Selenium- Reference Material

Submitted by: V & V Team

iteamic

#209, 28th Cross, 7th Block, Jayanagar
Bangalore, 560 082
Tel: 91 (80) 2663 4841
www.iteamic.com

Document Version No.:	1.0
Approved By:	
Date of Release:	

This document is created using IPAL V1.0 Selenium-Reference Material V 1.0

Table of Contents

A. Document Change History

Note: This section is to be maintained by the Project team

Ver.	Author	Change Description	Reviewed By	Approved By	Approval Date
1.0	Tarun Kumar Bhadauria	Document Created	Ashwith Rai		03-July-2008

1. Introduction

Selenium is a test tool for web applications. Selenium tests run directly in a browser, just like real users do. Selenium can be used for unit-testing, regression testing, smoke-testing, integration and acceptance testing of web applications in a variety of browsers and platforms as following –

Supported Platforms:

- Windows:
  - Internet Explorer 6.0 and 7.0
  - Firefox 0.8 to 2.0
  - Mozilla Suite 1.6+, 1.7+
  - Seamonkey 1.0
  - Opera 8 & 9
- Mac OS X:
  - Safari 2.0.4+
  - Firefox 0.8 to 2.0
  - Camino 1.0a1
  - Mozilla Suite 1.6+, 1.7+
  - Seamonkey 1.0
- Linux:
  - Firefox 0.8 to 2.0
  - Mozilla Suite 1.6+, 1.7+
  - Konqueror
  - Opera 8 & 9

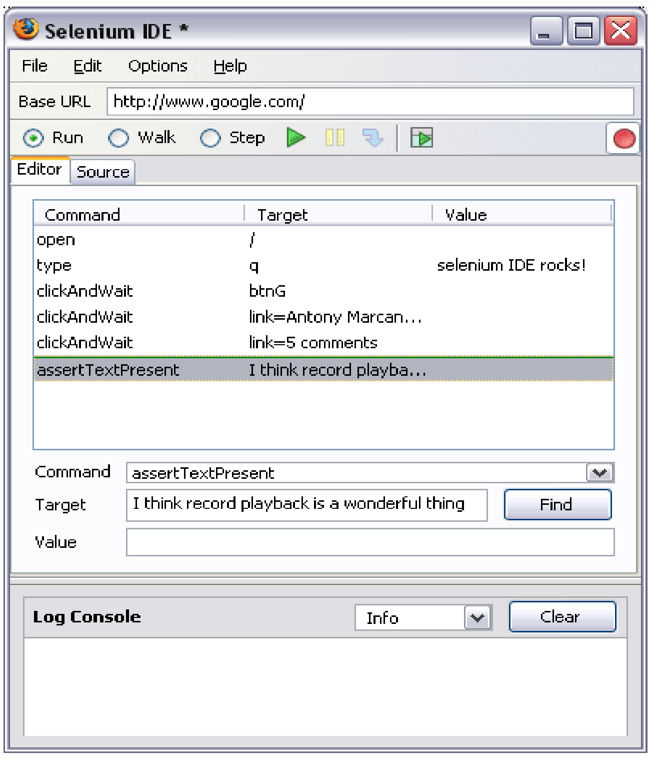
There are two modes of operation for Selenium - ***Core*** and ***Remote Control*** *(RC)*. Remote Control mode also has a related capability called *Selenium* ***Grid*** that allows throwing hardware at tests to make it all faster.

2. Selenium IDE

Selenium IDE is an integrated development environment for Selenium tests. It is implemented as a Firefox extension, and allows to record, edit, and debug tests. Selenium IDE includes the entire Selenium Core, allowing to easily and quickly record and play back tests in the actual environment that they will run.

Features:

- Easy record and playback,
- Intelligent field selection will use IDs, names, or Xpath as needed,
- Auto complete for all common Selenium commands,
- Debug and set breakpoints,
- Save tests as HTML, Ruby scripts, or any other format,
- Option to automatically assert the title of every page,

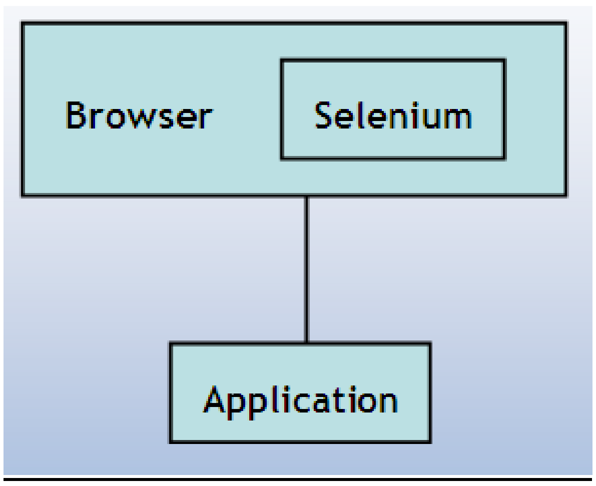


Base URL: http://www.google.com/

Command	Target	Value
open	/	
type	q	selenium IDE rocks!
clickAndWait	btnG	
clickAndWait	link=Antony Marcan...	
clickAndWait	link=5 comments	
assertTextPresent	I think record playba...	

Command: assertTextPresent
Target: I think record playback is a wonderful thing
Value:
Log Console: Info

Selenium IDE



Selenium IDE in Action

Selenium IDE would work with Browser which in tern interacts with application to simulate user actions.

Object Locators used in IDE

HTML-ID's

Id=LoginButton

xpath= xpathExpression

Locate an element using an XPath expression. XPath locators must

begin with "//".

xpath=//img[@alt='The image alt text']

xpath=//table[@id='table1']//tr[4]/td[2]

link= textPattern

Select the link (anchor) element which contains text matching the specified pattern.

link=The link text

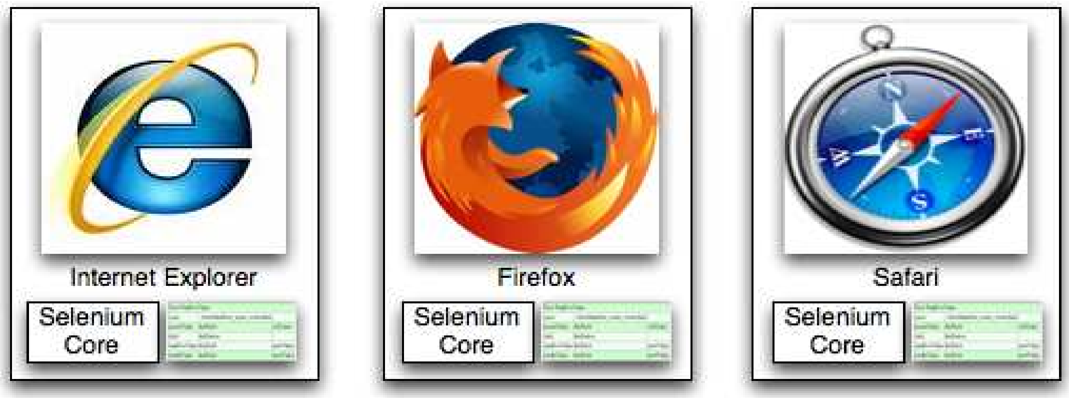
3. Selenium Core

Selenium Core is written in pure JavaScript/DHTML. Selenium Core uses JavaScript and Iframes to embed a test automation engine in your browser. This technique should work with any JavaScript-enabled browser.

Selenium Core tests directly into application web server, allowing the tests to run in any supported browser on the client-side. Thus, one must have write access to server to install Selenium Core.

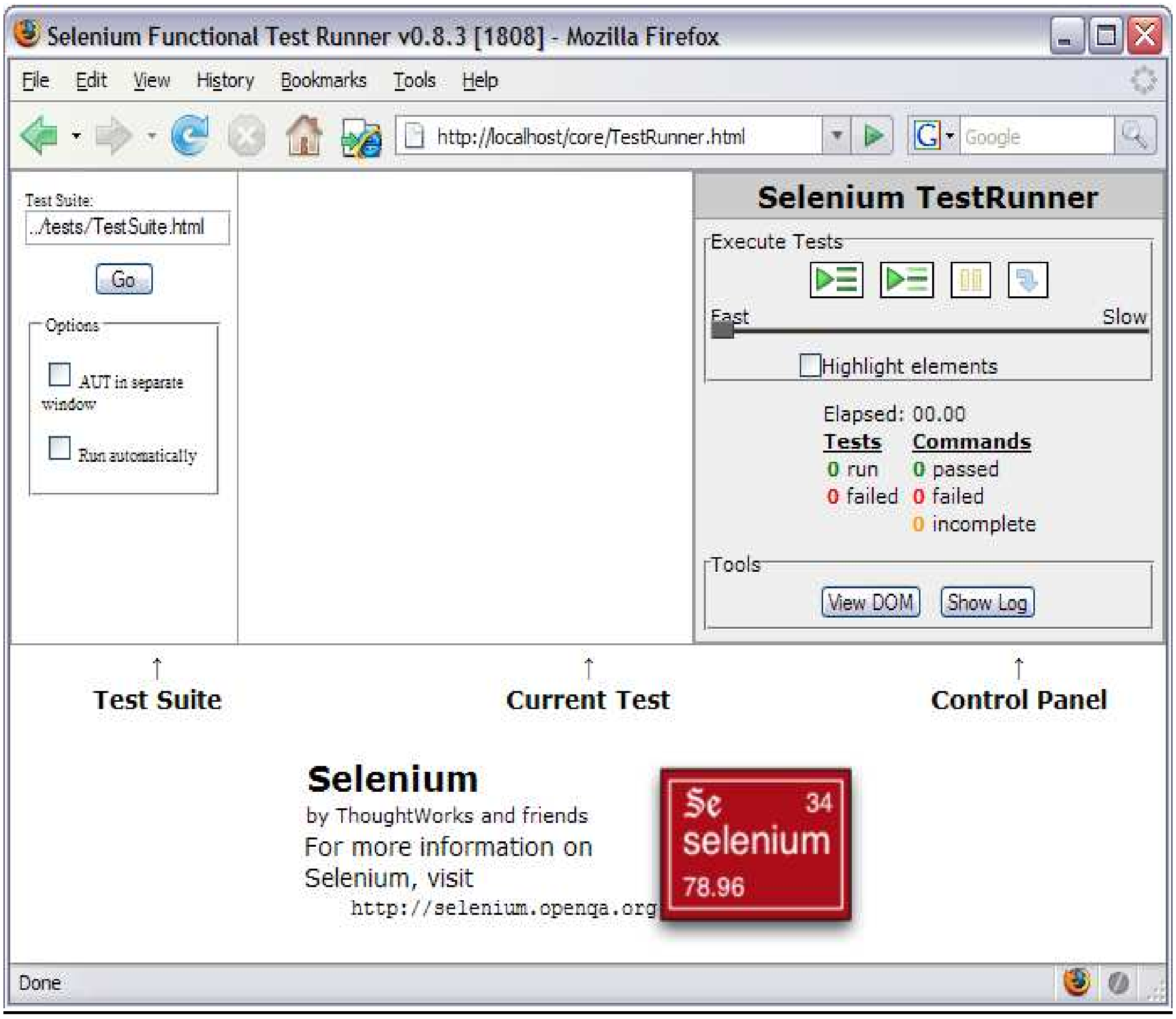
That means that one can't use Selenium Core (pure DHTML/JavaScript) to write a test of google.com this is because Selenium Core is pure DHTML/JavaScript, and so it is bound by JavaScript's security restrictions. This restriction is called same origin policy. The same origin policy states that JavaScript is allowed only to read/modify HTML from the *same origin* as its source.

Despite the soundness of the policy, it creates a problem for JavaScript automated tests. If one writes a .js file designed to test google.com, the same origin policy denies the right to run that .js file with google.com; instead, one has to somehow install that .js file *on google.com* in order to write automated tests against it.



Selenium Core

Selenium IDE embeds Selenium Core internally. Test Runner of Selenium can be driven from IDE itself.



<u>**Selenium Test Runner in Action**</u>

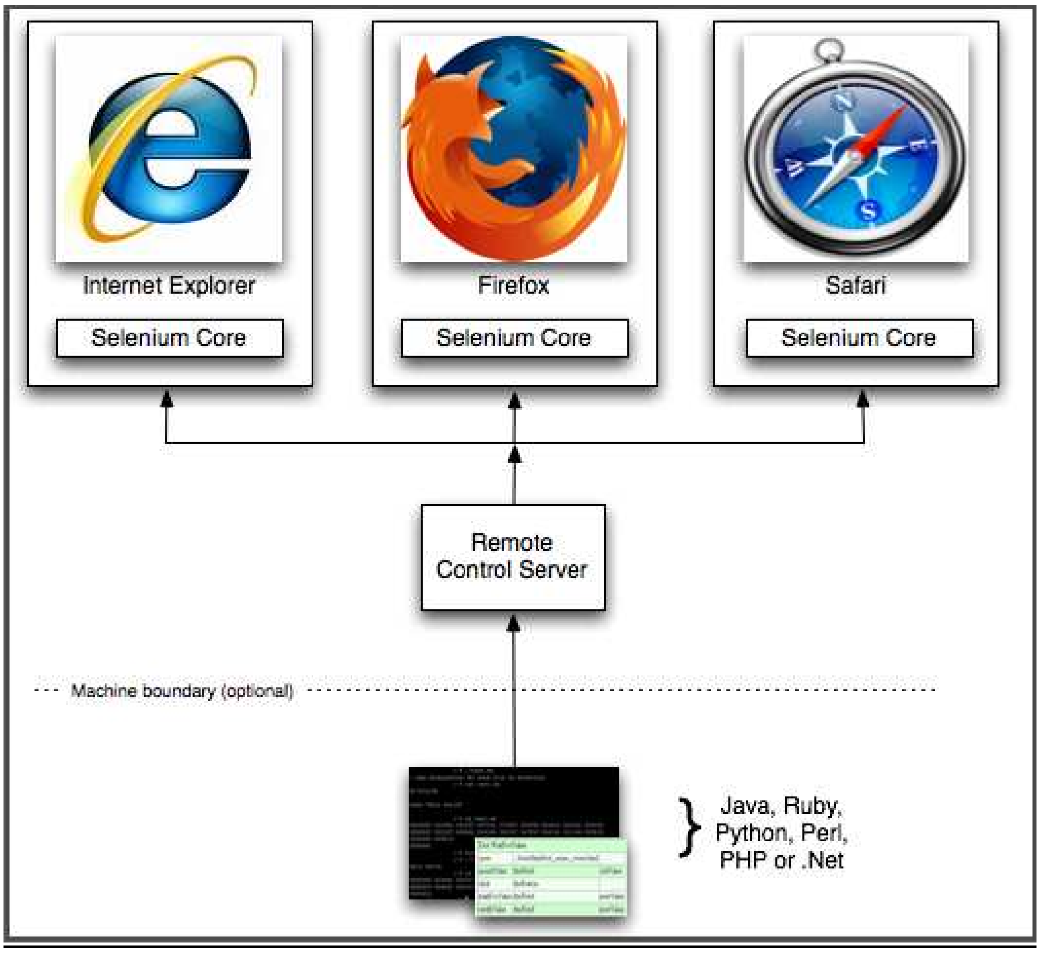
Test Runner is always used to run tests coded in HTML format. It is advisable to use this for our trial and error exercise during our initial test case development. We would have a fair idea on user action simulation of the recorded script.

4. Selenium Remote Control

Selenium Remote Control (RC) is a test tool that allows writing automated web application UI tests in any programming language against any HTTP website using any mainstream JavaScript-enabled browser.

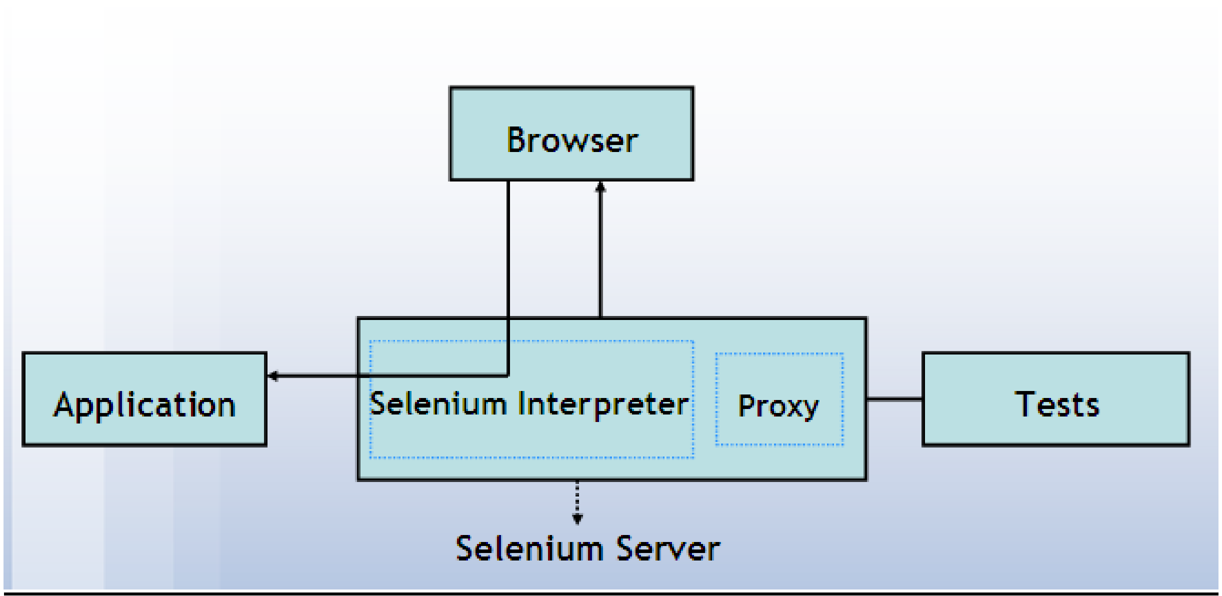
Selenium RC comes in two parts.

1. A server which can automatically launch and kill supported browsers, and acts as a HTTP proxy for web requests from those browsers. This Server bundles Selenium Core.
2. Client libraries for your favorite computer language. Using these libraries tests can be coded in following programming languages - Java, .NET, Perl, PHP, Python or Ruby.



Selenium RC

Selenium server acts as a Client Configured Proxy for the browser. Selenium Server doesn't simply fetch the page from the remote server, but instead automatically returns its own page. That makes the browser think that the remote server served up JS, which allows selenium to "inject" arbitrary JavaScript in to the domain being tested with out actually modifying the domain.



Selenium RC in Action

Selenium Server is written in Java, and requires the Java Runtime Environment (JRE) version 1.5.0 or higher in order to start. Selenium RC has two modes of operations –

- **Interactive Mode:**

  In interactive mode, commands are typed into the Selenium Server command window; this allows to immediately seeing the results of running command in a working browser.

- **Proxy Injection Mode:**

  Selenium Tests can not be run against multiple domains using Interactive mode. If tests are to be run against multiple domains then proxy injection mode should be used.

  The two experimental "elevated security privilege" browser launchers allow to test applications on any web site, including SSL/HTTPS websites, and allow your tests to freely change domains. These browsers are:

  - *iehta: Launches Internet Explorer as an HTML Application (HTA).
  - *chrome: Launches Firefox using a chrome URL.

The two experimental "proxy injection mode" browser launchers are:

- *piiexplore
- *pifirefox

"Proxy injection" mode is a new highly experimental feature for 0.9.0. In "normal mode" there are two automated test windows--one for Selenium, and one for application under test (AUT)--in proxy injection mode we eliminate the separate Selenium window, in favor of "injecting" Selenium into every HTML page. By injecting ourselves into the HTML, we have increased control over the AUT, but this comes with some risk, because we're also modifying the AUT in order to test it.

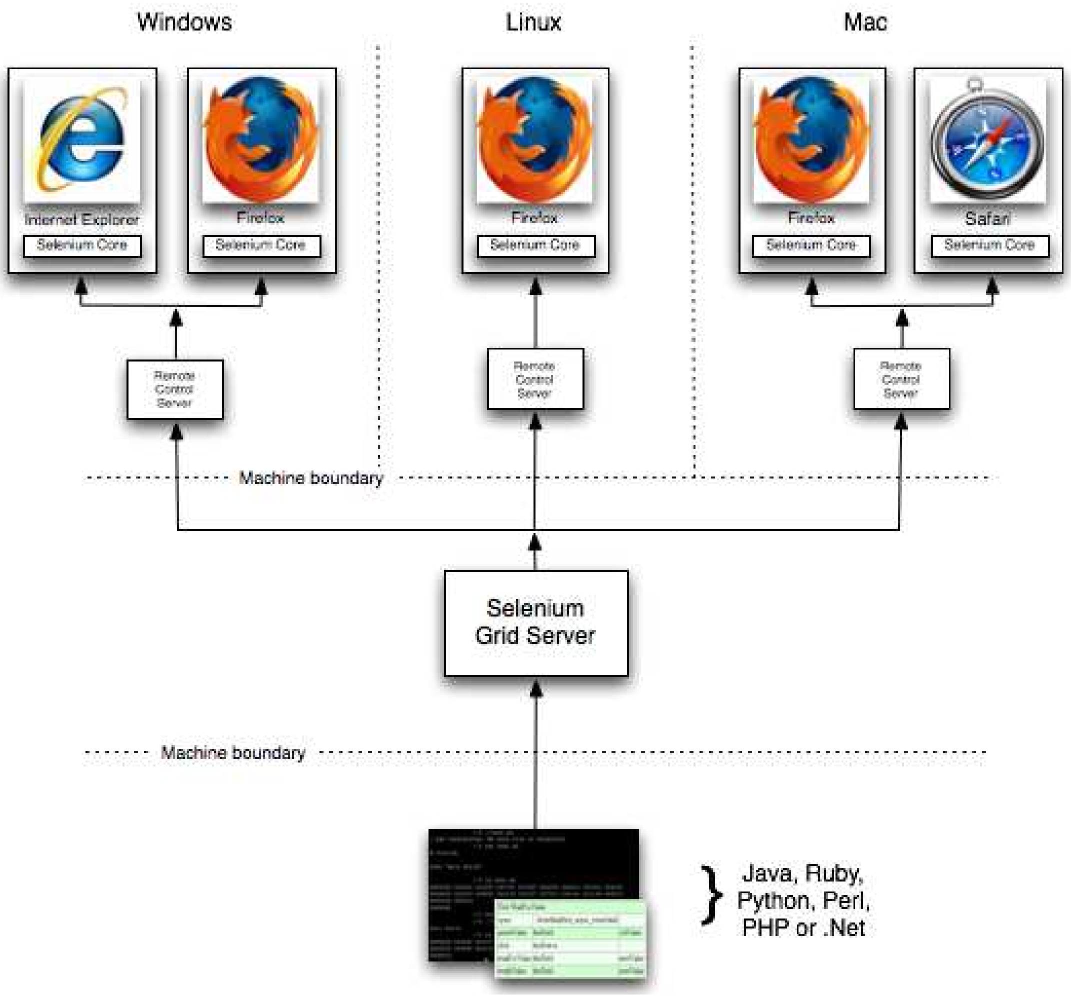
To use ProxyInjection mode, we need to start the Selenium Server with a special command line argument, like this:

Java -jar selenium-server. jar -proxyInjectionMode

5. Selenium Grid

Selenium Grid is an open-source tool that dramatically speeds up web testing by leveraging existing computing infrastructure. It allows you to run multiple tests in parallel and on multiple machines, cutting down the time required for running web acceptance tests.

Selenium Grid allows you to run multiple instances of Selenium Remote Control in parallel. It makes all these Selenium Remote Controls appear as a single one, so your tests do not have to worry about the actual infrastructure. Selenium Grid cuts down on the time required to run a Selenium test suite to a fraction of the time that a single instance of Selenium instance would take to run.



Selenium Grid

We can choose the number of nodes, the versions of each operating system. Mac OS Tiger and Leopard can be nodes in the same farm. Similarly Windows XP, Vista and Server 2003 can be nodes in the same farm.

6. Which Selenium Tool to Use

	Selenium IDE	Selenium Remote Control	Selenium Core	Selenium Core HTA
Browser Support	Firefox Only	Many	All	IE Only
Requires Remote Installation	No	No	Yes	No
Supports HTTPS/SSL	Yes	Yes*	Yes	Yes
Supports Multiple Domains	Yes	Yes*	No	Yes
Requires Java	No	Yes	No	No
Saves Test Results to Disk	No**	Yes	No	Yes
Language Support	Selenese Only	Many	Selenese Only	Selenese Only

* = experimental support is available in Selenium RC
** = theoretically possible, but not currently implemented

Note that even though Selenium RC requires Java, one can write RC tests in .NET, Perl, Python, and Ruby as well, but one still needs Java around to run the proxy.

- Not requiring remote installation and Language Support make Selenium RC most viable Selenium Tool.
- Since Selenium RC provides proxy server hence it needs not be installed on the server on which application is deployed.
- Selenium Remote Control allows to write tests in any programming language, including Java, .NET, Perl, Python and Ruby. Selenese has a number of strict limitations: it has no conditionals (no "if" statements), and it has no loops (no "for" statements). This can make writing complicated tests difficult

7. Selenium tool constraints.

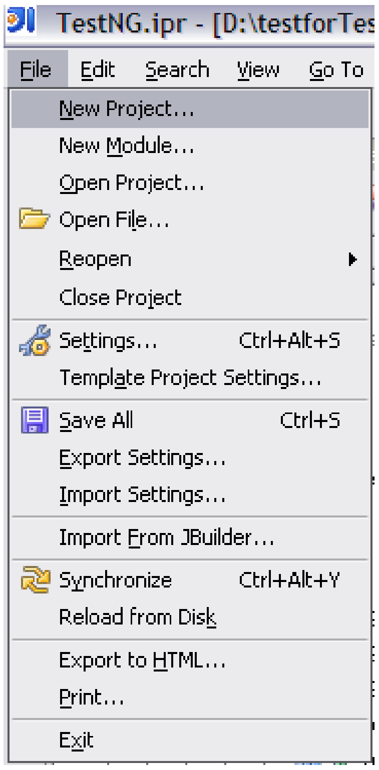
- This tool is used only of web applications.
- It does not have inbuilt reporting functionality.
- As it is an open source tool, this does not has commercial support.
- Cross domain testing browsers are experimental as of now.

8. Selenium Configuration with IntelliJ

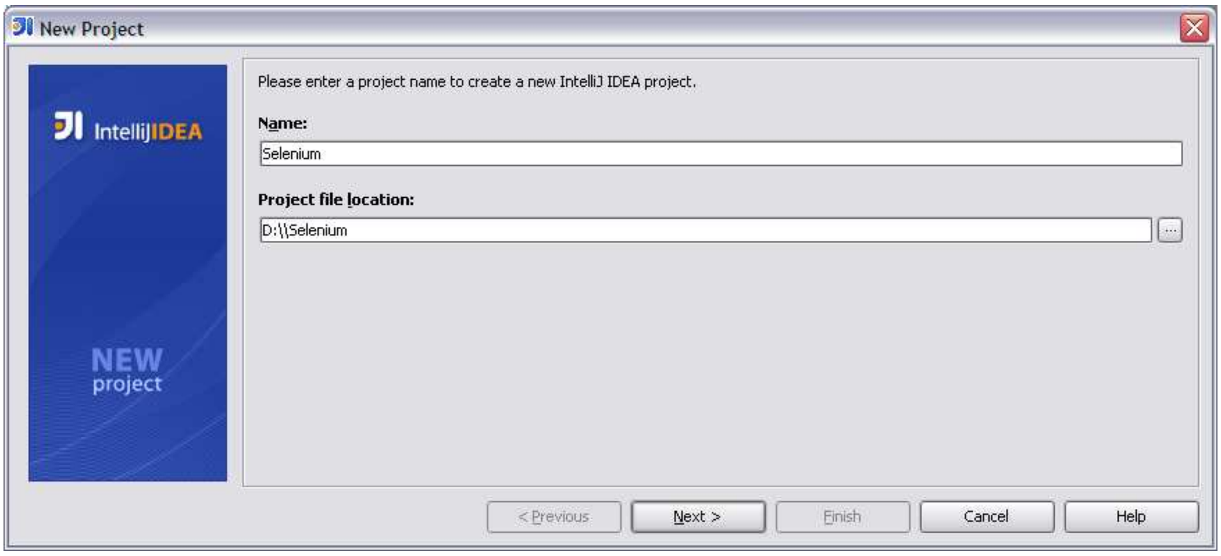
- Download Selenium RC from OpenQA site
- Start any java IDE.
- Create new project.
- Add to your project classpath selenium-java-client-driver.jar
- Record your test to from Selenium IDE and translate it to java code (Selenium IDE has automation translation feature).
- Run selenium server from console (You need initialized java environment variable to do this) like: java -jar selenium-server -proxyInjectionMode.
- Run your test in IDE.

These points have been delineated below with reference to IntelliJ IDEA:

Open a New Project in IntelliJ IDEA -



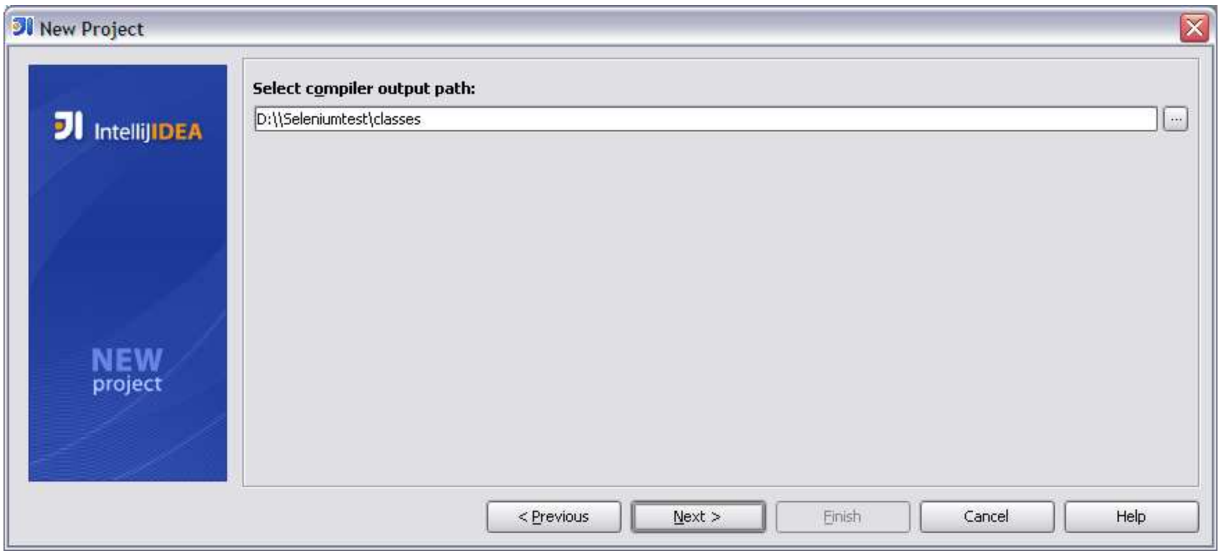
Give name and location to Project -


Please enter a project name to create a new IntelliJ IDEA project.

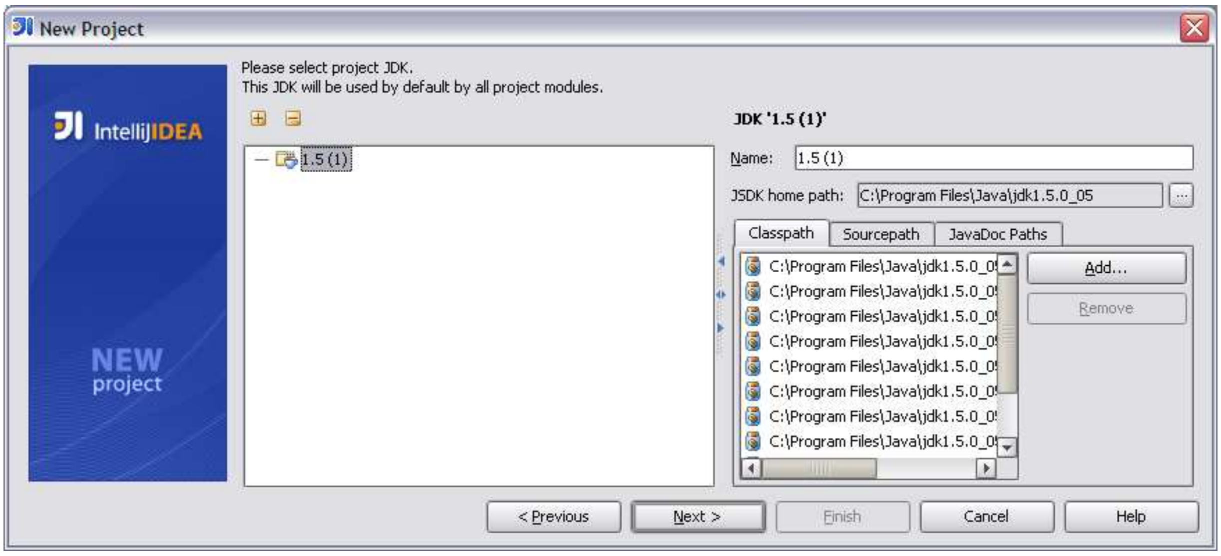
Name: Selenium

Project file location: D:\\Selenium

Click Next and provide compiler output path –


Select compiler output path: D:\\Seleniumtest\classes

Click Next and select the JDK to be used –


New Project

Please select project JDK.
This JDK will be used by default by all project modules.

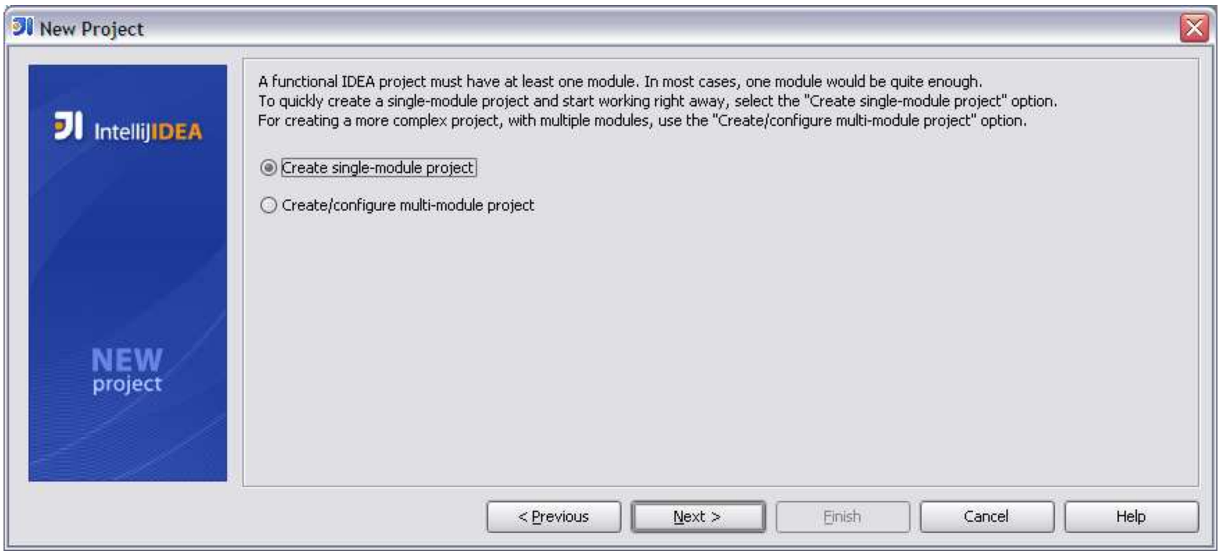
1.5 (1)

JDK '1.5 (1)'

Name: 1.5 (1)

JSDK home path: C:\Program Files\Java\jdk1.5.0_05

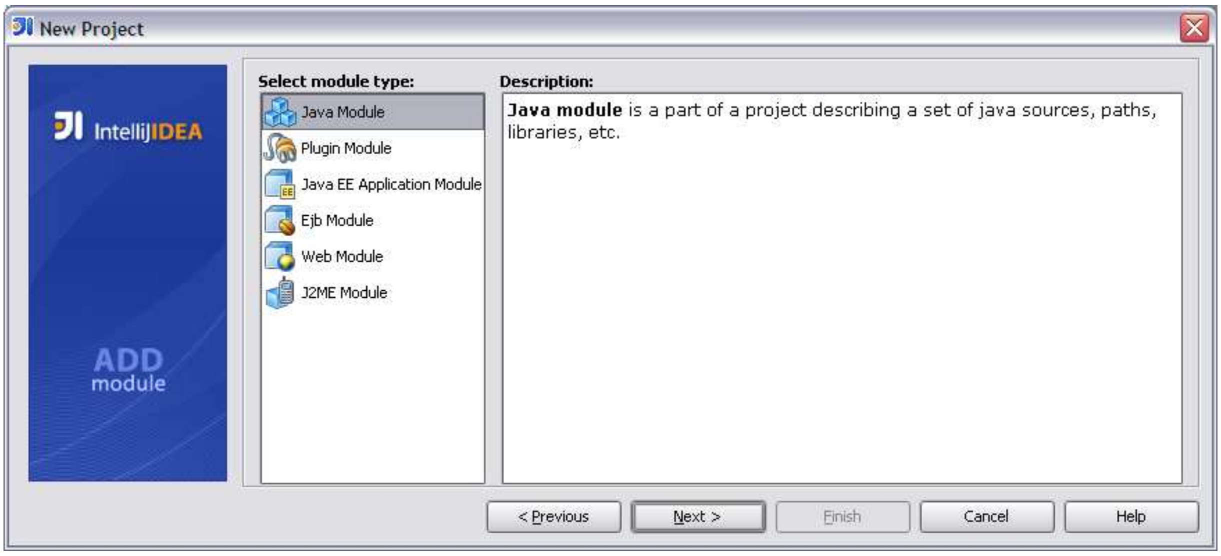
Click Next and select Single Module Project –


New Project

A functional IDEA project must have at least one module. In most cases, one module would be quite enough.
To quickly create a single-module project and start working right away, select the "Create single-module project" option.
For creating a more complex project, with multiple modules, use the "Create/configure multi-module project" option.

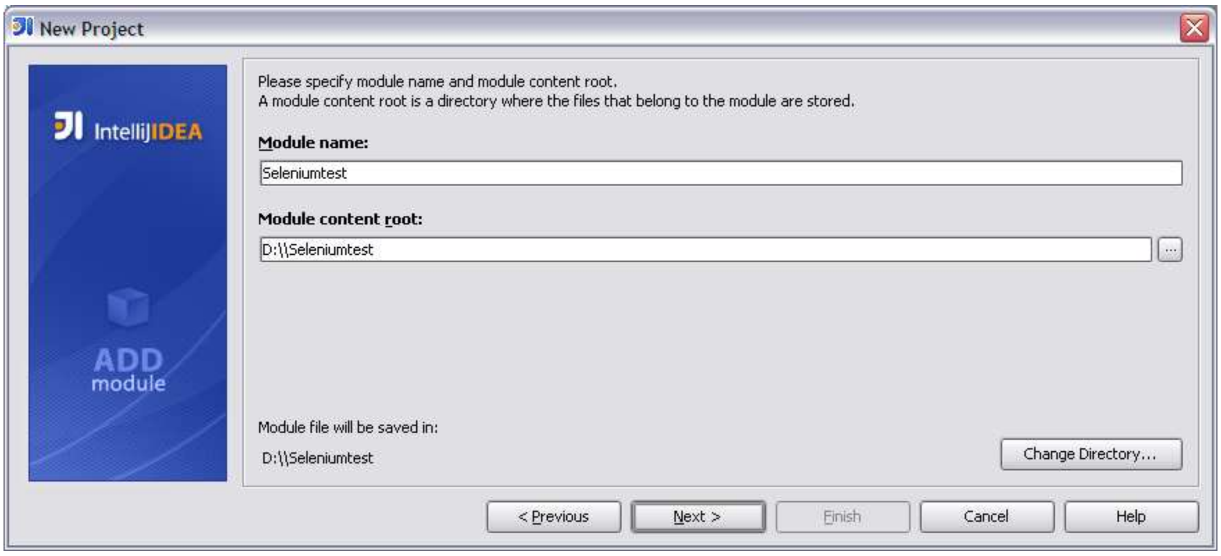
[✓ Create single-module project]
Create/configure multi-module project

Click Next and select Java module –



Java module is a part of a project describing a set of java sources, paths, libraries, etc.

Click Next and provide Module name and Module content root –



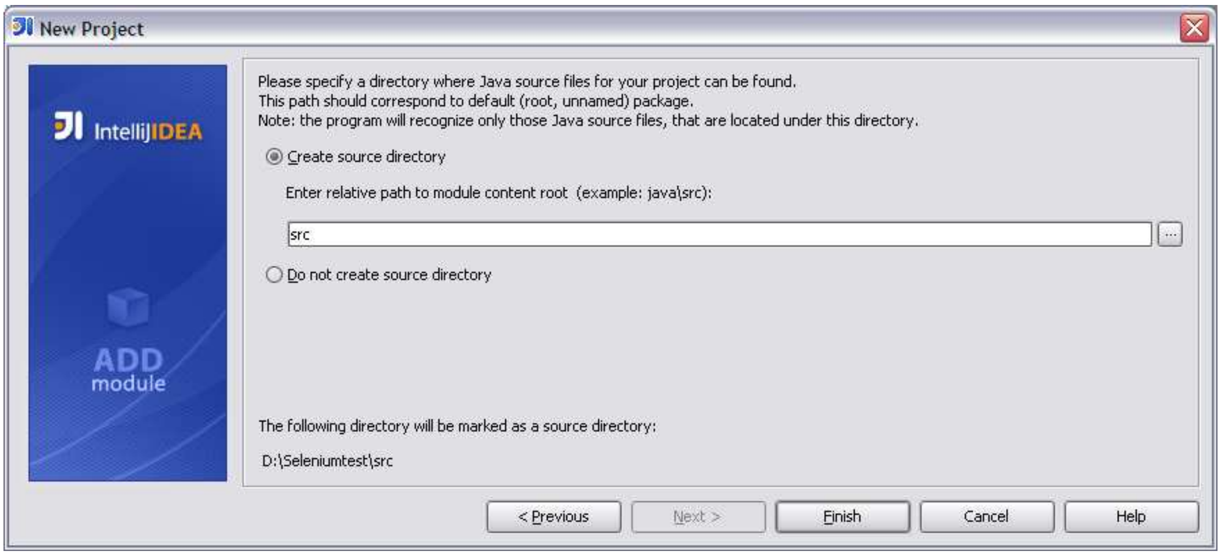
Please specify module name and module content root.
A module content root is a directory where the files that belong to the module are stored.

Module name: Seleniumtest

Module content root: D:\\Seleniumtest

Module file will be saved in:
D:\\Seleniumtest

Click Next and select Source directory –



Please specify a directory where Java source files for your project can be found.
This path should correspond to default (root, unnamed) package.
Note: the program will recognize only those Java source files, that are located under this directory.

(●) Create source directory

Enter relative path to module content root (example: java\src):

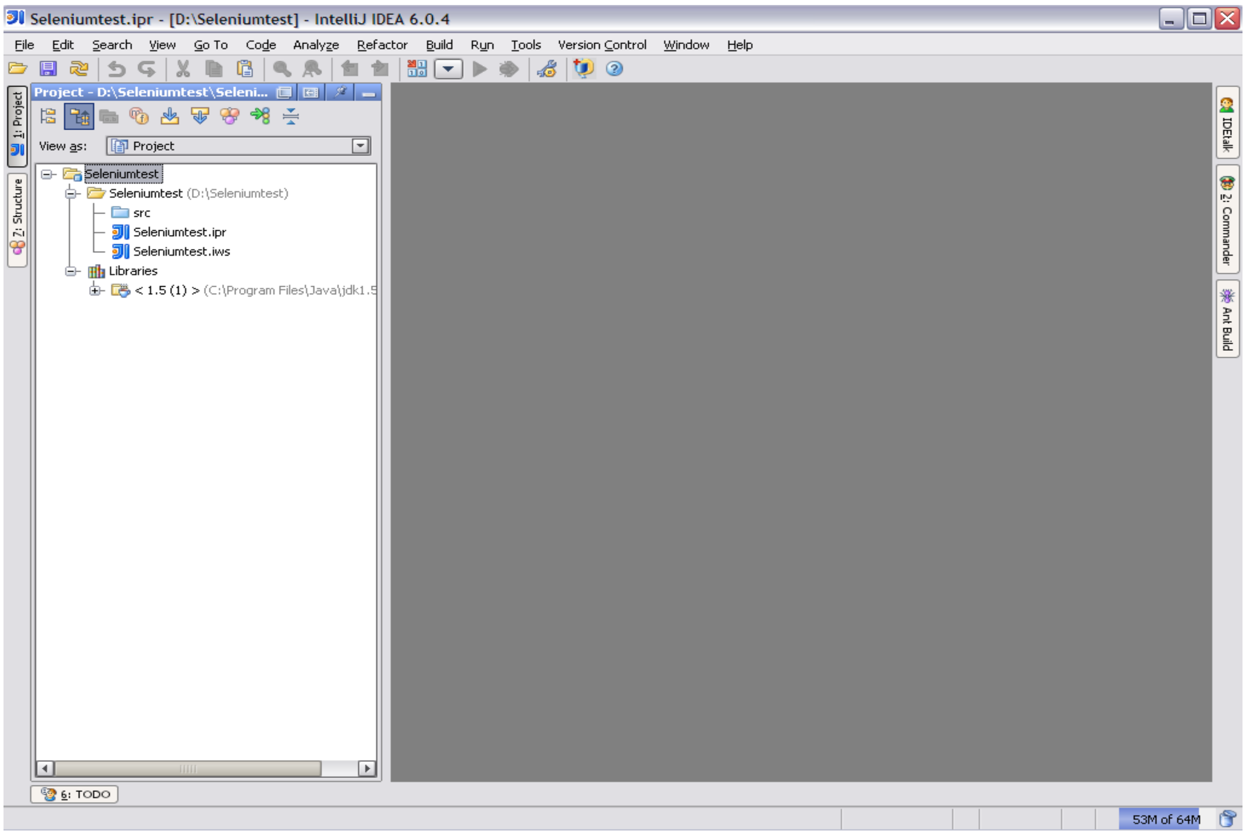
src

( ) Do not create source directory

The following directory will be marked as a source directory:

D:\Seleniumtest\src

At last click Finish. This will launch the Project Pan.

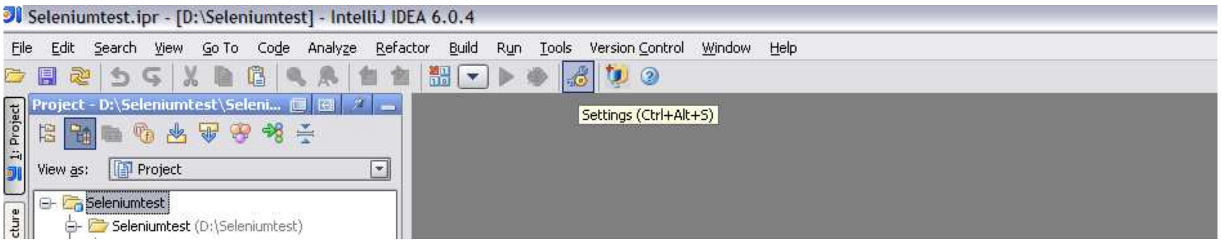


Seleniumtest.ipr - [D:\Seleniumtest] - IntelliJ IDEA 6.0.4

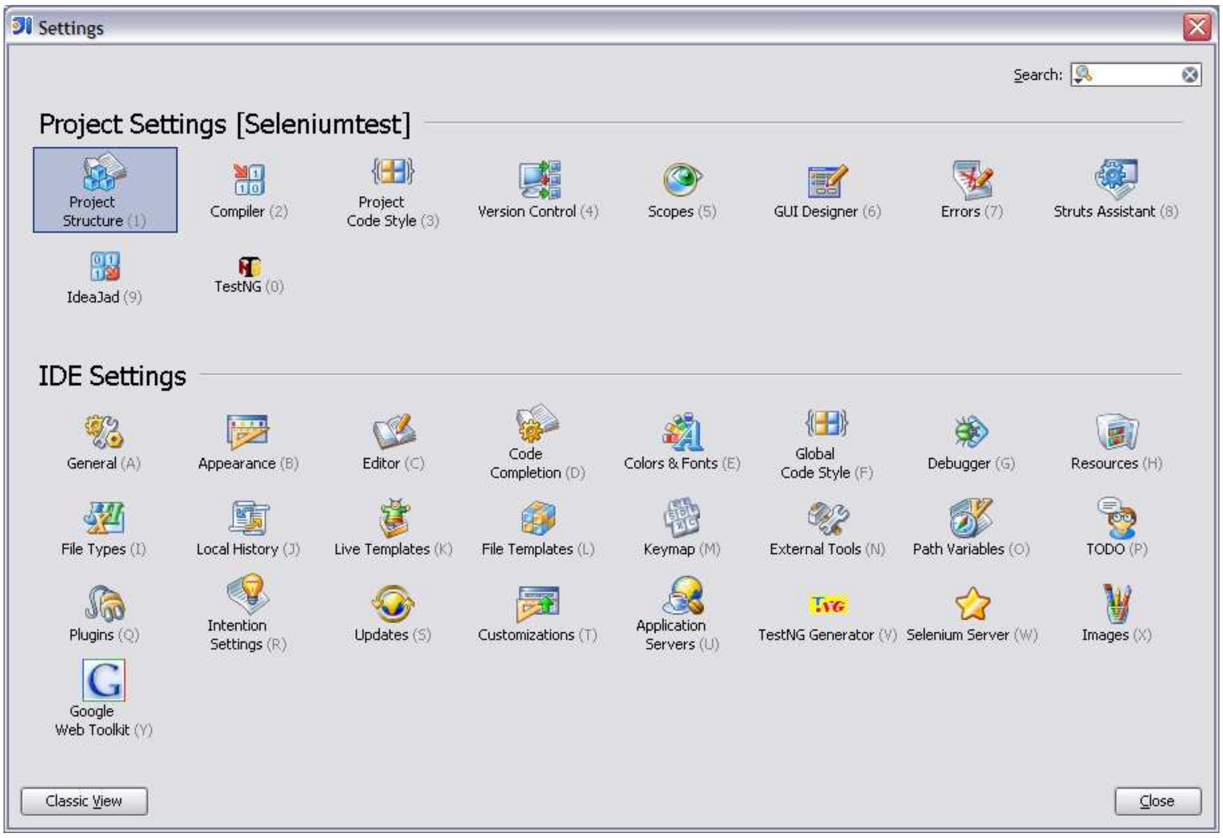
```
Seleniumtest
  Seleniumtest (D:\Seleniumtest)
    src
    Seleniumtest.ipr
    Seleniumtest.iws
  Libraries
    < 1.5 (1) > (C:\Program Files\Java\jdk1.5
```

Adding Libraries to Project:

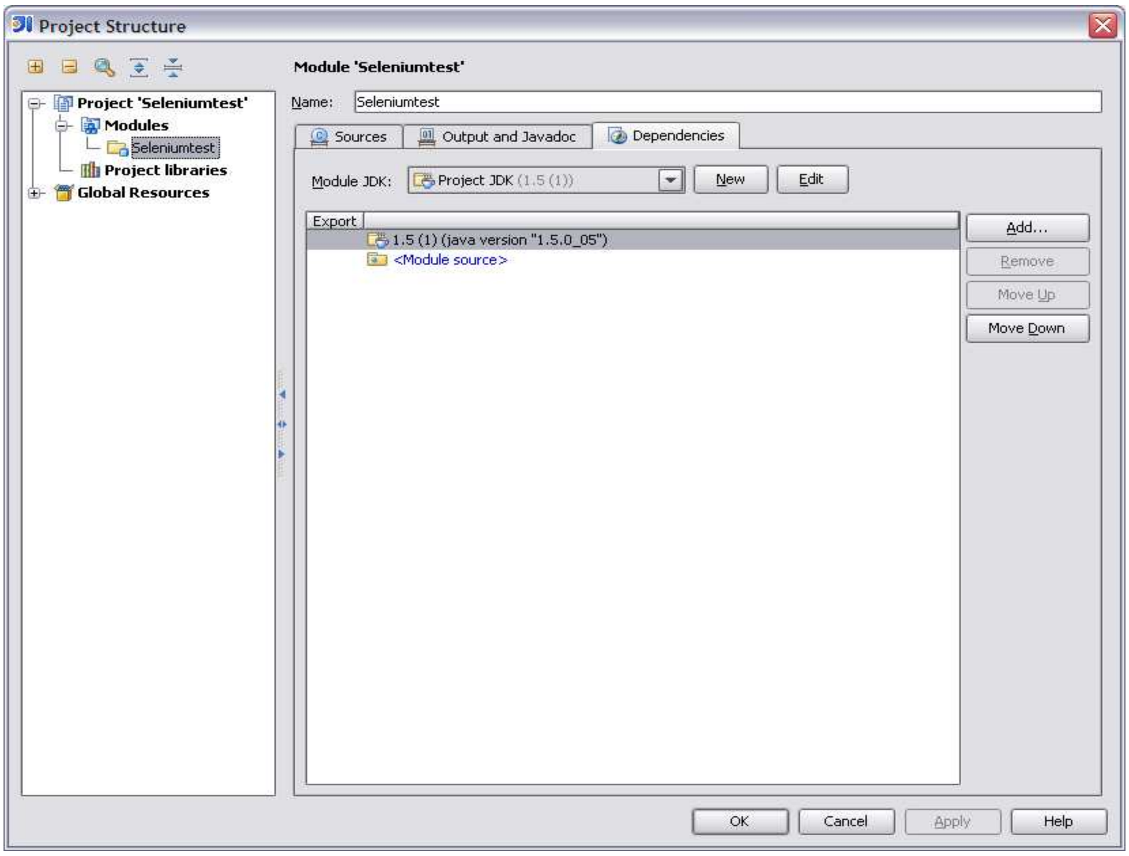
Click on Settings button in the Project Tool bar –


Seleniumtest.ipr - [D:\Seleniumtest] - IntelliJ IDEA 6.0.4
File Edit Search View Go To Code Analyze Refactor Build Run Tools Version Control Window Help
Settings (Ctrl+Alt+S)
Project - D:\Seleniumtest\Seleni...
View as: Project
Seleniumtest
Seleniumtest (D:\Seleniumtest)

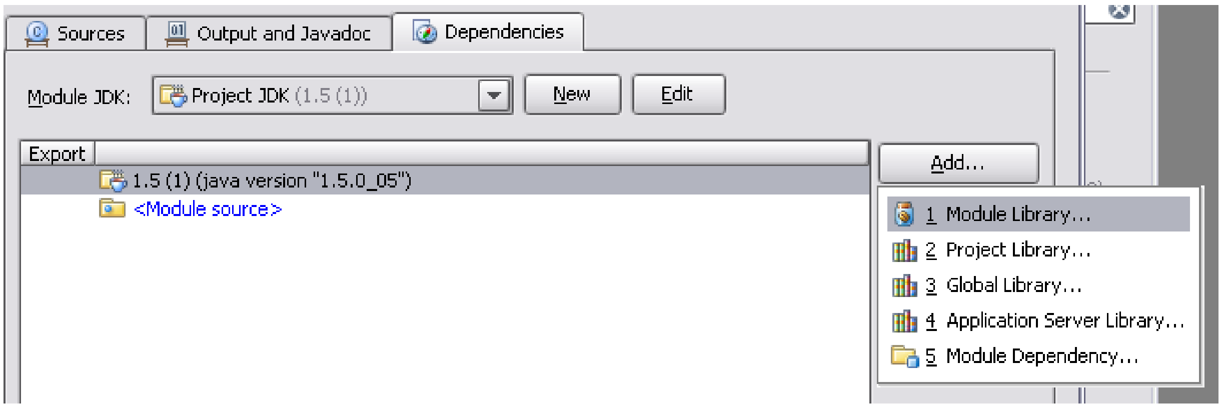
Click on Project Structure in Settings pan –



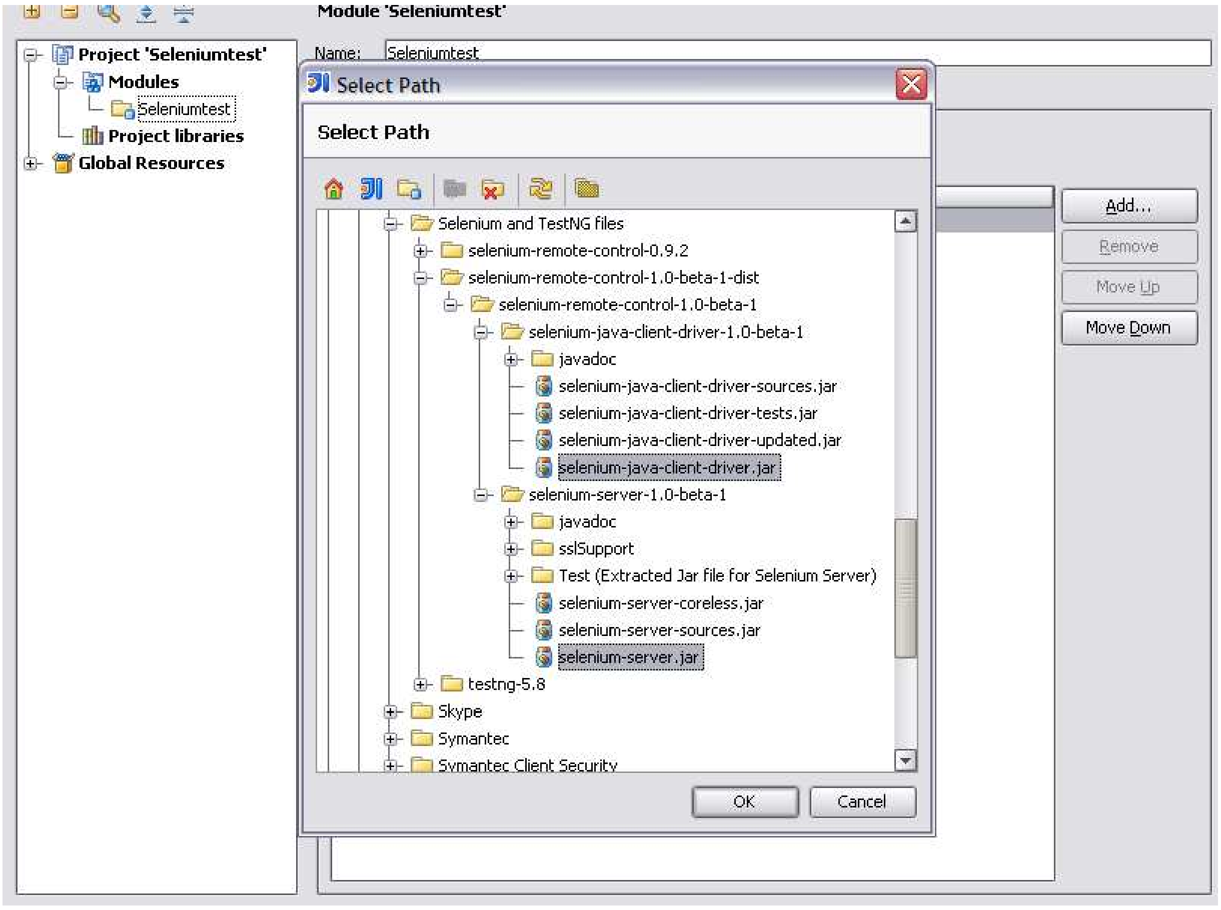
Select Module in Project Structure and browse to Dependencies tab –



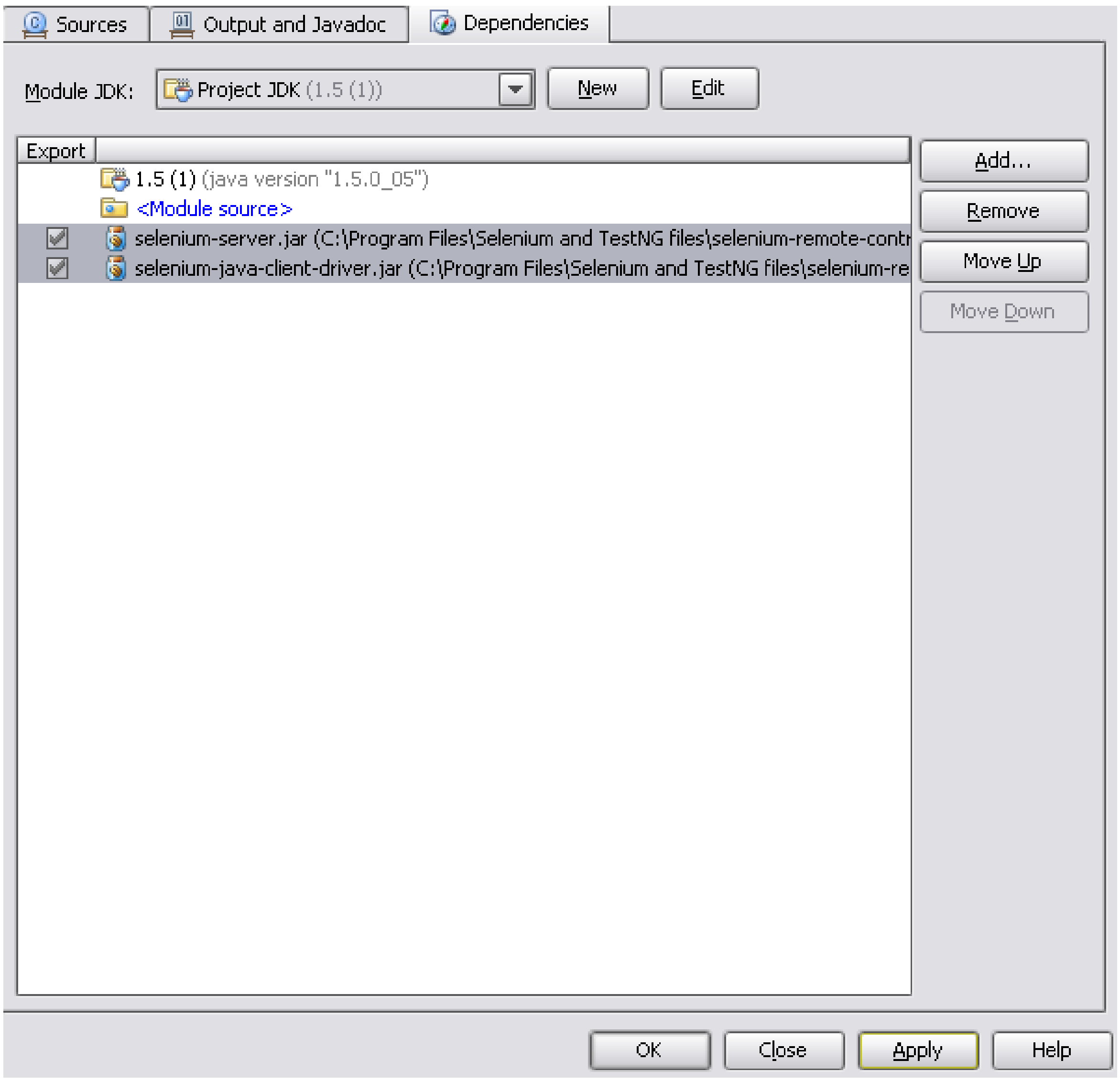
Click on Add button followed by the click on Module Library –



Browse to the Selenium directory and select selenium-java-client-driver.jar and selenium-server.jar. (Multiple Jars can be selected b holding down the control key.) –


Module 'Seleniumtest'
Name: Seleniumtest
Project 'Seleniumtest'
Modules
Seleniumtest
Project libraries
Global Resources
Select Path
Select Path
Selenium and TestNG files
selenium-remote-control-0.9.2
selenium-remote-control-1.0-beta-1-dist
selenium-remote-control-1.0-beta-1
selenium-java-client-driver-1.0-beta-1
javadoc
selenium-java-client-driver-sources.jar
selenium-java-client-driver-tests.jar
selenium-java-client-driver-updated.jar
selenium-java-client-driver.jar
selenium-server-1.0-beta-1
javadoc
sslSupport
Test (Extracted Jar file for Selenium Server)
selenium-server-coreless.jar
selenium-server-sources.jar
selenium-server.jar
testng-5.8
Skype
Symantec
Symantec Client Security
OK
Cancel
Add...
Remove
Move Up
Move Down

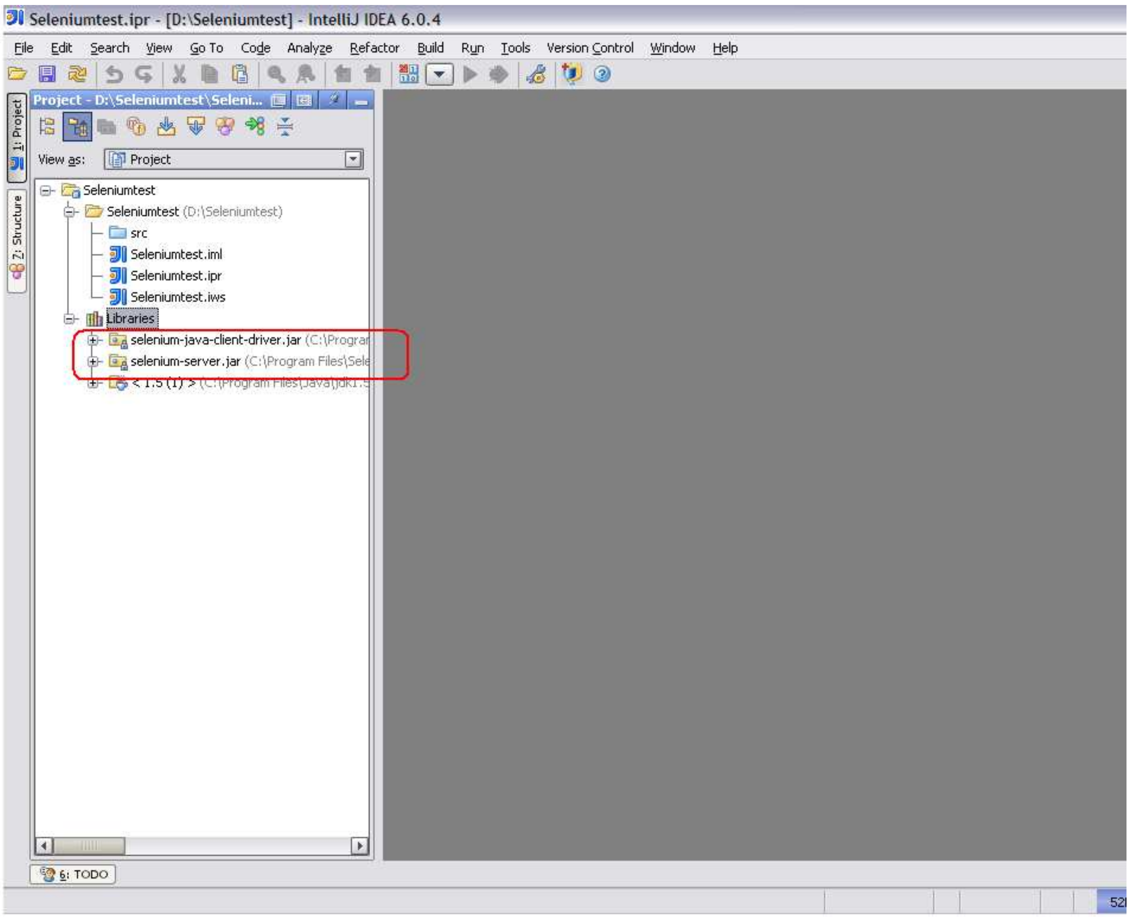
Select both jar files in project pan and click on Apply button –



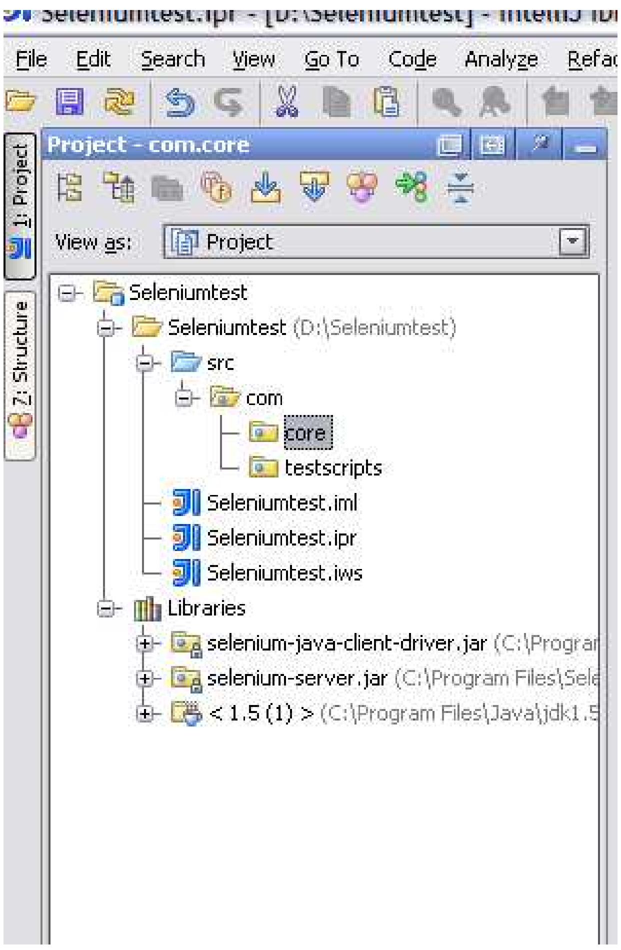
Module JDK: Project JDK (1.5 (1))

1.5 (1) (java version "1.5.0_05")
<Module source>
☑ selenium-server.jar (C:\Program Files\Selenium and TestNG files\selenium-remote-contr
☑ selenium-java-client-driver.jar (C:\Program Files\Selenium and TestNG files\selenium-re

Now click ok on Project Structure followed by click on Close on Project Settings pan. Added jars would appear in project Library as following –

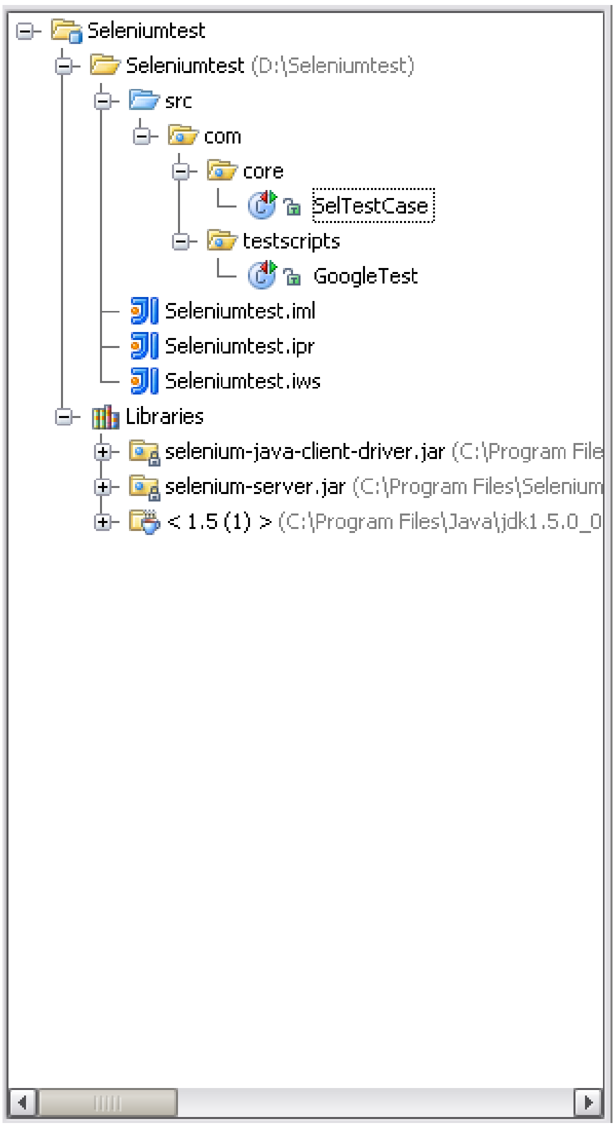

Seleniumtest.ipr - [D:\Seleniumtest] - IntelliJ IDEA 6.0.4
File Edit Search View Go To Code Analyze Refactor Build Run Tools Version Control Window Help
Project - D:\Seleniumtest\Seleni...
View as: Project
Seleniumtest
Seleniumtest (D:\Seleniumtest)
src
Seleniumtest.iml
Seleniumtest.ipr
Seleniumtest.iws
Libraries
selenium-java-client-driver.jar (C:\Progra
selenium-server.jar (C:\Program Files\Sele
< 1.5 (1) > (C:\Program Files\Java\jdk1.5
1: Project
7: Structure
6: TODO

Create the directory structure in src folder as following –


Seleniumtest.ipr - [D:\Seleniumtest] - IntelliJ ID
File Edit Search View Go To Code Analyze Refac
1: Project
7: Structure
Project - com.core
View as: Project
Seleniumtest
Seleniumtest (D:\Seleniumtest)
src
com
core
testscripts
Seleniumtest.iml
Seleniumtest.ipr
Seleniumtest.iws
Libraries
selenium-java-client-driver.jar (C:\Progra
selenium-server.jar (C:\Program Files\Sele
< 1.5 (1) > (C:\Program Files\Java\jdk1.5

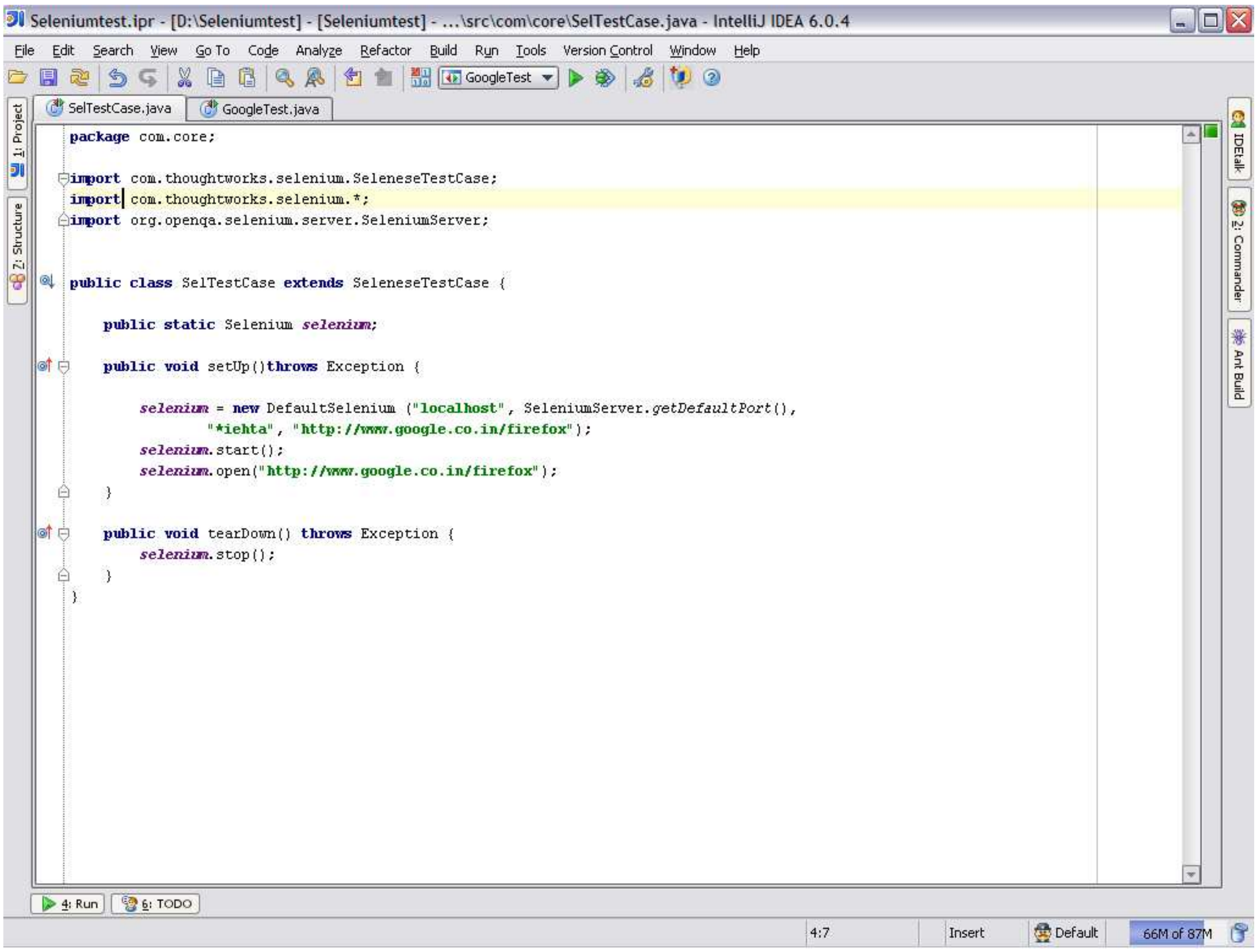
Note: This is not any hard and fast convention and just a convention hence it might differ from project to project.

Most basically *'core'* contains the SelTestCase class which is used to create Selenium object and fire up the browser. *'testscripts'* package contains the test classes which extend the SelTestCase class. Hence extended structure would look as following –



```
Seleniumtest
  Seleniumtest (D:\Seleniumtest)
    src
      com
        core
          SelTestCase
        testscripts
          GoogleTest
    Seleniumtest.iml
    Seleniumtest.ipr
    Seleniumtest.iws
  Libraries
    selenium-java-client-driver.jar (C:\Program File
    selenium-server.jar (C:\Program Files\Selenium
    < 1.5 (1) > (C:\Program Files\Java\jdk1.5.0_0
```

Here SelTestCase class extends **SeleneseTestCase** class of Selenium API which in turns extends **TestCase** class of Junit API. Hence capabilities of Junit would be available in Test Scripts. Most basic SelTestCase class can have following statements –

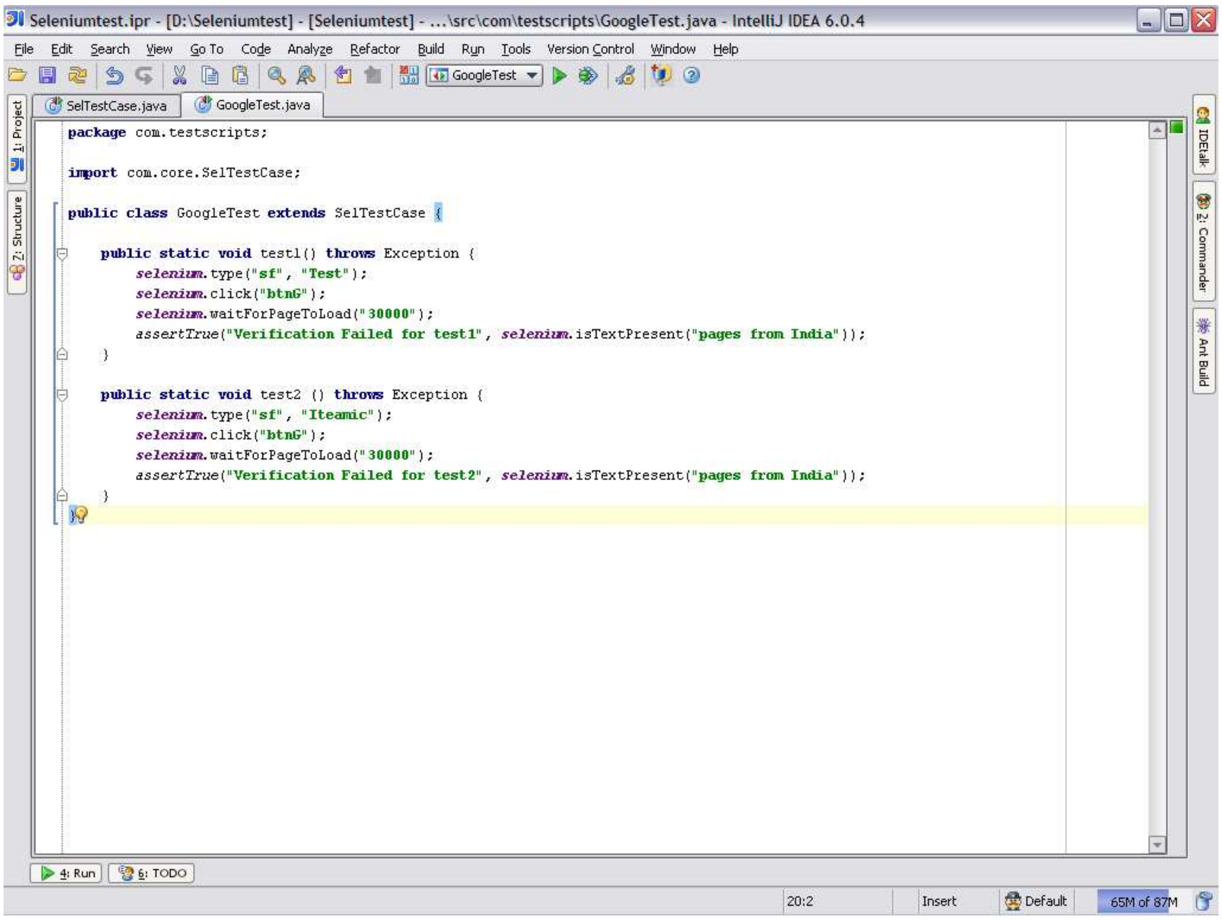

Seleniumtest.ipr - [D:\Seleniumtest] - [Seleniumtest] - ...\src\com\core\SelTestCase.java - IntelliJ IDEA 6.0.4
File Edit Search View Go To Code Analyze Refactor Build Run Tools Version Control Window Help
GoogleTest
SelTestCase.java
GoogleTest.java
4: Run
6: TODO
4:7
Insert
Default
66M of 87M

```java
package com.core;

import com.thoughtworks.selenium.SeleneseTestCase;
import com.thoughtworks.selenium.*;
import org.openqa.selenium.server.SeleniumServer;


public class SelTestCase extends SeleneseTestCase {

    public static Selenium selenium;

    public void setUp()throws Exception {

        selenium = new DefaultSelenium ("localhost", SeleniumServer.getDefaultPort(),
                "*iehta", "http://www.google.co.in/firefox");
        selenium.start();
        selenium.open("http://www.google.co.in/firefox");
    }

    public void tearDown() throws Exception {
        selenium.stop();
    }
}
```

Here ***setup*** and ***tearDown*** methods of SeleneseTestCase class are overridden in SelTestCase class.

setup method fires up browser before each test method and ***tearDown*** method closes the browser after each test method.

Test script for one of test method is as following –

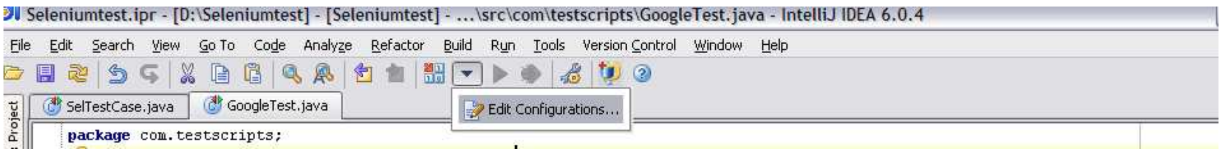

Seleniumtest.ipr - [D:\Seleniumtest] - [Seleniumtest] - ...\src\com\testscripts\GoogleTest.java - IntelliJ IDEA 6.0.4
File Edit Search View Go To Code Analyze Refactor Build Run Tools Version Control Window Help
GoogleTest
SelTestCase.java
GoogleTest.java
1: Project
Z: Structure
IDEtalk
2: Commander
Ant Build
4: Run
6: TODO
20:2
Insert
Default
65M of 87M

```java
package com.testscripts;

import com.core.SelTestCase;

public class GoogleTest extends SelTestCase {

    public static void test1() throws Exception {
        selenium.type("sf", "Test");
        selenium.click("btnG");
        selenium.waitForPageToLoad("30000");
        assertTrue("Verification Failed for test1", selenium.isTextPresent("pages from India"));
    }

    public static void test2 () throws Exception {
        selenium.type("sf", "Iteamic");
        selenium.click("btnG");
        selenium.waitForPageToLoad("30000");
        assertTrue("Verification Failed for test2", selenium.isTextPresent("pages from India"));
    }
}
```

Notice that test class GoogleTest extends the SelTestCase class and going by this way each test class would extend the SelTestCase class. Going forward in a project all common methods would be kept in SelTestCase class hence each of the class can use it. This is known as ***abstraction*** in java. Establishing JDBC connection can be its example.

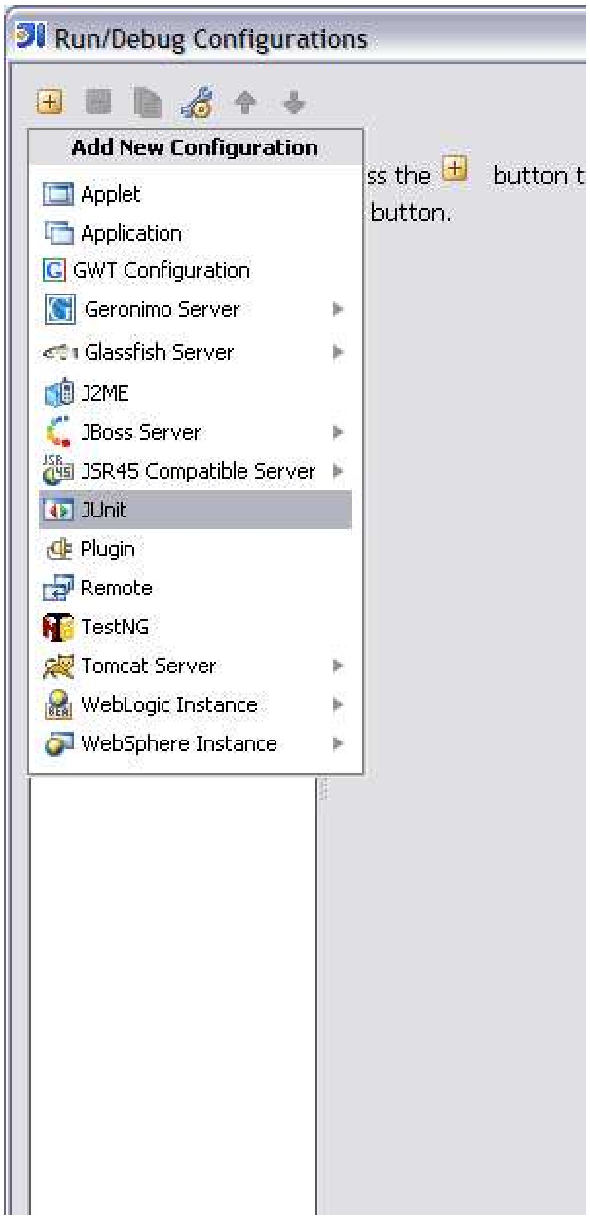
Running the Test:

Click on Edit configuration in Tool Bar -

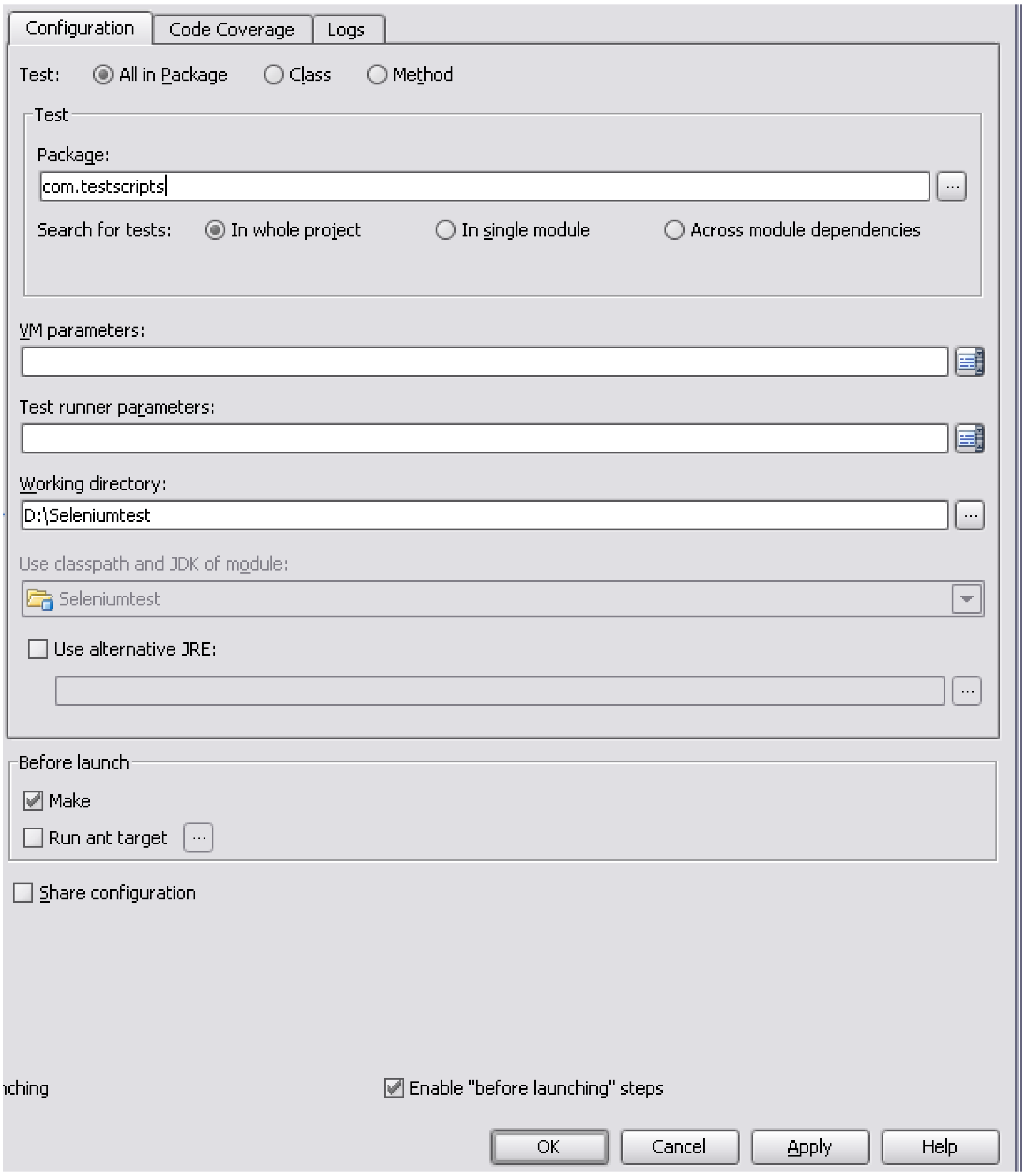

Seleniumtest.ipr - [D:\Seleniumtest] - [Seleniumtest] - ...\src\com\testscripts\GoogleTest.java - IntelliJ IDEA 6.0.4
File Edit Search View Go To Code Analyze Refactor Build Run Tools Version Control Window Help
SelTestCase.java
GoogleTest.java
Edit Configurations...
Project

```java
package com.testscripts;
```

Click on Add symbol and select JUnit from drop down –



Select the package for which tests are to be run and click on Apply button followed by click on Ok button -



Test: [✓ All in Package] Class Method

Test

Package: com.testscripts

Search for tests: [✓ In whole project] In single module Across module dependencies

VM parameters:

Test runner parameters:

Working directory: D:\Seleniumtest

Use classpath and JDK of module: Seleniumtest

☐ Use alternative JRE:

Before launch

✓ Make

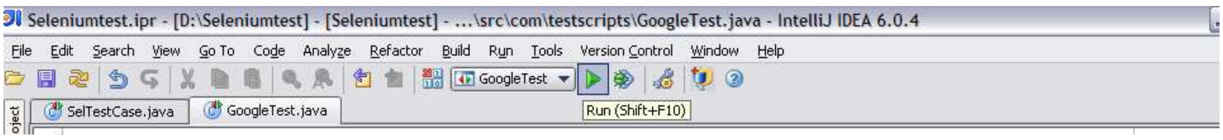
☐ Run ant target

☐ Share configuration

ching

✓ Enable "before launching" steps

Now click on **Run** button to run the tests –


Seleniumtest.ipr - [D:\Seleniumtest] - [Seleniumtest] - ...\src\com\testscripts\GoogleTest.java - IntelliJ IDEA 6.0.4
File Edit Search View Go To Code Analyze Refactor Build Run Tools Version Control Window Help
GoogleTest
SelTestCase.java
GoogleTest.java
Run (Shift+F10)

9. Further Readings and Scope

http://www.openqa.org/

http://safsdev.sourceforge.net/FRAMESDataDrivenTestAutomationFrameworks.htm

http://www.junit.org/

http://www.testng.org/

http://release.openqa.org/selenium-remote-control/0.9.2/doc/java/com/thoughtworks/selenium/package-summary.html

http://junit.sourceforge.net/javadoc/