web page .

In this tutorial, we will learn about the xpath and different XPath expression to find the complex or dynamic elements, whose attributes changes dynamically on refresh or any operations.

What is XPath

XPath is defined as **XML path. It is a syntax or language for finding any element on the web page using XML path expression.** XPath is used to find the location of any element on a webpage using HTML DOM structure. The basic format of XPath is explained below with screen shot.



Syntax for XPath:

XPath contains the path of the element situated at the web page.
Standard syntax for creating XPath is.

```
Xpath=//tagname[@attribute='value']
```

- **//** : Select current node.
- **Tagname**: Tagname of the particular node.
- **@**: Select attribute.
- **Attribute**: Attribute name of the node.
- **Value**: Value of the attribute.

To find the element on web pages accurately there are different types of locators:

XPath Locators	Find different elements on web page
ID	To find the element by ID of the element
Classname	To find the element by Classname of the element
Name	To find the element by name of the element
Link text	To find the element by text of the link
XPath	XPath required for finding the dynamic element and traverse between various elements of the web page
CSS path	CSS path also locates elements having no name, class or ID.

Types of X-path

There are two types of XPath:

- 1) Absolute XPath .**
- 2) Relative XPath .**

Absolute XPath :

It is the direct way to find the element, but the disadvantage of the absolute XPath is that if there are any changes made in the path of the element then that XPath gets failed.

The key characteristic of XPath is that it begins with the single forward slash(/), which means you can select the element from the root node.

Below is the example of an absolute xpath expression of the element shown in the below screen.

Absolute xpath:

```
html/body/div[1]/section/div[1]/div/div/div/div[1]/div/div/div/div/div[3]/div[1]/div/h4[1]/b
```

The screenshot shows the FirePath extension for Firefox. At the top, there's a navigation bar with tabs like Element, TESTING, SAP, and others. Below the tabs is a toolbar with icons for Top Window, Highlight, and XPath. The XPath field contains the absolute XPath expression: `html/body/div[1]/section/div[1]/div/div/div/div[1]/div/div/div/div/div[3]/div[1]/div/h4[1]/b`. This expression is highlighted with a red box. Below the toolbar, the DOM tree is displayed. A blue box highlights the node `Testing`, which is the target of the selected XPath. A speech bubble labeled "Absolute Path" points to the highlighted node in the tree. Another orange speech bubble labeled "Element" points to the "Element" tab in the toolbar. At the bottom left, a red box highlights the message "1 matching node".

Relative xpath:

For Relative Xpath the path starts from the middle of the HTML DOM structure. It starts with the double forward slash (//), which means it

can search the element anywhere at the webpage.

You can start from the middle of the HTML DOM structure and no need to write long xpath.

Below is the example of a relative XPath expression of the same element shown in the below screen. This is the common format used to find element through a relative XPath.

Relative xpath: `//*[@class='featured-box']//*[text()='Testing']`

The screenshot shows a web page with a navigation bar and two main sections: 'TESTING' and 'SAP'. The 'TESTING' section contains links for QTP, Selenium, and Mobile App Testing. The 'SAP' section contains links for SAP Beginner, ABAP, HR/HCM, and FICO. Below the page is the FirePath extension interface. The 'XPath' field contains the relative XPath expression `//*[text()='Testing']`. The FirePath tree view shows the DOM structure with the matching node highlighted. A speech bubble labeled 'Element' points to the 'TESTING' header, and another speech bubble labeled 'Relative Path' points to the FirePath toolbar.

What are XPath axes.

XPath axes search different nodes in XML document from current context node. XPath Axes are the methods used to find dynamic elements, which otherwise not possible by normal XPath method having no ID , Classname, Name, etc.

Axes methods are used to find those elements, which dynamically change on refresh or any other operations. There are few axes methods

commonly used in Selenium Webdriver like child, parent, ancestor, sibling, preceding, self, etc.

Using XPath Handling complex & Dynamic elements in Selenium

1) Basic XPath:

XPath expression select nodes or list of nodes on the basis of attributes like **ID** , **Name**, **Classname**, etc. from the XML document as illustrated below.

```
Xpath=//input[@name='uid']
```

Here is a link to access the page <http://demo.guru99.com/v1/>

The screenshot shows a browser window with a login form. The FirePath extension is active, highlighting the user ID input field with the XPath expression `./input[@name='uid']`. An orange callout bubble labeled "Basic xpath" points to the highlighted path in the FirePath interface. Another callout bubble labeled "Element" points to the user ID input field on the page. The browser's address bar shows the URL <http://demo.guru99.com/v1/>.

Some more basic xpath expressions:

```
Xpath=//input[@type='text']
Xpath= //label[@id='message23']
```

```
Xpath= //input[@value='RESET']  
Xpath=//*[@class='barone']  
Xpath=/a[@href='http://demo.guru99.com/']  
Xpath= //img[@src='//cdn.guru99.com.../Images/home/java.png']
```

2) Contains() : Contains() is a method used in XPath expression. It is used when the value of any attribute changes dynamically, for example, login information.

The contain feature has an ability to find the element with partial text as shown in below example.

In this example, we tried to identify the element by just using partial text value of the attribute. In the below XPath expression partial value 'sub' is used in place of submit button. It can be observed that the element is found successfully.

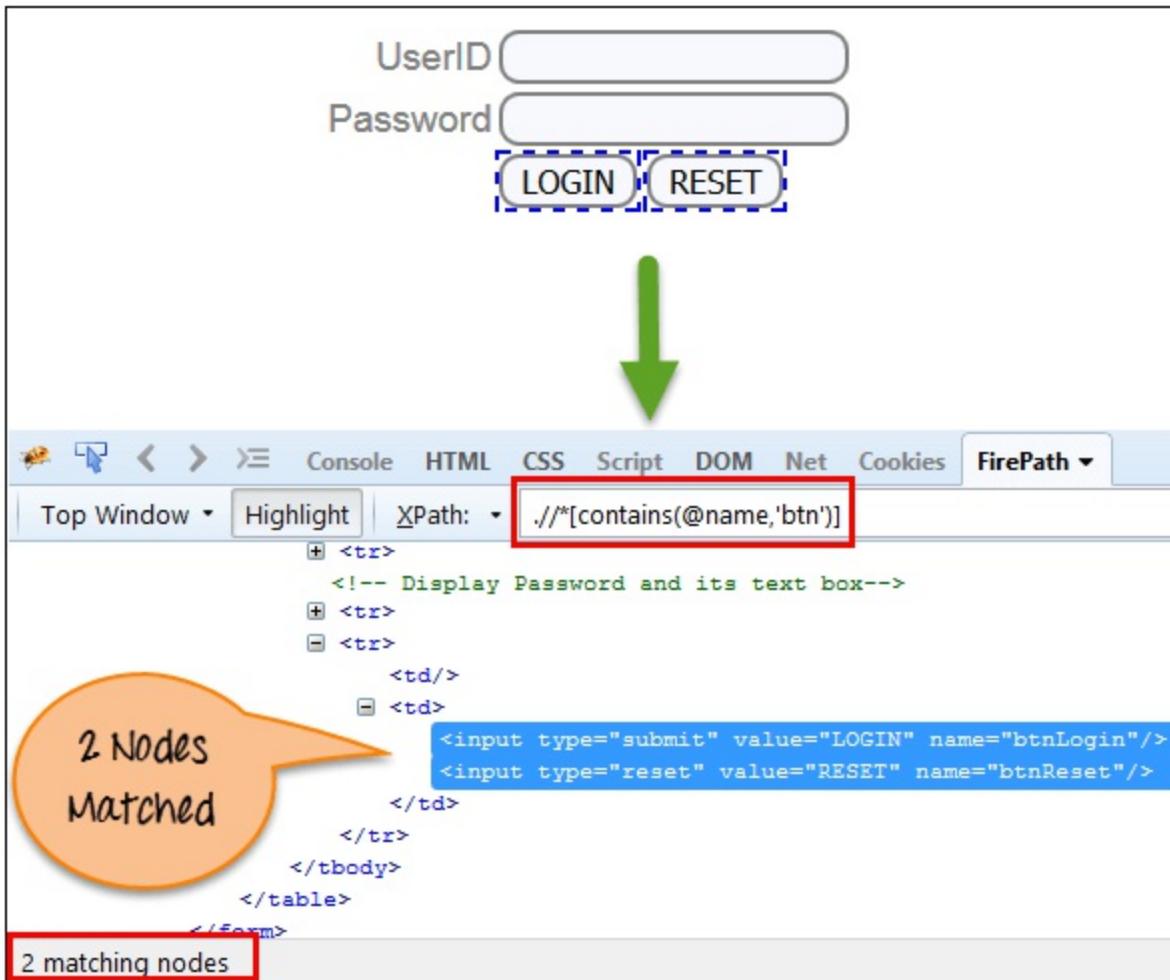
Complete value of 'Type' is 'submit' but using only partial value 'sub'.

```
Xpath=//*[@contains(@type, 'sub')]
```

Complete value of 'name' is 'btnLogin' but using only partial value 'btn'.

```
Xpath=//*[@contains(@name, 'btn')]
```

In the above expression, we have taken the 'name' as an attribute and 'btn' as an partial value as shown in the below screenshot. This will find 2 elements (LOGIN & RESET) as their 'name' attribute begins with 'btn'.



Similarly, in the below expression, we have taken the 'id' as an attribute and 'message' as a partial value. This will find 2 elements ('User-ID must not be blank' & 'Password must not be blank') as its 'name' attribute begins with 'message'.

```
Xpath=//*[@id='message']
```

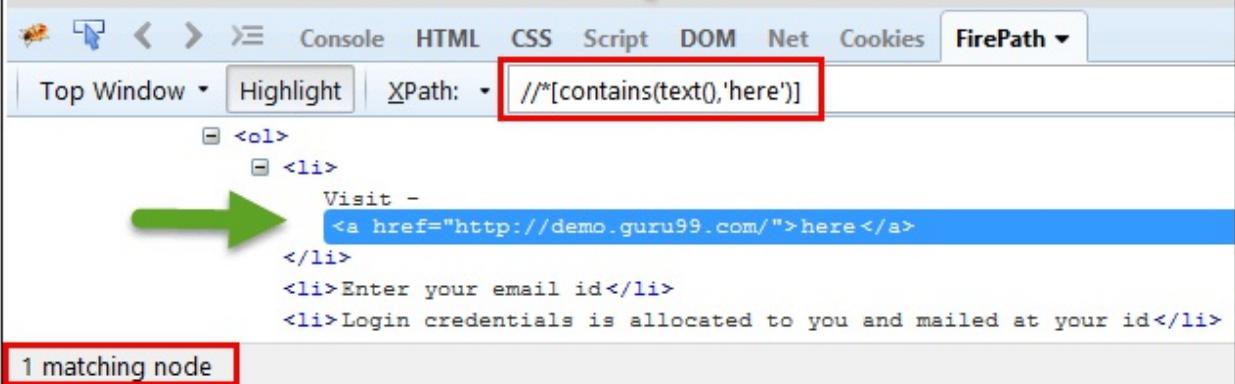
The screenshot shows a login form with two validation messages: "User-ID must not be blank" and "Password must not be blank". Below the form is a FirePath tool interface. The XPath expression `//*[contains(@id,'message')]` is highlighted. The tool shows two matching nodes: one for the User-ID validation message and one for the Password validation message. A green arrow points from the validation messages in the browser to the matching nodes in the FirePath tool.

In the below expression, we have taken the "text" of the link as an attribute and 'here' as a partial value as shown in the below screenshot. This will find the link ('here') as it displays the text 'here'.

```
Xpath=//*[contains(text(),'here')]
Xpath=//*[contains(@href,'guru99.com')]
```

Steps To Generate Access

1. Visit - [here](#)
 2. Enter your email id



Top Window ▾ Highlight XPath: **//*[contains(text(),'here')]**

```

<ol>
  <li>
    Visit -
    <a href="http://demo.guru99.com/">here</a>
  </li>
  <li>Enter your email id</li>
  <li>Login credentials is allocated to you and mailed at your id</li>

```

1 matching node

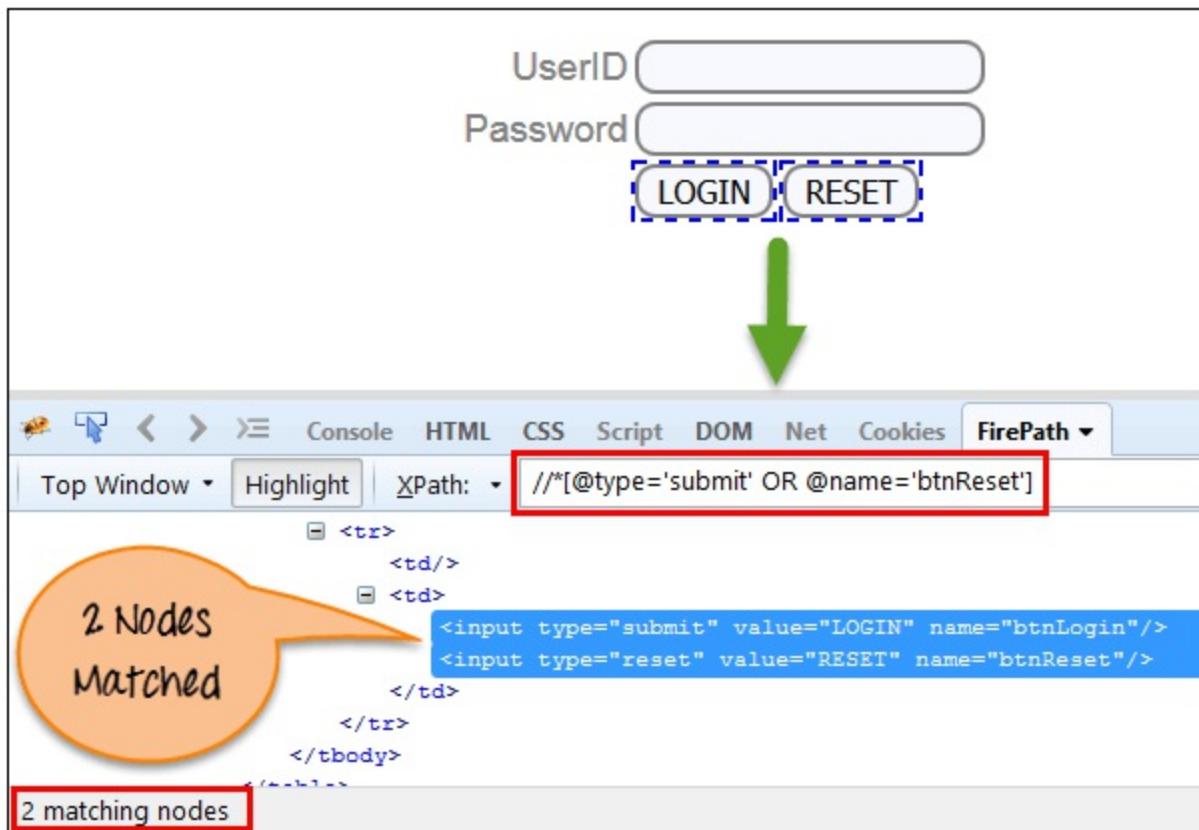
3) Using OR & AND:

In OR expression, two conditions are used, whether 1st condition OR 2nd condition should be true. It is also applicable if any one condition is true or maybe both. Means any one condition should be true to find the element.

In the below XPath expression, it identifies the elements whose single or both conditions are true.

```
Xpath=//*[@type='submit' OR @name='btnReset']
```

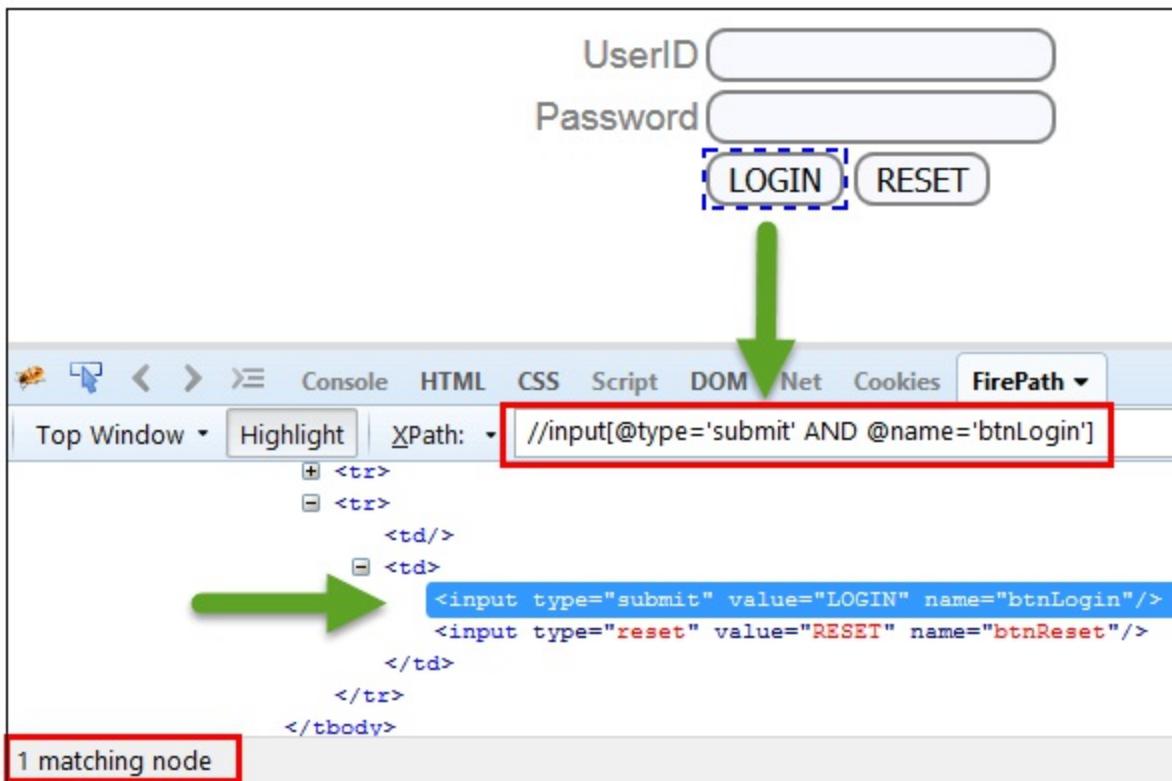
Highlighting both elements as "LOGIN" element having attribute 'type' and "RESET" element having attribute 'name'.



In AND expression, two conditions are used, both conditions should be true to find the element. It fails to find element if any one condition is false.

```
Xpath=//input[@type='submit' and @name='btnLogin']
```

In below expression, highlighting 'LOGIN' element as it having both attribute 'type' and 'name'.



4) Start-with function: Start-with function finds the element whose attribute value changes on refresh or any operation on the webpage. In this expression, match the starting text of the attribute is used to find the element whose attribute changes dynamically. You can also find the element whose attribute value is static (not changes).

For example :- Suppose the ID of particular element changes dynamically like:

`Id=" message12"`

`Id=" message345"`

`Id=" message8769"`

and so on.. but the initial text is same. In this case, we use Start-with expression.

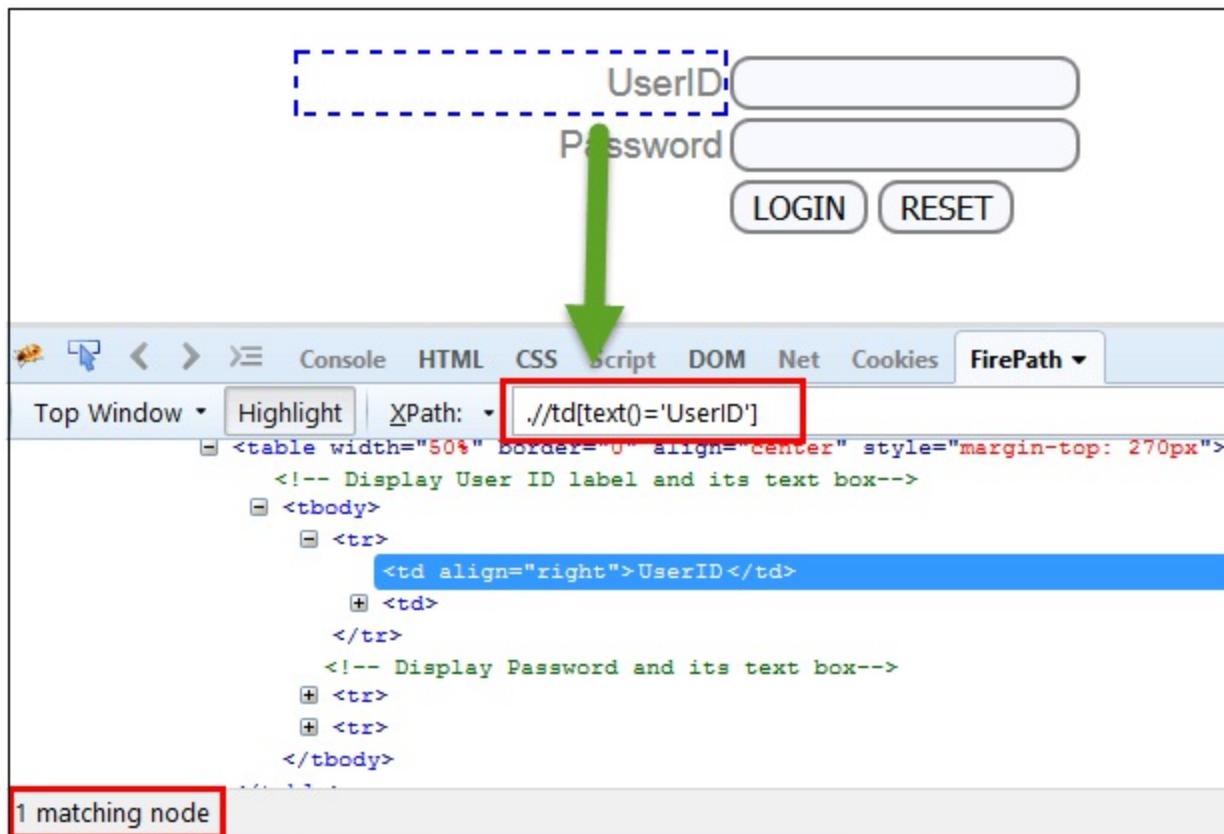
In the below expression, there are two elements with an id starting "message"(i.e., 'User-ID must not be blank' & 'Password must not be blank'). In below example, XPath finds those element whose 'ID' starting with 'message'.

```
Xpath=//label[starts-with(@id, 'message')]
```

The screenshot shows a browser window with a login form. The form has fields for 'UserID' and 'Password', each with an associated error message: 'User-ID must not be blank' and 'Password must not be blank'. Below the form are 'LOGIN' and 'RESET' buttons. At the top, a toolbar includes 'Console', 'HTML', 'CSS', 'Script', 'DOM', 'Net', 'Cookies', and 'FirePath'. The 'FirePath' tab is selected, showing the DOM tree. A red box highlights the XPath expression '/label[starts-with(@id, 'message')]' in the 'XPath' input field. A green arrow points from this field to the 'message' labels in the tree. An orange callout bubble with the text 'Id starting with 'message'' points to one of the 'message' labels. The tree shows two matching nodes, both labeled 'message'. The status bar at the bottom indicates '2 matching nodes'.

5) Text(): In this expression, with text function, we find the element with exact text match as shown below. In our case, we find the element with text "UserID".

```
Xpath=//td[text()='UserID']
```



6) XPath axes methods: These XPath axes methods are used to find the complex or dynamic elements. Below we will see some of these methods.

For illustrating these XPath axes method, we will use the Guru99 bank demo site.

a) Following: Selects all elements in the document of the current `node()` [UserID input box is the current node] as shown in the below screen.

```
Xpath=//*[@type='text']//following::input
```



There are 3 "input" nodes matching by using "following" axis-password, login and reset button. If you want to focus on any particular element then you can use the below XPath method:

```
Xpath=//*[ @type = 'text' ]//following::input[1]
```

You can change the XPath according to the requirement by putting [1], [2].....and so on.

With the input as '1', the below screen shot finds the particular node that is 'Password' input box element.

The screenshot shows a browser window for 'Guru99 Bank'. At the top, there's a header bar with the bank's name. Below it is a login form with fields for 'UserID' and 'Password', and buttons for 'LOGIN' and 'RESET'. An orange speech bubble points to the 'Password' input field with the text 'Showing particular Node'. Below the form, a FirePath toolbar is open. The 'XPath' dropdown shows the expression `//*[@type='text']//following::input[1]`. The FirePath interface highlights the entire row of the table containing the password input field, indicating that the XPath expression matches all elements in that row.

b) Ancestor: The ancestor axis selects all ancestors element (grandparent, parent, etc.) of the current node as shown in the below screen.

In the below expression, we are finding ancestors element of the current node("ENTERPRISE TESTING" node).

```
Xpath=//*[text()='Enterprise Testing']//ancestor::div
```

The screenshot shows a 'Tutorials Library' page with a table structure. The table has four main sections: 'TESTING', 'SAP', 'LIVE PROJECTS', and 'MUST LEARN!'. The 'TESTING' section contains 13 items. An orange speech bubble points to this section with the text '13 Nodes matched'. Below the table, a FirePath toolbar is visible. The 'XPath' dropdown shows the expression `//*[text()='Enterprise Testing']//ancestor::div`. The FirePath interface highlights the entire table, indicating that the XPath expression matches all its rows.

There are 13 "div" nodes matching by using "ancestor" axis. If you

want to focus on any particular element then you can use the below XPath, where you change the number 1, 2 as per your requirement:

```
Xpath=//*[text()='Enterprise Testing']//ancestor::div[1]
```

You can change the XPath according to the requirement by putting [1], [2].....and so on.

c) Child : Selects all children elements of the current node (Java) as shown in the below screen.

```
Xpath=//*[@id='java_technologies']/child::li
```

The screenshot shows the FirePath extension in the Chrome developer tools. The DOM tree is displayed with several sections highlighted by dashed blue boxes: 'TESTING', 'SAP', 'LIVE PROJECTS', and 'MUST LEARN!'. The 'TEST MANAGEMENT' section is also visible. The 'xpath using child' text is highlighted in an orange callout bubble pointing to the search bar. The search bar contains the XPath expression //*[@id='java_technologies']/child::li. Below the search bar, it says '71 matching nodes'. The bottom of the screenshot shows the browser's address bar with the URL https://www.guru99.com.

There are 71 "li" nodes matching by using "child" axis. If you want to focus on any particular element then you can use the below xpath:

```
Xpath=//*[@id='java_technologies']/child::li[1]
```

You can change the xpath according to the requirement by putting [1], [2].....and so on.

d) Preceding: Select all nodes that come before the current node as shown in the below screen.

In the below expression, it identifies all the input elements before "LOGIN" button that is **Userid** and **password** input element.

```
Xpath=//*[@type='submit']//preceding::input
```

The screenshot shows a browser window with the FirePath extension active. The FirePath toolbar at the top includes options like Highlight, XPath, CSS, Script, DOM, Net, Cookies, and FirePath. The XPath dropdown shows the expression //*[@type='submit']//preceding::input. Below the toolbar, the DOM tree is displayed. Two input elements under the 'tbody' node are highlighted with a red border. A callout bubble labeled 'Showing 2 Nodes' points to these inputs. Another callout bubble labeled 'xpath using preceding' points to the highlighted XPath expression in the toolbar. At the bottom left of the DOM tree, a red box highlights the text '2 matching nodes'.

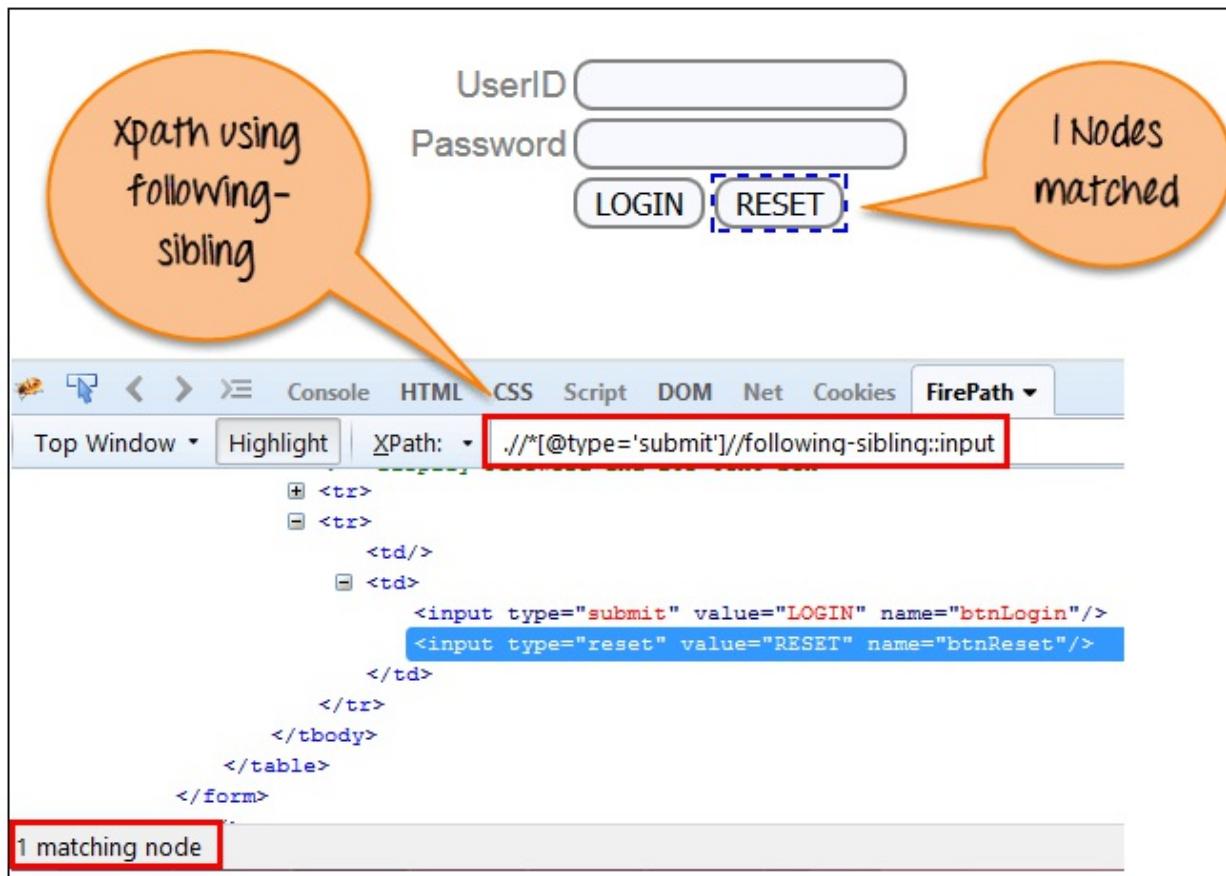
There are 2 "input" nodes matching by using "preceding" axis. If you want to focus on any particular element then you can use the below XPath:

```
Xpath=//*[@type='submit']//preceding::input[1]
```

You can change the xpath according to the requirement by putting [1], [2].....and so on.

e) Following-sibling: Select the following siblings of the context node. Siblings are at the same level of the current node as shown in the below screen. It will find the element after the current node.

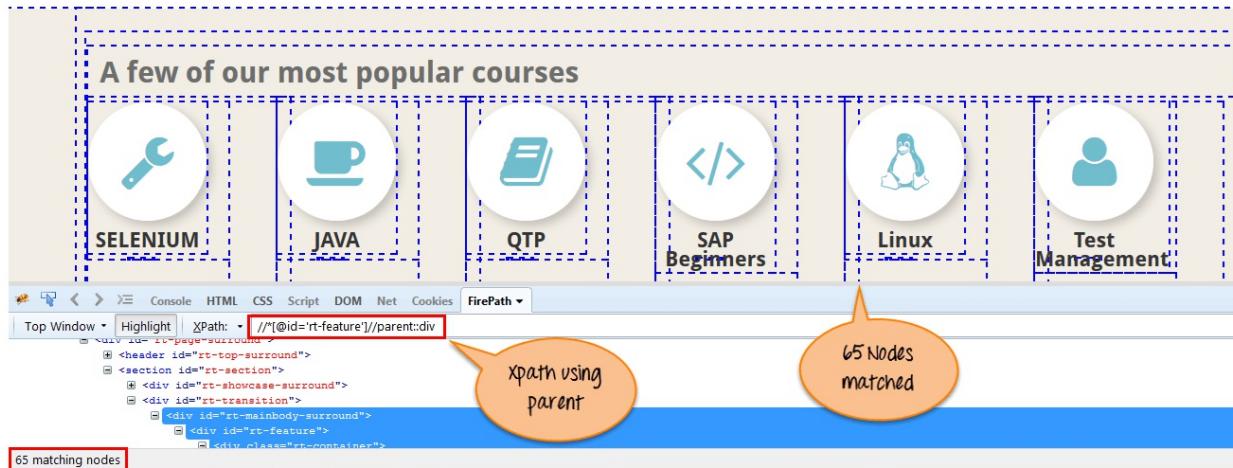
```
xpath=//*[@type='submit']//following-sibling::input
```



One input nodes matching by using "following-sibling" axis.

f) Parent: Selects the parent of the current node as shown in the below screen.

```
Xpath=//*[@id='rt-feature']//parent::div
```

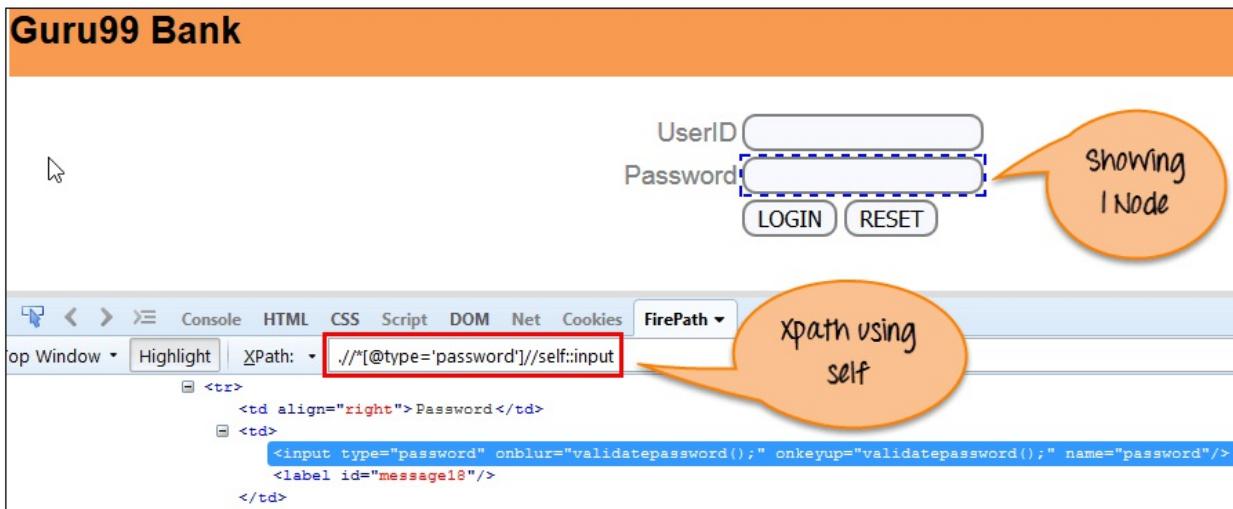


There are 65 "div" nodes matching by using "parent" axis. If you want to focus on any particular element then you can use the below XPath:

```
Xpath=//*[@id='rt-feature']//parent::div[1]
```

You can change the XPath according to the requirement by putting [1], [2].....and so on.

g) Self: Selects the current node or 'self' means it indicates the node itself as shown in the below screen.



One node matching by using "self" axis. It always finds only one node

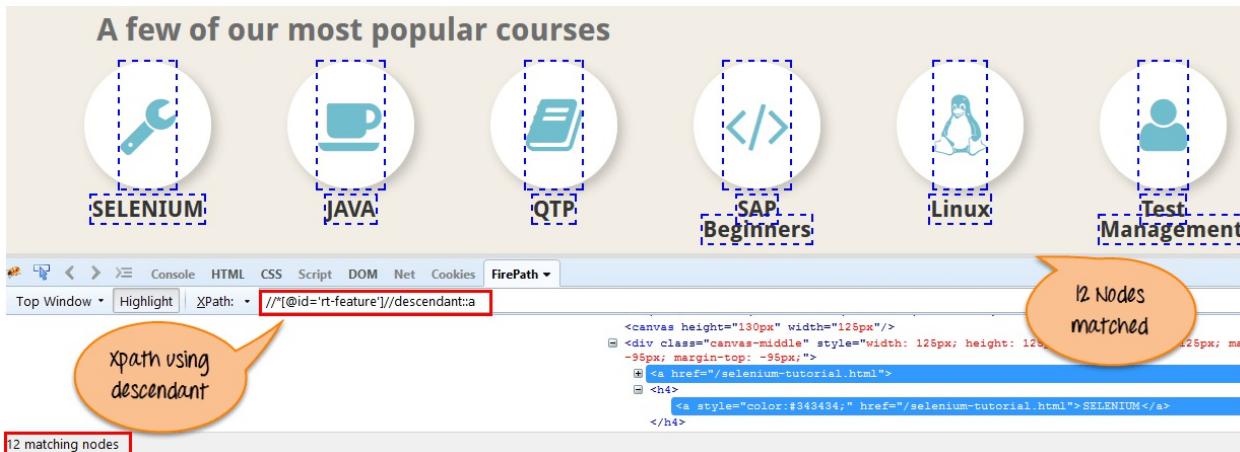
as it represents self-element.

```
Xpath =//*[@type='password']//self::input
```

h) Descendant: Selects the descendants of the current node as shown in the below screen.

In the below expression, it identifies all the element descendants to current element ('Main body surround' frame element) which means down under the node (child node , grandchild node, etc.).

```
Xpath=//*[@id='rt-feature']//descendant::a
```



There are 12 "link" nodes matching by using "descendant" axis. If you want to focus on any particular element then you can use the below XPath:

```
Xpath=//*[@id='rt-feature']//descendant::a[1]
```

You can change the XPath according to the requirement by putting [1], [2].....and so on.

Summary:

XPath is required to find an element on the web page as to do an

operation on that particular element.

- There are two types of XPath:
 - **Absolute XPath**
 - **Relative XPath**
- XPath Axes are the methods used to find dynamic elements, which otherwise not possible to find by normal XPath method
- XPath expression select nodes or list of nodes on the basis of attributes like ID , Name, Classname, etc. from the XML document .

Chapter 17: How TestNG makes Selenium tests easier

What is TestNG?

TestNG is an automation testing framework in which NG stands for "Next Generation". TestNG is inspired from JUnit which uses the annotations (@).

- Using TestNG you can generate a proper report, and you can easily come to know how many test cases are passed, failed and skipped.
- You can execute failed test case separately. For example.
 - Suppose, you have five test cases, one method is written for each test case (Assume that the program is written using the main method without using testNG). When you run this program first, three methods are executed successfully, and the fourth method is failed. Then correct the errors present in the fourth method, now you want to run only fourth method because first three methods are anyway executed successfully. This is not possible without using TestNG.
- The TestNG provides an option, i.e., testing-failed.xml file in test-output folder. If you want to run only failed test cases means you run this XML file. It will execute only failed test cases.

Beside above concept, you will learn more on TestNG, like what are the Advantages of TestNG, how to create test methods using @test annotations, how to convert these classes into testing suite file and execute through the eclipse as well as from the command line.

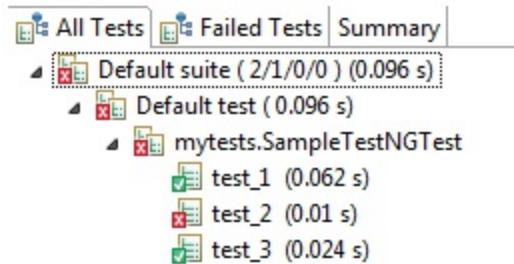
Why Use TestNG with Selenium?

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

Most Selenium users use this more than Junit because of its advantages. There are so many features of TestNG, but we will only focus on the most important ones that we can use in Selenium.

Following are key features of TestNG

- Generate the report in a proper format including a number of test cases runs, the number of test cases passed, the number of test cases failed, and the number of test cases skipped.
- Multiple test cases can be grouped more easily by converting them into testng.xml file. In which you can make priorities which test case should be executed first.
- The same test case can be executed multiple times without loops just by using keyword called 'invocation count.'
- Using testng, you can execute multiple test cases on multiple browsers, i.e., cross browser testing.
- The testing framework can be easily integrated with tools like Maven, Jenkins, etc.
- Annotations used in the testing are very easy to understand ex: @BeforeMethod, @AfterMethod, @BeforeTest, @AfterTest
- WebDriver has no native mechanism for generating reports. TestNG can generate the report in a readable format like the one shown below.



- TestNG simplifies the way the tests are coded. There is no more need for a static main method in our tests. The sequence of actions is regulated by easy-to-understand annotations that do not require methods to be static.

**USUAL STRUCTURE
(somewhat difficult to read)**

```

public class myclass {

    public static String baseUrl = "http://newtours.demoaut.com/";
    public static WebDriver driver = new FirefoxDriver();

    public static void main(String[] args) {
        driver.get(baseUrl);
        verifyHomepageTitle();
        driver.quit();
    }

    public static void verifyHomepageTitle() {
        String expectedTitle = "Welcome: Mercury Tours";
        String actualTitle = driver.getTitle();
        try {
            Assert.assertEquals(actualTitle, expectedTitle);
            System.out.println("Test Passed");
        } catch (Throwable e) {
            System.out.println("Test Failed");
        }
    }
}

```

TestNG structure (easier to understand)

```
public class SampleTestNGTest {
    public String baseUrl = "http://newtours.demoaut.com/";
    public WebDriver driver;

    @BeforeTest
    public void setBaseURL() {
        driver = new FirefoxDriver();
        driver.get(baseUrl);
    }

    @Test
    public void verifyHomepageTitle() {
        String expectedTitle = "Welcome: Mercury Tours";
        String actualTitle = driver.getTitle();
        Assert.assertEquals(actualTitle, expectedTitle);
    }

    @AfterTest
    public void endSession() {
        driver.quit();
    }
}
```

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

Advantages of TestNG over JUnit

There are three major advantages of TestNG over JUnit:

- Annotations are easier to understand
- Test cases can be grouped more easily
- Parallel testing is possible

Annotations in TestNG are lines of code that can control how

the method below them will be executed. They are always preceded by the @ symbol. A very early and quick example is the one shown below.

These are 2 examples of annotations

```
@Test(priority = 0)
public void goToHomepage() {
    driver.get(baseUrl);
    Assert.assertEquals(driver.getTitle(), "Welcome: Mercury Tours");
}

@Test(priority = 1)
public void logout() {
    driver.findElement(By.linkText("SIGN-OFF")).click();
    Assert.assertEquals("Sign-on: Mercury Tours", driver.getTitle());
}
```

The example above simply says that the method `goToHomepage()` should be executed first before `logout()` because it has a lower priority number

Annotations will be discussed later in the section named "Annotations used in TestNG," so it is perfectly ok if you do not understand the above example just yet. It is just important to note for now that annotations in TestNG are easier to code and understand than in JUnit.

The ability to run tests in parallel is available in TestNG but not in JUnit, so it is the more preferred framework of testers using Selenium Grid.

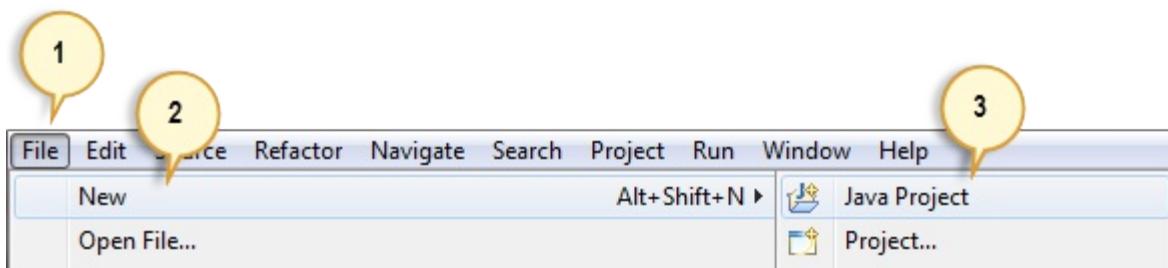
First test case using annotations

Before we create a test case, we should first setup a new TestNG

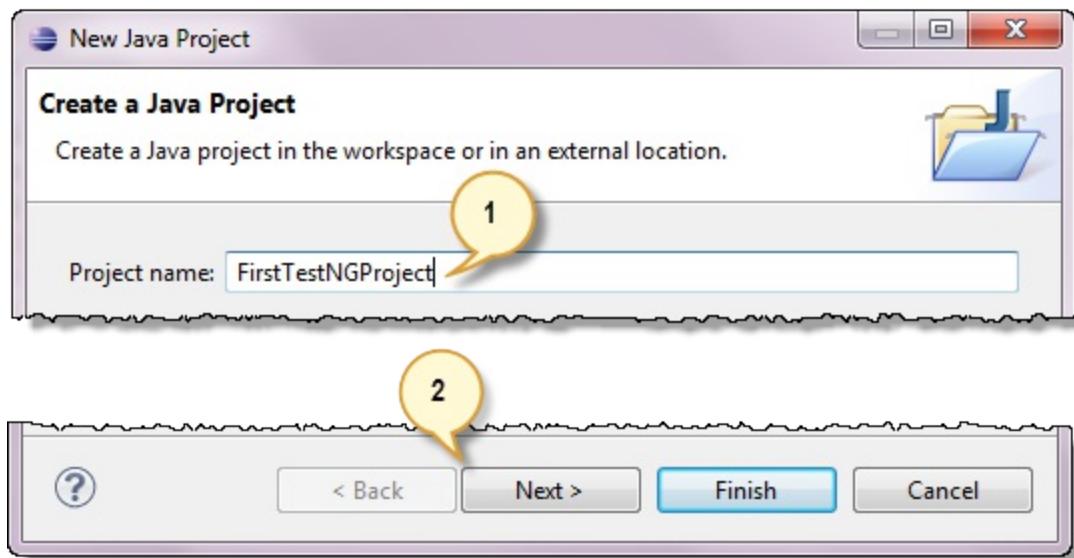
Project in Eclipse and name it as "FirstTestNGProject".

Setting up a new TestNG Project

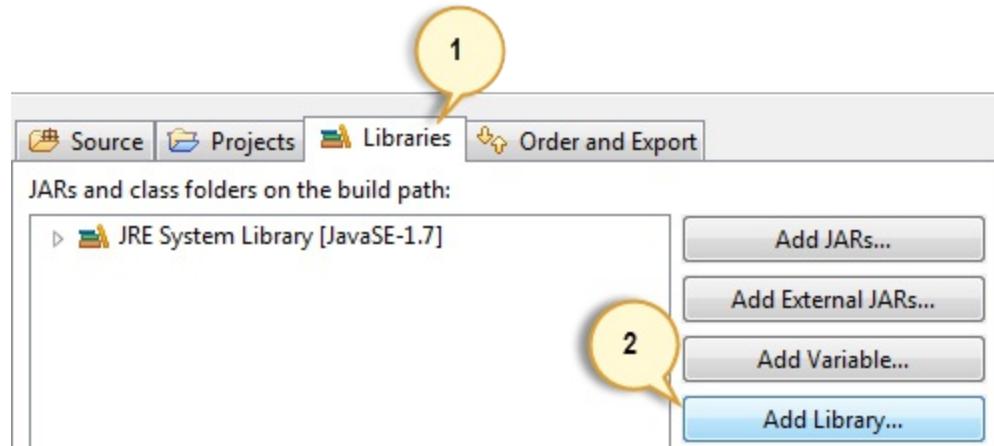
Step 1: Click File > New > Java Project



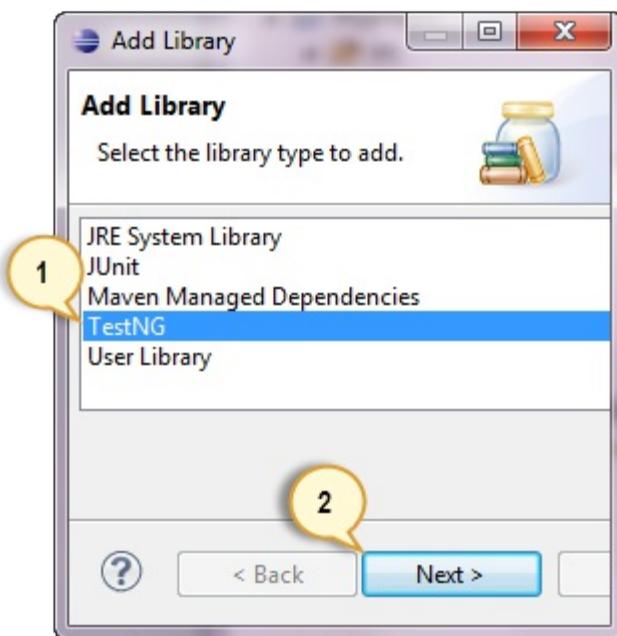
Step 2: Type "FirstTestNGProject" as the Project Name then click Next.



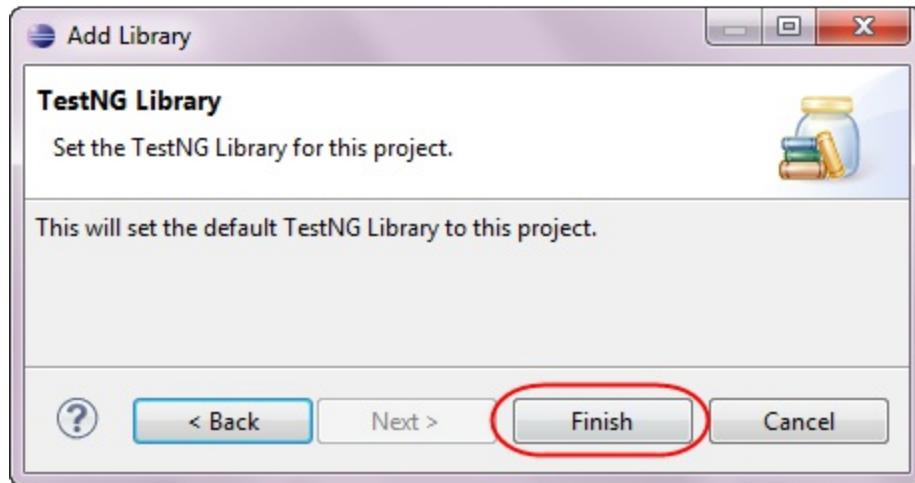
Step 3: We will now start to import the TestNG Libraries onto our project. Click on the "Libraries" tab, and then "Add Library..."



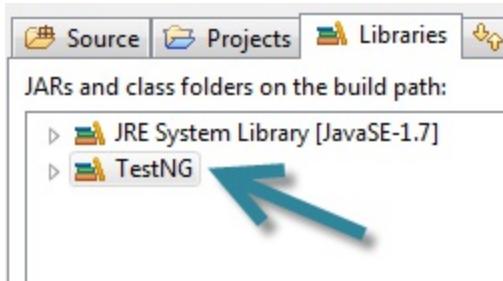
Step 4: On the Add Library dialog, choose "TestNG" and click Next.



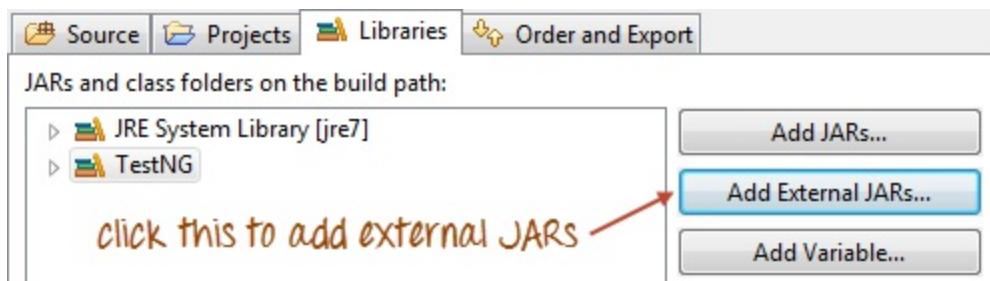
Step 5: Click Finish.



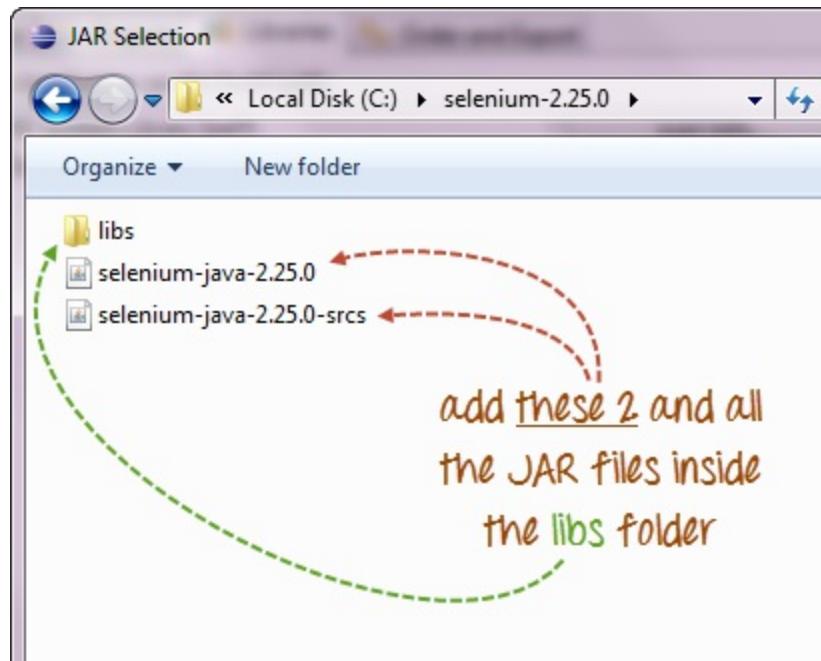
You should notice that TestNG is included on the Libraries list.



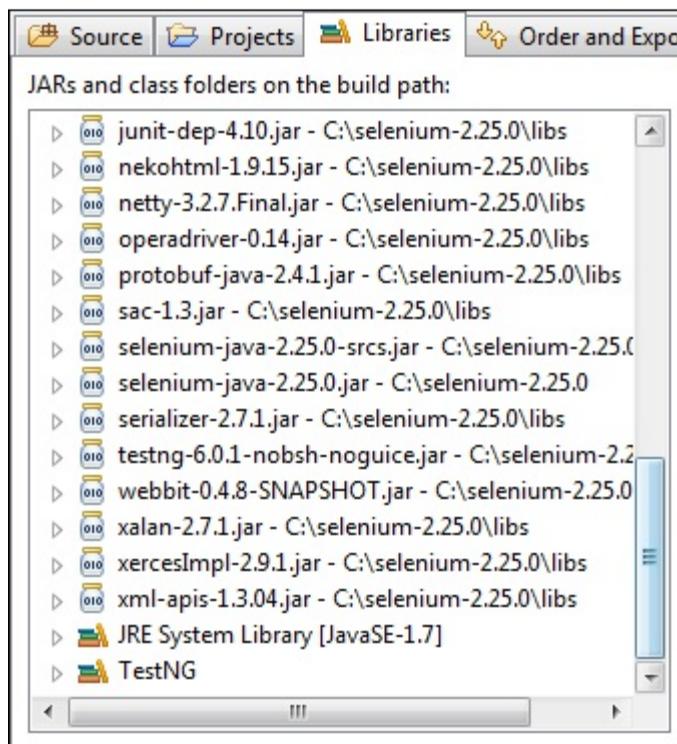
Step 6: We will now add the JAR files that contain the Selenium API. These files are found in the Java client driver that we downloaded from <http://docs.seleniumhq.org/download/> when we were installing Selenium and Eclipse in the previous chapters.



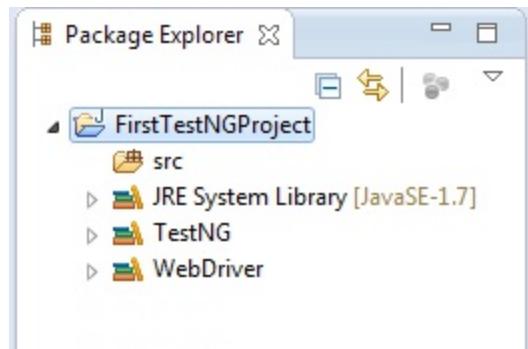
Then, navigate to where you have placed the Selenium JAR files.



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



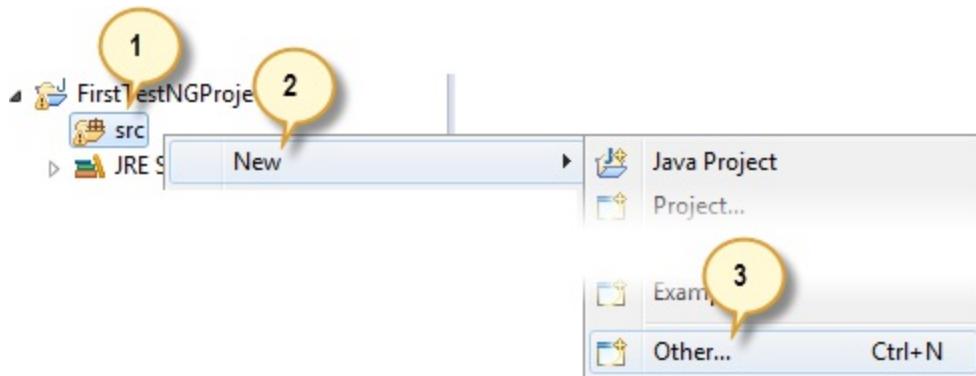
Step 7: Click Finish and verify that our FirstTestNGProject is visible on Eclipse's Package Explorer window.



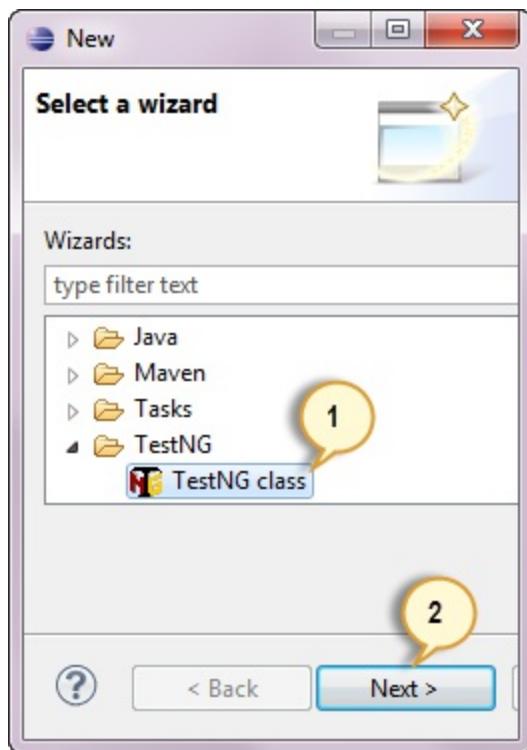
Creating a New TestNG Test File

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

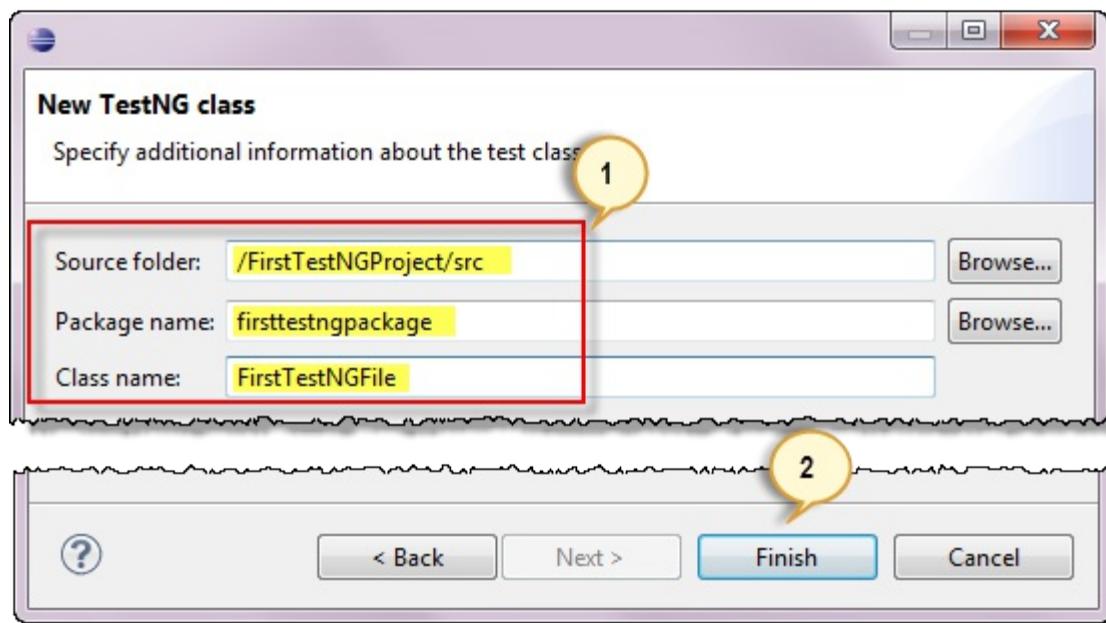
Step 1: Right-click on the "src" package folder then choose New > Other...



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.



```
FirstTestNGFile.java
FirstTestNGProject > src > firsttestngpackage
1 package firsttestngpackage;
2
3 import org.testng.annotations.Test;
4
5 public class FirstTestNGFile {
6     @Test
7     public void f() {
8     }
9 }
10
```

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.

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

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

    @Test
    public void verifyHomepageTitle() {

        System.out.println("launching firefox browser");
        System.setProperty("webdriver.firefox.marionette",
driverPath);
        driver = new FirefoxDriver();
        driver.get(baseUrl);
        String expectedTitle = "Welcome: Mercury Tours";
        String actualTitle = driver.getTitle();
```

```
        Assert.assertEquals(actualTitle, expectedTitle);
        driver.close();
    }
}
```

Notice the following.

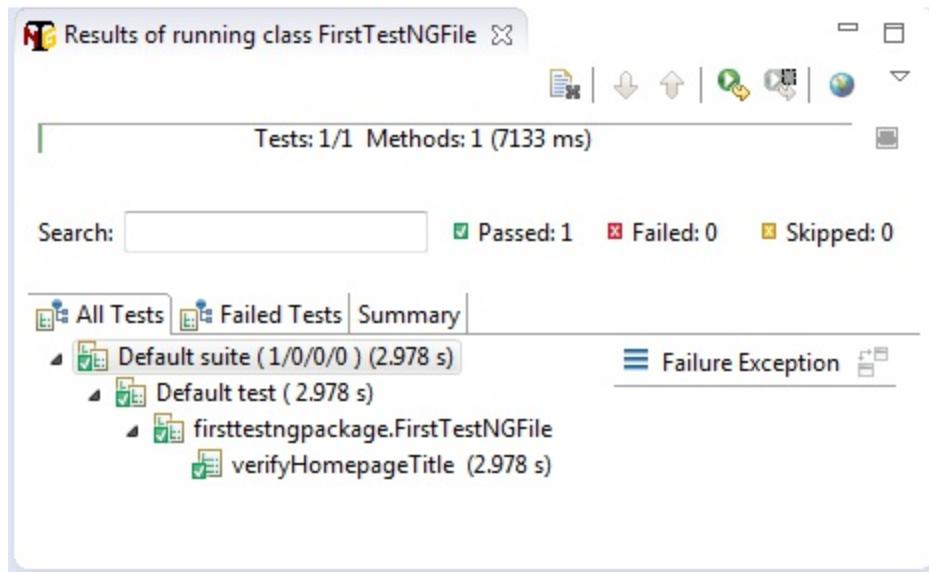
- TestNG does not require you to have a main() method.
- Methods need not be static.
- We used the @Test annotation. **@Test is used to tell that the method under it is a test case.** In this case, we have set the verifyHomepageTitle() method to be our test case, so we placed an '@Test' annotation above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

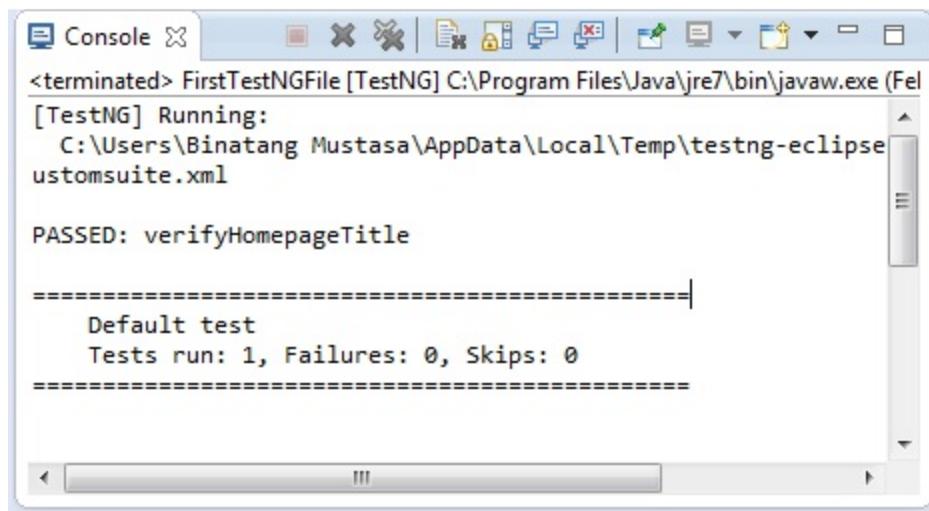
Running the Test

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.

TESTNG Results Window

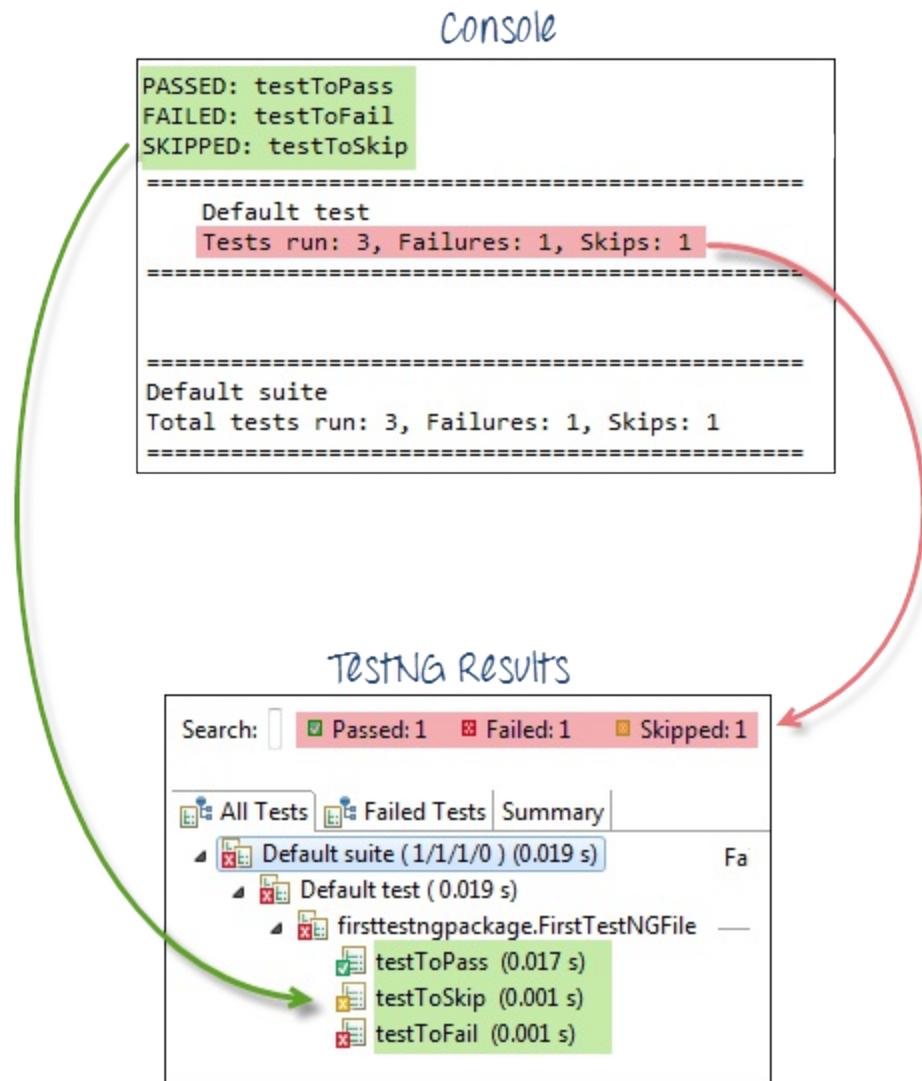


Console window



Checking reports created by TestNG

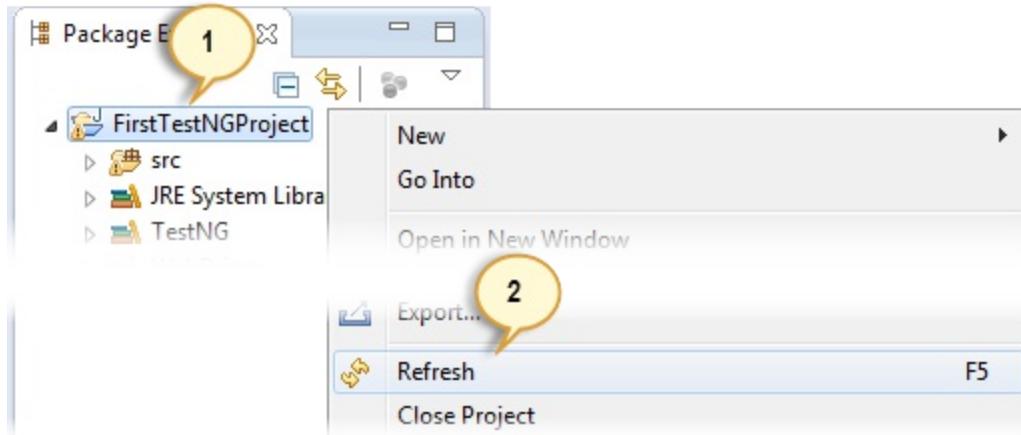
The Console window in Eclipse gives a text-based report of our test case results while the TestNG Results window gives us a graphical one.



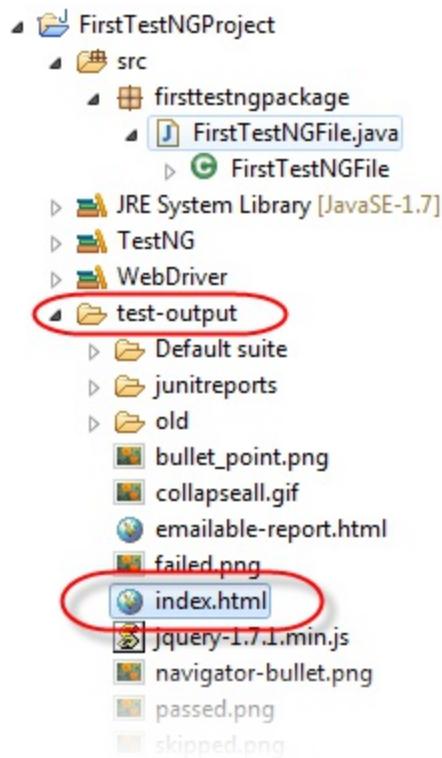
Generating HTML Reports

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.

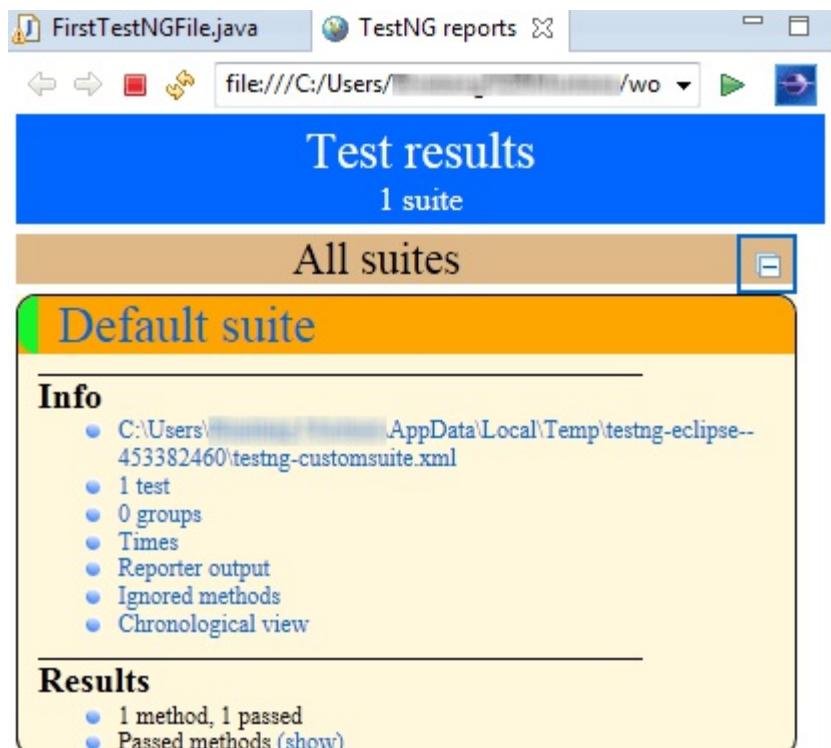


Step 2: Notice that a "test-output" folder was created. Expand it and look for an index.html file. This HTML file is a report of the results of the most recent test run.



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.

```

public class FirstTestNGFile {

    @Test
    public void c_test() {
        Assert.fail();
    }

    @Test
    public void a_test() {
        Assert.assertTrue(true);
    }

    @Test
    public void b_test() {
        throw new SkipException("Skipping b_test...");
    }
}

```

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

The screenshot shows the TestNG report interface. At the top, it says "Default suite". Below that is an "Info" section with the following details:

- C:\Users\...\\AppData\Local\Temp\testng-eclipse-127915423\testng-customsuite.xml
- 1 test
- 0 groups
- Times
- Reporter output
- Ignored methods
- Chronological view**

A red arrow points from the "Chronological view" option in the info section down to a table below. The table has a header "Methods in chronological order" and a sub-header "firsttestingpackage.FirstTestNGFile". It lists three methods: "a_test", "b_test", and "c_test". The "c_test" row is highlighted with a red box. A red arrow points from the "c_test" row in the table back up to the "Chronological view" option in the info section.

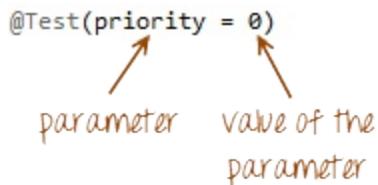
Methods in chronological order		
firsttestingpackage.FirstTestNGFile	a_test	0 ms
	b_test	17 ms
	c_test	20 ms

tests were executed alphabetically

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.

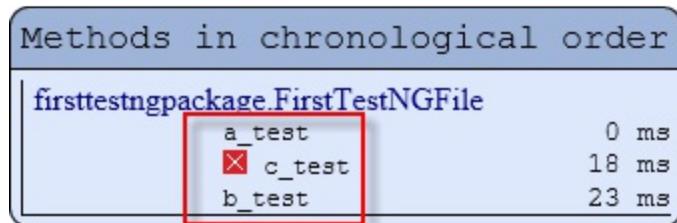


TestNG will execute the @Test annotation with the lowest priority value up to the largest. There is no need for your priority values to be consecutive.

```

public class FirstTestNGfile {
    @Test(priority = 3) ← the 2nd least priority value so
    public void c_test() {           this will be executed 2nd
        Assert.fail();
    }
    @Test(priority = 0) ← this has the lowest priority value
    public void a_test() {           so this will be executed first
        Assert.assertTrue(true);
    }
    @Test(priority = 7) ← largest priority value so this will
    public void b_test() {           be executed last
        throw new SkipException("Skipping b_test...");
    }
}
  
```

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.

```
@Test(priority = 0, alwaysRun = true)
```



@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.

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

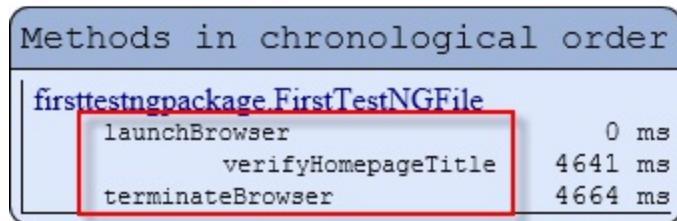
Applying the logic presented by the table and the code above, we can predict that the sequence by which methods will be executed is:

- 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.

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

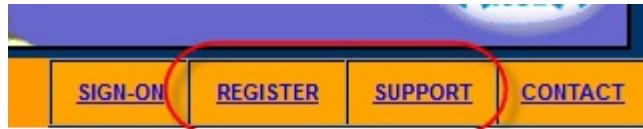
Run the code above and notice that



@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.

```
package firsttestngpackage;
import org.openqa.selenium.*;
```

```
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.*;
@Test
public class firsttestngfile {
    public String baseUrl =
"http://demo.guru99.com/test/newtours/";
    String driverPath = "C:\\geckodriver.exe";
    public WebDriver driver;
    public String expected = null;
    public String actual = null;
    @BeforeTest
    public void launchBrowser() {
        System.out.println("launching firefox browser");
        System.setProperty("webdriver.firefox.marionette",
driverPath);
        driver= new FirefoxDriver();
        driver.get(baseUrl);
    }

    @BeforeMethod
    public void verifyHomepageTitle() {
        String expectedTitle = "Welcome: Mercury Tours";
        String actualTitle = driver.getTitle();
        Assert.assertEquals(actualTitle, expectedTitle);
    }
    @Test(priority = 0)
    public void register(){
        driver.findElement(By.linkText("REGISTER")).click() ;
        expected = "Register: Mercury Tours";
        actual = driver.getTitle();
        Assert.assertEquals(actual, expected);
    }
    @Test(priority = 1)
    public void support() {
        driver.findElement(By.linkText("SUPPORT")).click() ;
        expected = "Under Construction: Mercury Tours";
        actual = driver.getTitle();
        Assert.assertEquals(actual, expected);
    }
    @AfterMethod
    public void goBackToHomepage ( ) {
        driver.findElement(By.linkText("Home")).click() ;
```

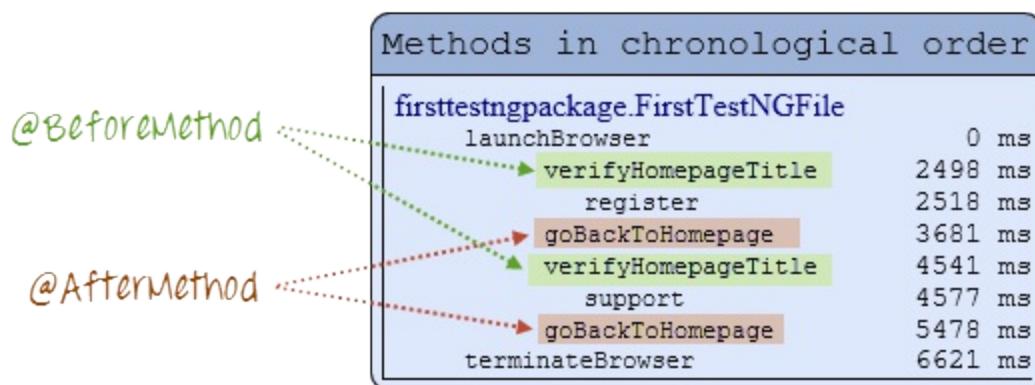
```

    }

    @AfterTest
    public void terminateBrowser(){
        driver.close();
    }
}

```

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



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

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.

Chapter 18: Handling Date Time Picker using Selenium

For DateTime selection, HTML5 has a new control shown below.

Select Date Select time

Open this page in Chrome

Birthday (date and time):

Above page can be accessed here

If we see the DOM of the DateTime Picker control, there will be only one input box for both date and time.

Open this page in Chrome

Date Time

Birthday (date and time):

There is only one control for both Date and Time

Screenshot of the browser's developer tools showing the DOM structure:

```

<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <form action="birthdate.php" method="post" name="bdate">
      <h3>Open this page in Chrome</h3>
      " Birthday (date and time): "
      <input type="datetime-local" name="bdaytime">
      <input type="submit">
    </form>
  </body>
</html>

```

The code shows a single `<input type="datetime-local" name="bdaytime">` element that spans both the date and time fields, which corresponds to the visual representation in the screenshot.

So to handle this type of control first we will fill date without separating with delimiter, i.e. if date is 09/25/2013, then we will pass 09252013 to the input box. Once done, we will shift focus from date to time by pressing 'tab' & fill time.

If we need to fill 02:45 PM , we will pass it a '0245PM' to the same input box.

The code for datepicker looks like this -

```
import java.util.List;  
  
import java.util.concurrent.TimeUnit;  
  
import org.openqa.selenium.By;  
  
import org.openqa.selenium.Keys;  
  
import org.openqa.selenium.WebDriver;  
  
import org.openqa.selenium.WebElement;  
  
import org.openqa.selenium.chrome.ChromeDriver;  
  
import org.testng.annotations.Test;  
  
public class DateTimePicker {  
  
    @Test  
  
    public void dateTimePicker(){  
  
        System.setProperty("webdriver.chrome.driver",  
"chromedriver.exe");  
  
        WebDriver driver = new ChromeDriver();  
  
        driver.manage().timeouts().implicitlyWait(10,  
TimeUnit.SECONDS);
```

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

//Find the date time picker control

WebElement dateBox =
driver.findElement(By.xpath("//form//input[@name='bdaytime']"));

//Fill date as mm/dd/yyyy as 09/25/2013

dateBox.sendKeys("09252013");

//Press tab to shift focus to time field

dateBox.sendKeys(Keys.TAB);

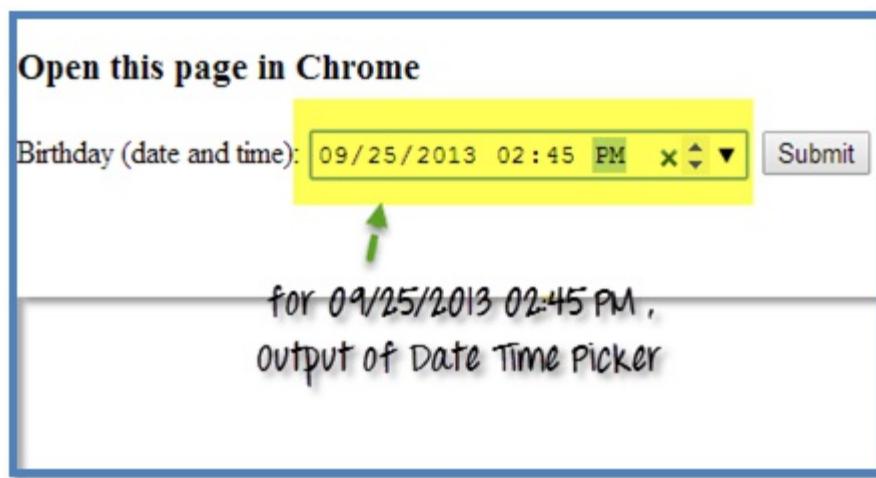
//Fill time as 02:45 PM

dateBox.sendKeys("0245PM");

}

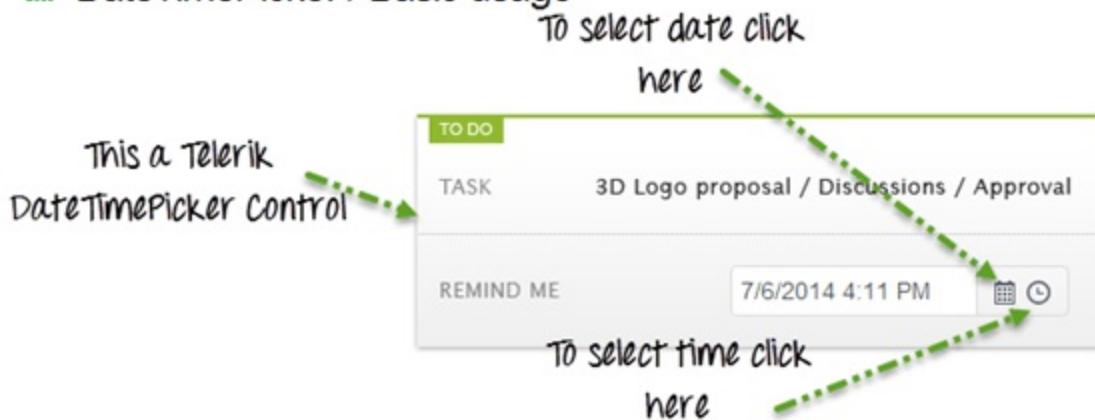
}
```

Output will be like-

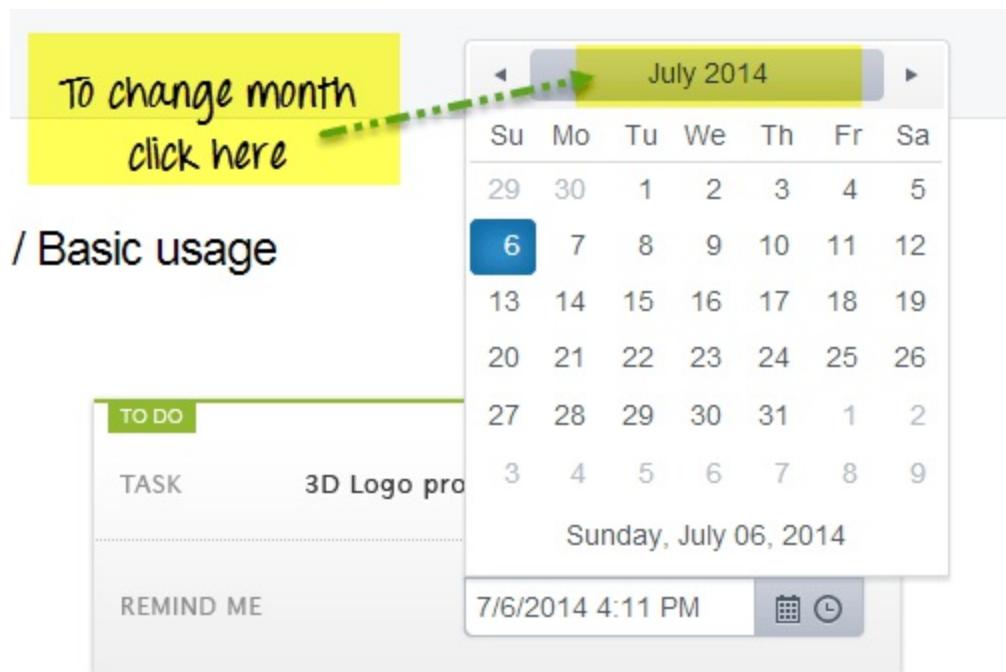


Lets look at another Calendar example. We will use Telerik DateTimePicker control. Can be accessed here

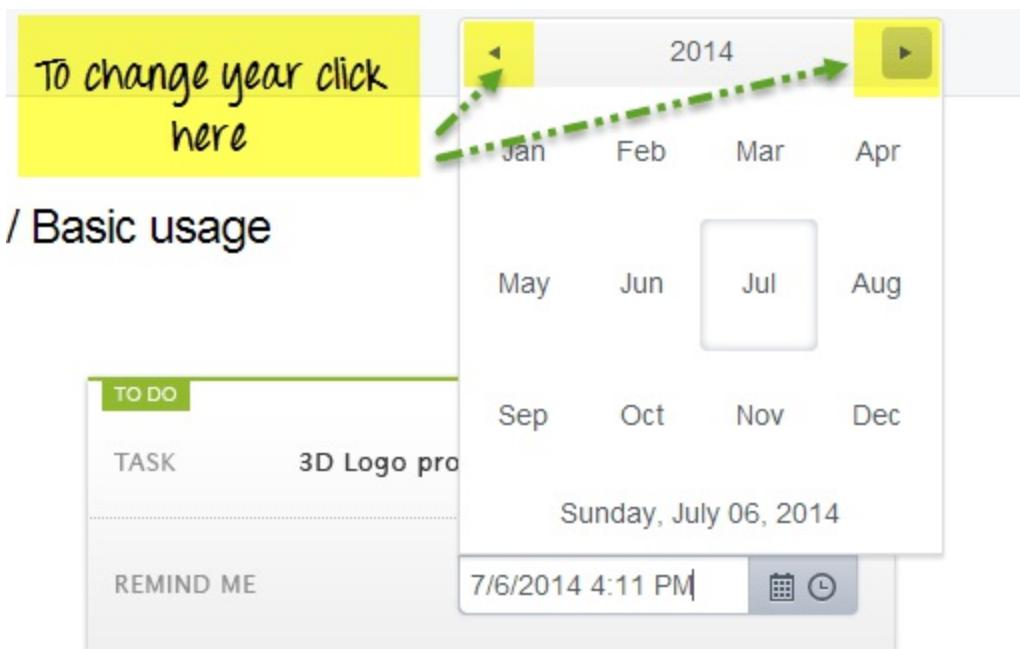
DatePickerController / Basic usage



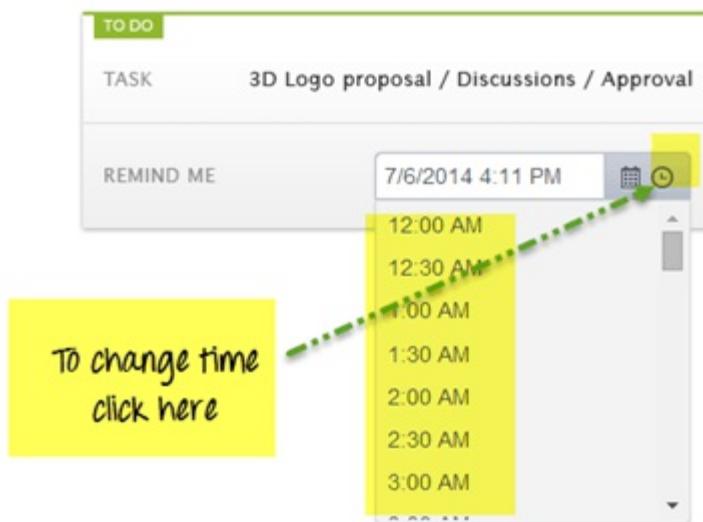
Here if we need to change the month, we have to click on the middle of the calendar header.



Similarly if we need to change the year then we can do it by clicking next or previous links on the datepicker.



And finally for changing the time we can select correct time from the dropdown (Note: Here time is selected in a gap of 30 min. i.e., 12:00, 12:30, 1:00, 1:30 etc.).



A complete example looks like-

```
import java.util.Calendar;
```

```
import java.util.List;
import java.util.concurrent.TimeUnit;
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 DatePicker {
    @Test
    public void testDatePicker() throws Exception{
        //DAte and Time to be set in textbox
        String dateTime ="12/07/2014 2:00 PM";
        WebDriver driver = new FirefoxDriver();
        driver.manage().window().maximize();
        driver.get("http://demos.telerik.com/kendo-
ui/datetimepicker/index");
        driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
        //button to open calendar
        WebElement selectDate =
driver.findElement(By.xpath("//span[@aria-
controls='datetimepicker_dateview']"));
        selectDate.click();
```

```
//button to move next in calendar

    WebElement nextLink =
driver.findElement(By.xpath("//div[@id='datetimepicker_dateview']
//div[@class='k-header']//a[contains(@class, 'k-nav-next')]"));

    //button to click in center of calendar header

    WebElement midLink =
driver.findElement(By.xpath("//div[@id='datetimepicker_dateview']
//div[@class='k-header']//a[contains(@class, 'k-nav-fast')]"));

    //button to move previous month in calendar

    WebElement previousLink =
driver.findElement(By.xpath("//div[@id='datetimepicker_dateview']
//div[@class='k-header']//a[contains(@class, 'k-nav-prev')]"));

    //Split the date time to get only the date part

    String date_dd_MM_yyyy[] = (dateTime.split(" ")
[0]).split("/");

    //get the year difference between current year and year
to set in calander

    int yearDiff = Integer.parseInt(date_dd_MM_yyyy[2])-
Calendar.getInstance().get(Calendar.YEAR);

    midLink.click();

    if(yearDiff!=0){

        //if you have to move next year

        if(yearDiff>0){

            for(int i=0;i< yearDiff;i++){

                System.out.println("Year Diff->" + i);

                nextLink.click();
            }
        }
    }
}
```

```
    }

    //if you have to move previous year

    else if(yearDiff<0){

        for(int i=0;i< (yearDiff*(-1));i++){

            System.out.println("Year Diff->" + i);

            previousLink.click();

        }

    }

    Thread.sleep(1000);

    //Get all months from calendar to select correct one

    List<WebElement> list_AllMonthToBook =
driver.findElements(By.xpath("//div[@id='datetimepicker_dateview']//table//tbody//td[not(contains(@class, 'k-other-month'))]"));

list_AllMonthToBook.get(Integer.parseInt(date_dd_MM_yyyy[1])-1).click();

    Thread.sleep(1000);

    //get all dates from calendar to select correct one

    List<WebElement> list_AllDateToBook =
driver.findElements(By.xpath("//div[@id='datetimepicker_dateview']//table//tbody//td[not(contains(@class, 'k-other-month'))]"));

list_AllDateToBook.get(Integer.parseInt(date_dd_MM_yyyy[0])-1).click();
```

```
//FOR TIME

WebElement selectTime =
driver.findElement(By.xpath("//span[@aria-
controls='datetimepicker_timeview']"));

//click time picker button
selectTime.click();

//get list of times

List<WebElement> allTime =
driver.findElements(By.xpath("//div[@data-role='popup']
[contains(@style, 'display: block')]//ul//li[@role='option']"));

dateTime = dateDateTime.split(" ")[1]+" "+dateDateTime.split(
")")[2];

//select correct time

for (WebElement webElement : allTime) {

    if(webElement.getText().equalsIgnoreCase(dateTime))

    {

        webElement.click();

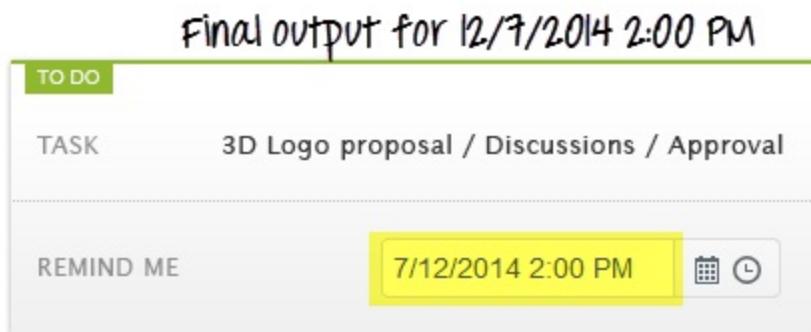
    }

}

}

}
```

Output will be like



Chapter 19: Alert & Popup handling in Selenium

In this tutorial, we will learn about different types of alert found in web application Testing and how to handle Alert in Selenium WebDriver. We will also see how do we accept and reject the alert depending upon the alert types.

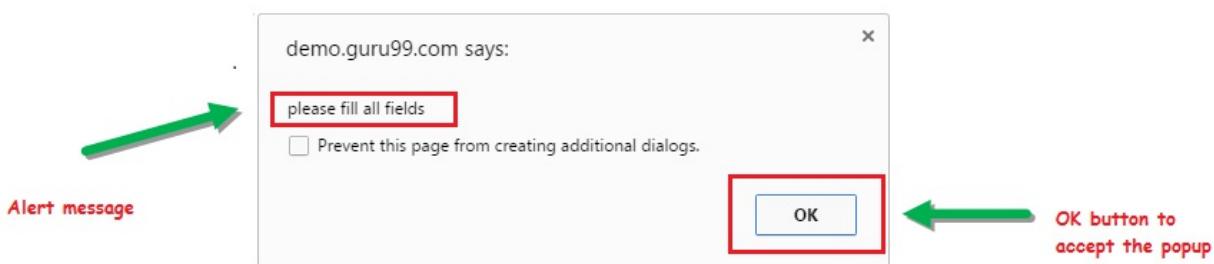
What is Alert?

Alert is a small message box which displays on-screen notification to give the user some kind of information or ask for permission to perform certain kind of operation. It may be also used for warning purpose.

Here are few alert types:

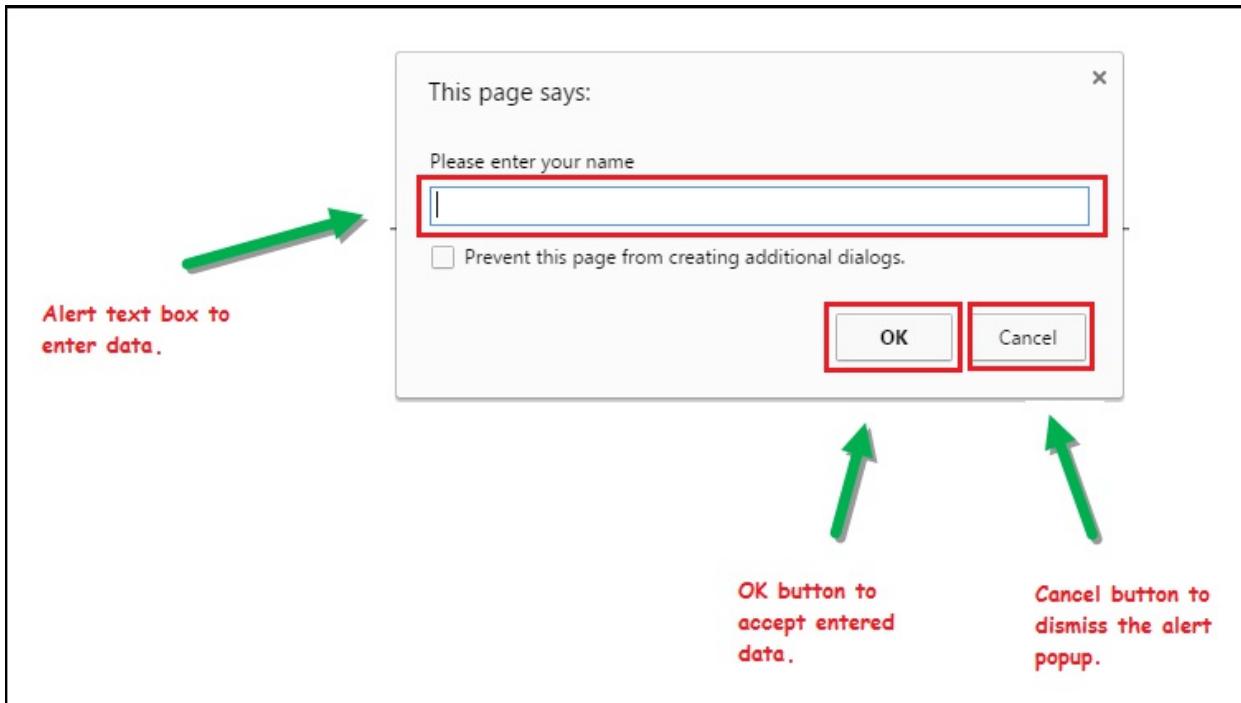
1) Simple Alert

This simple alert displays some information or warning on the screen.



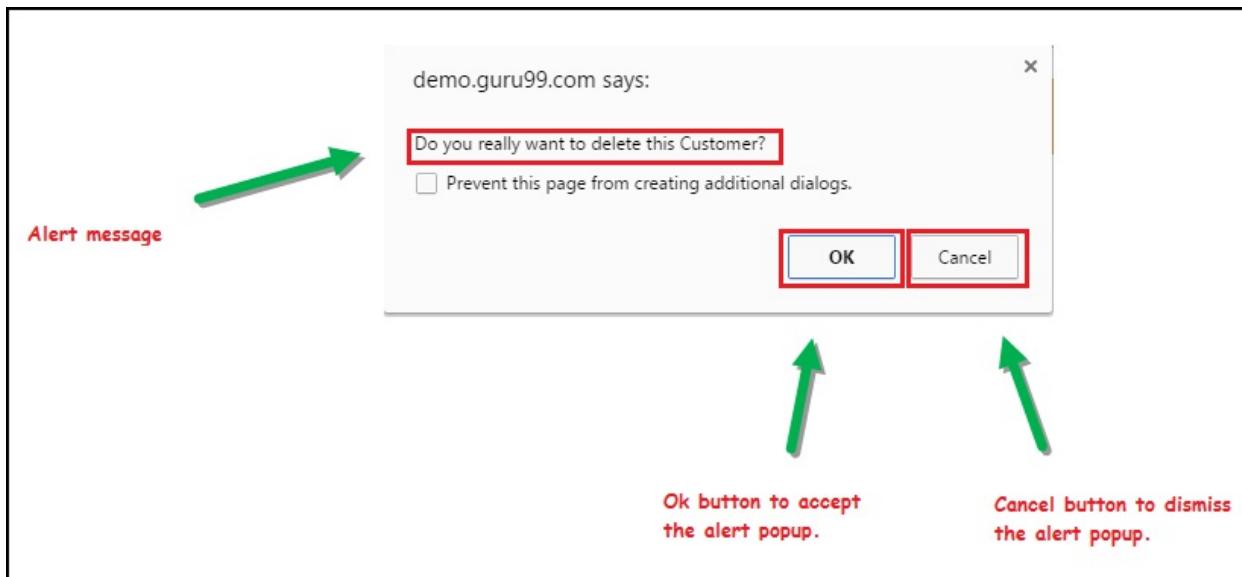
2) Prompt Alert.

This Prompt Alert asks some input from the user and selenium webdriver can enter the text using sendkeys(" input.... ").



3) Confirmation Alert.

This confirmation alert asks permission to do some type of operation.



How to handle Alert in Selenium WebDriver

Alert interface provides the below few methods which are widely used in Selenium Webdriver.

- 1) void dismiss() // To click on the 'Cancel' button of the alert.

```
driver.switchTo().alert().dismiss();
```

- 2) void accept() // To click on the 'OK' button of the alert.

```
driver.switchTo().alert().accept();
```

- 3) String getText() // To capture the alert message.

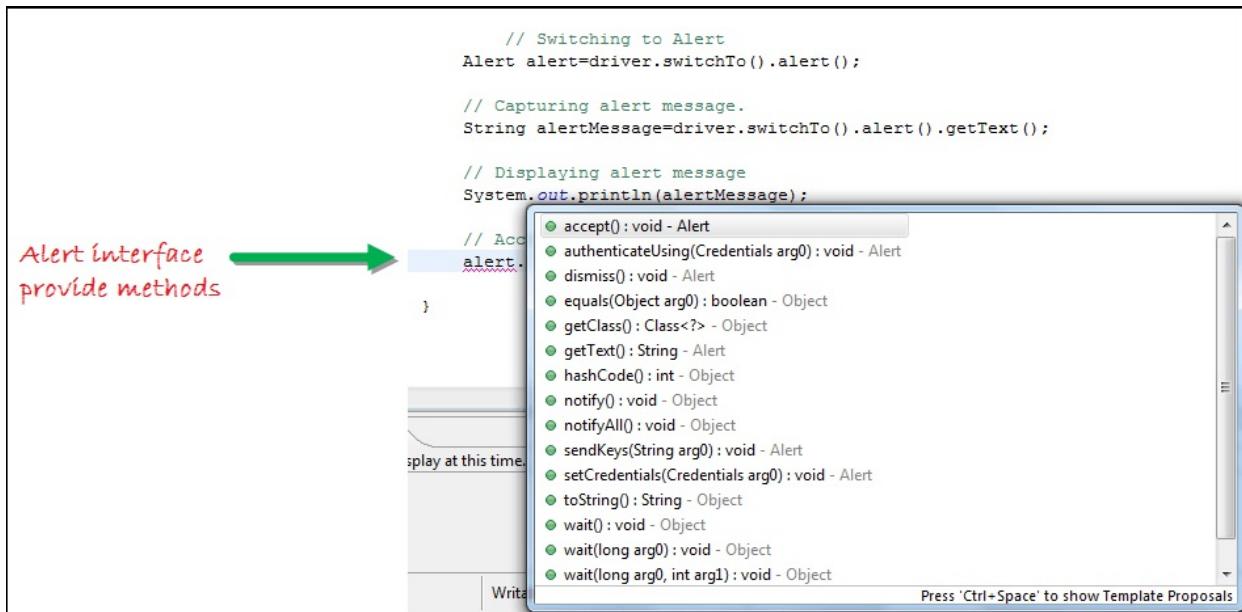
```
driver.switchTo().alert().getText();
```

- 4) void sendKeys(String stringToSend) // To send some data to alert box.

```
driver.switchTo().alert().sendKeys("Text");
```

You can see a number of Alert methods are displayed as shown in below screen suggested by Eclipse.

We can easily switch to alert from the main window by using Selenium's **.switchTo()** method.



```

// Switching to Alert
Alert alert=driver.switchTo().alert();

// Capturing alert message.
String alertMessage=driver.switchTo().alert().getText();

// Displaying alert message
System.out.println(alertMessage);

// Accepting alert
alert.accept();
}

```

Alert interface provide methods

accept(): void - Alert
 authenticateUsing(Credentials arg0): void - Alert
 dismiss(): void - Alert
 equals(Object arg0): boolean - Object
 getClass(): Class<?> - Object
 getText(): String - Alert
 hashCode(): int - Object
 notify(): void - Object
 notifyAll(): void - Object
 sendKeys(String arg0): void - Alert
 setCredentials(Credentials arg0): void - Alert
 toString(): String - Object
 wait(): void - Object
 wait(long arg0): void - Object
 wait(long arg0, int arg1): void - Object

Press 'Ctrl+Space' to show Template Proposals

Now we automate the given below scenario.

In this scenario, we will use Guru99 demo site to illustrate Selenium Alert handling.

Step 1) Launch the web browser and open the site "http://demo.guru99.com/test/delete_customer.php"

Step 2) Enter Any Customer id.

Delete Customer Form

Customer ID

← Enter ID

Step 3) After entering the customer ID, Click on the "Submit" button.

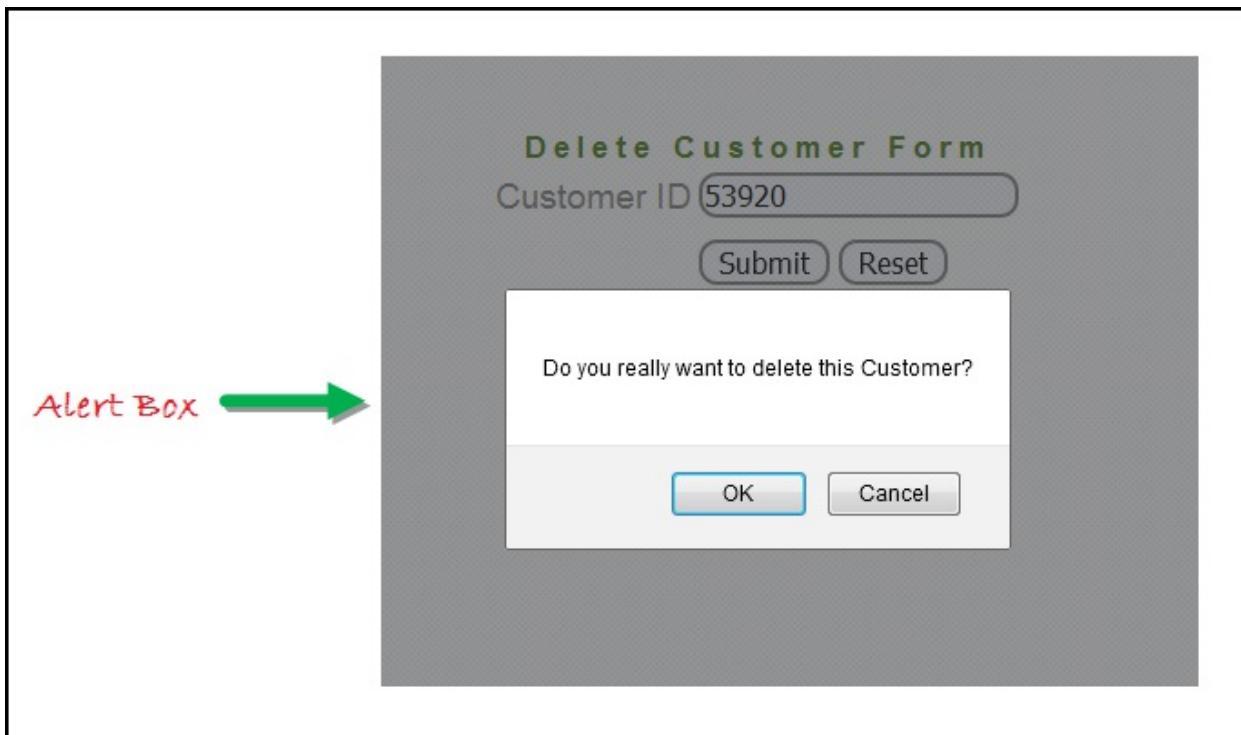
Delete Customer Form

Customer ID

↑

Click on Submit

Step 4) Reject/accept the alert.



Handling Alert in Selenium Webdriver using above scenario

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.NoAlertPresentException;
import org.openqa.selenium.Alert;

public class AlertDemo {

    public static void main(String[] args) throws
NoAlertPresentException, InterruptedException {
System.setProperty("webdriver.chrome.driver", "G:\\chromedriver.e
xe");
    WebDriver driver = new ChromeDriver();

    // Alert Message handling

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

```
driver.findElement(By.name("cusid")).sendKeys("53920");
driver.findElement(By.name("submit")).submit();

// Switching to Alert
Alert alert = driver.switchTo().alert();

// Capturing alert message.
String alertMessage=
driver.switchTo().alert().getText();

// Displaying alert message
System.out.println(alertMessage);
Thread.sleep(5000);

// Accepting alert
alert.accept();
}

}
```

Output :

When you execute the above code, it launches the site. Try to delete Customer ID by handling confirmation alert that displays on the screen, and thereby deleting customer id from the application.

How to handle Selenium Pop-up window using Webdriver

In automation, when we have multiple windows in any web application, the activity may need to switch control among several windows from one to other in order to complete the operation. After completion of the operation, it has to return to the main window i.e. parent window. We will see this further in the article with an example.

In selenium web driver there are methods through which we can handle multiple windows.

Driver.getWindowHandles();

To handle all opened windows by web driver, we can use "Driver.getWindowHandles()" and then we can switch window from one window to another in a web application. Its return type is Iterator<String>.

Driver.getWindowHandle();

When the site opens, we need to handle the main window by **driver.getWindowHandle()**. This will handle the current window that uniquely identifies it within this driver instance. Its return type is String.

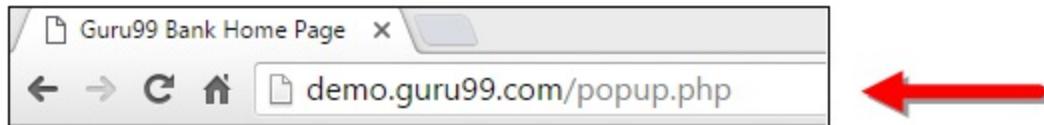
To handle multiple windows in Selenium WebDriver, We follow the following steps.

Now, we will automate the given below scenario to see how to handle multiple windows using Selenium Webdriver.

In this scenario, we will use "Guru99" demo site to illustrate window handling.

Step 1) Launch the site.

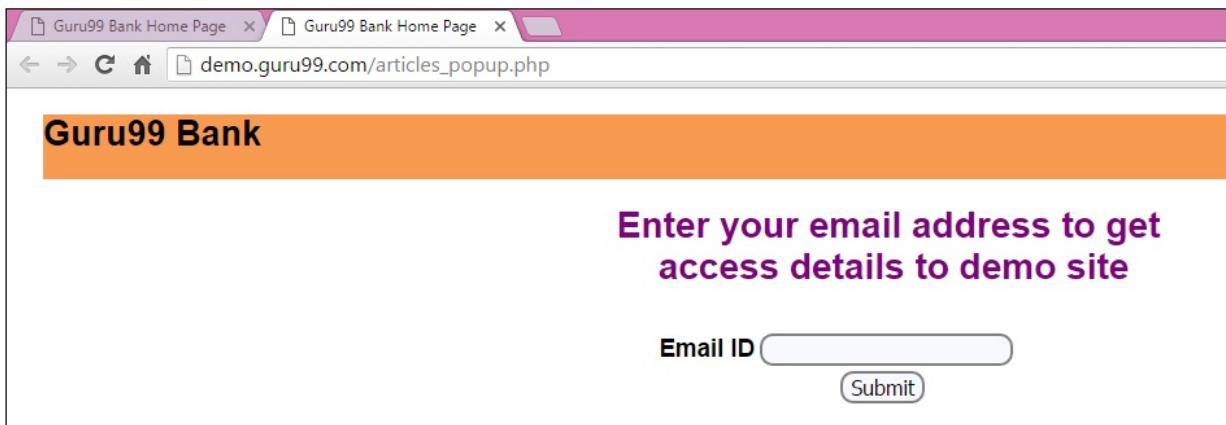
Launch the browser and open the site "
http://demo.guru99.com/popup.php"

**Step 2)** Click on link "Click Here".

When the user clicks on the " Click Here " link, new child window opens.

**Step 3)** New Child Window opens.

A new window opens, ask the user to enter email id and submit the page.

**Step 4)** Enter your email ID and submit.

Enter your email address to get access details to demo site

Email ID 



Step 5) Display the Access Credentials on submitting the page.

Access details to demo site.

User ID :	mngr780
Password :	YremurE

This access is valid only for 20 days.



When you execute the code, you will see the child window is open in new tab.

1. Close the Child window on which credentials are displayed.



2. Switch to the parent window.



Handling multiple windows in selenium webdriver using above scenario.

```
import java.util.Iterator;
import java.util.Set;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class WindowHandle_Demo {

    public static void main(String[] args) throws
InterruptedException {
        WebDriver driver=new FirefoxDriver();

        //Launching the site.
        driver.get("http://demo.guru99.com/popup.php");
    }
}
```

```

driver.manage().window().maximize();

driver.findElement(By.xpath("//*
[contains(@href, 'popup.php')]")).click();

String MainWindow=driver.getWindowHandle();

// To handle all new opened window.
Set<String> s1=driver.getWindowHandles();
Iterator<String> i1=s1.iterator();

while(i1.hasNext())
{
    String ChildWindow=i1.next();

    if(!MainWindow.equalsIgnoreCase(ChildWindow))
    {

        // Switching to Child window
        driver.switchTo().window(ChildWindow);
        driver.findElement(By.name("emailid"))
            .sendKeys("gaurav.3n@gmail.com");
    }

    driver.findElement(By.name("btnLogin")).click();

        // Closing the Child Window.
        driver.close();
    }
}

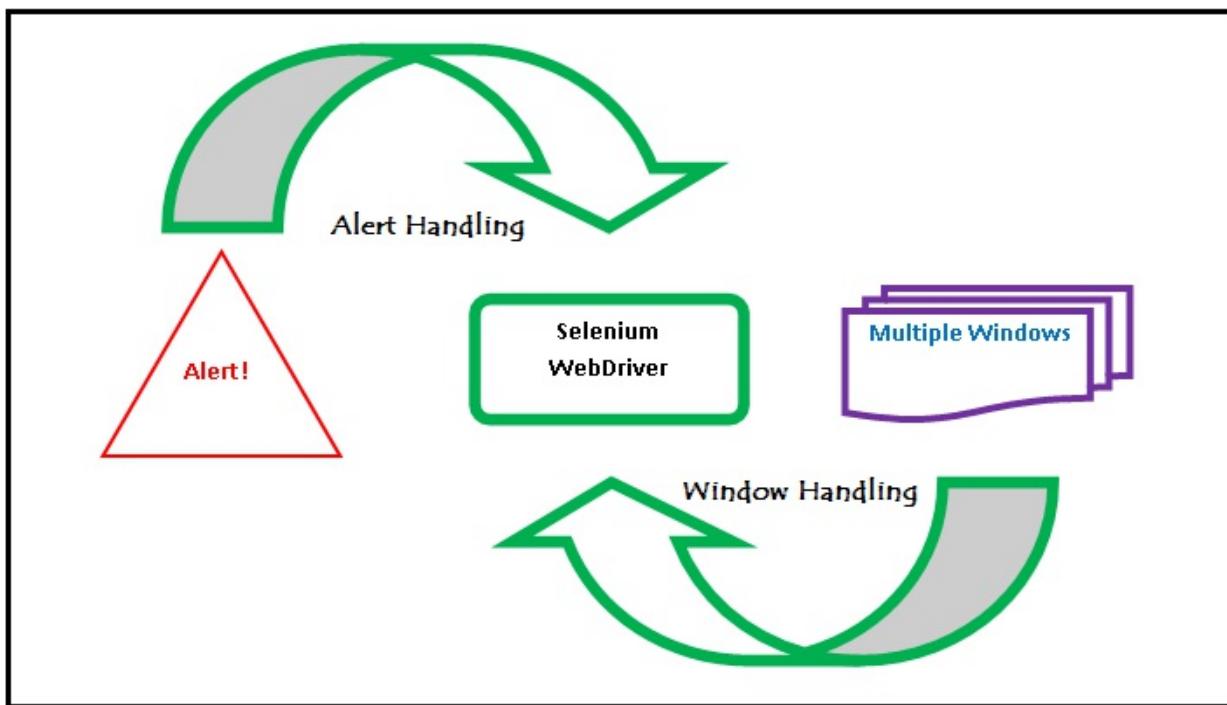
// Switching to Parent window i.e Main Window.
driver.switchTo().window(MainWindow);
}
}

```

Output:

When you execute the above code, it launches the site and on clicking the link "Click here," it opens up a child window in a new tab. You can close the child window, and switch to the parent window once the operation is completely done. Hence handling more than one window

in the application.



Conclusion:

- We defined the types of alert and shown them with a screen shot.
- Demonstrated handling the Alert with Selenium WebDriver using particular scenario.
- Handled multiple windows with Selenium WebDriver using particular scenario.

Chapter 20: Handling Dynamic Web Tables Using Selenium WebDriver

There are two types of HTML tables published on the web-

1. **Static tables:** Data is static i.e. Number of rows and columns are fixed.
2. **Dynamic tables:** Data is dynamic i.e. Number of rows and columns are NOT fixed.

Below is an example of a dynamic table of Sales. Based on input date filters, number of rows will get altered. So, it is dynamic in nature.

Here we have a Dynamic Sales Table whose rows will get changed based on input

	Order #	Purchased On	Bill to Name	Ship to Name
Any ▾		From: <input type="text" value="04/2/2011"/> To: <input type="text" value="04/22/20"/>		
<input type="checkbox"/>	100004183	Apr 21, 2017 5:32:33 PM	BERRY BERRYTEN	BERRY BERRYTEN
<input type="checkbox"/>	100004182	Apr 21, 2017 3:16:39 PM	abc def	abc def
<input type="checkbox"/>	100004181	Apr 21, 2017 3:14:17 PM	abc def	abc def
<input type="checkbox"/>	100004180	Apr 21, 2017 3:04:27 PM	abc def	abc def
<input type="checkbox"/>	100004179	Apr 21, 2017 6:00:08 AM	SUBHAM BISWAS	SUBHAM BISWAS
<input type="checkbox"/>	100004178	Apr 21, 2017 5:55:25 AM	SUBHAM BISWAS	SUBHAM BISWAS
<input type="checkbox"/>	100004177	Apr 21, 2017 5:54:03 PM	Phuong Phuong	Phuong Phuong

Handling static table is easy, but dynamic table is a little bit difficult as rows and columns are not constant.

Using X-Path to Locate Web Table

Elements

Before we locate web element, first let's understand-

What is a web element?

Web elements are nothing but HTML elements like textbox, dropdowns radio buttons, submit buttons, etc. These HTML elements are written with **start** tag and ends with an **end** tag.

For Example,

`<p> My First HTML Document</p>.`

Steps for getting X-path of web element that we want to locate.

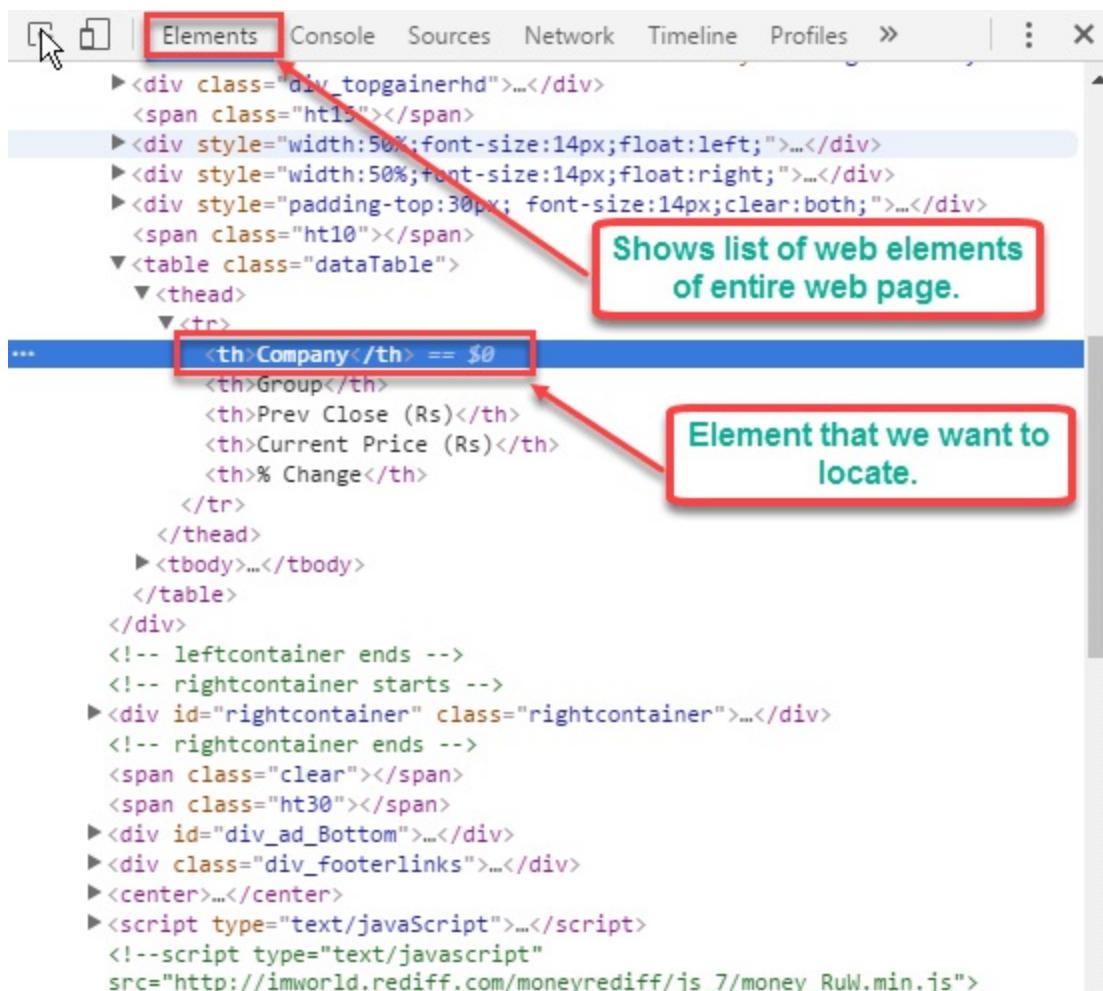
Step 1) In Chrome, Go to

<http://money.rediff.com/gainers/bsc/daily/groupa>

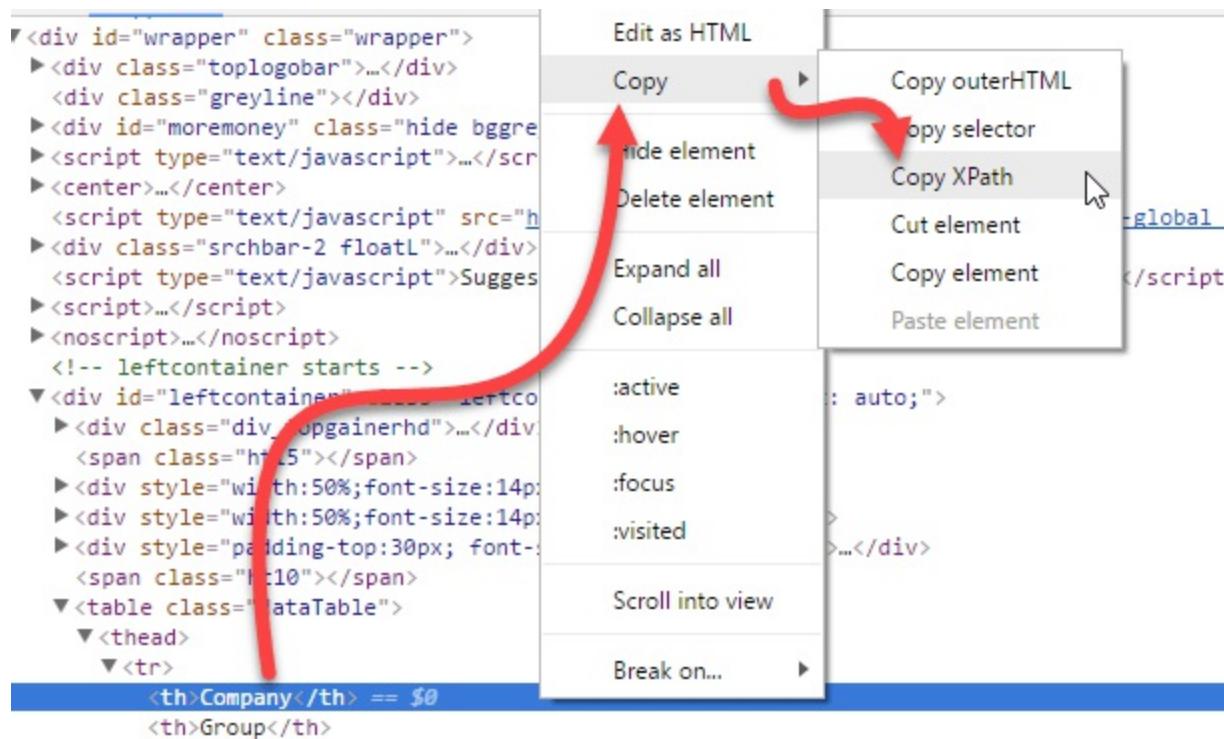
All	Group A	Group B	Group M	Group T	Group MT	Group Z
	Company	Group	Prev Close (Rs)	Current Price (Rs)	% Change	
Den Networks Ltd.	A	72.55	77.60	+ 6.96		
United Breweries Ltd	A	821.10	874.35	+ 6.49		
Balrampur Chini	A	102.40	108.50	+ 5.96		
Sintex Industrie	A	78.65	82.90	+ 5.40		
Thermax	A	822.90	865.00	+ 5.12		

Step 2) Right click on web element whose x-path is to be fetched. In our case, right click on "Company" Select Inspect option. The

following screen will be shown -



Step 3) Right Click on highlighted web element > Select Copy -> Copy x-path option.



Step 4) Use the copied X-path "//*
[@id='leftcontainer']/table/thead/tr/th [1]" in Selenium WebDriver to locate the element.

Example: Fetch number of rows and columns from Dynamic WebTable

When the table is dynamic in nature, we cannot predict its number of rows and columns.

Using Selenium web driver, we can find

- Number of Rows and columns of web table
- X row or Y column's data.

Below is program for fetching total number of rows and columns of web table.

```

import java.text.ParseException;
import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
public class Noofrowsandcols {

    public static void main(String[] args) throws ParseException {
        WebDriver wd;
        System.setProperty("webdriver.chrome.driver", "D:\\Guru 99 Tech writing\\Guru99 Examples\\chromedriver.exe");
        wd = new ChromeDriver();
        wd.get("http://money.rediff.com/gainers/bsc/dailygroupa?");

        //No.of Columns
        List <WebElement> col = wd.findElements(By.xpath(".//*[@id='leftcontainer']/table/thead/tr/th"));
        System.out.println("No of cols are : " + col.size());

        //No.of rows
        List <WebElement> rows = wd.findElements(By.xpath(".//*[@id='leftcontainer']/table/tbody/tr/td[1]"));
        System.out.println("No of rows are : " + rows.size());
        wd.close();
    }
}

```

Using x-path to locate rows and columns.

```

import java.text.ParseException;
import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class Noofrowsandcols {
    public static void main(String[] args) throws ParseException {
        WebDriver wd;

        System.setProperty("webdriver.chrome.driver", "G://chromedriver.exe");
        wd= new ChromeDriver();

        wd.get("http://money.rediff.com/gainers/bsc/dailygroupa?");
        //No.of Columns
        List col = wd.findElements(By.xpath(".//*[@id='leftcontainer']/table/thead/tr/th"));
        System.out.println("No of cols are : " + col.size());
        //No.of rows
        List rows = wd.findElements(By.xpath(".//*[@id='leftcontainer']/table/tbody/tr/td[1]"));
        System.out.println("No of rows are : " + rows.size());
        wd.close();
    }
}

```

```

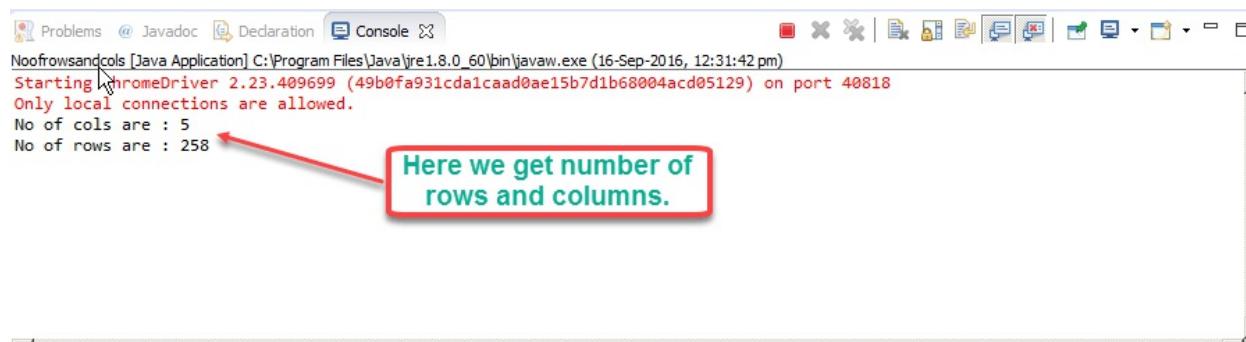
    }
}

```

Code Explanation:

- Here we have first declared Web Driver object "wd" & initialized it to chrome driver.
 - We use List <WebElement> to total number of columns in "col".
 - findElements command returns a list of ALL the elements matching the specified locator
 - using findElements and X-path //*[@id=\\\"leftcontainer\\\"]/table/thead/tr/th we get all the columns
 - Similarly, we repeat the process for rows.
- .

Output:



```

Problems @ Javadoc Declaration Console
Noofrowsandcols [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (16-Sep-2016, 12:31:42 pm)
Starting ChromeDriver 2.23.409699 (49b0fa931cda1caad0ae15b7d1b68004acd05129) on port 40818
Only local connections are allowed.
No of cols are : 5
No of rows are : 258

```

Here we get number of rows and columns.

Example: Fetch cell value of a particular row and column of the Dynamic Table

Let's assume we need 3rd row of the table and its second cell's data.
See the table below-

Top Gainers BSE | NSE India **Table update details** Last updated: 16 Sep, 12:37

Gainers		Losers		
Daily	Weekly	Monthly	Daily	Weekly
Table's 3rd row and 2nd cell's data				
All	Group A	Group B	Group M	Group T
Balrampur Chini	A	102.40	111.85	+ 9.23
Den Networks Ltd.	A	72.55	77.70	+ 7.10
United Breweries Ltd	A	821.10	873.60	+ 6.39
Thermax	A	822.90	866.50	+ 5.30
Reliance Infrastruct	A	582.20	610.00	+ 4.77
Sintex Industrie	A	78.65	82.35	+ 4.70
Jammu & Kashmir Bank	A	78.20	81.60	+ 4.35
Shree Renuka Sugars	A	15.80	16.45	+ 4.11
M&M Fin.Services	A	338.90	350.55	+ 3.44
Page Industries Ltd.	A	14,588.35	15,060.60	+ 3.24
South Indian Ban	A	23.35	24.10	+ 3.21
Kajaria Ceramics	A	1,332.35	1,374.50	+ 3.16
Anvind Ltd	A	215.65	225.40	+ 3.10

In above table, data is regularly updated after some span of time. The data you try retrieve will be different from the above screenshot. However, the code remains the same. Here is sample program to get the 3rd row and 2nd column's data.

```

import java.util.concurrent.TimeUnit;
public class Randomrowandcell {
    public static void main(String[] args)
{
    WebDriver wd;
    System.setProperty("webdriver.chrome.driver", "D:\\Guru 99 Tech writing\\Guru99 Examples\\chromedriver.exe");
    wd = new ChromeDriver();
    wd.get("http://money.rediff.com/gainers/bsc/daily/groupa?");
    wd.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
    WebElement baseTable = wd.findElement(By.tagName("table"));
}

//To find third row of table
WebElement tableRow = baseTable.findElement(By.xpath("//*[@id='leftcontainer']/table/tbody/tr[3]"));

WebElement tableRowText = baseTable.findElement(By.xpath("//*[@id='leftcontainer']/table/tbody/tr[3]/td[1]"));

String rowtext = tableRow.getText();
System.out.println("Third row of table : "+rowtext);

//To get 3rd row's 2nd column data
WebElement cellIneed = tableRow.findElement(By.xpath("//*[@id='leftcontainer']/table/tbody/tr[3]/td[2]"));
String valueIneed = cellIneed.getText();
System.out.println("Cell value is : "+valueIneed);
wd.close();
} //*[@id='leftcontainer']/table/tbody/tr[3]
}

```

To find table's 3rd row and 2nd cell's data we have used here xpath to locate it

```

import java.text.ParseException;
import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import java.util.concurrent.TimeUnit;

public class RowandCell {
    public static void main(String[] args) throws ParseException
{
    WebDriver wd;

System.setProperty("webdriver.chrome.driver","G://chromedriver.exe");
        wd= new ChromeDriver();

wd.get("http://money.rediff.com/gainers/bsc/daily/groupa?");
        wd.manage().timeouts().implicitlyWait(20,
TimeUnit.SECONDS);
        WebElement baseTable =
wd.findElement(By.tagName("table"));

//To find third row of table
WebElement tableRow =

```

```

baseTable.findElement(By.xpath("//*
[@id=\"leftcontainer\"]/table/tbody/tr[3]"));
    String rowtext = tableRow.getText();
    System.out.println("Third row of table :
"+rowtext);

    //to get 3rd row's 2nd column data
    WebElement cellIneed =
tableRow.findElement(By.xpath("//*
[@id=\"leftcontainer\"]/table/tbody/tr[3]/td[2]"));
    String valueIneed = cellIneed.getText();
    System.out.println("Cell value is : " +
valueIneed);
    wd.close();
}
}

```

Code Explanation:

- Table is located using locator property "tagname".
- Using XPath "//*[@id=\\"leftcontainer\\"]/table/tbody/tr[3]" find the 3rd row and gets its text using getText () function
- Using Xpath "//*
[@id=\\"leftcontainer\\"]/table/tbody/tr[3]/td[2]" find the 2nd cell in 3rd row and gets its text using getText () function

Output:



Example: Get Maximum of all the

Values in a Column of Dynamic Table

In this example, we will get the maximum of all values in a particular column.

Refer the following table -

Top Gainers		BSE NSE <small>new</small>			Last updated: 16 Sep, 15:59		
Gainers		Losers					
		Daily	Weekly	Monthly	Daily	Weekly	Monthly
All Group A Group B Group M Group T Group MT Group Z							
Company	Group	Prev Close (Rs)	Current Price (Rs)	% Change			
Wockhardt Ltd.	A	839.50	909.00	+ 8.28			
Thermax	A	822.90	889.75	+ 8.12			
Prestige Estates Pro	A	183.80	198.40	+ 7.94			
Den Networks Ltd.	A	72.55	78.20	+ 7.79			
GE T&D India	A	324.80	345.30	+ 6.31			
Repco Home Finance L	A	831.90	881.50	+ 5.96			
Firstsource Solution	A	41.65	43.95	+ 5.52			
Jammu & Kashmir Bank	A	78.20	82.35	+ 5.31			
TVS Motor Co. Ltd.	A	321.10	338.10	+ 5.29			
Arvind Ltd.	A	315.55	331.25	+ 4.98			
Balrampur Chini	A	102.40	107.05	+ 4.54			
Sintex Industrie	A	78.65	82.15	+ 4.45			
Alembic Pharmaceutic	A	628.80	655.00	+ 4.17			
Edelweiss Fin. Ser	A	106.90	111.20	+ 4.02			
Mangalore Refine	A	82.80	85.85	+ 3.68			

Here is the code

```

import java.text.NumberFormat;
public class maxfromtable {
    public static void main(String[] args) throws ParseException {
        String max;
        double m=0,r=0;
        WebDriver wd;
        System.setProperty("webdriver.chrome.driver", "D:\\Guru 99 Tech writing\\Guru99 Examples\\chromedriver.exe");
        wd = new ChromeDriver();
        wd.get("http://money.rediff.com/gainers/bsc/daily/groupa?");
        //No.of Columns
        List <WebElement> col = wd.findElements(By.xpath("//*[@id='leftcontainer']/table/thead/tr/th"));
        System.out.println("Total No of columns are : " + col.size());
        //No.of rows
        List <WebElement> rows = wd.findElements(By.xpath("//*[@id='leftcontainer']/table/tbody/tr/td[1]"));
        System.out.println("Total No of rows are : " + rows.size());
        for(int i =1;i<rows.size();i++)
        {
            max= wd.findElement(By.xpath("html/body/div[1]/div[5]/table/tbody/tr[" + (i+1) + "]/td[4]")).getText();
            NumberFormat f =NumberFormat.getNumberInstance();
            Number num = f.parse(max);
            max = num.toString();
            m = Double.parseDouble(max);
            if(m>r)
            {
                r= m;
            }
        }
        System.out.println("Maximum Current price is : " + r);
        wd.close();
    }
}

```

Here we have to change x-path to get next row. This will scan all rows to find maximum from current price.

```

import java.text.ParseException;
import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import java.text.NumberFormat;

public class MaxFromTable {
    public static void main(String[] args) throws ParseException
    {
        WebDriver wd;

        System.setProperty("webdriver.chrome.driver", "G://chromedriver.exe");
        wd= new ChromeDriver();

        wd.get("http://money.rediff.com/gainers/bsc/daily/groupa?");
        String max;
        double m=0, r=0;
    }
}

```

```

        //No. of Columns
        List col = wd.findElements(By.xpath(".//*
[@id='leftcontainer']/table/thead/tr/th"));
        System.out.println("Total No of columns are : " +
col.size());
        //No.of rows
        List rows = wd.findElements(By.xpath (" .//*
[@id='leftcontainer']/table/tbody/tr/td[1]"));
        System.out.println("Total No of rows are : " +
rows.size());
        for (int i =1;i<rows.size();i++)
        {
            max=
wd.findElement(By.xpath("html/body/div[1]/div[5]/table/tbody/tr[
" + (i+1)+ "]/td[4]").getText();
            NumberFormat f
=NumberFormat.getNumberInstance();
            Number num = f.parse(max);
            max = num.toString();
            m = Double.parseDouble(max);
            if(m>r)
            {
                r=m;
            }
        }
        System.out.println("Maximum current price is :
"+ r);
    }
}

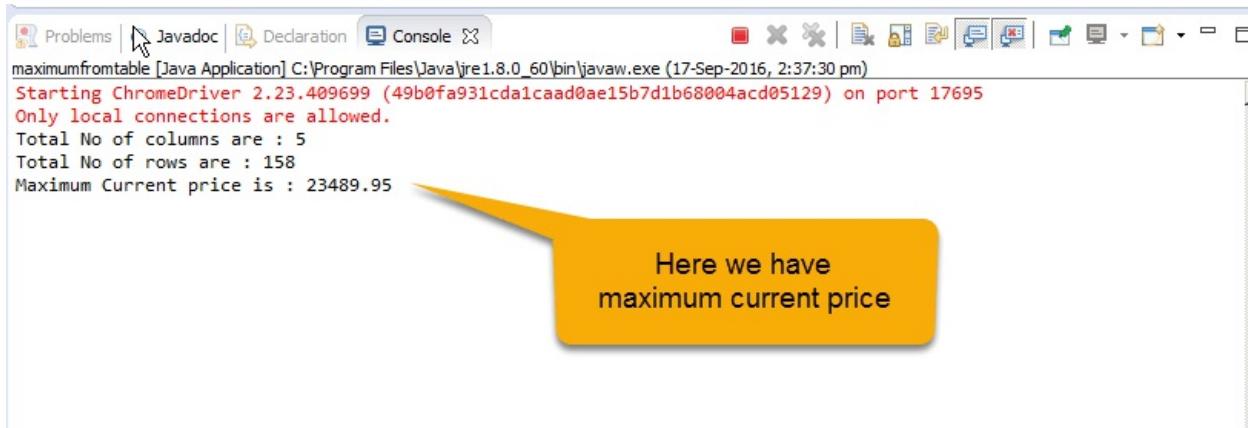
```

Code Explanation:

- Using chrome driver we locate the web table and get total number of row using XPath ".//*
[@id='leftcontainer']/table/tbody/tr/td[1]"
- Using for loop, we iterate through total number of rows and fetch values one by one. To get next row we use (i+1) in XPath
- We compare old value with new value and maximum value is

printed at the end of for loop

OutPut



```
maximumfromtable [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (17-Sep-2016, 2:37:30 pm)
Starting ChromeDriver 2.23.409699 (49b0fa931cda1caad0ae15b7d1b68004acd05129) on port 17695
Only local connections are allowed.
Total No of columns are : 5
Total No of rows are : 158
Maximum Current price is : 23489.95
```

Here we have
maximum current price

Example: Get all the values of a Dynamic Table

Consider the following table <http://demo.guru99.com/test/table.html>

1	2	3
4	5	6
5	6	
6	7	8
7		

The number of columns for each row is different.

Here row number 1, 2 and 4 have 3 cells, and row number 3 has 2 cells, and row number 5 has 1 cell.

We need to get values of all the cells

Here is the code:

```

import java.util.List;
public class dynamic_table_data_extract {
    public static void main(String args[])
    {
        WebDriver wd;
        System.setProperty("webdriver.chrome.driver", "D:\\Guru 99 Tech writing\\Guru99 Examples\\chromedriver.exe");
        wd = new ChromeDriver();
        wd.manage().window().maximize();
        wd.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
        wd.get("file:///D:/Guru%2099%20Tech%20writing/Guru99%20Examples/Guru99%20Samples/src/tableexample.html");
        //To locate table.
        WebElement mytable = wd.findElement(By.xpath("//html/body/table/tbody"));
        //To locate rows of table.
        List<WebElement> rows_table = mytable.findElements(By.tagName("tr"));
        //To calculate no of rows In table.
        int rows_count = rows_table.size();
        //Loop will execute till the last row of table.
        for (int row=0; row<rows_count; row++){
            //To locate columns(cells) of that specific row.
            List<WebElement> Columns_row = rows_table.get(row).findElements(By.tagName("td"));
            //To calculate no of columns(cells) In that specific row.
            int columns_count = Columns_row.size();
            System.out.println("Number of cells In Row "+row+" are "+columns_count);
            //Loop will execute till the last cell of that specific row.
            for (int column=0; column<columns_count; column++){
                //To retrieve text from that specific cell.
                String celtext = Columns_row.get(column).getText();
                System.out.println("Cell Value Of row number "+row+" and column number "+column+" Is "+celtext);
            }
            System.out.println("-----");
        }
    }
}

```

Give proper path of HTML file location saved on your system. It's for dynamic web table structure.

To locate table get table's x-path.

Loop is used to get last cell of specific row

```

import java.text.ParseException;
import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.chrome.ChromeDriver;

public class NofRowsColumns {
    public static void main(String[] args) throws ParseException
    {
        WebDriver wd;

        System.setProperty("webdriver.chrome.driver", "G://chromedriver.exe");
        wd = new ChromeDriver();
        wd.manage().timeouts().implicitlyWait(5,

```

```

TimeUnit.SECONDS);
        wd.get("http://demo.guru99.com/test/table.html");
        //To locate table.
        WebElement mytable =
wd.findElement(By.xpath("//html/body/table/tbody"));
        //To locate rows of table.
        List < WebElement > rows_table =
mytable.findElements(By.tagName("tr"));
        //To calculate no of rows In table.
        int rows_count = rows_table.size();
        //Loop will execute till the last row of table.
        for (int row = 0; row < rows_count; row++) {
            //To locate columns(cells) of that specific row.
            List < WebElement > Columns_row =
rows_table.get(row).findElements(By.tagName("td"));
            //To calculate no of columns (cells). In that
specific row.
            int columns_count = Columns_row.size();
            System.out.println("Number of cells In Row " + row +
" are " + columns_count);
            //Loop will execute till the last cell of that
specific row.
            for (int column = 0; column < columns_count;
column++) {
                // To retrieve text from that specific cell.
                String celtext =
Columns_row.get(column).getText();
                System.out.println("Cell Value of row number " +
row + " and column number " + column + " Is " + celtext);
            }
            System.out.println("-----");
        }
    }
}

```

Code Explanation:

- rows_count gives the total number of rows
- for each row we get the total number of columns using
rows_table.get(row).findElements(By.tagName("td"));

- We iterate through each column and of each row and fetch values.

Output:

```

Problems @ Javadoc Declaration Console
dynamic_table_data_extract [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (17-Sep-2016, 2:22:28 pm)
Starting ChromeDriver 2.23.409699 (49b0fa931cda1caad0ae15b7d1b68004acd05129) on port 10144
Only local connections are allowed.

Number of cells In Row 0 are 3
Cell Value Of row number 0 and column number 0 Is 1
Cell Value Of row number 0 and column number 1 Is 2
Cell Value Of row number 0 and column number 2 Is 3
-----
Number of cells In Row 1 are 3
Cell Value Of row number 1 and column number 0 Is 4
Cell Value Of row number 1 and column number 1 Is 5
Cell Value Of row number 1 and column number 2 Is 6
-----
Number of cells In Row 2 are 2
Cell Value Of row number 2 and column number 0 Is 5
Cell Value Of row number 2 and column number 1 Is 6
-----
Number of cells In Row 3 are 3
Cell Value Of row number 3 and column number 0 Is 6
Cell Value Of row number 3 and column number 1 Is 7
Cell Value Of row number 3 and column number 2 Is 8
-----
Number of cells In Row 4 are 1
Cell Value Of row number 4 and column number 0 Is 7
-----
```

Summary

- Static web tables are consistent in nature. i.e. they do have fixed number of rows as well as Cell data.
- Dynamic web tables are inconsistent i.e. they do not have fixed number of rows and cells data.
- Using selenium web driver, we can handle dynamic web tables easily.
- Selenium Webdriver allows us to access dynamic web tables by their X-path

Chapter 21: Using Contains, Sibling, Ancestor to Find Element in Selenium

If a simple XPath is not able to find a complicated web element for our test script, we need to use the functions from XPath 1.0 library. With the combination of these functions, we can create more specific XPath. Let's discuss 3 such functions –

1. Contains
2. Sibling
3. Ancestor

Let's study them in detail -

Contains: By using 'contains' function in XPath, we can extract all the elements which matches a particular text value.

Ex. Here we are searching an anchor .contains text as 'SAP M'.

```
//h4/a[contains(text(), 'SAP M')]"
```

'contains' is a function in xpath to get element which contains value similar to the given value

```
$x("//a[contains(text(), 'SAP MM'))")
[<a href="/sap-mm-training-tutorials.html" style="color:#343434;">SAP MM</a>]
```

Sibling: Using sibling keyword, we can fetch a web element on the basis of some other element.

Example: Here on the basis of sibling element of 'a' we are finding 'h4'

```
//div[@class='canvas-graph']//a[@href='/accounting.html'][i[@class='icon-usd']]//following-sibling::h4"
```

When we need to find element on the basis of some other control we can use following-sibling keyword of xpath.

Here in this example we are trying to get 'h4' on the basis of 'a'

```
<top frame>
$ //div[@class='canvas-graph']//a[@href='/accounting.html'][i[@class='icon-usd']]//following-sibling::h4"
[> h4...</h4>]
```

Ancestor: To find an element on the basis of the parent element we can use ancestor attribute of XPath.



Lets understand these 3 functions using an example –

Test Steps

Note: Since the date of creation of tutorial the Homepage of Guru99 has been updated so use the demo site instead to run tests

1. Go to <http://demo.guru99.com/test/guru99home/>
2. In the section 'A few of our most popular courses', search all Web Elements which are sibling of a WebElement whose text is 'SELENIUM'
3. We will find element using contains , ancestor and sibling function



USING Contains and Sibling

```
import java.util.List;
import java.util.concurrent.TimeUnit;
```

```
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 SiblingAndParentInXpath {

    @Test

    public void testSiblingAndParentInXpath(){

        WebDriver driver;
        String driverPath = "C:\\geckodriver.exe";
        System.setProperty("webdriver.firefox.marionette",
driverPath);
        driver = new FirefoxDriver();

        driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
        driver.get("http://demo.guru99.com/test/guru99home/");

        //Search element inside 'Popular course' which are
        sibling of control 'SELENIUM' ,Here first we will find a h2
        whose text is ''A few of our most popular courses' ,then we move
        to its parent element which is a 'div' , inside this div we will
        find a link whose text is 'SELENIUM' then at last we will find
        all of the sibling elements of this link('SELENIUM')
        List <WebElement> dateBox =
        driver.findElements(By.xpath("//h2[contains(text(),'A few of our
        most popular
        courses')]/parent::div//div[//a[text()='SELENIUM']]//following-
        sibling::div[@class='rt-grid-2 rt-omega']"));

        //Print all the which are sibling of the the element
        named as 'SELENIUM' in 'Popular course'
        for (WebElement webElement : dateBox) {
            System.out.println(webElement.getText());
        }

        driver.close();
    }
}
```

Output will be like:

```

Console > Results of running class DatepickerTest.java
<terminated> Datepicker [TestNG] C:\Program Files (x86)\Java\jre7\bin\
C:\Users\Babu\AppData\Local\Temp\testing-eclipse-10723\

Starting ChromeDriver (v2.6.2324) port 26084
JAVA
QTP
SAP Beginners
Linux
PASSSED: testSiblingAndP

=====
Default test
Tests run: 1, Failures: 0, Skips: 0
=====
```

Using Ancestor function

We can achieve the same functionality with the help of a function 'ancestor' as well.

Now suppose we need to Search All elements in 'Popular course' section with the help of ancestor of the anchor whose text is 'SELENIUM'

Here our xpath query will be like

```
//div[./a[text()='SELENIUM']]//ancestor::div[@class='rt-grid-2 rt-omega']/following-sibling::div
```

Complete Code

```

import java.util.List;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
```

```
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.Test;

public class AncestorInXpath{

    @Test

    public void testAncestorInXpath(){

        WebDriver driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
        driver.get("http://demo.guru99.com/test/guru99home/");

        //Search All elements in 'Popular course' section
        //with the help of ancestor of the anchor whose
text is 'SELENIUM'

        List <WebElement> dateBox =
driver.findElements(By.xpath("//div[./a[text()='SELENIUM']]//ancestor::div[@class='rt-grid-2 rt-omega']/following-
sibling::div"));

        //Print all the which are sibling of the element named
as 'SELENIUM' in 'Popular course'

        for (WebElement webElement : dateBox) {
            System.out.println(webElement.getText());
        }

        driver.quit();
    }
}
```

Output will look like-

```
Starting ChromeDriver (v2.6.232923) on port 4712
JAVA
QTP
SAP Beginners
Linux
PASSED: testAncestorInXpath
=====
Default test
Tests run: 1, Failures: 0, Skips: 0
=====
```

Summary:

- There are some situations when regular xpath cannot be used to find element. In these situations we need different functions from xpath query.
- There are some important xpath functions like contains, parent, ancestors, following-sibling etc.
- With the help of these functions we can create complex xpath expressions.

Chapter 22: Implicit & Explicit Waits in Selenium

In selenium "Waits" play an important role in executing tests. In this tutorial, you will learn various aspects of both "Implicit" and "Explicit" waits in Selenium.

Why Do We Need Waits In Selenium?

Most of the web applications are developed using Ajax and Javascript. When a page is loaded by the browser the elements which we want to interact with may load at different time intervals.

Not only it makes this difficult to identify the element but also if the element is not located it will throw an "**ElementNotVisibleException**" exception. Using Waits, we can resolve this problem.

Let's consider a scenario where we have to use both implicit and explicit waits in our test. Assume that implicit wait time is set to 20 seconds and explicit wait time is set to 10 seconds.

Suppose we are trying to find an element which has some "**ExpectedConditions**" (Explicit Wait), If the element is not located within the time frame defined by the Explicit wait(10 Seconds), It will use the time frame defined by implicit wait(20 seconds) before throwing an "**ElementNotVisibleException**".

Selenium Web Driver Waits

1. Implicit Wait
2. Explicit Wait

Implicit Wait

Selenium Web Driver has borrowed the idea of implicit waits from Watir.

The implicit wait will tell to the web driver to wait for certain amount of time before it throws a "No Such Element Exception". The default setting is 0. Once we set the time, web driver will wait for that time before throwing an exception.

In the below example we have declared an implicit wait with the time frame of 10 seconds. It means that if the element is not located on the web page within that time frame, it will throw an exception.

To declare implicit wait:

Syntax:

```
driver.manage().timeouts().implicitlyWait(TimeOut,  
TimeUnit.SECONDS);
```

```
package guru.test99;  
import java.util.concurrent.TimeUnit;  
import org.openqa.selenium.By;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.chrome.ChromeDriver;  
import org.testng.annotations.Test;  
public class AppTest {  
  
    protected WebDriver driver;
```

```
@Test
public void guru99tutorials() throws
InterruptedException
{
    System.setProperty
("webdriver.chrome.driver", ".\\chromedriver.exe" );
    driver = new ChromeDriver();

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS) ;
    String eTitle = "Demo Guru99 Page";
    String aTitle = "" ;
    // launch Chrome and redirect it to the Base URL
    driver.get("http://demo.guru99.com/test/guru99home/" );
    //Maximizes the browser window
    driver.manage().window().maximize() ;
    //get the actual value of the title
    aTitle = driver.getTitle();
    //compare the actual title with the expected title
    if (aTitle.equals(eTitle))
    {
        System.out.println( "Test Passed" ) ;
    }
    else {
        System.out.println( "Test Failed" );
    }
    //close browser
    driver.close();
}
}
```

Explanation of Code

In the above example,

Consider Following Code:

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

Implicit wait will accept 2 parameters, the first parameter will accept the time as an integer value and the second parameter will accept the

time measurement in terms of SECONDS, MINUTES, MILISECOND, MICROSECONDS, NANOSECONDS, DAYS, HOURS, etc.

Explicit Wait

The explicit wait is used to tell the Web Driver to wait for certain conditions (**Expected Conditions**) or the maximum time exceeded before throwing an "**ElementNotVisibleException**" exception.

The explicit wait is an intelligent kind of wait, but it can be applied only for specified elements. Explicit wait gives better options than that of an implicit wait as it will wait for dynamically loaded Ajax elements.

Once we declare explicit wait we have to use "**ExpectedConditions**" or we can configure how frequently we want to check the condition using **Fluent Wait**. These days while implementing we are using **Thread.Sleep()** generally it is not recommended to use

In the below example, we are creating reference wait for "**WebDriverWait**" class and instantiating using "**WebDriver**" reference, and we are giving a maximum time frame of 20 seconds.

Syntax:

```
WebDriverWait wait = new  
WebDriverWait(WebDriverReference,TimeOut);
```

```
package guru.test99;  
  
import java.util.concurrent.TimeUnit;  
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.support.ui.ExpectedConditions;
```

```
import org.openqa.selenium.support.ui.WebDriverWait;
import org.testng.annotations.Test;

public class AppTest2 {
    protected WebDriver driver;
    @Test
    public void guru99tutorials() throws
InterruptedException
    {
        System.setProperty
("webdriver.chrome.driver",".\\chromedriver.exe" );
        driver = new ChromeDriver();
        WebDriverWait wait=new WebDriverWait(driver, 20);
        String eTitle = "Demo Guru99 Page";
        String aTitle = "" ;
        // launch Chrome and redirect it to the Base URL
        driver.get("http://demo.guru99.com/test/guru99home/" );
        //Maximizes the browser window
        driver.manage().window().maximize() ;
        //get the actual value of the title
        aTitle = driver.getTitle();
        //compare the actual title with the expected title
        if (aTitle.contentEquals(eTitle))
        {
            System.out.println( "Test Passed" ) ;
        }
        else {
            System.out.println( "Test Failed" );
        }
        WebElement guru99seleniumlink;
        guru99seleniumlink=
wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(
"/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/
div/div[2]/div[2]/div/div/div/div/div/div[1]/div/div/a/i")));
        guru99seleniumlink.click();
    }
}
```

Explanation of Code

Consider Following Code:

```
WebElement guru99seleniumlink;
guru99seleniumlink=
wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(
"/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/
div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i")));
guru99seleniumlink.click();
```

In the above example, wait for the amount of time defined in the "**WebDriverWait**" class or the "**ExpectedConditions**" to occur whichever occurs first.

The above Java code states that we are waiting for an element for the time frame of 20 seconds as defined in the "**WebDriverWait**" class on the webpage until the "**ExpectedConditions**" are met and the condition is "**visibilityofElementLocated**".

The following are the Expected Conditions that can be used in Explicit Wait

1. alertIsPresent()
2. elementSelectionStateToBe()
3. elementToBeClickable()
4. elementToBeSelected()
5. frameToBeAvailableAndSwitchToIt()
6. invisibilityOfTheElementLocated()
7. invisibilityOfElementWithText()
8. presenceOfAllElementsLocatedBy()
9. presenceOfElementLocated()
10. textToBePresentInElement()
11. textToBePresentInElementLocated()
12. textToBePresentInElementValue()

13. titleIs()
14. titleContains()
15. visibilityOf()
16. visibilityOfAllElements()
17. visibilityOfAllElementsLocatedBy()
18. visibilityOfElementLocated()

Fluent Wait

The fluent wait is used to tell the web driver to wait for a condition, as well as the **frequency** with which we want to check the condition before throwing an "ElementNotVisibleException" exception.

Frequency: Setting up a repeat cycle with the time frame to verify/check the condition at the regular interval of time

Let's consider a scenario where an element is loaded at different intervals of time. The element might load within 10 seconds, 20 seconds or even more than that if we declare an explicit wait of 20 seconds. It will wait till the specified time before throwing an exception. In such scenarios, the fluent wait is the ideal wait to use as this will try to find the element at different frequency until it finds it or the final timer runs out.

Syntax:

```
Wait wait = new FluentWait(WebDriver reference)
    .withTimeout(timeout, SECONDS)
    .pollingEvery(timeout, SECONDS)
    .ignoring(Exception.class);
```

```
package guru.test99;
```

```
import org.testng.annotations.Test;
import java.util.NoSuchElementException;
import java.util.concurrent.TimeUnit;
import java.util.function.Function;

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.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.FluentWait;
import org.openqa.selenium.support.ui.Wait;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.testng.annotations.Test;

public class AppTest3 {
    protected WebDriver driver;
    @Test
    public void guru99tutorials() throws
InterruptedException
    {
        System.setProperty
("webdriver.chrome.driver",".\\chromedriver.exe" );
        String eTitle = "Demo Guru99 Page";
        String aTitle = "" ;
        driver = new ChromeDriver();
        // launch Chrome and redirect it to the Base URL
        driver.get("http://demo.guru99.com/test/guru99home/" );
        //Maximizes the browser window
        driver.manage().window().maximize() ;
        //get the actual value of the title
        aTitle = driver.getTitle();
        //compare the actual title with the expected title
        if (aTitle.contentEquals(eTitle))
        {
            System.out.println( "Test Passed" ) ;
        }
        else {
            System.out.println( "Test Failed" );
        }

        Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
            .withTimeout(30, TimeUnit.SECONDS)
```

```

        .pollingEvery(5, TimeUnit.SECONDS)
        .ignoring(NoSuchElementException.class);
    WebElement clickseleniumlink = wait.until(new
Function<Webdriver, WebElement>(){

    public WebElement apply(WebDriver driver ) {
        return
driver.findElement(By.xpath("/html/body/div[1]/section/div[2]/di
v/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/di
v[1]/div/div/a/i"));
    }
});
//click on the selenium link
clickseleniumlink.click();
//close~ browser
driver.close() ;
}
}

```

Explanation of Code

Consider Following Code:

```

Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
    .withTimeout(30, TimeUnit.SECONDS)
    .pollingEvery(5, TimeUnit.SECONDS)
    .ignoring(NoSuchElementException.class);

```

In the above example, we are declaring a fluent wait with the timeout of 30 seconds and the frequency is set to 5 seconds by ignoring "**NoSuchElementException**"

Consider Following Code:

```

public WebElement apply(WebDriver driver ) {
    return
driver.findElement(By.xpath("/html/body/div[1]/section/div[2]/di
v/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/di
v[1]/div/div/a/i"));
}

```

We have created a new function to identify the Web Element on the page. (Ex: Here Web Element is nothing but the selenium link on the webpage).

Frequency is set to 5 seconds and the maximum time is set to 30 seconds. Thus this means that it will check for the element on the web page at every 5 seconds for the maximum time of 30 seconds. If the element is located within this time frame it will perform the operations else it will throw an "**ElementNotVisibleException**"

Difference between Implicit Wait Vs Explicit Wait

Implicit Wait	Explicit Wait
<ul style="list-style-type: none"> Implicit Wait time is applied to all the elements in the script 	<ul style="list-style-type: none"> Explicit Wait time is applied only to those elements which are intended by us
<ul style="list-style-type: none"> In Implicit Wait, we need not specify "ExpectedConditions" on the element to be located 	<ul style="list-style-type: none"> In Explicit Wait, we need to specify "ExpectedConditions" on the element to be located
<ul style="list-style-type: none"> It is recommended to use when the elements are located with the time frame specified in implicit wait 	<ul style="list-style-type: none"> It is recommended to use when the elements are taking long time to load and also for verifying the property of the element like(visibilityOfElementLocated, elementToBeClickable,elementToBeSelected)

Conclusion:

Implicit, Explicit and Fluent Wait are the different waits used in selenium. Usage of these waits are totally based on the elements which

are loaded at different intervals of time. It is always **not recommended** to use **Thread.Sleep()** while Testing our application or building our framework.

Chapter 23: Parameterization using XML and DataProviders: Selenium

As we create software, we always wish it should work differently with a different set of data. When it comes to Testing the same piece of software, we can't be unfair to test it with just one set of data. Here again, we need to verify that our system is taking all set of combinations which it expected to support.

Here comes Parameterization in the picture. To pass multiple data to the application at runtime, we need to parameterize our test scripts.

This concept which we achieve by parameterization is called **Data Driven Testing**.

Type of Parameterization in TestNG-

To make parameterization more clear, we will go through the parameterization options in one the most popular framework for Selenium Webdriver - **TestNG**.

There are **two ways** by which we can achieve parameterization in TestNG

1. With the help of **Parameters annotation** and **TestNG XML file**.

```
@Parameters({"name", "searchKey"})
```

2. With the help of **DataProvider** annotation.

```
@DataProvider(name="SearchProvider")
```



Parameters from Testng.xml can be suite or test level

Parameter from DataProvider can take Method and ITestContext as the parameter.

Let's study them in detail -

Parameters annotation with Testng.xml

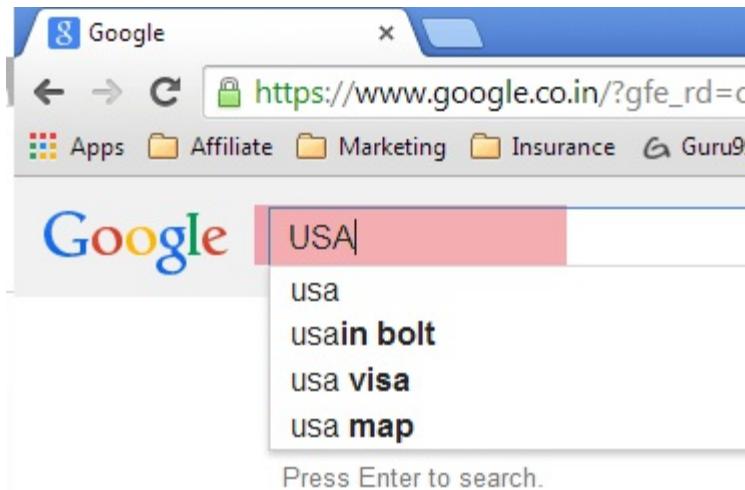
Select parameterization using annotations when you do want to deal with complexity & the number of input combinations are less.

Let see how this works

Test Scenario

Step 1) Launch browser & go to Google.com

Step 2) Enter a search keyword



Step 3) Verify the inputted value is same as that provided by our test data

Step 4) Repeat 2 & 3 until all values are inputted

Test Author	SearchKey
Guru99	India
Krishna	USA
Bhupesh	China

Here is an example of how to do it WITHOUT parameters

```
package parameters;

import org.testng.annotations.Test;
import org.testng.AssertJUnit;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
```

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

public class NoParameterWithTestNGXML {
    String driverPath = "C:\\\\geckodriver.exe";
    WebDriver driver;

    @Test
    public void testNoParameter() throws InterruptedException{
        String author = "guru99";
        String searchKey = "india";

        System.setProperty("webdriver.firefox.marionette",
driverPath);
        driver= new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);

        driver.get("https://google.com");
        WebElement searchText =
driver.findElement(By.name("q"));
        //Searching text in google text box
        searchText.sendKeys(searchKey);

        System.out.println("Welcome ->" +author+ " Your search key
is->" +searchKey);
        System.out.println("Thread will sleep now");

        Thread.sleep(3000);
        System.out.println("Value in Google Search Box =
"+searchText.getAttribute("value") +" :: Value given by input =
"+searchKey);
        //verifying the value in google search box

AssertJUnit.assertTrue(searchText.getAttribute("value").equalsIgnoreCase(searchKey));
}
}
```

A Study, the above example. Just imagine how complex the code will

become when we do this for 3 input combinations

Now, let's parameterize this using TestNG

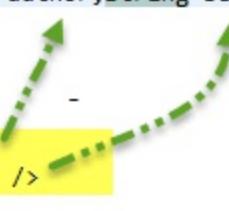
To do so, you will need to

- Create an XML file which will store the parameters
- In the test, add annotation @Parameters

```
@Test
@Parameters({"author","searchKey"})
public void testParameterWithXML(String author, String searchKey)

<suite name="TestSuite" thread-count="1" >
<parameter name="author" value="demo" />
<parameter name="searchKey" value="India" />
<test name="testGuru">
<classes>
<class name="parameters.ParameterWithTestNGXML">
</class>
</classes>
</test>
</suite>
```

Here 'author' parameter from xml will map with the 'author' parameter in test method



Here is the complete code

Test Level TestNG.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="TestSuite" thread-count="3" >
<parameter name="author" value="Guru99" />
<parameter name="searchKey" value="India" />
<test name="testGuru">
<parameter name="searchKey" value="UK" />
<classes>
<class name="parameters.ParameterWithTestNGXML">
</class>
```

```
</classes>
</test>
</suite>
```

ParameterWithTestNGXML.java File

```
package parameters;

import org.testng.AssertJUnit;
import java.util.concurrent.TimeUnit;

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.Optional;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class ParameterWithTestNGXML {
    String driverPath = "C:\\\\geckodriver.exe";
    WebDriver driver;
    @Test
    @Parameters({"author","searchKey"})
    public void testParameterWithXML( @Optional("Abc") String
author,String searchKey) throws InterruptedException{

        System.setProperty("webdriver.firefox.marionette",
driverPath);
        driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
        driver.get("https://google.com");

        WebElement searchText =
driver.findElement(By.name("q"));
        //Searching text in google text box
        searchText.sendKeys(searchKey);

        System.out.println("Welcome ->"+author+" Your search key
is->"+searchKey);
```

```

System.out.println("Thread will sleep now");
Thread.sleep(3000);
System.out.println("Value in Google Search Box =
"+searchText.getAttribute("value") +" :: Value given by input =
"+searchKey);
//verifying the value in google search box

AssertJUnit.assertTrue(searchText.getAttribute("value").equalsIgnoreCase(searchKey));

}
}

```

Instructions to run the script, select the XML file and Run as Test NG Suite

Right Click on .xml file -> Run as -> Testing Suite (Note : Suite)



Now, parameters can be defined at 2 levels

1. Suite level – The parameters inside the <suite> tag of TestNG XML file will be a suite level parameter.
2. Test Level -- The parameters inside the <Test> tag of testing XML file will be a Test level parameter.

Here is the same test with suite level parameters

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="TestSuite" thread-count="1" >
  <parameter name="author" value="demo" />
  <parameter name="searchKey" value="India" />
  <test name="testGuru">
    <classes>
      <class name="parameters.ParameterWithTestNGXML">
        </class>
      </classes>
    </test>
  </suite>

```

NOTE: In case if the parameter name is same in suite level and test level then test level parameter will get preference over suite level. So, in that case, all the classes inside that test level will share the overridden parameter, and other classes which are outside the test level will share suite level parameter.

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="TestTest" thread-count="1" >
  <parameter name="author" value="demo" />
  <parameter name="searchKey" value="India" />
  <test name="testGuru">
    <parameter name="searchKey" value="UK" />
    <classes>
      <class name="parameters.ParameterWithTestNGXML">
        </class>
      </classes>
    </test>
  </suite>

```

Same
parameter in
suite and test
level

Troubleshooting

Issue # 1 Parameter value in testng.xml cannot be typecasted to the corresponding test method's parameter it will throw an error.

Consider the following example

Here author is a string type but in test method , it is expecting a int. So We will get Number format exception.

```

<parameter name="author" value="Guru99" />
<parameter name="searchKey" value="India" />

@Test
@Parameters({"author", "searchKey"})
public void testParameterWithXML( int name, String searchKey) throws

```

Search: Passed: 0 Failed: 0 Skipped: 1

All Tests Failed Tests Summary

- LoginTestSuite (0/0/1/0) (0.016 s)
 - testGuru (0.016 s)
 - parameters.ParameterWithTestNGXML
 - testParameterWithXML (0.016 s)

Failure Exception

java.lang.NumberFormatException: For input string: "Guru99"

Here, 'author' attribute is equal to 'Guru99' which is a string and in corresponding test method its expecting an integer value, so we will get an exception here.

Issue # 2 Your @Parameters do not have a corresponding value in testing.xml.

You can solve this situation by adding **@optional annotation** in the corresponding parameter in the test method.

Optional parameter with default value for author as
'GURU99'

```

@Test
@Parameters({"author", "searchKey"})
public void testParameterWithXML( @Optional("Guru99") String author, String searchKey)

```

Issue # 3: You want to test multiple values of the same parameter using Testng.xml

The Simple answer is this can not be done! You can have multiple different parameters, but each parameter can only have a single value.

This helps prevent hardcoding values into the script. This makes code reusable. Think of it as config files for your script. If you want to use multiple values for a parameter use DataProviders

Parameters using Dataprovider

@Parameters annotation is easy but to test with multiple sets of data we need to use Data Provider.

To fill thousand's of web forms using our testing framework we need a different methodology which can give us a very large dataset in a single execution flow.

This data driven concept is achieved by **@DataProvider** annotation in TestNG.

'name' attribute of a dataprovider is optional



```
@DataProvider(name="SearchProvider")
public Object[][] getDataFromDataprovider(){
```

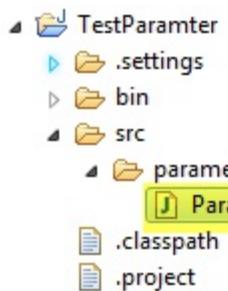
It has only one **attribute 'name'**. If you do not specify the name attribute then the DataProvider's name will be same as the corresponding method name.

Data provider returns **a two-dimensional JAVA object** to the test method and the test method, will invoke M times in a M*N type of object array. For example, if the DataProvider returns an array of 2*3 objects, the corresponding testcase will be invoked 2 times with 3 parameters each time.

return type of DataProvider is 2d array object

```
@DataProvider(name="SearchProvider")
public Object[][] getDataFromDataprovider(){
```

Complete Example



This is the final project structure

```
package parameters;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

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

public class ParameterByDataprovider {
    WebDriver driver;
    String driverPath = "C:\\geckodriver.exe";

    @BeforeTest
    public void setup(){
        //Create firefox driver object
        System.setProperty("webdriver.firefox.marionette",
driverPath);
```

```
        driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
        driver.get("https://google.com");
    }

/** Test case to verify google search box
 * @param author
 * @param searchKey
 * @throws InterruptedException
 */

@Test(dataProvider="SearchProvider")
public void testMethod(String author, String searchKey)
throws InterruptedException{
    {
        WebElement searchText =
driver.findElement(By.name("q"));
        //search value in google searchbox
        searchText.sendKeys(searchKey);
        System.out.println("Welcome ->" +author+ " Your search key
is->" +searchKey);
        Thread.sleep(3000);
        String testValue = searchText.getAttribute("value");
        System.out.println(testValue +"::::"+searchKey);
        searchText.clear();
        //Verify if the value in google search box is correct

Assert.assertTrue(testValue.equalsIgnoreCase(searchKey));
    }
}
/***
 * @return Object[][] where first column contains 'author'
 * and second column contains 'searchKey'
 */
@DataProvider(name="SearchProvider")
public Object[][] getDataFromDataprovider(){
return new Object[][]
{
    { "Guru99", "India" },
    { "Krishna", "UK" },
    { "Bhupesh", "USA" }
}
```

```

    };
}

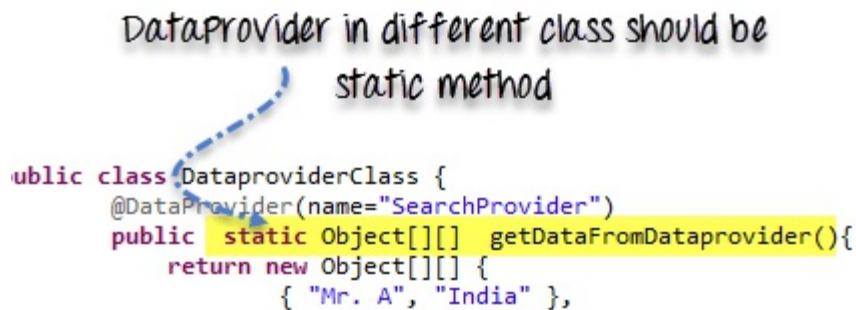
}

```

Invoke DataProvider from different class

By default, DataProvider resides in the same class where test method is or its base class. To put it in some other class we need to make data provider method as static and in test method we need to add an attribute **dataProviderClass** in **@Test** annotation.

DataProvider in different class should be
static method

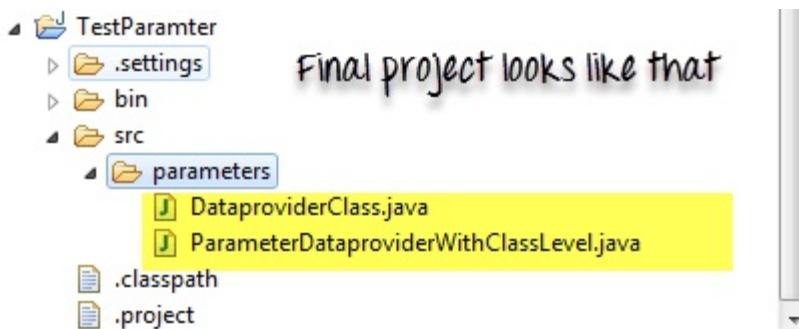


```

public class DataproviderClass {
    @DataProvider(name="SearchProvider")
    public static Object[][] getDataFromDataprovider(){
        return new Object[][] {
            {"Mr. A", "India"},

```

Code Example



TestClass ParameterDataproviderWithClassLevel.java

```

package parameters;

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;

```

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class ParameterDataproviderWithClassLevel {
    WebDriver driver;
    String driverPath = "C:\\geckodriver.exe";

    @BeforeTest
    public void setup(){

        System.setProperty("webdriver.firefox.marionette", driverPath);
        driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10,
        TimeUnit.SECONDS);
        driver.get("https://google.com");
    }

    @Test(dataProvider="SearchProvider",dataProviderClass=DataproviderClass.class)
    public void testMethod(String author,String searchKey)
    throws InterruptedException{

        WebElement searchText =
        driver.findElement(By.name("q"));
        //Search text in google text box
        searchText.sendKeys(searchKey);
        System.out.println("Welcome ->"+author+" Your search key
is->"+searchKey);
        Thread.sleep(3000);
        //get text from search box
        String testValue = searchText.getAttribute("value");
        System.out.println(testValue +"::::"+searchKey);
        searchText.clear();
        //verify if search box has correct value

        Assert.assertTrue(testValue.equalsIgnoreCase(searchKey));
    }
}
```

DataproviderClass.java

```
package parameters;

import org.testng.annotations.DataProvider;
public class DataproviderClass {
    @DataProvider(name="SearchProvider")
    public static Object[][] getDataFromDataprovider(){
        return new Object[][] {
            { "Guru99", "India" },
            { "Krishna", "UK" },
            { "Bhupesh", "USA" }
        };
    }
}
```

Types of Parameters in Dataprovider

There are two type of parameters supported by DataProvider method.

Method- If the **SAME** DataProvider should behave differently with different test method , use Method parameter.

```
@DataProvider(name="SearchProvider")
public Object[][] getDataFromDataprovider(Method m){
    if(m.getName().equalsIgnoreCase("testMethodA")){
        return new Object[][] {
            { "Guru99", "India" },
            { "Krishna", "UK" },
            { "Bhupesh", "USA" }
        };
    } else{
        return new Object[][] {
            { "Canada" },
            { "Russia" },
            { "Japan" }
        };
    }
}
```

Method parameter in data provider



In the following example ,

- We check if method name is testMethodA.

- If yes return one set of value
- Else return another set of value

```
package parameters;

import java.lang.reflect.Method;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class ParameterByMethodInDataprovider{

    WebDriver driver;
    String driverPath = "C:\\geckodriver.exe";

    @BeforeTest
    public void setup(){
        System.setProperty("webdriver.firefox.marionette",
driverPath);
        driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
        driver.get("https://google.com");
    }

    @Test(dataProvider="SearchProvider")
    public void testMethodA(String author, String searchKey)
throws InterruptedException{

    WebElement searchText =
driver.findElement(By.name("q"));
    //Search text in search box
    searchText.sendKeys(searchKey);
    //Print author and search string
    System.out.println("Welcome ->"+author+" Your search key
is->"+searchKey);
}
```

```

        Thread.sleep(3000);
        String testValue = searchText.getAttribute("value");
        System.out.println(testValue +"::::"+searchKey);
        searchText.clear();
        //Verify if google text box is showing correct value

Assert.assertTrue(testValue.equalsIgnoreCase(searchKey));
}

@Test(dataProvider="SearchProvider")
public void testMethodB(String searchKey) throws
InterruptedException{
{
    WebElement searchText =
driver.findElement(By.name("q"));
    //Search text in search box
    searchText.sendKeys(searchKey);
    //Print only search string
    System.out.println("Welcome ->Unknown user Your
search key is->" +searchKey);
    Thread.sleep(3000);
    String testValue = searchText.getAttribute("value");
    System.out.println(testValue +"::::"+searchKey);
    searchText.clear();
    //Verify if google text box is showing correct value

Assert.assertTrue(testValue.equalsIgnoreCase(searchKey));
}
}

/**
 * Here DataProvider returning value on the basis of test
method name
 * @param m
 * @return
 */
@DataProvider(name="SearchProvider")
public Object[][] getDataFromDataprovider(Method m){
    if(m.getName().equalsIgnoreCase("testMethodA")){
        return new Object[][] {
            { "Guru99", "India" },
            { "Krishna", "UK" },
            { "Bhupesh", "USA" }
        }
    }
}

```

```

        };
    else{
        return new Object[][] {
            { "Canada" },
            { "Russia" },
            { "Japan" }
        };
    }
}

```

Here is the output

```

Welcome ->Guru99 Your search key is->India
India::::India
Welcome ->Krishna Your search key is->UK
UK::::UK
Welcome ->Bhupesh Your search key is->USA
USA::::USA
Welcome ->Unknown user Your search key is->Canada
Canada::::Canada
Welcome ->Unknown user Your search key is->Russia
Russia::::Russia
Welcome ->Unknown user Your search key is->Japan
Japan::::Japan
PASSED: testMethodA("Guru99", "India")
PASSED: testMethodA("Krishna", "UK")
PASSED: testMethodA("Bhupesh", "USA")
PASSED: testMethodB("Canada")
PASSED: testMethodB("Russia")
PASSED: testMethodB("Japan")

=====
Default test
Tests run: 6, Failures: 0, Skips: 0
=====


```

Output

ITestContext- It can use to create different parameters for test cases based on groups.

In real-life, you can use ITestContext to vary parameter values based on Test Methods, hosts, configurations of the test.

```

@DataProvider(name="SearchProvider")
public Object[][] getDataFromDataprovider(ITestContext c){
Object[][] groupArray = null;
for (String group : c.getIncludedGroups()) {
if(group.equalsIgnoreCase("A")){
    groupArray = new Object[][] {
        { "Guru99", "India" },
        { "Krishna", "UK" },
        { "Bhupesh", "USA" }
    };
    break;
}
else if(group.equalsIgnoreCase("B")){
    groupArray = new Object[][] {
        { "Canada" },
        { "Russia" },
        { "Japan" }
    };
    break;
}
}
return groupArray;
}

```

ITestContext Parameter
in Data provider



In the following code example

- We have 2 groups A & B
- Each test method is assigned to a group
- If value of group is A, a particular data set is returned
- If value of group is B, another data set is returned

```

package parameters;

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.ITestContext;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class ParameterByITestContextInDataprovider {

```

```
WebDriver driver;
String driverPath = "C:\\geckodriver.exe";
@BeforeTest(groups={"A", "B"})
public void setup(){

System.setProperty("webdriver.firefox.marionette", driverPath);
driver = new FirefoxDriver();

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("https://google.com");
}

@Test(dataProvider="SearchProvider", groups="A")
public void testMethodA(String author, String searchKey)
throws InterruptedException{
{
//search google textbox
WebElement searchText =
driver.findElement(By.name("q"));
//search a value on it
searchText.sendKeys(searchKey);
System.out.println("Welcome ->" +author+
Your search key is->"+searchKey);
Thread.sleep(3000);
String testValue =
searchText.getAttribute("value");
System.out.println(testValue
+";:::" +searchKey);
searchText.clear();
//verify correct value in searchbox

Assert.assertTrue(testValue.equalsIgnoreCase(searchKey));
}
}

@Test(dataProvider="SearchProvider", groups="B")
public void testMethodB(String searchKey) throws
InterruptedException{
{
//find google search box
WebElement searchText =
driver.findElement(By.name("q"));
//search a value on it
```

```

        searchText.sendKeys(searchKey);
        System.out.println("Welcome ->Unknown
user Your search key is->" +searchKey);
        Thread.sleep(3000);
        String testValue =
searchText.getAttribute("value");
        System.out.println(testValue
+ ":::::" +searchKey);
        searchText.clear();
        //verify correct value in searchbox

Assert.assertTrue(testValue.equalsIgnoreCase(searchKey));
    }
}

/**
 * Here the DAtaProvider will provide Object array on
the basis on ITestContext
 * @param c
 * @return
 */
@DataProvider(name="SearchProvider")
public Object[][] getDataFromDataprovider(ITestContext
c){
    Object[][] groupArray = null;
    for (String group : c.getIncludedGroups()) {
        if(group.equalsIgnoreCase("A")){
            groupArray = new Object[][] {
                { "Guru99", "India" },
                { "Krishna", "UK" },
                { "Bhupesh", "USA" }
            };
            break;
        }
        else if(group.equalsIgnoreCase("B")){
            groupArray = new Object[][] {
                { "Canada" },
                { "Russia" },
                { "Japan" }
            };
            break;
        }
    }
}

```

```
    }
    return groupArray;
}
}
```

Summary:

- **Parameterization** is required to create **Data Driven Testing**.
- TestNG supports two kinds of parameterization,
using **@Parameter+TestNG.xml** and using **@DataProvider**
- In **@Parameter+TestNG.xml** parameters can be placed in
suite level and test level. If

The same parameter name is declared in both places; test level
parameter will get preference over suite level parameter.

- using **@Parameter+TestNG.xml** only one value can be set at a
time, but **@DataProvider** returns **an 2d array of Object**.
- If **DataProvider** is present in the different class then the class
where the test method resides, **DataProvider** should be **static
method**.
- There are two parameters supported
by **DataProvider** are **Method** and **ITestContext**.

Chapter 24: All About Excel in Selenium: POI & JXL

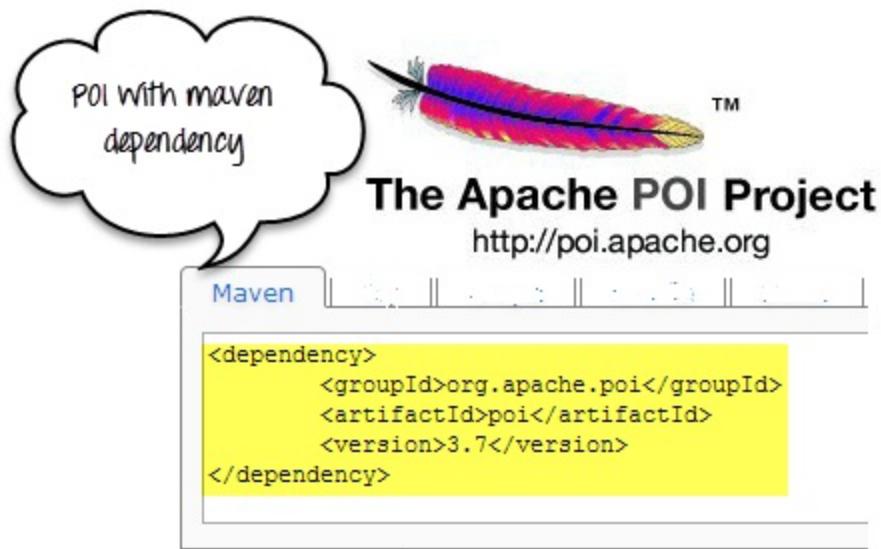
File IO is a critical part of any software process. We frequently create a file, open it & update something or delete it in our Computers. Same is the case with Selenium Automation. We need a process to manipulate files with Selenium.

Java provides us different classes for File Manipulation with Selenium. In this tutorial, we are going to learn how can we read and write on Excel file with the help of Java IO package and Apache POI library.



Exporting Excel

- **How to handle excel file using POI (Maven POM Dependency)**



To read or write an Excel, Apache provides a very famous library POI. This library is capable enough to read and write both **XLS** and **XLSX** file format of Excel.

To read **XLS** files, an **HSSF** implementation is provided by POI library.

To read **XLSX**, **XSSF** implementation of **POI library** will be the choice. Let's study these implementations in detail.

If you are using Maven in your project, the Maven dependency will be

```

<dependency>
<groupId>org.apache.poi</groupId>
<artifactId>poi</artifactId>
<version>3.9</version>
</dependency>

```

Or you can simply download the latest version POI jars from <http://poi.apache.org/download.html> & download the latest zip file



The Apache POI Project
<http://poi.apache.org>

Apache | POI

- Overview
 - Home
 - Download
 - Components
 - Text Extraction
 - Encryption support
 - Case Studies
 - Legal
- Help
 - Javadocs
 - FAQ
 - Mailing Lists
 - Bug Database
 - Changes Log
- Getting Involved
 - Subversion Repository
 - How To Build
 - Contribution Guidelines
 - Who We Are
- Components
 - Excel
 - Word
 - Powerpoint
 - OpenOffice
 - OLE2
 - OLE2 Data Sources (HPSF)
 - Outlook (HSMF)
 - Visio (HDGF)
 - TNEF (HMEF)
 - Publisher (HPBF)
- Apache Wide
 - Apache Software Foundation

Available Downloads

This page provides instructions on how to download and verify the Apache POI see [the project homepage](#).

- [The latest stable release is Apache POI 3.10-FINAL](#)
- [Archives of all prior releases](#)

Apache POI releases are available under the [Apache License, Version 2.0](#). See [the project homepage](#) for more information. To insure that you have downloaded the true release you should [verify the integrity](#) of the file.

8 February 2014 - POI 3.10-FINAL available

The Apache POI team is pleased to announce the release of 3.10-FINAL. This is the first release of the year 2014. A full list of changes is available in the [change log](#). People interested should also check out the [Apache POI mailing lists](#). The POI source release as well as the pre-built binary deployment packages are available at [http://poi.apache.org](#) under the package name "org.apache.poi" and Version "3.10-FINAL".

Binary Distribution

- [poi-bin-3.10-FINAL-20140208.tar.gz](#) (16MB, [signed](#))
 MD5 checksum: 818d1e99a2efe539ba49f622b554950c
 SHA1 checksum: 29b61005d780e00604fb9053fd46ab1b8b18392f
- [poi-bin-3.10-FINAL-20140208.zip](#) (23MB, [signed](#))
 MD5 checksum: c394bc9a31c9697d31-e929fc2150a962
 SHA1 checksum: 704f103bca893e3d4df5c2cc95d275b0cd5d0b33

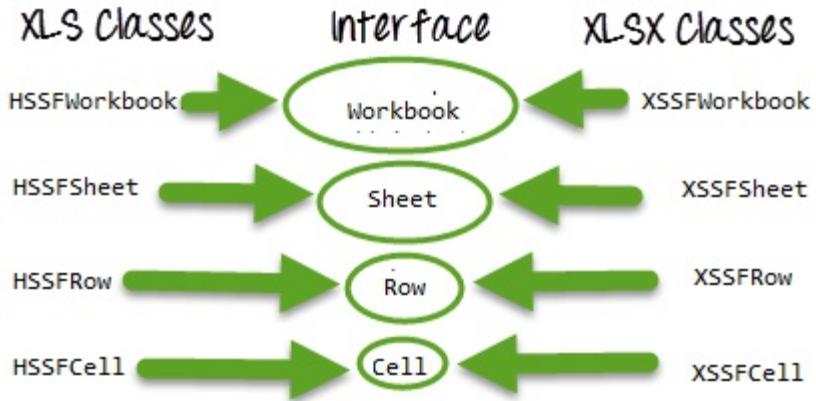
Download this

When you download the zip file for this jar, you need to unzip it and add these all jars to the class path of your project.

docs	6/30/2014 5:01 PM	File folder	
lib	6/30/2014 5:01 PM	File folder	
ooxml-lib	6/30/2014 5:00 PM	File folder	
LICENSE	1/16/2014 10:07 AM	File	27 KB
NOTICE	1/16/2014 10:07 AM	File	1 KB
poi-3.10-FINAL-20140208.jar	2/1/2014 7:30 PM	Executable Jar File	1,906 KB
poi-examples-3.10-FINAL-20140208.jar	2/1/2014 7:30 PM	Executable Jar File	306 KB
poi-excelant-3.10-FINAL-20140208.jar	2/1/2014 7:30 PM	Executable Jar File	30 KB
poi-ooxml-3.10-FINAL-20140208.jar	2/1/2014 7:30 PM	Executable Jar File	1,008 KB
poi-ooxml-schemas-3.10-FINAL-20140208.jar	2/1/2014 7:30 PM	Executable Jar File	4,831 KB
poi-scratchpad-3.10-FINAL-20140208.jar	2/1/2014 7:30 PM	Executable Jar File	1,212 KB
ooxml folder			
dom4j-1.6.1.jar	1/16/2014 10:12 AM	Executable Jar File	307 KB
stax-api-1.0.1.jar	1/16/2014 10:13 AM	Executable Jar File	26 KB
xmlbeans-2.3.0.jar	1/16/2014 10:13 AM	Executable Jar File	2,605 KB
lib folder			
commons-codec-1.5.jar	1/16/2014 10:12 AM	Executable Jar File	72 KB
commons-logging-1.1.jar	1/16/2014 10:12 AM	Executable Jar File	52 KB
junit-4.11.jar	1/16/2014 10:12 AM	Executable Jar File	240 KB
log4j-1.2.13.jar	1/16/2014 10:12 AM	Executable Jar File	350 KB

Add all these Files

Classes and Interfaces in POI:



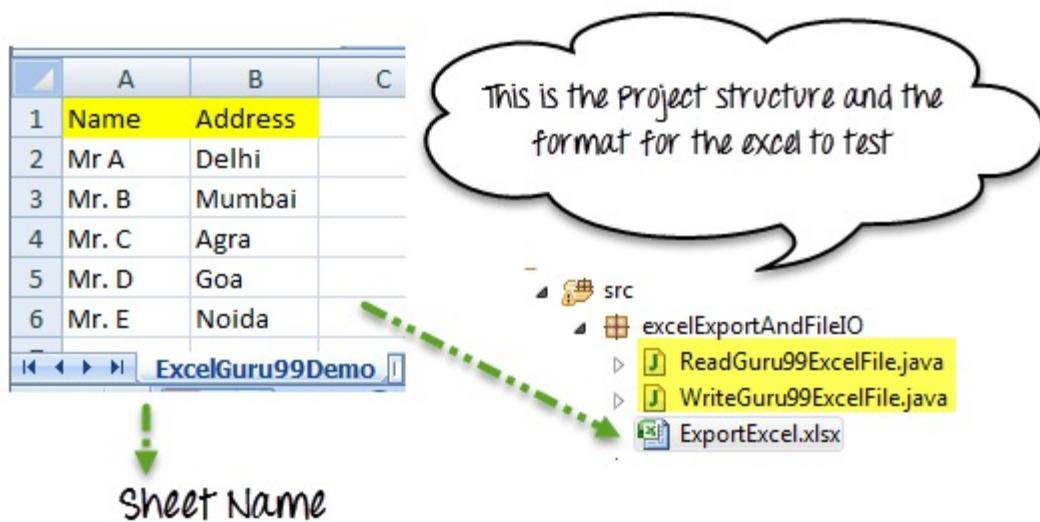
Following is a list of different Java Interfaces and classes in **POI** for reading **XLS** and **XLSX** file-

- **Workbook:** XSSFWorkbook and HSSFWorkbook classes implement this interface.

- **XSSFWorkbook:** Is a class representation of XLSX file.
- **HSSFWorkbook:** Is a class representation of XLS file.
- **Sheet:** XSSFSheet and HSSFSheet classes implement this interface.
- **XSSFSheet:** Is a class representing a sheet in an XLSX file.
- **HSSFSheet:** Is a class representing a sheet in an XLS file.
- **Row:** XSSFRow and HSSFRow classes implement this interface.
- **XSSFRow:** Is a class representing a row in the sheet of XLSX file.
- **HSSFRow:** Is a class representing a row in the sheet of XLS file.
- **Cell:** XSSFCell and HSSFCell classes implement this interface.
- **XSSFCell:** Is a class representing a cell in a row of XLSX file.
- **HSSFCell:** Is a class representing a cell in a row of XLS file.

Read/Write operation-

For our example, we will consider below given Excel file format



Read data from Excel file

Complete Example: Here we are trying to read data from Excel file

```
package excelExportAndFileIO;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import org.apache.poi.hssf.usermodel.HSSFWorkbook;

import org.apache.poi.ss.usermodel.Row;

import org.apache.poi.ss.usermodel.Sheet;

import org.apache.poi.ss.usermodel.Workbook;

import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class ReadGuru99ExcelFile {

    public void readExcel(String filePath, String fileName, String sheetName) throws IOException{
        //Create an object of File class to open xlsx file
        File file = new File(filePath+"\\"+fileName);
        //Create an object of FileInputStream class to read excel file
        FileInputStream inputStream = new FileInputStream(file);
        Workbook guru99Workbook = null;
        //Find the file extension by splitting file name in substring and getting only extension name
        String fileExtensionName =
```

```
fileName.substring(fileName.indexOf("."));

//Check condition if the file is xlsx file

if(fileExtensionName.equals(".xlsx")){
    //If it is xlsx file then create object of XSSFWorkbook
    class

        guru99Workbook = new XSSFWorkbook(inputStream);

    }

//Check condition if the file is xls file

else if(fileExtensionName.equals(".xls")){
    //If it is xls file then create object of XSSFWorkbook
    class

        guru99Workbook = new HSSFWorkbook(inputStream);

    }

//Read sheet inside the workbook by its name

Sheet guru99Sheet = guru99Workbook.getSheet(sheetName);

//Find number of rows in excel file

int rowCount = guru99Sheet.getLastRowNum()-
guru99Sheet.getFirstRowNum();

//Create a loop over all the rows of excel file to read it

for (int i = 0; i < rowCount+1; i++) {

    Row row = guru99Sheet.getRow(i);

    //Create a loop to print cell values in a row

    for (int j = 0; j < row.getLastCellNum(); j++) {
```

```
//Print Excel data in console

System.out.print(row.getCell(j).getStringCellValue()+"|| ");
}

System.out.println();

}

}

//Main function is calling readExcel function to read data
from excel file

public static void main(String...strings) throws
IOException{

//Create an object of ReadGuru99ExcelFile class

ReadGuru99ExcelFile objExcelFile = new
ReadGuru99ExcelFile();

//Prepare the path of excel file

String filePath =
System.getProperty("user.dir")+"\\src\\excelExportAndFileIO";

//Call read file method of the class to read data

objExcelFile.readExcel(filePath, "ExportExcel.xlsx", "ExcelGuru99D
emo");
}

}
```

Note: We are not using the Testng framework here. Run the class as Java Application

The screenshot shows an IDE interface with two tabs: 'ReadGuru99ExcelFile.java' and 'Console'. The Java code in the editor is as follows:

```
//It is XL  
guru99Workbook  
}  
//Check condit  
else if(fileE  
//If it is  
guru99Work
```

The 'Console' tab shows the output of the code, which reads data from an Excel file:

Name	Address
Mr A	Delhi
Mr. B	Mumbai
Mr. C	Agra
Mr. D	Goa

A blue arrow points from the text 'output of reading excel' to the console output.

Write data on Excel file

Complete Example: Here we are trying to write data from Excel file by adding new row in Excel file

```
package excelExportAndFileIO;  
  
import java.io.File;  
  
import java.io.FileInputStream;  
  
import java.io.FileOutputStream;  
  
import java.io.IOException;  
  
import org.apache.poi.hssf.usermodel.HSSFWorkbook;  
  
import org.apache.poi.ss.usermodel.Cell;  
  
import org.apache.poi.ss.usermodel.Row;  
  
import org.apache.poi.ss.usermodel.Sheet;  
  
import org.apache.poi.ss.usermodel.Workbook;  
  
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
```

```
public class WriteGuru99ExcelFile {  
  
    public void writeExcel(String filePath, String  
fileName, String sheetName, String[] dataToWrite) throws  
IOException{  
  
        //Create an object of File class to open xlsx file  
  
        File file = new File(filePath+"\\"+fileName);  
  
        //Create an object of FileInputStream class to read  
excel file  
  
        FileInputStream inputStream = new FileInputStream(file);  
  
        Workbook guru99Workbook = null;  
  
        //Find the file extension by splitting file name in  
substring and getting only extension name  
  
        String fileExtensionName =  
fileName.substring(fileName.indexOf("."));  
  
        //Check condition if the file is xlsx file  
  
        if(fileExtensionName.equals(".xlsx")){  
  
            //If it is xlsx file then create object of XSSFWorkbook  
class  
  
            guru99Workbook = new XSSFWorkbook(inputStream);  
  
        }  
  
        //Check condition if the file is xls file  
  
        else if(fileExtensionName.equals(".xls")){  
  
            //If it is xls file then create object of  
XSSFWorkbook class
```

```
        guru99Workbook = new HSSFWorkbook(inputStream);

    }

//Read excel sheet by sheet name

Sheet sheet = guru99Workbook.getSheet(sheetName);

//Get the current count of rows in excel file

int rowCount = sheet.getLastRowNum()-sheet.getFirstRowNum();

//Get the first row from the sheet

Row row = sheet.getRow(0);

//Create a new row and append it at last of sheet

Row newRow = sheet.createRow(rowCount+1);

//Create a loop over the cell of newly created Row

for(int j = 0; j < row.getLastCellNum(); j++){

    //Fill data in row

    Cell cell = newRow.createCell(j);

    cell.setCellValue(dataToWrite[j]);

}

//Close input stream

inputStream.close();

//Create an object of FileOutputStream class to create write data in excel file

FileOutputStream outputStream = new FileOutputStream(file);
```

```
//write data in the excel file  
  
guru99Workbook.write(outputStream);  
  
//close output stream  
  
outputStream.close();  
  
}  
  
  
  
  
public static void main(String...strings) throws  
IOException{  
  
    //Create an array with the data in the same order in  
    //which you expect to be filled in excel file  
  
    String[] valueToWrite = {"Mr. E", "Noida"};  
  
    //Create an object of current class  
  
    WriteGuru99ExcelFile objExcelFile = new  
    WriteGuru99ExcelFile();  
  
    //Write the file using file name, sheet name and the  
    //data to be filled  
  
    objExcelFile.writeExcel(System.getProperty("user.dir")+"\\src\\e  
    xcelExportAndFileIO", "ExportExcel.xlsx", "ExcelGuru99Demo", valueT  
    oWrite);  
  
}
```

} After Writing Excel File

```
public static void main(String...strings) throws
    //Create an array with the data in the same order
    String[] valueToWrite = {"Mr. E","Noida"};
    //Create an object of current class
    WriteGuru99ExcelFile objExcelFile = new WriteGuru99ExcelFile();
    //Write the file using file name , sheet name
    objExcelFile.writeExcel(System.getProperty("user.dir") + "ExcelGuru99Demo.xlsx", "Sheet1", valueToWrite);
}
```

	Name	Address
2	Mr A	Delhi
3	Mr. B	Mumbai
4	Mr. C	Agra
5	Mr. D	Goa
6	Mr. E	Noida

New Entry in excel file(name='Mr E' ,Address='Noida')

Excel Manipulation using JXL API

JXL A Java Excel API - A Java API to read, write, and modify Excel



JXL is also another famous jar for reading writing Excel files. Now a day's POI is used in most of the projects, but before POI, JXL was only Java API for Excel manipulation. It is a very small and simple API.

TIPS: *My suggestion is not to use JXL in any new project because the library is not in active development from 2010 and lack of the feature in compare to POI API.*

Download JXL:

If you want to work with JXL, you can download it from this link

<http://sourceforge.net/projects/jexcelapi/files/jexcelapi/2.6.12/>

Click on this link to download JXL

Name	Modified	Size	Downloads / Week
Parent folder			
2_6_12_releasenotes.txt	2009-10-26	606 Bytes	66  
jexcelapi_2_6_12.zip	2009-10-26	2.5 MB	1,030  
jexcelapi_2_6_12.tar.gz	2009-10-26	1.9 MB	70  
Totals: 3 Items		4.4 MB	1,166

You can also get demo example inside this zipped file for JXL.

Some of the features:

- JXL is able to read Excel 95, 97, 2000, XP, 2003 workbook.
- We can work with English, French, Spanish, German.
- Copying a Chart and image insertion in Excel is possible

Drawback:

- We can write Excel 97 and later only (writing in Excel 95 is not supported).
- JXL does not support XLSX format of excel file.
- It Generates spreadsheet in Excel 2000 format.

Summary:

- Excel file can be read by Java IO operation. For that, we need to use **Apache POI Jar**.
- There are two kinds of a workbook in Excel file, **XLSX** and **XLS** files.

- POI has different Interfaces Workbook, Sheet, Row, Cell.
- These interfaces are implemented by corresponding **XLS** (**HSSFWorkbook**, **HSSFSheet**, **HSSFRow**, **HSSFCell**) and **XLSX** (**XSSFWorkbook**, **XSSFSheet**, **XSSFRow**, **XSSFCell**) file manipulation classes.
- JXL is another API for Excel manipulation.
- JXL cannot work with XLSX format of excel.

Chapter 25: Page Object Model (POM) & Page Factory in Selenium: Ultimate Guide

Before we learn about Page Object Model, let's understand -

Why POM?

Starting an UI Automation in Selenium WebDriver is NOT a tough task. You just need to find elements, perform operations on it.

Consider this simple script to login into a website

```
public class NoPOMTest99GuruLogin {  
  
    /**  
     * This test case will login in http://demo.guru99.com/V4/  
     * Verify login page title as guru99 bank  
     * Login to application  
     * Verify the home page using Dashboard message  
     */  
    @Test(priority=0)  
    public void test_Home_Page_Appear_Correct(){  
        WebDriver driver = new FirefoxDriver();  
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);  
        driver.get("http://demo.guru99.com/V4/");  
        //Find user name and fill user name  
        driver.findElement(By.name("uid")).sendKeys("demo");  
        //find password and fill it  
        driver.findElement(By.name("password")).sendKeys("password");  
        //click login button  
        driver.findElement(By.name("btnLogin")).click();  
        String homeText = driver.findElement(By.xpath("//table//tr[@class='heading3']")).getText();  
        //verify login success  
        Assert.assertTrue(homeText.toLowerCase().contains("guru99 bank"));  
    }  
}
```

① Find user name and fill it

② Find password and fill it

③ Find Login button and click it

④ Find home page text and get it

⑤ Verify home page has text 'GURU99 Bank'

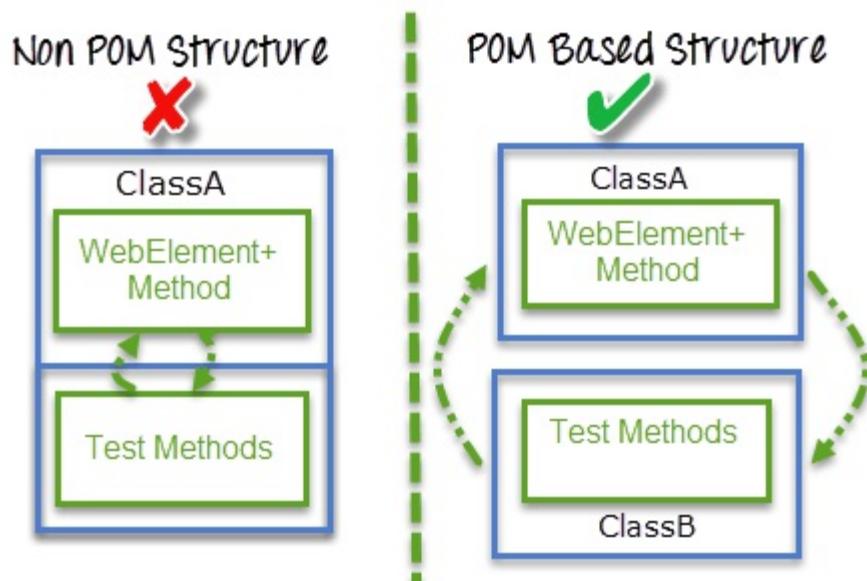
As you can observe, all we are doing is finding elements and filling values for those elements.

This is a small script. Script maintenance looks easy. But with time test suite will grow. As you add more and more lines to your code, things become tough.

The chief problem with script maintenance is that if 10 different scripts are using the same page element, with any change in that element, you need to change all 10 scripts. This is time consuming and error prone.

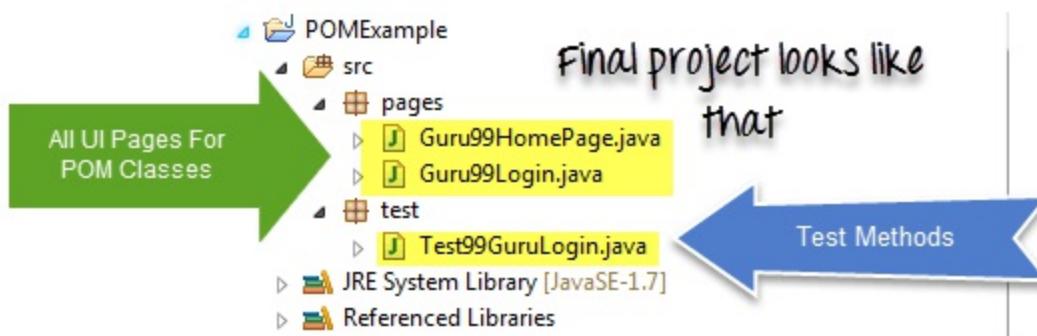
A better approach to script maintenance is to create a separate class file which would find web elements, fill them or verify them. This class can be reused in all the scripts using that element. In future, if there is a change in the web element, we need to make the change in just 1 class file and not 10 different scripts.

This approach is called **Page Object Model(POM)**. It helps make the code **more readable, maintainable, and reusable**.



What is POM?

- **Page Object Model** is a design pattern to create **Object Repository** for web UI elements.
- Under this model, for each web page in the application, there should be corresponding page class.
- This Page class will find the WebElements of that web page and also contains Page methods which perform operations on those WebElements.
- Name of these methods should be given as per the task they are performing, i.e., if a loader is waiting for the payment gateway to appear, POM method name can be `waitForPaymentScreenDisplay()`.



Advantages of POM

1. Page Object Pattern says operations and flows in the UI should be separated from verification. This concept makes our code cleaner and easy to understand.
2. The Second benefit is the **object repository is independent of test cases**, so we can use the same object repository for a different purpose with different tools. For example, we can

integrate POM with TestNG/JUnit for functional Testing and at the same time with JBehave/Cucumber for acceptance testing.

3. Code becomes less and optimized because of the reusable page methods in the POM classes.
4. **Methods get more realistic names** which can be easily mapped with the operation happening in UI. i.e. if after clicking on the button we land on the home page, the method name will be like 'gotoHomePage()'.

How to implement POM?

Simple POM:

It's the basic structure of Page object model (POM) where all Web Elements of the **AUT** and the method that operate on these Web Elements are maintained inside a class file. A task like **verification** should be **separate** as part of Test methods.

```
public class Guru99Login { I Page class in object repository
    WebDriver driver;
    By user99GuruName = By.name("uid");
    By password99Guru = By.name("password");
    By titleText = By.className("barone");
    By login = By.name("btnLogin");

    public Guru99Login(WebDriver driver){
        this.driver = driver;
    }
    //Set user name in textbox
    public void setUserName(String strUserName){
2        driver.findElement(user99GuruName).sendKeys(strUserName); 3
    }
}
```

Complete Example

TestCase: Go to Guru99 Demo Site.

Step 1) Go to Guru99 Demo Site

Goto Guru99 Demo site



Step 2) In home page check text "**Guru99 Bank**" is present

Verify if you found this title 'Guru99 Bank' in Login Page

Guru99 Bank

Step 3) Login into application

Enter user name

User ID

Password

LOGIN RESET

Click Login

Enter password

Step 4) Verify that the Home page contains text as "Manger Id: demo"

Welcome To Manager's Page of Guru99 Bank

Manger Id : demo

User Name should appear on Home Page

Here are we are dealing with 2 pages

1. Login Page
2. Home Page (shown once you login)

Accordingly, we create 2 POM classes

Guru99 Login page POM

```
package pages;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

public class Guru99Login {

    WebDriver driver;

    By user99GuruName = By.name("uid");

    By password99Guru = By.name("password");

    By titleText =By.className("barone");

    By login = By.name("btnLogin");

    public Guru99Login(WebDriver driver){

        this.driver = driver;

    }

    //Set user name in textbox

    public void setUserName(String strUserName){

driver.findElement(user99GuruName).sendKeys(strUserName);

    }

    //Set password in password textbox

    public void setPassword(String strPassword){
```

```
driver.findElement(password99Guru).sendKeys(strPassword);

}

//Click on login button

public void clickLogin(){

    driver.findElement(login).click();

}

//Get the title of Login Page

public String getLoginTitle(){

    return    driver.findElement(titleText).getText();

}

/** 

 * This POM method will be exposed in test case to login in
the application

 * @param strUserName

 * @param strPasword

 * @return

 */

public void loginToGuru99(String strUserName, String
strPasword){

    //Fill user name
```

```
    this.setUserName(strUserName);

    //Fill password

    this.setPassword(strPasword);

    //Click Login button

    this.clickLogin();

}

}
```

Guru99 Home Page POM

```
package pages;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

public class Guru99HomePage {

    WebDriver driver;

    By HomePageUserName =
By.xpath("//table//tr[@class='heading3']");

    public Guru99HomePage(WebDriver driver){

        this.driver = driver;

    }

}
```

```
//Get the User name from Home Page

    public String getHomePageDashboardUserName(){

        return
driver.findElement(homePageUserName).getText();

    }

}
```

Guru99 Simple POM Test case

```
package test;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.Assert;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

import pages.Guru99HomePage;

import pages.Guru99Login;

public class Test99GuruLogin {

    WebDriver driver;

    Guru99Login objLogin;

    Guru99HomePage objHomePage;

    @BeforeTest
```

```
public void setup(){

    driver = new FirefoxDriver();

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

    driver.get("http://demo.guru99.com/V4/");

}

/**

 * This test case will login in http://demo.guru99.com/V4/
 * Verify login page title as guru99 bank
 * Login to application
 * Verify the home page using Dashboard message
 */

@Test(priority=0)

public void test_Home_Page_Appear_Correct(){

    //Create Login Page object
    objLogin = new Guru99Login(driver);
    //Verify login page title
    String loginPageTitle = objLogin.getLoginTitle();

    Assert.assertTrue(loginPageTitle.toLowerCase().contains("guru99
bank"));

    //login to application
    objLogin.loginToGuru99("mgr123", "mgr!23");
}
```

```

// go the next page

objHomePage = new Guru99HomePage(driver);

//Verify home page

Assert.assertTrue(objHomePage.getHomePageDashboardUserName().toLowerCase().contains("manger id : mgr123"));

}

```

What is Page Factory?

Page Factory is an inbuilt Page Object Model concept for Selenium WebDriver but it is very optimized.

Here as well, we follow the concept of separation of Page Object Repository and Test Methods. Additionally, with the help of PageFactory class, we use annotations **@FindBy** to find WebElement. We use initElements method to initialize web elements

WebElements are identify by
@FindBy Annotation

static initElements method of
PageFactory class for
initializing WebElement

```

@FindBy(xpath = "/table//tr[@class='heading3']")
WebElement homePageUserName;

public Guru99HomePage(WebDriver driver){
    this.driver = driver;
    //This initElements method will create all WebElements
    PageFactory.initElements(driver, this);
}

```

@FindBy can accept **tagName**, **partialLinkText**, **name**,

linkText, id, css, className, xpath as attributes.

Let's look at the same example as above using Page Factory

Guru99 Login page with Page Factory

```
package PageFactory;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
public class Guru99Login {

    /**
     * All WebElements are identified by @FindBy annotation
     */
    WebDriver driver;
    @FindBy(name="uid")
    WebElement user99GuruName;

    @FindBy(name="password")
    WebElement password99Guru;

    @FindBy(className="barone")
    WebElement titleText;
```

```
@FindBy(name="btnLogin")  
WebElement login;  
  
public Guru99Login(WebDriver driver){  
    this.driver = driver;  
    //This initElements method will create all WebElements  
    PageFactory.initElements(driver, this);  
}  
  
//Set user name in textbox  
  
public void setUserName(String strUserName){  
    user99GuruName.sendKeys(strUserName);  
  
}  
  
//Set password in password textbox  
  
public void setPassword(String strPassword){  
    password99Guru.sendKeys(strPassword);  
}  
  
//Click on login button
```

```
public void clickLogin(){

    login.click();

}

//Get the title of Login Page

public String getLoginTitle(){

    return    titleText.getText();

}

/***
 * This POM method will be exposed in test case to login in
the application

 * @param strUserName

 * @param strPasword

 * @return

 */

public void loginToGuru99(String strUserName, String
strPasword){

    //Fill user name

    this.setUserName(strUserName);

    //Fill password

    this.setPassword(strPasword);

    //Click Login button

    this.clickLogin();
```

```
 }  
}
```

Guru99 Home Page with Page Factory

```
package PageFactory;  
  
import org.openqa.selenium.WebDriver;  
  
import org.openqa.selenium.WebElement;  
  
import org.openqa.selenium.support.FindBy;  
  
import org.openqa.selenium.support.PageFactory;  
  
public class Guru99HomePage {  
  
    WebDriver driver;  
  
    @FindBy(xpath="//table//tr[@class='heading3']")  
  
    WebElement homePageUserName;  
  
  
  
    public Guru99HomePage(WebDriver driver){  
  
        this.driver = driver;  
  
        //This initElements method will create all WebElements  
  
        PageFactory.initElements(driver, this);  
  
    }  
  
  
  
    //Get the User name from Home Page
```

```
public String getHomePageDashboardUserName(){  
    return     homePageUserName.getText();  
}  
}
```

Guru99 TestCase with Page Factory concept

```
package test;  
  
import java.util.concurrent.TimeUnit;  
  
import org.openqa.selenium.WebDriver;  
  
import org.openqa.selenium.firefox.FirefoxDriver;  
  
import org.testng.Assert;  
  
import org.testng.annotations.BeforeTest;  
  
import org.testng.annotations.Test;  
  
import PageFactory.Guru99HomePage;  
  
import PageFactory.Guru99Login;  
  
public class Test99GuruLoginWithPageFactory {  
  
    WebDriver driver;  
  
    Guru99Login objLogin;  
  
    Guru99HomePage objHomePage;
```

```
@BeforeTest

public void setup(){

    driver = new FirefoxDriver();

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

    driver.get("http://demo.guru99.com/V4/");

}

/**
 * This test go to http://demo.guru99.com/V4/
 * Verify login page title as guru99 bank
 * Login to application
 * Verify the home page using Dashboard message
 */

@Test(priority=0)

public void test_Home_Page_Appear_Correct(){

    //Create Login Page object
    objLogin = new Guru99Login(driver);

    //Verify login page title
    String loginPageTitle = objLogin.getLoginTitle();

    Assert.assertTrue(loginPageTitle.toLowerCase().contains("guru99
bank"));
}
```

```

//login to application

objLogin.loginToGuru99("mgr123", "mgr!23");

// go the next page

objHomePage = new Guru99HomePage(driver);

//Verify home page

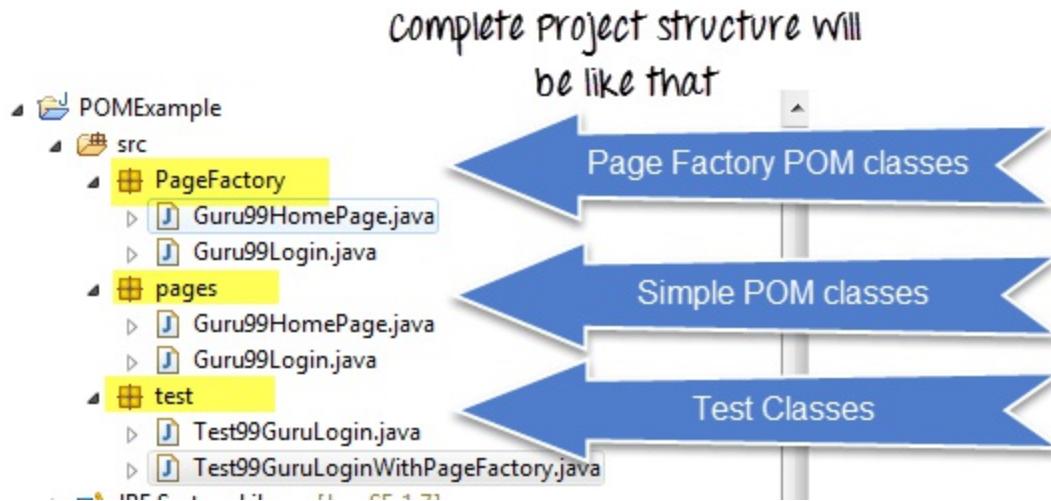
Assert.assertTrue(objHomePage.getHomePageDashboardUserName().toLowerCase().contains("manger id : mgr123"));

}

}

```

Complete Project Structure will look like the diagram:



AjaxElementLocatorFactory

One of the key advantages of using Page Factory pattern is

AjaxElementLocatorFactory Class.

It is working on lazy loading concept, i.e. a timeout for a WebElement will be assigned to the Object page class with the help of AjaxElementLocatorFactory .

Here, when an operation is performed on an element the wait for its visibility starts from that moment only. If the element is not found in the given time interval, Test Case execution will throw 'NoSuchElementException' exception.

after 100 sec if element is not visible to perform an operation , timeout exception will appear

```
AjaxElementLocatorFactory factory = new AjaxElementLocatorFactory(driver, 100);
PageFactory.initElements(factory, this);
```

This is a lazy loading,wait will start only if we perform operation on control

Summary

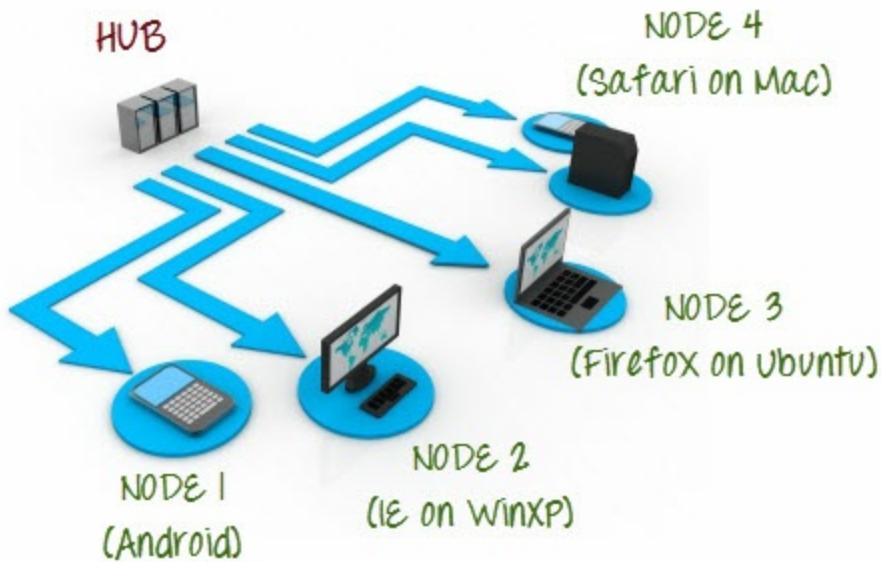
1. Page Object Model is an Object Repository design pattern in Selenium WebDriver.
2. POM creates our testing code maintainable, reusable.
3. Page Factory is an optimized way to create object repository in POM concept.
4. AjaxElementLocatorFactory is a lazy load concept in Page Factory pattern to identify WebElements only when they are used in any operation.

Chapter 26: Introduction to Selenium Grid

What is Selenium Grid?

Selenium Grid is a part of the Selenium Suite that specializes in 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-O.S combinations.
- **Save time in the 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.

Selenium Grid Architecture

Selenium Grid has a Hub and Node Architecture.

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 O.S 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.

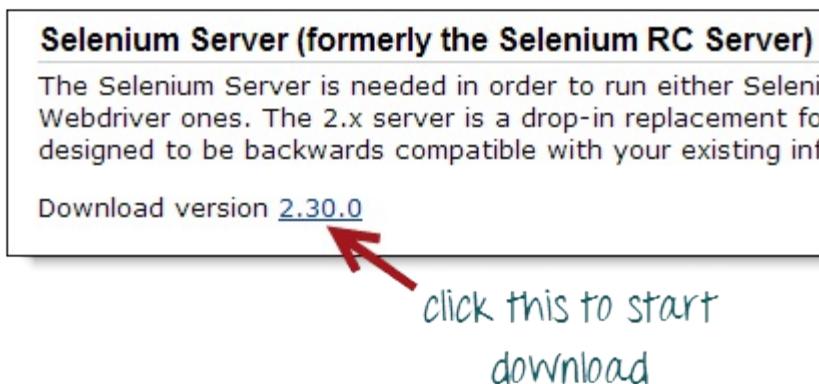
Selenium grid can be set up in two different ways; one through command line and the other through JSON config file.

How to Set Up Selenium Grid? Using Command Line

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 from here.



Step 2

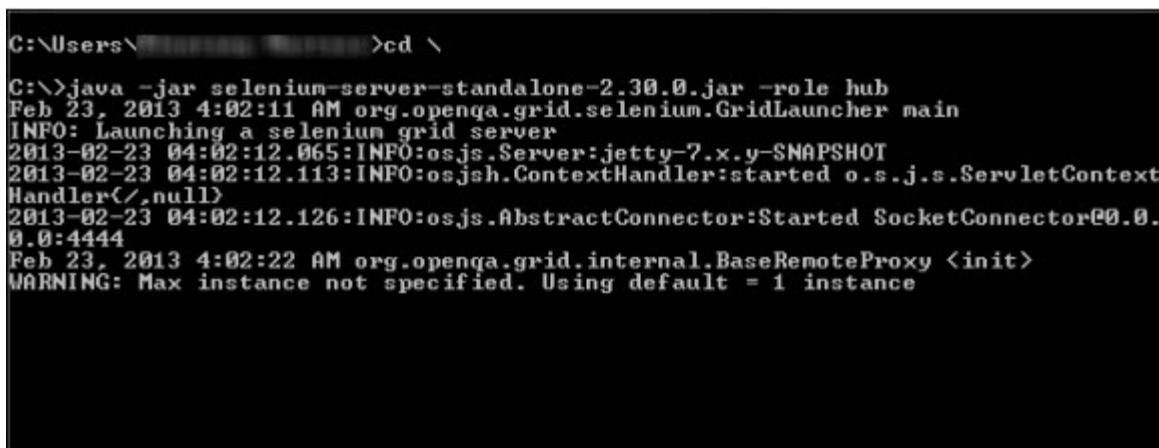
You can place the Selenium Server .jar file anywhere in your HardDrive. But for the purpose of this tutorial, place it on the C drive of both Machine A and Machine B. After doing this, you are now done installing Selenium Grid. The following steps will launch the hub and the node.

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.

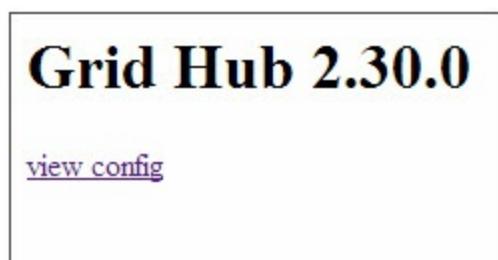
- On the command prompt, type **java -jar selenium-server-standalone-2.30.0.jar -role hub**
- The hub should successfully be launched. Your command prompt should look similar to the image below



```
C:\>cd \  
C:\>java -jar selenium-server-standalone-2.30.0.jar -role hub  
Feb 23, 2013 4:02:11 AM org.openqa.grid.selenium.GridLauncher main  
INFO: Launching a selenium grid server  
2013-02-23 04:02:12.065:INFO:osjs.Server:jetty-7.x.y-SNAPSHOT  
2013-02-23 04:02:12.113:INFO:osjsh.ContextHandler:started o.s.j.s.ServletContext  
Handler</, null>  
2013-02-23 04:02:12.126:INFO:osjs.AbstractConnector:Started SocketConnector@0.0.  
0.0:4444  
Feb 23, 2013 4:02:22 AM org.openqa.grid.internal.BaseRemoteProxy <init>  
WARNING: Max instance not specified. Using default = 1 instance
```

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 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`

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.
- NOTE: You now have to give path to the Gecko driver if using Firefox. Here is updated code that needs to be used

```
java -Dwebdriver.gecko.driver="C:\geckodriver.exe" -jar selenium-server-standalone-3.4.0.jar -role webdriver -hub http://192.168.1.3:4444/grid/register -port 5566
```

```
C:\> java -jar selenium-server-standalone-2.30.0.jar -role webdriver -hub http://192.168.1.3:4444/grid/register -port 5566
```



IP address of the machine
where the hub is running

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

```
C:\>java -jar selenium-server-standalone-2.30.0.jar -role webdriver -hub http://192.168.1.3:4444/grid/register -port 5566
Feb 23, 2013 4:34:39 PM org.openqa.grid.selenium.GridLauncher main
INFO: Launching a selenium grid node
16:34:40.733 INFO - Java: Oracle Corporation 23.7-b01
16:34:40.733 INFO - OS: Windows XP 5.1 x86
16:34:40.733 INFO - v2.30.0, with Core v2.30.0. Built from revision dc1ef9c
16:34:40.874 INFO - RemoteWebDriver instances should connect to: http://127.0.0.1:5566/wd/hub
16:34:40.874 INFO - Version Jetty/5.1.x
16:34:40.874 INFO - Started HttpContext[/selenium-server/driver,/selenium-server/driver]
16:34:40.889 INFO - Started HttpContext[/selenium-server,/selenium-server]
16:34:40.889 INFO - Started HttpContext[/,/]
16:34:40.889 INFO - Started org.openqa.jetty.servlet.ServletHandler@ed3512

16:34:40.889 INFO - Started HttpContext[/wd,/wd]
16:34:40.905 INFO - Started SocketListener on 0.0.0.0:5566
16:34:40.905 INFO - Started org.openqa.jetty.Server@1f77497
16:34:40.905 INFO - using the json request : {"class":"org.openqa.grid.common.RegistrationRequest","capabilities":[{"platform":"XP","seleniumProtocol":"WebDriver","browserName":"firefox","maxInstances":5},{"platform":"XP","seleniumProtocol":"WebDriver","browserName":"chrome","maxInstances":5}, {"platform":"WINDOWS","seleniumProtocol":"WebDriver","browserName":"internet explorer","maxInstances":1}, {"configuration":{"port":5566,"register":true,"host":"192.168.1.4","proxy":"org.openqa.grid.selenium.proxy.DefaultRemoteProxy","maxSession":5,"role":"webdriver","hubHost":"192.168.1.3","registerCycle":5000,"hub":"http://192.168.1.3:4444/grid/register","hubPort":4444,"url":"http://192.168.1.4:5566","remoteHost":"http://192.168.1.4:5566"}]}
16:34:40.905 INFO - starting auto register thread. Will try to register every 500 ms.
16:34:40.905 INFO - Registering the node to hub :http://192.168.1.3:4444/grid/register
16:34:46.171 INFO - Executing: org.openqa.selenium.remote.server.handler.Status@1e5a771 at URL: /status
16:34:46.186 INFO - Done: /status
```

Step 6

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

Grid Hub 2.30.0

DefaultRemoteProxy

listening on http://192.168.1.4:5566

test session time out after 300 sec.

Supports up to 5 concurrent tests from:



[view config](#)

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

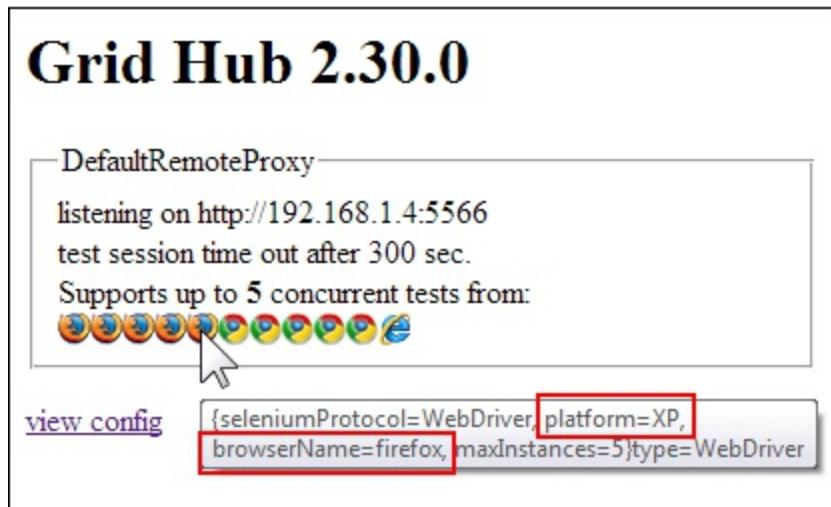
```
import org.openqa.selenium.DesiredCapabilities;
```

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

```
import java.net.MalformedURLException;
import java.net.URL;
import org.openqa.selenium.remote.RemoteWebDriver;
```

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** showed 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).

```
DesiredCapabilities capability = DesiredCapabilities.firefox();
capability.setBrowserName("firefox");
capability.setPlatform(Platform.XP);
```

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.

WE USED **RemoteWebDriver** and not **FirefoxDriver**

```
 WebDriver driver;
driver = new RemoteWebDriver(
    new URL("http://192.168.1.4:5566/wd/hub"), capability);
```

IP address and port on Machine B

Running a Sample Test Case on the Grid

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

```

import org.openqa.selenium.*;
import org.openqa.selenium.remote.DesiredCapabilities;
import java.net.MalformedURLException;
import java.net.URL;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.testng.Assert;
import org.testng.annotations.*;

public class Grid_2 {
    WebDriver driver;
    String baseUrl, nodeURL;

    @BeforeTest
    public void setUp() throws MalformedURLException {
        baseUrl = "http://newtours.demoaut.com/";
        nodeURL = "http://192.168.1.4:5566/wd/hub";
        DesiredCapabilities capability = DesiredCapabilities.firefox();
        capability.setBrowserName("firefox");
        capability.setPlatform(Platform.XP);
        driver = new RemoteWebDriver(new URL(nodeURL), capability);
    }

    @AfterTest
    public void afterTest() {
        driver.quit();
    }

    @Test
    public void simpleTest() {
        driver.get(baseUrl);
        Assert.assertEquals("Welcome: Mercury Tours", driver.getTitle());
    }
}

```

The test should pass.



Selenium grid configuration using JSON File:

JSON stands for Javascript Object Notation. It is a standard format used for interchange of data between browser and the web server. Selenium has an in built JSON config file that can be used to set up selenium grid.

Below are the steps to configure selenium grid using JSON config file.

Step 1) Download the code for JSON config file using the below path

<https://github.com/SeleniumHQ/selenium/blob/master/java/server/>

Here is the code

```
{
    "port": 4444,
    "newSessionWaitTimeout": -1,
    "servlets" : [],
    "withoutServlets": []}
```

```

    "custom": {},
    "capabilityMatcher":
"org.openqa.grid.internal.utils.DefaultCapabilityMatcher",
        "registryClass":
"org.openqa.grid.internal.DefaultGridRegistry",
            "throwOnCapabilityNotPresent": true,
            "cleanUpCycle": 5000,
            "role": "hub",
            "debug": false,
            "browserTimeout": 0,
            "timeout": 1800
}

```

Step 2) Copy and paste the code on a text editor such as Notepad with the extension '.json'

Step 3) Launch the hub using the below command using command prompt

```
java -jar selenium-server-standalone-2.53.1.jar -role hub -hubConfig hubconfig.json
```

NOTE: Selenium stand alone jar file and the json file must be present on the same directory

Step 4) The below screen will appear which indicates the hub is set up successfully

```
C:\windows\system32\cmd.exe - java -jar selenium-server-standalone-2.53.1.jar -role hub -hubConfig hubconfig.json
C:\Selenium Grid>java -jar selenium-server-standalone-2.53.1.jar -role hub -hubConfig hubconfig.json
16:20:53.092 INFO - Launching Selenium Grid hub
2017-12-02 16:20:55.209:INFO::main: Logging initialized @2567ms
16:20:55.221 INFO - Will listen on 4444
16:20:55.257 INFO - Will listen on 4444
2017-12-02 16:20:55.261:INFO:osjs.Server:main: jetty-9.2.z-SNAPSHOT
2017-12-02 16:20:55.289:INFO:osjsh.ContextHandler:main: Started o.s.j.s.ServletContextHandler@10d59286{/,,null,AVAILABLE}
2017-12-02 16:20:55.321:INFO:osjs.ServerConnector:main: Started ServerConnector@6483f5ae{HTTP/1.1}{0.0.0.0:4444}
2017-12-02 16:20:55.321:INFO:osjs.Server:main: Started @2680ms
16:20:55.321 INFO - Nodes should register to http://192.168.43.223:4444/grid/register/
16:20:55.321 INFO - Selenium Grid hub is up and running
```

Step 5) Open web browser and try connecting to the below URL

<http://192.168.43.223:4444/grid/console>

NOTE: The URL may vary from machine to machine. URL followed by 'Nodes should register to' on the above screen must be used.

Step 6) Grid Console screen will appear which indicates the hub is up and running



Configure NODES using JSON:

Step 1) Download the sample node configuration file provided by Selenium using the below URL

<https://github.com/SeleniumHQ/selenium/blob/selenium-2.53.0/java/server/src/org/openqa/grid/common/defaults/DefaultNode.json>

NOTE: IF YOU ARE USING SELENIUM 3.0 OR ABOVE, THE BELOW JSON FILE MUST BE USED

<https://github.com/SeleniumHQ/selenium/blob/master/java/server/defaults/DefaultNodeWebDriver.json>

Step 2) Copy and paste the code into a text editor such as Notepad and save the file with the extension '.json'

Step 3)

Run the below command on command prompt

```
Java -Dwebdriver.chrome.driver="chromedriver.exe" -Dwebdriver.ie.driver="IEDriverServer.exe" -Dwebdriver.gecko.driver="geckodriver.exe" -jar selenium-server-standalone-2.53.1.jar -role node -nodeConfig node1.json
```

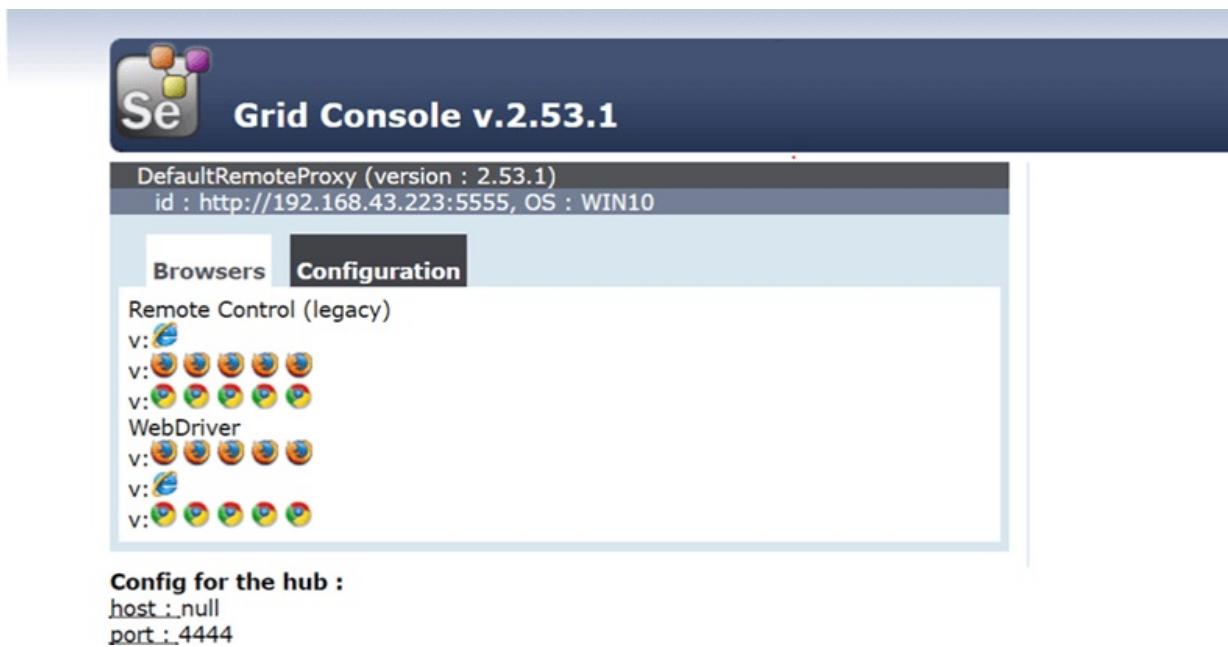
NOTE: Before running the command, please make sure the drivers for each of the browsers have been downloaded onto local machine directory

Step 4) The below screen will appear when enter key is pressed



```
C:\Windows\system32\cmd.exe - java -Dwebdriver.chrome.driver="chromedriver.exe" -Dwebdriver.ie.driver="IEDriverServer.exe" -Dwebdriver.gecko.driver="geckodriver.exe" -jar selenium-server-standalone-2.53.1.jar -role node -nodeConfig node1.json
C:\Selenium Grid>java -Dwebdriver.chrome.driver="chromedriver.exe" -Dwebdriver.ie.driver="IEDriverServer.exe" -Dwebdriver.gecko.driver="geckodriver.exe" -jar selenium-server-standalone-2.53.1.jar -role node -nodeConfig node1.json
16:44:08.246 INFO - Launching a Selenium Grid node
16:44:09.986 INFO - Java: Oracle Corporation 25.151-b12
16:44:09.986 INFO - OS: Windows 10 10.0 amd64
16:44:09.910 INFO - v2.53.1, with Core v2.53.1. Built from revision a36b8b1
16:44:09.950 INFO - Driver class not found: com.opera.core.systems.OperaDriver
16:44:09.950 INFO - Driver provider com.opera.core.systems.OperaDriver is not registered
16:44:09.958 INFO - Driver provider org.openqa.selenium.safari.SafariDriver registration is skipped: registration capabilities Capabilities [{browserName=safari, version=, platform=MAC}] does not match the current platform WIN10
16:44:09.958 INFO - Driver class not found: org.openqa.selenium.htmlunit.HtmlUnitDriver
16:44:09.958 INFO - Driver provider org.openqa.selenium.htmlunit.HtmlUnitDriver is not registered
16:44:09.982 INFO - Selenium Grid node is up and ready to register to the hub
16:44:10.010 INFO - Starting auto registration thread. Will try to register every 5000 ms.
16:44:10.010 INFO - Registering the node to the hub: http://192.168.43.223:4444/grid/register
16:44:10.030 INFO - The node is registered to the hub and ready to use
```

Step 5) Navigate to the grid console to verify if the nodes have been successfully configured



The browsers are displayed on the grid console. Hence we can be sure that nodes are configured successfully.

Sample Test Cases On Selenium Grid:

```
package com.objectrepository.demo;
import org.openqa.selenium.*;
import org.openqa.selenium.remote.DesiredCapabilities;
import java.net.MalformedURLException;
import java.net.URL;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.testng.Assert;
import org.testng.annotations.*;

public class SeleniumGrid {

    WebDriver driver;
    String baseURL, nodeURL;

    @BeforeTest
    public void setUp() throws MalformedURLException {
        baseURL = "http://demo.guru99.com/test/guru99home/";
        nodeURL = "http://192.168.43.223:4444/wd/hub";
        DesiredCapabilities capability =
DesiredCapabilities.chrome();
```

```

        capability.setBrowserName("chrome");
        capability.setPlatform(Platform.WIN10);
        driver = new RemoteWebDriver(new URL(nodeURL),
capability);
    }

    @AfterTest
    public void afterTest() {
        driver.quit();
    }
    @Test
    public void sampleTest() {
        driver.get(baseUrl);
        driver.get(baseUrl);

        if (driver.getPageSource().contains("MOBILE TESTING"))
{
            Assert.assertTrue(true, "Mobile Testing Link
Found");
        } else {
            Assert.assertTrue(false, "Failed: Link not found");
        }
    }
}

```

The above code launches chrome browser and navigates to the URL specified in the 'baseUrl' variable. It verified a link name 'Mobile Testing' is displayed on the page

URL on the 'nodeURL' variable can be modified to reflect the IP Address of the remote machine. Test result can be verified on the default TestNG report generated

Summary

- Selenium Grid is used to run multiple tests simultaneously on

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.

Chapter 27: Maven & Jenkins with Selenium: Complete Tutorial

What is Jenkins?

Jenkins is the leading open-source continuous integration tool developed by Hudson lab. It is cross-platform and can be used on Windows, Linux, Mac OS and Solaris environments. Jenkins is written in Java. Jenkin's chief usage is to monitor any job which can be SVN checkout, cron or any application states. It fires pre-configured actions when a particular step occurs in jobs.

Important Features of Jenkins

- Change Support: Jenkins generates the list of all changes done in repositories like SVN.
- Permanent links: Jenkins provides direct links to the latest build or failed build that can be used for easy communication
- Installation: Jenkins is easy to install either using direct installation file (exe) or war file to deploy using application server.
- Email integration: Jenkins can be configured to email the content of the status of the build.
- Easy Configuration: To configure various tasks on Jenkins is easy.
- TestNG test: Jenkins can be configured to run the automation test build on Testng after each build of SVN.
- Multiple VMs: Jenkins can be configured to distribute the build

on multiple machines.

- Project build: Jenkins documents the details of jar, version of jar and mapping of build and jar numbers.
- Plugins: 3rd party plugin can be configured in Jenkins to use features and additional functionality.

Why Jenkins and Selenium?

- Running Selenium tests in Jenkins allows you to run your tests every time your software changes and deploy the software to a new environment when the tests pass.
- Jenkins can schedule your tests to run at specific time.
- You can save the execution history and Test Reports.
- Jenkins supports Maven for building and Testing a project in continuous integration.

Why Maven & Jenkins

Selenium WebDriver is great for browser automation. But, when using it for testing and building a test framework, it feels underpowered.

Integrating Maven with Selenium provides following benefits

Apache Maven provides support for managing the full lifecycle of a test project.

- Maven is used to define project structure, dependencies, build, and test management.
- Using pom.xml(Maven) you can configure dependencies needed for building testing and running code.
- Maven automatically downloads the necessary files from the repository while building the project.

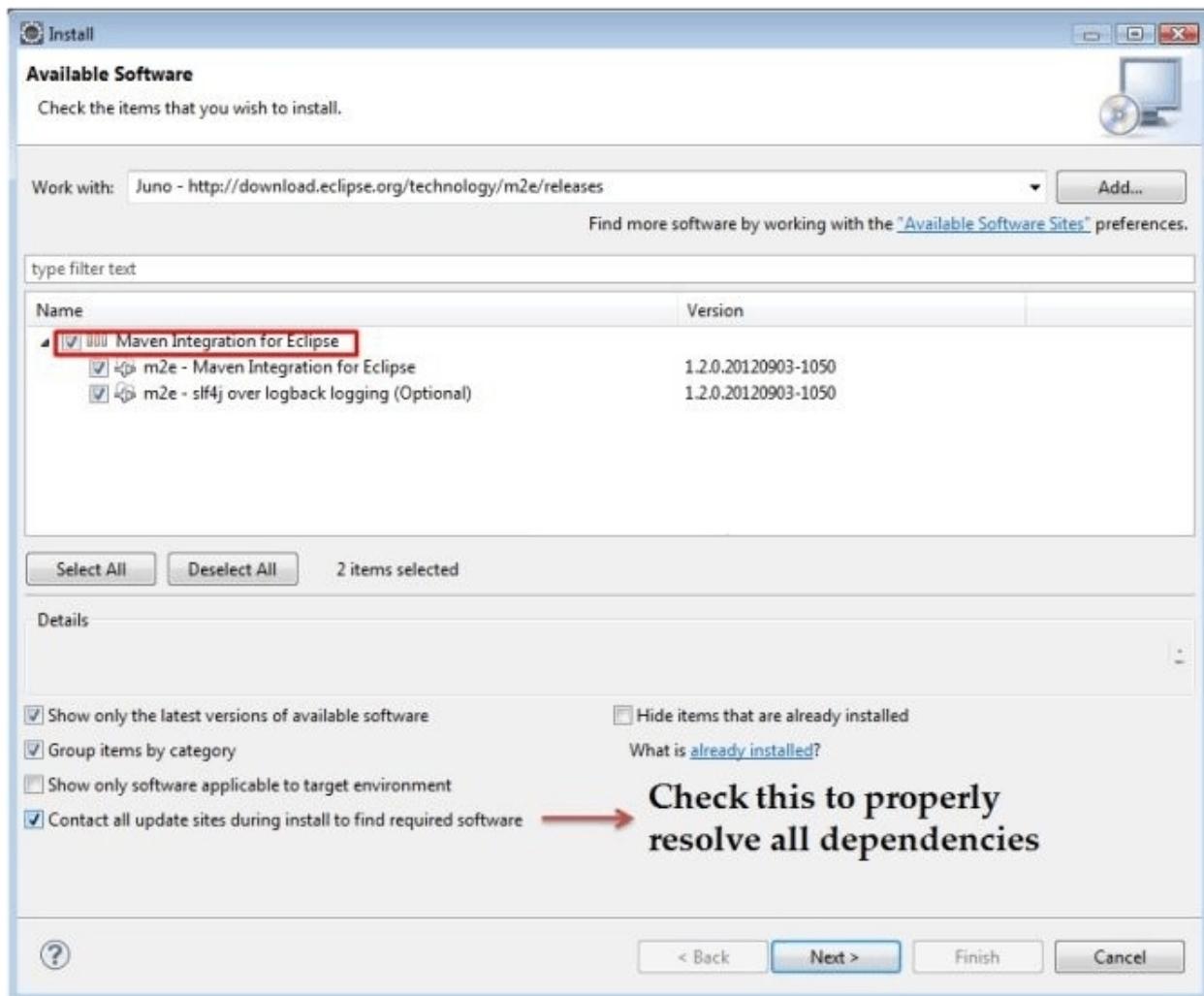
Steps to install Maven and use it with TestNG Selenium

For this tutorial, we will use Eclipse (Juno) IDE for Java Developers to set up Selenium WebDriver Project. Additionally, we need add m2eclipse plugin to Eclipse to facilitate the build process and create pom.xml file.

Let's add m2eclipse plugin to Eclipse with following steps:

Step1) In Eclipse IDE, select **Help | Install New Software** from Eclipse Main Menu.

Step 2) On the Install dialog, Enter the URL <http://download.eclipse.org/technology/m2e/releases/>. Select **Work with** and m2e plugin as shown in the following screenshot:



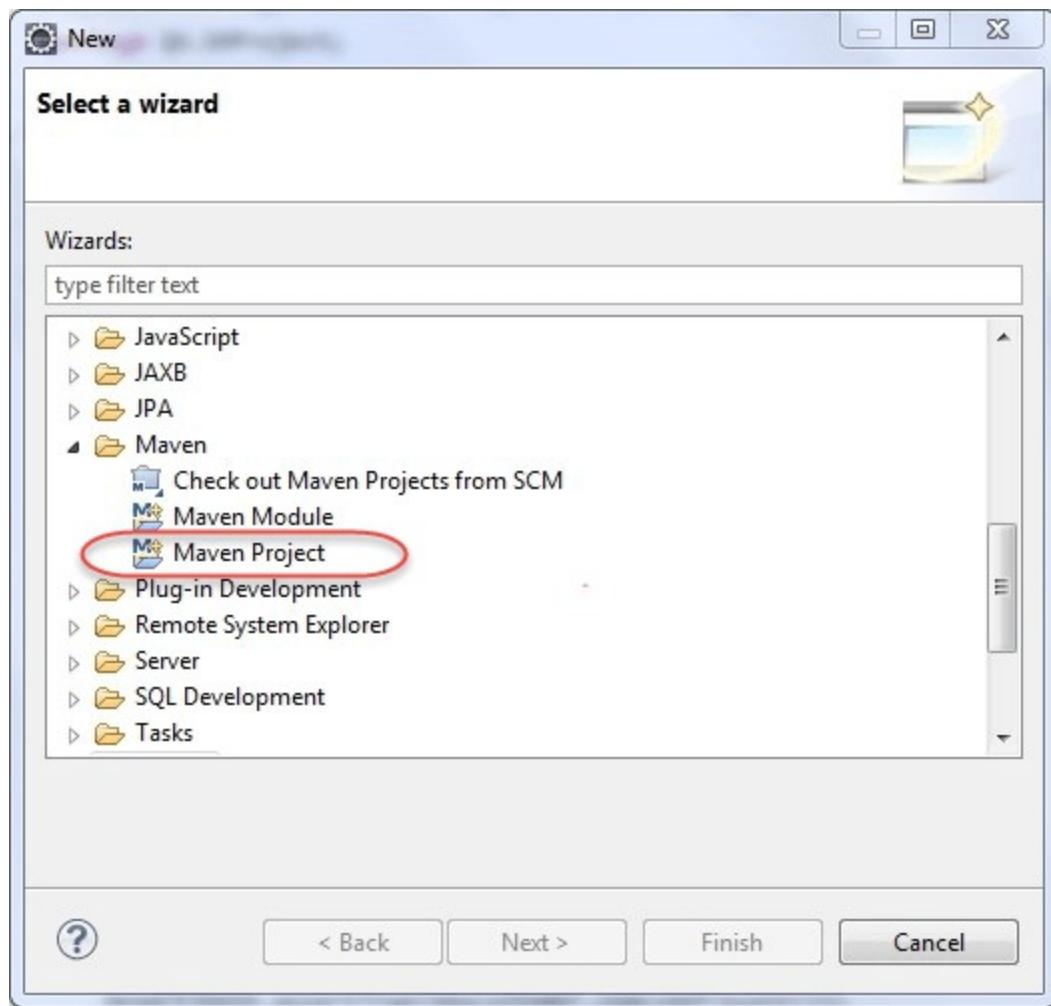
Step 3) Click on **Next** button and finish installation.

Configure Eclipse with Maven

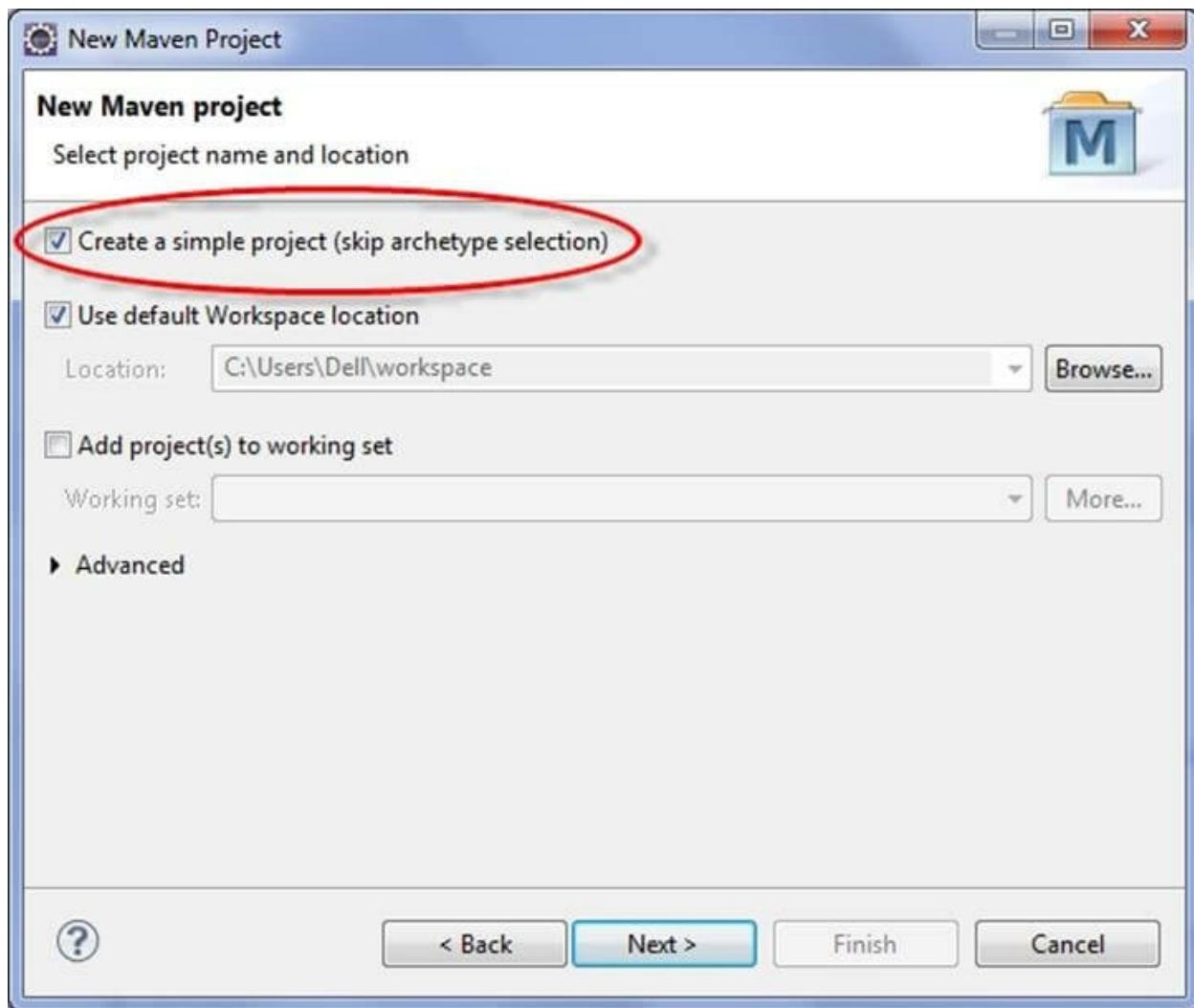
With m2e plugin is installed, we now need create Maven project.

Step 1) In Eclipse IDE, create a new project by selecting **File | New | Other** from Eclipse menu.

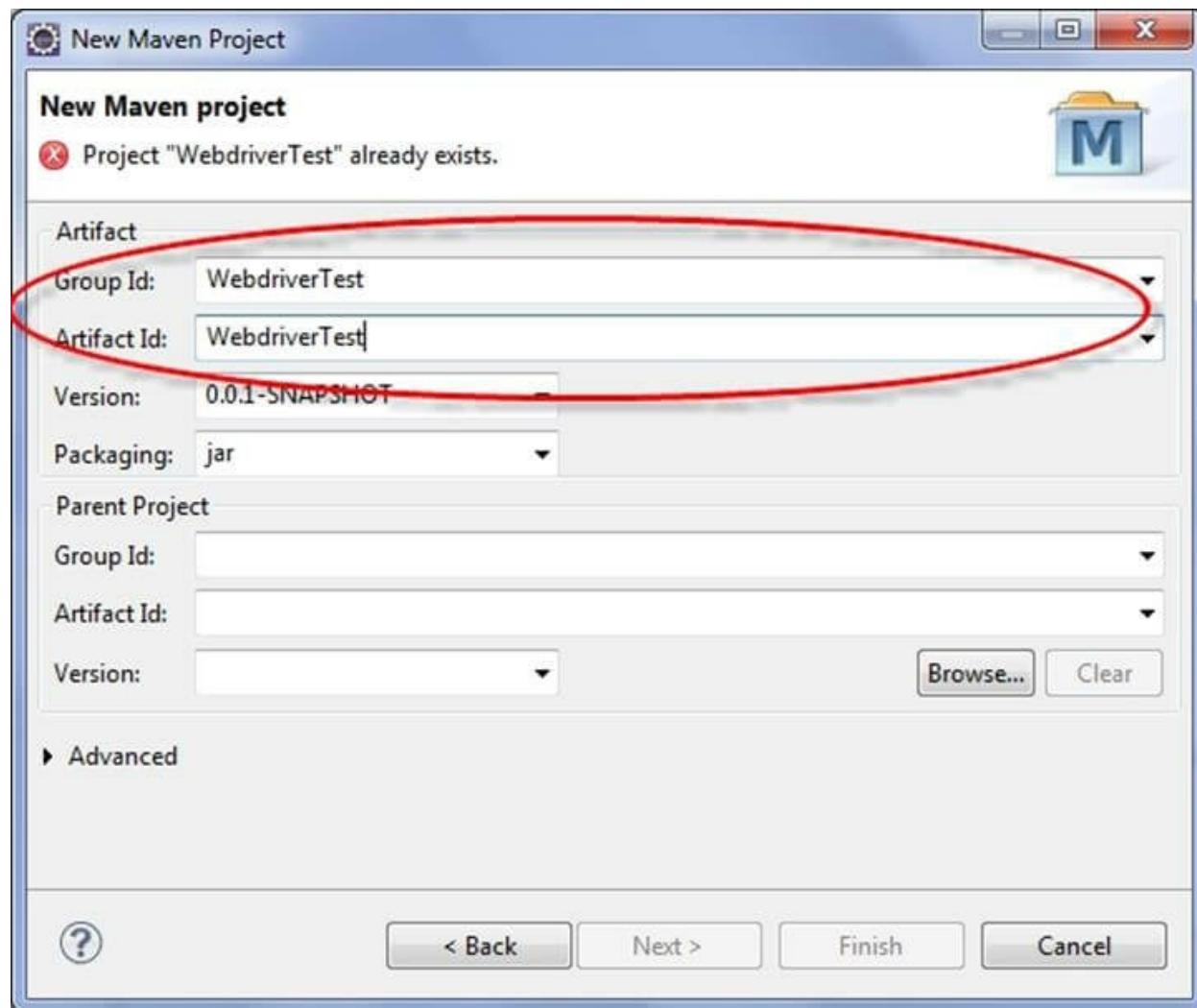
Step 2) On the **New** dialog, select **Maven | Maven Project** and click **Next**



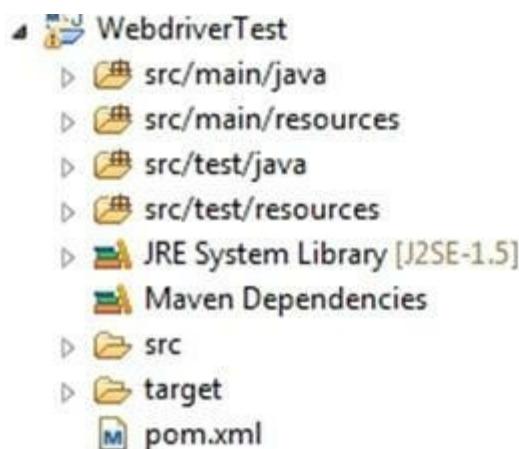
Step 3) On the New Maven Project dialog select the Create a simple project and click Next



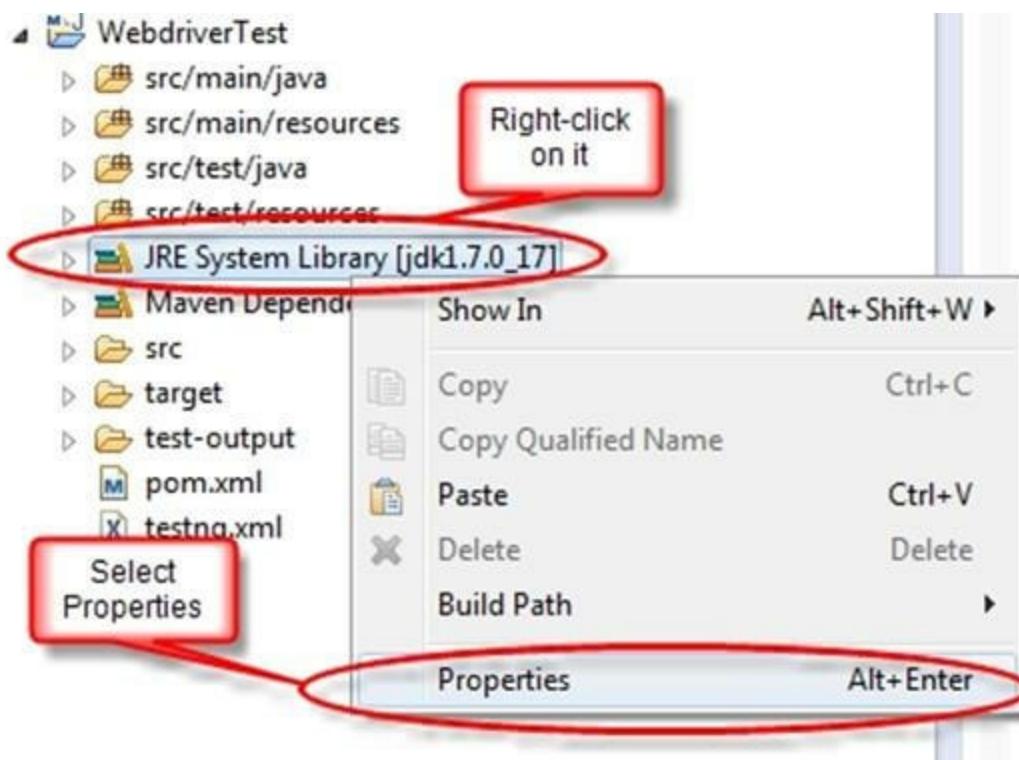
Step 4) Enter WebdriverTest in **Group Id:** and **Artifact Id:** and click finish



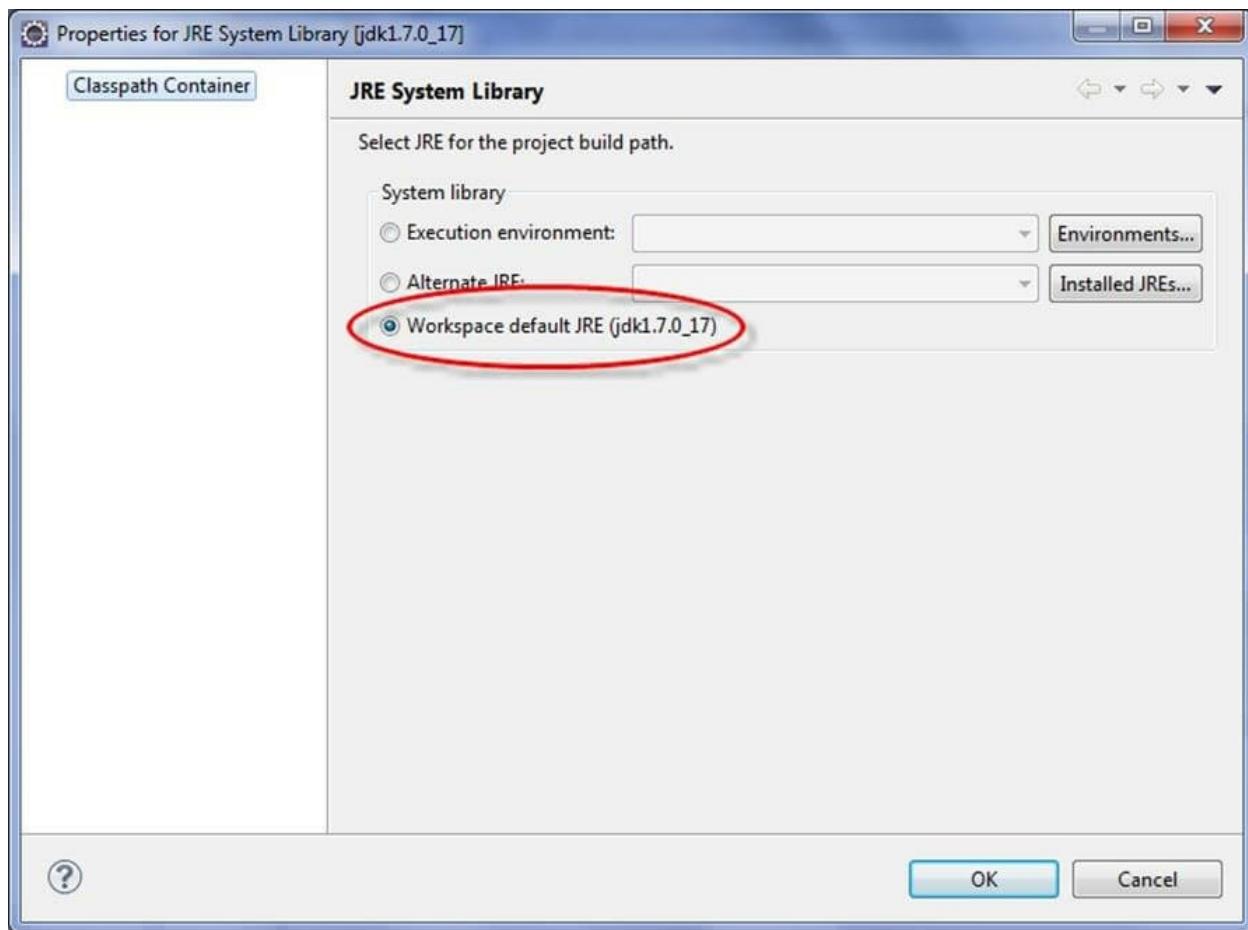
Step 5) Eclipse will create **WebdriverTest** with following structure:



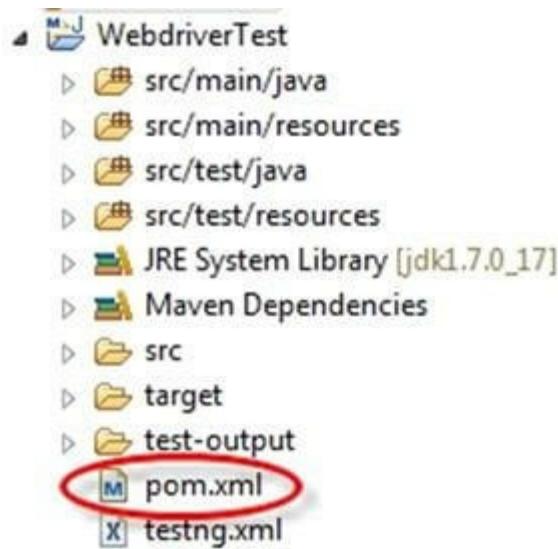
Step 6) Right-click on **JRE System Library** and select the **Properties** option from the menu.



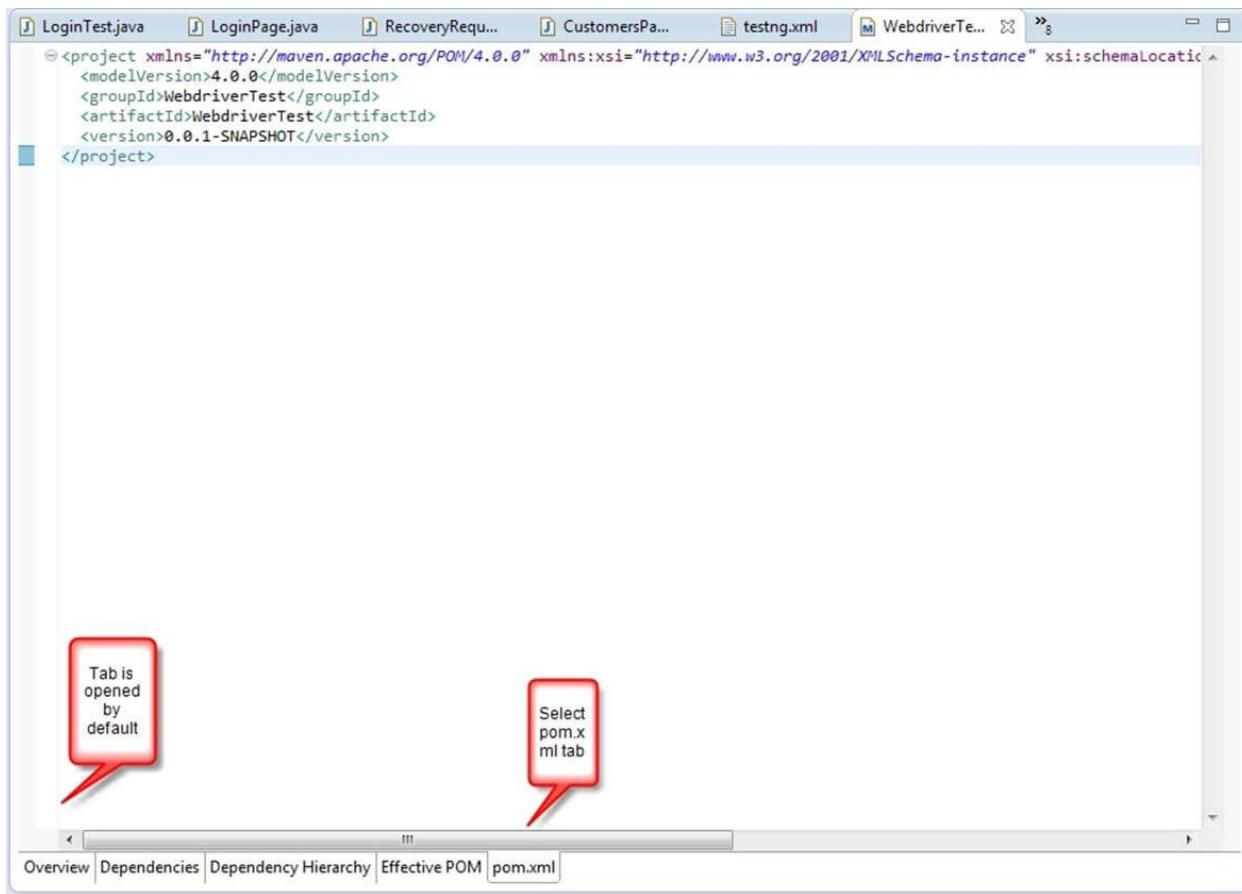
On the **Properties for JRE System Library** dialog box, make sure **Workspace default JRE** is selected and click OK



Step 7). Select pom.xml from Project Explorer..



pom.xml file will Open in Editor section

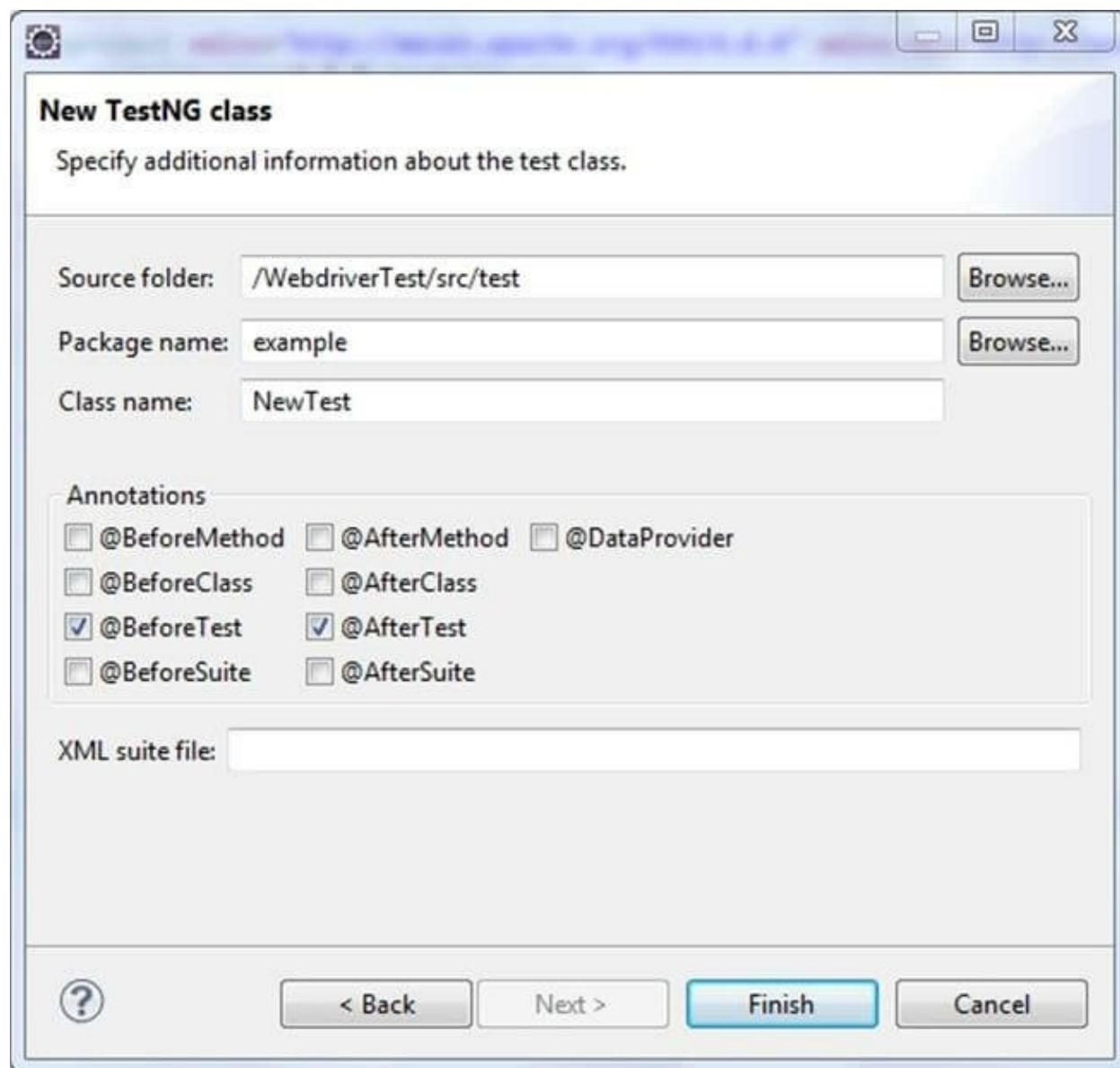


Step 8) Add the Selenium, Maven, TestNG, Junit dependencies to pom.xml in the <project> node:

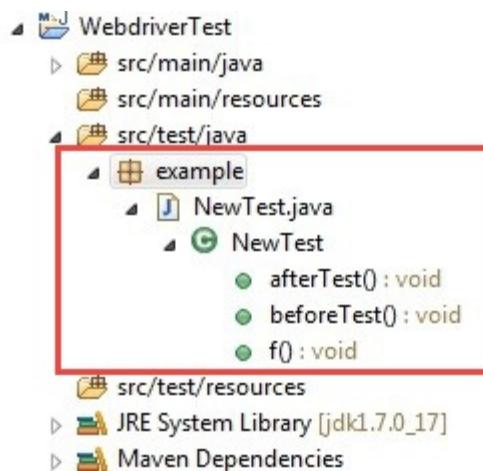
```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>2.45.0</version>
    </dependency>
<dependency>
```

```
<groupId>org.testng</groupId>
<artifactId>testng</artifactId>
<version>6.8</version>
<scope>test</scope>
</dependency>
</dependencies>
```

Step 9) Create a New TestNG Class. Enter Package name as "example" and "NewTest" in the **Name:** textbox and click on the **Finish** button as shown in the following screenshot:



Step 10). Eclipse will create the NewTest class as shown in the following screenshot:



Step 11) Add the following code to the **NewTest** class:

This code will verify the title of Guru99 Selenium Page

```
package example;

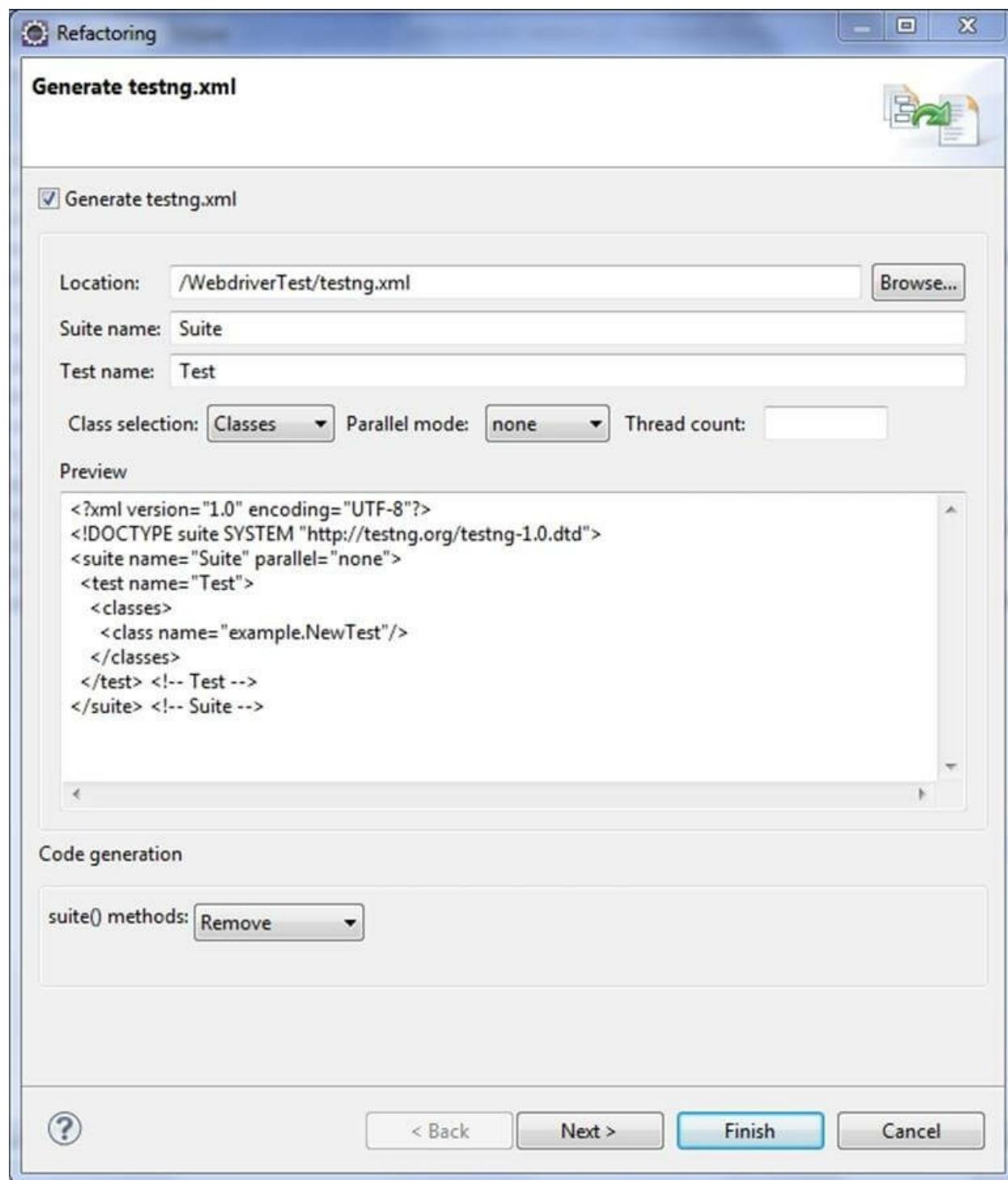
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.AfterTest;
public class NewTest {
    private WebDriver driver;
    @Test
    public void testEasy() {

driver.get("http://demo.guru99.com/test/guru99home/");
        String title = driver.getTitle();
        Assert.assertTrue(title.contains("Demo
Guru99 Page"));
    }
    @BeforeTest
    public void beforeTest() {
        driver = new FirefoxDriver();
```

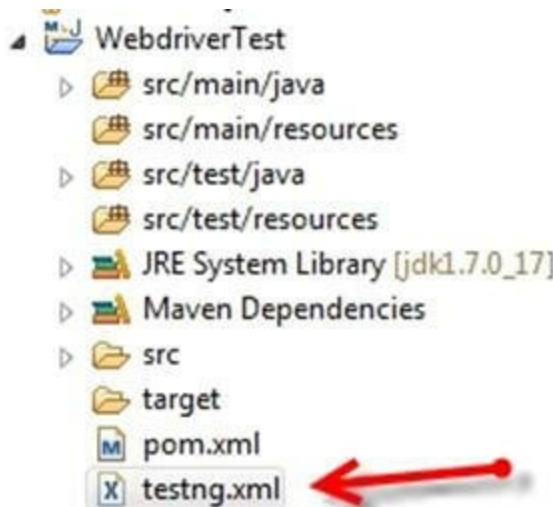
```
    }
    @AfterTest
    public void afterTest() {
        driver.quit();
    }
}
```

Step 12) Right-click on the WebdriverTest and select **TestNG | Convert to TestNG**.

Eclipse will create testng.xml which says that you need to run only one test with the name **NewTest** as shown in the following screenshot:

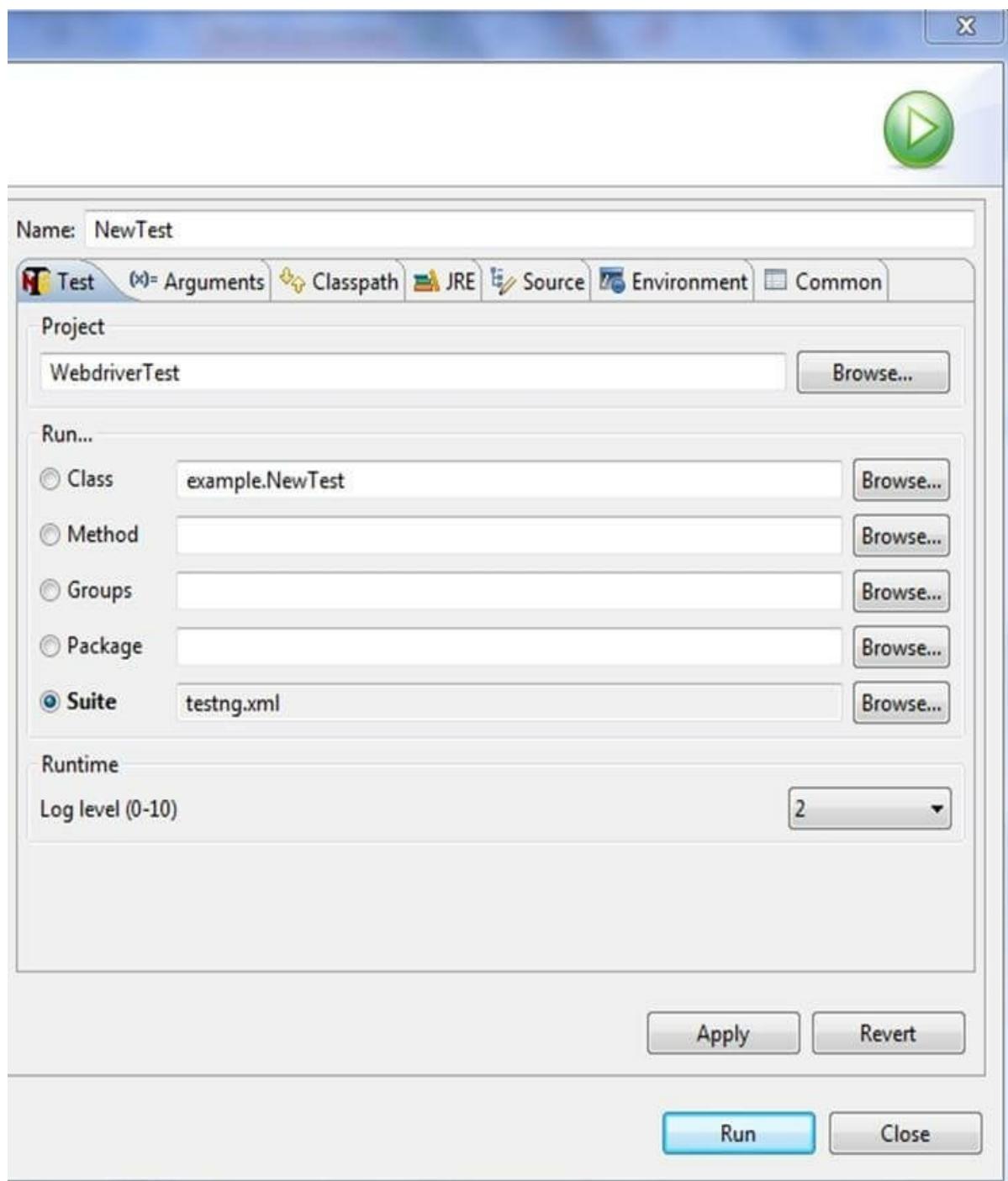


Update the project and make sure that file appears in the tree **Package Explorer** (right click on the project - Refresh).



Step 13) Now you need to run test through this **testng.xml**.

So, go to the **Run Configurations** and create a new launch **TestNG**, select the project and field **Suite** as **testng.xml** and click Run



Make sure that build finished successfully.

Step 14). Additionally, we need to add

1. maven-compiler-plugin

2. maven-surefire-plugin
3. testng.xml

to pom.xml.

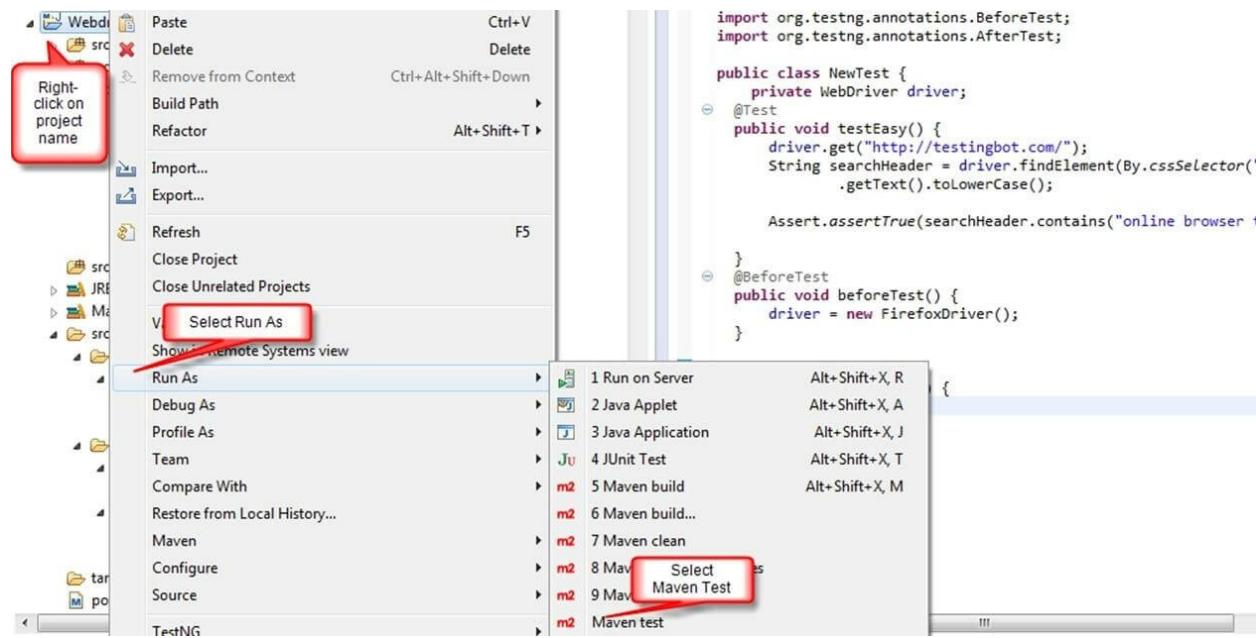
The maven-surefire-plugin is used to configure and execute tests. Here plugin is used to configure the testing.xml for TestNG test and generate test reports.

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

```
17 <plugins>
18   <plugin>
19     <groupId>org.apache.maven.plugins</groupId>
20     <artifactId>maven-compiler-plugin</artifactId>
21     <version>2.3.2</version>
22     <configuration>
23       <source>1.7</source>
24       <target>1.7</target>
25     </configuration>
26   </plugin>
27   <plugin>
28     <groupId>org.apache.maven.plugins</groupId>
29     <artifactId>maven-surefire-plugin</artifactId>
30     <version>2.12</version>
31     <inherited>true</inherited>
32     <configuration>
33       <suiteXmlFiles>
34         <suiteXmlFile>testng.xml</suiteXmlFile>
35       </suiteXmlFiles>
36     </configuration>
37   </plugin>
38 </plugins>
```

Step 15) To run the tests in the Maven lifecycle, Right-click on the WebdriverTest and select **Run As | Maven test**. Maven will execute

test from the project.



Make sure that build finished successfully.

Steps to Install Jenkins and configure it to Run Maven with TestNG Selenium

Installation

Step 1) Go to <http://jenkins-ci.org/> and download correct package for your OS. Install Jenkins.

Download Jenkins

Release Long-Term Support Release

Java Web Archive (.war)

Latest and greatest (1.607)
[changelog](#) | [past releases](#) | [RC](#)

[upgrading from Hudson?](#)

Or native package

-  Windows
-  Ubuntu/Debian
-  Red Hat/Fedora/CentOS
-  Mac OS X
-  openSUSE
-  FreeBSD
-  OpenBSD
-  Solaris/OpenIndiana
-  Gentoo

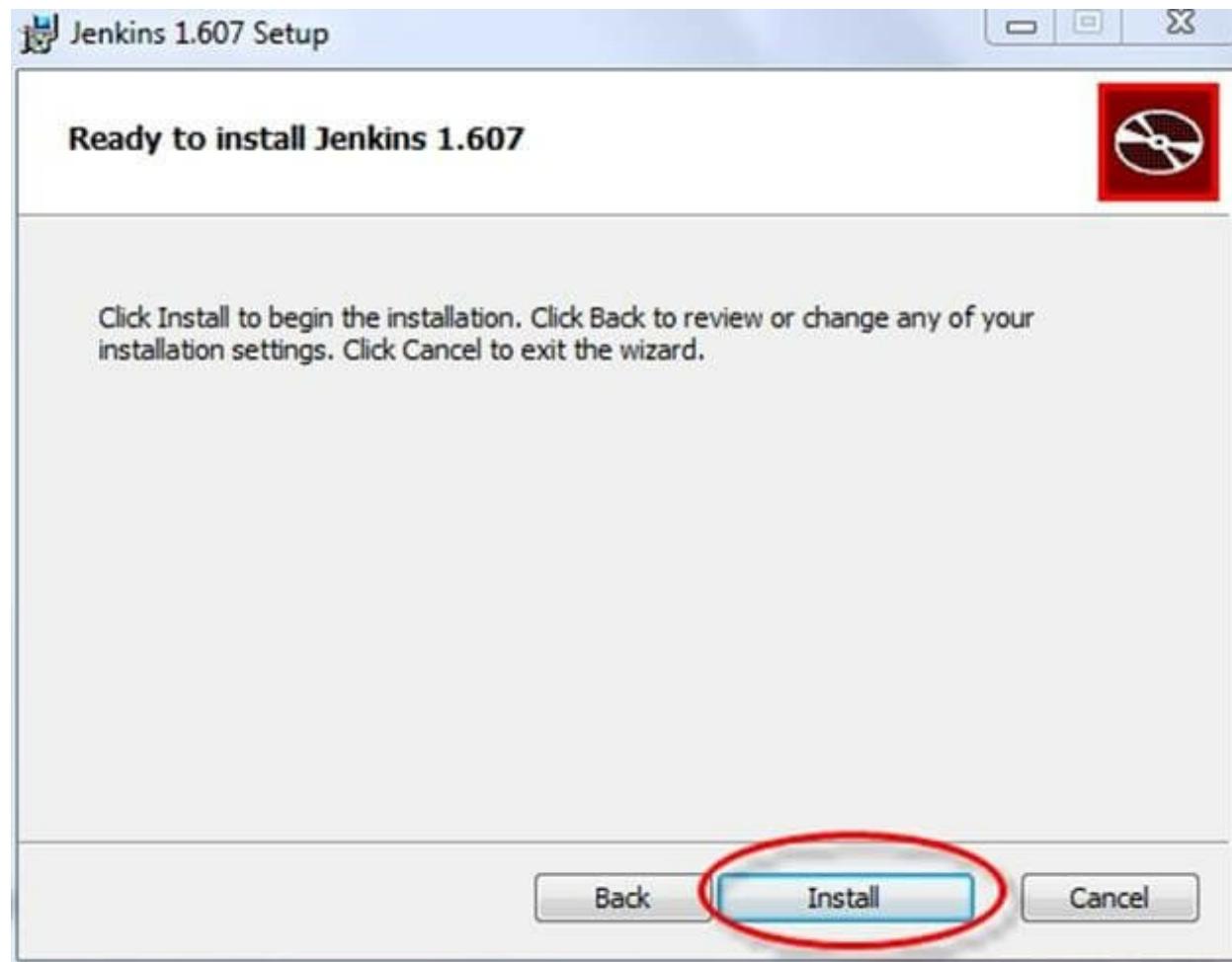
Step 2) Unzip Jenkins to specified folder. Run exe file as shown in following screenshot:

Имя	Дата изменения	Тип	Размер
jenkins-1.607	30.03.2015 20:28	Пакет установщи...	99 143 КБ
setup	30.03.2015 20:28	Приложение	471 КБ

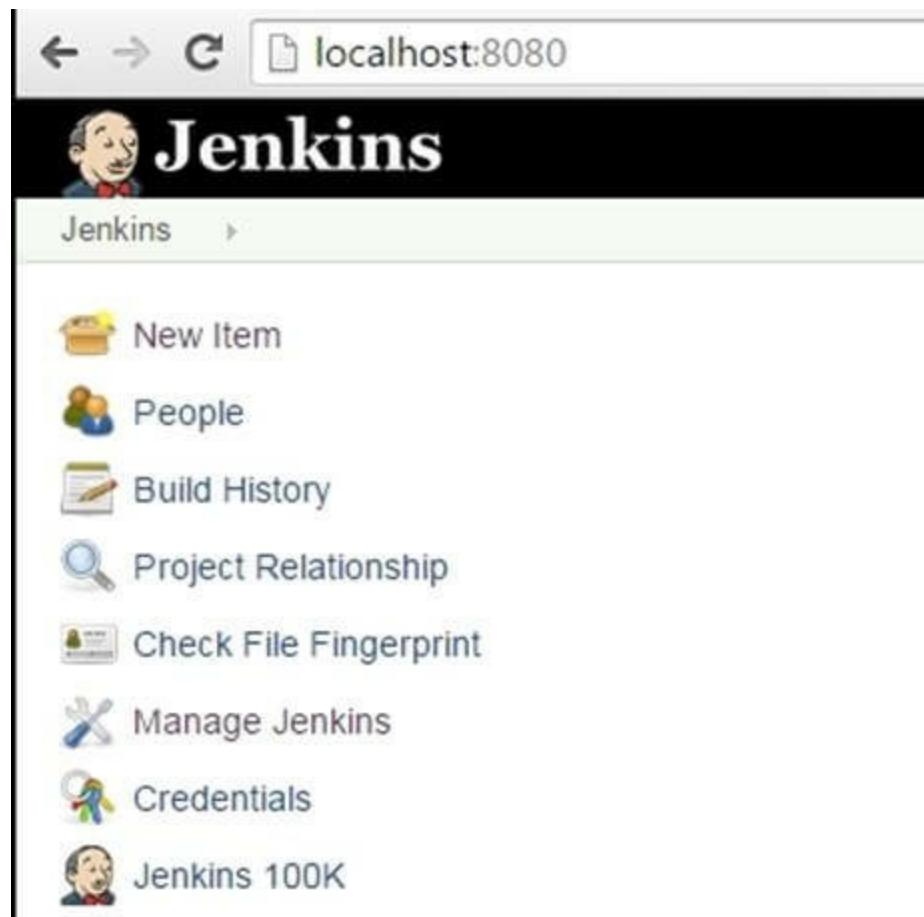
Step 3) In Jenkins 1.607 Setup window click on **Next** button.



Step 4) Click on **Install** button in the end.



Step 5) Once installation is done, navigate to the Jenkins Dashboard (<http://localhost:8080> by default) in the browser window.



Step 6) Click on the **New Item** link to create a CI job.



Step 7) Select the Maven project radio button as shown in the following screenshot:

Item name

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

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

External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This Jenkins as a dashboard of your existing automation system. See [the documentation for more details](#).

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

Copy existing Item
Copy from

Using the Build a **Maven Project** option, Jenkins supports building and testing Maven projects.

Step 6) Click on OK button. A new job with name "WebdriverTest" is created in Jenkins Dashboard.



Step 7) Go to **Manage Jenkins => Configure System** as shown in the following screenshot.

Manage Jenkins

- Configure System** Configure global settings and paths.
- Configure Global Security** Secure Jenkins; define who is allowed to access Jenkins.
- Reload Configuration from Disk** Discard all the loaded data in memory and reload config files directly on disk.
- Manage Plugins** Add, remove, disable or enable plugins that can extend Jenkins.
- System Information** Displays various environmental information to help troubleshoot.

Click on JDK installations and configure JDK as in the following screenshot:

JDK installations	Name	java 1.7.0
	JAVA_HOME	C:\Program Files\Java\jdk1.7.0_17

Install automatically **Delete JDK**

Add JDK

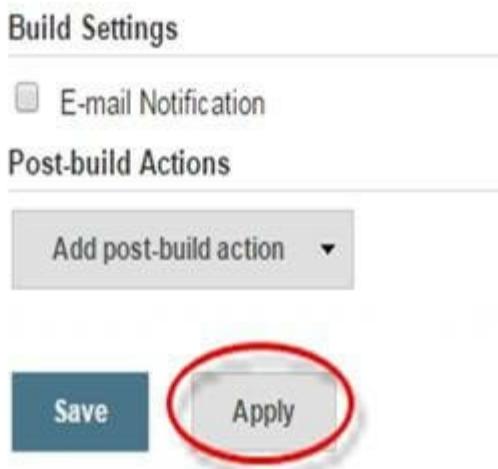
List of JDK installations on this system

Step 8) Go to the **Build** section of new job.

- In the **Root POM** textbox, enter full path to pom.xml
- In Goals and options section, enter "clean test"



Step 9) Click on **Apply** button.



Step 10) On the WebdriverTest project page, click on the **Build Now** link.

Maven project WebdriverTest

Back to Dashboard | Status | Changes | Workspace | Build Now | Delete Maven project | Configure | Modules

Workspace | Recent Changes

Permalinks

Build History | trend

Maven will build the project. It will then have TestNG execute the test cases.

Step 11) Once the build process is completed, in Jenkins Dashboard click on the **WebdriverTest** project

S	W	Name ↓	Last Success	Last Failure
		Tests	3 days 18 hr - #13	2 days 22 hr - #18
		<u>WebdriverTest</u>	19 hr - #9	19 hr - #7

Icon: S M L

Step 12) The WebdriverTest project page displays the build history and links to the results as shown in the following screenshot:

Maven project WebdriverTest

Build history

Build History

Recent Changes

Latest Test Result (no failures)

Latest Test Result (no failures)

Permalinks

- [Last build \(#9\), 19 hr ago](#)
- [Last stable build \(#9\), 19 hr ago](#)
- [Last successful build \(#9\), 19 hr ago](#)
- [Last failed build \(#7\), 19 hr ago](#)
- [Last unstable build \(#8\), 19 hr ago](#)
- [Last unsuccessful build \(#8\), 19 hr ago](#)

Build History

#9 Mar 31, 2015 11:54 PM
#8 Mar 31, 2015 11:48 PM
#7 Mar 31, 2015 11:45 PM
#6 Mar 31, 2015 3:17 PM
#5 Mar 31, 2015 12:12 PM
#4 Mar 30, 2015 12:53 AM
#3 Mar 30, 2015 12:42 AM
#2 Mar 29, 2015 11:17 PM
#1 Mar 29, 2015 9:26 PM

RSS for all RSS for failures

Step 13) Click on the "Latest Test Result" link to view the test results as shown in the following screenshot:

Test Result

0 failures (-1)

Module	Fail	(diff)	Total	(diff)
ToolsQA-DemoMavenProject	0	-1	6	

Step 14). Select specific build, and you will see the current status by clicking on "**console output**".

```
-----  
T E S T S  
-----  
Running TestSuite  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.748 sec  
  
Results :  
  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0  
  
[JENKINS] Recording test results  
log4j:WARN No appenders could be found for logger (org.apache.commons.beanutils.converters.BooleanConverter).  
log4j:WARN Please initialize the log4j system properly.  
[pool-1-thread-1 for channel] INFO org.apache.maven.cli.event.ExecutionEventLogger - -----  
[pool-1-thread-1 for channel] INFO org.apache.maven.cli.event.ExecutionEventLogger - BUILD SUCCESS  
[pool-1-thread-1 for channel] INFO org.apache.maven.cli.event.ExecutionEventLogger - -----  
[pool-1-thread-1 for channel] INFO org.apache.maven.cli.event.ExecutionEventLogger - Total time: 01:26 min  
[pool-1-thread-1 for channel] INFO org.apache.maven.cli.event.ExecutionEventLogger - Finished at: 2015-03-29T21:28:51+03:00  
[pool-1-thread-1 for channel] INFO org.apache.maven.cli.event.ExecutionEventLogger - Final Memory: 25M/52M  
[pool-1-thread-1 for channel] INFO org.apache.maven.cli.event.ExecutionEventLogger - -----  
  
Ожидая пока Jenkins закончит сбор данных  
[JENKINS] Archiving C:\Users\Dell\workspace\FirstWebdriverTest\pom.xml to ToolsQA/DemoMavenProject/0.0.1-SNAPSHOT/DemoMavenProject-  
0.0.1-SNAPSHOT.pom  
channel stopped  
Finished: SUCCESS
```

Scheduling Jenkins for automatic execution.

Scheduling builds(Selenium Tests) is one of the important features of Jenkins where it automatically triggers the build, based on defined criteria. Jenkins provides multiple ways to trigger the build process under the Build Trigger configuration.

For example:

Enter `0 23 * * *` in the Schedule textbox as shown in the following screenshot. This will trigger the build process every day at 11 p.m.

Build Triggers

The screenshot shows the 'Build Triggers' section of a Jenkins job configuration. It includes three checkboxes: 'Build whenever a SNAPSHOT dependency is built' (unchecked), 'Build after other projects are built' (unchecked), and 'Build periodically' (checked and circled). Below the checkboxes is a 'Schedule' field containing the cron expression '0 23 * * *'. A red oval highlights the 'Build periodically' checkbox.

Using Jenkins without Maven

To run pure TestNG script in Jenkins, enter the following in build

```
D:>java -cp "Path to lib folder\lib\*;Path to bin folder\bin"
org.testng.TestNG testng.xml
```

The screenshot shows the 'Build' section of a Jenkins job configuration. It features a 'Execute Windows batch command' step. The 'Command' field contains the following text:
D:>java -cp "Path to lib folder\lib*;Path to bin folder\bin" org.testng.TestNG testng.xml
Below the command field is a link 'See the list of available environment variables'. To the right of the command field is a 'Delete' button. At the bottom of the build step section are 'Add build step' and 'Delete' buttons. The 'Post-build Actions' and 'Add post-build action' sections are also visible at the bottom.

- Click on Save button.
- Note: The actual path of lib and bin folder need to add in above command.
- After saving the command, Jenkins will build project in

predefined time, and this command will be run using TestNG.

- Result will be stored in custom report HTML file that can be sent via email with a Jenkins configuration
- Output of the code will be

The screenshot shows the Jenkins Test results interface. At the top, it says "Test results" and "1 suite". The suite listed is "temp.CopyOftestng1" with a status of "LiveLink105VM13". The left sidebar shows "All suites" and "Default suite" selected. Under "Info", it lists the XML file path: "C:\Users\hiteshkumar_panchani\AppData\Local\Temp\testng-eclipse-1972880750\testng-customsuite.xml". It also shows "1 test", "0 groups", "0 Times", "Reporter output", "Ignored methods", and "Chronological view". The "Results" section shows "1 method, 1 passed" and "Passed methods (hide) (show)". The main pane displays the XML code for the suite. Below the XML are sections for "Tests for Default suite" (listing "Default test (1 class)"), "Groups for Default suite", and "Times for Default suite".

Benefits of using Jenkins

1. Early issue finding – Bug can be detected in early phase of the software development
2. Automatic integration – no separate effort required to integrate all changes
3. Installer – a deployable system available at any point of development
4. Records – part build records maintained
5. Support and Plugins: One of the reasons for Jenkin's popularity is the availability of large community support. Also, lots of ready-made plugins are available which help you expand its functionality.

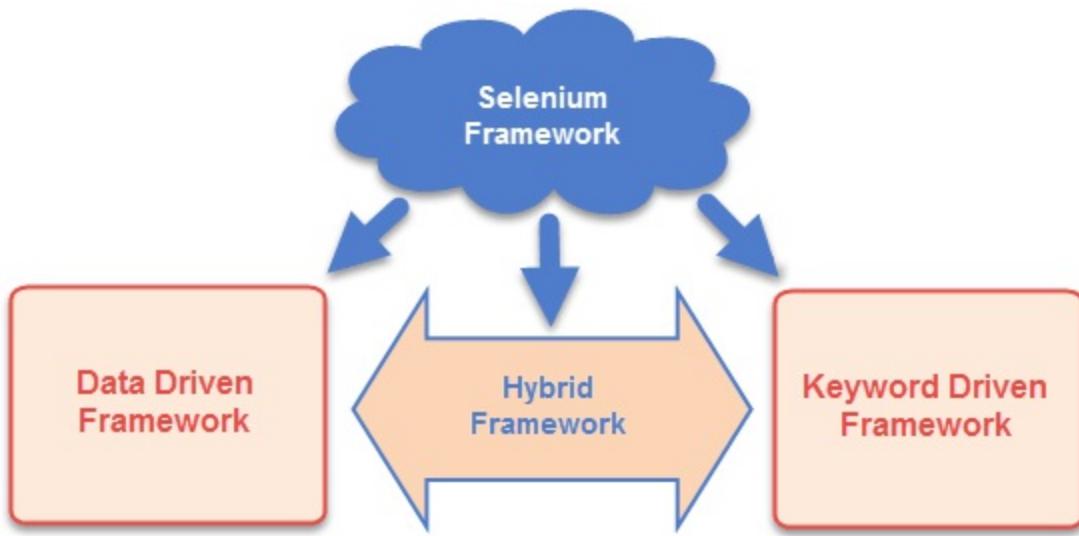
Chapter 28: Creating Keyword & Hybrid Frameworks with Selenium

What is Selenium Framework?

Selenium Framework is a code structure that helps to make code maintenance easy. Without frameworks, we will place the “code” as well as “data” in the same place which is neither re-usable nor readable. Using Frameworks, produce beneficial outcomes like increased code re-usage, higher portability, reduced script maintenance cost, higher code readability, etc.

There are mainly three type of frameworks created by Selenium WebDriver to automate manual test cases

- Data Driven Test Framework
- Keyword Driven Test Framework
- Hybrid Test Framework

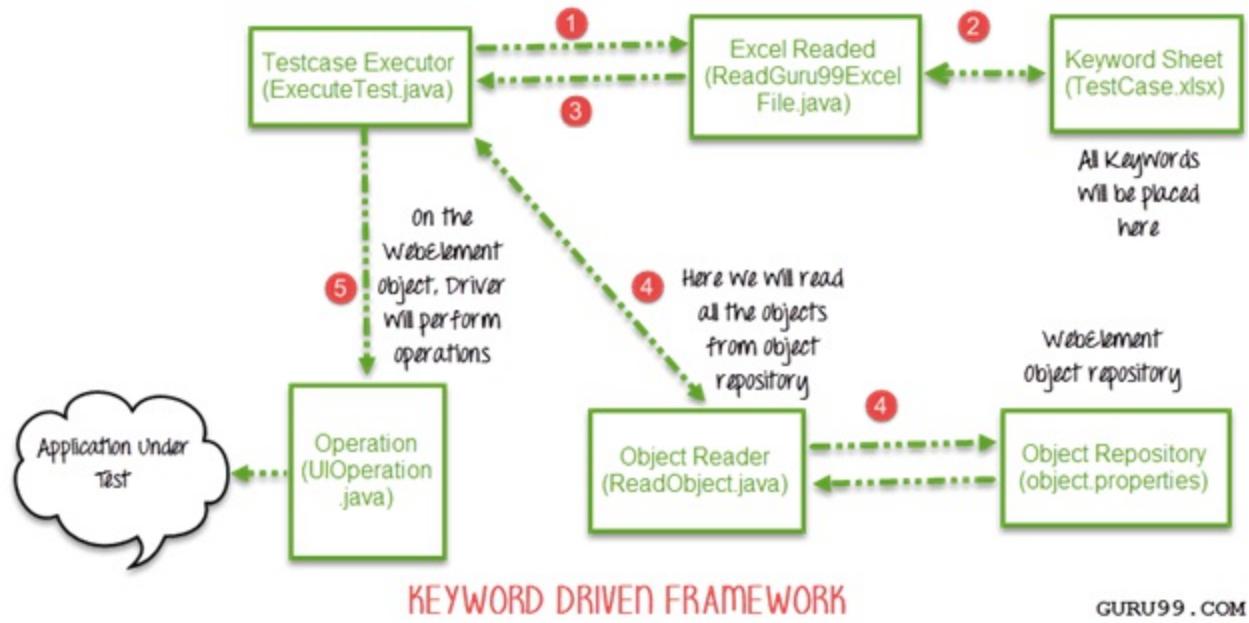


Data Driven Test Framework

In data driven framework all of our test data is generated from some external files like Excel, CSV, XML or some database table. We already learned about Data Driven Testing in our previous tutorial

Keyword Driven Test Framework:

In keyword driven test framework, all the operations and instructions are written in some external file like Excel worksheet. Here is how the complete framework looks like



As you can see it's a 5 step framework. Let's study it stepwise in detail

Step 1)

- The driver script Execute.java will call ReadGuru99ExcelFile.java
- ReadGuru99ExcelFile.java has POI script to read data from an Excel

Step 2)

- ReadGuru99ExcelFile.java will read data from TestCase.xlsx
- Here is how the sheet looks like-

Testcase Name	Keywords	Object Name	Object Type can be xpath, name,css etc.	Value for textbox area, url etc
TestCase	Keyword	Object	ObjectType	value
Reset Login In Application	GOTOURL	username	name	url
	SETTEXT	password	name	Demo
	CLICK	resetButton	name	testPassword
Login In Application	GOTOURL	username	name	url
	SETTEXT	password	name	Demo
	CLICK	loginButton	name	testPassword

Excel Sheet For Keyword Driven Test

- According to the keywords written in Excel file, the framework will perform the operation on UI.
- For example, we need to click a button 'Login.' Correspondingly, our Excel will have a keyword 'Click.' Now the AUT can have hundreds of button on a page, to identify a Login button, in Excel we will input Object Name as loginButton & object type as a name (see highlighted the row in above image). The Object Type could be Xpath, name CSS or any other value

Step 3) ReadGuru99ExcelFile.java will pass this data to the driver script Execute.java

Step 4)

- For all of our UI web elements, we need to create an object repository where we will place their element locator (like Xpath, name, CSS path, class name etc.)



- Execute.java (our driver script) will read the entire Object Repository and store it in a variable
- To read this object repository, we need a ReadObject class which has a getObjectRepository method to read it.

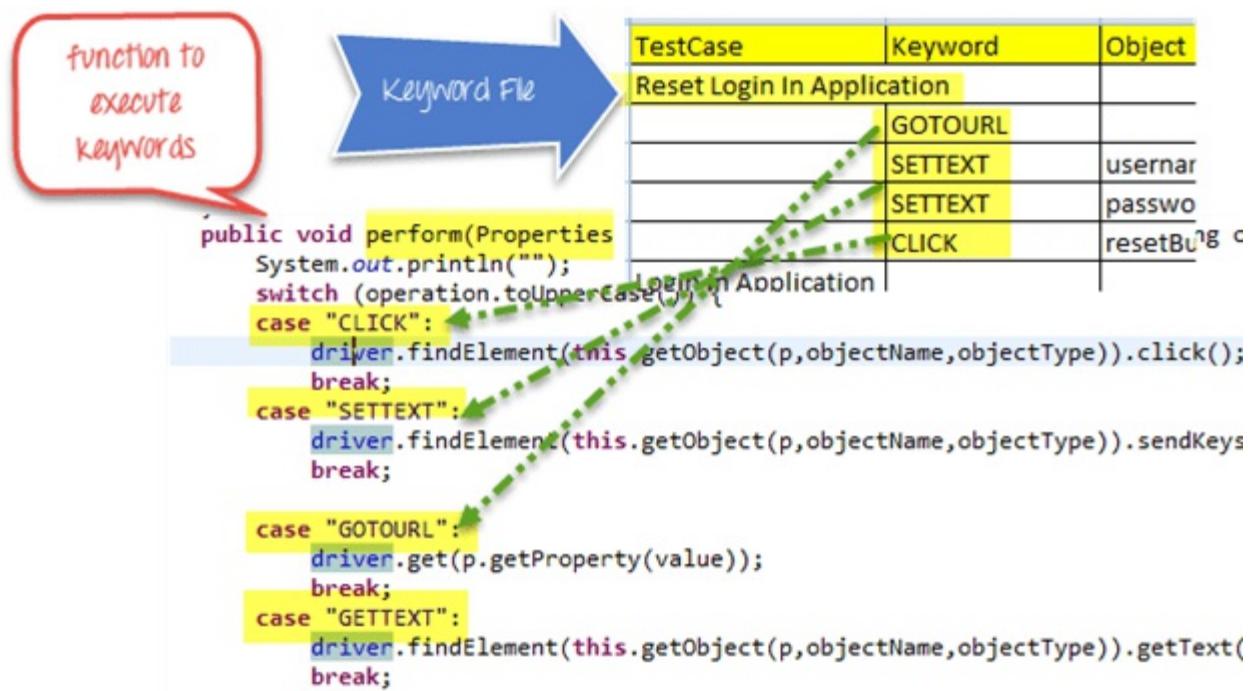
```
public class ReadObject {
    Properties p = new Properties();
    public Properties getObjectRepository() throws
        //Read object repository file
        InputStream stream = new FileInputStream(ne
        //load all objects|
        p.load(stream);
        return p;
}
```

This class will be used to read object repository file

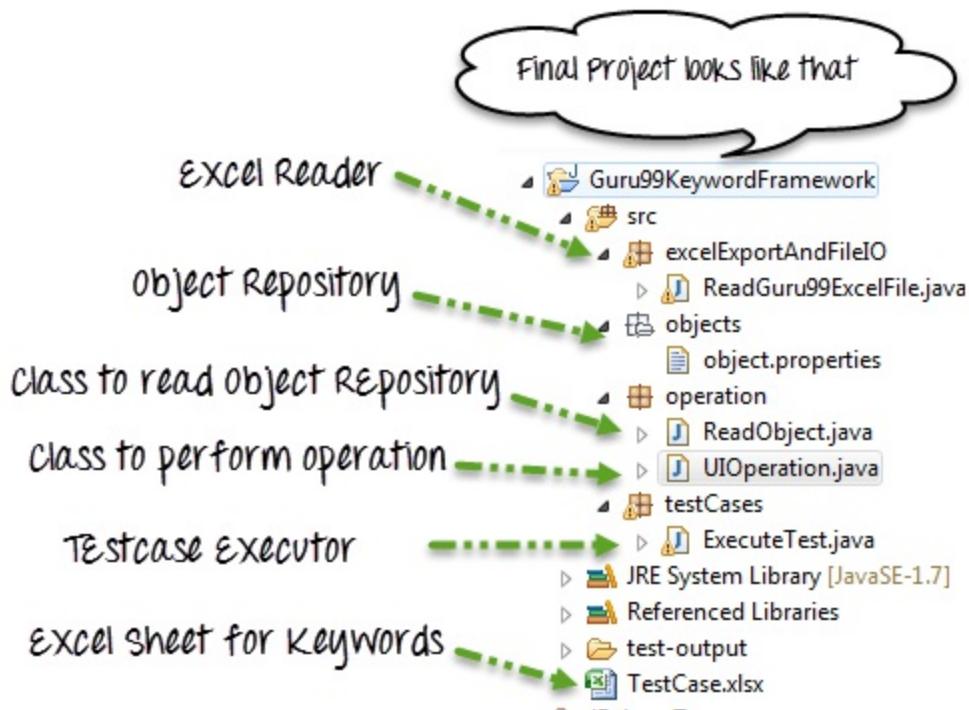
NOTE: You may think why do we need to create an object repository. The answer helps in code maintenance. For example, we are using the button with name = btnlogin in 10 different test cases. In future, the developer decides to change the name from btnlogin to submit. You will have to make a change in all the 10 test cases. In the case of an object repository, you will make the change just once in the repository.

Step 5)

- The driver will pass the data from Excel & Object Repository to UIOperation class
- UIOperation class has functions to perform actions corresponding to keywords like CLICK, SETTEXT etc... mentioned in the excel
- UIOperation class is a Java class which has the actual implementation of the code to perform operations on web elements



The complete project will look like-



Let's look into an example:

Test Scenario

- We are executing 2 test cases
- Test Case 1:
 - Goto <http://demo.guru99.com/V4/>
 - Enter User ID
 - Enter Password
 - Click Reset
- Test Case 2:
 - Goto <http://demo.guru99.com/V4/>
 - Enter User ID
 - Enter Password
 - Click Login

object.properties

url=http://demo.guru99.com/V4/

username=uid

password=password

title=barone

loginButton=btnLogin

resetButton=btnReset

ReadGuru99ExcelFile.java

```
package excelExportAndFileIO;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
public class ReadGuru99ExcelFile {

    public Sheet readExcel(String filePath, String
fileName, String sheetName) throws IOException{
        //Create a object of File class to open xlsx file
        File file = new File(filePath+"\\"+fileName);
        //Create an object of FileInputStream class to read excel
file
        FileInputStream inputStream = new FileInputStream(file);
        Workbook guru99Workbook = null;
        //Find the file extension by splitting file name in substring
and getting only extension name
        String fileExtensionName =
fileName.substring(fileName.indexOf("."));
        //Check condition if the file is xlsx file
        if(fileExtensionName.equals(".xlsx")){
            //If it is xlsx file then create object of XSSFWorkbook
```

```

class
    guru99Workbook = new XSSFWorkbook(inputStream);
}
//Check condition if the file is xls file
else if(fileExtensionName.equals(".xls")){
    //If it is xls file then create object of XSSFWorkbook
class
    guru99Workbook = new HSSFWorkbook(inputStream);
}
//Read sheet inside the workbook by its name
Sheet guru99Sheet = guru99Workbook.getSheet(sheetName);
return guru99Sheet;
}
}

```

ReadObject.java

```

package operation;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;
public class ReadObject {
    Properties p = new Properties();
    public Properties getObjectRepository() throws IOException{
        //Read object repository file
        InputStream stream = new FileInputStream(new
File(System.getProperty("user.dir")+"\\src\\objects\\object.properties"));
        //load all objects
        p.load(stream);
        return p;
    }
}

```

UIOperation.java

```
package operation;
```

```
import java.util.Properties;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
public class UIOperation {
    WebDriver driver;
    public UIOperation(WebDriver driver){
        this.driver = driver;
    }
    public void perform(Properties p,String operation,String
objectName,String objectType,String value) throws Exception{
        System.out.println("");
        switch (operation.toUpperCase()) {
            case "CLICK":
                //Perform click
                driver.findElement(this.getObject(p,objectName,objectType)).click();
                break;
            case "SETTEXT":
                //Set text on control
                driver.findElement(this.getObject(p,objectName,objectType)).sendKeys(value);
                break;

            case "GOTOURL":
                //Get url of application
                driver.get(p.getProperty(value));
                break;
            case "GETTEXT":
                //Get text of an element
                driver.findElement(this.getObject(p,objectName,objectType)).getText();
                break;
            default:
                break;
        }
    }
    /**
     * Find element BY using object type and value
     * @param p
```

```
* @param objectName
* @param objectType
* @return
* @throws Exception
*/
private By getObject(Properties p, String objectName, String
objectType) throws Exception{
    //Find by xpath
    if(objectType.equalsIgnoreCase("XPATH")){
        return By.xpath(p.getProperty(objectName));
    }
    //find by class
    else if(objectType.equalsIgnoreCase("CLASSNAME")){
        return By.className(p.getProperty(objectName));
    }
    //find by name
    else if(objectType.equalsIgnoreCase("NAME")){
        return By.name(p.getProperty(objectName));
    }
    //Find by css
    else if(objectType.equalsIgnoreCase("CSS")){
        return By.cssSelector(p.getProperty(objectName));
    }
    //find by link
    else if(objectType.equalsIgnoreCase("LINK")){
        return By.linkText(p.getProperty(objectName));
    }
    //find by partial link
    else if(objectType.equalsIgnoreCase("PARTIALLINK")){
        return
By.partialLinkText(p.getProperty(objectName));
    }
}
```

```
        {
            throw new Exception("Wrong object type");
        }
    }
}
```

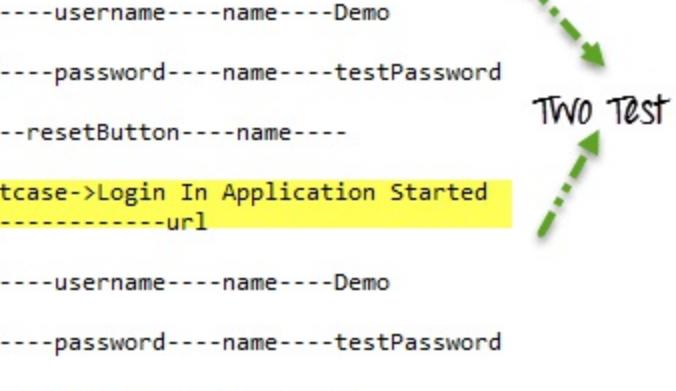
ExecuteTest.java

```
package testCases;
import java.util.Properties;
import operation.ReadObject;
import operation.UIOperation;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.Test;
import excelExportAndFileIO.ReadGuru99ExcelFile;
public class ExecuteTest {
    @Test
        public void testLogin() throws Exception {
            // TODO Auto-generated method stub
    WebDriver webdriver = new FirefoxDriver();
    ReadGuru99ExcelFile file = new ReadGuru99ExcelFile();
    ReadObject object = new ReadObject();
    Properties allObjects = object.getObjectRepository();
    UIOperation operation = new UIOperation(webdriver);
    //Read keyword sheet
    Sheet guru99Sheet =
    file.readExcel(System.getProperty("user.dir")+"\\\" , "TestCase.xls
x" , "KeywordFramework");
    //Find number of rows in excel file
    int rowCount = guru99Sheet.getLastRowNum()-guru99Sheet.getFirstRowNum();
    //Create a loop over all the rows of excel file to read it
    for (int i = 1; i < rowCount+1; i++) {
        //Loop over all the rows
        Row row = guru99Sheet.getRow(i);
        //Check if the first cell contain a value, if yes, That
means it is the new testcase name
        if(row.getCell(0).toString().length()==0){
```

```
//Print testcase detail on console
        System.out.println(row.getCell(1).toString()+"----"+
row.getCell(2).toString()+"----"+
                row.getCell(3).toString()+"----"+
row.getCell(4).toString());
        //Call perform function to perform operation on UI
        operation.perform(allObjects,
row.getCell(1).toString(), row.getCell(2).toString(),
                row.getCell(3).toString(),
row.getCell(4).toString());
    }
    else{
        //Print the new testcase name when it started
        System.out.println("New Testcase-
>"+row.getCell(0).toString() +" Started");
    }
}
}
```

After execution, output will look like -

```
New Testcase->Reset Login In Application Started  
GOTOURL-----url  
  
SETTEXT----username----name----Demo  
  
SETTEXT----password----name----testPassword  
  
CLICK----resetButton----name----  
  
New Testcase->Login In Application Started  
GOTOURL-----url  
  
SETTEXT----username----name----Demo  
  
SETTEXT----password----name----testPassword  
  
CLICK----loginButton----name----  
  
PASSED: testLogin
```



Hybrid Test Framework

Hybrid Test framework is a concept where we are using the advantage

of both Keyword and Data driven framework.

Here for keywords, we will use Excel files to maintain test cases, and for test data, we can use data provider of Testng framework.

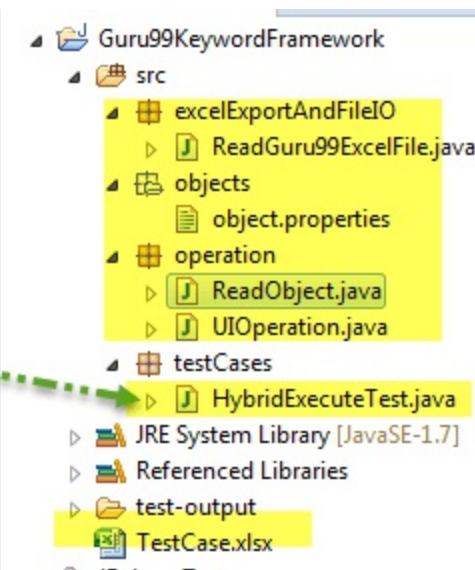
```

    @DataProvider(name="hybridData")
    public Object[][] getDataFromDataprovider() throws
    Object[][] object = null;
    ReadGuru99ExcelFile file = new ReadGuru99ExcelFile();
    //Read keyword sheet
    Sheet sheet = file.getSheet("Keywords");
    List<List<String>> l;
    @Test(dataProvider="hybridData")
    public void testLogin(String testcaseName, String keyword)
        // TODO Auto-generated method stub
        if(testcaseName!=null&&testcaseName.length()!=0){
            webdriver=new FirefoxDriver();
        }
        ReadObject object = new ReadObject();
        Properties allObjects = object.getObjectRepository();
        UIOperation operation = new UIOperation(webdriver);
        //Call perform function to perform operation on UI
        operation.perform(allObjects, keyword, obj
        objectType, value);
    
```

This test case is same as keyword driven , but we also have a dataprovider

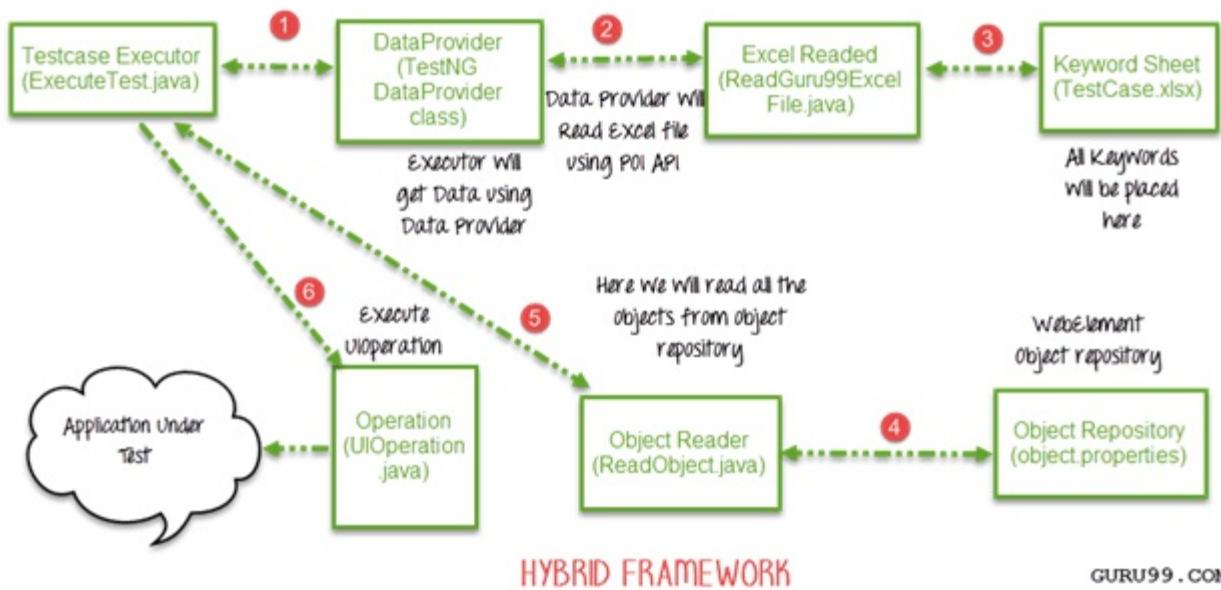
Here in our hybrid framework, we don't need to change anything in Keyword driven framework, here we just need to replace ExecuteTest.java file with HybridExecuteTest.java file.

only Testcase is changed in hybrid test case , all other thing will work without any changes



This HybridExecuteTest file has all the code for keyword driven with data provider concept.

The complete pictorial representation of hybrid framework will look like



HybridExecuteTest.java

```
package testCases;
import java.io.IOException;
import java.util.Properties;
import operation.ReadObject;
import operation.UIOperation;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;
import excelExportAndFileIO.ReadGuru99ExcelFile;
public class HybridExecuteTest {
    WebDriver webdriver = null;
    @Test(dataProvider="hybridData")
    public void testLogin(String testcaseName, String
```

```
keyword, String objectName, String objectType, String value) throws
Exception {
    // TODO Auto-generated method stub

    if(testcaseName!=null&&testcaseName.length()!=0){
        webdriver=new FirefoxDriver();
    }
    ReadObject object = new ReadObject();
    Properties allObjects = object.getObjectRepository();
    UIOperation operation = new UIOperation(webdriver);
    //Call perform function to perform operation on UI
    operation.perform(allObjects, keyword, objectName,
        objectType, value);

}
@DataProvider(name="hybridData")
public Object[][] getDataFromDataprovider() throws
IOException{
    Object[][] object = null;
    ReadGuru99ExcelFile file = new ReadGuru99ExcelFile();
//Read keyword sheet
Sheet guru99Sheet =
file.readExcel(System.getProperty("user.dir")+"\\\" , "TestCase.xls"
x", "KeywordFramework");
//Find number of rows in excel file
    int rowCount = guru99Sheet.getLastRowNum()-guru99Sheet.getFirstRowNum();
    object = new Object[rowCount][5];
    for (int i = 0; i < rowCount; i++) {
        //Loop over all the rows
        Row row = guru99Sheet.getRow(i+1);
        //Create a loop to print cell values in a row
        for (int j = 0; j < row.getLastCellNum(); j++) {
            //Print excel data in console
            object[i][j] = row.getCell(j).toString();
        }
    }
    System.out.println("");
    return object;
}
}
```

Summary:

- We can create three types of test framework using Selenium WebDriver.
- These are Data Driven, Keyword Driven, and Hybrid test framework.
- We can achieve Data-driven framework using TestNG's data provider.
- In Keyword driven framework, keywords are written in some external files like excel file and java code will call this file and execute test cases.
- The hybrid framework is a mix of keyword driven and data driven framework.

Chapter 29: Database Testing using Selenium: Step by Step Guide

Selenium Webdriver is limited to Testing your applications using Browser. To use Selenium Webdriver for Database Verification you need to use the JDBC ("Java Database Connectivity").

JDBC (Java Database Connectivity) is a SQL level API that allows you to execute SQL statements. It is responsible for the connectivity between the Java Programming language and a wide range of databases. The JDBC API provides the following classes and interfaces

- Driver Manager
- Driver
- Connection
- Statement
- ResultSet
- SQLException

In order to test your Database using Selenium, you need to observe the following 3 steps

1. Make a connection to the Database
2. Send Queries to the Database
3. Process the results

Make a connection to the Database

Send Queries to the Database

Process the results

1) Make a connection to the Database

In order to make a connection to the database the syntax is

```
DriverManager.getConnection(URL, "userid", "password")
```

Here,

- Userid is the username configured in the database
- Password of the configured user
- URL is of format jdbc:<dbtype>://ipaddress:portnumber/db_name"
- <dbtype>- The driver for the database you are trying to connect. To connect to oracle database this value will be "oracle"

For connecting to database with name "emp" in MYSQL URL will

bejdbc:mysql://localhost:3036/emp

And the code to create connection looks like

```
Connection con =  
DriverManager.getConnection(dbUrl,username,password);
```

You also need to load the JDBC Driver using the code

```
Class.forName("com.mysql.jdbc.Driver");
```

2) Send Queries to the Database

Once connection is made, you need to execute queries.

You can use the Statement Object to send queries.

```
Statement stmt = con.createStatement();
```

Once the statement object is created use the executeQuery method to execute the SQL queries

```
stmt.executeQuery(select * from employee);
```

3) Process the results

Results from the executed query are stored in the ResultSet Object.

Java provides loads of advance methods to process the results. Few of the methods are listed below

Method name	Description
String getString()	Method is used to fetch the string type data from the result set
int getInt()	Method is used to fetch the integer type data from the result set
double getDouble()	Method is used to fetch the double type data from the result set
Date getDate()	Method is used to fetch the Date type object from the result set
boolean next()	Method is used to move to the next record in the result set
boolean previous()	Method is used to move to the previous record in the result set
boolean first()	Method is used to move to the first record in the result set
boolean last()	Method is used to move to the last record in the result set
boolean absolute(int rowNumber)	Method is used to move to the specific record in the result set

Example of Database Testing with Selenium

Step 1) Install MySQL Server and MySQL Workbench

Check out the complete guide to Mysql & MySQL Workbench here

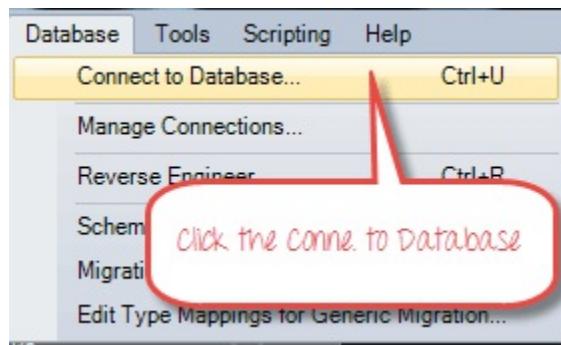
While installing MySQL Server, please note the database

- Username
- Password
- Port Number

It will be required in further steps.

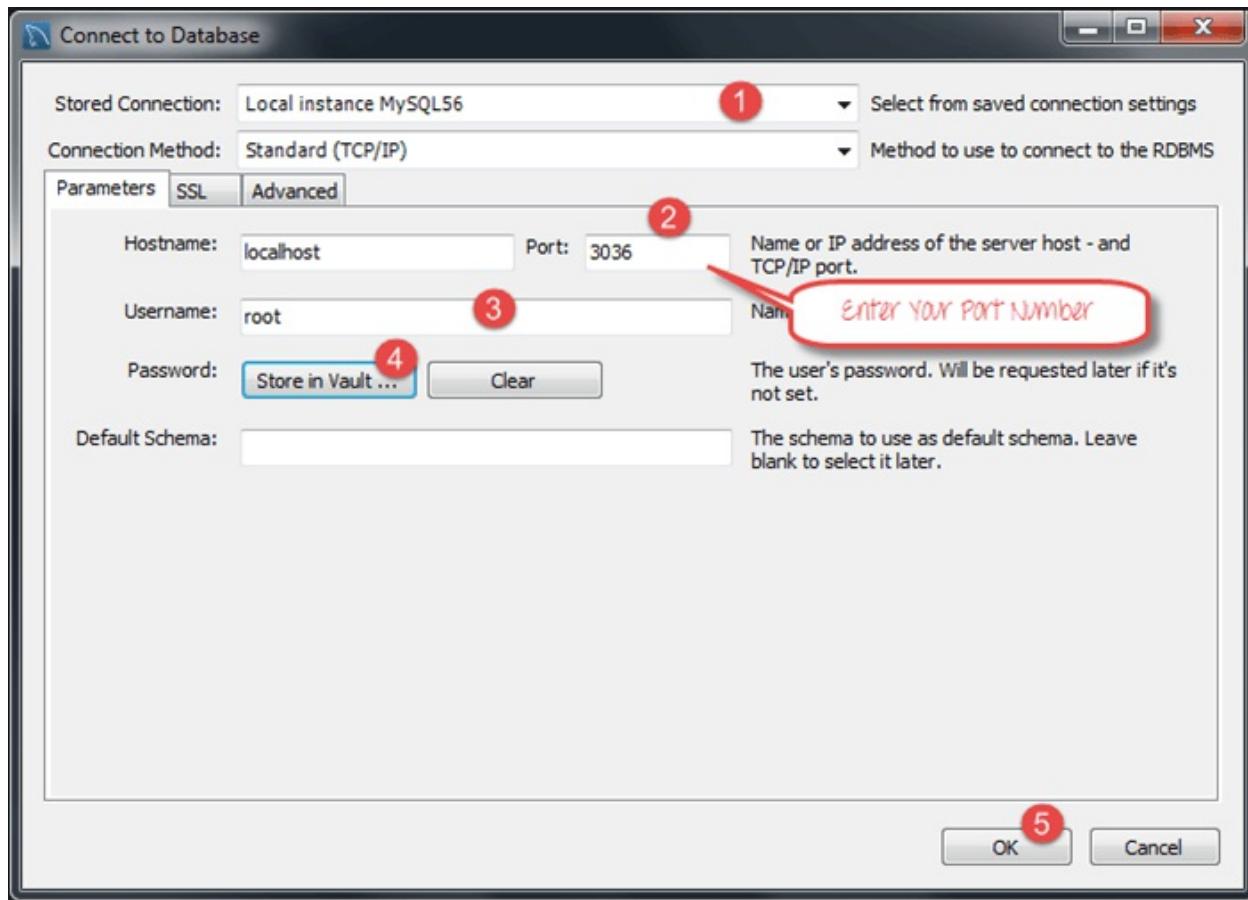
MySQL Workbench makes it easy to administer the database without the need to code SQL. Though, you can also use the MySQL Terminal to interact with the database.

Step 2) In MySQL WorkBench, connect to your MySQL Server



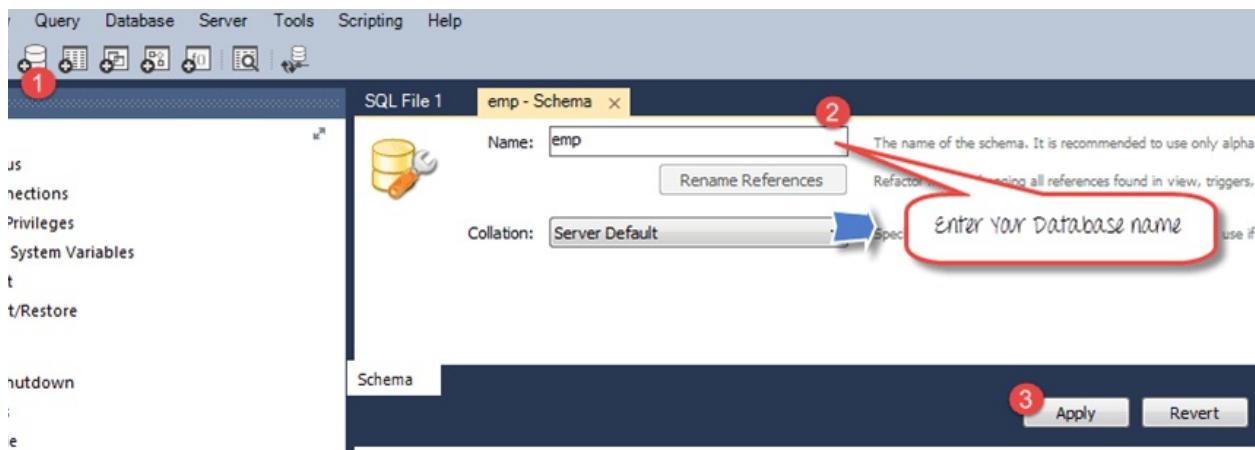
In the next screen,

1. Select Local Instance of MySQL
2. Enter Port Number
3. Enter Username
4. Enter Password
5. Click OK



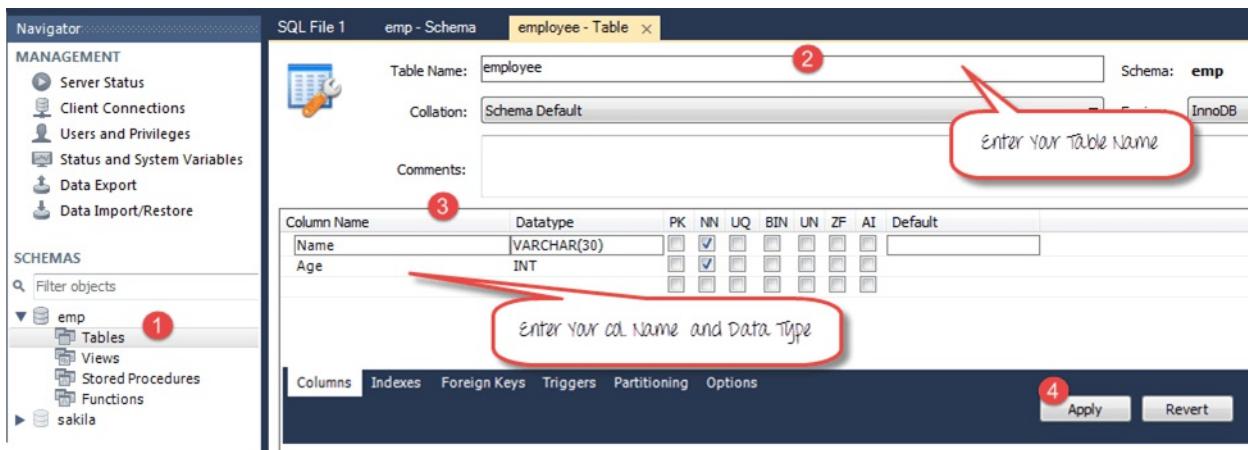
Step 3) To Create Database,

1. Click create Schema Button
2. Enter Name of Schema/Database
3. Click Apply

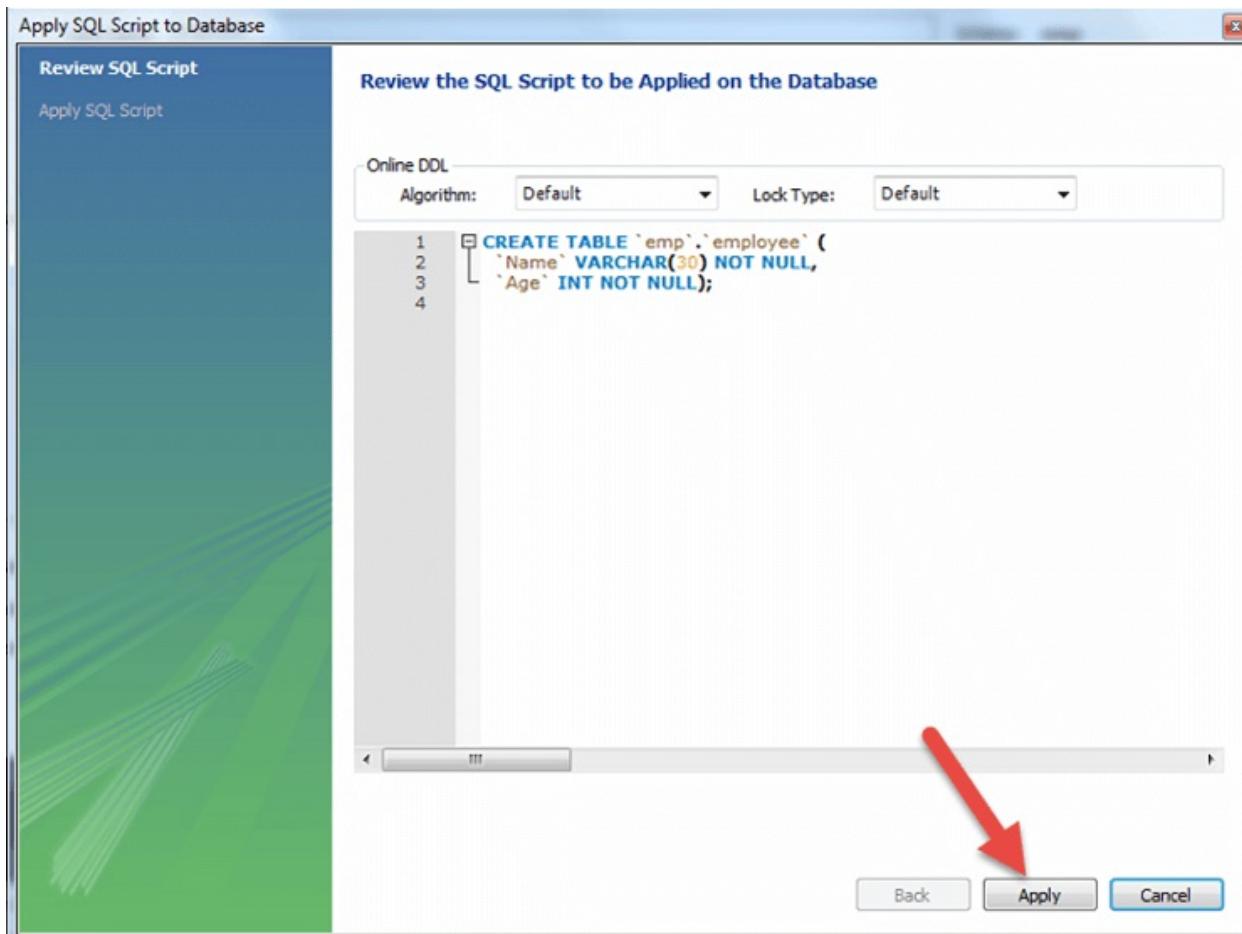


Step 4) In the navigator menu,

1. Click on Tables, beneath the emp database
2. Enter Table name as employee
3. Enter Fields as Name and Age
4. Click Apply



You will see the following pop-up. Click Apply



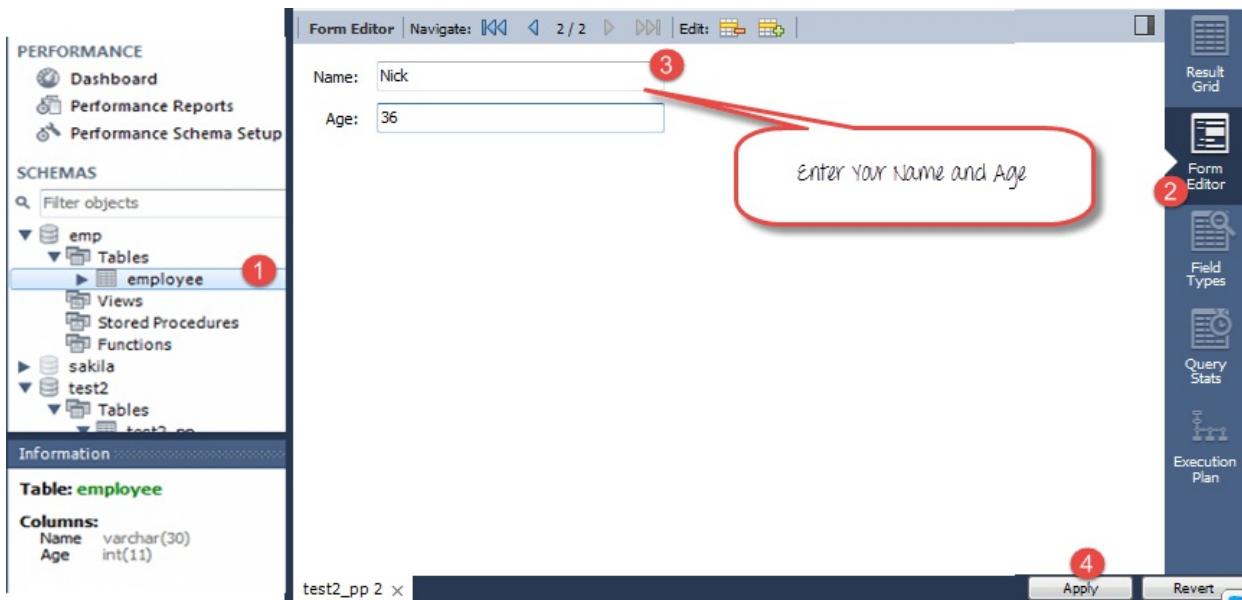
Step 5) We will create following data

Name	Age
Top	25
Nick	36
Bill	47

To create data into the Table

1. In navigator, select the employee table

2. In right pane, click Form Editor
3. Enter Name and Age
4. Click Apply



Repeat the process until all data is created

```

1 •   SELECT `employee`.`Name`,
2           `employee`.`Age`
3   FROM `emp`.`employee`;
4
5

```

	Name	Age
Top	25	
Nick	Nick	36
Bill	Bill	47

Step 6) Download the MySQL JDBC connector here

The screenshot shows a GitHub project page for 'find-ur-pal'. The 'Downloads' tab is selected. A red arrow points to the title 'the driver for connecting MySQL to Java'. Below it, the file 'mysql-connector-java-5.1.18-bin.jar' is listed with a download count of 771 KB. To the right of the file link is a detailed description of how to add it to an Eclipse project.

Download: the driver for connecting MySQL to Java
7 people starred this download

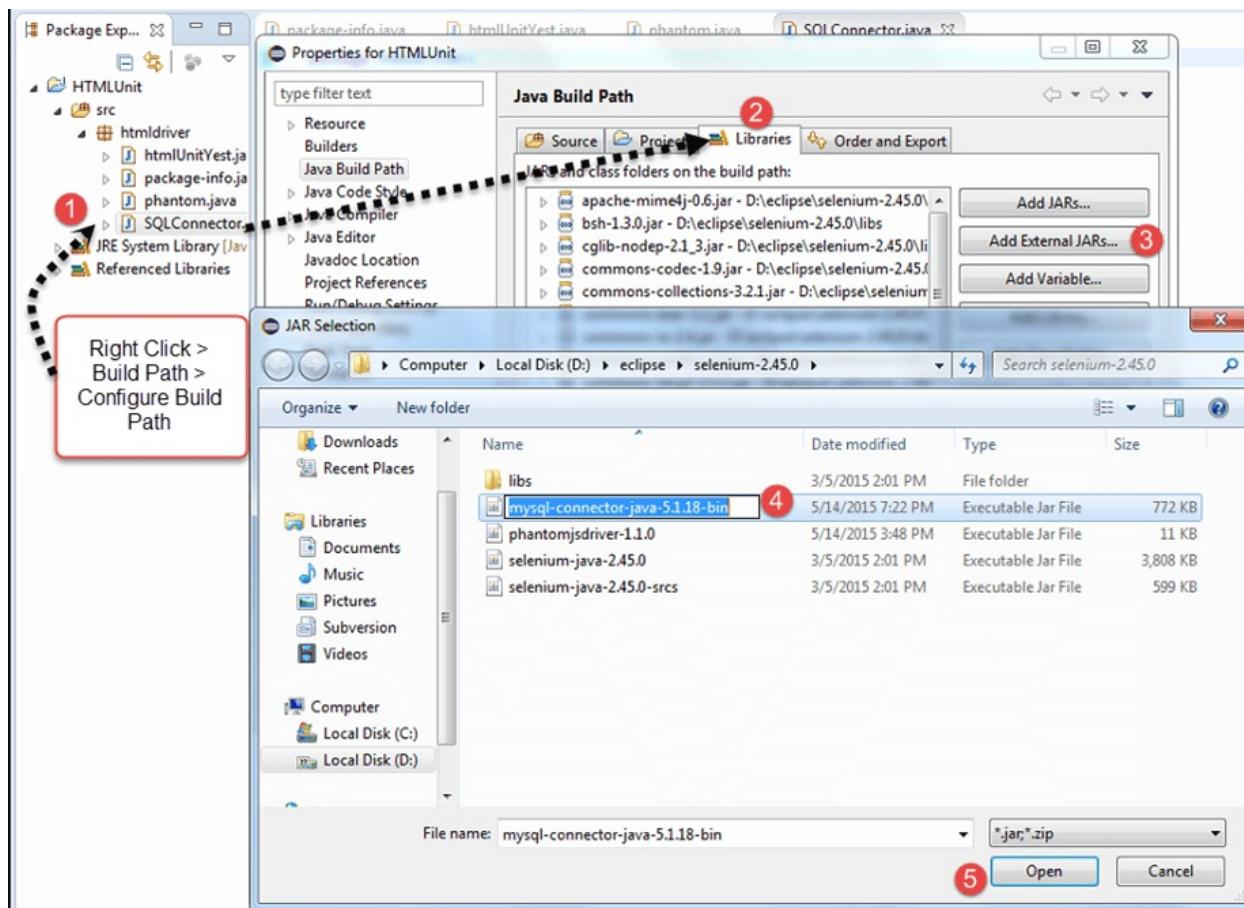
File: mysql-connector-java-5.1.18-bin.jar 771 KB

Description: This is a jar file which needs to be copied to the classpath. These are the steps to follow. Right-click on the project name in eclipse. Click on Build Path -> Configure Build Path. Choose Library Add External Jar Browse and say Open Open You're Done. Your program can now access the Jar file.

SHA1 Checksum: 85dfedad243dc0303ad7ae3a323c39421d220690 [What's this?](#)

Step 7) Add the downloaded Jar to your Project

1. Right click on your Java File. Then click on Build Pathà Configure build path
2. Select the libraries
3. Click on add external JARs
4. You can see MySQL connector java in your library
5. Click on open to add it to the project



Step 8) Copy the following code into the editor

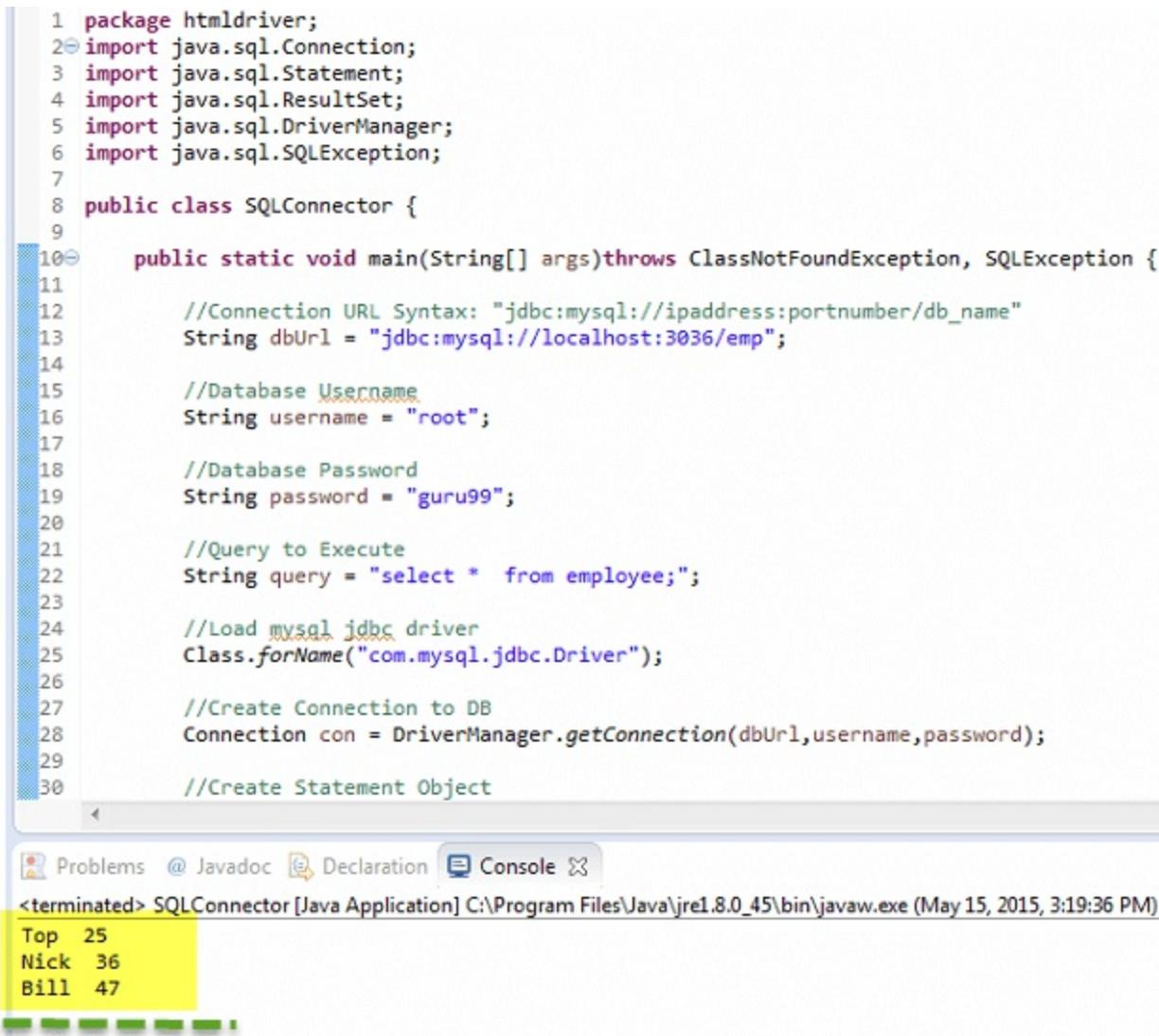
```

Package htmldriver;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.DriverManager;
import java.sql.SQLException;
public class SQLConnector {
    public static void main(String[] args) throws
ClassNotFoundException, SQLException {
        //Connection URL Syntax:
"jdbc:mysql://ipaddress:portnumber/db_name"
        String dbUrl =
"jdbc:mysql://localhost:3036/emp";
        //Database Username
        String username = "root";
    }
}

```

```
//Database Password  
String password = "guru99";  
  
//Query to Execute  
String query = "select * from  
employee;";  
  
//Load mysql jdbc driver  
Class.forName("com.mysql.jdbc.Driver");  
  
//Create Connection to DB  
Connection con =  
DriverManager.getConnection(dbUrl,username,password);  
  
//Create Statement Object  
Statement stmt = con.createStatement();  
  
// Execute the SQL Query. Store results  
in ResultSet  
ResultSet rs= stmt.executeQuery(query);  
  
// While Loop to iterate through all  
data and print results  
while (rs.next()) {  
    String myName =  
    rs.getString(1);  
    String myAge = rs.getString(2);  
    System.out.println(myName+ "  
"+myAge);  
}  
// closing DB Connection  
con.close();  
}
```

Step 8) Execute the code, and check the output



```

1 package htmldriver;
2 import java.sql.Connection;
3 import java.sql.Statement;
4 import java.sql.ResultSet;
5 import java.sql.DriverManager;
6 import java.sql.SQLException;
7
8 public class SQLConnector {
9
10    public static void main(String[] args) throws ClassNotFoundException, SQLException {
11
12        //Connection URL Syntax: "jdbc:mysql://ipaddress:portnumber/db_name"
13        String dbUrl = "jdbc:mysql://localhost:3036/emp";
14
15        //Database Username
16        String username = "root";
17
18        //Database Password
19        String password = "guru99";
20
21        //Query to Execute
22        String query = "select * from employee;";
23
24        //Load mysql jdbc driver
25        Class.forName("com.mysql.jdbc.Driver");
26
27        //Create Connection to DB
28        Connection con = DriverManager.getConnection(dbUrl,username,password);
29
30        //Create Statement Object

```

The screenshot shows an IDE interface with the following details:

- Code Area:** Displays the Java code for the `SQLConnector` class.
- Console Tab:** Shows the output of the `main` method execution.
- Output Content:**

```

Problems @ Javadoc Declaration Console X
<terminated> SQLConnector [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (May 15, 2015, 3:19:36 PM)
Top 25
Nick 36
Bill 47

```

Summary of Steps for Selenium Database Testing

Step 1) Make a connection to the Database using method.

`DriverManager.getConnection(URL, "userid", "password")`

Step 2) Create Query to the Database using the Statement Object.

`Statement stmt = con.createStatement();`

Step 3) Send the query to database using execute query and store the results in the ResultSet object.

```
ResultSet rs = stmt.executeQuery("select * from employee;");
```

Java provides lots of built-in methods to process the SQL Output using the ResultSet Object

Chapter 30: Handling Iframes in Selenium

What is Iframe?

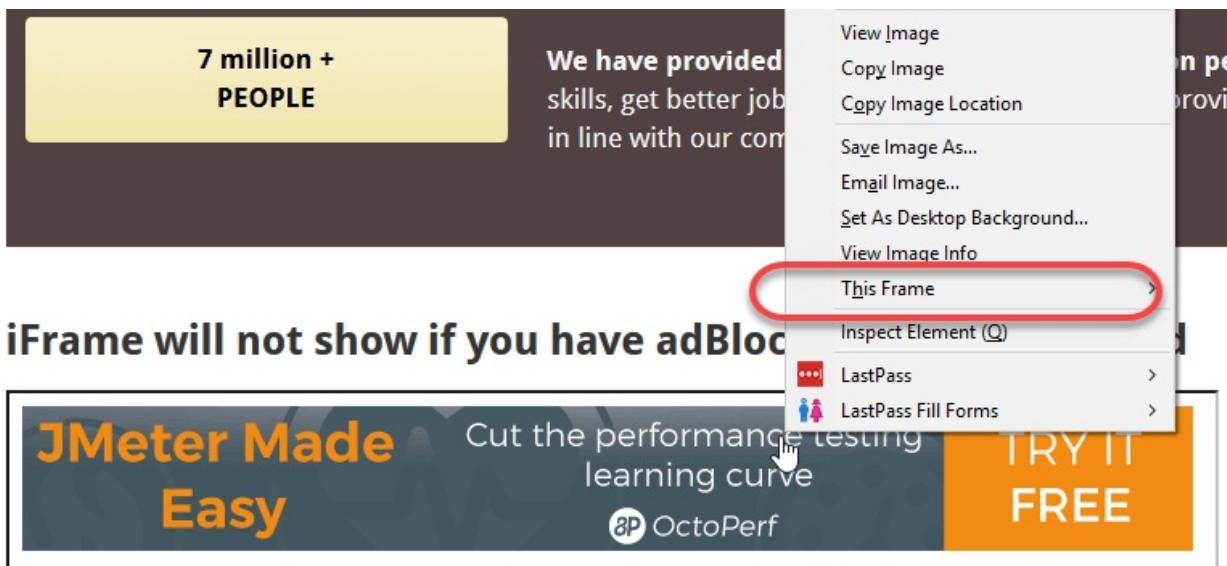
IFrame is a web page which is embedded in another web page or an HTML document embedded inside another HTML document.

The IFrame is often used to insert content from another source, such as an advertisement, into a Web page. The <**iframe**> tag specifies an inline frame.

How to identify the iframe:

We cannot detect the frames by just seeing the page or by inspecting Firebug.

Observe the below image, Advertisement being displayed is an Iframe, we cannot locate or recognize that by just inspecting using Firebug. So the question is how can you identify the iframe?



We can identify the iframes using methods given below:

- Right click on the element, If you find the option like 'This Frame' then it is an iframe.(Please refer the above diagram)
- Right click on the page and click 'View Page Source' and Search with the 'iframe', if you can find any tag name with the 'iframe' then it is meaning to say the page consisting an iframe.

In above diagram, you can see that '**This Frame**' option is available upon right clicking, so we are now sure that it is an iframe.

We can even identify total number of iframes by using below snippet.

```
Int size = driver.findElements(By.tagName("iframe")).size();
```

How to switch over the elements in iframes using Web Driver commands:

Basically, we can switch over the elements in frames using 3 ways.

- **By Index**
- **By Name or Id**
- **By Web Element**

Switch to the frame by index:

Index is one of the attributes for the Iframe through which we can switch to it.

Index of the iframe starts with '0'.

Suppose if there are 100 frames in page, we can switch to the iframe by using index.

- driver.switchTo().frame(0);
- driver.switchTo().frame(1);

Switch to the frame by Name or ID:

Name and ID are attributes of iframe through which we can switch to the it.

- driver.switchTo().frame("iframe1");
- driver.switchTo().frame("id of the element");

Example of Switching to iframe through ID:

Let's take an example of iframe displaying in the below image. Our requirement is to click the iframe.

We can access this iframe through this below
URL:<http://demo.guru99.com/test/guru99home/>



It is impossible to click iframe directly through XPath since it is an iframe. First we have to switch to the frame and then we can click using xpath.

Step 1)

```
WebDriver driver = new FirefoxDriver();

driver.get("http://demo.guru99.com/test/guru99home/");
driver.manage().window().maximize();
```

- We initialise the Firefox driver.
- Navigate to the "guru99" site which consist the iframe.
- Maximized the window.

Step 2)

```
driver.switchTo().frame("a077aa5e");
```

- In this step we need to find out the id of the iframe by inspecting through Firebug.
- Then switch to the iframe through ID.

Step 3)

```
driver.findElement(By.xpath("html/body/a/img")).click();
```

- Here we need to find out the xpath of the element to be clicked.

- Click the element using web driver command shown above.

Here is the complete code:

```
public class SwitchToFrame_ID {  
    public static void main(String[] args) {  
  
        WebDriver driver = new FirefoxDriver();  
        //navigates to the Browser  
  
        driver.get("http://demo.guru99.com/test/guru99home/");  
        // navigates to the page consisting an iframe  
  
        driver.manage().window().maximize();  
        driver.switchTo().frame("a077aa5e"); //switching  
        the frame by ID  
  
        System.out.println("*****We are  
        switch to the iframe*****");  
  
        driver.findElement(By.xpath("html/body/a/img")).click();  
        //Clicks the iframe  
  
        System.out.println("*****We are  
        done*****");  
    }  
}
```

Output:

Browser navigates to the page consisting the above iframe and clicks on the iframe.

Switch to the frame by Web Element:

We can even switch to the iframe using web element .

- `driver.switchTo().frame(WebElement);`

How to switch back to the Main Frame

We have to come out of the iframe.

To move back to the parent frame, you can either use `switchTo().parentFrame()` or if you want to get back to the main (or most parent) frame, you can use `switchTo().defaultContent();`

```
driver.switchTo().parentFrame();
driver.switchTo().defaultContent();
```

How to switch over the frame, if we CANNOT switch using ID or Web Element:

Suppose if there are 100 frames in the page, and there is no ID available, in this case, we just don't know from which iframe required element is being loaded (It is the case when we do not know the index of the frame also).

The solution for the above concern is, we must find the index of the iframe through which the element is being loaded and then we need to switch to the iframe through the index.

Below are the steps for finding the index of the Frame by which the element is being loaded by using below snippet

Step 1)

```
WebDriver driver = new FirefoxDriver();
driver.get("http://demo.guru99.com/test/guru99home/");
driver.manage().window().maximize();
```

- Initialise the Firefox driver.
- Navigate to the "guru99" site which consisting the iframe.
- Maximized the window.

Step 2)

```
int size = driver.findElements(By.tagName("iframe")).size();
```

- The above code finds the total number of iframes present inside the page using the tagname 'iframe'.

Step 3)

Objective for this step would be finding out the index of iframe.

```
for(int i=0; i<=size; i++){
    driver.switchTo().frame(i);
    int
total=driver.findElements(By.xpath("html/body/a/img")).size();
    System.out.println(total);
    driver.switchTo().defaultContent();}
```

Above "forloop" iterates all the iframes in the page and it prints '1' if our required iframe was found else returns '0'.

Here is the complete code till step 3:

```
public class IndexOfIframe {
public static void main(String[] args) {
    WebDriver driver = new FirefoxDriver();

driver.get("http://demo.guru99.com/test/guru99home/");
    driver.manage().window().maximize();
    //driver.manage().timeouts().implicitlyWait(100,
TimeUnit.SECONDS);
    int size =
driver.findElements(By.tagName("iframe")).size();

    for(int i=0; i<=size; i++){
        driver.switchTo().frame(i);
        int
total=driver.findElements(By.xpath("html/body/a/img")).size();
```

```
        System.out.println(total);
        driver.switchTo().defaultContent();}}}
```

Execute this program and output would be like below:

Output:

1

0

0

0

0

0

Verify the output, you can find the series of 0's and 1's.

- Wherever you find the '1' in output that is the index of Frame by which the element is being loaded.
- Since the index of the iframe starts with '0' if you find the 1 in the 1st place, then the index is 0.
- If you find 1 in 3rd place, the index is 2.

We can comment out the for loop, once we found the index.

Step 4)

```
driver.switchTo().frame(0);
```

- Once you find the index of the element, you can switch over the frame using above command.

- driver.switchTo().frame(index found from the Step 3);

Step5)

```
driver.findElement(By.xpath("html/body/a/img")).click();
```

- The above code will clicks the iframe or element in the iframe.

So the complete code would be like below:

```
public class SwitchToframe {
public static void main(String[] args) throws
NoSuchElementException{
    WebDriver driver = new FirefoxDriver();

driver.get("http://demo.guru99.com/test/guru99home/");
    driver.manage().window().maximize();
    //int size =
driver.findElements(By.tagName("iframe")).size();

/*for(int i=0; i<=size; i++){
    driver.switchTo().frame(i);
    int
total=driver.findElements(By.xpath("html/body/a/img")).size();
    System.out.println(total);
    driver.switchTo().defaultContent(); //switching back
from the iframe
}*/

    //Commented the code for finding the index of
the element
    driver.switchTo().frame(0); //Switching to the frame
    System.out.println("*****We are switched to
the iframe*****");

driver.findElement(By.xpath("html/body/a/img")).click();

    //clicking the element in line with
Advertisement
    System.out.println("*****We are
done*****");
```

{
}**Output:**

Browser navigates to the page consisting the above iframe and clicks on the iframe.

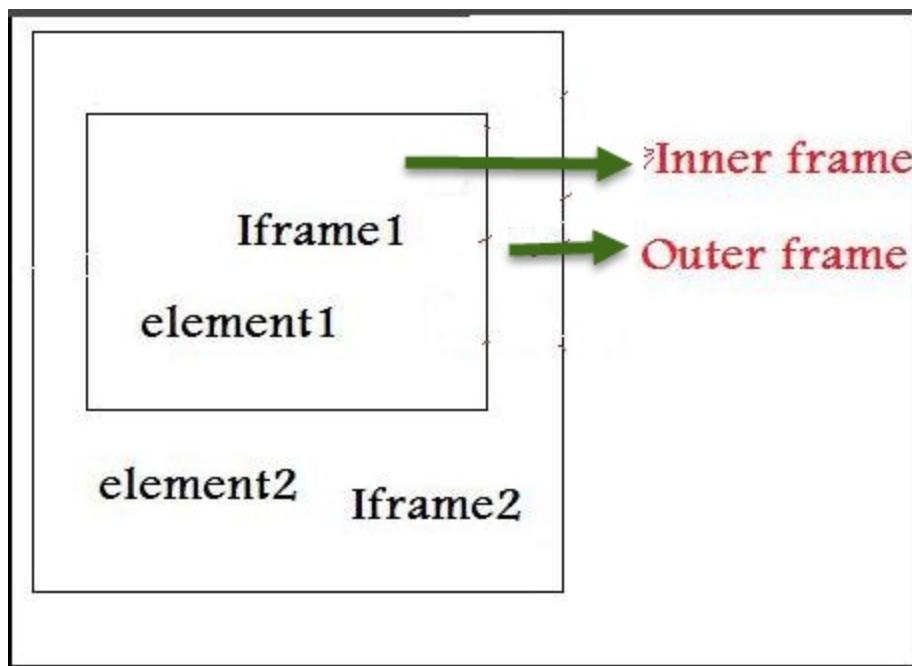
Concept of Nested Frames(Frames inside Frames):

Let's assume that there are two frames one inside other like shown in below image and our requirement is printing the text in the outer frame and inner frame.

In the case of nested frames,

- At first we must switch to the outer frame by either Index or ID of the iframe
- Once we switch to the outer frame we can find the total number of iframes inside the outer frame, and
- We can switch to the inner frame by any of the known methods.

While exiting out of the frame, we must exit out in the same order as we entered into it from the inner frame first and then outer frame.



The Html code for the above nested frame is as shown below.

```

<div class="iframe">
  <div style="overflow:auto;">
    <div id="iframewrapper" class="iframewrapper">
      <iframe id="iframeResult" frameborder="0">
        <!DOCTYPE html>
        <html>
          <head>
            <body>
              <iframe width="200" height="200" src="demo_iframe.htm">
            </body>
          </html>
    
```

The above HTML code clearly explains the iframe tag (highlighted in green) within another iframe tag, indicating presence of nested iframes.

Below are the steps for switching to outer frame and printing the text on outer frames:

Step 1)

```
WebDriver driver=new FirefoxDriver();
```

```

        driver.get("Url");
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(2,
TimeUnit.SECONDS);
        int size =
driver.findElements(By.tagName("iframe")).size();
        System.out.println("Total Frames --" + size);

        // prints the total number of frames
        driver.switchTo().frame(0); // Switching the
Outer Frame
        System.out.println
(driver.findElement(By.xpath("xpath of the outer element
")).getText());

```

- Switch to the outer Frame.
- Prints the text on outer frame.

Once we switch to the outer frame, we should know whether any inner frame present inside the outer frame

Step 2)

```

size = driver.findElements(By.tagName("iframe")).size();
// prints the total number of frames inside outer frame
System.out.println("Total Frames --" + size);

```

- Finds the total number of iframes inside outer frame.
- If size was found '0' then there is no inner frame inside the frame.

Step 3)

```

driver.switchTo().frame(0); // Switching to innerframe
System.out.println(driver.findElement(By.xpath("xpath of the
inner element ")).getText());

```

- Switch to the inner frame
- Prints the text on the inner frame.

Here is the complete code:

```

public class FramesInsideFrames {
public static void main(String[] args) {
WebDriver driver=new FirefoxDriver();
    driver.get("Url");
    driver.manage().window().maximize();
    driver.manage().timeouts().implicitlyWait(2,
TimeUnit.SECONDS);

        int size =
driver.findElements(By.tagName("iframe")).size();
        System.out.println("Total Frames --" + size);

            // prints the total number of frames
            driver.switchTo().frame(0); // Switching the
Outer Frame
            System.out.println
(driver.findElement(By.xpath("xpath of the outer element
")).getText());

            //Printing the text in outer frame
            size =
driver.findElements(By.tagName("iframe")).size();
            // prints the total number of frames inside outer
frame

            System.out.println("Total Frames --" + size);
            driver.switchTo().frame(0); // Switching to
innerframe

System.out.println(driver.findElement(By.xpath("xpath of the
inner element ")).getText());

            //Printing the text in inner frame
            driver.switchTo().defaultContent();
        }
}

```

Output:

The output of the above code would print the text in the Inner frame and Outer frame.

Chapter 31: Cross Browser Testing using Selenium

What is Cross Browser Testing?

Cross Browser Testing is a type of functional test to check that your web application works as expected in different browsers.

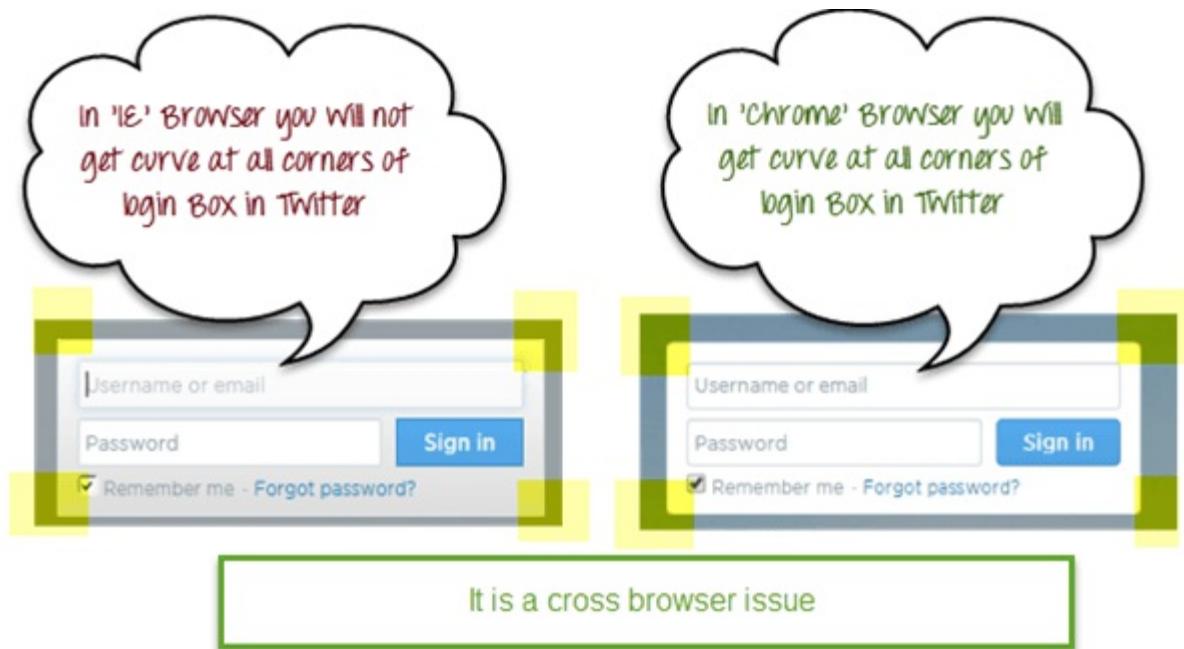


Why do we need Cross Browser Testing?

Web-based applications are totally different from Windows applications. A web application can be opened in any browser by the end user. For example, some people prefer to open <http://twitter.com> in **Firefox browser**, while others can be using

Chrome browser or IE.

In the diagram below you can observe that in **IE**, the login box of Twitter is not showing curve at all corners, but we are able to see it in Chrome browser.



So we need to ensure that the web application will work as expected in all popular browsers so that more people can access it and use it.

This motive can be fulfilled with Cross Browser Testing of the product.

Reason Cross Browser Issues

1. Font size mismatch in different browsers.
2. JavaScript implementation can be different.
3. CSS,HTML validation difference can be there.
4. Some browser still not supporting HTML5.
5. Page alignment and div size.
6. Image orientation.

7. Browser incompatibility with OS. Etc.

How to perform Cross Browser Testing

If we are using Selenium WebDriver, we can automate test cases using Internet Explorer, FireFox, Chrome, Safari browsers.

To execute test cases with different browsers in the same machine at the same time we can integrate Testng framework with Selenium WebDriver.

Your testing.xml will look like that,

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="TestSuite" thread-count="2" parallel="tests" >
  <test name="ChromeTest">
    <parameter name="browser" value="Chrome" />
    <classes>
      <class name="parallelTest.CrossBrowserScript">
      </class>
    </classes>
  </test>
  <test name="FirefoxTest">
    <parameter name="browser" value="Firefox" />
    <classes>
      <class name="parallelTest.CrossBrowserScript">
      </class>
    </classes>
  </test>
</suite>

```

Test will run parallel

1st Test name is 'ChromeTest'

Parameter is 'chrome'

2st Test name is 'FirefoxTest'

Parameter is 'Firefox'

This is the TestNGxml for cross Browser Testing

This testing.xml will map with the Test Case which will look like that

Parameter will be pass from testNG.xml

```

@BeforeTest
@Parameters("browser")
public void setup(String browser) throws Exception{
    if(browser.equalsIgnoreCase("firefox")){
        driver = new FirefoxDriver();
    }
    else if(browser.equalsIgnoreCase("chrome")){
        System.setProperty("webdriver.chrome.driver", ".\\chromedriver.exe");
        driver = new ChromeDriver();
    }
}

```

check parameter value
and create webDriver
according it

Here because the testing.xml has two Test tags ('ChromeTest','FirefoxTest'),this test case will execute two times for 2 different browsers.

First Test 'ChromeTest' will pass the value of parameter 'browser' as 'chrome' so ChromeDriver will be executed. This test case will run on Chrome browser.

Second Test 'FirefoxTest' will pass the value of parameter 'browser' as 'Firefox' so FirefoxDriver will be executed. This test case will run on FireFox browser.

Complete Code:

Guru99CrossBrowserScript.java

```

package parallelTest;

import java.util.concurrent.TimeUnit;
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.edge.EdgeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.BeforeTest;

```

```
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class CrossBrowserScript {

    WebDriver driver;

    /**
     * This function will execute before each Test tag in
     testng.xml
     * @param browser
     * @throws Exception
     */
    @BeforeTest
    @Parameters("browser")
    public void setup(String browser) throws Exception{
        //Check if parameter passed from TestNG is
'firefox'
        if(browser.equalsIgnoreCase("firefox")){
            //create firefox instance

System.setProperty("webdriver.firefox.marionette",
".\\geckodriver.exe");
            driver = new FirefoxDriver();
        }
        //Check if parameter passed as 'chrome'
        else if(browser.equalsIgnoreCase("chrome")){
            //set path to chromedriver.exe

System.setProperty("webdriver.chrome.driver",".\\chromedriver.ex
e");
            //create chrome instance
            driver = new ChromeDriver();
        }
        //Check if parameter passed as 'Edge'
        else
if(browser.equalsIgnoreCase("Edge")){
            //set path to Edge.exe

System.setProperty("webdriver.edge.driver",".\\MicrosoftWebDrive
r.exe");
            //create Edge instance
            driver = new
```

```

EdgeDriver();
        }
    else{
        //If no browser passed throw exception
        throw new Exception("Browser is not
correct");
    }
    driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
}

@Test
public void testParameterWithXML() throws
InterruptedException{
    driver.get("http://demo.guru99.com/V4/");
    //Find user name
    WebElement userName =
driver.findElement(By.name("uid"));
    //Fill user name
    userName.sendKeys("guru99");
    //Find password
    WebElement password =
driver.findElement(By.name("password"));
    //Fill password
    password.sendKeys("guru99");
}
}

```

testing.xml

```

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

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

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

<test name="ChromeTest">

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

<classes>

```

```
<class name="parallelTest.CrossBrowserScript">

</class>

</classes>

</test>

<test name="FirefoxTest">

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

<classes>

<class name="parallelTest.CrossBrowserScript">

</class>

</classes>

</test>

<test name="EdgeTest">

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

<classes>

<class name="parallelTest.CrossBrowserScript">

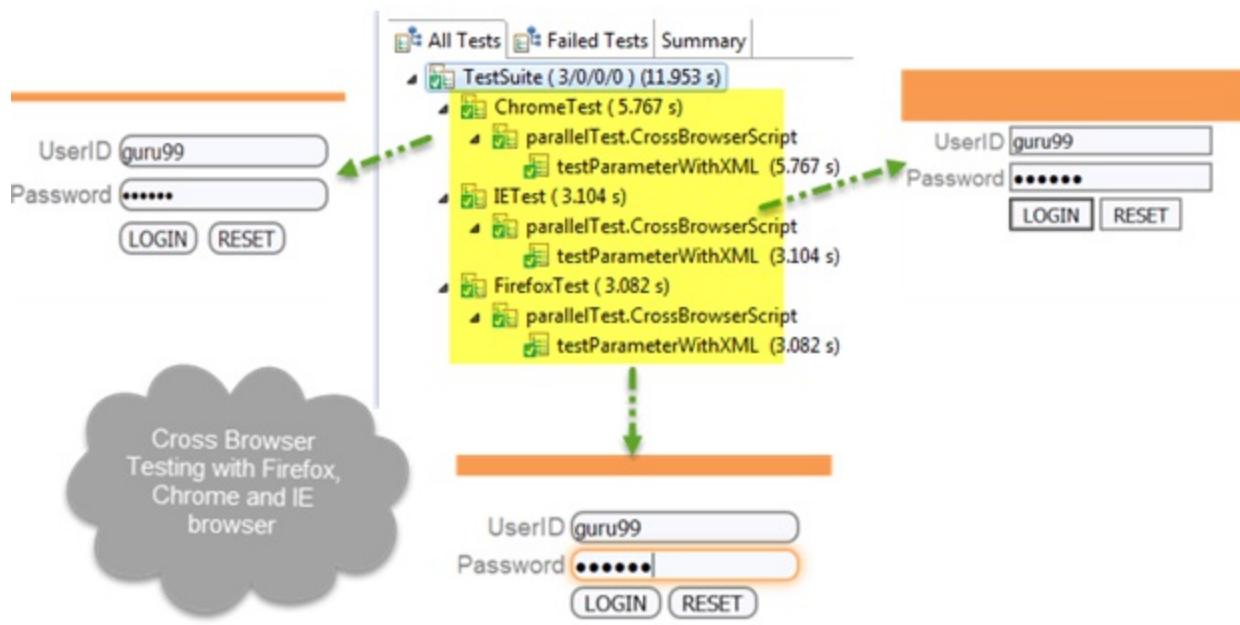
</class>

</classes>

</test>

</suite>
```

NOTE: To run the test, Right click on the **testing.xml**, Select Run As, and Click TestNG



Summary

1. Cross browser Testing is a technique to test web application with different web browsers.
2. Selenium can support different type of browsers for automation.
3. Selenium can be integrated with TestNG to perform Multi Browser Testing.
4. From parameters in testing.xml we can pass browser name, and in a test case, we can create WebDriver reference accordingly.

Note: The given program was built & tested on selenium 3.0.1, Chrome 56.0.2924.87 , Firefox 47.0.2 & Microsoft Edge 14.14393. If the programs give an error, please update the driver

Chapter 32: PDF , Emails and Screenshot of Test Reports in Selenium

Before we look into anything else, let's first understand -

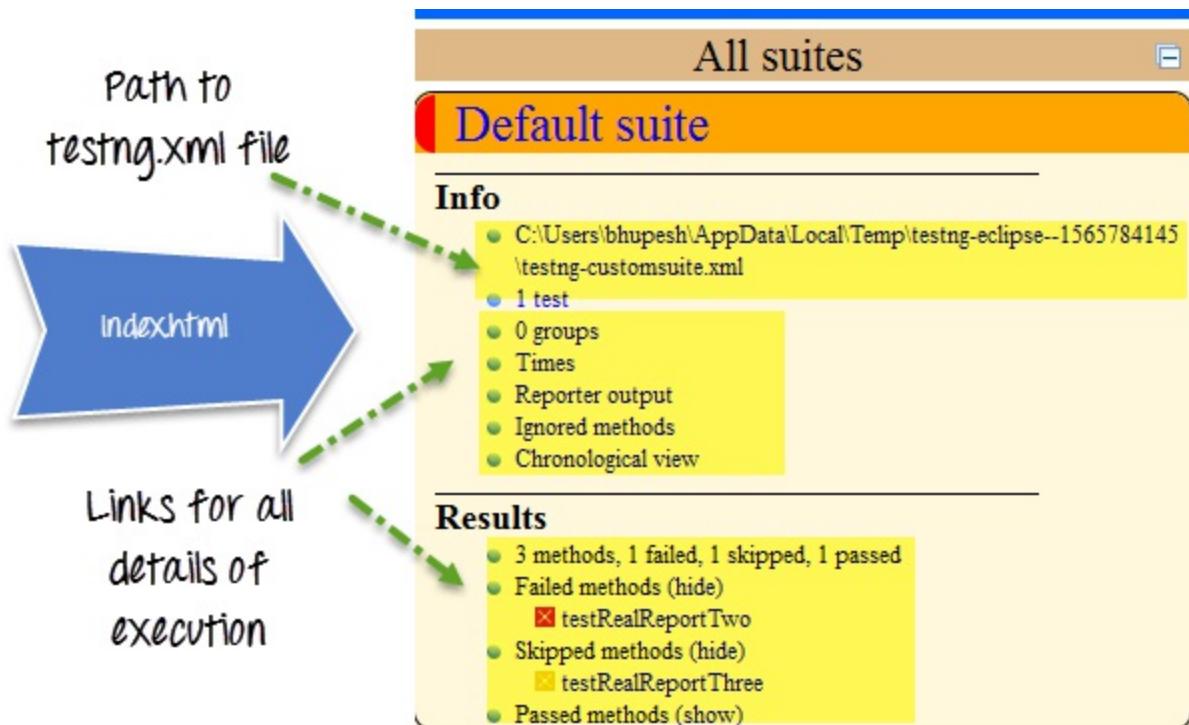
Why do we need reporting?

When we are using Selenium or any other automation tool, we are performing operations on the web application. But our purpose of automation is not just to exercise the Application Under Test. We, as an automation tester are supposed to test the application, find bugs and report it to the development team or higher management. Here the reporting gets importance for software Testing process

TestNG Reporting

TestNG library provides a very handy reporting feature. After execution, Testng will generate a test-output folder at the root of the project. This folder contains two type of Reports-

Index.html: This is the complete report of current execution which contains information like an error, groups, time, reporter logs, testng XML files.



emailable-report.html: This is the summarize report of the current test execution which contains Test Case message in green (for pass test cases) and red(for failed test cases) highlight.

Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups
Default suite					
Default test	1	1	1	37	

emailable-report.html

Test Detail in Report

How to customize TestNG Report

TestNG reporting is quite handy but still, sometimes we need some less data in reports or want to display reports in some other format

like pdf, excel, etc. or want to change report's layout.

There can be two ways we can customize TestNG report

- Using ITestListener Interface:
- Using IReporter Interface:



ITestListener Interface

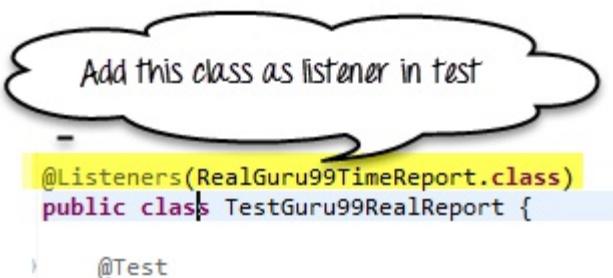
We use this interface when we need to customize real time report. In other words, if we are executing the bunch of test cases in a TestNG suite and we want to get the report of each test case, then after each test case we need to implement ITestListener interface. This interface will override onTestFailure, onTestStart

, onTestSkipped method to send the correct status of the current test case.



Here are the steps we will follow

- Create a class say RealGuru99Report and implement iTestListener in it.
- Implement methods of iTestListener
- Create test method and add RealGuru99Report class as a listener in Test Method class.



Code Example

RealGuru99TimeReport.java is the real time reporting class. It will implement ITestListener interface for reporting

```
package testNGReport.realTimeReport;

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

public class RealGuru99TimeReport implements ITestListener{
```

```
@Override  
  
public void onStart(ITestContext arg0) {  
  
    System.out.println("Start Of Execution(TEST)->" + arg0.getName());  
  
}  
  
@Override  
  
public void onTestStart(ITestResult arg0) {  
  
    System.out.println("Test Started->" + arg0.getName());  
  
}  
  
@Override  
  
public void onTestSuccess(ITestResult arg0) {  
  
    System.out.println("Test Pass->" + arg0.getName());  
  
}  
  
@Override  
  
public void onTestFailure(ITestResult arg0) {  
  
    System.out.println("Test Failed->" + arg0.getName());  
  
}  
  
@Override  
  
public void onTestSkipped(ITestResult arg0) {  
  
    System.out.println("Test Skipped->" + arg0.getName());  
  
}  
  
@Override
```

```
public void onFinish(ITestContext arg0) {  
  
System.out.println("END Of Execution(TEST)->"+arg0.getName());  
  
}  
  
@Override  
  
public void  
onTestFailedButWithinSuccessPercentage(ITestResult arg0) {  
  
// TODO Auto-generated method stub  
  
}  
  
}
```

TestGuru99RealReport.java is the test case for real report

```
package testNGReport.realTimeReport;  
  
import org.testng.Assert;  
  
import org.testng.annotations.Listeners;  
  
import org.testng.annotations.Test;  
  
@Listeners(RealGuru99TimeReport.class)  
  
public class TestGuru99RealReport {  
  
    @Test  
  
    public void testRealReportOne(){  
  
        Assert.assertTrue(true);  
    }  
}
```

```
}

@Test
public void testRealReportTwo(){
    Assert.assertTrue(false);
}

//Test case depends on failed testcase= testRealReportTwo
@Test(dependsOnMethods="testRealReportTwo")
public void testRealReportThree(){

}

}
```

The output will look like-

```

public class RealGuru99TimeReport implements ITestListener {
    @Override
    public void onStart(ITestContext arg0) {
        System.out.println("Start Of Execution(TEST)->" +
    }
    @Override
    public void onTestStart(ITestResult arg0) {
        System.out.println("Test Started->" + arg0.getName());
    }
    @Override
    public void onTestSuccess(ITestResult arg0) {
        System.out.println("Test Pass->" + arg0.getName());
    }
    @Override
    public void onTestFailure(ITestResult arg0) {
        System.out.println("Test Failed->" + arg0.getName());
    }

    @Override
    public void onTestSkipped(ITestResult arg0) {
        System.out.println("Test Skipped->" + arg0.getName());
    }

    @Override
    public void onFinish(ITestContext arg0) {
        System.out.println("END Of Execution(TEST)->" + arg0.getName());
    }
}

```

Real time report in Console

```

Start Of Execution(TEST)->Default
Test Started->testRealReportOne
Test Pass->testRealReportOne
Test Started->testRealReportTwo
Test Failed->testRealReportTwo
Test Skipped->testRealReportThree
END Of Execution(TEST)->Default te
PASSED: testRealReportOne
FAILED: testRealReportTwo
java.lang.AssertionError: expected
at org.testng.Assert.fail(
at org.testng.Assert.failN

```

These reports created in between execution

IReporter Interface

If we want to customize final test report generated by TestNG, we need to implement IReporter interface. This interface has only one method to implement generateReport. This method has all the information of a complete test execution in the List<ISuite>, and we can generate the report using it.

Code Example

Guru99Reporter.java is the file used to customize report

```

package testNGReport.iReporterReport;

import java.util.Collection;
import java.util.Date;
import java.util.List;

```

```
import java.util.Map;
import java.util.Set;
import org.testng.IReporter;
import org.testng.IResultMap;
import org.testng.ISuite;
import org.testng.ISuiteResult;
import org.testng.ITestContext;
import org.testng.ITestNGMethod;
import org.testng.xml.XmlSuite;
public class Guru99Reporter implements IReporter{
    @Override
    public void generateReport(List<XmlSuite> arg0, List<ISuite> arg1,
        String outputDirectory) {
        // Second parameter of this method ISuite will contain
        all the suite executed.
        for (ISuite iSuite : arg1) {
            //Get a map of result of a single suite at a time
            Map<String, ISuiteResult> results =
iSuite.getResults();
            //Get the key of the result map
            Set<String> keys = results.keySet();
            //Go to each map value one by one
```

```
for (String key : keys) {  
  
    //The Context object of current result  
  
    ITestContext context =  
results.get(key).getTestContext();  
  
    //Print Suite detail in Console  
  
    System.out.println("Suite Name->" + context.getName()  
                      + " :: Report output Directory-  
->" + context.getOutputDirectory())  
  
    + " :: Suite Name->" +  
context.getSuite().getName()  
  
    + " :: Start Date Time for execution-  
->" + context.getStartDate()  
  
    + " :: End Date Time for execution-  
->" + context.getEndDate());  
  
  
    //Get Map for only failed test cases  
  
    IResultMap resultMap = context.getFailedTests();  
  
    //Get method detail of failed test cases  
  
    Collection<ITestNGMethod> failedMethods =  
resultMap.getAllMethods();  
  
    //Loop one by one in all failed methods  
  
    System.out.println("-----FAILED TEST CASE-----  
-");  
  
    for (ITestNGMethod iTestNGMethod : failedMethods) {  
  
        //Print failed test cases detail
```

```
System.out.println("TESTCASE NAME-  
>"+iTestNGMethod.getMethodName()  
  
                                +"\\nDescription-  
>"+iTestNGMethod.getDescription()  
  
                                +"\\nPriority-  
>"+iTestNGMethod.getPriority()  
  
                                +"\\n:Date->"+new  
Date(iTestNGMethod.getDate()));  
  
    }  
  
}  
  
}  
  
}
```

TestGuru99ForReporter.java is a demo for Custom reporting

```
package testNGReport.iReporterReport;

import org.testng.Assert;

import org.testng.annotations.Listeners;

import org.testng.annotations.Test;

//Add listener to listen report and write it when testcas
finished

@Listeners(value=Guru99Reporter.class)
```

```
public class TestGuru99ForReporter {  
  
    @Test(priority=0,description="testReporterOne")  
  
    public void testReporterOne(){  
  
        //Pass test case  
  
        Assert.assertTrue(true);  
  
    }  
  
    @Test(priority=1,description="testReporterTwo")  
  
    public void testReporterTwo(){  
  
        //Fail test case  
  
        Assert.assertTrue(false);  
  
    }  
}
```

Output will be like-

```

Suite Name->Default test::Report out
-----FAILED TEST CASE-----
TESTCASE NAME->testReporterTwo
Description->testReporterTwo
Priority->1
>Date->Sat Jun 28 12:30:49 IST 2014
[TestNG] Time taken by testNGReport.i
[TestNG] Time taken by org.testng.rep
[TestNG] Time taken by org.testng.rep

Failed Testcase detail
Reporter detail

Suite Detail
estContext context = results.get(key).getTestContext();
Print Suite detail in Console
ystem.out.println("Suite Name->" + context.getName()
+ " ::Report output Ditectory->" + context.getOutputDirectory()
+ " ::Suite Name->" + context.getSuite().getName()
+ " ::Start Date Time for execution->" + context.getStartDate()
+ " ::End Date Time for execution->" + context.getEndDate());

//Get Map for only failed test cases
IResultMap resultMap = context.getFailedTests();
//Get method detail of failed test cases
Collection<ITestNGMethod> failedMethods = resultMap.getAllMethods();
//Loop one by one in all failed methods
System.out.println("-----FAILED TEST CASE-----");
for (ITestNGMethod iTestNGMethod : failedMethods) {
//Print failed test cases detail
System.out.println("TESTCASE NAME->" + iTestNGMethod.getMethodName()
+ "\nDescription->" + iTestNGMethod.getDescription()
+ "\nPriority->" + iTestNGMethod.getPriority()
+ "\nDate->" + new Date(iTestNGMethod.getDate()));
}

```

PDF and Email of Reports

The above report implementation is quite simple and clear to get you started with report customization.

But in corporate environment, you will need to create highly customized reports. Here is the scenario we will be dealing with

1. Create Custom Report in PDF form
2. Take Screenshots ONLY on Errors. Link to screenshots in PDF
3. Send Email of the PDF

The PDF report looks like this

Default test TESTNG RESULTS

Wed Jul 02 01:21:11 IST 2014

The screenshot shows a TestNG results report with two main sections: 'FAILED TESTS' and 'PASSED TESTS'. The 'FAILED TESTS' section has two rows, each with a yellow background. The 'PASSED TESTS' section has one row with a green background. Cloud-shaped callouts point from labels to specific parts of the report: 'Test Class Name' points to the first column of the 'FAILED TESTS' table; 'Test Methods' points to the second column of the same table; 'Passed Testcases' points to the first column of the 'PASSED TESTS' table; 'Test case Exception' points to the fourth column of the 'FAILED TESTS' table; and 'Link for Snapshot of failed Test Cases' points to the fourth column of the 'FAILED TESTS' table, where it highlights a Java assertion error message.

FAILED TESTS			
Class	Method	Time (ms)	Exception
[TestClass name=class test.TestGuru99ForReporter]	testReporterOne	9756	java.lang.AssertionErr or: expected [true] but found [false]{Screen Shot}
[TestClass name=class test.TestGuru99ForReporter]	testReporterTwo	2810	java.lang.AssertionErr or: expected [true] but found [false]{Screen Shot}
PASSED TESTS			
[TestClass	testReporter1Two	3030	

To create pdf report we need a Java API **IText**. Download it here . There is another custom listener class which is actually implementing this IText jar and creating a pdf report for us. Download it here

Above figure shows the default format of the PDF report generated. You can customize it

Here is how we will approach this

Step 1) Create a Base Class

Step 2) Customize JyptensionListerner.Java (PDF creation code)

Step 3) Create a TestGuru99PDFEmail.java which will execute test cases , create PDF

Step 4) Append code to TestGuru99PDFEmail.java to send PDF report via email

Let's look into these steps

Step 1) Create Base Class

This base class has functions to create WebDriver and Take Screenshot

```
package PDFEmail;

import java.io.File;

import org.apache.commons.io.FileUtils;

import org.openqa.selenium.OutputType;

import org.openqa.selenium.TakesScreenshot;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class BaseClass {

    static WebDriver driver;

    public static WebDriver getDriver(){

        if(driver==null){

            WebDriver driver ;

            System.setProperty("webdriver.firefox.marionette", "C:\\geckodriver.exe");
            driver = new FirefoxDriver();

        }

        return driver;

    }

}
```

```
/*
 * This function will take screenshot
 * @param webdriver
 * @param filePath
 * @throws Exception
 */

public static void takeSnapShot(WebDriver webdriver, String
filePath) throws Exception{

    //Convert web driver object to TakeScreenshot
    TakesScreenshot scrShot =((TakesScreenshot)webdriver);

    //Call getScreenshotAs method to create image file
    File
SrcFile=scrShot.getScreenshotAs(OutputType.FILE);

    //Move image file to new destination
    File DestFile=new File(filePath);

    //Copy file at destination
    FileUtils.copyFile(SrcFile, DestFile);

}

}
```

Step 2) Customize JyptensionListener.java

We will stick with the default report format. But we will make 2 customizations

- Adding code to instruct JyperionListener to take screenshot on Error
- Attaching the link of the screenshot take in the PDF report

```
public void onTestFailure(ITestResult result) {
    Log("onTestFailure("+result+ ")");
    String file = System.getProperty("user.dir")+"\\\"+ " + "screenshot" + (new Random().nextInt()) + ".png";
    try {
        BaseClass.takeSnapShot(BaseClass.getDriver(), file);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    if (this.failTable == null) {
        this.failTable = new PdfPTable(new float[]{.3f, .3f, .1f, .3f});
        this.failTable.setTotalWidth(20f);
        Paragraph p = new Paragraph("FAILED TESTS", new Font(Font.TIMES_ROMAN, Font.DEFAULTSIZE, Font.BOLD));
        p.setAlignment(Element.ALIGN_CENTER);
        PdfPCell cell = new PdfPCell(p);
        cell.setColspan(4);
        cell.setBackgroundColor(Color.RED);
    }
}
```

Add code to attach the screenshot to the PDF report

```
Throwable throwable = result.getThrowable();
if (throwable != null) {
    this.throwableMap.put(new Integer(throwable.hashCode()), throwable);
    this.nbExceptions++;

    Chunk imbd = new Chunk("[SCREEN SHOT]", new Font(Font.TIMES_ROMAN, Font.DEFAULTSIZE, Font.UNDERLINE));
    imbd.setAction(new PdfAction("file:///"+file));
    Paragraph excep = new Paragraph(
        throwable.toString());
    excep.add(imbd);
```

Step 3) Create a TestGuru99PDFEmail.java which will execute test cases , create PDF

- Here we will add JyperionListener.class as listener
- We will Execute 3 test cases.
- Using Assert.assertTrue we will fail 2 test cases while passing just one.
- Screenshot will be taken for the failed test cases only as per our customizations

```
package PDFEmail;

import java.util.Properties;

import javax.activation.DataHandler;

import javax.activation.DataSource;

import javax.activation.FileDataSource;

import javax.mail.BodyPart;

import javax.mail.Message;

import javax.mail.MessagingException;

import javax.mail.Multipart;

import javax.mail.Session;

import javax.mail.Transport;

import javax.mail.internet.AddressException;

import javax.mail.internet.InternetAddress;

import javax.mail.internet.MimeBodyPart;

import javax.mail.internet.MimeMessage;

import javax.mail.internet.MimeMultipart;

import org.openqa.selenium.WebDriver;

import org.testng.Assert;

import org.testng.annotations.AfterSuite;

import org.testng.annotations.Listeners;

import org.testng.annotations.Test;

import reporter.JyperionListener;
```

```
//Add listener for pdf report generation  
  
@Listeners(JyperionListener.class)  
  
public class TestGuru99PDFReport extends BaseClass {  
  
    WebDriver driver;  
  
    //Testcase failed so screen shot generate  
  
    @Test  
  
    public void testPDFReportOne(){  
  
        driver = BaseClass.getDriver();  
  
        driver.get("http://google.com");  
  
        Assert.assertTrue(false);  
  
    }  
  
    //Testcase failed so screen shot generate  
  
    @Test  
  
    public void testPDFReportTwo(){  
  
        driver = BaseClass.getDriver();  
  
        driver.get("http://guru99.com");  
  
        Assert.assertTrue(false);  
  
    }  
  
    //Test test case will be pass, so no screen shot on it
```

```
@Test  
  
public void testPDFReportThree(){  
  
    driver = BaseClass.getDriver();  
  
    driver.get("http://demo.guru99.com");  
  
    Assert.assertTrue(true);  
  
}
```

Step 4) Append code to TestGuru99PDFEmail.java to send PDF report via email

- We will use the annotation @AfterSuite to send email of the PDF report
- We will be sending email using Gmail
- To enable Email, need to import many library files like mail.jar, pop3.jar, smtp.jar, etc
- Before you execute this, do enter the from, to email address and password

```
//After complete execution send pdf report by email  
  
@AfterSuite  
  
public void tearDown(){  
  
    sendPDFReportByGMail("FROM@gmail.com", "password",  
"TO@gmail.com", "PDF Report", "");  
  
}  
  
/**  
 * Send email using java
```

```
* @param from
* @param pass
* @param to
* @param subject
* @param body
*/
private static void sendPDFReportByGMail(String from, String
pass, String to, String subject, String body) {
Properties props = System.getProperties();
String host = "smtp.gmail.com";
props.put("mail.smtp.starttls.enable", "true");
props.put("mail.smtp.host", host);
props.put("mail.smtp.user", from);
props.put("mail.smtp.password", pass);
props.put("mail.smtp.port", "587");
props.put("mail.smtp.auth", "true");
Session session = Session.getDefaultInstance(props);
MimeMessage message = new MimeMessage(session);
try {
//Set from address
message.setFrom(new InternetAddress(from));
message.addRecipient(Message.RecipientType.TO, new
```

```
InternetAddress(to);

//Set subject

message.setSubject(subject);

message.setText(body);

BodyPart objMessageBodyPart = new MimeBodyPart();

objMessageBodyPart.setText("Please Find The Attached Report
File!");

Multipart multipart = new MimeMultipart();

multipart.addBodyPart(objMessageBodyPart);

objMessageBodyPart = new MimeBodyPart();

//Set path to the pdf report file

String filename = System.getProperty("user.dir")+"\\Default
test.pdf";

//Create data source to attach the file in mail

DataSource source = new FileDataSource(filename);

objMessageBodyPart.setDataHandler(new DataHandler(source));

objMessageBodyPart.setFileName(filename);

multipart.addBodyPart(objMessageBodyPart);

message.setContent(multipart);

Transport transport = session.getTransport("smtp");

transport.connect(host, from, pass);

transport.sendMessage(message, message.getAllRecipients());

transport.close();
```

```
}

catch (AddressException ae) {

ae.printStackTrace();

}

catch (MessagingException me) {

me.printStackTrace();

}

}

}
```

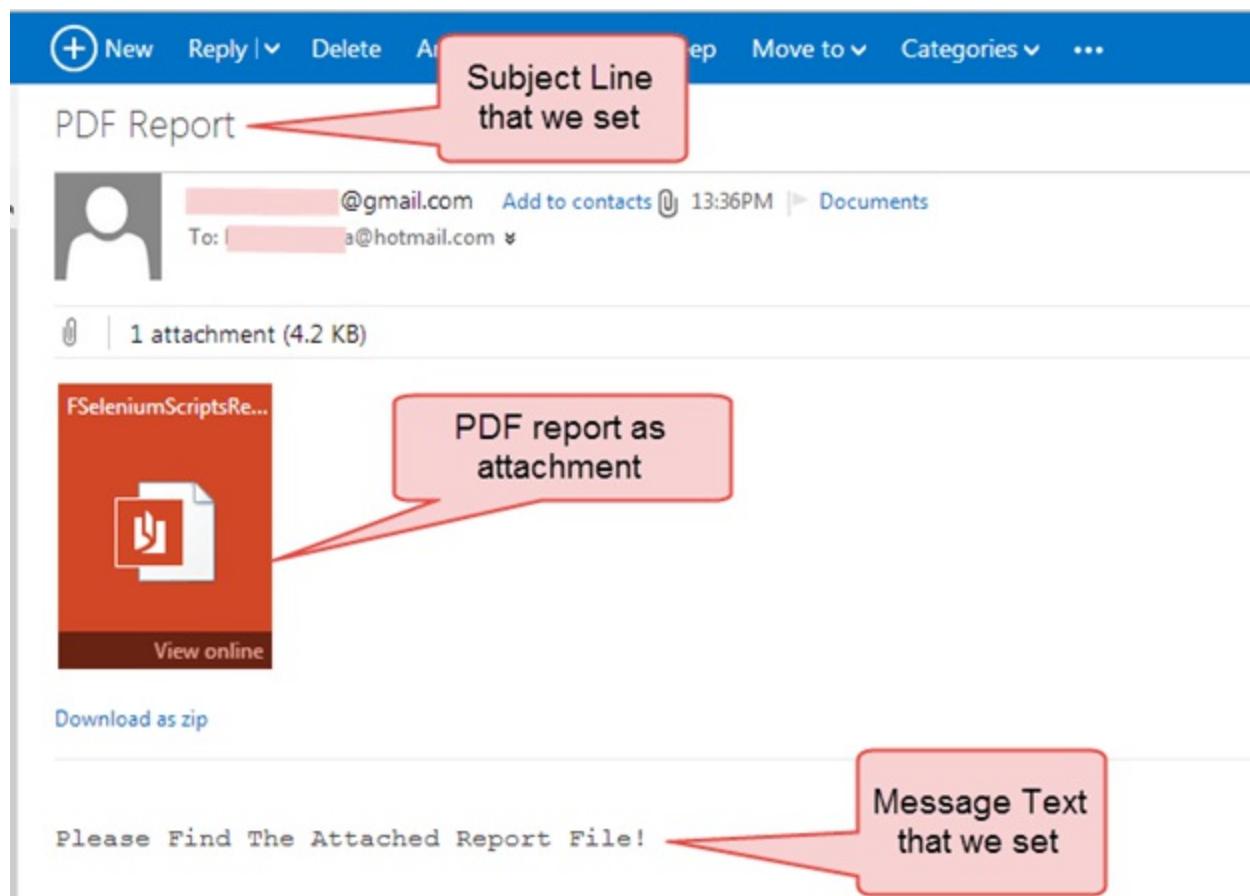
Download the complete project here

Note: When we click on the screen shot link in pdf, it shows security dialog. We have to allow this dialog to open pdf.

If a test case is fail and you want to see its screenshot , then on clicking {Screen Shot} link in pdf report you will get this message box. You have to click Allow



The email so generated will look like this



Summary:

- TestNG has an inbuilt reporting ability in it.
- After a complete execution of test cases, TestNG generates a test-output folder in the root of the project.
- In the test-output folder, there are two main reports, index.html, and emailable-report.html.
- To customize TestNG report we need to implement two interfaces, ITestListener and IReporter.
- If we need to get a report in between execution, we need ITestListener.
- For creating a final report after complete execution, we need to implement IReporter.
- Taking the screenshot, in Selenium WebDriver, we need to type

cast WebDriver to TakesScreenShot interface.

- To generate pdf reports we need to add IText jar in the project.

Chapter 33: How to Take Screenshot in Selenium WebDriver

Screenshots are desirable for bug analysis. Selenium can automatically take screenshots during execution. You need to type cast WebDriver instance to TakesScreenshot.



Taking Screenshot in Selenium is a 3 Step process

Step 1) Convert web driver object to TakeScreenshot

```
TakesScreenshot scrShot =((TakesScreenshot)webdriver);
```

Step 2) Call `getScreenshotAs` method to create image file

```
File SrcFile=scrShot.getScreenshotAs(OutputType.FILE);
```

Step 3) Copy file to Desired Location

Example: In this example we will take screenshot of <http://demo.guru99.com/V4/> & save it as C:/Test.png

```
package Guru99TakeScreenshot;

import java.io.File;

import org.apache.commons.io.FileUtils;

import org.openqa.selenium.OutputType;

import org.openqa.selenium.TakesScreenshot;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.Test;

public class Guru99TakeScreenshot {

    @Test

    public void testGuru99TakeScreenShot() throws Exception{

        WebDriver driver ;

        System.setProperty("webdriver.firefox.marionette", "C:\\geckodriver.exe");
        driver = new FirefoxDriver();

        //goto url

        driver.get("http://demo.guru99.com/V4/");

        //Call take screenshot function

        this.takeSnapShot(driver, "c://test.png") ;

    }

    /**

```

```
* This function will take screenshot  
* @param webdriver  
* @param fileWithPath  
* @throws Exception  
*/  
  
public static void takeSnapShot(WebDriver webdriver, String  
fileWithPath) throws Exception{  
  
    //Convert web driver object to TakeScreenshot  
  
    TakesScreenshot scrShot =((TakesScreenshot)webdriver);  
  
    //Call getScreenshotAs method to create image file  
  
    File  
SrcFile=scrShot.getScreenshotAs(OutputType.FILE);  
  
    //Move image file to new destination  
  
    File DestFile=new File(fileWithPath);  
  
    //Copy file at destination  
  
    FileUtils.copyFile(SrcFile, DestFile);  
}  
}
```

NOTE: Selenium version 3.9.0 and above does not provide Apache Commons IO JAR. You can simply download them here and call them in your project

Chapter 34: Sessions, Parallel run and Dependency in Selenium

To understand how to run scripts in parallel, let's first understand

Why do we need Session Handling?

During test execution, the Selenium WebDriver has to interact with the browser all the time to execute given commands. At the time of execution, it is also possible that, before current execution completes, someone else starts execution of another script, in the same machine and in the same type of browser.

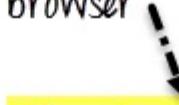


In such situation, we need a mechanism by which our two different executions should not overlap with each other. This can be achieved using Session Handling in Selenium.

How to achieve Session Handling in Selenium WebDriver?

If you check the source code of Selenium WebDriver, you will find a variable named as 'sessionId'. Whenever we create a new instance of a WebDriver object, a new 'sessionId' will be generated and attached with that particular Firefox/Chrome/IE Driver () .

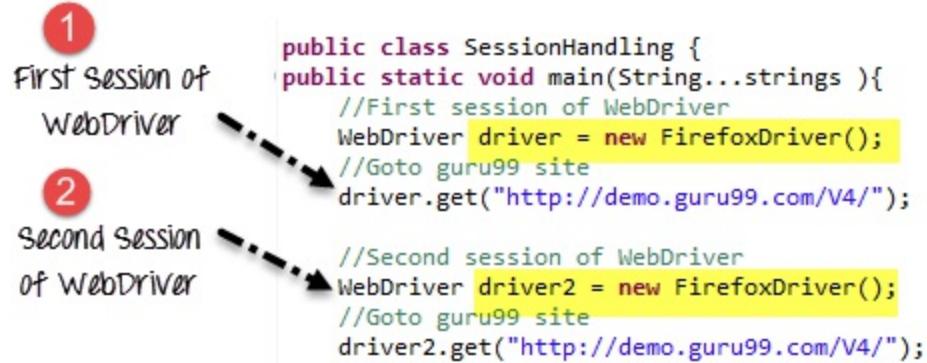
Here new session will generate for firefox
browser



```
webdriver=new FirefoxDriver();
```

So anything we do after this will execute only in that particular Firefox browser session.

For each driver there will be new session



As this is an in-built functionality, there is no explicit need to assign the session id

Code Example: Here two different sessions will be generated for two different WebDriver.

```

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class SessionHandling {
    public static void main(String...strings ){
        //First session of WebDriver
        WebDriver driver = new FirefoxDriver();
        //Goto guru99 site
        driver.get("http://demo.guru99.com/V4/");

        //Second session of WebDriver
        WebDriver driver2 = new FirefoxDriver();
        //Goto guru99 site
        driver2.get("http://demo.guru99.com/V4/");
    }
}

```

Run Scripts in Parallel

There are situations where you want to run multiple tests at the same

time.

In such cases, one can use "parallel" attribute

As the thread count is three, three WebDriver instances can run parallel in three different browsers

```

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="TestSuite" thread-count="3" parallel="methods" >
    <test name="testGuru">

```

Test cases can run parallel

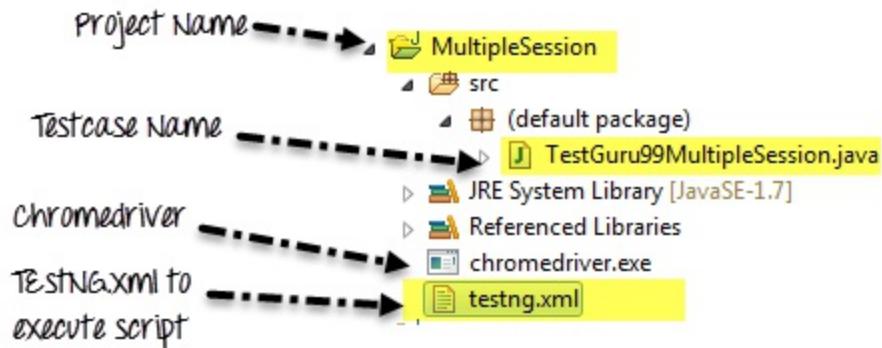
The parallel attribute of suite tag can accept four values:

tests	All the test cases inside <test> tag of Testing xml file will run parallel.
classes	All the test cases inside a Java class will run parallel
methods	All the methods with @Test annotation will execute parallel.
instances	Test cases in same instance will execute parallel but two methods of two different instances will run in different thread.

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

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

The Complete project will look like:



TestGuru99MultipleSession.java

```

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.Test;
public class TestGuru99MultipleSession {
    @Test
    public void executeSessionOne(){
        //First session of WebDriver
        System.setProperty("webdriver.chrome.driver","chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        //Goto guru99 site
        driver.get("http://demo.guru99.com/V4/");
        //find user name text box and fill it
        driver.findElement(By.name("uid")).sendKeys("Driver 1");
    }

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

```

```

    }

    @Test
        public void executSessionThree(){
            //Third session of WebDriver

System.setProperty("webdriver.chrome.driver","chromedriver.exe")
;
            WebDriver driver = new ChromeDriver();
            //Goto guru99 site
            driver.get("http://demo.guru99.com/V4/");
            //find user name text box and fill it
            driver.findElement(By.name("uid")).sendKeys("Driver 3");

        }
}

```

TestNG.XML

```

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

```

Test Case order and Dependency

You can set order and dependency of Test Case execution.

Suppose you have two test cases , 'testGuru99TC1' and 'testGuru99TC2' and you want to execute test case 'testGuru99TC2' before 'testGuru99TC1'. In that case we will use 'dependsOnMethods'

attribute to make dependency and order of execution.

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

Summary

- A new sessionID is created for a new instance of WebDriver.
- One session will bind with one particular browser.
- Using attribute thread and parallel, you run your scripts in parallel.
- You can use attribute dependency to set the order to test execution

Chapter 35: Tutorial on Log4j and LogExpert with Selenium

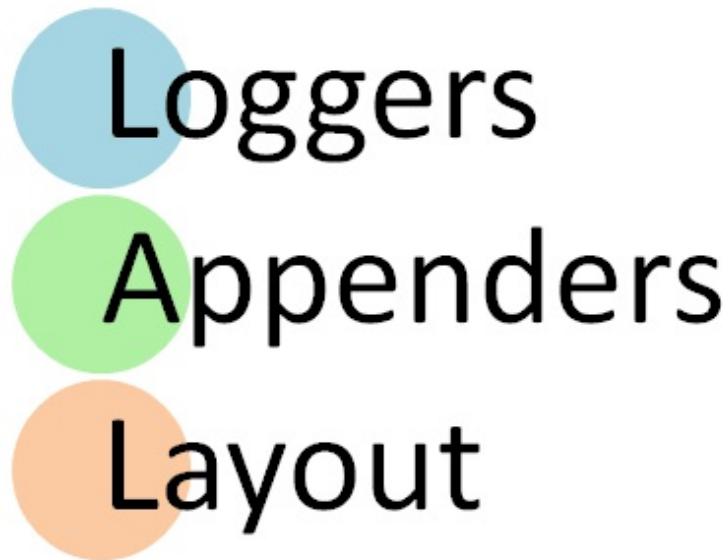
Introduction Log4j

Log4j is a fast, flexible and reliable logging framework (APIS) written in Java developed in early 1996. It is distributed under the Apache Software License. Log4J has been ported to the C, C++, C#, Perl, Python, Ruby and Eiffel Languages. It is a tool used for small to large scale Selenium Automation projects.

Why use Log4j?

- It is an open source
- With Log4j, it is possible to store the flow details of our Selenium Automation in a file or databases
- Log4j is used for large as well as small projects
- In Log4j, we use log statements rather than SOPL statements in the code to know the status of a project while it is executing

Log4j has three principal components



The diagram consists of three large, bold, black text labels: 'Loggers', 'Appenders', and 'Layout'. Each label is positioned above a corresponding colored circle: a blue circle for 'Loggers', a green circle for 'Appenders', and an orange circle for 'Layout'.

1. **Loggers:** It is responsible for logging information. To implement loggers into a project following steps need to be performed -
 - **Create an instance for logger class:** Logger class is a Java-based utility that has got all the generic methods already implemented to use log4j
 - **Define the Log4j level:** Primarily there are five kinds of log levels
 1. All - This level of logging will log everything (it turns all the logs on)
 2. DEBUG – print the debugging information and is helpful in development stage
 3. INFO – print informational message that highlights the progress of the application
 4. WARN – print information regarding faulty and unexpected system behavior.
 5. ERROR – print error message that might allow system to continue
 6. FATAL – print system critical information which are causing the application to crash

7. OFF – No logging

2. **Appenders:** It is used to deliver LogEvents to their destination.

It decides what will happen with log information. In simple words, it is used to write the logs in file. Following are few types of Appenders

1. ConsoleAppender logs to standard output
2. File appender prints logs to some file
3. Rolling file appender to a file with maximum size

Note: In log4j properties we can call appender with any name.

There are other appenders as well but we will restrict to these few.

3. **Layouts:** It is responsible for formatting logging information in different styles.

The Logger class provides different methods to handle logging activities. It provides two static methods for obtaining a Logger Object.

- **Public static Logger getRootLogger()**
- **Public static Logger getLogger(String name)**

How log4j is configured?

To configure log4j we have to decide which appender to implement. Accordingly, parameters of appender will be set.

- We will use DEBUG level and RollingFileAppender
- We will do two configurations or logs,
 - First: root logger, that will write all system generated logs in file name i.e. Selenium.logs
 - Second: Will write the information generated by manual

commands in code into the file name- Manual.logs

- Layout will be PatternLayout

#Root logger

```
log4j.rootLogger=DEBUG, file
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=D:\\Guru99\\src\\Selenium.logs
log4j.appender.file.maxFileSize=900KB
log4j.appender.file.maxBackupIndex=5
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p
%c<strong>{1}</strong>:%L - %m%n
log4j.appender.file.Append=false
```

#Application Logs

```
log4j.logger.devpinoyLogger=DEBUG, dest1
log4j.appender.dest1=org.apache.log4j.RollingFileAppender
log4j.appender.dest1.maxFileSize=900KB
log4j.appender.dest1.maxBackupIndex=6
log4j.appender.dest1.layout=org.apache.log4j.PatternLayout
log4j.appender.dest1.layout.ConversionPattern=%d{dd/MM/yyyy
HH:mm:ss} %c %m%n
log4j.appender.dest1.File=D:\\ Guru99\\src\\Manual.logs
log4j.appender.dest1.Append=false
```

In the above code, we have configured log4j to log in two different files named as Selenium.log and Manual.log.

- file and dest1 are the two identifiers.
- "File" is used to give file name in which logs will be saved
- "maxFileSize" is used to configure the maximum size of the log file. When file reaches this size, a new file will be created with the same name and the old file name will be add as an Index to it.
- "maxBackupIndex" is used to configure maximum number of files to be backup.

- "layout" is used to set the format of the log file.
- "Append" is used to set append function. If it is set to false, than every time a new file will be created rather than old file will be used for logging

How log4j is used within script?

In code, we have used "log" as a reference variable referencing getLogger method of Logger Class

```
Logger log = Logger.getLogger("devpinoyLogger");
```

Use "log" referencing variable and debug method to log the information we want.

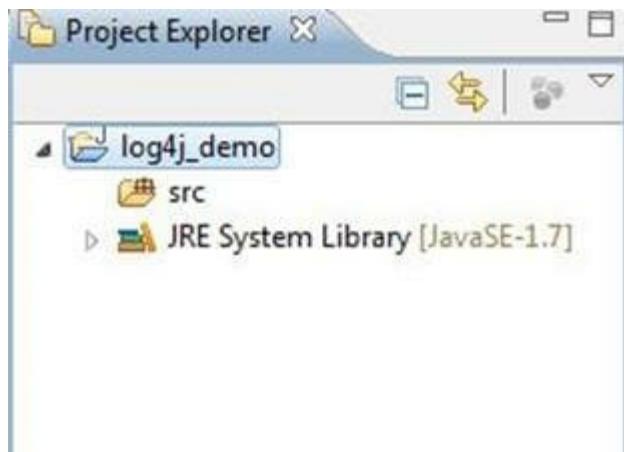
```
log.debug("--information--");
```

What is a LogExpert tool?

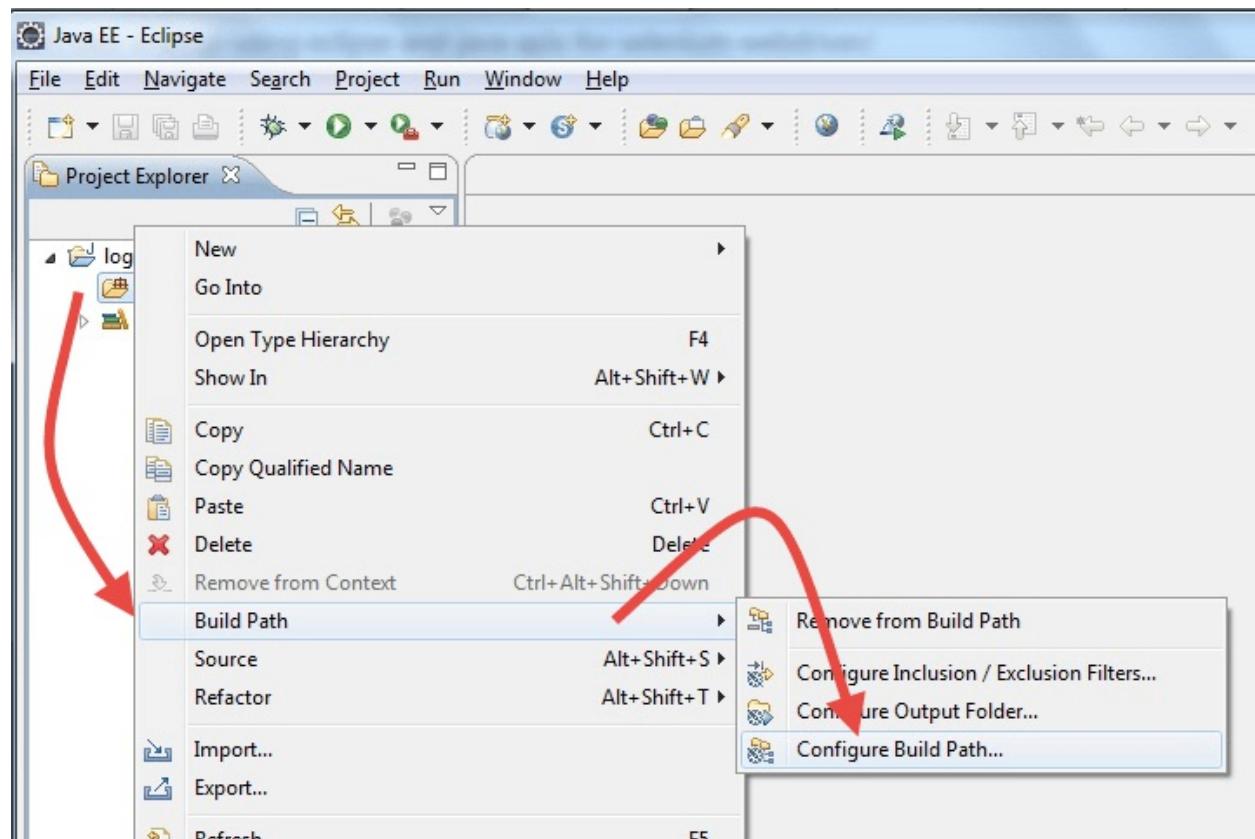
1. LogExpert tool is a tool for Windows developed to tail the logs
2. It is free and open source log viewer.
3. It is a log analysis tool with multiple features like searching, filtering, bookmarking and highlighting the logs
4. In this tool logs, files get automatically updated when opened
5. In this tool, we can open multiple log file in different tabs
6. We can also put comments on bookmarks, and there is the shortcut key to navigate in between different bookmarks. We can also see complete bookmark list and navigate from there
7. Shortcuts of the tool are given in help file so that they can be referred to the tool.

Steps to use Log4j with Selenium

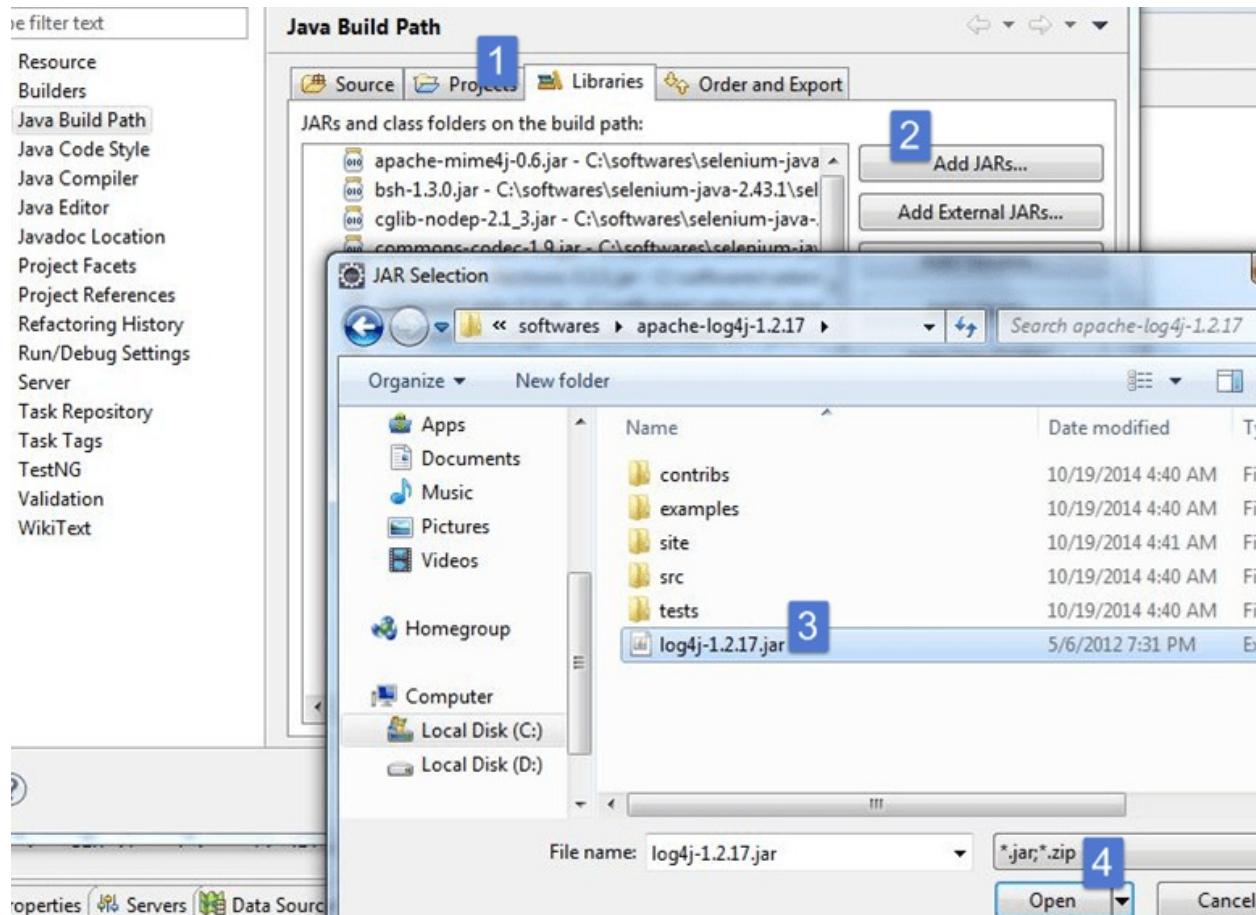
Step 1) In Eclipse create a new project with name log4j_demo



Step 2) Right click on src -> Build Path -> Configure Build Path



Step 2) Click on Libraries and Add Log4J Library . You can download it from <https://logging.apache.org/log4j/1.2/download.html>



Step 3) Create a new file. This file will include all the log4j configuration

1. Right click on src -> New -> Other -> General -> File
2. Give the file name as "log4j.properties"
3. Click on Finish

Create two more files and give them names such as Selenium.logs and Manual.logs. These files will contain all the logs created by system and manually logged statements



Step 4) In log4j.properties copy the entire configuration.

```
log4j.properties X
#Root logger option

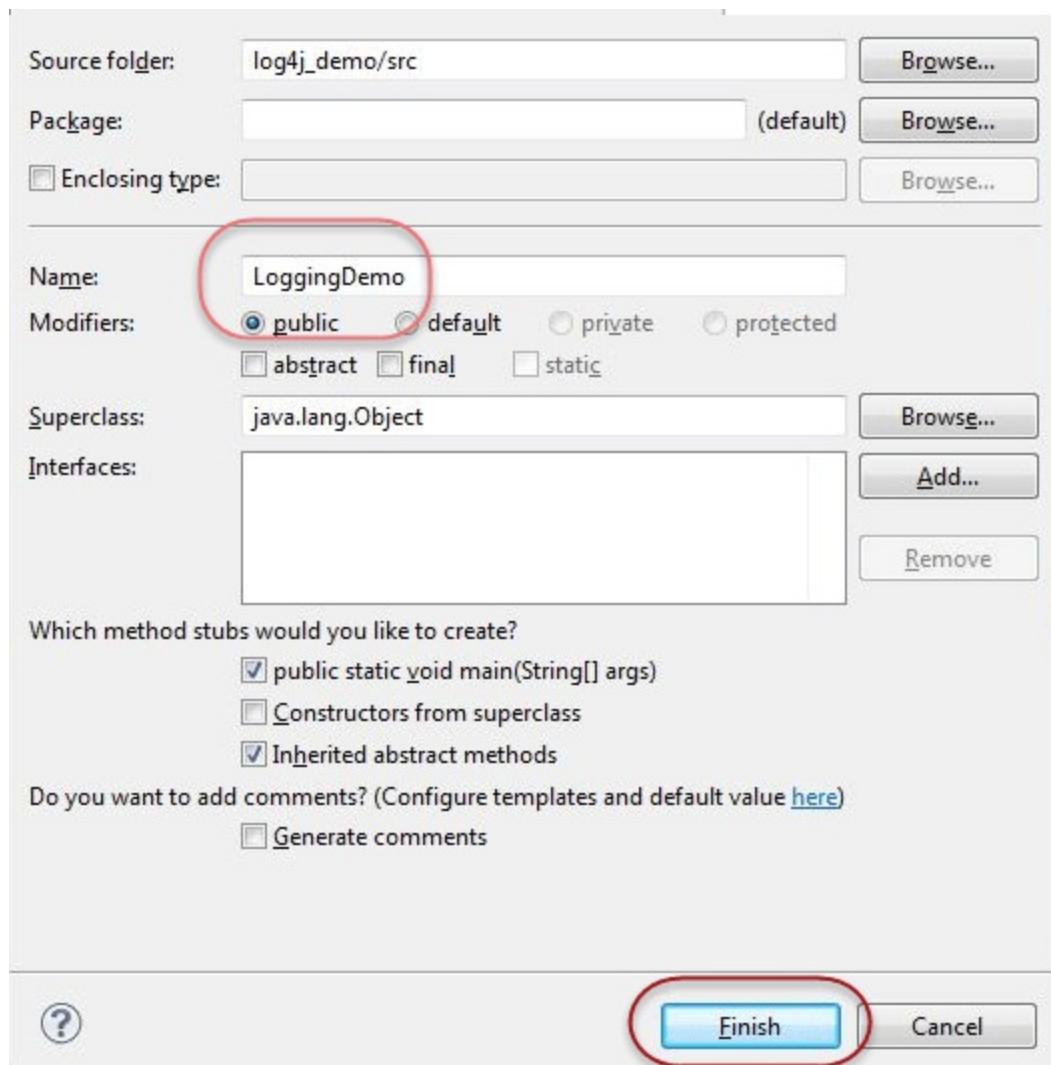
log4j.rootLogger=debug,file
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=D:\\selenium_log4j\\log4j_demo\\src\\Selenium.logs
log4j.appender.file.maxFileSize=5000KB
log4j.appender.file.maxBackupIndex=1
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c<strong>{1}</strong>:%L - %m%n
log4j.appender.file.Append=false

#Application Logs

log4j.logger.devpinoyLogger=DEBUG,dest1
log4j.appender.dest1=org.apache.log4j.RollingFileAppender
log4j.appender.dest1.maxFileSize=5000KB
log4j.appender.dest1.maxBackupIndex=3
log4j.appender.dest1.layout=org.apache.log4j.PatternLayout
log4j.appender.dest1.layout.ConversionPattern=%d{dd/MM/yyyy HH:mm:ss} %c %m%n
log4j.appender.dest1.File=D:\\selenium_log4j\\log4j_demo\\src\\Manual.logs
log4j.appender.dest1.Append=false
```

Step 5) Create main class:

1. Right click on default package -> New -> Class
2. Give the class name and click on finish



Step 6) Copy the following code in to the main class

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.apache.log4j.Logger;

public class LoggingDemo {
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        WebDriver driver = new FirefoxDriver();
        Logger log = Logger.getLogger("devpinoyLogger");
```

```

        driver.get("http://healthunify.com/bmicalculator/");
            log.debug("opening webiste");
        driver.manage().timeouts().implicitlyWait(20,
TimeUnit.SECONDS);
            log.debug("entrng weight");

driver.findElement(By.name("wg")).sendKeys("87");
log.debug("selecting kilograms");

driver.findElement(By.name("opt1")).sendKeys("kilograms");
log.debug("selecting height in feet");
driver.findElement(By.name("opt2")).sendKeys("5");
log.debug("selecting height in inchs");
driver.findElement(By.name("opt3")).sendKeys("10");
log.debug("Clicking on calculate");
driver.findElement(By.name("cc")).click();

        log.debug("Getting SIUnit value");
String SIUnit =
driver.findElement(By.name("si")).getAttribute("value");
log.debug("Getting USUnit value");
String USUnit =
driver.findElement(By.name("us")).getAttribute("value");
log.debug("Getting UKUnit value");
String UKUnit =
driver.findElement(By.name("uk")).getAttribute("value");
log.debug("Getting overall description");
String note =
driver.findElement(By.name("desc")).getAttribute("value");

System.out.println("SIUnit = " + SIUnit);
System.out.println("USUnit = " + USUnit);
System.out.println("UKUnit = " + UKUnit);
System.out.println("note = " + note);
        driver.quit();
    }
}

```

In the above code, we visit <http://healthunify.com/bmicalculator/> and verify BMI calculator. The weight entered is 87KG and the height is 5

Feet 10 inches. The script checks output in SE, US and UK units.

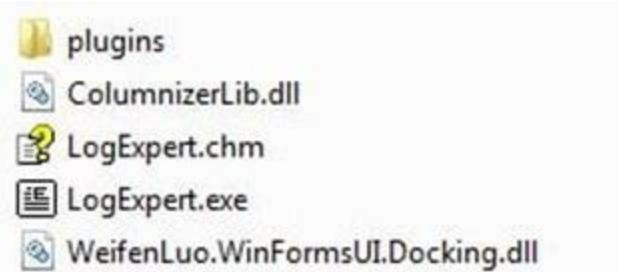
Using Logger.getLogger("devpinoyLogger") we create system level logs

Using log.debug method we store data into Manual.log

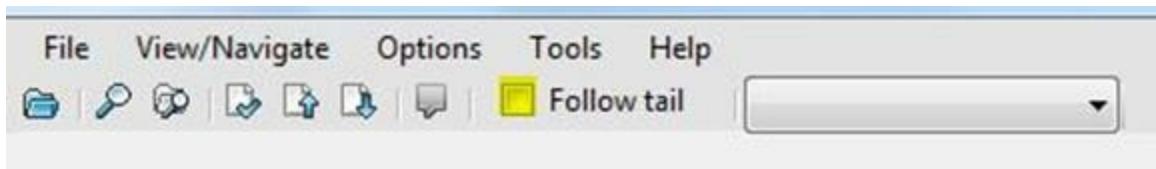
Step 7) Run the script. Open the location of Manual and Selenium logs to check logging data.

How LogExpert tool can be used to analyze logs

1. Download the tool from <http://logexpert.codeplex.com/> . Go to LogExpert download folder



2. Open LogExpert.exe
3. Click on File -> Open and Browse to the path where Manual.log and Selenium.log files are stored. Select the file
4. Select the "Follow tail" option



Selecting follow tail option enables tailing of logs which means LogExpert automatically updates the log file when script is in

execution phase. If we use any other editor like notepad then we have to close and reopen the file again and again to update the logs. But with ExpertTool in Follow Tail Mode this is not required.

Following images shows the layout of the logs

The screenshot shows the LogExpert application window. The title bar reads "LogExpert - D:\selenium_log4j\log4j_demo\bin\Manual.logs". The menu bar includes File, View/Navigate, Options, Tools, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, Find, and Copy. A checkbox labeled "Follow tail" is checked. The main area displays two tabs: "Manual.logs" (selected) and "Selenium.logs". The "Manual.logs" tab contains a table with two columns: "Line" and "Text". The text log entries are:

Line	Text
1	11/03/2015 23:46:10 devpinoyLogger opening website
2	11/03/2015 23:46:11 devpinoyLogger entering weight
3	11/03/2015 23:46:16 devpinoyLogger selecting kilograms
4	11/03/2015 23:46:17 devpinoyLogger selecting height in feet
5	11/03/2015 23:46:17 devpinoyLogger selecting height in inches
6	11/03/2015 23:46:17 devpinoyLogger Clicking on calculate
7	11/03/2015 23:46:17 devpinoyLogger Getting SIUnit value
8	11/03/2015 23:46:18 devpinoyLogger Getting USUnit value
9	11/03/2015 23:46:18 devpinoyLogger Getting UKUnit value
10	11/03/2015 23:46:18 devpinoyLogger Getting overall description

At the bottom, status bars show "10 lines", "575 bytes", and "1".

The screenshot shows the LogExpert application window. The title bar reads "LogExpert - D:\selenium_log4j\log4j_demo\bin\Selenium.logs". The menu bar includes File, View/Navigate, Options, Tools, and Help. Below the menu is a toolbar with icons for Open, Save, Find, Copy, Paste, and Follow tail. The main interface has two tabs: "Manual.logs" (selected) and "Selenium.logs". The "Selenium.logs" tab displays a list of log entries. Each entry consists of a line number and a timestamp followed by a log message. The log messages are primarily DEBUG level entries from org.apache.http components, detailing the creation and configuration of HttpClientConnectionManager, MainClientExec, and HttpClients, along with various header settings and connection details. The bottom of the window shows status information: "1049 lines", "141 KB", and page number "1".

Line	Text
92	23:46:10,744 DEBUG org.apache.http.impl.conn.PoolingHttpClientConnectionManager{1}:219 - Conne
93	23:46:10,744 DEBUG org.apache.http.impl.conn.PoolingHttpClientConnectionManager{1}:250 - Conne
94	23:46:10,744 DEBUG org.apache.http.impl.execchain.MainClientExec{1}:217 - Opening connection (
95	23:46:10,745 DEBUG org.apache.http.impl.conn.HttpClientConnectionOperator{1}:120 - Connecting :
96	23:46:10,746 DEBUG org.apache.http.impl.conn.HttpClientConnectionOperator{1}:127 - Connection :
97	23:46:10,747 DEBUG org.apache.http.impl.execchain.MainClientExec{1}:238 - Executing request PO:
98	23:46:10,747 DEBUG org.apache.http.impl.execchain.MainClientExec{1}:243 - Target auth state: UN
99	23:46:10,747 DEBUG org.apache.http.impl.execchain.MainClientExec{1}:249 - Proxy auth state: UN
100	23:46:10,747 DEBUG org.apache.http.headers{1}:124 - http-outgoing-2 >> POST /hub/session/7f98c
101	23:46:10,748 DEBUG org.apache.http.headers{1}:127 - http-outgoing-2 >> Content-Type: applicati
102	23:46:10,748 DEBUG org.apache.http.headers{1}:127 - http-outgoing-2 >> Content-Length: 30
103	23:46:10,748 DEBUG org.apache.http.headers{1}:127 - http-outgoing-2 >> Host: 127.0.0.1:7055
104	23:46:10,748 DEBUG org.apache.http.headers{1}:127 - http-outgoing-2 >> Connection: Keep-Alive
105	23:46:10,749 DEBUG org.apache.http.headers{1}:127 - http-outgoing-2 >> User-Agent: Apache-Http
106	23:46:10,749 DEBUG org.apache.http.headers{1}:127 - http-outgoing-2 >> Accept-Encoding: gzip,defl
107	23:46:10,749 DEBUG org.apache.http.wire{1}:72 - http-outgoing-2 >> "POST /hub/session/7f98ca7f
108	23:46:10,749 DEBUG org.apache.http.wire{1}:72 - http-outgoing-2 >> "Content-Type: application/
109	23:46:10,750 DEBUG org.apache.http.wire{1}:72 - http-outgoing-2 >> "Content-Length: 30[\r][\n]
110	23:46:10,750 DEBUG org.apache.http.wire{1}:72 - http-outgoing-2 >> "Host: 127.0.0.1:7055[\r][\n]
111	23:46:10,750 DEBUG org.apache.http.wire{1}:72 - http-outgoing-2 >> "Connection: Keep-Alive[\r]
112	23:46:10,750 DEBUG org.apache.http.wire{1}:72 - http-outgoing-2 >> "User-Agent: Apache-HttpCli
113	23:46:10,750 DEBUG org.apache.http.wire{1}:72 - http-outgoing-2 >> "Accept-Encoding: gzip,defl
114	23:46:10,751 DEBUG org.apache.http.wire{1}:72 - http-outgoing-2 >> "[\r][\n]"
115	23:46:10,751 DEBUG org.apache.http.wire{1}:86 - http-outgoing-2 >> "[{"ms":20000,"type":"implic
116	23:46:10,770 DEBUG org.apache.http.wire{1}:72 - http-outgoing-2 >> "HTTP/1.1 200 OK[\r][\n]"

Using LogExpert tool, one can debug logs created by the selenium

webdriver as in this tool once can

- search for any text and regular expression,
- create bookmark and comment them and also can navigate between bookmarks which is not possible in any other tool,
- Filter the logs and search for text ranges and also can apply another filter to the previous filtered logs,
- Highlight different line based on some certain words.

This tool also helps to partition the data into different columns.

Chapter 36: Selenium with HTMLUnit Driver & PhantomJS

Selenium Web driver is a web automation tool which enables you to run the tests against different browsers. These browsers can be Internet Explorer, Firefox or Chrome. To use a particular browser with Selenium you need corresponding driver.

At test run, Selenium launches the corresponding browser called in script and executes test steps. You can see the browser and the test execution in action.

What Is Headless Browser?

A headless browser is a web-browser **without a graphical user interface**. This program will behave just like a browser but will not show any GUI.

Some of the examples of Headless Drivers include

- HtmlUnit
- Ghost
- PhantomJS
- ZombieJS
- Watir-webdriver

In this tutorial we will focus on HtmlUnit and PhatomJS

HTMLUnitDriver

HTML UnitDriver is the most light weight and fastest implementation headless browser for of WebDriver. It is based on HtmlUnit. It is known as **Headless Browser Driver**. It is same as Chrome, IE, or FireFox driver, but it does not have GUI so one cannot see the test execution on screen.

Features of HTML unit driver

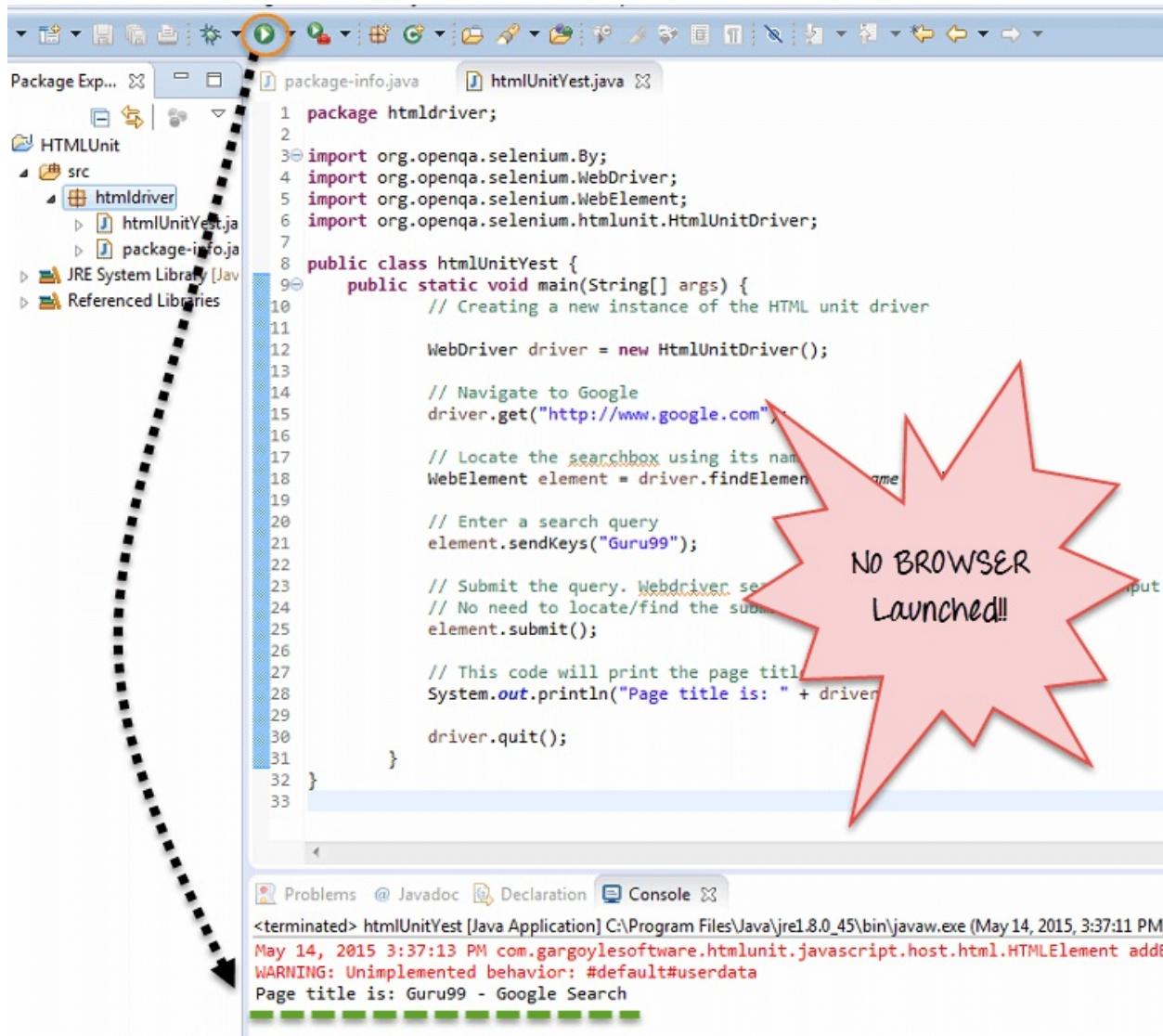
- Support for the HTTPS and HTTP protocols
- Support for HTML responses (clicking links, submitting forms, walking the DOM model of the HTML document etc.)
- Support for cookies
- Proxy server support
- Support for basic and NTLM authentication
- Excellent JavaScript support
- Support for submit methods GET and POST
- Ability to customize the request headers being sent to the server
- Ability to determine whether failing responses from the server should throw exceptions or should be returned as pages of the appropriate type

Steps to Use HTMLUnit Driver with Selenium

Step 1) In Eclipse, copy the following code. Add the standard selenium library files to the project. No additional jar files are required.

```
package htmldriver;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.htmlunit.HtmlUnitDriver;
public class htmlUnitYest {
    public static void main(String[] args) {
        // Creating a new instance of the HTML unit
        WebDriver driver = new HtmlUnitDriver();
        // Navigate to Google
        driver.get("http://www.google.com");
        // Locate the searchbox
        // using its name
        WebElement element =
        driver.findElement(By.name("q"));
        // Enter a search query
        element.sendKeys("Guru99");
        // Submit the query. Webdriver
        // searches for the form using the text input element automatically
        // No need to locate/find the submit button
        element.submit();
        // This code will print the page title
        System.out.println("Page title is: " +
        driver.getTitle());
        driver.quit();
    }
}
```

Step 2) Run the code. You will observe no browser is launched and results are shown in console.



```

1 package htmldriver;
2
3 import org.openqa.selenium.By;
4 import org.openqa.selenium.WebDriver;
5 import org.openqa.selenium.WebElement;
6 import org.openqa.selenium.htmlunit.HtmlUnitDriver;
7
8 public class htmlUnitYest {
9     public static void main(String[] args) {
10         // Creating a new instance of the HTML unit driver
11
12         WebDriver driver = new HtmlUnitDriver();
13
14         // Navigate to Google
15         driver.get("http://www.google.com");
16
17         // Locate the searchbox using its name
18         WebElement element = driver.findElement(By.name("q"));
19
20         // Enter a search query
21         element.sendKeys("Guru99");
22
23         // Submit the query. WebDriver sends the enter key
24         // No need to locate/find the submit button
25         element.submit();
26
27         // This code will print the page title
28         System.out.println("Page title is: " + driver.getTitle());
29
30         driver.quit();
31     }
32 }
33

```

Console output:

```

<terminated> htmlUnitYest [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (May 14, 2015, 3:37:11 PM)
May 14, 2015 3:37:13 PM com.gargoylesoftware.htmlunit.javascript.host.html.HTMLInputElement addEventList
WARNING: Unimplemented behavior: #default#userdata
Page title is: Guru99 - Google Search

```

Benefits of Html Unit Driver:

- Since it is not using any GUI to test, your tests will run in background without any visual interruption
- Compared to all other instances execution is faster
- To run your tests through HtmlUnit driver you can also select other browser versions
- It is platform independent and easier to run several tests concurrently. Ideal for Load Testing.

Limitations:

- It cannot emulate other browsers JavaScript behavior

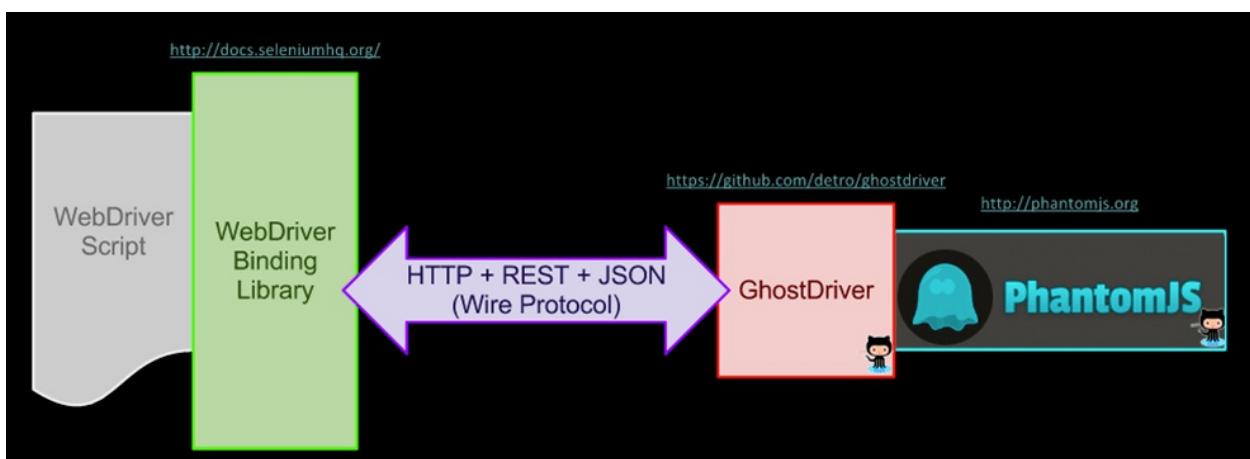
PhantomJS

PhantomJS is a headless browser with JavaScript API. It is an optimal solution for Headless Website Testing, access and manipulate webpages & comes with the standard DOM API.

In order to use PhantomJS with Selenium, one has to use GhostDriver. **GhostDriver** is a implementation of Webdriver Wire protocol in simple JS for PhantomJS.

The latest release of PhatomJS has **integrated** GhostDriver and **there is no need to separately install it.**

Here is how the system works-



Steps to run Selenium with PhantomJS

Step 1) You need Eclipse with Selenium installed

Step 2) Download PhantomJS here

Please take a moment to [improve this document](#) with anything that could be useful to other developers.

Documentation

Get Started

- [Download](#)
- [Build](#)
- [Releases](#)
- [Release Names](#)
- [REPL](#)

Learn

- [Quick Start](#)
- [Headless Testing](#)
- [Screen Capture](#)
- [Network Monitoring](#)
- [Page Automation](#)
- [Inter Process Communication](#)
- [Command Line Interface](#)

Get Help

- [Troubleshooting](#)
- [FAQ](#)

Explore

- [Examples](#)

Download

Note Binary packages are made available on a volunteer basis. There is no guarantee that a binary package for a given platform will be ready. The packagers are volunteers who release them as soon as they can after each release and they give their best effort to make the binaries available.

Download service is kindly provided by [Bitbucket](#). Previous releases can also be found at [Google Code Project Hosting](#).

Windows

Download [phantomjs-2.0.0-windows.zip](#) (19.4 MB) and extract (unzip).

The executable `phantomjs.exe` is ready to use.

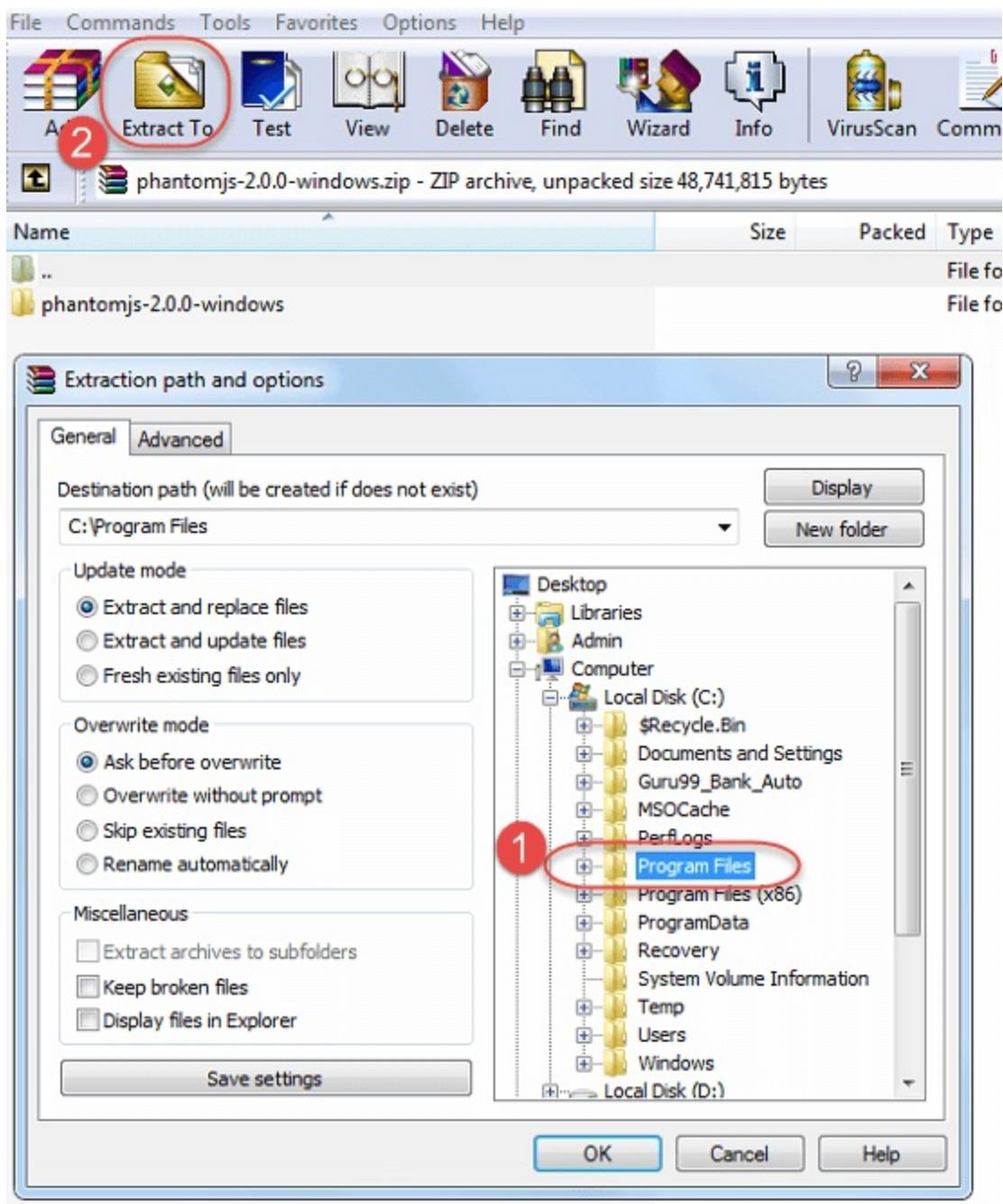
Note: For this static build, the binary is self-contained with no external dependencies. It requires a fresh install of Windows Vista or later versions. There is no requirement for any other libraries.

Mac OS X

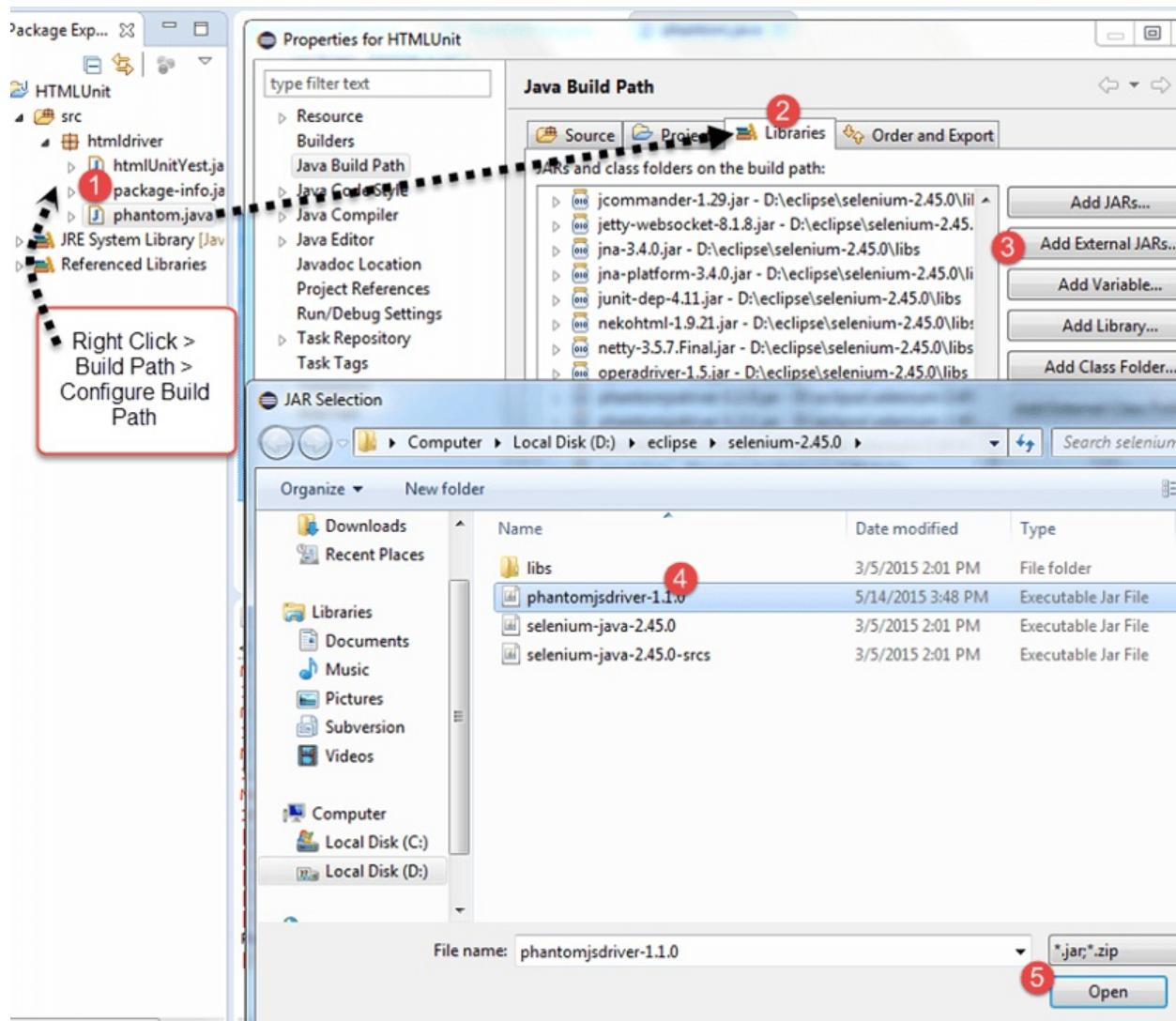
Download [phantomjs-2.0.0-macosx.zip](#) (13.7 MB) and extract (unzip).

The binary `bin/phantomjs` is ready to use.

Step 3) Extract the downloaded folder to Program Files



Step 4) Download the PhantomJS Driver from here. Add the jar to your project



Step 5) Paste the following code in eclipse

```

package htmldriver;
import java.io.File;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.phantomjs.PhantomJSDriver;

public class phantom {
    public static void main(String[] args) {
        File file = new File("C:/Program
Files/phantomjs-2.0.0-windows/bin/phantomjs.exe");
        System.setProperty("phantomjs.binary.path",
    }
}

```

```
file.getAbsolutePath());
        WebDriver driver = new PhantomJSDriver();
        driver.get("http://www.google.com");
        WebElement element =
driver.findElement(By.name("q"));
        element.sendKeys("Guru99");
        element.submit();
        System.out.println("Page title is: " +
driver.getTitle());
        driver.quit();
    }
}
```

Step 6) Run the code. You will observe the output is shown in console and no browser is launched.

NOTE: At first run, based on your settings, you may get security warning from Windows to allow to run PhantomJS. Click on Allow Access.

The screenshot shows the Eclipse IDE interface. In the top tab bar, three files are listed: package-info.java, htmlUnitYest.java, and phantom.java. The phantom.java file is currently open and displayed in the editor. The code implements a main method that sets up a PhantomJS driver, navigates to Google, sends the text 'Guru99' to the search input, and prints the page title. The code is annotated with line numbers from 1 to 20.

Below the editor is the Eclipse Console view, which displays the terminal output of the Java application. The output shows the application starting up, connecting to the PhantomJS service, and successfully performing a search for 'Guru99' on Google, printing the result 'Page title is: Guru99 - Google Search'. The application then shuts down.

```

1 package htmldriver;
2 import java.io.File;
3 import org.openqa.selenium.By;
4 import org.openqa.selenium.WebDriver;
5 import org.openqa.selenium.WebElement;
6 import org.openqa.selenium.phantomjs.PhantomJSDriver;
7
8 public class phantom {
9     public static void main(String[] args) {
10         File file = new File("C:/Program Files/phantomjs-2.0.0-windows/bin/phantomjs.exe");
11         System.setProperty("phantomjs.binary.path", file.getAbsolutePath());
12         WebDriver driver = new PhantomJSDriver();
13         driver.get("http://www.google.com");
14         WebElement element = driver.findElement(By.name("q"));
15         element.sendKeys("Guru99");
16         element.submit();
17         System.out.println("Page title is: " + driver.getTitle());
18         driver.quit();
19     }
20 }

```

```

terminated> phantom [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (May 14, 2015, 4:24:22 PM)
May 14, 2015 4:24:22 PM org.openqa.selenium.phantomjs.PhantomJSDriverService <init>
INFO: executable: C:\Program Files\phantomjs-2.0.0-windows\bin\phantomjs.exe
May 14, 2015 4:24:22 PM org.openqa.selenium.phantomjs.PhantomJSDriverService <init>
INFO: port: 7115
May 14, 2015 4:24:22 PM org.openqa.selenium.phantomjs.PhantomJSDriverService <init>
INFO: arguments: [--webdriver=7115, --webdriver-logfile=D:\eclipse\workspace\HTMLUnit\phantomjsd]
May 14, 2015 4:24:22 PM org.openqa.selenium.phantomjs.PhantomJSDriverService <init>
INFO: environment: {}
[INFO - 2015-05-14T10:54:24.834Z] GhostDriver - Main - running on port 7115
[INFO - 2015-05-14T10:54:26.067Z] Session [96513350-fa27-11e4-8966-0503e9e3b0e4] - page.setting
[INFO - 2015-05-14T10:54:26.067Z] Session [96513350-fa27-11e4-8966-0503e9e3b0e4] - page.customH
[INFO - 2015-05-14T10:54:26.068Z] Session [96513350-fa27-11e4-8966-0503e9e3b0e4] - Session.nego
[INFO - 2015-05-14T10:54:26.068Z] SessionManagerReqHand - _postNewSessionCommand - New Session
Page title is: Guru99 - Google Search
[INFO - 2015-05-14T10:54:51.447Z] ShutdownReqHand - _handle - About to shutdown

```

Many organization uses Phantom.JS for various purpose, for example,

- Headless Testing
- Screen Capture
- Page Automation
- Network Monitoring
- To render dashboard screenshots for their users
- To run Unit tests on command line

- To generate employee handbooks from HTML to PDF
- Combined with QUnit for the test suite

Summary

To test application rapidly in various browsers and without any visual interruption, headless browser Testing is used. Due to its speed, accuracy and easy to access features, HTML unit driver and PhantomJS are gaining popularity for headless browser testing. By following some simple steps you get to know how easily these tools can be integrated with other tools and can execute the test code.

Chapter 37: Using Robot API with Selenium

Why Robot Class?

In certain Selenium Automation Tests, there is a need to control keyboard or mouse to interact with OS windows like Download pop-up, Alerts, Print Pop-ups, etc. or native Operation System applications like Notepad, Skype, Calculator, etc.

Selenium Webdriver cannot handle these OS pop-ups/applications.

In Java version 1.3 Robot Class was introduced. Robot Class can handle OS pop-ups/applications.

Benefits of Robot Class

1. Robot Class can simulate Keyboard and Mouse Event
2. Robot Class can help in upload/download of files when using selenium web driver
3. Robot Class can easily be integrated with current automation framework (keyword, data-driven or hybrid)

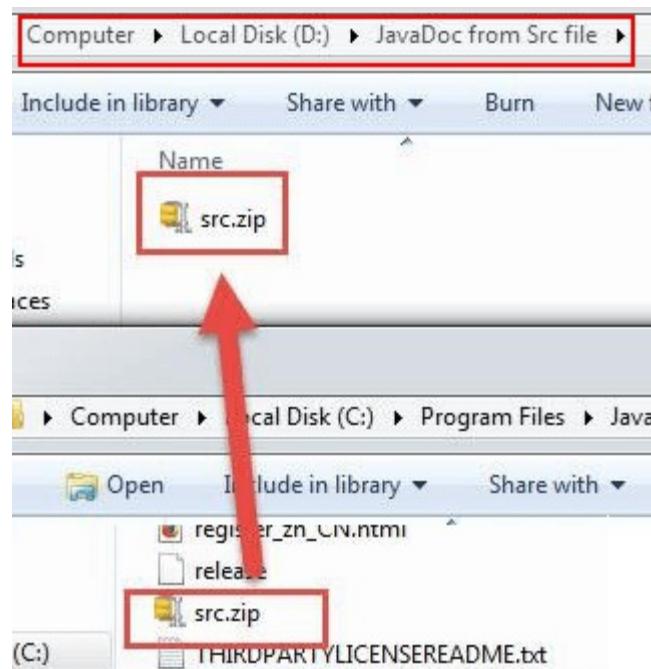
Documentation of Robot Class

Robot Class documentation will help you to understand the basic definition, syntax and usage of all methods, and functions available in

Robot Class . You can view the documentation on Official Oracle website, or you can create the documentation on your local machine.

To create the documentation on local machine, follow the steps below-

Step 1) You will find the src.zip file in JDK folder. Copy src.zip and extract the same in some other folder or directory (say D: or E:)



Step 2) Extract src folder and Navigate to (path till src folder)/**src/java.awt**

Step 3) Copy the current location of awt folder and open command prompt.

Step 4) In cmd, change your current directory location to awt folder and type 'javadoc *.java' as shown below

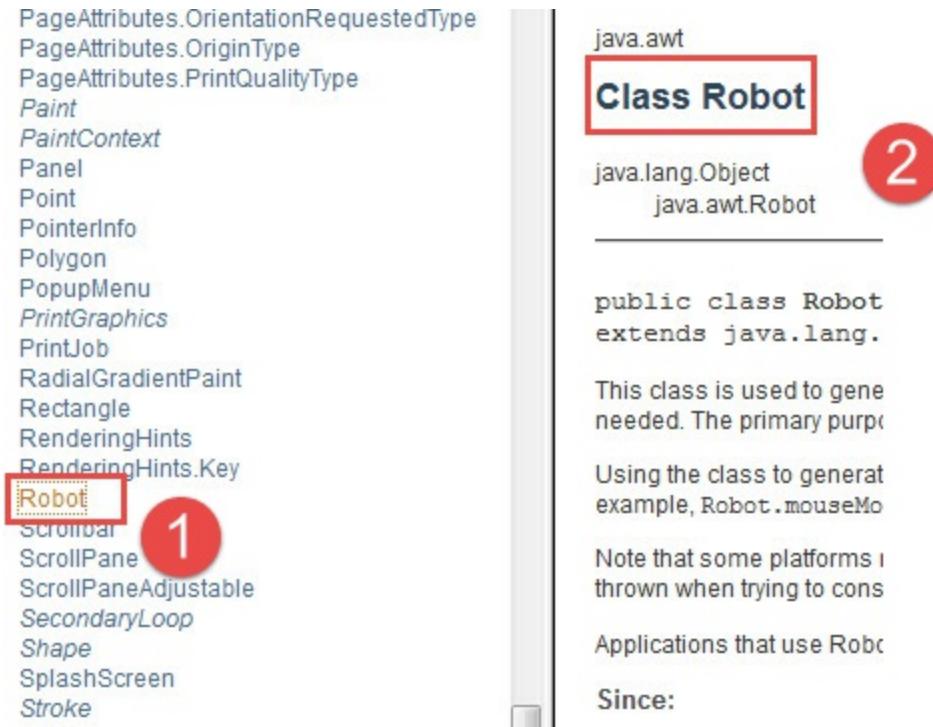


Wait a while for the system to process, once completed you will see few HTML files in awt folder.

Step 5) Open index.html

```
ndow
Window.java:3207: warning - Tag @link: can't find getCenterPoint in Graphic
ronment
Generating \java\awt\Window.AccessibleAWTWindow.html...
Generating \java\awt\Window.Type.html...
Generating \java\awt\package-frame.html...
Generating \java\awt\package-summary.html...
Generating \java\awt\package-tree.html...
Generating \constant-values.html...
Generating \serialized-form.html...
Dialog.java:129: warning - Tag @see: can't find isDisplayable() in Component
Frame.java:336: warning - Tag @see: can't find isDisplayable() in Component
Building index for all the packages and classes...
Generating \overview-tree.html...
Generating \index-all.html...
Generating \deprecated-list.html...
Building index for all classes...
Generating \allclasses-frame.html...
Generating \allclasses-noframe.html...
Generating \index.html...
Generating \help-doc.html...
5 errors
827 warnings
D:\JavaDoc from Src file\src\java\awt>index.html
```

Step 6) Here's you have full documentation of awt package, from the left navigation bar click on 'Robot' hyperlink (See 1 marked in below image).



Here you can also see all the methods and interfaces of Robot Class
(See 2 marked in above image).

Understanding Robot Class internal methods and usage

Robot Class methods can be used to interact with keyboard/mouse events while doing browser automation. Alternatively AutoIT can be used, but its drawback is that it generates an executable file (exe) which will only work on windows, so it is not a good option to use.

Some commonly and popular used methods of Robot Class during web automation:

- **keyPress(): Example:** `robot.keyPress(KeyEvent.VK_DOWN)` : This method will press down arrow key of Keyboard
- **mousePress() : Example :**

`robot.mousePress(InputEvent.BUTTON3_DOWN_MASK)` : This method will press the right click of your mouse.

- `mouseMove()` : **Example:** `robot.mouseMove(point.getX(), point.getY())` : This will move mouse pointer to the specified X and Y coordinates.

- `keyRelease()` : **Example:**

`robot.keyRelease(KeyEvent.VK_DOWN)` : This method with release down arrow key of Keyboard

- `mouseRelease()` : **Example:**

`robot.mouseRelease(InputEvent.BUTTON3_DOWN_MASK)` : This method will release the right click of your mouse

Sample code to automate common use cases using Robot Class

- Lets take example of web site
<http://spreadsheetpage.com/index.php/file/C35/P10/> wherein after you click on a web element
`(./a[@href=contains(text(),'yearly-calendar.xls'])` a O.S download pop-up appears.
- To handle this we use Robot class (by creating an instance of Robot Class in your code say **Robot robot = new Robot()** . Robot class us present in AWT package of JDK.
- To press down arrow key of Keyboard we use **(robot.keyPress(KeyEvent.VK_DOWN))**
- To press TAB key of keyboard (we use **robot.keyPress(KeyEvent.VK_TAB))**
- To press Enter key we use **(robot.keyPress(KeyEvent.VK_ENTER)).**

Here is a sample code

```
import java.awt.AWTException;
import java.awt.Robot;
import java.awt.event.KeyEvent;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

class Excercise1 {

    public static void main(String[] args) throws
AWTException, InterruptedException {
        WebDriver driver = new FirefoxDriver();

driver.get("http://spreadsheetpage.com/index.php/file/C35/P10/")
; // sample url

driver.findElement(By.xpath("./a[@href=contains(text(), 'yearly-
calendar.xls')]]")).click();
        Robot robot = new Robot(); // Robot class throws AWT
Exception
        Thread.sleep(2000); // Thread.sleep throws
InterruptedException
        robot.keyPress(KeyEvent.VK_DOWN); // press arrow
down key of keyboard to navigate and select Save radio button

        Thread.sleep(2000); // sleep has only been used to
showcase each event separately
        robot.keyPress(KeyEvent.VK_TAB);
        Thread.sleep(2000);
        robot.keyPress(KeyEvent.VK_TAB);
        Thread.sleep(2000);
        robot.keyPress(KeyEvent.VK_TAB);
        Thread.sleep(2000);
        robot.keyPress(KeyEvent.VK_ENTER);
        // press enter key of keyboard to perform above selected
action
    }
}
```

How to execute Robot Class code

using TestNG

Since, now you aware of basic methods of Robot Class so let's understand few more complex methods -

Suppose you **do not** want to use the **click method** for clicking at web element.

In such cases, you can use mouseMove method of the Robot class.

Step 1) mouseMove method takes x and y coordinates as parameters like **robot.mouseMove(630, 420)** where 630 indicates x-axis and 420 indicate y-axis. So, this method will move your mouse pointer from the current location to mentioned x and y intersection point.

Step 2) Next, we need to press the mouse button. We can use the method **mousePress** like

robot.mousePress(InputEvent.BUTTON1_DOWN_MASK) .

Step 3) After press, the mouse needs to be released. We can use **robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK)** in order to release left click of a mouse.

Running code using testNG:

Running code using Testng requires maven dependency of testNG or referenced library of TestNG jar file.

TestNG maven dependency:

```
<dependency>
<groupId>org.testng</groupId>
<artifactId>testng</artifactId>
<version>6.1.1</version>
```

```
</dependency>
```

After adding maven dependency or jar file. You need to import Test annotation of testNG. Once it is all done, just Right click on the program code and click on Run As then click on TestNG... and you will find that code will start its execution using testNG API.

Here is the code

```
import java.awt.AWTException;
import java.awt.Robot;
import java.awt.event.InputEvent;
import java.awt.event.KeyEvent;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.Test;

public class Excercise1 {

    @Test
    public static void execution() throws InterruptedException,
    AWTException {
        WebDriver driver = new FirefoxDriver();
        driver.manage().window().maximize();

        driver.get("http://spreadsheetpage.com/index.php/file/C35/P10/")
        ; // sample url
        Robot robot = new Robot();
        robot.mouseMove(630, 420); // move mouse point to
        specific location
        robot.delay(1500); // delay is to make code wait
        for mentioned milliseconds before executing next step
        robot.mousePress(InputEvent.BUTTON1_DOWN_MASK); // press
        left click
        robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK); // release left click
        robot.delay(1500);
        robot.keyPress(KeyEvent.VK_DOWN); // press keyboard
        arrow key to select Save radio button
        Thread.sleep(2000);
        robot.keyPress(KeyEvent.VK_ENTER);
```

```
        // press enter key of keyboard to perform above selected
action
    }
}
```

Disadvantages of Robot Class

Robot framework has few disadvantages mentioned below:

1. Keyword/mouse event will only work on current instance of Window. E.g. suppose a code is performing any robot class event, and during the code execution user has moved to some other screen then keyword/mouse event will occur on that screen.
2. Most of the methods like mouseMove are screen resolution dependent so there might be a chance that code working on one machine might not work on other.

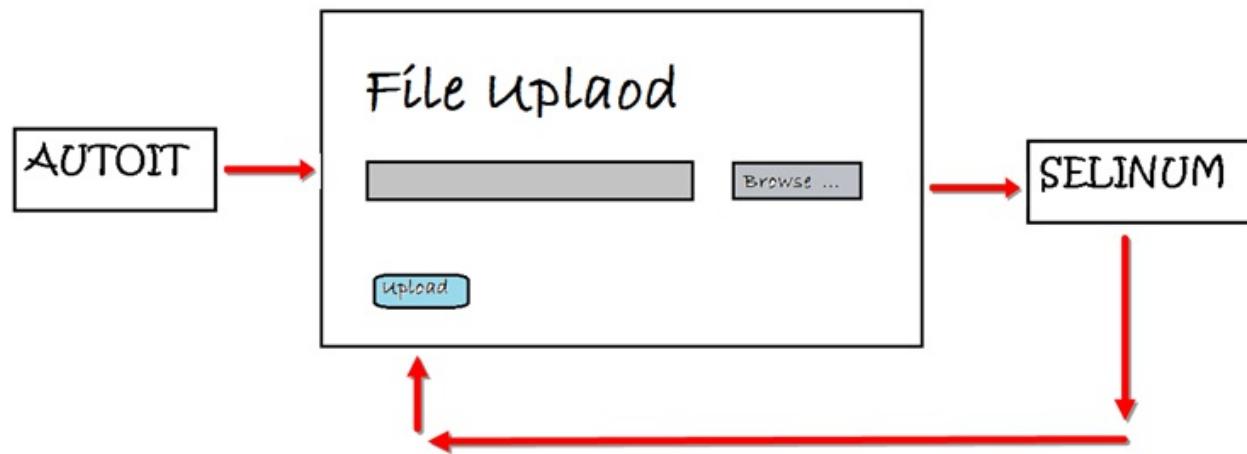
Summary

Robot class in AWT package is used to generate keyboard/mouse events to interact with OS windows and native apps.

The primary purpose of Robot is to support selenium automated tests project build in Java platform

Chapter 38: How to use AutoIT with Selenium

Selenium is an open source tool that is designed to automate web-based applications on different browsers but to handle window GUI and non HTML popups in application. AutoIT is required as these window based activity are not handled by Selenium.



AutoIt v3 is also freeware. It uses a combination of mouse movement, keystrokes and window control manipulation to automate a task which is not possible by selenium webdriver.

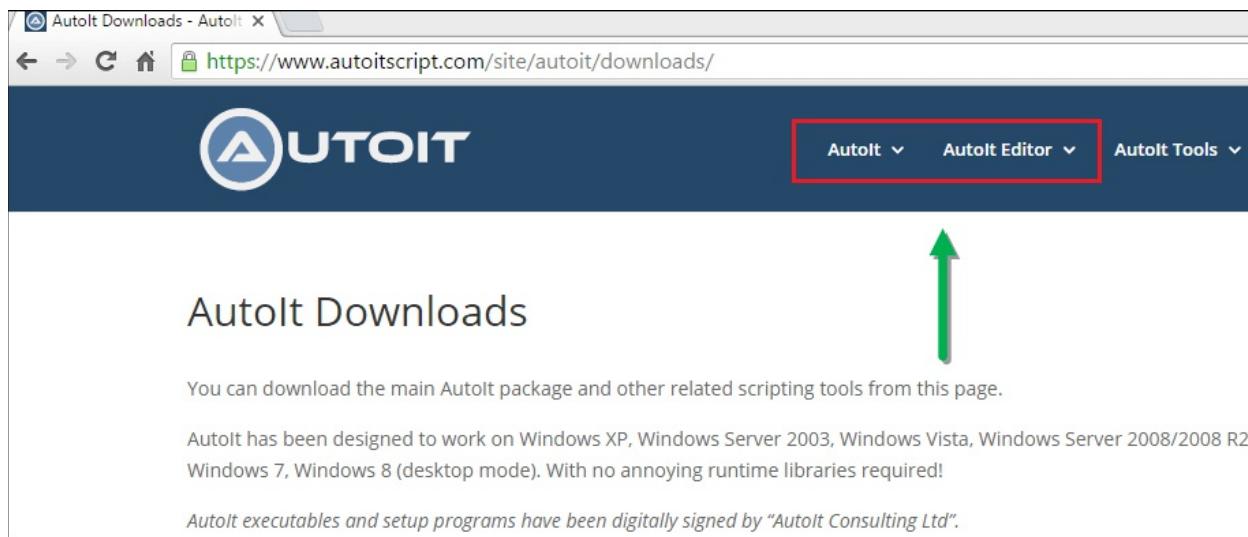
Moving ahead we will learn how to upload a file in selenium webdriver using autoIT. Here we need three tools in order to this.

- Selenium Webdriver
- AutoIT editor and element identifier
- The window that you want to automate

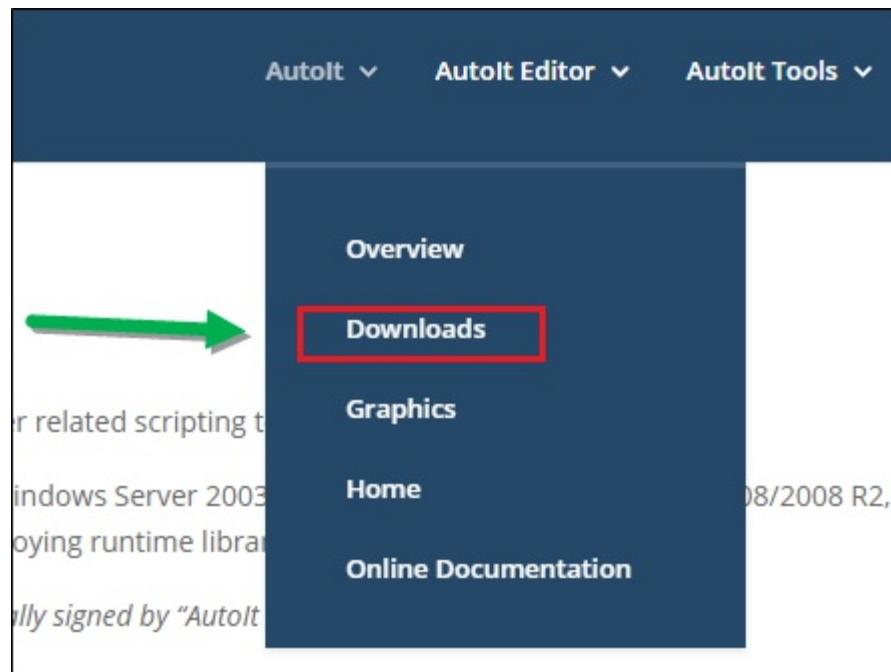
How to download and install AutoIT

Step 1): Go to this link.

Step 2): Hover on 'Autoit' and 'Autoit Editor' dropdown.



Step 3) Click on 'AutoIT' Downloads option.



Step 4): Download "Autoit" by clicking on 'Download Autoit' button .

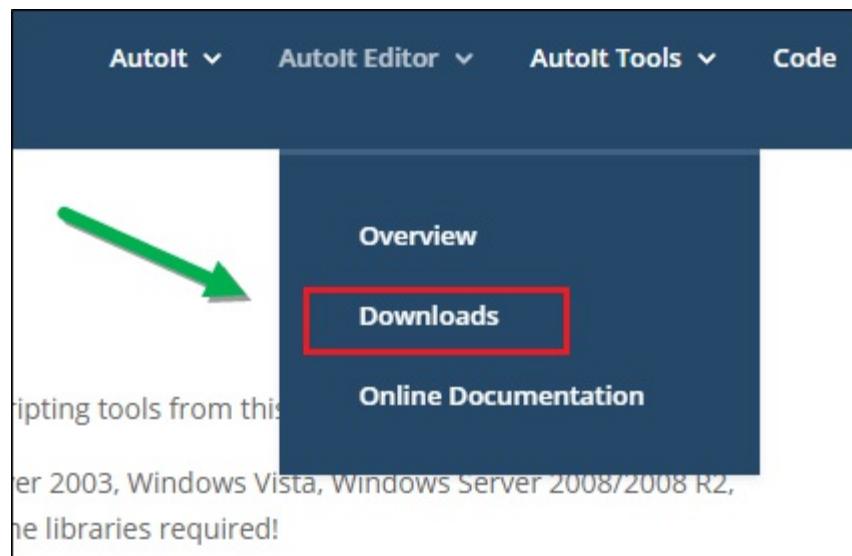
Current Versions

✓ **Latest version:** v3.3.14.2

Updated: 18 September, 2015

History: [View changelog](#)

Software	Download
<p>AutoIt Full Installation. Includes x86 and x64 components, and:</p> <ul style="list-style-type: none">• AutoIt program files, documentation and examples.• Aut2Exe – Script to executable converter. Convert your scripts into standalone .exe files!• AutoItX – DLL/COM control. Add AutoIt features to your favorite programming and scripting languages! Also features a C# assembly and PowerShell CmdLets.• Editor – A cut down version of the SciTE script editor package to get started. Download the package below for the full version!	

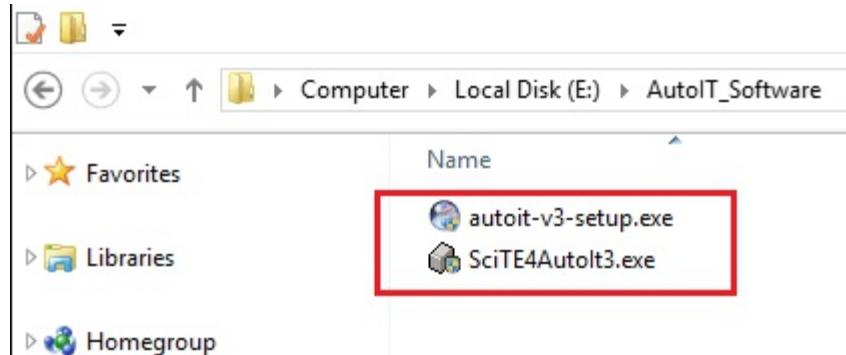
Step 5): Now download "Autoit editor" by clicking on 'Downloads' button .

Step 6): Click on the link as shown below.

The screenshot shows a web browser window with the URL <https://www.autoitscript.com/site/autoit-script-editor/downloads/>. The page title is "AutoIt Script Editor Downloads". Below it, there's a section titled "Current Versions" with a table. The first row of the table has a red border around the "File" column, which contains the link [SciTE4AutoIt3.exe \(5151Kb\)](#). A green arrow points to this link. The table has three columns: "File", "Date updated", and "Notes". The "Notes" column for the first row states: "Installer containing SciTE and all configuration files plus utilities. [Update History](#). Definition files included: AutoIt v3.3.14.2 and BETA v3.3.15.0".

File	Date updated	Notes
SciTE4AutoIt3.exe (5151Kb)	20-9-2015	Installer containing SciTE and all configuration files plus utilities. Update History . Definition files included: AutoIt v3.3.14.2 and BETA v3.3.15.0

After download you will get two setup file as shown in below screen, first is **AutoIt version 3** setup and second is **Scitaautoit3** .

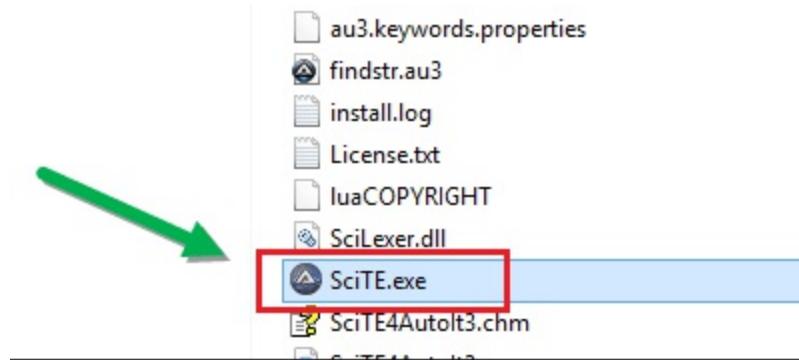


Step 6): For Installing AutoIT-Click on both AutoIT setup one by one

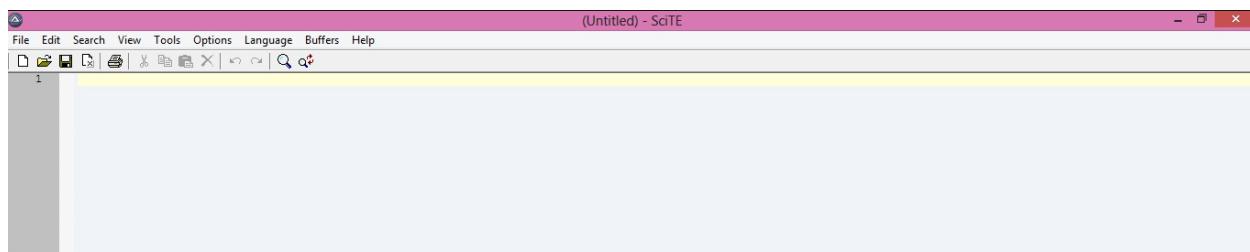
.

Step 7): After successfully installation - open up AutoIT Editor.

Go to 'C:\Program Files (x86)\AutoIt3\SciTE'

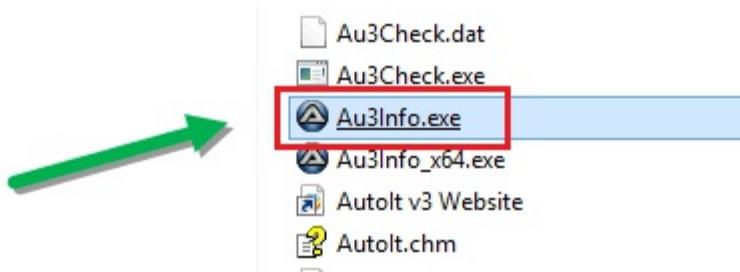


and click on 'SciTE.exe' file, the AutoIT editor opens as shown in below screen.

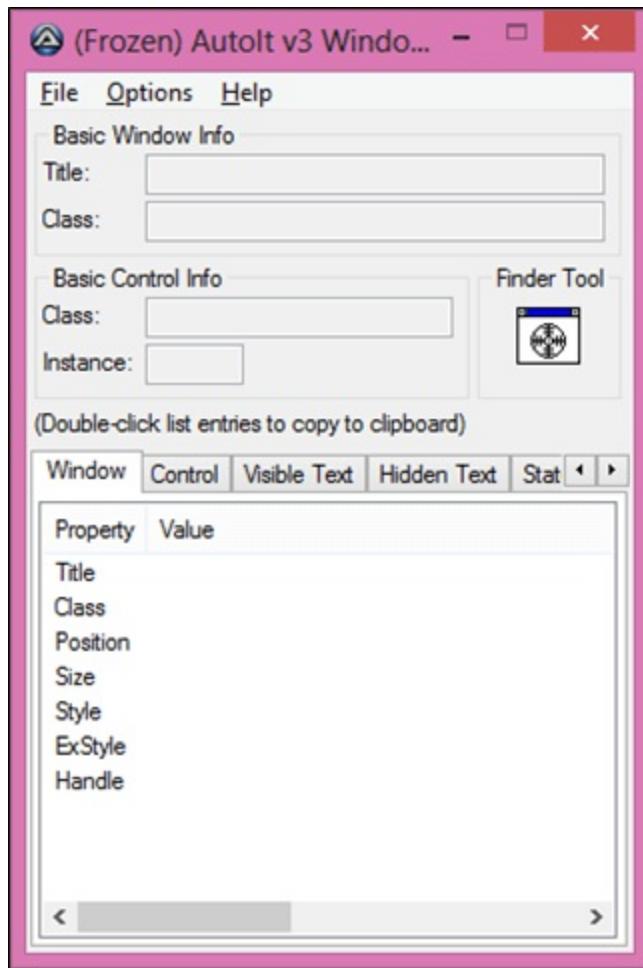


Step 8) : Now opens element Identifier .

Go to 'C:\Program Files (x86)\AutoIt3 '



And click on 'Au3Info.exe' file, the element identifier opens as shown in below screen.



Note: Once you done with this element identifier you need to close manually, it will not close automatically.

Finding element through element Identifier and writing script on AutoIT editor.

Under this, we will see how to find element on file uploader window through AutoIT Element Identifier (Element identifier is a tool like selenium IDE, identifier find the element of window GUI or non HTML popups and provide the attribute of element like **title**, **class**,

instance) and how to write script on AutoIT editor using 3 methods.

For Example: We will use "Write to us" page of guru99 to upload resume (Doc file).

After clicking on 'Choose File' button from the "Write to us" page, we need to call AutoIT script. The control immediately transferred to autoit after clicking 'Choose File' by the below statement which takes care of uploading part.

```
Runtime.getRuntime().exec("E:\\AutoIT\\FileUpload.exe");
```

Finally, when we run selenium script-it will fill the form-> upload resume-> Submit form.

💡Knowledge shared is Knowledge Multiplied

[Create A Course](#) [Create on Article](#)

Write For Us

Name *

E-mail *

Upload Your Resume *

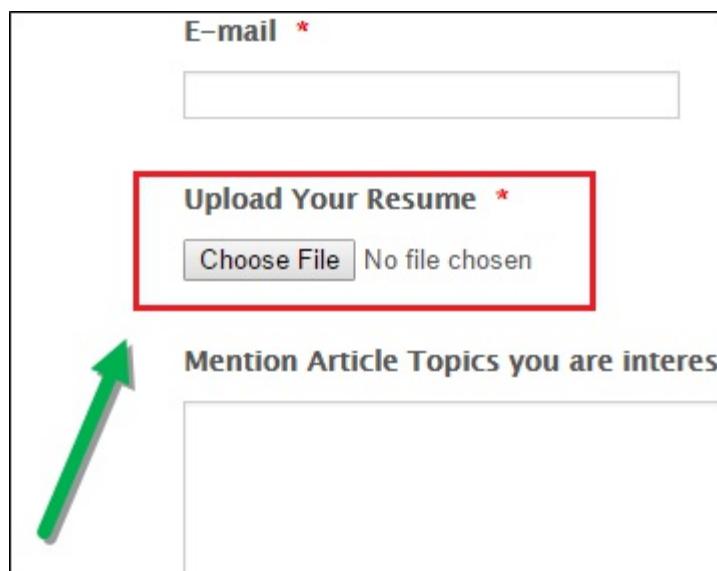
No file chosen

Mention Article Topics you are interested in writing *

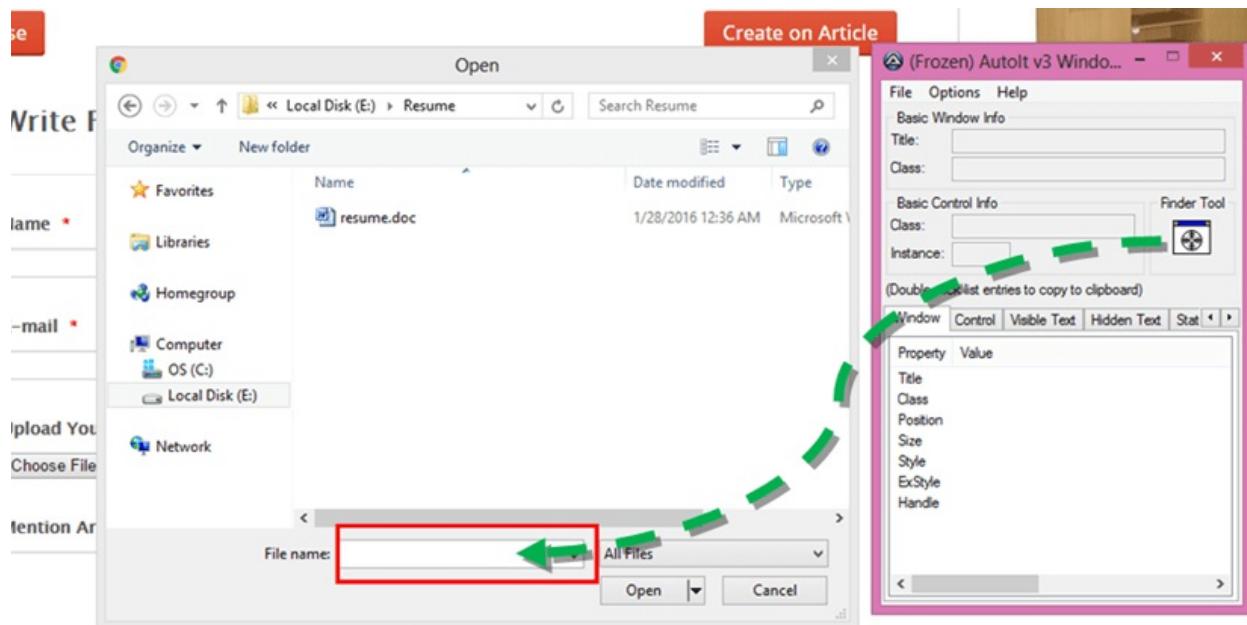
Step 1): Now open element Identifier- Go to 'C:\Program Files (x86)\AutoIt3' and click on 'Au3Info.exe' file, the element identifier window opens as shown in below screen.

The screenshot shows a web browser window with the URL www.guru99.com/become-an-instructor.html. The page title is "Knowledge shared is Knowledge Multiplied". There are two main buttons at the top: "Create A Course" and "Create on Article". Below them is a section titled "Write For Us". This section contains fields for "Name *", "E-mail *", and "Upload Your Resume *". The "Upload Your Resume" field has a "Choose File" button which is highlighted with a red box. A green arrow points from the bottom left towards this red box. To the right of the form, a separate window titled "(Frozen) AutoIt v3 Windo..." is open, displaying basic window and control information. At the bottom of the page is a "Submit" button.

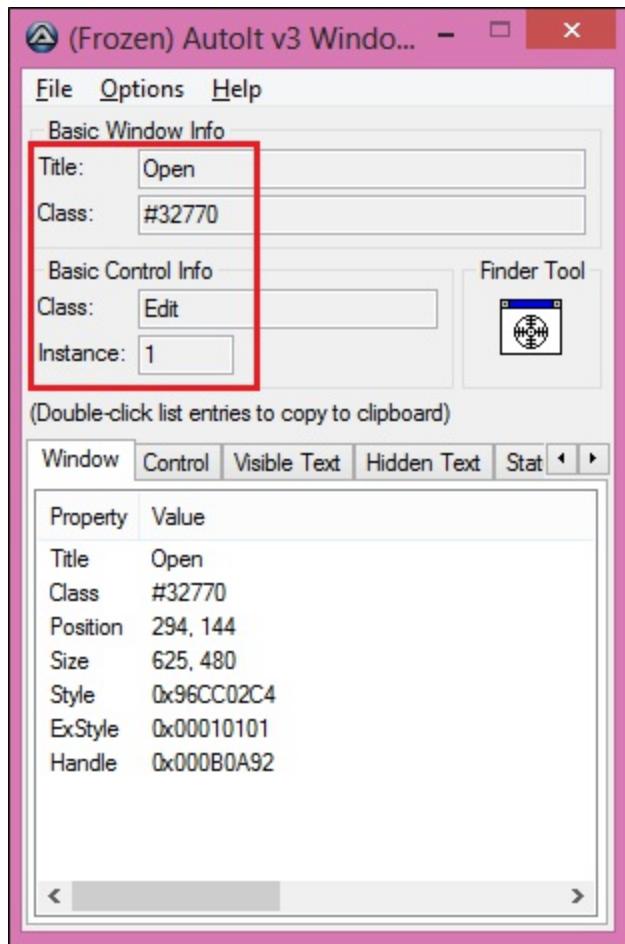
Step 2): Now open file uploader window by clicking on 'Choose File' which is windows activity.



Step 3): Drag the finder tool on the " File Name" box element of file uploader window to find the basic attributes info as shown in the below screen with the arrow.



We can get the value of attributes i.e. **title='Open'**, **class='Edit'** and **instance='1'** as shown below. These values are used in writing AutoIT script as explained in below step 4.



Step 4: Now open AutoIT script editor, goto 'C:\Program Files (x86)\AutoIt3\SciTE' and click on 'SciTE.exe' as shown in step 7 from the 1st topic.

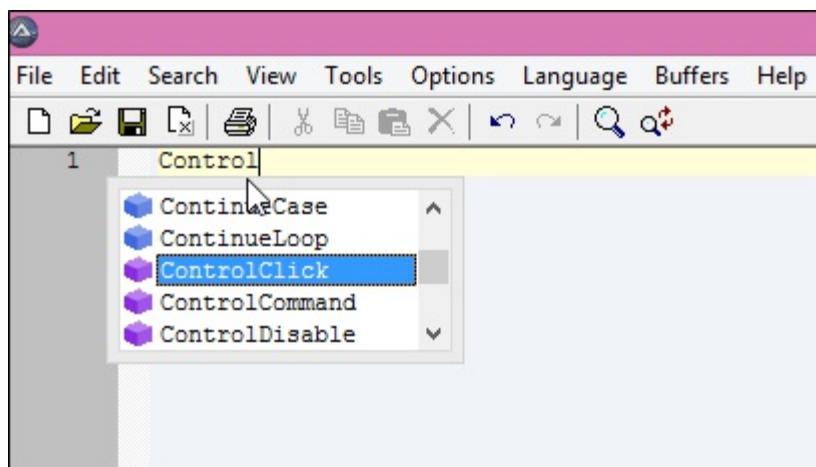
Start writing a script for selecting a file to upload.

There are lots of method available which we can use in a script according to the requirement, but right now we will focus on the below methods as these methods are required for writing file upload script:

1. **ControlFocus(" title "," text ",controlID) //Sets input focus to a given control on a window.**
2. **ControlSetText(" title "," text ",controlID , " File path which need to upload ") // Sets text of a control.**

3. ControlClick(" title "," text ",controlID) //Sends a mouse click command to a given control.

You can see a number of methods are displayed as shown in below screen. The good feature of AutoIT is that it is somewhat like Eclipse that suggests you some of the methods.



Here in the AutoIT editor, we have selected "**control focus**" method. Element identifier is already opened and minimized as the element is already identified in above step 3. We can open it by maximizing it.

Now, we will take the values from element identifier for 'ControlFocus' and 'ControlSetText' methods as these methods works on same element i.e. 'File name' text box but for 'ControlClick' method need to capture values of different element i.e. 'Open' button.

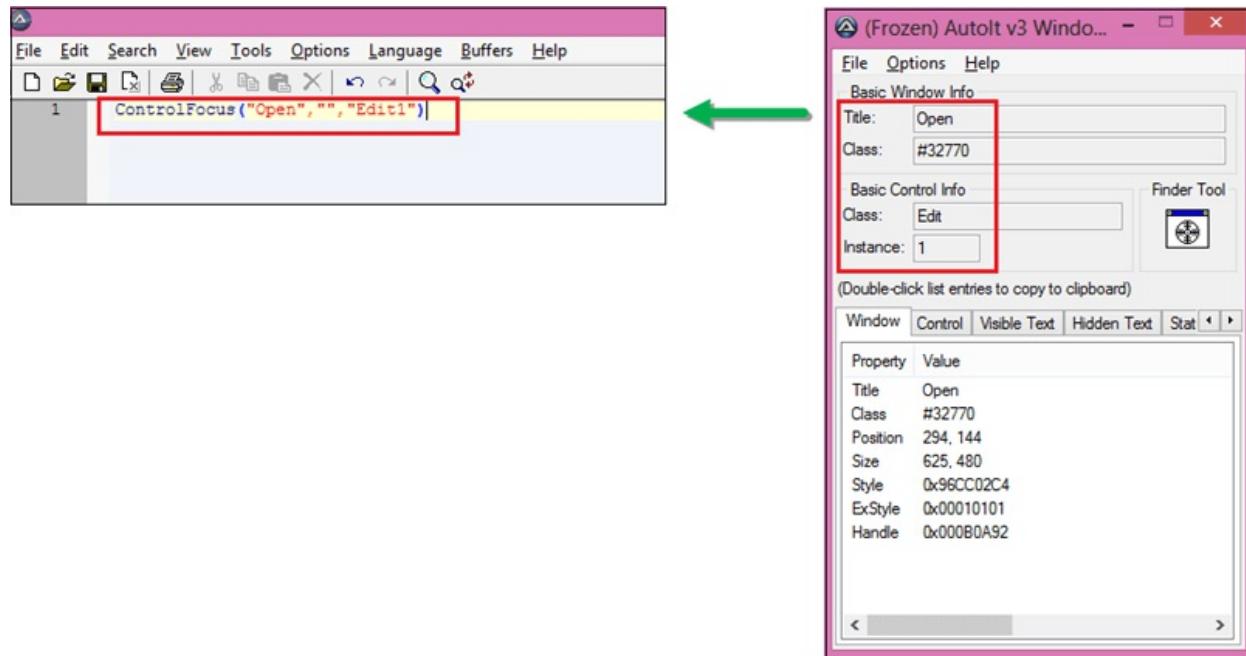
Parameter values for **ControlFocus** method:

This method sets focus to the 'file name' text box of the file uploader window.

- 1st parameter **title** is " Open ".
- We ignore 2nd parameter, the **text** is not required.

- 3rd parameter **controlID** is the combination of class='Edit' and Instance='1' i.e., . 'Edit1.'

```
ControlFocus("Open", "", "Edit1") // This method sets input focus to 'File name' text box.
```

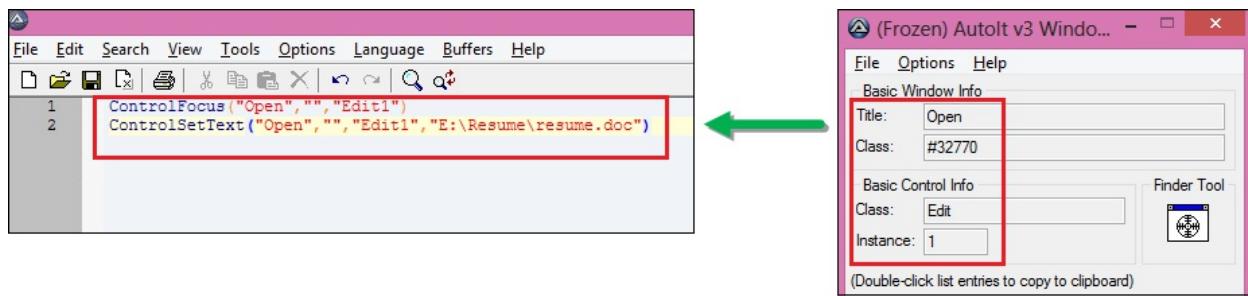


Parameter values for **ControlSetText** method :

This method is used to define the path of a file which we need to upload in 'file name' text box. In another way, we can say that this method is used to set the text to the input element.

- 1st parameter **title** is " Open ".
- We ignore 2nd parameter, the **text** is not required.
- 3rd parameter **controlID** is the combination of class='Edit' and Instance='1' i.e., " Edit1 ".
- 4th parameter **new text**, we pass the path of the file which we need to upload.

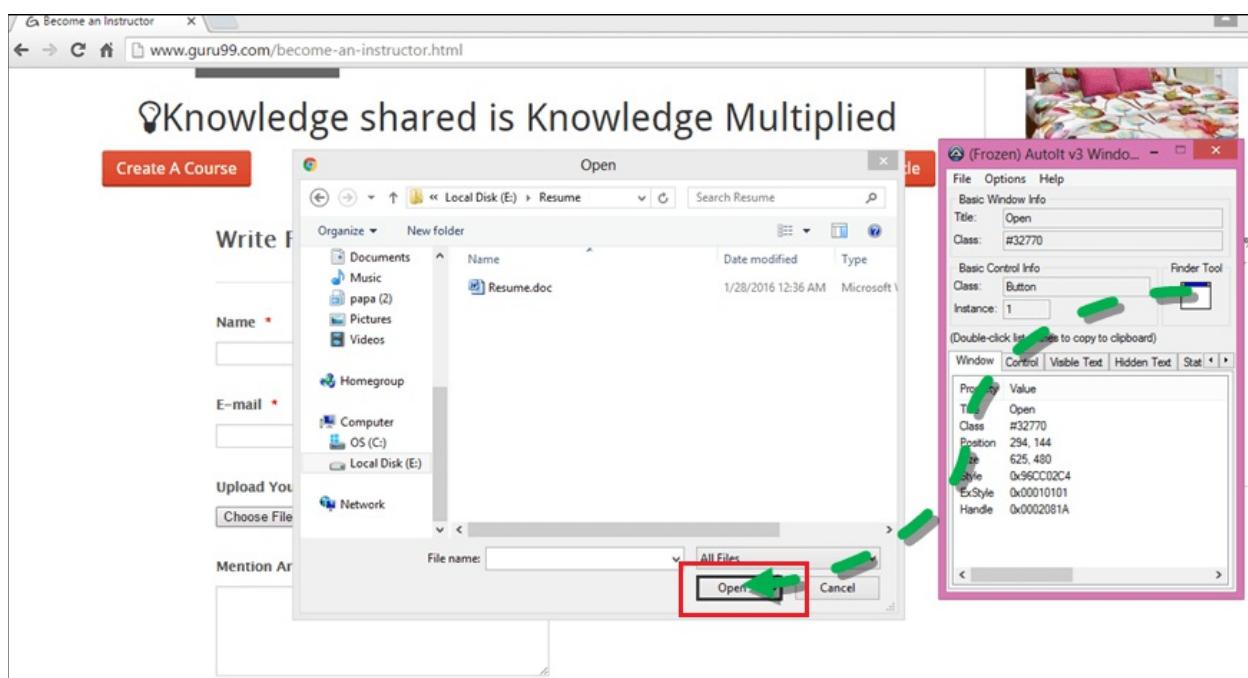
```
ControlSetText("Open", "", "Edit1", "E:\Resume\resume.doc") // This method input file path of a control.
```



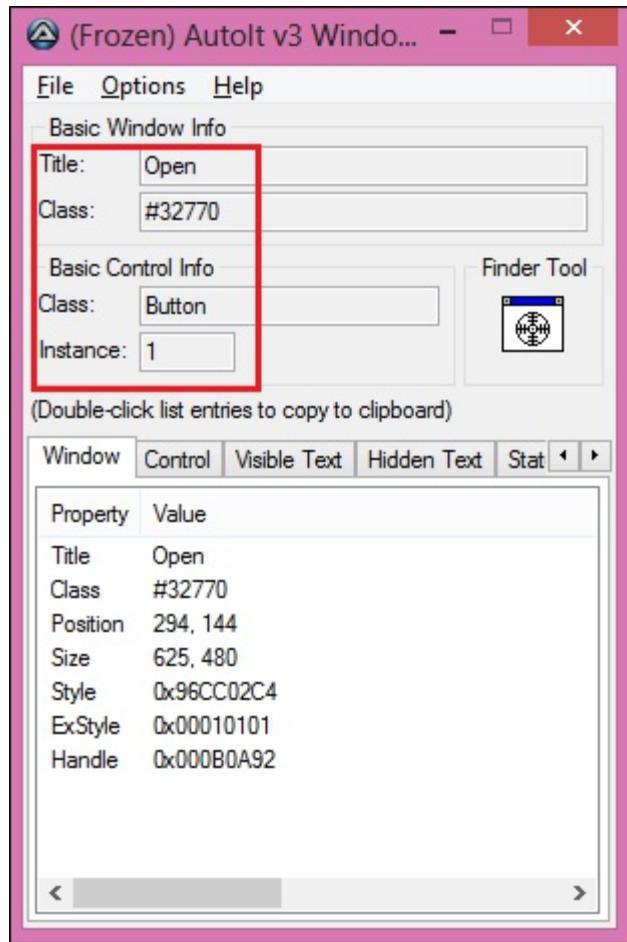
After following the above step, don't close the windows (editor and element identifier), keep it remain open. You again need to open file uploader window as to find attributes of 'Open' Button as shown in below step 5.

Step 5: Now drag the finder tool on the "Open" button element of file uploader window to find the basic attribute information.

Previous values (i.e. attributes of 'File name' text box) overwrite with new values of 'Open' button. You can see the class attribute is now changed to "button" which was previously "edit" in AutoIT element identifier window.



We can get the value of attributes i.e. **title='Open'**, **class='Button'** and **instance='1'** as shown below. These values are used in writing Autoit script as explained in below.

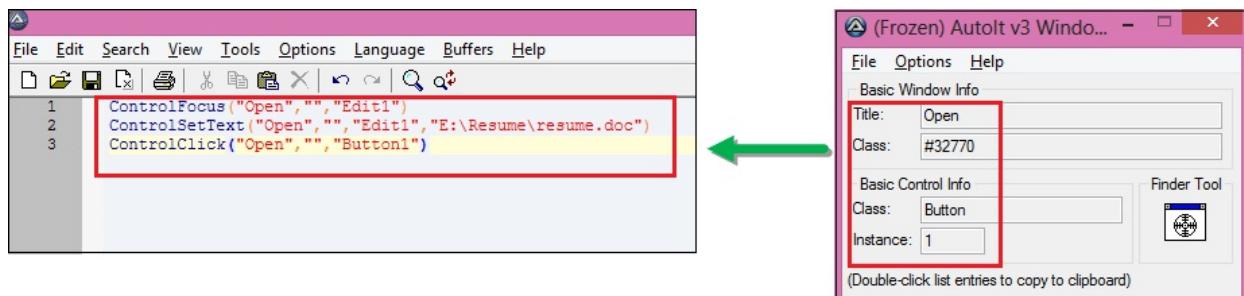


Parameter values for **ControlClick** method:

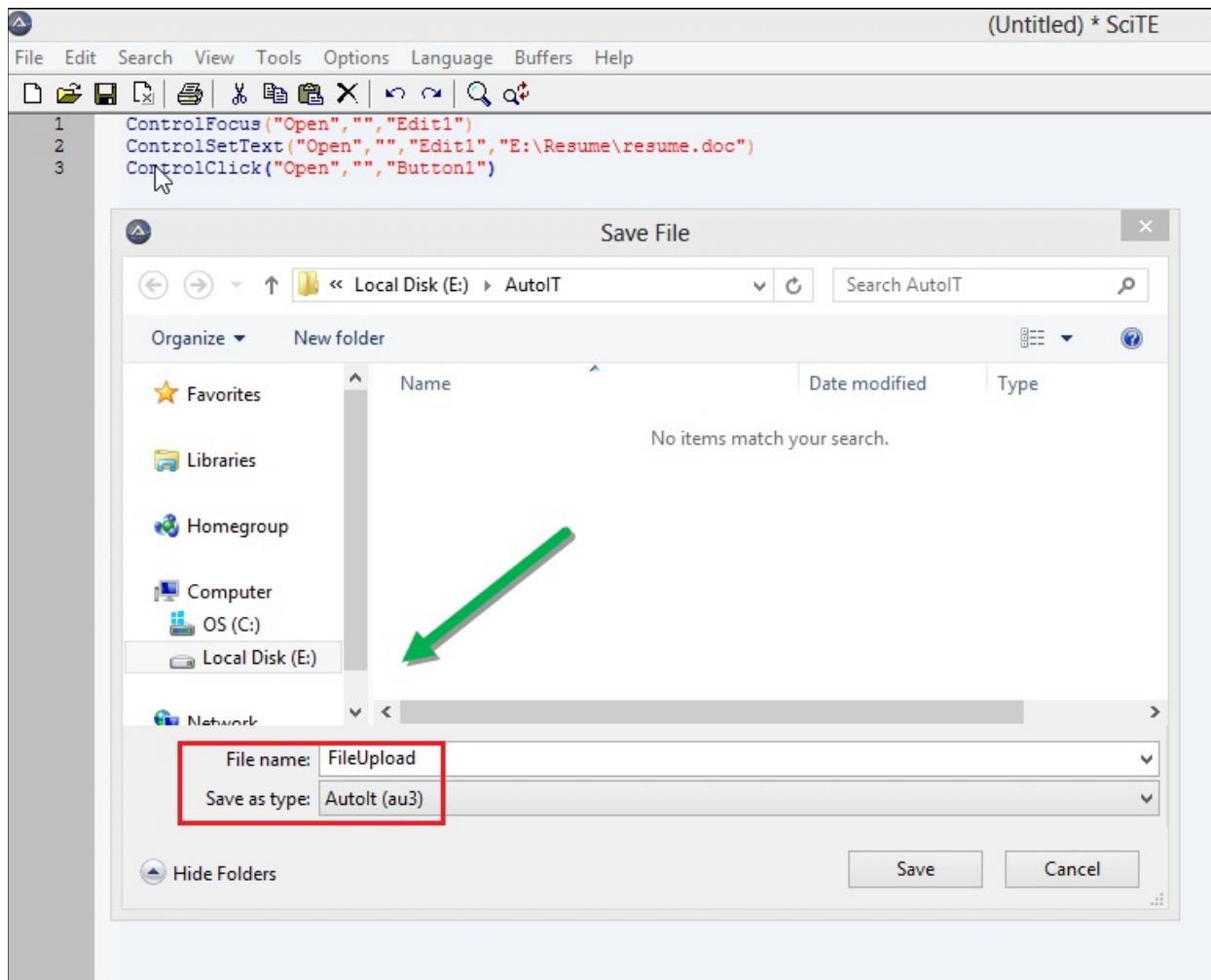
This method clicks on 'Open' button of the file uploader window.

- 1st parameter **title** is " Open ".
- We ignore 2nd parameter; the **text** is not required.
- 3rd parameter **controlID** is the combination of class and Instance, i.e., " Button1 ".

```
ControlClick("Open", "", "Button1") //This method click on 'Open'
button of file uploader.
```

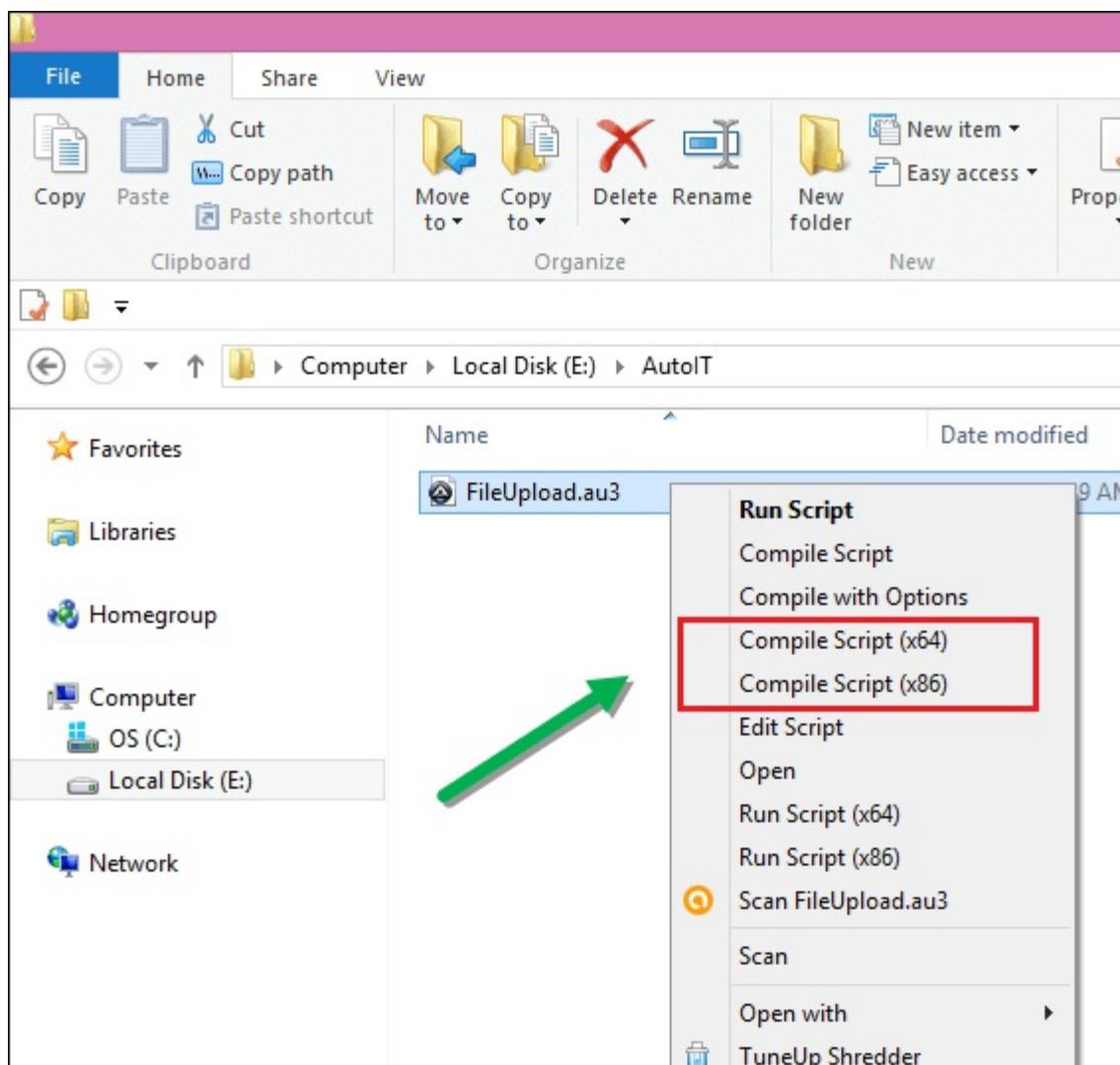


Step 6): You can see in below screen that AutoIT script is completed to handle file uploader. Now you can close the element identifier and save the script as " FileUpload " at the given location (E:\AutoIT).

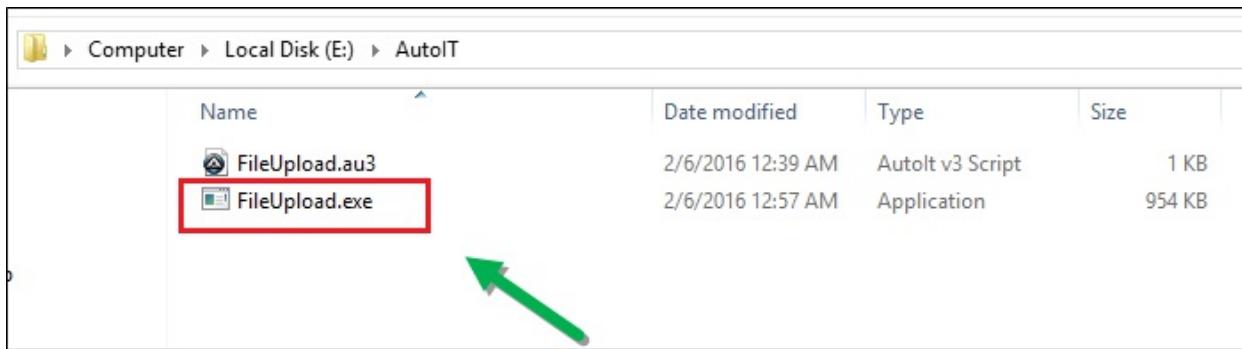


Now you can't execute this script directly, you need to compile this script.

For compiling this script, you have two options " **compile script x64** " and " **compile script x86** ", if you have windows 32-bit machine then u go with " **compile script x86** " and for windows 64-bit machine then u go with " **compile script x64** ."



Step 7): 'FileUpload exe' file generated after compilation, you can see in the below screen. Now we can use this file in Selenium webdriver script.

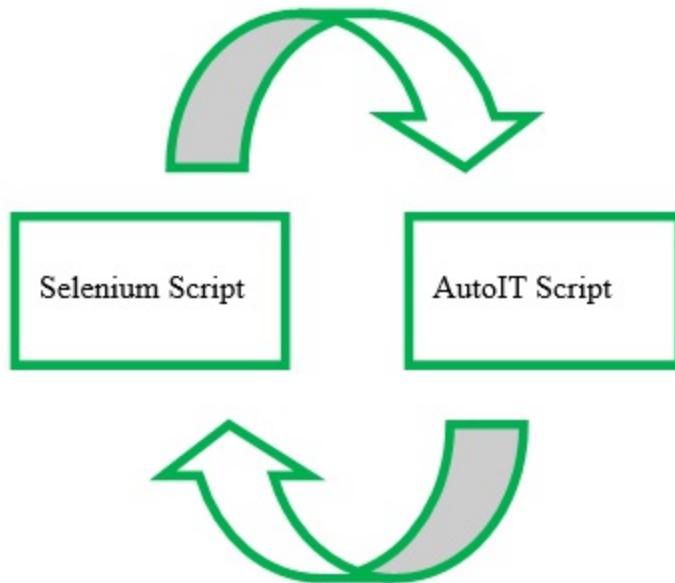


Now we will use this AutoIT script in Selenium web driver. Check below for output.

AutoIT Upload file in Selenium Webdriver

In Selenium script, we find the elements of the form and fill the data in each element as required and upload 'resume.doc' file by executing AutoIT exe file generated from AutoIT script and then allow to submit the form in selenium script.

- Open Eclipse and start writing code.
- When selenium clicks on Choose File button, file uploader box opens.
- Then we need to call AutoIT script, the control immediately transferred to AutoIT in order to upload a file and then control send back to selenium as shown below.



Step 1): Develop selenium script in eclipse.

- **Runtime** class allows the script to interface with the environment in which the script is running.
- **getRuntime()** get the current runtime associated with this process.
- **exec()** methods execute the AutoIT script (FileUpload.exe) .

```
Runtime.getRuntime().exec("E:\\\\AutoIT\\\\FileUpload.exe");
```

above line will call **AutoIT script** in selenium and upload file .

```

import java.io.IOException;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class FileUpload {
    public static void main(String[] args) throws IOException {
        WebDriver driver=new FirefoxDriver();
        driver.get("http://www.guru99.com/become-an-instructor.html");
        driver.findElement(By.id("postjob")).click();
        driver.findElement(By.id("input_3")).sendKeys("Gaurav");
        driver.findElement(By.id("id_4")).sendKeys("test.test@gmail.com");
        driver.findElement(By.id("input_4")).click();
        // below line execute the AutoIT script .
        Runtime.getRuntime().exec("E:\\AutoIT\\FileUpload.exe");
        driver.findElement(By.id("input_6")).sendKeys("AutoIT in Selenium");
        driver.findElement(By.id("input_2")).click();
        driver.close();
    }
}

```



Step 2) : Execute the Selenium script in Eclipse.

```

import java.io.IOException;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class FileUpload {
    public static void main(String[] args) throws IOException {
        WebDriver driver=new FirefoxDriver();
        driver.get("http://demo.guru99.com/test/autoit.html");
        driver.findElement(By.id("postjob")).click();

        driver.findElement(By.id("input_3")).sendKeys("Gaurav");
        driver.findElement(By.id("id_4")).sendKeys("test.test@gmail.com")
    }
}

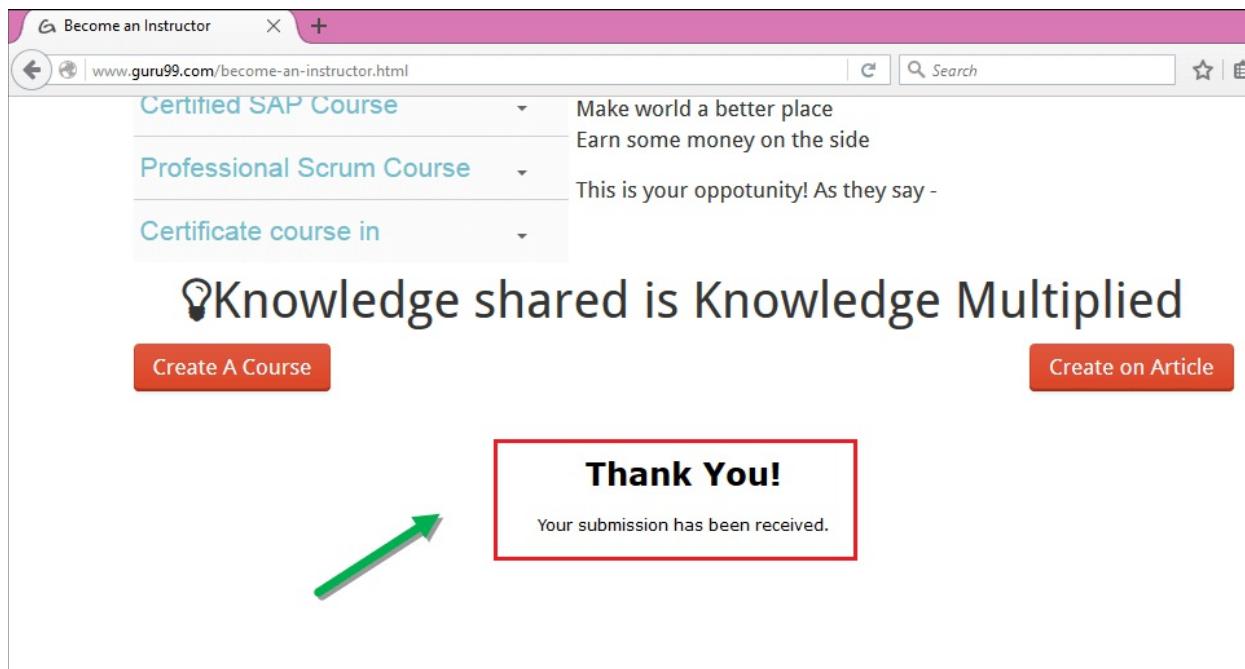
```

```

);
    driver.findElement(By.id("input_4")).click();
    // below line execute the AutoIT script .
    Runtime.getRuntime().exec("E:\\AutoIT\\FileUpload.exe");
    driver.findElement(By.id("input_6")).sendKeys("AutoIT in
Selenium");
    driver.findElement(By.id("input_2")).click();
    driver.close();
}
}
}

```

Step 3): Verify the output, resume.doc file uploaded successfully and thank you message will be displayed.



Conclusion:

- Downloaded and installed Element Identifier and AutoIT editor.
- Opened the site on which to do the operation.
- Element Identifier identifies the elements of file uploader window.
- Prepared AutoIT script in the editor with the help of Element

identifier.

- Autoit script is used in selenium webdriver script.
- Executed the selenium script.
- Output: Successfully the file uploaded.

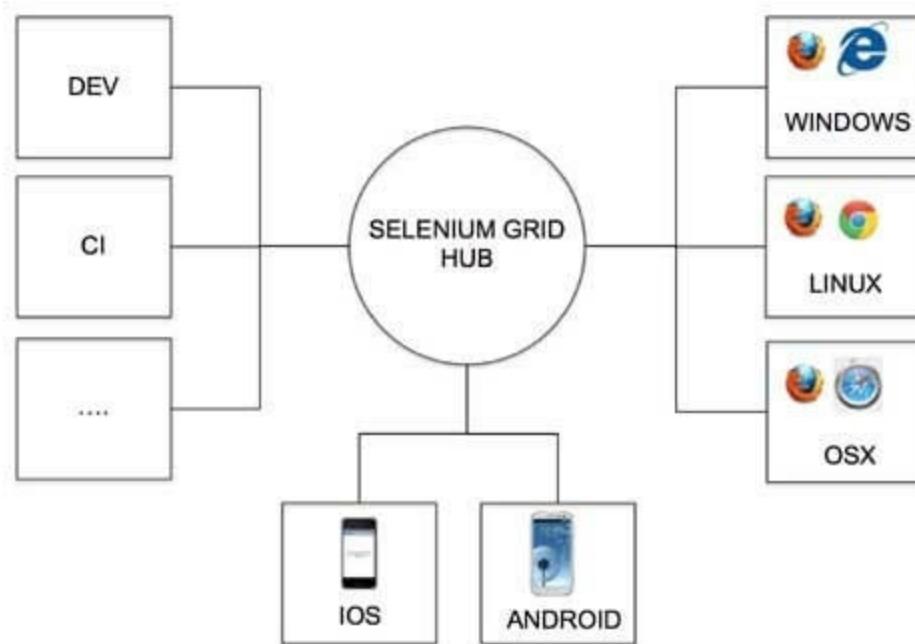
Chapter 39: Desired Capabilities in Selenium

Every Testing scenario should be executed on some specific testing environment. The testing environment can be a web browser, Mobile device, mobile emulator, mobile simulator, etc.

The Desired Capabilities Class helps us to tell the webdriver, which environment we are going to use in our test script.

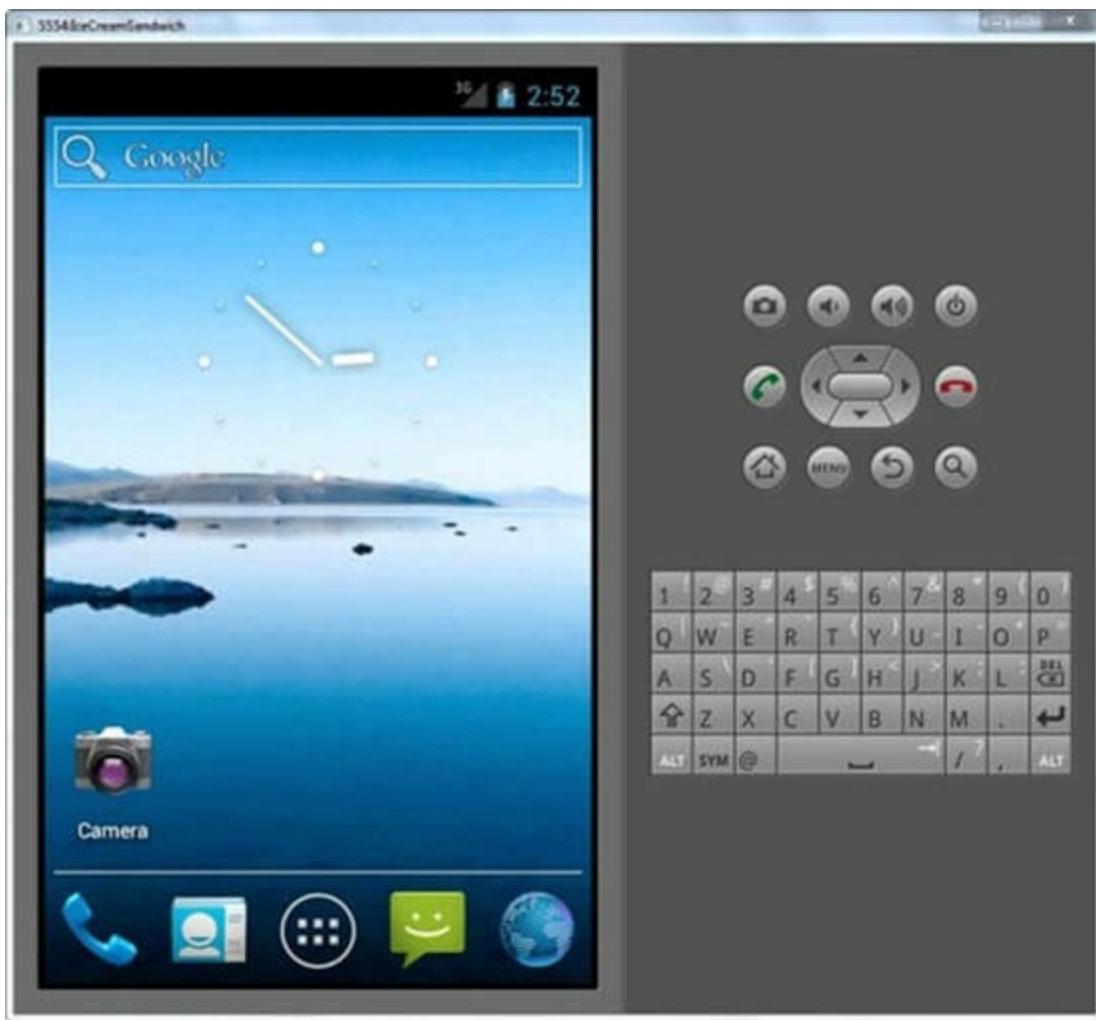
The **setCapability method** of the DesiredCapabilities Class, which is explained in the later part of the tutorial, can be used in Selenium Grid. It is used to perform a parallel execution on different machine configurations.

Ex: Grid



It is used to set the browser properties (Ex. Chrome, IE), Platform Name (Ex. Linux, Windows) that are used while executing the test cases.

In the case of mobile automation, as we perform the tests on different varieties of mobile devices, the Mobile Platform (ex. iOS, Android) Platform Version (Ex. 3.x,4.x in Android) can be set.



The above emulator example shows the platform set which is android and the platform version set which is IceCream Sandwich (4.x).

What is Desired Capability

The desired capability is a series of key/value pairs that stores the browser properties like browsername, browser version, the path of the browser driver in the system, etc. to determine the behaviour of the browser at run time.

- Desired capability can also be used to configure the driver instance of Selenium WebDriver.
- We can configure driver instance like FirefoxDriver, ChromeDriver, InternetExplorerDriver by using desired capabilities.

Desired Capabilities are more useful in cases like:

- In mobile application automation, where the browser properties and the device properties can be set.
- In Selenium grid when we want to run the test cases on a different browser with different operating systems and versions.

Different types of Desired Capabilities Methods

Here we will see a different type of desired capabilities methods and see how to use one of this method "**setCapability Method**".

1. **getBrowserName()**

```
public java.lang.String getBrowserName()
```

2. **setBrowserName()**

```
public void setBrowserName(java.lang.String browserName)
```

3. **getVersion()**

```
public java.lang.String getVersion()
```

4. **setVersion()**

```
public void setVersion(java.lang.String version)
```

5. **getPlatform()**

```
public Platform getPlatform()
```

6. **setPlatform()**

```
public Platform getPlatform()
```

7. **getCapability Method**

The getCapability method of the DesiredCapabilities class can be used to get the capability that is in use currently in the system.

```
public java.lang.Object getCapability(java.lang.String  
capabilityName)
```

8. **setCapabilityMethod**

The setCapability() method of the Desired Capabilities class can be used to set the device name, platform version, platform name, absolute path of the app under test (the .apk file of the app(Android) under test), app Activity (in Android) and appPackage(java).

"**setCapability method**" in Java has the below declarations:

```
setCapability : public void setCapability(java.lang.String  
capabilityName,boolean value)
```

```
setCapability :public void setCapability(java.lang.String  
capabilityName,java.lang.String value)
```

```
setCapability :public void setCapability(java.lang.String  
capabilityName,Platform value)
```

```
setCapability :public void setCapability(java.lang.String  
key,java.lang.Object value)
```

Example for set capability method

Let us consider an example where we want to run our Test Case on Internet explorer browser to open www.gmail.com website using Selenium Webdriver.

Following is the code.

```
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.ie.InternetExplorerDriver;  
  
public class IETestforDesiredCapabilities {  
  
    public static void main(String[] args) {  
  
        WebDriver IEdriver = new InternetExplorerDriver();  
        driver.manage().window().maximize();  
        driver.get("http://gmail.com");  
  
        driver.quit();  
    }  
  
}
```

Now run this code from Eclipse and check out the console.

Output:

It will throw the following error when above code is executed. The error occurs because the path to the browser driver (IE in the above case) is not set. The browser could not be located by the selenium code.

The path to the driver executable must be set by the

webdriver.ie.driver system property; formore information, see

<http://code.google.com/p/selenium/wiki/InternetExplorerDriver>

The latest version can be downloaded from

<http://code.google.com/p/selenium/downloads/list>

Dec 11, 2012 12:59:43PM

**org.openqa.selenium.ie.InternetExplorerDriverServer
initializeLib**

WARNING: This method of starting the IE driver is deprecated and will be removed in selenium 2.26. Please download the IEDriverServer.exe from <http://code.google.com/p/selenium/downloads/list> and ensure that it is in your PATH.

Solution:

The solution for the above problem is given in the warning section of the error itself.

- Download the Internet ExplorerDriver standalone server for 32bit or 64bit.
- Save the driver in a suitable location in the system.
- Set the path for the driver using the **System.setProperty** method.
- It is used to set the IE driver with the webdriver property. It helps to locate the driver executable file that is stored in the system location. (Ex:"C:\IEDriverLocation\IEDriver.exe")

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
```

```
public class ITestforDesiredCapabilities {  
  
    public static void main(String[] args) {  
  
        //it is used to define IE capability  
        DesiredCapabilities capabilities =  
        DesiredCapabilities.internetExplorer();  
  
        capabilities.setCapability(CapabilityType.BROWSER_NAME, "IE");  
        capabilities.setCapability(InternetExplorerDriver.  
            INTRODUCE_FLAKINESS_BY_IGNORING_SECURITY_DOMAINS, true);  
  
        System.setProperty("webdriver.ie.driver",  
            "C:\\IEDriverServer.exe");  
  
        //it is used to initialize the IE driver  
        WebDriver driver = new InternetExplorerDriver(capabilities);  
  
        driver.manage().window().maximize();  
  
        driver.get("http://gmail.com");  
  
        driver.quit();  
    }  
}
```

Code Explanation:

In the code above,

- The import statements is to import the required packages for the selenium web driver, required packages for the Internet Explorer driver, packages for the desired capabilities.
- setCapability takes the various capabilities as input variables which are then used by the web driver to launch the application in the desired environment.

- `setProperty` is used to set the path where the driver is located. Web Driver then locates the required driver.
- Gmail website is opened in the Internet Explorer browser by using "get" method.

Output:

The test case on Internet explorer browser will run successfully using Selenium Webdriver.

Conclusion

The Desired Capabilities class will help to set an environment to define the behaviour of the browser/environment on which the test can be executed.

It helps to launch our application in the desired environment having the capabilities that we desire to use.

Chapter 40: SSL Certificate Error Handling in Selenium

SSL (Secure Socket Layer) Certificate ensures secure transformation of data across the server and client application using strong encryption standard or digital signature. One has to install an SSL certificate or a code signing certificate.

What is SSL Certificate

SSL (Secure Sockets Layer) is a standard security protocol for establishing a secure connection between the server and the client which is a browser.

There are number of benefits of using SSL certificate like,

- One can increase their users' and customer's trust in order to enhance the business' growth rapidly
- These certificates help to secure online transactions and customers sensitive information like credit-card/debit-card data, etc.
- Signing certificate tends to get a maximum number of downloads and good reviews from users.

SSL-secured websites begin with **https://** and you can see a lock icon or green address bar if the connection is securely established.

For example, if you want to do some transaction via net banking or want to purchase a Mobile phone through e-commerce site such as

Flipkart or Amazon.

What happens between the Web Browser and Server

1. A browser tries to connect with a website secured with SSL. The browser requests the webserver to identify itself
2. The server sends the browser a copy of its SSL certificate
3. The browser verifies whether the SSL certificate is genuine. If so, it sends a message to the server
4. The server sends back a digitally signed acknowledgment to start an SSL encrypted session
5. The encrypted data is shared between the server and the browser

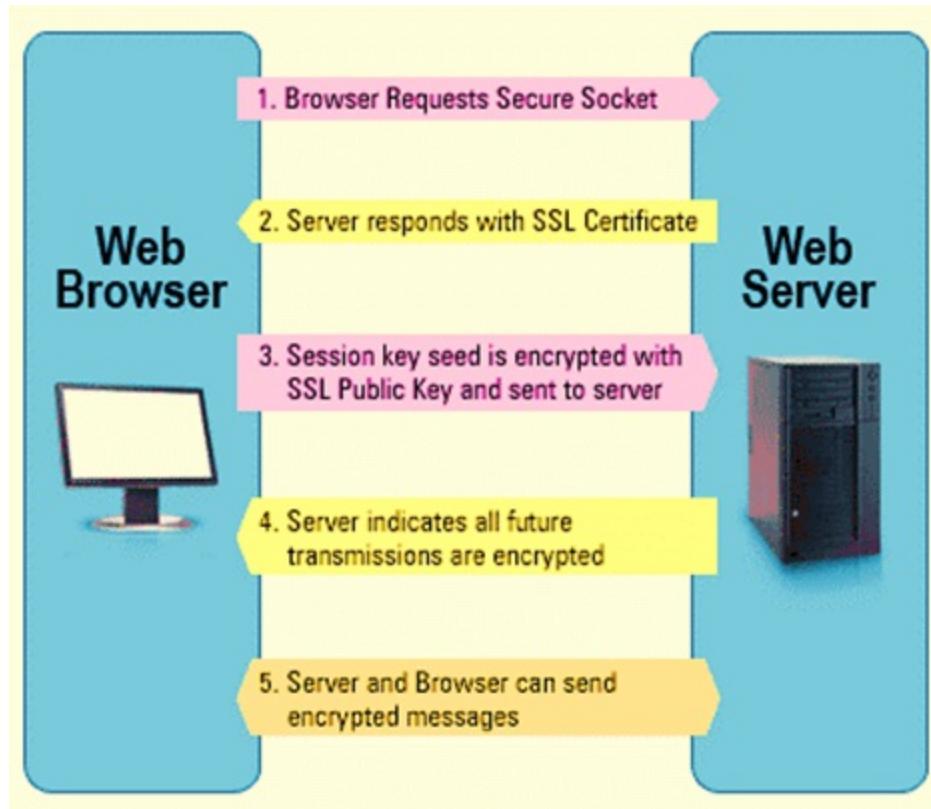
In doing so, you need to transmit sensitive information such as credit card numbers or login credentials and that has to transmit securely so that it cannot be hacked or intercept.

For example

1. Type **https://netbanking.hdfcbank.com/netbanking/** .
2. Hit Enter.
3. You will see a green address bar in the browser as below :-



How Does the SSL Certificate Create a Secure Connection



1. **Browser** sends HTTPS request to the server.
2. Now Server must provide some identification to Browser to prove that it is trusted. This can be done by sending a copy of its SSL certificate to the browser.
3. **Each Browser has its own list of Trusted CA's.** Browser checks the certificate root against its list of trusted CAs and that the certificate is unexpired, unrevoked, and that the common name is valid for the website that it is connecting to.
4. If the browser trusts the certificate, an encrypted session is created between the server and the browser.
5. Server and Browser can send encrypted messages

Types of SSL Certificates

Browser and the server use SSL Certificate mechanism to be able to

establish a secure connection. This connection involves verification of three types of certificates.

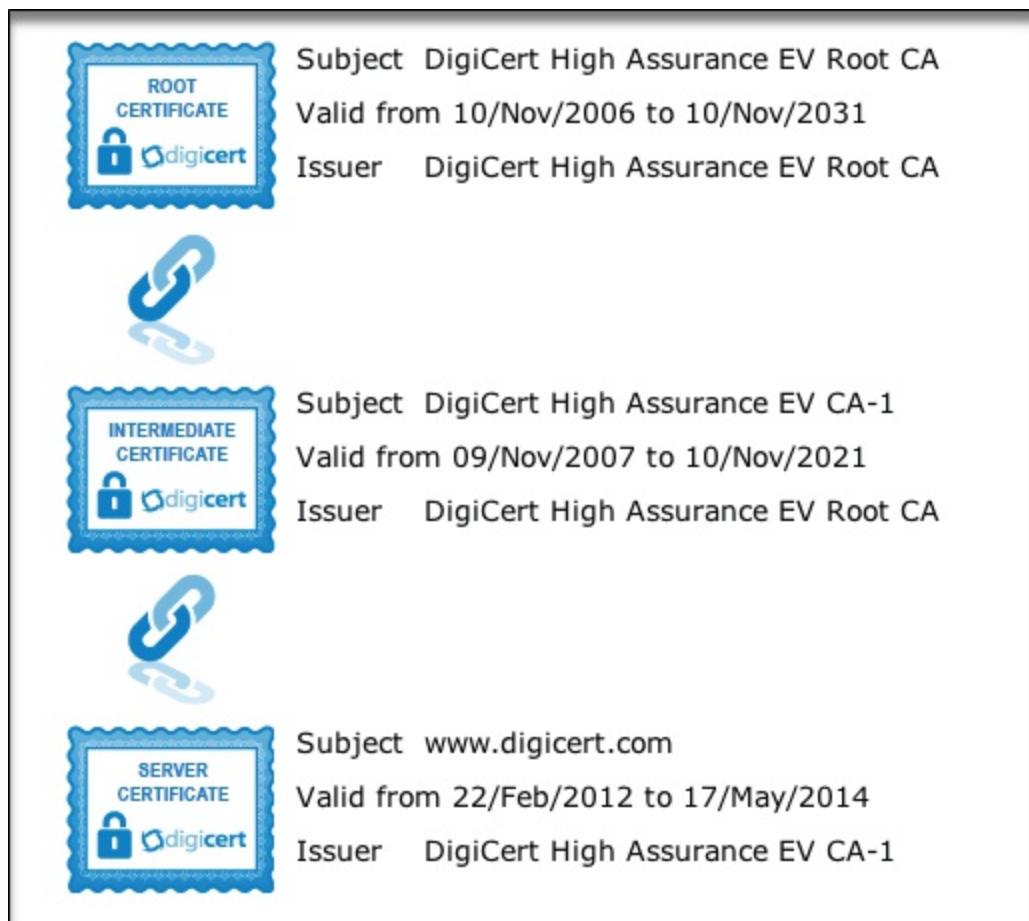
- Root
- Intermediate
- Server Certificate

Process of getting SSL Certificate

The process of getting SSL certificate includes below steps:-

1. First, you must create CSR (create a Certificate Signing Request) request.
2. CSR request creates CSR data file, which is sent to SSL certificate issuer known as CA (Certificate Authority).
3. The CA uses the CSR data files to create SSL certificate for your server.
4. After receiving the SSL certificate, you have to install it on your server.
5. An intermediate certificate is also needed to be installed which ties yours SSL certificate with CA's root certificate.

The below image represent all the three certificate- **Root, Intermediate, and Server Certificate.**



How SSL certificates are verified

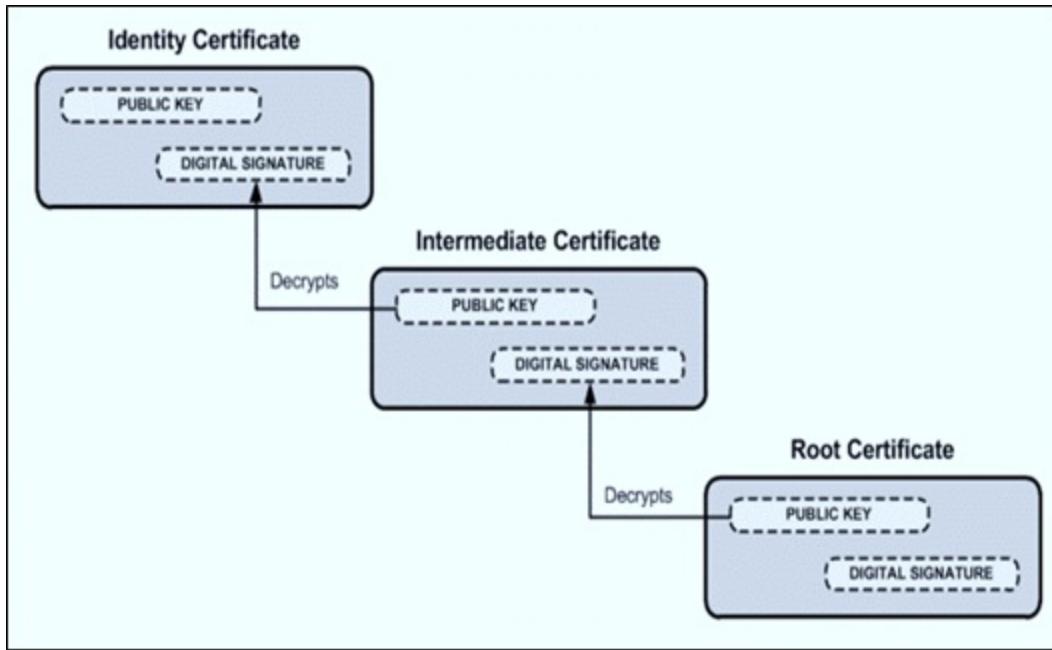
SSL works through a combination of programs and encryption/decryption routine that exist on the web server computer and web server browser.

SSL certificate basically contains below information.

1. Subject which is the identity of the website owner.
2. Validity information- a public and a private key.

The Private and public key are two uniquely related cryptographic keys (numbers). Whatever is encrypted by a public key may only be

decrypted by a private key.



When a secure connection is not established between the server and client due to the certificate, following SSL certificate error will be manifested.

Types of SSL Certificate Error

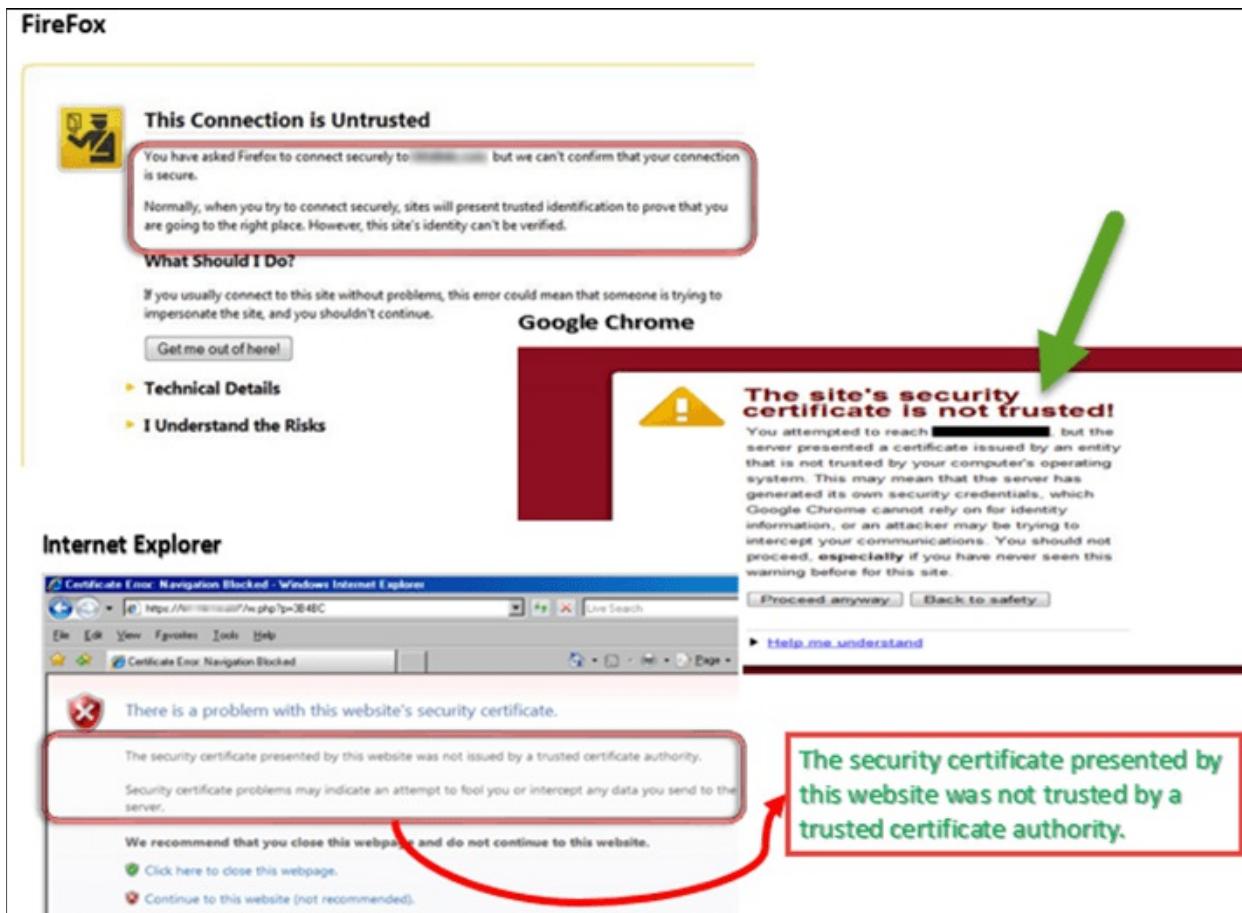
Suppose you type some https request in the browser and get a message such as "This connection is Untrusted" or the "The site's security certificate is not trusted" depending upon the browser you are using. Then such error is subject to SSL certificate error.

Now, if the browser is unable to establish a secured connection with the requested certificate, then the browser will throw "Untrusted Connection" exception as below and ask the user to take appropriate action.

The types of error you likely to see due to certificate in different

browsers may be somewhat like this

1. **FireFox** - This connection is untrusted
1. **Google Chrome** -This site security is not trusted
3. **Internet Explorer (IE)** - This security certificate presented by this website was not trusted by a trusted certificate authority (CA)



How to handle SSL Certificate Error using Selenium Webdriver

Suppose we have written some test scripts and while executing the script, we caught in the situation as "Untrusted Connection" above then how do we handle the exception purely through automation.

In such case, we have to adjust our script in such a way that it will take care of SSL Exception by itself.

The scripts need to be modified according to the type of browser instance we are using. These when desired capabilities comes in picture.

Desired Capabilities is used to configure the driver instance of Selenium Webdriver. Through Desired Capabilities, one can configure all driver instance like ChromeDriver, FirefoxDriver, and Internet Explorer.

As of now we don't have any specific URL to create the above scenario, but I am providing steps that we can add in the Selenium Script to handle the above situation "Untrusted Connection."

SSL Certificate Error Handling in Firefox

For handling SSL certificate error in Firefox, we need to use desired capabilities of Selenium Webdriver and follow the following steps.

Step 1): First we need to create a new firefox profile say "**myProfile**". You can refer google to learn "How to create" firefox profile. It is simple and easy.

Step 2): Now access myProfile in the script as below and create the FirefoxProfile object.

```
ProfilesIni prof = new ProfilesIni()  
FirefoxProfile ffProfile= prof.getProfile ("myProfile")
```

Step 3): Now we need to set "**setAcceptUntrustedCertificates**" and "**setAssumeUntrustedCertificateIssuer**" properties in the Fire Fox profile.

```
ffProfile.setAcceptUntrustedCertificates(true)
ffProfile.setAssumeUntrustedCertificateIssuer(false)
```

Step 4): Now use the FireFox profile in the FireFox driver object.

```
WebDriver driver = new FirefoxDriver (ffProfile)
```

Note: "setAcceptUntrustedCertificates" and "setAssumeUntrustedCertificateIssuer" are capabilities to handle the certificate errors in web browsers.

SSL Certificate Error Handling in Chrome

For handling SSL error in Chrome, we need to use desired capabilities of Selenium Webdriver. The below code will help to accept all the SSL certificate in chrome, and the user will not receive any SSL certificate related error using this code.

We need to create instance of DesiredCapabilities class as below:-

```
DesiredCapabilities handlSSLErr = DesiredCapabilities.chrome ()
handlSSLErr.setCapability (CapabilityType.ACCEPT_SSL_CERTS,
true)
WebDriver driver = new ChromeDriver (handlSSLErr);
```

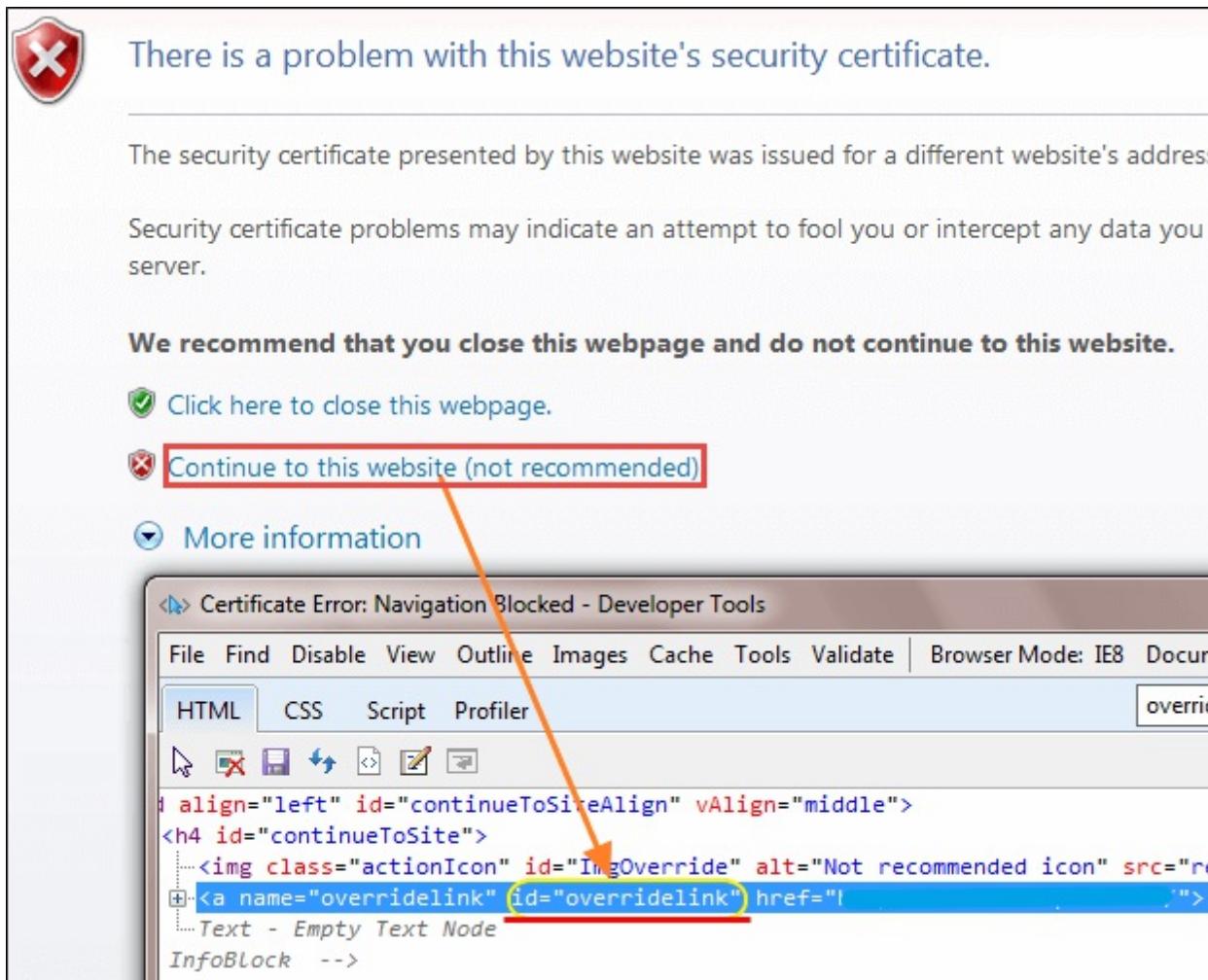
SSL Certificate Error Handling in IE

Unlike handling SSL certificates in Chrome browser and Firefox, in IE, you may have to handle it using javascript.

To handle SSL certificate in IE, you can handle this situation in two ways,

1. In this, you will click the link "**Continue to this website (not recommended)**". In the following we will see how to handle SSL error in IE.

Observe SSL certificate error in IE browser you will find "Continue to this website (not recommended)" link. This link has ID "override link". You can view the ID in HTML mode using F12.



Click on the link using `driver.navigate()` method with JavaScript as below :-

```
driver.navigate().to
("javascript:document.getElementById('overridelink').click()");
```

2. The second method is quite similar to chrome SSL Handling code

```
DesiredCapabilities capabilities = new DesiredCapabilities();
capabilities.setCapability(CapabilityType.ACCEPT_SSL_CERTS,
true);
System.setProperty("webdriver.ie.driver", "IEDriverServer.exe");
WebDriver driver = new InternetExplorerDriver(capabilities);
```

The above code will help to handle SSL certificate error in IE.

Summary:

- SSL (Secure Sockets Layer) is a standard security protocol for establishing secure connection between the server and the client
- Browser and the server use SSL Certificate mechanism to be able to establish a secure connection.
- SSL works through a combination of programs and encryption/decryption routine that exist on the web server computer and web server browser.
- When secure connection is not established between the server and client due to certificate SSL certificate error will occur
- Need to adjust our script in such a way that it will take care of SSL Exception/error by itself through Selenium Web driver.

Chapter 41: Handling Ajax call in Selenium Webdriver

Ajax is a technique used for creating fast and dynamic web pages. This technique is asynchronous and uses a combination of Javascript and XML .

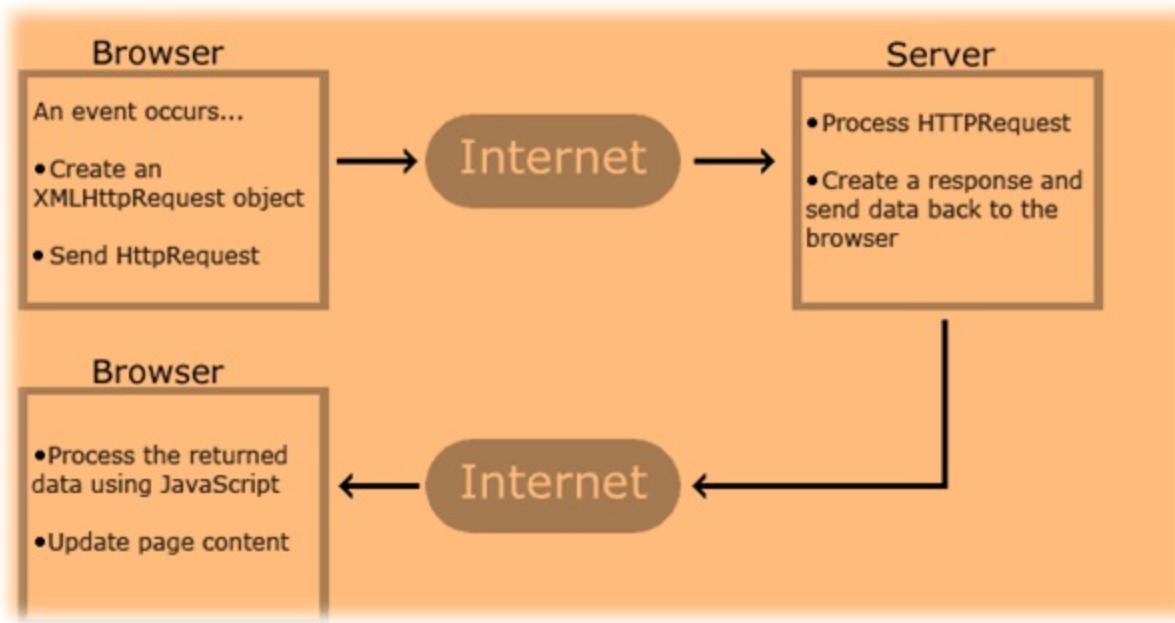
It will updates the part/s of a web page without reloading the whole page.

Some of the famous applications that uses AJAX technique are Gmail, Google Maps, Facebook, Youtube, etc.

What is Ajax

AJAX stands for **Asynchronous JavaScript & XML**, and it allows the Web page to retrieve small amounts of data from the server without reloading the entire page.

For example, when you click on submit button, JavaScript will make a request to the server, interpret the result and update the current screen without reloading the webpage.



- An Ajax call is an asynchronous request initiated by the browser that does not directly result in a page transition. It means, if you fire an Ajax request, the user can still work on the application while the request is waiting for a response.
- AJAX sends HTTP requests from the client to server and then process the server's response, without reloading the entire page. So when you make an AJAX call, you **are not pretty sure about the time taken by the server to send you a response.**

From a tester's point of view, if you are checking the content or the element to be displayed , you need to wait till you get the response. During AJAX call the data is stored in XML format and retrieved from the server.

How to handle Ajax call Using Selenium Webdriver

The **biggest challenge in handling Ajax call is knowing the loading time for the web page.** Since the loading of the web page will last only for a fraction of seconds, it is difficult for the tester to test such application through automation tool. For that, Selenium Webdriver has to use the wait method on this Ajax Call.

So by executing this wait command, selenium will suspend the execution of current Test Case and wait for the expected or new value. When the new value or field appears, the suspended test cases will get executed by Selenium Webdriver.

Following are the wait methods that Selenium Webdriver can use

1. **Thread.Sleep()**

- Thread.Sleep () is not a wise choice as it suspends the current thread for the specified amount of time.
- In AJAX, you can never be sure about the exact wait time. So, your test will fail if the element won't show up within the wait time. Moreover, it increases the overhead because calling Thread.sleep(t) makes the current thread to be moved from the running queue to the waiting queue.
- After the time 't' reached, the current thread will move from the waiting queue to the ready queue, and then it takes some time to be picked by the CPU and be running.

2. **Implicit Wait()**

- This method tells webdriver to wait if the element is not available immediately, but this wait will be in place for the entire time the browser is open. So any search for the elements on the page could take the time the implicit wait is set for.

3. **Explicit Wait()**

- Explicit wait is used to freeze the test execution till the time a

particular condition is met or maximum time lapses.

4. WebdriverWait

- It can be used for any conditions. This can be achieved with WebDriverWait in combination with ExpectedCondition
- The best way to wait for an element dynamically is checking for the condition every second and continuing to the next command in the script as soon as the condition is met.

But the problem with all these waits is, you have to mention the time out unit. What if the element is still not present within the time? So there is one more wait called Fluent wait.

5. Fluent Wait

- This is an implementation of the Wait interface having its timeout and polling interval. Each FluentWait instance determines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition.

Challenges in Handling Ajax Call in Selenium Webdriver

- Using "pause" command for handling Ajax call is not completely reliable. Long pause time makes the test unacceptably slow and increases the Testing time. Instead, "waitforcondition" will be more helpful in testing Ajax applications.
- It is difficult to assess the risk associated with particular Ajax applications
- Given full freedom to developers to modify Ajax application makes the testing process challenging
- Creating automated test request may be difficult for testing tools

as such AJAX application often use different encoding or serialization technique to submit POST data.

An Example for Ajax HANDLING

```
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.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.testng.Assert;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;

public class Ajaxdemo {

    private String URL =
"http://demo.guru99.com/test/ajax.html";

    WebDriver driver;
    WebDriverWait wait;

    @BeforeClass
    public void setUp() {

System.setProperty("webdriver.chrome.driver",".\\chromedriver.exe");
        //create chrome instance
        driver = new ChromeDriver();
        driver.manage().window().maximize();
        driver.navigate().to(URL);
    }

    @Test
    public void test_AjaxExample() {

        By container = By.cssSelector(".container");
        wait = new WebDriverWait(driver, 5);
```

```
wait.until(ExpectedConditions.presenceOfElementLocated(container));
//Get the text before performing an ajax call
WebElement noTextElement =
driver.findElement(By.className("radiobutton"));
String textBefore =
noTextElement.getText().trim();

//Click on the radio button
driver.findElement(By.id("yes")).click();

//Click on Check Button
driver.findElement(By.id("buttoncheck")).click();

/*Get the text after ajax call*/
WebElement TextElement =
driver.findElement(By.className("radiobutton"));

wait.until(ExpectedConditions.visibilityOf(TextElement));
String textAfter = TextElement.getText().trim();

/*Verify both texts before ajax call and after
ajax call text.*/
Assert.assertNotEquals(textBefore, textAfter);
System.out.println("Ajax Call Performed");

String expectedText = "Radio button is checked
and it's value is Yes";

/*Verify expected text with text updated after
ajax call*/
Assert.assertEquals(textAfter, expectedText);
driver.close();
}

}
```

Summary:

- AJAX allows the Web page to retrieve small amounts of data from the server without reloading the entire page.
- To test Ajax application, different wait methods should be applied
 - ThreadSleep
 - Implicit Wait
 - Explicit Wait
 - WebdriverWait
 - Fluent Wait
- Creating automated test request may be difficult for testing tools as such AJAX application often use different encoding or serialization technique to submit POST data.

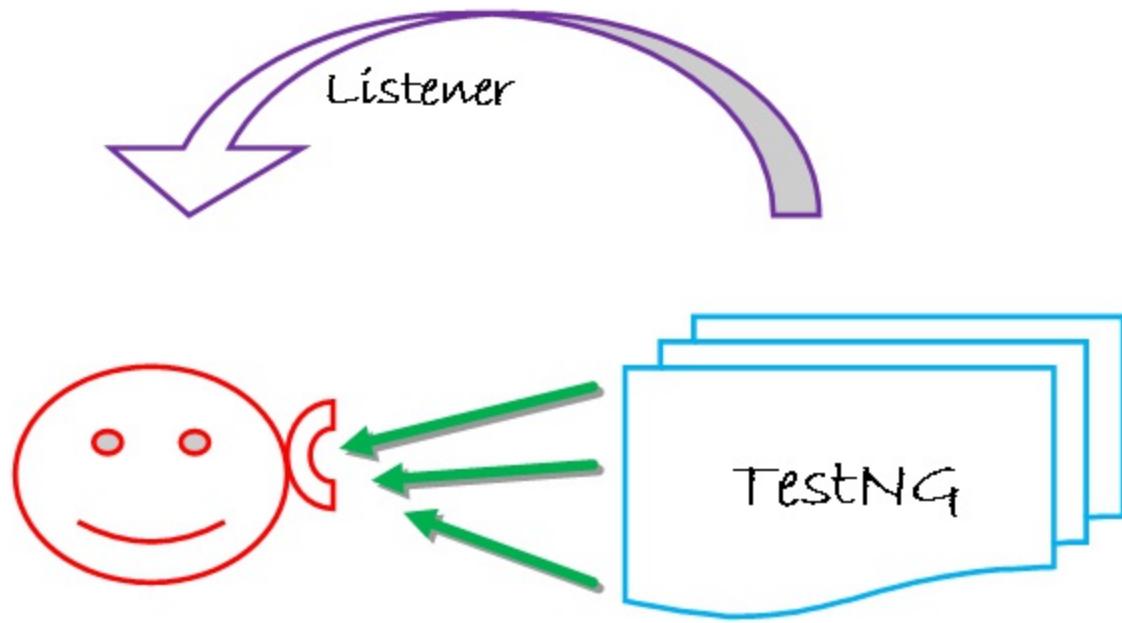
Chapter 42: Listeners and their use in Selenium WebDriver

There are two main listeners.

1. WebDriver Listeners
2. TestNG Listeners

What is Listeners in Selenium WebDriver?

Listener is defined as interface that modifies the default TestNG's behavior. As the name suggests Listeners "listen" to the event defined in the selenium script and behave accordingly. It is used in selenium by implementing Listeners Interface. It allows customizing TestNG reports or logs. There are many types of TestNG listeners available.



Types of Listeners in TestNG

There are many types of listeners which allows you to change the TestNG's behavior.

Below are the few TestNG listeners:

1. `IAnnotationTransformer`,
2. `IAnnotationTransformer2`,
3. `IConfigurable`,
4. `IConfigurationListener`,
5. `IExecutionListener`,
6. `IHookable`,
7. `IInvokedMethodListener`,
8. `IInvokedMethodListener2`,
9. `IMethodInterceptor`,
10. `IReporter`,

11. ISuiteListener,
12. ITestListener .

Above Interface are called TestNG Listeners. These interfaces are used in selenium to generate logs or customize the Testing reports.

In this tutorial, we will implement the ITestListener.

ITestListener has following methods

- **OnStart-** OnStart method is called when any Test starts.
- **onTestSuccess-** onTestSuccess method is called on the success of any Test.
- **onTestFailure-** onTestFailure method is called on the failure of any Test.
- **onTestSkipped-** onTestSkipped method is called on skipped of any Test.
- **onTestFailedButWithinSuccessPercentage-** method is called each time Test fails but is within success percentage.
- **onFinish-** onFinish method is called after all Tests are executed.

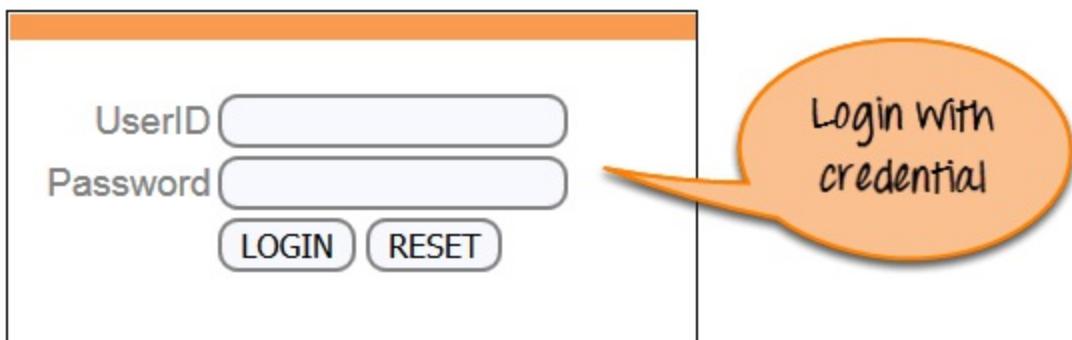
Test Scenario:

In this test scenario, we will automate Login process and implement the 'ItestListener'.

1. Launch the Firefox and open the site
"http://demo.guru99.com/V4/ "



1. Login to the application.



Steps to create a TestNG Listener

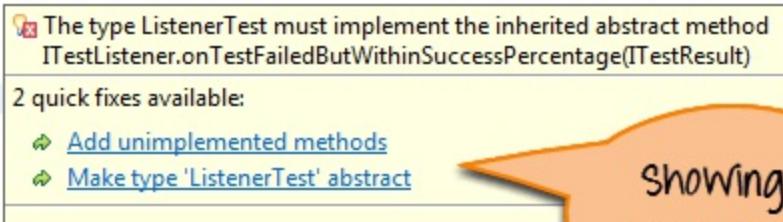
For the above test scenario, we will implement Listener.

Step 1) Create class "Listener_Demo" and implements 'ITestListener'. Move the mouse over redline text, and Eclipse will suggest you 2 quick fixes as shown in below screen:

```

package Listener_Demo;

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

public class ListenerTest implements ITestListener
{
    
}

```

Showng 2
quick fixes

Just click on "Add unimplemented methods". Multiple unimplemented methods (without a body) is added to the code. Check below-

```

package Listener_Demo;

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

public class ListenerTest implements ITestListener
{

    @Override
    public void onFinish(ITestContext arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onStart(ITestContext arg0) {
        // TODO Auto-generated method stub
    }

    @Override

```

```
public void  
onTestFailedButWithinSuccessPercentage(ITestResult arg0) {  
    // TODO Auto-generated method stub  
  
}  
  
@Override  
public void onTestFailure(ITestResult arg0) {  
    // TODO Auto-generated method stub  
  
}  
  
@Override  
public void onTestSkipped(ITestResult arg0) {  
    // TODO Auto-generated method stub  
  
}  
  
@Override  
public void onTestStart(ITestResult arg0) {  
    // TODO Auto-generated method stub  
  
}  
  
@Override  
public void onTestSuccess(ITestResult arg0) {  
    // TODO Auto-generated method stub  
  
}  
}
```

Let's modify the 'ListenerTest' class. In particular, we will modify following methods-

onTestFailure, onTestSkipped, onTestStart, onTestSuccess, etc.

The modification is simple. We just print the name of the Test.

Logs are created in the console. It is easy for the user to understand which test is a pass, fail, and skip status.

After modification, the code looks like-

```
package Listener_Demo;

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

public class ListenerTest implements ITestListener
{

    @Override
    public void onFinish(ITestContext Result)
    {

    }

    @Override
    public void onStart(ITestContext Result)
    {

    }

    @Override
    public void
onTestFailedButWithinSuccessPercentage(ITestResult Result)
    {

    }

    // When Test case get failed, this method is called.
    @Override
    public void onTestFailure(ITestResult Result)
    {
        System.out.println("The name of the testcase failed is
:"+Result.getName());
    }

    // When Test case get Skipped, this method is called.
    @Override
    public void onTestSkipped(ITestResult Result)
    {
```

```
System.out.println("The name of the testcase Skipped is  
:"+Result.getName());  
}  
  
// When Test case get Started, this method is called.  
@Override  
public void onTestStart(ITestResult Result)  
{  
System.out.println(Result.getName()+" test case started");  
}  
  
// When Test case get passed, this method is called.  
@Override  
public void onTestSuccess(ITestResult Result)  
{  
System.out.println("The name of the testcase passed is  
:"+Result.getName());  
}  
}
```

Step 2) Create another class "TestCases" for the login process automation. Selenium will execute this 'TestCases' to login automatically.

```
package Listener_Demo;  
  
import org.openqa.selenium.By;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.firefox.FirefoxDriver;  
import org.testng.Assert;  
import org.testng.annotations.Listeners;  
Import org.testng.annotations.Test;  
  
public class TestCases {  
WebDriver driver= new FirefoxDriver();  
  
// Test to pass as to verify listeners .  
@Test  
public void Login()  
{
```

```

        driver.get("http://demo.guru99.com/V4/");
        driver.findElement(By.name("uid")).sendKeys("mngr34926");
        driver.findElement(By.name("password")).sendKeys("amUpenu");
        driver.findElement(By.name("btnLogin")).click();
    }

// Forcefully failed this test as to verify listener.
@Test
public void TestToFail()
{
    System.out.println("This method to test fail");
    Assert.assertTrue(false);
}
}

```

Step 3) Next, implement this listener in our regular project class i.e. "TestCases". There are two different ways to connect to the class and interface.

The first way is to use Listeners annotation (@Listeners) as shown below:

```
@Listeners(Listener_Demo.ListenerTest.class)
```

We use this in the class "TestCases" as shown below.

So Finally the class " TestCases " looks like after using Listener annotation:

```

package Listener_Demo;

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

@Listeners(Listener_Demo.ListenerTest.class)

```

```

public class TestCases {
    WebDriver driver= new FirefoxDriver();

    //Test to pass as to verify listeners.
    @Test
    public void Login()
    {
        driver.get("http://demo.guru99.com/V4/");
        driver.findElement(By.name("uid")).sendKeys("mngr34926");
        driver.findElement(By.name("password")).sendKeys("amUpenu");
        driver.findElement(By.id("")).click();
    }

    //Forcefully failed this test as verify listener.
    @Test
    public void TestToFail()
    {
        System.out.println("This method to test fail");
        Assert.assertTrue(false);
    }
}

```

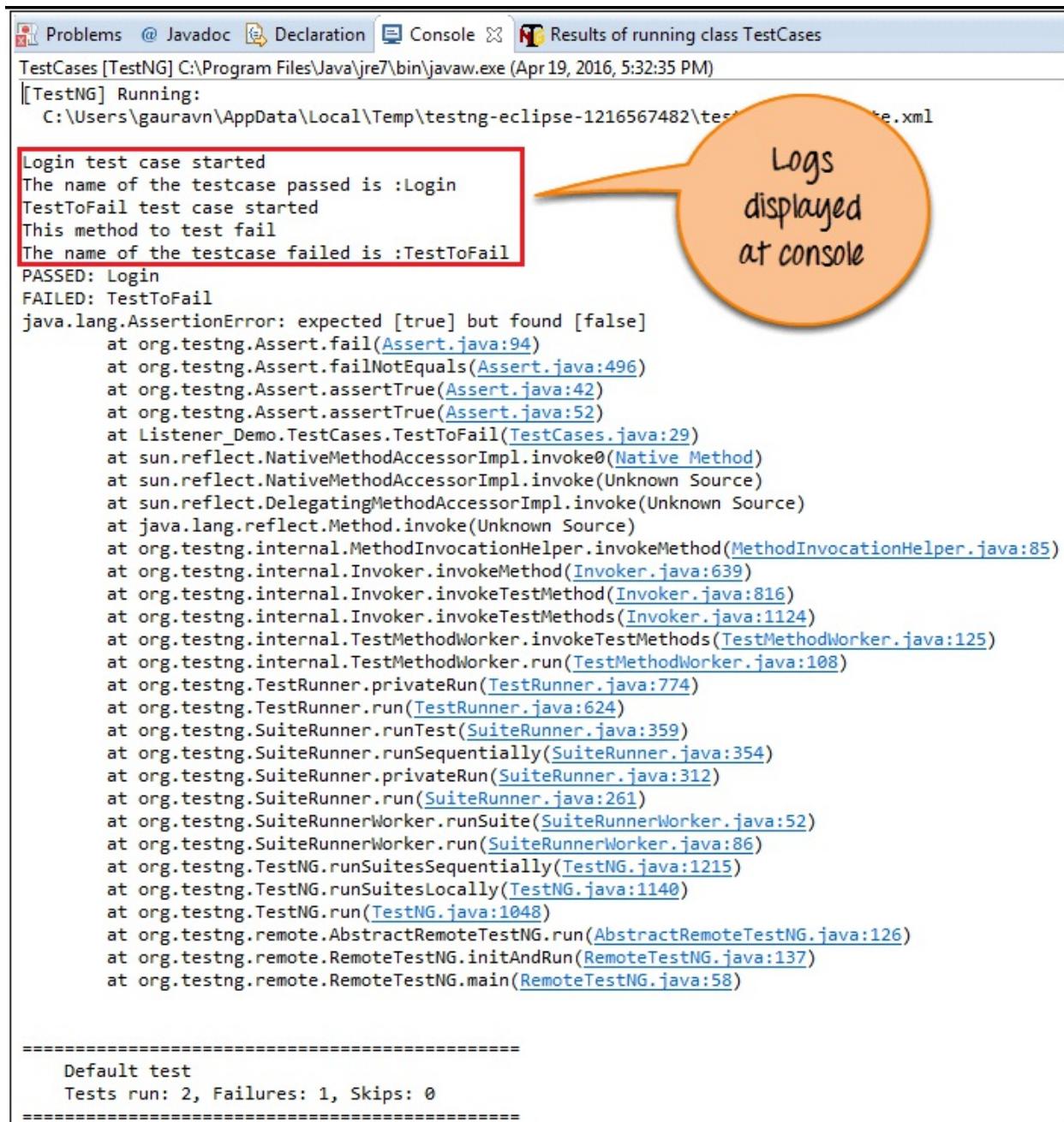
The project structure looks like:



Step 4): Execute the "TestCases" class. Methods in class "ListenerTest" are called automatically according to the behavior of methods annotated as @Test.

Step 5): Verify the Output that logs displays at the console.

Output of the 'TestCases' will look like this:



The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output window displays the results of running the 'TestCases' class using TestNG. A red box highlights the first few lines of the log, which include test case start-ups and failure details. An orange callout bubble points to this red box with the text 'Logs displayed at console'. The full log output is as follows:

```

Problems @ Javadoc Declaration Console Results of running class TestCases
TestCases [TestNG] C:\Program Files\Java\jre7\bin\javaw.exe (Apr 19, 2016, 5:32:35 PM)
[TestNG] Running:
C:\Users\gauravn\AppData\Local\Temp\testng-eclipse-1216567482\testng-customsuite.xml

Login test case started
The name of the testcase passed is :Login
TestToFail test case started
This method to test fail
The name of the testcase failed is :TestToFail
PASSED: Login
FAILED: TestToFail
java.lang.AssertionError: expected [true] but found [false]
    at org.testng.Assert.fail(Assert.java:94)
    at org.testng.Assert.failNotEquals(Assert.java:496)
    at org.testng.Assert.assertTrue(Assert.java:42)
    at org.testng.Assert.assertTrue(Assert.java:52)
    at Listener_Demo.TestCases.TestToFail(TestCases.java:29)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
    at java.lang.reflect.Method.invoke(Unknown Source)
    at org.testng.internal.MethodInvocationHelper.invokeMethod(MethodInvocationHelper.java:85)
    at org.testng.internal.Invoker.invokeMethod(Invoker.java:639)
    at org.testng.internal.Invoker.invokeTestMethod(Invoker.java:816)
    at org.testng.internal.Invoker.invokeTestMethods(Invoker.java:1124)
    at org.testng.internal.TestMethodWorker.invokeTestMethods(TestMethodWorker.java:125)
    at org.testng.internal.TestMethodWorker.run(TestMethodWorker.java:108)
    at org.testng.TestRunner.privateRun(TestRunner.java:774)
    at org.testng.TestRunner.run(TestRunner.java:624)
    at org.testng.SuiteRunner.runTest(SuiteRunner.java:359)
    at org.testng.SuiteRunner.runSequentially(SuiteRunner.java:354)
    at org.testng.SuiteRunner.privateRun(SuiteRunner.java:312)
    at org.testng.SuiteRunner.run(SuiteRunner.java:261)
    at org.testng.SuiteRunnerWorker.runSuite(SuiteRunnerWorker.java:52)
    at org.testng.SuiteRunnerWorker.run(SuiteRunnerWorker.java:86)
    at org.testng.TestNG.runSuitesSequentially(TestNG.java:1215)
    at org.testng.TestNG.runSuitesLocally(TestNG.java:1140)
    at org.testng.TestNG.run(TestNG.java:1048)
    at org.testng.remote.AbstractRemoteTestNG.run(AbstractRemoteTestNG.java:126)
    at org.testng.remote.RemoteTestNG.initAndRun(RemoteTestNG.java:137)
    at org.testng.remote.RemoteTestNG.main(RemoteTestNG.java:58)

=====
Default test
Tests run: 2, Failures: 1, Skips: 0
=====
```

[TestNG] Running:

C:\Users\gauravn\AppData\Local\Temp\testng-eclipse-1058076918\testng-customsuite.xml

Login Test Case started

The name of the testcase passed is:Login

TestToFail test case started

This method to test fail

The name of the testcase failed is:TestToFail

PASSED: Login

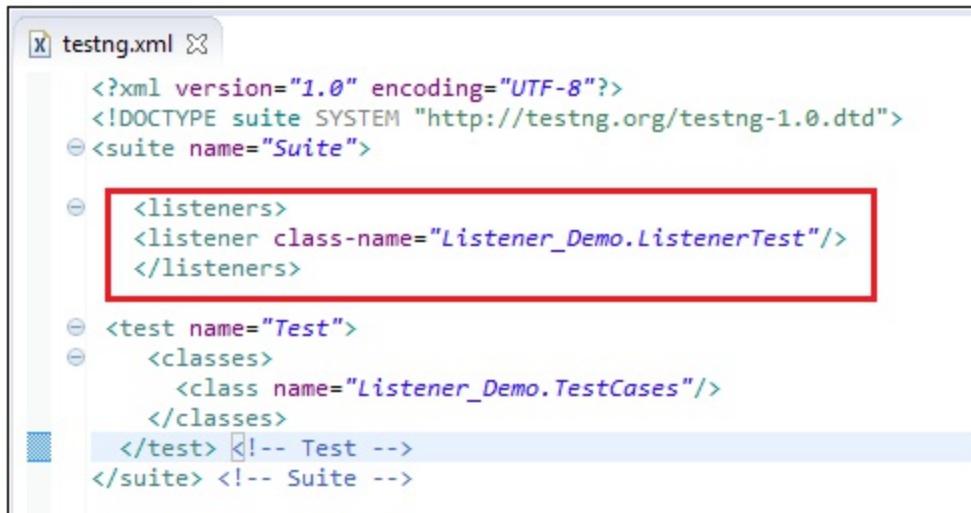
FAILED: TestToFail

java.lang.AssertionError: expected [true] but found [false]

Use of Listener for multiple classes.

If project has multiple classes adding Listeners to each one of them could be cumbersome and error prone.

In such cases, we can create a testng.xml and add listeners tag in XML.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
    <listeners>
        <listener class-name="Listener_Demo.ListenerTest"/>
    </listeners>
    <test name="Test">
        <classes>
            <class name="Listener_Demo.TestCases"/>
        </classes>
    </test> <!-- Test -->
</suite> <!-- Suite -->
```

This listener is implemented throughout the test suite irrespective of

the number of classes you have. When you run this XML file, listeners will work on all classes mentioned. You can also declare any number of listener class.

Summary:

Listeners are required to generate logs or customize TestNG reports in Selenium Webdriver.

- There are many types of listeners and can be used as per requirements.
- Listeners are interfaces used in selenium web driver script
- Demonstrated the use of Listener in Selenium
- Implemented the Listeners for multiple classes

Chapter 43: Execute JavaScript based code using Selenium Webdriver

In Selenium Webdriver, locators like XPath, CSS, etc. are used to identify and perform operations on a web page.

In case, these locators do not work you can use JavaScriptExecutor. You can use JavaScriptExecutor to perform an desired operation on a web element.

Selenium support javaScriptExecutor. There is no need for an extra plugin or add-on. You just need to import **(org.openqa.selenium.JavascriptExecutor)** in the script as to use JavaScriptExecutor .

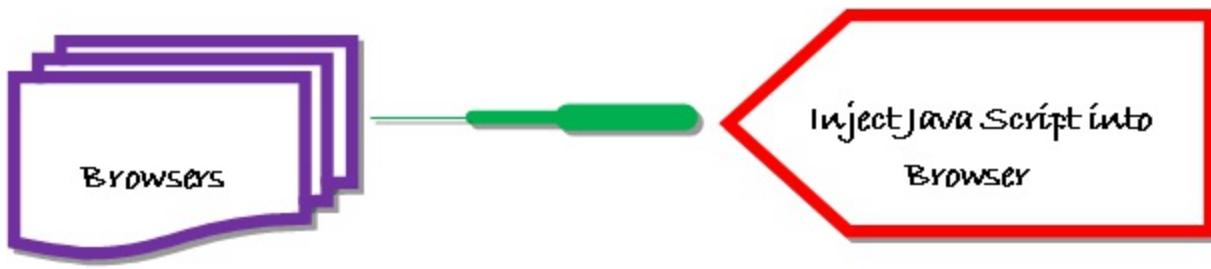
We will discuss JavaScriptExecutor and its execution in Selenium Webdriver in this tutorial.

What is JavaScriptExecutor

JavaScriptExecutor is an Interface that helps to execute JavaScript through Selenium Webdriver.

JavaScriptExecutor provides two methods "executescript" & "executeAsyncScript"

to run javascript on the selected window or current page.



1. **executeAsyncScript**

With Asynchronous script, your page renders more quickly. Instead of forcing users to wait for a script to download before the page renders. This function will execute an asynchronous piece of JavaScript in the context of the currently selected frame or window in Selenium. The JS so executed is single-threaded with a various callback function which runs synchronously.

2. **executeScript**

This method executes JavaScript in the context of the currently selected frame or window in Selenium. The script used in this method runs in the body of an anonymous function (a function without a name). We can also pass complicated arguments to it.

The script can return values. Data types returned are

- Boolean
- Long
- String
- List
- WebElement.

The basic syntax for JavascriptExecutor is given below:

Syntax:

```
JavascriptExecutor js = (JavascriptExecutor) driver;  
js.executeScript(Script,Arguments);
```

- Script – This is the JavaScript that needs to execute.
- Arguments – It is the arguments to the script. It's optional.

Example demonstrating various operations performed by JavaScriptExecutor

Example of executeAsyncScript

Using the executeAsyncScript, helps to improve the performance of your test. It allows writing test more like a normal coding.

The execSync blocks further actions being performed by the Selenium browser but execAsync does not block action. It will send a callback to the server-side Testing suite once the script is done. It means everything inside the script will be executed by the browser and not the server.

Example: Performing a sleep in the browser under test.

In this scenario, we will use "Guru99" demo site to illustrate executeAsyncScript. In this example, you will

- Launch the browser.
- Open site "http://demo.guru99.com/V4/".
- Application waits for 5 sec to perform a further action.

Step 1) Capture the start time before waiting for 5 seconds (5000 milliseconds) by using executeAsyncScript() method.

Step 2) Then, use executeAsyncScript() to wait 5 seconds.

Step 3) Then, get the current time.

Step 4) Subtract (current time – start time) = passed time.

Step 5) Verify the output it should display more than 5000 milliseconds

```
import java.util.concurrent.TimeUnit;

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

public class JavaSE_Test {

    @Test
    public void Login()
    {

        WebDriver driver= new FirefoxDriver();

        //Creating the JavascriptExecutor interface object by
        Type casting
        JavascriptExecutor js = (JavascriptExecutor)driver;

        //Launching the Site.
        driver.get("http://demo.guru99.com/V4/");

        //Maximize window
        driver.manage().window().maximize();

        //Set the Script Timeout to 20 seconds
        driver.manage().timeouts().setScriptTimeout(20,
TimeUnit.SECONDS);

        //Declare and set the start time
        long start_time = System.currentTimeMillis();
```

```
//Call executeAsyncScript() method to wait for 5  
seconds  
  
js.executeAsyncScript("window.setTimeout(arguments[arguments.length - 1], 5000);");  
  
        //Get the difference (currentTime - startTime) of  
times.  
        System.out.println("Passed time: " +  
(System.currentTimeMillis() - start_time));  
  
    }  
}
```

Output: Successfully displayed the passed time more than 5 seconds(5000 miliseconds) as shown below:

[TestNG] Running:

C:\Users\gauravn\AppData\Local\Temp\testng-eclipse-387352559\testng-customsuite.xml

log4j:WARN No appenders could be found for logger
(org.apache.http.client.protocol.RequestAddCookies).

log4j:WARN Please initialize the log4j system properly.

log4j:WARN See
<http://logging.apache.org/log4j/1.2/faq.html#noconfig> for more info.

Passed time: 5022

PASSED: Login

=====

Default test

Tests run: 1, Failures: 0, Skips: 0

=====

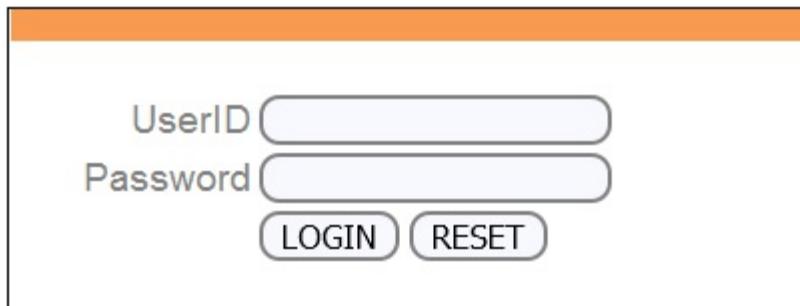
Example of executeScript

For executeScript, we will see three different example one by one.

1) Example: Click a button to login and generate Alert window using JavaScriptExecutor.

In this scenario, we will use "Guru99" demo site to illustrate JavaScriptExecutor. In this example,

- Launch the web browser
- open the site "http://demo.guru99.com/V4/" and
- login with credentials



- Display alert window on successful login.

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

public class JavaSE_Test {
```

```
@Test
public void Login()
{
    WebDriver driver= new FirefoxDriver();

    //Creating the JavascriptExecutor interface object by
Type casting
    JavascriptExecutor js = (JavascriptExecutor)driver;

    //Launching the Site.
    driver.get("http://demo.guru99.com/V4/");

    WebElement button
=driver.findElement(By.name("btnLogin"));

    //Login to Guru99

driver.findElement(By.name("uid")).sendKeys("mngr34926");

driver.findElement(By.name("password")).sendKeys("amUpenu");

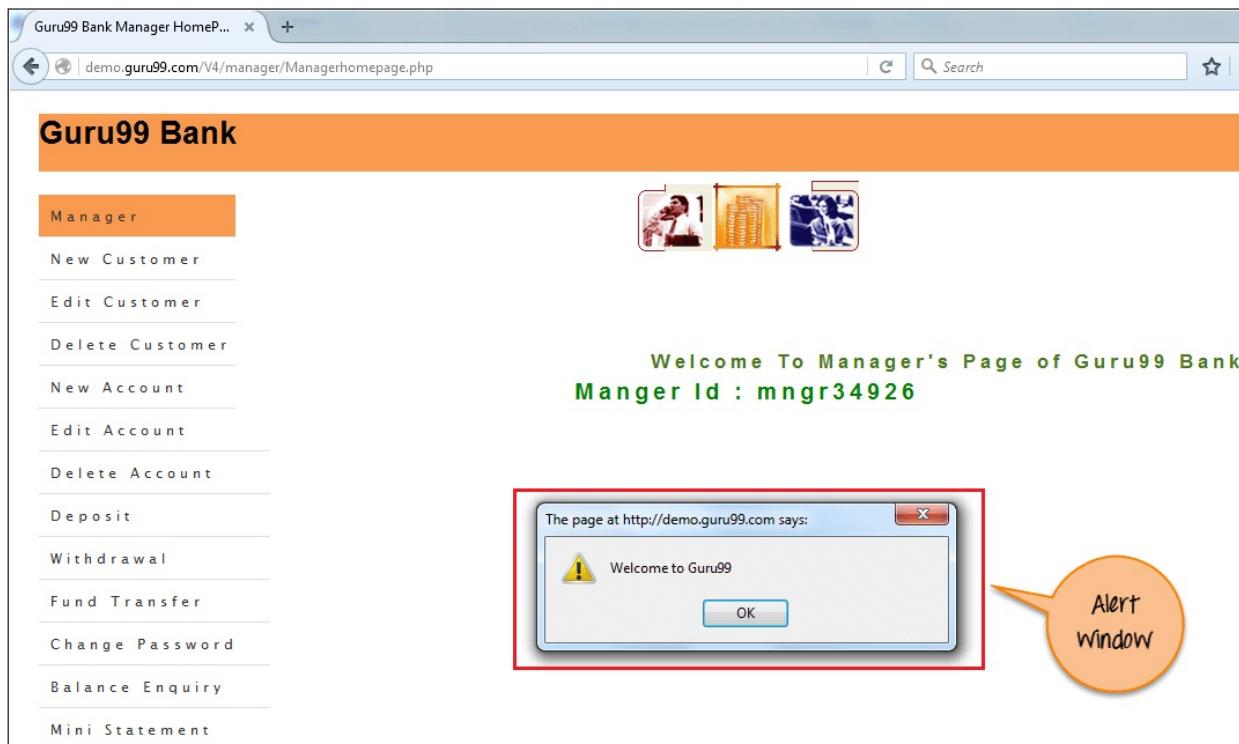
    //Perform Click on LOGIN button using JavascriptExecutor
    js.executeScript("arguments[0].click()", button);

    //To generate Alert window using JavascriptExecutor.
Display the alert message
    js.executeScript("alert('Welcome to Guru99');");
}

}
```

Output: When the code is executed successfully. You will observe

- Successful click on login button and the
- Alert window will be displayed (see image below).



2) Example: Capture Scrape Data and Navigate to different pages using JavaScriptExecutor.

Execute the below selenium script. In this example,

- Launch the site
- Fetch the details of the site like URL of the site, title name and domain name of the site.
- Then navigate to a different page.

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

public class JavaSE_Test {

    @Test
    public void Login()
    {
```

```
WebDriver driver= new FirefoxDriver();

        //Creating the JavascriptExecutor interface object by
Type casting
        JavascriptExecutor js = (JavascriptExecutor)driver;

        //Launching the Site.
        driver.get("http://demo.guru99.com/V4/");

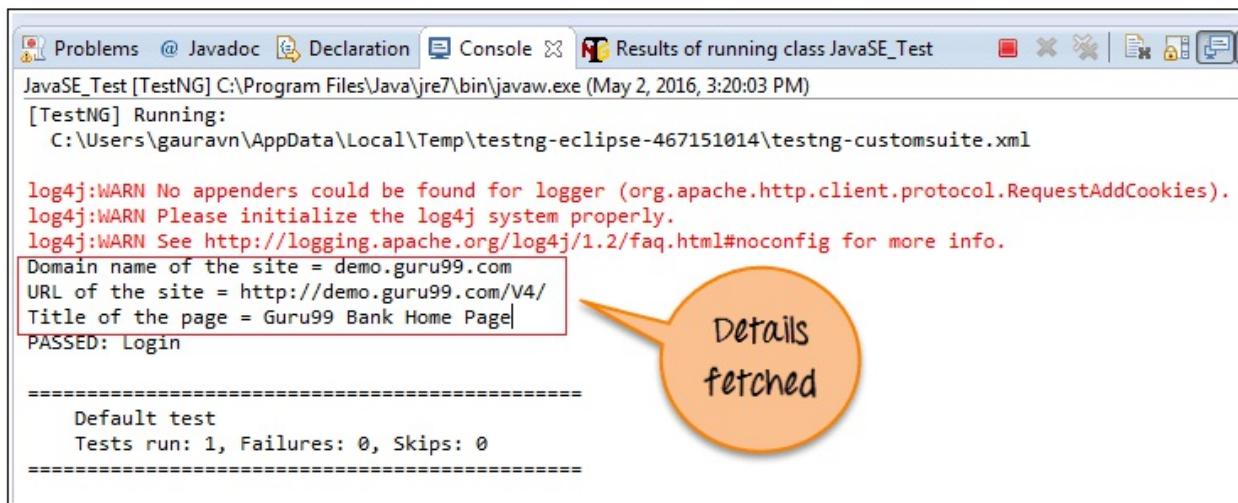
        //Fetching the Domain Name of the site. Tostring()
change object to name.
        String DomainName = js.executeScript("return
document.domain;").toString();
        System.out.println("Domain name of the site =
"+DomainName);

        //Fetching the URL of the site. Tostring() change object
to name
        String url = js.executeScript("return
document.URL;").toString();
        System.out.println("URL of the site = "+url);

        //Method document.title fetch the Title name of the site.
Tostring() change object to name
        String TitleName = js.executeScript("return
document.title;").toString();
        System.out.println("Title of the page = "+TitleName);

        //Navigate to new Page i.e to generate access page.
(launch new url)
        js.executeScript("window.location =
'http://demo.guru99.com/'");
    }
}
```

Output: When above code is executed successfully, it will fetch the details of the site and navigate to different page as shown below.



The screenshot shows the Eclipse IDE interface with the 'Results of running class JavaSE_Test' tab selected. The output window displays the following text:

```
JavaSE_Test [TestNG] C:\Program Files\Java\jre7\bin\javaw.exe (May 2, 2016, 3:20:03 PM)
[TestNG] Running:
C:\Users\gauravn\AppData\Local\Temp\testng-eclipse-467151014\testng-customsuite.xml

log4j:WARN No appenders could be found for logger (org.apache.http.client.protocol.RequestAddCookies).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.

Domain name of the site = demo.guru99.com
URL of the site = http://demo.guru99.com/V4/
Title of the page = Guru99 Bank Home Page
PASSED: Login

=====
Default test
Tests run: 1, Failures: 0, Skips: 0
=====
```

An orange callout bubble with the text "Details fetched" points to the highlighted section of the log output.

[TestNG] Running:

C:\Users\gauravn\AppData\Local\Temp\testng-eclipse-467151014\testng-customsuite.xml

log4j:WARN No appenders could be found for logger
(org.apache.http.client.protocol.RequestAddCookies).

log4j:WARN Please initialize the log4j system properly.

log4j:WARN See
<http://logging.apache.org/log4j/1.2/faq.html#noconfig> for more info.

Domain name of the site = demo.guru99.com

URL of the site = http://demo.guru99.com/V4/

Title of the page = Guru99 Bank Home Page

PASSED: Login

=====

Default test

Tests run: 1, Failures: 0, Skips: 0

=====



3) Example: Scroll Down using JavaScriptExecutor.

Execute the below selenium script. In this example,

- Launch the site
- Scroll down by 600 pixel

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

public class JavaSE_Test {

    @Test
    public void Login()
    {
        WebDriver driver= new FirefoxDriver();

        //Creating the JavascriptExecutor interface object by
        Type casting
        JavascriptExecutor js = (JavascriptExecutor)driver;
```

```
//Launching the Site.  
driver.get("http://moneyboats.com/");  
  
//Maximize window  
driver.manage().window().maximize();  
  
//Vertical scroll down by 600 pixels  
js.executeScript("window.scrollBy(0,600)");  
}  
}
```

Output: When above code is executed, it will scroll down by 600 pixels (see image below).

Peoples' quote on Resume Templates

EASY TO USE

Designer Templates are easy to use and Eye catcher. Thanks Resume for the Awesome Design

AWESOME !

Specially designed Templates Must use for every job seeker

THANKS RESUME

Thanks for these awesome resume templates. I got my dream job, thanks a lot !!

IMPRESSIVE

Make a perfect first Impression with Awes...

Scrolled
down to
600 pixels

Summary:

JavaScriptExecutor is used when Selenium Webdriver fails to click on any element due to some issue.

- JavaScriptExecutor provides two methods "executescript" & "executeAsyncScript" to handle.
- Executed the JavaScript using Selenium Webdriver.
- Illustrated how to click on an element through JavaScriptExecutor, if selenium fails to click on element due to

some issue.

- Generated the 'Alert' window using JavaScriptExecutor.
- Navigated to the different page using JavaScriptExecutor.
- Scrolled down the window using JavaScriptExecutor.
- Fetched URL, title, and domain name using JavaScriptExecutor.

Chapter 44: Using Selenium with Python

Selenium supports Python and thus can be utilized with Selenium for testing.

- Python is easy compared to other programming languages, having far less verbose.
- The Python APIs empower you to connect with the browser through Selenium.
- Selenium sends the standard Python commands to different browsers, despite variation in their browser's design.

You can run Python scripts for Firefox, Chrome, IE, etc.on different Operating Systems.

What is Python?

Python is a high-level object-oriented scripting language. It is designed in a user-friendly manner. Python uses simple English keywords, which is easy to interpret. It has less syntax complications than any other programming languages.

See some of the examples in the table below.

Keyword	Meaning	Usage
elif	Else if	Else if
else	Else	if: X; elif: Y; else: J
except	do this ,If an exception happens,	except ValueError, a: print a
exec	Run string as Python	exec 'print "hello world !"'

What is Selenium?

Selenium is a tool to test your web application. You can do this in various ways, for instance

- Permit it to tap on buttons
- Enter content in structures
- Skim your site to check whether everything is "OK" and so on.

Why to choose Python over Java in Selenium

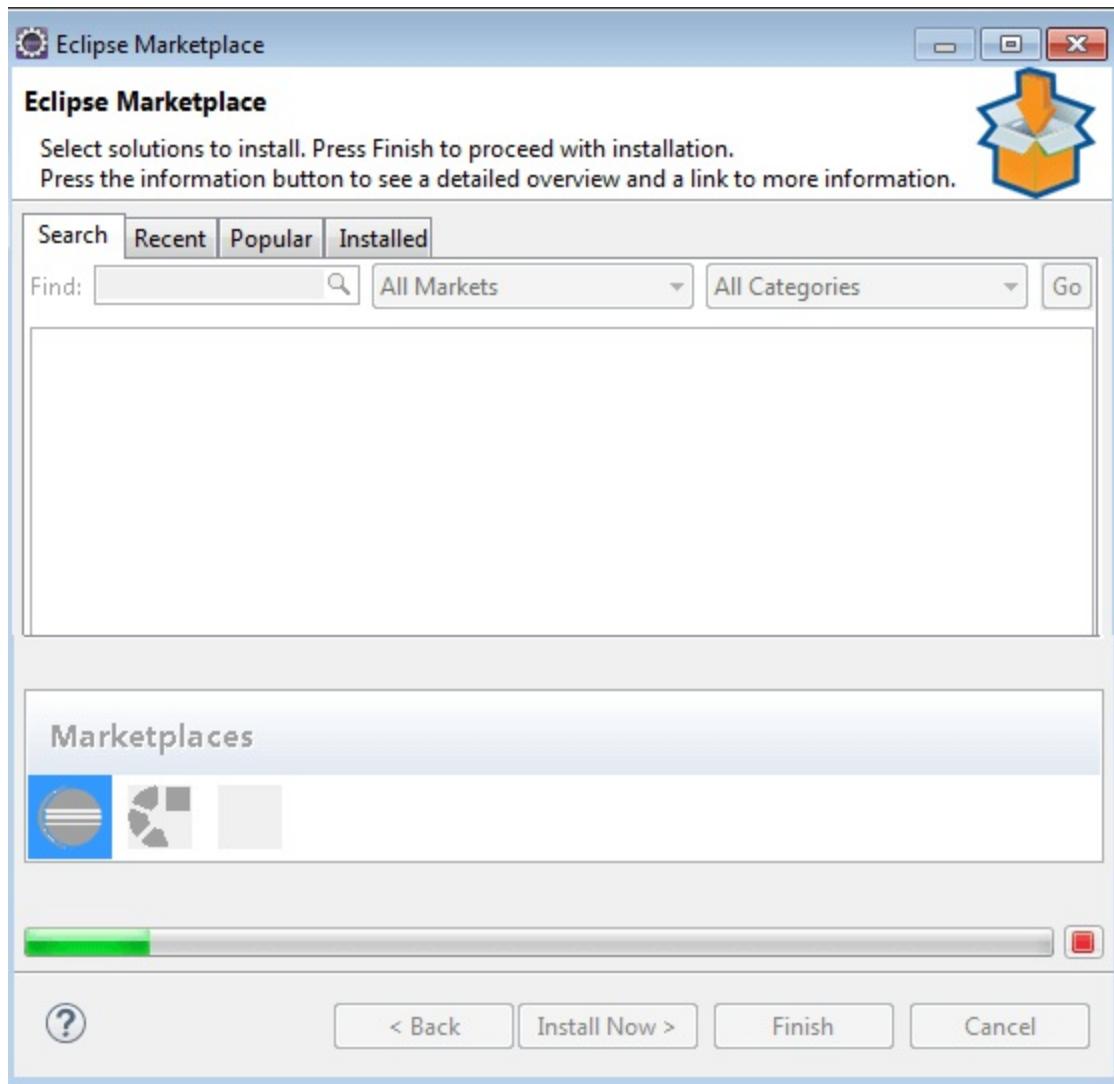
Few points that favor Python over Java to use with Selenium is,

1. Java programs tend to run slower compared to Python programs.
2. Java uses traditional braces to start and ends blocks, while Python uses indentation.
3. Java employs static typing, while Python is dynamically typed.
4. Python is simpler and more compact compared to Java.

Install and Configure PyDev in Eclipse

PyDev is Python development environment for Eclipse.

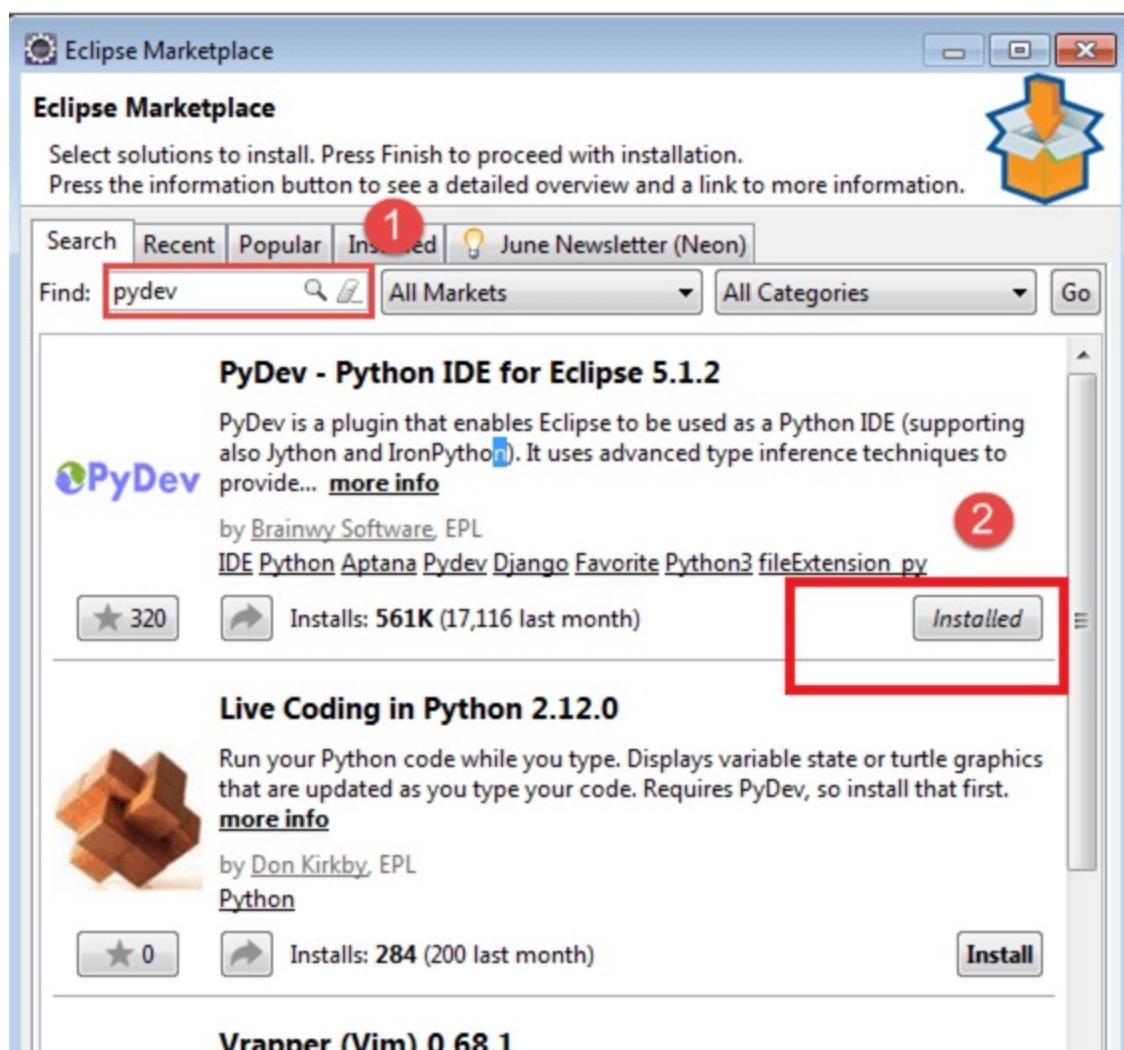
Step 1) Got to Eclipse Marketplace. Help > Eclipse Marketplace



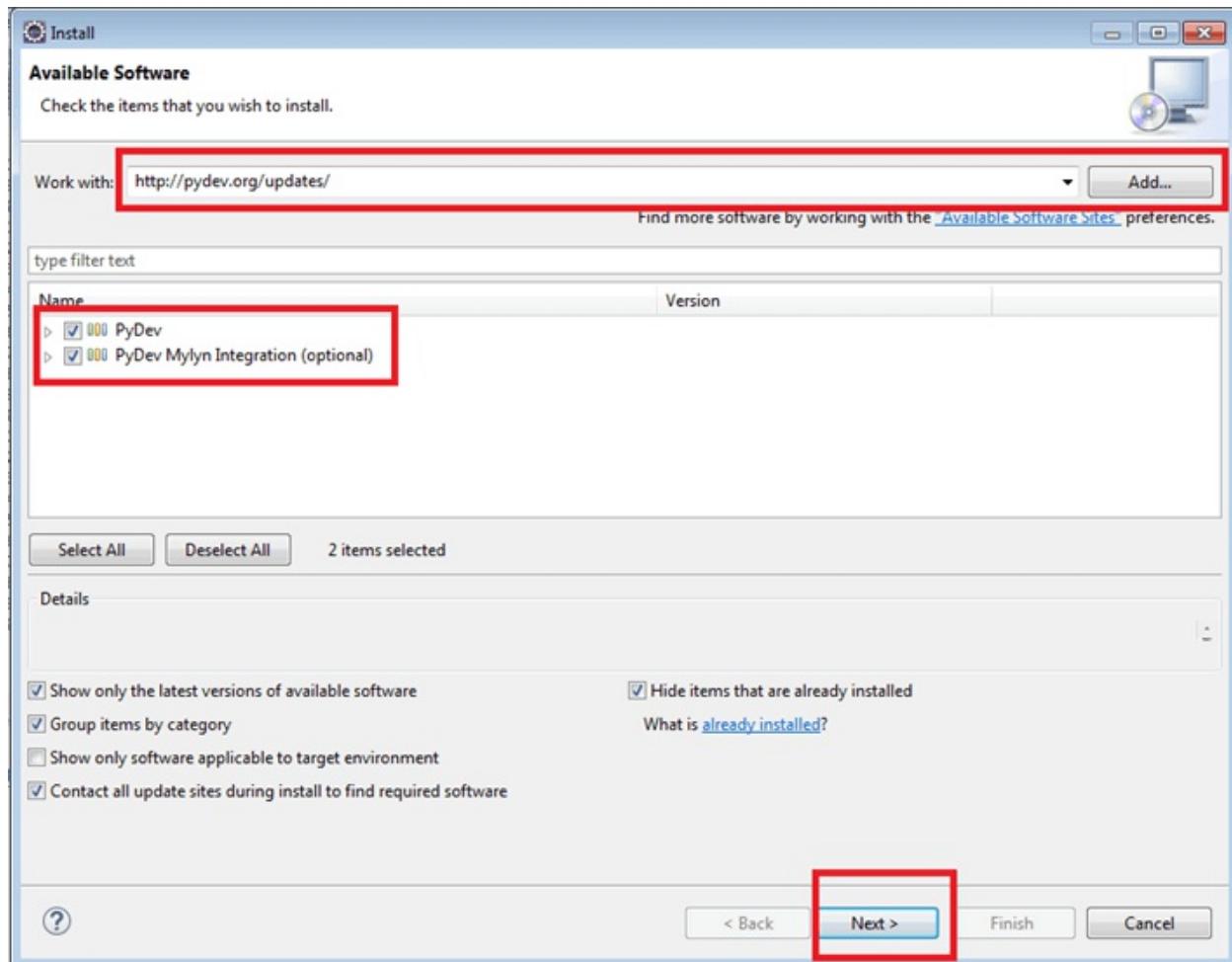
Now once the plugin 'eclipse market place' is opened. The next step is to install "pydev IDE" for eclipse.

Step 2) In this step,

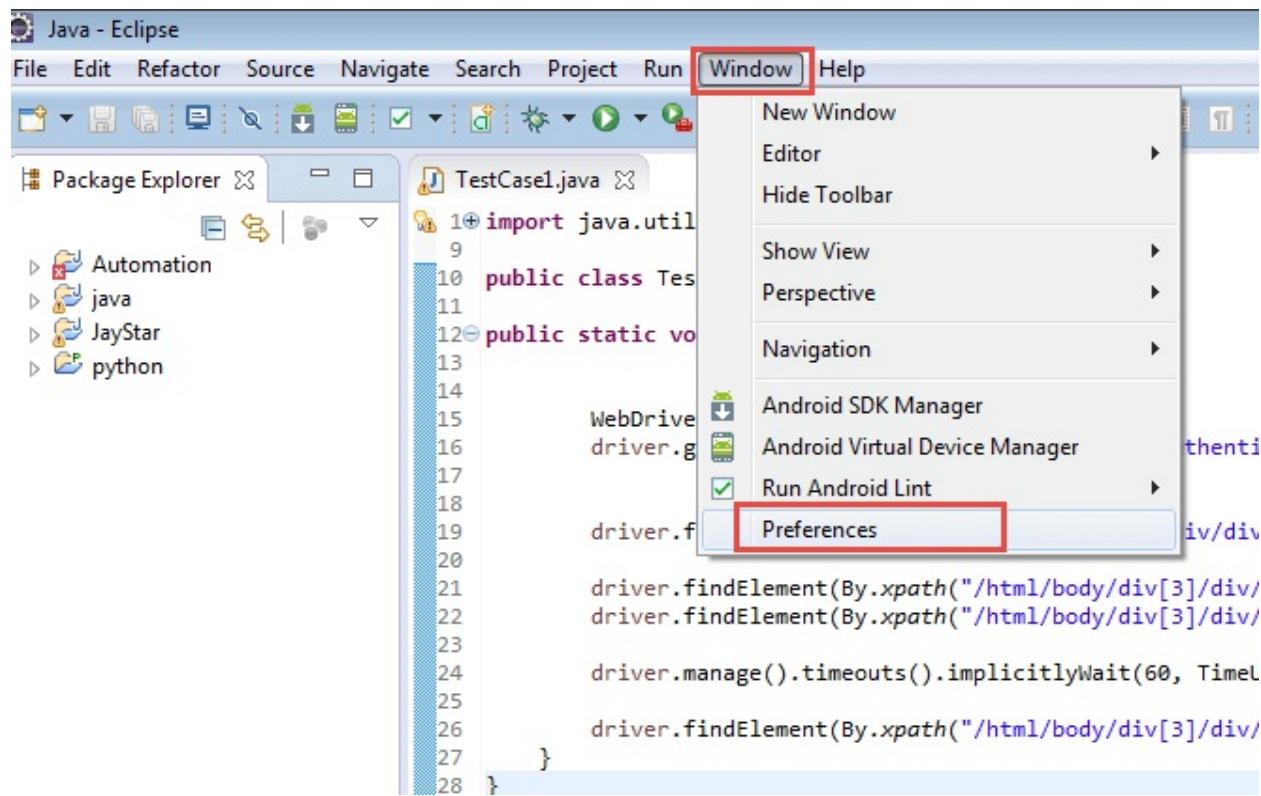
1. Search for "pydev" in search box and then
2. Click install(In my system it is already installed).



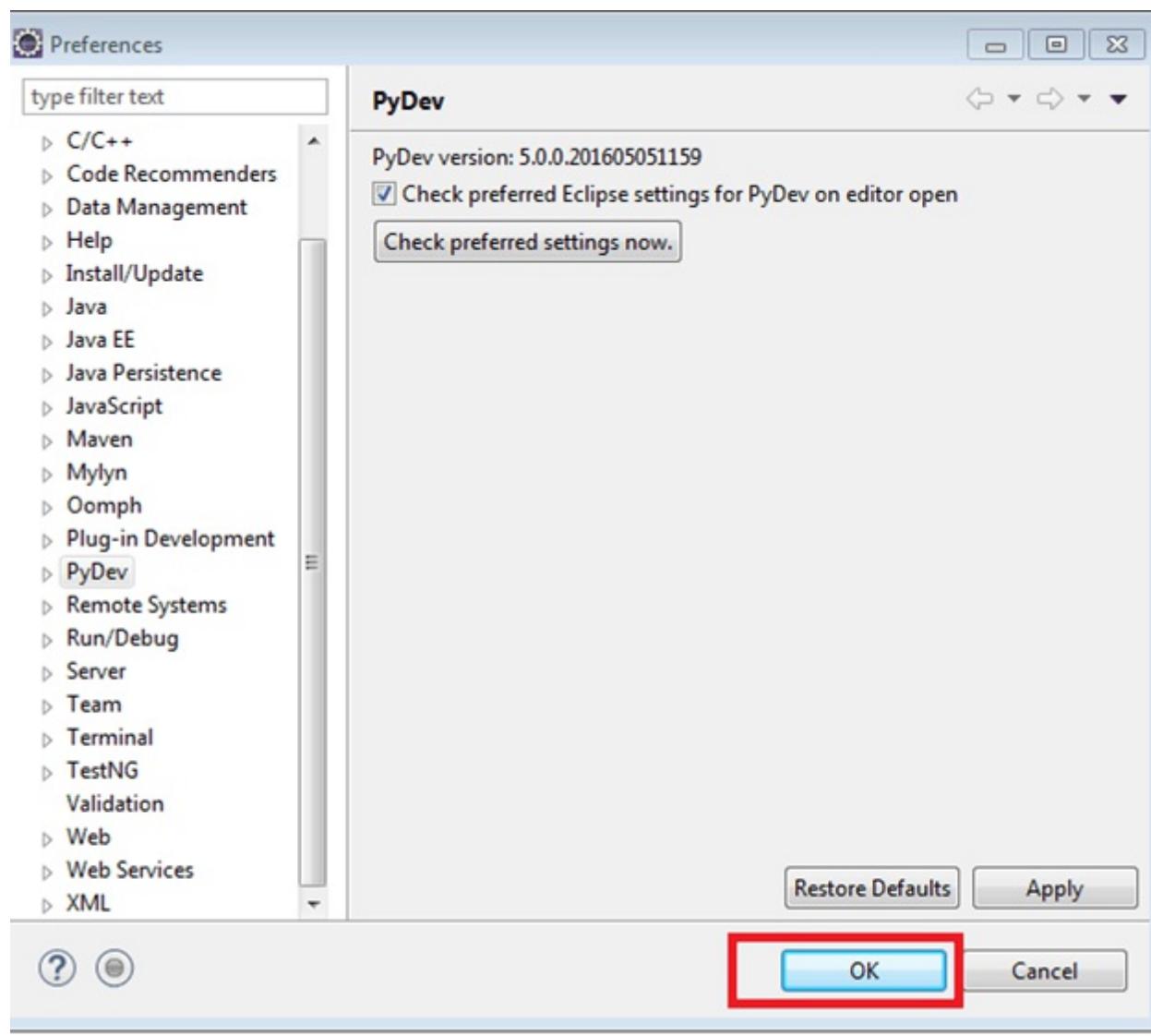
Step 3) Select the checkbox button. It says 'PyDev.' The first check box is mandatory while the second one is optional. After marking the checkbox, press 'Next'.



Step 4) Now, in this step you will set preferences. With the help of preference option, you can use Python as per the project need.

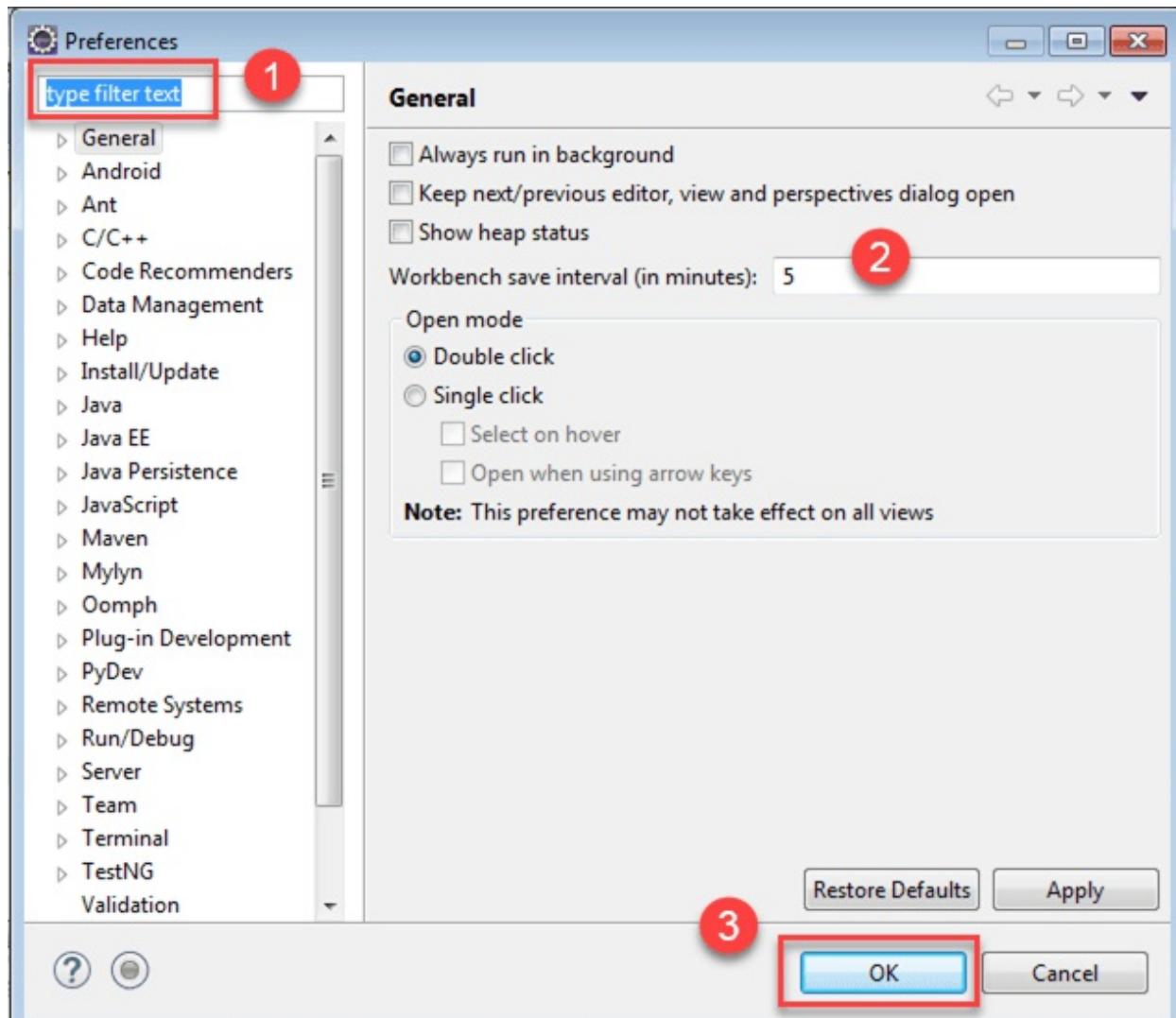


Go to Windows > Preferences > Interpreter-Python. Click on "OK" button.

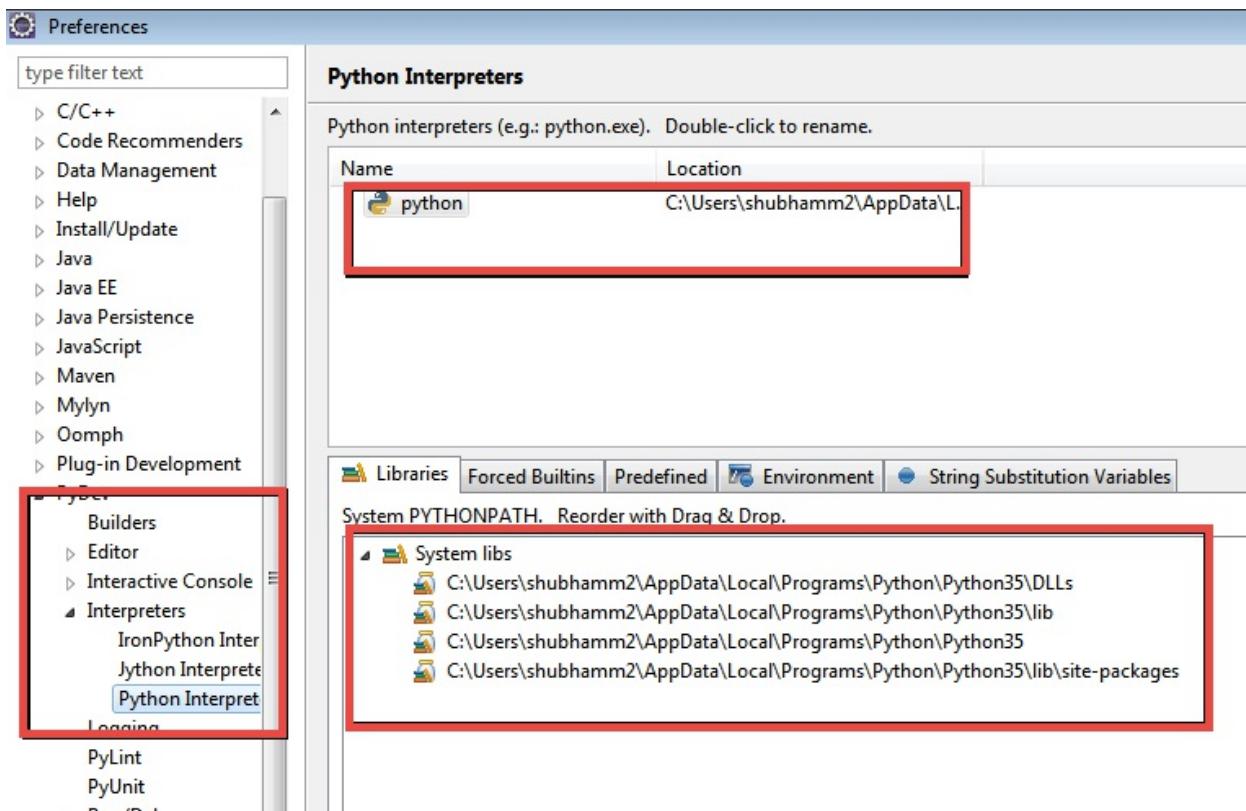


A new window will open when you click on 'OK' button. In this window, follow the following steps.

- Under interpreter dropdown, you select the option Interpreter-Python. It helps in running Python scripts.
- Also, set workbench time interval. When a build is performed, the workbench will automatically save all resources that is changed since the last build.
- Click on 'OK' button.



When you click on "OK" button, it sets the default Python Interpreter. It is just like you need to set java compiler for running a Java code. To change the interpreter name, double click on Python Tab.



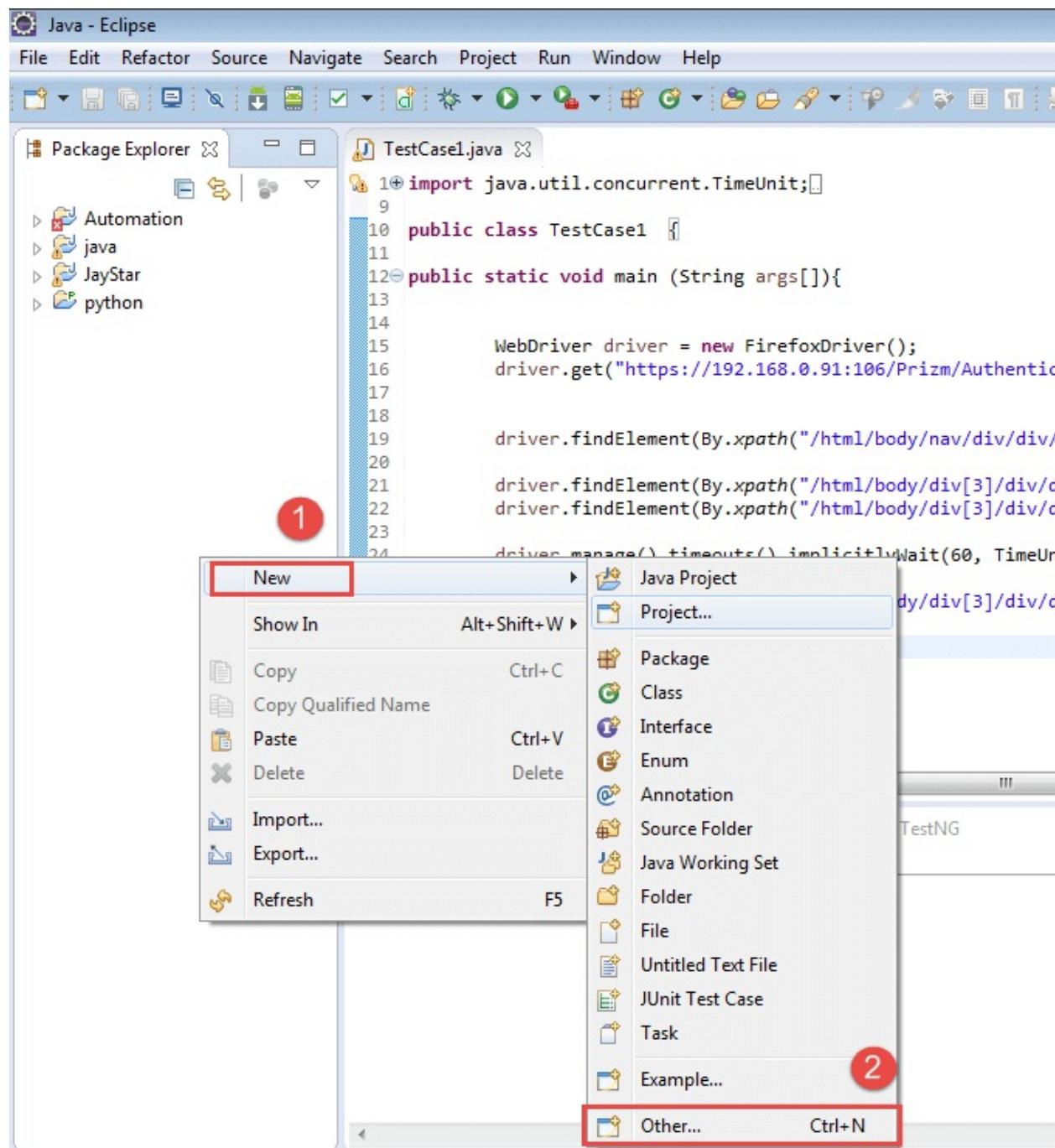
Step 5) In this step, give the "interpreter name" and the "exe file name" of Python.

1. Click on 'Browse' and find python.exe "C:\Python27\python.exe".
2. Click 'OK' button.

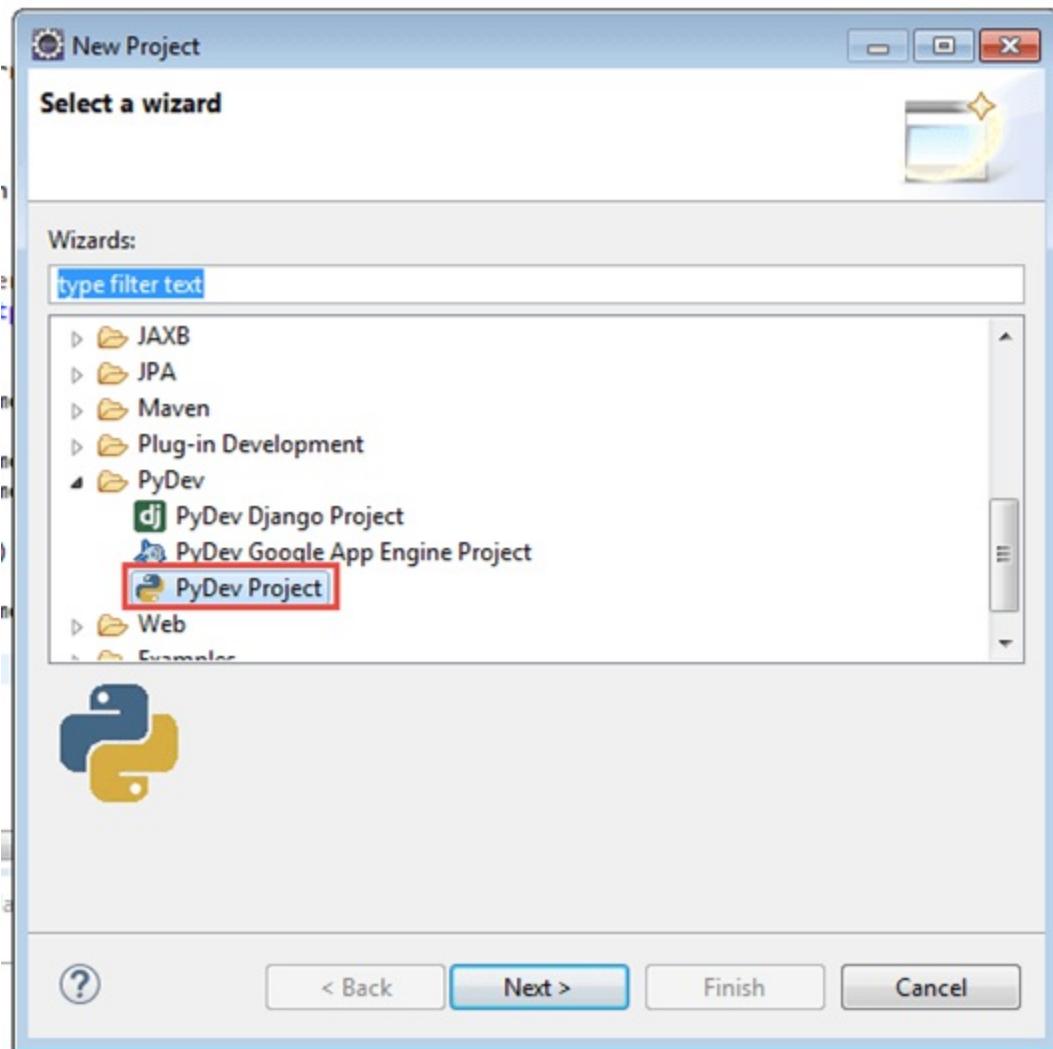


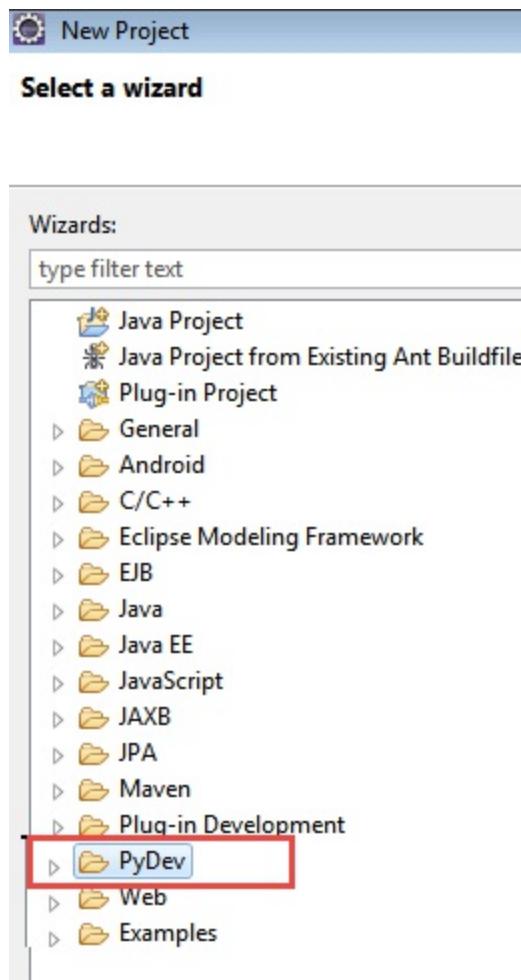
Step 6) Make a New Project in Python. In this step,

1. Right click Package Explorer > New >
2. Select option others.



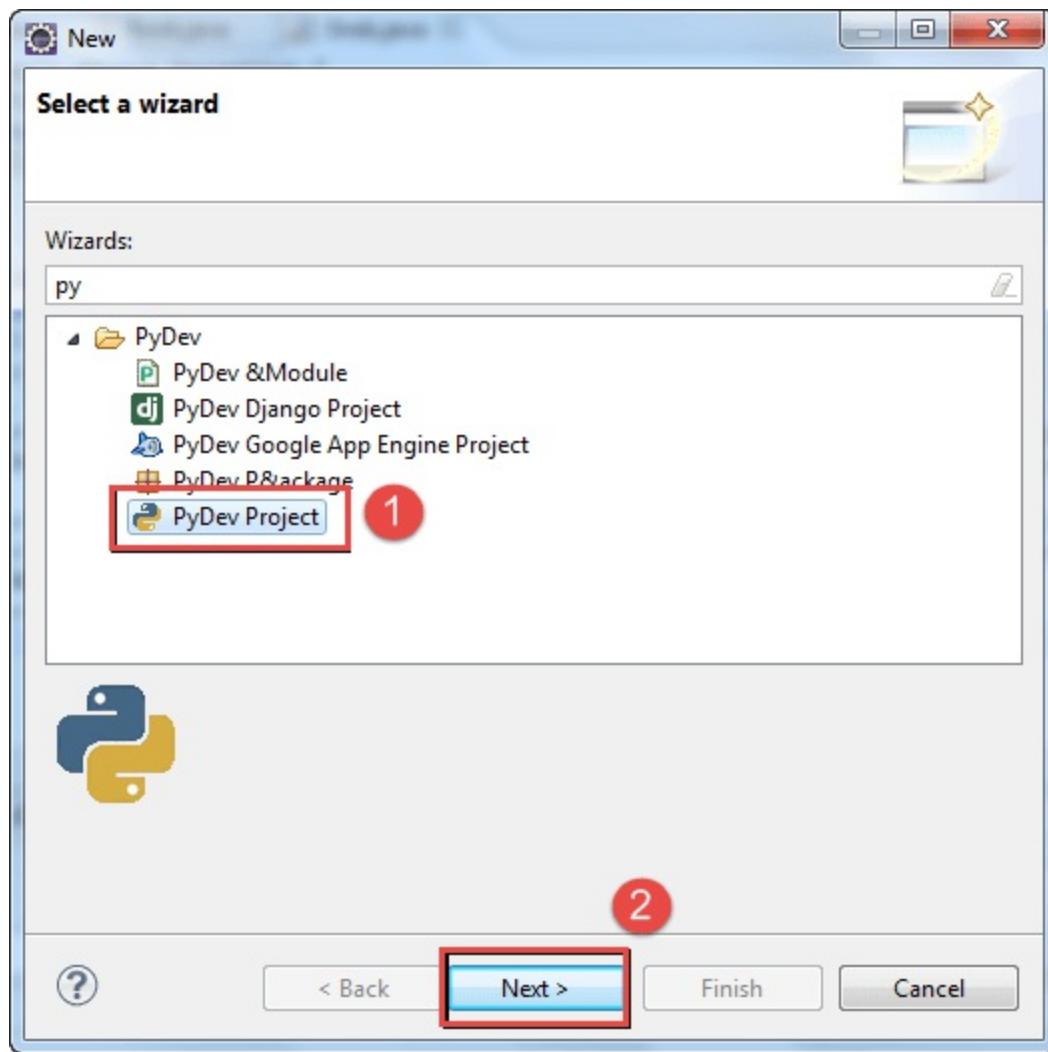
You can see the new Python(PyDev) project is created.





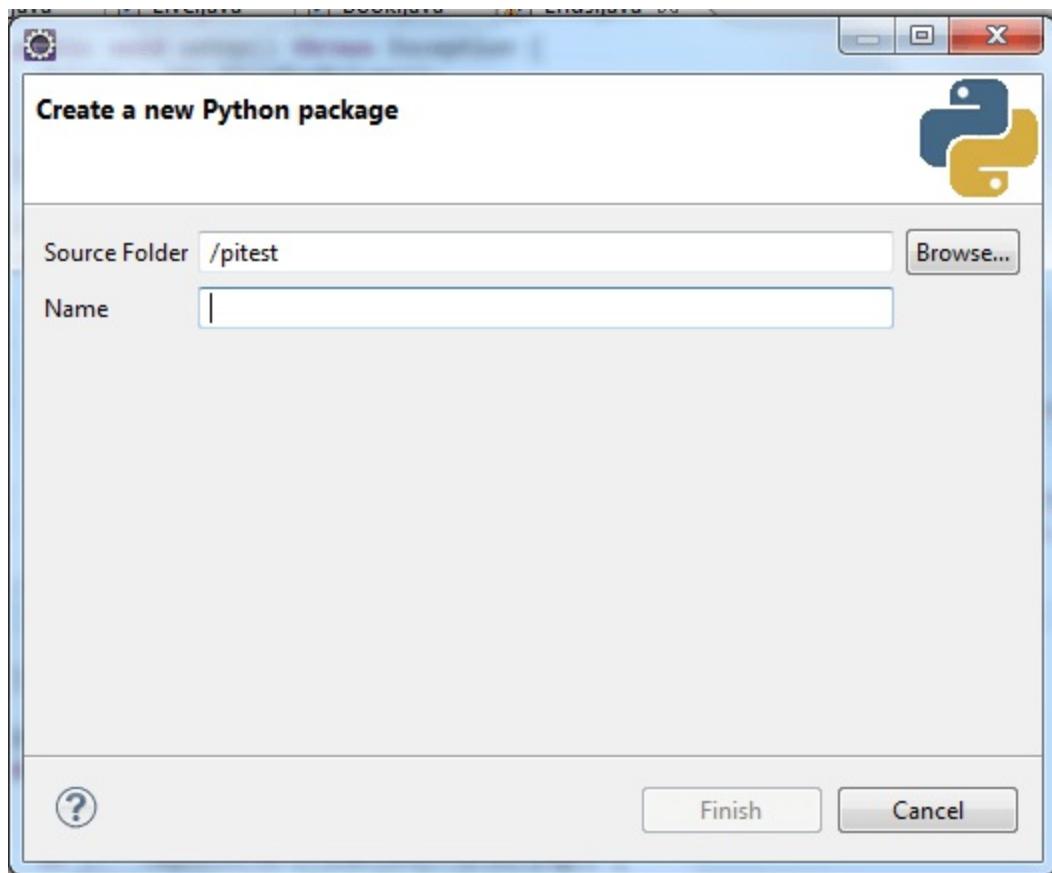
Step 7) In this step,

1. Select 'PyDev Project' and
2. Press 'Next' button.

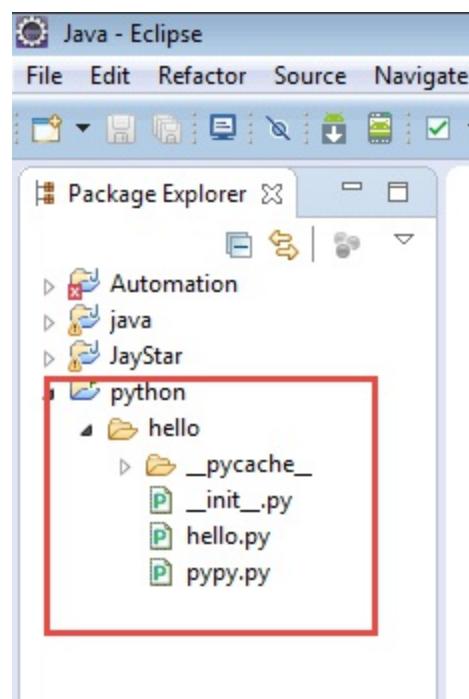


After creating 'PyDev Project', you will create a new Python package.

Step 8) Create a new Python package. After entering the name, click on "Finish" button.

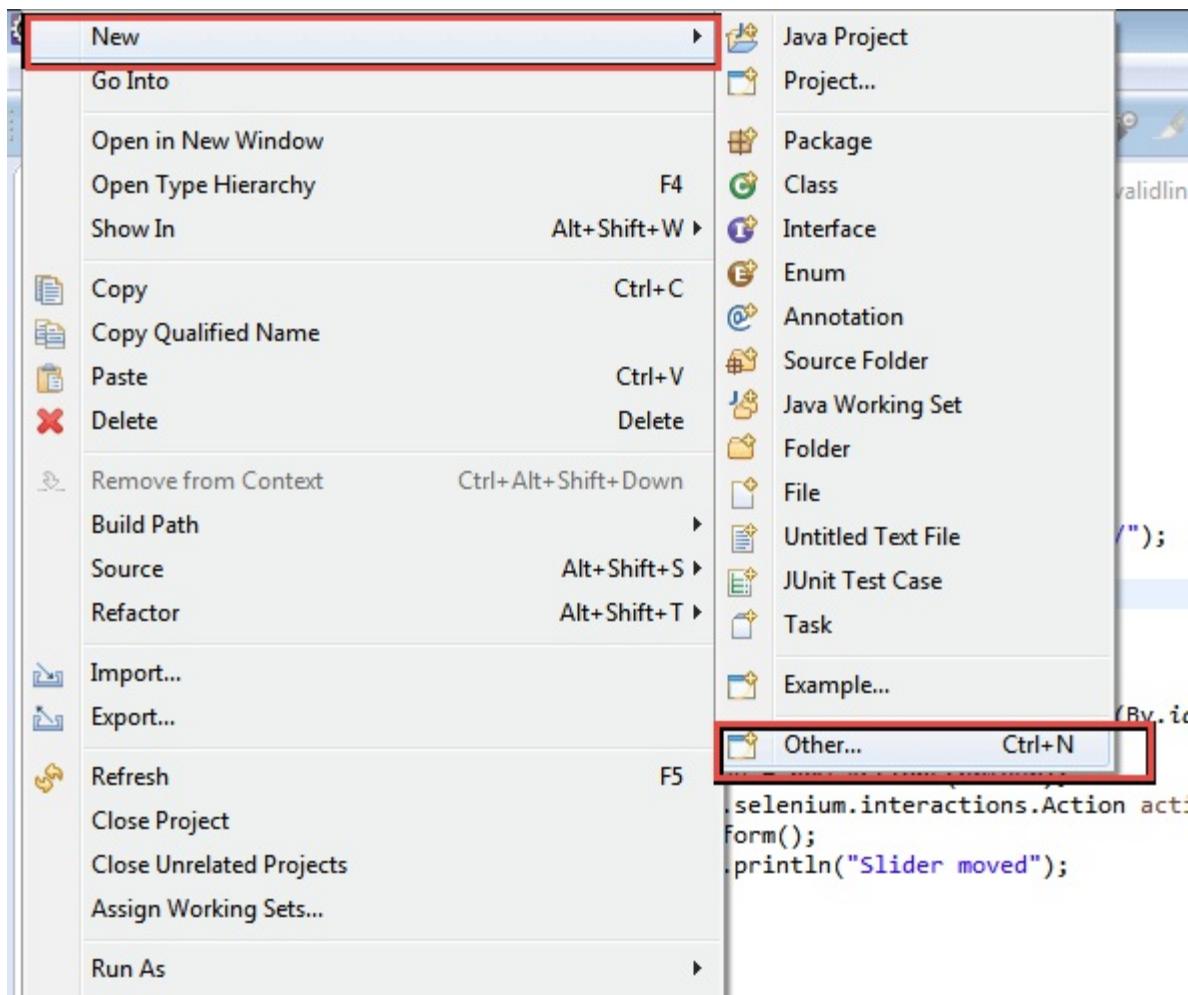


If you see in below screenshot, a new package is created.

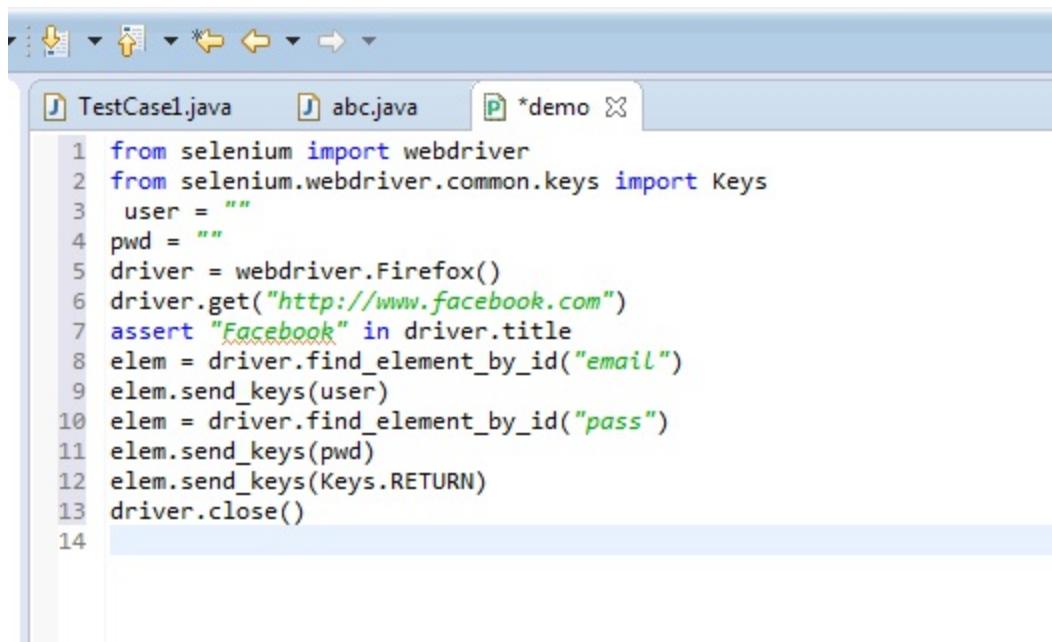


After creating a new package, the next step is to create PyDev Module. The module contains some Python files for initialization. These files or functions from the module can be imported into other module. So, there will be no need to re-write the program again.

Step 9) Create a new PyDev module. Right click on package > New > Other > PyDev module.



Step 10) Write your Python code.



```
1 from selenium import webdriver
2 from selenium.webdriver.common.keys import Keys
3 user = ""
4 pwd = ""
5 driver = webdriver.Firefox()
6 driver.get("http://www.facebook.com")
7 assert "Facebook" in driver.title
8 elem = driver.find_element_by_id("email")
9 elem.send_keys(user)
10 elem = driver.find_element_by_id("pass")
11 elem.send_keys(pwd)
12 elem.send_keys(Keys.RETURN)
13 driver.close()
14
```

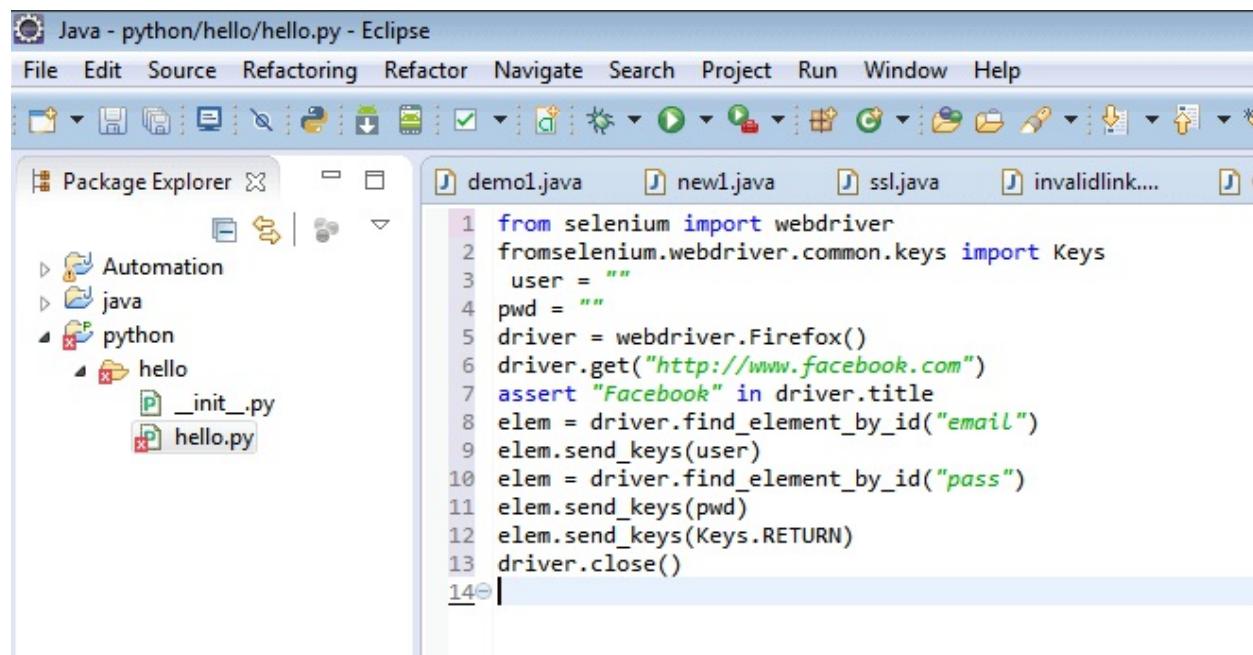
Create Test Scripts in Selenium with Python

- In this example, we did automation for "Facebook login page" using the Firefox driver.

EXAMPLE 1

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
user = ""
pwd = ""
driver = webdriver.Firefox()
driver.get("http://www.facebook.com")
assert "Facebook" in driver.title
elem = driver.find_element_by_id("email")
elem.send_keys(user)
elem = driver.find_element_by_id("pass")
elem.send_keys(pwd)
elem.send_keys(Keys.RETURN)
driver.close()
```

Snapshot of the Code



```

Java - python/hello/hello.py - Eclipse
File Edit Source Refactoring Refactor Navigate Search Project Run Window Help
Package Explorer demo1.java new1.java ssl.java invalidlink...
Automation java python hello __init__.py hello.py

1 from selenium import webdriver
2 from selenium.webdriver.common.keys import Keys
3 user = ""
4 pwd = ""
5 driver = webdriver.Firefox()
6 driver.get("http://www.facebook.com")
7 assert "Facebook" in driver.title
8 elem = driver.find_element_by_id("email")
9 elem.send_keys(user)
10 elem = driver.find_element_by_id("pass")
11 elem.send_keys(pwd)
12 elem.send_keys(Keys.RETURN)
13 driver.close()
14 |

```

Explanation of the code

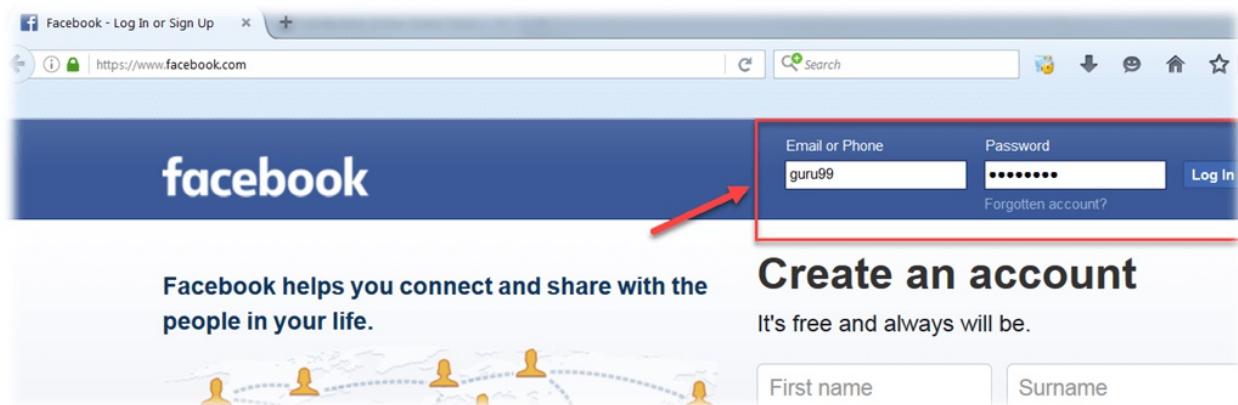
- **Code line 1:** From selenium module import webdriver
- **Code line 2:** From selenium module import Keys
- **Code line 3:** User is a blank variable which will be used to store values of username.
- **Code line 4:** pwd is also a blank (here it is empty, but the user can provide values in it) variable. This will be used to store values of the password.
- **Code line 5:** In this line, we are initializing "FireFox" by making an object of it.
- **Code line 6:** The "driver.get method" will explore to a page given by the URL. WebDriver will hold up until the page has completely been loaded (that is, the "onload" occasion has let go), before returning control to your test or script.
- **Code line 7:** "Asserts" keyword is used to verify the conditions. In this line, we are confirming whether the title is correct or not.

For that, we will compare the title with the string which is given.

- **Code line 8:** In this line, we are finding the element of the textbox where the "email" has to be written.
- **Code line 9:** Now we are sending the values to the email section
- **Code line 10:** Same for the password
- **Code line 11:** Sending values to the password section
- **Code line 12:** Elem.send_keys(Keys.RETURN) is used to press enter after the values are inserted
- **Code line 13:** Close

OUTPUT

The values of the username "guru99" and password entered.



The Facebook page will login with email and password. Page opened (see image below)



EXAMPLE 2

In this example,

- We will open a login page.
- Fill the required field "username" and "password".
- Then validate if the login was successful or not.

```
from selenium import webdriver
from selenium.common.exceptions import TimeoutException

browser = webdriver.Firefox()
browser.get( www.facebook.com )

username = browser.find_element_by_id( "guru99" )
password = browser.find_element_by_id( "password@123" )
submit   = browser.find_element_by_id( "submit" )
```

```
username.send_keys( "me" )
password.send_keys( "mykewlpass" )

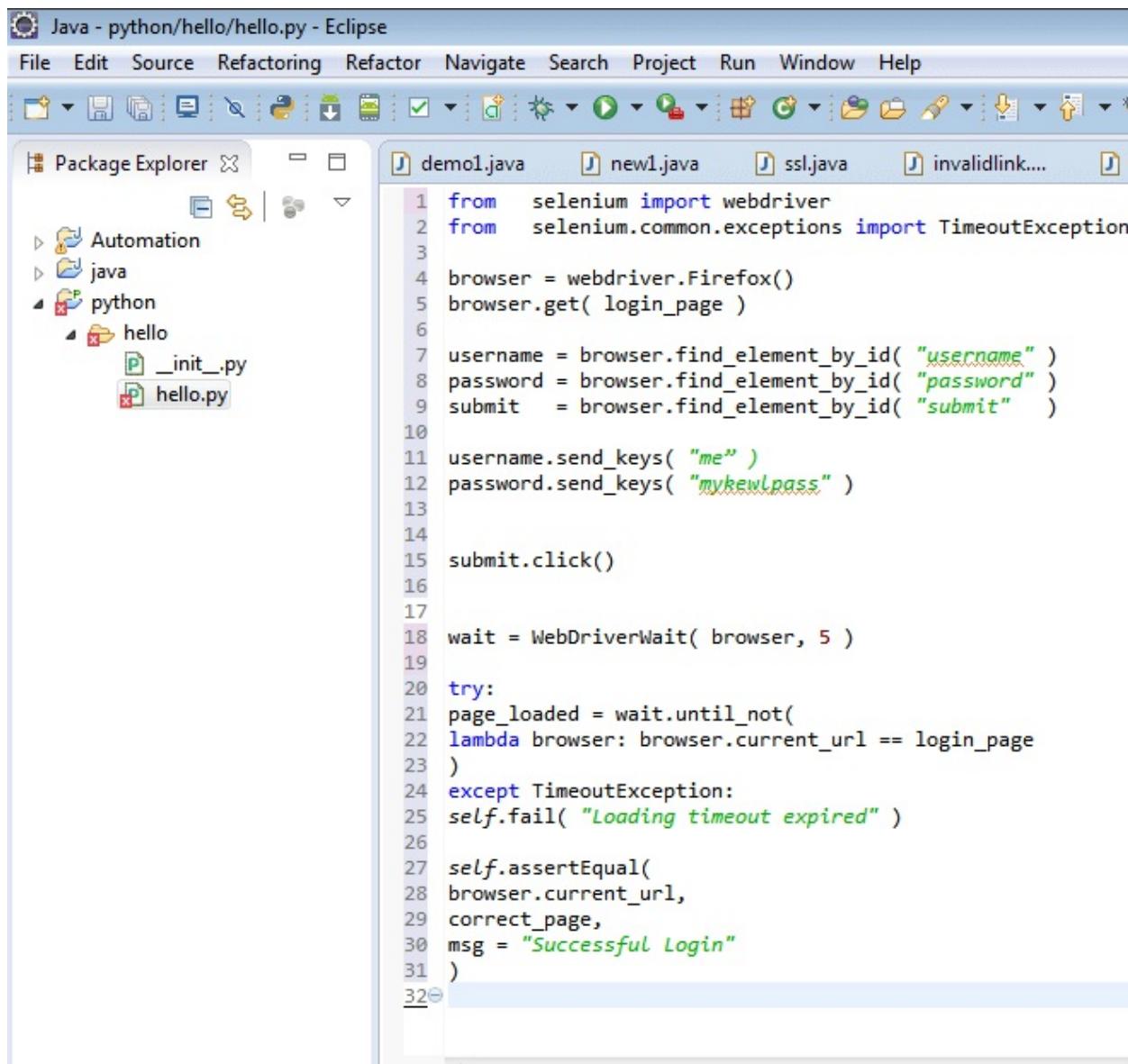
submit.click()

wait = WebDriverWait( browser, 5 )

try:
page_loaded = wait.until_not(
lambda browser: browser.current_url == login_page
)
except TimeoutException:
self.fail( "Loading timeout expired" )

self.assertEqual(
browser.current_url,
correct_page,
msg = "Successful Login"
)
```

Snapshot of the code



The screenshot shows the Eclipse IDE interface with the title bar "Java - python/hello/hello.py - Eclipse". The menu bar includes File, Edit, Source, Refactoring, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, Copy, Paste, and Run. The Package Explorer view on the left shows a project structure with folders Automation, java, python, and hello. Inside the hello folder are files __init__.py and hello.py. The hello.py file is open in the editor, displaying Python code for Selenium web automation. The code uses the selenium package to initialize a Firefox browser, navigate to a login page, and interact with form elements to perform a login. It includes exception handling for timeouts and asserts the correct page title.

```

from selenium import webdriver
from selenium.common.exceptions import TimeoutException
browser = webdriver.Firefox()
browser.get( login_page )
username = browser.find_element_by_id( "username" )
password = browser.find_element_by_id( "password" )
submit = browser.find_element_by_id( "submit" )
username.send_keys( "me" )
password.send_keys( "mykewlpass" )
submit.click()
wait = WebDriverWait( browser, 5 )
try:
    page_loaded = wait.until_not(
        lambda browser: browser.current_url == login_page
    )
except TimeoutException:
    self.fail( "Loading timeout expired" )
self.assertEqual(
    browser.current_url,
    correct_page,
    msg = "Successful Login"
)

```

Explanation of the code:

- **Code line 1-2:** Import selenium package
- **Code line 4:** Initialize Firefox by creating an object
- **Code line 5:** Get login page (Facebook)
- **Code line 7-9:** Fetch username, password input boxes and submit button.
- **Code line 11-12:** Input text in username and password input boxes

- **Code line 15:** Click on the "Submit" button
- **Code line 18:** Create wait object with a timeout of 5 sec.
- **Code line 20 -30:** Test that login was successful by checking if the URL in the browser changed. Assert that the URL is now the correct post-login page

Note: For the above scenarios there will be no output. Since no valid URL is used in the example.

Summary:

- Selenium is an open-source web-based automation tool.
- Python language is used with Selenium for testing. It has far less verbose and easy to use than any other programming language
- The Python APIs empower you to connect with the browser through Selenium
- Selenium can send the standard Python commands to different browsers, despite variation in their browser's design.

Chapter 45: How to use intelliJ & Selenium Webdriver

IntelliJ is an IDE that helps you to write better and faster code. IntelliJ can be used in the option to Java bean and Eclipse.

What is IntelliJ

IntelliJ IDEA is a Java Integrated Development Environment (IDE). It is used for software development. It is developed by JetBrains. It comes under apache2 licenced 'community edition' as well as 'proprietary commercial edition'. It is the finest available Java IDEs. It provides facilities like advanced code navigation and code refactoring capabilities.

The advantage of using IntelliJ is that

- It quickly generates getter and setter methods for object attributes.
- With simple keystrokes, you can wrap a statement in a try-catch or if-else block.
- The IDE delivers inbuilt packaging tools like gradle, SBT, grunt, bower, etc.
- Database like SQL, ORACLE, PostgreSQL, Microsoft SQL Server can be accessed directly from the IDE.
- It supports different languages like Java, Javascript, Clojure, etc.
- It is supported with different operating systems like Windows,

Linux, etc. It can be downloaded from JetBrains official website.

Pre-requisites to IntelliJ with selenium webdriver

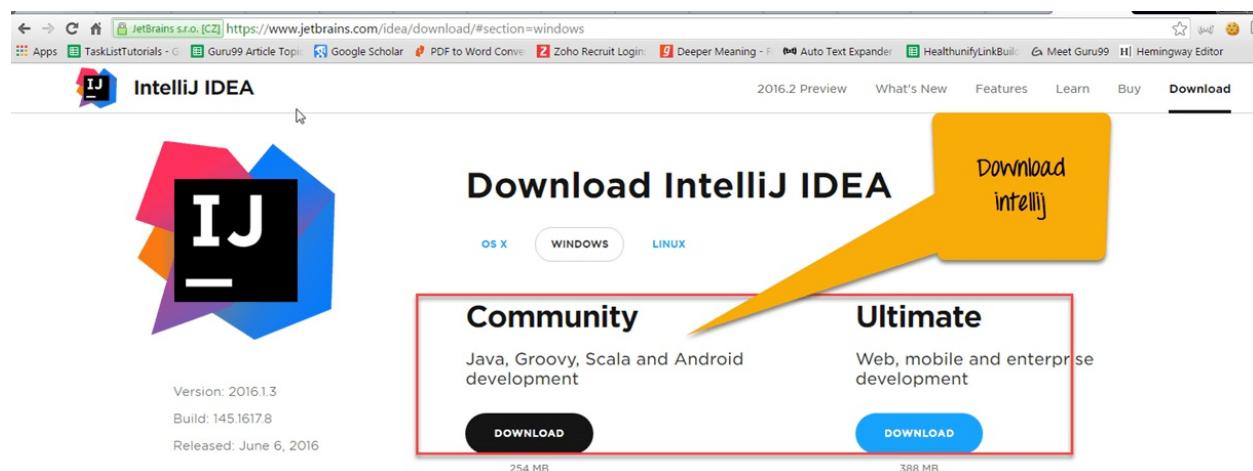
For that, we need to have some pre-requisites which are as follow.

- IntelliJ
- Any Web browser (preferably Mozilla Firefox)
- JDK (Java Development Kit)
- Selenium .jar files

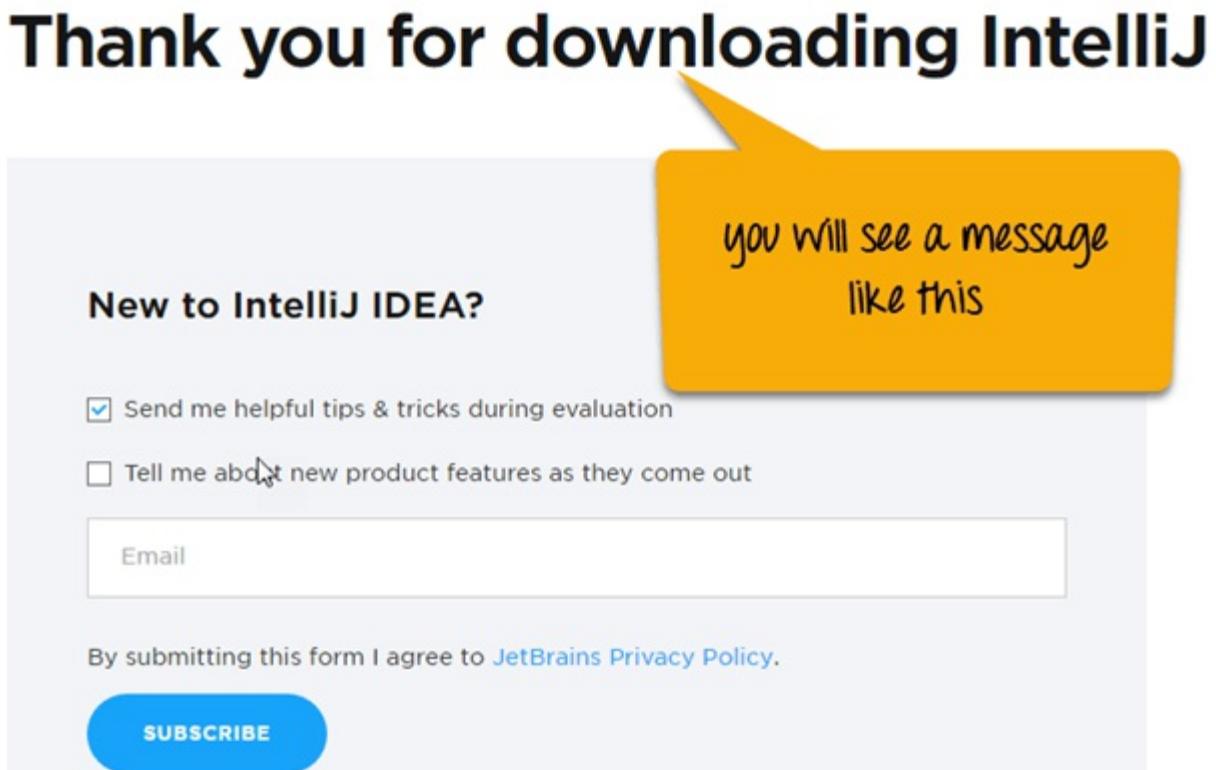
The required jar files can be downloaded from Selenium.org official site. After download, the file extracts the .jar files into the desired directory.

How to Download & Install IntelliJ

Step 1) To download IntelliJ visit the jetbrains site. Here we have selected "Community" Version. You can select "ultimate" version for mobile, web and enterprise development.



Step 2) When you begin downloading, you will see a message like this.



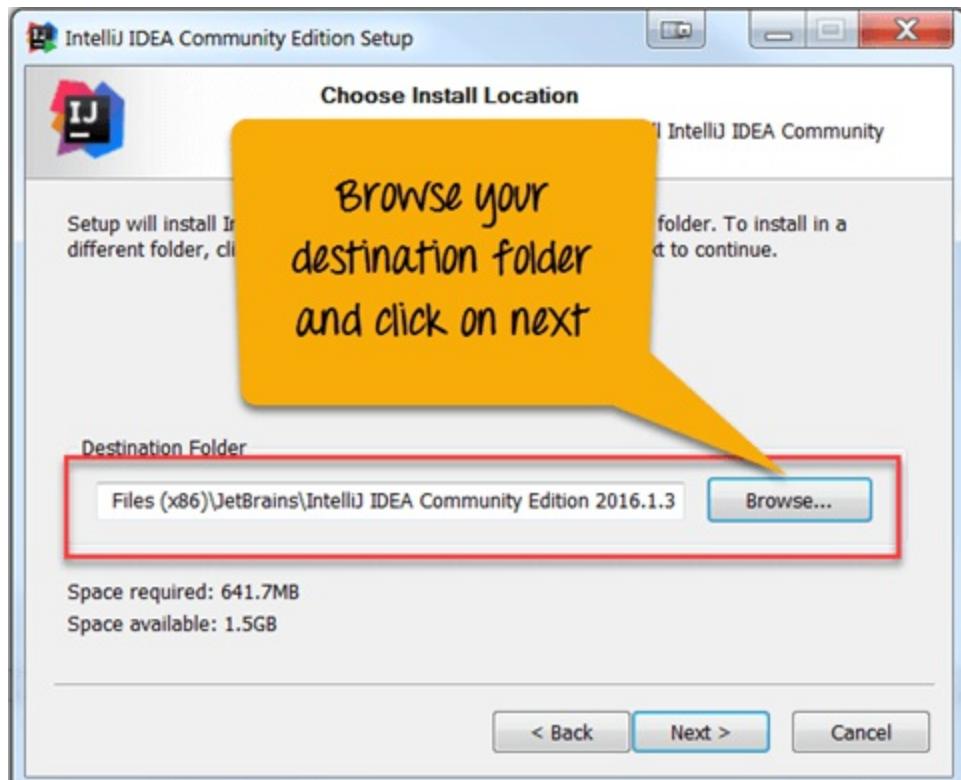
Step 3) In next step, a pop-up window will open. Click on 'run' button.



Step 4) In this step, click on 'next' button in the setup wizard.

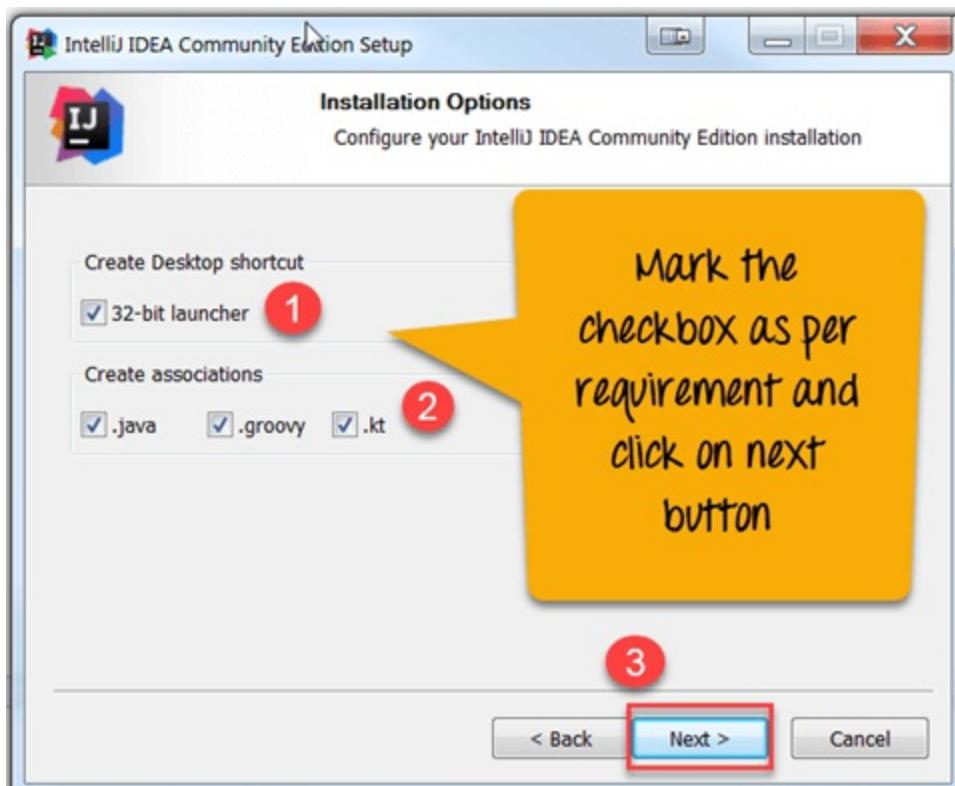


Step 5) Another pop-up window will open. Browse your destination folder and click on 'next' button.

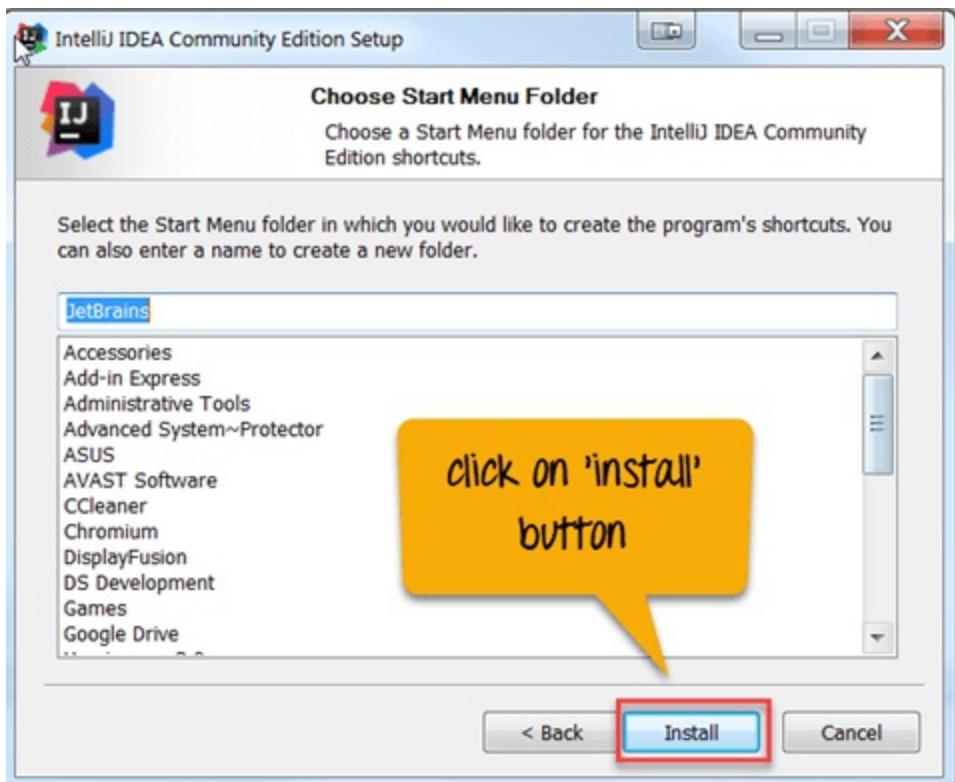


Step 6) In this step,

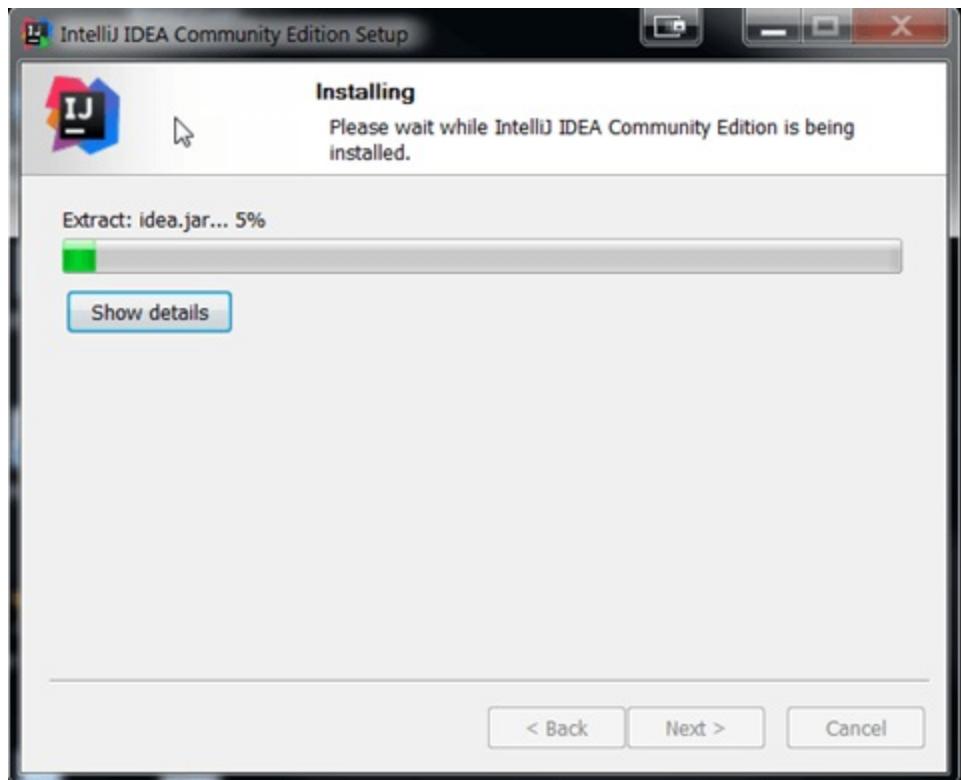
1. Mark the checkbox – 32-bit launcher
2. Mark the checkbox for language as per your requirement
3. Click on 'next' button



Step 7) In next step, click on 'Install' button.

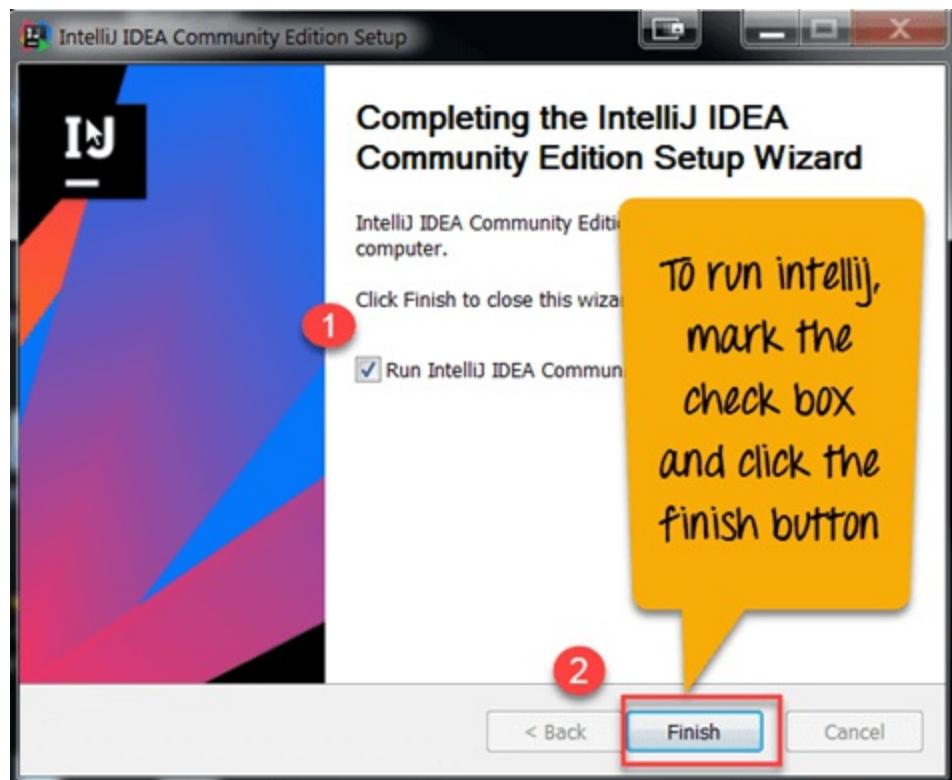


You can see IntelliJ installing process is in progress.



Step 8) In this step,

1. To run IntelliJ, mark the checkbox and
2. Click the 'Finish' button



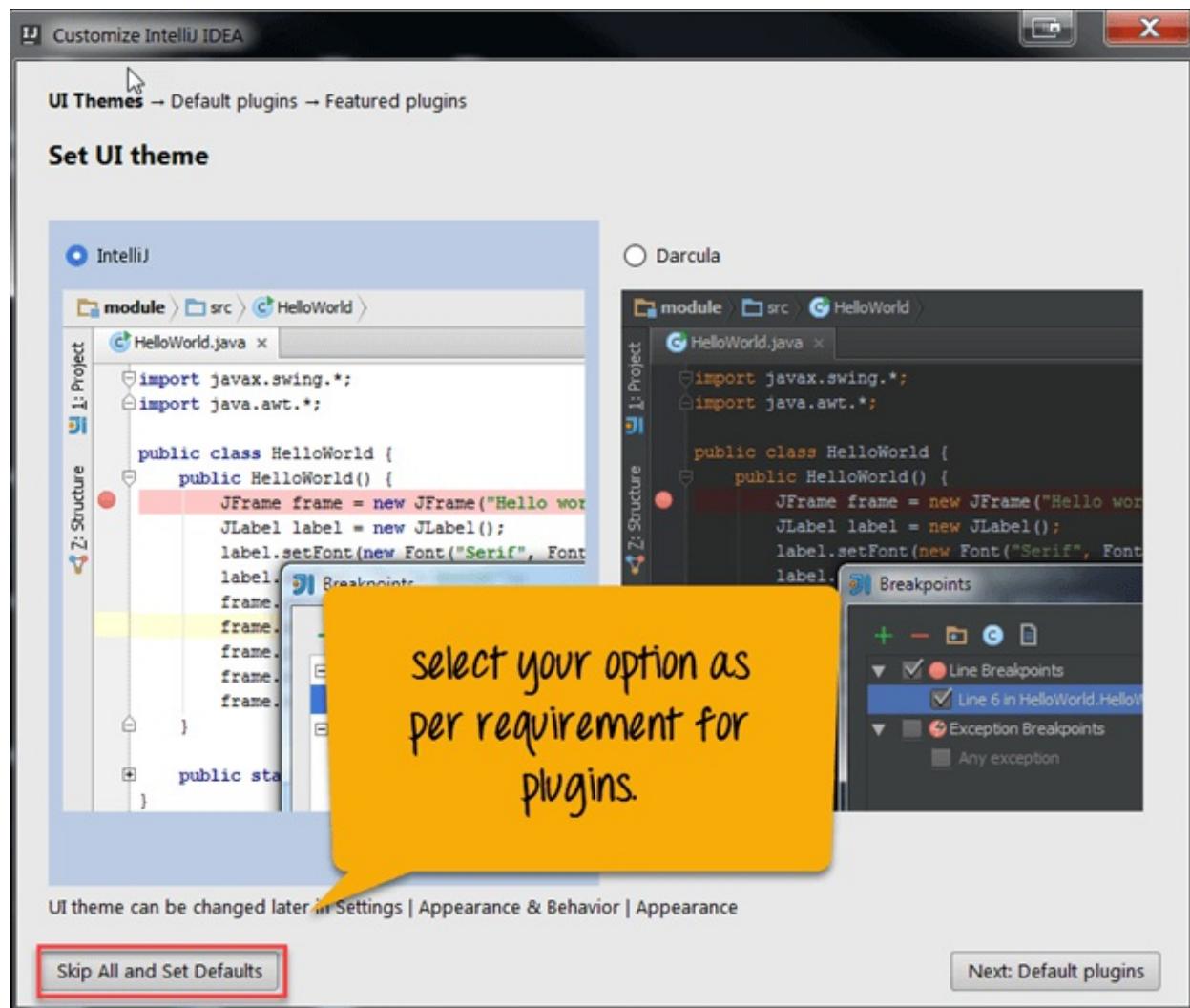
Step 9) If you already have an older version of IntelliJ installed in your system. You can import setting from older version to the newer version. Since we don't have any previous version installed. We will select the second option.



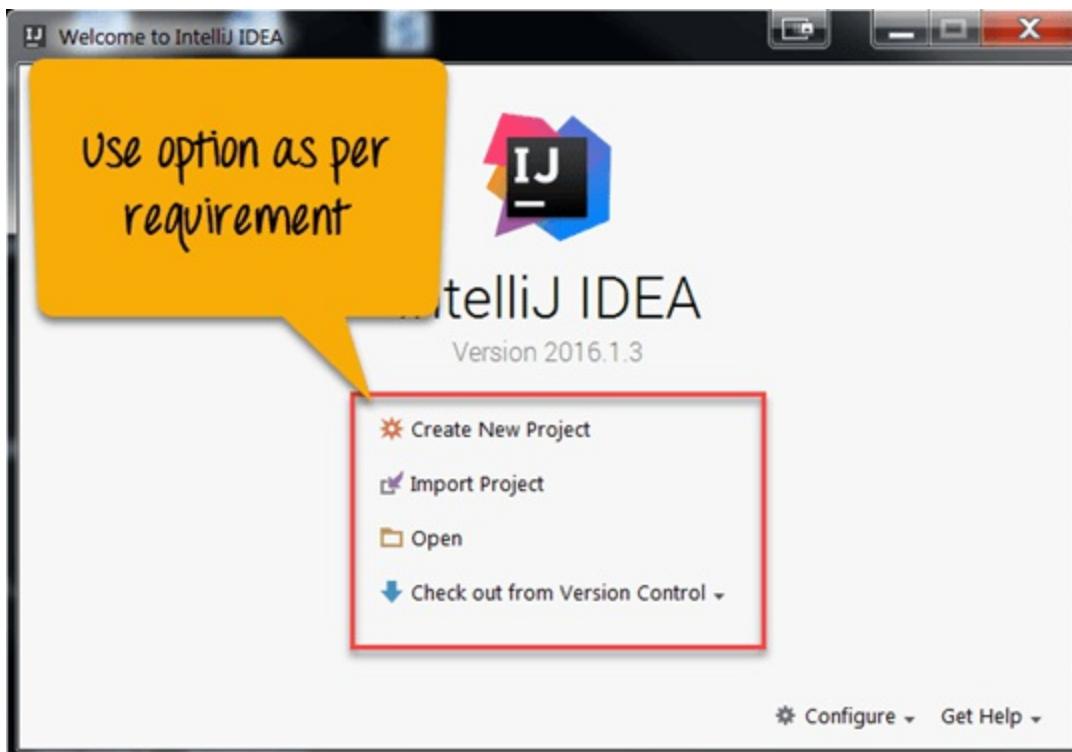
Step 10) When you click on 'ok' button in the previous step, it will ask for Jetbrain privacy policy agreement. Click on 'Accept' button.



Step 11) In this step, you can set plugin setting.

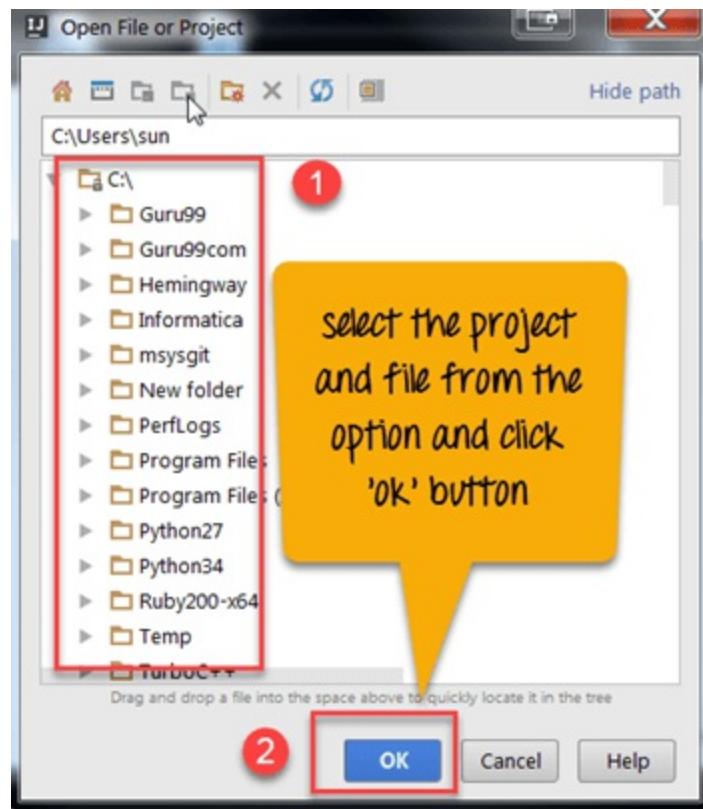


Step 12) In next step, select the option as per requirement. You will see options like create a new project, import project, open, etc.

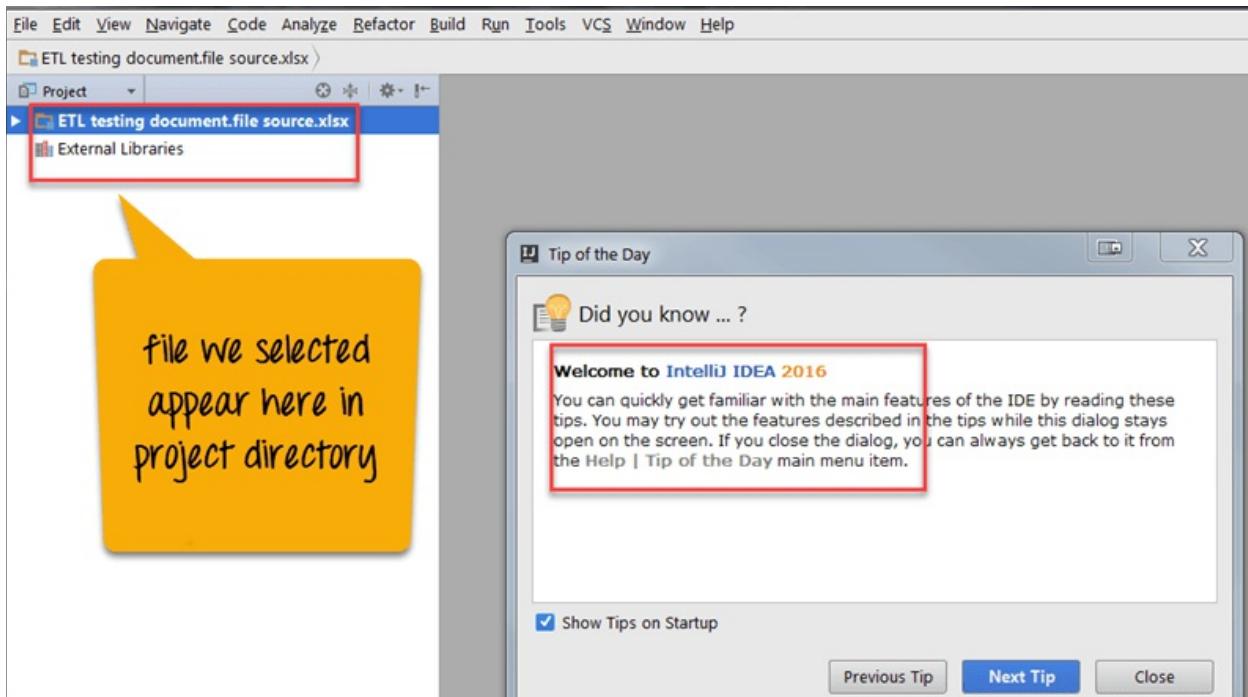


Step 13) In this step,

1. Select the 'Project' and 'file' from the library and
2. Click on 'OK' button



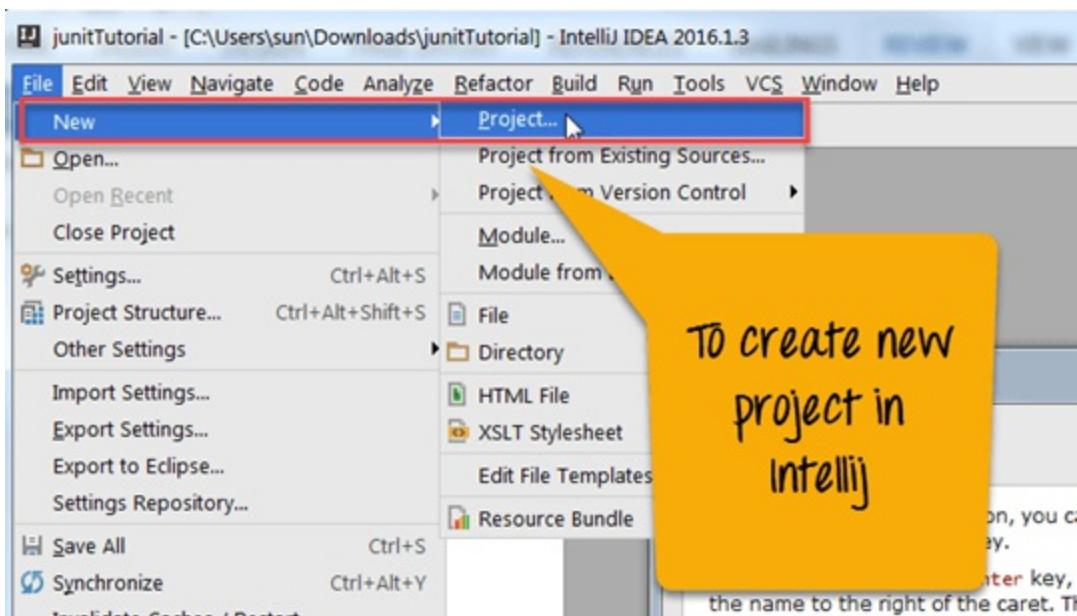
Step 14) In this step, the file we selected in the previous step appears in the project directory.

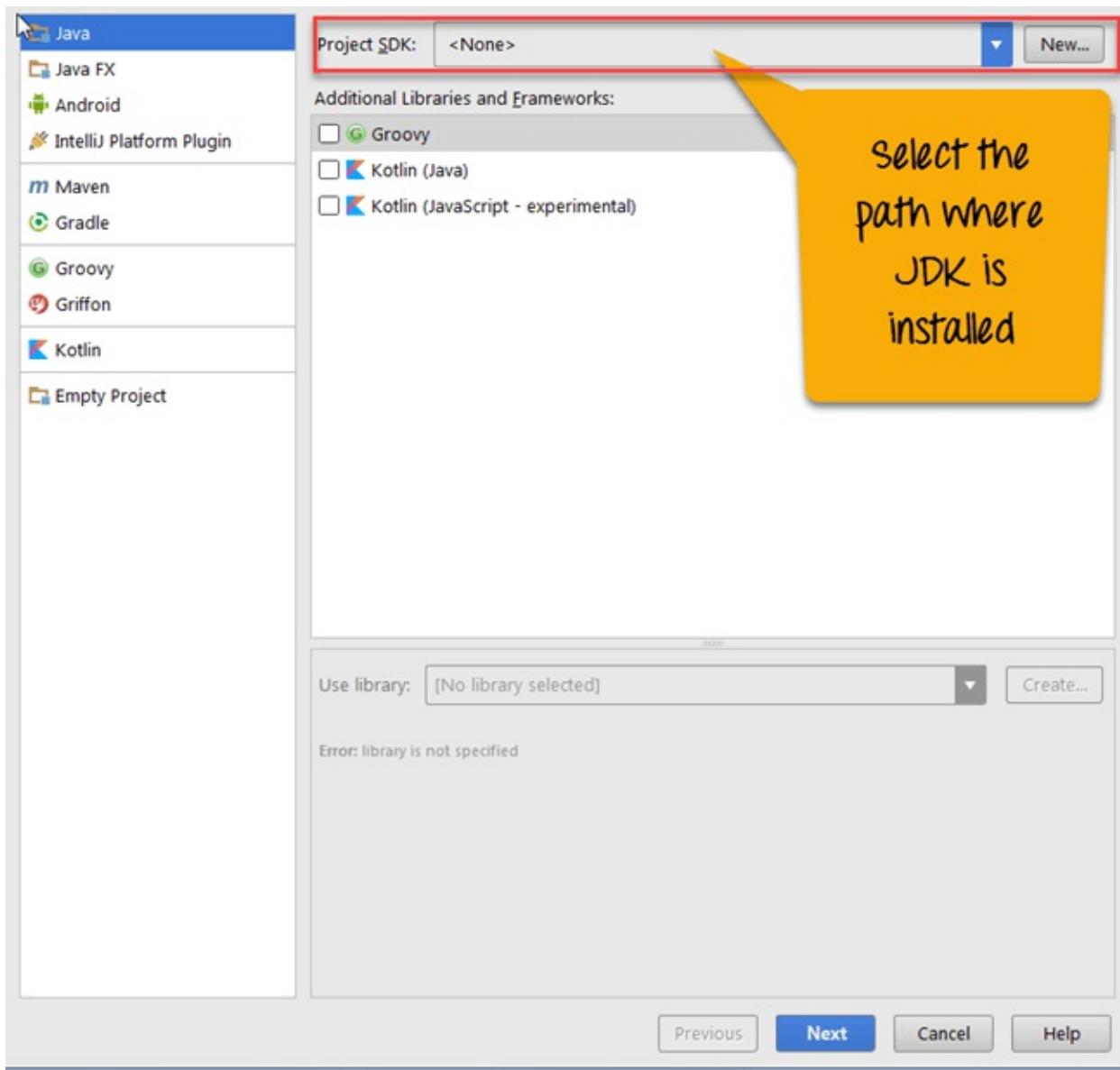


Configure IntelliJ to Support Selenium

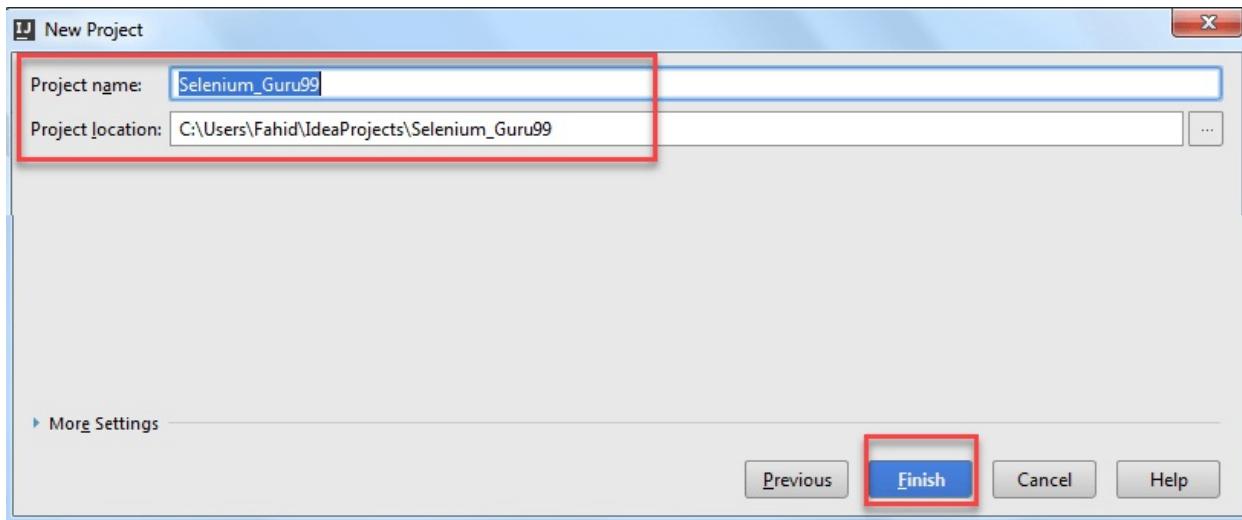
To support Selenium, you need to configure IntelliJ. For that follow the following steps.

Step 1) Launch your IntelliJ IDE and make a new Project. Select File -> New -> Project



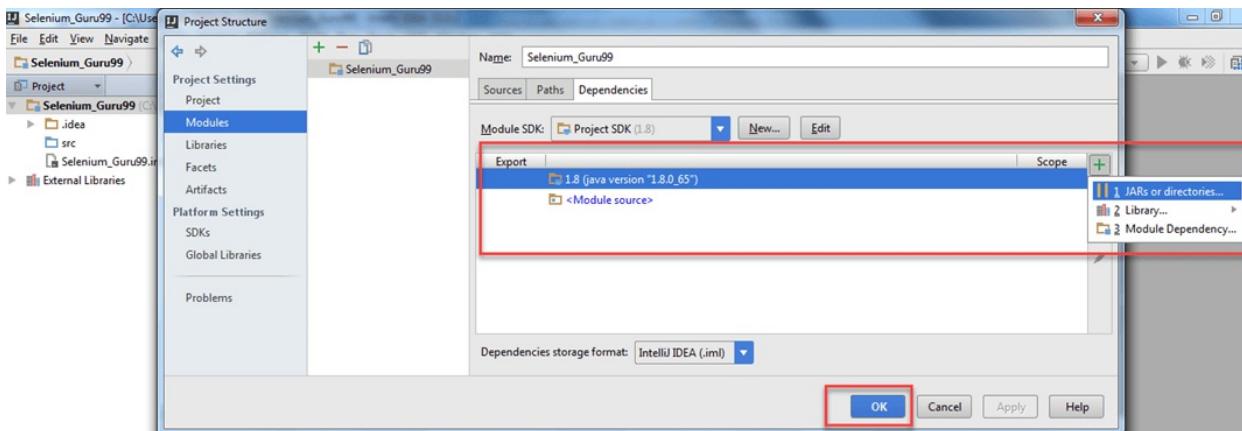


Step 2) In the previous step when you Click -> Next. A new screen will open. In this screen, give project name. In our case, we have given name Selenium_Guru99. Then Click -> Finish. Your project has been created in IntelliJ.

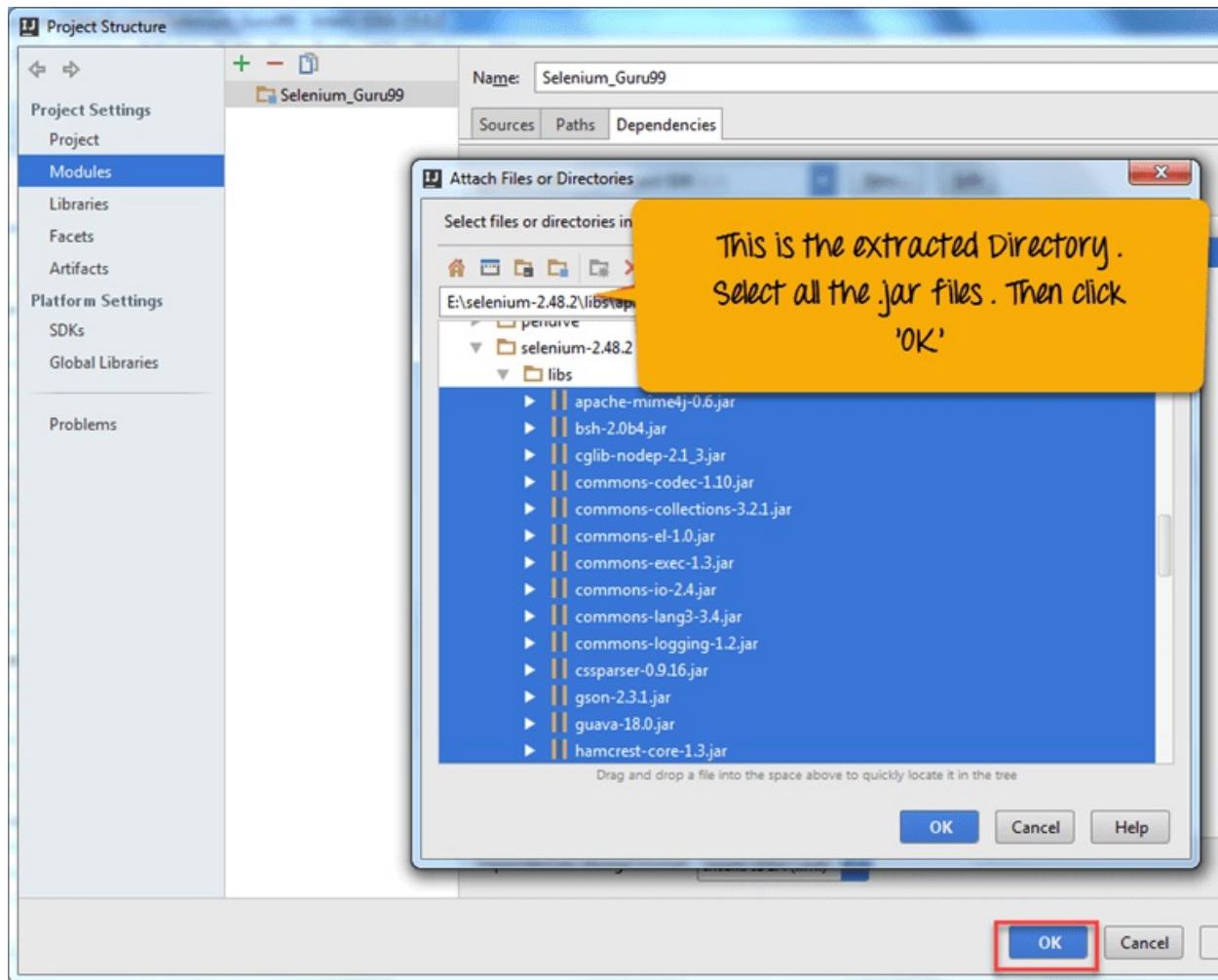


Step 3) Now You need to add the Selenium's .jar files into intelliJ as external libraries.

For this Click go to File -> Project Structure -> in a project setting tab look for Modules -> Dependencies -> Click on '+' Sign -> Select for JARs or directories.

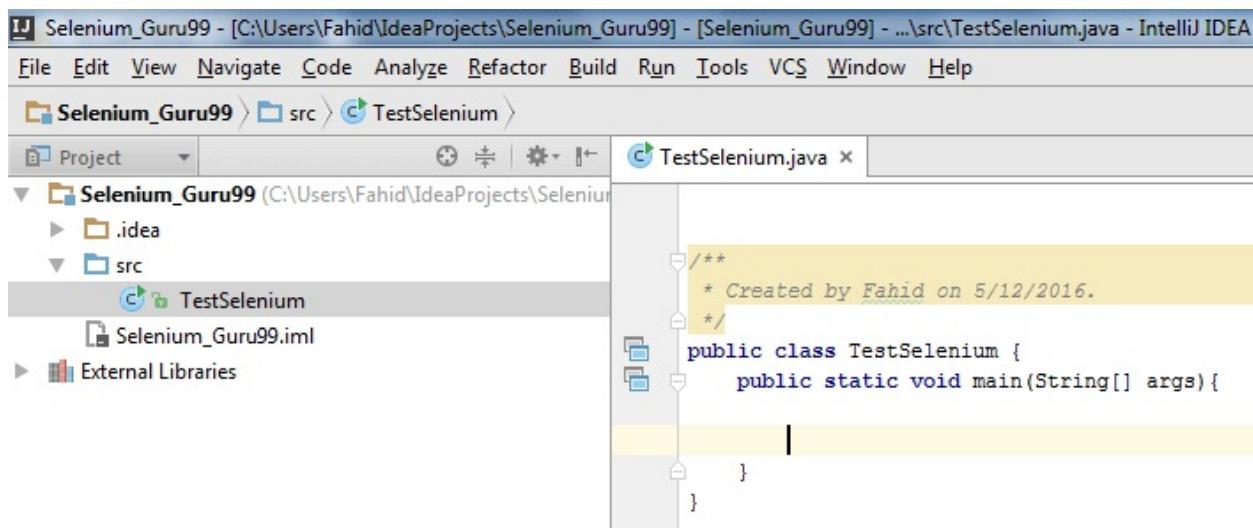


Step 4) Select all the selenium .jar files from the directory and subdirectory /lib, where you have extracted after download.



Now, you have successfully added the .jar files into intelliJ. If you see your project structure, then you will notice that your project's /src directory is empty.

Step 5) Right Click on /src directory -> New -> Java Class. Your project structure will look as shown below.



```
package TestSelenium;
public class TestSelenium {
    public static void main(String[] args) {
}
```

Example

We will use the site <http://demo.guru99.com/>.

In this test scenario

- We will launch the URL
- Enter Invalid Email ID
- Click the 'Submit' button
- The output will be as shown below- 'Email id is not valid'

Guru99 Bank

Enter your email address to get access details to demo site

Email ID abc.gmail.com Email ID is not valid

Submit

In above result, you can see that

- When we run the code, Firefox instance is open.
- At code level, we have provided an email to webelement. Which is an input field (abc.gmail.com).
- When Selenium Webdriver clicks the 'submit' button, email id is verified by guru99 site.
- As we said that unregistered email would show message "Email ID is not valid."

Following is java code for test1.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

public class TestSelenium {
    public static void main(String[] args){
        FirefoxDriver driver=new
FirefoxDriver();
        driver.get("http://demo.guru99.com/");
    }
}
```

```
        WebElement  
element=driver.findElement(By.xpath("//input[@name='emailid']"))  
;  
  
element.sendKeys("abc@gmail.com");  
  
        WebElement  
button=driver.findElement(By.xpath("//input[@name='btnLogin']"))  
;  
        button.click();  
    }  
}
```

Summary

- IntelliJ IDEA is a Java Integrated Development Environment (IDE).
- It does have facilities of advanced code navigation and code refactoring capabilities.
- The advantage of using IntelliJ is
 - Quickly generate getter and setter methods
 - With simple key strokes, you can wrap a statement in a try-catch or if-else block
 - It supports different languages like Java, JavaScript , Clojure, etc.
 - It supports different O.S like Windows, Linux, etc.In
- It comes with inbuilt plugins and packaging tools
- To use with Selenium, you need to configure IntelliJ

Chapter 46: Test Case Priority in TestNG

TestNG is a Testing framework, that covers different types of test designs like a unit test, functional test, end to end test, UI test and integration test.

You can run a single or multiple test cases in your Testng code.

If test priority is not defined while, running multiple test cases, TestNG assigns all @Test a priority as zero(0).

Now, while running; lower priorities will be scheduled first.

Demo of TestNG code without Priority

Let's take a scenario where sequencing will be required in order to pass all test cases:

Scenario: Generate a code where you are required to perform a Google search with a specific keyword say "Facebook". Now, verify that Browser title is changed to "Facebook - Google Search".

Note: Each step which you code should be in separate methods

Method 1: Open Browser say Firefox (openBrowser())

Method 2: Launch Google.com (launchGoogle())

Method 3: Perform a search using "Facebook" (performSearchAndClick1stLink())

Method 4: Verify Google search page title (FaceBookPageTitleVerification())

Code for our scenario:

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

public class Priority_In_testNG {
    WebDriver driver;

    // Method 1: Open Brower say Firefox
    @Test
    public void openBrowser() {
        driver = new FirefoxDriver();
    }

    // Method 2: Launch Google.com
    @Test
    public void launchGoogle() {
        driver.get("http://www.google.co.in");
    }

    // Method 3: Perform a search using "Facebook"
    @Test
    public void performSearchAndClick1stLink() {
        driver.findElement(By.xpath(".//*[@title='Search']")).sendKeys("Facebook");
    }

    // Method 4: Verify Google search page title.
    @Test
    public void FaceBookPageTitleVerification() throws
Exception {
        driver.findElement(By.xpath(".//*[@value='Search']")).click();
        Thread.sleep(3000);
    }
}
```

```

Assert.assertEquals(driver.getTitle().contains("Facebook - Google Search"), true);
}
}

```

Explanation of Code

As mentioned above we have created 4 test cases for performing each action in an independent methods.

- The first method (**openBrowser**) states to initialize Firefox browser.
- The second method (**launchGoogle**) states that launch Google.com is in the initialized browser.
- The third method (**performSearchAndClick1stLink**) states that perform a search in the search box (with xpath `("//*[
[@title='Search']]")` with a search term as **Facebook** and
- The fourth and last method (**FaceBookPageTitleVerification**) states that click on search icon of Google and verify that browser title has been changed to **Facebook - Google Search**.

Now run this code using testNG as shown in the video you will find all the Test Case are failing. The reason for failure: as there is a dependency of previous test case to pass, only than current running test case will be passed.

In this case,

- First method which is executed is **openBrowser()**. It got passed because it does not have any dependency.
- Second method executed is **FaceBookPageTitleVerification()**; it is failing because we are trying to click search button and verifying browser title.

- You can see that if search activity is not process then how any other step can get passed. Hence, this is the reason my test cases are failing.

PASSED: openBrowser

FAILED: FaceBookPageTitleVerification

FAILED: launchGoogle

FAILED: performSearchAndClick1stLink

If we don't mention any priority, testng will execute the @Test methods based on alphabetical order of their method names irrespective of their place of implementation in the code.

```
package com.guru.testngannotations;

import org.testng.annotations.Test;

public class TestNG_Priority_Annotations {

    @Test
    public void c_method(){
        System.out.println("I'm in method C");
    }
    @Test
    public void b_method(){
        System.out.println("I'm in method B");
    }
    @Test
    public void a_method(){
        System.out.println("I'm in method A");
    }
    @Test
    public void e_method(){
        System.out.println("I'm in method E");
    }
    @Test
```

```
public void d_method(){
System.out.println("I'm in method D");
}
}
```

Output

```
I'm in method A
I'm in method B
I'm in method C
I'm in method D
I'm in method E
```

Though we defined the methods in a random manner (c, b, a, e, d), testng executed the methods based on their method names by considering alphabetical order and the same was reflected in the output as well.

Importance of Priority in running testNG methods

As you have seen in the previous example that sequencing required in order to pass this scenario, so we will be modifying the previous piece of code with **Priority Parameter** so that each test should run against to the priority assigned to them.

Now as you can see we have assigned the Priority to each test case means test case will the lower priority value will be executed first.

Priority in testNG in action

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

```

public class Priority_In_testNG {
    WebDriver driver;

    // Method 1: Open Browser say Firefox
    @Test (priority=1)
    public void openBrowser() {
        driver = new FirefoxDriver();
    }

    // Method 2: Launch Google.com
    @Test (priority=2)
    public void launchGoogle() {
        driver.get("http://www.google.co.in");
    }

    // Method 3: Perform a search using "Facebook"
    @Test (priority=3)
    public void performSearchAndClick1stLink() {
        driver.findElement(By.xpath(".//*[@@title='Search']")).sendKeys("Facebook");
    }

    // Method 4: Verify Google search page title.
    @Test (priority=4)
    public void FaceBookPageTitleVerification() throws Exception
    {
        driver.findElement(By.xpath(".//*[@@value='Search']")).click();
        Thread.sleep(3000);
        Assert.assertEquals(driver.getTitle().contains("Facebook - Google Search"), true);
    }
}

```

Explanation of Code

After assigning priority to each testcases, run the above code using testNG as shown in Video-2 mentioned below.

Here, you can see that test cases are prioritized. Test case having lower priority are executed first i.e. now there is a sequential execution

according to priority in the test cases. Hence, all test cases are passing now.

Note the console of eclipse:

Output :

PASSED: openBrowser

PASSED: launchGoogle

PASSED: performSearchAndClick1stLink

PASSED: FaceBookPageTitleVerification

Number 0 has the highest priority(it'll be executed first) and the priority goes on based on the given number i.e., 0 has the highest priority than 1. 1 has the highest priority than 2 and so on.

```
package com.guru.testngAnnotations;
import org.testng.annotations.Test;

public class TestNG_Priority_Annotations {

    @Test(priority=6)
    public void c_method(){
        System.out.println("I'm in method C");
    }
    @Test(priority=9)
    public void b_method(){
        System.out.println("I'm in method B");
    }
    @Test(priority=1)
    public void a_method(){
        System.out.println("I'm in method A");
    }
    @Test(priority=0)
    public void e_method(){
```

```

        System.out.println("I'm in method E");
    }
    @Test(priority=3)
    public void d_method(){
        System.out.println("I'm in method D");
    }

}

```

Output

```

I'm in method E
I'm in method A
I'm in method D
I'm in method C
I'm in method B

```

Here we have provided the priorities as 0,1,3,6,9. So, method having 0 as priority is executed first and then method having priority-1 and so on. Here alphabetical order method name won't be considered as we provided the priorities

Methods with Same Priority:

There may be a chance that methods may contain same priority. In those cases, testng considers the alphabetical order of the method names whose priority is same.

```

package com.guru.testngannotations;
import org.testng.annotations.Test;

public class TestNG_Priority_Annotations {

    @Test(priority=6)
    public void c_method(){
        System.out.println("I'm in method C");
    }
    @Test(priority=9)

```

```

public void b_method(){
    System.out.println("I'm in method B");
}
@Test(priority=6)
public void a_method(){
    System.out.println("I'm in method A");
}
@Test(priority=0)
public void e_method(){
    System.out.println("I'm in method E");
}
@Test(priority=3)
public void d_method(){
    System.out.println("I'm in method D");
}

}

```

Output

```

I'm in method E
I'm in method D
I'm in method A
I'm in method C
I'm in method B

```

Here ‘e’ and ‘d’ are executed based on their priority values. But the methods ‘a’ and ‘c’ contains the same priority value(6). So, here testng considers the alphabetical order of ‘a’ and ‘c’ and executes them accordingly.

Combining both prioritized(having same priority) and non-prioritized methods:

In this case, we'll cover two cases in one testng class.

1. Methods having same priority value.
2. More than one non-prioritized methods.

```
package com.guru.testngannotations;

import org.testng.annotations.Test;

public class TestNG_Priority_Annotations {

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

Output:

```
I'm in method B
I'm in method C
I'm in method E
I'm in method A
I'm in method D
PASSED: b_method
PASSED: c_method
PASSED: e_method
PASSED: a_method
PASSED: d_method
```

Explanation:

First preference: Non-prioritized methods: 'c' and 'b': Based on alphabetical order 'b' was executed first and then 'c'.

Second preference: Prioritized methods: 'a', 'e' and 'd': 'e' was executed first as it was having highest priority(0). As the priority of 'a' and 'd' methods were same, testng considered the alphabetical order of their methods names. So, between them, 'a' was executed first and then 'd'.

Case-sensitive in TestNG

Just for your information there is a standard syntax for defining priority in testNG i.e. **@Test (priority=4)**, suppose you are defining it in some other syntax say **@Test (PRIORITY=1)** then your IDE will show it as a compilation error. Refer image below:

```
// Method 1: Open Browser say Firefox
@Test (PRIORITY=1)
public void openBrowser() {
    driver = new FirefoxDriver();
}

// Method 2: Launch Google.com
@Test (PRIORITY=2)
public void launchGoogle() {
    driver.get("http://www.google.com");
}

// Method 3: Search Facebook
@Test (PRIORITY=3)
public void searchFacebook() {
    driver.findElement(By.name("q")).sendKeys("Facebook");
    driver.findElement(By.name("q")).sendKeys(Keys.RETURN);
}
```

The screenshot shows a Java code editor with three test methods. The first method uses `@Test (PRIORITY=1)`. A tooltip appears over the `PRIORITY` keyword, stating: "The attribute PRIORITY is undefined for the annotation type Test". Below the tooltip, a list of 24 quick fixes is shown, with the first one being "Change to 'priority'". The code editor's status bar at the bottom right shows the text "endKeys("Facebook");".

Conclusion:

As you have seen that if there is a requirement to run a set of test-case in specific sequence then it can be easily done using **Priority** using

testNG as a run tool.

Chapter 47: TestNG: Execute multiple test suites

TestNG enables you to run test methods, test classes and test cases in parallel inside your project. By performing parallel execution, we can reduce the 'execution time' as tests are started and executed simultaneously in different threads.

Here we will see how to run multiple classes (aka different suites) using TestNG.

Creating a TestNG.xml file for executing test

In order to do that follow the below steps.

1. Create a new project in eclipse
2. Create two packages in the projects (name them as com.suite1 and com.suite2)
3. Create a class in each package (name them as Flipkart.java and Snapdeal.java) and copy the below code in respective classes
4. Create a new file in your project and name it as testing.xml (Make sure you've installed testing plugin for eclipse, instructions available here). Testng.xml contains all configuration (classnames, testnames and suitnames).

Flipkart.java

```
package com.suite1;
```

```
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.Alert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;

public class Flipkart{

    WebDriver driver = new FirefoxDriver();
    String username = ""; // Change to your username and
password
    String password = "";

    // This method is to navigate flipkart URL
    @BeforeClass
    public void init() {
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(60,
TimeUnit.SECONDS);

driver.navigate().to("https://www.flipkart.com");
    }

    // To log in flipkart
    @Test
    public void login() {

driver.findElement(By.partialLinkText("Login")).click();
        driver.findElement(
            By.cssSelector(".fk-input.login-
form-input.user-email"))
            .sendKeys(username);
        driver.findElement(
            By.cssSelector(".fk-input.login-
form-input.user-pwd"))
            .sendKeys(password);
    }
}
```

```
        driver.findElement(By.cssSelector(".submit-
btn.login-btn.btn")).click();
    }

    // Search For product
    @Test
    public void searchAndSelectProduct() {
        driver.findElement(By.id("fk-top-search-
box")).sendKeys("moto g3");
        driver.findElement(
            By.cssSelector("search-bar-
submit.fk-font-13.fk-font-bold"))
            .click();

        // select the first item in the search results
        String css = ".gd-row/browse-grid-row:nth-of-
type(1) > div:nth-child(1)>div>div:nth-child(2)>div>a";
        driver.findElement(By.cssSelector(css)).click();
    }

    @Test
    public void buyAndRemoveFromCart() {
        driver.findElement(
            By.cssSelector(".btn-express-
checkout.btn-big.current"))
            .click();
        driver.findElement(By.cssSelector(".remove.fk-
inline-block")).click();
        Alert a = driver.switchTo().alert();
        a.accept();
    }

    @Test
    public void logout() {
        Actions s = new Actions(driver);
        WebElement user =
driver.findElement(By.partialLinkText(username));
        s.moveToElement(user).build().perform();

        driver.findElement(By.linkText("Logout")).click();
    }

    @AfterClass
```

```
    public void quit() {
        driver.close();
    }
}
```

SnapDeal.java

```
package com.suite2;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.Alert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;

public class SnapDeal {

    WebDriver driver = new FirefoxDriver();
    String username = ""; // Change to your username and
password
    String password = "";
    String pinCode = "";

    // This method is to navigate snapdeal URL
    @BeforeClass
    public void init() {
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(60,
TimeUnit.SECONDS);

    driver.navigate().to("https://www.snapdeal.com");
    }

    // To log in flipkart
    @Test
    public void login() {
```

```
driver.findElement(By.xpath("//button[text()='Login']")).click();
;

        driver.switchTo().frame("loginIframe");

driver.findElement(By.cssSelector("div[onClick='getLoginForm()']"))
.click();

driver.findElement(By.id("j_username")).sendKeys(username);

driver.findElement(By.id("j_password_login")).sendKeys(password);
;

driver.findElement(By.id("signin_submit")).click();

        driver.switchTo().defaultContent();
}

// Search For product
@Test
public void searchAndSelectProduct() {
    driver.findElement(By.cssSelector(".col-xs-
20.searchformInput.keyword"))
        .sendKeys("iphone 6s");
    driver.findElement(By.cssSelector(".sd-icon.sd-
icon-search")).click();

    // select the first item in the search results
    String css = ".product_grid_row:nth-of-
type(1)>div:nth-child(1)";
    driver.findElement(By.cssSelector(css)).click();
}

@Test
public void buyAndRemoveFromCart() {

driver.findElement(By.xpath("//li[contains(text(), 'Silver')]"))
.click();
```

```

        driver.findElement(By.id("pincode-
check")).sendKeys(pinCode);
        driver.findElement(By.id("buy-button-
id")).click();

driver.findElement(By.cssSelector("i[title='Delete
Item']")).click();
    Alert a = driver.switchTo().alert();
    a.accept();
}

@Test
public void logout() {

    driver.findElement(By.linkText("START SHOPPING
NOW")).click();
    Actions s = new Actions(driver);
    WebElement user =
driver.findElement(By.cssSelector(".sd-icon.sd-icon-user"));
    s.moveToElement(user).build().perform();

driver.findElement(By.linkText("Logout")).click();
}

@AfterClass
public void quit() {
    driver.close();
}
}

```

TestNg.xml

```

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

<suite thread-count="1" verbose="1" name="Gmail Suite"
annotations="JDK" parallel="tests">

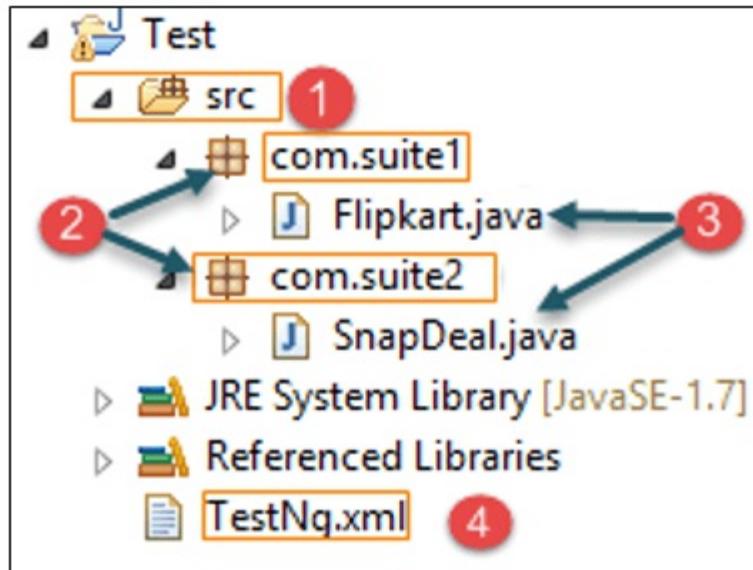
<test name="flipkart">
    <classes>
        <class name="com.suite1.Flipkart"/>
    </classes>

```

```
</test>

<test name="Myntra">
    <classes>
        <class name="com.suite2.SnapDeal"/>
    </classes>
</test>
</suite>
```

Final project structure looks like below,



Parallel execution in TestNG

After creating xml file as shown above, in next step, we will execute the parallel test. Below is the code.

```

<!DOCTYPE suite > 1 TEM "http://testing.org/testng-1.0.dtd"
<suite thread-count="1" verbose="1" name="Gmail Suite" annotations="JDK" parallel="tests"> 2
  <test name="flipkart">
    <classes>
      <class name="com.suite1.Flipkart"/>
    </classes>
  </test> 3

  <test name="Myntra">
    <classes>
      <class name="com.suite2.SnapDeal"/>
    </classes>
  </test> 4
</suite>

```

The diagram illustrates the annotations in the TestNG XML configuration file. Annotations 1 and 2 are grouped together and point to a red box labeled 'Name of the test'. Annotations 3 and 4 are also grouped together and point to another red box labeled 'Name of the classes'.

1) thread-count: This is used for parallel execution, based on the number script. It will execute in parallel or sequential order.

2) verbose: It is used to log the execution details in the console. The value should be 1-10. The log details in the console window will get more detailed and clearer as you increase the value of the verbose attribute in the testng.xml configuration file.

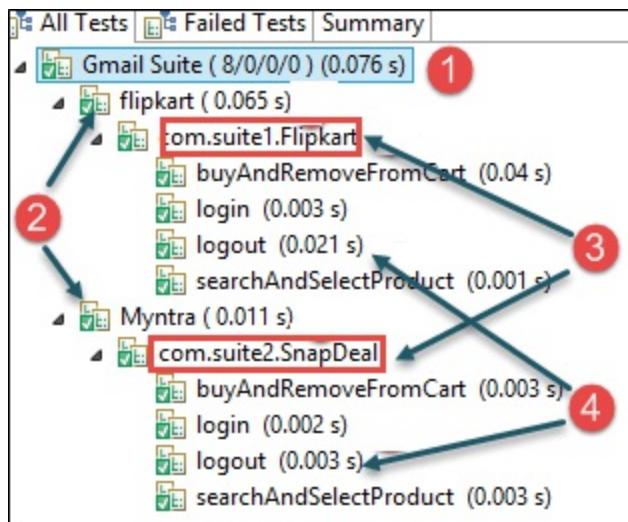
3) name: Name of the suite. Here it is "Gmail Suite"

4) Parallel: To run scripts parallel, value can be tests/classes/methods/suites. Default value is **none**

Right click on the testing.xml and select run as testing, once successful you'll see all the results

When you execute the above code, you will get the following output.

Output:



1 name of the suite given in testng.xml

```
<suite thread-count="1" verbose="1" name="Gmail Suite" annotations="JDK" parallel="tests">
```

2 name of the test given in testng.xml

```
<test name="flipkart">
```

3 name of the class given in testng.xml

```
<class name="com.suite1.Flipkart"/>
```

4 method names annotated with @Test in .java file

```
@Test  
public void buyAndRemoveFromCart() {
```

Likewise, it will execute test suite for snap deal as well.

Conclusion:

Here we have seen how to use Testng to execute parallel test. TestNG gives an option to execute multiple test in parallel in a single configuration file (XML).

Chapter 48: Introduction to TestNG Groups

TestNG is a Testing framework that covers different types of test designs like unit, functional, end to end, UI and integration test.

You can run a single or multiple packages (package here means to encapsulate a group of classes in a proper director format) by creating XML and run it through maven.

TestNG groups with Example

We use groups in Testng when,

- We don't want to define test methods separately in different classes (depending upon functionality) and
- At the same time want to ignore (not to execute) some test cases as if they does not exist in the code.
- So to carry out this we have to Group them. This is done by using "include" and "exclude" mechanism supported in testNG.

In below example, we have shown the syntax of how to use groups in the XML file.

```
@Test (groups = { "bonding", "strong_ties" })
```

Here we are using 2 group names i.e. "bonding" and "strong_ties" (these are logical name that can be altered as per your wish).

<groups> tag defines the starting of groups in XML.

Customize your XML to pick the mentioned group from the test classes. Below mentioned is the syntax of how to declare groups in XML file e.g.

```
<groups>
  <run>
    <include name="bonding" />
  </run>
</groups>
```

So, let us assume that there are 10 test methods in a class.

Out of them,

- 6 methods are tagged in "bonding" group and
- 4 are in "strong_ties" group

Moving forward, we are going to set maven/Java path and use the Eclipse IDE to demonstrate the usage of groups using XML files in Java based maven project.

Set maven and Java path in environment variable (for windows user)

Please refer <https://www.guru99.com/maven-jenkins-with-selenium-complete-tutorial.html>

<https://www.guru99.com/install-java.html>

Introduction to XML and how to make an XML files

- XML (Extensible Markup Language) file in Maven framework contains the information of one or more tests and is defined by the **tag <suite>**.
- Test information in XML is represented by **tag <test>** and can contain one or more TestNG classes.
- A Java class which contains **@Test** annotation above test methods is defined as TestNG methods.

Multiple tags are used in a sequence to build a working testNG xml like `<suite>`, `<test>` and `<class>`

- First is `<suite>` tag, which holds a logical name which defines full information to testNG reported to generate execution report.
- Second is `<test name="Guru 99 Smoke Test Demo">`, note it is logical name which holds the information of test execution report like pass, fail, skip test cases and other information like total time for execution and group info
- Third is `<class name="com.group.guru99.TC_Class1" />`, com.group.guru99 is the package used, and Test Class name is TC_Class1.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
    <suite name="Suite">
        <test name="Guru 99 Smoke Test Demo">
            <groups>
                <run>
                    <include
name="strong_ties" />
                </run>
            </groups>
            <classes>
                <class
name="com.group.guru99.TC_Class1" />
            </classes>
        </test>
    </suite>

```

```
</suite>
```

We will be using this XML for upcoming video downside.

Another mechanism instead of Grouping is "exclude" or "include" in test XML

Suppose you are finding the usage of group mechanism complex then testNG XML facilitate the functionality to exclude/include a test.

- **Exclude Tag:** Syntax for exclude tag `<exclude name="${TEST_CASE_NAME}" />`
- **Include Tag:** Syntax for include tag `<include name="${TEST_CASE_NAME}" />`

Note: We can include/exclude multiple test cases once at a time, and it works with Groups as well.

How to run code using XML file (video demo)

Explanation of the Java Code and XML with the **group**, **exclude** and **include** the tag in XML.

- **Scenario:** Launch Guru99 demo Banking site, verify few thing's on login page after that enter credentials and re-verify few new thing on the application when logged in.

Welcome To Manager's Page of Guru99 Bank
Manger Id : mngr28642

Note: Each step which you code should be declared in separate methods, but when executed, it will execute test methods depending upon the entries in the XML file.

Method 1: Initialize Browser and launch URL (tco1LaunchURL())

Method 2: Verify Login Page Heading (tco2VerifyLoginPage())

Method 3: Enter userName and Password on login form (tco3EnterCredentials())

Method 4: Verify the presence of Manager ID on User Dashboard (tco4VerifyLoggedInPage())

Method 5: Verify few more links on User DashBoard (tco5VerifyHyperlinks())

Code for our scenario:

```
package com.group.guru99;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
```

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

public class TC_Class1 {
    public static final WebDriver webDriver = new
FirefoxDriver();

    String launchPageHeading = "//h3[text()='Guru99 Bank']";
    final String userName_element = "//input[@name='uid']",
password_element = "//input[@name='password']",
            signIn_element = "//input[@name='btnLogin']";
    final String userName_value = "mngr28642", password_value =
"ydAnate";
    final String managerID = "//td[contains(text(), 'Manger
Id')]";
    final String newCustomer =
"//a[@href='addcustomerpage.php']", fundTransfer =
"//a[@href='FundTransInput.php']";

    /**
     * This test case will initialize the webDriver
     */
    @Test(groups = { "bonding", "strong_ties" })
    public void tc01LaunchURL() {
        webDriver.manage().window().maximize();
        webDriver.manage().timeouts().implicitlyWait(20,
TimeUnit.SECONDS);
        webDriver.get("http://www.demo.guru99.com/V4/");
    }

    /**
     * Will check the presence of Heading on Login Page
     */
    @Test(groups = { "bonding" })
    public void tc02VerifyLaunchPage() {

Assert.assertTrue(webDriver.findElement(By.xpath(launchPageHeadi
ng)).isDisplayed(),
            "Home Page heading is not displayed");
        System.out.println("Home Page heading is displayed");
    }
}
```

```

    /**
     * This test case will enter User name, password and will
     then click on
     * signIn button
     */
    @Test(groups = { "bonding", "strong_ties" })
    public void tc03EnterCredentials() {

        webDriver.findElement(By.xpath(userName_element)).sendKeys(userName_value);

        webDriver.findElement(By.xpath(password_element)).sendKeys(password_value);
            webDriver.findElement(By.xpath(signIn_element)).click();
    }

    /**
     * This test case will verify manger's ID presence on
     DashBoard
     */
    @Test(groups = { "strong_ties" })
    public void tc04VerifyLoggedInPage() {

        Assert.assertTrue(webDriver.findElement(By.xpath(managerID)).isDisplayed(),
                            "Manager ID label is not displayed");
        System.out.println("Manger Id label is displayed");
    }

    /**
     * This test case will check the presence of presence of New
     customer link
     * And FundTransfer link in Left pannel
     */
    @Test(groups = { "bonding" })
    public void tc05VerifyHyperlinks() {

        Assert.assertTrue(webDriver.findElement(By.xpath(newCustomer)).isEnabled(),
                            "New customer hyperlink is not displayed");
        System.out.println("New customer hyperlink is
displayed");
    }
}

```

```
Assert.assertTrue(webDriver.findElement(By.xpath(fundTransfer)).  
isEnabled(),  
        "Fund Transfer hyperlink is not displayed");  
System.out.println("Fund Transfer hyperlink is  
displayed");  
}  
}
```

Please Note: The credentials are only valid for 20 days, so if you are trying to run code on your local machine, so you might face invalid credentials error. Please find below steps to generate your login credentials:

1. Launch <http://www.demo.guru99.com>
2. Enter your email id in the box.
3. Click enter and see your login details on screen.

Explanation of Code:

As mentioned above, we have created 5 test cases for performing each action in independent methods.

You can observe that to every method, we have associated a group parameter holding some value in it.

Basically, these are the name of the differentiating groups i.e. "strong_ties" & "bonding".

- First and Third methods are tagged to "bonding", "strong_ties" which means if XML is updated in any of the group, this Test Case will run.
- The second method is only tagged to "bonding" group it means

that if XML is updated with bonding group. Only in that case this test case will run.

- Fourth Test case is tagged to strong_ties group, which means this test case will only run if XML is updated with strong_ties group name.
- Last but not the least fifth test case is attached to bonding group, which means this test case will only run if XML is updated with bonding group name.

So overall, we have 4 scenarios;

1. We want to run all test cases irrespective of the group name. In this case, we will remove Group tag from running XML.
2. We want to run test case few test that are related only to either of groups i.e. strong_ties or bonding
- In continuation to video, now we have included group name in XML, you can see only test cases specific to that group is only running.
3. We are using Exclude mechanism to exclude the test case:
 - You see that we have used exclude few test case (tco2) by writing their name in running XML. In final result mentioned test cases did not run.
4. Last, we are using include test mechanism to include the test cases (tco1LaunchURL, tco3EnterCredentials and tco5VerifyHyperlinks)

Conclusion

We have learned here relatively a new way for running test cases using XML in Maven project.

We started by providing a brief introduction on testNG and continued

with the full technical specification of Groups, exclude and include.

Chapter 49: Verify Tooltip Using Selenium WebDriver

The tooltip is a text that appears when a mouse hovers over an object like a link, an image, a button, text area, etc. in a web page. The text often gives more information about the object on which it appears.

Tooltips were traditionally implemented as a 'title' attribute to an element. The value of this attribute was shown as a tooltip on mouse-hover. This is a static text giving information of the element with no styling.

Now, there are many plugins available for 'tool tips' implementation. Advanced tooltips with styling, rendering, images and links are being implemented using JavaScript/JQuery plugins or using CSS Tooltips.

- For accessing or verifying the static tooltips which are implemented using the HTML "title" attribute, we can simply use the `getAttribute("title")` method of the WebElement. The returned value of this method (which is the tooltip text) is compared with an expected value for verification.
- For other forms of tooltip implementations, we will have to use the "Advanced User Interactions API" provided by the Web Driver to create the mouse hover effect and then retrieve the tooltip for the element.

A Brief of the Advanced User Interactions API:

Advanced User Interactions API provides the API for user actions like drag and drop, hovering, multi selecting, key press and release and other actions using keyboard or mouse on a webpage.

You can refer this link for more details on the API.

<https://seleniumhq.github.io/selenium/docs/api/java/index.html?org/openqa/selenium/interactions/Actions.html>

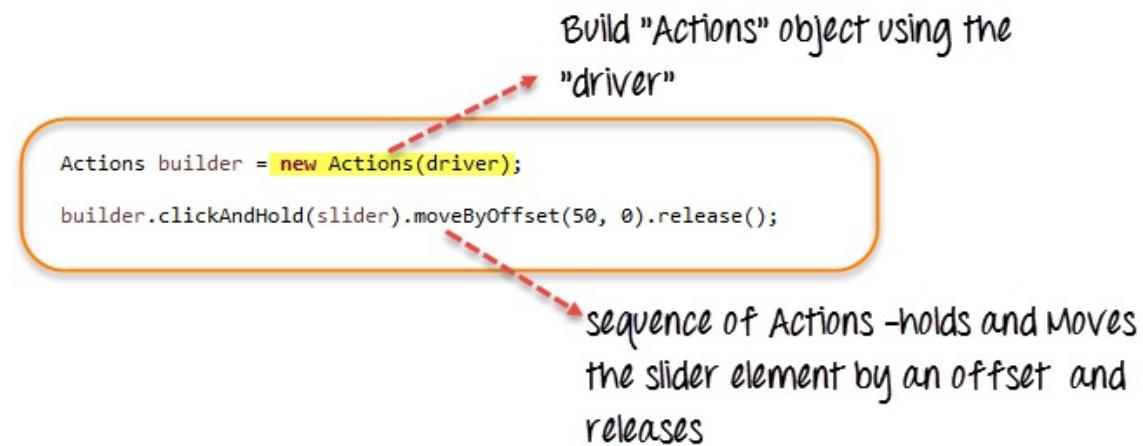
Here, let's see how to use a couple of classes and methods we would need to move a slider element by an offset.

Step 1) In order to use the API, the following packages/classes needs to be imported:

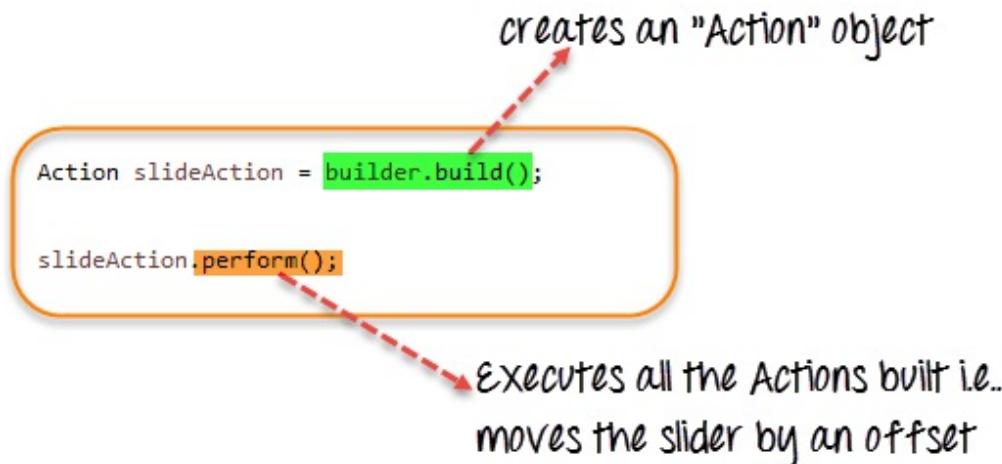
```
import org.openqa.selenium.interactions.Action;  
import org.openqa.selenium.interactions.Actions;
```

Step 2) Create an object of "Actions" class and build the Sequence of user actions. Actions class is used to build the sequence of user actions like moveToElement(), dragAndDrop() etc. Various methods related to user actions are provided by API.

The driver object is provided as a parameter to its constructor.



Step 3) Create an Action Object using the build() method of "Actions" class. Call the perform() method to execute all the actions built by the Actions object(builder here).



We have seen how to use some of the user Actions methods provided by the API - clickAndHold(element), moveByOffset(10,0), release(). The API provides many such methods.

Refer to the link for more details.

Verifying Tool Tips

Let's see the demonstration of accessing and verifying the tool tips in the simple scenario

- Scenario 1: Tooltip is implemented using the "title" attribute
- Scenario 2: Tooltip is implemented using a jQuery plugin.

Scenario 1: HTML 'title' Attribute

For this case, let's take the example site -

<http://demo.guru99.com/test/social-icon.html>.

We will try to verify the tooltip of the "github" icon at the top right of the page.



In order to do it, we will first find the element and get its 'title' attribute and verify with the expected tool tip text.

Since, we are assuming the tool tip is in the "title" attribute, we are not even automating the mouse hover effect but simply retrieving the attribute's value using the "getAttribute()" method.

The github icon at the top right header of the practice form page



```

String expectedTooltip = "Github";

// Find the Github icon at the top right of the header
WebElement github = driver.findElement(By.xpath("//[@class='soc-ico show-round']/a[4]")); ①

//get the value of the "title" attribute of the github icon
String actualTooltip = github.getAttribute("title"); ②

//Assert the tooltip's value is as expected
Assert.assertEquals(expectedTooltip, actualTooltip); ③



④


```

1) Find the "github" icon element.

2) Get its "title" attribute.

3) Assert the expected value against the actual one.

Here is the code

```

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

public class ToolTip {
    public static void main(String[] args) {

        String baseUrl = "http://demo.guru99.com/test/social-
icon.html";

        System.setProperty("webdriver.chrome.driver", "G:\\chromedriver.e
xe");
        WebDriver driver = new ChromeDriver();
        driver.get(baseUrl);
        String expectedTooltip = "Github";

        // Find the Github icon at the top right of the header
        WebElement github = driver.findElement(By.xpath("//*
[@class='soc-ico show-round']/a[4]"));

        //get the value of the "title" attribute of the github
        //icon
        String actualTooltip = github.getAttribute("title");
    }
}

```

```
//Assert the tooltip's value is as expected
System.out.println("Actual Title of Tool
Tip"+actualTooltip);
if(actualTooltip.equals(expectedTooltip)) {
    System.out.println("Test Case Passed");
}
driver.close();
}
```

Explanation of code

1. Find the WebElement representing the "github" icon.
2. Get its "title" attribute using the getAttribute() method.
3. Assert the value against the expected tooltip value.

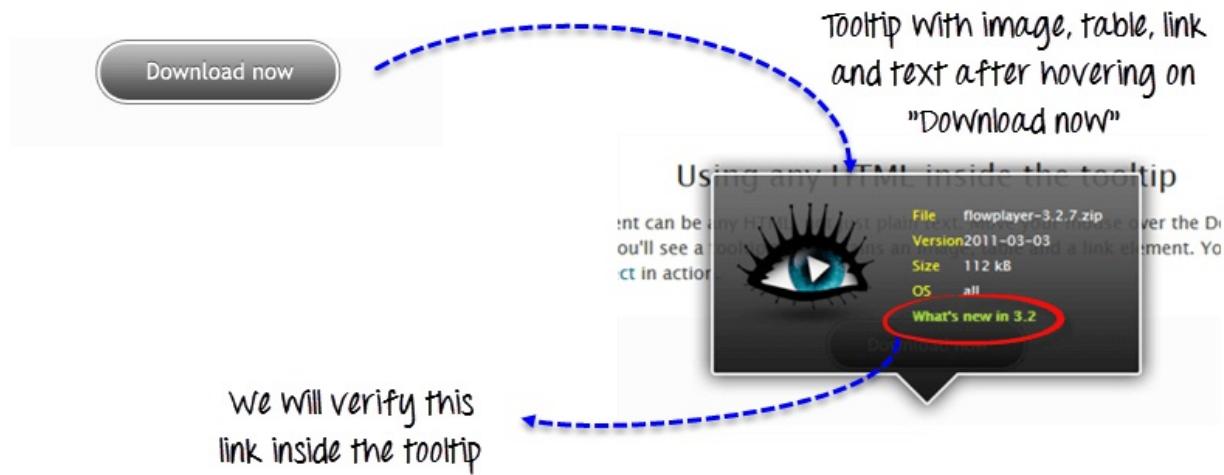
Scenario 2: JQuery Plugin:

There are a plenty of JQuery plugins available to implement the tooltips, and each one has a slightly different form of implementation.

Some plugins expect the tooltip HTML to be present all the time next to the element for which the tooltip is applicable whereas the others create a dynamic "div" tag, which appears on the fly while hovering over the element.

For our demonstration, let's consider the "jQuery Tools Tooltip" way of tooltip implementation.

Here in the URL – <http://demo.guru99.com/test/tooltip.html> you can see the demo where on mouse hovering over "Download now", we get an advanced tooltip with an image, callout background, a table and a link inside it which is clickable.



If you look at the source below, you can see that the div tag representing the tooltip is always present next to the "Download now" link's tag. But, the code inside the script tag below controls when it needs to popup.

```


<!-- trigger element. a regular workable link -->
    <a id="download_now">Download now</a>
    <!-- tooltip element -->
    <div class="tooltip" style="position: absolute; top: 311.5px; left: 352.017px; opacity: 0; display: none;">
        
        <table style="margin:0">
            <a href="/release-notes">What's new in 3.2</a> -----> This is the link that appears inside the tooltip
        </table>
    </div>
    <script> $(document).ready(function() { $("#download_now").tooltip({ effect: 'slide'}); }); </script>
</div>


```

Tool tip's div tag present next to the "Download now" link
The script tag that controls the display of tooltip's div tag

Let's try to verify just the link text in the tooltip for our demonstration here.

We will first find the WebElement corresponding to the "Download now". Then using the Interactions API, we will move to the element (mouse-hover). Next, we will find the WebElement that corresponds to the link inside the displayed tooltip and verify it against the expected

text.

```

String expectedToolTip = "What's new in 3.2";
WebElement download = driver.findElement(By.xpath(".//*[@id='download_now']")); ①

Actions builder = new Actions(driver);

builder.clickAndHold().moveToElement(download);
builder.moveToElement(download).build().perform(); ②

WebElement toolTipElement = driver.findElement(By.xpath(".//*[@class='box']/div/a")); ③

System.out.println("Tool Tip Text "+toolTipElement.getText());

Assert.assertEquals(toolTipElement.getText(), expectedToolTip); ④

```

- 1) Find the "Download now" Element
- 2) Mouse hover the element using interactions API
- 3) Find the link tag inside the Tooltip that opens
- 4) Assert the link's text against the expected one.

Here is the code

```

import org.openqa.selenium.interactions.Action;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.*;

public class JqueryToolTip {
    public static void main(String[] args) {

        String baseUrl =
"http://demo.guru99.com/test/tooltip.html";

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

        WebDriver driver = new ChromeDriver();
        String expectedTooltip = "What's new in 3.2";
        driver.get(baseUrl);

        WebElement download = driver.findElement(By.xpath(".//*[@id='download_now']"));
        Actions builder = new Actions (driver);

        builder.clickAndHold().moveToElement(download);

```

```

        builder.moveToElement(download).build().perform();

        WebElement toolTipElement =
driver.findElement(By.xpath("./*[@class='box']/div/a"));
        String actualTooltip = toolTipElement.getText();

        System.out.println("Actual Title of Tool Tip
"+actualTooltip);
        if(actualTooltip.equals(expectedTooltip)) {
            System.out.println("Test Case Passed");
        }
        driver.close();
    }
}

```

Code Explanation

1. Find the WebElement that corresponds to the element "download now" that we will mouse-hover.
2. Using the Interactions API, mouse hover on to the "Download now".
3. Assuming the tooltip is displayed, find the WebElement that corresponds to the link inside the tooltip i.e. the "a" tag.
4. Verify the link's tooltip text retrieved using the getText() against an expected value we have stored in "expectedToolTip"

Summary:

In this tutorial, you have learnt how to access Tooltips using Selenium Web driver.

- Tool Tips are implemented in different ways –
 - The basic implementation is based on HTML's "title" attribute. getAttribute(title) gets the value of the tooltip.
 - Other tool tip implementation's like JQuery, CSS tooltips

require Interactions API to create mouse hover effect

- Advanced User Interactions API
 - `moveToElement(element)` of Actions class is used to mouse hover an element.
 - `Build()` method of Actions class builds the sequence of user actions into an Action object.
 - `Perform()` of Action class executes all the sequence of user actions at once.
- In order to verify a tooltip, we have to first mouse-hover the element, then find the element that corresponds to the tool tip and get its text or other values to verify against the expected values.

Chapter 50: Flash Testing with Selenium

In Selenium Automation, if the elements are not found by the general locators **like id, class, name, etc.** then **XPath** is required to find an element on the web page to perform operation on that particular element. But in Flash testing, XPath fails to access flash object. So Flashwebdriver object is required to find flash object in any application.



Here you will see how to execute flash Testing and how to do flash testing with Selenium.

What is Flash Testing?

Flash Testing is testing type used to check the flash based video, games, movies, etc. are working as expected. In other words, testing the functionality of the flash is known as **Flash Testing**. Flash is very popular software developed by Mircomedia (now acquired by Adobe). It is used to develop games, applications, graphic based animations, movie, Mobile games, programs, etc. In

Pre-requisite-Below are the requirements in order to test the flash application

1. Flash Application.
2. Support web browser.
3. Adobe Flash player plugins.

Tools-Below are the testing tools which are useful in flash testing.

1. Selenium
2. Soap UI
3. TestComplete
4. Test Studio etc.

Selenium is a very popular tool for web testing. You can create framework across different platform and in a different language. It's an open source tool and can be downloaded from official website. It is easy to configure, use and implement.

How Flash testing is different from other element

- **Why flash object capturing is difficult? How is it resolved?**

Flash is an outdated technology. It is difficult to capture a flash object as it is different from HTML. Also, Flash is an embedded SWF file (Small Web Format). It is also difficult to access Flash object on a mobile device.

Developing flash is more difficult than developing HTML page with

the SEO (Search engine optimization) perspective because flash is not fully readable by the search engine. However, advanced technologies like HTML 5 are introduced to overcome the problems like performance and security.

- **What ways flash application is tested.**

Flash Applications are tested in two ways:

- **Manual** – You can test the Flash object by executing test cases manually as it is simple and easy to test. After bug fixation, you make sure that flash is working properly as expected and provide sign off.
- **Automation** – You use to write a script using any automation tool like Selenium, SoapUI, TestComplete, etc. and execute the script.
- **Difference between the Flash and other element.**

As mentioned above, the main difference between flash and other element is that Flash is embedded in SWF files, while other elements are embedded in HTML files. That's why HTML is easy to capture compared to flash.

How can You get flash object ID of flash movie / flash app

In any Web pages, the < object > tag is used for any embedded multimedia (like Flash, ActiveX, Video etc.). It implies "embed" within an HTML document. This tag defines a container embedded in < object /> or < embed /> tags in an HTML) for interactive content or external application. Object name is used to locate flash object on

webpages.

For instance, in below example you can see the flash movie is defined in an "embed" tag in HTML document or file.

Example:

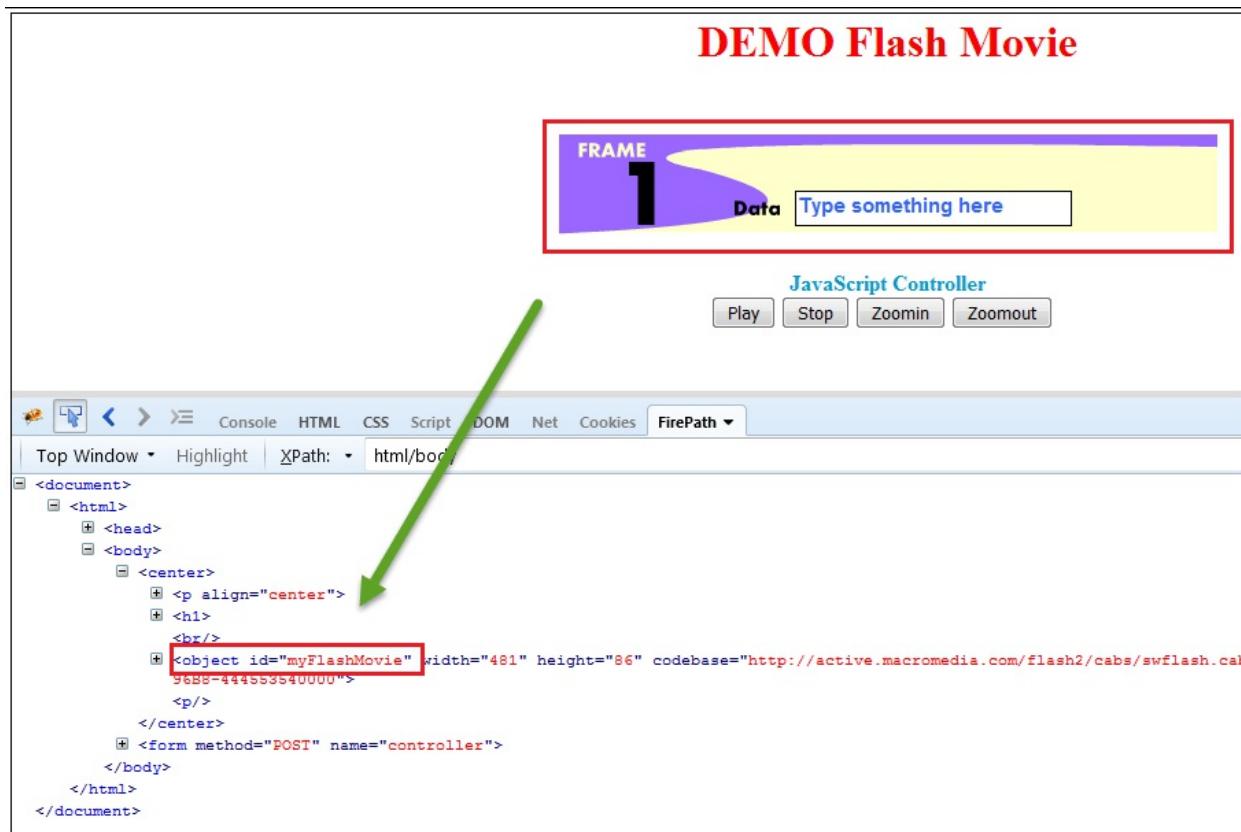
```
/* Html page*/
<html>
<body marginwidth="0" marginheight="0">
<embed width="100%" height="100%" name="plugin"
src="http://video/movie_play.swf" type="application/flash"/>
</body>
</html>
```

Using object ID to find Flash elements.

You can use flash attributes like object id to locate the flash object. And thereby you can perform operations on it as required like play, stop, etc.

As already discussed, Flash objects cannot be accessed using XPath. So in order to do any action on these objects, developer needs to assign appropriate object ID.

Below screen shows the object ID "MyFlashMovie" is assigned for the Flash:

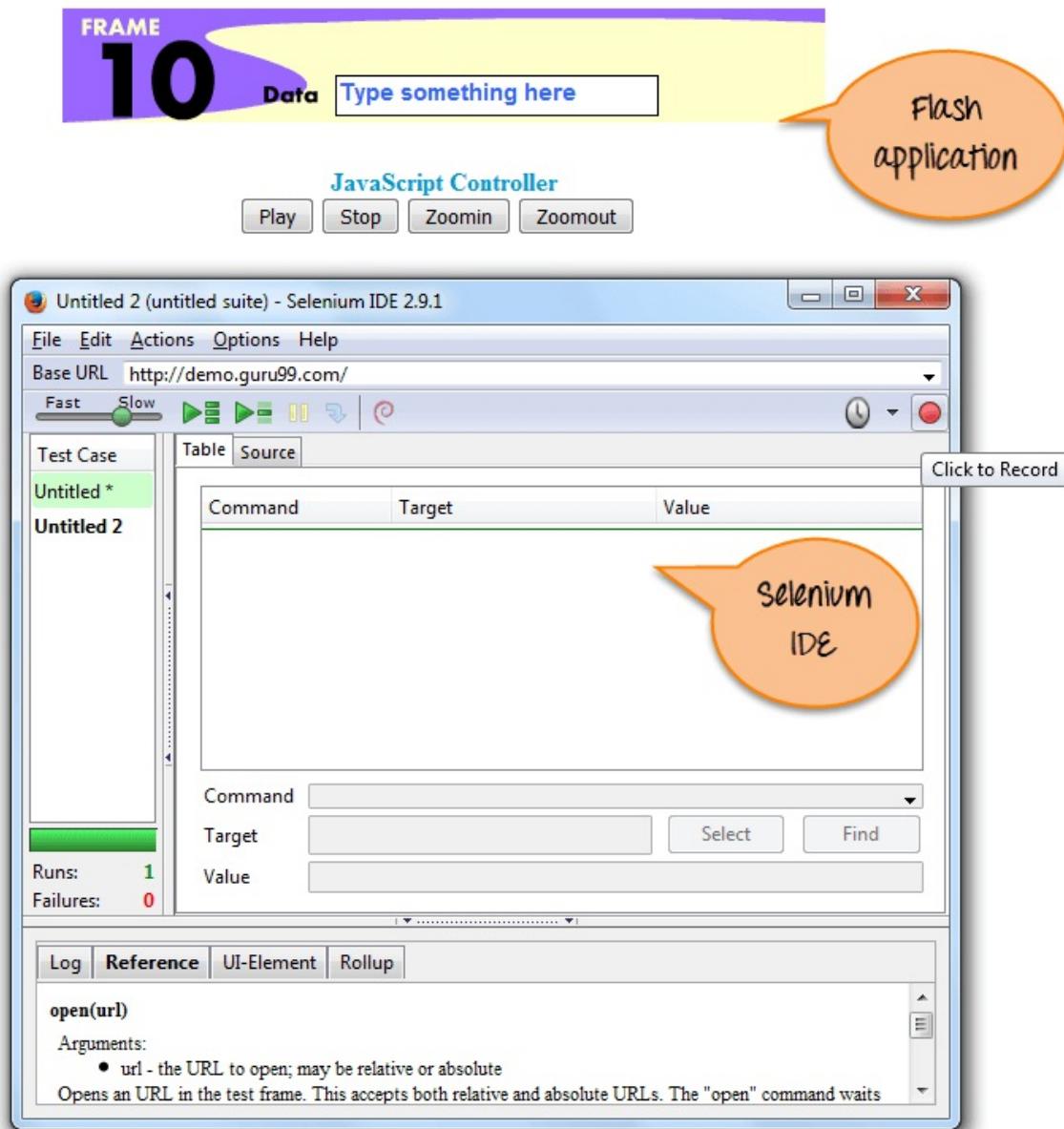


How to automate Flash using Selenium IDE recording

You can also automate the flash using Selenium IDE.

Step 1) You need to open flash application and then Selenium IDE as shown in below screen:

DEMO Flash Movie



Step 2) Now click on "record red button" on the right-hand side and start doing operation on Flash movie and then you will find the recorded script as shown below:

DEMO Flash Movie

The screenshot shows the Selenium IDE interface with the following details:

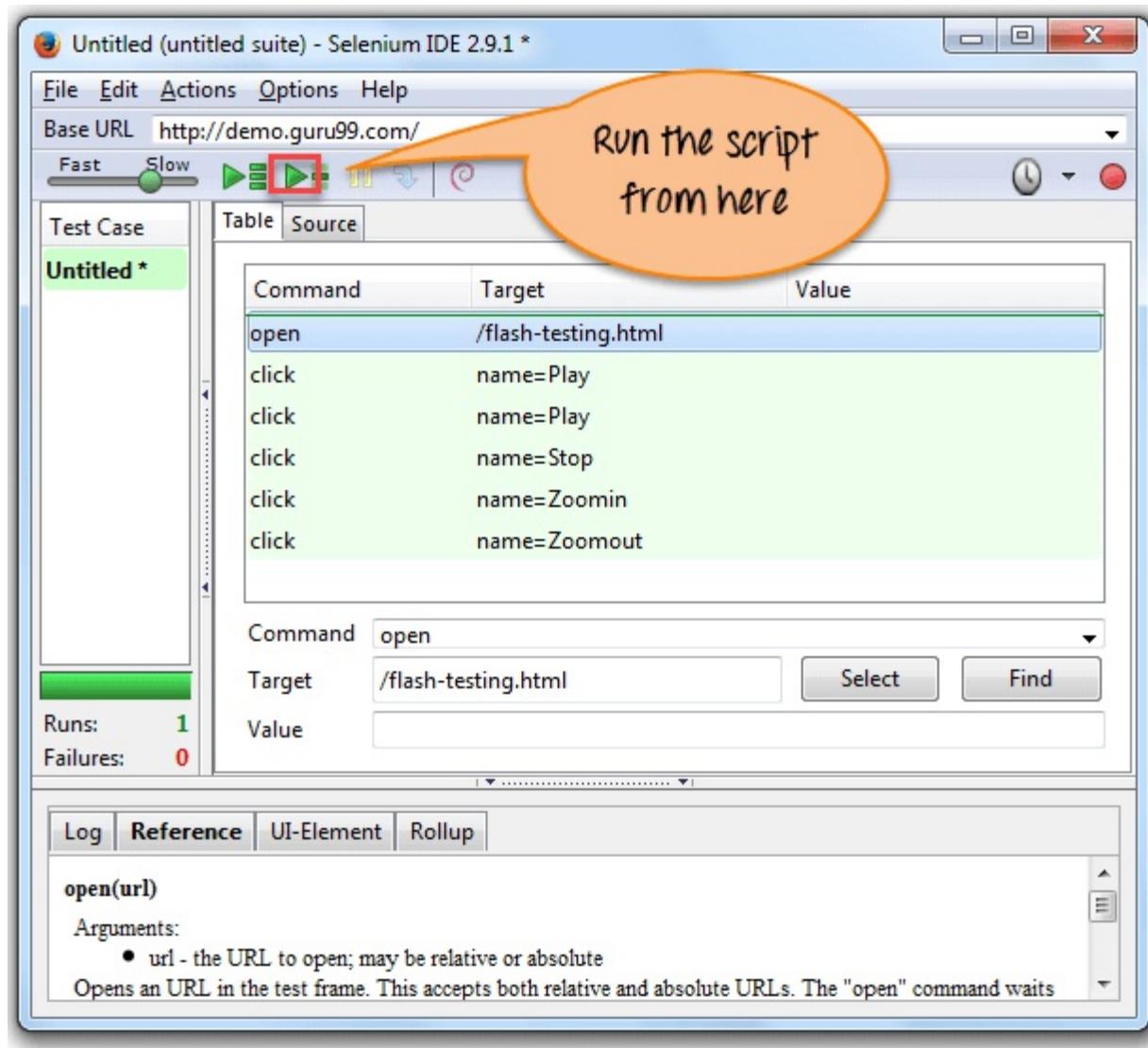
- Top Frame Content:** A Flash movie frame titled "FRAME 10". It contains a text input field with the placeholder "Type something here". Below the frame are buttons for "Play", "Stop", "Zoomin", and "Zoomout".
- Selenium IDE Window:**
 - Title Bar:** Untitled (untitled suite) - Selenium IDE 2.9.1 *
 - Menu Bar:** File, Edit, Actions, Options, Help
 - Toolbar:** Includes a speed slider (Fast to Slow), several execution buttons (green arrows, etc.), and a red "Run" button.
 - Test Case List:** Untitled * (highlighted in green).
 - Table View:** Shows a recorded script in a table format:

Command	Target	Value
open	/flash-testing.html	
click	name=Play	
click	name=Play	
click	name=Stop	
click	name=Zoomin	
click	name=Zoomout	
 - Bottom Panel:** Shows the "open" command details with arguments: url (the URL to open). It also includes a log section with the command "open(url)" and its description: "Opens an URL in the test frame. This accepts both relative and absolute URLs. The 'open' command waits".

A large orange speech bubble with the text "Recorded Script created" is overlaid on the right side of the table view.

After recording, if the user wants to execute the script then they can click on the "green run button" as shown in below screen. The

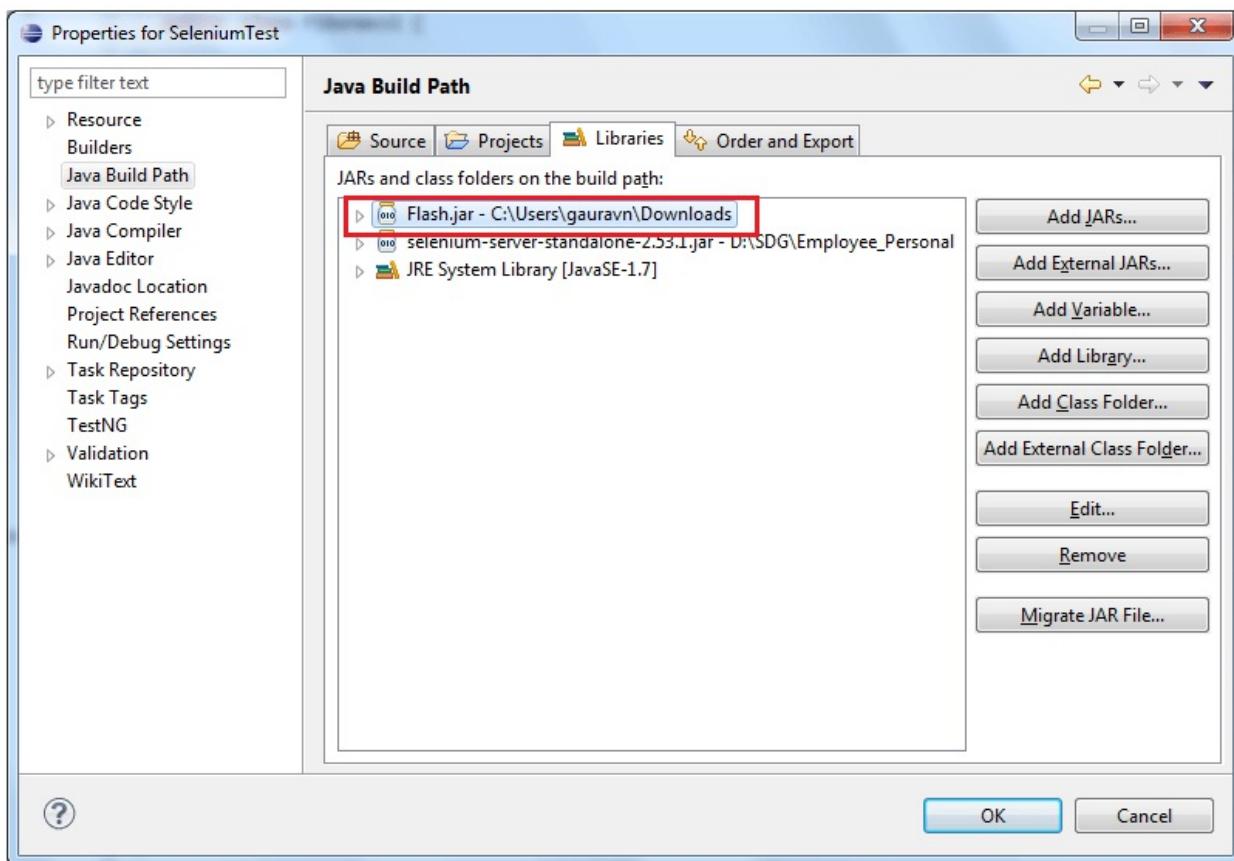
Selenium IDE will execute the script step by step.



How to automate Flash using Selenium Webdriver.

You can also automate the flash using Selenium web driver through the Flashwebdriver object and then call a method to operate flash object. You need to download flashwebdriver jar files:

Step 1) After download, add the jar file in your project as shown in below screen.



Step 2) Under flash jar file there is a separate flashobjectwebdriver class. Implement the flashWebdriver "myFlashmovie" in your selenium script as shown below in screen.

```
<br/>
+ <object id="myFlashMovie" width="481" height="86" codebase="http://a
96B8-444553540000">
<p>
</center>
+ <form method="POST" name="controller">

// Open firefox browser
FirefoxDriver driver = new FirefoxDriver();

// Maximize browser
driver.manage().window().maximize();

driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);

// Under Flash jar file there is separate FlashObjectWebDriver class
FlashObjectWebDriver flashApp = new FlashObjectWebDriver(driver, "myFlashMovie");

// Pass the URL of video
driver.get("http://demo.guru99.com/flash-testing.html");
```

After adding web driver class "MyFlashMovie," you can access the Flash object.

When to automate flash testing

Usually, you need to Automate Flash testing when the flash object is not easily accessible. This result in testing gets abort and hence fails to test Flash object.

Creating selenium script for Flash testing.

Step 1) You use the "Guru99" flash movie to test the flash scenario.

<http://demo.guru99.com/test/flash-testing.html>



Step 2) Write a script in Selenium eclipse and execute it. Below code when executed will do following things

- Open the Firefox browser,
- Launch the guru99 flash site,
- Play the flash movie and
- Then stop the movie.

```
import org.openqa.selenium.firefox.FirefoxDriver;
import Flash.FlashObjectWebDriver;
public class Flash {
    public static void main(String[] args) throws
InterruptedException {
        // Open firefox browser
        FirefoxDriver driver = new FirefoxDriver();
        // Maximize browser
        driver.manage().window().maximize();
        // Under Flash jar file there is separate
FlashObjectWebDriver class
        FlashObjectWebDriver flashApp = new
FlashObjectWebDriver(driver, "myFlashMovie");
        // Pass the URL of video
        driver.get("http://demo.guru99.com/test/flash-
testing.html");
        Thread.sleep(5000);
        flashApp.callFlashObject("Play");
        Thread.sleep(5000);
        flashApp.callFlashObject("StopPlay");
```

```

        Thread.sleep(5000);

flashApp.callFlashObject("SetVariable", "/:message", "Flash
testing using selenium Webdriver");

System.out.println(flashApp.callFlashObject("GetVariable", "/:mes
sage"));
    }
}

```

Step 3) : Execute the above script.

Output : On execution of the above script the flash movie starts to play and Stop etc.

Challenges in Flash Testing

- Automating flash app is a challenge. To automate flash app, You can use FlexMonkium which is an add-on for Selenium IDE.
- You might face issue to enable record / playback Flex apps using Selenium-Flexmonkium integration. Solution is that user needs to install and integrate Flex monkium to selenium IDE carefully. Proper installation will enable record to automate flash apps.

Summary:

- In Flash testing, You need to check the flash video, games, movies, etc. are working as expectation or not.
- You use flash attributes like object id to locate the flash object . And thereby you can perform operations on it as required like play, stop, etc.
- Main difference between flash and other element is that Flash is embedded in SWF files, while other elements are embedded in HTML files
- You need to Automate Flash testing normally when the flash

object is not easily accessible.

- Tools useful in flash testing are
 1. Selenium
 2. Soap UI
 3. TestComplete
 4. Test Studio etc.
- Automating flash app is a challenge. To automate flash app, you can use FlexMonkium which is an add-on for Selenium IDE.

Chapter 51: How to Find Broken links using Selenium Webdriver

What are Broken Links?

Broken links are links or URLs that are not reachable. They may be down or not functioning due to some server error

An URL will always have a status with 2xx which is valid. There are different HTTP status codes which are having different purposes. For an invalid request, HTTP status is 4xx and 5xx.

4xx class of status code is mainly for client side error, and 5xx class of status codes is mainly for the server response error.

We will most likely be unable to confirm if that link is working or not until we click and confirm it.

Why should you check Broken links?

You should always make sure that there are no broken links on the site because the user should not land into an error page.

The error happens if the rules are not updated correctly, or the requested resources are not existing at the server.

Manual checking of links is a tedious task, because each webpage may

have a large number of links & manual process has to be repeated for all pages.

An Automation script using Selenium that will automate the process is a more apt solution.

How to check the Broken Links and images

For checking the broken links, you will need to do the following steps.

1. Collect all the links in the web page based on `<a>` tag.
2. Send HTTP request for the link and read HTTP response code.
3. Find out whether the link is valid or broken based on HTTP response code.
4. Repeat this for all the links captured.

Code to Find the Broken links on a webpage

Below is the web driver code which tests our use case:

```
package automationPractice;

import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Iterator;
import java.util.List;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
```

```
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class BrokenLinks {

    private static WebDriver driver = null;

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        String homePage = "http://www.zlti.com";
        String url = "";
        HttpURLConnection huc = null;
        int respCode = 200;

        driver = new ChromeDriver();

        driver.manage().window().maximize();

        driver.get(homePage);

        List<WebElement> links =
driver.findElements(By.tagName("a"));

        Iterator<WebElement> it = links.iterator();

        while(it.hasNext()){

            url = it.next().getAttribute("href");

            System.out.println(url);

            if(url == null || url.isEmpty()){
System.out.println("URL is either not configured for anchor tag
or it is empty");
                continue;
            }

            if(!url.startsWith(homePage)){
                System.out.println("URL belongs to another
domain, skipping it.");
                continue;
            }
        }
    }
}
```

```
try {
    huc = (HttpURLConnection)(new
URL(url).openConnection());

    huc.setRequestMethod("HEAD");

    huc.connect();

    respCode = huc.getResponseCode();

    if(respCode >= 400){
        System.out.println(url+" is a broken link");
    }
    else{
        System.out.println(url+" is a valid link");
    }

} catch (MalformedURLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}

driver.quit();

}
```

Explaining the code

Step 1: Import Packages

Import below package in addition to default packages:

```
import java.net.HttpURLConnection;
```

Using the methods in this package, we can send HTTP requests and capture HTTP response codes from the response.

Step 2: Collect all links in web page

Identify all links in a webpage and store them in List.

```
List<WebElement> links = driver.findElements(By.tagName("a"));
```

Obtain Iterator to traverse through the List.

```
Iterator<WebElement> it = links.iterator();
```

Step 3: Identifying and Validating URL

In this part, we will check if URL belongs to Third party domain or whether URL is empty/null.

Get href of anchor tag and store it in url variable.

```
url = it.next().getAttribute("href");
```

Check if URL is null or Empty and skip the remaining steps if the condition is satisfied.

```
if(url == null || url.isEmpty()){
    System.out.println("URL is either not configured
for anchor tag or it is empty");
    continue;
}
```

Check whether URL belongs to a main domain or third party. Skip the remaining steps if it belongs to third party domain.

```
if(!url.startsWith(homePage)){
    System.out.println("URL belongs to another domain,
```

```

skipping it.");
        continue;
}

```

Step 4: Send http request

HttpURLConnection class has methods to send HTTP request and capture HTTP response code. So, output of openConnection() method (URLConnection) is type casted to HttpURLConnection.

```
huc = (HttpURLConnection)(new URL(url).openConnection());
```

We can set Request type as "HEAD" instead of "GET". So that only headers are returned and not document body.

```
huc.setRequestMethod("HEAD");
```

On invoking connect() method, actual connection to url is established and the request is sent.

```
huc.connect();
```

Step 5: Validating Links

Using getResponseCode() method we can get response code for the request

```
respCode = huc.getResponseCode();
```

Based on response code we will try to check link status.

```

if(respCode >= 400){
    System.out.println(url+" is a broken link");
}
else{
    System.out.println(url+" is a valid link");
}

```

{}

Thus, we can obtain all links from web page and print whether links are valid or broken.

Hope this tutorial helps you in checking Broken links using selenium.

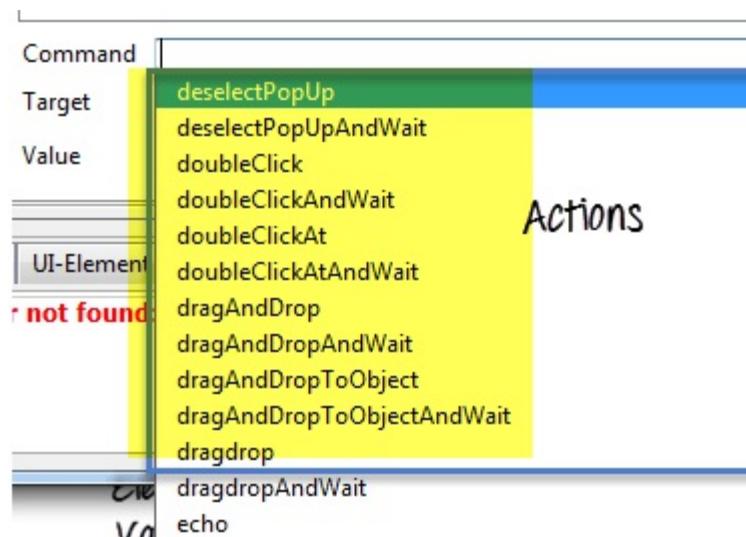
TroubleShooting

In an isolated case, the first link accessed by the code could be the "Home" Link. In such case, driver.navigate.back() action will show a blank page as the 1st action is opening a browser. The driver will not be able to find all other links in a blank browser. So IDE will throw an exception and rest of the code will not execute. This can be easily handled using an If loop.

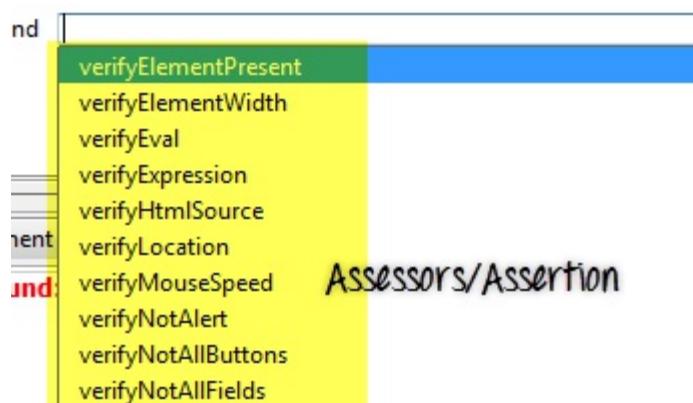
Chapter 52: Selenium Core Extensions

To understand extensions, lets first understand the three pillars of selenium IDE

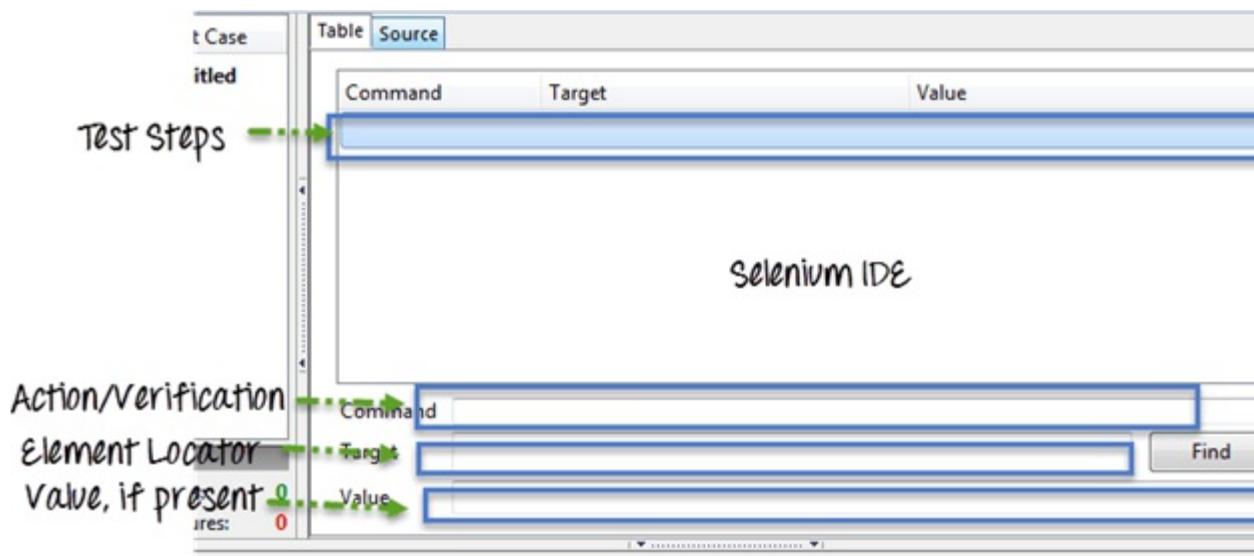
1. Action: What operation you are performing on UI Screen



2. Assessors/Assertion: What verification you do on data you get from UI



3. Locator Strategy: How can we find the element in UI.



Now, Selenium IDE has a very mature library with plenty of Actions, Assertion/Assessors and Locator Strategies.

But sometimes we need to add some more functionality to it for our project requirements. In that situation, we can expand this library by adding our custom extensions. These custom extensions are called 'User Extension'.

For example, we need an Action which can convert the text to upper case before filling it in a web element. You cannot find this Action in the default Action library. In such case you can create your own 'User Extension'. In this tutorial, we will learn how to create user extension to convert Text to Upper Case

Requirement to create Selenium user extension:

To create user extension for Selenium IDE, we need to know the basic

concept of JavaScript and Java Script prototype object concept.

Prototype concept require

```
// The "inDocument" is a the document you are searching.
PageBot.prototype.locateElementByValueRepeated = function(text, inDocument) {
    // Create the text to search for
```

To create your user extension, you need to create Java script methods and add them to the selenium object prototype and PageBot object prototype.

How Selenium IDE recognizes User Extension?

After adding of User Extension to Selenium IDE when we start Selenium IDE, all of these extensions in javascript prototype get loaded, and Selenium IDE recognizes them by their name.

How to Create User Extension

Step 1) Action- all actions are started by "do", i.e. if the action is for upper case text than its name will be **doTextUpperCase**. When we add this action method in Selenium IDE, Selenium IDE will itself create a wait method for this action. So in this case when we create **doTextUpperCase** action, Selenium IDE will create a corresponding wait function as **TextUpperCaseAndWait**. It can accept two parameters

Example: Upper Case Text Action

```
Selenium.prototype.doTextUpperCase = function(locator, text) {
    // Here findElement is itself capable to handle all type of
```

locator(xpath,css,name,id,className), We just need to pass the locator text

```

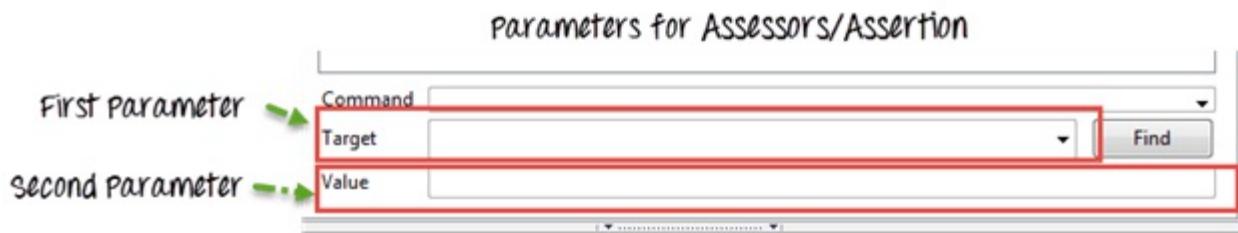
    var element = this.page().findElement(locator);

    // Create the text to type
    text = text.toUpperCase();

    // Replace the element text with the new text
    this.page().replaceText(element, text);
};
```

Step 2) Assessors/Assertion- All assessors registered in selenium object prototype will be prefixed

by "get" or "is" Ex. getValueFromCompoundTable ,
isValueFromCompoundTable .It can accept two parameters, one for target and other for value field in test case.



For each Assessor, there will be corresponding verification functions prefixed by "verify," "assert" and the wait function prefix by "waitFor"

Example: For Upper Case Text assessors

```

Selenium.prototype.assertTextUpperCase = function(locator, text)
{
    // All locator-strategies are automatically handled by
    "findElement"
    var element = this.page().findElement(locator);

    // Create the text to verify
    text = text.toUpperCase();
```

```

    // Get the actual element value
    var actualValue = element.value;

    // Make sure the actual value matches the expected
    Assert.matches(expectedValue, actualValue);
};

Selenium.prototype.isTextEqual = function(locator, text) {
    return this.getText(locator).value === text;
};

Selenium.prototype.getTextValue = function(locator, text) {
    return this.getText(locator).value;
};

```

Step 3) Locator strategy- If we wish to create our own function to locate an element then

we need to extend PageBot prototype with a function with prefix "locateElementBy."

It will take two parameters, first will be the locator string and second will be the document

where it needs to be searched.

Example: For Upper Case Text Locator

```

// The "inDocument" is a document you are searching.
PageBot.prototype.locateElementByUpperCase = function(text,
inDocument) {
    // Create the text to search for
    var expectedValue = text.toUpperCase();

    // Loop through all elements, looking for ones that have
    // a value === our expected value
    var allElements = inDocument.getElementsByTagName("*");
// This star '*' is a kind of regular expression it will go
through every element (in HTML DOM every element surely have a

```

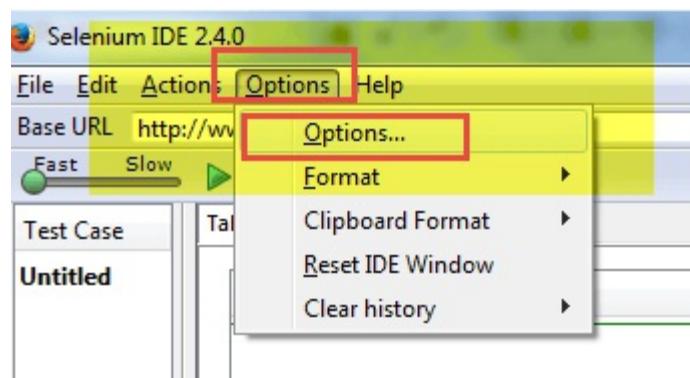
tag name like<body>, <a>, <h1>, <table>, <tr>, <td> etc.). Here our motive is to find an element which matched with the Upper Case text we have passed so we will search it with all elements and when we get match we will have the correct web element.

```
for (var i = 0; i < allElements.length; i++) {  
    var testElement = allElements[i];  
    if (testElement.innerHTML && testElement.innerHTML ===  
expectedValue) {  
        return testElement;  
    }  
}  
return null;  
};
```

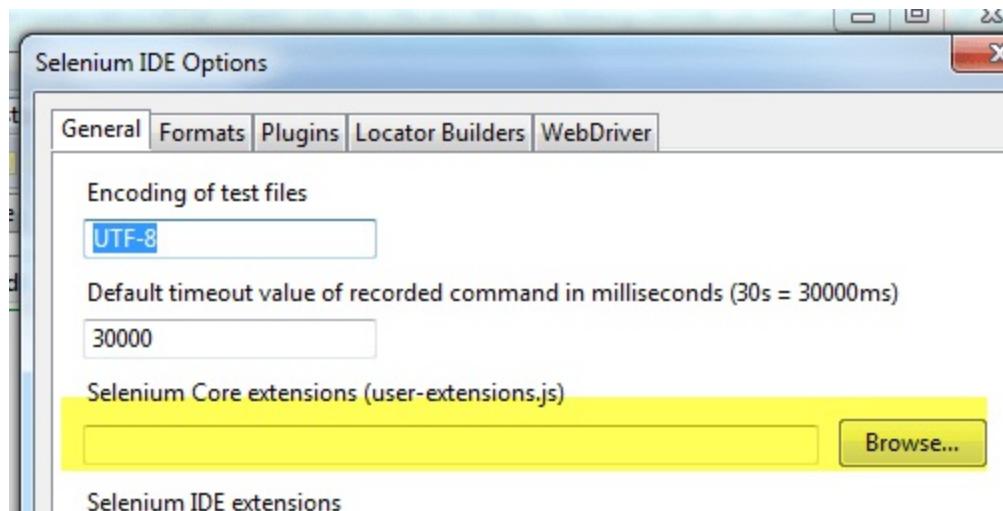
How to use newly created core extension?

1. Go to Selenium IDE

Click on Options -> Options...



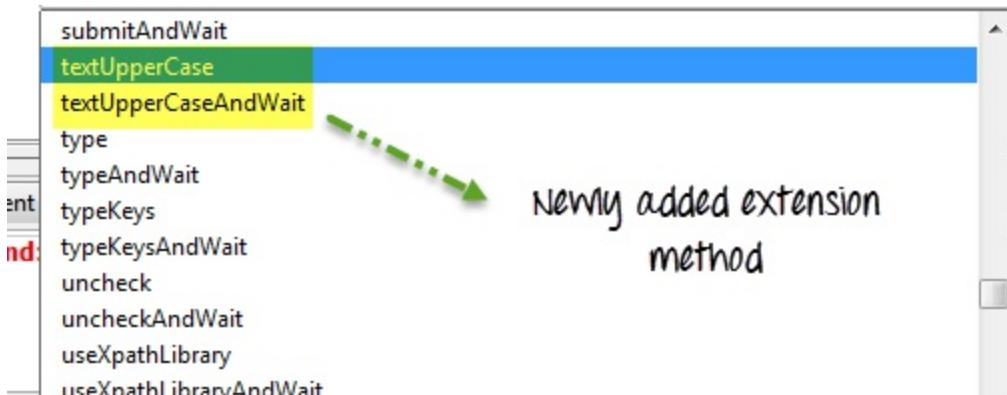
2. In General section select the location of the newly created Selenium Core Extension



3. Click OK and restart Selenium IDE



4. You will find the extension in the command list



Here is a list of popular extensions/plug-in used in Selenium IDE

Name	Purpose

Favorites	To mark a test suite as favorite and execute them in one click
Flex Pilot X	For Flex based automation
FlexMonkium	For Adobe Flex based recording and playback Testing in Selenium IDE
File Logging	For saving logs in a file
Flow Control	To control test execution flow
Highlight Elements	To highlight a web control
Implicit Wait	To wait for an element for certain time limit
ScreenShot on Fail	Take a screenshot on failure
Test Results	Save Test Case result for a test suite in one click

You can get these all and many more from SeleniumHQ official site's download section

<http://docs.seleniumhq.org/download/>

Summary:

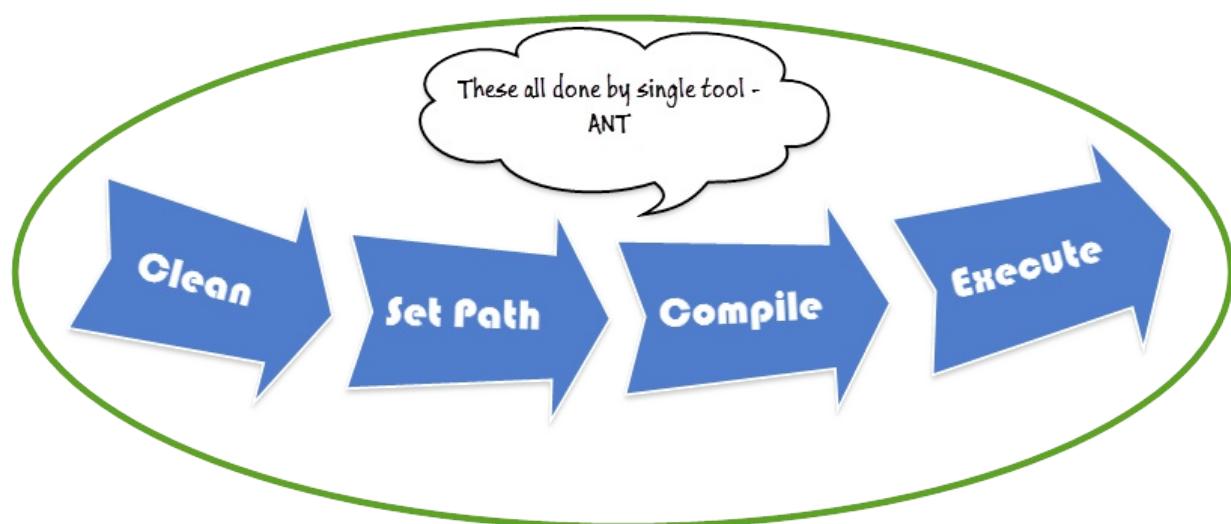
- There is three part of Selenium IDE, Action, Assessors/Assertion, Locator strategy.
- User extension is created, when Selenium IDE is not fulfilling the current requirement.
- To create user extension it is required to add javascript to selenium's object prototype.
- After creation of extension, it is required to add it in Selenium IDE and restart IDE.

Chapter 53: Using Apache Ant with Selenium

What is Apache Ant?

While creating a complete software product, one needs to take care different third party API, their classpath, cleaning previous executable binary files, compiling our source code, execution of source code, creation of reports and deployment code base etc. If these tasks are done one by one manually, it will take an enormous time, and the process will be prone to errors.

Here comes the importance of a build tool like Ant. It stores, executes and automates all process in a sequential order mentioned in Ant's configuration file (usually build.xml).



Benefit of Ant build

1. Ant creates the application life cycle i.e. clean, compile, set dependency, execute, report, etc.
2. Third party API dependency can be set by Ant i.e. other Jar file's class path is set by Ant build file.
3. A complete application is created for End to End delivery and deployment.
4. It is a simple build tool where all configurations can be done using XML file and which can be executed from the command line.
5. It makes your code clean as configuration is separate from actual application logic.

How to install Ant

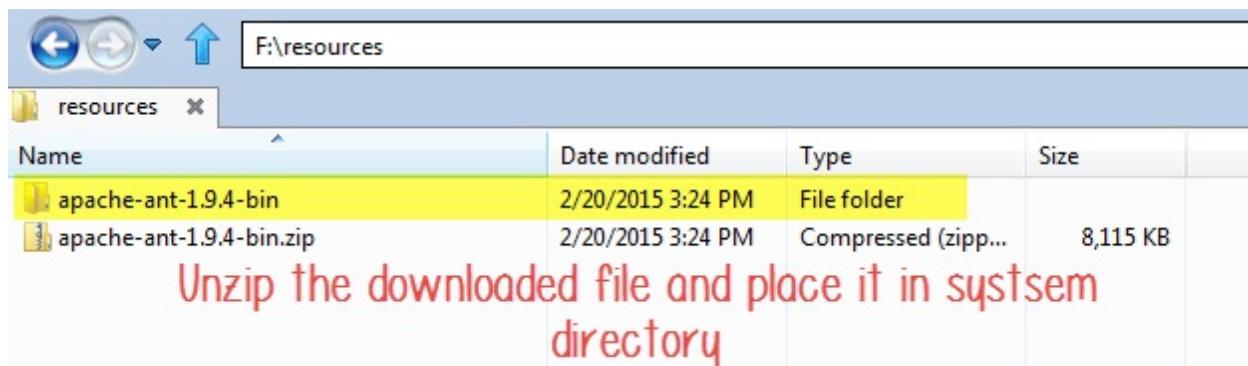
Steps to install Ant in Windows is as follows

Step 1) Go to <http://ant.apache.org/bindownload.cgi> Download .zip file from apache-ant-1.9.4-bin.zip

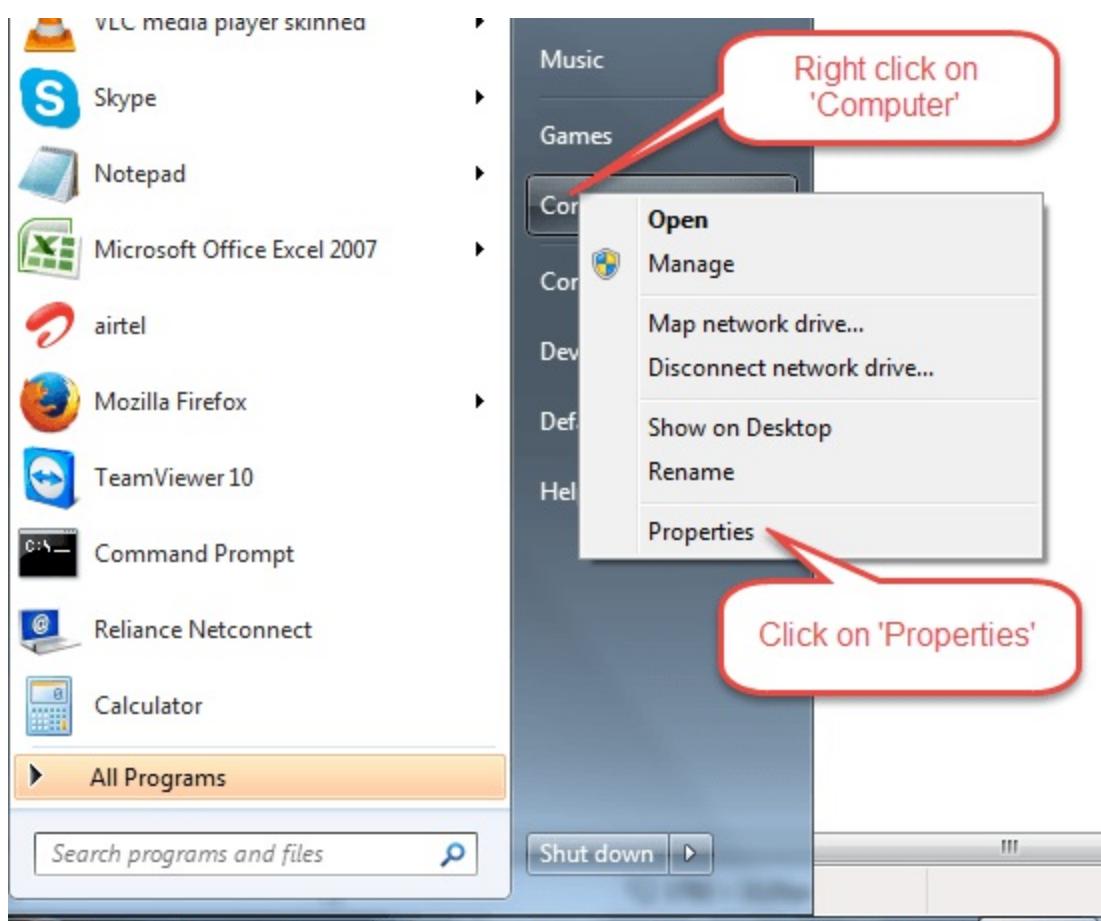
- .zip archive: [apache-ant-1.9.4-bin.zip \[PGP\] \[SHA1\] \[SHA512\] \[MD5\]](#)
- .tar.gz archive: [apache-ant-1.9.4-bin.tar.gz \[PGP\] \[SHA1\] \[SHA512\] \[MD5\]](#)
- .tar.bz2 archive: [apache-ant-1.9.4-bin.tar.bz2 \[PGP\] \[SHA1\] \[SHA512\] \[MD5\]](#)

Click here to download ANT

Step 2) Unzip the folder and go to and copy path to the root of unzipped folder



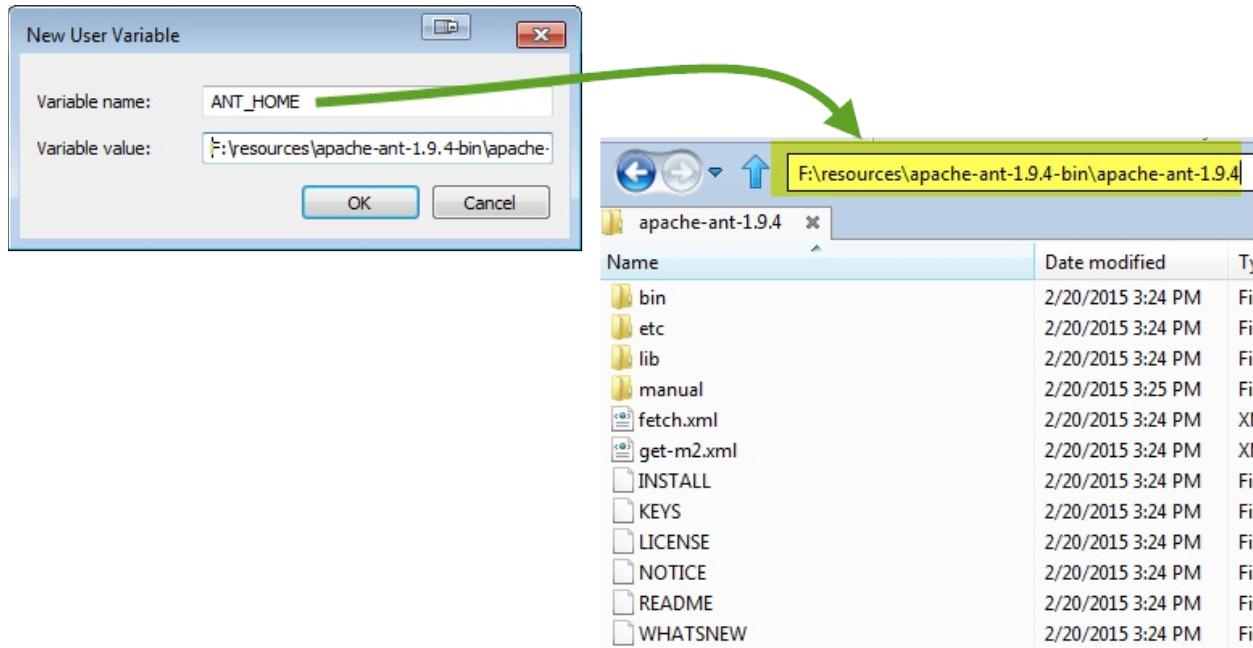
Step 3) Go to Start -> Computer -> right click here and select 'Properties' then click on Advanced System Settings



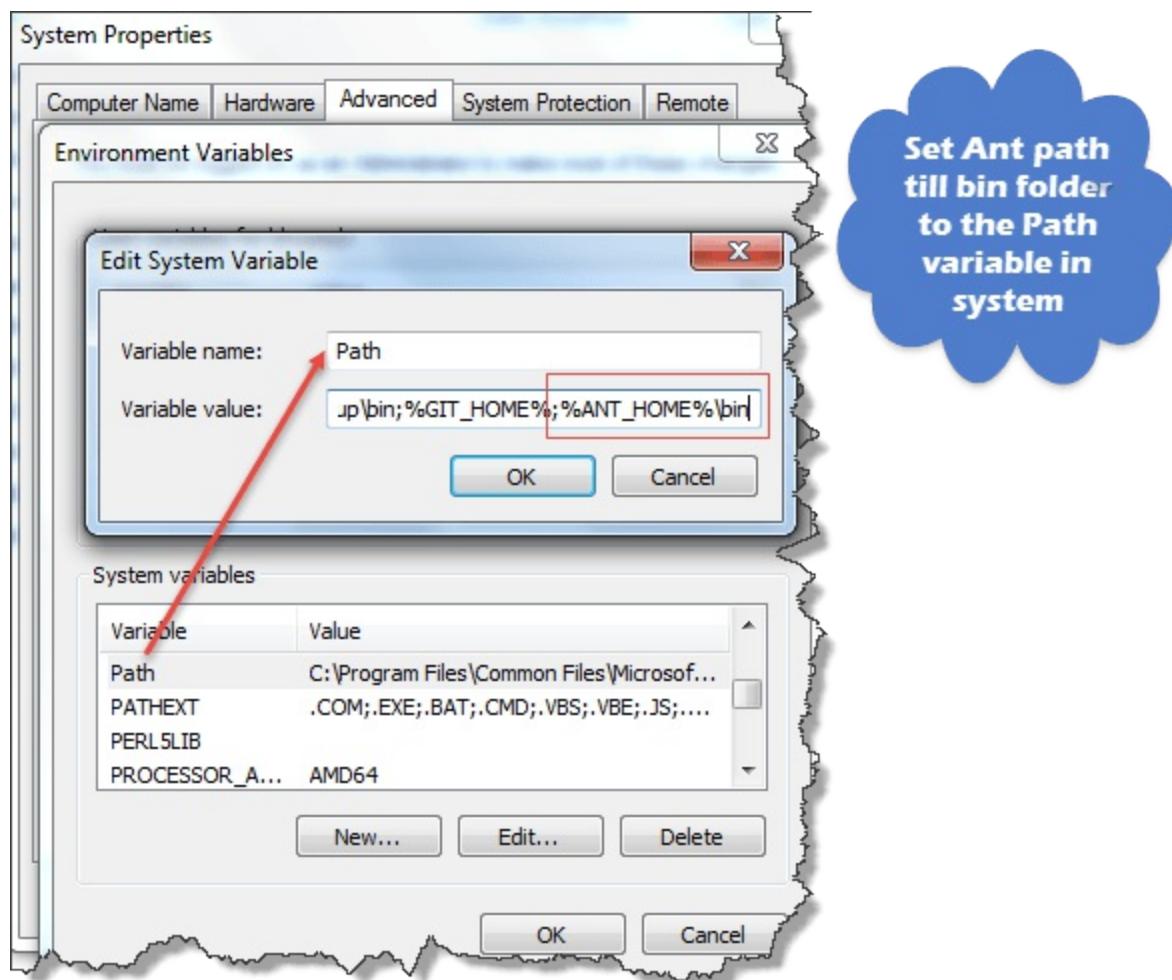
Step 4) A new window opens. Click on 'Environment Variable...' button.



Step 5) Click 'New...' button and set variable name as 'ANT_HOME' and variable value as the root path to unzipped folder and click OK.



Step 6) now select 'Path' variable from the list and click 'Edit' and append; %ANT_HOME%\bin.



Restart system one time and you are ready to use Ant build tool now.

Step 7) To check the version of your Ant using command line:

Ant –version

```
C:\>ant -version
Apache Ant(TM) version 1.9.2 compiled on...
C:\>
```

This command will show Ant version

Understanding Build.xml

Build.xml is the most important component of Ant build tool. For a Java project, all cleaning, setup, compilation and deployment related task are mentioned in this file in XML format. When we execute this XML file using command line or any IDE plugin, all instructions written into this file will get executed in 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 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}/YT0peration"/>
<property name="src.dir" value="${basedir}/src"/>
```

- Target tags used as steps that will execute in sequential order. 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 files logically which are in the common location

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

- pathelement tag will set the path to the root of common location

where all files are stored

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

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

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

- fileset tag used to set classpath for different third party jar in our project

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

- Echo tag is used to print text on the console

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

- Delete tag will clean data from given folder

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

- mkdir tag will create a new directory

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

- 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="test.Main"/>
</manifest>
```

- 'depends' attribute used to make one target to depend on another target

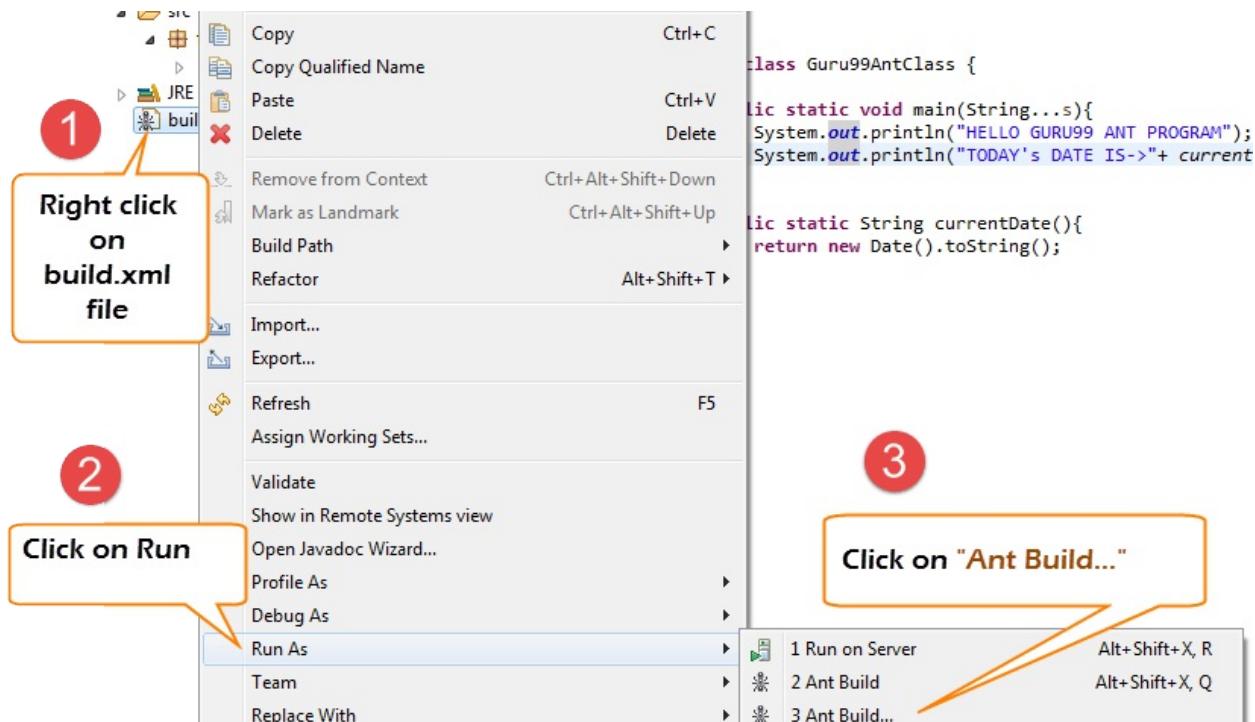
```
<target name="run" depends="compile">
```

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

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

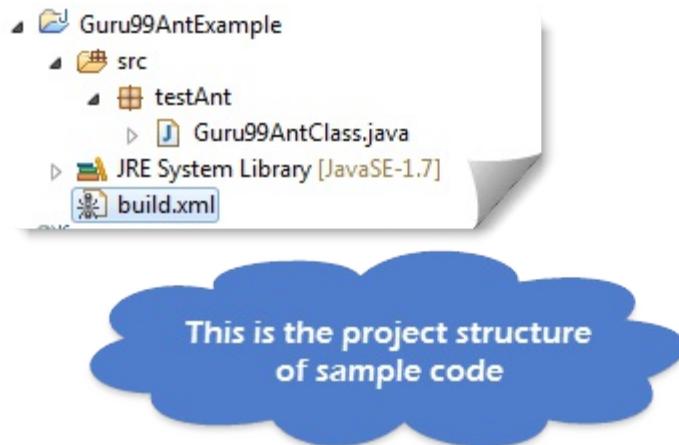
Run Ant using Eclipse plugin

To run Ant from eclipse go to build.xml file -> right click on file -> Run as... -> click Build file



Example:

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



Here in this example we have 4 targets

1. Set class path for external jars,
2. Clean previously complied code
3. Compile existing java code
4. Run the code

Guru99AntClass.class

```
package testAnt;
import java.util.Date;

public class Guru99AntClass {
    public static void main(String...s){
        System.out.println("HELLO GURU99 ANT PROGRAM");
        System.out.println("TODAY's DATE IS->" +
currentDate());
    }
    public static String currentDate(){
        return new Date().toString();
    }
}
```

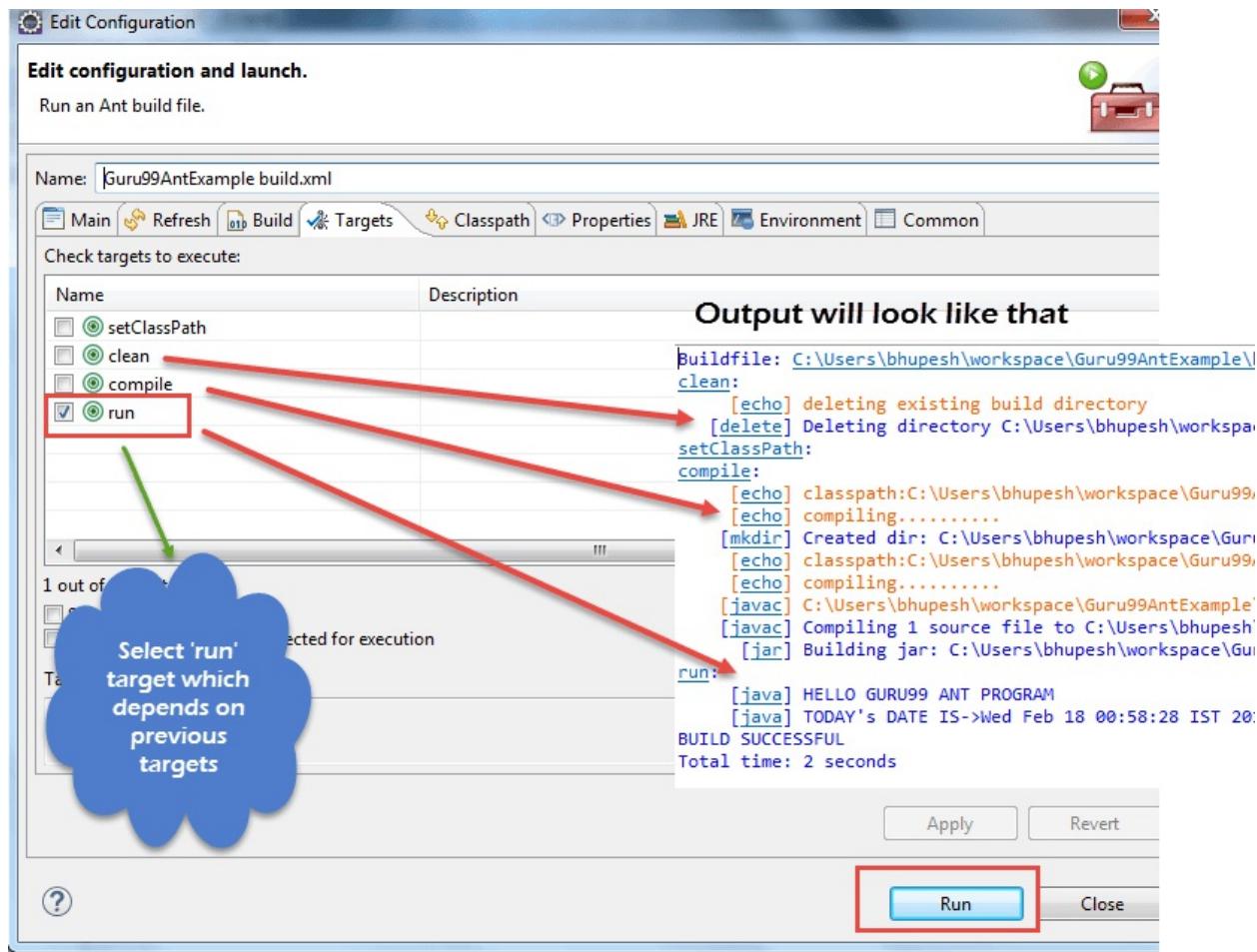
```
    }  
}
```

Build.xml

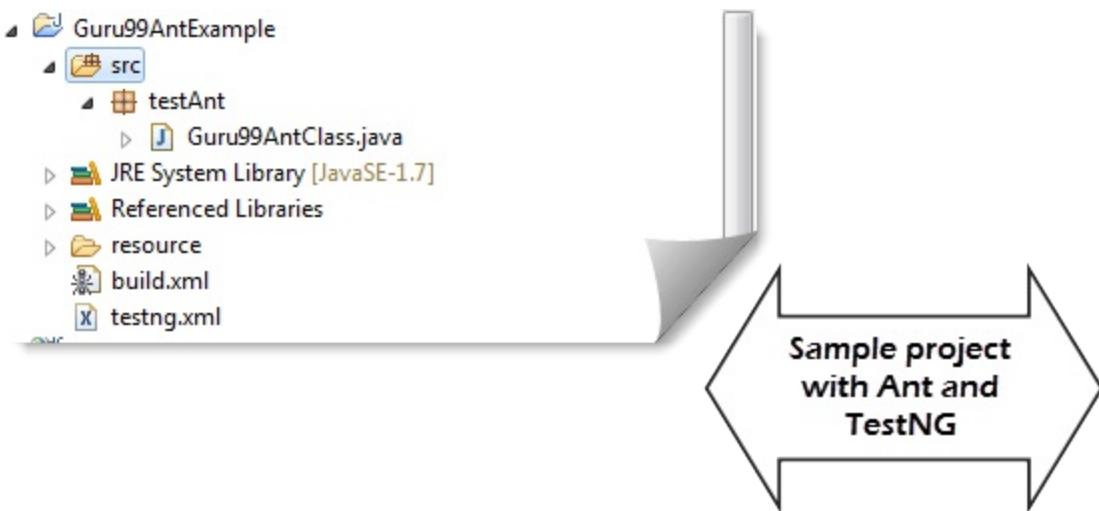
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<!--Project tag used to mention the project name, and basedir  
attribute will be the root directory of the application-->  
  
<project name="YTMonetize" basedir=".">  
    <!--Property tags will be used as variables in build.xml  
file to use in further steps-->  
  
    <property name="build.dir" value="${basedir}/build"/>  
    <property name="external.jars" value=".\\resources"/>  
        <property name="ytoperation.dir"  
value="${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="${ytoperation.dir}/YT0peration.jar" basedir="${build.dir}">
    <!--manifest tag will set your main class
for execution-->
    <manifest>

<attribute name="Main-Class" value="testAnt.Guru99AntClass"/>
</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}/YT0peration.jar" fork="true"/>
</target>
</project>
```



How to Execute TestNG code using Ant



Here we will create a class with Testng methods and set class path for Testing in build.xml.

Now to execute testng method we will create another testng.xml file and call this file from build.xml file.

Step 1) We create a "Guru99AntClass.class**" in package **testAnt****

Guru99AntClass.class

```
package testAnt;
import java.util.Date;
import org.testng.annotations.Test;
public class Guru99AntClass {
    @Test
        public void Guru99AntTestNGMethod(){
            System.out.println("HELLO GURU99 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="GURU99TestNGAnt">
        <classes>
            <class name="testAnt.Guru99AntClass">
        </class>
    </classes>
</test>
</suite>
```

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

```
<target name="runGuru99TestNGAnt" 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

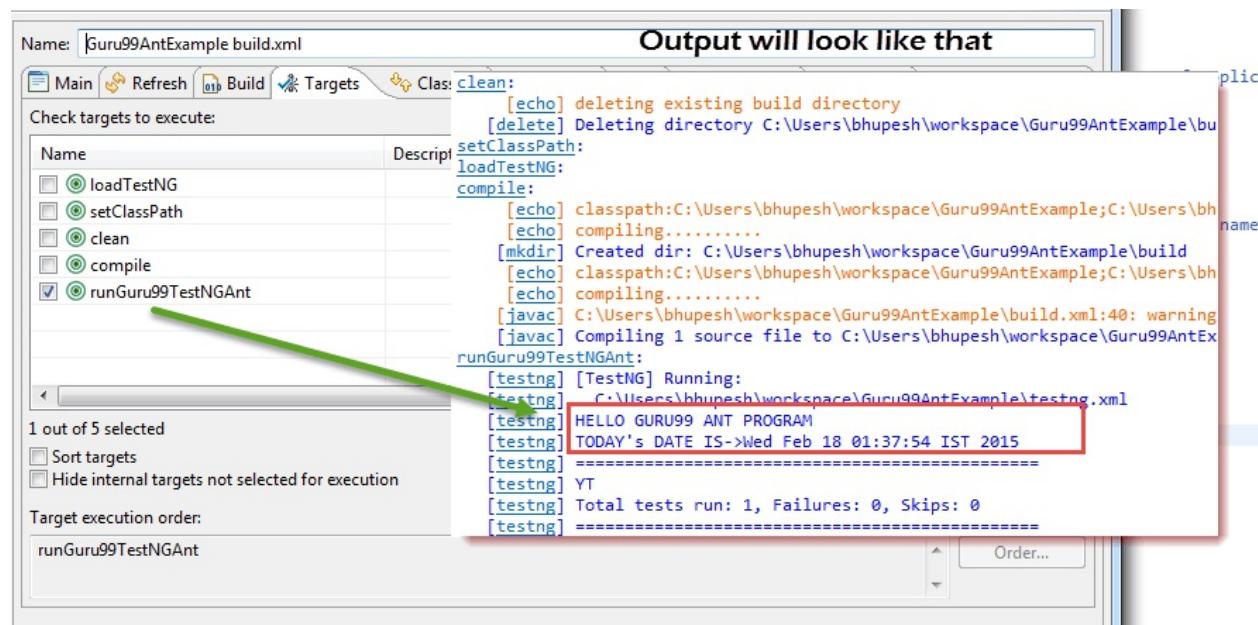
```

<?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="${src.dir}">
    <classpath refid="classpath_jars"/>
    </javac>
</target>
<target name="runGuru99TestNGAnt" depends="compile">
    <!-- testing tag will be used to execute testng
code using corresponding testng.xml file -->
    <testng
classpath="${test.classpath};${build.dir}">
    <xmfileset
dir="${basedir}" includes="testng.xml"/>
    </testng>
</target>
</project>
```

Step 6) Output



The screenshot shows the Eclipse IDE interface for an Ant build. The 'Targets' view on the left lists several targets: 'loadTestNG', 'setClassPath', 'clean', 'compile', and 'runGuru99TestNGAnt'. The 'runGuru99TestNGAnt' target is checked. The 'Output will look like that' view on the right displays the command-line output of running this target:

```

[echo] deleting existing build directory
[delete] Deleting directory C:\Users\bhupesh\workspace\Guru99AntExample\bu
[setClassPath]
[loadTestNG]
[compile]
[echo] classpath:C:\Users\bhupesh\workspace\Guru99AntExample;C:\Users\bh
[echo] compiling.....
[mkdir] Created dir: C:\Users\bhupesh\workspace\Guru99AntExample\build
[echo] classpath:C:\Users\bhupesh\workspace\Guru99AntExample;C:\Users\bh
[echo] compiling.....
[javac] C:\Users\bhupesh\workspace\Guru99AntExample\build.xml:40: warning
[javac] Compiling 1 source file to C:\Users\bhupesh\workspace\Guru99AntEx
[runGuru99TestNGAnt]
[testing] [TestNG] Running:
[testing] C:\Users\bhupesh\workspace\Guru99AntExample\testng.xml
[testing] HELLO GURU99 ANT PROGRAM
[testing] TODAY's DATE IS->Wed Feb 18 01:37:54 IST 2015
[testing] =====
[testing] YT
[testing] Total tests run: 1, Failures: 0, Skips: 0
[testing] =====

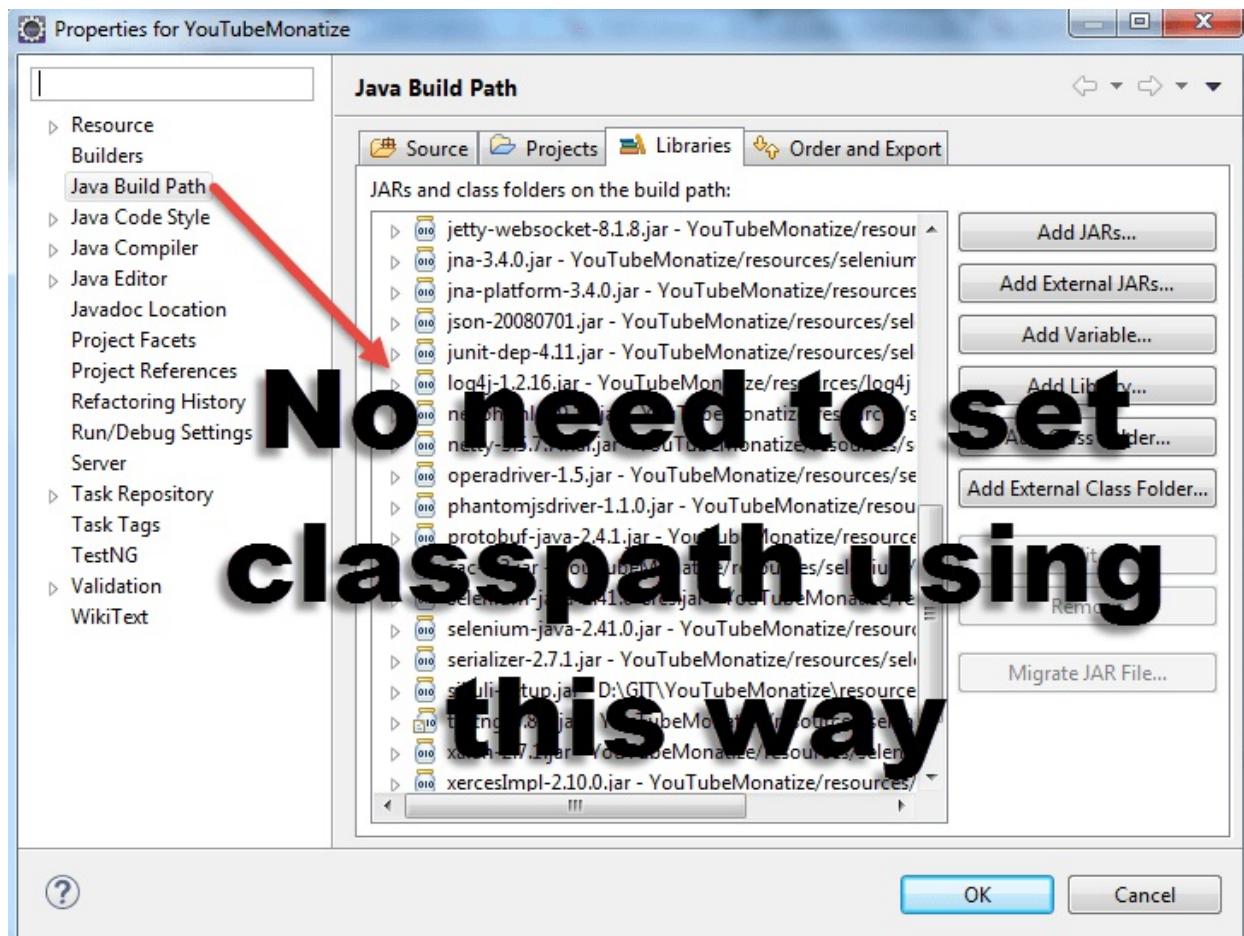
```

Ant with Selenium Webdriver:

So far, we have learned that using ANT we can put all third party jars in a particular location in the system and set their path for our project. Using this method we are setting all 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 class path of it manually in our application.

So now you can ignore below-mentioned traditional way to set classpaths for project.



Example:

We are going to modify the previous example

Step 1) Set the property selenium.jars to 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="${basedir}/build"/>
            <!-- put testing related jar in the resource folder -->
            <property name="external.jars"
value=".\\resource"/>
<!-- put selenium related jar in resource folder -->
        <property name="selenium.jars" value=".\\selenium"/>
            <property name="src.dir"
value="${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="${src.dir}">
            <classpath refid="classpath_jars"/>
        </javac>
</target>
<target name="runGuru99TestNGAnt" depends="compile">
    <!-- testing tag will be used to execute testng
code using corresponding testng.xml file -->
    <testng
classpath="${test.classpath};${build.dir}">
        <xmfileset dir="${basedir}"
includes="testng.xml"/>
    </testng>
</target>
</project>

```

Step 4) Now change previously created class Guru99AntClass.java with new code.

Here in this example our steps using Selenium are:

1. Go to http://demo.guru99.com/test/guru99home/
2. Read all courses links one by one
3. Print all courses hyperlink on console.

Guru99AntClass.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;

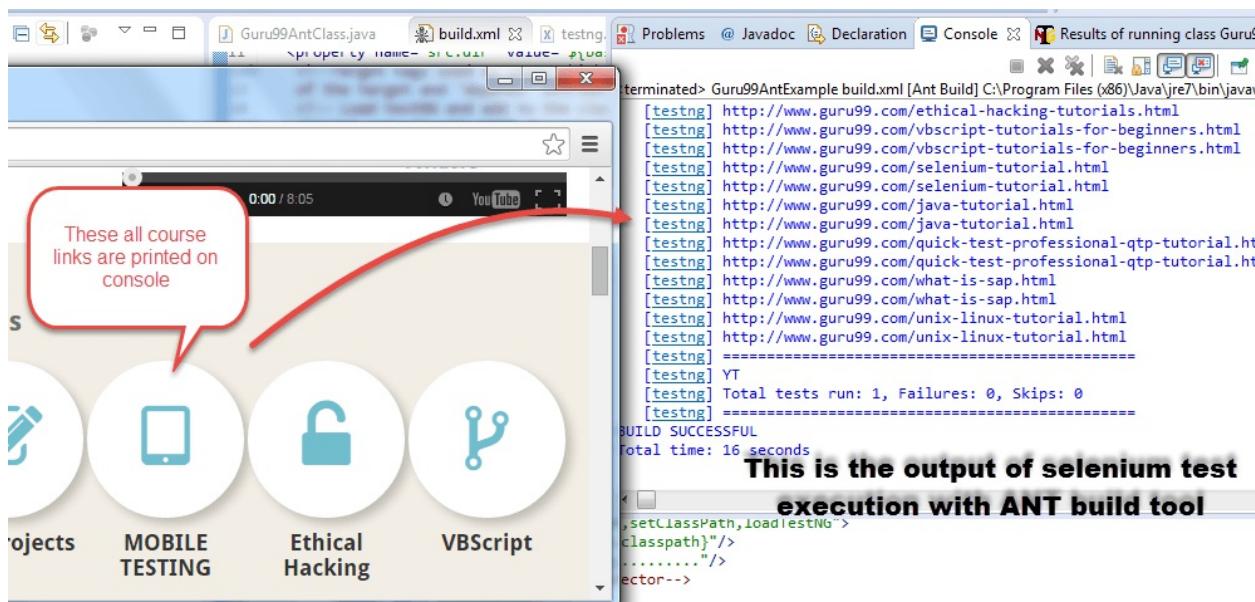
public class Guru99AntClass {

    @Test
        public void Guru99AntTestNGMethod(){
            WebDriver driver = new FirefoxDriver();

            driver.get("http://demo.guru99.com/test/guru99home/");
            List<WebElement> listAllCourseLinks =
            driver.findElements(By.xpath("//div[@class='canvas-
middle']//a"));
            for(WebElement webElement : listAllCourseLinks) {

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

Step 5) After successful execution output will look like:



Summary:

Ant is a build tool for Java.

Ant used for code compilation, deployment, execution process.

Ant can be downloaded from Apache website.

Build.xml file used to configure execution targets using Ant.

Ant can be run from the command line or suitable IDE plugin like eclipse.

Chapter 54: Using Selenium with Github

Git Hub is a Collaboration platform. It is built on top of git. It allows you to keep both local and remote copies of your project. A project which you can publish it among your team members as they can use it and update it from there itself.

Upload script to GITHUB



Advantages of Using Git Hub For Selenium.

- When multiple people work on the same project they can update project details and inform other team members simultaneously.
- Jenkins can help us to regularly build the project from the remote repository this helps us to keep track of failed builds.

Before we start selenium and git hub integration, we need to install the following components.

1. Jenkins Installation.
2. Maven Installation.
3. Tomcat Installation.

You can find this installation steps in the following links:

- 1) Maven and Jenkins installation Guide (</maven-jenkins-with-selenium-complete-tutorial.html>)
- 2) Tomcat Installation Guide (</apache.html>)

Git Binaries Installation

Now let us start by installing "Git Binaries".

Step 1) Launch the Browser and navigate to URL- <https://git-scm.com/>

Step 2) Download the latest stable release.

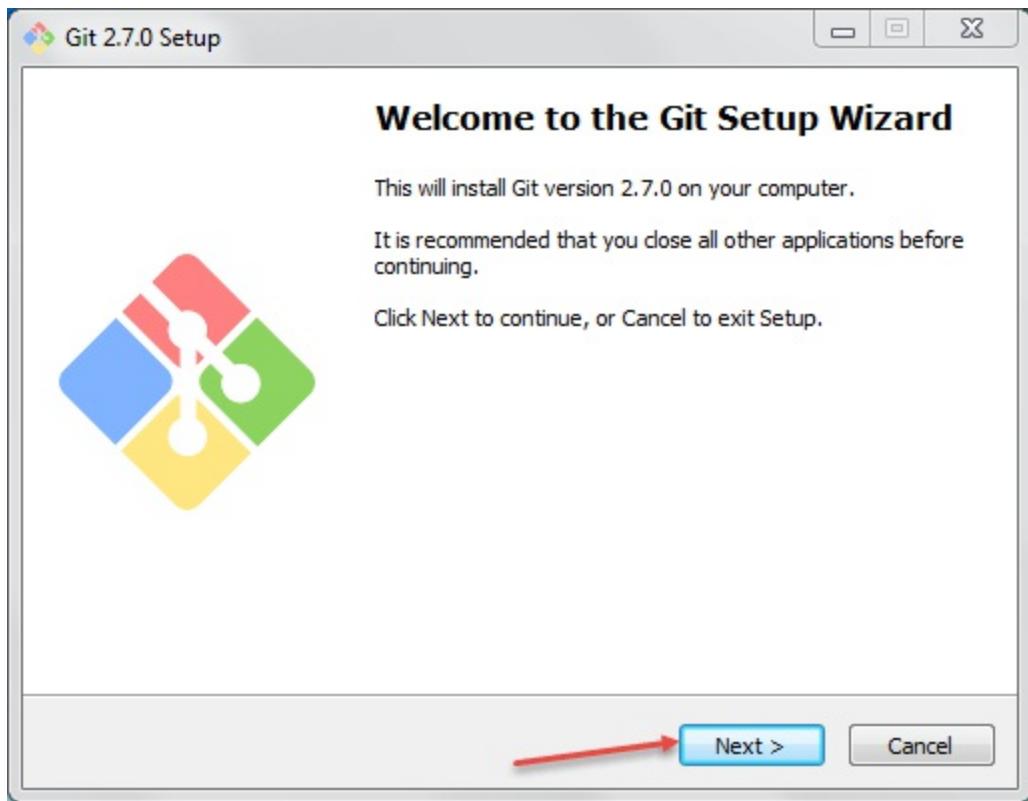
Step 3) Click on downloads for windows once the file is downloaded we can begin with our installation.



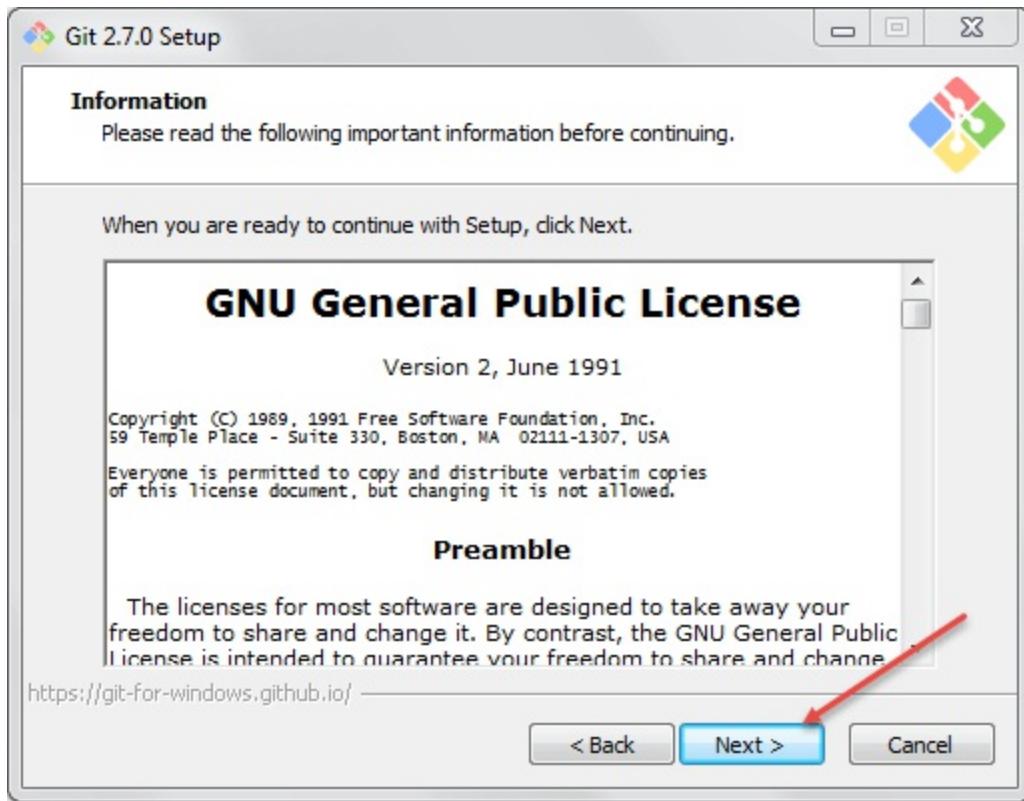
Step 4) Go to the download location or icon and run the installer.

Step 5) Click through welcome and General Public license.

Step 6) Click on "next" button in git setup wizard



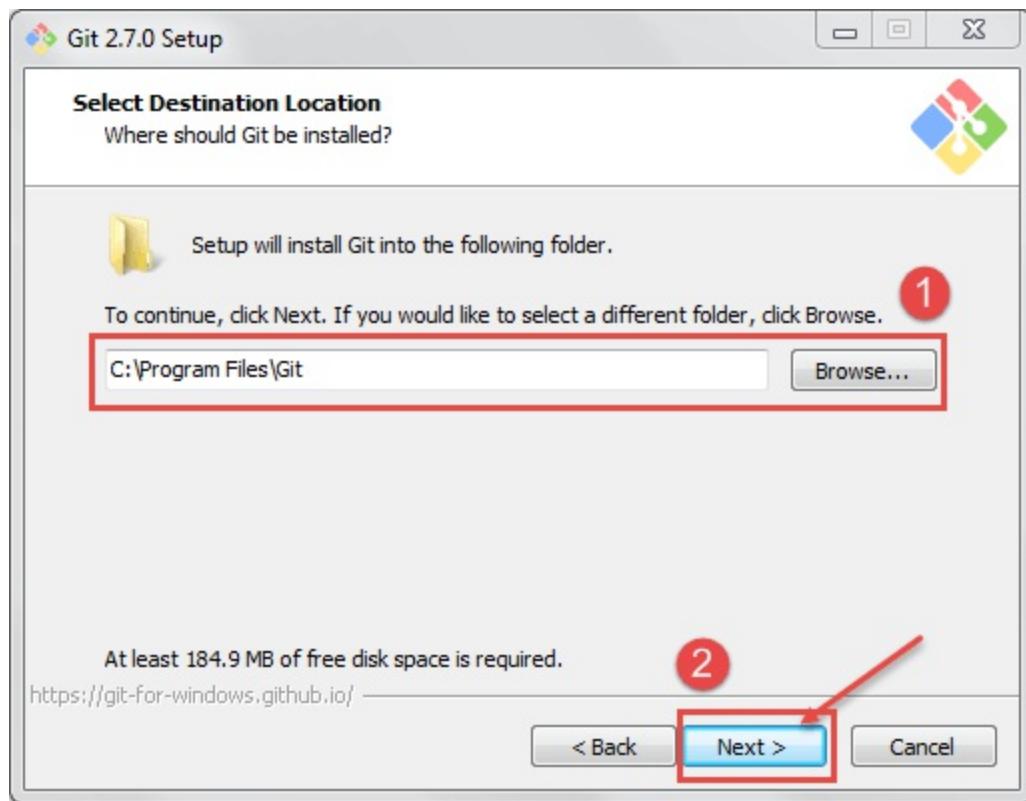
Step 7) Read the GNU General Public License and click on next



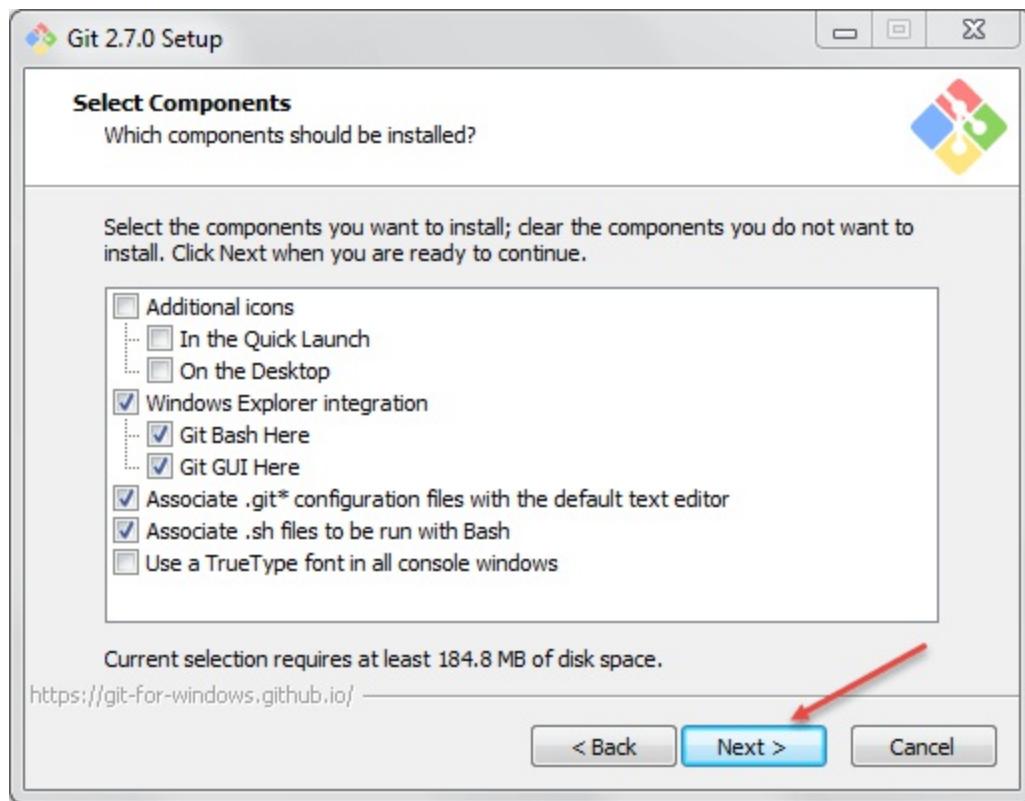
Another window will pop up,

Step 8) In this step,

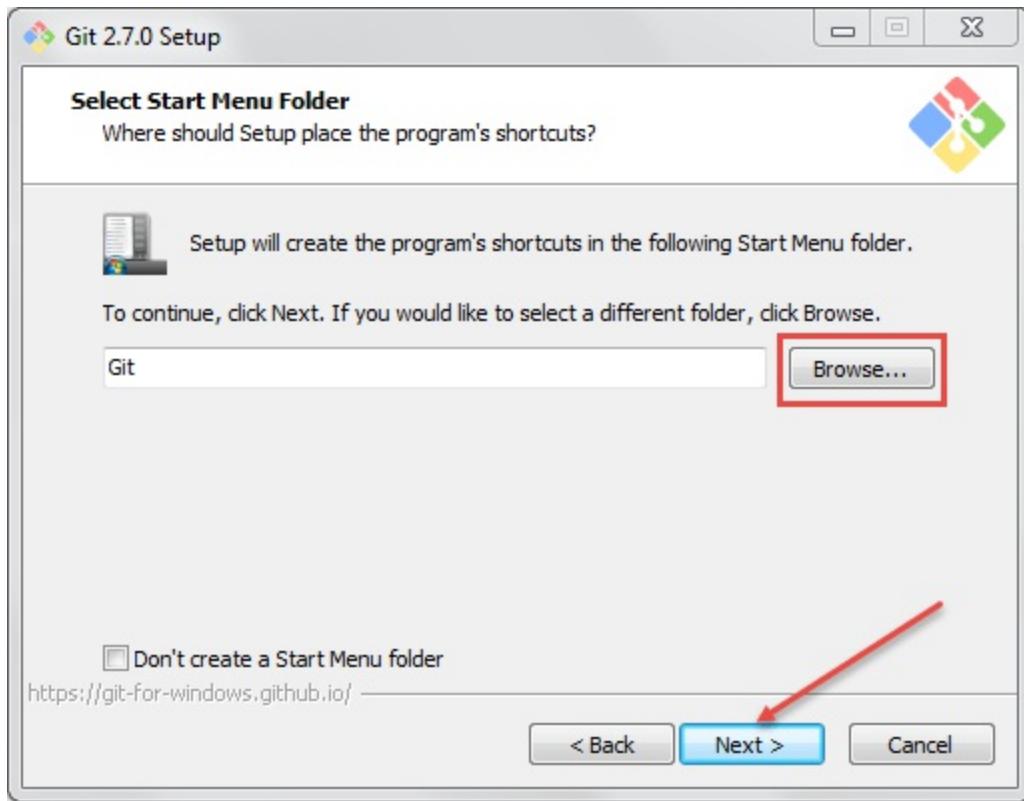
1. Select the Directory where you want to install "Git Binaries" and
2. Click on next button



Step 9) Select the component which you want to install and click on next



Step 10) If you want to create a start menu folder for Git, leave the setting default and click on next.



Step 11) In this step,

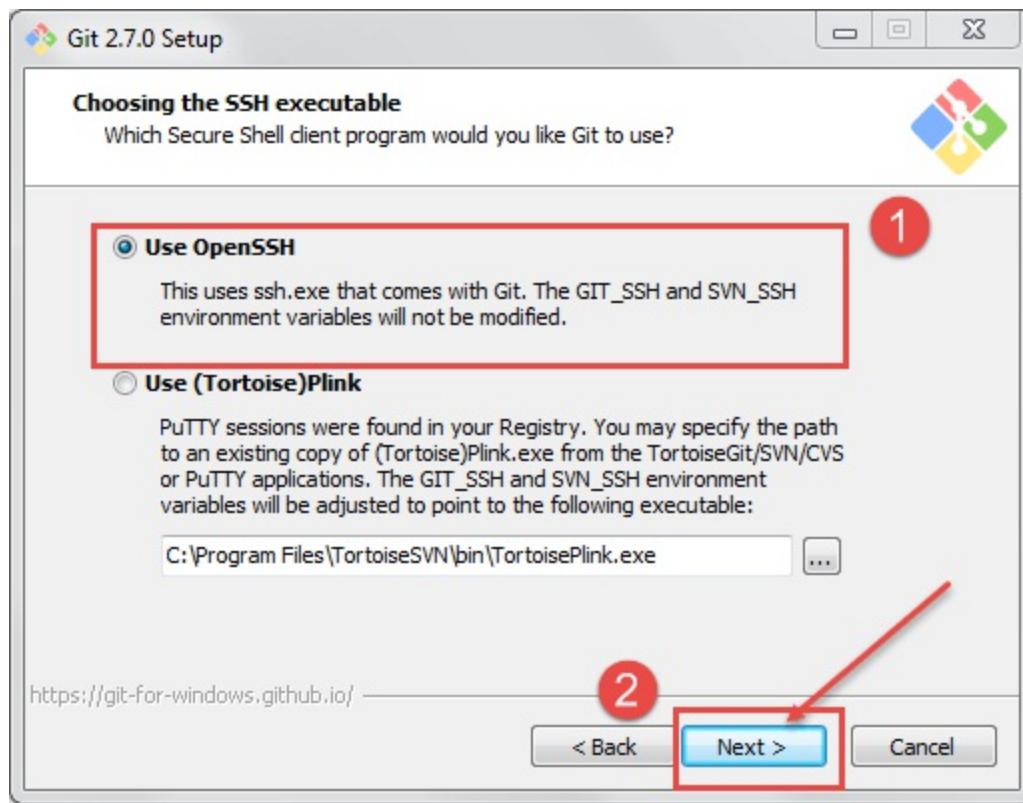
1. Select Use Git from the Windows Command Prompt to run Git from the command line and
2. Click on next.



Leave the default setting and click on next to install.

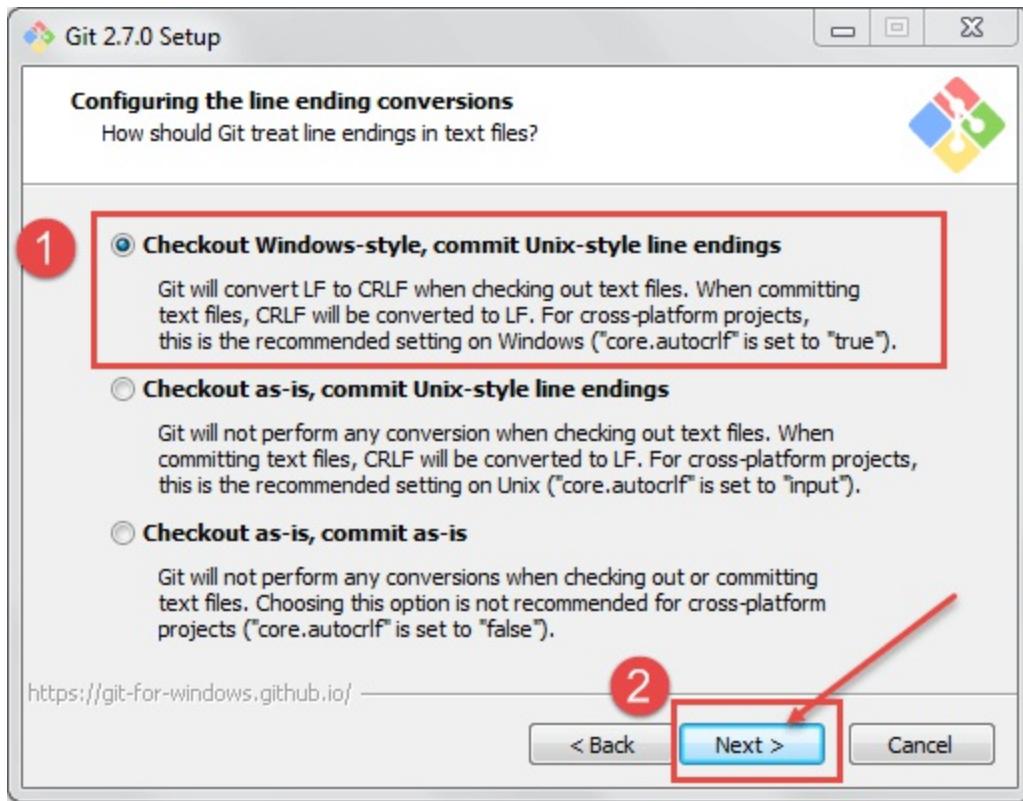
Step 12) In this step,

1. Select Use Open SSH It will help us to execute the command from the command line, and it will set the environmental path.
2. Click on next button.



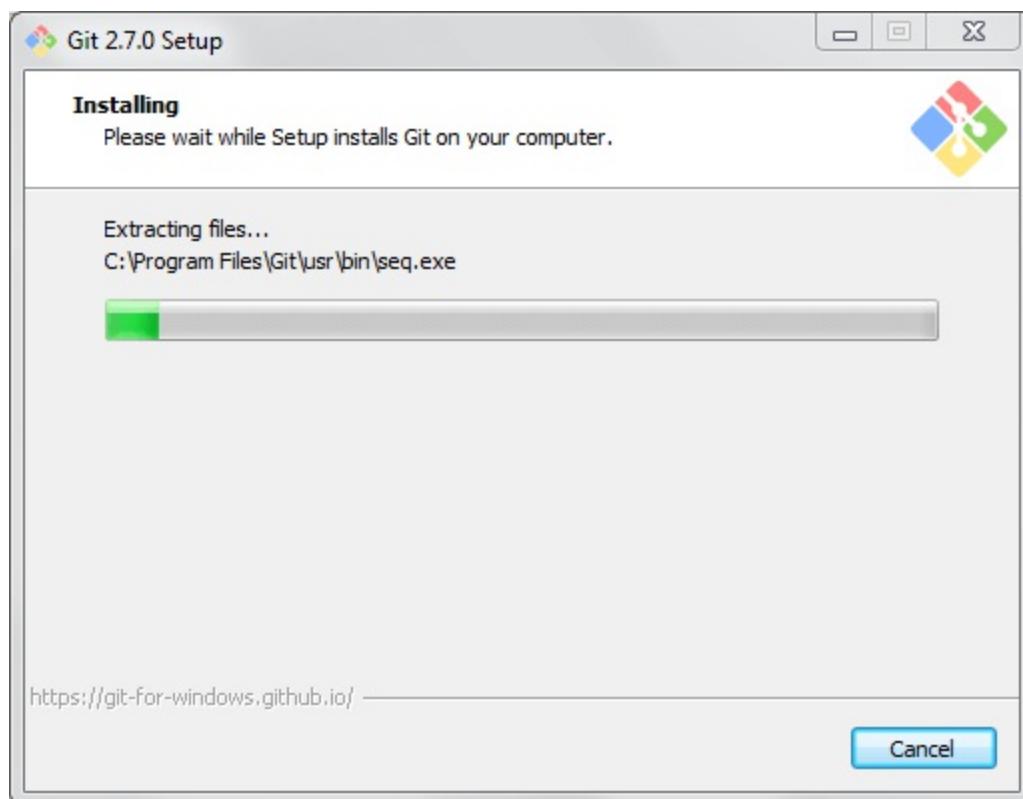
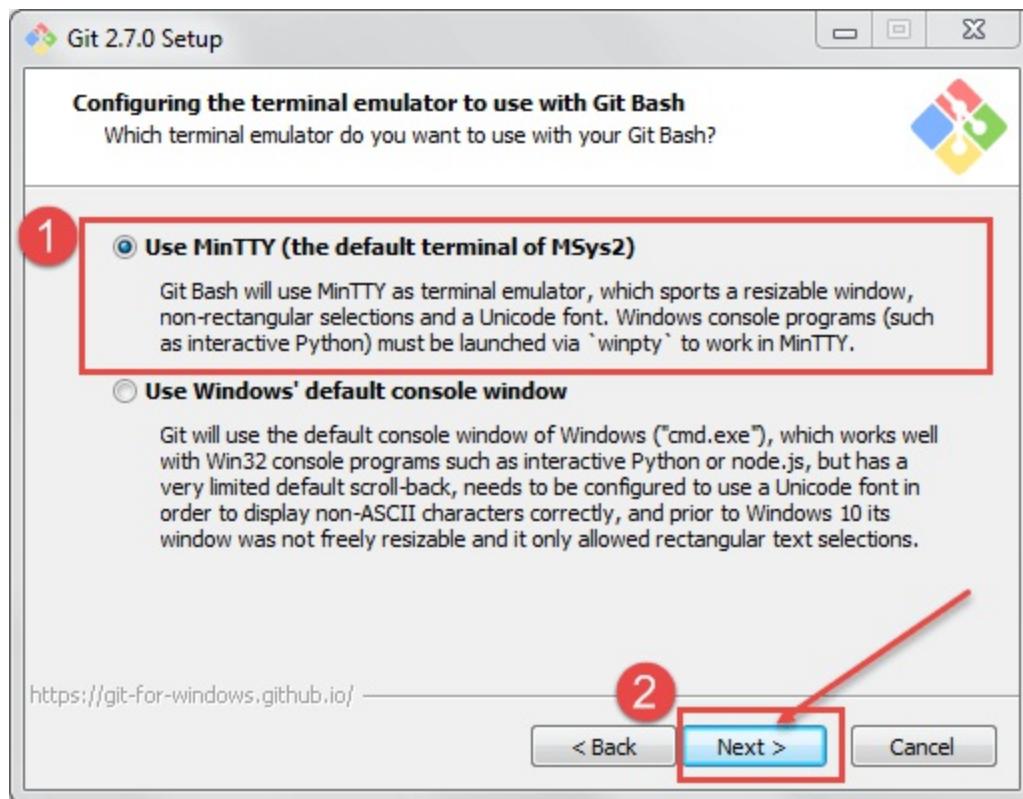
Step 13) In this step,

1. Select "Checkout windows-style, commit Unix-style line ending".
(how the git hub should treat line endings in text files).
2. Click on next button.



Step 14) In this step,

1. Select Use MinTTY is the default terminal of MSys2 for Git Bash
2. Click on next button



Once git is installed successfully, you can access the git.

Open Command prompt and type "git" and hit "Enter" If you see below screen means it is installed successfully

```

Administrator: C:\windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\CP042756>git
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  reset     Reset current HEAD to the specified state
  rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
  bisect    Use binary search to find the commit that introduced a bug
  grep      Print lines matching a pattern
  log       Show commit logs
  show      Show various types of objects
  status    Show the working tree status

grow, mark and tweak your common history
  branch   List, create, or delete branches
  checkout Switch branches or restore working tree files
  commit   Record changes to the repository
  diff     Show changes between commits, commit and working tree, etc
  merge   Join two or more development histories together
  rebase   Forward-port local commits to the updated upstream head
  tag     Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
  fetch   Download objects and refs from another repository
  pull    Fetch from and integrate with another repository or a local
          repository
  push    Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.

C:\Users\CP042756>

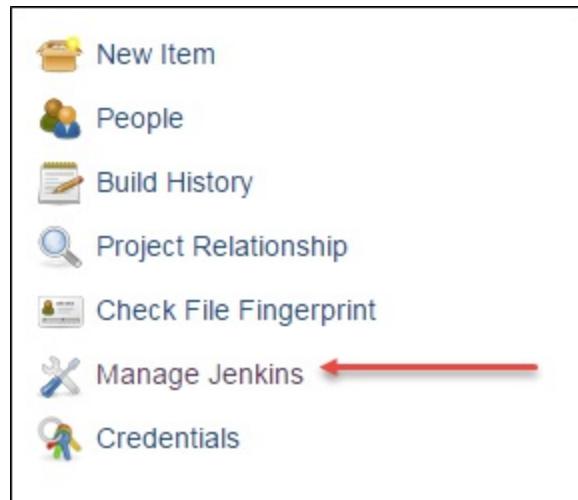
```

Jenkins Git Plugin Install

Now let's start with Jenkins Git Plugin Installation.

Step 1) Launch the Browser and navigate to your Jenkins.

Step 2) Click on Manage Jenkins.



Step 3) Click on Manage Plugins, it will open another window

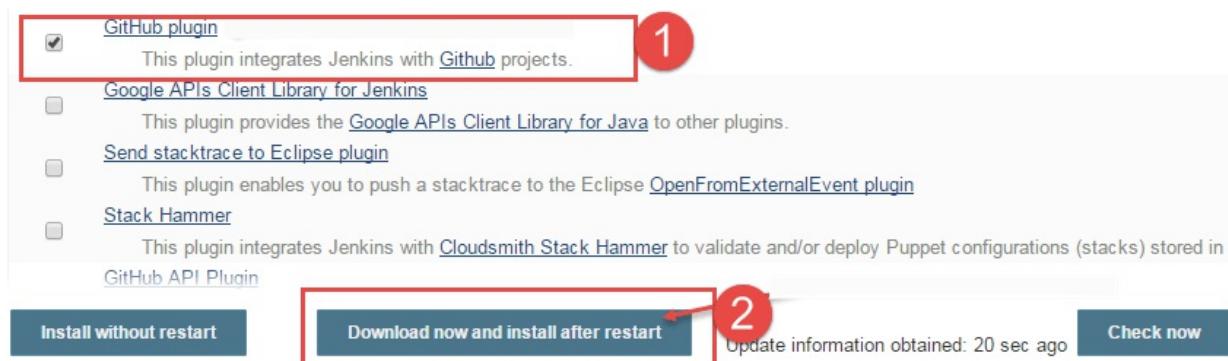


Step 4) Click on Available TAB



Step 5) In this step,

1. Select GitHub plugin then
2. Click on Download now and install after restart button.



Now it will install the following plugins.

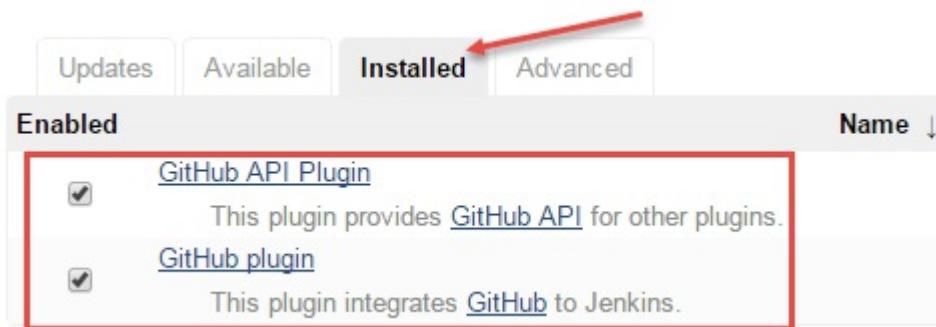
Once the Installation is finished. Restart your Tomcat server by calling the "shutdown.bat" file

Installing Plugins/Upgrades

Preparation	
	<ul style="list-style-type: none"> • Checking internet connectivity • Checking update center connectivity
GitHub API Plugin	● Pending
Credentials Plugin	● Pending
SSH Credentials Plugin	● Pending
Git client plugin	● Pending
SCM API Plugin	● Pending
Mailer Plugin	● Pending
JUnit Plugin	● Pending
Matrix Project Plugin	● Pending
Git plugin	● Pending
Token Macro Plugin	● Pending
Plain Credentials Plugin	● Pending
GitHub plugin	● Pending

After Restarting the tomcat and Jenkins we can see plugins are

installed in the "Installed" TAB.



Setting Up our Eclipse with GitHub Plugin

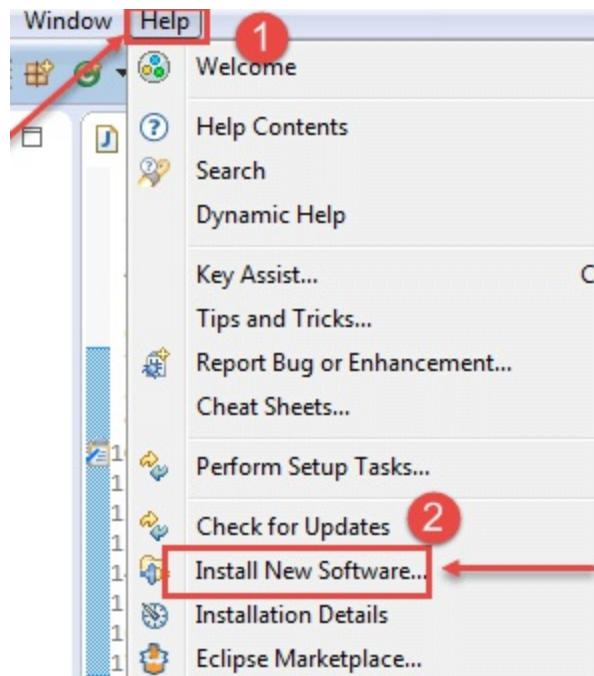
Now let's install GitHub Plugin for Eclipse.

URI for EGit Plugin location

<http://download.eclipse.org/egit/updates>

Step 1) Launch Eclipse and then

1. Click on help button then
2. Click on install new software

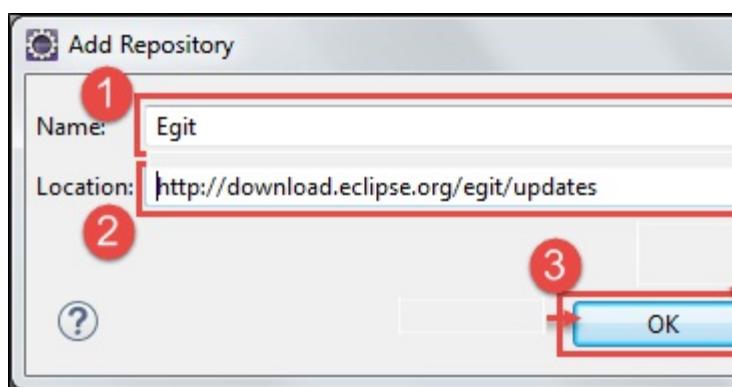


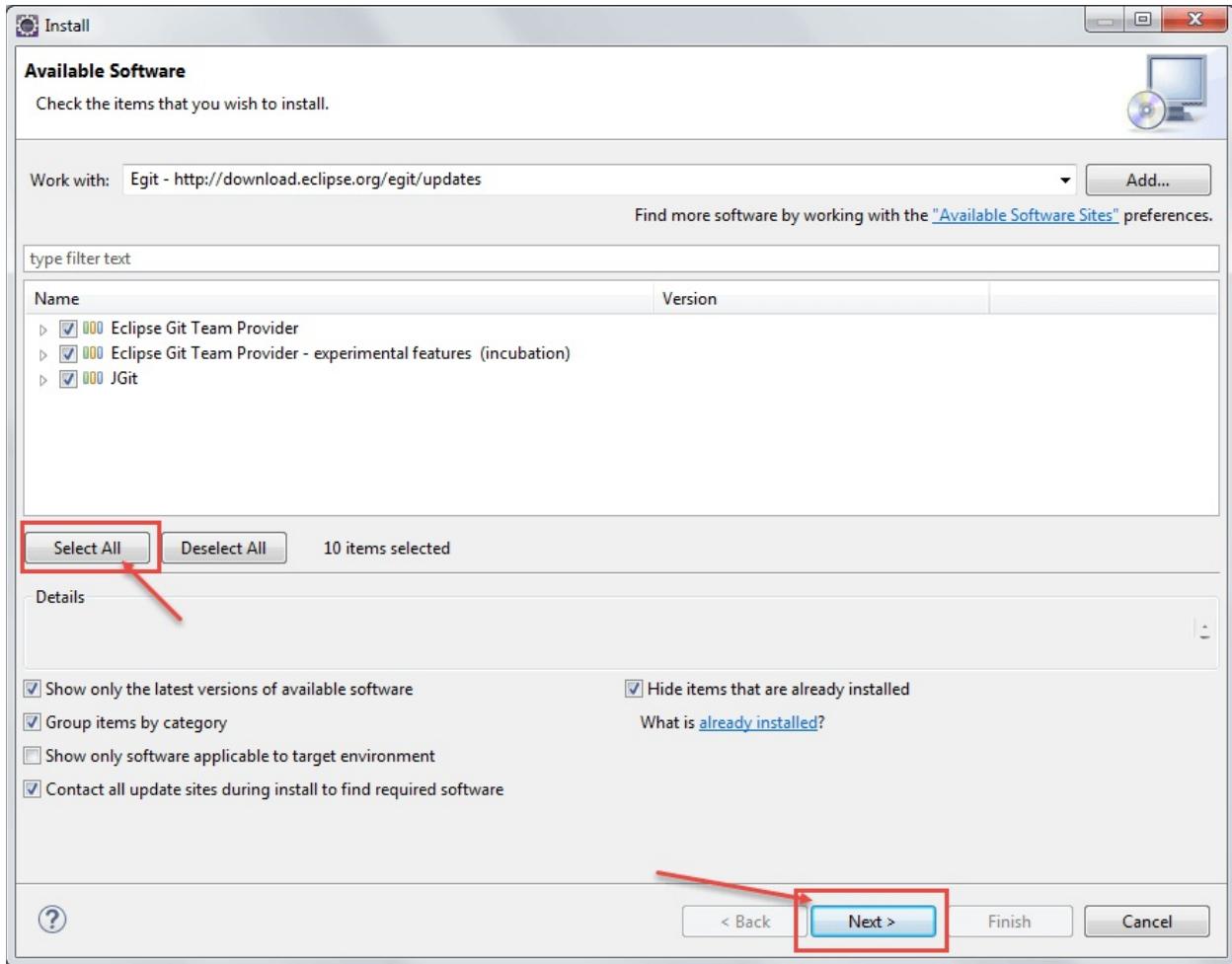
Step 2) The below screen will open once we click on the install new software. Now click on add



Step 3) In this step,

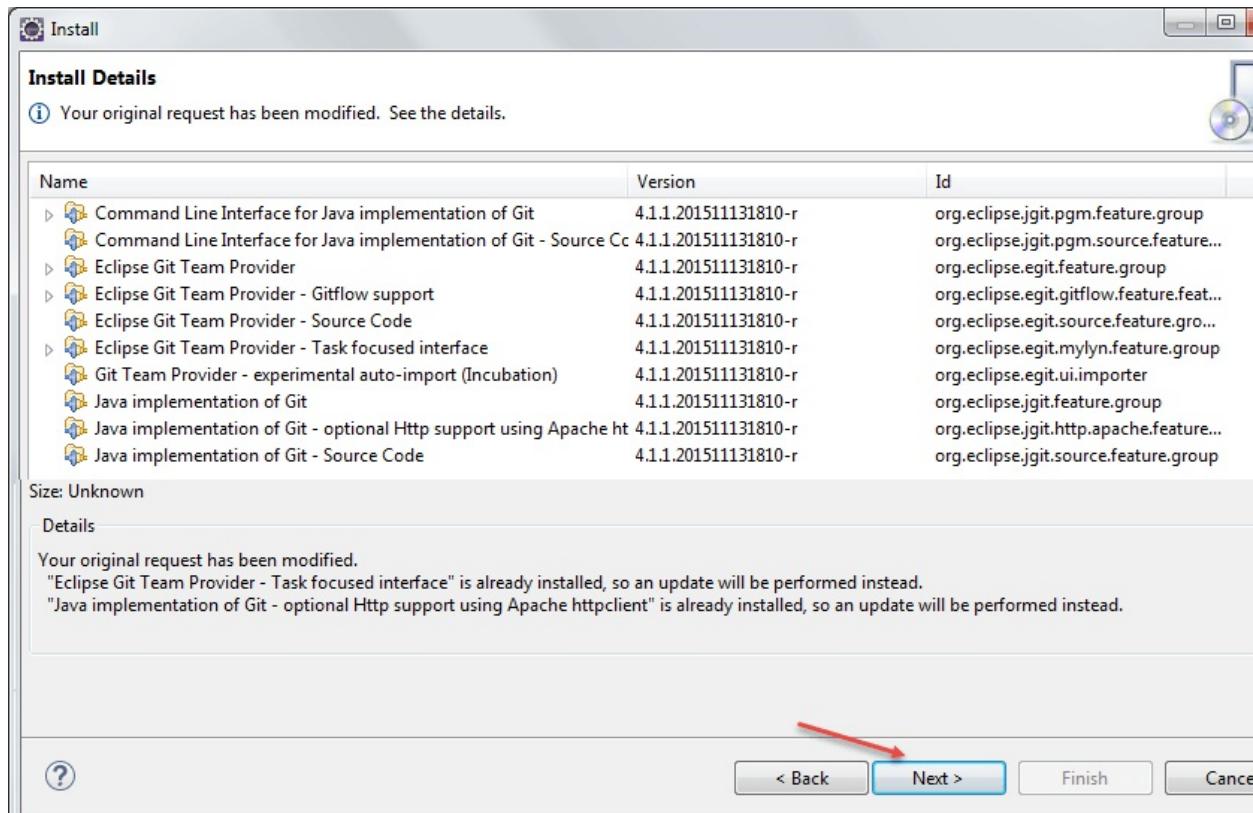
1. Type the name "EGIT" and
2. Enter the location <http://download.eclipse.org/egit/updates> then
3. Click on ok.



Step 4) Then click on select all and next

Step 5) Click on next and click accept the license agreement then finish the installation.

Then restart the eclipse.



Building a repository on Git

Step 1) Navigate to Git Hub URI: <https://github.com/> sign up for git hub

Step 2) Once you have been successfully signed up then click on create new repository



Step 3) In this step,

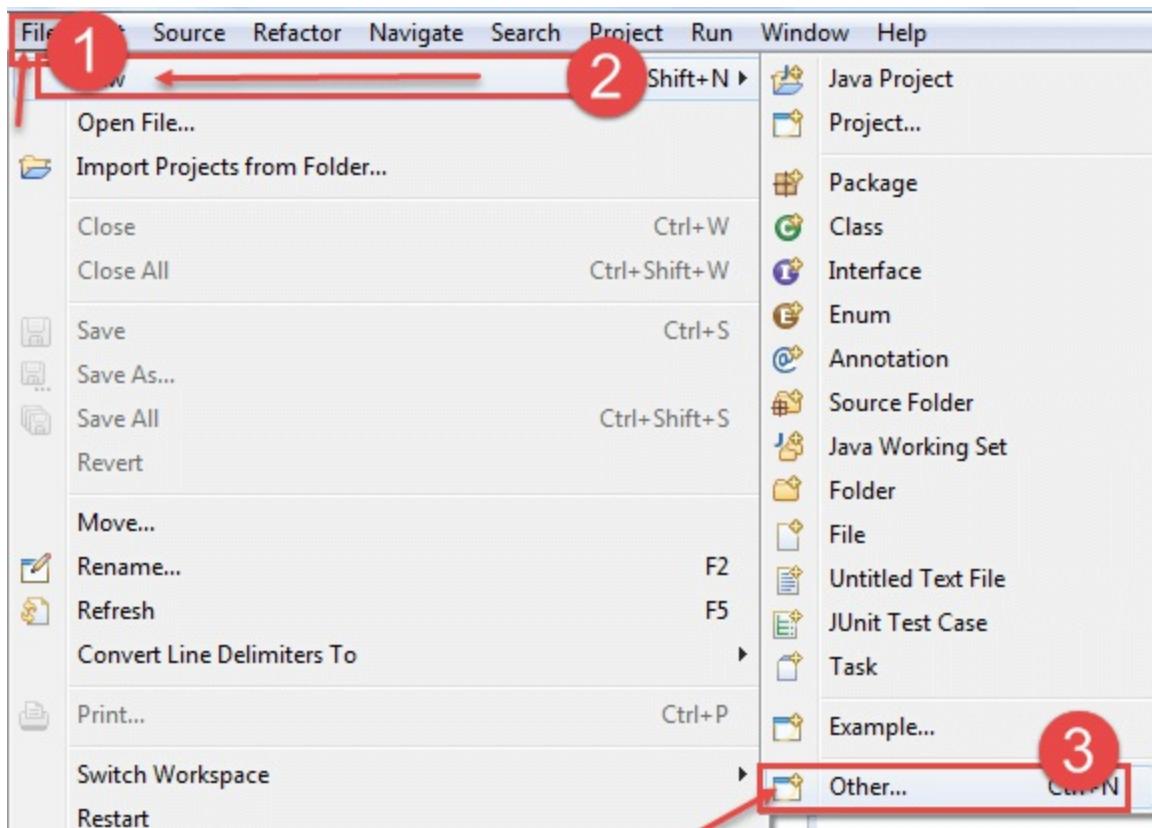
1. Enter the name of the repository and
2. click on create repository

The screenshot shows the GitHub 'Create repository' interface. A red box highlights the 'Repository name' input field, which contains a placeholder 'repository-name'. A red circle with the number '1' is positioned above the input field. Another red box highlights the green 'Create repository' button at the bottom left, and a red circle with the number '2' is positioned above it. Other visible fields include 'Owner' (set to 'chaitanyap97'), 'Description (optional)', repository visibility options ('Public' selected), and initialization checkboxes.

Testing Example Of Using Selenium with Git Hub.

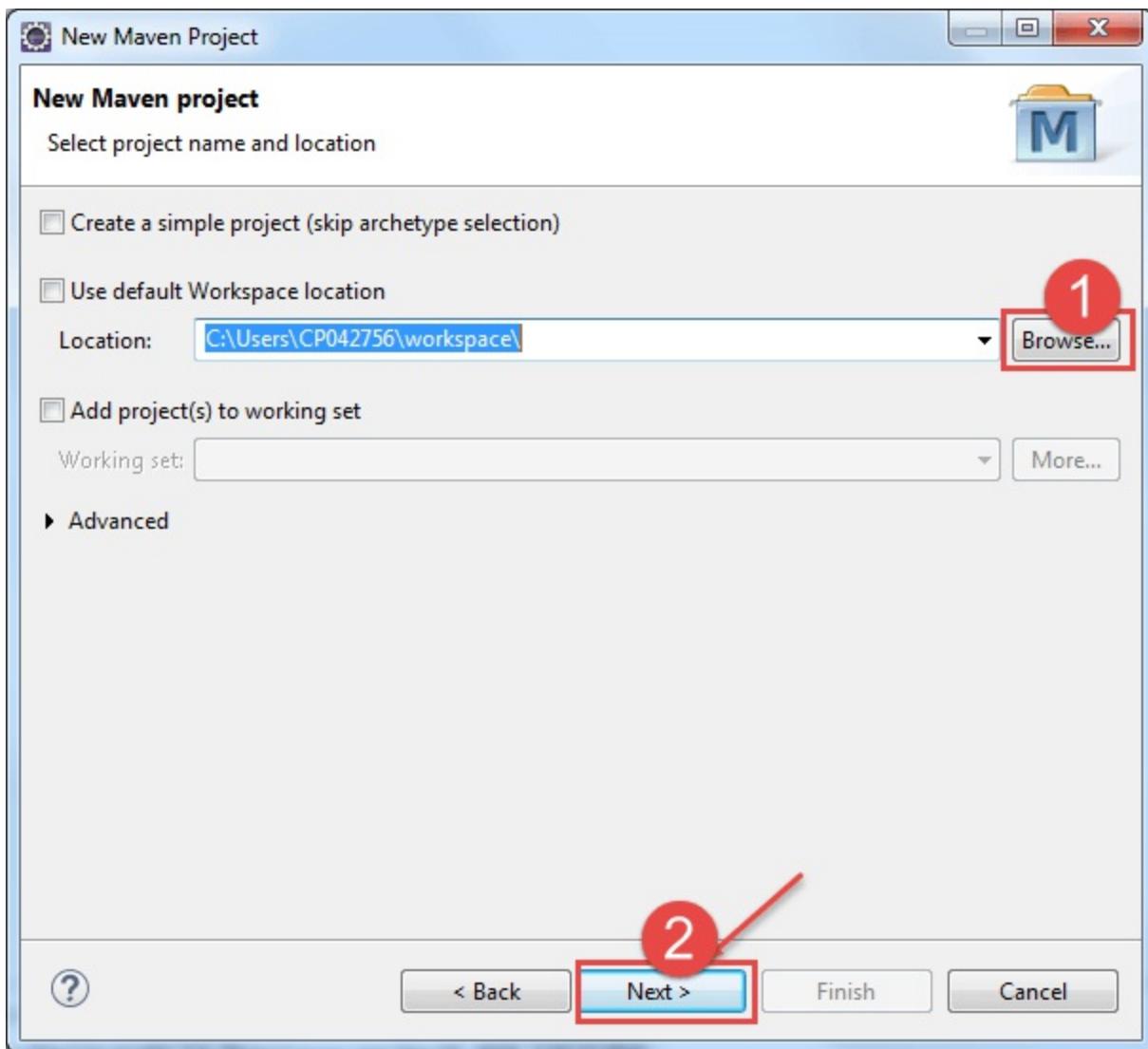
Step 1) Once we are done with the new repository, Launch Eclipse

1. Click on file
2. Then click on new button and then
3. Click on other



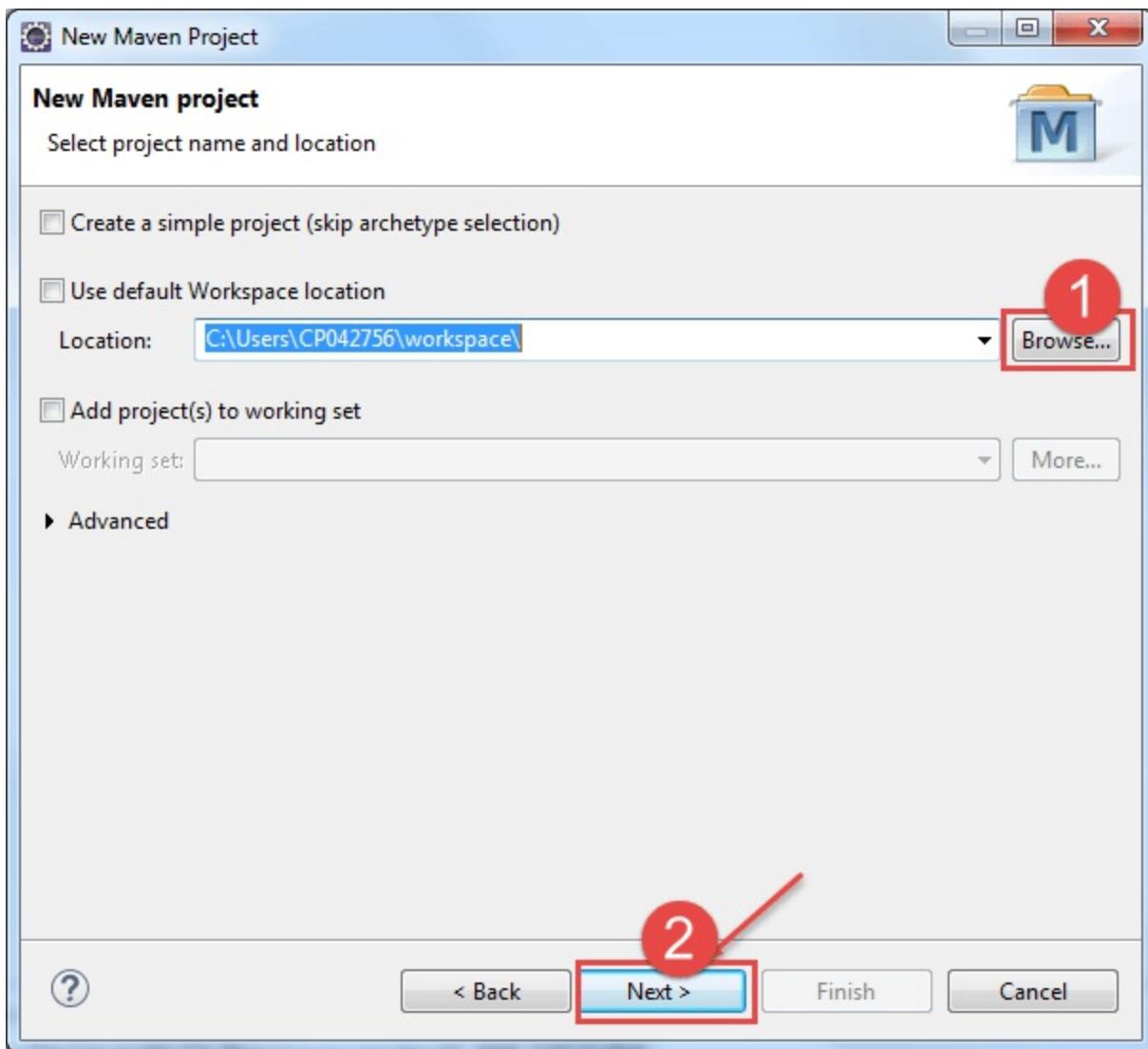
Step 2) In this step,

1. Select Maven Project and browse the location.
2. Click on next

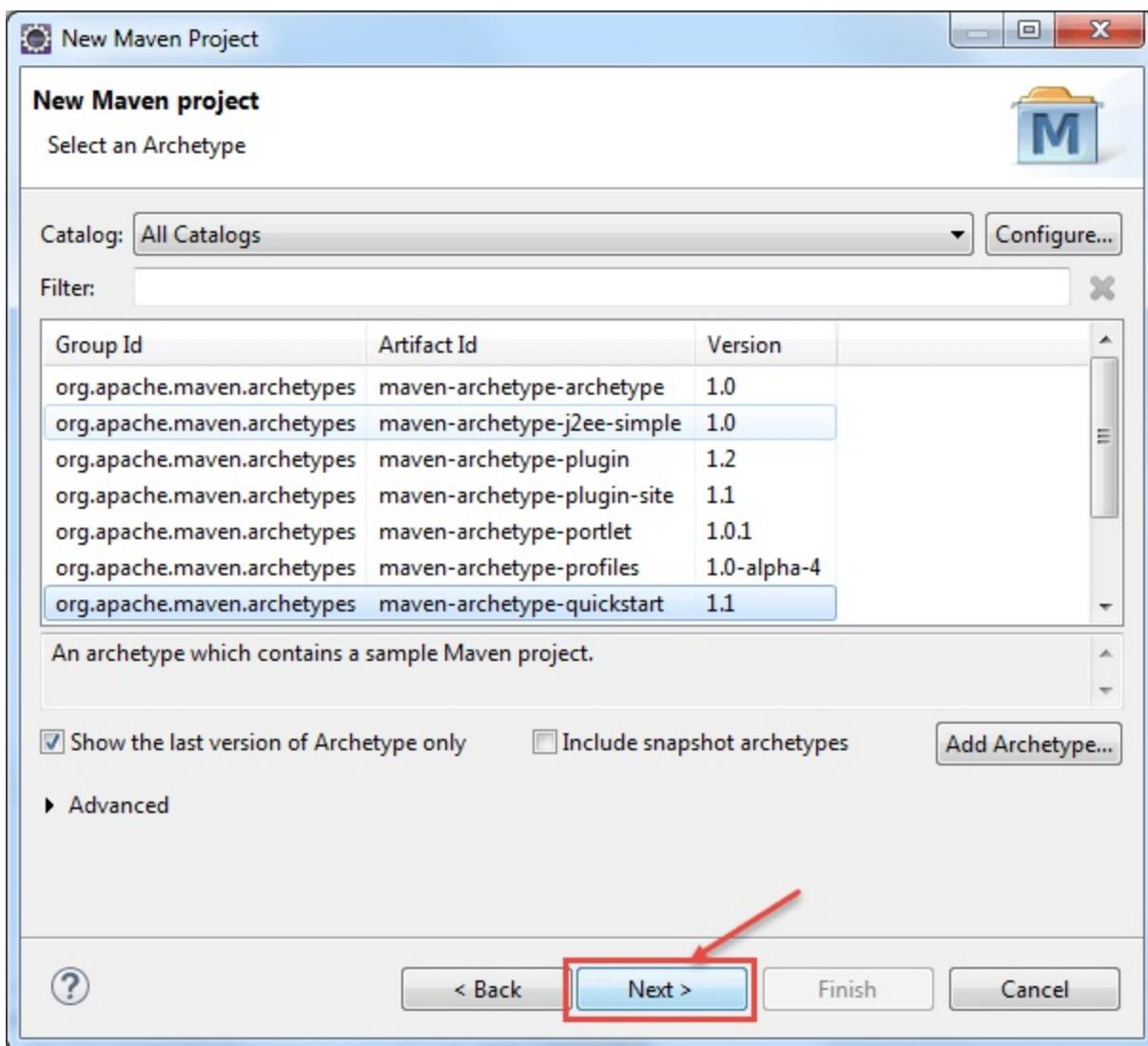


Step 3) In this step,

1. Select project name and location then
2. Click on next

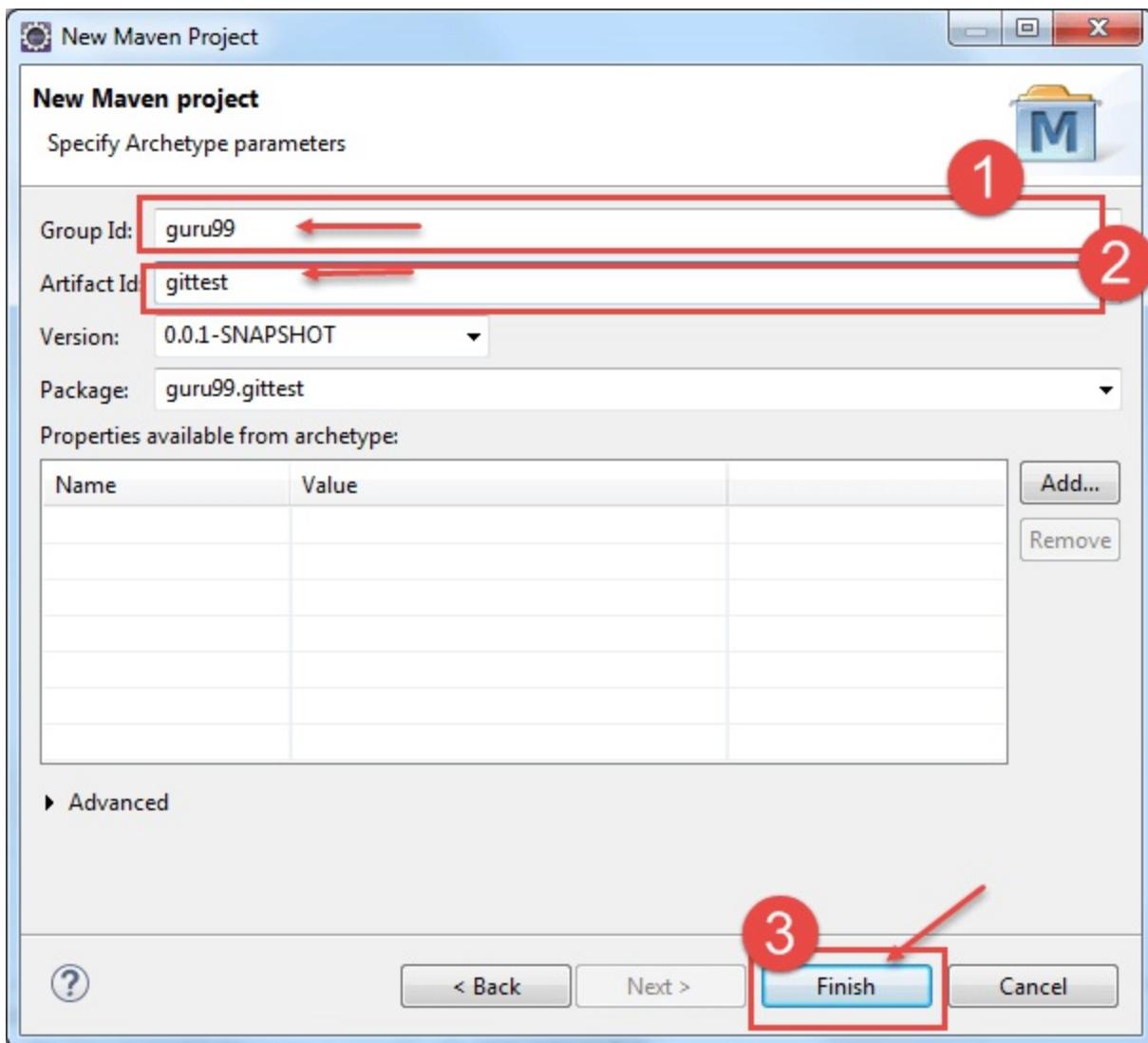


Step 4) Click on next



Step 5) In this step,

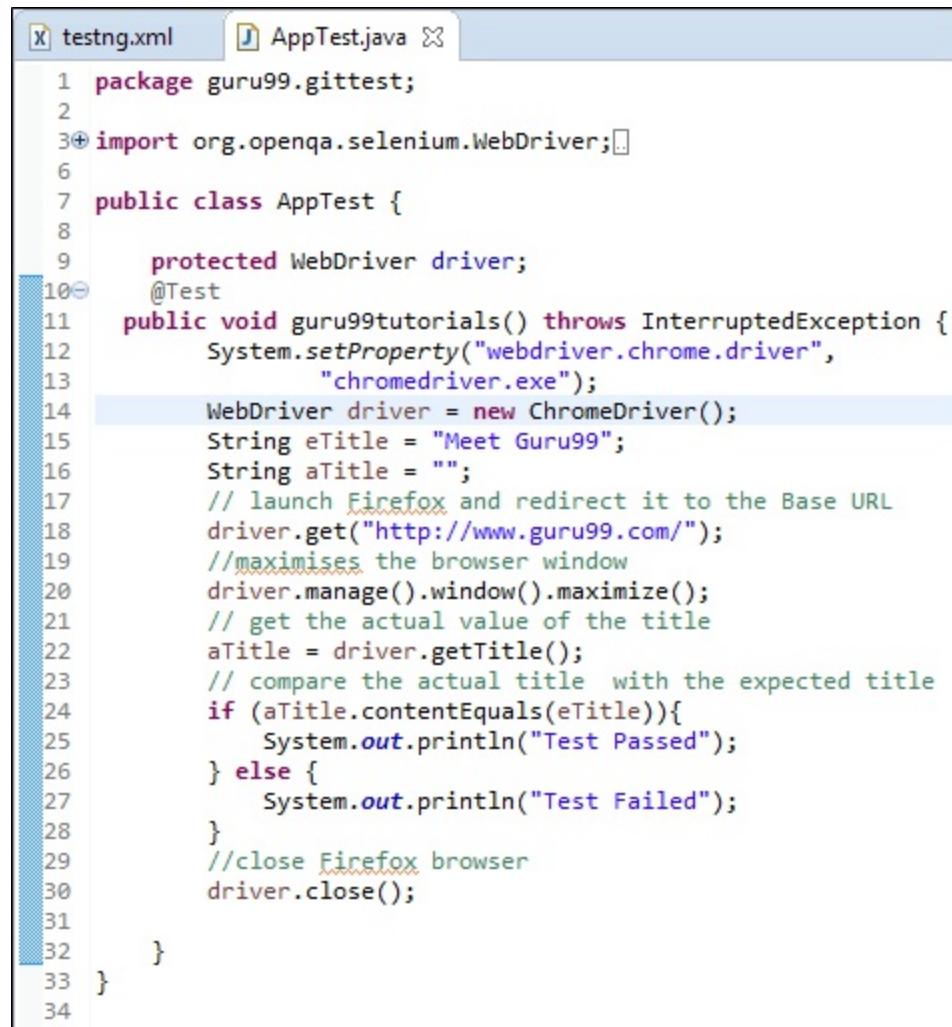
1. Enter Group Id and
2. Artifact Id and
3. Click on Finish button.



As soon as you click on finish button, a project will be created.

Step 6)

Now let's create a sample script



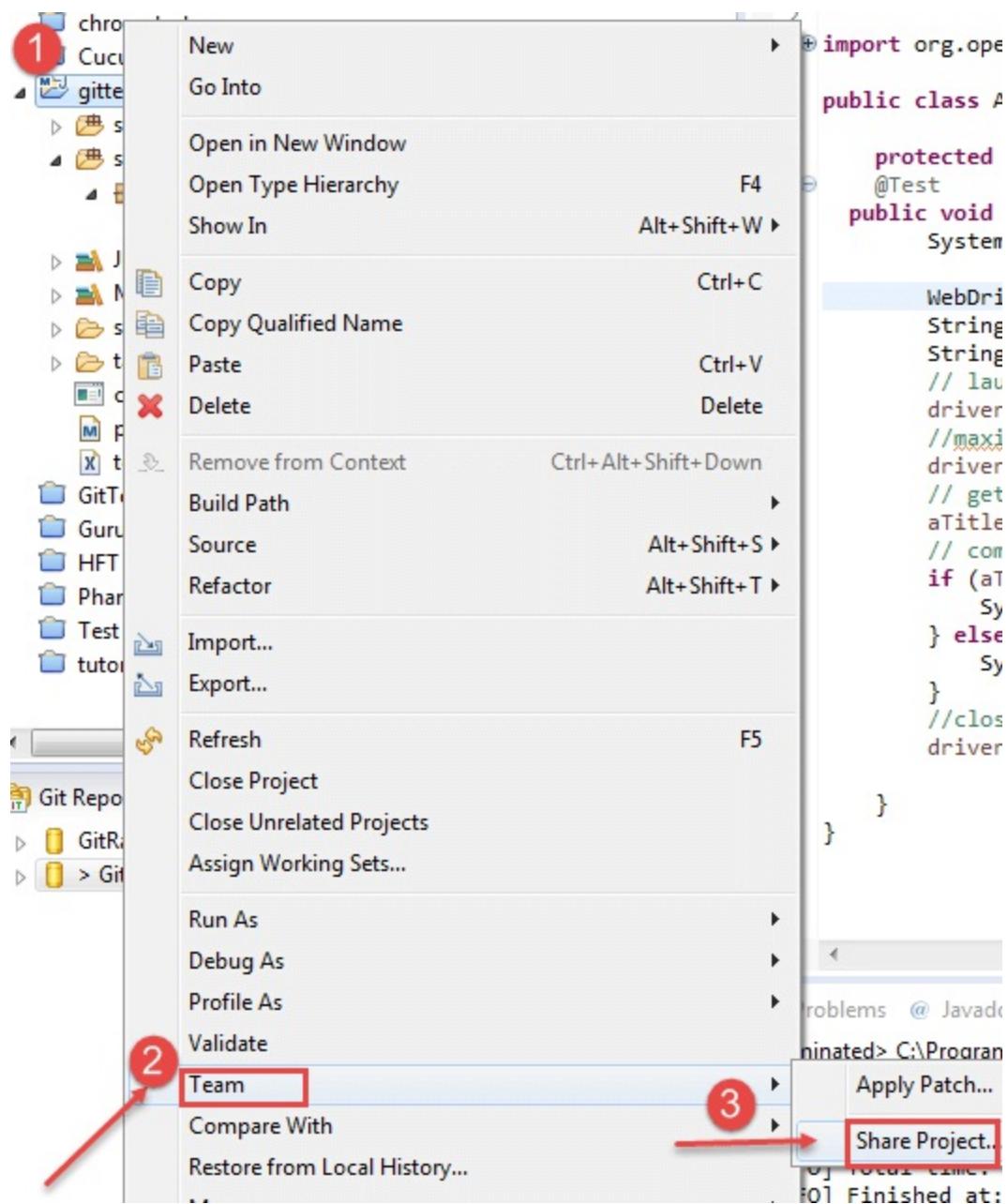
The screenshot shows the Eclipse IDE interface with two tabs open: 'testing.xml' and 'AppTest.java'. The 'AppTest.java' tab is active and displays the following Java code:

```
1 package guru99.gittest;
2
3+ import org.openqa.selenium.WebDriver;
4
5
6 public class AppTest {
7
8     protected WebDriver driver;
9
10    @Test
11    public void guru99tutorials() throws InterruptedException {
12        System.setProperty("webdriver.chrome.driver",
13                            "chromedriver.exe");
14        WebDriver driver = new ChromeDriver();
15        String eTitle = "Meet Guru99";
16        String aTitle = "";
17        // launch Firefox and redirect it to the Base URL
18        driver.get("http://www.guru99.com/");
19        //maximises the browser window
20        driver.manage().window().maximize();
21        // get the actual value of the title
22        aTitle = driver.getTitle();
23        // compare the actual title with the expected title
24        if (aTitle.contentEquals(eTitle)){
25            System.out.println("Test Passed");
26        } else {
27            System.out.println("Test Failed");
28        }
29        //close Firefox browser
30        driver.close();
31
32    }
33 }
34 }
```

Let's push the code/local repository to Git Hub.

Step 7) In this step,

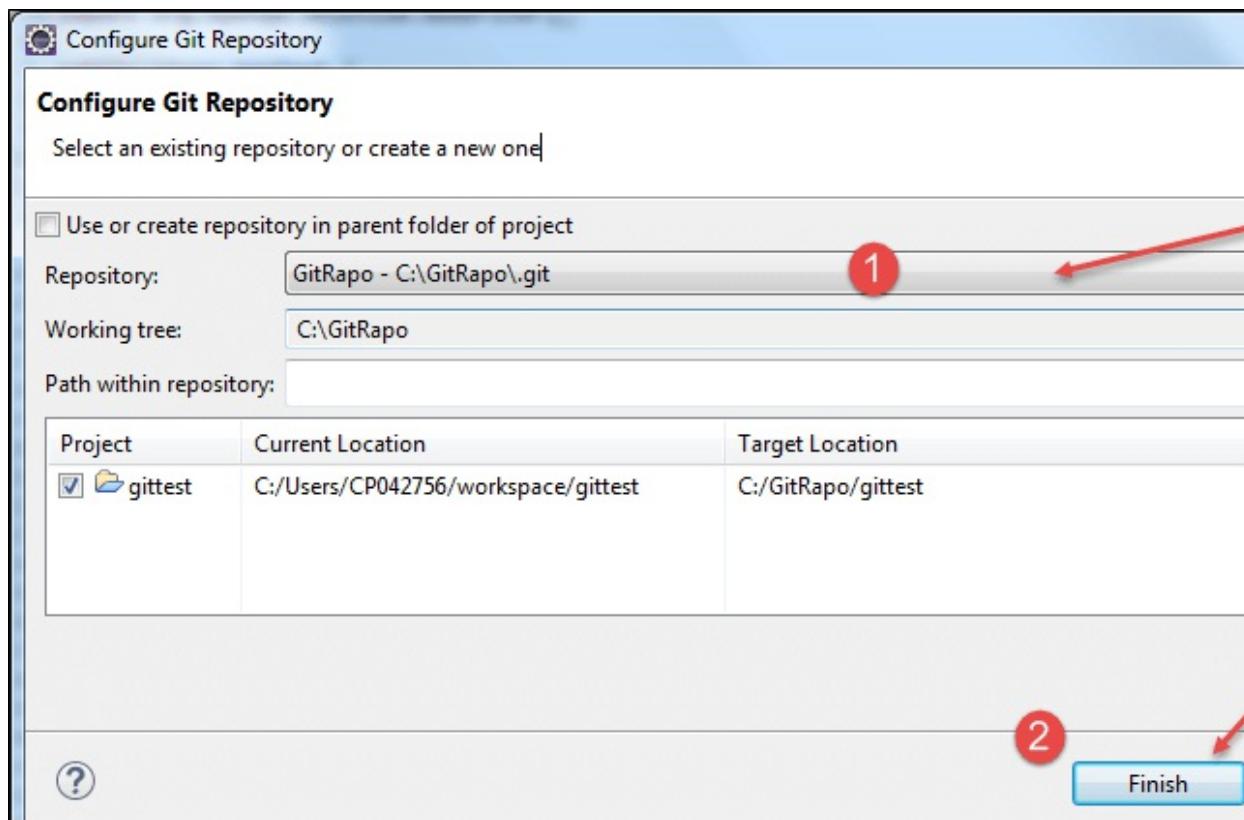
1. Open eclipse and then navigate to the project
2. Right-click on the project and Select "team" then
3. Select share project



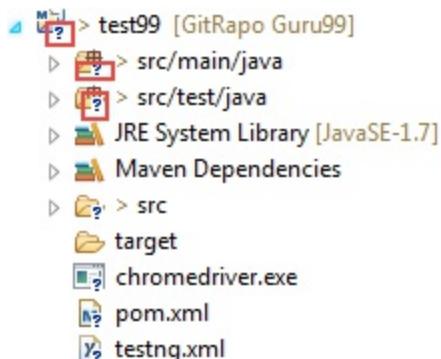
Step 8) Once we click on the "Share Project" in above screen, we will get another window

In this step,

1. Select the local repository and
2. Click on finish.



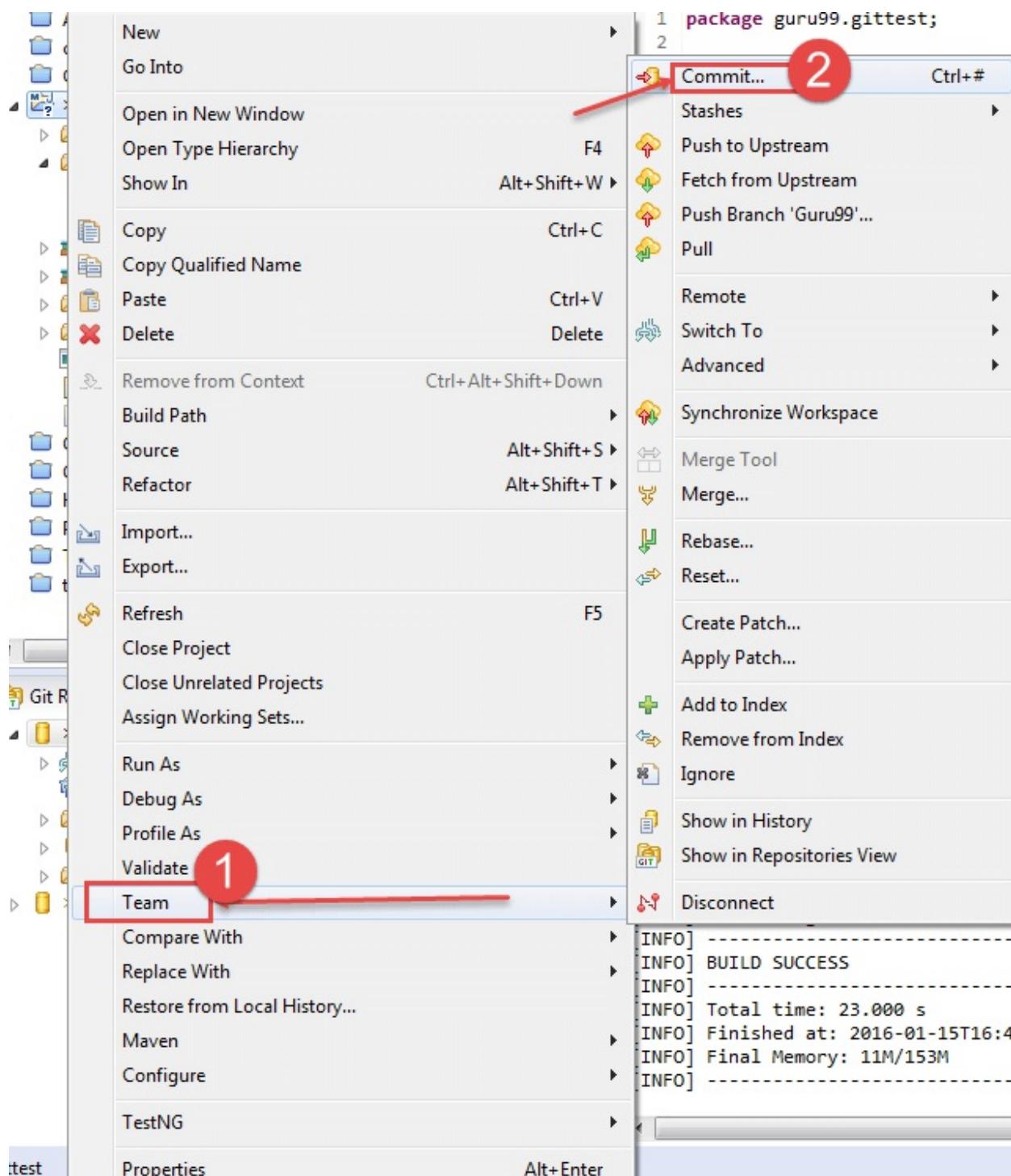
Once we click on Finish, we can see the change in the project structure that we have created a local repository.



Now it's time to push our code to Git Hub Repository

Step 9) In this step,

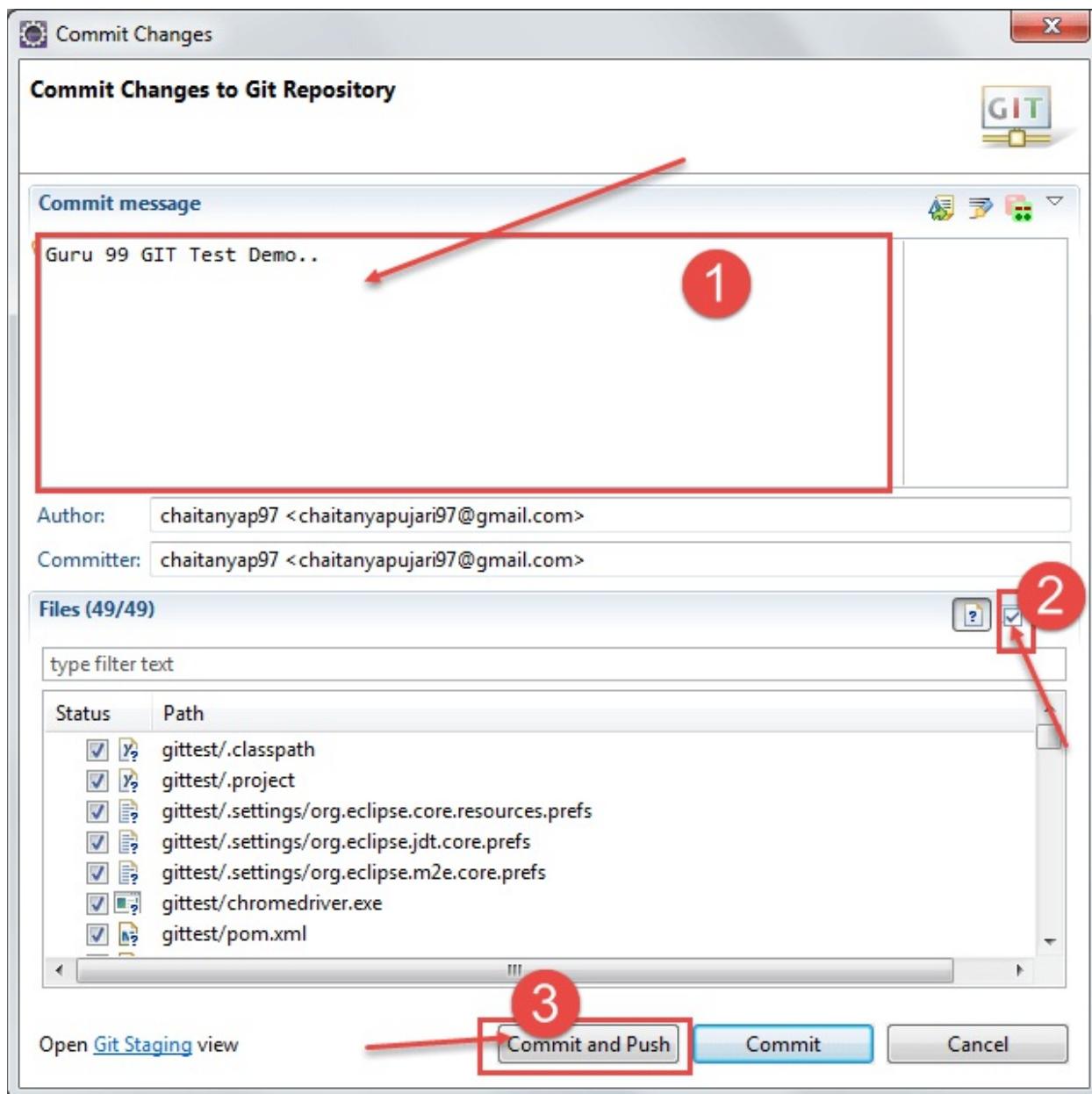
1. Right-click on the project and team then
2. Click on commit



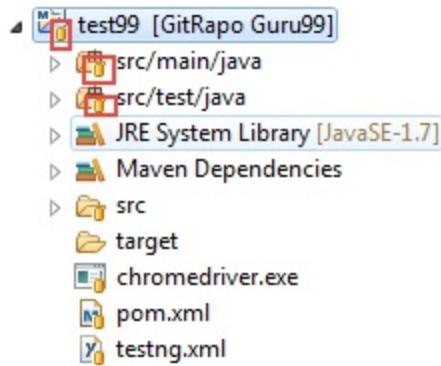
Step 10) In this step,

1. Enter a commit message and
2. Select the files which we want to send to Git Hub repository

3. Click on commit and push



Once you are done with it, you could see the icons in the project is being changed it says that we have successfully pushed and committed our code to Git Hub

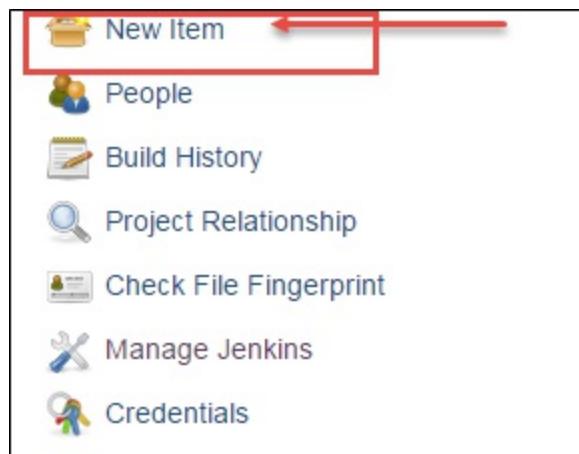


We can verify in the Git hub in the repository that our project is successfully pushed into repository

Now it's time for executing our project from Git Hub in Jenkins

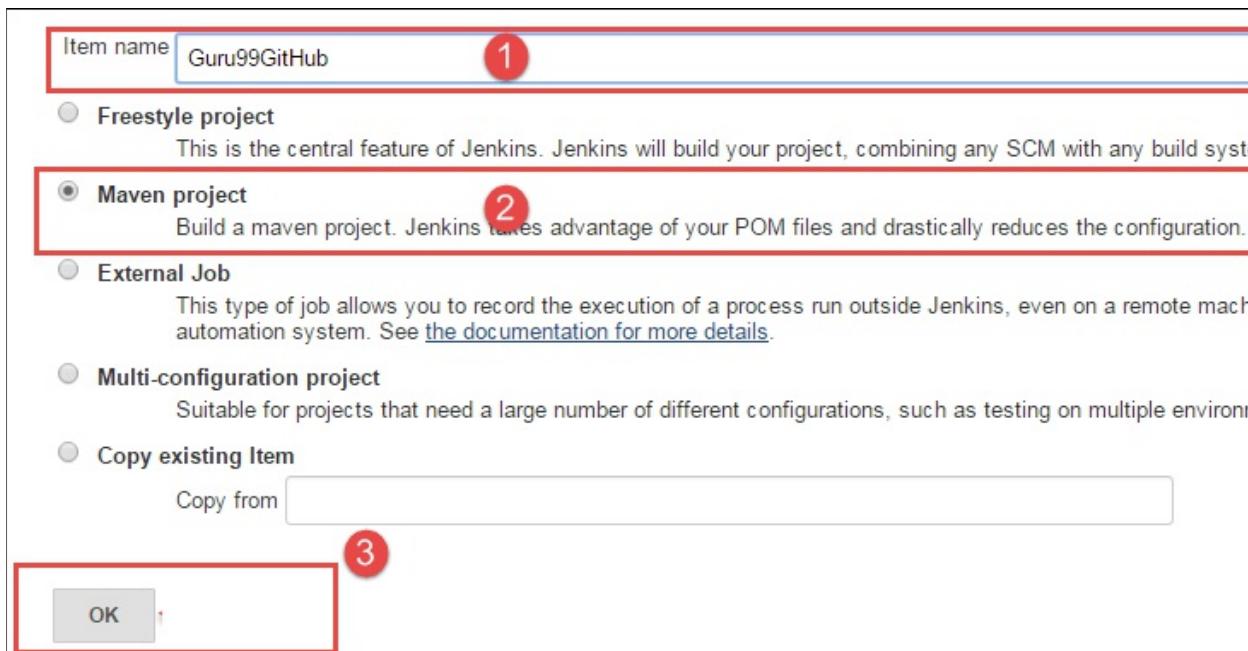
Step 11) Launch browser and open your Jenkins.

Step 12) Click on new Item.



Step 13) In this step,

1. Enter Item name
2. Select Maven Project
3. Click on ok button



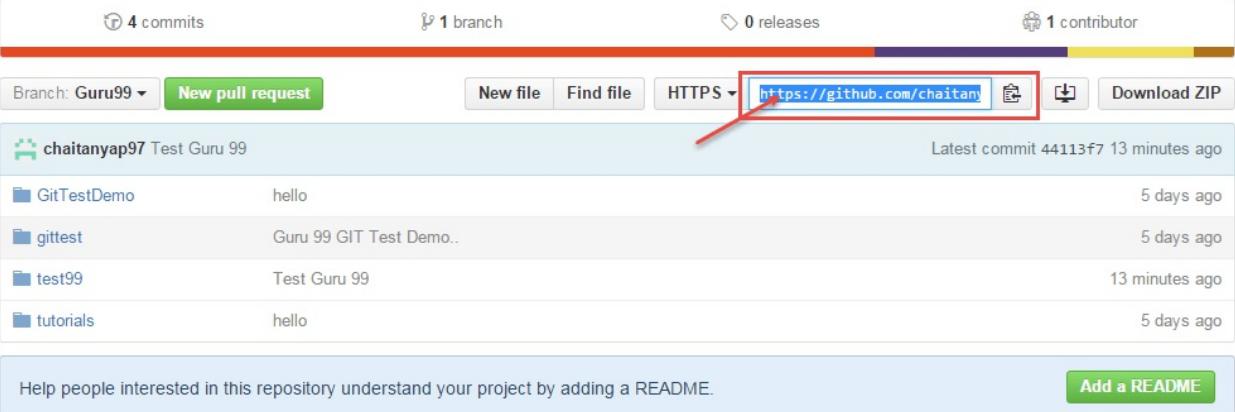
Step 14) In this step, we will configure Git Hub in Jenkins

1. Click on Git and
2. Enter the Repository URI
3. Click on Add repository

If you have multiple repositories in Git Hub, you need to add name Refspec field of the repository.



We can get the URI in Git Hub



The screenshot shows a GitHub repository page for 'chaitanyap97 Test Guru 99'. At the top, there are statistics: 4 commits, 1 branch, 0 releases, and 1 contributor. Below this is a navigation bar with 'Branch: Guru99', 'New pull request', 'New file', 'Find file', 'HTTPS' (with a dropdown menu), and download options ('Download ZIP'). A red box highlights the 'HTTPS' dropdown menu, which contains the URL 'https://github.com/chaitanyap97/Test-Guru-99'. The main content area lists repository files: 'GitTestDemo' (hello), 'gittest' (Guru 99 GIT Test Demo..), 'test99' (Test Guru 99), and 'tutorials' (hello). A message at the bottom encourages adding a README, with a 'Add a README' button.

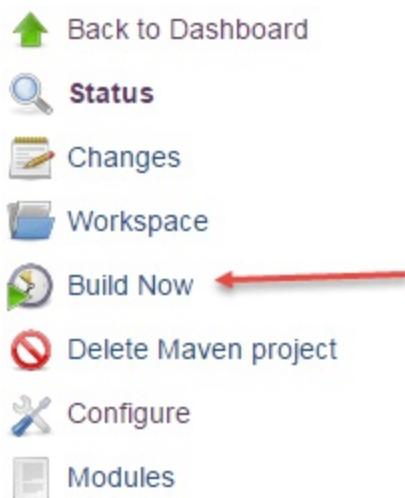
Step 15) In this step,

1. Add the pom.xml file location in the textbox and
2. Specify the goals and options for Maven then
3. Select option on how to run the test
4. Click on save button.

The screenshot shows the Jenkins build configuration interface. It includes sections for Pre Steps, Build, Post Steps, Build Settings, and Post-build Actions. Annotations are present: a red circle labeled '1' is over the 'Root POM' field containing 'gittest\pom.xml'; a red circle labeled '2' is over the 'Goals and options' field containing 'clean test'; a red circle labeled '3' is over the 'Run regardless of' radio button; and a red circle labeled '4' is over the 'Save' button, with a red arrow pointing from it towards the 'Build Now' button in the navigation menu below.

Step 16) once we click on save below screen will appear,

Now we can build our project click on build.

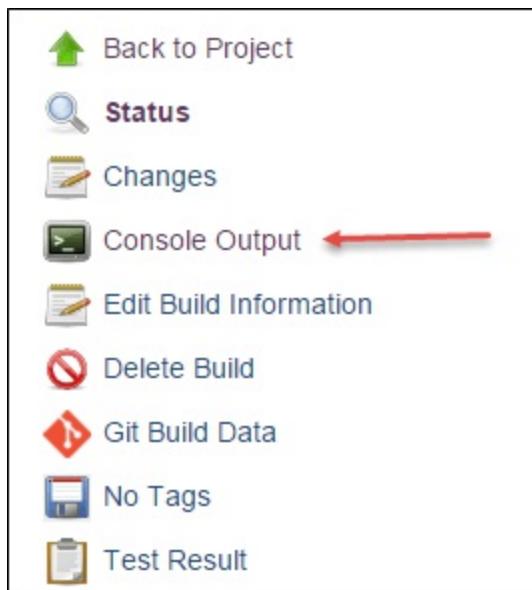


Step 17) It will show the Build, click on build Number or the build

date.



Step 18) once we click on build number below screen will appear where we can see the console output in this step, click on the console output.



Finally, we can verify that our build is successfully completed/executed.

```
Results :  
  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0  
[JENKINS] Recording test results  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 43.864 s  
[INFO] Finished at: 2016-01-15T16:05:14+05:30  
[INFO] Final Memory: 24M/217M  
[INFO] -----
```

Chapter 55: Handling Cookies in Selenium WebDriver

A HTTP cookie is comprised of information about the user and their preferences. It stores information using a key-value pair. It is a small piece of data sent from Web Application and stored in Web Browser, while the user is browsing that website.

[Click here to learn about cookie testing.](#)

Selenium Query Commands for cookies

In Selenium Webdriver, we can query and interact with cookies with below built-in method:

```
driver.manage().getCookies();    // Return The List of all Cookies  
driver.manage().getCookieNamed(arg0); //Return specific cookie according to name  
driver.manage().addCookie(arg0);   //Create and add the cookie  
driver.manage().deleteCookie(arg0); // Delete specific cookie  
driver.manage().deleteCookieNamed(arg0); // Delete specific cookie according Name  
driver.manage().deleteAllCookies(); // Delete all cookies
```

Why Handle Cookies in Selenium?

Each cookie is associated with a name, value, domain, path, expiry, and the status of whether it is secure or not. In order to validate a

client, a server parses all of these values in a cookie.

When Testing a web application using selenium web driver, you may need to create, update or delete a cookie.

For example, when automating Online Shopping Application, you many need to automate test scenarios like place order, View Cart, Payment Information, order confirmation, etc.

If cookies are not stored, you will need to perform login action every time before you execute above listed test scenarios. This will increase your coding effort and execution time.

The solution is to store cookies in a File. Later, retrieve the values of cookie from this file and add to it your current browser session. As a result, you can skip the login steps in every Test Case because your driver session has this information in it.

The application server now treats your browser session as authenticated and directly takes you to your requested URL.

Demo: Cookie handling in Selenium.

We will use <http://demo.avactis.com> for our demo purpose.

This will be a 2 step process.

Step 1) Login into application and store the authentication cookie generated.

Step 2) Used the stored cookie, to again login into application without using userid and password.

Step 1) Storing cookie information.

```
package CookieExample;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.util.Set;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.Cookie;

public class cookieRead{

    public static void main(String[] args)
    {
        WebDriver driver;

System.setProperty("webdriver.chrome.driver","G://chromedriver.exe");
        driver=new ChromeDriver();

driver.get("http://demo.guru99.com/test/cookie/selenium_aut.php");
    }

    // Input Email id and Password If you are already
    Register

    driver.findElement(By.name("username")).sendKeys("abc123");

    driver.findElement(By.name("password")).sendKeys("123xyz");
        driver.findElement(By.name("submit")).click();

    // create file named Cookies to store Login Information
    File file = new File("Cookies.data");
    try
    {
        // Delete old file if exists
        file.delete();
    }
}
```

```
        file.createNewFile();
        FileWriter fileWrite = new FileWriter(file);
        BufferedWriter Bwrite = new
BufferedWriter(fileWrite);
        // loop for getting the cookie information

        // loop for getting the cookie information
for(Cookie ck : driver.manage().getCookies())
{
    Bwrite.write((ck.getName()+"."+ck.getValue()+"."+ck.getDomain()+
";"+ck.getPath()+"."+ck.getExpiry()+"."+ck.isSecure()));
    Bwrite.newLine();
}
Bwrite.close();
fileWrite.close();

    }
catch(Exception ex)
{
    ex.printStackTrace();
}
}
```

Code Explanation:

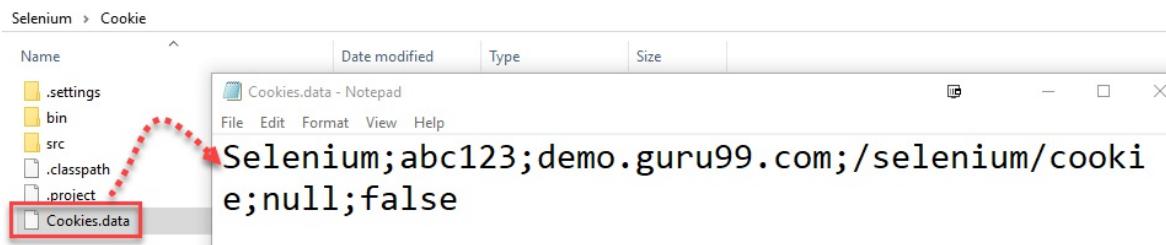
- Create WebDriver instance
 - We visit the website using the
driver.get("http://demo.guru99.com/test/cookie/selenium_aut.pl")
 - Login into the Application
 - Read the cookie information using

```
driver.manage().getCookies();
```

- Store the cookie information using `FileWriter` Class to write streams of characters and `BufferedWriter` to write the text into a file to create into a file `Cookies.data`
 - "Cookies.data" file stores all cookies information along with

"Name, Value, Domain, Path". We can retrieve this information and login into the application without entering the login credentials.

- Once you run above code the Cookie.data file is created into the project folder structure as shown in below screen. Open the Cookie.data file, you can see login credential of the AUT is saved in the format of Cookie, see below-highlighted screen



Step 2) Using stored cookie to login into the application.

Now, we will access the cookie generated in step 1 and use the cookie generated to authenticate our session in the application

```
package CookieExample;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.Date;
import java.util StringTokenizer;
import org.openqa.selenium.Cookie;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class CookieWrite
{
    public static void main(String[] args){
        WebDriver driver;
```

```
System.setProperty("webdriver.chrome.driver", "G://chromedriver.exe");
        driver=new ChromeDriver();
    try{

        File file = new File("Cookies.data");
        FileReader fileReader = new FileReader(file);
        BufferedReader Buffreader = new
BufferedReader(fileReader);
        String strline;
        while((strline=Buffreader.readLine())!=null){
            StringTokenizer token = new
StringTokenizer(strline,";");
            while(token.hasMoreTokens()){
                String name = token.nextToken();
                String value = token.nextToken();
                String domain = token.nextToken();
                String path = token.nextToken();
                Date expiry = null;

                String val;
                if(!(val=token.nextToken()).equals("null")){
                    {
                        expiry = new Date(val);
                    }
                    Boolean isSecure = new Boolean(token.nextToken()).
booleanValue();
                    Cookie ck = new
Cookie(name,value,domain,path,expiry,isSecure);
                    System.out.println(ck);
                    driver.manage().addCookie(ck); // This will add the
stored cookie to your current session
                }
            }
        }catch(Exception ex){
            ex.printStackTrace();
        }

driver.get("http://demo.guru99.com/test/cookie/selenium_aut.php"
);
}
```

OUTPUT: You are taken directly to the login success screen without entering the input user id and password

NOTE: Use hard refresh in case you see the login page after executing the above script.

Conclusion

Thus, you can avoid entering the username and password on the server validating them again and again for each test with the help of Selenium Webdriver, and thereby saves a lot of time.

Chapter 56: Using SoapUI with Selenium

SoapUI is the most popular open source functional Testing tool for Api Testing. It provides complete Test coverage and supports all standard protocols and technologies.

What is SOAP?

SOAP is a simple XML-based protocol. It allows applications to exchange information over HTTP. It uses Web services description language(WSDL) language for communication. Other applications can also interact with web services using WSDL interface.

What is SOAPUI?

SOAPUI is an open source cross-platform web service testing tool. The SOAPUI-Pro has extra functionality for companies dealing with critical web services. Web services play a significant role in Internet applications.

Selenium

- **Selenium:** - It is a test tool to automate browsers across many platforms.
- **Selenium Webdriver:** - It makes direct calls to the browsers. It uses browser's native support for automation.

Selenium with SoapUI

The simplest and easiest way to integrate Selenium with Soapui is to use Groovy. SoapUI extensively supports Groovy.

Groovy is an object-oriented scripting language. Groovy includes all the Java libraries. So all Java related keywords and functions can be used in the groovy script directly. It integrates with JVM (Java Virtual Machine).

Pre-requisites for using Selenium with SoapUI

- Download Groovy SDK:
- Install Java SDK
- Install Selenium
- Install SoapUI Pro

Call the SoapUI Testcase runner in Selenium.

The below code will be used to call SoapUI testcase. It will set the properties of city and corresponding zip codes. When the code is executed, it will get the value of cities and zip codes. Also, display the failure count that does not match with the corresponding city and zip code. This code will run in Selenium.

Note: "usePropertyFileFlag=true" here instead of using a static property file to store zip code and city. The information of zip code and city will pass at runtime dynamically by setProjectProperties() method.

Instructions to run the code.

- Start up SoapUI
- Start a new test case
- Add a new groovy step.
- Copy paste the sample code into the step.
- Click on Play.
- You can see Firefox starting up and navigating to Google. After that, you can see SoapUI log entries.
- Code runs using Junit

```

@when("<I use the weather service to get the weather
information")
    public void
i_use_the_weather_service_to_get_the_information() {
    Set<Entry<String, string>> set =
zipAndCities.entrySet();
    while (iterator.hasNext()) {
        Entry<String, String> entry = iterator.next();
        String zipCode = entry.getKey();
        String city = entry.getValue();
        String[] prop =
{"usePropertyFileFlag=true", "zipCode=" +zipCode, "city=" +city};

        try{
            SoapUITestCaseRunner soapUITestCaseRunner = new
SoapUITestCaseRunner();

soapUITestCaseRunner.setProjectFile("src/test/resources/WeathersS
oapTest-soapui-project.xml");
            soapUITestCaseRunner.setProjectProperties(prop);
            soapUITestCaseRunner.setTestSuite("TestSuite1");
            soapUITestCaseRunner.setTestCase("TestCase1");
            soapUITestCaseRunner.run();

        } catch (Exception e) {
            System.err.println("checking" + zipCode + "
failed!");
            failureCount++;
            zipCodes.append(zipCode + " [" + city +"] ");
            e.printStackTrace();
        }finally{

```

```

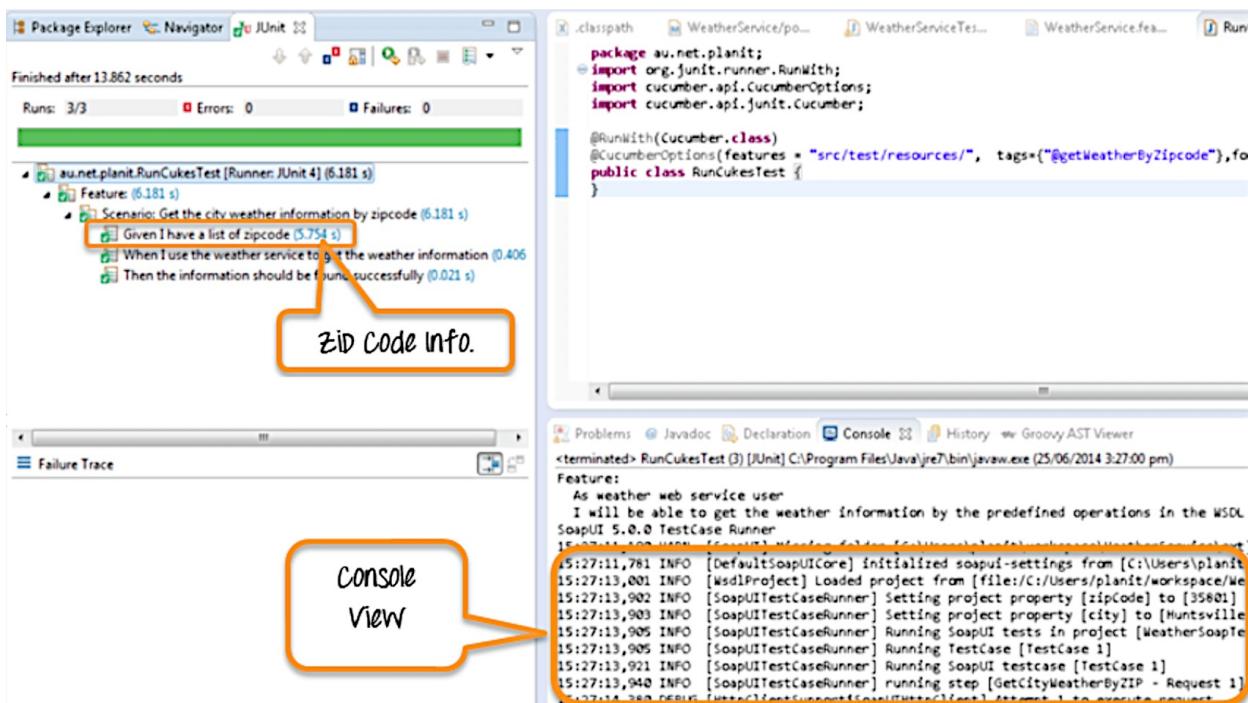
        totalCount++;
    }
}
}
}

```

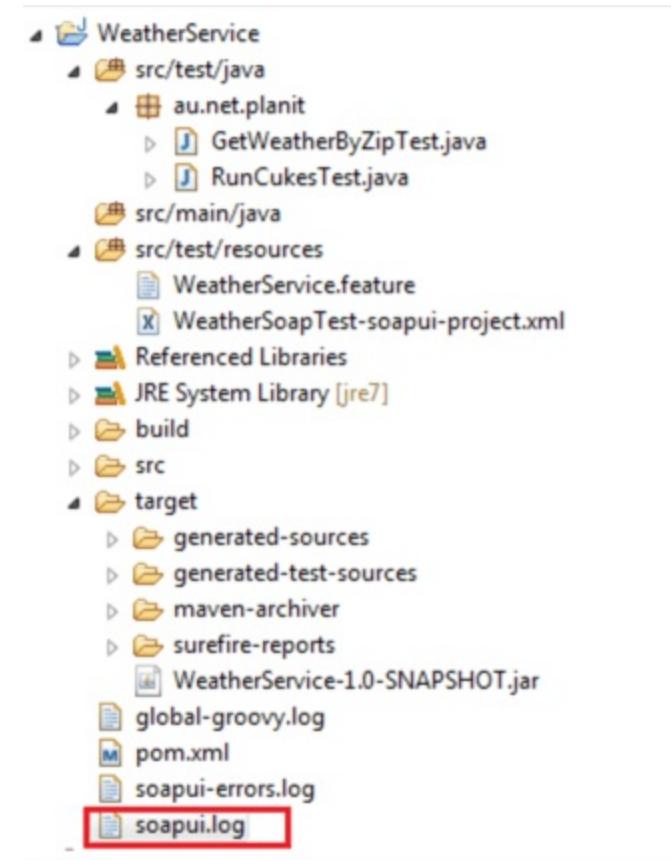
The console view enables us to have a glance at all the test cases executed. You will find a list of zip codes, cities fetched and passed into the SoapUI Test Case 1.

View SoapUI Log file

Log files record every action occurred in the operating system or software application. To view, SoapUI log files. Go to the main directory and you will see a file name "soapui.log."



In SoapUI, the log file is located in the bin folder in the installation directory. E.g C:\Program files\SmartBear\soapUI-Pro-4.0.1\bin



When you open this log file by clicking on it, it will look similar to the screenshot below.

```

2014-06-25 16:36:16,142 WARN [SoapUI] Missing folder [C:\Users\planit\workspace\WeatherService\ext] for external libraries
2014-06-25 16:36:17,581 INFO [DefaultSoapUICore] initialized soapui-settings from [C:\Users\planit\soapui-settings.xml]
2014-06-25 16:36:21,526 INFO [WsdlProject] Loaded project from [file:/C:/Users/planit/workspace/WeatherService/src/test/resources/WeatherSoap
2014-06-25 16:36:23,488 INFO [SoapUITestCaseRunner] Setting project property [zipCode] to [35801]
2014-06-25 16:36:23,489 INFO [SoapUITestCaseRunner] Setting project property [city] to [Huntsville]
2014-06-25 16:36:23,492 INFO [SoapUITestCaseRunner] Running SoapUI tests in project [WeatherSoapTest]
2014-06-25 16:36:23,493 INFO [SoapUITestCaseRunner] Running TestCase [TestCase 1]
2014-06-25 16:36:23,533 INFO [SoapUITestCaseRunner] Running SoapUI testcase [TestCase 1]
2014-06-25 16:36:23,578 INFO [SoapUITestCaseRunner] running step [GetCityWeatherByZIP - Request 1]
2014-06-25 16:36:26,562 INFO [SoapUITestCaseRunner] Assertion [SOAP Response] has status VALID
2014-06-25 16:36:26,563 INFO [SoapUITestCaseRunner] Assertion [Match content of [ResponseText]] has status VALID
2014-06-25 16:36:26,563 INFO [SoapUITestCaseRunner] Assertion [Match content of [City]] has status VALID
2014-06-25 16:36:26,564 INFO [SoapUITestCaseRunner] Finished running SoapUI testcase [TestCase 1], time taken: 2928ms, status: FINISHED
2014-06-25 16:36:26,565 INFO [SoapUITestCaseRunner] TestCase [TestCase 1] finished with status [FINISHED] in 2928ms
2014-06-25 16:36:26,578 INFO [WsdlProject] Loaded project from [file:/C:/Users/planit/workspace/WeatherService/src/test/resources/WeatherSoap
2014-06-25 16:36:26,604 INFO [SoapUITestCaseRunner] Setting project property [zipCode] to [72201]
2014-06-25 16:36:26,605 INFO [SoapUITestCaseRunner] Setting project property [city] to [Little Rock]
2014-06-25 16:36:26,606 INFO [SoapUITestCaseRunner] Running SoapUI tests in project [WeatherSoapTest]
2014-06-25 16:36:26,606 INFO [SoapUITestCaseRunner] Running TestCase [TestCase 1]
2014-06-25 16:36:26,607 INFO [SoapUITestCaseRunner] Running SoapUI testcase [TestCase 1]
2014-06-25 16:36:26,608 INFO [SoapUITestCaseRunner] running step [GetCityWeatherByZIP - Request 1]
2014-06-25 16:36:26,906 INFO [SoapUITestCaseRunner] Assertion [SOAP Response] has status VALID
2014-06-25 16:36:26,906 INFO [SoapUITestCaseRunner] Assertion [Match content of [ResponseText]] has status VALID
2014-06-25 16:36:26,906 INFO [SoapUITestCaseRunner] Assertion [Match content of [City]] has status VALID
2014-06-25 16:36:26,907 INFO [SoapUITestCaseRunner] Finished running SoapUI testcase [TestCase 1], time taken: 297ms, status: FINISHED
2014-06-25 16:36:26,907 INFO [WsdlProject] TestCase [TestCase 1] finished with status [FINISHED] in 297ms
2014-06-25 16:36:26,925 INFO [SoapUITestCaseRunner] Loaded project from [file:/C:/Users/planit/workspace/WeatherService/src/test/resources/WeatherSoap
2014-06-25 16:36:26,943 INFO [SoapUITestCaseRunner] Setting project property [zipCode] to [99501]
2014-06-25 16:36:26,944 INFO [SoapUITestCaseRunner] Setting project property [city] to [Anchorage]

```

Summary

- Soap is simple XML-based protocol. It allows the exchange of information over HTTP.
- SoapUI is an open source cross-platform web service testing tool.
- Selenium is a suite of test tools to automate browsers across many platforms.
- Selenium Webdriver makes direct calls to the browsers. It uses browsers native support for automation.
- Selenium integrates with SoapUI using Groovy.

Chapter 57: XSLT Report in Selenium

The test report is the most important feature of the Selenium framework.

In Selenium, Testng provides its default reporting system. To enhance the reporting feature further XSLT Report is helpful. It also has more user-friendly UI and detail description for the test suite result.

What is XSLT

XSLT stands for **Extensible Stylesheet Language Transformations**.

XSLT is a language for transforming XML documents into other XML documents (XHTML) that are used by a browser.

With XSLT, we can customize the output file. This can be done by adding/removing attributes and elements in the XML file. This help to interpret result quickly. All browser support XSLT. It uses XPath to navigate through elements and attributes in XML documents.

Below are the most popularly used XSL element in programming:

<xsl:stylesheet> It defines that this document is an XSLT style sheet document.

<xsl:if> is used to put a conditional test against the content of the XML file.

`<xsl:template>` is used to build templates.

`<xsl:apply-templates>` is used to apply templates to elements.

`<xsl:choose>` is used in conjunction with `<xsl: otherwise >` and `<xsl:when >` to express multiple conditions.

`<xsl:for-each>` is used to select every XML element of a specified node.

`<xsl:value-of>` is used to extract the value of a selected node.

`<xsl:sort>` is used to sort the output.

Pre-requisite to generate XSLT report

Following is the pre-requisite to generate XSLT report.

- 1) ANT build tool should be install (Its necessary to install ANT for XSLT reporting feature). ANT is used to compile the source code and creating the build. It is also very much extensible. Refer this link for steps to download and install ANT.
- 2) XSLT package downloaded.
- 3) Selenium script that should be executed by TestNG.

We will discuss XSLT report in Selenium Webdriver during this tutorial.

Generate XSLT Report in Selenium

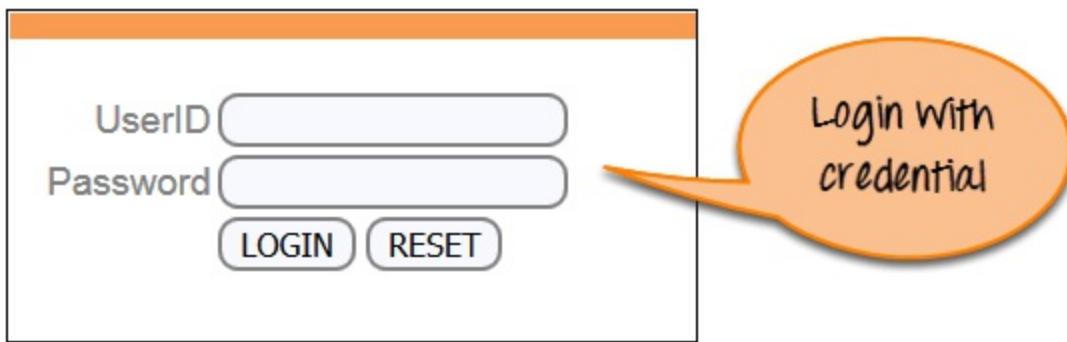
In this scenario, we will use Guru99 demo site to illustrate Generate XSLT report.

Scenario: You will automate and generate XSLT report for the following scenario

- Launch the web browser
- Launch the Firefox and open the site "http://demo.guru99.com/V4/ "



- Login to the application.



- Log out from the application.



Now we will generate XSLT report in selenium as given in below steps.

Step 1: For the above-mentioned scenario. Create and execute the Selenium script for Guru99 demo site.

```

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

public class Testing {
    WebDriver driver= new FirefoxDriver();

    @Test(priority=1)
    public void Login()
    {
        //Launching the Site.
        driver.get("http://demo.guru99.com/V4/");

        //Login to Guru99

        driver.findElement(By.name("uid")).sendKeys("mngr34926");

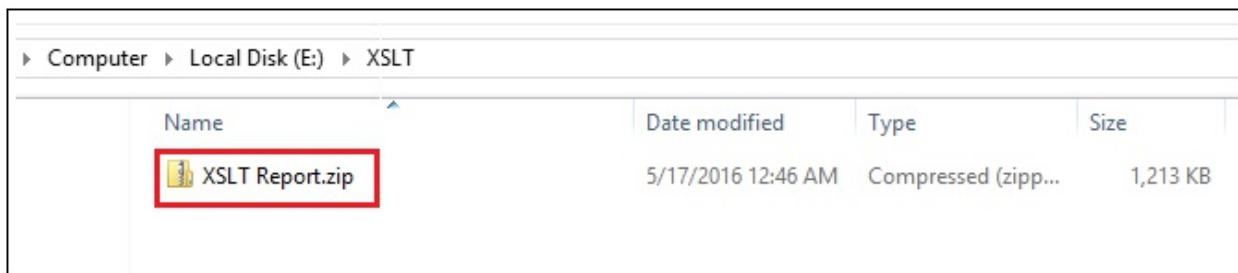
        driver.findElement(By.name("password")).sendKeys("amUpenu");
        driver.findElement(By.name("btnLogin")).click();
        //Verifying the manager home page
        Assert.assertEquals(driver.getTitle(),"Guru99 Bank
Manager HomePage" );
    }
}

```

```
@Test(priority=2)
public void verifytitle()
{
    //Verifying the title of the home page
    Assert.assertEquals(driver.getTitle(),"Guru99 Bank
Manager HomePage" );
}

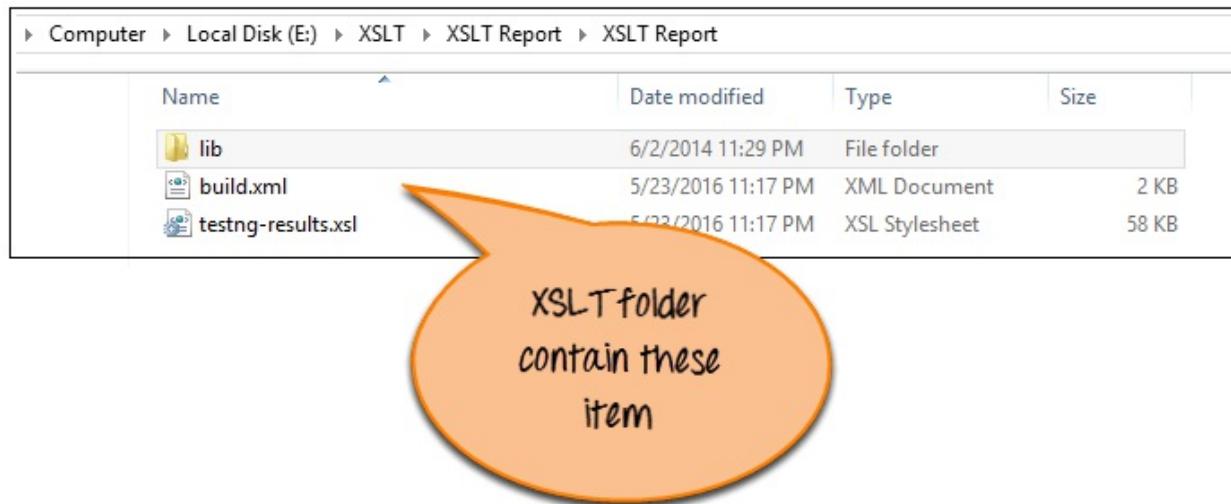
@Test(priority=3)
public void Logout()
{
    driver.findElement(By.linkText("Log out")).click();
    Alert alert=driver.switchTo().alert();
    alert.accept();
    //Verifying the title of the logout page
    Assert.assertEquals(driver.getTitle(),"Guru99 Bank Home
Page" );
}
}
```

Step 2): Download the XSLT report package from this link:

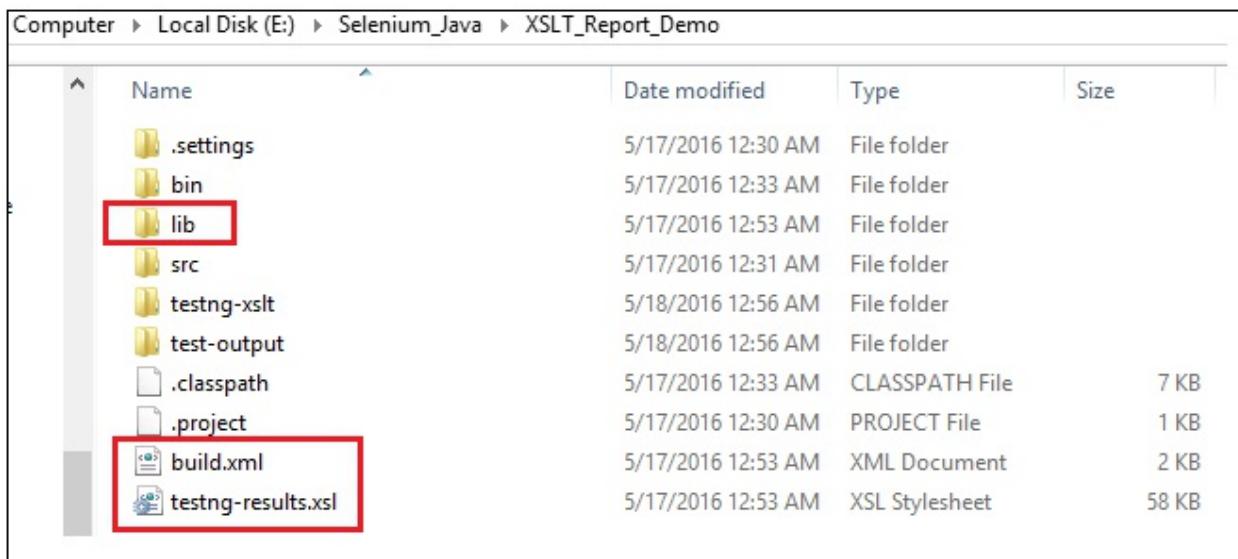


Unzip the above folder you will get below items:

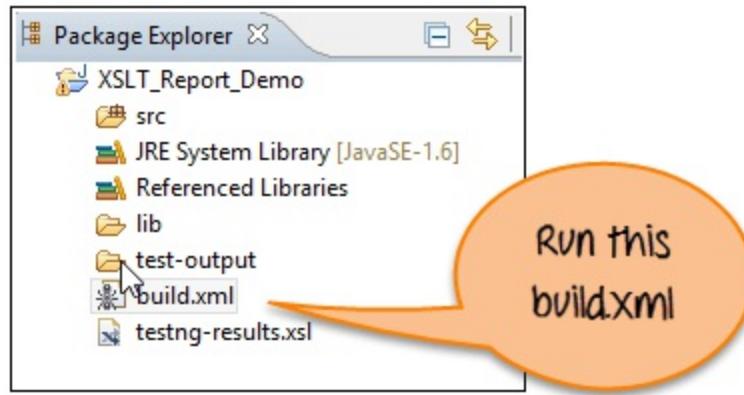
- build.xml
- testng-results.xsl



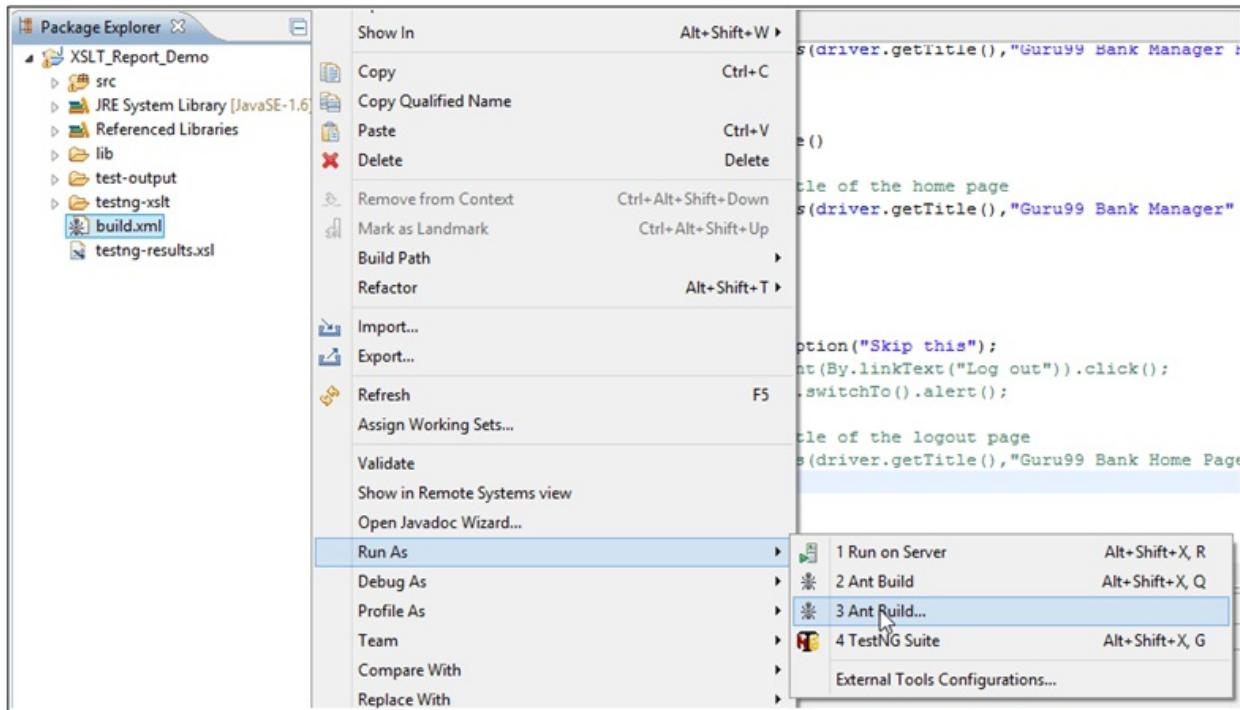
Step 3): Unzip the folder and copy all files and paste at the project home directory as shown in below screen.



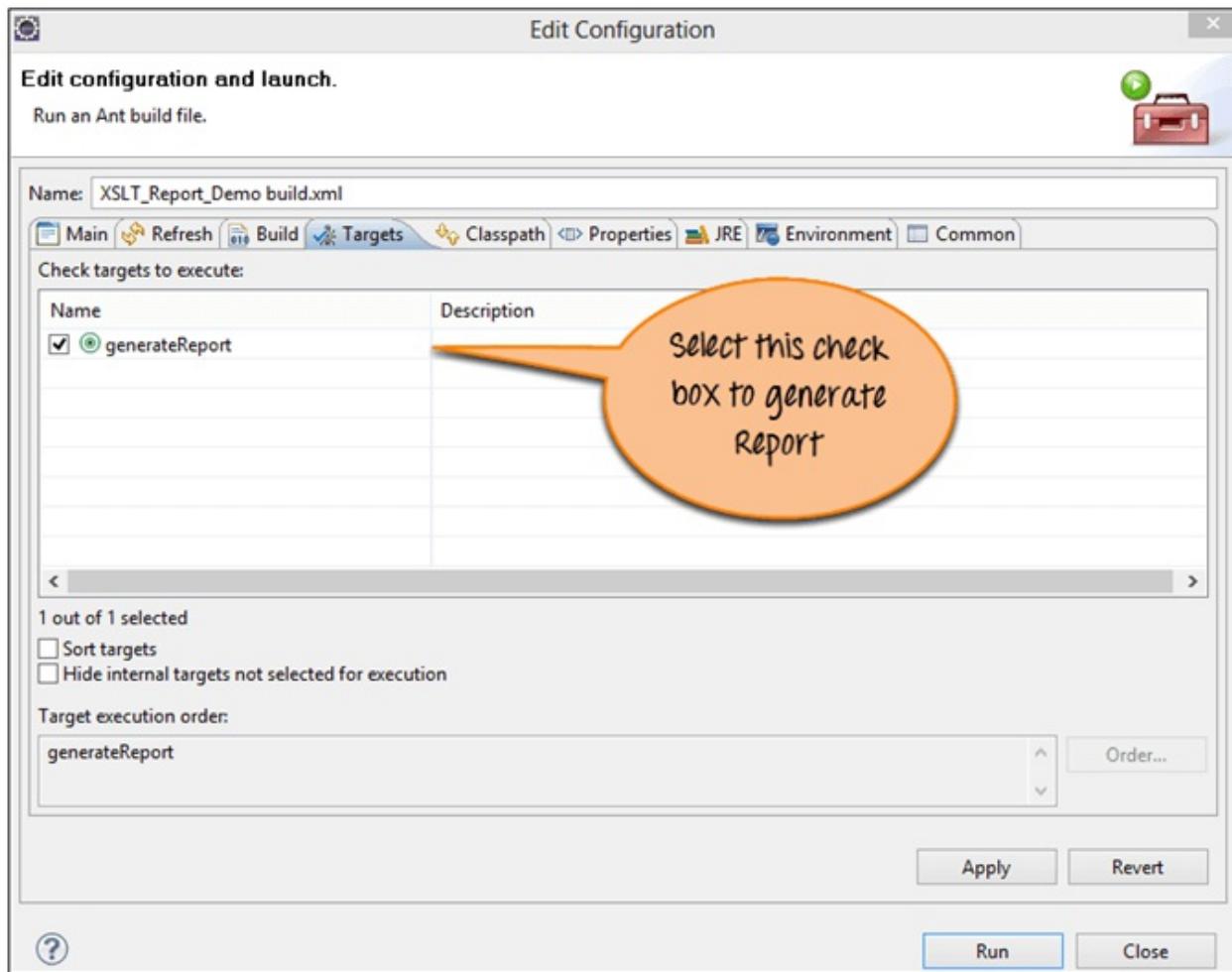
Step 4): In this step run the build.xml file from eclipse as shown below:



Right click on the build.xml then click on run as Ant build.



Then a new window opens. Now select option 'generateReport'.



Click on Run button. It should generate the report.

Verifying XSLT Report

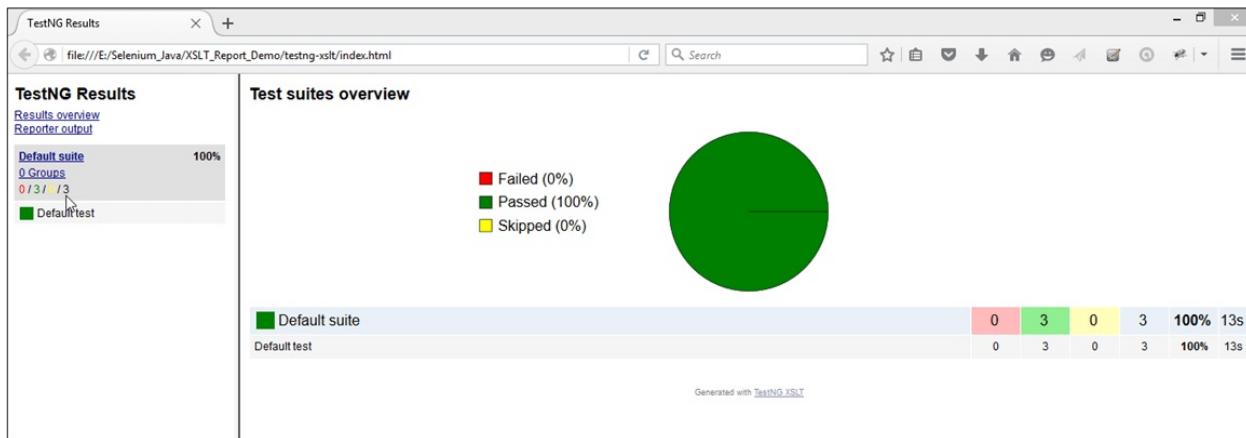
Once build is successful and moved to project home directory. You will find the **testng-xslt** folder.

Name	Date modified	Type	Size
.settings	5/17/2016 12:30 AM	File folder	
bin	5/17/2016 12:33 AM	File folder	
lib	5/17/2016 12:53 AM	File folder	
src	5/17/2016 12:31 AM	File folder	
testng-xslt	5/18/2016 12:34 AM	File folder	
test-output	5/17/2016 12:59 AM	File folder	
.classpath	5/17/2016 12:33 AM	CLASSPATH File	7 KB
.project	5/17/2016 12:30 AM	PROJECT File	1 KB
build.xml	5/17/2016 12:53 AM	XML Document	2 KB
testng-results.xsl	5/17/2016 12:53 AM	XSL Stylesheet	58 KB

Inside this folder you will find **index.html** file as shown below:

Name	Date modified	Type	Size
Default suite.html	5/18/2016 12:34 AM	HTML File	13 KB
Default suite_Default test.html	5/18/2016 12:34 AM	HTML File	13 KB
Default suite_groups.html	5/18/2016 12:34 AM	HTML File	2 KB
index.html	5/18/2016 12:34 AM	HTML File	1 KB
main.js	5/18/2016 12:34 AM	JavaScript File	7 KB
navigation.html	5/18/2016 12:34 AM	HTML File	3 KB
overview.html	5/18/2016 12:34 AM	HTML File	4 KB
overview-chart.svg	5/18/2016 12:34 AM	SVG Document	2 KB
reporterOutput.html	5/18/2016 12:34 AM	HTML File	1 KB
style.css	5/18/2016 12:34 AM	Cascading Style S...	3 KB

Now open this HTML file in any browser like Firefox or Chrome, which support javascript. You will find the report as shown in below screen. The pie chart report represents test status more clearly. The filter feature allows the user to filter the result as per the set criteria.



You will find the pie chart showing the percentage of passed, failed and skipped test.

To display the result in regular format click on the **Default suite** from the left-hand side of the pane. It should show the details of each test as shown in below screen:



Now we forcefully make a test pass, fail and skip.

To view a report of each type for the test result, we need to do some changes in below methods.

1. **verifytitle()** : In the Assert, we pass the wrong expected page title. When the code is executed, it does not match the expected title. Hence making the test fail.
2. **Logout()** : In this method, we forcefully skip the test by using

skipexception. So that when the code is executed, this method will get skip.

By doing so, we are trying to show XSLT report with the help of pie chart. It will show the test result for a pass, fail and skip test.

```
@Test(priority=2)
public void verifytitle()
{
    //Verifying the title of the home page
    Assert.assertEquals(driver.getTitle(),"Guru99 Bank
Manager" );
}
```



passed the
wrong expected
result

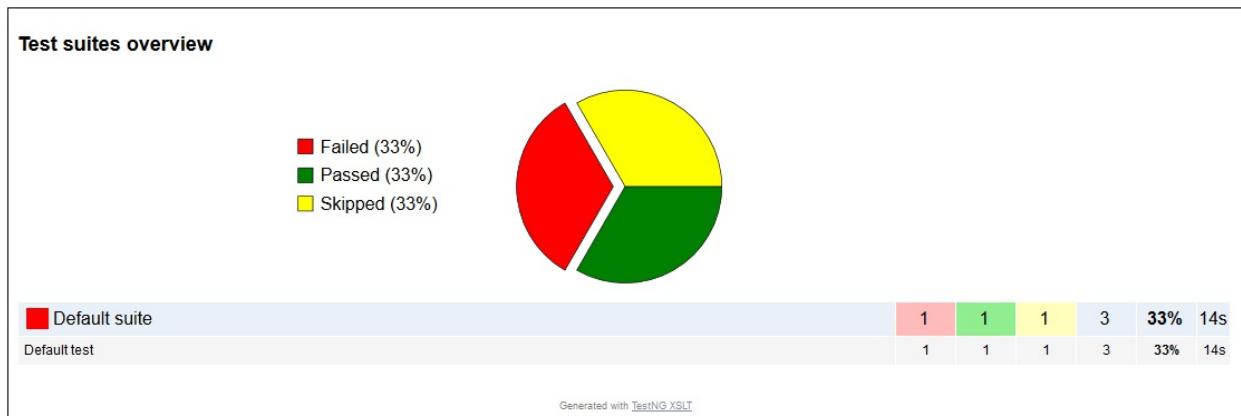
```
@Test(priority=3)
public void Logout()
{
    throw new SkipException("Skip this");
}
```



Replaced the
body with
skipexception

Now we have one test for each type of result status, i.e., pass, fail and skip.

After execution of script and build.xml. Verify the XSLT report as shown in the below screen:



The test report is more user-friendly report and easy to understand. You can also filter the result by selecting the check box in the below screen.

Test case Default test					
<input checked="" type="checkbox"/> Group by class					
<input checked="" type="checkbox"/> All		<input checked="" type="checkbox"/> Failed	<input checked="" type="checkbox"/> Passed	<input checked="" type="checkbox"/> Skipped	<input checked="" type="checkbox"/> Config
Testing					
Name		Started	Duration	Exception	
Login()		00:55:46	14s		
verifytitle()		00:56:01	15 ms	java.lang.AssertionError: expected [Guru99 Bank Manager] but found [Guru99 Bank Manager HomePage]	
Logout()		00:56:01	0 ms	org.testng.SkipException: Skip this	

Note: In the screenshot the 'config' option display the test for which the configuration is done. In big project, there are lots of configuration code. So usually it is used in big projects.

Summary:

XSLT report is required to enhance the TestNG reporting feature in a very user-friendly way.

- XSLT stands for Extensible Stylesheet Language Transformations.
- Download and installation of ANT build refer to given link.
- Generated the XSLT report in selenium and executed the build.xml from eclipse.
- Verify the XSLT report from project folder.
- Verify the XSLT report of each type of result status.

Chapter 58: Firefox Profile - Selenium WebDriver

Firefox profile is the collection of settings, customization, add-ons and other personalization settings that can be done on the Firefox Browser. You can customize Firefox profile to suit your Selenium automation requirement.

Also, Firefox or any other browser handles the SSL certificates settings. So automating them makes a lot of sense along with the test execution code.

In short a profile is a user's personal settings. When you want to run a reliable automation on a Firefox browser, it is recommended to make a separate profile.

Location of your profile folder in the disk

Firefox profile is just like different users using Firefox. Firefox saves personal information such as bookmarks, passwords, and user preferences which can be edited, deleted or created using the program manager.



Location of profile is as follows

- For windows 7 > /AppData/MozillaFirefoxProfile_name.default
- For Linux > /.mozilla/firefox/profile_name.default/
- For Mac OS X >
~/Library/ApplicationSupport/Firefox/Profiles/profile_name.def

In order to run a successful Selenium Test, a Firefox profile should be -

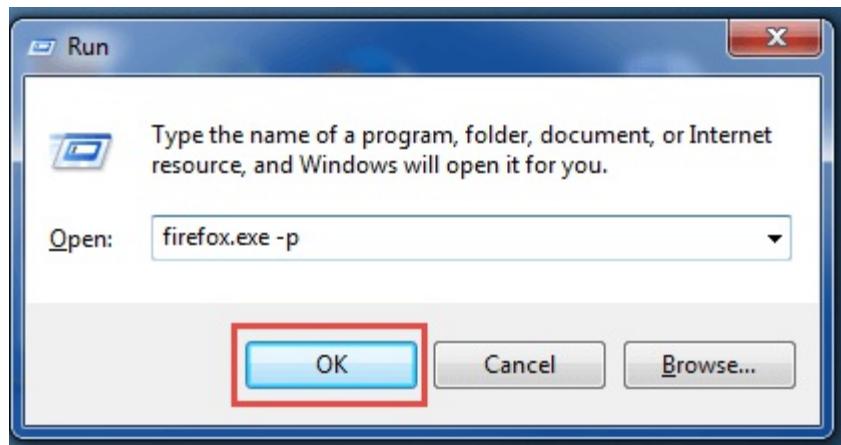
- Easy to load
- Proxy settings if required
- Other user-specific settings based on automation needs

How to create a Firefox profile

Let see step by step how to create a Firefox profile.

Step 1) First of all close the Firefox if open.

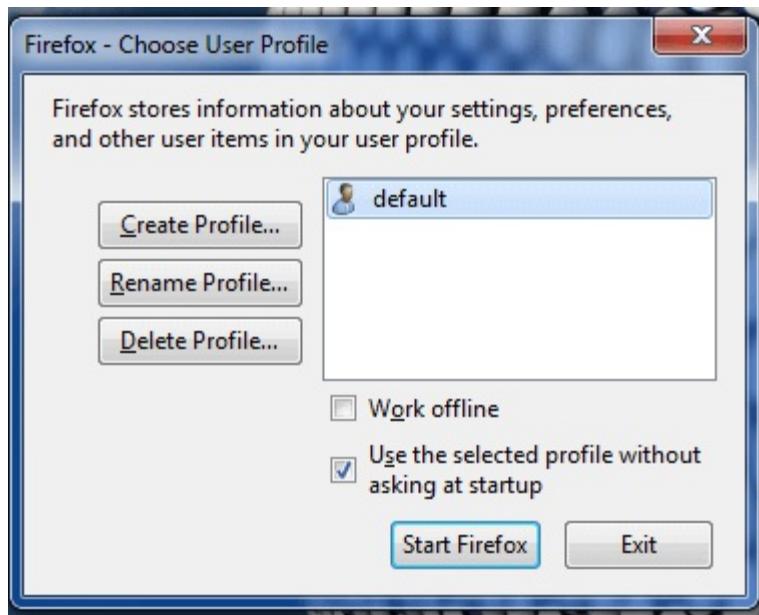
Step 2) Open Run (windows key + R) and type firefox.exe –p and click OK



Note: If it doesn't open you can try using full path enclosed in quotes.

- On 32 bit- Windows: "C:Program FilesMozilla Firefox.exe" -p
- On 64 bit : Windows: "C:Program Files(x86)Mozilla Firefox.exe"
-p

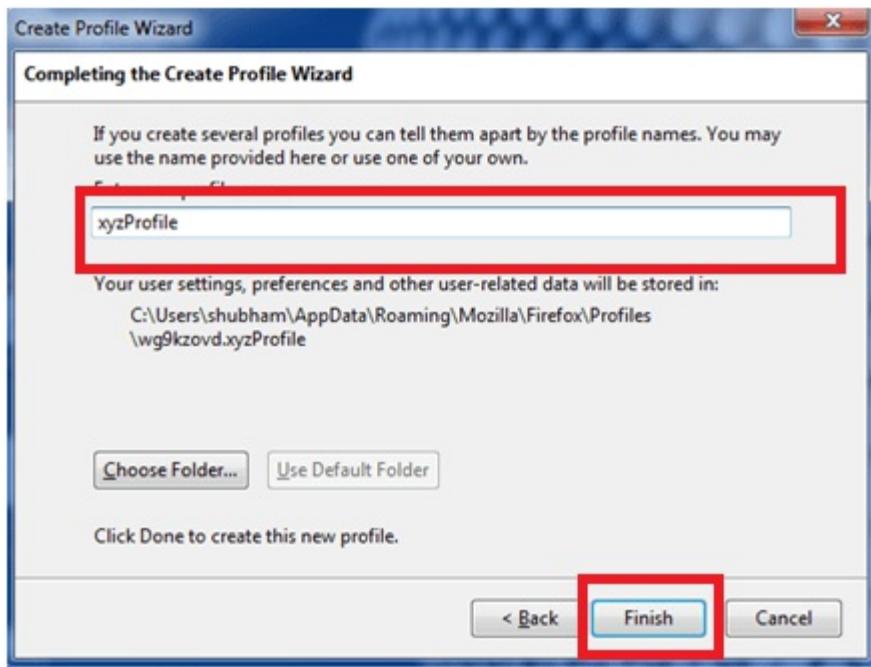
Step 3) A dialogue box will open named Firefox – choose user profile



Step 4) Select option "Create Profile" from the window, and a wizard will open. Click on next



Step 5) Give your profile name which you want to create and click on finish button



Now your profile is ready you can select your profile and open Firefox.

You will notice that the new Firefox window will not show any of your Bookmarks and Favorite icons.

Note: The last selected profile, will load automatically at next Firefox launch. You will need to restart profile manager if you wish to change profiles.

Automation Script for Selenium

To access newly created Firefox profile in Selenium Webdriver software test, we need to use webdrivers inbuilt class 'profilesIni' and its method getProfile as shown below.

Selenium code for the profile

This is a code to implement a profile, which can be embedded in the selenium code.

```
ProfilesIni profile = new ProfilesIni();
```

```
// this will create an object for the Firefox profile
```

```
FirefoxProfile myprofile = profile.getProfile("xyzProfile");
```

```
// this will Initialize the Firefox driver
```

```
WebDriver driver = new FirefoxDriver(myprofile)
```

Let see the implementation of this code in following examples.

Firefox Profile Example 1

```

1 // import the package
2 import java.io.File;
3 import java.util.concurrent.TimeUnit;
4 import org.openqa.selenium.WebDriver;
5 import org.openqa.selenium.firefox.FirefoxDriver;
6 import org.openqa.selenium.firefox.FirefoxProfile;
7 import org.openqa.selenium.firefox.internal.ProfilesIni;
8 public class FirefoxProfile {
9     public static void main(String[] args) {
10         ProfilesIni profile = new ProfilesIni();
11         FirefoxProfile myprofile = profile.getProfile("xyzProfile");
12         // Initialize Firefox driver
13         WebDriver driver = new FirefoxDriver(myprofile);
14         //Maximize browser window
15         driver.manage().window().maximize();
16         //Go to URL which you want to navigate
17         driver.get("http://www.google.com");
18         //Set timeout for 5 seconds so that the page may load properly within that time
19         driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
20         //close firefox browser
21         driver.close();
22     }
23 }
24 }
```

EXPLANATION FOR THE CODE:

Below is the explanation of code line by line.

- **Code line 2-7:** First of all we need to import the package required to run the selenium code.
- **Code line 8:** Make a public class "FirefoxProfile."
- **Code line 9:** Make an object (you need to have basic knowledge of oops concepts).
- **Code line 10-11:** We need to initialize Firefox profile with the object of myprofile .
- **Code line 13:** Create object for Firefox
- **Code line 15:** Maximize window.
- **Code line 17:** Driver.get use to navigate to given URL .
- **Code line 19:** Set timeout is used to wait for some time so that browser may load the page before proceeding to next page.
- **Code line 21:** Close Firefox.

Let's see one more example.

Firefox Profile Example 2

```

1 import java.io.File;
2 import java.util.concurrent.TimeUnit;
3 import org.openqa.selenium.WebDriver;
4 import org.openqa.selenium.firefox.FirefoxDriver;
5 import org.openqa.selenium.firefox.FirefoxProfile;
6 import org.openqa.selenium.firefox.internal.ProfilesIni;
7
8 public class FirefoxProfile2{
9 public static void main(String[] args) {
10
11 // Create object for FirefoxProfile
12 FirefoxProfile myprofile=newFirefoxProfile (newFile("\c:users\AppData\MozillaFirefoxProfile_name.default "));
13 // Initialize Firefox driver
14 WebDriver driver = new FirefoxDriver(myprofile);
15 //Maximize browser window
16 driver.manage().window().maximize();
17 //Go to URL
18 driver.get("http://www.google.com");
19 //Set timeout
20 driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
21 //close firefox browser
22 driver.close();
23 }
```

Explanation for the code:

Below is the explanation of code line by line.

- **Code line 1-6:** First of all we need to import the package required to run the selenium code.
- **Code line 8:** Make a public class FirefoxProfile 2 .
- **Code line 12:** Make the object of myprofile by referring to the exact path .
- **Code line 14:** Create object for firefox
- **Code line 16:** Maximize window.
- **Code line 18:** Driver.get use to navigate to given URL .
- **Code line 20:** Set timeout is used to wait for some time so that browser may load the page before proceeding to next page.
- **Code line 22:** Close Firefox.

Summary:

- Automating Firefox profile makes a lot of sense as such they handles SSL certificates settings.
- Firefox profile can be customized to suit your Selenium automation requirement.
- Firefox profile should be such that it should be easy to load and have some user-specific proxy settings to run a good test.
- To access newly created Firefox profile in Selenium Webdriver software test, we need to use webdrivers inbuilt class 'profilesIni' and its method getProfile.

Chapter 59: Breakpoints and Startpoints in Selenium

Breakpoints are used to check the execution of your code. Whenever you implement a breakpoint in your code, the execution will stop right there. This helps you to verify that your code is working as expected. Breakpoints are usually shown in the UI along with the source code.

Breakpoints in Selenium

Breakpoints in Selenium helps in debugging.

There are two methods to set breakpoints,

- In the first method,
 - Right click on the command and select the 'Toggle Breakpoint'. You can also use shortcut key "B" from the keyboard.
 - You can set a breakpoint just before the Test Case you want to examine.
 - After setting breakpoints, click on the Run button to run the test case from the start to the breakpoint.
 - Repeat the same step to deselect the Breakpoint.
- In the second method,
 - Select Menu Bar -> 'Actions' -> select the Toggle Breakpoint. To deselect repeat the same step.

To demonstrate, let us consider the following scenario. Validate 'username' and 'password' when clicked on the 'Sign in' button.

Methods to implement breakpoints in Selenium

First Method:

Step 1) Launch Firefox and Selenium IDE.

Step 2) Type the Base URL as --->

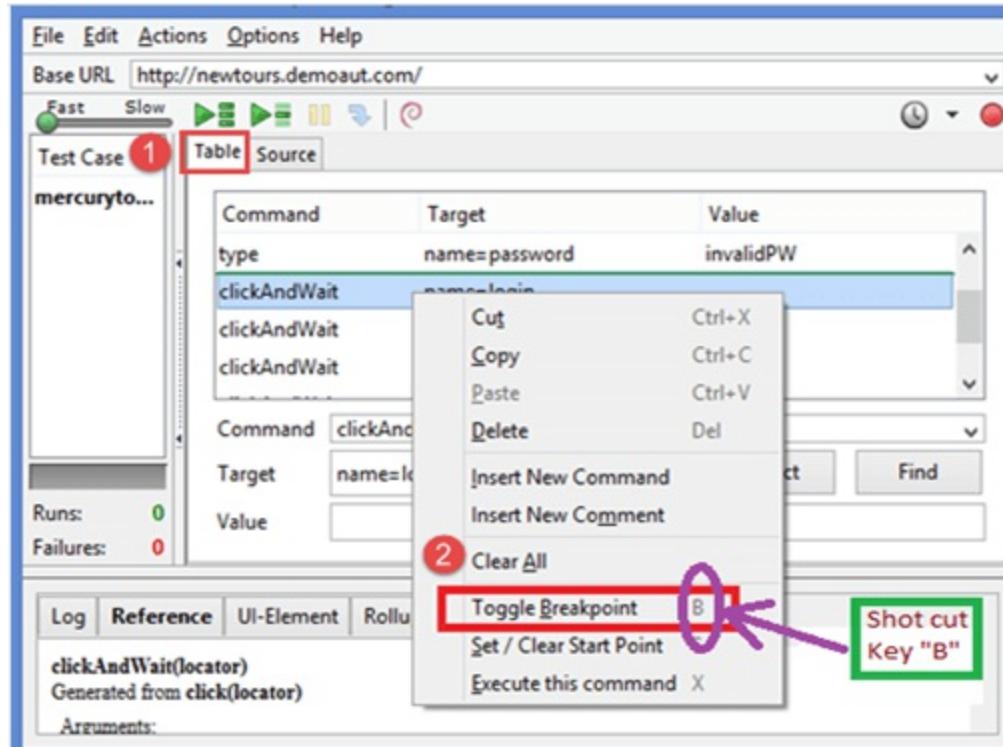
http://newtours.demoaut.com/

Step 3) Click on the Record button (marked in the red box in the screenshot below).

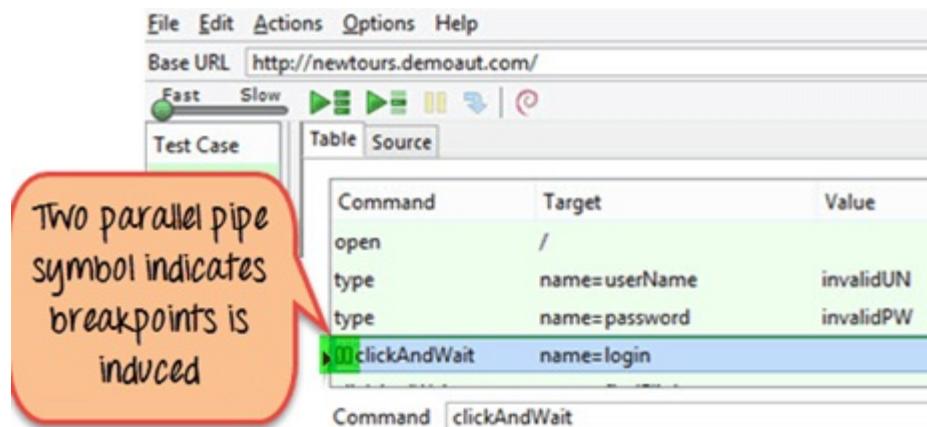


Step 4) In this step,

1. Under the tab "Table" right click on the command ("clickandwait")
2. Under the "Command" column select the 'Toggle Breakpoint'. You can also use the shortcut key "B" from the keyboard.



When you toggle breakpoint, it will open another window as shown below. You will see two yellow pipes mark, in front of "clickandwait", under 'Command' column.



It indicates two things,

- The yellow pipe shows that the test case was paused at that point. So when you click the 'Run' button, the execution starts from the beginning of the test case to this point. After that, one need to

start executing manually.

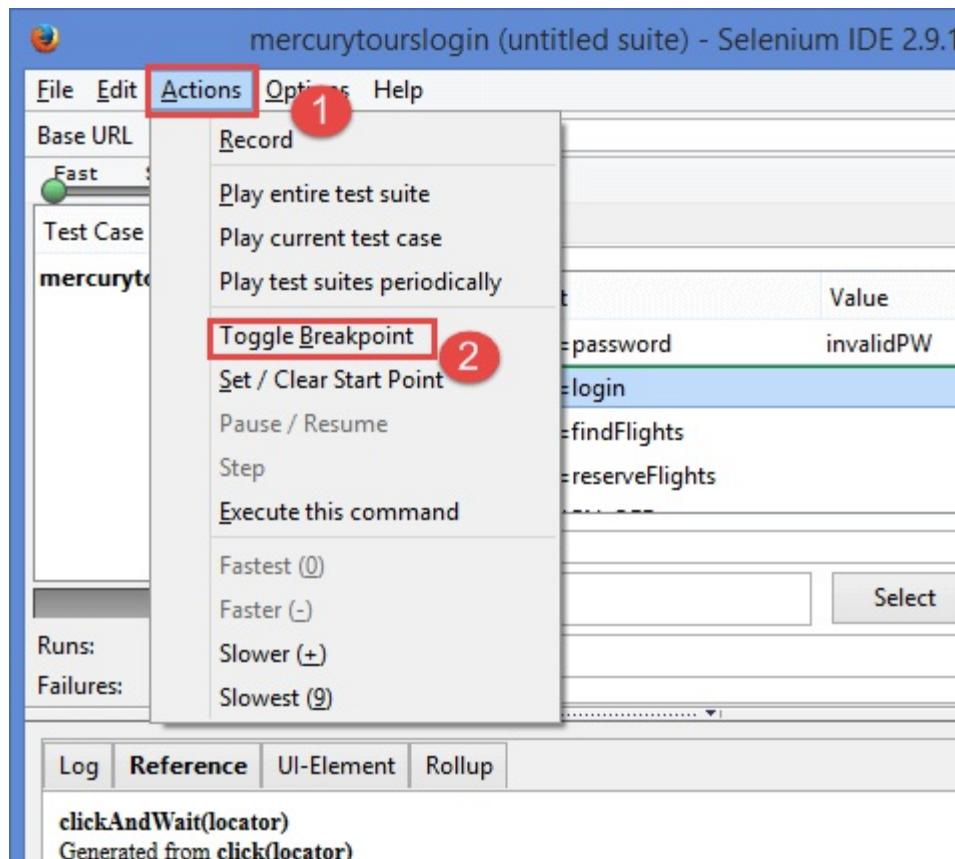
- A Test Script can have multiple breakpoints which can be set in the same manner as shown above.

Second Method:

Step 1) Follow the above steps 1, 2 & 3 mentioned in the First method.

Step 2) In this step,

1. Click on option 'Actions' from Menu bar and
2. Click on the option "Toggle Breakpoint".



This is all about the breakpoints in Selenium.

Start Point in Selenium

In Selenium, Start Point indicates the point from where the execution should begin. Start Point can be used when you want to run the testscript from the middle of the code or a breakpoint.

To understand this, let us take an example of the login scenario. Suppose, if you have to login into the website and perform series of tests and then try to debug one of those tests.

In this case, you have to login once and then re-run your tests as you are developing them. You can set the Start Points after login function. So everytime you perform a new test it will begin executing after the login function.

Start Point can be selected by two methods:

1. Right click on any command under the 'Command' column in selenium IDE. Select the option 'Set/clear Start Point'. You can also use the shortcut key 'S' from the keyboard to mark the start point (shown as a green triangle in the screen shot). Repeat the same step to deselect the Start Point .
2. Click "Actions" -> 'Set/Clear Start Point'. This will select the Start Point or repeat the same step to deselect them.

Methods to set Start Point in Selenium

Let see the first method with an example,

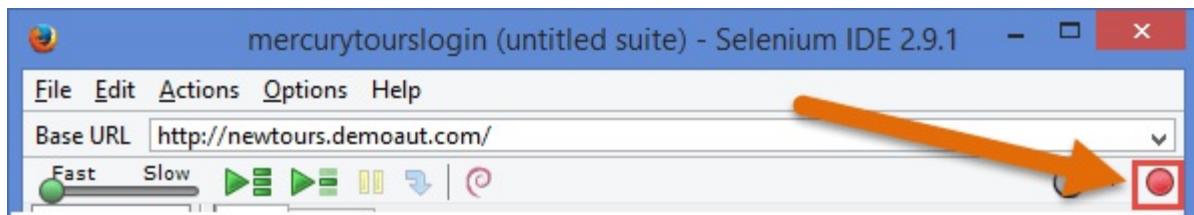
First Method:

Step 1) Launch Firefox and Selenium IDE.

Step 2) Type the Base URL as --->

http://newtours.demoaut.com/

Step 3) Click on the Record button (marked in red rectangle box in the screen shot below).



Step 4) In this step,

1. Under the tab "Table" right click on the command "clickandwait"
2. Now select the option 'Set/Clear Start Point'. You can also use the short key "S" from the keyboard to select the same Set/Clear Start Point .

The screenshot shows the QTP Test Case Editor interface. A context menu is open over a command in the test case table. The menu items are:

- Cut (Ctrl+X)
- Copy (Ctrl+C)
- Paste (Ctrl+V)
- Delete (Del)
- Insert New Command
- Insert New Comment
- Clear All
- Toggle Breakpoint**
- Set / Clear Start Point** (highlighted with a red box and number 2)

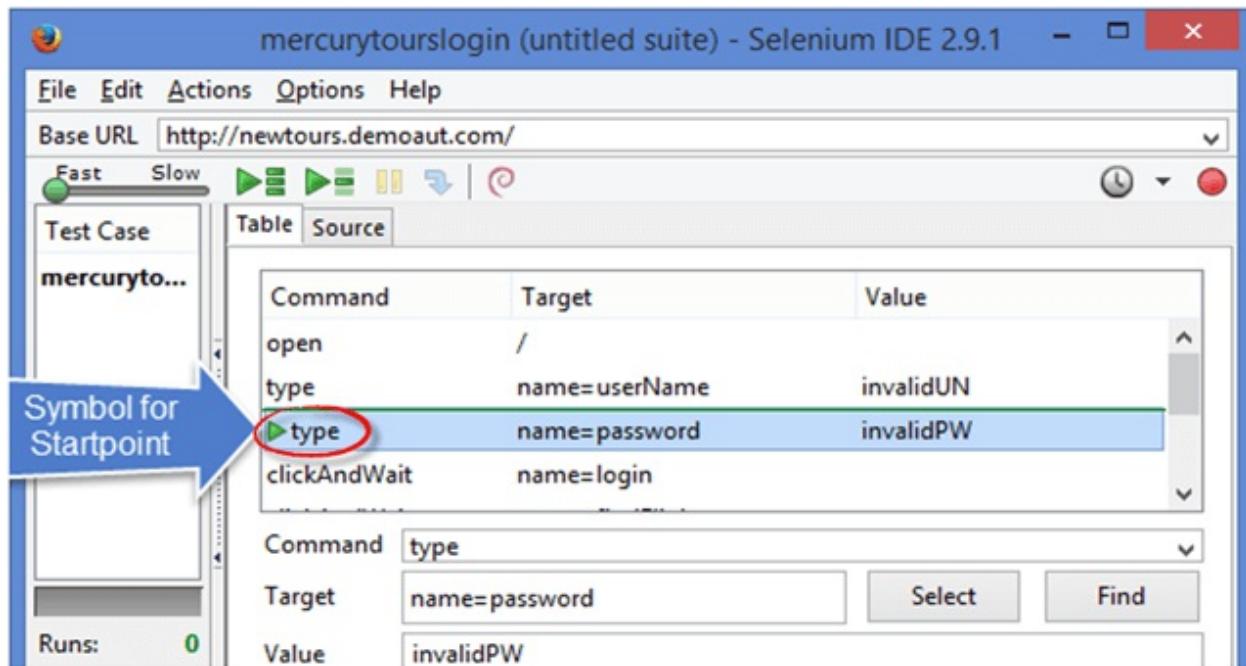
The 'Table' tab is selected in the top navigation bar. The test case table contains the following commands:

Command	Target	Value
open	/	
type	name=userName	invalidUN
type	name=password	invalidPW
clickAndWait	name=login	

Below the table, the 'clickAndWait(locator)' command is shown with its description and arguments.

When you click on Set/ Clear Start Point, it will open another window. In this window, you can see the green Triangle symbol before "type" under 'Command' column.

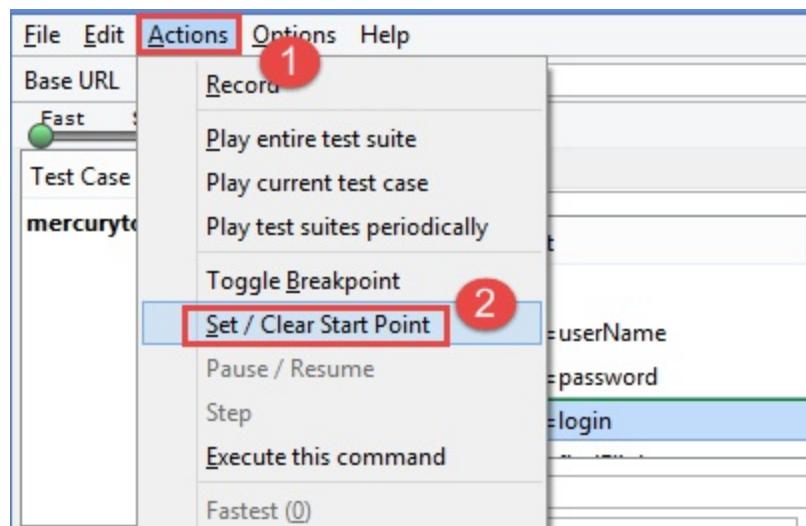
This triangle symbol indicates that the test case starts at this point. So when you click 'Run' button, the execution starts from that point onwards.



Note: There can only be one Start Point in a single test script. Also, Start Point is dependent on the currently displayed page. The execution will fail if the user is on the wrong page.

Second Method:

1. Follow the above steps 1, 2 & 3 mentioned in previous (First) method.
2. Next, follow the following steps,
 1. Click on option 'Actions' from Menu bar and
 2. Click on the option "Set/Clear Start Point ".



Summary

- Breakpoints and Start Point feature help in the debugging process. It helps to start or pause any given test at a particular point of instance. This helps to observe the behavior of the test script.
- In a single test script, there can only be one Start Point as compared to many Breakpoints.

Chapter 60: Top 100 Selenium Interview Questions & Answers

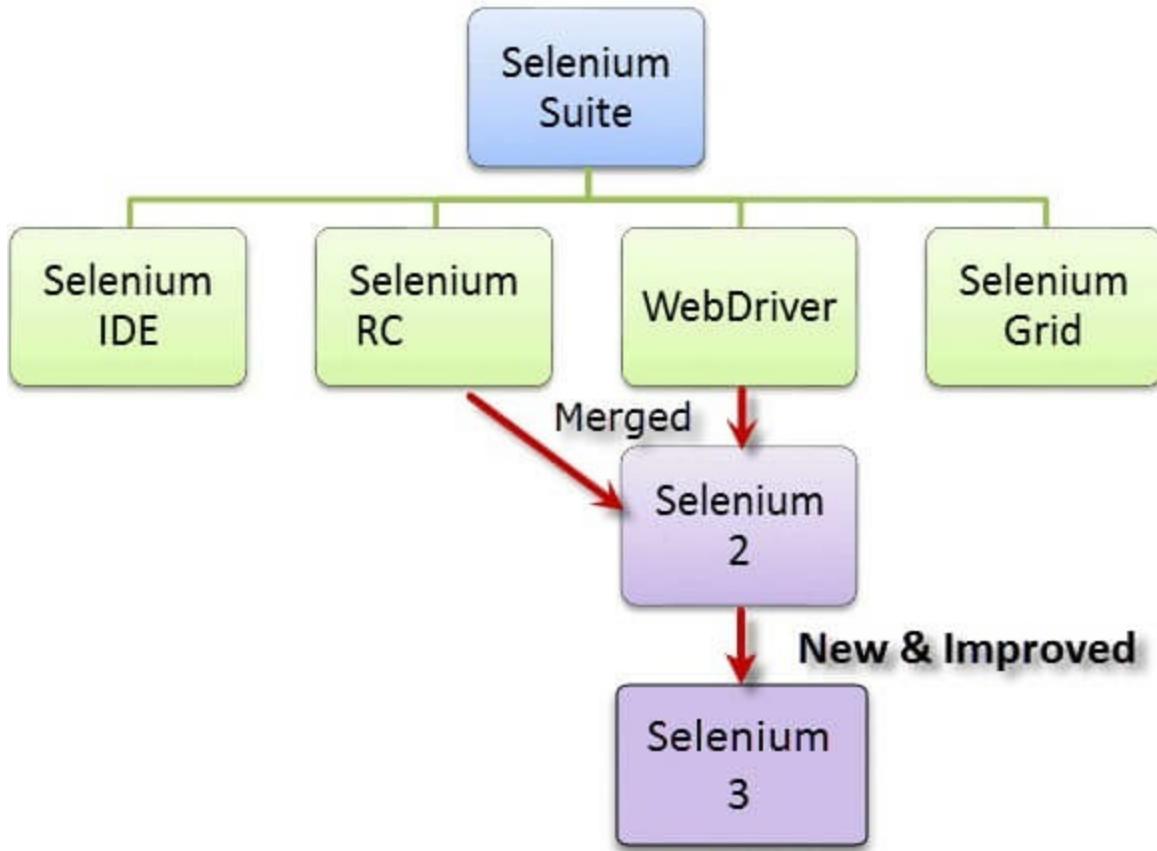
1) What is Selenium and what is composed of?

Selenium is a suite of tools for automated web testing. It is composed of

- **Selenium IDE (Integrated Development Environment)** : It is a tool for recording and playing back. It is a firefox plugin
- **WebDriver and RC**: It provide the APIs for a variety of languages like Java, .NET, PHP, etc. With most of the browsers Webdriver and RC works.
- **Grid**: With the help of Grid you can distribute tests on multiple machines so that test can be run parallel which helps in cutting down the time required for running in browser test suites

2) What is Selenium 2.0 ?

Web Testing tools Selenium RC and WebDriver are consolidated in single tool in Selenium 2.0



3) Mention what is Selenium 3.0?

Selenium 3.0 is the latest version of Selenium. It has released 2 beta versions of selenium 3.0 with few of the below changes:

Here are few new features added to Selenium 3.0

Beta 1 updates	Beta 2 updates (Only for Java)
<ul style="list-style-type: none"> Minimum Java version is now 8+ 	<ul style="list-style-type: none"> System property webdriver.firefox.marionette now forces the server in marionette or legacy firefox driver mode, ignoring any related Desired Capability
<ul style="list-style-type: none"> It will support for Firefox Via Mozilla's geckodriver 	<ul style="list-style-type: none"> Grid fixes NPE's on registration when -browser not specified
<ul style="list-style-type: none"> Support for Edge is provided by MS 	<ul style="list-style-type: none"> Update GeckoDriver –port argument in all bindings

- It now supports Safari on MacOS via Apple's own Safari driver

4) How will you find an element using Selenium?

In Selenium every object or control in a web page is referred as an elements, there are different ways to find an element in a web page they are

- ID
- Name
- Tag
- Attribute
- CSS
- Linktext
- PartialLink Text
- Xpath etc

5) List out the test types that are supported by Selenium?

For web based application testing selenium can be used

The test types can be supported are

- a) Functional, Learn More about Functional Testing.
- b) Regression

For post release validation with continuous integration automation tool could be used

- a) Jenkins

- b) Hudson
- c) Quick Build
- d) CruiseCont

6) Explain what is assertion in Selenium and what are the types of assertion?

Assertion is used as a verification point. It verifies that the state of the application conforms to what is expected. The types of assertion are “assert”, “verify” and “waifFor”.

7) Mention what is the use of X-path?

X-Path is used to find the WebElement in web pages. It is also useful in identifying the dynamic elements.

Refer Complete Guide on XPath

8) Explain the difference between single and double slash in X-path?

Single slash ‘/’

- Single slash (/) start selection from the document node
- It allows you to create ‘absolute’ path expressions

Double Slash ‘//’

- Double slash (//) start selection matching anywhere in the document
- It enables to create ‘relative’ path expressions

9) List out the technical challenges with Selenium?

Technical challenges with Selenium are

- Selenium supports only web based applications
- It does not support the Bitmap comparison
- For any reporting related capabilities have to depend on third party tools
- No vendor support for tool compared to commercial tools like HP UFT
- As there is no object repository concept in Selenium, maintainability of objects becomes difficult

10) What is the difference between type keys and type commands ?

TypeKeys() will trigger JavaScript event in most of the cases whereas .type() won't. Type key populates the value attribute using JavaScript whereas .typekeys() emulates like actual user typing

11) What is the difference between verify and assert commands?

Assert: Assert allows to check whether an element is on the page or not. The test will stop on the step failed, if the asserted element is not available. In other words, the test will terminated at the point where check fails.

Verify: Verify command will check whether the element is on the page, if it is not then the test will carry on executing. In verification, all the commands are going to run guaranteed even if any of test fails.

12) What is JUnit Annotations and what are different types of annotations which are useful ?

In JAVA a special form of syntactic meta-data can be added to Java

source code, this is known as Annotations. Variables, parameters, packages, methods and classes are annotated some of the Junit annotations which can be useful are

- Test
- Before
- After
- Ignore
- BeforeClass
- AfterClass
- RunWith

13) While using click command can you use screen coordinate?

To click on specific part of element, you would need to use clickAT command. ClickAt command accepts element locator and x, y coordinates as arguments-

clickAt (locator, cordString)

14) What are the advantages of Selenium?

- It supports C#, PHP, Java, Perl, Python
- It supports different OS like Windows, Linux and Mac OS
- It has got powerful methods to locate elements (Xpath, DOM , CSS)
- It has highly developer community supported by Google

15) Why testers should opt for Selenium and not QTP?

Selenium is more popular than QTP as

- Selenium is an open source whereas QTP is a commercial tool

- Selenium is used specially for testing web based applications while QTP can be used for testing client server application also
- Selenium supports Firefox, IE, Opera, Safari on operating systems like Windows, Mac, Linux etc. however QTP is limited to Internet Explorer on Windows.
- Selenium supports many programming languages like Ruby, Perl, Python whereas QTP supports only VB script

16) What are the four parameter you have to pass in Selenium?

Four parameters that you have to pass in Selenium are

- Host
- Port Number
- Browser
- URL

17) What is the difference between setSpeed() and sleep() methods?

Both will delay the speed of execution.

Thread.sleep () : It will stop the current (java) thread for the specified period of time. Its done only once

- It takes a single argument in integer format

Ex: `thread.sleep(2000)`- It will wait for 2 seconds

- It waits only once at the command given at sleep

SetSpeed () : For specific amount of time it will stop the execution for every selenium command.

- It takes a single argument in integer format

Ex: `selenium.setSpeed("2000")`- It will wait for 2 seconds

- Runs each command after setSpeed delay by the number of milliseconds mentioned in set Speed

This command is useful for demonstration purpose or if you are using a slow web application

18) What is same origin policy? How you can avoid same origin policy?

The “**Same Origin Policy**” is introduced for security reason, and it ensures that content of your site will never be accessible by a script from another site. As per the policy, any code loaded within the browser can only operate within that website’s domain.

To avoid “Same Origin Policy” proxy injection method is used, in proxy injection mode the Selenium Server acts as a client configured **HTTP proxy**, which sits between the browser and application under test and then masks the AUT under a fictional URL

19) What is heightened privileges browsers?

The purpose of heightened privileges is similar to Proxy Injection, allows websites to do something that are not commonly permitted. The key difference is that the browsers are launched in a special mode called heightened privileges. By using these browser mode, Selenium core can open the AUT directly and also read/write its content without passing the whole AUT through the Selenium RC server.

20) How you can use “submit” a form using Selenium ?

You can use “submit” method on element to submit form-

```
element.submit();
```

Alternatively you can use click method on the element which does form submission

21) What are the features of TestNG and list some of the functionality in TestNG which makes it more effective?

TestNG is a testing framework based on JUnit and NUnit to simplify a broad range of testing needs, from Unit Testing to Integration Testing. And the functionality which makes it efficient testing framework are

- Support for annotations
- Support for data-driven testing
- Flexible test configuration
- Ability to re-execute failed test cases

22) Mention what is the difference between Implicit wait and Explicit wait?

Implicit Wait: Sets a timeout for all successive Web Element searches. For the specified amount of time it will try looking for element again and again before throwing a NoSuchElementException. It waits for elements to show up.

Explicit Wait : It is a one-timer, used for a particular search.

23) Which attribute you should consider throughout the script in frame for “if no frame Id as well as no frame name”?

You can use.....driver.findElements(By.xpath("//iframe"))....

This will return list of frames.

You will need to switch to each and every frame and search for locator which we want.

Then break the loop

24) Explain what is the difference between find elements () and find element () ?

find element ():

It finds the first element within the current page using the given “locating mechanism”. It returns a single WebElement

findElements () : Using the given “locating mechanism” find all the elements within the current page. It returns a list of web elements.

25) Explain what are the JUnits annotation linked with Selenium?

The JUnits annotation linked with Selenium are

- @Before public void method() – It will perform the method () before each test, this method can prepare the test
- @Test public void method() – Annotations @Test identifies that this method is a test method environment
- @After public void method()- To execute a method before this annotation is used, test method must start with test@Before

26) Explain what is Datadriven framework and Keyword driven?

Datadriven framework: In this framework, the test data is separated and kept outside the Test Scripts, while Test Case logic

resides in Test Scripts. Test data is read from the external files (Excel Files) and are loaded into the variables inside the Test Script. Variables are used for both for input values and for verification values.

Keyworddriven framework: The keyword driven frameworks requires the development of data tables and keywords, independent of the test automation. In a keyword driven test, the functionality of the application under test is documented in a table as well as step by step instructions for each test.

27) Explain how you can login into any site if it's showing any authentication popup for password and username?

Pass the username and password with url

- Syntax-`http://username:password@url`
- ex- `http://creyate:tom@www.gmail.com`

28) Explain how to assert text of webpage using selenium 2.0 ?

```
WebElement el = driver.findElement(By.id("ElementID"))
```

```
//get test from element and stored in text variable
```

```
String text = el.getText();
```

```
//assert text from expected
```

```
Assert.assertEquals("Element Text", text);
```

29) Explain what is the difference between Borland Silk and Selenium?

Silk Test Tool	Selenium Test Tool
<ul style="list-style-type: none"> Borland Silk test is not a free testing tool 	<ul style="list-style-type: none"> Selenium is completely free test automation tool
<ul style="list-style-type: none"> Silk test supports only Internet Explorer and Firefox 	<ul style="list-style-type: none"> Selenium supports many browsers like Internet Explorer, Firefox, Safari, Opera and so on
<ul style="list-style-type: none"> Silk test uses test scripting language 	<ul style="list-style-type: none"> Selenium suite has the flexibility to use many languages like Java, Ruby, Perl and so on
<ul style="list-style-type: none"> Silk test can be used for client server applications 	<ul style="list-style-type: none"> Selenium can be used for only web application

30) What is Object Repository ?

An object repository is an essential entity in any UI automations which allows a tester to store all object that will be used in the scripts in one or more centralized locations rather than scattered all over the test scripts.

31) Explain how Selenium Grid works?

Selenium Grid sent the tests to the hub. These tests are redirected to Selenium Webdriver, which launch the browser and run the test. With entire test suite, it allows for running tests in parallel.

32) Can we use Selenium grid for performance testing?

Yes. But not as effectively as a dedicated Performance Testing tool like Loadrunner.

33) List the advantages of Webdriver over Selenium Server?

- If you are using Selenium-WebDriver, you don't need the Selenium Server as it is using totally different technology
- Selenium Server provides Selenium RC functionality which is used for Selenium 1.0 backwards compatibility

- Selenium Web driver makes direct calls to browser using each browsers native support for automation, while Selenium RC requires selenium server to inject Javascript into the browser

34) Mention what are the capabilities of Selenium WebDriver or Selenium 2.0 ?

WebDriver should be used when requiring improvement support for

- Handling multiple frames, pop ups , multiple browser windows and alerts
- Page navigation and drag & drop
- Ajax based UI elements
- Multi browser testing including improved functionality for browser not well supported by Selenium 1.0

35) While injecting capabilities in webdriver to perform tests on a browser which is not supported by a webdriver what is the limitation that one can come across?

Major limitation of injecting capabilities is that “findElement” command may not work as expected.

36) Explain how you can find broken images in a page using Selenium Web driver ?

To find the broken images in a page using Selenium web driver is

- Get XPath and get all the links in the page using tag name
- In the page click on each and every link
- Look for 404/500 in the target page title

37) Explain how you can handle colors in web driver?

To handle colors in web driver you can use

Use getCssValue(argin) function to get the colors by sending ‘color’ string as an argument

38) Using web driver how you can store a value which is text box?

You can use following command to store a value which is text box using web driver

```
driver.findElement(By.id("your Textbox")).sendKeys("your keyword");
```

39) Explain how you can switch between frames?

To switch between frames webdrivers [**driver.switchTo().frame()**] method takes one of the three possible arguments

- A number: It selects the number by its (zero-based) index
- A number or ID: Select a frame by its name or ID
- Previously found WebElement: Using its previously located WebElement select a frame

40) Mention 5 different exceptions you had in Selenium web driver?

The 5 different exceptions you had in Selenium web drivers are

- WebDriverException
- NoAlertPresentException
- NoSuchWindowException
- NoSuchElementException

- TimeoutException

41) Explain using Webdriver how you can perform double click ?

You can perform double click by using

- **Syntax- Actions act = new Actions (driver);**
- **act.doubleClick(webElement);**

42) How will you use Selenium to upload a file ?

You can use “type” command to type in a file input box of upload file. Then, you have to use “Robot” class in JAVA to make file upload work.

43) Which web driver implementation is fastest?

HTMLUnit Driver implementation is fastest, HTMLUnitDriver does not execute tests on browser but plain http request, which is far quick than launching a browser and executing tests

44) Explain how you can handle frames using Selenium 2.0 ?

To bring control on HTML frame you can use “SwitchTo” frame method-

```
driver.switchTo().frame("frameName");
```

To specify a frame you can use index number

```
driver.switchTo().frame("parentFrame.4.frameName");
```

This would bring control on frame named- “frameName” of the 4th sub frame names “parentFrame”

45) What is the difference between getWindowhandles() and getWindowhandle() ?

getWindowhandles(): It is used to get the address of all the open browser and its return type is Set<String>

getWindowhandle(): It is used to get the address of the current browser where the control is and return type is string

46) Explain how you can switch back from a frame?

To switch back from a frame use method defaultContent()

Syntax-`driver.switchTo().defaultContent();`

47) List out different types of locators?

Different types of locators are

- By.id()
- By.name()
- By.tagName()
- By.className()
- By.linkText()
- By.partialLinkText()
- By.xpath
- By.cssSelector()

48) What is the command that is used in order to display the values of a variable into the output console or log?

- In order to display a constant string, command can be used is `echo <constant string>`

- If order to display the value of a variable you can use command like echo \${variable name}>>

Above is using PHP. If you are using Java, replace echo with System.out.println

49) Explain how you can use recovery scenario with Selenium?

Recovery scenarios depends upon the programming language you use. If you are using Java then you can use exception handling to overcome same. By using “Try Catch Block” within your Selenium WebDriver Java tests

50) Explain how to iterate through options in test script?

To iterate through options in test script you can loop features of the programming language, for example to type different test data in a text box you can use “for” loop in Java

```
// test data collection in an array
```

```
String[ ] testData = { "test1" , "test2" , "test3" } ;
```

```
// iterate through each test data
```

```
For (string s: test data) { selenium.type ( "elementLocator" , testData ) ;  
}
```

51) How can you prepare customized html report using TestNG in hybrid framework ?

There are three ways

- Junit: With the help of ANT
- TestNG: Using inbuilt default.html to get the HTML report. Also XST reports from ANT, Selenium, Testng combinations
- Using our own customized reports using XSL jar for converting XML content to HTML

52) From your test script how you can create html test report?

To create html test report there are three ways

- TestNG: Using inbuilt default.html to get the HTML report. Also XLST reports from ANT, Selenium, TestNG combination
- JUnit: With the help of ANT
- Using our own customized reports using XSL jar for converting XML content to HTML

53) Explain how you can insert a break point in Selenium IDE ?

In Selenium IDE to insert a break point

- Select “Toggle break point” by right click on the command in Selenium IDE
- Press “B” on the keyboard and select the command in Selenium IDE
- Multiple break points can be set in Selenium IDE

54) Explain in Selenium IDE how can you debug the tests?

- Insert a break point from the location from where you want to execute test step by step
- Run the test case

- At the given break point execution will be paused
- To continue with the next statement click on the blue button
- Click on the “Run” button to continue executing all the commands at a time

55) What is Selenese and what are the types of Selenese ?

Selenese is a selenium set of command which are used for running the test

There are three types of Selenese

- Actions: It is used for performing the operations and interactions with the target elements
- Assertions: It is used as a check points
- Accessors: It is used for storing the values in a variable

56) Explain what are the limitations of Selenium IDE?

The limitations of Selenium IDE

- Exceptional handling is not present
- Selenium IDE uses only HTML languages
- External databases reading is not possible with IDE
- Reading from the external files like .txt, .xls is not possible
- Conditional or branching statements execution like if,else, select statements is not possible

57) What are the two modes of views in Selenium IDE ?

Either Selenium IDE can be opened as a pop up window or in side bar

58) In selenium IDE what are the element locators that can

be used to locate elements on web page?

In selenium there are mainly 4 locators that are used

- X-path locators
- Css locators
- Html id
- Html name

59) In Selenium IDE how you can generate random numbers and dates for test data ?

In Selenium IDE you can generate random numbers by using Java Script

type

```
css=input#s
```

```
javascript{Math.random()}
```

And for

type

```
css=input#s
```

```
javascript{new Date()}
```

60) How you can convert any Selenium IDE tests from Selenese to another language?

You can use the format option of Selenium IDE to convert tests into another programming language

61) Using Selenium IDE is it possible to get data from a particular html table cell ?

You can use the “storeTable” command

Example store text from cell 0,2 from an html table

storeTable

Css=#table 0.2

textFromCell

62) Explain what can cause a Selenium IDE test to fail?

- When a locator has changed and Selenium IDE cannot locate the element
- When element Selenium IDE waiting to access did not appear on the web page and the operation timed out
- When element Selenium IDE was trying to access was not created

63) Explain how you can debug the tests in Selenium IDE ?

- Insert a break point from the location where you want to execute step by step
- Run the test case
- At the given break point execution will be paused
- To continues with the next step click on the Blue button
- To run commands at a time click on run button

64) From Selenium IDE how you can execute a single line?

From Selenium IDE single line command can be executed in two ways

- Select “Execute this command” by right clicking on the command in Selenium IDE
- Press “X” key on the keyboard after selecting the command in Selenium IDE

65) In which format does source view shows your script in Selenium IDE ?

In Selenium IDE source view shows your script in XML format

66) Explain how you can insert a start point in Selenium IDE?

In two ways selenium IDE can be set

- Press “S” key on the keyboard and select the command in Selenium IDE
- In Selenium IDE right click on the command and select “Set / Clear Start Point”

67) What if you have written your own element locator and how would you test it?

To test the locator one can use “Find Button” of Selenium IDE, as you click on it, you would see on screen an element being highlighted provided your element locator is right or else an error message will be displayed

68) What is regular expressions? How you can use regular expressions in Selenium ?

A regular expression is a special text string used for describing a search pattern. In Selenium IDE regular expression can be used with

the keyword- **regexp**: as a prefix to the value and patterns needs to be included for the expected values.

69) What are core extension ?

If you want to “extend” the defualt functionality provided by Selenium Function Library , you can create a Core Extension. They are also called “User Extension”. You can even download ready-made Core Extension created by other Selenium enthusiasts.

70) How will you handle working with multiple windows in Selenium ?

We can use the command **selectWindow** to switch between windows. This command uses the title of Windows to identify which window to switch to.

71) How will you verify the specific position of an web element

You can use verifyElementPositionLeft & verifyElementPositionTop. It does a pixel comparison of the position of the element from the Left and Top of page respectively

72) How can you retrieve the message in an alert box ?

You can use the storeAlert command which will fetch the message of the alert pop up and store it in a variable.

73) What is selenium RC (Remote Control)?

Selenium IDE have limitations in terms of browser support and language support. By using Selenium RC limitation can be diminished.

- On different platforms and different web browser for automating web application selenium RC is used with languages like Java, C#, Perl, Python
- Selenium RC is a java based and using any language it can interact with the web application
- Using server you can bypass the restriction and run your automation script running against any web application

74) Why Selenium RC is used?

Selenium IDE does not directly support many functions like condition statements, Iteration, logging and reporting of test results, unexpected error handling and so on as IDE supports only HTML language. To handle such issues Selenium RC is used it supports the language like Perl, Ruby, Python, PHP using these languages we can write the program to achieve the IDE issues.

75) Explain what is the main difference between web-driver and RC ?

The main difference between Selenium RC and Webdriver is that, selenium RC injects javascript function into browsers when the page is loaded. On the other hand, Selenium Webdriver drives the browser using browsers built in support

76) What are the advantages of RC?

Advantages of RC are

- Can read or write data from/ to .xls, .txt, etc
- It can handle dynamic objects and Ajax based UI elements
- Loops and conditions can be used for better performance and

flexibility

- Support many Programming languages and Operating Systems
- For any JAVA script enabled browser Selenium RC can be used

77) Explain what is framework and what are the frameworks available in RC?

A collection of libraries and classes is known as Framework and they are helpful when testers has to automate test cases. NUnit, JUnit, TestNG, Bromine, RSpec, unittest are some of the frameworks available in RC .

78) How can we handle pop-ups in RC ?

To handle pop-ups in RC , using selectWindow method, pop-up window will be selected and windowFocus method will let the control from current window to pop-up windows and perform actions according to script

79) What are the technical limitations while using Selenium RC?

Apart from “same origin policy” restriction from js, Selenium is also restricted from exercising anything that is outside browser.

80) Can we use Selenium RC to drive tests on two different browsers on one operating system without Selenium Grid?

Yes, it is possible when you are not using JAVA testing framework. Instead of using Java testing framework if you are using java client driver of selenium then TestNG allows you to do this. By using “parallel=test” attribute you can set tests to be executed in parallel and can define two different tests, each using different browser.

81) Why to use TestNG with Selenium RC ?

If you want full automation against different server and client platforms, You need a way to invoke the tests from a command line process, reports that tells you what happened and flexibility in how you create your test suites. TestNG gives that flexibility.

82) Explain how you can capture server side log Selenium Server?

To capture server side log in Selenium Server, you can use command

- `java -jar .jar -log selenium.log`

83) Other than the default port 4444 how you can run Selenium Server?

You can run Selenium server on java-jar selenium-server.jar-port other than its default port

84) How Selenium grid hub keeps in touch with RC slave machine?

At predefined time selenium grid hub keeps polling all RC slaves to make sure it is available for testing. The deciding parameter is called “remoteControlPollingIntervalSeconds” and is defined in “grid_configuration.yml”file

85) Using Selenium how can you handle network latency ?

To handle network latency you can use
`driver.manage.pageloadingtime` for network latency

86) To enter values onto text boxes what is the command

that can be used?

To enter values onto text boxes we can use command **sendKeys()**

87) How do you identify an object using selenium?

To identify an object using Selenium you can use

`isElementPresent(String locator)`

`isElementPresent` takes a locator as the argument and if found returns a Boolean

88) In Selenium what are Breakpoints and Startpoints?

- **Breakpoints:** When you implement a breakpoint in your code, the execution will stop right there. This helps you to verify that your code is working as expected.
- **Startpoints:** Startpoint indicates the point from where the execution should begin. Startpoint can be used when you want to run the testscript from the middle of the code or a breakpoint.

89) Mention why to choose Python over Java in Selenium?

Few points that favor Python over Java to use with Selenium is,

- Java programs tend to run slower compared to Python programs.
- Java uses traditional braces to start and ends blocks, while Python uses indentation.
- Java employs static typing, while Python is dynamically typed.
- Python is simpler and more compact compared to Java.

90) Mention what are the challenges in Handling Ajax Call in Selenium Webdriver?

The challenges faced in Handling Ajax Call in Selenium Webdriver are

- Using "pause" command for handling Ajax call is not completely reliable. Long pause time makes the test unacceptably slow and increases the testing time. Instead, "waitforcondition" will be more helpful in testing Ajax applications.
- It is difficult to assess the risk associated with particular Ajax applications
- Given full freedom to developers to modify Ajax application makes the testing process challenging
- Creating automated test request may be difficult for testing tools as such AJAX application often use different encoding or serialization technique to submit POST data.

91) Mention what is IntelliJ?

IntelliJ is an IDE that helps you to write better and faster code for Selenium. IntelliJ can be used in the option to Java bean and Eclipse.

92) Mention in what ways you can customize TestNG report?

You can customize TestNG report in two ways,

- Using ITestListener Interface
- Using IReporter Interface

93) To generate pdf reports mention what Java API is required?

To generate pdf reports, you need Java API IText.

94) Mention what is Listeners in Selenium WebDriver?

In Selenium WebDriver, Listeners "listen" to the event defined in the selenium script and behave accordingly. It allows customizing TestNG reports or logs. There are two main listeners i.e. WebDriver Listeners and TestNG Listeners.

95) Mention what are the types of Listeners in TestNG?

The types of Listeners in TestNG are,

- IAnnotationTransformer
- IAnnotationTransformer2
- IConfigurable
- IConfigurationListener
- IExecutionListener
- IHookable
- IIInvokedMethodListener
- IIInvokedMethodListener2
- IMETHODInterceptor
- IReporter
- ISuiteListener
- ITestListener

96) Mention what is desired capability? How is it useful in terms of Selenium?

The desired capability is a series of key/value pairs that stores the browser properties like browser name, browser version, the path of the browser driver in the system, etc. to determine the behavior of the browser at run time.

For Selenium,

- It can be used to configure the driver instance of Selenium WebDriver.
- When you want to run the test cases on a different browser with different operating systems and versions.

97) For Database Testing in Selenium Webdriver what API is required?

For Database Testing in Selenium Webdriver, you need JDBC (Java Database Connectivity) API. It allows you to execute SQL statements.

98) Mention when to use AutoIT?

Selenium is designed to automate web-based applications on different browsers. But to handle window GUI and non-HTML popups in the application you need AutoIT. know more about How to use AutoIT with Selenium

99) Mention why do you need Session Handling while working with Selenium?

While working with Selenium, you need Session Handling. This is because, during test execution, the Selenium WebDriver has to interact with the browser all the time to execute given commands. At the time of execution, it is also possible that, before current execution completes, someone else starts execution of another script, in the same machine and in the same type of browser. So to avoid such situation you need Session Handling.

100) Mention what are the advantages of Using Git Hub For Selenium?

The advantages of Using Git Hub for Selenium are

- Multiple people when they work on the same project they can update project details and inform other team members simultaneously.
- Jenkins can help you to build the project from the remote repository regularly. This helps you to keep track of failed builds.

Chapter 61: Using Cucumber with Selenium

In this tutorial, you will learn how to integrate Cucumber with Selenium Webdriver.

What is Cucumber?

Cucumber is a testing approach which supports Behavior Driven Development (BDD). It explains the behavior of the application in a simple English text using Gherkin language.

Learn more at - <https://www.guru99.com/cucumber-tutorials.html>

What is Selenium?

Selenium is an automation tool for Functional Testing of the web-based application. Selenium supports different language like java, ruby, python C#, etc.

Learn more at - <https://www.guru99.com/selenium-tutorial.html>

Why use Cucumber with Selenium?

Cucumber and Selenium are two popular technologies.

Most of the organizations use Selenium for functional testing. These organizations which are using Selenium want to integrate Cucumber with selenium as Cucumber makes **it easy to read and to**

understand the application flow.

Cucumber tool is based on the Behavior Driven Development framework that **acts as the bridge** between the following people:

1. Software Engineer and Business Analyst.
2. Manual Tester and Automation Tester.
3. Manual Tester and Developers.

Cucumber also **benefits the client to understand the application code** as it uses Gherkin language which is in Plain Text. Anyone in the organization can understand the behavior of the software. The syntax's of Gherkin is in simple text which is readable and understandable.



Prerequisite for using Cucumber with Selenium.

Before we start integration cucumber with selenium we need the following items:

Selenium jar files:

- Selenium-server-standalone

Can be downloaded at <http://www.seleniumhq.org/download/>

SeleniumHQ
Browser Automation

[edit this page](#) [search selenium:](#)

[Projects](#) [Download](#) [Documentation](#)

[Selenium Downloads](#)
[Previous Releases](#)
[Source Code](#)
[Maven Information](#)

Downloads

Below is where you can find the latest releases of all the Selenium components ([previous releases](#), [source code](#), and additional information for [Maven user](#) build tool).

Selenium Standalone Server

The Selenium Server is needed in order to run Remote Selenium WebDriver, capable of running Selenium RC directly, rather than through emulation WebDriverBackedSelenium interface.

[Download version 3.5.3](#) Click here to download Selenium

[To run Selenium tests exported from](#)
[To use the Selenium Server in a Grid configuration](#)

The Internet Explorer Driver Server

Jar files For Cucumber :

- Cucumber-core
- Cucumber-html
- cobertura code coverage
- Cucumber-java
- Cucumber-junit
- Cucumber-jvm-deps
- Cucumber-reporting
- Hemcrest-core
- Gherkin
- Junit

Can be downloaded at <http://mvnrepository.com/search?q=Cucumber>

You need to search files and download them one by one individually.

For example, we will show you to download one of the jar files, i.e., "Cucumber-core."

Click on the above download link. It redirects to the below site. Now search the particular jar, i.e. 'Cucumber Core' as shown screenshot below:

MVNREPOSITORY

Indexed Artifacts (7.74M)

Found 15112 results

Sort: relevance | popular | newest

1. Cucumber Core
info.cukes » cucumber-core
cucumber-core

Last Release on Jan 16, 2012

2. Cucumber JVM: Core
io.cucumber » cucumber-core
Cucumber JVM: Core

In the next page, Click of the version 1.2.2,

Version	Repository	Usages	Date
%24%257Bcucum...	Central	0	(Jan, 2012)
\$%7Bcucumber-...	Central	0	(Oct, 2013)
1.2.5	Central	43	(Sep, 2016)
1.2.4	Central	40	(Jul, 2015)
1.2.3	Central	9	(Jul, 2015)
1.2.2	Central	26	(Jan, 2015)
1.2.0	Central	13	(Oct, 2014)
1.1.8	Central	17	(Jun, 2014)

An orange callout points to the '1.2.2' row with the text 'Click here to download 1.2.2 ver.'

In the next screen, click on download to get the 'Cucumber Core' jar file.

Home » info.cukes » cucumber-core » 1.2.2

Note: There is a new version for this artifact

New Version	%24%257Bcucumber-jvm.version%257D
-------------	-----------------------------------

Cucumber Core » 1.2.2

cucumber-core

License	MIT
Date	(Jan 13, 2015)
Files	Download (JAR) (207 KB)
Repositories	Central Sonatype Releases
Used By	85 artifacts



Note: For your ease, we have bundled the jar files required to be download from Maven here. With time these jars maybe updated and become incompatible. You are requested to download them using the method illustrated above.

Automation Testing Using Cucumber with Selenium.

Let's study steps to use cucumber with selenium step by step. Here we will cover 3 scenarios:

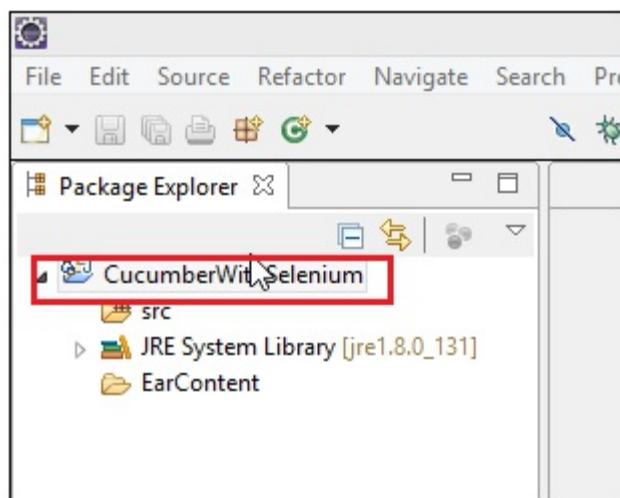
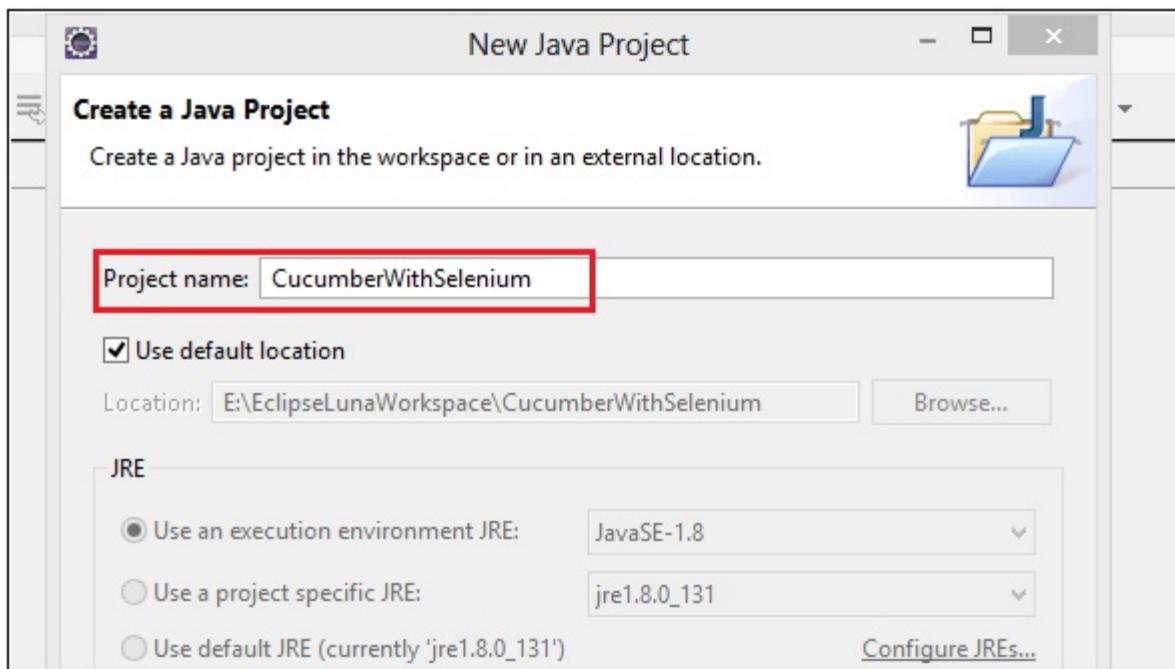
- Scenario 1: Print text in the console.
- Scenario 2: Enter login Credential and reset the value.
- Scenario 3: Enter login Credential on Guru99 & reset the value.
Do this for 3 sets of data.

Scenario 1: Print text in the console.

In this scenario, we just print the text in the console by using Cucumber.

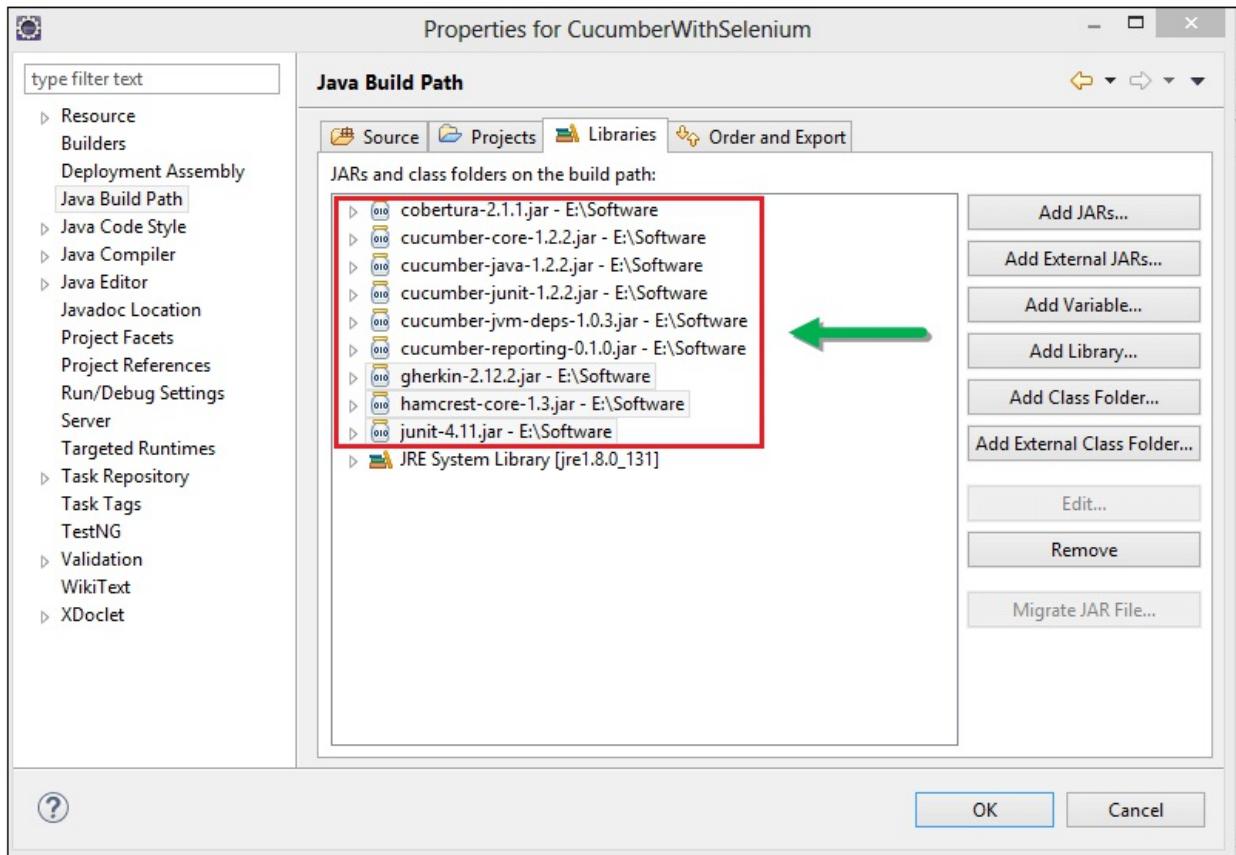
Step 1) Create Project in eclipse.

Create Java project with the name "CucumberWithSelenium" as shown in the below screenshot.



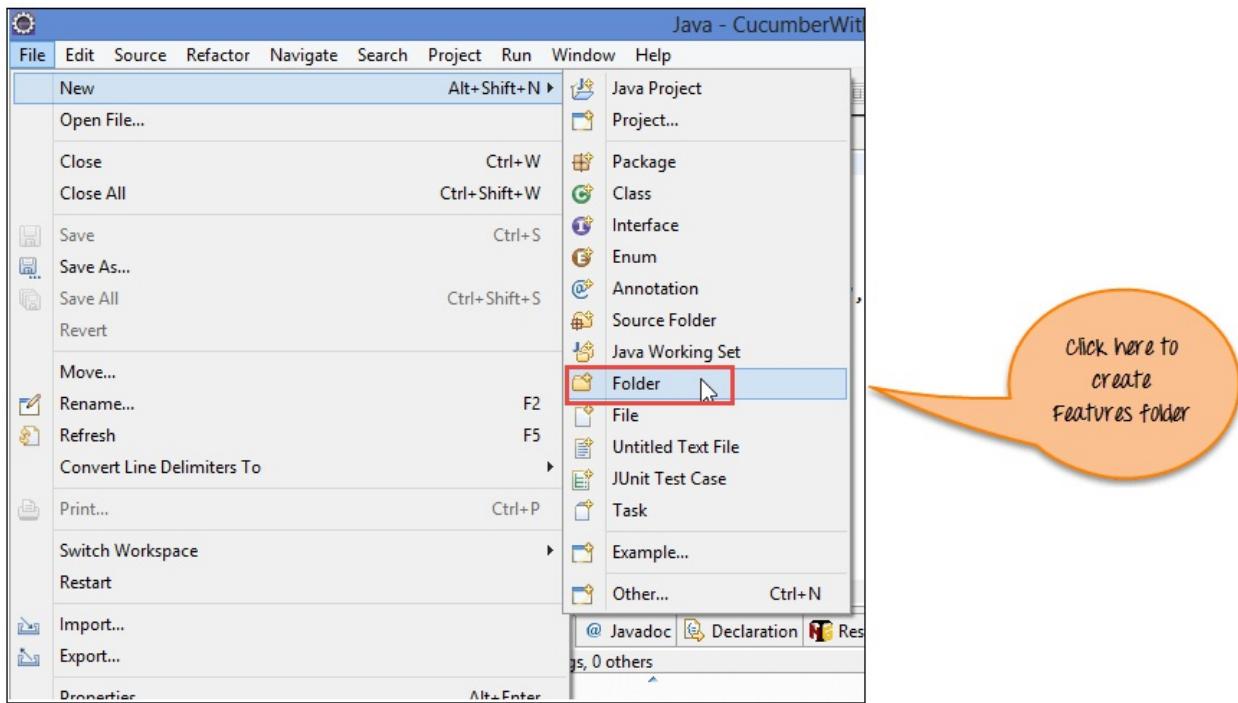
Step 2) Adding Jar files in the project.

Right Click on the Project > Select Properties > Go to Java Build Path.
Add all the libraries downloaded earlier.



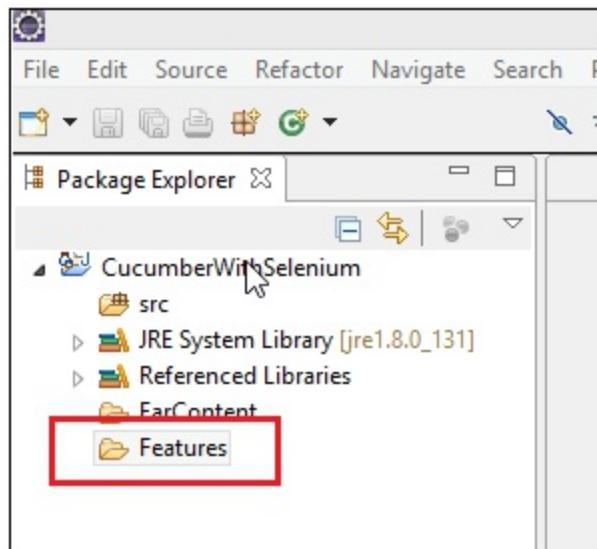
Step 3) Creating feature file

For creating feature file first create features folder as shown below screenshot.

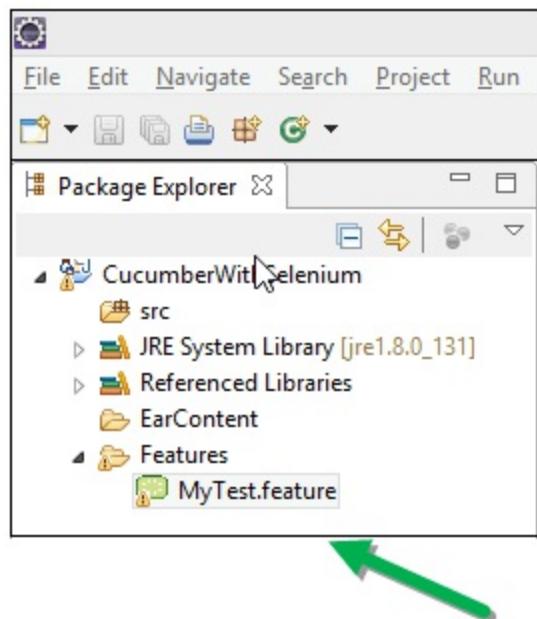


Now Enter Folder name 'Features' and click on 'Finish' Button.





Now, create feature file in the 'Features' folder with the name of "MyTest.feature" - Process is similar to creating a folder



Note: You may need to install the Cucumber Eclipse Plugin for this to work. Goto -- Helps->Install New Software->copy paste the link <http://cucumber.github.io/cucumber-eclipse/update-site/> and install

Step 4) Write scenarios.

Below lines are written in 'MyTest.feature' file using the Gherkin language as shown below:

```
Feature: Reset functionality on login page of Application

Scenario: Verification of Reset button

Given Open the Firefox and launch the application

When Enter the Username and Password

Then Reset the credential
```

Code Explanation

Line 1) In this line we write business functionality.

Line 2) In this line we write a scenario to test.

Line 3) In this line we define the precondition.

Line 4) In this line we define the action we need to perform.

Line 4) In this line we define the expected outcome or result.

Step 5) Writing selenium testrunner script.

Here we create 'TestRunner' package and then 'Runner.java' class file under it.

```
package TestRunner;

import org.junit.runner.RunWith;
```

```

import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)
@CucumberOptions(features="Features",glue={"StepDefinition"})
public class Runner
{
}

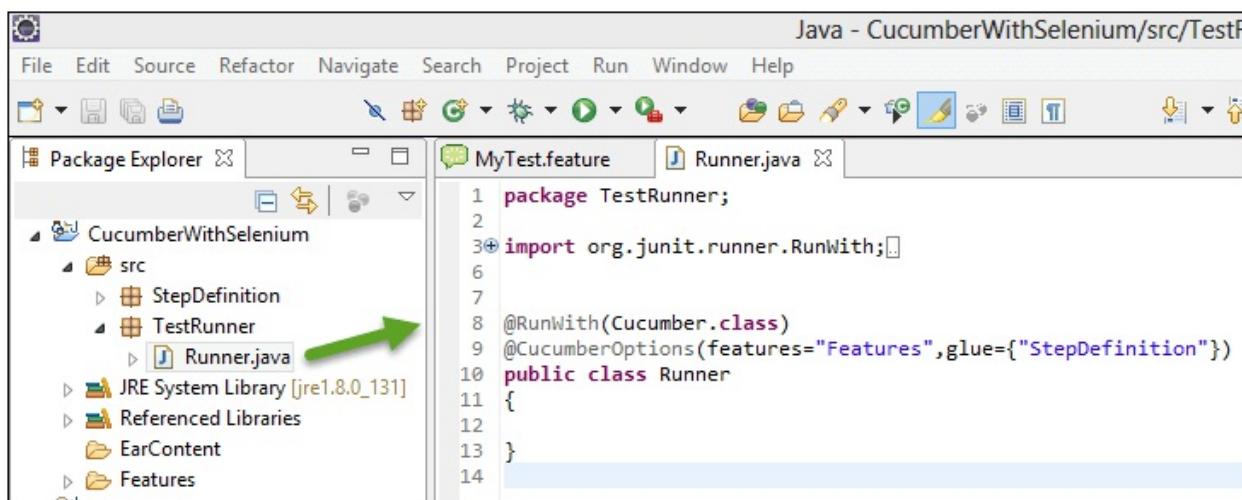
```

In the above code we run the cucumber test by using the following annotations:

@RunWith() annotation tells about the test runner class to start executing our tests.

@CucumberOptions() annotation is used to set some properties for our cucumber test like feature file, step definition, etc.

Screenshot of the TestRunner file.



Step 6) Creating Step Definition script.

Now here we create 'StepDefinition' package and then 'Steps.java' script file under it. Here we actually write a selenium script to carry

out the test under Cucumber methods.

```
package StepDefinition;

import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;

public class Steps {

    @Given("^Open the Firefox and launch the application$")
    public void open_the_Firefox_and_launch_the_application()
throws Throwable
    {
        System.out.println("This Step open the Firefox and
launch the application.");
    }

    @When("^Enter the Username and Password$")
    public void enter_the_Username_and_Password() throws
Throwable
    {
        System.out.println("This step enter the Username and
Password on the login page.");
    }

    @Then("^Reset the credential$")
    public void Reset_the_credential() throws Throwable
    {
        System.out.println("This step click on the Reset
button.");
    }
}
```

In the above code, the class is created with the name 'Steps.' Cucumber annotation is used to map with feature file. Each annotation method is defined:

@Given annotation define method to open firefox and launch the

application

@When annotation define method to enter the username and password

@Then annotation define method to reset the credential

Under each method, we are only printing a message.

Below is the screenshot of the 'Steps.java' script and project tree, how it looks like.

```

Java - CucumberWithSelenium/src/StepDefinition/Steps.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer MyTest.feature Runner.java Steps.java
CucumberWithSelenium
  src
    StepDefinition
      Steps.java
    TestRunner
      Runner.java
  JRE System Library [jre1.8.0_131]
  Referenced Libraries
  EarContent
  Features
    MyTest.feature

1 package StepDefinition;
2
3 import cucumber.api.java.en.Given;
4 import cucumber.api.java.en.Then;
5 import cucumber.api.java.en.When;
6
7 public class Steps {
8
9   @Given("^Open the Firefox and launch the application$")
10  public void open_the_Firefox_and_launch_the_application() throws Throwable
11  {
12    System.out.println("This Step open the Firefox and launch the application.");
13  }
14
15  @When("^Enter the Username and Password$")
16  public void enter_the_Username_and_Password() throws Throwable
17  {
18    System.out.println("This step enter the Username and Password on the login page.");
19  }
20
21  @Then("^Reset the credential$")
22  public void Reset_the_credential() throws Throwable
23  {
24    System.out.println("This step click on the Reset button.");
25  }
26
27 }
28

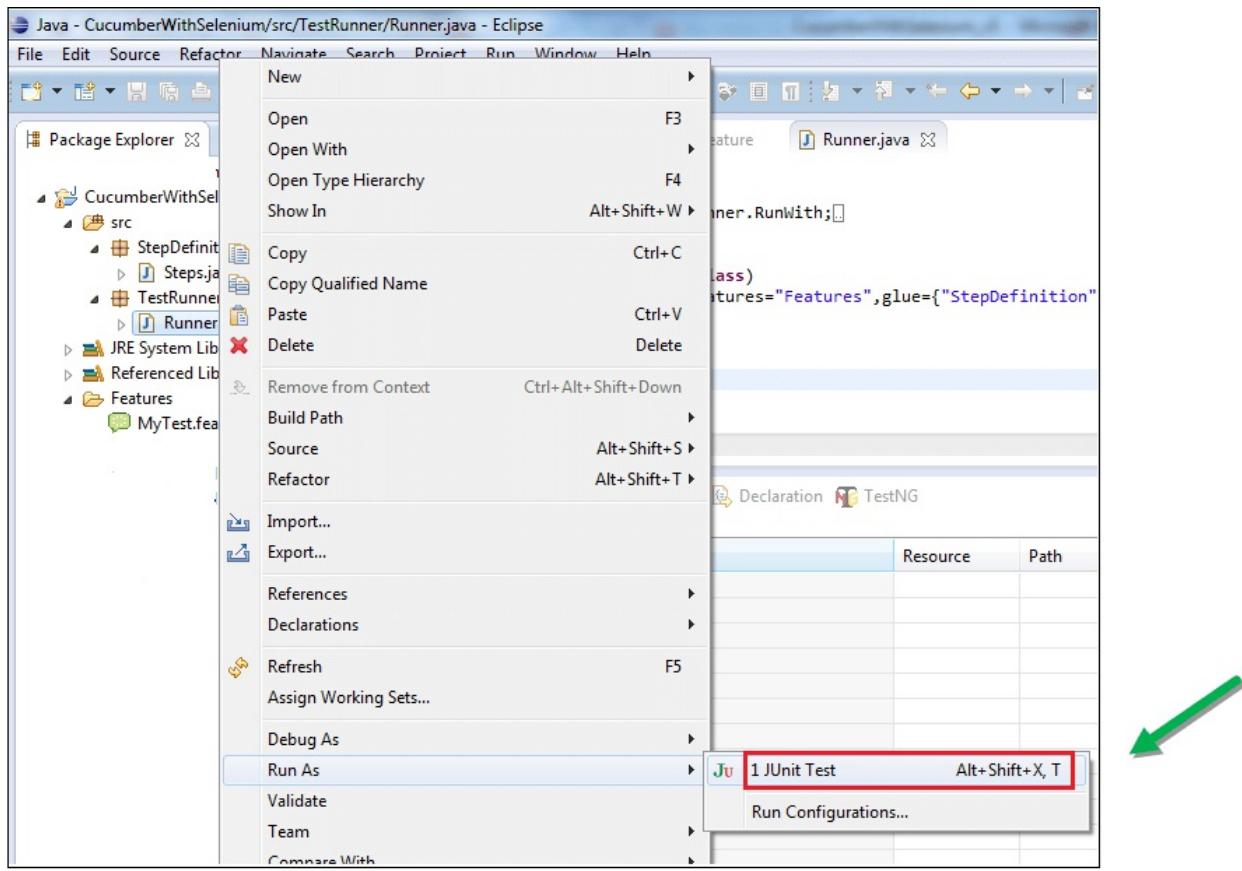
```

only printing message in this scenario

Note: Step definition is nothing but the steps you want to perform under this cucumber method.

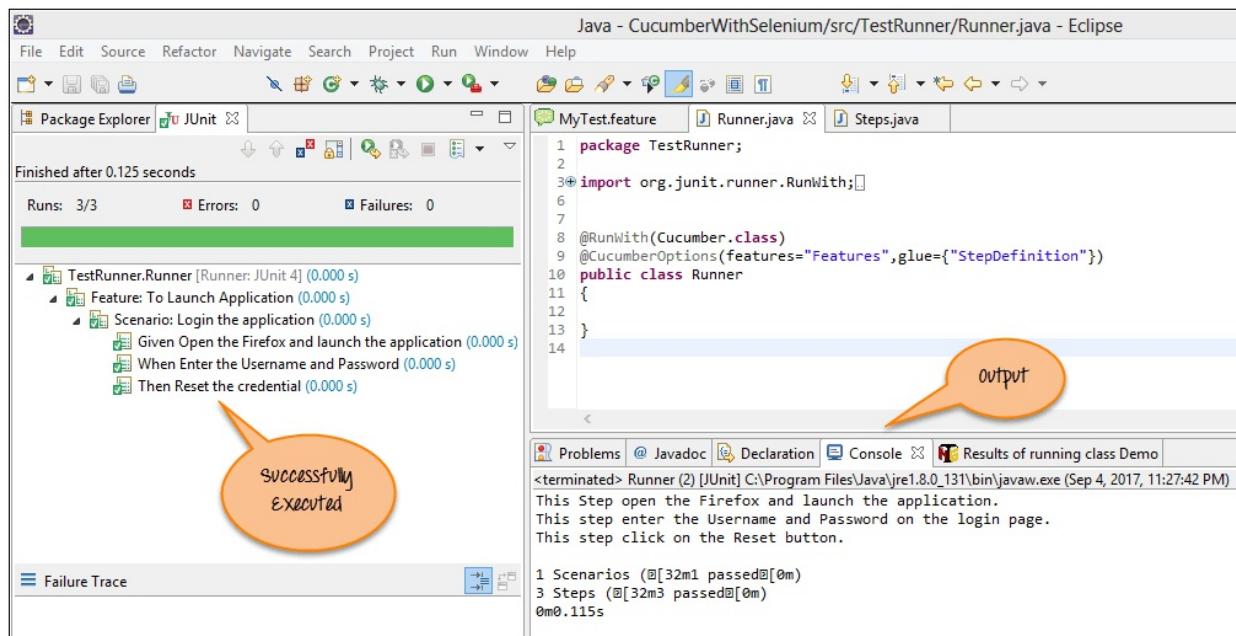
Step 7) Executing the Script.

The user can execute this script from Test runner script, i.e. 'Runner.java' as shown in below screenshot.



Step 8) Analyze the output.

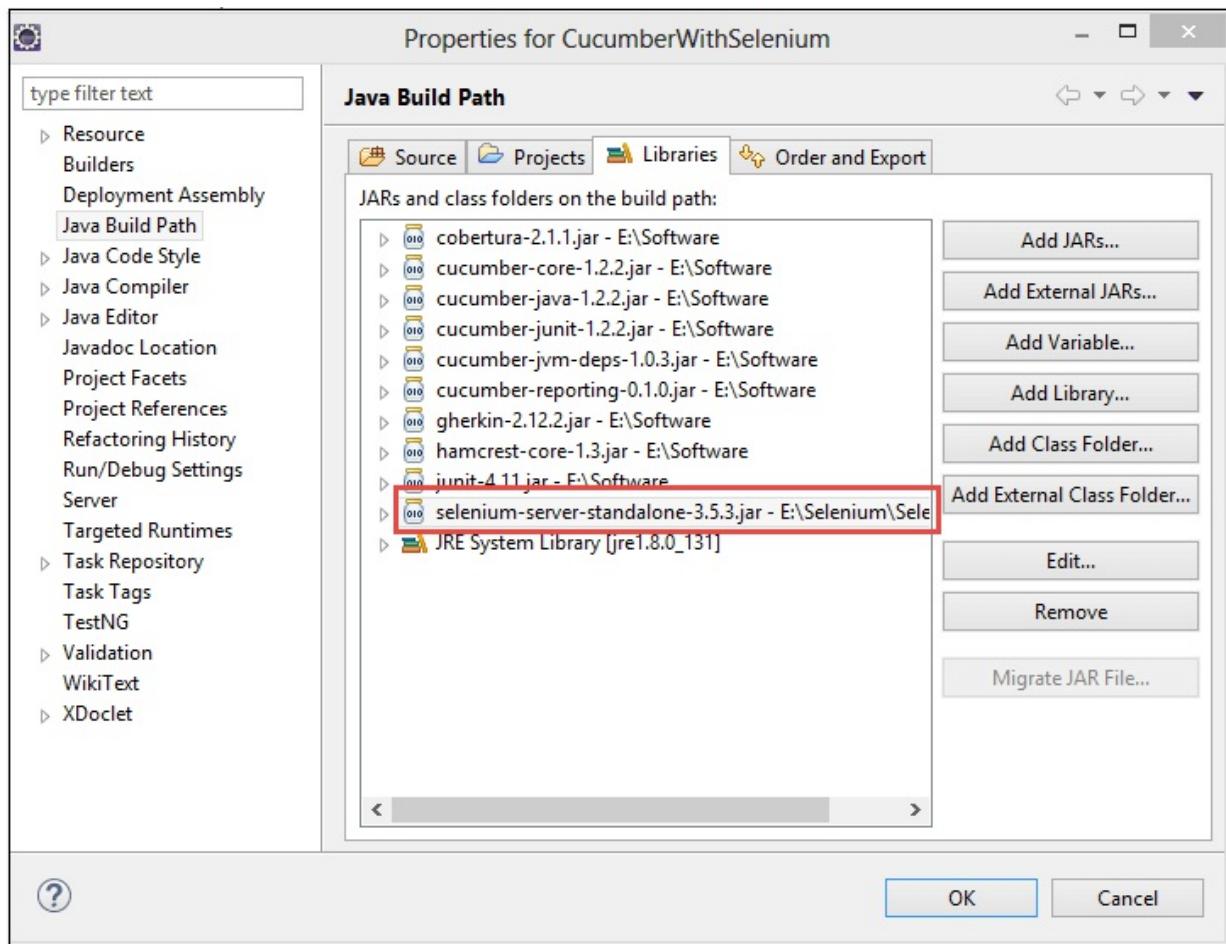
On executing the 'Runner.java' script, it displays the text on the console. It is the same text defined in 'Steps.java' script.



Scenario 2: Enter login Credential and reset the value.

Here we will just enter Credential on Guru99 demo login page and reset the value

For Scenario 2 we need to update only 'Steps.java' script. Here we actually write the selenium script as shown below steps. First, we need to add Selenium jar file to this project.



Step 1) Here we update the 'Steps.java' script as shown in the below code and screenshot.

```
package StepDefinition;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;

public class Steps {

    WebDriver driver;
```

```
@Given("^Open the Firefox and launch the application$")
public void open_the_Firefox_and_launch_the_application()
throws Throwable
{
    System.setProperty("webdriver.gecko.driver",
"E://Selenium//Selenium_Jars//geckodriver.exe");
    driver= new FirefoxDriver();
    driver.manage().window().maximize();
    driver.get("http://demo.guru99.com/v4");
}

@When("^Enter the Username and Password$")
public void enter_the_Username_and_Password() throws
Throwable
{
    driver.findElement(By.name("uid")).sendKeys("username12");

    driver.findElement(By.name("password")).sendKeys("password12");
}

@Then("^Reset the credential$")
public void Reset_the_credential() throws Throwable
{
    driver.findElement(By.name("btnReset")).click();
}
}
```

Screenshot of the above selenium script.

```

Java - CucumberWithSelenium/src/StepDefinition/Steps.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer Runner.java
CucumberWithSelenium
  src
    StepDefinition
      Steps.java
    TestRunner
      Runner.java
  JRE System Library [jre1.8.0_131]
  Referenced Libraries
  EarContent
  Features
    MyTest.feature
  2 import org.openqa.selenium.By;
  3 import org.openqa.selenium.WebDriver;
  4 import org.openqa.selenium.firefox.FirefoxDriver;
  5 import cucumber.api.java.en.Given;
  6 import cucumber.api.java.en.Then;
  7 import cucumber.api.java.en.When;
  8
  9 public class Steps {
 10
 11     WebDriver driver;
 12
 13     @Given("^Open the Firefox and launch the application$")
 14     public void open_the_Firefox_and_launch_the_application() throws Throwable
 15     {
 16         System.setProperty("webdriver.firefox.marionette", "E:/Selenium/Selenium_Jars/geckodriver.exe");
 17         driver= new FirefoxDriver();
 18         driver.manage().window().maximize();
 19         driver.get("www.demo.guru99.com/v4");
 20     }
 21
 22     @When("^Enter the Username and Password$")
 23     public void enter_the_Username_and_Password() throws Throwable
 24     {
 25         driver.findElement(By.name("uid")).sendKeys("username12");
 26         driver.findElement(By.name("password")).sendKeys("password12");
 27     }
 28
 29     @Then("^Reset the credential$")
 30     public void Reset_the_credential() throws Throwable
 31     {
 32         driver.findElement(By.name("btnReset")).click();
 33     }
 34 }
 35

```

Step 2) Execute the script.

After updating we run the Runner.java.

Step 3) Analyze the output.

In the output you can see the following:

- Browser launched.
- Guru99 bank demo site gets opened.
- Username and password are placed on the login page.
- Reset the values.

Guru99 Bank

User ID

Password

Steps To Generate Access

1. Visit - [here](#)
2. Enter your email id
3. Login credentials is allocated to you and mailed at your id
4. Login credentials are only valid for 20 days! So Hurry Up and quickly complete your tasks

Scenario 3: Enter login Credential on Guru99 & reset the value. Do this for 3 sets of data.

Here we need to update both the 'Step.java' and the feature file.

Step 1) Update the feature file as shown below:

Here we update the feature file with 'Scenario Outline' and 'examples' syntax.

Feature: Reset functionality on login page of Application

Scenario Outline: Verification of reset button with numbers of credential

Given Open the Firefox and launch the application

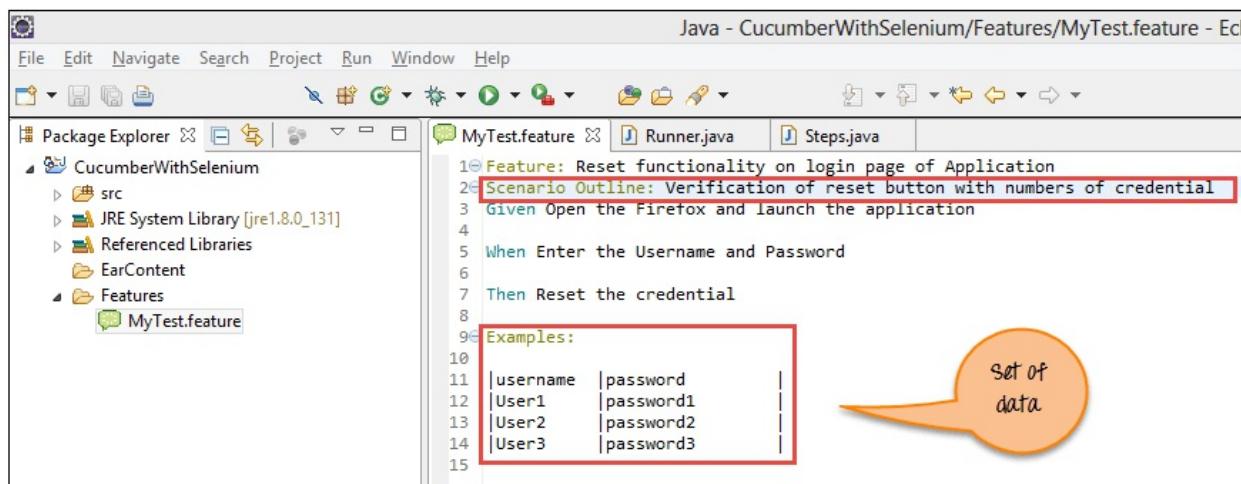
When Enter the Username <username>and Password <password>

Then Reset the credential

Examples:

username	password	
User1	password1	
User2	password2	
User3	password3	

// In this line we define the set of data.



Step 2) Now update the Step.java script.

Here we update the methods as to pass the parameters, updated script shown below:

```

package StepDefinition;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

import cucumber.api.java.en.Given;

```

```
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;

public class Steps {

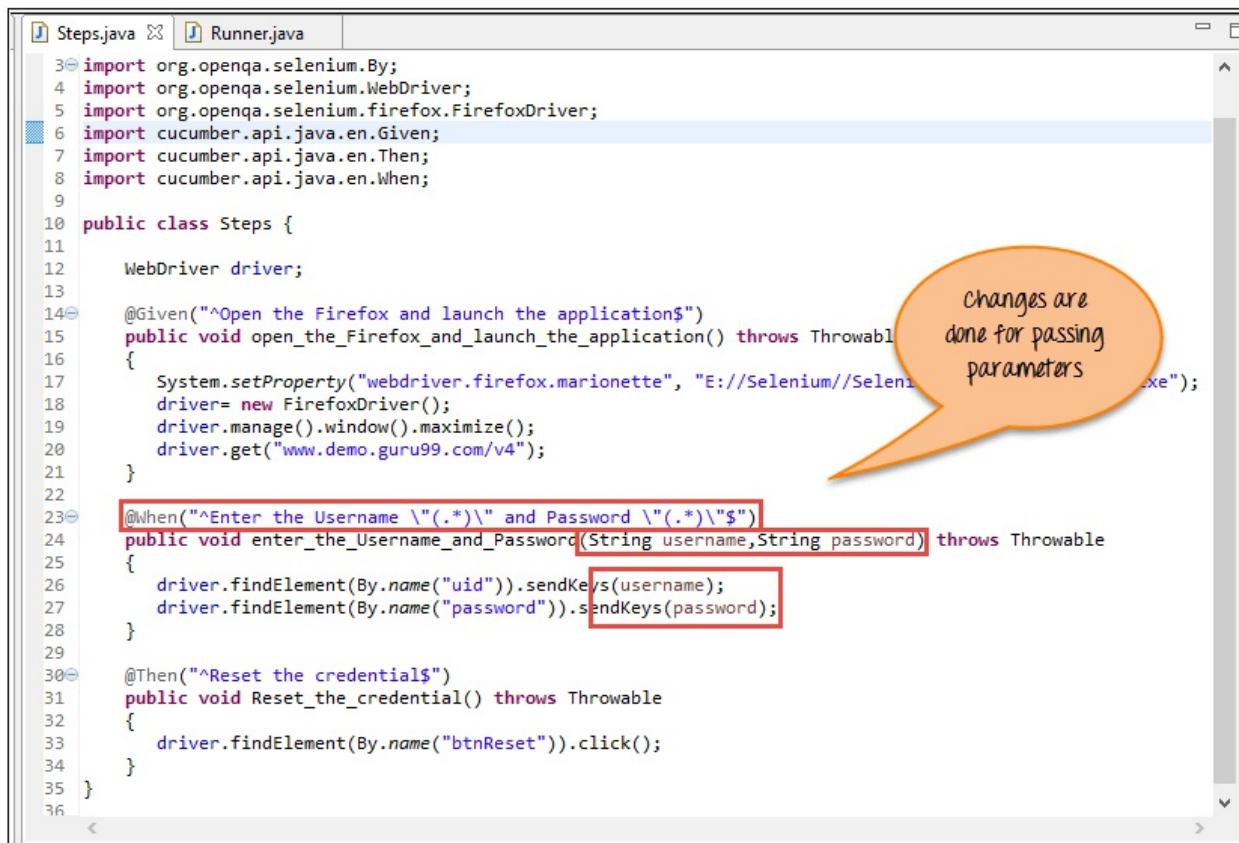
    WebDriver driver;

    @Given("^Open the Firefox and launch the application$")
    public void open_the_Firefox_and_launch_the_application()
throws Throwable
    {
        System.setProperty("webdriver.gecko.driver",
"E://Selenium//Selenium_Jars//geckodriver.exe");
        driver= new FirefoxDriver();
        driver.manage().window().maximize();
        driver.get("www.demo.guru99.com/v4");
    }

    @When("^Enter the Username \"(.*)\" and Password \"(.*)\"$")
    public void enter_the_Username_and_Password(String
username,String password) throws Throwable
    {
        driver.findElement(By.name("uid")).sendKeys(username);

        driver.findElement(By.name("password")).sendKeys(password);
    }

    @Then("^Reset the credential$")
    public void Reset_the_credential() throws Throwable
    {
        driver.findElement(By.name("btnReset")).click();
    }
}
```



```

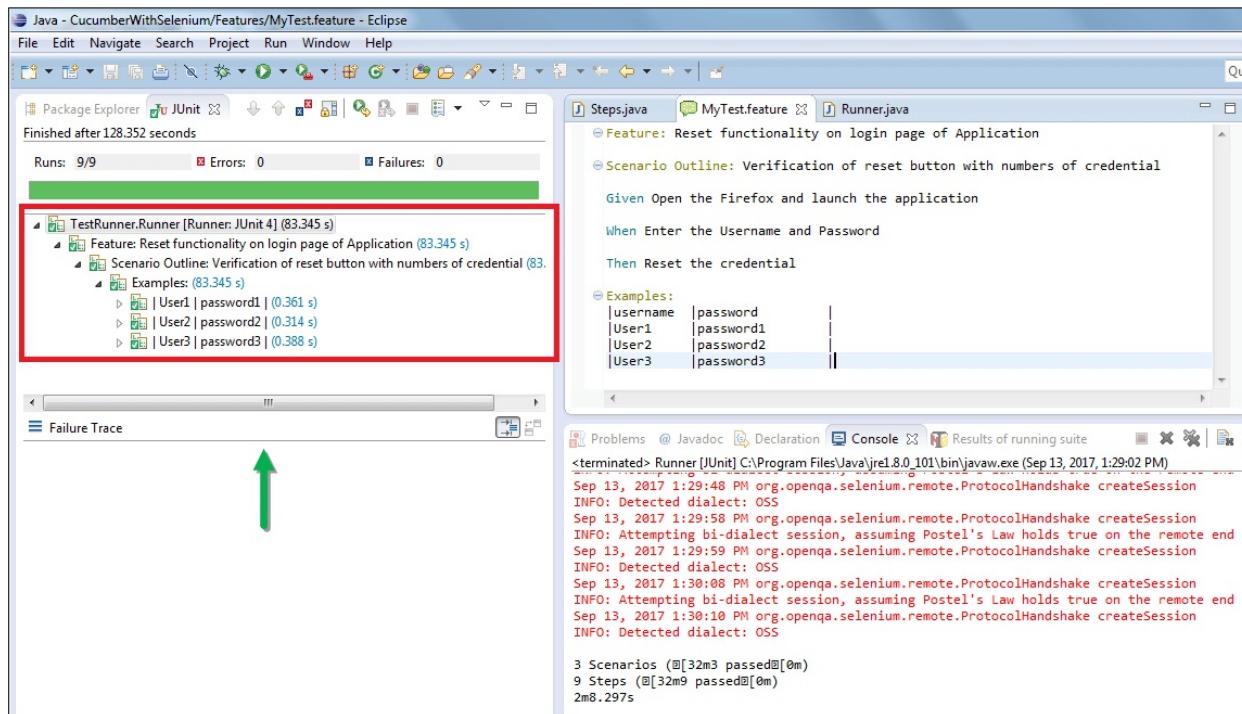
1 Steps.java 2 Runner.java
3 import org.openqa.selenium.By;
4 import org.openqa.selenium.WebDriver;
5 import org.openqa.selenium.firefox.FirefoxDriver;
6 import cucumber.api.java.en.Given;
7 import cucumber.api.java.en.Then;
8 import cucumber.api.java.en.When;
9
10 public class Steps {
11
12     WebDriver driver;
13
14     @Given("^Open the Firefox and launch the application$")
15     public void open_the_Firefox_and_launch_the_application() throws Throwable
16     {
17         System.setProperty("webdriver.firefox.marionette", "E:/Selenium//Selene
18         driver= new FirefoxDriver();
19         driver.manage().window().maximize();
20         driver.get("www.demo.guru99.com/v4");
21     }
22
23     @When("^Enter the Username \"(.*)\" and Password \"(.*)\"$")
24     public void enter_the_Username_and_Password(String username, String password) throws Throwable
25     {
26         driver.findElement(By.name("uid")).sendKeys(username);
27         driver.findElement(By.name("password")).sendKeys(password);
28     }
29
30     @Then("^Reset the credential$")
31     public void Reset_the_credential() throws Throwable
32     {
33         driver.findElement(By.name("btnReset")).click();
34     }
35 }
36

```

Changes are done for passing parameters

Step 3) Now execute the updated script.

Below screen shows the successful execution of the script and time taken by each set of data.



Step 4) Analyze the output.

In the output you can see the following:

Below output gets repeated for the number of data sets, i.e., 3 sets.

- Browser launched.
- Guru99 bank demo site gets opened.
- Username and password are placed on the login page.
- Reset the values.

Guru99 Bank

User ID

Password

Steps To Generate Access

1. Visit - [here](#)
2. Enter your email id
3. Login credentials is allocated to you and mailed at your id
4. Login credentials are only valid for 20 days! So Hurry Up and quickly complete your tasks

Conclusion.

Cucumber is a very popular BDD tool. It is easy to read and can be understood by all stakeholders including technical and non-technical person.

Cucumber can be integrated with Selenium using following 3 steps

1. Create feature file in which define the feature and scenarios step by step using Gherkin language.
2. Create Testrunner file. In this file, we integrated Cucumber with selenium. We execute this script.
3. Create Step definition, the actual selenium script defined under this package.

Chapter 62: Drag and Drop action in Selenium

Some web application, have a functionality to drag web elements and drop them on defined area or element. We can automate drag and drop of such elements using Selenium Webdriver.

Syntax for drag and drop.

The Actions class has two methods that support Drag and Drop. Let's study them-

```
Actions.dragAndDrop(Source locator, Destination locator)
```

In dragAndDrop method, we pass the two parameters -

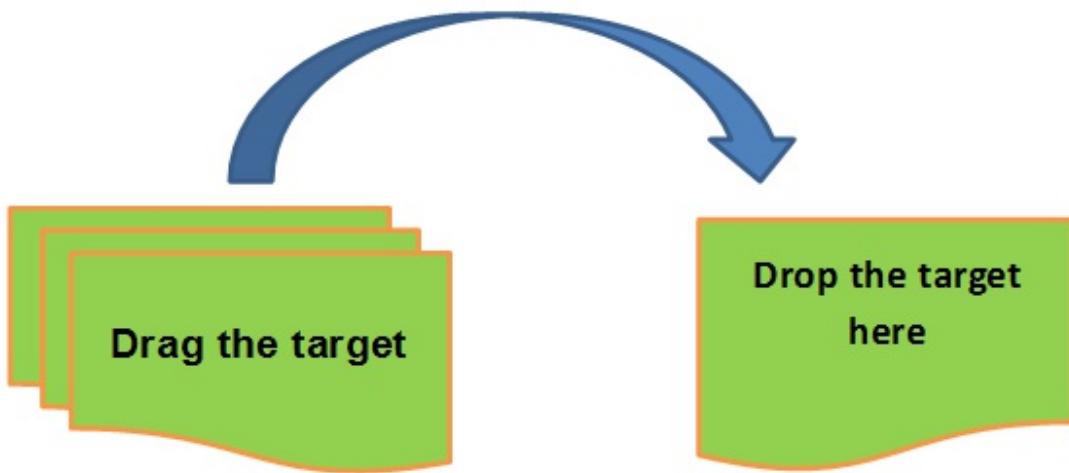
1. First parameter "Source locator" is the element which we need to drag
2. Second parameter "Destination locator" is the element on which we need to drop the first element

```
Actions.dragAndDropBy(Source locator, x-axis pixel of
Destination locator, y-axis pixel of Destination locator)
```

dragAndDropBy method we pass the 3 parameters -

1. First parameter "Source locator" is the element which we need to drag
2. The second parameter is x-axis pixel value of the 2nd element on which we need to drop the first element.
3. The third parameter is y-axis pixel value of the 2nd element on

which we need to drop the first element.

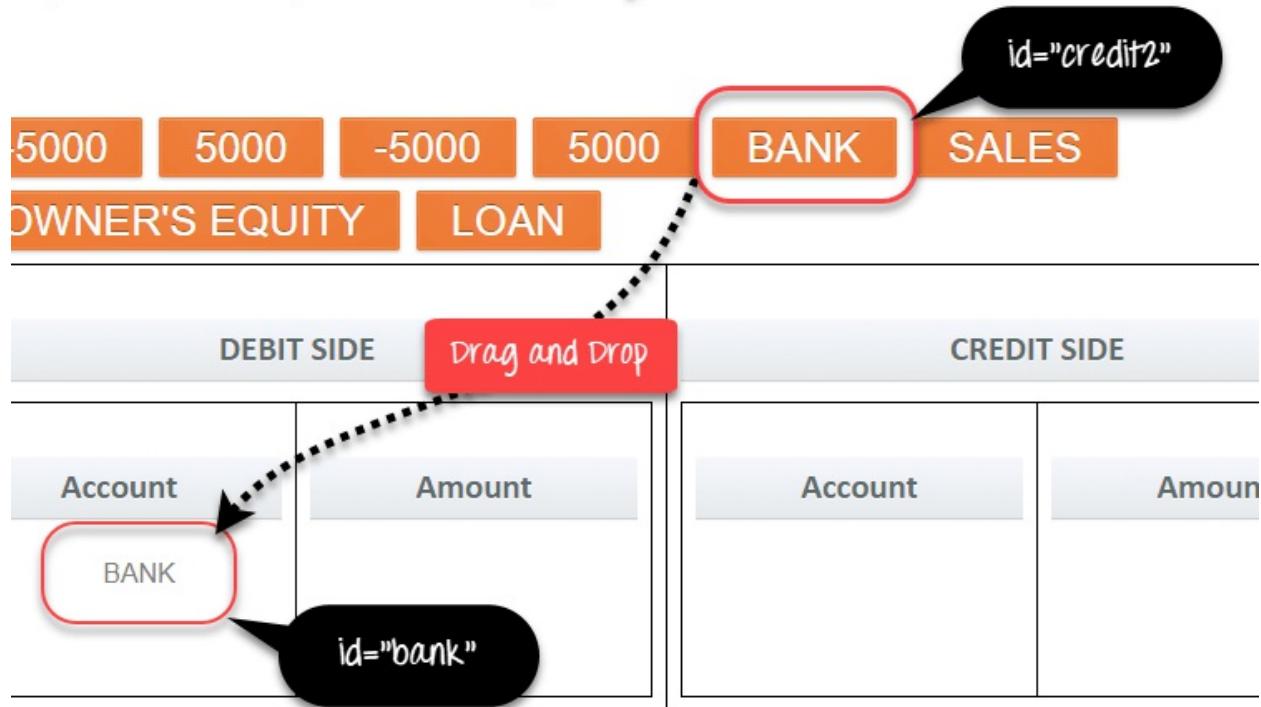


Let's practically show you the drag and drop of an element using the selenium webdriver with following 3 scenarios

- Scenario 1: BANK element is dragged and dropped on the specific element by DragAndDrop method.
- Scenario 2: BANK element is dragged and dropped on the specific element by DragAndDrop method.
- Scenario 3: Few elements are dragged and dropped and then verify the message is displayed or not.

Scenario 1: BANK element is dragged and dropped on the specific cell by DragAndDrop method.

In the following code, we launch the given URL in Firefox browser and then drag the BANK element and drop on the DEBIT SIDE block through dragAndDrop method.



```

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.interactions.Actions;
import org.testng.annotations.Test;

public class DragAndDrop {

    WebDriver driver;

    @Test
    public void DragnDrop()
    {
        System.setProperty("webdriver.chrome.driver",
E://Selenium//Selenium_Jars//chromedriver.exe ");
        driver= new ChromeDriver();

        driver.get("http://demo.guru99.com/test/drag_drop.html");

        //Element which needs to drag.
        WebElement From=driver.findElement(By.xpath("//*
[@id='credit2']/a"));
    }
}

```

```

    //Element on which need to drop.
    WebElement To=driver.findElement(By.xpath("//*
[@id='bank']/li"));

    //Using Action class for drag and drop.
    Actions act=new Actions(driver);

    //Dragged and dropped.
    act.dragAndDrop(From, To).build().perform();
}
}

```

Code Explanation: In the above code we launch the given URL in Firefox browser and then drag the BANK element and drop on the DEBIT SIDE block through dragAndDrop method. Explained briefly below:

First, we capture the 1st element which we need to drag in variable "From."

```
WebElement From=driver.findElement(By.xpath("//*
[@id='credit2']/a));
```

Second, we capture the 2nd element on which we need to drop the 1st element in variable "To".

```
WebElement To=driver.findElement(By.xpath("//*
[@id='bank']/li));
```

Third, we create object of Actions class as we use methods of Actions class.

```
Actions act=new Actions(driver);
```

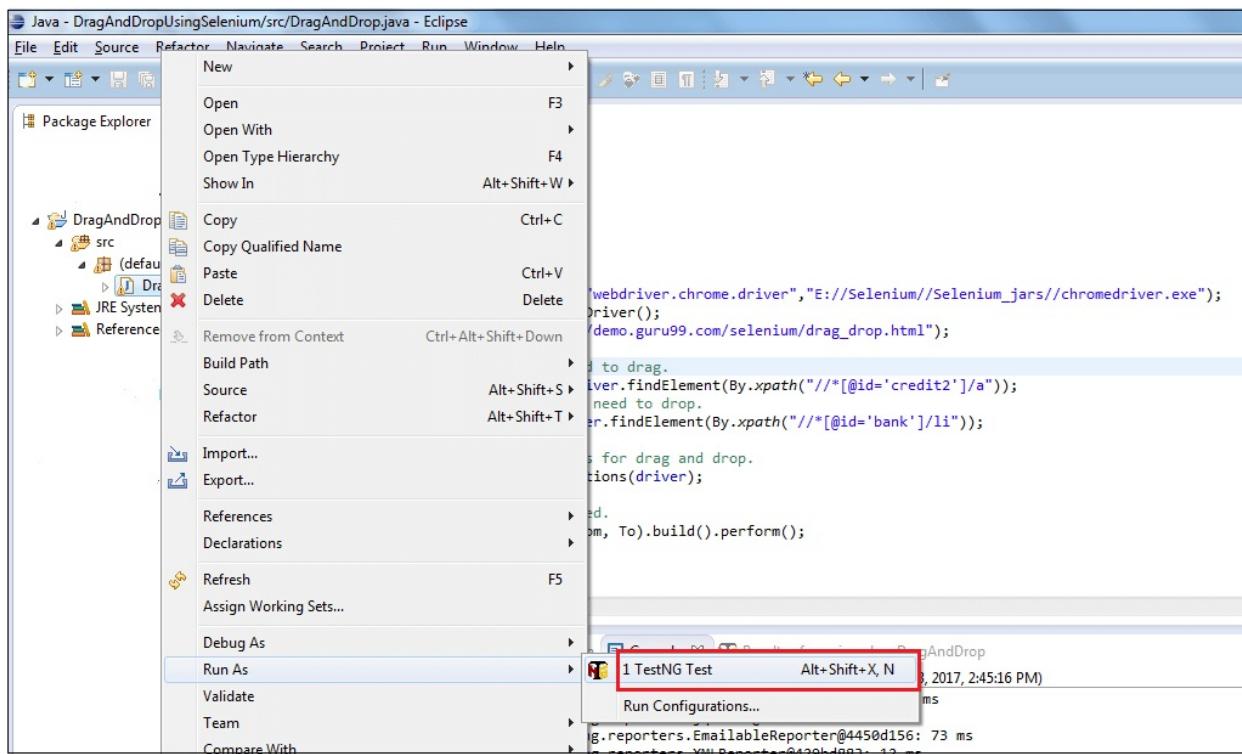
For drag and drop element we use dragAndDrop method of Actions class and passes the parameters as the first element(Sourcelocator)

"From" and the second element(Destinationlocator) "To". Below line will drag the 1st element and drop it on the 2nd element.

```
act.dragAndDrop(From, To).build().perform();
```

Execution of the script.

Now you can execute the above script one by one from eclipse as shown in below screenshot.



Here is the output when you run the script

Selenium Drag and Drop Example:

Drag the following blocks into empty cells below

Note: Only certain blocks are permitted to be dropped in specific cells in the table



Scenario 2: BANK element is dragged and dropped on the specific cell by DragAndDrop method.

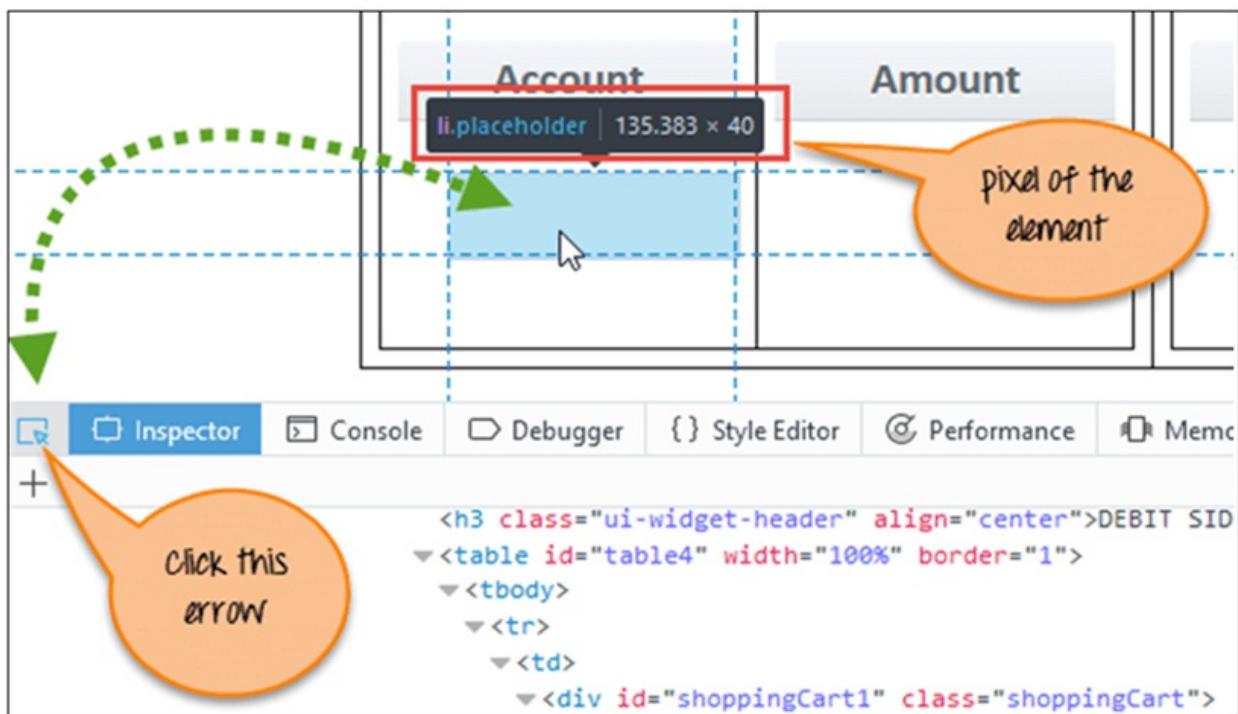
In this scenario, we launch the given URL in the browser and then drag the BANK element and drop on the DEBIT SIDE block through dragAndDropBy method. To dragAndDropBy, we need to find the pixel of the element.

How to find Pixel?

Open the URL in Chrome or FireFox and click on the Blue color arrow.

Next click on any element for which you want to know the pixel.

You will find the pixel above the element as shown in below screenshot.



```

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.interactions.Actions;
import org.testng.annotations.Test;

public class DragAndDrop {

    WebDriver driver;
    @Test
    public void DragnDrop()
    {

System.setProperty("webdriver.chrome.driver", "E://Selenium//Sel
nium_Jars//chromedriver.exe");
        driver= new ChromeDriver();

driver.get("http://demo.guru99.com/test/drag_drop.html");

        //Element(BANK) which need to drag.
        WebElement From=driver.findElement(By.xpath("//*
[@id='credit2']/a"));
    }
}

```

```

//Using Action class for drag and drop.
Actions act=new Actions(driver);

//Drag and Drop by Pixel.
act.dragAndDropBy(From,135, 40).build().perform();
}

}

```

NOTE: The pixels values change with screen resolution and browser size. This method is hence not reliable and not widely used.

Scenario 3: Few elements are dragged and dropped and then verify the message is displayed or not.

In the following code, we launch the given URL in the browser and then drag the elements like BANK, SALES, 500 and drop on the respective block. Once done we verify the output message.

```

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.interactions.Actions;
import org.testng.annotations.Test;

public class DragAndDrop {

    WebDriver driver;
    @Test
    public void DragnDrop()
    {
        System.setProperty("webdriver.chrome.driver",
E://Selenium//Selenium_Jars//chromedriver.exe");
        driver= new ChromeDriver();

driver.get("http://demo.guru99.com/test/drag_drop.html");

```

```
//Element(BANK) which need to drag.  
WebElement From1=driver.findElement(By.xpath("//*  
[@id='credit2']/a"));  
  
//Element(DEBIT SIDE) on which need to drop.  
WebElement To1=driver.findElement(By.xpath("//*  
[@id='bank']/li"));  
  
//Element(SALES) which need to drag.  
WebElement From2=driver.findElement(By.xpath("//*  
[@id='credit1']/a"));  
  
//Element(CREDIT SIDE) on which need to drop.  
WebElement To2=driver.findElement(By.xpath("//*  
[@id='loan']/li"));  
  
//Element(500) which need to drag.  
WebElement From3=driver.findElement(By.xpath("//*  
[@id='fourth']/a"));  
  
//Element(DEBIT SIDE) on which need to drop.  
WebElement To3=driver.findElement(By.xpath("//*  
[@id='amt7']/li"));  
  
//Element(500) which need to drag.  
WebElement From4=driver.findElement(By.xpath("//*  
[@id='fourth']/a"));  
  
//Element(CREDIT SIDE) on which need to drop.  
WebElement To4=driver.findElement(By.xpath("//*  
[@id='amt8']/li"));  
  
//Using Action class for drag and drop.  
Actions act=new Actions(driver);  
  
//BANK drag and drop.  
act.dragAndDrop(From1, To1).build().perform();  
  
//SALES drag and drop.  
act.dragAndDrop(From2, To2).build().perform();  
  
//500 drag and drop debit side.
```

```

        act.dragAndDrop(From3, To3).build().perform();

        //500 drag and drop credit side.
        act.dragAndDrop(From4, To4).build().perform();

        //Verifying the Perfect! message.

if(driver.findElement(By.xpath("//a[contains(text(), 'Perfect')]"))
)).isDisplayed())
{
    System.out.println("Perfect Displayed !!!");
}
else
{
    System.out.println("Perfect not Displayed !!!");
}
}

```

Summary

- In the above tutorials, we illustrate the drag and drop functionality of the web application through Action methods in Webdriver:
- dragAndDrop(Source locator, Destination locator)
- dragAndDropBy(Source locator, x-axis pixel of Destination locator, y-axis pixel of Destination locator)
- To drag and drop the element first we used DragAndDrop method from the Actions class in which we pass the 2 parameters, 1st parameter is the element which we need to drag, and 2nd parameter is the element on which we need to drop the 1st element.
- Second, we used the dragAndDropBy method from the Actions class in which we pass the 3 parameters, the 1st parameter is the element which we need to drag, 2nd parameter is the x-axis pixel value of the 2nd element, 3rd parameter is the y-axis pixel value of

the 2nd element.

Chapter 63: Selenium C# Webdriver Tutorial for Beginners

Selenium Overview:

Selenium is an open-source, web Automation Testing tool that supports multiple browsers and multiple operating systems. It allows testers to use multiple programming languages such as Java, C#, Python, .Net, Ruby, PHP, and Perl for coding automated tests.

C # Overview:

C# is an object-oriented programming language derived from C++ and Java. C# allows developers to build applications using Visual Studio on .Net platform. The following are the key features of C#.

1. It is an Object-Oriented programming language
2. It supports the development of console, windows and web-based applications
3. It provides features such as Encapsulation, Inheritance, and Polymorphism.

Basic Syntax of C #:

A program in C # need to contain the following sections

1. Namespace declaration
2. Classes
3. Class Attributes and Methods
4. Main method
5. Program statements

Example-

Below is a sample C# program to print the text 'Guru99' on the system console.

```
using System;
namespace FirstProgram {
class DemoPrint {
static void main(){
    Console.WriteLine("Guru99");
}
}
```

Explanation:

- A namespace in C# is a collection of multiple classes. Each namespace must be declared using the keyword 'using'.
- The first statement of the code includes the namespace 'System' into our program. System namespace defines the fundamental classes and events used in C#. The namespace to be used depends on the program requirement.
- The second statement is declaring a namespace "FirstProgram" for the class "DemoPrint." A C# file can contain multiple classes within the same namespace.
- The third statement includes the class declaration. A class may contain multiple attributes and multiple methods.
- The fourth statement includes a declaration of the Main method.

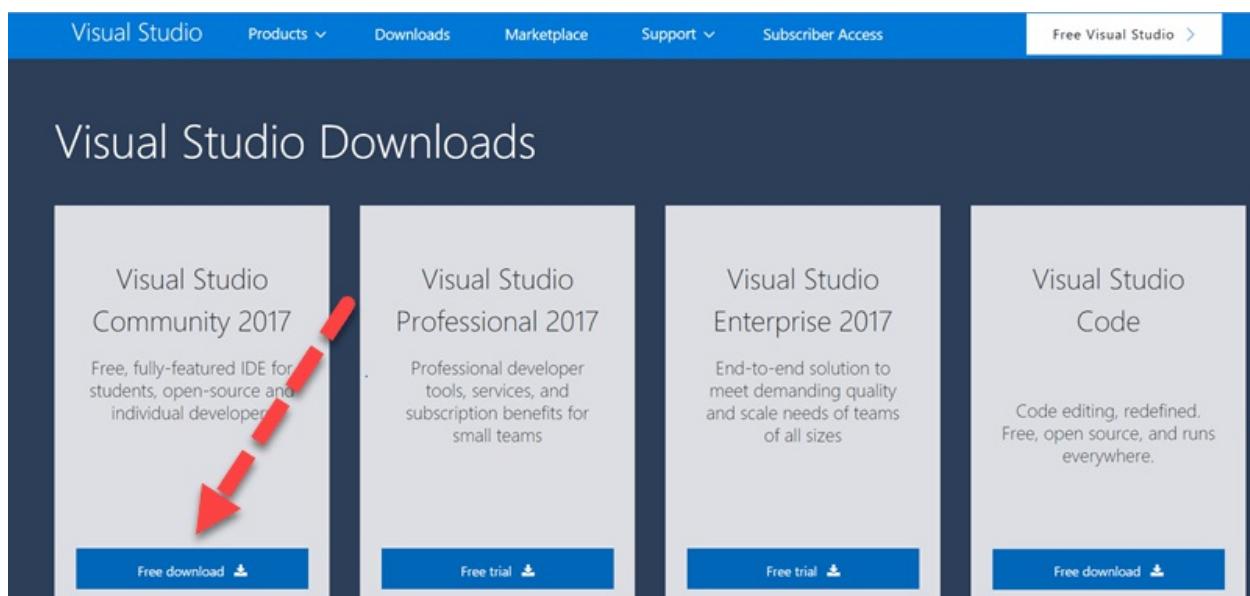
- The main method is the entry point of execution for each class.
- The last statement is the C# syntax used for printing a statement to console. WriteLine is a method of the class 'Console'.

Set Up Visual Studio with Selenium WebDriver:

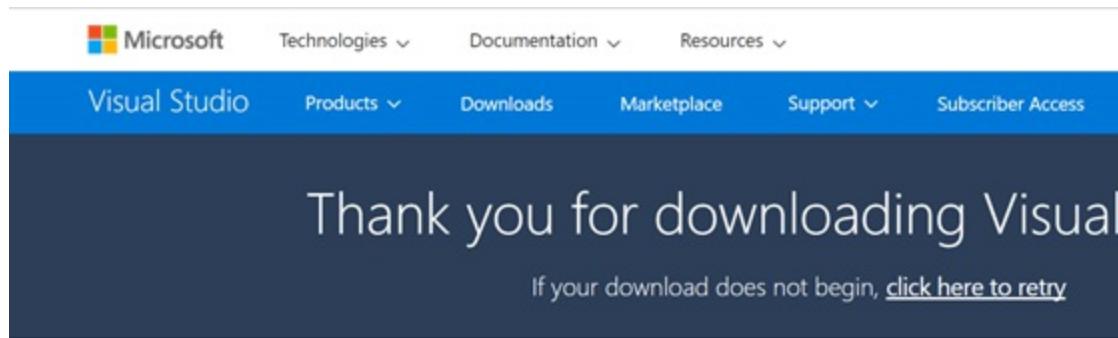
Visual Studio is an Integrated Development Environment (IDE) that is used by developers to build applications across multiple platforms such as Windows, Android, iOS and Cloud-based applications.

Step 1) Navigate to the URL

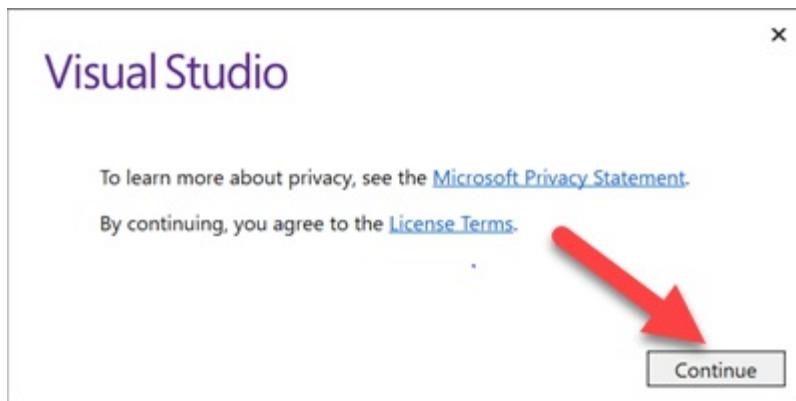
<https://www.visualstudio.com/downloads/> and Click on the 'Free download' button displayed on Visual Studio Community 2017 tab



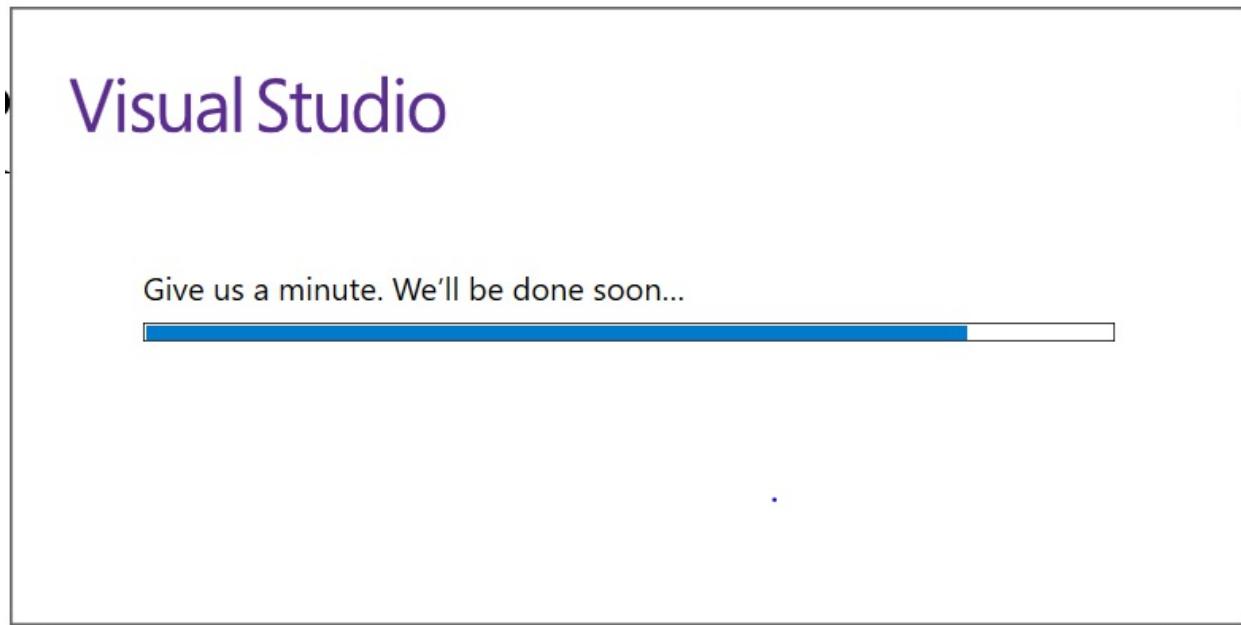
Step 2) Open the exe downloaded. Click on 'Yes' if asked for Admin Rights.



Step 3) The below popup will appear. Click on 'Continue' button.



Files will be downloaded as shown in the popup below.

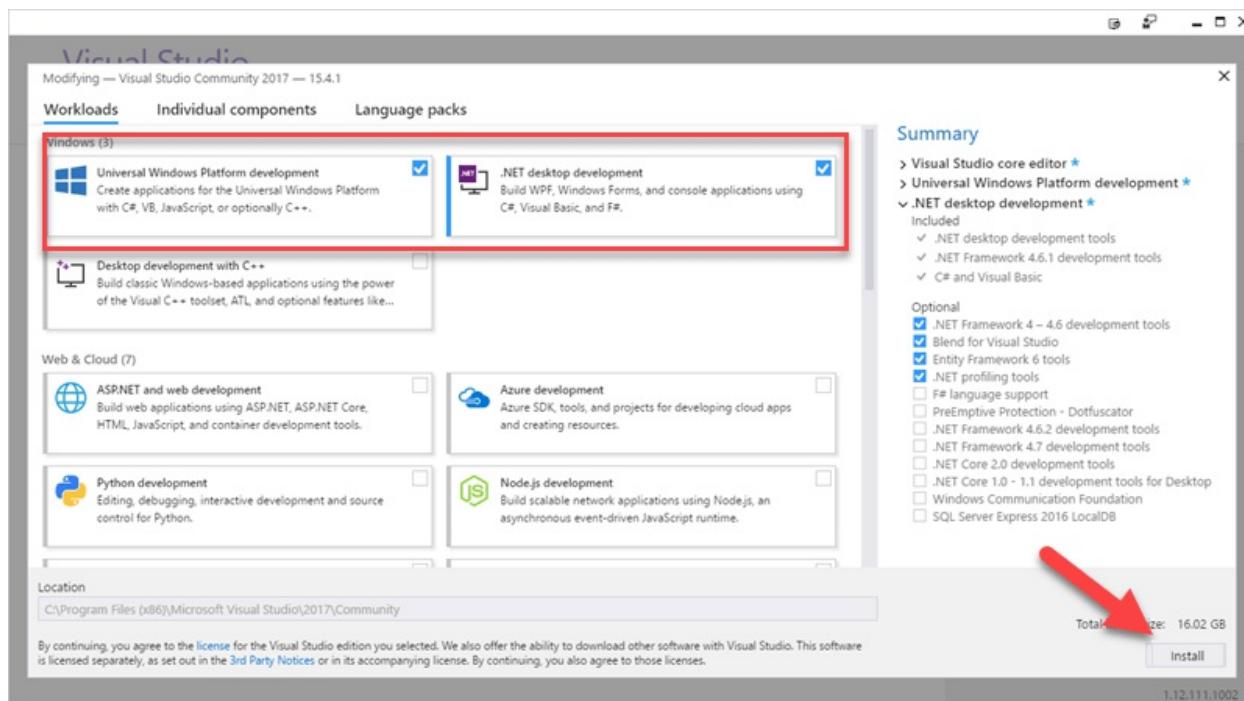


Step 4) In the next screen,

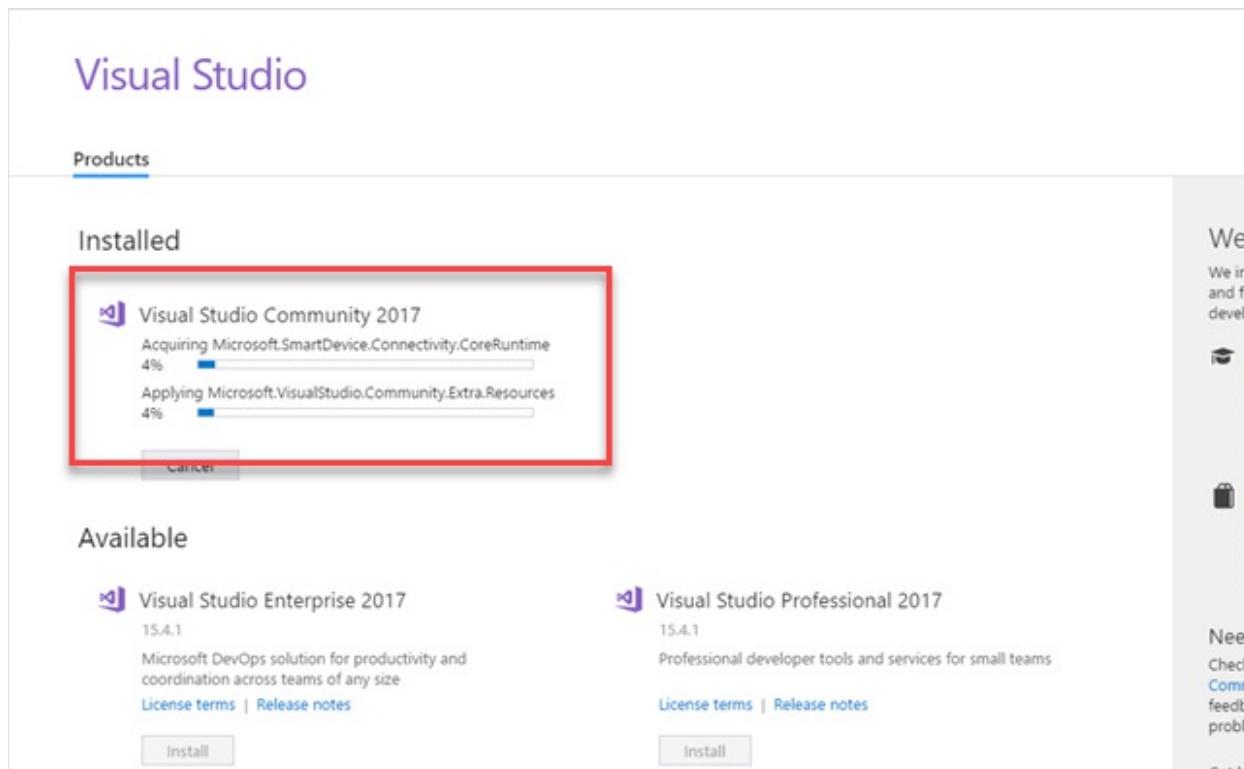
Select the checkboxes for

- Universal Windows Platform development
- Net desktop development

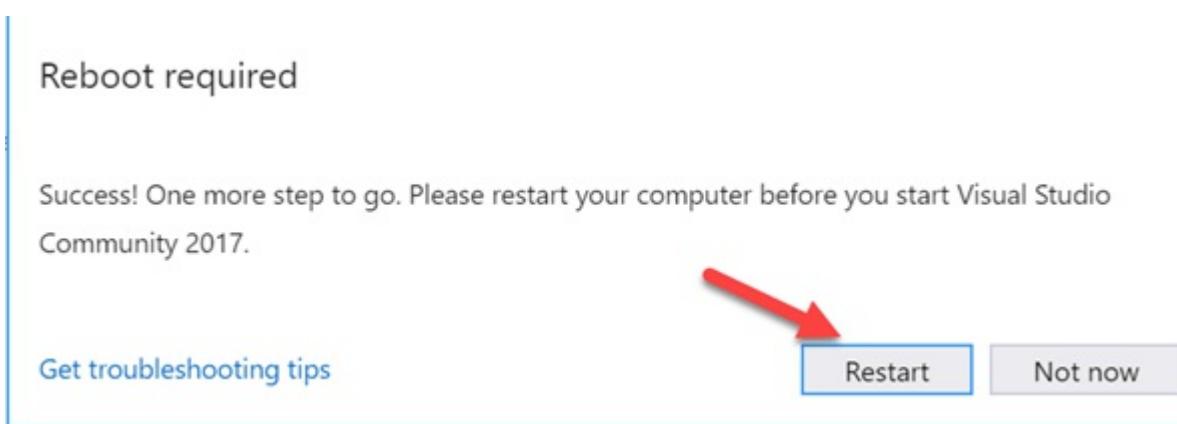
Click on 'Install.'



Wait for installation of each component to complete. Files are 16GB in size and will take time.



Step 5) The below pop up will be displayed. Click on 'Restart' button.



Step 6) Once the machine is restarted, search for "Visual Studio 2017" on the start menu and click on the search result. The following popup will appear. Click on "Not now, maybe later" link if you do not have an existing account.

Visual Studio

Welcome!

Connect to all your developer services.

Sign in to start using your Azure credits, publish code to a private Git repository, sync your settings, and unlock the IDE.

[Learn more](#)

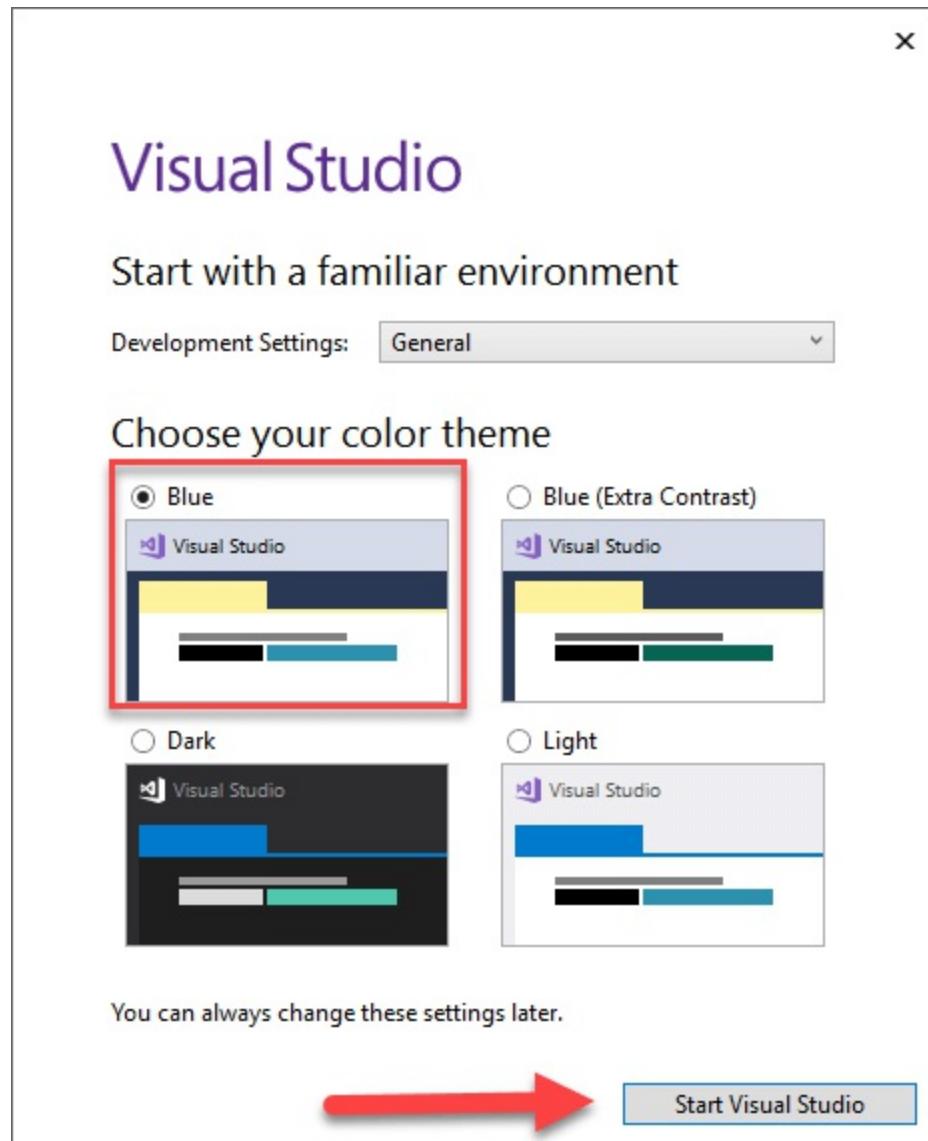
[Sign in](#)

Don't have an account? [Sign up](#)

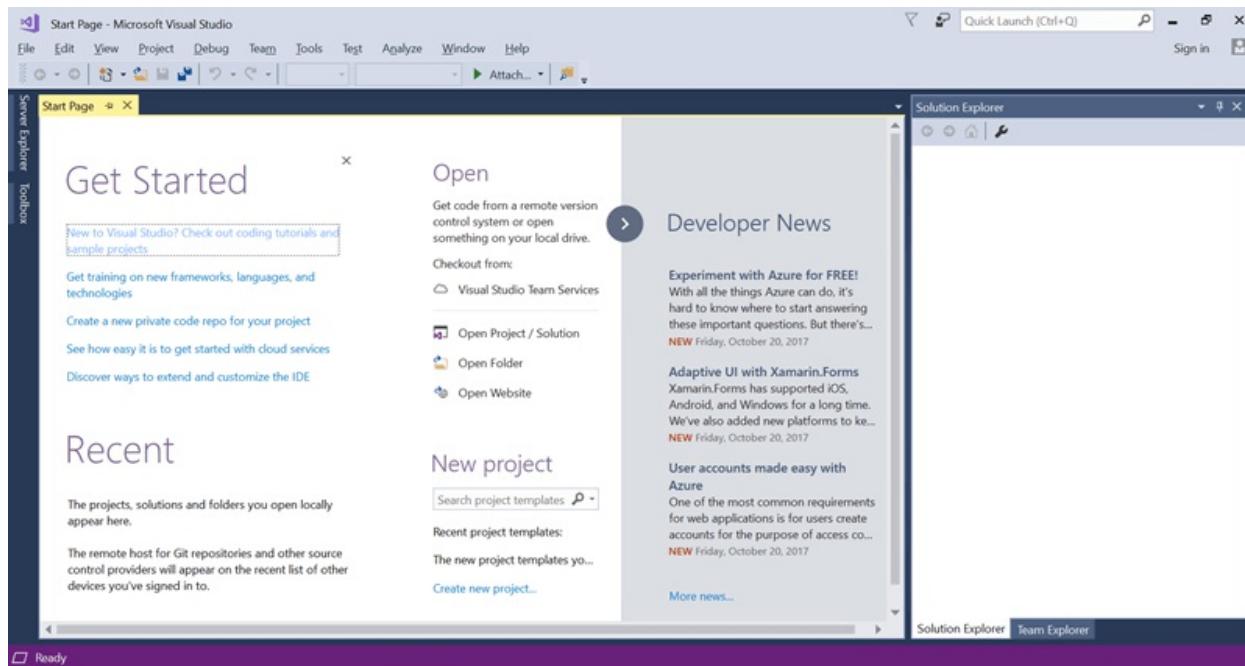
[Not now, maybe later.](#)

Step 7) In the next screen,

- Select color theme of your liking
- Click the button "Start Visual Studio"

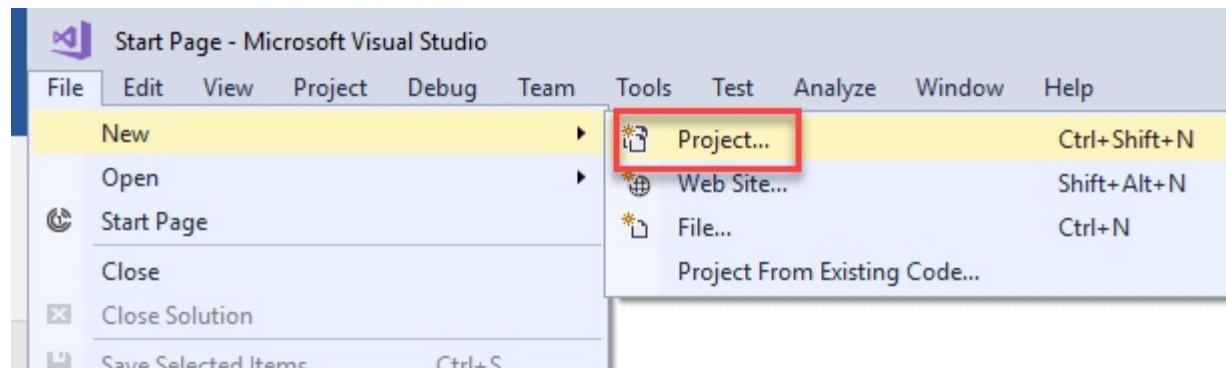


Step 8) Visual Studio 'Get Started' screen will appear.



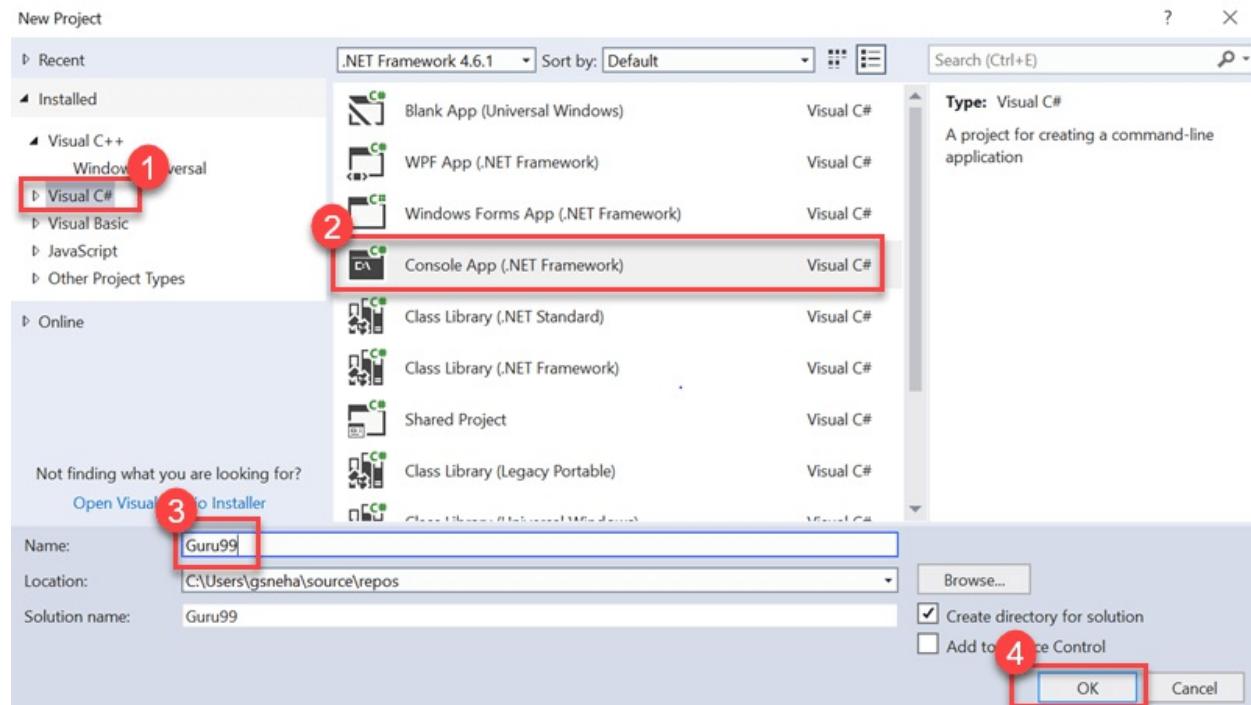
Create a new project in Visual Studio:

Step 1) In the File Menu, Click New > Project

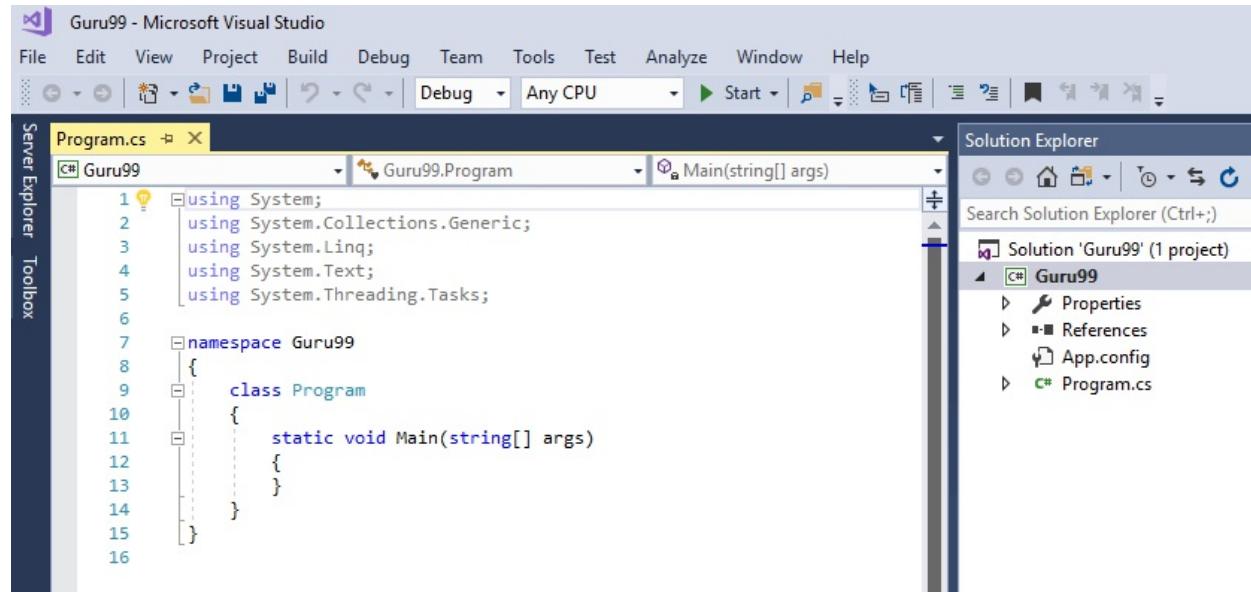


Step 2) In the next screen,

1. Select the option 'Visual C#'
2. Click on Console App (.Net Framework)
3. Enter name as "Guru99"
4. Click OK

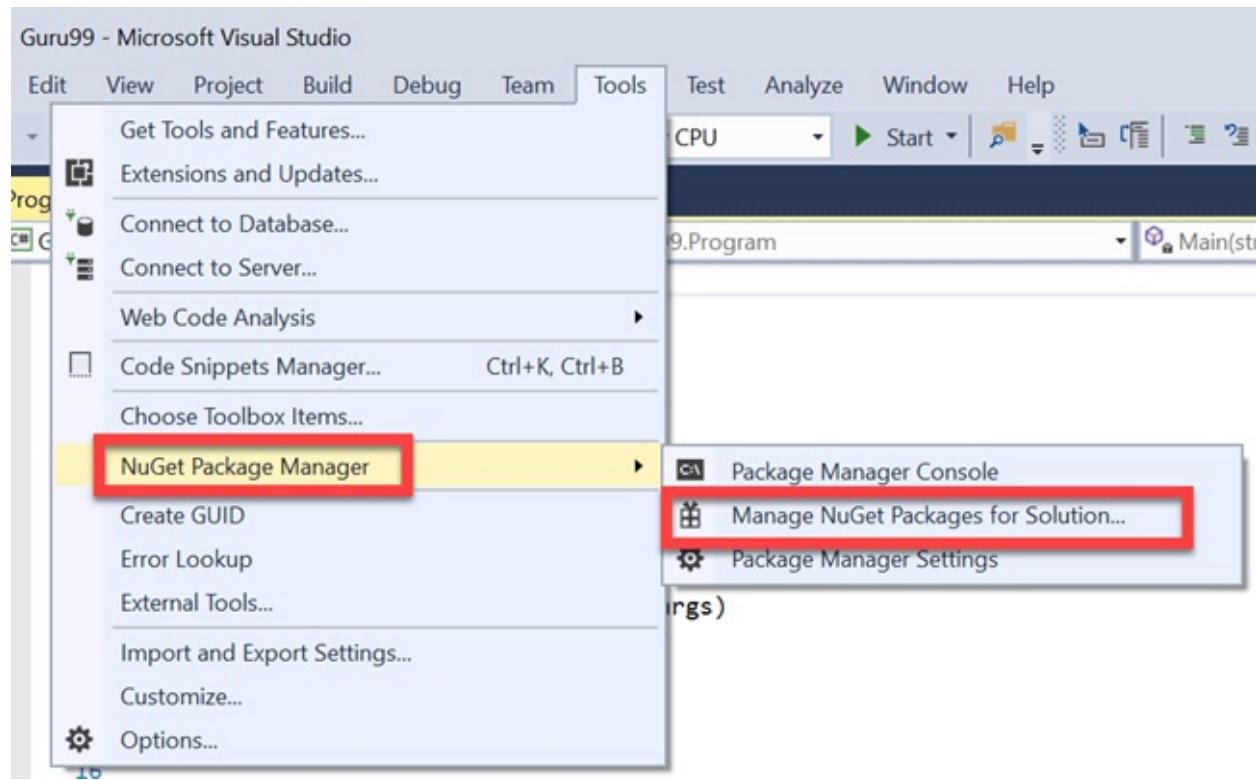


Step 3) The below screen will be displayed once the project is successfully created.



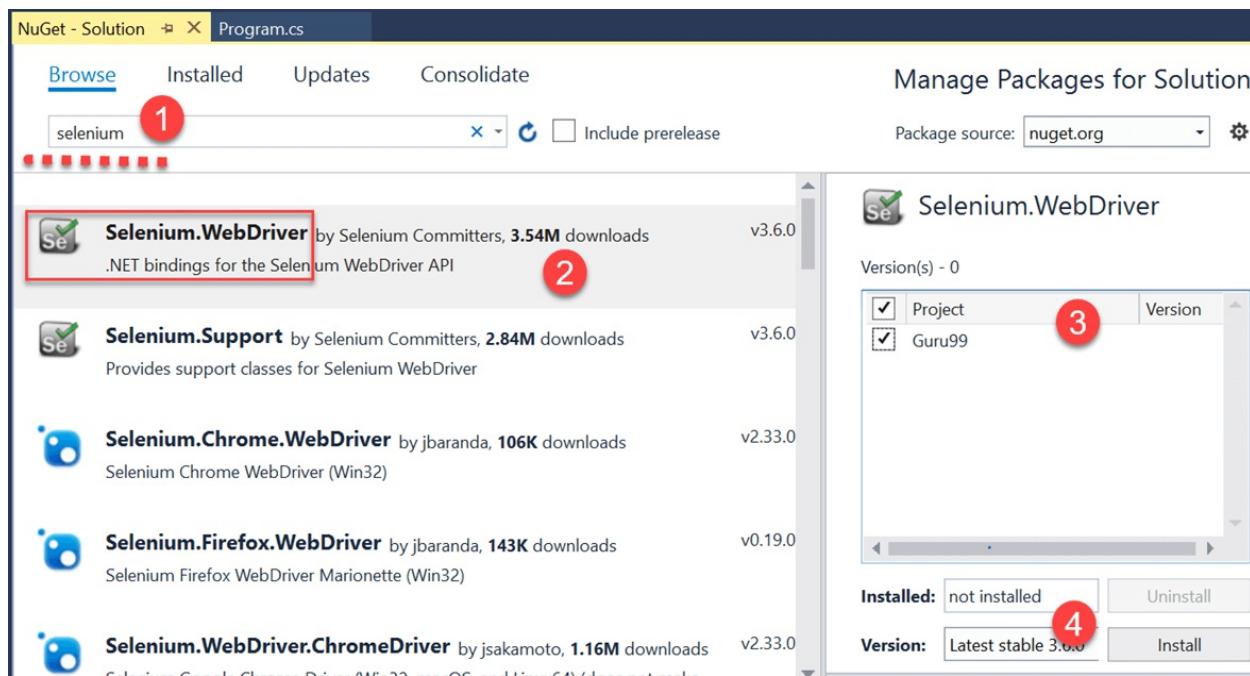
Set up Visual Studio with Selenium WebDriver:

Step 1) Navigate to Tools -> NuGet Package Manager -> Manage NuGet Packages for Solution

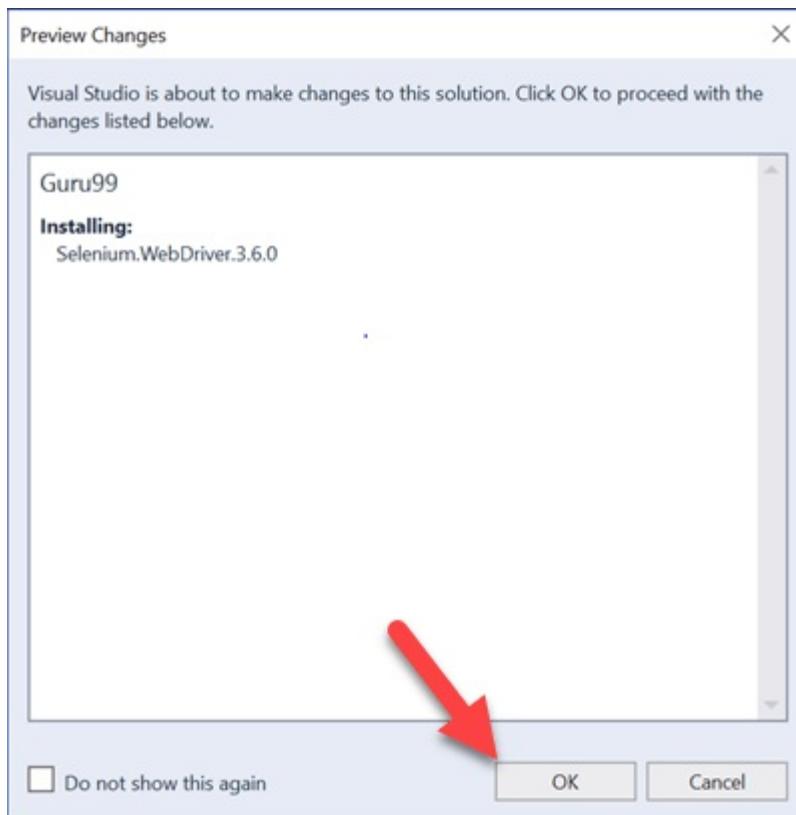


Step 2) In the next screen

1. Search for Selenium on the resultant screen
2. Select the first search result
3. Check the project checkbox
4. Click on 'Install'

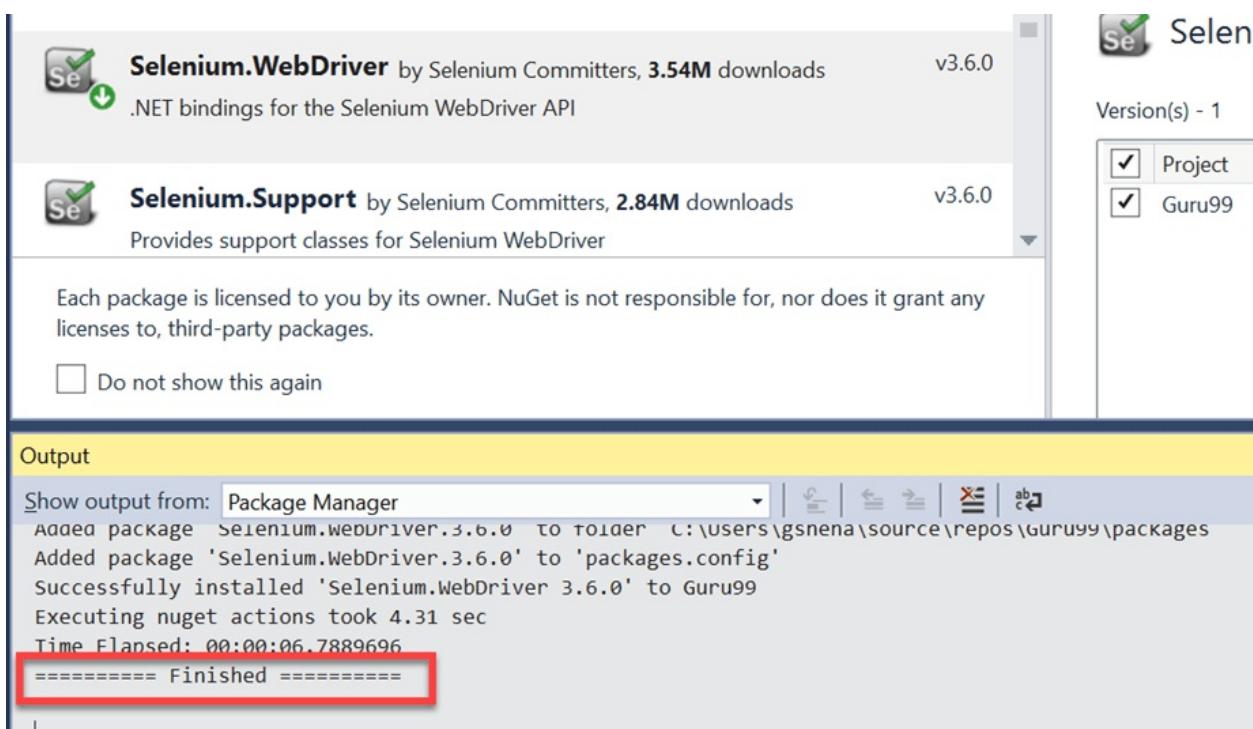


Step 3) Click on 'OK' button in the pop-up screen



Step 4) The below message will be displayed once the package is

successfully installed.



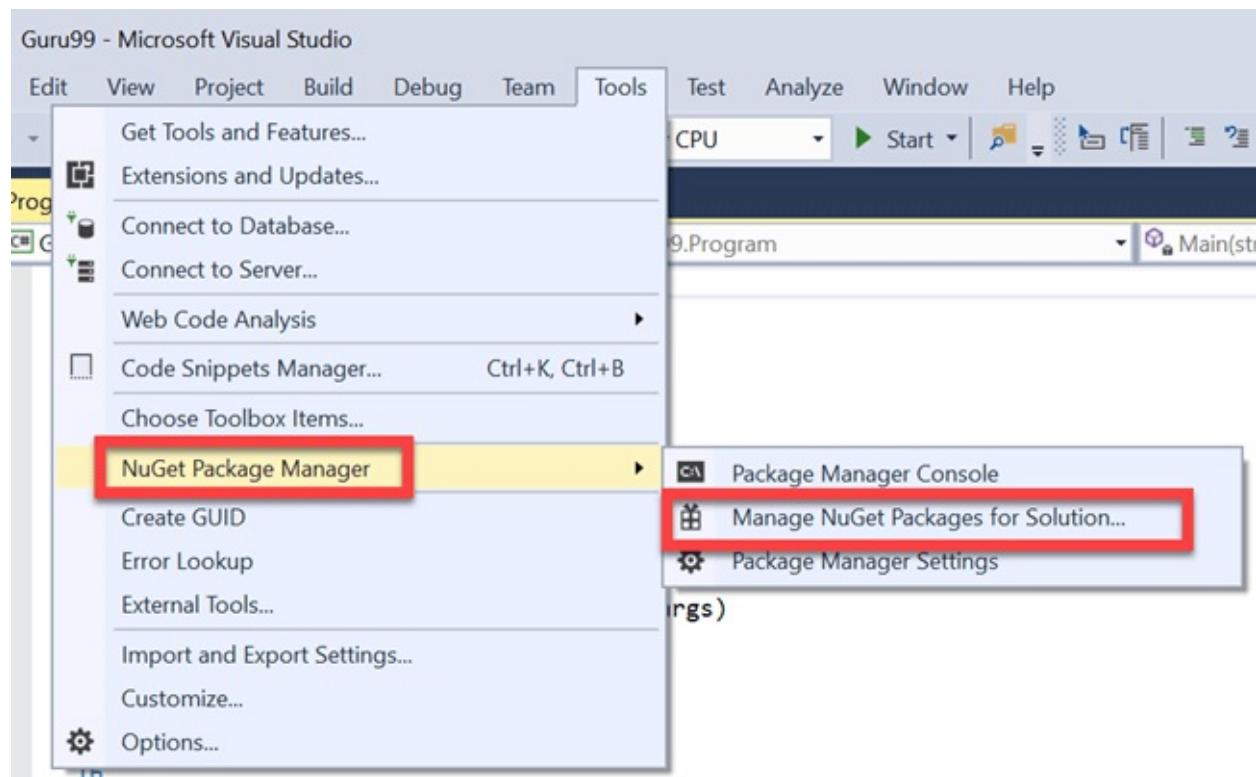
NUnit Framework: Overview

NUnit is the Unit Testing framework supported by Visual Studio and Selenium WebDriver. NUnit is the most widely used Unit Testing framework for .Net applications. NUnit presents the test results in a readable format and allows a tester to debug the automated tests.

We need to install NUnit Framework and NUnit Test Adapter onto Visual Studio inorder to use it.

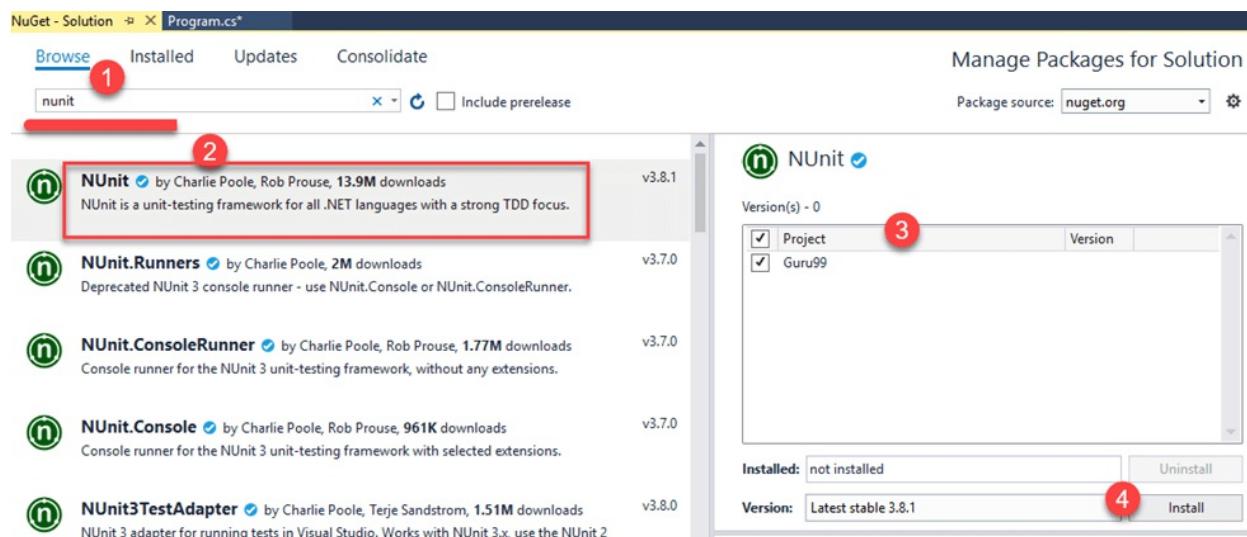
Steps to install NUnit Framework:

1. Navigate to Tools -> NuGet Package Manager -> Manage NuGet Packages for Solution

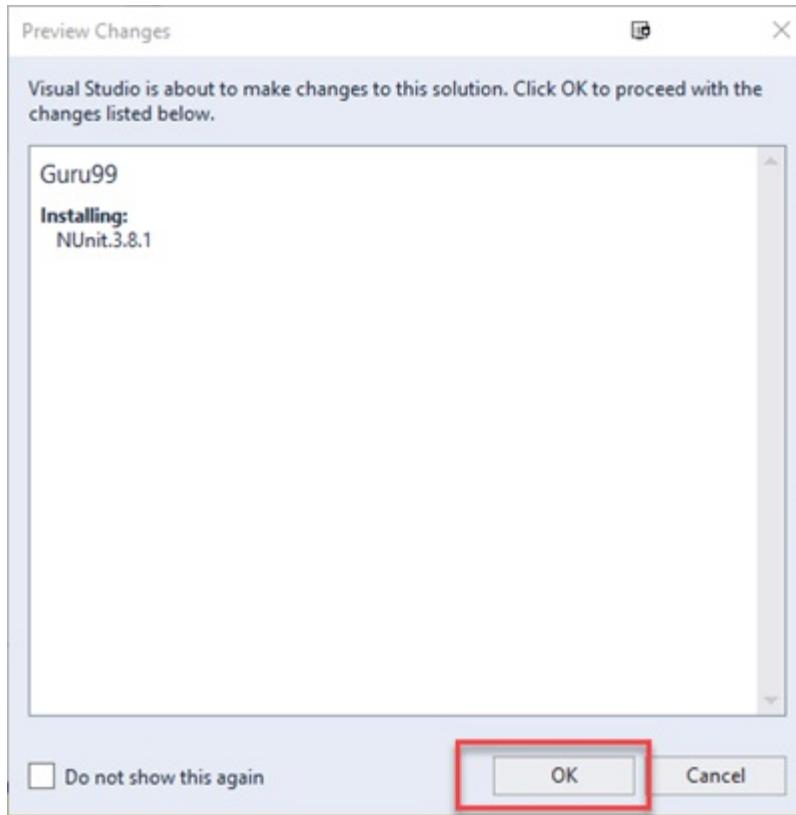


Step 2) In the next window

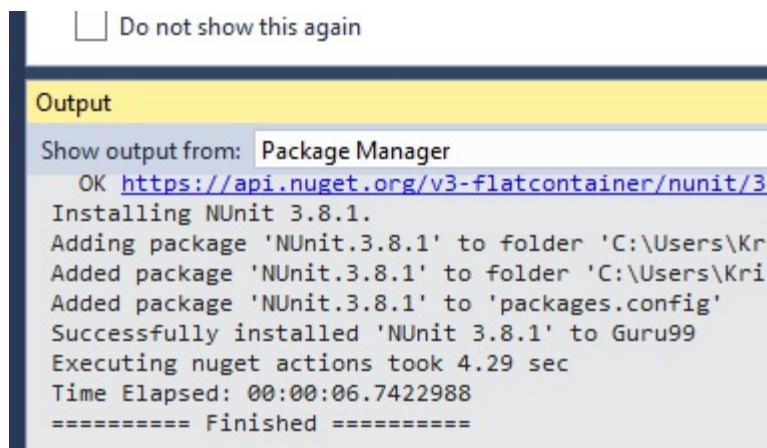
1. Search for NUnit
2. Select the search result
3. Select Project
4. Click Install



Step 3) The below popup will appear. Click on 'Ok' button.



Step 4) The below message will appear once the installation is complete.

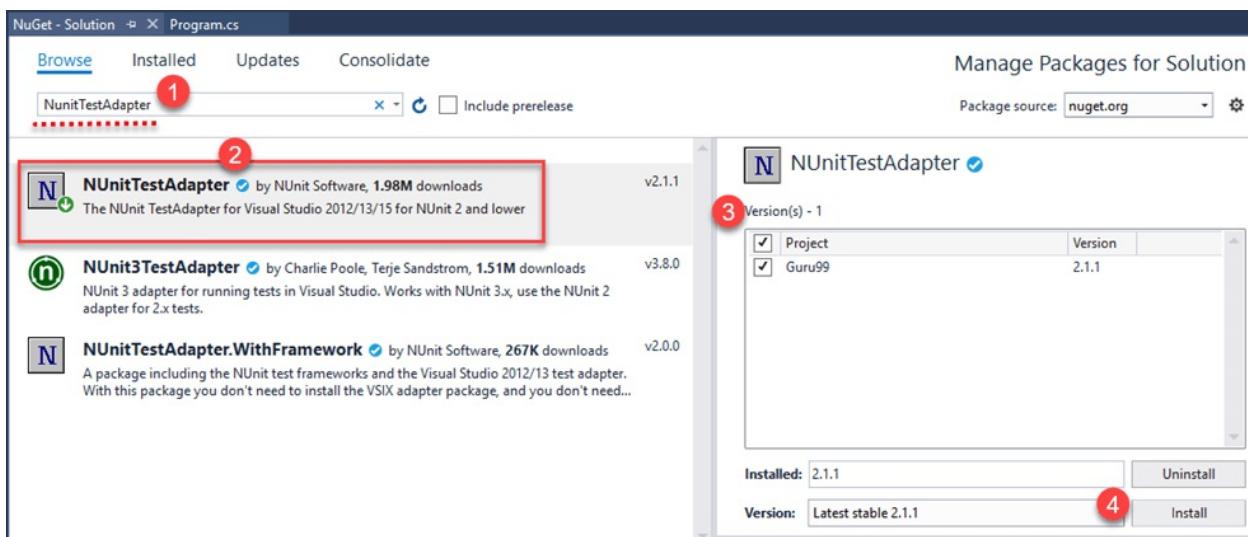


Steps to download NUNIT Test Adapter

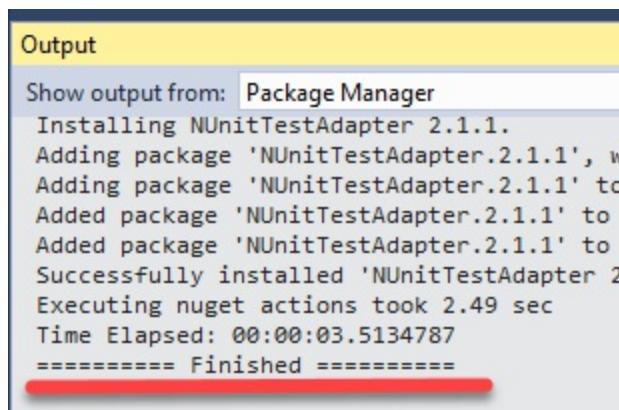
Please note that the below steps work only for 32-bit machines. For 64-bit machines, you need to download the 'NUnit3 Test Adapter' by following the same process as mentioned below.

Step 1) Navigate to Tools ->NuGet Package Manager -> Manage NuGet Packages for Solution. In that screen,

1. Search NUnitTestAdapter
2. Click Search Result
3. Select Project
4. Click Install



Step 2) Click OK on the confirmation pop-up. Once install is done you will see the following message-



The screenshot shows the 'Output' window from a development environment. The title bar says 'Output'. Below it, there is a dropdown menu labeled 'Show output from: Package Manager'. The main area displays the following log:

```
Show output from: Package Manager
Installing NUnitTestAdapter 2.1.1.
Adding package 'NUnitTestAdapter.2.1.1', w
Adding package 'NUnitTestAdapter.2.1.1' to
Added package 'NUnitTestAdapter.2.1.1' to
Added package 'NUnitTestAdapter.2.1.1' to
Successfully installed 'NUnitTestAdapter 2
Executing nuget actions took 2.49 sec
Time Elapsed: 00:00:03.5134787
===== Finished =====
```

Selenium and NUnit framework:

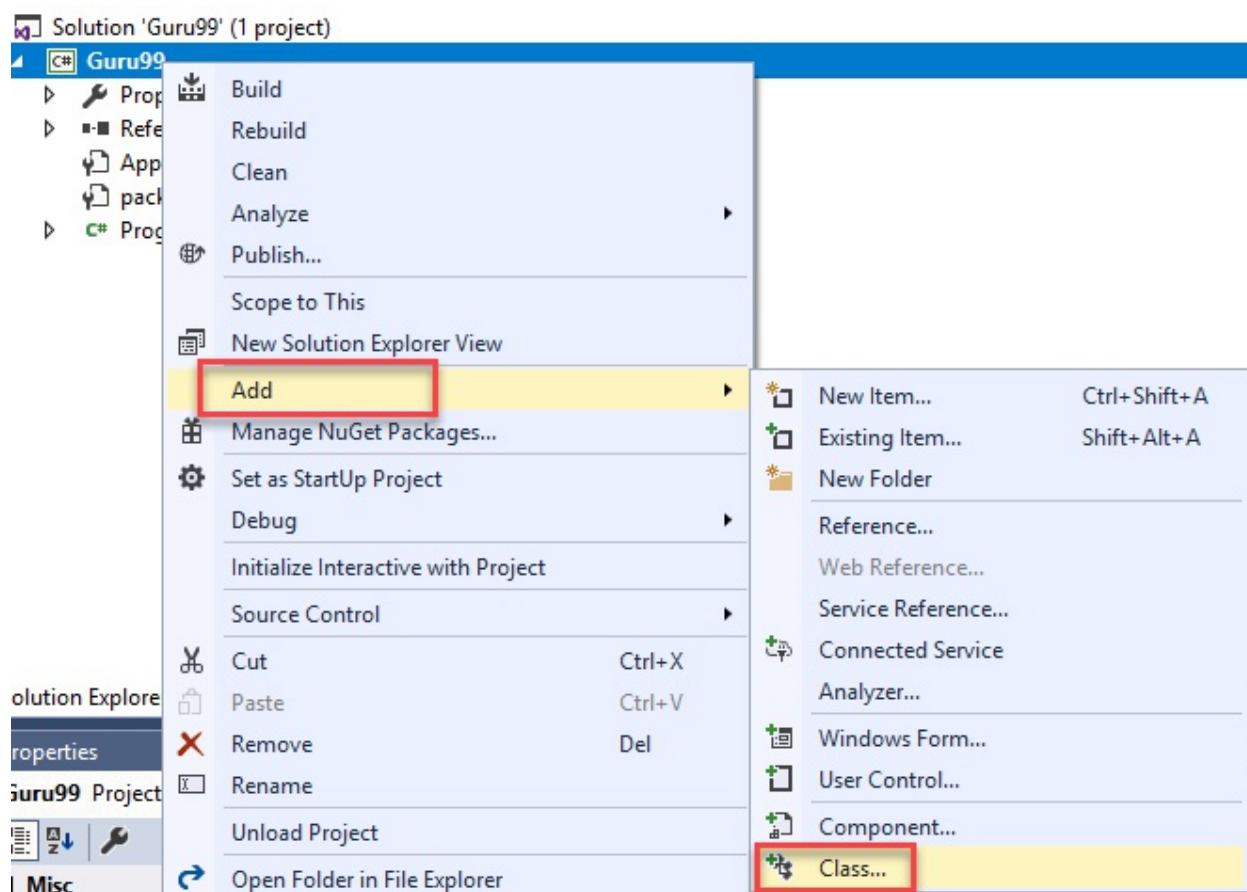
Integration of selenium with NUnit framework allows a tester to differentiate between various test classes. NUnit also allows testers to use annotations such as SetUp, Test, and TearDown to perform actions before and after running the test.

NUnit framework can be integrated with Selenium by creating a NUNIT test class and running the test class using NUNIT framework.

The below are the steps needed to create and run a test class using NUNIT framework.

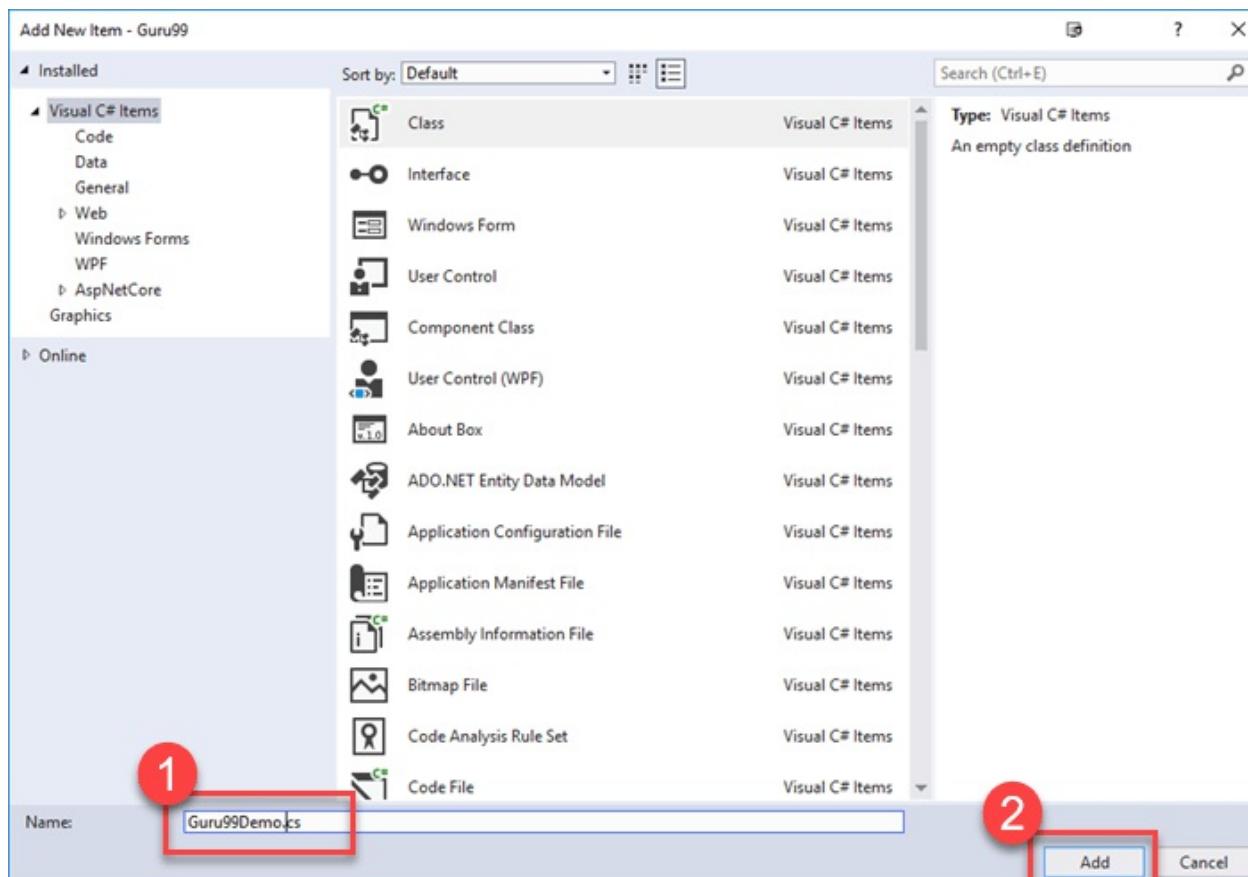
Steps to create a NUNIT Test class in Selenium:

Step 1) In the Solution Explorer, Right click on project > Add > Class

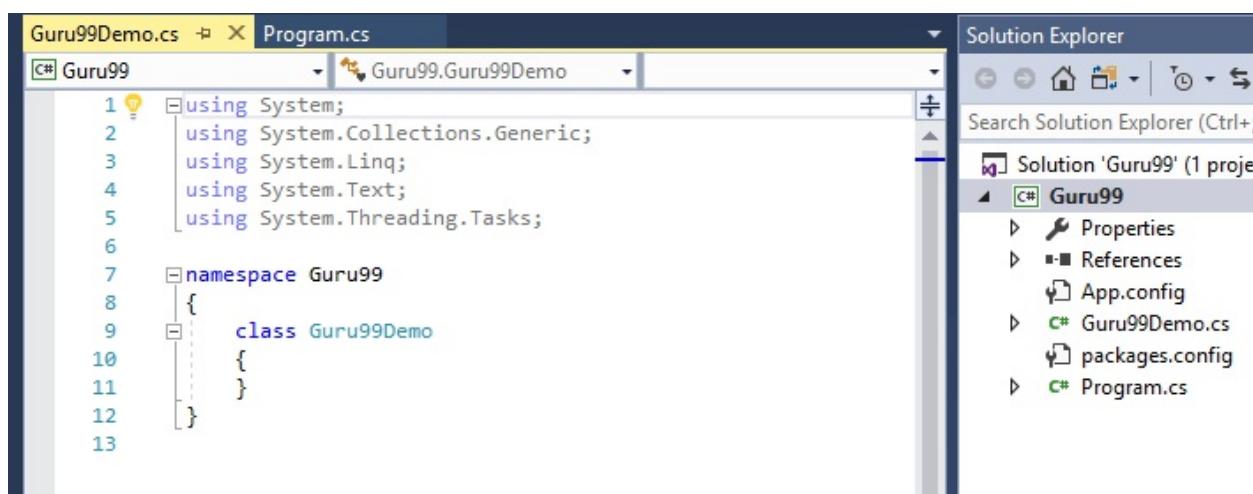


Step 2) Class creation window will appear.

1. Provide a name to the class
2. Click on Add button



Step 3) The below screen will appear.



Step 4) Add the following code to the created class. Please note that you need to specify the location of 'chromedriver.exe' file during chrome driver initialization.

```
using NUnit.Framework;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Firefox;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

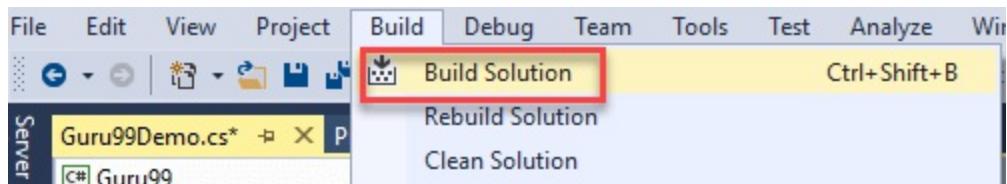
namespace Guru99Demo
{
    class Guru99Demo
    {
        IWebDriver driver;

        [SetUp]
        public void startBrowser()
        {
            driver = new ChromeDriver("D:\\3rdparty\\chrome");
        }

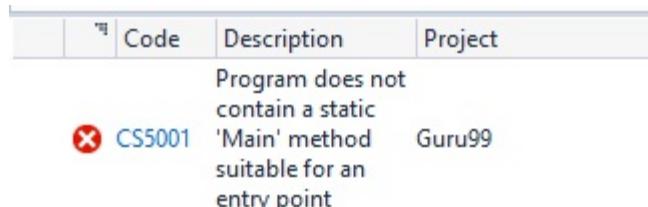
        [Test]
        public void test()
        {
            driver.Url = "http://www.google.co.in";
        }

        [TearDown]
        public void closeBrowser()
        {
            driver.Close();
        }
    }
}
```

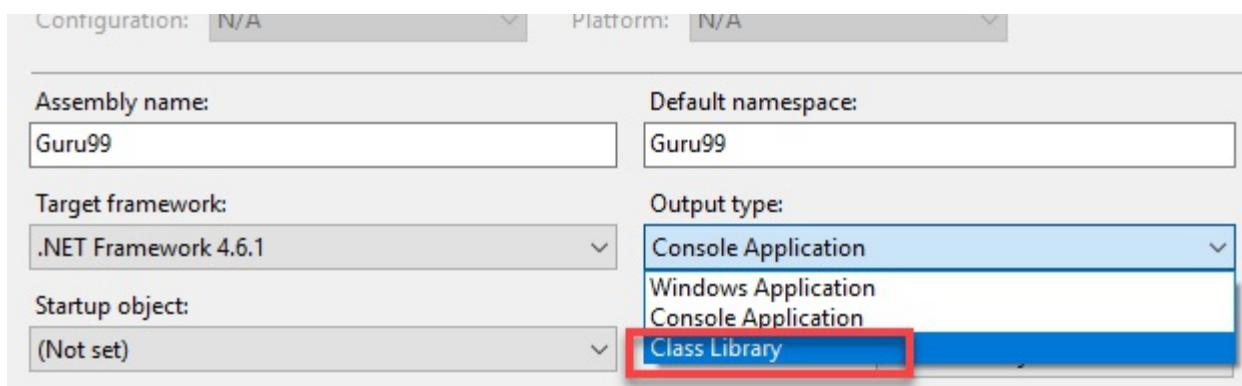
Step 4) Click on 'Build' -> 'Build Solution'



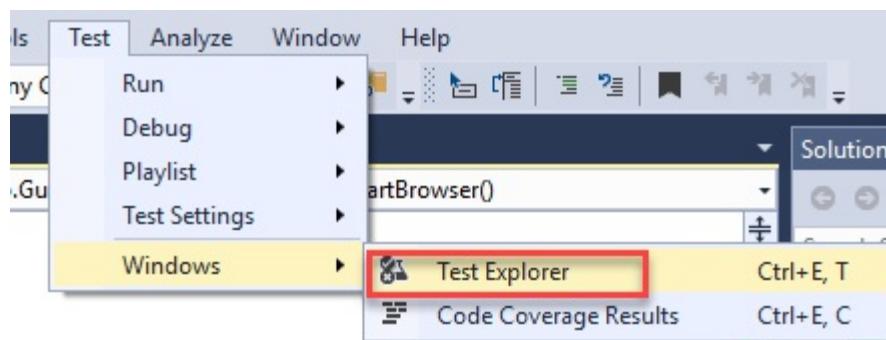
NOTE: You may get an error like "Does not contain a static 'main' method suitable for an entry point" when you build



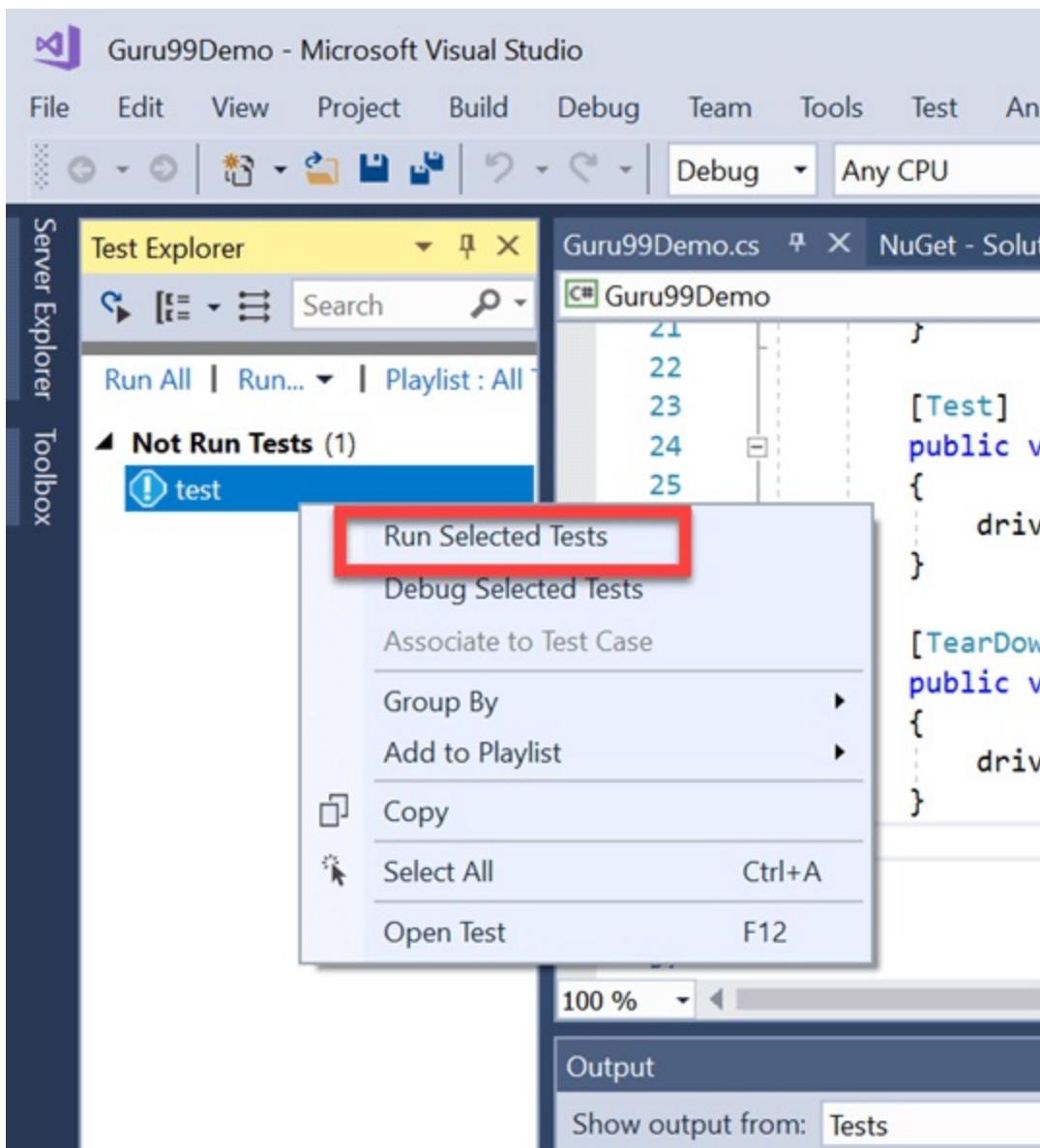
To resolve this Got to Project > Properties and change Output Type to "Class Library." The default is "Console Application."



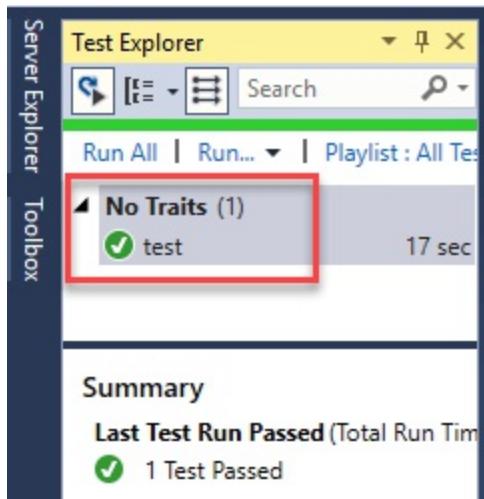
Step 5) Once the build is successful, we need to open the Test Explorer window. Click on Test -> Windows -> Test Explorer



Step 6) Test Explorer window opens with the list of available tests.
Right-click on Test Explorer and select Run Selected Tests



Step 7) Selenium must open the browser with specified URL and close the browser. Test case status will be changed to 'Pass' on the Test Explorer window.



Selenium WebDriver Commands in C#:

C# uses the interface 'IWebDriver' for browser interactions. The following are the category of commands available in C#.

1. Browser commands
2. Web Element commands
3. Dropdown commands

Let's study them one by one

Browser commands:

The following are the list of browser commands available in C#.

Command Name	Description	Syntax
Url Command	This command is used to open a specified URL in the browser.	<code>driver.Url = "https://www.guru99.com"</code>

Title Command	This command is used to retrieve the page title of the web page that is currently open	<code>String title = driver.Title</code>
PageSource Command	This command is used to retrieve the source code of web page that is currently open.	<code>String pageSource = driver.PageSource</code>
Close Command	This command is used to close the recently opened browser instance.	<code>driver.Close();</code>
Quit Command	This command is used to close all open browser instances	<code>driver.Quit();</code>
Back Command	This command is used to navigate to the previous page of browser history.	<code>driver.Navigate().Back();</code>
Forward Command	This command is used to navigate to the next page of browser history.	<code>driver.Navigate().Forward()</code>
Refresh Command	This command is used to perform browser refresh.	<code>driver.Navigate().Refresh()</code>

Webelement Commands:

A Webelement represents all the elements on a web page. They are represented by HTML tags. Each of the buttons, textboxes, links, images, tables, and frames fall under Webelements. Operations on web elements can be triggered using the IWebelement interface. To interact with a Webelement, we need to find the element on the webpage and then perform operations on it. Tools like Firebug and Firepath can be used to identify the Xpath of Webelement.

The following are the list of Webelement commands available in C#.

Command Name	Description	Syntax
Click command	This command is used to click on a Webelement. For the element to be clickable, the element must be visible on the webpage. This command is used for checkbox and radio button operations as well.	<pre>IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); element.Click();</pre>
Clear command	This command is specifically used for clearing the existing contents of textboxes.	<pre>IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); element.Clear();</pre>
SendKeys command	This command is used to input a value onto text boxes. The value to be entered must be passed as a parameter to	<pre>IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); element.SendKeys("guru99");</pre>
Displayed command	This command is used to identify if a specific element is displayed on the webpage. This command returns a Boolean value; true or false depending on the visibility of web element.	<pre>IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); Boolean status = element.Displayed;</pre>
Enabled command	This command is used to identify if a particular web element is enabled on the web page. This command returns a Boolean value; true or false as a result.	<pre>IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); Boolean status = element.Enabled;</pre>

Selected command	This command is used to identify if a particular web element is selected. This command is used for checkboxes, radio buttons, and select operations.	IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); Boolean status = element.Selected;
Submit command:	This command is similar to click command, The difference lies in whether the HTML form has a button with the type Submit. While the click command clicks on any button, submit command clicks on the only the buttons with type submit.	IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); element.submit();
Text command	This command returns the innertext of a Webelement. This command returns a string value as a result.	IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); String text=element.Text;
TagName command	This command returns the HTML tag of a web element. It returns a string value as the result.	IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); String tagName = element.TagName;
GetCSSValue Command:	This method is used to return the color of a web element on the form of a rgba string (Red, Green, Blue, and Alpha).	IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); String color = element.getCSSValue; Output- If the color of element is red, output would be rgba(255,0,0,1)

Dropdown Commands:

Dropdown operations in C# can be achieved using the SelectElement class.

The following are the various dropdown operations available in C#.

Command Name	Description	Syntax
SelectByText Command	This command selects an option of a dropdown based on the text of the option.	<pre>IWebElement element = driver.FindElement(By.xpath("xpath of WebElement")); SelectElement select = new SelectElement(element); select.SelectByText("Guru99");</pre>
SelectByIndex Command	This command is used to select an option based on its index. Index of dropdown starts at 0.	<pre>IWebElement element = driver.FindElement(By.xpath("xpath of WebElement")); SelectElement select = new SelectElement(element); select.SelectByIndex("4");</pre>
SelectByValue Command	This command is used to select an option based on its option value.	<pre>IWebElement element = driver.FindElement(By.xpath("xpath of WebElement")); SelectElement select = new SelectElement(element); select.SelectByValue("Guru99");</pre>
Options Command	This command is used to retrieve the list of options displayed in a dropdown.	<pre>IWebElement element = driver.FindElement(By.xpath("xpath of WebElement")); SelectElement select = new SelectElement(element); List<IWebElement> options = select.Options; int size = options.Count; for(int i=0;i<options.size();i++) { String value = size.elementAt(i).Text; Console.WriteLine(value); }</pre> <p>The above code prints all the options onto console within a</p>

		dropdown.
IsMultiple command	This command is used to identify if a dropdown is a multi select dropdown; A multi select dropdown enables user to select more than one option in a dropdown at a time. This command returns a Boolean value.	IWebElement element = driver.FindElement(By.xpath("xpath of WebElement")); SelectElement select = new SelectElement(element); Boolean status = select.IsMultiple();
DeSelectAll command	This command is used in multi select dropdowns. It clears the options that have already been selected.	IWebElement element = driver.FindElement(By.xpath("xpath of WebElement")); SelectElement select = new SelectElement(element); select.DeSelectAll();
DeSelectByIndex command	This command deselects an already selected value using its index.	IWebElement element = driver.FindElement(By.xpath("xpath of WebElement")); SelectElement select = new SelectElement(element); select.DeSelectByIndex("4");
DeSelectByValue command	This command deselects an already selected value using its value.	IWebElement element = driver.FindElement(By.xpath("xpath of WebElement")); SelectElement select = new SelectElement(element); select.DeSelectByValue("Guru99");
DeSelectByText command	This command deselects an already selected value using its text.	IWebElement element = driver.FindElement(By.xpath("xpath of WebElement")); SelectElement select = new SelectElement(element); select.DeSelectByText("Guru99");

Code Samples

Example 1: Click on a link using XPATH Locator:

Test Scenario:

1. Navigate to Demo Guru99 web page -
<http://demo.guru99.com/test/guru99home/>
2. Maximize the window
3. Click on the 'Testing' menu
4. Close the browser

```
using NUnit.Framework;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Guru99Demo
{
    class CSS
    {
        IWebDriver m_driver;

        [Test]
        public void cssDemo()
        {
            m_driver = new ChromeDriver("D:\\\\3rdparty\\\\chrome");
            m_driver.Url =
"http://demo.guru99.com/test/guru99home/";
            m_driver.Manage().Window.Maximize();
            IWebElement link =
m_driver.FindElement(By.XPath(".//*[@id='rt-
header']//div[2]/div/ul/li[2]/a"));
            link.Click();
            m_driver.Close();
        }
    }
}
```

```
    }  
}
```

Example 2: Entering data into TextBox and Click on a button using XPATH locator:

Test Scenario:

1. Navigate to Guru 99 demo page -
<http://demo.guru99.com/test/guru99home/>
2. Enter data into email text box
3. Click on sign up button

```
using NUnit.Framework;  
using OpenQA.Selenium;  
using OpenQA.Selenium.Chrome;  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace Guru99Demo  
{  
    class CSS  
    {  
        IWebDriver m_driver;  
  
        [Test]  
        public void cssDemo()  
        {  
            m_driver = new ChromeDriver("G:\\\\");  
            m_driver.Url =  
"http://demo.guru99.com/test/guru99home/";  
            m_driver.Manage().Window.Maximize();  
  
            // Store locator values of email text  
box and sign up button
```

```

        IWebElement emailTextBox =
m_driver.FindElement(By.XPath(".//*[@id='philadelphia-field-
email']"));
        IWebElement signUpButton =
m_driver.FindElement(By.XPath(".//*[@id='philadelphia-field-
submit']"));

        emailTextBox.SendKeys("test123@gmail.com");
        signUpButton.Click();

    }
}
}

```

Example 3: Entering data into TextBox and Click on a button using CSS locator:

Test Scenario:

1. Navigate to Guru 99 demo page -
<http://demo.guru99.com/test/guru99home/>
2. Enter data into email text box
3. Click on sign up button

```

using NUnit.Framework;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Guru99Demo
{
    class CSS
    {
        IWebDriver m_driver;

```

```

[Test]
    public void cssDemo()
{
    m_driver = new ChromeDriver("G:\\\\");
    m_driver.Url =
"http://demo.guru99.com/test/guru99home/";
    m_driver.Manage().Window.Maximize();

                // Store locator values of email text
box and sign up button
    IWebElement emailTextBox =
m_driver.FindElement(By.CssSelector("input[id=philadelphia-
field-email]"));
    IWebElement signUpButton =
m_driver.FindElement(By.CssSelector("input[id=philadelphia-
field-submit]"));

    emailTextBox.SendKeys("test123@gmail.com");
    signUpButton.Click();

}
}
}

```

Example 4: Select a value in the dropdown:

Test Scenario:

1. Navigate to Guru 99 demo page -
<http://demo.guru99.com/test/guru99home/>
2. Click on SAP link
3. Enter data onto name and email text boxes
4. Select a value from the Course dropdown
5. Close the browser

```

using NUnit.Framework;
using OpenQA.Selenium;

```

```

using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Support.UI;

namespace Guru99Demo
{
    class TestSelect
    {
        IWebDriver m_driver;

        [Test]
        public void selectDemo()
        {
            m_driver = new ChromeDriver("G:\\\\");
            m_driver.Url =
"http://demo.guru99.com/test/guru99home/";
            m_driver.Manage().Window.Maximize();

            IWebElement course =
m_driver.FindElement(By.XPath(".//*[@id='awf_field-91977689']"));
            var selectTest = new
SelectElement(course);
            // Select a value from the dropdown
            selectTest.SelectByValue("sap-abap");

        }
    }
}

```

Summary:

- In order to use Selenium with C#, you need to install Visual Studio.
- NUnit is the Unit Testing framework supported by Visual Studio and Selenium WebDriver
- We need to install NUnit Framework and NUnit Test Adapter onto Visual Studio inorder to use it.

- NUnit framework can be integrated with Selenium by creating a NUnit test class and running the test class using NUnit framework.
- NUnit also allows testers to use annotations such as SetUp, Test, and TearDown to perform actions before and after running the test.
- Selenium WebDriver Commands can be categorized into Browser commands, WebElement Commands and Dropdown Commands.

Chapter 64: Creating Object Repository in Selenium WebDriver

What is an Object Repository?

An object repository is a common storage location for all objects. In Selenium WebDriver context, objects would typically be the locators used to uniquely identify web elements.

The major advantage of using object repository is the segregation of objects from test cases. If the locator value of one webelement changes, only the object repository needs to be changed rather than making changes in all test cases in which the locator has been used. Maintaining an object repository increases the modularity of framework implementation.

Types of Object Repositories in Selenium Web Driver

Selenium WebDriver does not offer an in-built object repository by default. However, object repositories can be built using the key-value pair approach wherein the key refers to the name given to the object and value refers to the properties used to uniquely identify an object within the web page.

The following are the types of object repositories that can be created in

Selenium WebDriver.

1. Object Repository using Properties file
2. Object Repository using XML file

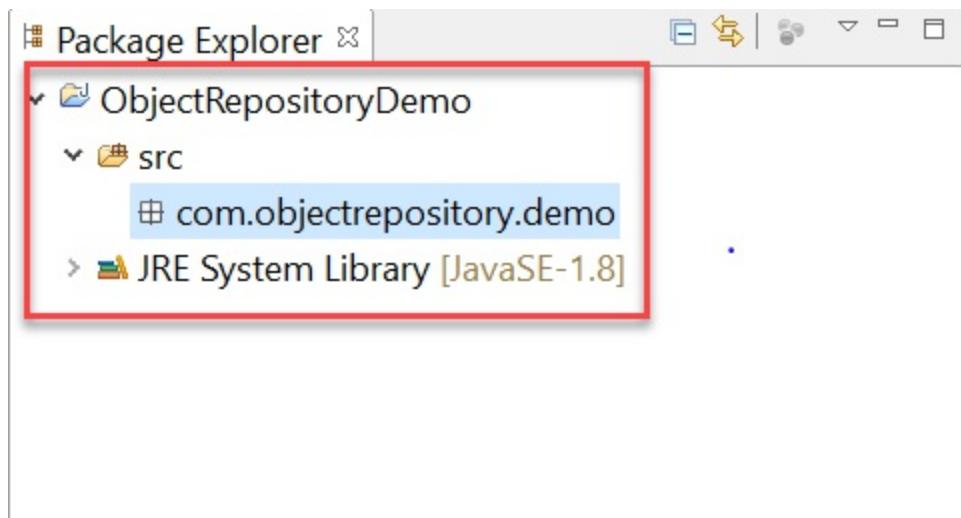
Selenium Web Driver Object repository using Properties file

In this approach, properties file is a text file wherein data is stored on the form of key-value pairs. The below tutorial will address the following topics.

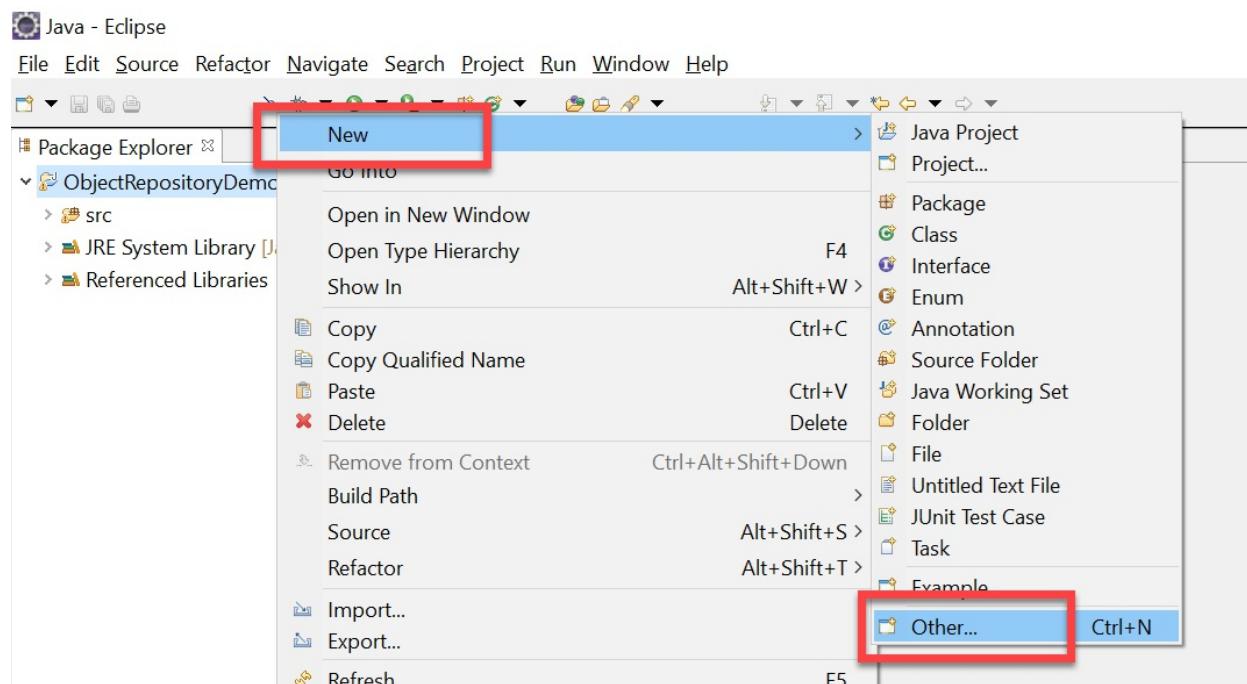
- Creating a properties file in eclipse
- Storing data onto properties file
- Reading data from properties file
- Using properties file in test scripts

Step 1) Creating a properties file in eclipse

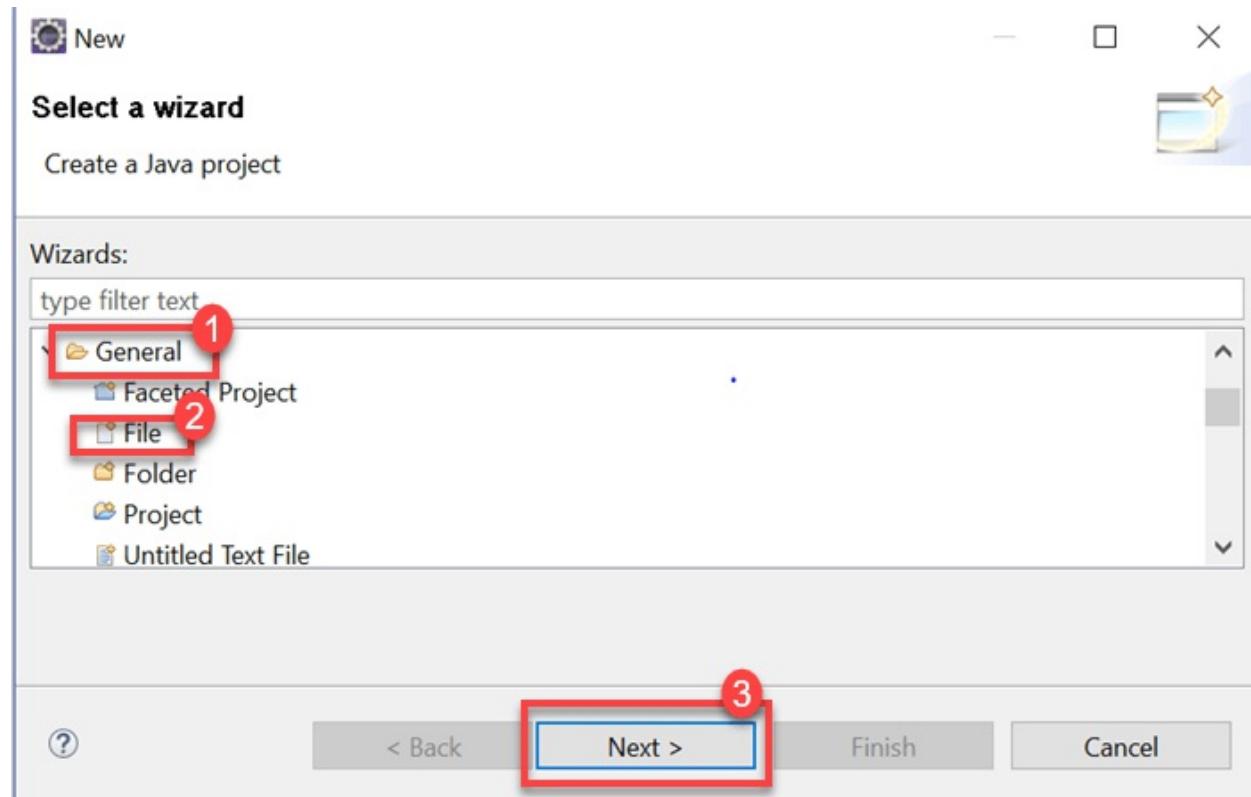
1. To start with, the below java project structure needs to be created in eclipse. Project name and package name can be any valid names.



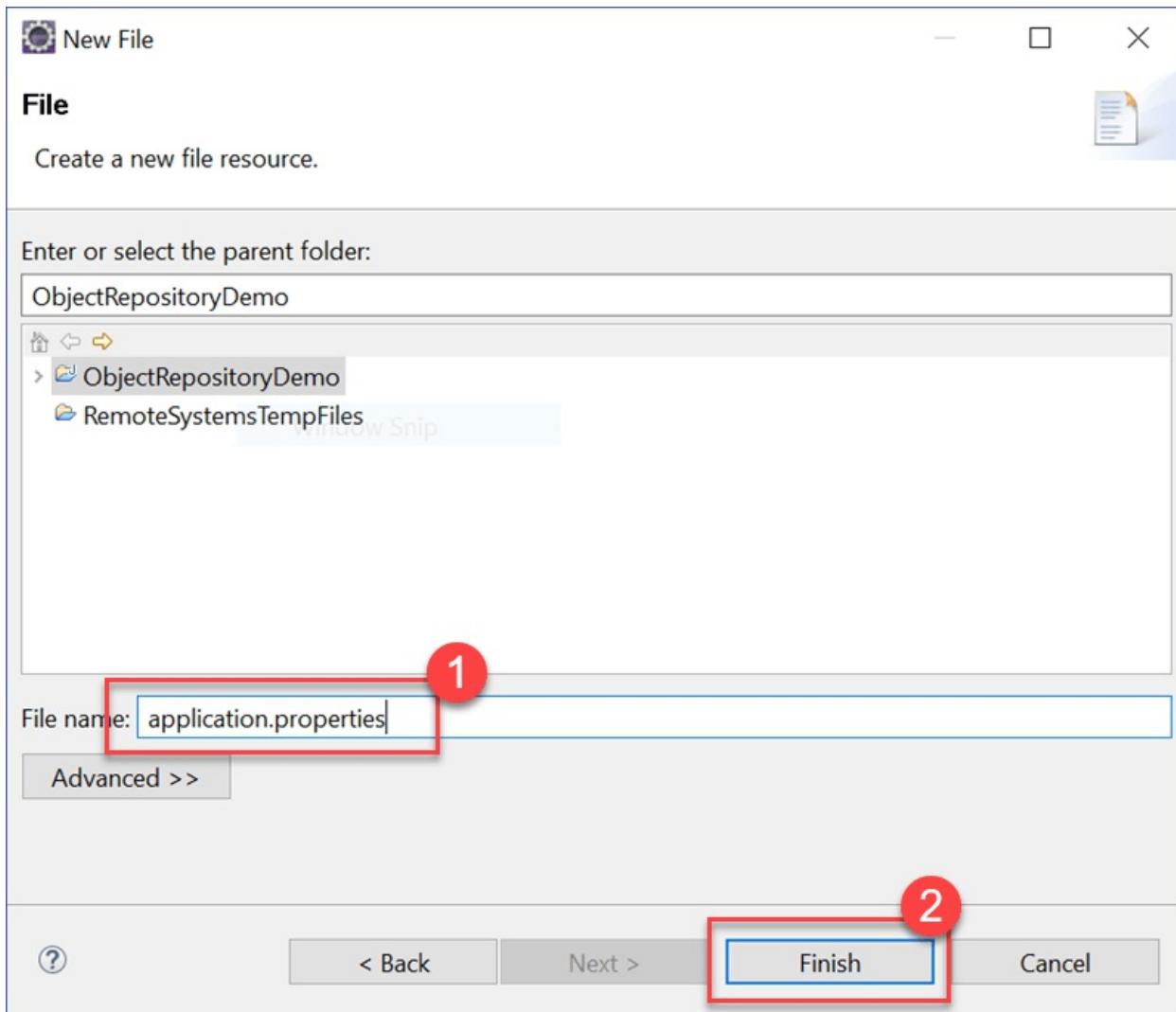
2. Right-click on the main project folder and Select New-> Other



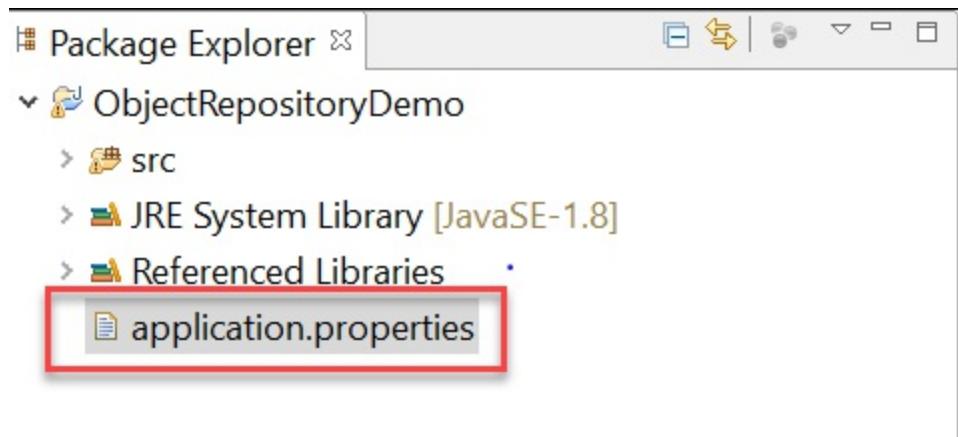
3. In the next window, select General -> File and click on 'Next' button



4. Provide a valid file name with the extension '.properties' on the new file resource window and click on 'Finish' button



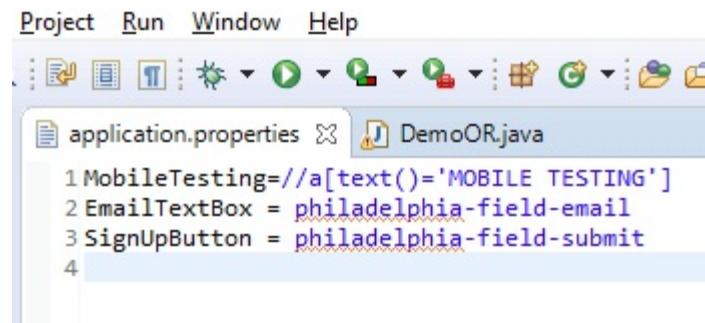
5. A file named 'application.properties' must be displayed on Project Structure



Step 2) Storing data onto properties file

1. Data is stored in properties file in the form of key-value pairs, with the key being unique across the file.
2. We will try to use the properties file to identify webelements using locator values.
3. Open application.properties file in Eclipse and store the following data

```
MobileTesting=//a[text()='MOBILE TESTING']
EmailTextBox = philadelphia-field-email
SignUpButton = philadelphia-field-submit
```



4) For this tutorial, the following demo website is being used:
<http://demo.guru99.com/test/guru99home/>. Here is Test scenario:

- Click on Mobile Testing link using XPATH
- Navigate back
- Enter data onto email textbox using ID
- Click on the Sign Up button using ID

Step 3) Reading data from properties file

1. Reading data from properties file can be done using the built-in Properties class provided in java.util package.
2. Initially, an object of Properties class need to be created as below

```
Properties obj = new Properties();
```

3. We need to create an object of FileInputStream class with the path to properties file

```
FileInputStream objfile = new  
FileInputStream(System.getProperty("user.dir")+"\\application.pr  
operties");
```

4. Reading data from properties file can be done using load method offered by Properties class in java. The below code demonstrates the usage of load method.

```
Properties obj = new Properties();  
FileInputStream objfile = new  
FileInputStream(System.getProperty("user.dir")+"\\application.pr  
operties");  
obj.load(objfile);  
String mobileTesting = obj.getProperty("MobileTesting");
```

The string 'mobileTesting' will contain the XPATH to identify the Mobile Testing link within the webpage.

Step 4) Using properties file in test scripts

Properties file can be used in test scripts by reading data from a properties file and passing the data as a parameter to the findElement method. The below code demonstrates the usage of data read from properties file in test scripts.

```
driver.findElement(By.xpath(obj.getProperty("MobileTesting"))).c  
lick();  
driver.findElement(By.id(obj.getProperty("EmailTextBox"))).sendKeys("testguru99@gmail.com");  
driver.findElement(By.id(obj.getProperty("SignUpButton"))).click();
```

The below is the complete code used for the above test scenario.

```
package com.objectrepository.demo;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Properties;

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

public class DemoOR {

    public static void main(String[] args) throws IOException {

        // Create WebDriver Instance
        WebDriver driver;

        System.setProperty("webdriver.chrome.driver", "G:\\chromedriver.exe");
        driver = new ChromeDriver();
        driver.get("http://demo.guru99.com/test/guru99home/");
        driver.manage().window().maximize();
        // Load the properties File
        Properties obj = new Properties();
        FileInputStream objfile = new
FileInputStream(System.getProperty("user.dir")+"\\application.properties");
        obj.load(objfile);
        // Navigate to link Mobile Testing and Back

        driver.findElement(By.xpath(obj.getProperty("MobileTesting"))).click();
        driver.navigate().back();
        // Enter Data into Form

        driver.findElement(By.id(obj.getProperty("EmailTextBox"))).sendKeys("testguru99@gmail.com");

        driver.findElement(By.id(obj.getProperty("SignUpButton"))).click
```

```
();  
}  
  
}
```

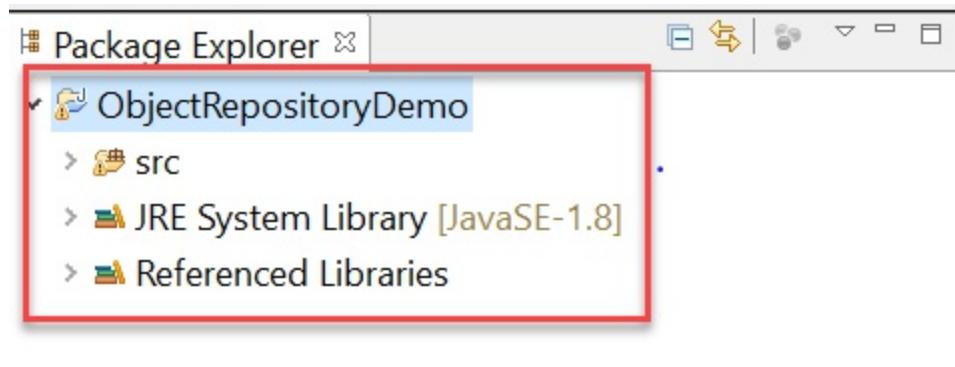
Selenium WebDriver Object Repository Using XML File

XML stands for Extensible Markup Language. An XML File uses Document Object Model(DOM) as the basic structure. XML File format will replicate the HTML format upon which the webpage is constructed. Below is the list of topics that will be covered.

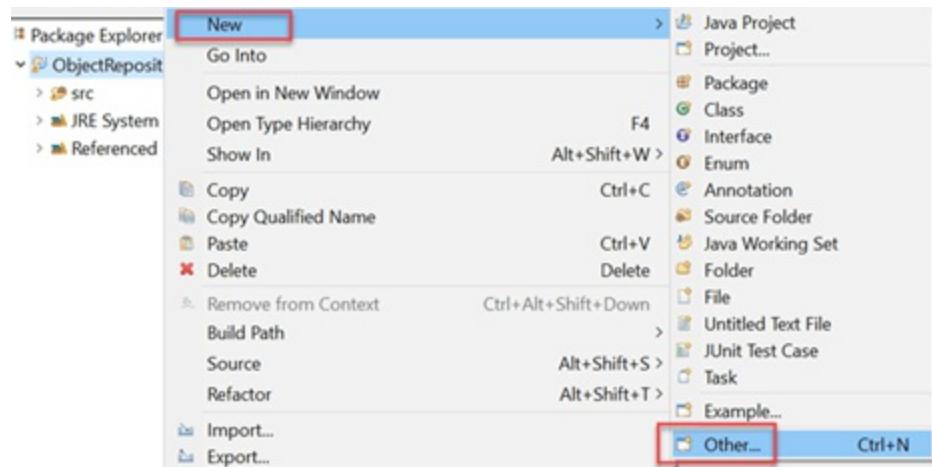
- Creating an XML file in eclipse
- Storing data onto XML file
- Reading data from XML file
- Using XML file in test scripts

Step 1) Creating an XML file in eclipse

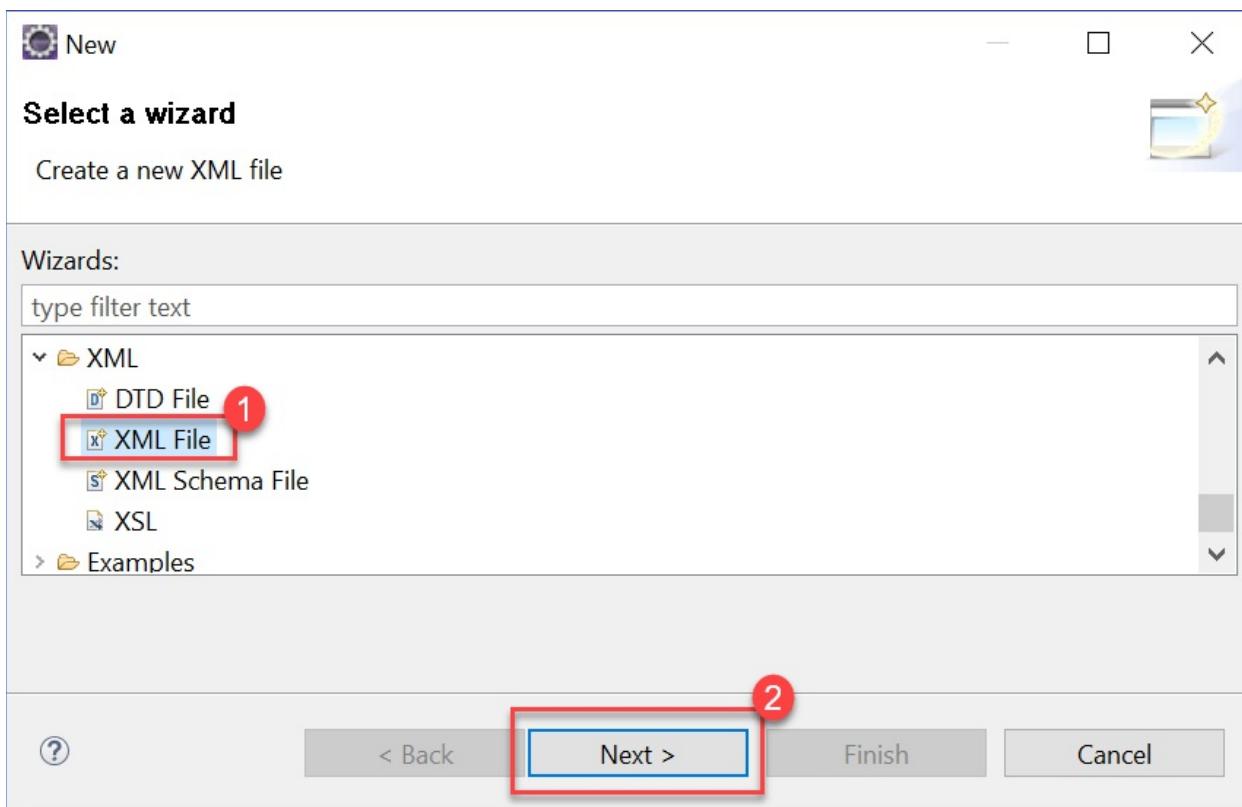
1. The below java project structure needs to be created in Eclipse.



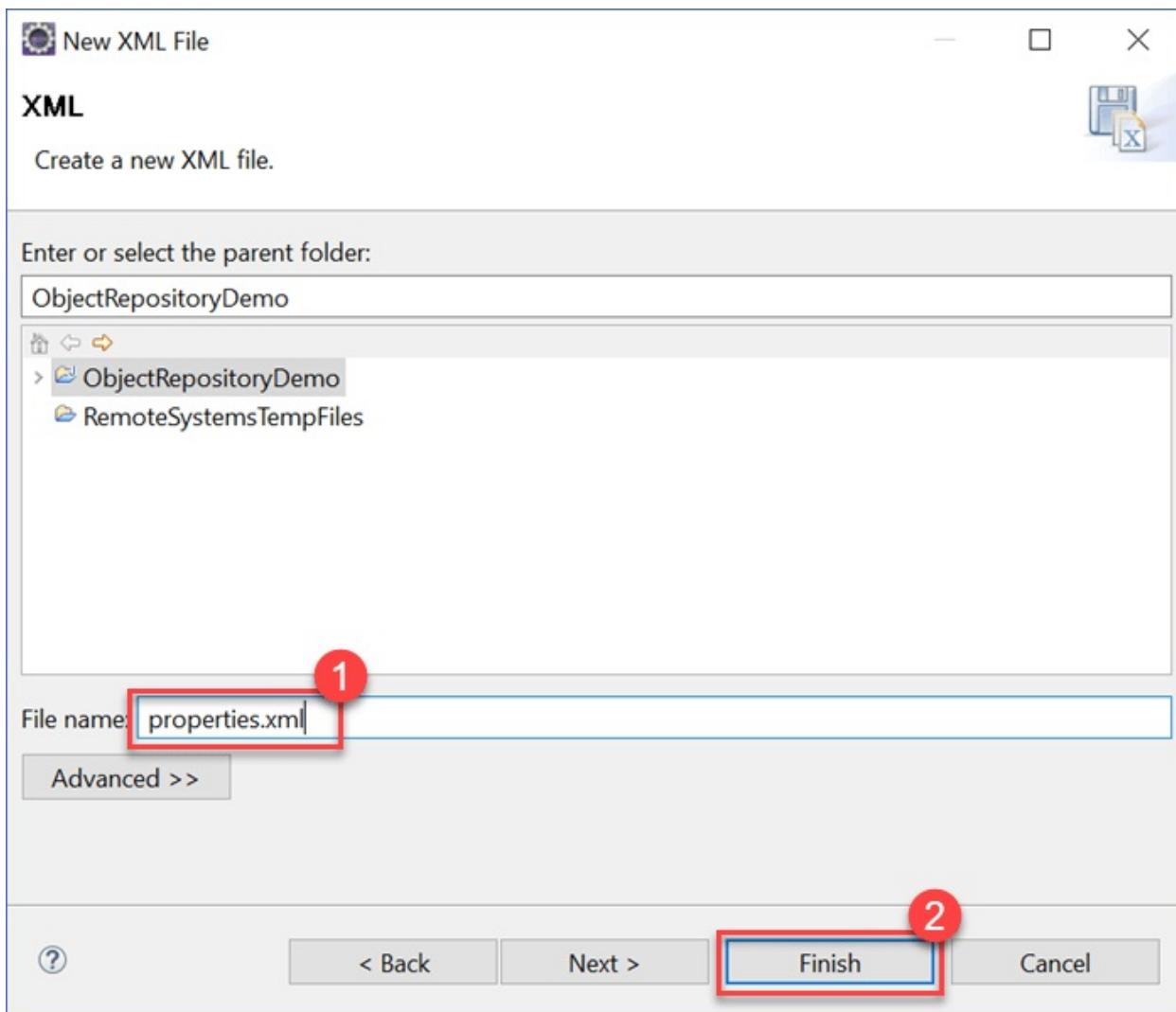
2. Right-click on the project folder, select New -> Other



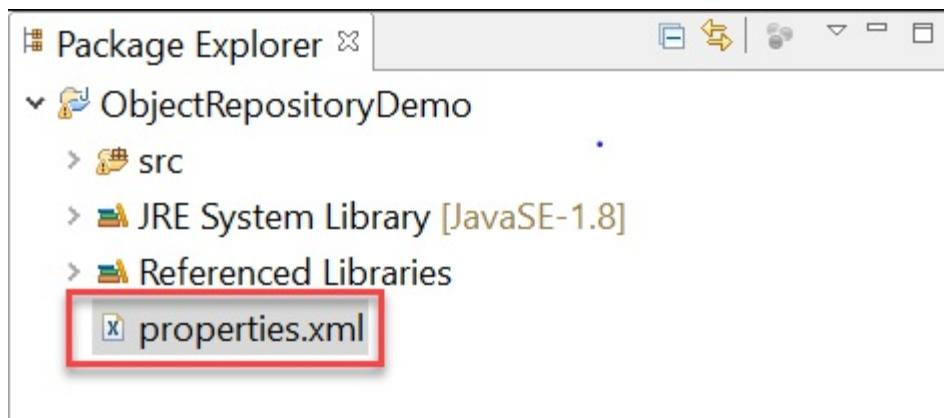
3. Select the XML File within the XML folder and click on 'Next' button



4. Enter a valid XML File name and click on 'Finish' button



5. An XML file will be added to the project folder as shown below



Step 2) Storing data onto XML file

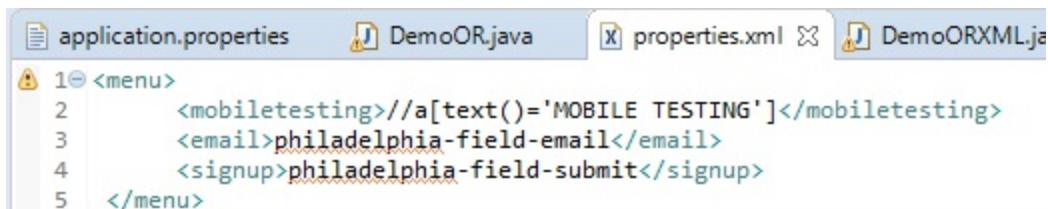
Data can be stored in XML file in the form of Document Object Model (DOM). For simplicity sake, we can use the below test scenario as an example.

- Click on Mobile Testing link using XPATH
- Navigate Back to Home page
- Enter data onto email textbox using ID
- Click on the Sign Up button using ID

The below is the format of XML File to be used.

```
<menu>
    <mobiletesting>//a[text()='MOBILE TESTING']
</mobiletesting>
    <email> philadelphia-field-email</email>
    <signup> philadelphia-field-submit </signup>
</menu>
```

Store the above XML code in properties.xml



In the design tab you will see

Node	Content
menu	
mobiletesting	//a[text()='MOBILE TESTING']
email	philadelphia-field-email
signup	philadelphia-field-submit

Step 3) Reading data from XML file

1. Reading data from XML file can be accomplished using the built-in 'dom4j' class in java. Please note that you need to add the below JAR files into the buildpath of your project before proceeding with the code.

- jaxen.jar
- dom4j-1.6.jar

2. Below is the code to read data from XML file.

```
File inputFile = new
File(System.getProperty("user.dir") +"\\properties.xml");
SAXReader saxReader = new SAXReader();
Document document = saxReader.read(inputFile);
String mobileTesting =
document.selectSingleNode("//menu/mobiletesting").getText();
String emailTextBox =
document.selectSingleNode("//menu/email").getText();
String signUpButton =
document.selectSingleNode("//menu/signup").getText();
```

3. Initially, we need to create a File object and pass it as a parameter to the 'read' method of SAXReader class. Once the XML file data is read successfully, we can access individual nodes of XML document using the 'selectSingleNode' method.

Step 4) Using XML file in test scripts

XML file can be used in test scripts by reading data from XML file and passing the data as parameter to the findElement method. The below code demonstrates the usage of data read from XML file in test scripts.

```
driver.findElement(By.xpath(mobileTesting)).click();
driver.findElement(By.id(emailTextBox)).sendKeys("testguru99@gma
il.com");
driver.findElement(By.id(signUpButton)).click();
```

The below code demonstrates the use of XML file in selenium WebDriver

```
package com.objectrepository.demo;
import java.io.*;
import java.util.*;
import org.dom4j.*;
import org.dom4j.io.SAXReader;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class DemoORXML {

    public static void main(String[] args) throws DocumentException {
        // Creating WebDriver Instance
        WebDriver driver;

        System.setProperty("webdriver.chrome.driver", "G:\\chromedriver.e
xe");
        driver = new ChromeDriver();
        driver.get("http://demo.guru99.com/test/guru99home/");
        driver.manage().window().maximize();
        // Reading XML File
        File inputFile = new File(System.getProperty("user.dir")
+"\\properties.xml");
        SAXReader saxReader = new SAXReader();
        Document document = saxReader.read(inputFile);
```

```
String mobileTesting =
document.selectSingleNode("//menu/mobiletesting").getText();
String emailTextBox =
document.selectSingleNode("//menu/email").getText();
String signUpButton =
document.selectSingleNode("//menu/signup").getText();

//Navigating to Mobile Testing and back
driver.findElement(By.xpath(mobileTesting)).click();
driver.navigate().back();
//Entering Form Data
driver.findElement(By.id(emailTextBox)).sendKeys("testguru99@gma
il.com");
driver.findElement(By.id(signUpButton)).click();

}
}
```

Summary:

- An object repository is a common storage location for all objects
- Selenium WebDriver does not offer an in-built object repository by default
- You can create 2 Types of Object Repository in Selenium
 1. Object Repository using Properties file
 2. Object Repository using XML file
- The properties file is a text file wherein data is stored on the form of key-value pairs
- XML File format will replicate the HTML format upon which the webpage is constructed.

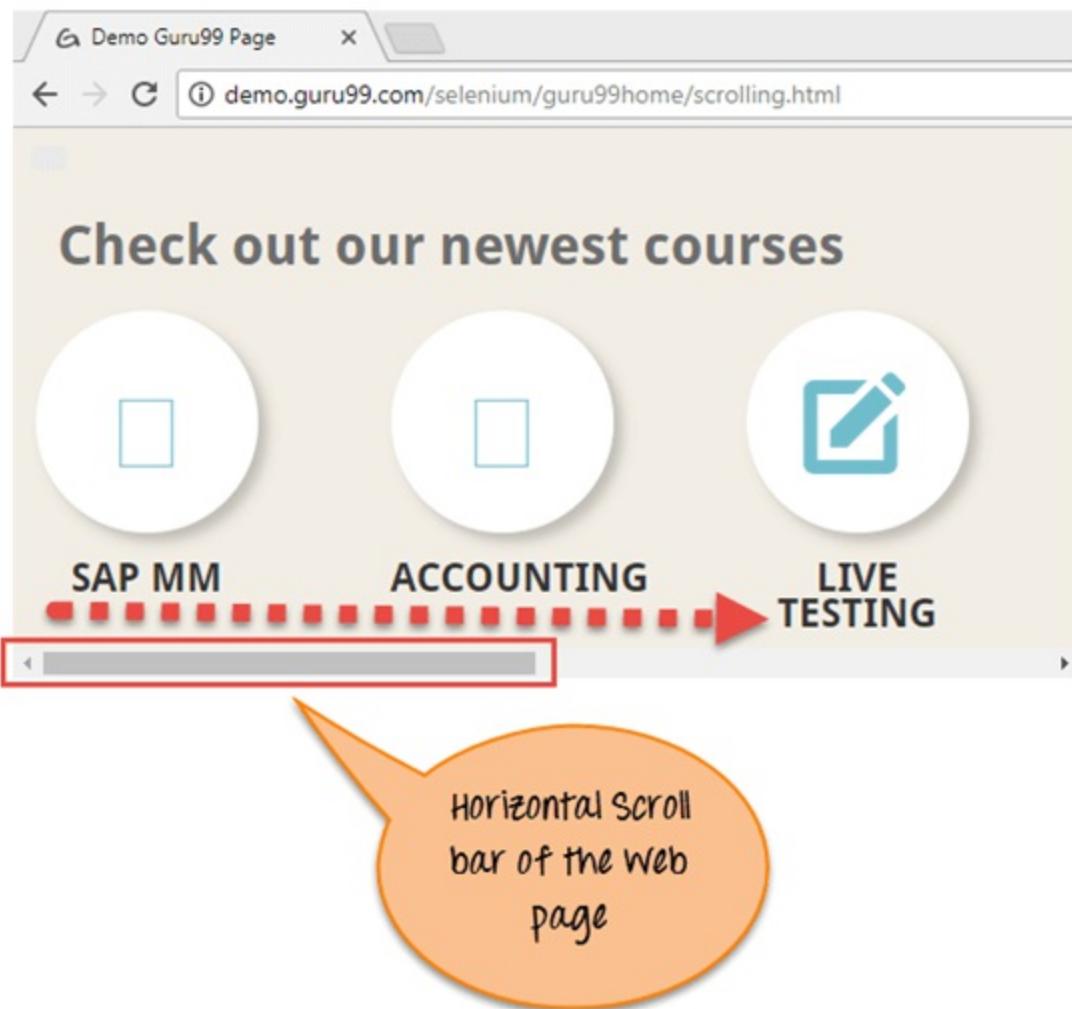
Chapter 65: Scroll UP or Down a page in Selenium Webdriver

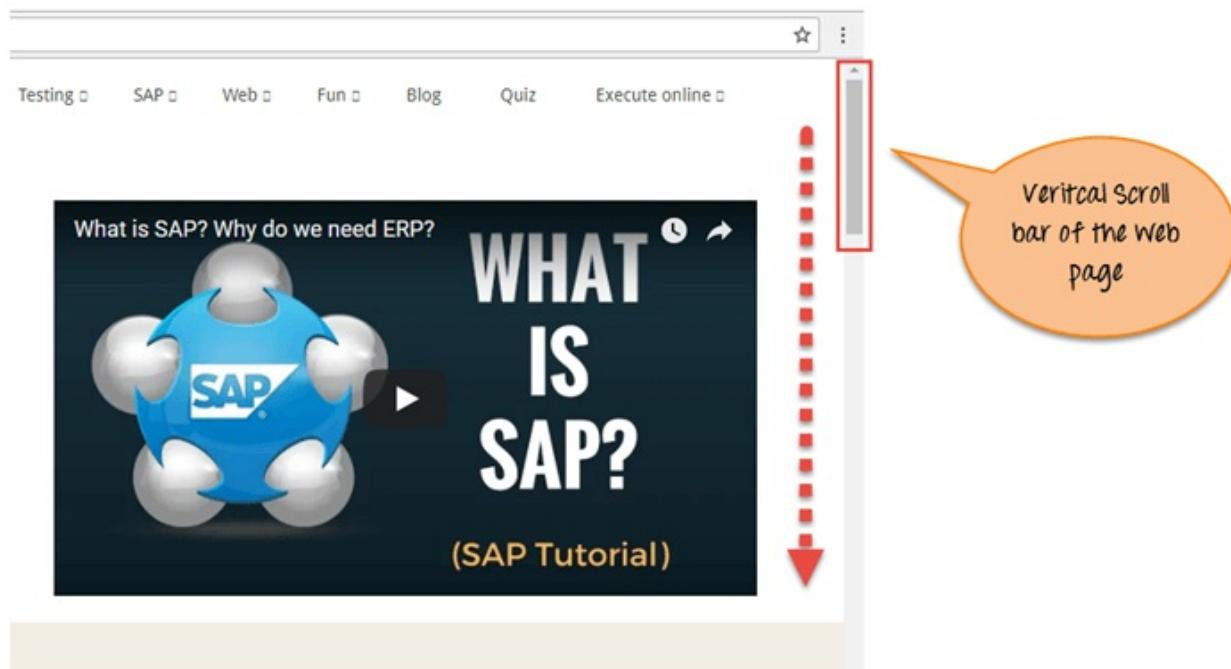
What is a Scrollbar?

A Scrollbar is a lets you move around screen in horizontal or vertical direction if the current page scroll does not fit the visible area of the screen. It is used to move the window up and down.

Selenium Webdriver does not require scroll to perform actions as it manipulates DOM. But in certain web pages, elements only become visible once the user have scrolled to them. In such cases scrolling may be necessary.

Scroll bar is of two type : **Horizontal** and **vertical** scroll bar as shown in below screenshot.





Scroll in Selenium

To scroll using Selenium, you can use `JavaScriptExecutor` interface that helps to execute JavaScript methods through Selenium Webdriver

Learn more about `JavaScriptExecutor`

Syntax :

```
JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript(Script, Arguments);
```

- Script – This is the JavaScript that needs to execute.
- Arguments – It is the arguments to the script. It's optional.

Selenium Script to scroll down the page

Let's, see the scroll down a web page using the selenium webdriver with following 3 scenarios :

- Scenario 1: To scroll down the web page by pixel.
- Scenario 2: To scroll down the web page by the visibility of the element.
- Scenario 3: To scroll down the web page at the bottom of the page.
- Scenario 4: Horizontal scroll on the web page.

Scenario 1: To scroll down the web page by pixel.

Selenium Script

```
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.Test;

public class ScrollByPixel {

    WebDriver driver;
    @Test
    public void ByPixel() {
        System.setProperty("webdriver.chrome.driver",
"E://Selenium//Selenium_Jars//chromedriver.exe");
        driver = new ChromeDriver();

        JavascriptExecutor js = (JavascriptExecutor) driver;

        // Launch the application
        driver.get("http://demo.guru99.com/test/guru99home/");

        //To maximize the window. This code may not work with
        Selenium 3 jars. If script fails you can remove the line below
        driver.manage().window().maximize();

        // This will scroll down the page by 1000 pixel
        vertical
            js.executeScript("window.scrollBy(0,1000)");
    }
}
```

Script Description: In the above code first we launch the given URL in Chrome browser. Next, scroll the page by 1000 pixels through executeScript. Javascript method ScrollBy() scrolls the web page to the specific number of pixels.

The syntax of ScrollBy() methods is :

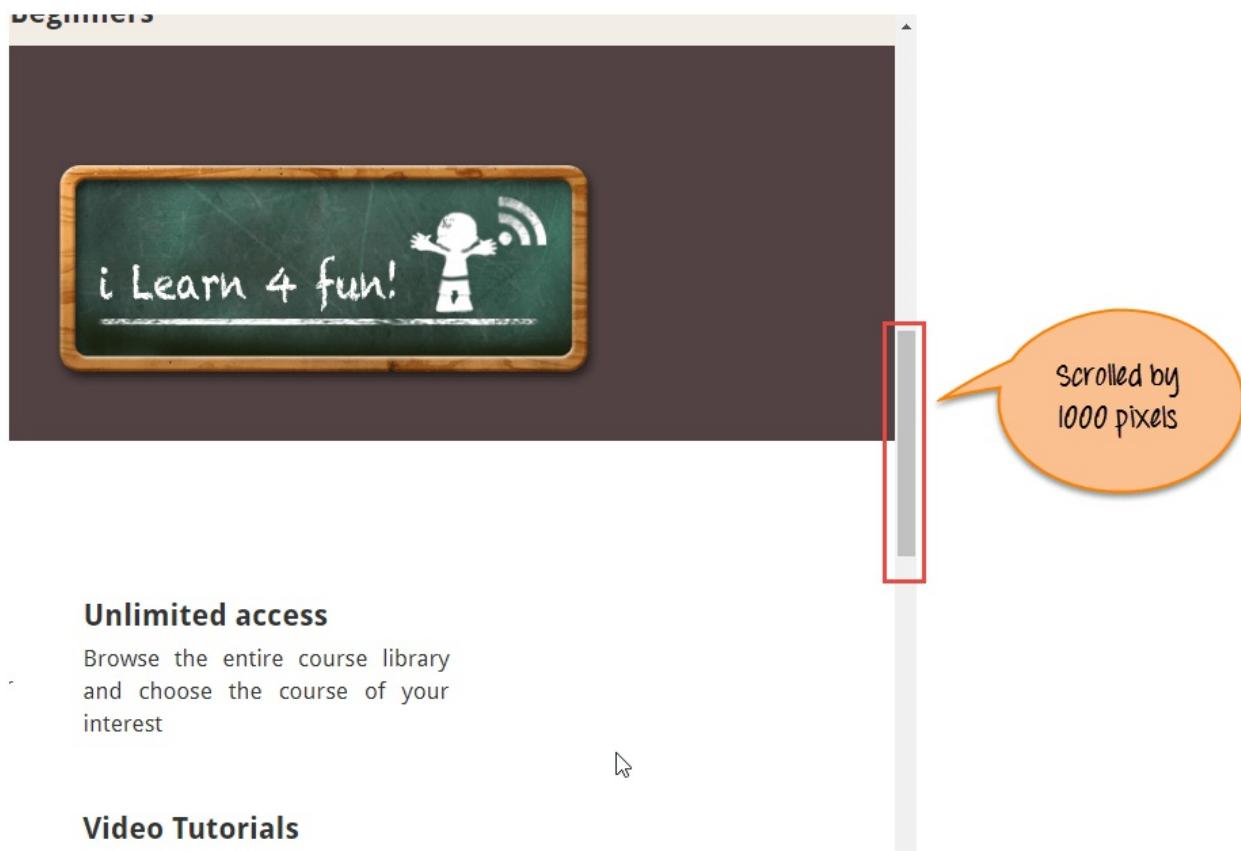
```
executeScript("window.scrollBy(x-pixels,y-pixels)");
```

x-pixels is the number at x-axis, it moves to the left if number is positive and it move to the right if number is negative .y-pixels is the number at y-axis, it moves to the down if number is positive and it move to the up if number is in negative .

Example:

```
js.executeScript("window.scrollBy(0,1000)"); //Scroll vertically  
down by 1000 pixels
```

Output analysis : Here is the output when you execute the above script .



Scenario 2: To scroll down the web page by the visibility of the element.

Selenium Script

```
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.Test;

public class ScrollByVisibleElement {

    WebDriver driver;
    @Test
    public void ByVisibleElement() {
        System.setProperty("webdriver.chrome.driver",
"G://chromedriver.exe");
```

```
driver = new ChromeDriver();
JavascriptExecutor js = (JavascriptExecutor) driver;

//Launch the application
driver.get("http://demo.guru99.com/test/guru99home/");

//Find element by link text and store in variable
Element
WebElement Element =
driver.findElement(By.linkText("Linux"));

//This will scroll the page till the element is found
js.executeScript("arguments[0].scrollIntoView()", 
Element);
}
}
```

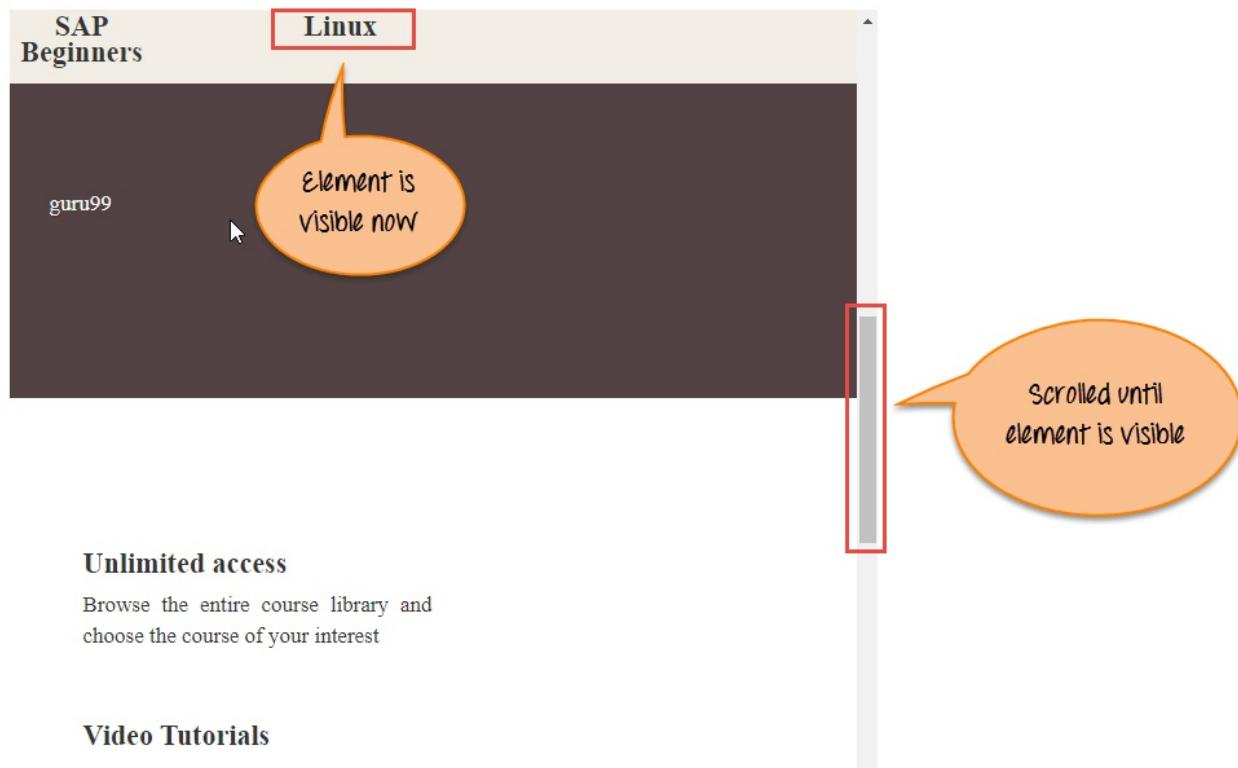
Script Description: In the above code, we first launch the given url in Chrome browser. Next, scroll the page until the mentioned element is visible on the current page. Javascript method scrollIntoView() scrolls the page until the mentioned element is in full view :

```
js.executeScript("arguments[0].scrollIntoView()",Element );
```

"arguments[0]" means first index of page starting at 0.

Where an " Element " is the locator on the web page.

Output analysis : Here is the output when you execute the above script .



Scenario 3: To scroll down the web page at the bottom of the page.

Selenium Script

```
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.Test;

public class ScrollByPage {

    WebDriver driver;
    @Test
    public void ByPage() {
        System.setProperty("webdriver.chrome.driver",
"E://Selenium//Selenium_Jars//chromedriver.exe");
        driver = new ChromeDriver();

        JavascriptExecutor js = (JavascriptExecutor) driver;
```

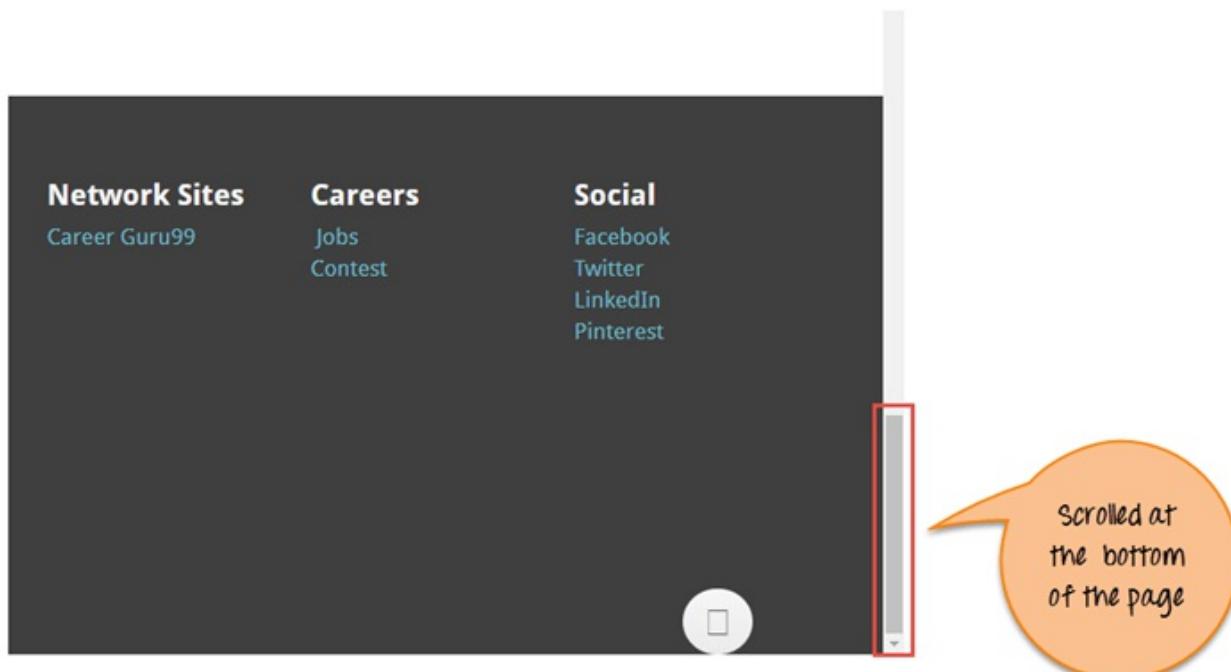
```
// Launch the application  
driver.get("http://demo.guru99.com/test/guru99home/");  
  
//This will scroll the web page till end.  
js.executeScript("window.scrollTo(0,  
document.body.scrollHeight)");  
}  
}
```

Script Description : In the above code, we first launch the given url in Chrome browser. Next, scroll till the bottom of the page. Javascript method scrollTo() scroll the till the end of the page .

```
js.executeScript("window.scrollTo(0,  
document.body.scrollHeight)");
```

"document.body.scrollHeight" returns the complete height of the body i.e web page.

Output analysis: Here is the output when you execute the above script.



Scenario 4: Horizontal scroll on the web page.

Selenium Script

```
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.Test;

public class HorizontalScroll {

    WebDriver driver;
    @Test
    public void ScrollHorizontally() {
        System.setProperty("webdriver.chrome.driver",
"E://Selenium//Selenium_Jars//chromedriver.exe");
        driver = new ChromeDriver();

        JavascriptExecutor js = (JavascriptExecutor) driver;
        // Launch the application

        driver.get("http://demo.guru99.com/test/guru99home/scrolling.htm
l");
        WebElement Element =
driver.findElement(By.linkText("VBScript"));

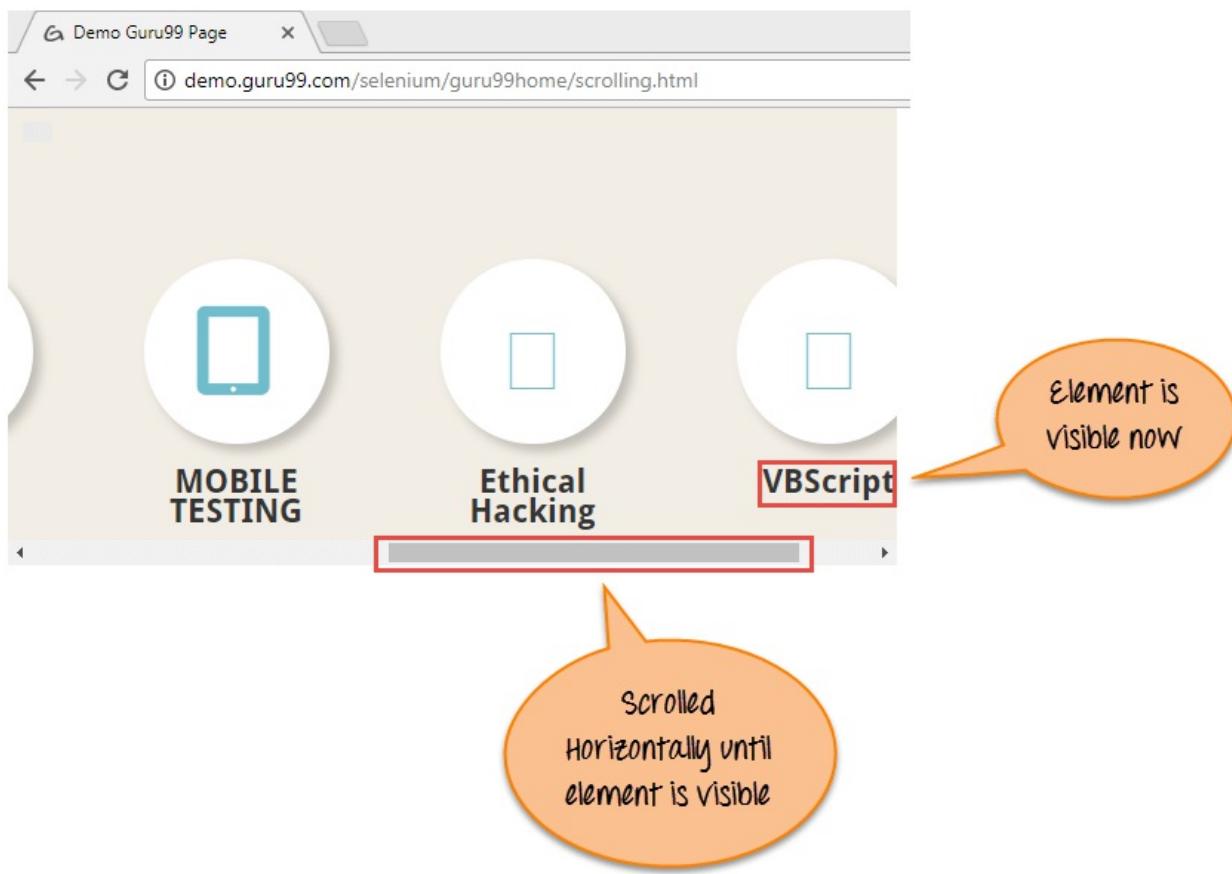
        //This will scroll the page Horizontally till the
element is found
        js.executeScript("arguments[0].scrollIntoView();",
Element);
    }
}
```

Script Description : In the above code, we first launch the given url in Chrome browser. Next, scroll the page horizontally until the mentioned element is visible on the current page. Javascript method

scrollIntoView() scrolls the page until the mentioned element is in full view :

```
js.executeScript("arguments[0].scrollIntoView();",Element );
```

Output analysis: Here is the output when you execute the above script.



Summary

- In the above tutorial, we illustrate the scroll of the web page through different scenarios.
- In the first scenario, we showed the scroll down on page by pixel.
- In the second scenario, we showed the scroll down of page until

the visible of the element.

- In the third scenario, we showed the scroll down of page at the bottom of the page.
- In the fourth scenario, illustrated the horizontal scroll on the web page.

Chapter 66: File Upload using Sikuli in Selenium Webdriver

Introduction to Sikuli

Sikuli is an open source GUI based automation tool. It is used to interact with elements of a web page and handling windows based popups. It uses the technique of 'Image Recognition' to interact with elements of the web page and windows popups. Sikuli considers all the elements of a web page as images and recognizes the elements based on their images. Sikuli is preferred when UI elements are stable and are not constantly changing.

Sikuli Integration with Selenium Webdriver

Sikuli can be integrated with selenium webdriver using the Sikuli JAR file.

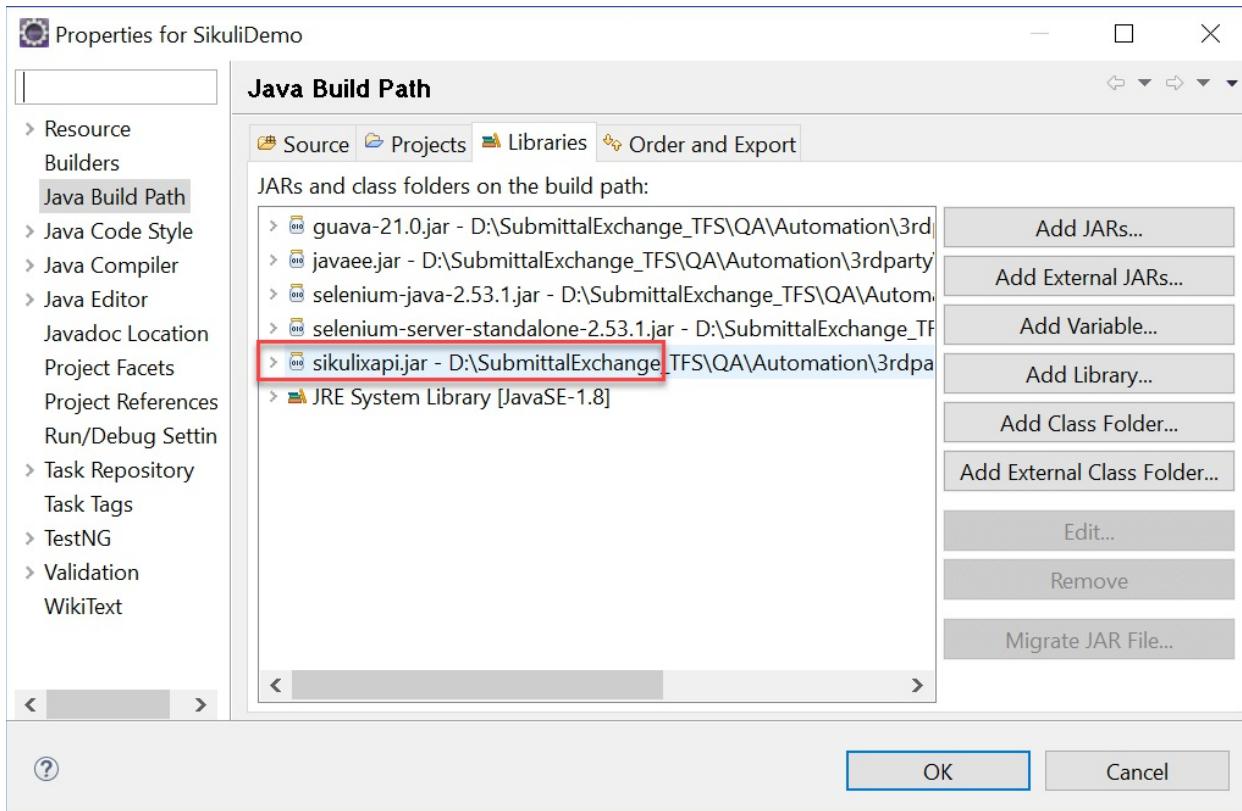
The below sequence is the list of steps to configure Sikuli with selenium webdriver.

Step 1) Download the Sikuli JAR file from the below URL.

<https://mvnrepository.com/artifact/com.sikulix/sikulixapi/1.1.0>

Extract the contents of the ZIP file to a folder.

Step 2) Create a new JAVA project in Eclipse and add the JAR file to build path, along with selenium jar files using Right Click on the project -> Build Path -> Configure Build Path



Once you have added the JAR file to project build path, classes provided by Sikuli can be used.

Screen class in Sikuli

Screen class is the base classes for all the methods provided by Sikuli. Screen class contains predefined methods for all the commonly performed operations on screen elements such as click, double-click, providing input to a text box, hover, etc. The below is the list of commonly used methods provided by Screen class.

Method	Description	Syntax
--------	-------------	--------

Click	This method is used to click on an element on the screen using image name as the parameter.	Screen s = new Screen(); s.click("QA.png");
doubleClick	This method is used to double click on an element. It accepts image name as the parameter.	Screen s = new Screen(); s.doubleClick("QA.png");
Type	This method is used to provide input value to an element. It accepts the image name and text to be sent as parameters.	s.type("QA.png","TEXT");
Hover	This method is used to hover over an element. It accepts image name as the parameter.	s.hover("QA.png");
Find	This method is used to find a specific element on the screen. It accepts image name as the parameter.	s.find("QA.png");

Pattern class in Sikuli

Pattern class is used to associate the image file with additional attributes to uniquely identify the element. It takes the path of the image as a parameter.

```
Pattern p = new Pattern("Path of image");
```

The following are the most commonly used methods of Pattern class.

Method	Description	Syntax
getFileName	Returns the file name contained in the Pattern object.	Pattern p = new Pattern("D:\Demo\QA.png"); String filename = p.getFileName();

similar	This method returns a new Pattern object with similarity set to a specified value. It accepts the similarity value between 0 to 1 as a parameter. Sikuli looks for all elements that fall within the specified similarity range and returns a new pattern object.	Pattern p1 = p.similar(0.7f);
Exact	This method returns a new pattern object with similarity set to 1. It looks only for an exact match of the specified element.	Pattern p1 = p.exact();

Code Example for File Upload using Sikuli

Below code explains the use of Sikuli for file upload in Firefox.

```
package com.sikuli.demo;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.sikuli.script.FindFailed;
import org.sikuli.script.Pattern;
import org.sikuli.script.Screen;
import org.openqa.selenium.chrome.ChromeDriver;

public class SikuliDemo {

    public static void main(String[] args) throws FindFailed {

        System.setProperty("webdriver.chrome.driver",
"D:\\chromedriver.exe");
        String filepath = "D:\\Guru99Demo\\Files\\";
        String inputFilePath = "D:\\Guru99Demo\\Files\\";
        Screen s = new Screen();
        Pattern fileInputTextBox = new Pattern(filepath +
"FileTextBox.PNG");
        Pattern openButton = new Pattern(filepath +
"OpenButton.PNG");
        WebDriver driver;
```

```
// Open Chrome browser
driver = new ChromeDriver();

driver.get("http://demo.guru99.com/test/image_upload/index.php")
;

        // Click on Browse button and handle windows pop up
using Sikuli
        driver.findElement(By.xpath(".//*
[@id='photoimg']")).click();
        s.wait(fileInputTextBox, 20);
        s.type(fileInputTextBox, filePath + "Test.docx");
        s.click(openButton);

        // Close the browser
        driver.close();

    }

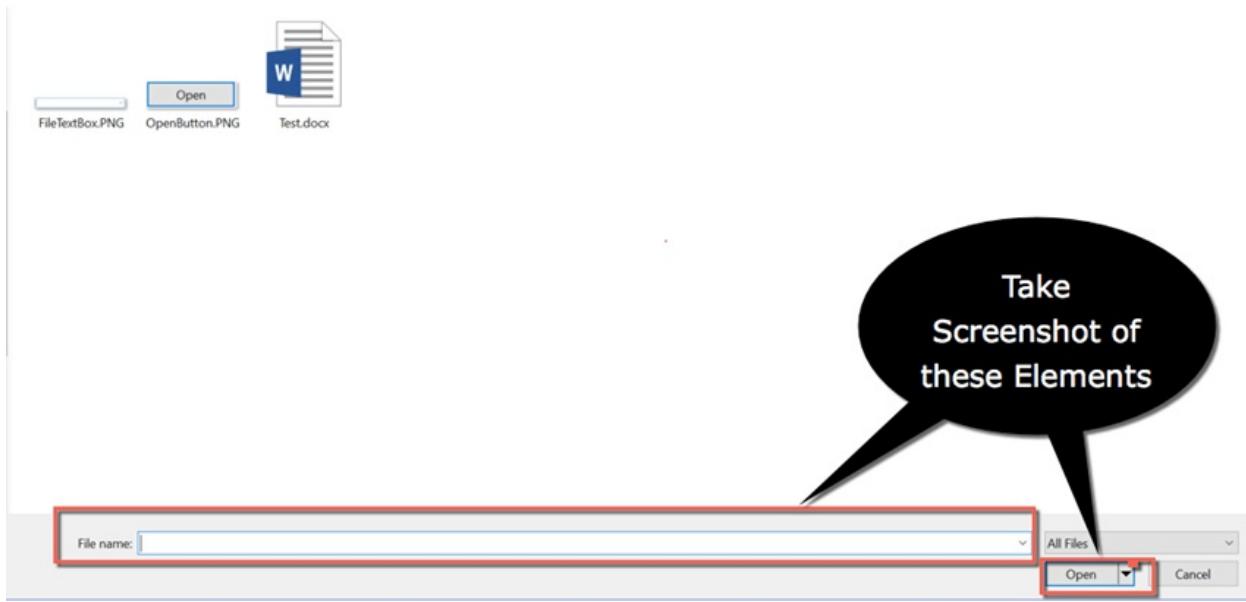
}
```

Code Explanation:

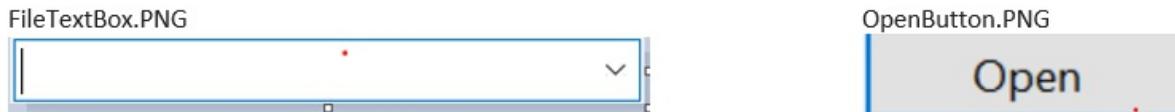
Step 1) The first statement involves setting the driver executable path for chrome.

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

Step 2) Use a screengrab tool such as Snipping Tool to take screenshots of windows popup 'FileTextBox' and 'Open' button.



This is how your screenshot should look like:-



Images for windows file input text box and open button are stored onto 'FileTextBox.PNG' and 'OpenButton.PNG'.

Sikuli uses the technique of Image Recognition to recognize elements on the screen. It finds elements on screen solely based on their images.

Example: If you want to automate the operation of opening notepad, then you need to store the image of a desktop icon for notepad onto a PNG file and perform click operation on it.

In our case, it recognizes the file input text box and opens button on Windows popup using the images stored. **If the screen resolution changes from image capture to test script execution, the behavior of Sikuli would be inconsistent. Hence it is always advisable to run the test script on the same resolution at**

which images are captured. Change in pixel size of images will result in Sikuli throwing a FindFailed exception.

Step 3) The next statements include the creation of objects for Screen and Pattern classes. Create a new screen object. Set the path of the file you want to upload as a parameter to the Pattern object.

```
Screen s = new Screen();
Pattern fileInputTextBox = new Pattern(filepath +
"FileTextBox.PNG");
Pattern openButton = new Pattern(filepath + "OpenButton.PNG");
```

Step 4) The below statements involve opening chrome browser with the URL: http://demo.guru99.com/test/image_upload/index.php

```
driver = new ChromeDriver();
driver.get("http://demo.guru99.com/test/image_upload/index.php")
;
```

The above URL is a demo application to demonstrate file upload functionality.

Step 5) Click the choose file button using below statement

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

Step 6) Wait for the windows popup to appear. Wait method is used to handle the delay associated with opening windows pop up after clicking on the browse button.

```
s.wait(fileInputTextBox, 20);
```

Step 7) Type the file path onto input file text box and click on Open button

```
s.type(fileInputTextBox, filePath + "Test.docx");
```

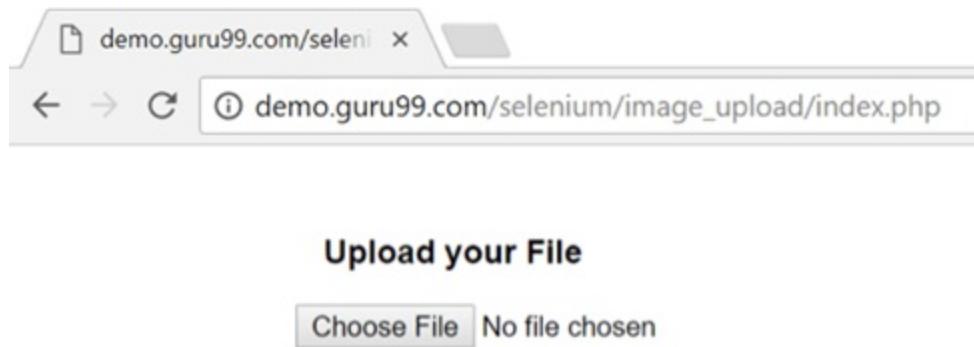
```
s.click(openButton);
```

Step 8) Close the browser

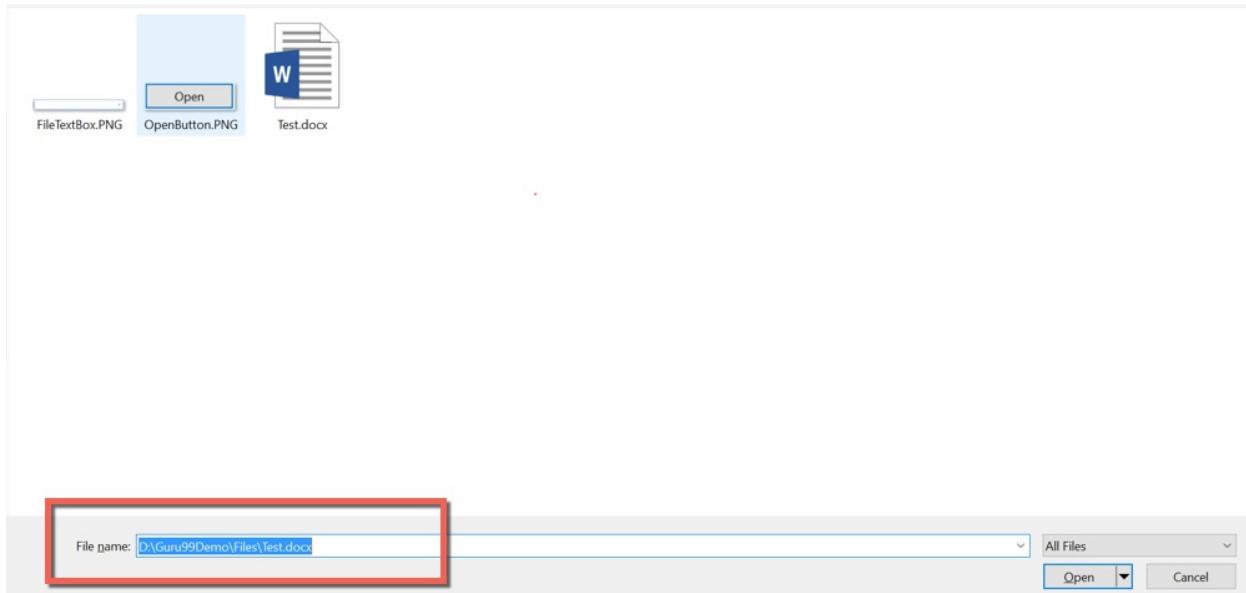
```
driver.close();
```

Output:

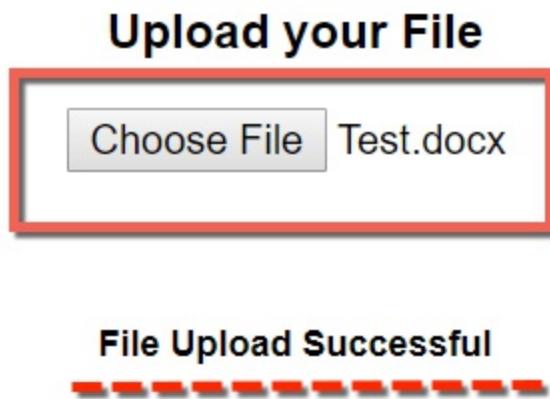
Initially, script opens chrome browser



Clicks on the 'Choose File' button, windows file popup screen will appear. Enters data into File Input textbox and clicks on 'Open' button



Below screen is displayed once the file upload is complete and closes the browser



Conclusion:

Sikuli is used to handle flash objects on a web page and windows popups with ease. Sikuli is best used when the elements on user interface do not change frequently. Owing to this disadvantage, from an automation testing perspective, Sikuli is given less preference compared to other frameworks such as Robot and AutoIT.

Chapter 67: Gecko (Marionette) Driver

Selenium: Download, Install, Use with Firefox

What is Gecko Driver?

The term Gecko stands for a Web Browser engine that is inbuilt within Mozilla Firefox browser. Gecko driver acts as a proxy between Web Driver enabled clients(Eclipse, Netbeans, etc.) and Mozilla Firefox browser. In short, Gecko driver acts as a link between Selenium Web Driver tests and Mozilla Firefox browser.

Before Selenium 3, Mozilla Firefox browser was the default browser for Selenium. After Selenium 3, testers need to initialize the script to use Firefox using GeckoDriver explicitly. Selenium uses W3C Webdriver protocol to send requests to GeckoDriver, which translates them into a protocol named Marionette. Firefox will understand the commands transmitted in the form of Marionette protocol and executes them.



Advantage of using Gecko Driver

Selenium Webdriver version 2.53 is not compatible with Mozilla Firefox version 47.0+. The Firefox driver used in earlier versions of Mozilla Firefox will be discontinued, and only the GeckoDriver implementation would be used. Hence testers are forced to use GeckoDriver if they want to run automated tests on Mozilla Firefox version 47.0+. But the big question - what is the advantage?

The major advantage of using GeckoDriver as opposed to the default Firefox driver is **Compatibility**. GeckoDriver uses **W3C WebDriver protocol** to communicate with Selenium. W3C is a universally defined standard for Web Driver. This means Selenium Developers (People who code Selenium base) need not create a new version of Web Driver for each browser version. The same Web Driver can be used for multiple browser versions. Hence, GeckoDriver is preferred compared to the earlier implementation of Firefox driver.

Download and Install Gecko Driver:

Gecko Driver is available as an executable file that can be downloaded on the system. The following are the list of steps to download gecko driver.

Step 1) At this page

<https://github.com/mozilla/geckodriver/releases> ,Select the appropriate version for GeckoDriver download based on your operating system

v0.19.1

 AutomatedTester released this on Nov 1, 2017 · 11 commits to master since this release

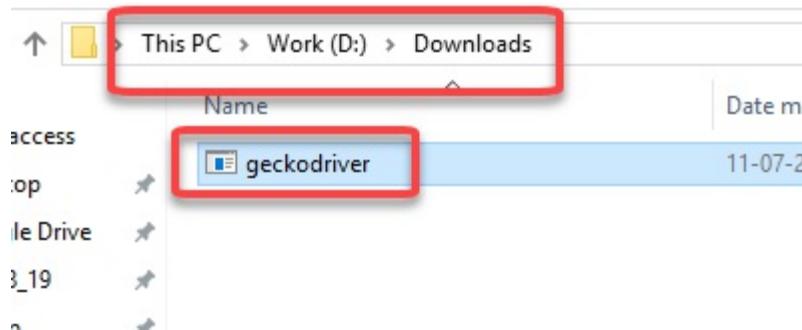
Assets

 geckodriver-v0.19.1-arm7hf.tar.gz	2.19 MB
 geckodriver-v0.19.1-linux32.tar.gz	2.23 MB
 geckodriver-v0.19.1-linux64.tar.gz	2.19 MB
 geckodriver-v0.19.1-macos.tar.gz	1.26 MB
 geckodriver-v0.19.1-win32.zip	1.98 MB
 geckodriver-v0.19.1-win64.zip	2.03 MB
 Source code (zip)	
 Source code (tar.gz)	

Step 2) Once the ZIP file download is complete, extract the contents of ZIP File onto a file folder



Step 3) Note the location where you extracted the driver. Location will be used later to instantiate the driver.



Ways to initialize GeckoDriver:

There are three different ways to initialize GeckoDriver.

1. Using DesiredCapabilities:

First, set the system property for Gecko Driver.

Syntax:

```
System.setProperty("webdriver.gecko.driver", "Path to  
geckdriver.exe file");
```

Example:

```
System.setProperty("webdriver.gecko.driver", "D:\\Downloads\\Geck  
oDriver.exe");
```

Next, set Desired Capabilities.

Desired Capabilities help Selenium to understand the browser name, version and operating system to execute the automated tests. Below is the code to set gecko driver using DesiredCapabilities class.

```
DesiredCapabilities capabilities =  
DesiredCapabilities.firefox();  
capabilities.setCapability("marionette", true);
```

Here is the complete code

```
System.setProperty("webdriver.gecko.driver", driverPath);  
DesiredCapabilities capabilities =  
DesiredCapabilities.firefox();  
capabilities.setCapability("marionette", true);  
driver= new FirefoxDriver(capabilities);
```

2. Using marionette property:

Gecko driver can also be initialized using marionette property as below

```
System.setProperty("webdriver.firefox.marionette", "D:\\Downloads\\GeckoDriver.exe");
```

If gecko driver is initialized using the above method, code for desired capabilities is **not** required.

3. Using FirefoxOptions:

Mozilla Firefox version 47+ has marionette driver as a legacy system. Taking advantage of this, marionette driver can be called using Firefox Options as below

```
FirefoxOptions options = new FirefoxOptions();
options.setLegacy(true);
```

Code for launching firefox using Gecko driver :

```
package com.guru99.demo;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.remote.DesiredCapabilities;

public class GeckoDriverDemo {

    String driverPath = "D:\\Guru99Demo\\GeckoDriver.exe";
    public WebDriver driver;

    @Before
    public void startBrowser() {
        System.setProperty("webdriver.gecko.driver",
driverPath);
        DesiredCapabilities capabilities =
```

```
DesiredCapabilities.firefox();
capabilities.setCapability("marionette", true);
driver = new FirefoxDriver(capabilities);

}

@Test
public void navigateToUrl() {

driver.get("http://demo.guru99.com/selenium/guru99home/");
}

@After
public void endTest() {
    driver.quit();
}

}
```

Code Explanation:

@Before method:

Initially, we need to set the system property for gecko driver to the geckdriver.exe file download location. We need to set the marionette property to true for Selenium to use Marionette protocol to communicate with Gecko Driver. Finally, we need to start the Firefox browser instance using the object for Desired Capabilities.

The below statements help to achieve the above task.

```
System.setProperty("webdriver.gecko.driver", driverPath);
DesiredCapabilities capabilities =
DesiredCapabilities.firefox();
capabilities.setCapability("marionette",true);
driver= new FirefoxDriver(capabilities);
```

@Test method:

We are navigating to user-specified URL using the inbuilt "get" method provided by Selenium web driver. The below statement help to achieve the same.

```
driver.get("http://demo.guru99.com/selenium/guru99home/");
```

@After method:

Finally, we are closing the browser instance using the quit method.

```
driver.quit();
```

Modify a script for non- Gecko to Gecko:

Non-gecko driver script used before Selenium 3 was straightforward. We need to create an instance of Firefox driver and use the instance variable.

```
@Before  
public void startBrowser() {  
    driver = new FirefoxDriver();  
}
```

To convert to gecko, you need to simply add one line of code

```
@Before  
public void startBrowser() {  
    System.setProperty("webdriver.firefox.marionette",  
    "D:\\Downloads\\GeckoDriver.exe");  
    driver = new FirefoxDriver();  
}
```

Common exceptions occurred while using Gecko Driver:

Following is a list of common exceptions that occur while using Gecko Driver and with resolution.

1. The path to driver executable must be set by webdriver.gecko.driver system property:

This exception occurs when user tries to instantiate Firefox driver without setting the system property for gecko driver. This is usually done by beginners to Selenium who are not aware of the changes made from Selenium 3 to Selenium previous versions.

The resolution for the above exception is to set the system property for gecko driver with the location of geckodriver.exe file as below

```
System.setProperty("webdriver.gecko.driver",  
"D:\\Downloads\\\\geckodriver.exe");
```

Please note that you need to set the property of gecko driver before creating an instance of Mozilla Firefox driver.

2. Firefox Not Connected Exception:

```
org.openqa.selenium.firefox.NotConnectedException: Unable to  
connect to host 127.0.0.1 on port 7055 after 45000 ms.
```

This exception usually occurs when Firefox version has been upgraded to the latest version. The resolution for this exception is to update the selenium jar file and gecko driver to the latest version and use the same.

3. Session Not Created Exception:

```
org.openqa.selenium.SessionNotCreatedException: Unable to create new remote session.
```

This exception occurs due to compatibility issues between Selenium and Gecko driver. Gecko driver works with Firefox version 47 or above. It can be resolved by updating Firefox version to 47 or above.

4. Connection Refused Exception:

```
WebDriver Exception: Connection Refused
```

This exception is the message generated when web driver is unable to establish a connection with Firefox. It can be resolved using any one of the following techniques.

- Use driver.quit() method to destroy earlier instances of web driver
- Clean the browser cache before executing your automated tests
- Clean the project workspace within Eclipse IDE
- Always use the latest version of selenium gecko driver and most recent version of Firefox browser

Chapter 68: Find Element and Find Elements in Selenium

Why do you need Find Element/s command?

Interaction with a web page requires a user to locate the web element. Find Element command is used to uniquely identify a (one) web element within the web page. Whereas, Find Elements command is used to uniquely identify the list of web elements within the web page. There are multiple ways to uniquely identify a web element within the web page such as ID, Name, Class Name, Link Text, Partial Link Text, Tag Name and XPATH.

FindElement command syntax:

Find Element command takes in the By object as the parameter and returns an object of type WebElement. By object in turn can be used with various locator strategies such as ID, Name, Class Name, XPATH etc. Below is the syntax of FindElement command in Selenium web driver.

```
WebElement elementName =  
driver.findElement(By.LocatorStrategy("LocatorValue"));
```

Locator Strategy can by any of the following values.

- ID

- Name
- Class Name
- Tag Name
- Link Text
- Partial Link Text
- XPATH

Locator Value is the unique value using which a web element can be identified. It is the responsibility of developers and testers to make sure that web elements are uniquely identifiable using certain properties such as ID or name.

Example:

```
WebElement loginLink = driver.findElement(By.linkText("Login"));
```

FindElements command syntax:

Find Elements command takes in By object as the parameter and returns a list of web elements. It returns an empty list if there are no elements found using the given locator strategy and locator value. Below is the syntax of find elements command.

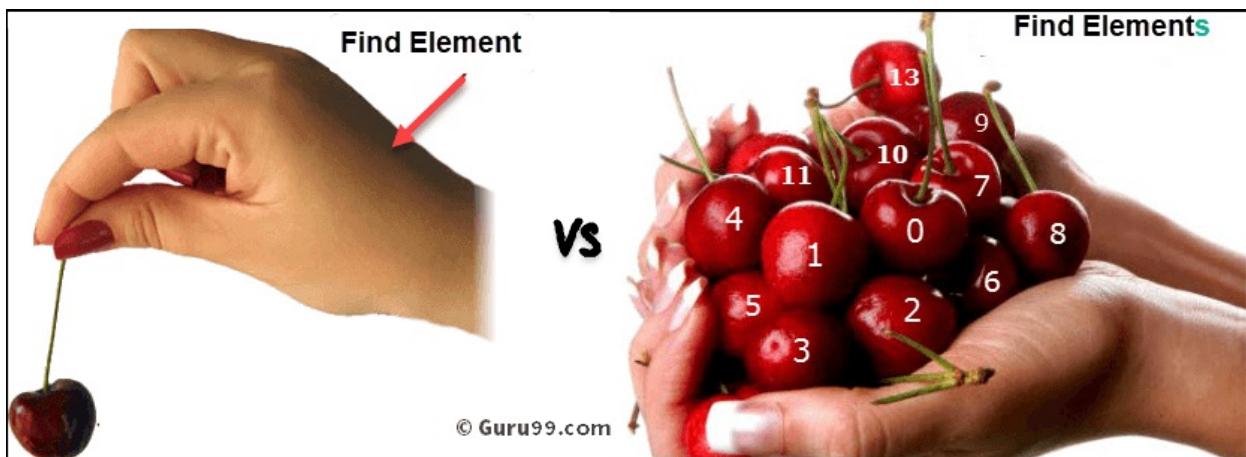
```
List<WebElement> elementName =  
driver.findElements(By.LocatorStrategy("LocatorValue"));
```

Example:

```
List<WebElement> listOfElements =  
driver.findElements(By.xpath("//div"));
```

Find element Vs Find elements

Below are the major differences between find element and find elements commands.



Find Element	Find Elements
Returns the first most web element if there are multiple web elements found with the same locator	Returns a list of web elements
Throws exception NoSuchElementException if there are no elements matching the locator strategy	Returns an empty list if there are no web elements matching the locator strategy
It will only find one web element	It will find a collection of elements whose match the locator strategy.
Not Applicable	Each Web element is indexed with a number starting from 0 just like an array

Example

Find Element

The following application is used for demo purpose

<http://demo.guru99.com/test/ajax.html>

Scenario:

1. Open the AUT

2. Find and click radio button

```
package com.sample.stepdefinitions;

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

public class NameDemo {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        System.setProperty("webdriver.chrome.driver",
"D:\\\\3rdparty\\\\chrome\\\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        driver.manage().window().maximize();

        driver.get("http://demo.guru99.com/test/ajax.html");

        // Find the radio button for "No" using its ID and click
on it
        System.out.println(By.Name("name"));

    }

}
```

Find Elements command:

Scenario:

1. Open the URL for Application Under Test
2. Find the text of radio buttons and print it onto the output console

```
package com.sample.stepdefinitions;

import java.util.List;

import org.openqa.selenium.By;
```

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

public class NameDemo {

    public static void main(String[] args) {

        System.setProperty("webdriver.chrome.driver",
"X://chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        driver.get("http://demo.guru99.com/test/ajax.html");
        List <WebElement> elements = driver.findElements(By.name("name"));
        System.out.println("Number of elements:" +
elements.size());

        for (int i = 0; i < elements.size(); i++) {
            System.out.println("Radio button text:" +
elements.get(i).getAttribute("value"));
        }
    }
}
```

Summary:

- Find Element command returns the web element that matches the first most element within the web page.
- Find Elements command returns a list of web elements that match the criteria.
- Find Element command throws NoSuchElementException exception if it does not find the element matching the criteria.
- Find Elements command returns an empty list if there are no elements matching the criteria