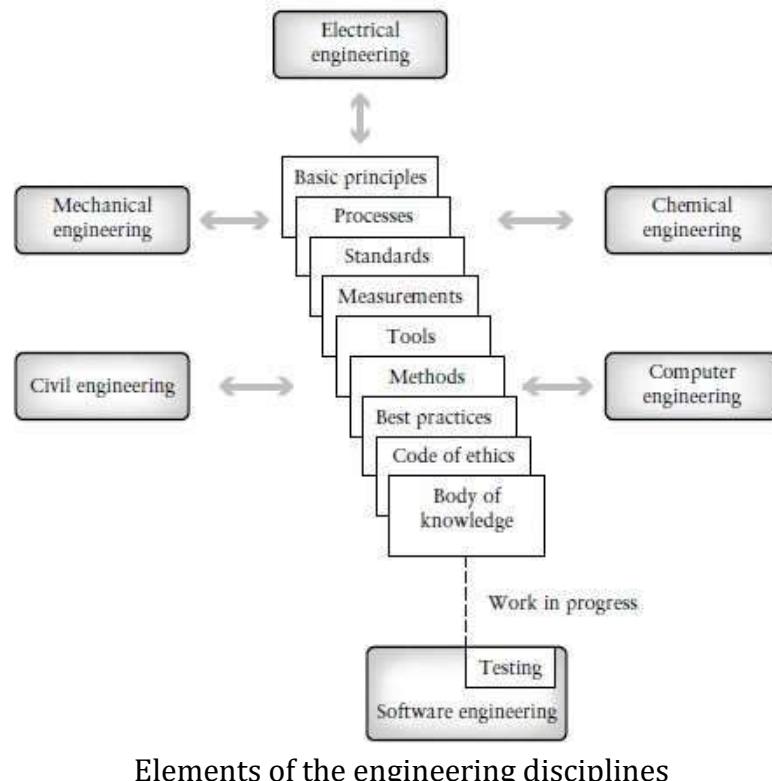


Software Testing Class Notes

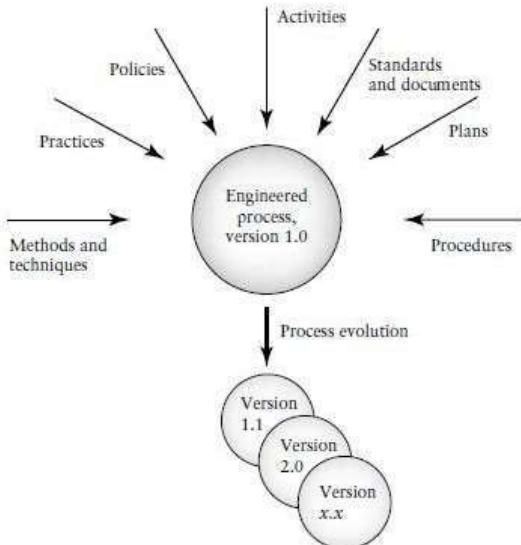
1.1 The Evolving Profession of Software Engineering

- the development process is well understood;
- projects are planned;
- life cycle models are defined and adhered to;
- standards are in place for product and process;
- measurements are employed to evaluate product and process quality;
- components are reused;



The Role of Process in Software Quality

Process, in the software engineering domain, is the set of methods, practices, standards, documents, activities, policies, and procedures that software engineers use to develop and maintain a software system and its associated artifacts, such as project and test plans, design documents, code, and manuals.

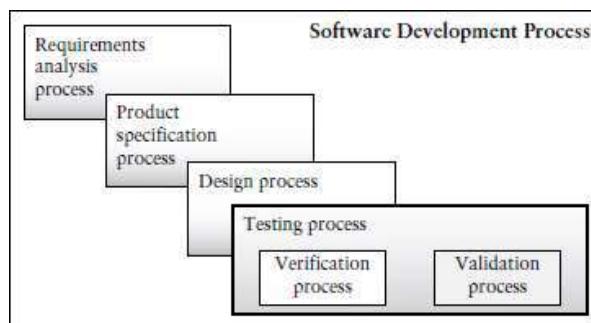


Components of an engineering process

Testing as a Process

Validation is the process of evaluating a software system or component during, or at the end of, the development cycle in order to determine whether it satisfies specified requirements.

Verification is the process of evaluating a software system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.



Example process embedded in the software development

Testing is generally described as a group of procedures carried out to evaluate some aspect of a piece of software.

Testing can be described as a process used for revealing defects in software, and for establishing that the software has attained a specified degree of quality with respect to selected attributes.

An Overview of the Testing Maturity Model

1. There is a demand for software of high quality with low defects;
2. Process is important in the software engineering discipline;
3. Software testing is an important software development sub process;
4. Existing software evaluation and improvement models have not adequately addressed testing issues.

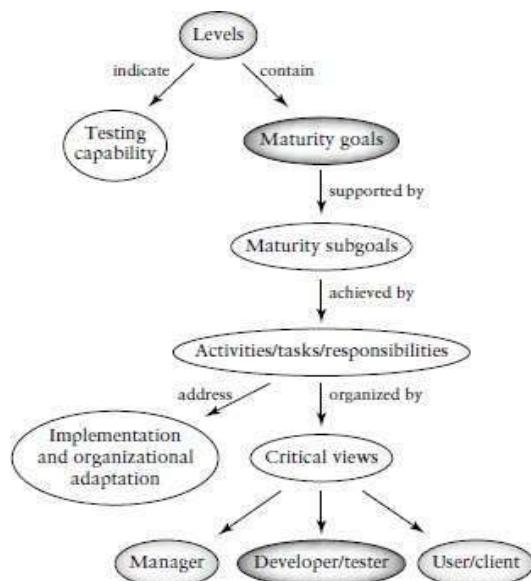
In the testing domain possible benefits of test process improvement are the following:

- smarter testers
- higher quality software
- the ability to meet budget and scheduling goals
- improved planning
- the ability to meet quantifiable testing goals

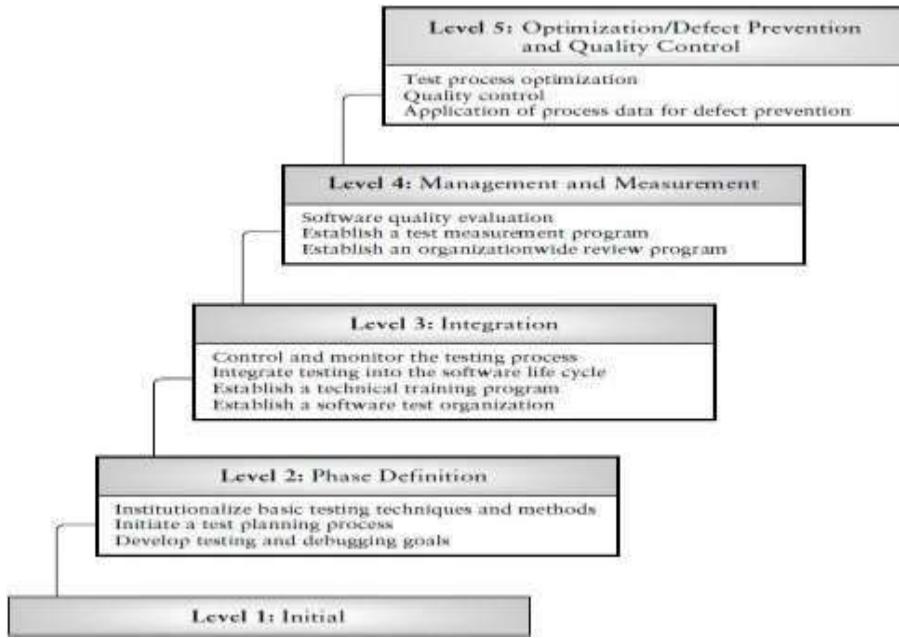
TMM Levels

A set of *maturity goals*. The maturity goals identify testing improvement goals that must be addressed in order to achieve maturity at that level. To be placed at a level, an organization must satisfy the maturity goals at that level. The TMM levels and associated maturity goals are shown in Figure.

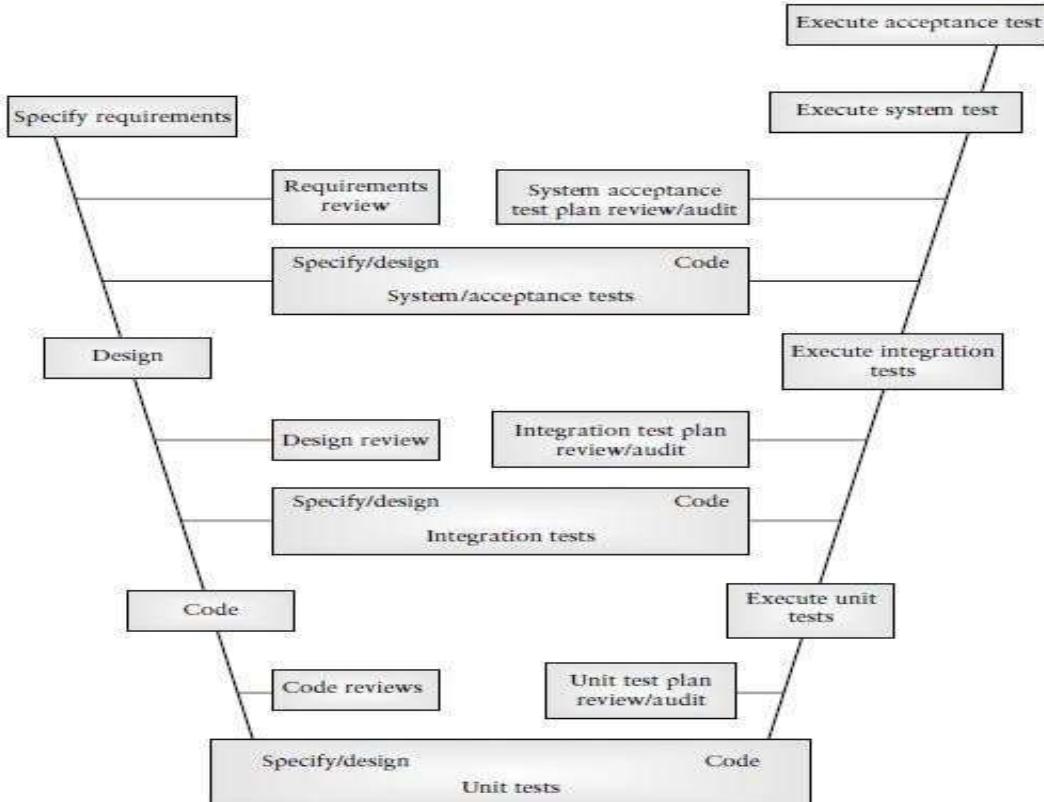
- *Supporting maturity subgoals*. They define the scope, boundaries and needed accomplishments for a particular level.
- *Activities, tasks and responsibilities (ATR)*. The ATRs address implementation and organizational adaptation issues at each TMM



The internal structure of TMM maturity levels



The s-level structure of the testing maturity model



The Extended/Modified V-model

1.2 Testing Fundamentals Error & Faults (Defects)

- An error is a mistake, misconception, or misunderstanding on the part of a software developer.
- A fault (defect) is introduced into the software as the result of an error. It is an anomaly in the software that may cause it to behave incorrectly, and not according to its specification.

Failures

- A failure is the inability of a software system or component to perform its required functions within specified performance requirements

Test Cases

- A test case in a practical sense is a test-related item which contains the following information:
- *A set of test inputs.* These are data items received from an external source by the code under test. The external source can be hardware, software, or human.
- *Execution conditions.* These are conditions required for running the test, for example, a certain state of a database, or a configuration of a hardware device.
- *Expected outputs.* These are the specified results to be produced by the code under test.

Test

- A test is a group of related test cases, or a group of related test cases and test procedures

Test Oracle

- A test oracle is a document, or piece of software that allows testers to determine whether a test has been passed or failed.

Test Bed

- A test bed is an environment that contains all the hardware and software needed to test a software component or a software system.

Software Quality

- Quality relates to the degree to which a system, system component, or process meets specified requirements.
- Quality relates to the degree to which a system, system component, or process meets customer or user needs, or expectations.
- A metric is a quantitative measure of the degree to which a system, system component, or process possesses a given attribute
- A quality metric is a quantitative measurement of the degree to which an item possesses a given quality attribute

Software Quality Assurance Group

- The software quality assurance (SQA) group is a team of people with the necessary training and skills to ensure that all necessary actions are taken during the development process so that the resulting software conforms to

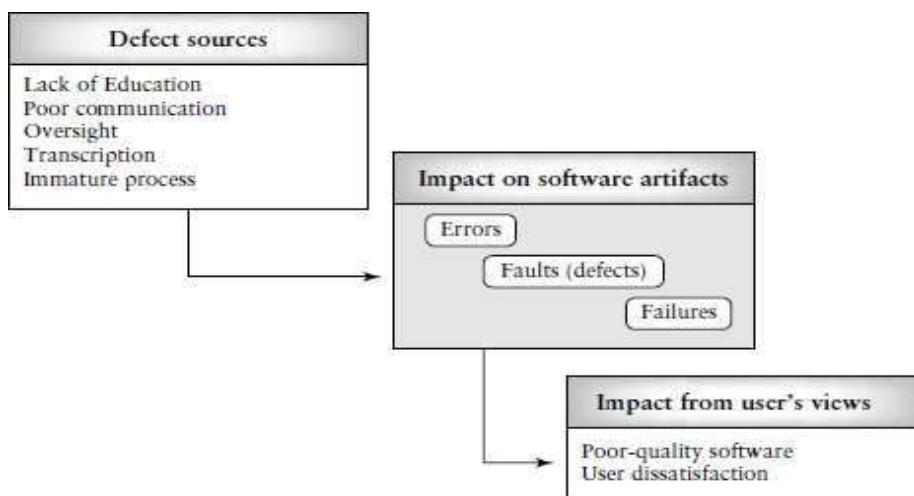
established technical requirements.

Reviews

- A review is a group meeting whose purpose is to evaluate a software artifact or a set of software artifacts.

1.3 DEFECTS, HYPOTHESES, AND TESTS

Origins of Defects



Hypotheses

- design test cases;
- design test procedures;
- assemble test sets;
- select the testing levels (unit, integration, etc.)
- appropriate for the tests;
- evaluate the results of the tests

Defect Classes, the Defect Repository, and Test Design

- Requirements and Specification Defects
- Design Defects
- Coding Defects
- Testing Defects

Requirements and Specification Defects

- Functional Description Defects
- Feature Defects
- Feature Interaction Defects
- Interface Description Defects

Design Defects

- Algorithmic and Processing Defects
- Control, Logic, and Sequence Defects
- Data Defects
- Module Interface Description Defects
- Functional Description Defects

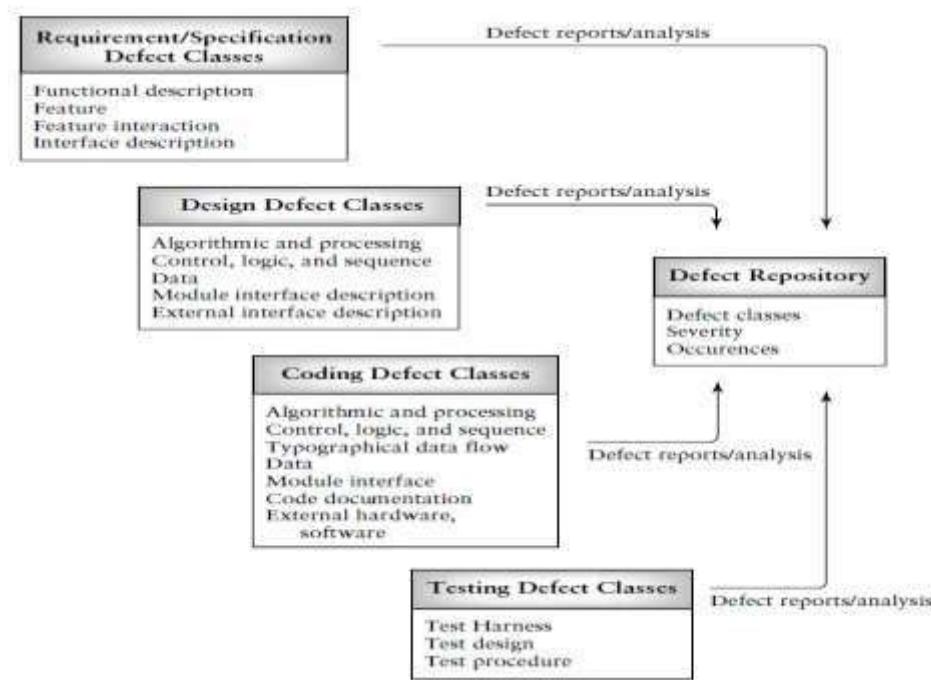
- External Interface Description Defects

Coding Defects

- Algorithmic and Processing Defects
- Control, Logic and Sequence Defects
- Typographical Defects
- Initialization Defects
- Data-Flow Defects
- Data Defects
- Module Interface Defects
- Code Documentation Defects
- External Hardware, Software Interfaces Defects

Testing Defects

- Test Harness Defects
- Test Case Design and Test Procedure Defects



Defect Examples: The Coin Problem

- A precondition is a condition that must be true in order for a software component to operate properly.
- In this case a useful precondition would be one that states for example: `number_of_coins > 0`
- **A postcondition is a condition that must be true when a software component completes its operation properly.**

- A useful postcondition would be:
number_of_dollars, number_of_cents >= 0.

Specification for Program calculate_coin_value

This program calculates the total dollars and cents value for a set of coins. The user inputs the amount of pennies, nickels, dimes, quarters, half-dollars, and dollar coins held. There are six different denominations of coins. The program outputs the total dollar and cent values of the coins to the user.

Inputs: number_of_coins is an integer
 Outputs: number_of_dollars is an integer
 number_of_cents is an integer

A sample specification with defects.

```
Design Description for Program calculate_coin_values

Program calculate_coin_values
number_of_coins is integer
total_coin_value is integer
number_of_dollars is integer
number_of_cents is integer
coin_values is array of six integers representing
each coin value in cents
initialized to: 1,5,10,25,25,100
begin

    initialize total_coin_value to zero
    initialize loop_counter to one
    while loop_counter is less than six
    begin
        output "enter number of coins"
        read (number_of_coins )
        total_coin_value = total_coin_value +
        number_of_coins * coin_value[loop_counter]
        increment loop_counter
    end
    number_dollars = total_coin_value/100
    number_of_cents = total_coin_value - 100 * number_of_dollars
    output (number_of_dollars, number_of_cents)
end
```

A sample design specification with defects.

- **Control, logic, and sequencing defects.** The defect in this subclass arises from an incorrect “while” loop condition (should be less than or equal to six)
- **Data defects.** This defect relates to an incorrect value for one of the elements of the integer array, coin_values, which should read 1,5,10,25,50,100.

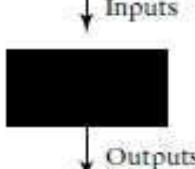
```
*****  
program calculate_coin_values calculates the dollar and cents  
value of a set of coins of different dominations input by the user  
denominations are pennies, nickels, dimes, quarters, half dollars,  
and dollars  
*****  
main ()  
{  
int total_coin_value;  
int number_of_coins = 0;  
int number_of_dollars = 0;  
int number_of_cents = 0;  
int coin_values = {1,5,10,25,25,100};  
{  
int i = 1;  
while ( i < 6)  
{  
    printf("input number of coins\n");  
    scanf ("%d", number_of_coins);  
    total_coin_value = total_coin_value +  
        (number_of_coins * coin_value[i]);  
}  
i = i + 1;  
number_of_dollars = total_coin_value/100;  
number_of_cents = total_coin_value - (100 * number_of_dollars);  
printf("%d\n", number_of_dollars);  
printf("%d\n", number_of_cents);  
}  
*****
```

A code example with defect

Test Case Design

Strategies and Methods for Test Case Design

-I Test Case Design Strategies

Test Strategy	Tester's View	Knowledge Sources	Methods
Black box		Requirements document Specifications Domain knowledge Defect analysis data	Equivalence class partitioning Boundary value analysis State transition testing Cause and effect graphing Error guessing
White box		High-level design Detailed design Control flow graphs Cyclomatic complexity	Statement testing Branch testing Path testing Data flow testing Mutation testing Loop testing

The two basic testing strategies

Using the Black Box Approach to Test Case Design

- black box test strategy where we are considering only inputs and outputs as a basis for designing test cases
- how do we choose a suitable set of inputs from the set of all possible valid and invalid inputs
- As a example, suppose you tried to test a single procedure that calculates the square root of a number.
- If you were to exhaustively test it you would have to try all positive input values.
- what about all negative numbers, fractions? These are also possible inputs.

2.1 Equivalence Class Partitioning

- A good approach to selecting test inputs is to use a method called equivalence class partitioning.
- advantages:
 - It eliminates the need for exhaustive testing, which is not feasible.
 - It guides a tester in selecting a subset of test inputs with a high probability of detecting a defect.
 - It allows a tester to cover a larger domain of inputs/outputs with a smaller subset selected from an equivalence class.

There are several important points related to equivalence class partitioning

- The tester must consider both valid and invalid equivalence classes.
- Equivalence classes may also be selected for output conditions.
- The derivation of input or outputs equivalence classes is a heuristic process.

List of Conditions

- If an input condition for the software-under-test is specified as a range of values
- If an input condition for the software-under-test is specified as a number of values

- If an input condition for the software-under-test is specified as a set of valid input values
- If an input condition for the software-under-test is specified as a *-must be* || condition

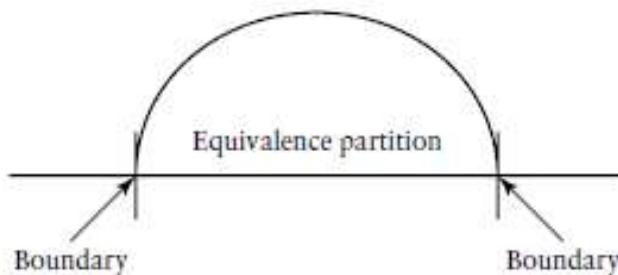
```

Function square_root
  message (x:real)
  when x >= 0.0
    reply (y:real)
      where y >= 0.0 & approximately (y*y,x)
  otherwise reply exception imaginary_square_root
end function

```

- EC1. The input variable x is real, valid.
- EC2. The input variable x is not real, invalid.
- EC3. The value of x is greater than 0.0, valid.
- EC4. The value of x is less than 0.0, invalid.

2.2 Boundary Value Analysis



Boundaries of an equivalence partition

An Example of the Application of Equivalence Class Partitioning and Boundary Value Analysis

- Widget identifiers into a widget data base
- We have three separate conditions that apply to the input:
 - (i) it must consist of alphanumeric characters
 - (ii) the range for the total number of characters is between 3 and 15,
 - (iii) the first two characters must be letters.
- First we consider condition 1, the requirement for alphanumeric characters. This is a *-must be* || condition. We derive two equivalence classes.
 - EC1. Part name is alphanumeric, valid.
 - EC2. Part name is not alphanumeric, invalid.
- Then we treat condition 2, the range of allowed characters 3–15.
 - EC3. The widget identifier has between 3 and 15 characters, valid.
 - EC4. The widget identifier has less than 3 characters, invalid.
 - EC5. The widget identifier has greater than 15 characters, invalid.

- Finally we treat the -must be|| case for the first two characters.
 - EC6. The first 2 characters are letters, valid.
 - EC7. The first 2 characters are not letters, invalid.

Condition	Valid equivalence classes	Invalid equivalence classes
1	EC1	EC2
2	EC3	EC4, EC5
3	EC6	EC7

Fig 2.3 Example equivalence class reporting table

- For example:
 - BLB**—a value just below the lower bound
 - LB**—the value on the lower boundary
 - ALB**—a value just above the lower boundary
 - BUB**—a value just below the upper bound
 - UB**—the value on the upper bound
 - AUB**—a value just above the upper bound
- For our example module the values for the bounds groups are:
 - BLB**—2
 - BUB**—14
 - LB**—3
 - UB**—15
 - ALB**—4
 - AUB**—16

Module name: Insert_Widget Module identifier: AP62-Mod4 Date: January 31, 2000 Tester: Michelle Jordan			
Test case identifier	Input values	Valid equivalence classes and bounds covered	Invalid equivalence classes and bounds covered
1	abc1	EC1, EC3(ALB) EC6	
2	ab1	EC1, EC3(LB), EC6	
3	abcdef123456789	EC1, EC3 (UB) EC6	
4	abcde123456789	EC1, EC3 (BUB) EC6	
5	abc*	EC3(ALB), EC6	EC2
6	ab	EC1, EC6	EC4(BLB)
7	abcdefg123456789	EC1, EC6	EC5(AUB)
8	a123	EC1, EC3 (ALB)	EC7
9	abcdef123	EC1, EC3, EC6 (typical case)	

Summary of test inputs using equivalence class partitioning and boundary value analysis for sample module

- A major weakness with equivalence class partitioning is that it does not allow testers to combine conditions.

2.3 Cause-and-Effect Graphing

- The tester must decompose the specification of a complex software component into lower-level units.
- For each specification unit, the tester needs to identify causes and their effects.
- From the cause-and-effect information, a Boolean cause-and-effect graph is created. Nodes in the graph are causes and effects.
- The graph may be annotated with constraints that describe combinations of causes and/or effects that are not possible due to environmental or syntactic constraints.
- The graph is then converted to a decision table.
- The columns in the decision table are transformed into test cases.

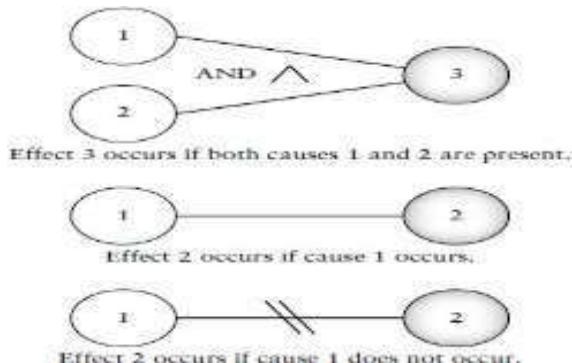


Fig 2.5 Sample of cause-and-effect graph notations

- The input conditions, or causes are as follows:

C1: Positive integer from 1 to 80

C2: Character to search for is in string The output conditions, or effects are:

E1: Integer out of range

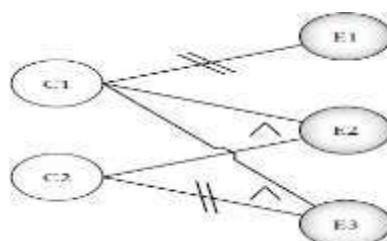
E2: Position of character in

string E3: Character not found

The rules or relationships can be described as follows: If C1 and C2, then E2.

If C1 and not C2, then

E3. If not C1, then E1.



Cause-and-effect graph for the character search example

- A decision table will have a row for each cause and each effect.
- Entries in the table can be represented by a -1|| for a cause or effect that is present, a -0|| represents the absence of a cause or effect, and a ---|| indicates a -don't care|| value.
- A decision table for our simple example is shown in Table 4.3 where C1, C2, C3 represent the causes, E1, E2, E3 the effects, and columns T1, T2, T3 the test cases.

	Inputs	Length	Character to search for	Outputs
T1	5	c		3
T2	5	w		Not found
T3	90			Integer out of range

	T1	T2	T3
C1	1	1	0
C2	1	0	—
E1	0	0	1
E2	1	0	0
E3	0	1	0

Decision table for character search example

2.4 State Transition Testing

- A state is an internal configuration of a system or component. It is defined in terms of the values assumed at a particular time for the variables that characterize the system or component.
- A finite-state machine is an abstract machine that can be represented by a state graph having a finite number of states and a finite number of transitions between states.

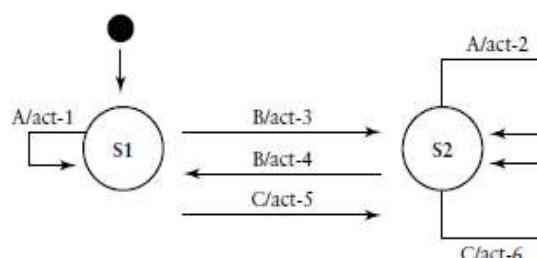


Fig. 2.8 Simple state transition graph

- For example, the transition from S1 to S2 occurs with input, or event B. Action 3 occurs as part of this state transition. This is represented by the symbol -B/act3.
- For the simple state machine in Figure 4.6 and Table 4.4 the transitions to be tested are:
 - Input A in S1
 - Input A in S2
 - Input B in S1
 - Input B in S2
 - Input C in S1
 - Input C in S2

	S1	S2
Inputs		
Input A	S1 (act-1)	S2 (act-2)
Input B	S2 (act-3)	S1 (act-4)
Input C	S2 (act-5)	S2 (act-6)

Fig 2.9 A state table for the machine

2.5 Error Guessing

- Designing test cases using the error guessing approach is based on the tester's/developer's past experience

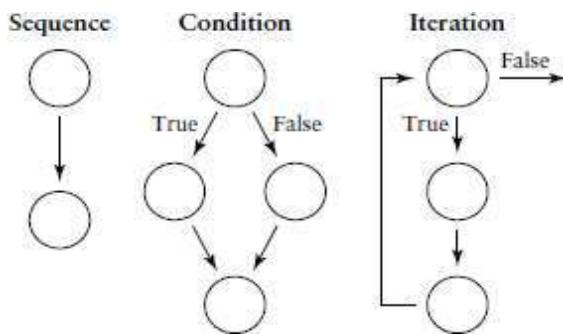
Strategies and Methods for Test Case Design

-II Using the White Box Approach to Test Case Design

- A test data set is statement, or branch, adequate if a test set T for program P causes all the statements, or branches, to be executed respectively.

2.6 Coverage and Control Flow Graphs

- program statements
- decisions/branches (these influence the program flow of control)
- conditions (expressions that evaluate to true/false, and do not contain any other true/false-valued expressions)
- combinations of decisions and conditions
- paths (node sequences in flow graphs)
- All structured programs can be built from three basic primes-sequential (e.g., assignment statements), decision (e.g., if/then/else statements), and iterative (e.g., while, for loops).



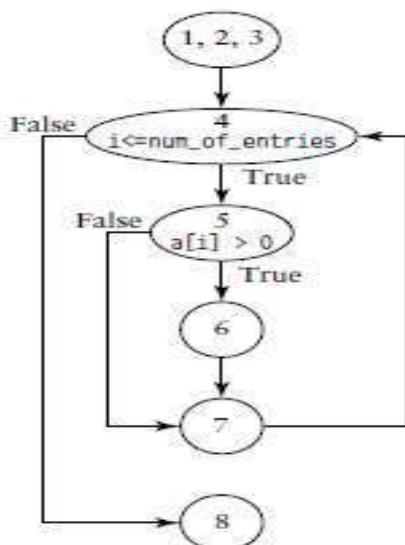
Representation of program

primes Covering Code Logic

```
/* pos_sum finds the sum of all positive numbers (greater than zero) stored in an integer array a. Input parameters are num_of_entries, an integer, and a, an array of integers with num_of_entries elements. The output parameter is the integer sum */
```

1. pos_sum(a, num_of_entries, sum)
2. sum = 0
3. inti = 1
4. while (i <= num_of_entries)
5. if a[i] > 0
6. sum = sum + a[i]
7. endif
8. i = i + 1
9. end while
10. end pos_sum

Code sample with branch and loop



A control flow graph representation for the code

Decision or branch	Value of variable i	Value of predicate	Test case: Value of a, num_of_entries
			a=1, -45,3 num_of_entries = 3
while	1	True	
	4	False	
if	1	True	
	2	False	

A test case for the code in 2.11 that satisfied the decision coverage criterion

```
if(age <65 and married true)
do X
do Y .....
else
do Z
```

- Condition 1: Age less than 65
- Condition 2: Married is true

Test cases for simple decision coverage

Value for age	Value for married	Decision outcome (compound predicate as a whole)	Test case ID
30	True	True	1
75	True	False	2

Test cases for condition coverage

Value for age	Value for married	Condition 1 outcome	Condition 2 outcome	Test case ID
75	True	False	True	2
30	False	True	False	3

2.7 Paths: Their Role in White Box-Based Test Design

A path is a sequence of control flow nodes usually beginning from the entry node of a graph through to the exit node.

8 White Box Test Design Approaches

2.8.1 Data Flow and White Box Test Design

We say a variable is defined in a statement when its value is assigned or changed.

- For example in the statements

Y= 26 * X
Read (Y)

- the variable Y is defined, that is, it is assigned a new value. In data flow notation this is indicated as a *def* for the variable Y .

We say a variable is used in a statement when its value is utilized in a statement. The value of the variable is not changed.

- They describe a predicate use (p-use) for a variable that indicates its role in a predicate. A computational use (c-use) indicates the variable's role as a part of a computation. In both cases the variable value is unchanged.
- For example, in the statement $Y = 26 * X$
- the variable X is used. Specifically it has a c-use.
- In the statement if ($X > 98$)
 $Y = \max$

- X has a predicate or p-use.

1	sum = 0	sum, def
2	read (n),	n, def
3	i = 1	i, def
4	while (i <= n)	i, n p-sue
5	read (number)	number, def
6.	sum = sum + number	sum, def, sum, number, c-use
7	i = i + 1	i, def, c-use
8	end while	
9	print (sum)	sum, c-use

Fig 2.14 Sample code with data flow information

2.8.2 Loop Testing

- Loop testing strategies focus on detecting common defects associated with these structures.
 - (i) zero iterations of the loop, i.e., the loop is skipped in its entirely;
 - (ii) one iteration of the loop;
 - (iii) two iterations of the loop;
 - (iv) k iterations of the loop where $k < n$;

- (v) $n - 1$ iterations of the loop;
- (vi) $n + 1$ iterations of the loop (if possible).

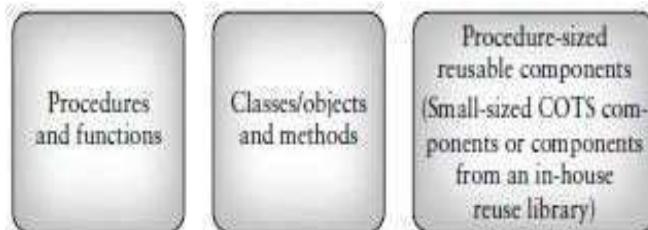
Levels of Testing

3.1 Unit Test: Functions, Procedures, Classes, and Methods as Units

- A unit is the smallest possible testable software component.

It can be characterized in several ways. For example, a unit in a typical

- procedure-oriented software system;
- performs a single cohesive function;
- can be compiled separately;
- is a task in a work breakdown structure (from the manager's point of view);
- contains code that can fit on a single page or screen.



some components suitable for unit test

Unit Test: The Need for Preparation

- To prepare for unit test the developer/tester must perform several tasks. These are:
 - (i) plan the general approach to unit testing;
 - (ii) design the test cases, and test procedures (these will be attached to the test plan);
 - (iii) define relationships between the tests;
 - (iv) prepare the auxiliary code necessary for unit test.

Unit Test Planning

- **Phase 1: Describe Unit Test Approach and Risks**

In this phase of unit testing planning the general approach to unit testing is outlined. The test planner::

- (i) identifies test risks;
- (ii) describes techniques to be used for designing the test cases for the units;
- (iii) describes techniques to be used for data validation and recording of test results;
- (iv) describes the requirements for test harnesses and other software that interfaces with the units to be tested, for example, any special objects needed for testing object-oriented units.

- **Phase 2: Identify Unit Features to be Tested**

- **Phase 3: Add Levels of Detail to the**

Plan Designing the Unit Tests

- Part of the preparation work for unit test involves unit test design. It is important to specify

- (i) the test cases (including input data, and expected outputs for each test case), and,
- (ii) the test procedures (steps required run the tests)

The Class as a Testable Unit: Special Considerations

- Issue 1: Adequately Testing Classes

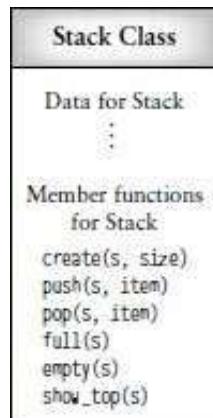


Fig 3.2 Sample stack class with multiple methods

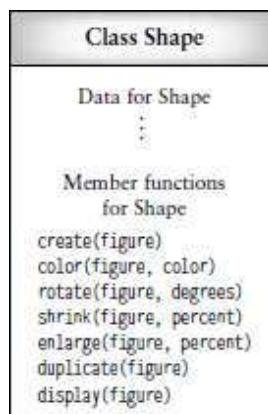
Issue 2: Observation of Object States and State Changes

- `empty(s)`, `push(s,item-1)`, `show_top(s)`, `push(s,item-2)`,
- `show_top(s)`, `push(s,item-3)`, `full(s)`, `show_top(s)`, `pop(s,item)`,
- `show_top(s)`, `pop(s,item)`, `show_top(s)`, `empty(s)`, ...

Issue 3: The Retesting of Classes—

I Issue 4: The Retesting of

Classes—II

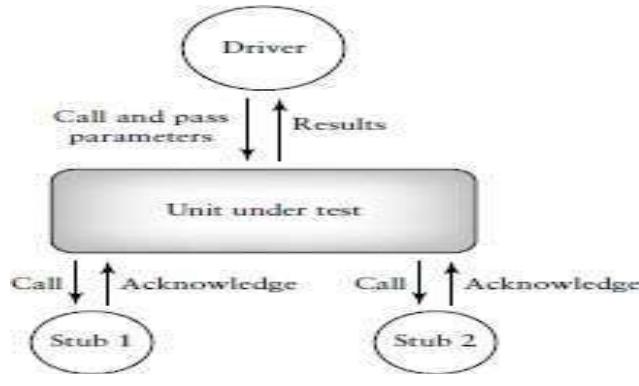


Sample shape class

The Test Harness

- The auxiliary code developed to support testing of units and components is called a test harness. The harness consists of drivers that call the target code

and stubs that represent modules it calls.



The test harness

For example, a driver could have the following options and combinations of options:

- (i) call the target unit;
- (ii) do 1, and pass inputs parameters from a table;
- (iii) do 1, 2, and display parameters;
- (iv) do 1, 2, 3 and display results (output parameters).

The stubs could also exhibit different levels of functionality. For example a stub could:

- (i) display a message that it has been called by the target unit;
- (ii) do 1, and display any input parameters passed from the target unit;
- (iii) do 1, 2, and pass back a result from a table;
- (iv) do 1, 2, 3, and display result from table.

Running the Unit Tests and Recording Results

Unit Test Worksheet			
Unit Name:			
Unit Identifier:			
Tester:			
Date:			
Test case ID	Status (run/not run)	Summary of results	Pass/fail

Summary work sheet for unit test

3.2 Integration Test: Goals

- Integration test for procedural code has two major goals:
 - (i) to detect defects that occur on the interfaces of units;
 - (ii) to assemble the individual units into working subsystems and finally a complete system that is ready for system test.

Integration Strategies for Procedures and Functions

- For conventional procedural/functional-oriented systems there are two major integration strategies—top-down and bottom-up.



Fig 3.6 Simple structure chart for integration test examples

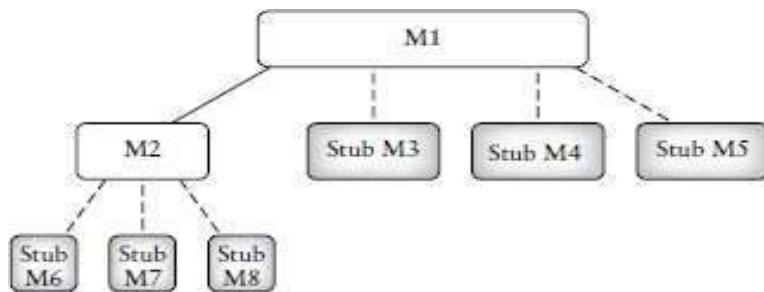
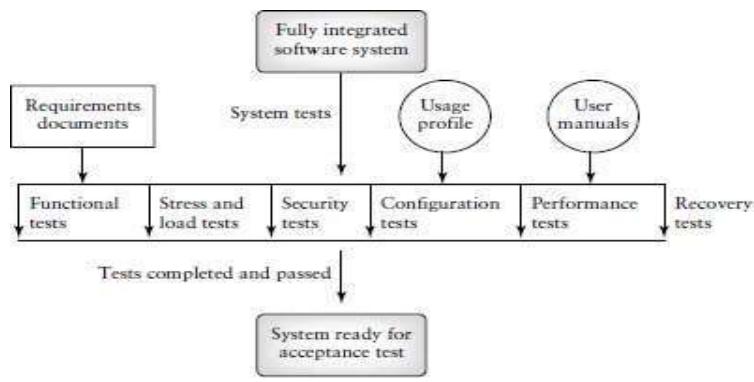


Fig 3.7 Top-down integration of modules M1 and M2

System Test: The Different Types

There are several types of system tests as shown on Figure. The types are as follows:

- Functional testing
- Performance testing
- Stress testing
- Configuration testing
- Security testing
- Recovery testing



Types of system test

Functional Testing

- All types or classes of legal inputs must be accepted by the software.
- All classes of illegal inputs must be rejected (however, the system should remain available).
- All possible classes of system output must exercised and examined.
- All effective system states and state transitions must be exercised and examined.
- All functions must be exercised.

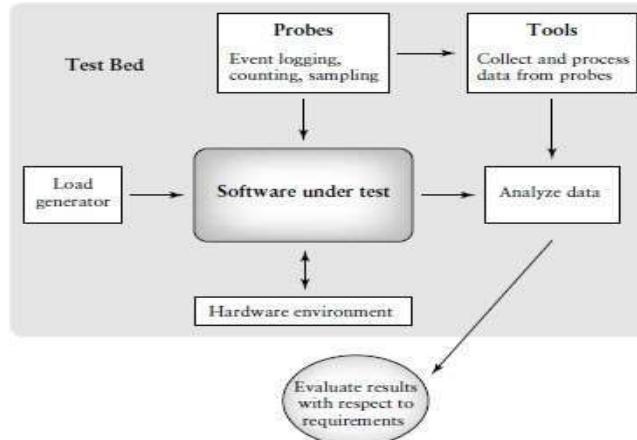
Performance Testing

1. Functional requirements.

Users describe what functions the software should perform. We test for compliance of these requirements at the system level with the functional-based system tests.

2. Quality requirements.

There are nonfunctional in nature but describe quality levels expected for the software. One example of a quality requirement is performance level. The users may have objectives for the software system in terms of memory use, response time, throughput, and delays.



Example of special resources needed for a performance test

Stress Testing

- When a system is tested with a load that causes it to allocate its resources in maximum amounts, this is called stress testing.
- For example, if an operating system is required to handle 10 interrupts / second and the load causes 20 interrupts / second, the system is being stressed. The goal of stress test is to try to break the system; find the circumstances under which it will crash. This is sometimes called “breaking the system.”

Configuration Testing

- Typical software systems interact with hardware devices such as disc drives, tape drives, and printers.

Configuration testing has the following objectives

- Show that all the configuration changing commands and menus work properly.
- Show that all interchangeable devices are really interchangeable, and that they each enter the proper states for the specified conditions.
- Show that the systems' performance level is maintained when devices are interchanged, or when they fail.

Security Testing

- Designing and testing software systems to insure that they are safe and secure is a big issue facing software developers and test specialists.

Computer software and data can be compromised by:

- i)** criminals intent on doing damage, stealing data and information, causing denial of service, invading privacy;
- (ii)** errors on the part of honest developers/maintainers who modify, destroy, or compromise data because of is information, misunderstandings, and/or lack of knowledge.

Damage can be done through various means such as:

- (i) viruses;**
 - (ii) trojan horses;**
 - (iii) trap doors;**
 - (iv) illicit channels.**
 - The effects of security breaches could be extensive and can cause:
- (i) loss of information;**
 - (ii) corruption of information;**
 - (iii) misinformation;**
 - (iv) privacy violations;**
 - (v) denial of service.**

Recovery Testing

- Recovery testing subjects a system to losses of resources in order to determine if it can recover properly from these losses.
- This type of testing is especially important for transaction systems, for example, on-line banking software.

Testers focus on the following areas during recovery testing

- **Restart.** *The current system state and transaction states are discarded.* The most recent checkpoint record is retrieved and the system initialized to the states in the checkpoint record. Testers must insure that all transactions have been

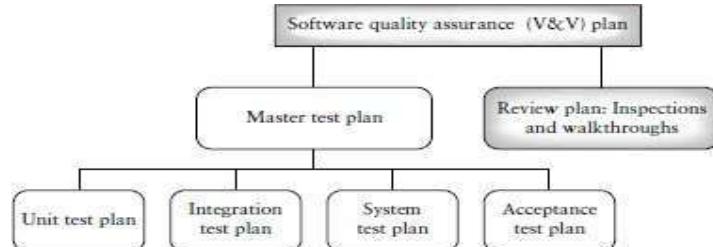
- reconstructed correctly and that all devices are in the proper state. The system should then be able to begin to process new transactions.
- ***Switchover.*** *The ability of the system to switch to a new processor* must be tested. Switchover is the result of a command or a detection of a faulty processor by a monitor.
 - In each of these testing situations all transactions and processes must be carefully examined to detect:
 - (i) **loss of transactions;**
 - (ii) **merging of transactions;**
 - (iii) **incorrect transactions;**
 - (iv) **an unnecessary duplication of a transaction.**

Regression Testing

- Regression testing is not a level of testing, but it is the *retesting of software* that occurs when changes are made to ensure that the new version of the software

Test Management

- A plan is a document that provides a framework or approach for achieving a set of goals.
- Milestones are tangible events that are expected to occur at a certain time in the project's lifetime. Managers use them to determine project status
- Test plans for software projects are very complex and detailed documents. The planner usually includes the following essential high-level items.
- *Overall test objectives*
 - *What to test (scope of the tests).*
 - *Who will test.*
 - *How to test.*
 - *When to test.*
 - *When to stop testing*



Test Plan Components

Test Plan Components	
1. Test plan identifier	16. Approvals
2. Introduction	
3. Items to be tested	
4. Features to be tested	
5. Approach	
6. Pass/fail criteria	
7. Suspension and resumption criteria	
8. Test deliverables	
9. Testing Tasks	
10. Test environment	
11. Responsibilities	
12. Staffing and training needs	
13. Scheduling	
14. Risks and contingencies	
15. Testing costs	

Components of a test plan

4.0 Test Planning				
4.1	Meet with project manager. Discuss test requirements.			
4.2	Meet with SQA group, client group. Discuss quality goals and plans.			
4.3	Identify constraints and risks of testing.			
4.4	Develop goals and objectives for testing. Define scope.			
4.5	Select test team.			
4.6	Decide on training required.			
4.7	Meet with test team to discuss test strategies, test approach, test monitoring, and controlling mechanisms.			
4.8	Develop the test plan document.			
4.9	Develop test plan attachments (test cases, test procedures, test scripts).			
4.10	Assign roles and responsibilities.			
4.11	Meet with SQA, project manager, test team, and clients to review test plan.			

A breakdown of testing planning element

Test Plan Attachments

Requirement identifier	Requirement description	Priority (scale 1-10)	Review status	Test ID
SR-25-13.5	Displays opening screens	8	Yes	TC-25-2 TC-25-5
SR-25-52.2	Checks the validity of user password	9	Yes	TC-25-18 TC-25-23

Example of entries in a requirements traceability matrix

Test Design Specification

- *Test Design Specification Identifier*
- *Features to Be Tested*
- *Approach Refinements*
- *Test Case Identification*
- *Pass/Fail Criteria*

Test Case specification

- *Test Case Specification Identifier*
- *Test Items*
- *Input Specifications*
- *Output Specifications*
- *Special Environmental Needs*
- *Special Procedural Requirements*
- *Intercase Dependencies*

Test procedure specification

- A procedure in general is a sequence of steps required to carry out a specific task.
- *Test Procedure Specification Identifier*
- *Purpose*
- *Specific Requirements*
- *Procedure Steps*

i) setup: to prepare for execution of the procedure;

(ii) start: to begin execution of the procedure;

proceed: to continue the execution of the procedure;

**(iv) measure: to describe how test measurements related to outputs will
be made;**

**(v) shut down: to describe actions needed to suspend the test when unexpected
events occur;**

**(vi) restart: to describe restart points and actions needed to restart the
procedure from these points;**

(vii) stop: to describe actions needed to bring the procedure to an orderly

halt;

(viii) wrap up: to describe actions necessary to restore the environment;

(ix) contingencies: plans for handling anomalous events if they occur

during execution of this procedure.

Locating Test Items: The Test Item Transmittal Report

(i) version/revision number of the item;

(ii) location of the item;

(iii) persons responsible for the item (e.g., the developer);

(iv) references to item documentation and the test plan it is related to;

(v) status of the item;

(vi) approvals—space for signatures of staff who approve the transmittal.

Reporting Test Results

- **Test Log**

- *Test Log Identifier*

- *Description*

- *Activity and Event Entries*

- *Execution description*

- *Procedure results*

- *Environmental information*

- *Anomalous events*

- *Incident report identifiers*

- **Test Incident Report**

1. *Test Incident Report identifier:* to uniquely identify this report.

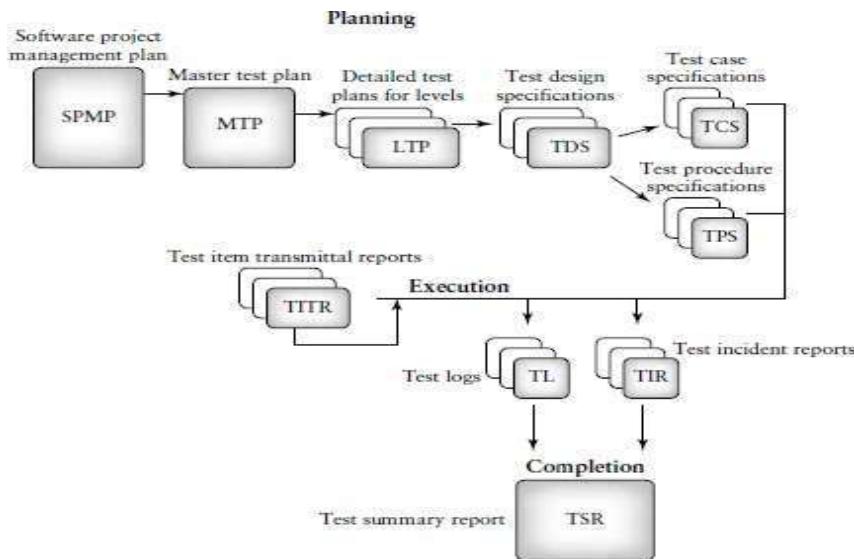
2. *Summary:* to identify the test items involved, the test procedures, test

cases, and test log associated with this report.

- 3. Incident description:** this should describe time and date, testers, observers, environment, inputs, expected outputs, actual outputs, anomalies, procedure step, environment, and attempts to repeat.
- 4. Impact:** what impact will this incident have on the testing effort, the test plans, the test procedures, and the test cases

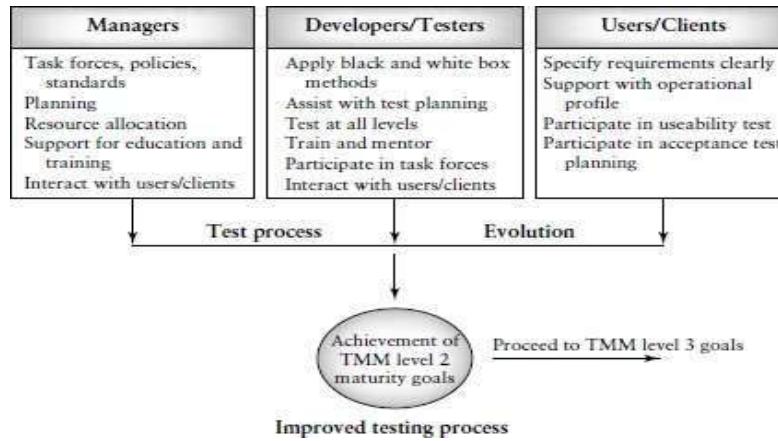
- **Test Summary Report**

1. *Test Summary Report identifier:* to uniquely identify this report.
 2. *Variances:* these are descriptions of any variances of the test items from their original design.
 3. *Comprehensiveness assessment:* the document author discusses the comprehensiveness of the test effort as compared to test objectives
- Summary of results:* the document author summarizes the testing results.
5. *Evaluation:* in this section the author evaluates each test item based on test results.
 6. *Summary of activities:* all testing activities and events are summarized.
 7. *Approvals:* the names of all persons who are needed to approve this document are listed with space for signatures and dates.



Test-related documents as recommended by IEEE[5]

The Role of the Three Critical Groups in Testing Planning and Test Policy Development



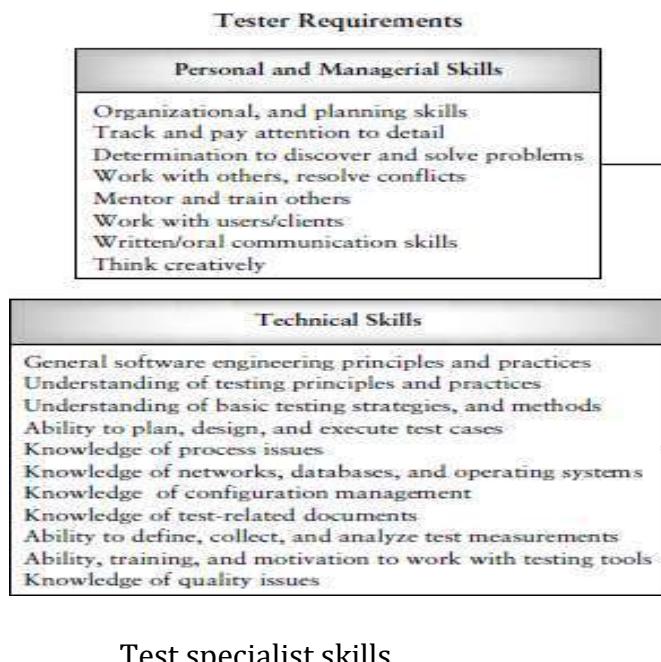
Reaching TMM level 2; summary of critical group roles

Introducing the Test Specialist

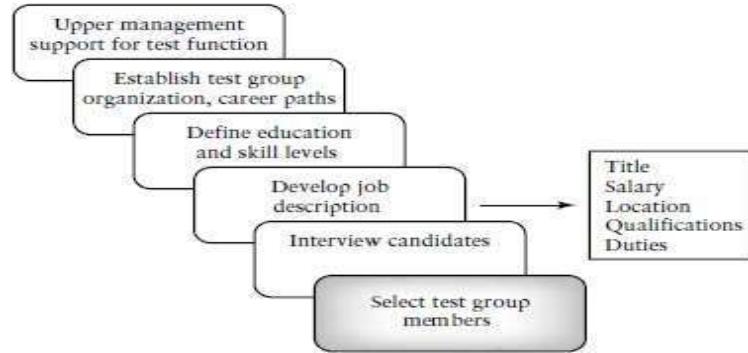
- maintenance and application of test policies;
- development and application of test-related standards;
- participating in requirements, design, and code reviews;
- test planning;
- test design;
- test execution;
- test measurement;
- test monitoring (tasks, schedules, and costs);
- defect tracking, and maintaining the defect repository;
- acquisition of test tools and equipment;
- identifying and applying new testing techniques, tools, and methodologies;
- mentoring and training of new test personnel;
- test reporting.

Skills Needed by a Test Specialist

- organizational, and planning skills;
- the ability to keep track of, and pay attention to, details;
- the determination to discover and solve problems;
- the ability to work with others and be able to resolve conflicts;
- the ability to mentor and train others;
- the ability to work with users and clients;
- strong written and oral communication skills;
- the ability to work in a variety of environments;
- the ability to think creatively



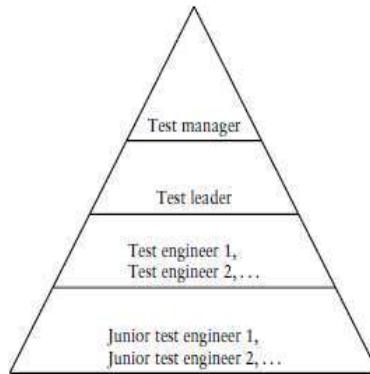
Building a Testing Group



Test specialist skills

The Structure of the Test Group

- maintain testing policy statements;
- plan the testing efforts;
- monitor and track testing efforts so that they are on time and within budget;
- measure process and product attributes;
- provide management with independent product and process quality information;
- design and execute tests with no duplication of effort;
- automate testing;
- participate in reviews to insure quality;
- work with analysts, designers, coders, and clients to ensure quality goals are meet;
- maintain a repository of test-related information;
- give greater visibility to quality issues organization wide;
- support process improvement efforts.



The test team hierarchy

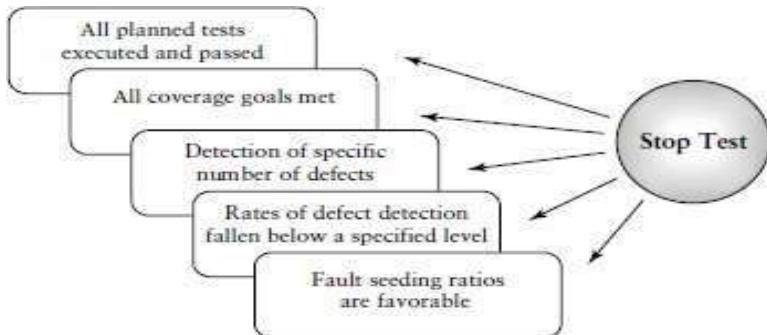
The Technical Training Program

- quality issues;
- measurement identification, collection, and analysis;
- testing techniques and methodologies;
- design techniques;
- tool usage (for all life cycle phases);
- configuration management;
- planning;
- process evaluation and improvement;
- policy development;
- technical review skills;
- software acquisition;
- project management skills;
- business skills
- communication skills.

Test Automation

Status Meetings, Reports, and Control Issues

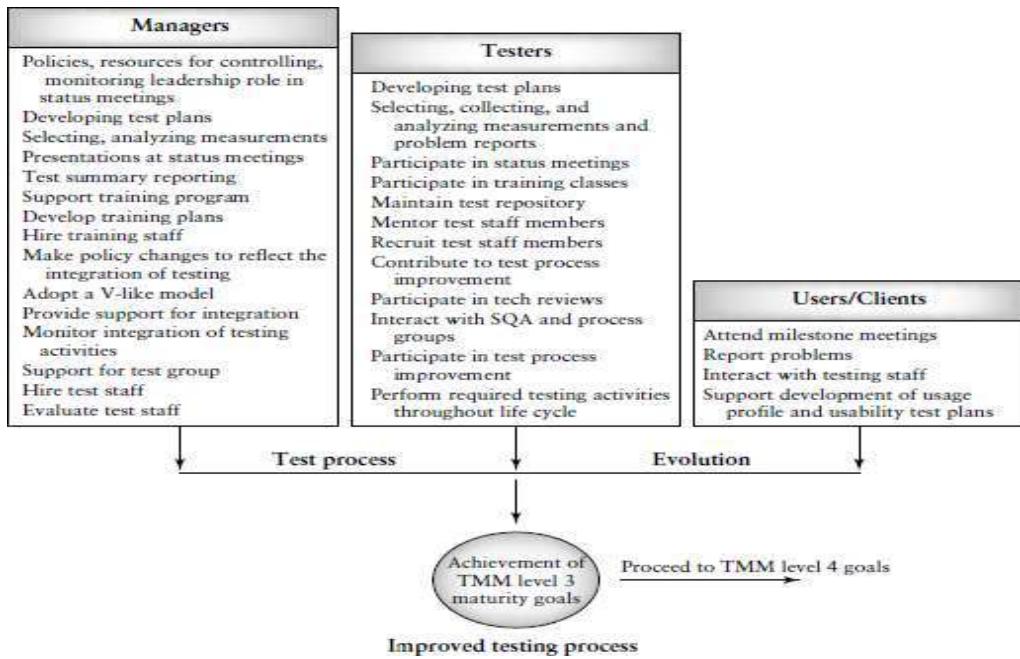
- All the Planned Tests That Were Developed Have Been Executed and Passed.
- All Specified Coverage Goals Have Been Met.
- The Detection of a Specific Number of Defects Has Been Accomplished.
- The Rates of Defect Detection for a Certain Time Period Have Fallen Below a Specified Level.
- Fault Seeding Ratios Are Favorable



Software Configuration Management

- **Identification of the Configuration Items**
- **Change Control**
- **Configuration status reporting**
- **Configuration audits**

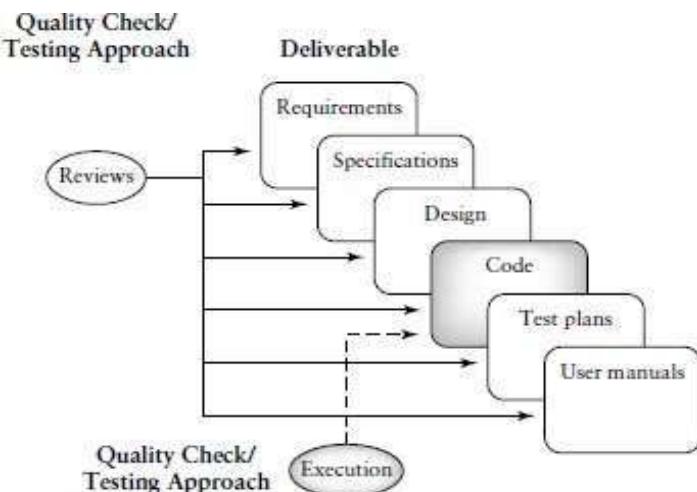
Controlling and Monitoring: Three Critical Views



Contributions of three critical groups to TMM level 3 maturity goals

REVIEWS AS A TESTING ACTIVITY

- i) testing policies with an emphasis on defect detection and quality, and measurements for controlling and monitoring;
- (ii) a test organization with staff devoted to defect detection and quality issues;
- (iii) policies and standards that define requirements, design, test plan, and other documents;
- (iv) organizational culture with a focus on quality products and quality processes.



Role of reviews in testing software deliverables

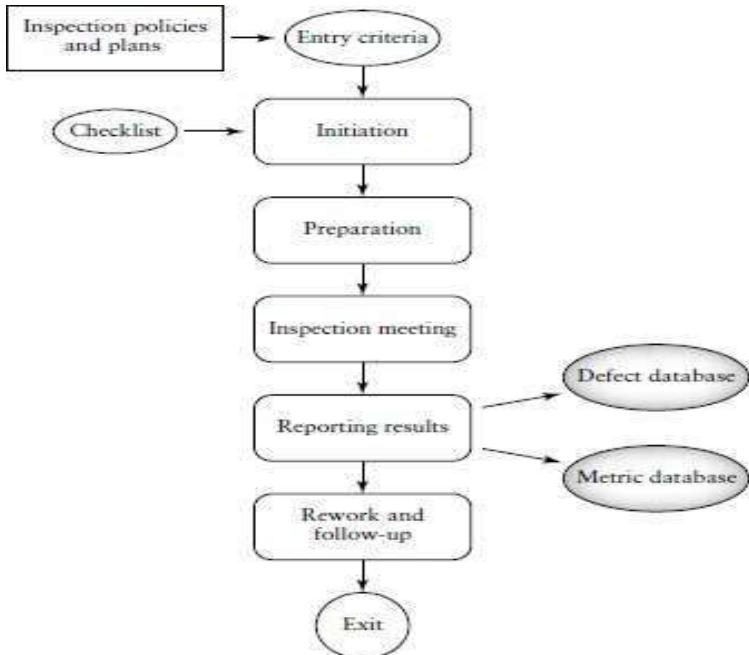
The many benefits of a review program are:

- higher-quality software;
- increased productivity (shorter rework time);
- closer adherence to project schedules (improved process control);
- increased awareness of quality issues;
- teaching tool for junior staff;
- opportunity to identify reusable software artifacts;
- reduced maintenance costs;
- higher customer satisfaction;
- more effective test planning;
- a more professional attitude on the part of the development staff.

Types of Reviews

- verify that a software artifact meets its specification;
 - to detect defects; and
 - check for compliance to standards.
-
- Inspections as a Type of Technical Review
 - Walkthroughs as a Type of

T e c h n i c a l R e v i e w



Steps in the inspection process

The Need for Review Policies

- i) testing policies with an emphasis on defect detection and quality, and measurements for controlling and monitoring;
- (ii) a test organization with staff devoted to defect detection and quality issues;
- (iii) policies and standards that define requirements, design, test plan, and other documents;
- (iv) organizational culture with a focus on quality products and quality processes.

EVALUATING SOFTWARE QUALITY: A QUANTITATIVE APPROACH

1. Quality relates to the degree to which a system, system component, or process meets specified requirements.

2. Quality relates to the degree to which a system, system component, or process, meets customer, or user, needs or expectations.

The software quality assurance (SQA) group is a team of people with the necessary training and skills to ensure that all necessary actions are taken during the development process so that the resulting software conforms to established technical requirements.

Quality Costs

- quality planning;
- test and laboratory equipment;
- training;
- formal technical reviews.

What Is Quality Control?

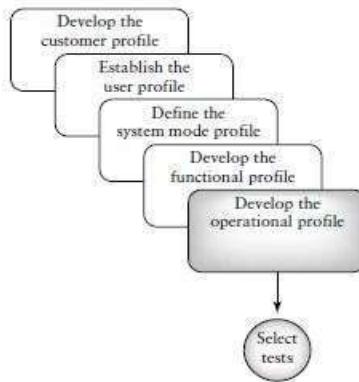
- Quality control consists of the procedures and practices employed to ensure that a work product or deliverable conforms to standards or requirements.
- Quality control is the set of activities designed to evaluate the quality of developed or manufactured products.

Quality control includes:

- policies and standards;
- review and audit procedures;
- a training program;
- dedicated, trained, and motivated staff (for testing and quality assurance);
- a measurement program;
- a planning process (for test and quality assurance);
- effective testing techniques, statistical techniques, and tools;
- process monitoring and controlling systems;
- a test process assessment system (TMM-AM);
- a configuration management system.

The Role of Operational Profiles and Usage Models in Quality Control

1. An operational profile is a quantitative characterization of how a software system will be used in its intended environment .
2. An operational profile is a specification of classes of inputs and the probability of their occurrence.



Steps to develop an operations profile

Software Reliability

- Software reliability is the ability of a system or component to perform its required functions under stated conditions for a specified period of time .
- Software reliability is the probability that a software system will operate without failure under given conditions for a given time interval.
- Availability is the degree to which a system or component is operational and accessible when required for use .
- Availability is the probability that a software system will be available for use.
- Trustworthiness is the probability that there are no errors in the software that will cause the system to fail catastrophically.

Measurements for Software Reliability

- $MTBF = MTTF + MTTR$
- mean time to failure, MTTF

- mean time to repair, MTTR
- mean time between failure, MTBF
- Reliability (R)

$$R = \text{MTBF} / (1 + \text{MTBF})$$

- Availability (A)

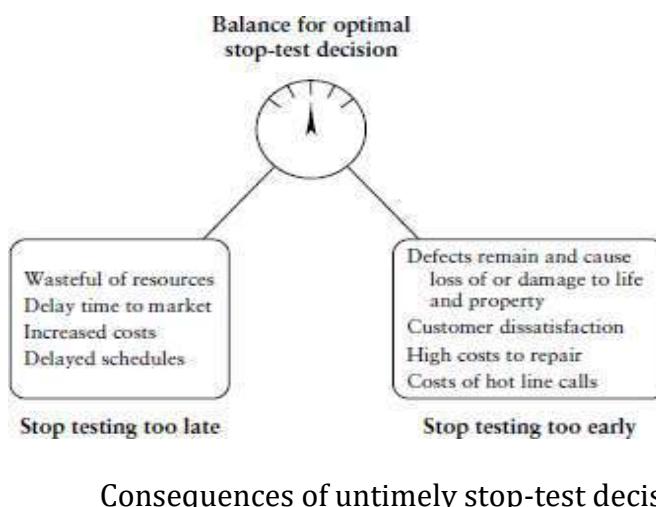
$$A = \text{MTBF} / (\text{MTBF} + \text{MTTR})$$

- The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changing environment.
- maintainability

$$(M) M = 1 / (1 + \text{MTTR})$$

Reliability, Quality Control, and Stop-Test Decisions

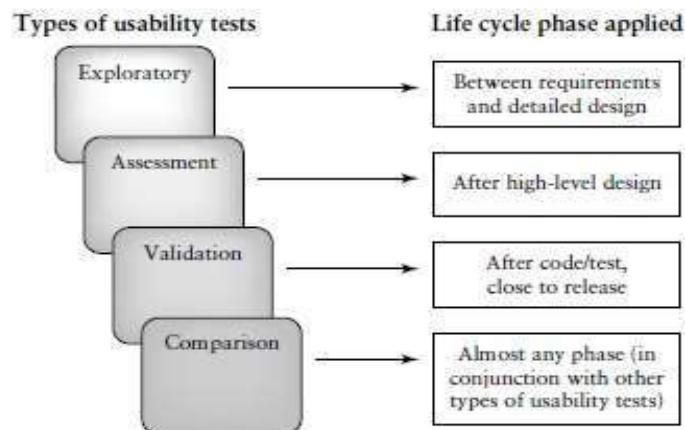
- (i) setting reliability goals in the requirements document (a reliability specification is the result);
- (ii) providing adequate time and resources in the test plan to allow for developing/modifying the usage profile, running the tests, and collecting, cataloging, and analyzing the reliability data;
- (iii) developing a usage or operational profile that accurately models usage patterns;



Usability Testing, and Quality Control

- Usability is a quality factor that is related to the effort needed to learn, operate, prepare input, and interpret the output of a computer program.
- Usability is a complex quality factor and can be decomposed according to IEEE standards into the sub factors
- Understandability: The amount of effort required to understand the software.
- Ease of learning: The degree to which user effort required to understand the software is minimized.
- Operability: The degree to which the operation of the software matches the purpose, environment, and physiological characteristics of users; this includes ergonomic factors such as color, shape, sound, font size, etc.
- Communicativeness: The degree to which the software is designed in accordance with the psychological characteristics of the users.

An Approach to Usability Testing



Types of usability tests.

Usability Testing: Resource Requirements

1. A usability testing laboratory
2. Trained personnel

3. Usability test planning.

Usability Tests and Measurements

- (i) open an existing document;
- (ii) add text to the document;
- (iii) modify the old text;
- (iv) change the margins in selected sections;
- (v) change the font size in selected sections;
- (vi) print the document;
- (vii) save the document.



SELENIUM

web browser automation

tutorialspoint
SIMPLY EASY LEARNING



tutorialspoint
SIMPLY EASY LEARNING

www.tutorialspoint.com



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

About the Tutorial

Selenium is an open-source tool that is used for test automation. It is licensed under Apache License 2.0. Selenium is a suite of tools that helps in automating only web applications.

This tutorial will give you an in-depth understanding of Selenium and its related tools and their usage.

Audience

This tutorial is designed for software testing professionals who would like to learn the basics of Selenium through practical examples. The tutorial contains enough ingredients to get you started with Selenium from where you can take yourself to higher levels of expertise.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of Java or any other object-oriented programming language. In addition, you should be well-versed with the fundamentals of testing concepts.

Copyright & Disclaimer

© Copyright 2014 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial.....	i
Audience	i
Prerequisites	i
Copyright & Disclaimer.....	i
Table of Contents.....	ii
 1. OVERVIEW	1
Introduction	1
Advantages of Selenium.....	2
Disadvantages of Selenium	3
 2. SELENIUM - IDE	4
Selenium – IDE	4
Download Selenium IDE	5
Features of Selenium IDE	7
Creating Selenium IDE Tests	8
Script Debugging	13
Inserting Verification Points	17
Pattern Matching	20
Selenium User Extensions	22
Different Browser Execution	25
 3. ENVIRONMENT SETUP	28
Download and Install Java.....	28
Download and Configure Eclipse	34
Configure FireBug and FirePath	36
Configure Selenium RC.....	40

Configure Selenium WebDriver	42
4. SELENIUM RC	44
What is Selenium RC?.....	44
Selenium RC Architecture.....	44
RC – Scripting	45
5. SELENESE COMMANDS.....	54
Actions	54
Accessors	58
Assertions	61
6. WEBDRIVER.....	65
Architecture	65
Selenium RC Vs WebDriver.....	66
Scripting using WebDriver	66
Most Used Commands	74
7. LOCATORS	76
Locators Usage	77
8. INTERACTIONS	84
User Interactions.....	84
Text Box Interaction.....	84
Radio Button Interaction.....	87
Check Box Interaction	89
Dropdown Interaction.....	91
Synchronization	93
Drag & Drop	95
Keyboard Actions	97

Selenium

Mouse Actions	97
Multi Select Action.....	98
Find All Links	101
9. TEST DESIGN TECHNIQUES	103
Page Object Model.....	103
POM Flow Diagram	103
Data Driven using Excel	107
Parameterization	112
Log4j Logging.....	118
Exception Handling	127
Multi Browser Testing	128
Capture Screenshots	135
Capturing Videos.....	141
10. TESTNG	148
What is TestNG?.....	148
Installing TestNG for Eclipse	148
Annotations in TestNG	152
TestNG-Eclipse Setup	155
First Test in TestNG	162
11. SELENIUM GRID	166
Architecture	167
Working with Grid	167
Configuring the Hub	168
Configuring the Nodes.....	169
Develop the Script and Prepare the XML File	174
Test Execution.....	181

1. OVERVIEW

Introduction

Selenium is an open-source and a portable automated software testing tool for testing web applications. It has capabilities to operate across different browsers and operating systems. Selenium is not just a single tool but a set of tools that helps testers to automate web-based applications more efficiently.

Let us now understand each one of the tools available in the Selenium suite and their usage.

Tool	Description
Selenium IDE	Selenium Integrated Development Environment (IDE) is a Firefox plugin that lets testers to record their actions as they follow the workflow that they need to test.
Selenium RC	Selenium Remote Control (RC) was the flagship testing framework that allowed more than simple browser actions and linear execution. It makes use of the full power of programming languages such as Java, C#, PHP, Python, Ruby, and PERL to create more complex tests.
Selenium WebDriver	Selenium WebDriver is the successor to Selenium RC which sends commands directly to the browser and retrieves results.
Selenium Grid	Selenium Grid is a tool used to run parallel tests across different machines and different browsers simultaneously which results in minimized execution time.

Advantages of Selenium

QTP and Selenium are the most used tools in the market for software automation testing. Hence it makes sense to compare the pros of Selenium over QTP.

Selenium	QTP
Selenium is an open-source tool.	QTP is a commercial tool and there is a cost involved in each one of the licenses.
Can be extended for various technologies that expose DOM.	Limited add-ons and needs add-ons for each one of the technologies.
Has capabilities to execute scripts across different browsers.	Can run tests in specific versions of Firefox, IE, and Chrome.
Can execute scripts on various operating systems.	Works only with Windows.
Supports mobile devices.	Supports mobile devices with the help of third-party tools.
Executes tests within the browser, so focus is NOT required while script execution is in progress.	Needs Focus during script execution, as the tool acts on the browser (mimics user actions).
Can execute tests in parallel with the use of Selenium Grids.	QTP cannot execute tests in parallel, however integrating QTP with QC allows testers to execute in parallel. QC is also a commercial tool.

Disadvantages of Selenium

Let us now discuss the pitfalls of Selenium over QTP.

Selenium	QTP
Supports only web-based applications.	Can test both web and desktop applications.
No feature such as Object Repository/Recovery Scenario	QTP has built-in object repositories and recovery scenarios.
No IDE, so the script development won't be as fast as QTP.	More intuitive IDE; automation can be achieved faster.
Cannot access controls within the browser.	Can access controls within the browser such as favorites bar, backward, and forward buttons.
No default test report generation.	Default test result generation within the tool.
For parameterization, users has to rely on the programming language.	Parameterization is built-in and easy to implement.

2. SELENIUM - IDE

Selenium –IDE

The Selenium-IDE (Integrated Development Environment) is an easy-to-use Firefox plug-in to develop Selenium test cases. It provides a Graphical User Interface for recording user actions using Firefox which is used to learn and use Selenium, but it can only be used with Firefox browser as other browsers are not supported.

However, the recorded scripts can be converted into various programming languages supported by Selenium and the scripts can be executed on other browsers as well.

The following table lists the sections that we are going to cover in this chapter. .

Title	Description
Download Selenium IDE	This section deals with how to download and configure Selenium IDE.
Selenium IDE Features	This section deals with the features available in Selenium IDE.
Creating Selenium IDE Tests	This section deals with how to create IDE tests using recording feature.
Selenium IDE Script Debugging	This section deals with debugging the Selenium IDE script.
Inserting Verification Points	This section describes how to insert verification points in Selenium IDE.

Selenium Pattern Matching	This section deals with how to work with regular expressions using IDE.
Selenium User Extensions	The Java script that allows users to customize or add new functionality.
Different Browser Execution	This section deals with how to execute Selenium IDE scripts on different browsers.

Download Selenium IDE

Step 1 : Launch Firefox and navigate to the following URL –

<http://seleniumhq.org/download/>.

Under the Selenium IDE section, click on the link that shows the current version number as shown below.



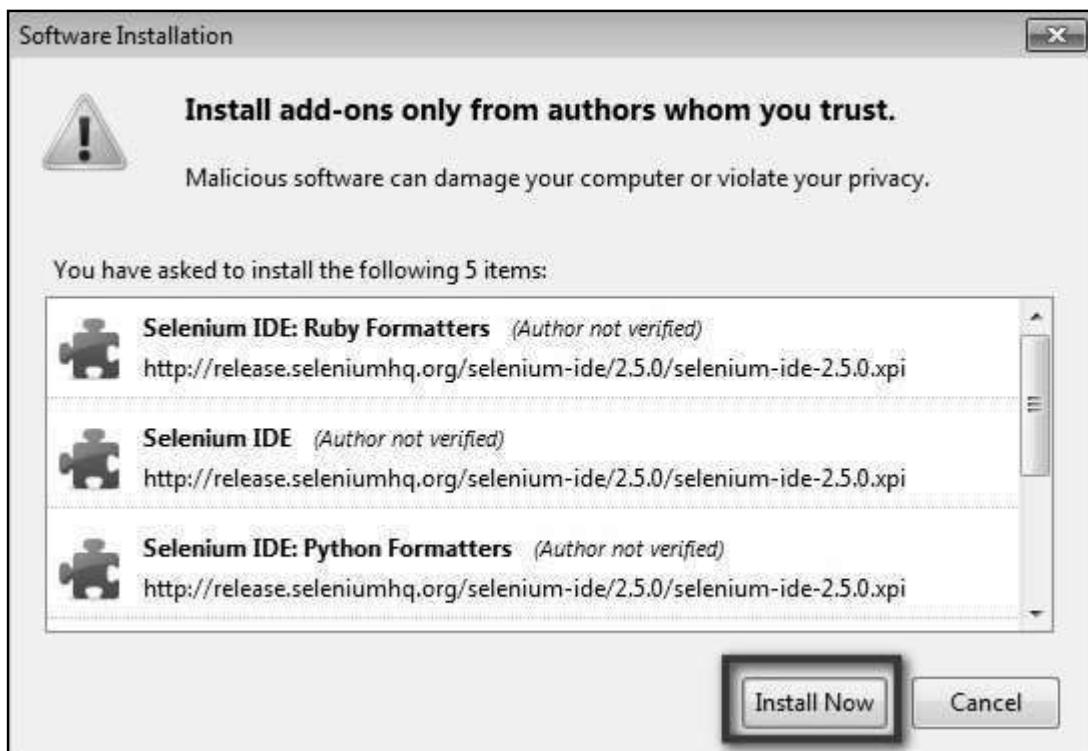
The screenshot shows the SeleniumHQ website with the following details:

- Header:** edit this page, search selenium: [text input], Go, Projects, Download, Documentation, Support, About.
- Left Sidebar:** Selenium Downloads, Latest Releases, Previous Releases, Source Code, Maven Information.
- Downloads Section:** Subtitle: Downloads. Description: Below is where you can find the latest releases of all the Selenium components. You can also find a list of previous releases, source code, and additional information for Maven users (Maven is a popular Java build tool). Sub-section: Selenium IDE. Description: Selenium IDE is a Firefox plugin which records and plays back user interactions with the browser. Use this to either create simple scripts or assist in exploratory testing. It can also export Remote Control or WebDriver scripts, though they tend to be somewhat brittle and should be overhauled into some sort of Page Object-y structure for any kind of resiliency.
- Buttons:** Download latest released version 2.5.0 released on 01/Jan/2014 or view the [Release Notes](#) and then [install some plugins](#). Download version under development unreleased (currently disabled).
- Donation:** Donate with PayPal, with payment method icons (MasterCard, Visa, American Express, Discover, Diners Club, Bank).

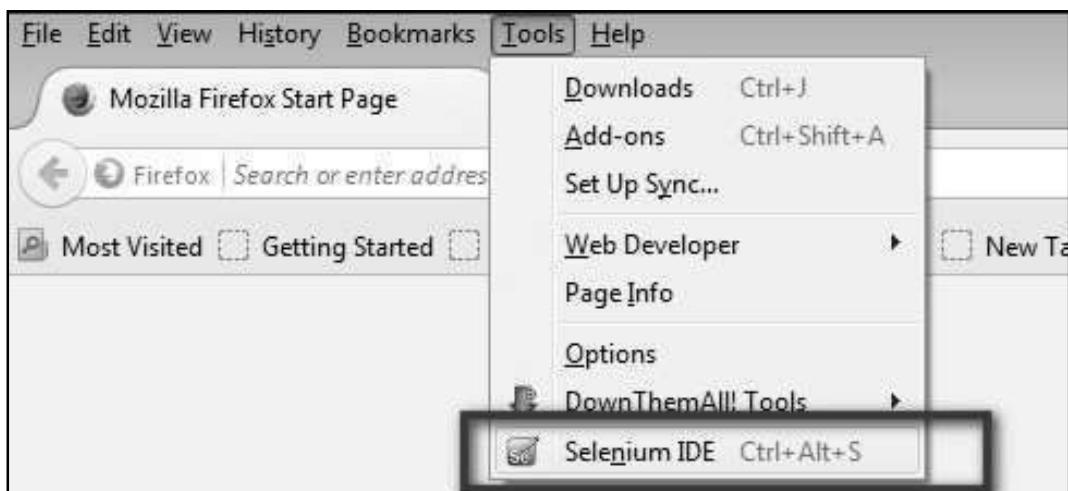
Step 2 : Firefox add-ons notifier pops up with allow and disallow options. User has to allow the installation.



Step 3 : The add-ons installer warns the user about untrusted add-ons. Click 'Install Now'.



Step 4 : The Selenium IDE can now be accessed by navigating to Tools >> Selenium IDE.

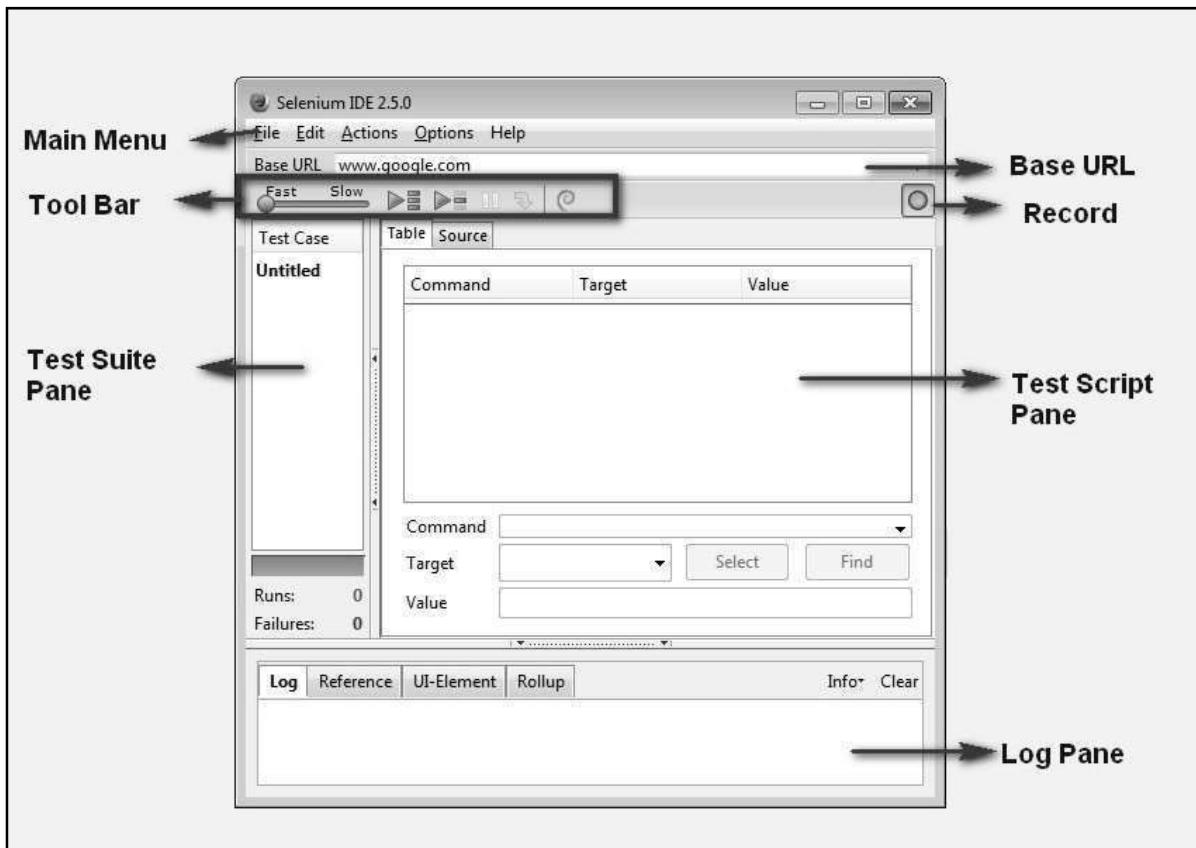


Step 5 : The Selenium IDE can also be accessed directly from the quick access menu bar as shown below.



Features of Selenium IDE

The following image shows the features of Selenium IDE with the help of a simple tooltip.



The features of the record tool bar are explained below.

Control	Control Name	Description
	Speed Control	This helps in controlling the speed of the test case runs.
	Run All	Executes the entire test suite that contains multiple test cases.
	Run	Executes the currently selected test.
	Pause/Resume	Allows user to pause or resume the script execution. Enabled only during the execution.
	Step	Helps user to debug the test by executing only one step of a test case at a time.
	Test Runner Mode	Allows user to execute the test case in a browser loaded with the Selenium-Core. It is an obsolete functionality that is likely to be deprecated.
	Apply Rollup Rules	This feature allows repetitive sequences of Selenium commands to be grouped into a single action.
	Record	This feature helps user to Records the user's browser actions.

Creating Selenium IDE Tests

The following steps are involved in creating Selenium tests using IDE:

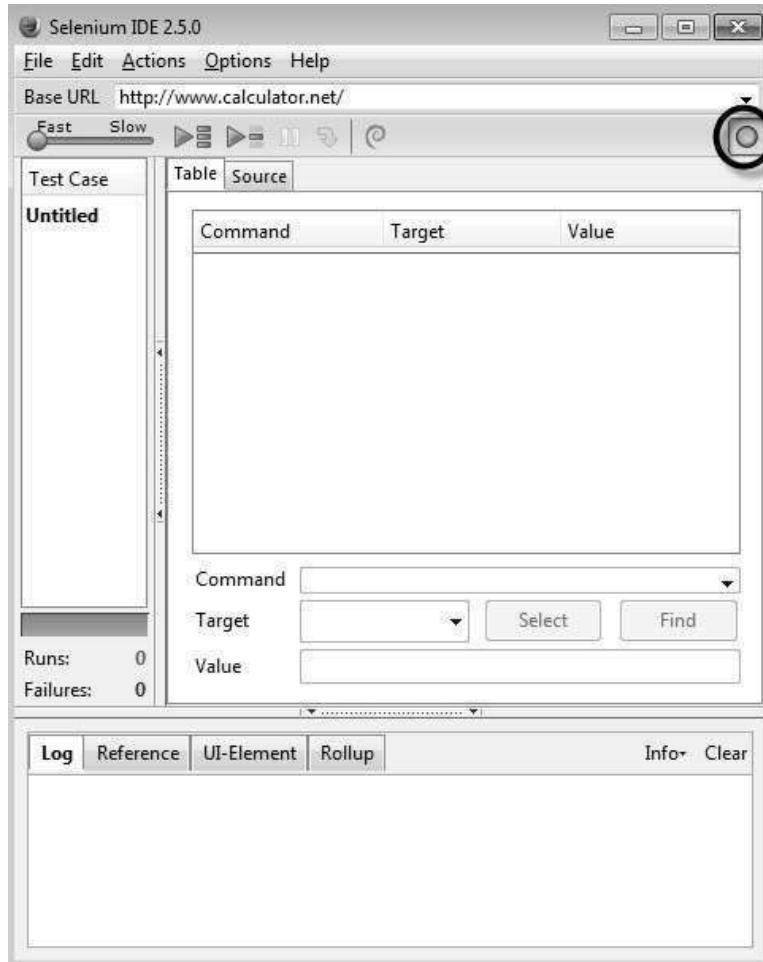
- Recording and adding commands in a test
- Saving the recorded test
- Saving the test suite
- Executing the recorded test

Recording and Adding Commands in a Test

We will use www.ncalculators.com to demonstrate the features of Selenium.

Step 1 : Launch the Firefox browser and navigate to the website – <http://www.ncalculators.com/>

Step 2 : Open Selenium IDE from the Tools menu and press the record button that is on the top-right corner.



Step 3 : Navigate to "Math Calculator" >> "Percent Calculator >> enter "10" as number1 and 50 as number2 and click "calculate".

Step 4 : User can then insert a checkpoint by right clicking on the webelement and select "Show all available commands" >> select "assert text css=b 5"

Step 5 : The recorded script is generated and the script is displayed as shown below.

Command	Target	Value
open	/	
clickAndWait	xpath=/a[contains(text(),'Math')][2]	
clickAndWait	link=Percent Calculator	
type	id=cpar1	10
type	id=cpar2	50
clickAndWait	css=input[type="image"]	
assertText	css=b	5

Saving the Recorded Test

Step 1 : Save the Test Case by navigating to "File" >> "Save Test" and save the file in the location of your choice. The file is saved as .HTML as default.

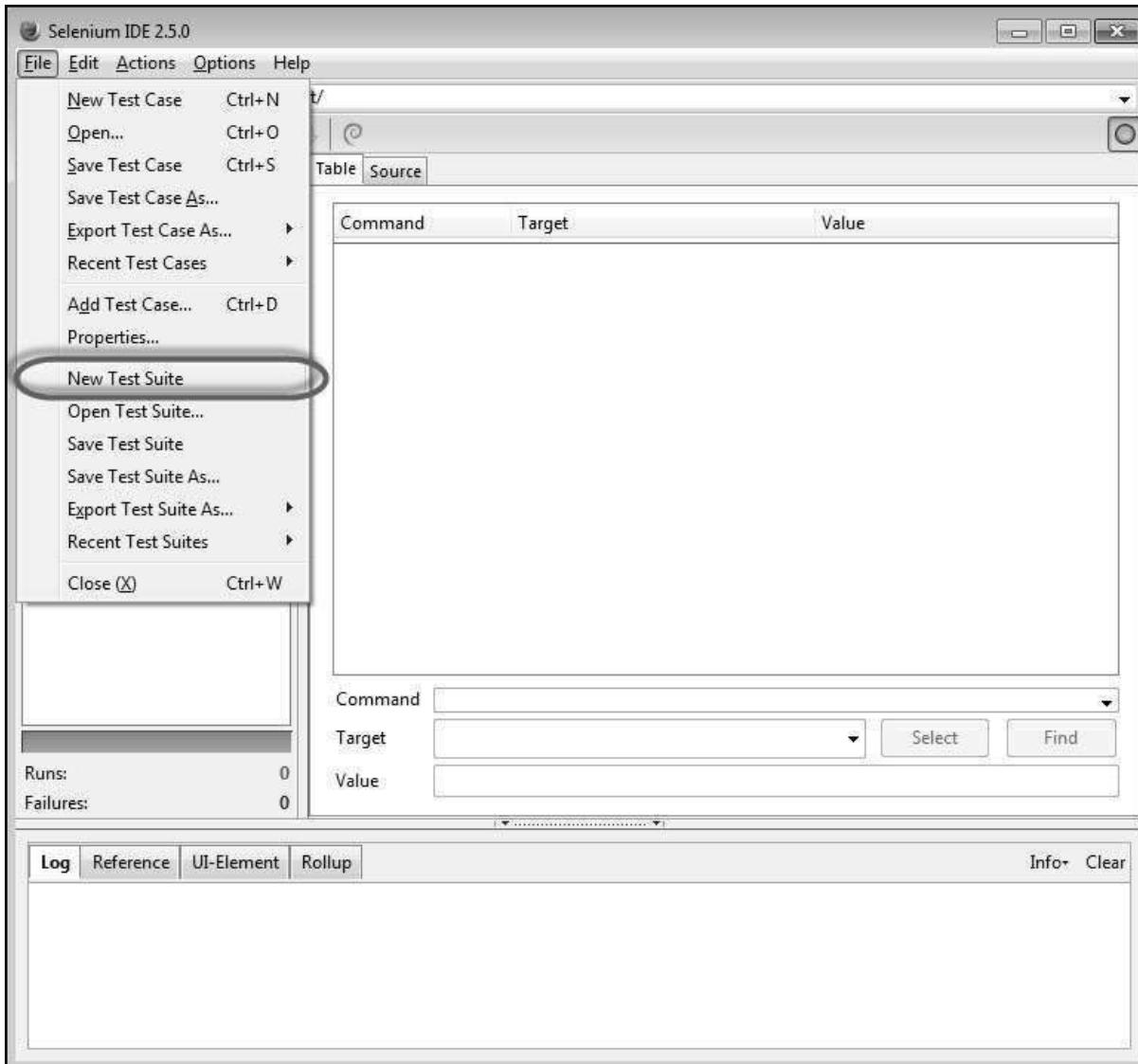
The test can also be saved with an extension htm, shtml, and xhtml.

Command	Target	Value
open	/	
clickAndWait	xpath=/a[contains(text(),'Math')][2]	
clickAndWait	link=Percent Calculator	
type	id=cpar1	10
type	id=cpar2	50
clickAndWait	css=input[type="image"]	
assertText	css=b	5

Saving the Test Suite

A test suite is a collection of tests that can be executed as a single entity.

Step 1 : Create a test suite by navigating to "File" >> "New Test Suite" as shown below.



Step 2 : The tests can be recorded one by one by choosing the option "New Test Case" from the "File" Menu.

Step 3 : The individual tests are saved with a name along with saving a "Test Suite".



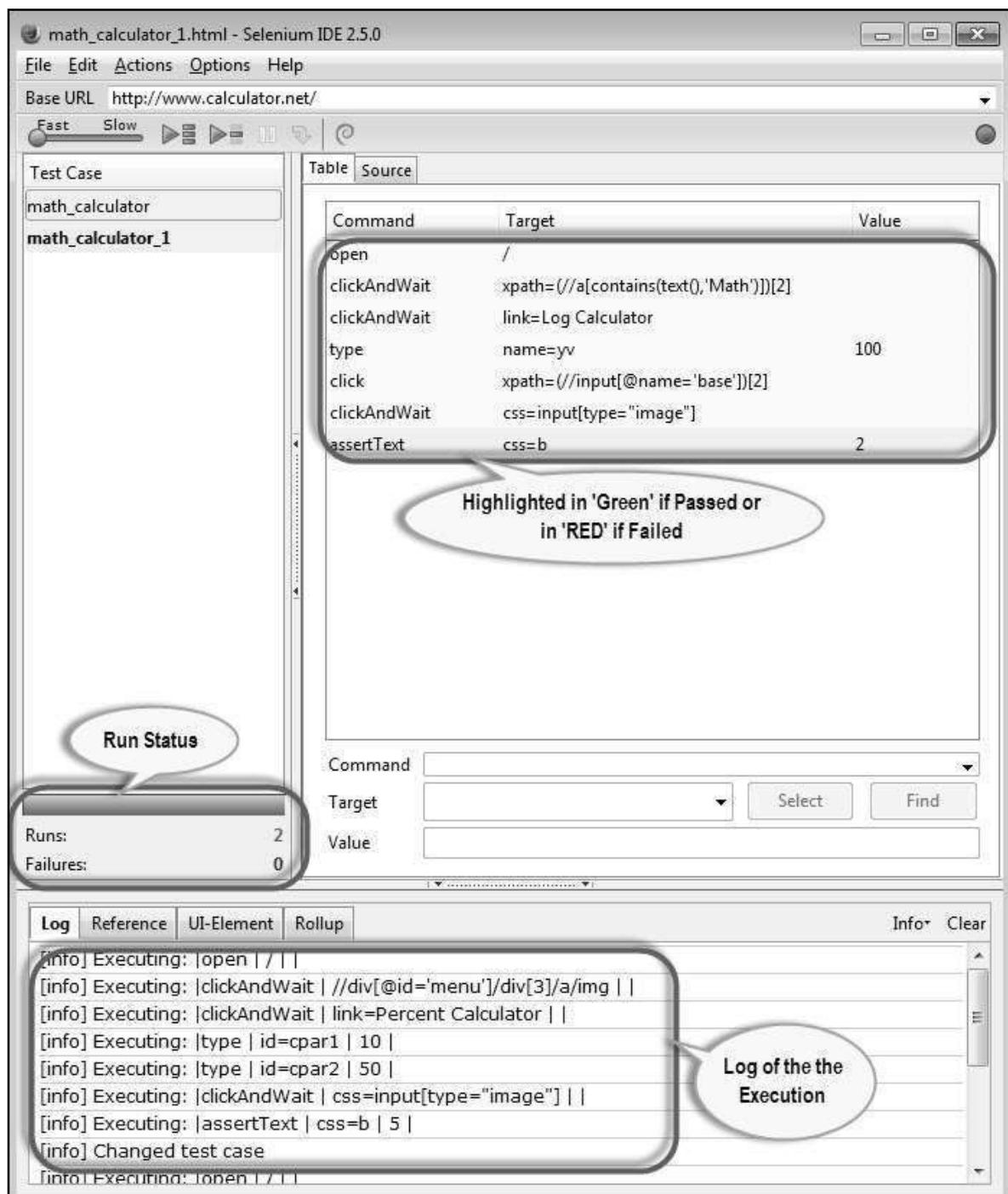
Executing the Recorded Test

The recorded scripts can then be executed either by clicking "Play entire suite" or "Play current test" button in the toolbar.

Step 1 : The Run status can be seen in the status pane that displays the number of tests passed and failed.

Step 2 : Once a step is executed, the user can see the result in the "Log" Pane.

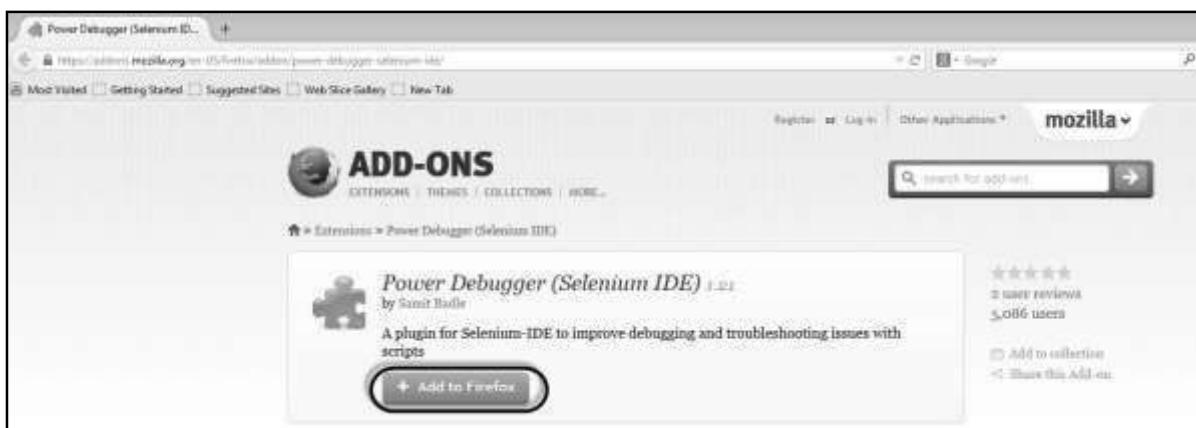
Step 3 : After executing each step, the background of the test step turns "Green" if passed and "Red" if failed as shown below.



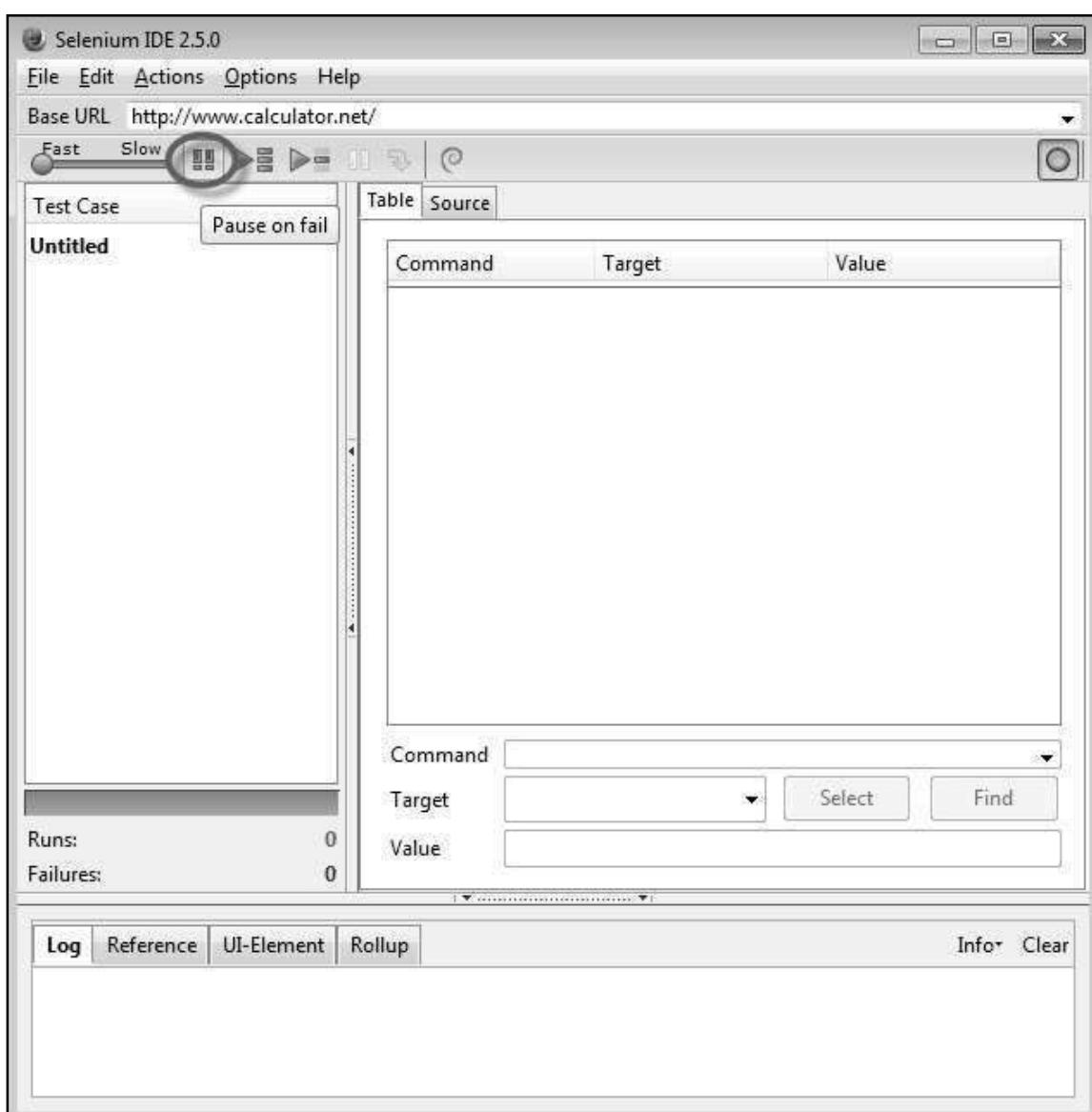
Script Debugging

Debugging is the process of finding and fixing errors in the test script. It is a common step in any script development. To make the process more robust, we can use a plugin "Power Debugger" for Selenium IDE.

Step 1 : To install Power Debugger for Selenium IDE, navigate to <https://addons.mozilla.org/en-US/firefox/addon/power-debugger-selenium-ide/> and click "Add to Firefox" as shown below.



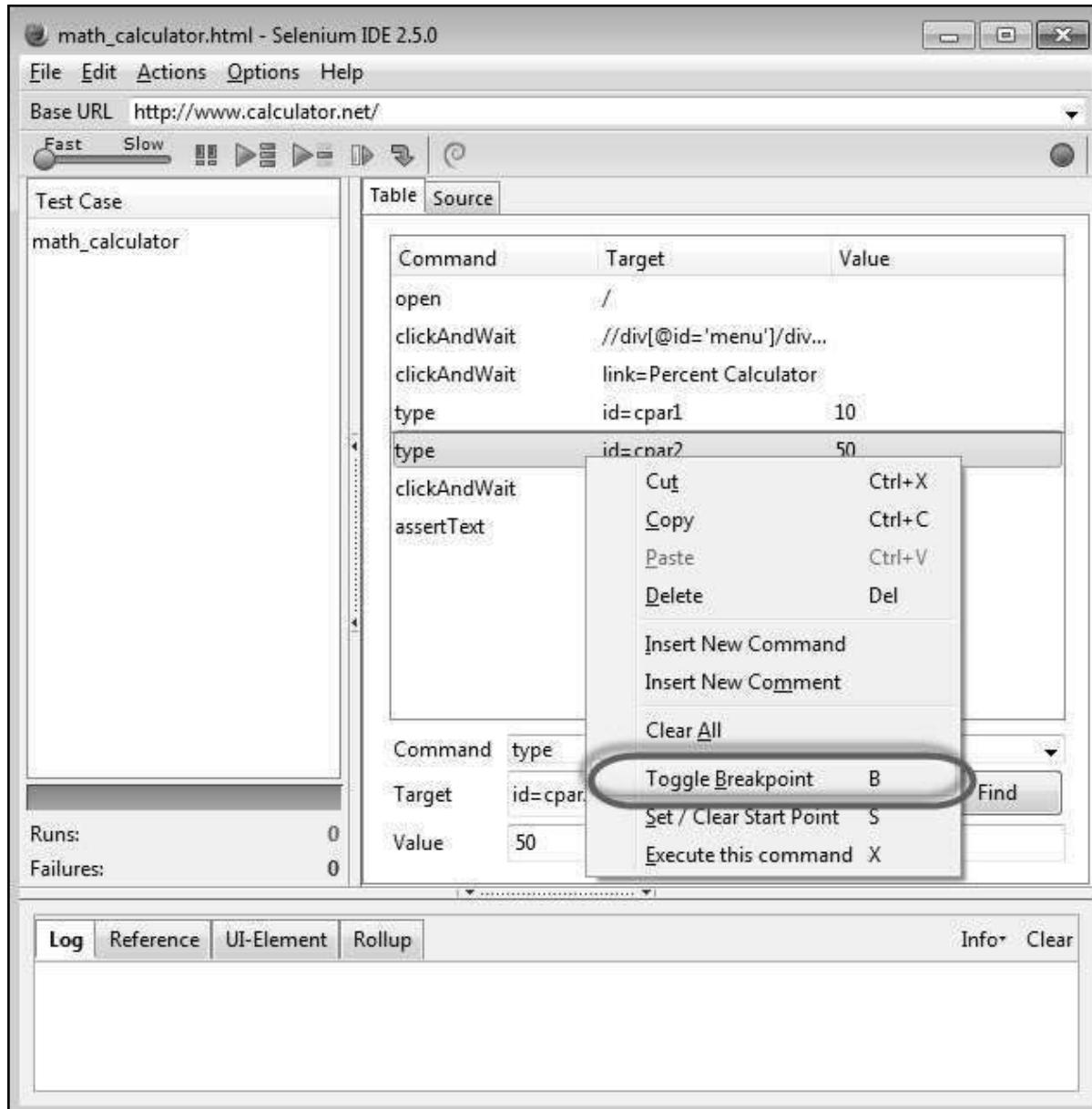
Step 2 : Now launch 'Selenium IDE' and you will notice a new icon, "Pause on Fail" on the recording toolbar as shown below. Click it to turn it ON. Upon clicking again, it would be turned "OFF".



Step 3 : Users can turn "pause on fail" on or off any time even when the test is running.

Step 4 : Once the test case pauses due to a failed step, you can use the resume/step buttons to continue the test execution. The execution will **NOT** be paused if the failure is on the last command of any test case.

Step 5 : We can also use breakpoints to understand what exactly happens during the step. To insert a breakpoint on a particular step, "Right Click" and select "Toggle Breakpoint" from the context-sensitive menu.



Step 6 : Upon inserting the breakpoint, the particular step is displayed with a pause icon as shown below.

The screenshot shows the Selenium IDE interface with the following details:

- Test Case:** math_calculator
- Table View (Main Area):**

Command	Target	Value
open	/	
clickAndWait	//div[@id='menu']/div...	
clickAndWait	link=Percent Calculator	
type	id=cpar1	10
type	id=cpar2	50
clickAndWait	css=input[type="imag...]	
assertText	css=b	5
- Source View (Bottom):**

```
Command: open
Target: /
Value: 
```
- Tooltip (Bottom):**

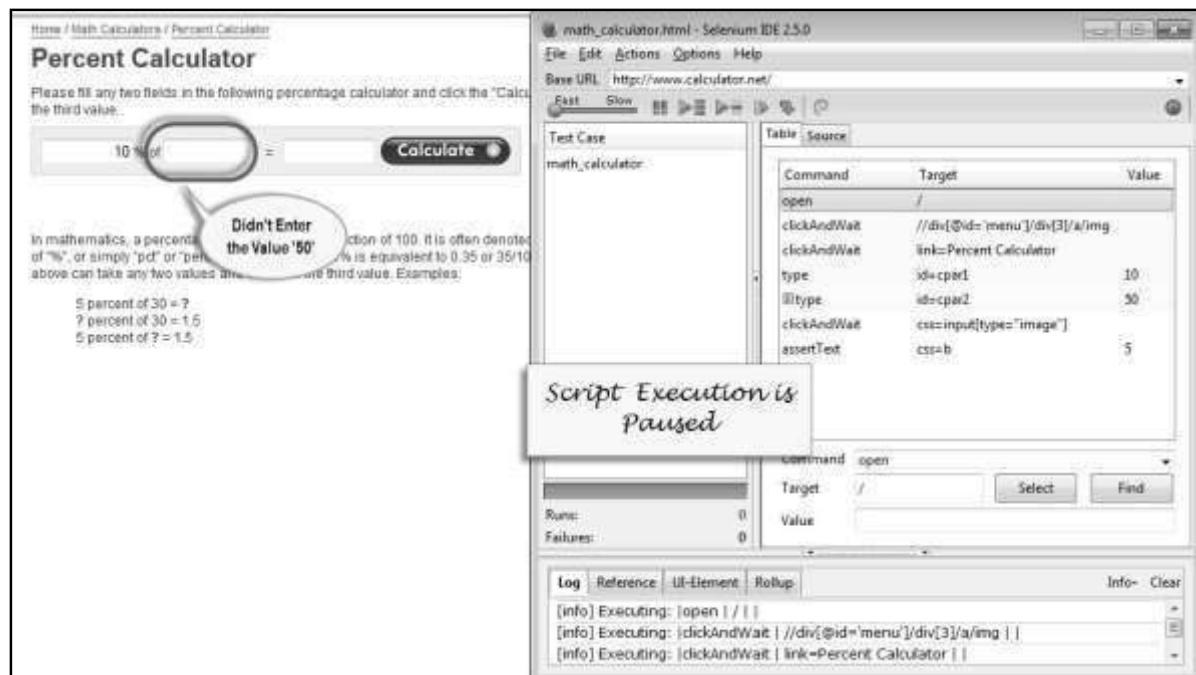
open(url)

Arguments:

 - url - the URL to open; may be relative or absolute

Opens an URL in the test frame. This accepts both relative and absolute URLs. The "open" command waits for the

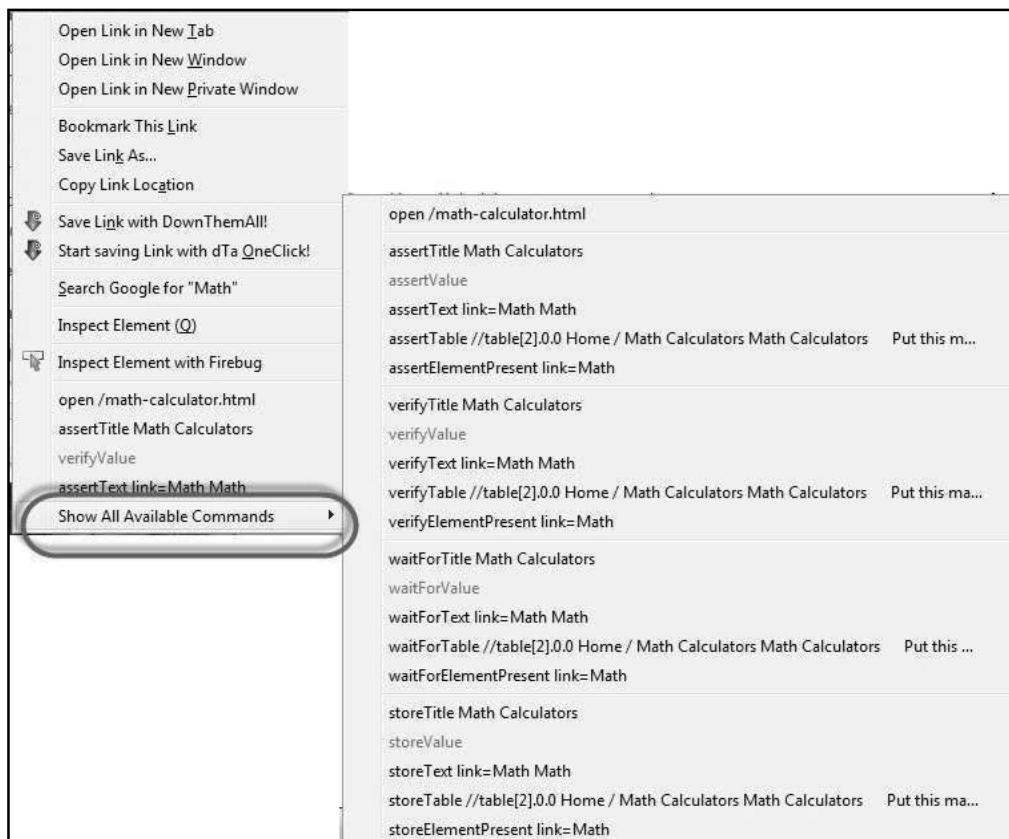
Step 7 : When we execute the script, the script execution is paused where the breakpoint is inserted. This will help the user to evaluate the value/presence of an element when the execution is in progress.



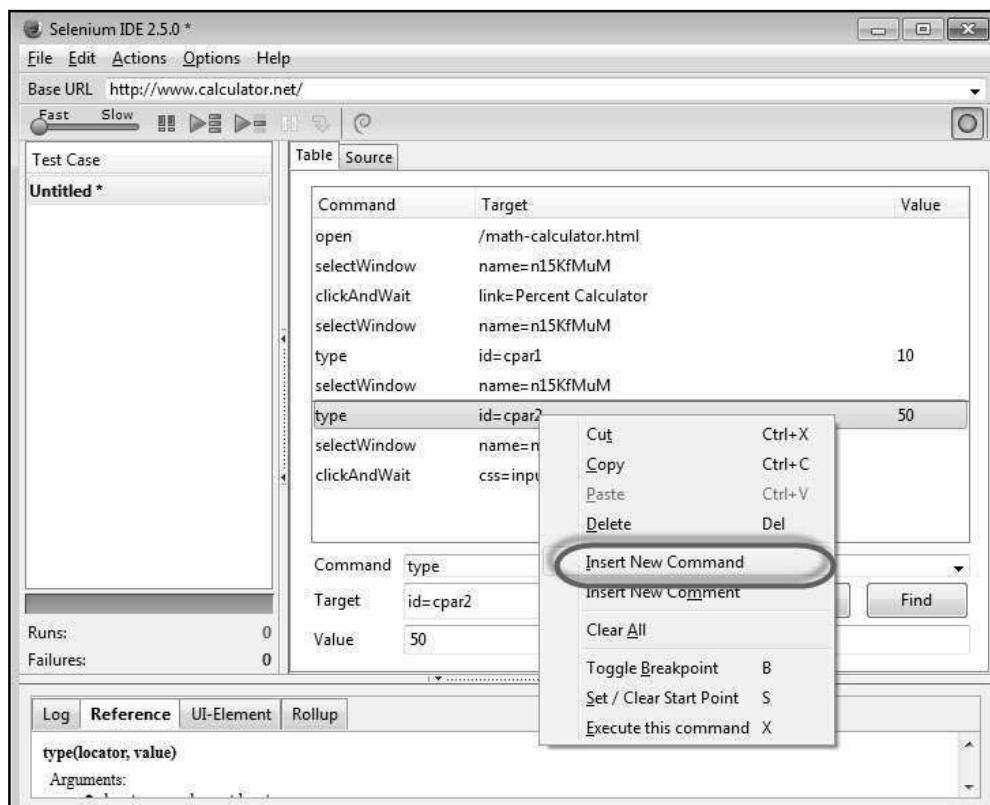
Inserting Verification Points

The test cases that we develop also need to check the properties of a web page. It requires assert and verify commands. There are two ways to insert verification points into the script.

To insert a verification point in recording mode, "Right click" on the element and choose "Show all Available Commands" as shown below.



We can also insert a command by performing a "Right-Click" and choosing "Insert New Command".



After inserting a new command, click 'Command' dropdown and select appropriate verification point from the available list of commands as shown below.

The screenshot shows the Selenium IDE interface with a test case titled 'Untitled *'. The test steps listed are:

Command	Target	Value
open	/math-calculator.html	
selectWindow	name=n15KfMuM	
clickAndWait	link=Percent Calculator	
selectWindow	name=n15KfMuM	
type	id=cpar1	10
selectWindow	name=n15KfMuM	
type	id=cpar2	50
selectWindow	name=n15KfMuM	
clickAndWait	css=input[type="image"]	

A context menu is open over the 'Command' dropdown in the bottom right corner, listing various verification commands:

- addLocationStrategy
- addLocationStrategyAndWait
- addScript
- addScriptAndWait
- addSelection
- addSelectionAndWait
- allowNativeXpath
- allowNativeXpathAndWait
- altKeyDown
- altKeyDownAndWait
- altKeyUp
- altKeyUpAndWait
- answerOnNextPrompt
- assertAlert

Given below are the mostly used verification commands that help us check if a particular step has passed or failed.

- verifyElementPresent
- assertElementPresent
- verifyElementNotPresent
- assertElementNotPresent
- verifyText
- assertText

- verifyAttribute
- assertAttribute
- verifyChecked
- assertChecked
- verifyAlert
- assertAlert
- verifyTitle
- assertTitle

Synchronization Points

During script execution, the application might respond based on server load, hence it is required for the application and script to be in sync. Given below are a few commands that we can use to ensure that the script and application are in sync.

- waitForAlertNotPresent
- waitForAlertPresent
- waitForElementPresent
- waitForElementNotPresent
- waitForTextPresent
- waitForTextNotPresent
- waitForPageToLoad
- waitForFrameToLoad

Pattern Matching

Like locators, patterns are a type of parameter frequently used by Selenium. It allows users to describe patterns with the help of special characters. Many a time, the text that we would like to verify are dynamic; in that case, pattern matching is very useful.

Pattern matching is used with all the verification point commands – verifyTextPresent, verifyTitle, verifyAlert, assertConfirmation, verifyText, and verifyPrompt.

There are three ways to define a pattern:

- globbing,
- regular expressions, and
- exact patterns.

Globbing

Most techies who have used file matching patterns in Linux or Windows while searching for a certain file type like *.doc or *.jpg would be familiar with term "globbing".

Globbing in Selenium supports only three special characters: *, ?, and [].

- * - matches any number of characters.
- ? - matches a single character.
- [] - called a character class, lets you match any single character found within the brackets. [0-9] matches any digit.

To specify a glob in a Selenium command, prefix the pattern with the keyword 'glob:'. For example, if you would like to search for the texts "tax year 2013" or "tax year 2014", then you can use the glob "tax year *" as shown below.

However the usage of "glob:" is optional while specifying a text pattern because globbing patterns are the default in Selenium.

Command	Target	Value
clickAndWait	link=search	
verifyTextPresent	glob: tax year *	

Exact Patterns

Patterns with the prefix 'exact:' will match the given text as it is. Let us say, the user wants an exact match with the value string, i.e., without the glob operator doing its work, one can use the 'exact' pattern as shown below. In this example, the operator '*' will work as a normal character rather than a pattern-matching wildcard character.

Command	Target	Value
clickAndWait	link=search	
verifyValue	exact: *.doc	

Regular Expressions

Regular expressions are the most useful among the pattern matching techniques available. Selenium supports the complete set of regular expression patterns that Javascript supports. Hence the users are no longer limited by *, ?, and [] globbing patterns.

To use RegEx patterns, we need to prefix with either "regexp:" or "regexpi:". The prefix "regexpi" is case-insensitive. The glob: and the exact: patterns are the subsets of the Regular Expression patterns. Everything that is done with glob: or exact: can be accomplished with the help of RegExp.

Example

For example, the following will test if an input field with the id 'name' contains the string 'tax year', 'Tax Year', or 'tax Year'.

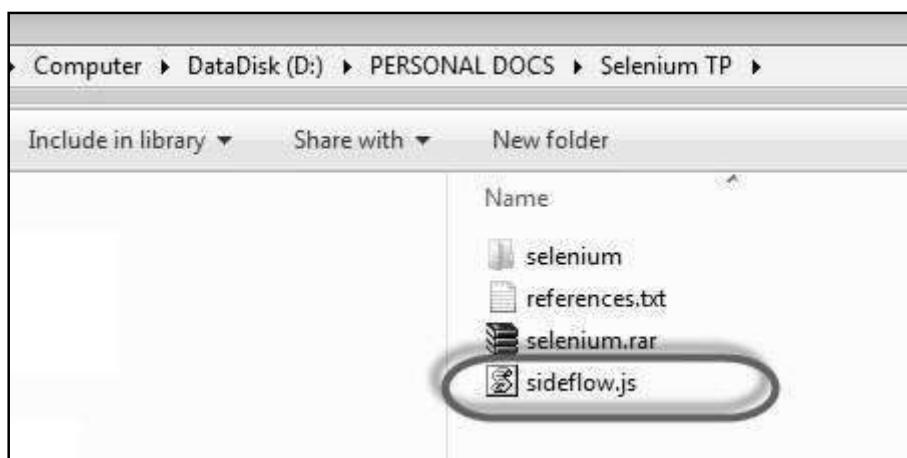
Command	Target	Value
clickAndWait	link=search	
verifyValue	id=name	regexp:[Tt]ax ([Yy]ear)

Selenium User Extensions

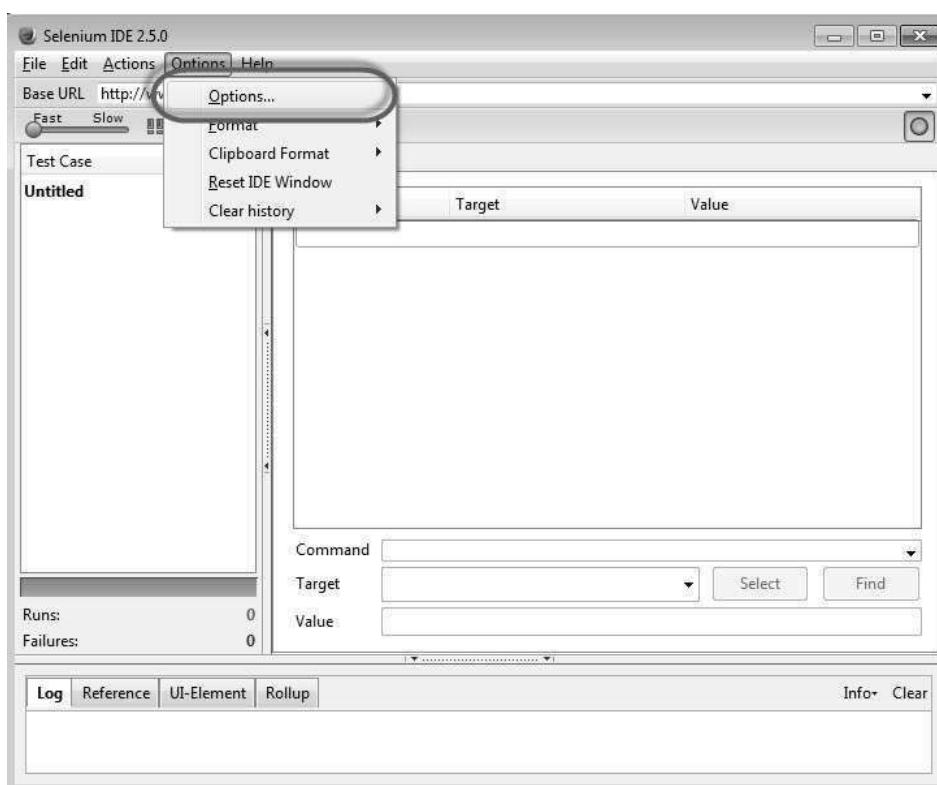
It is easy to extend Selenium IDE by adding customized actions, assertions, and locator-strategies. It is done with the help of JavaScript by adding methods to the Selenium object prototype. On startup, Selenium will automatically look through the methods on these prototypes, using name patterns to recognize which ones are actions, assertions, and locators.

Let us add a 'while' Loop in Selenium IDE with the help of JavaScript.

Step 1 : To add the js file, first navigate to <https://github.com/darrenderidder/sideflow/blob/master/sideflow.js> and copy the script and place save it as 'sideflow.js' in your local folder as shown below.



Step 2 : Now launch 'Selenium IDE' and navigate to "Options" >> "Options" as shown below.



Step 3 : Click the 'Browse' button under 'Selenium Core Extensions' area and point to the js file that we have saved in Step 1.



Step 4 : Restart Selenium IDE.

Step 5 : Now you will have access to a few more commands such as "Label", "While", etc.

Step 6 : Now we will be able to create a While loop within Selenium IDE and it will execute as shown below.

The screenshot shows the Selenium IDE interface with the following details:

- Title Bar:** try.html - Selenium IDE 2.5.0
- Menu Bar:** File Edit Actions Options Help
- Toolbar:** Base URL http://www.calculator.net/ (with a dropdown arrow), Fast/Slow speed slider, and various icons.
- Test Case List:** Test Case (highlighted) and try.
- try Test Case Content:**

Command	Target	Value
store	5	x
while	$\$\{x\}<10$	
echo	Value of x is \$x	
storeEval	new Number(storedVars['x'])+1	x
endWhile		
- Search/Filter:** Command: store, Target: 5, Value: x, Select, Find buttons.
- Log Window:**
 - Log tab selected.
 - Content:
 - [info] Executing: |echo | Value of x is \$x | |
 - [info] echo: Value of x is \$x
 - [info] Executing: |storeEval | new Number(storedVars['x'])+1 | x |
 - [info] script is: new Number(storedVars['x'])+1
 - [info] Executing: |endWhile | | |
 - [info] Executing: |while | \$\{x\}<10 | |

Different Browser Execution

Selenium scripts can run tests only against Firefox as the tool IDE itself is a plugin of Firefox. Tests developed using Selenium IDE can be executed against other browsers by saving it as Selenium WebDriver or Selenium Remote Control Script.

Step 1 : Open any saved Test in Selenium IDE.

Step 2 : Navigate to "File" menu and select "Export Test Suite As" and the options would be listed.

The screenshot shows the Selenium IDE interface. The 'File' menu is open, and the 'Export Test Suite As...' option is highlighted. A dropdown menu is displayed, listing several options: Ruby / RSpec / WebDriver, Ruby / Test::Unit / WebDriver, Ruby / RSpec / Remote Control, Java / JUnit 4 / WebDriver (which is circled), Java / JUnit 4 / WebDriver Backed, Java / JUnit 4 / Remote Control, Java / JUnit 3 / Remote Control, Java / TestNG / Remote Control, and C# / NUnit / WebDriver. Below the menu, the log window shows test execution details.

Condition	Target	Value
and	/	
idWait	//div[@id='menu']/div[3]/a/img	
idWait	link=Percent Calculator	
id=cpar1		10
id=cpar2		50
idWait	css=input[type="image"]	
ext	css=b	5

```

Log Reference UI-Element Rollup Info Clear
[info] Executing: |echo| Value of x is ${x}
[info] Executing: |storeEval| new Number(storedVars['x'])+1 | x |
[info] script is: new Number(storedVars['x'])+1
[info] Executing: |endWhile|||
[info] Executing: |while| ${x}<10|||
[info] Changed test case
  
```

Step 3 : Now let us export the script to "WebDriver" and save it with a name.

Step 4 : The saved WebDriver file is displayed as shown below.

```
1 import junit.framework.Test;
2 import junit.framework.TestSuite;
3
4 public class percentcalc {
5
6     public static Test suite() {
7         TestSuite suite = new TestSuite();
8         suite.addTestSuite(math_calculator.class);
9         return suite;
10    }
11
12    public static void main(String[] args) {
13        junit.textui.TestRunner.run(suite());
14    }
15 }
```

3. ENVIRONMENT SETUP

In order to develop Selenium RC or WebDriver scripts, users have to ensure that they have the initial configuration done. Setting up the environment involves the following steps.

- Download and Install Java
- Download and Configure Eclipse
- Configure FireBug and FirePath
- Configure Selenium RC
- Configure Selenium WebDriver

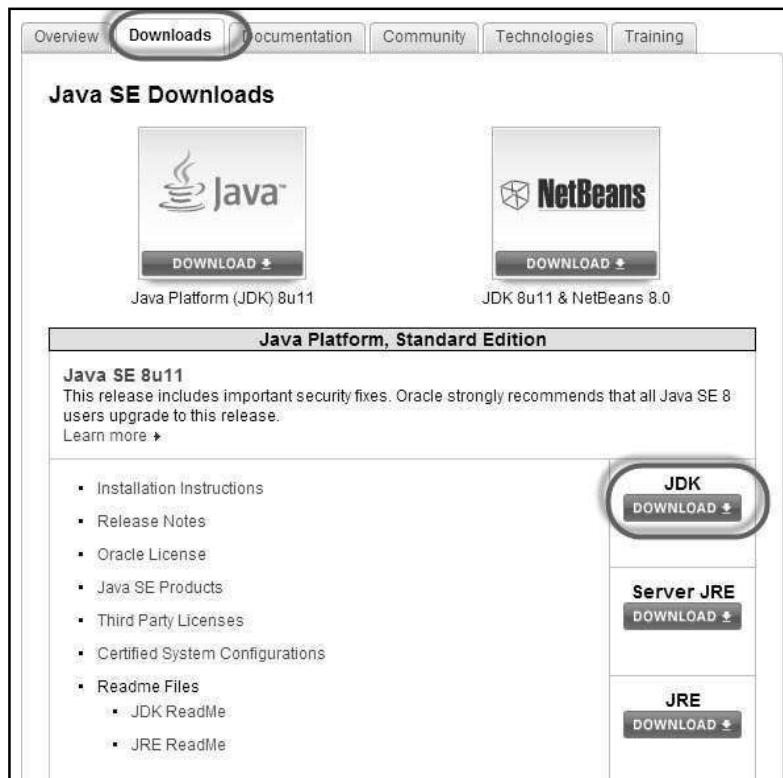
Download and Install Java

We need to have JDK (Java Development Kit) installed in order to work with Selenium WebDriver/Selenium. Let us see how to download and install Java.

Step 1: Navigate to the URL:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Step 2: Go to "Downloads" section and select "JDK Download".



Step 3: Select "Accept License Agreement" radio button.

- Overview
- Downloads
- Documentation
- Community
- Technologies
- Training

Java SE Development Kit 8 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

See also:

- Java Developer Newsletter (tick the checkbox under Subscription Center > Oracle Technology News)
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

JDK MD5 Checksum

Looking for JDK 8 on ARM?
JDK 8 for ARM downloads have moved to the [JDK 8 for ARM download page](#).

Java SE Development Kit 8u11

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 x86	133.58 MB	jdk-8u11-linux-i586.rpm
Linux x86	152.55 MB	jdk-8u11-linux-i586.tar.gz
Linux x64	133.89 MB	jdk-8u11-linux-x64.rpm
Linux x64	151.65 MB	jdk-8u11-linux-x64.tar.gz
Mac OS X x64	207.82 MB	jdk-8u11-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	135.66 MB	jdk-8u11-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	96.14 MB	jdk-8u11-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	135.7 MB	jdk-8u11-solaris-x64.tar.Z
Solaris x64	93.18 MB	jdk-8u11-solaris-x64.tar.gz
Windows x86	151.81 MB	jdk-8u11-windows-i586.exe
Windows x64	155.29 MB	jdk-8u11-windows-x64.exe

Step 4 : Select the appropriate installation. In this case, it is 'Windows 7-64' bit. Click the appropriate link and save the .exe file to your disk.

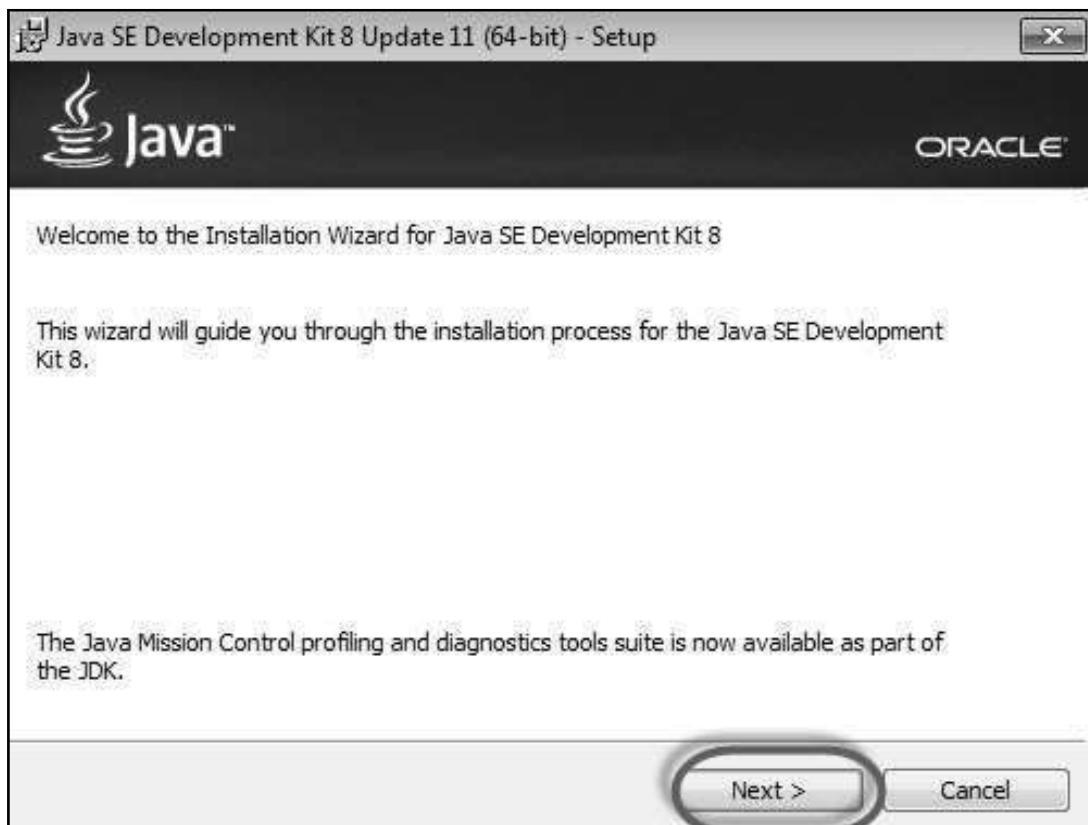
Java SE Development Kit 8u11

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

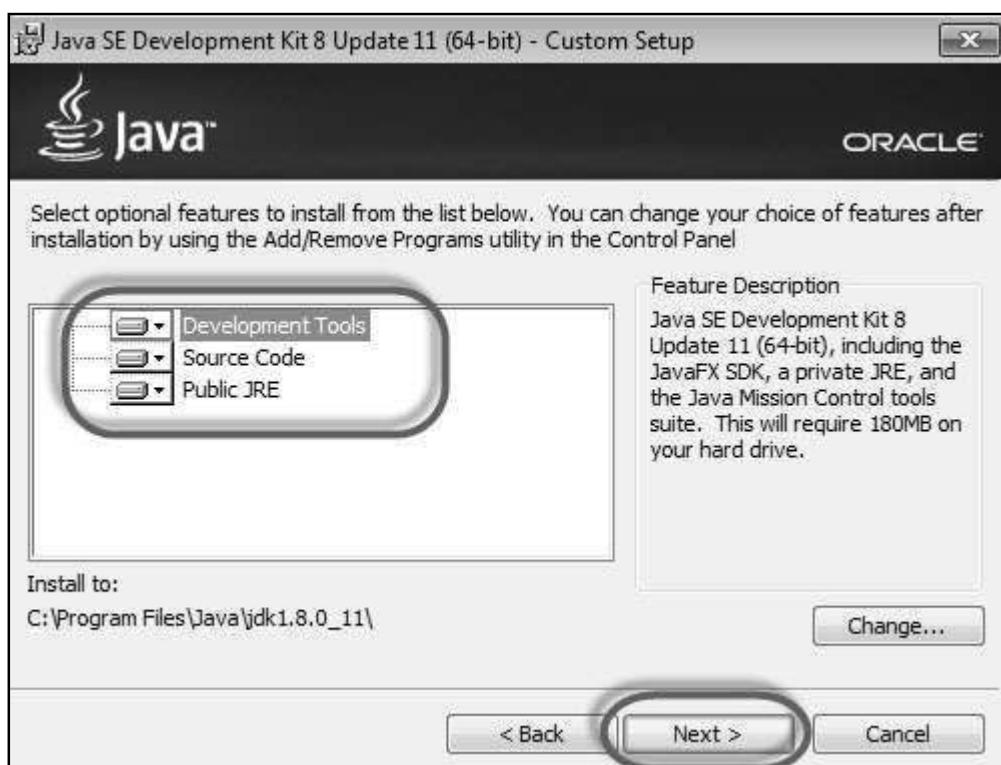
Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	133.58 MB	jdk-8u11-linux-i586.rpm
Linux x86	152.55 MB	jdk-8u11-linux-i586.tar.gz
Linux x64	133.89 MB	jdk-8u11-linux-x64.rpm
Linux x64	151.65 MB	jdk-8u11-linux-x64.tar.gz
Mac OS Xx64	207.82 MB	jdk-8u11-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	135.66 MB	jdk-8u11-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	96.14 MB	jdk-8u11-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	135.7 MB	jdk-8u11-solaris-x64.tar.Z
Solaris x64	93.18 MB	jdk-8u11-solaris-x64.tar.gz
Windows x86	151.81 MB	jdk-8u11-windows-i586.exe
Windows x64	155.29 MB	jdk-8u11-windows-x64.exe

Step 5 : Run the downloaded exe file to launch the Installer wizard. Click 'Next' to continue.



Step 6 : Select the features and click 'Next'.



Step 7 : The installer is extracted and its progress is shown in the wizard.



Step 8 : The user can choose the install location and click 'Next'.



Step 9 : The installer installs the JDK and new files are copied across.



Step 10 : The Installer installs successfully and displays the same to the user.



Step 11 : To verify if the installation was successful, go to the command prompt and just type 'java' as a command. The output of the command is shown below. If the Java installation is unsuccessful or if it had NOT been installed, it would throw an "unknown command" error.

```
C:\>java
Usage: java [-optional 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/jar files>
              A ; separated list of directories, JAR archives,
              and ZIP archives to search for class files.
    -D<name>=<value>
              set a system property
    -verbose:[class|gc|jni]
              enable verbose output
    -version   print product version and exit
    -version:<value>
              require the specified version to run
    -showversion print product version and continue
    -jre-restrict-search | -no-jre-restrict-search
              include/exclude user private JREs in the version search
    -? -help   print this help message
    -X       print help on non-standard options
    -real:<packagename>...!:<classname>
    -enableassertions[!:<packagename>...!:<classname>]
              enable assertions with specified granularity
    -da[!:<packagename>...!:<classname>]
    -disableassertions[!:<packagename>...!:<classname>]
              disable assertions with specified granularity
    -esa[!:<classname>]
    -enableassertions
              enable system assertions
    -dsa[!:<classname>]
    -disableassertions
              disable system assertions
    -agentlib:<library>[!<options>]
              load native agent library <library>, e.g., -agentlib:hprof
              see also, -agentlib:jdwp=help and -agentlib:hprof=help
    -agentpath:<pathname>[!<options>]
              load native agent library by full pathname
    -javaagent:<jarpath>[!<options>]
              load Java programming language agent, see java.lang.instrument
    -splash:<imagepath>
              show splash screen with specified image
See http://www.oracle.com/technetwork/java/javase/documentation/index.html for more details.
C:\>
```

Download and Configure Eclipse

Step 1 : Navigate to the URL: <http://www.eclipse.org/downloads/> and download the appropriate file based on your OS architecture.

Eclipse Luna (4.4) Release for Windows

Eclipse Standard 4.4, 206 MB
Downloaded 955,976 Times [Other Downloads](#)

Standard Eclipse package suited for Java and plug-in development plus adding new plugins; already includes Git, Marketplace Client, source code and...

[Windows 32 Bit](#) [Windows 64 Bit](#)

Step 2 : Click the 'Download' button.

Downloads Home

» Source code

» More Packages

Give Back to Eclipse

\$5 \$15

Eclipse downloads - mirror selection

All downloads are provided under the terms and conditions of the [Eclipse Foundation Software User Agreement](#) unless otherwise specified.

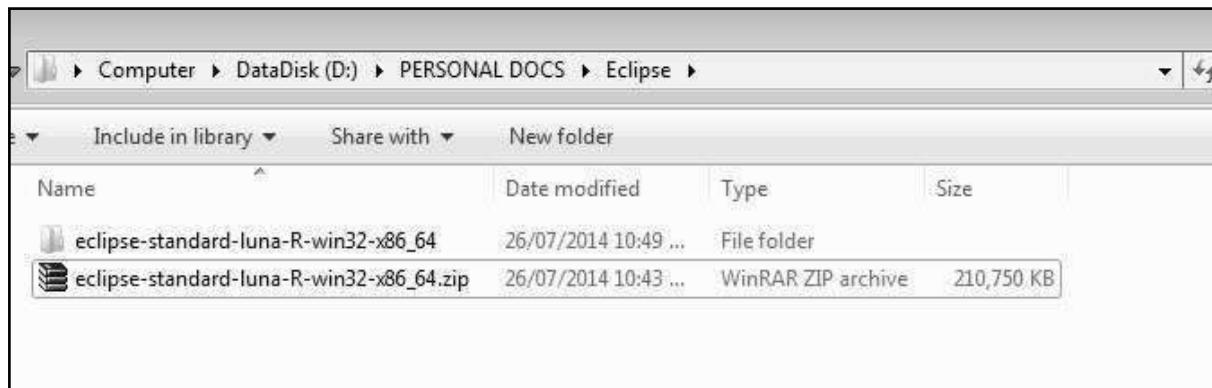
Download [eclipse-standard-luna-R-win32-x86_64.zip](#) from:

[China] Beijing Institute of Technology ([http](http://))

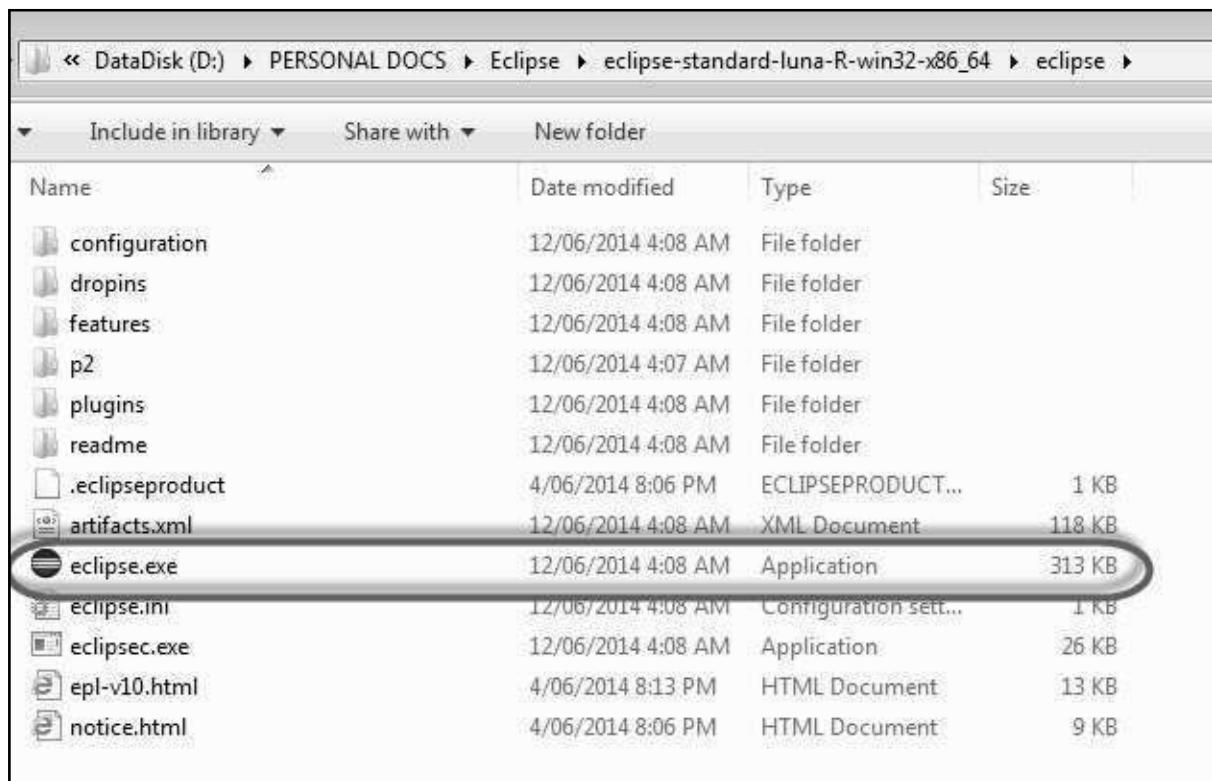
Checksums: [\[MD5\]](#) [\[SHA1\]](#) [\[SHA-512\]](#)

...or pick a mirror site below.

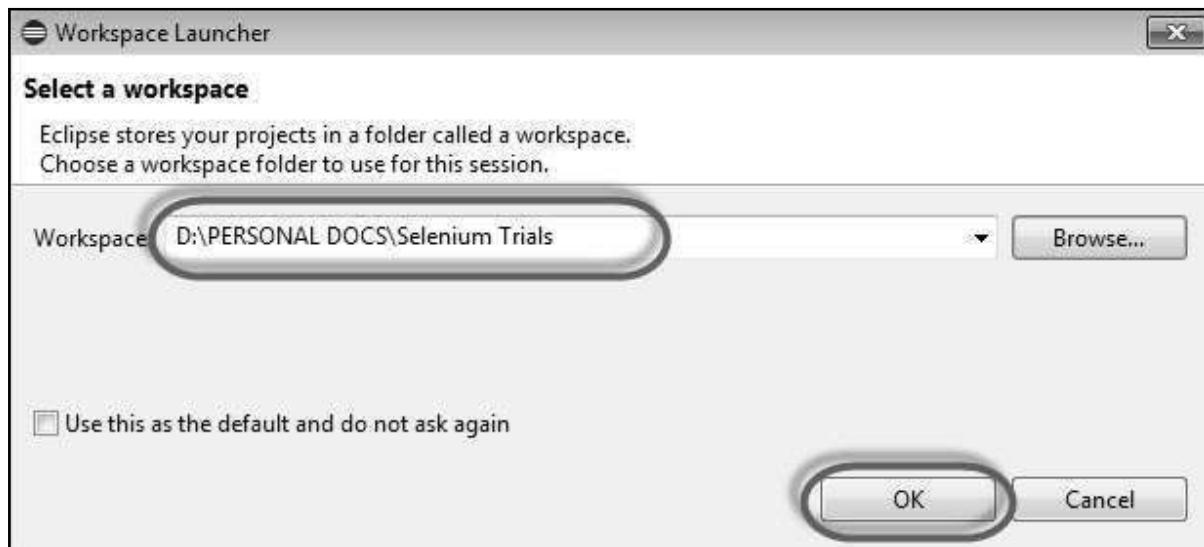
Step 3 : The download would be in a Zipped format. Unzip the contents.



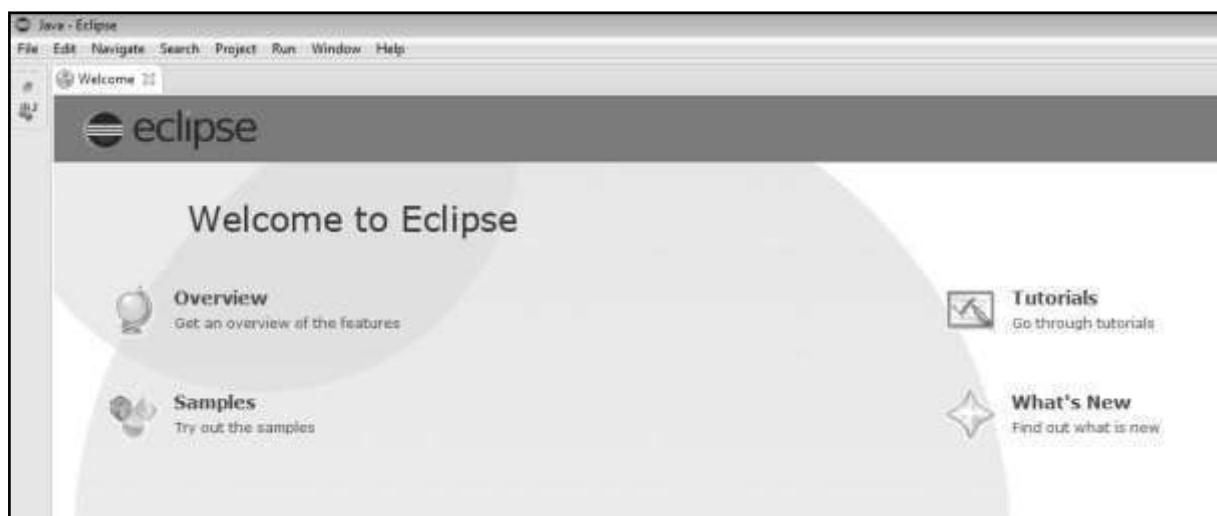
Step 4 : Locate Eclipse.exe and double click on the file.



Step 5 : To configure the workspace, select the location where the development has to take place.



Step 6 : The Eclipse window opens as shown below.



Configure FireBug and FirePath

To work with Selenium RC or WebDriver, we need to locate elements based on their XPath or ID or name, etc. In order to locate an element, we need tools/plugins.

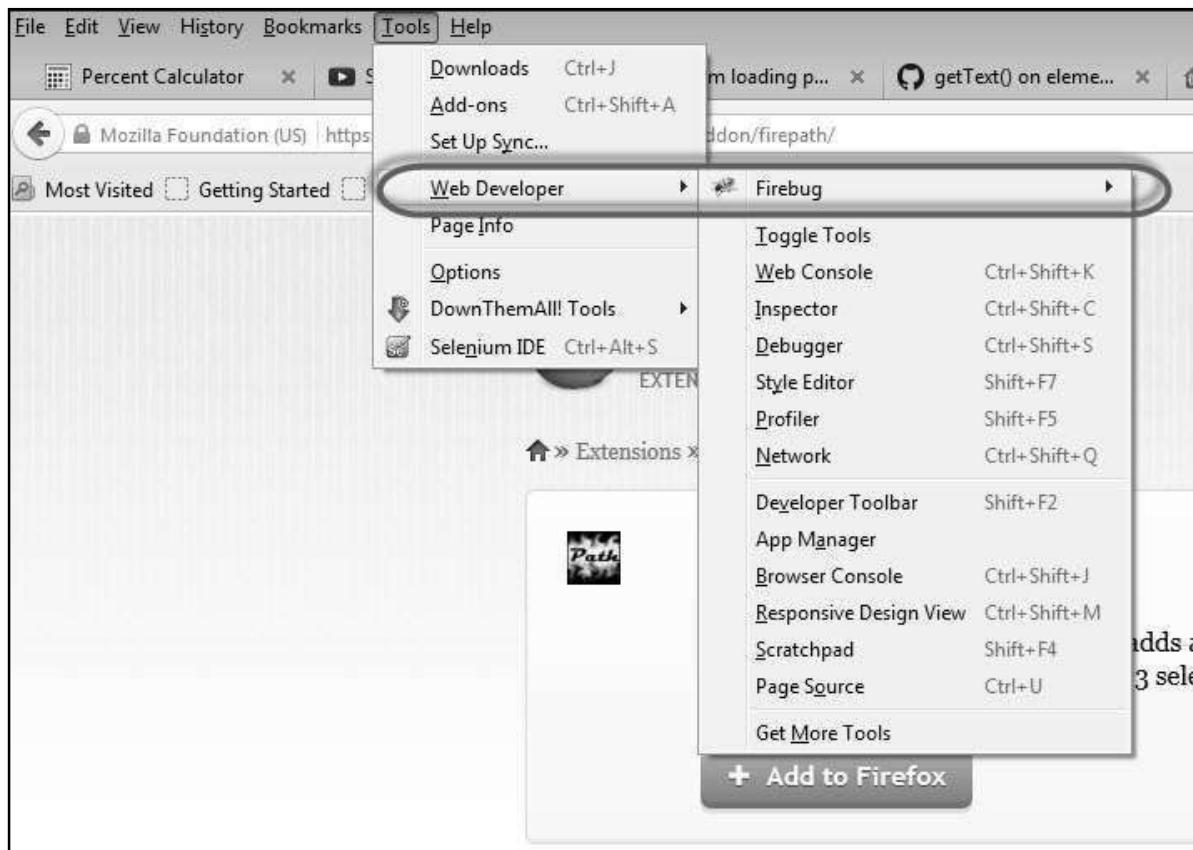
Step 1 : Navigate to the URL: <https://addons.mozilla.org/en-US/firefox/addon/firebug/> and download plugin.

The screenshot shows the Mozilla Add-ons website. At the top, there are links for 'Register' or 'Log in' and 'Other Applications'. The main header says 'ADD-ONS' with a search bar below it. A navigation bar indicates the user is in the 'Extensions' section under 'Firebug'. The main content area features a card for 'Firebug 2.0.2'. It includes a small icon of a bug, the version number, and the developer's name, Joe Hewitt, Jan Odvarko, robcoee, and FirebugWorkingGroup. Below this is a brief description: 'Firebug integrates with Firefox to put a wealth of development tools at your fingertips while you browse. You can edit, debug, and monitor CSS, HTML, and JavaScript live in any web page...'. A large button labeled '+ Add to Firefox' is prominently displayed. To the right of the card, there are user reviews: 1,655 reviews and 2,589,073 users. Below the reviews are buttons for 'Add to collection' and 'Share this Add-on'. At the bottom of the card, there's a note about donations: 'Enjoy this add-on?' followed by 'The developer of this add-on asks that you show your support by making a donation to the Mozilla Foundation.' A 'Contribute' button and a note '\$0.00 suggested' are also present.

Step 2 : The add-on installer is shown to the user and it is installed upon clicking the 'Install' button.



Step 3 : After installing, we can launch the plugin by navigating to "Web Developer" >> "Firebug".



Step 4 : FirePath, a plugin that works within Firebug, helps users to grab the 'XPath' of an element. Install FirePath by navigating to "<https://addons.mozilla.org/en-US/firefox/addon/firepath/>"

ADD-ONS

Extensions | Themes | Collections | MORE...

search for add-ons

FirePath 0.9.7
by Pierre Thiolence

FirePath is a Firebug extension that adds a development tool to edit, inspect and generate XPath 1.0 expressions, CSS 3 selectors and JQuery selectors (Sizzle selector engine).

+ Add to Firefox

★★★★★
16 user reviews
47,127 users

Add to collection | Share this Add-on

Step 5 : The add-on installer is shown to the user and it is installed upon clicking the 'Install' button.



Step 6 : Now launch "Firebug" by navigating to "Tools" >> "Webdeveloper" >> "Firebug".



Example

Now let us understand how to use FireBug and FirePath with an example. For demonstration, we will use www.google.com and capture the properties of the text box of "google.com".

Step 1 : First click on the arrow icon as highlighted in the following screenshot and drag it to the object for which we would like to capture the properties. The HTML/DOM of the object would be displayed as shown below. We are able to capture the 'ID' of the input text box with which we can interact.



Step 2 : To fetch the XPath of the object, go to 'firepath' tab and perform the following steps.

- Click the Spy icon.
- Select the Control for which we would like to capture the XPath.
- XPath of the selected control would be generated.



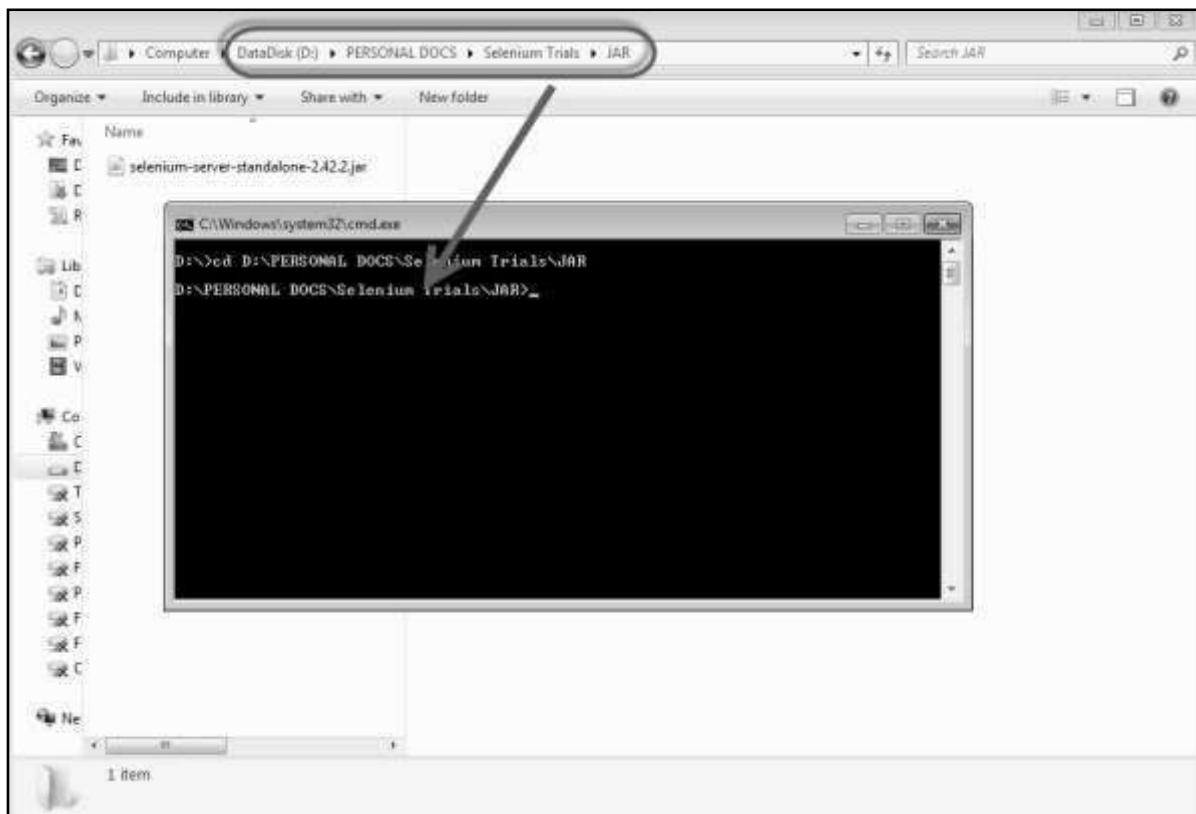
Configure Selenium RC

Now let us look at how to configure Selenium Remote control. We will understand how to develop scripts with Selenium RC in later chapters, however for now, we will understand just the configuration part of it.

Step 1 : Navigate to the Selenium downloads section <http://www.seleniumhq.org/download/> and download Selenium Server by clicking on its version number as shown below.

The screenshot shows the SeleniumHQ website's 'Downloads' page. On the left, there's a sidebar with links to 'Selenium Downloads', 'Latest Releases', 'Previous Releases', 'Source Code', and 'Maven Information'. Below that is a 'Donate to Selenium' section with a 'Donate' button and payment method icons (PayPal, Credit Card, Bank). The main content area has sections for 'Selenium IDE' and 'Selenium Server (formerly the Selenium RC Server)'. The 'Selenium Server' section includes a link to 'Download version 2.42.2'. A note at the bottom says 'To use the Selenium Server in a Grid configuration see the wiki page.'

Step 2 : After downloading, we need to start the Selenium Server. To do so, open command prompt and navigate to the folder where the downloaded JAR file is kept as shown below.



Step 3 : To start the server, use the command 'java -jar <<downloaded jar name >>' and if java JDK is installed properly, you would get a success message as shown below. Now we can start writing Selenium RC scripts.

```
C:\Windows\system32\cmd.exe - java -jar selenium-server-standalone-2.42.2.jar
D:\>cd D:\PERSONAL DOCS\Selenium Trials\JAR
D:\PERSONAL DOCS\Selenium Trials\JAR>java -jar selenium-server-standalone-2.42.2.jar
Jul 27, 2014 8:01:20 AM org.openqa.grid.selenium.GridLauncher main
INFO: Launching a standalone server
08:01:20.430 INFO - Java: Oracle Corporation 25.11-b03
08:01:20.431 INFO - OS: Windows 7 6.1 amd64
08:01:20.438 INFO - v2.42.2, with Core v2.42.2. Built from revision 6a6995d
08:01:20.542 INFO - RemoteWebDriver instances should connect to: http://127.0.0.1:4444/wd/hub
08:01:20.543 INFO - Version Jetty/5.1.x
08:01:20.546 INFO - Started HttpContext[/{selenium-server}/selenium-server]
08:01:20.626 INFO - Started org.openqa.jetty.jetty.servlet.ServletHandler@35851384
08:01:20.627 INFO - Started HttpContext[/{wd}/wd]
08:01:20.628 INFO - Started HttpContext[/{selenium-server}/driver]
08:01:20.628 INFO - Started HttpContext[/{},/]
08:01:20.633 INFO - Started SocketListener on 0.0.0.0:4444
08:01:20.633 INFO - Started org.openqa.jetty.Server@30dae81
```

Configure Selenium WebDriver

Now let us look at how to configure Selenium WebDriver. We will understand how to develop scripts with Selenium WebDriver in later chapters, however for now, we will understand just the configuration part of it.

Step 1 : Navigate to the selenium downloads section

<http://www.seleniumhq.org/download/> and download Selenium WebDriver by clicking on its version number as shown below.

The Internet Explorer Driver Server

This is required if you want to make use of the latest and greatest features of the WebDriver InternetExplorerDriver. Please make sure that this is available on your \$PATH (or %PATH% on Windows) in order for the IE Driver to work as expected.

Download version 2.42.0 for (recommended) [32 bit Windows IE](#) or [64 bit Windows IE CHangelog](#)

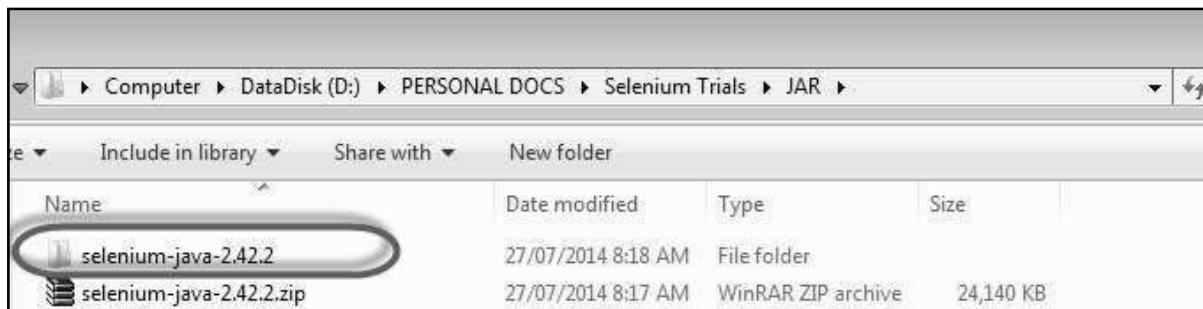
Selenium Client & WebDriver Language Bindings

In order to create scripts that interact with the Selenium Server (Selenium RC, Selenium Remote Webdriver) or create local Selenium WebDriver script you need to make use of language-specific client drivers. These languages include both 1.x and 2.x style clients.

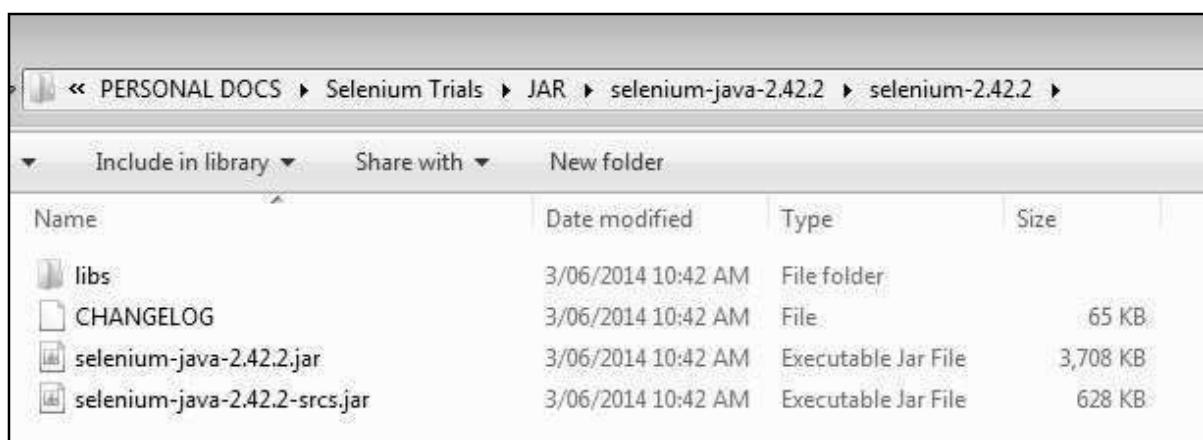
While language bindings for [other languages](#) exist, these are the core ones that are supported by the main project hosted on google code.

Language	Client Version	Release Date	Download	Change log	Javadoc
Java	2.42.2	2014-06-03	Download	Change log	Javadoc
C#	2.42.0	2014-05-27	Download	Change log	API docs
Ruby	2.42.0	2014-05-22	Download	Change log	API docs
Python	2.42.1	2014-05-27	Download	Change log	API docs
Javascript (Node)	2.42.0	2014-05-22	Download	Change log	API docs

Step 2 : The downloaded file is in Zipped format and one has to unzip the contents to map it to the project folder.



Step 3 : The Unzipped contents would be displayed as shown below. How to map it to the project folder and how to start scripting would be dealt in the WebDriver chapter.



4. SELENIUM RC

What is Selenium RC?

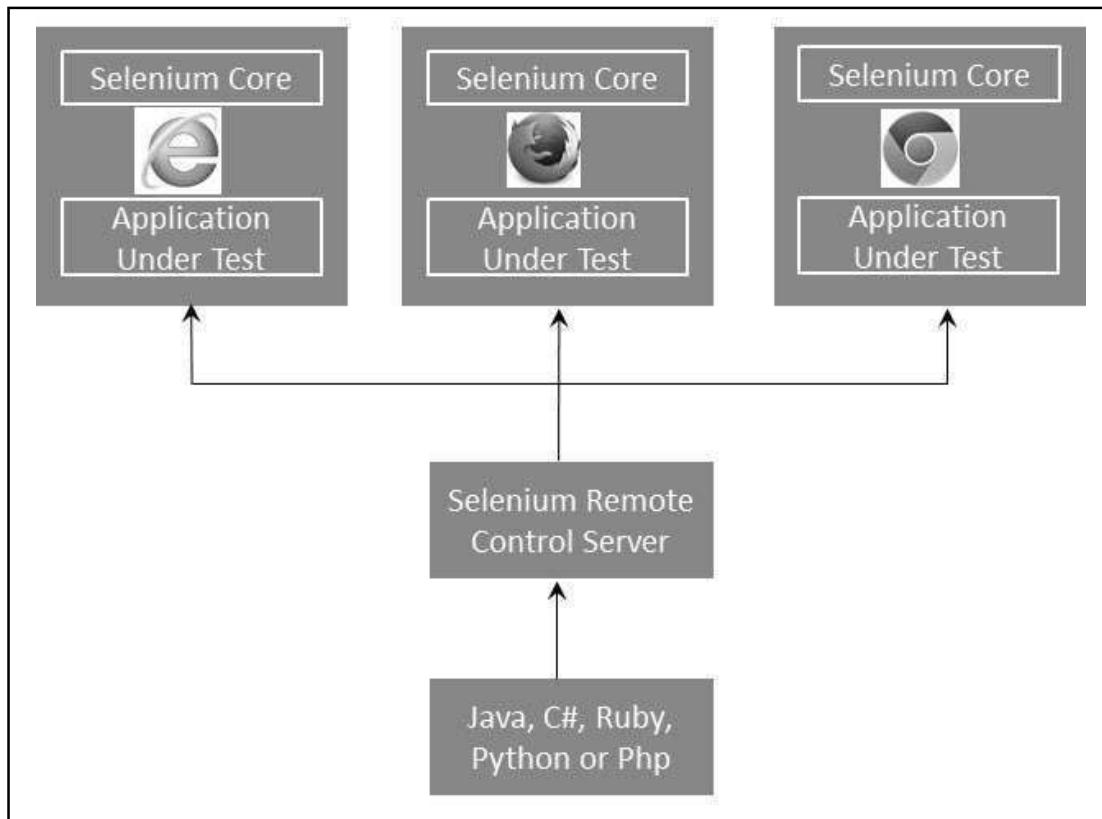
Selenium Remote Control (RC) was the main Selenium project that sustained for a long time before Selenium WebDriver (Selenium 2.0) came into existence. Now Selenium RC is hardly in use, as WebDriver offers more powerful features, however users can still continue to develop scripts using RC.

It allows us to write automated web application UI tests with the help of full power of programming languages such as Java, C#, Perl, Python, and PHP to create more complex tests such as reading and writing files, querying a database, and emailing test results.

Selenium RC Architecture

Selenium RC works in such a way that the client libraries can communicate with the Selenium RC Server passing each Selenium command for execution. Then the server passes the Selenium command to the browser using Selenium-Core JavaScript commands.

The browser executes the Selenium command using its JavaScript interpreter.



Selenium RC comes in two parts.

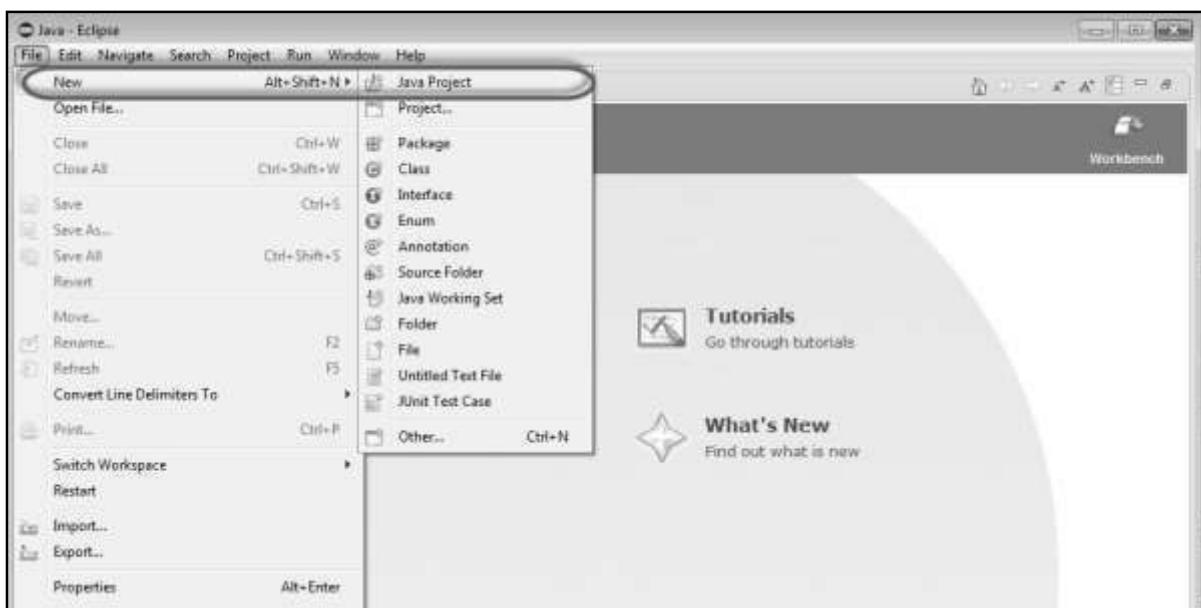
- The Selenium Server launches and kills browsers. In addition to that, it interprets and executes the Selenese commands. It also acts as an HTTP proxy by intercepting and verifying HTTP messages passed between the browser and the application under test.
- Client libraries that provide an interface between each one of the programming languages (Java, C#, Perl, Python, and PHP) and the Selenium-RC Server.

RC –Scripting

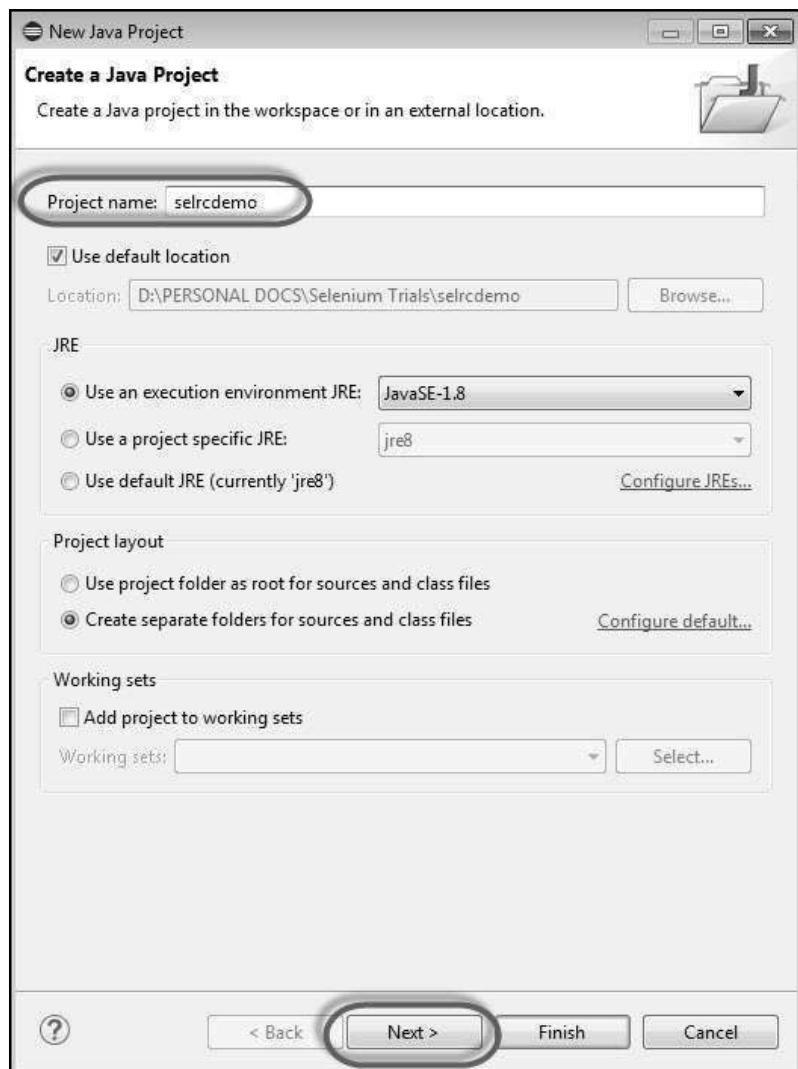
Now let us write a sample script using Selenium Remote Control. Let us use <http://www.calculator.net/> for understanding Selenium RC. We will perform a Percent calculation using 'Percent Calculator' that is present under the 'Math Calculators' module.

Step 1 : Start Selenium Remote Control (with the help of command prompt).

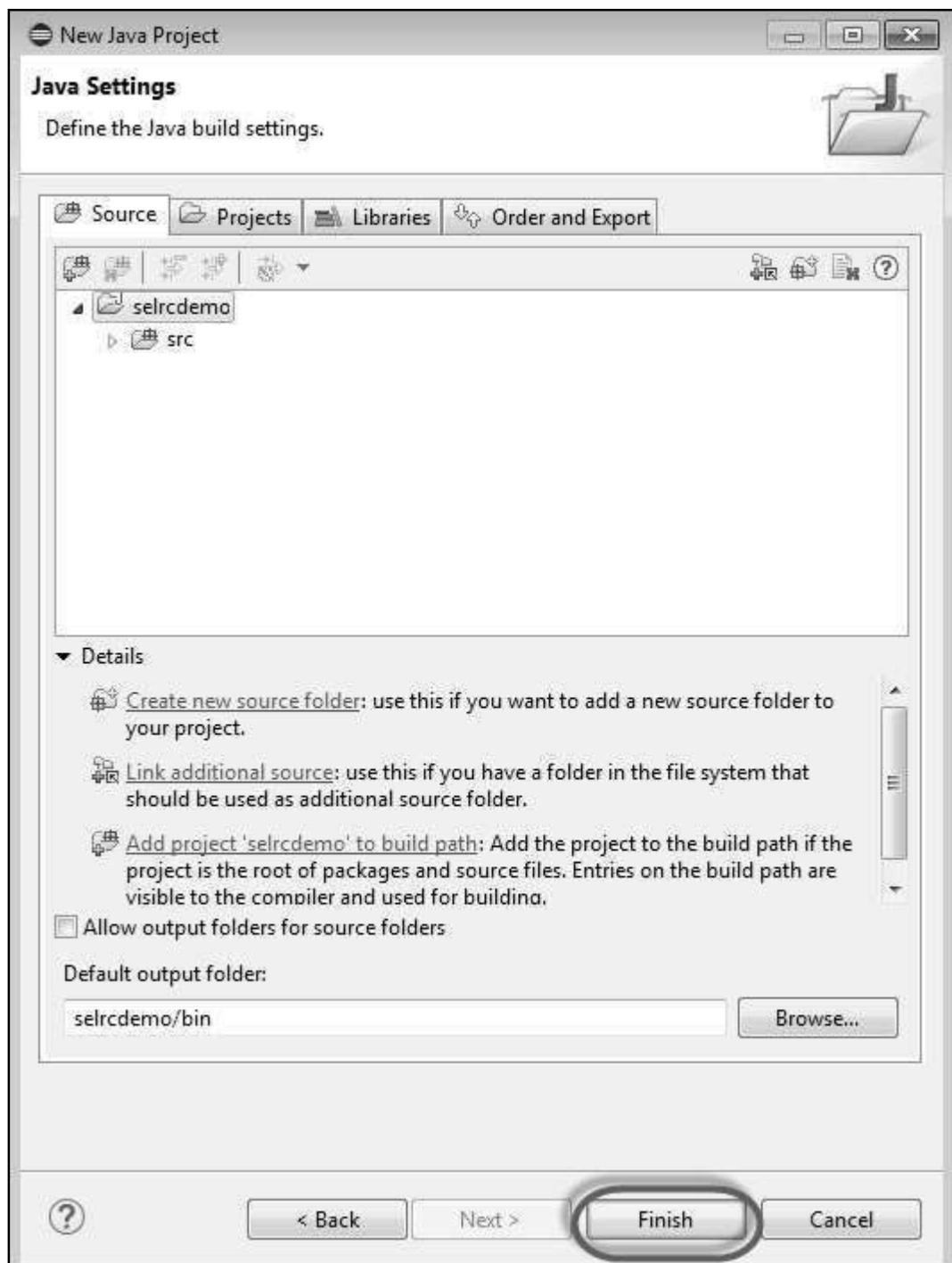
Step 2 : After launching Selenium RC, open Eclipse and create a "New Project" as shown below.



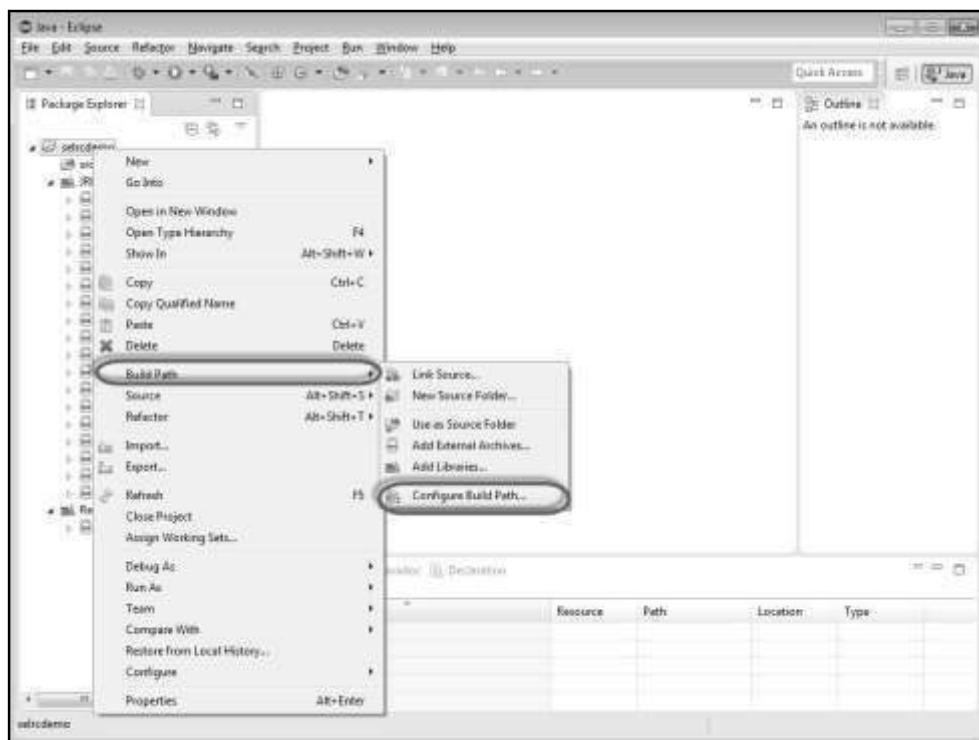
Step 3 : Enter the project name and click 'Next' button.



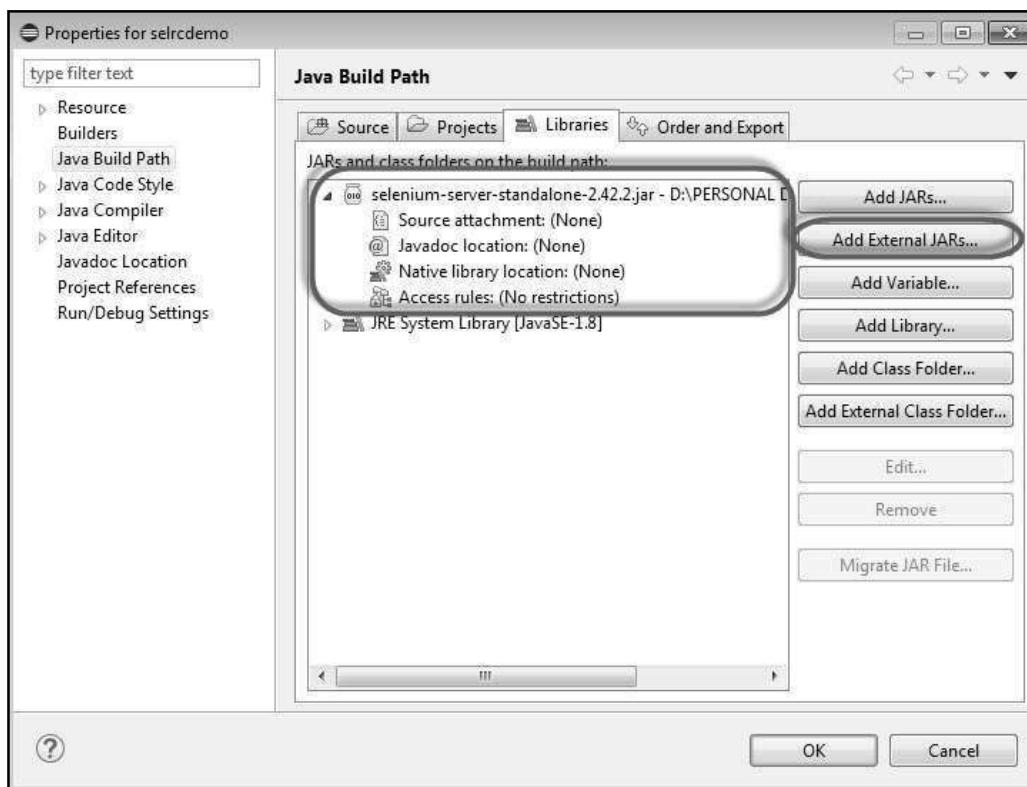
Step 4 : Verify the Source, Projects, Libraries, and Output folder and click 'Finish'.



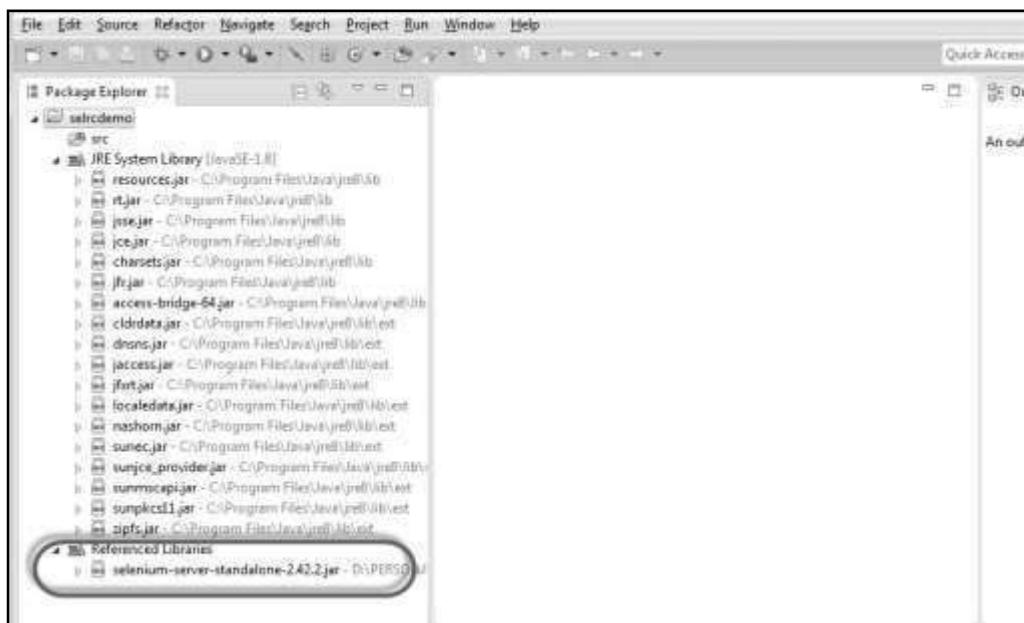
Step 5 : Right click on 'project' container and choose 'Configure Build Path'.



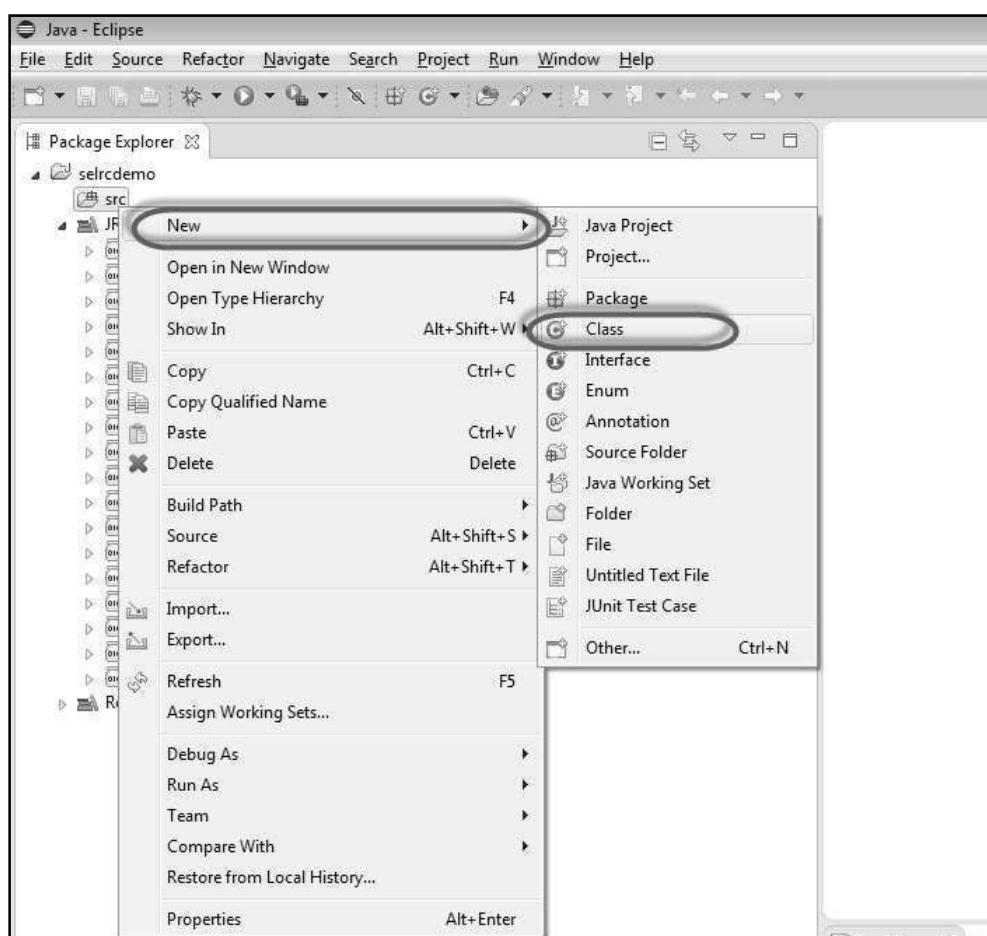
Step 6 : Properties for 'selrcdemo' opens up. Navigate to 'Libraries' tab and select 'Add External JARs'. Choose the Selenium RC jar file that we have downloaded and it would appear as shown below.



Step 7 : The referenced Libraries are shown as displayed below.



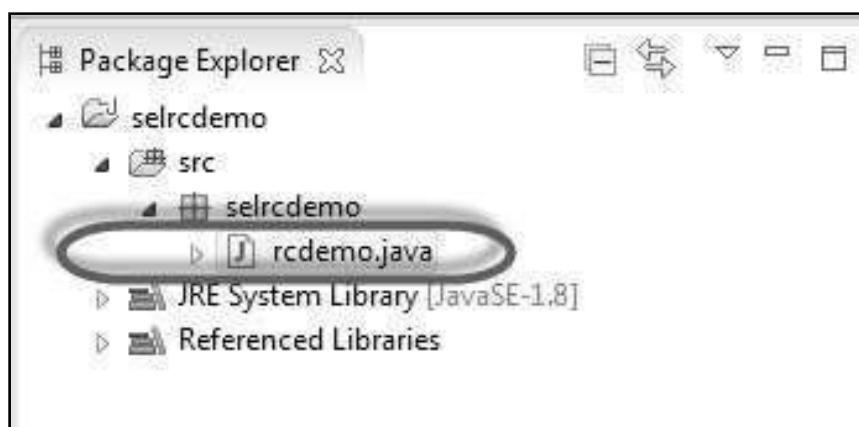
Step 8 : Create a new class file by performing a right click on 'src' folder and select 'New' >> 'class'.



Step 9 : Enter a name of the class file and enable 'public static void main' as shown below.



Step 10 : The Created Class is created under the folder structure as shown below.



Step 11 : Now it is time for coding. The following code has comments embedded in it to make the readers understand what has been put forth.

```

package seleniumrcdemo;

import com.thoughtworks.selenium.DefaultSelenium;
import com.thoughtworks.selenium.Selenium;

public class rcdemo
{
    public static void main(String[] args) throws InterruptedException
    {

        // Instantiate the RC Server
        Selenium selenium = new DefaultSelenium("localhost", 4444,
        "firefox", "http://www.calculator.net");
        selenium.start();                                // Start
        selenium.open("/");                            // Open the URL
        selenium.windowMaximize();

        // Click on Link Math Calculator
        selenium.click("xpath=//*[@id='menu']/div[3]/a");
        Thread.sleep(2500);                           // Wait for page load

        // Click on Link Percent Calculator
        selenium.click("xpath=//*[@id='menu']/div[4]/div[3]/a");
        Thread.sleep(4000);                           // Wait for page load

        // Focus on text Box
        selenium.focus("name=cpar1");

        // enter a value in Text box 1
        selenium.type("css=input[name='cpar1']", "10");
    }
}

```

```

// enter a value in Text box 2
selenium.focus("name=cpar2");
selenium.type("css=input[name='cpar2']", "50");

// Click Calculate button
selenium.click("xpath=//*[@id='content']/table/tbody/tr/td[2]/input");

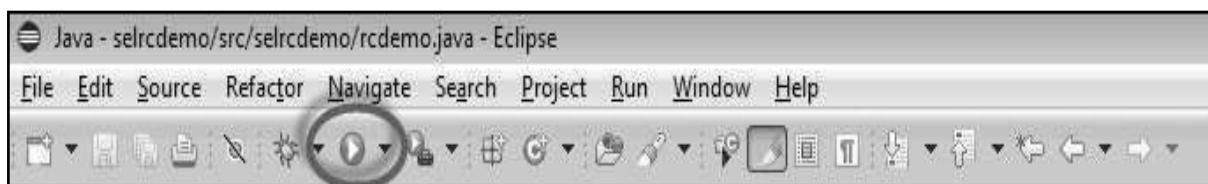
// verify if the result is 5
String result = selenium.getText("//*[@id='content']/p[2]");

if (result == "5")
{
    System.out.println("Pass");
}
else
{
    System.out.println("Fail");
}

}
}

```

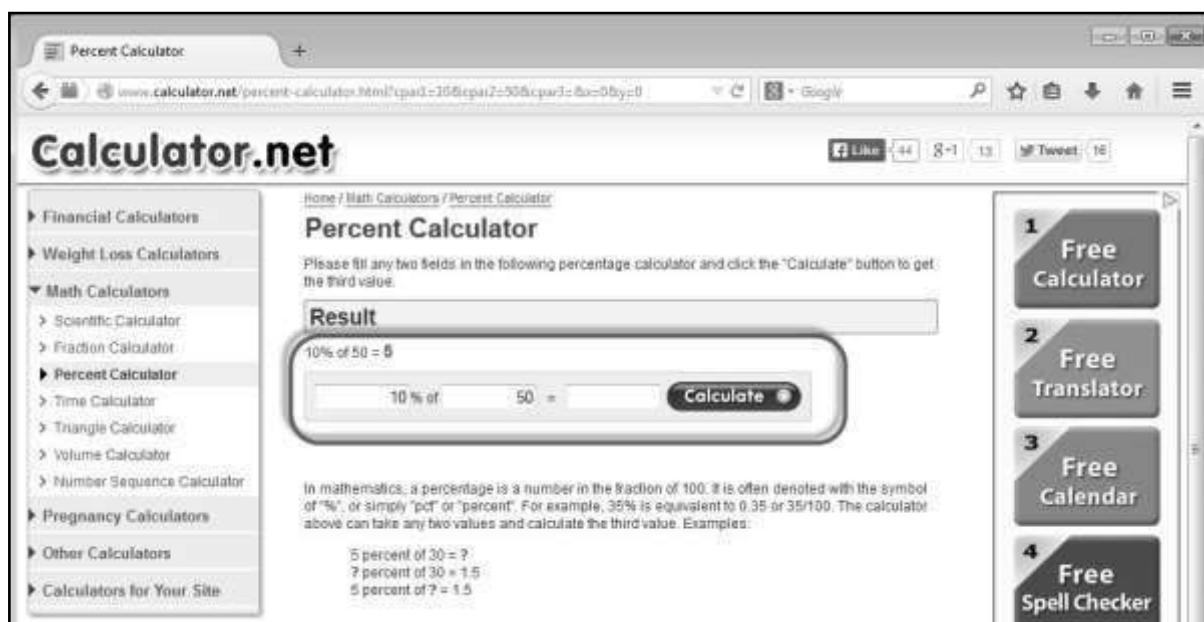
Step 11 : Now, let us execute the script by clicking the 'Run' Button.



Step 12 : The script would start executing and the user would be able to see the command history under the 'Command History' Tab.



Step 13 : The final state of the application is shown as below. The percentage is calculated and it displays the result on the screen as shown below.



Step 14 : The output of the test is printed on the Eclipse console as shown below, as we have printed the output to the console. In real time, the output is written to an HTML file or in a simple Text file.



5. SELENESE COMMANDS

A command refers to what Selenium has to do and the commands in Selenium are of three types:

- Actions
- Accessors
- Assertions

Actions

Actions are commands that manipulate the state of the application. Upon execution, if an action fails, the execution of the current test is stopped. For example, "click a link" and "select an option".

The following table lists the Selenium action commands that are used very frequently, however the list is not exhaustive.

Command/Syntax	Description
click (locator)	Clicks on a link, button, checkbox or radio button
clickAt (locator, coordString)	Clicks on an element with the help of locator and coordinates
close ()	Simulates the user clicking the "close" button in the title bar of a popup window or tab.
contextMenuAt (locator, coordString)	Simulates opening the context menu of the specified element from a

	specified location
doubleClick (locator)	Double clicks on a webelement based on the specified element.
dragAndDrop (locator, movementsString)	Drags an element and then drops it based on specified distance.
dragAndDropToObject (Dragobject, dropobject)	Drags an element and drops it on another element.
Echo (message)	Prints the specified message on console which is used for debugging.
fireEvent (locator,eventName)	Explicitly simulate an event, to trigger the corresponding "onevent" handler
focus (locator)	Move the focus to the specified element
highlight (locator)	Changes the background color of the specified element to yellow which is useful for debugging purposes.
mouseDown (locator)	Simulates a user pressing the left mouse button on the specified element.

mouseDownAt (locator, coordString)	Simulates a user pressing the left mouse button at the specified location on the specified element.
mouseUp (locator)	Simulates the event that occurs when the user releases the mouse button
mouseUpAt (locator, coordString)	Simulates the event that occurs when the user releases the mouse button at the specified location.
open (url)	Opens a URL in the specified browser and it accepts both relative and absolute URLs.
openWindow (url, windowID)	Opens a popup window. After opening the window, user need to activate it using the selectWindow command.
pause (waitTime)	Waits for the specified amount of time (in milliseconds)
refresh()	Simulates the user clicking the "Refresh" button on their browser.
select (selectLocator, optionLocator)	Select an option from a drop-down using an option

	locator.
selectWindow (windowID)	Selects a popup window using a window locator; once a popup window has been selected, all focus shifts to that window.
store (expression, variableName)	The name of a variable in which the result is to be stored and expression is the value to store.
type (locator, value)	Sets the value of an input field, similar to user typing action.
typeKeys (locator, value)	Simulates keystroke events on the specified element, as though you typed the value key-by-key.
waitForCondition (script, timeout)	Executes the specified JavaScript snippet repeatedly until it evaluates to "true".
waitForPageToLoad (timeout)	Waits for a new page to load.
waitForPopUp (windowID, timeout)	Waits for a popup window to appear and load.
windowFocus()	Gives focus to the currently

	selected window
windowMaximize()	Resize the currently selected window to take up the entire screen

Accessors

Accessors evaluate the state of the application and store the results in a variable which is used in assertions. For example, "storeTitle".

The following table lists the Selenium accessors that are used very frequently, however the list is not exhaustive.

Command/Syntax	Description
assertErrorOnNext (message)	Pings Selenium to expect an error on the next command execution with an expected message.
storeAllButtons (variableName)	Returns the IDs of all buttons on the page.
storeAllFields (variableName)	Returns the IDs of all input fields on the page.
storeAllLinks (variableName)	Returns the IDs of all links on the page.
storeAllWindowsIds (variableName)	Returns the IDs of all windows that the browser knows about in an array.

storeAllWindowTitles (variableName)	Returns the names of all windows that the browser knows about in an array.
storeAllWindowNames (variableName)	Returns the titles of all windows that the browser knows about in an array.
storeAttribute (attributeLocator, variableName)	Gets the value of an element attribute. The value of the attribute may differ across browsers.
storeBodyText (variableName)	Gets the entire text of the page.
storeConfirmation (variableName)	Retrieves the message of a JavaScript confirmation dialog generated during the previous action.
storeElementIndex (locator, variableName)	Get the relative index of an element to its parent (starting from 0).
storeLocation (variableName)	Gets the absolute URL of the current page.
storeSelectedIds (selectLocator, variableName)	Gets all element IDs for selected options in the specified select or multi-select element.

storeSelectedIndex (selectLocator, variableName)	Gets index (option number, starting at 0) for selected option in the specified select element.
storeSelectedLabel (selectLocator, variableName)	Gets label (visible text) for selected option in the specified select element..
storeSelectedValue (selectLocator, variableName)	Gets value (value attribute) for selected option in the specified select element.
storeSelectOptions (selectLocator, variableName)	Gets all labels in the specified select drop-down.
storeTable (tableCellAddress, variableName)	Gets the text from a cell of a table. The cellAddress syntax: tableLocator.row.column, where row and column start at 0.
storeText (locator, variableName)	Gets the text of an element. This works for any element that contains text.
storeTitle (variableName)	Gets the title of the current page.
storeValue (locator, variableName)	Gets the (whitespace-trimmed) value of an

	input field.
storeChecked (locator, variableName)	Gets whether a toggle-button (checkbox/radio) is checked.
storeElementPresent (locator, variableName)	Verifies that the specified element is somewhere on the page.
storeTextPresent (pattern, variableName)	Verifies that the specified text pattern appears somewhere on the rendered page shown to the user.
storeVisible (locator, variableName)	Determines if the specified element is visible.

Assertions

Assertions enable us to verify the state of an application and compares against the expected. It is used in 3 modes, viz. - "assert", "verify", and "waitFor". For example, "verify if an item from the dropdown is selected".

The following table lists the Selenium assertions that are used very frequently, however the list is not exhaustive.

Command/Syntax	Description
waitForErrorOnNext (message)	Waits for error; used with the accessor assertErrorOnNext.

verifySelected (selectLocator, optionLocator)	Verifies that the selected option of a drop-down satisfies the optionSpecifier.
waitForSelected (selectLocator, optionLocator)	Waits for getting the option selected; used with the accessor assertSelected.
waitForNotSelected (selectLocator, optionLocator)	Waits for not getting the option selected; used with the accessor assertSelected.
verifyAlert (pattern)	Verifies the alert text; used with the accessor storeAlert.
waitForAlert (pattern)	Waits for the alert; used with the accessor storeAlert.
verifyAllButtons (pattern)	Verifies the button; used with the accessor storeAllButtons.
waitForAllButtons (pattern)	Waits for the button to load; used with the accessor storeAllButtons.
verifyAllLinks (pattern)	Verifies all links; used with the accessor storeAllLinks.

waitForAllLinks (pattern)	Waits for all links; used with the accessor storeAllLinks.
verifyAllWindowIds (pattern)	Verifies the window id; used with the accessor storeAllWindowIds.
waitForAllWindowIds (pattern)	Waits the window id; used with the accessor storeAllWindowIds.
verifyAttribute (attributeLocator, pattern)	Verifies an attribute of an element; used with the accessor storeAttribute.
waitForAttribute (attributeLocator, pattern)	Waits for an attribute of an element; used with the accessor storeAttribute.
verifyBodyText(pattern)	Verifies the body text; used with the accessor storeBodyText.
waitForBodyText(pattern)	Waits for the body text; used with the accessor storeBodyText.
waitForConfirmation(pattern)	Waits for confirmation; used with the accessor storeConfirmationPresent

Locators

Element Locators help Selenium to identify the HTML element the command refers to. All these locators can be identified with the help of FirePath and FireBug plugin of Mozilla. Please refer the Environment Setup chapter for details.

- **identifier=id** Select the element with the specified "id" attribute and if there is no match, select the first element whose @name attribute is id.
- **id=id** Select the element with the specified "id" attribute.
- **name=name** Select the first element with the specified "name" attribute
- **dom=javascriptExpression** Selenium finds an element by evaluating the specified string that allows us to traverse through the HTML Document Object Model using JavaScript. Users cannot return a value but can evaluate as an expression in the block.
- **xpath=xpathExpression** Locate an element using an XPath expression.
- **link=textPattern** Select the link element (within anchor tags) which contains text matching the specified pattern.
- **css=cssSelectorSyntax** Select the element using css selector.

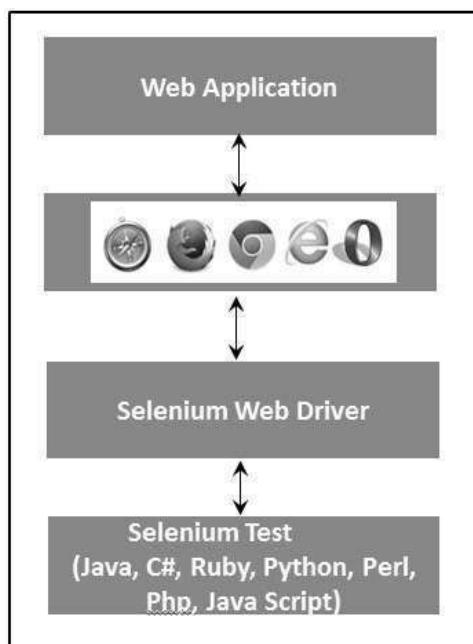
6. WEBDRIVER

WebDriver is a tool for automating testing web applications. It is popularly known as Selenium 2.0. WebDriver uses a different underlying framework, while Selenium RC uses JavaScript Selenium-Core embedded within the browser which has got some limitations. WebDriver interacts directly with the browser without any intermediary, unlike Selenium RC that depends on a server. It is used in the following context:

- Multi-browser testing including improved functionality for browsers which is not well-supported by Selenium RC (Selenium 1.0).
- Handling multiple frames, multiple browser windows, popups, and alerts.
- Complex page navigation.
- Advanced user navigation such as drag-and-drop.
- AJAX-based UI elements.

Architecture

WebDriver is best explained with a simple architecture diagram as shown below.



Selenium RC Vs WebDriver

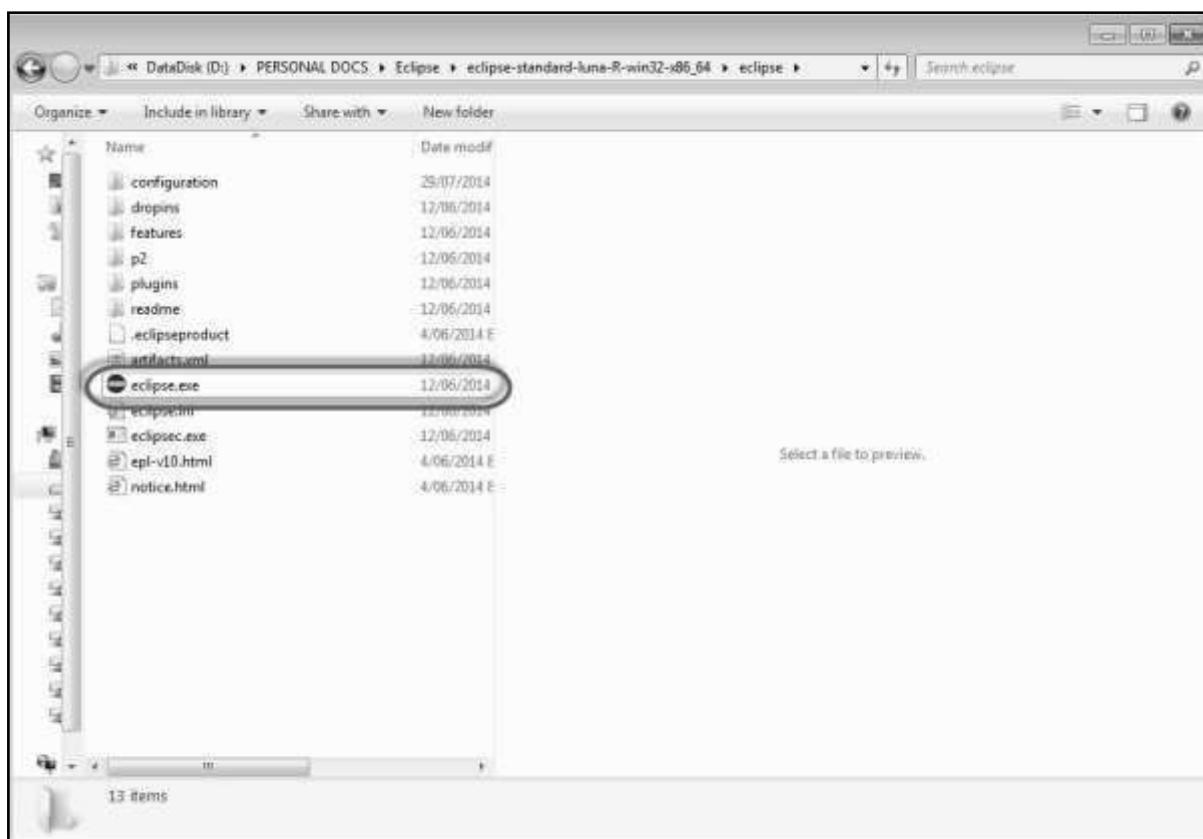
Selenium RC	Selenium WebDriver
The architecture of Selenium RC is complicated, as the server needs to be up and running before starting a test.	WebDriver's architecture is simpler than Selenium RC, as it controls the browser from the OS level.
Selenium server acts as a middleman between the browser and Selenese commands.	WebDriver interacts directly with the browser and uses the browser's engine to control it.
Selenium RC script execution is slower, since it uses a Javascript to interact with RC.	WebDriver is faster, as it interacts directly with the browser.
Selenium RC cannot support headless execution, as it needs a real browser to work with.	WebDriver can support the headless execution.
It's a simple and small API.	Complex and a bit large API as compared to RC.
Less object-oriented API.	Purely object-oriented API.
Cannot test mobile Applications.	Can test iPhone/Android applications.

Scripting using WebDriver

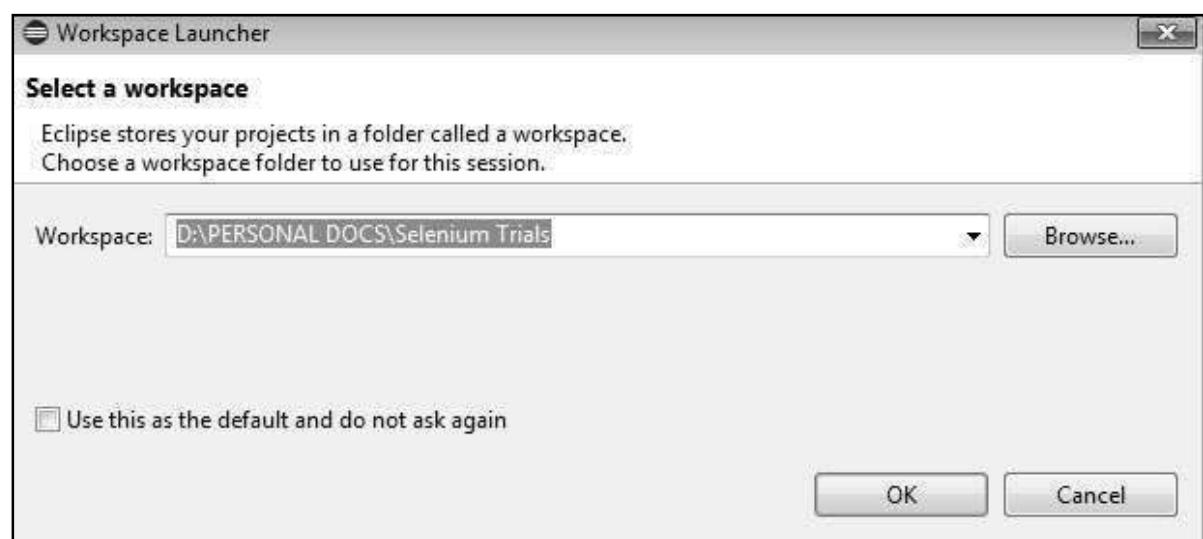
Let us understand how to work with WebDriver. For demonstration, we would use <http://www.calculator.net/>. We will perform a "Percent Calculator" which is

located under "Math Calculator". We have already downloaded the required WebDriver JAR's. Refer the chapter "Environmental Setup" for details.

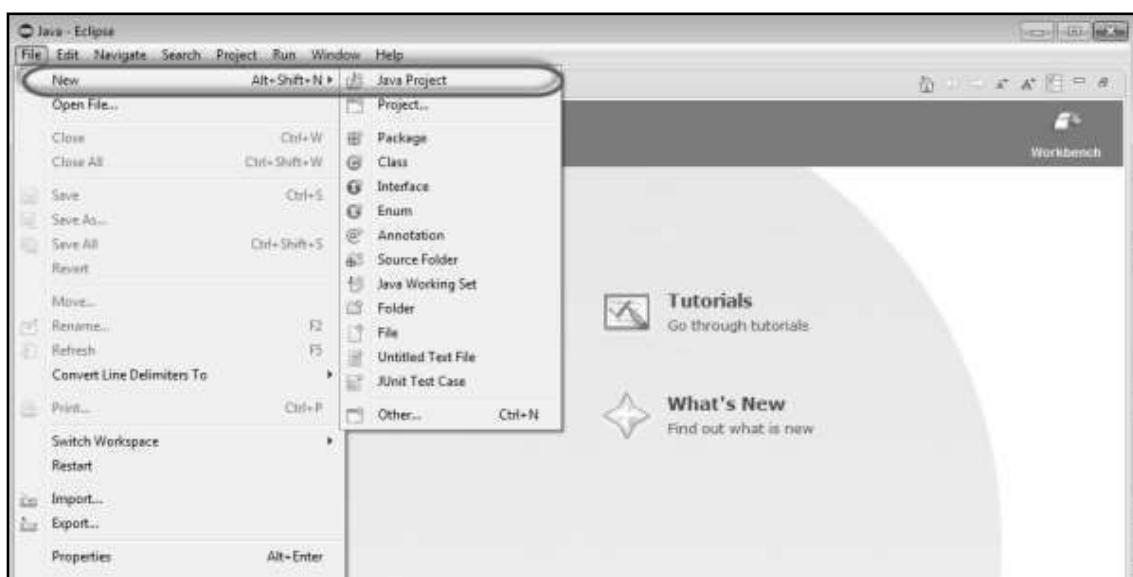
Step 1 : Launch "Eclipse" from the Extracted Eclipse folder.



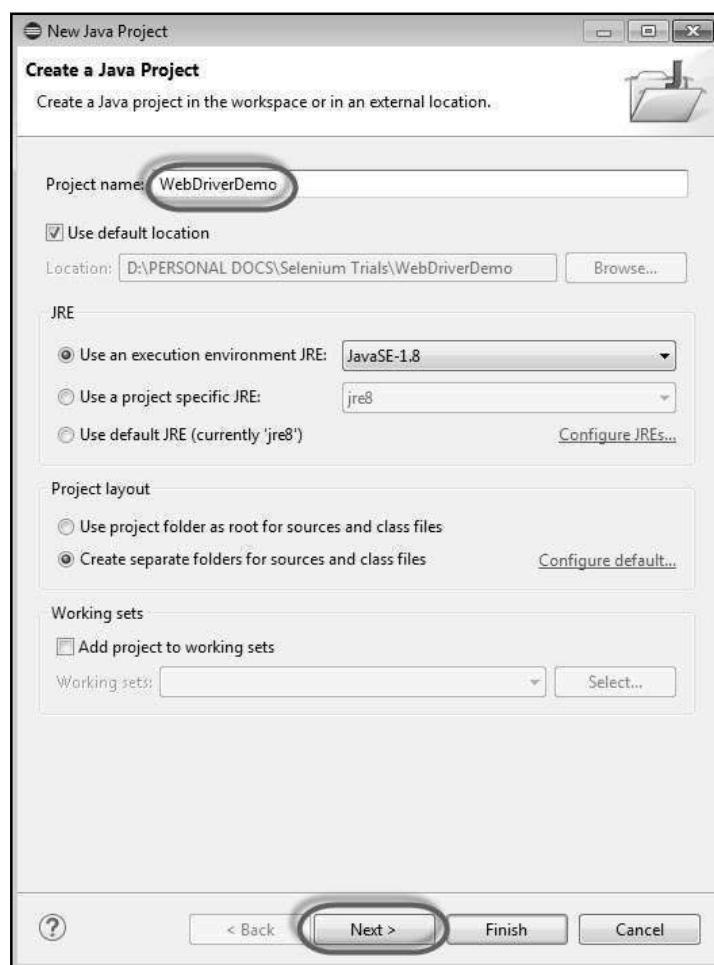
Step 2 : Select the Workspace by clicking the 'Browse' button.



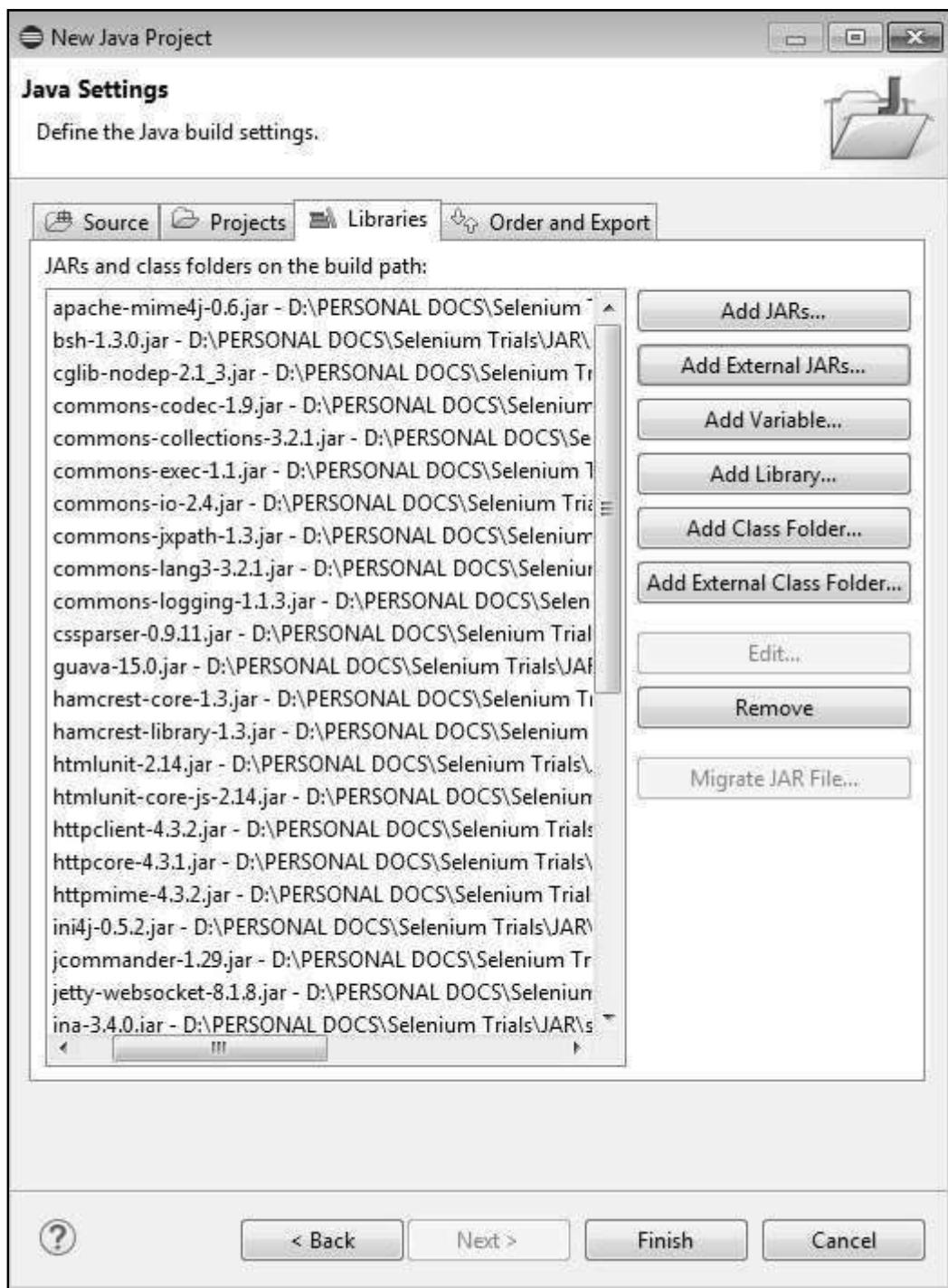
Step 3 : Now create a 'New Project' from 'File' menu.



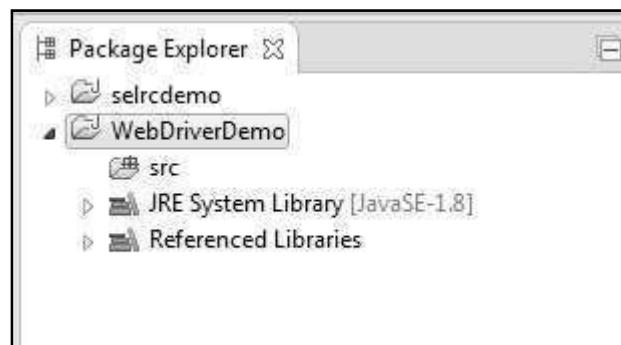
Step 4 : Enter the Project Name and Click 'Next'.



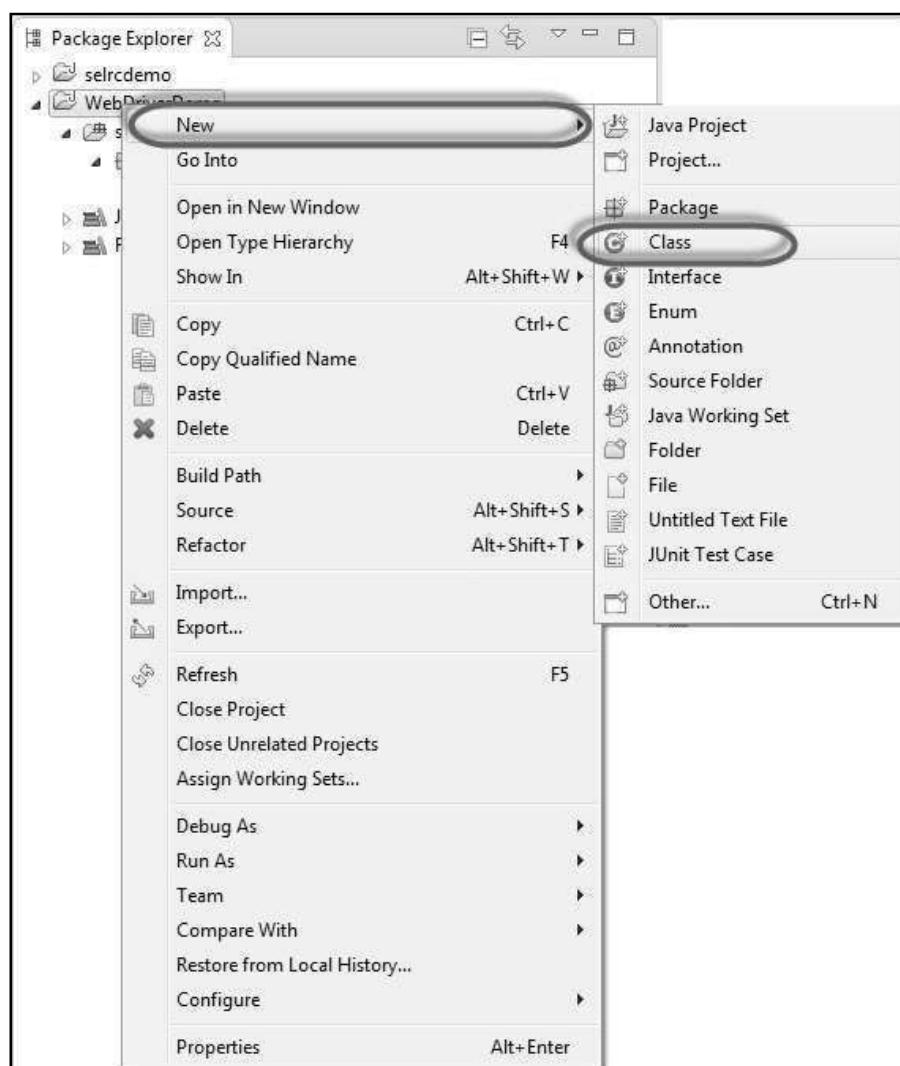
Step 5 : Go to Libraries Tab and select all the JAR's that we have downloaded. Add reference to all the JAR's of Selenium WebDriver Library folder and also selenium-java-2.42.2.jar and selenium-java-2.42.2-srscs.jar.



Step 6 : The Package is created as shown below.



Step 7 : Now right-click on the package and select 'New' >> 'Class' to create a 'Class'.



Step 8 : Now name the class and make it the main function.



Step 9 : The class outline is shown as below.

```

Package Explorer webdriverdemo.java
selcdemo
WebDriverDemo
src
  (default package)
    webdriverdemo.java
JRE System Library [JavaSE-1.8]
Referenced Libraries
  
```

```

1 public class webdriverdemo {
2
3     public static void main(String[] args) {
4         // TODO Auto-generated method stub
5
6     }
7
8 }
9
10
  
```

Step 10 : Now it is time to code. The following script is easier to understand, as it has comments embedded in it to explain the steps clearly. Please take a look at the chapter "Locators" to understand how to capture object properties.

```

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;

public class webdriverdemo
{
    public static void main(String[] args)
    {
        WebDriver driver = new FirefoxDriver();

        // Puts an Implicit wait, Will wait for 10 seconds
        // before throwing exception
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

        // Launch website
        driver.navigate().to("http://www.calculator.net/");

        // Maximize the browser
        driver.manage().window().maximize();

        // Click on Math Calculators
        driver.findElement(By.xpath(".//*[@id='menu']/div[3]/a")).click();
    }
}
  
```

```

// Click on Percent Calculators
driver.findElement(By.xpath("//*[@id='menu']/div[4]/div[3]/a")).click();

// Enter value 10 in the first number of the percent Calculator
driver.findElement(By.id("cpar1")).sendKeys("10");

// Enter value 50 in the second number of the percent Calculator
driver.findElement(By.id("cpar2")).sendKeys("50");

// Click Calculate Button
driver.findElement(By.xpath("//*[@id='content']/table/tbody
/tr/td[2]/input")).click();

// Get the Result Text based on its xpath
String result =
driver.findElement(By.xpath("//*[@id='content']/p[2]/span/font/b"))
.getText();

//Print a Log In message to the screen
System.out.println(" The Result is " + result);

//Close the Browser.
driver.close();
}
}

```

Step 11 : The output of the above script would be printed in Console.



Most Used Commands

The following table lists some of the most frequently used commands in WebDriver along with their syntax.

Command	Description
driver.get("URL")	To navigate to an application.
element.sendKeys("inputtext")	Enter some text into an input box.
element.clear()	Clear the contents from the input box.
select.deselectAll()	Deselect all OPTIONS from the first SELECT on the page.
select.selectByVisibleText("some text")	Select the OPTION with the input specified by the user.
driver.switchTo().window("windowName")	Move the focus from one window to another.
driver.switchTo().frame("frameName")	Switch from frame to frame.
driver.switchTo().alert()	Helps in handling alerts.
driver.navigate().to("URL")	Navigate to the URL.

driver.navigate().forward()	To navigate forward.
driver.navigate().back()	To navigate back.
driver.close()	Closes the current browser associated with the driver.
driver.quit()	Quits the driver and closes all the associated window of that driver.
driver.refresh()	Refreshes the current page.

7. LOCATORS

Locating elements in Selenium WebDriver is performed with the help of `findElement()` and `findElements()` methods provided by `WebDriver` and `WebElement` class.

- `findElement()` returns a `WebElement` object based on a specified search criteria or ends up throwing an exception if it does not find any element matching the search criteria.
- `findElements()` returns a list of `WebElements` matching the search criteria. If no elements are found, it returns an empty list.

The following table lists all the Java syntax for locating elements in Selenium WebDriver.

Method	Syntax	Description
By ID	<code>driver.findElement(By.id(<element ID>))</code>	Locates an element using the ID attribute
By name	<code>driver.findElement(By.name(<element name>))</code>	Locates an element using the Name attribute
By class name	<code>driver.findElement(By.className(<element class>))</code>	Locates an element using the Class attribute
By tag name	<code>driver.findElement(By.tagName(<htmltagname>))</code>	Locates an element using the

		HTML tag
By link text	driver.findElement(By.linkText(<linktext>))	Locates a link using link text
By partial link text	driver.findElement(By.partialLinkText(<linktext>))	Locates a link using the link's partial text
By CSS	driver.findElement(By.cssSelector(<css selector>))	Locates an element using the CSS selector
By XPath	driver.findElement(By.xpath(<xpath>))	Locates an element using XPath query

Locators Usage

Now let us understand the practical usage of each of the locator methods with the help of <http://www.calculator.net>

By ID

Here an object is accessed with the help of IDs. In this case, it is the ID of the text box. Values are entered into the text box using the sendkeys method with the help of ID(cdensity).

The screenshot shows the 'Mass Calculator' page on Calculator.net. A context menu is open over the density input field, highlighting the element's ID ('cdensity'). The browser's developer tools are visible at the bottom, showing the full HTML structure of the density input field.

```

<input id="cdensity" type="text" style="text-align: right;" value="8900" size="5" name="cdensity">

```

```
driver.findElement(By.id("cdensity")).sendKeys("10");
```

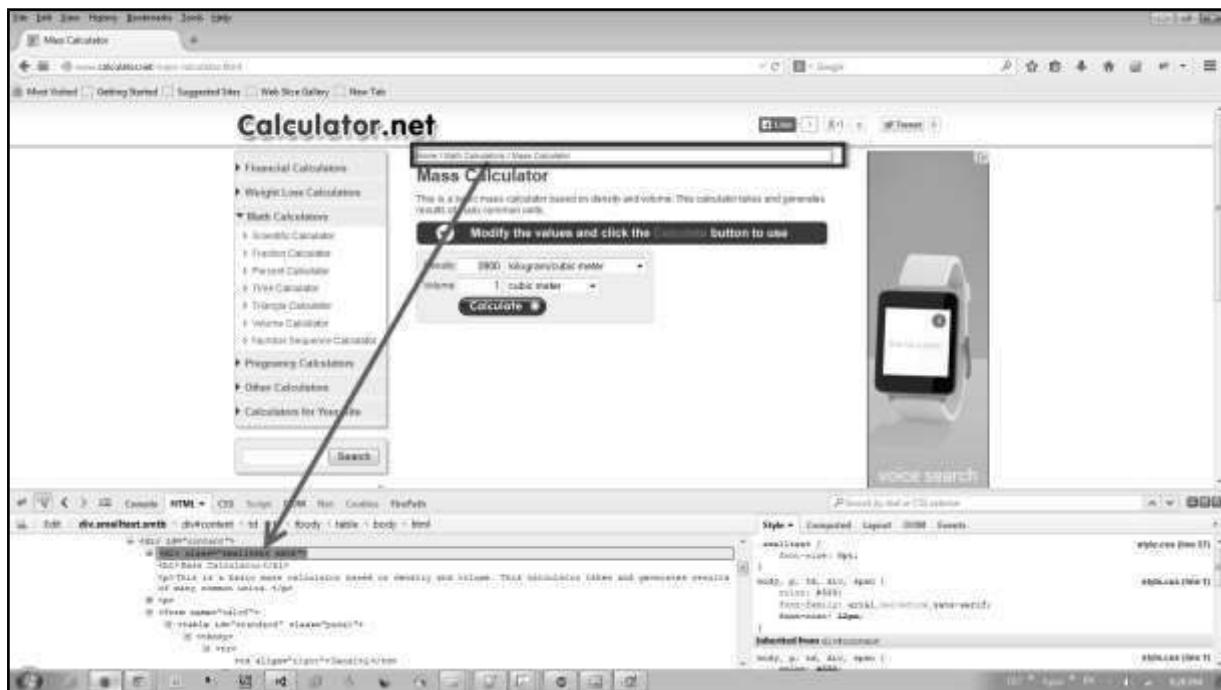
By Name

Here an object is accessed with the help of names. In this case, it is the name of the text box. Values are entered into the text box using the sendkeys method with the help of ID(cdensity).

```
driver.findElement(By.name("cdensity")).sendKeys("10");
```

By Class Name

Here an object is accessed with the help of Class Names. In this case, it is the Class name of the WebElement. The Value can be accessed with the help of the gettext method.



```
List<WebElement> byclass = driver.findElements(By.className("smalltext smtb"));
```

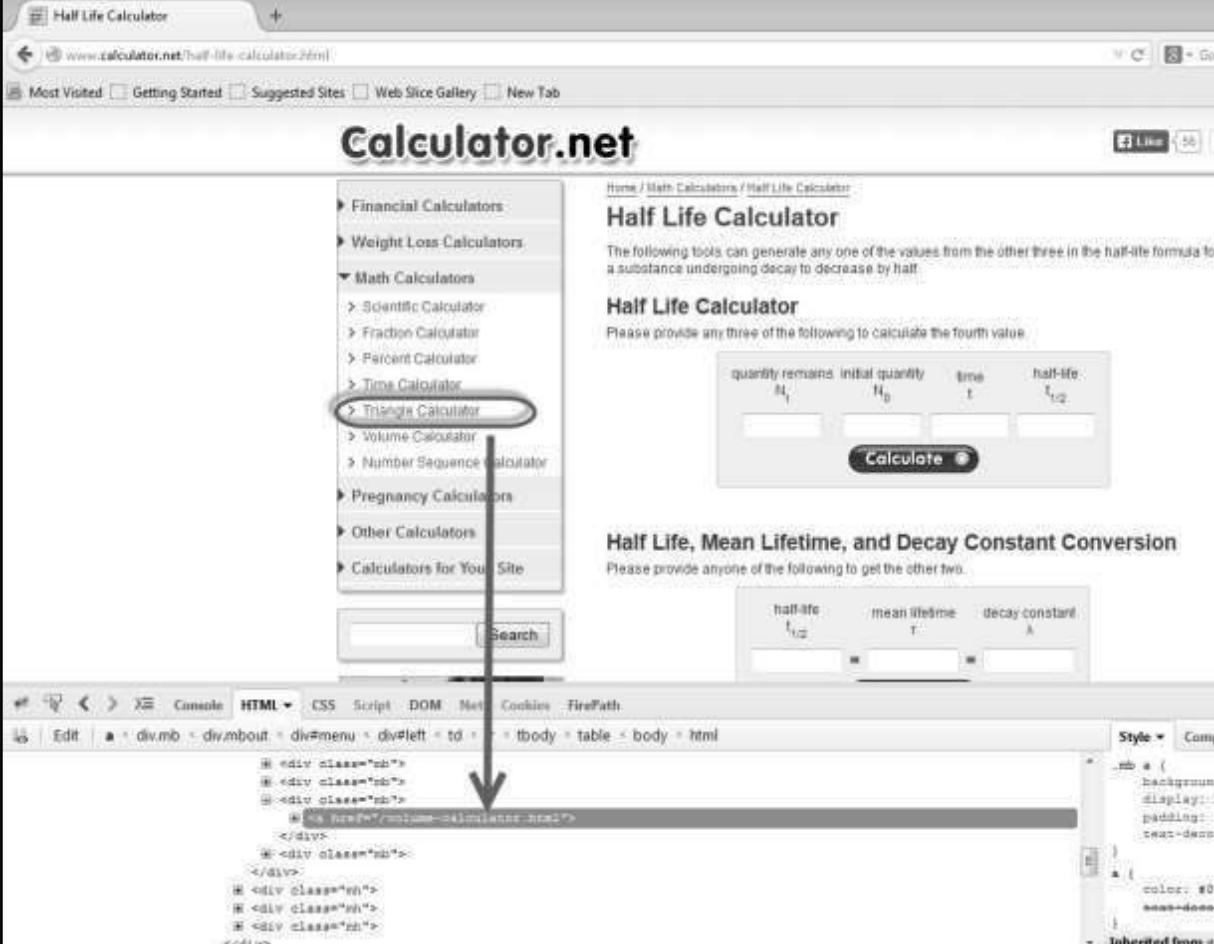
By Tag Name

The DOM Tag Name of an element can be used to locate that particular element in the WebDriver. It is very easy to handle tables with the help of this method. Take a look at the following code.

```
WebElement table = driver.findElement(By.id("calctable"));
List<WebElement> row = table.findElements(By.tagName("tr"));
int rowcount = row.size();
```

By Link Text

This method helps to locate a link element with matching visible text.

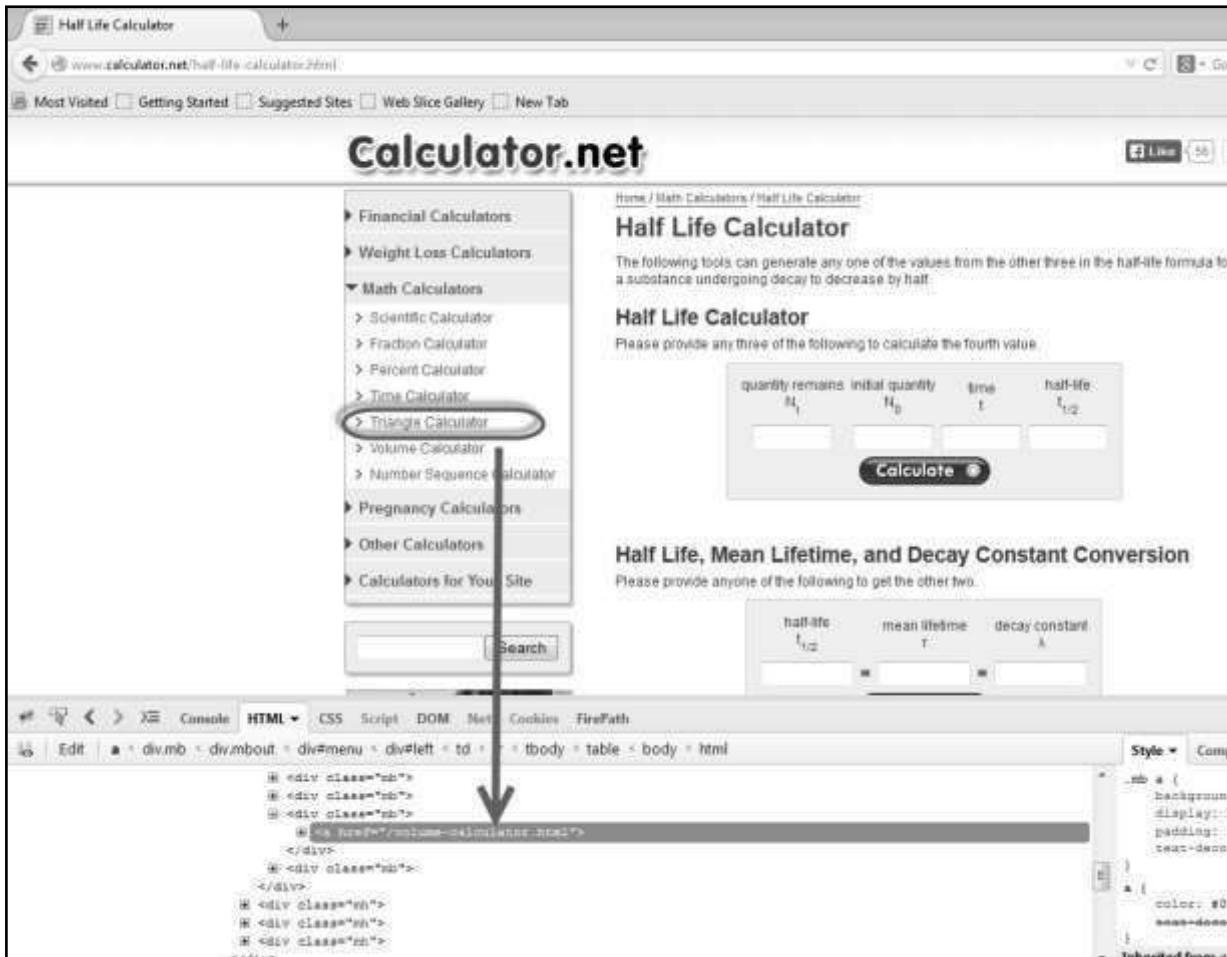


The screenshot shows a browser window for 'Half Life Calculator' on the 'calculator.net' website. The sidebar menu is open, showing various calculator categories. The 'Volume Calculator' link under the 'Triangle Calculator' section is highlighted with a red oval and a black arrow pointing to it from the bottom. The main content area displays the 'Half Life Calculator' tool and its conversion formulas. Below the screenshot, a code snippet in a box illustrates the Selenium command to click on this specific link:

```
driver.findElement(By.linkText("Volume")).click();
```

By Partial Link Text

This method helps locate a link element with partial matching visible text.



```
driver.findElements(By.partialLinkText("Volume")).click();
```

By CSS

The CSS is used as a method to identify the webobject, however NOT all browsers support CSS identification.

```
WebElement loginButton =  
driver.findElement(By.cssSelector("input.login"));
```

By XPath

XPath stands for XML path language. It is a query language for selecting nodes from an XML document. XPath is based on the tree representation of XML

documents and provides the ability to navigate around the tree by selecting nodes using a variety of criteria.

The screenshot shows the 'Half Life Calculator' page on calculator.net. The left sidebar has a 'Math Calculators' section expanded, showing various calculators like Scientific, Fraction, Percent, Time, Triangle, Volume, Number Sequence, Pregnancy, Other, and Calculators for Your Site. Below the sidebar is a search bar. The main content area is titled 'Half Life Calculator' and contains instructions: 'The following tools can generate any one of the values from the other three in the half-life formula for a substance undergoing decay to decrease by half'. It features two input forms. The first form is for calculating quantity remains (N_1) given initial quantity (N_0), time (t), and half-life ($t_{1/2}$). The second form is for calculating half-life ($t_{1/2}$) given mean lifetime (\bar{t}) and decay constant (k). At the bottom, there's a developer toolbar with tabs for Console, HTML, CSS, Script, DOM, Net, Cookies, and FirePath. The FirePath tab is active, showing the DOM tree with the path `//[@id='content']/table[1]/tbody/tr/td/table/tbody/tr[2]/td[1]/input`. The input field for quantity remains is highlighted with a red border in both the browser and the DOM tree.

```
driver.findElement(By.xpath("//*[@id='content']/table[1]/tbody/tr/td/table/tbody/tr[2]/td[1]/input")).sendKeys("100");
```

8. INTERACTIONS

User Interactions

Selenium WebDriver is the most frequently used tool among all the tools available in the Selenium tool set. Therefore it is important to understand how to use Selenium to interact with web apps. In this module, let us understand how to interact with GUI objects using Selenium WebDriver.

We need to interact with the application using some basic actions or even some advanced user action by developing user-defined functions for which there are no predefined commands.

Listed below are the different kinds of actions against those GUI objects:

- Text Box Interaction
- Radio Button Selection
- Check Box Selection
- Drop Down Item Selection
- Synchronization
- Drag & Drop
- Keyboard Actions
- Mouse Actions
- Multi Select
- Find All Links

Text Box Interaction

In this section, we will understand how to interact with text boxes. We can put values into a text box using the 'sendKeys' method. Similarly, we can also retrieve text from a text box using the getAttribute("value") command. Take a look at the following example.

The screenshot shows a dropdown menu on the left and a main content area on the right. The content area has a title 'Percent Calculator' and a form with three input fields and a 'Calculate' button. Below the form is some explanatory text and examples. A large black arrow points from the dropdown menu towards the Firebug-style DOM viewer at the bottom.

```

body < table.panel < div#content < td < tr < tbody < table < body < html
  ↘ <td valign="middle" align="center">
    <input id="cpar1" type="text" style="text-align: right;" value="" size="10" name="cpar1">
    % of
    <input id="cpar2" type="text" style="text-align: right;" value="" size="10" name="cpar2">
    =
    <input id="cpar3" type="text" style="text-align: right;" value="" size="10" name="cpar3">
  </td>

```

```

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;

public class webdriverdemo
{
    public static void main(String[] args) throws InterruptedException
    {
        WebDriver driver = new FirefoxDriver();

        // Puts an Implicit wait, Will wait for 10 seconds
        // before throwing exception
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

        // Launch website
        driver.navigate().to("http://www.calculator.net")
    }
}

```

```
/percent-calculator.html");

// Maximize the browser
driver.manage().window().maximize();

// Enter value 10 in the first number of the percent Calculator
driver.findElement(By.id("cpar1")).sendKeys("10");

Thread.sleep(5000);

// Get the text box from the application
String result =
driver.findElement(By.id("cpar1")).getAttribute("value");

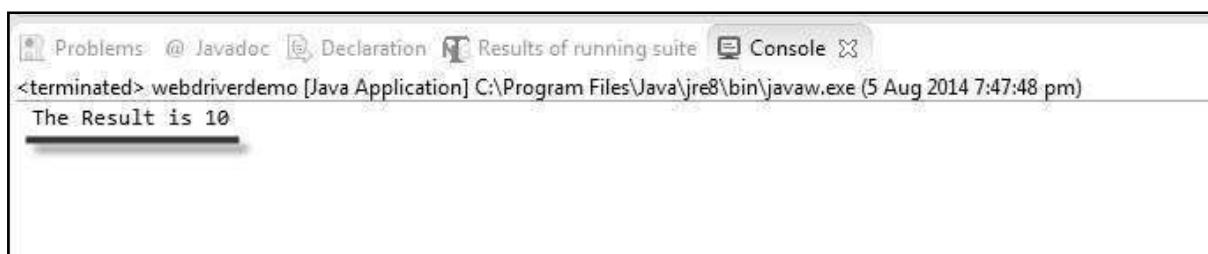
// Print a Log In message to the screen
System.out.println(" The Result is " + result);

// Close the Browser
driver.close();
}

}
```

Output

The output of the above script is displayed as shown below.



```
Problems @ Javadoc Declaration Results of running suite Console 
<terminated> webdriverdemo [Java Application] C:\Program Files\Java\jre8\bin\javaw.exe (5 Aug 2014 7:47:48 pm)
The Result is 10
```

Radio Button Interaction

In this section, we will understand how to interact with Radio Buttons. We can select a radio button option using the 'click' method and unselect using the same 'click' method.

Let us understand how to interact with radio buttons using <http://www.calculator.net/mortgage-payoff-calculator.html>. We can also check if a radio button is selected or enabled.

Home / Financial Calculators / Mortgage Payoff Calculator

Mortgage Payoff Calculator

The Mortgage Payoff Calculator is used to determine your current mortgage payoff schedule based on a fixed interest. You can choose to pay off your loan altogether, and see the costs, or view the effects of an additional monthly payment. This calculator can also give out the interest saved and the new loan schedule.

Modify the values and click the Calculate button to use

Existing Loan Amount	\$ 300000
Existing Loan Term	30 years
Interest Rate	6 %
Time Remaining	25 years 0 months
<input checked="" type="radio"/> Payoff altogether <input type="radio"/> Payoff with additional \$ 500 per month	
Calculate	

Summary

With additional \$500.00 monthly payment, you can payoff the loan in 15 years 8.00 months. It is 9 years 4.00 months earlier than the old schedule. You will save \$108,886.04 of interest.

```

SS Script DOM Net Cookies FirePath
body <table#cal.le.panel> div.leftinput <div#content <td <tr <tbody <table <body <html

r>
<td colspan="3">
  <input id="cpayoff1" type="radio" value="together" name="cmonthoryear">
  Payoff altogether
  <br>
  <input id="cpayoff2" type="radio" checked="" value="monthly" name="cmonthoryear">
  Payoff with additional $
  <input id="cadditional" type="text" style="text-align: right;" value="500" size="2" name="cadditional">
  per month

```

```
import java.util.concurrent.TimeUnit;  
import org.openqa.selenium.*;  
import org.openqa.selenium.firefox.FirefoxDriver;  
  
public class webdriverdemo  
{  
    public static void main(String[] args) throws InterruptedException  
    {
```

```

WebDriver driver = new FirefoxDriver();

// Puts an Implicit wait, will wait for 10 seconds
// before throwing exception
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

// Launch website
driver.navigate().to("http://www.calculator.net
/mortgage-payoff-calculator.html");
driver.manage().window().maximize();

// Click on Radio Button
driver.findElement(By.id("cpayoff1")).click();

System.out.println("The Output of the IsSelected " +
driver.findElement(By.id("cpayoff1")).isSelected());

System.out.println("The Output of the IsEnabled " +
driver.findElement(By.id("cpayoff1")).isEnabled());

System.out.println("The Output of the IsDisplayed " +
driver.findElement(By.id("cpayoff1")).isDisplayed());

driver.close();

// Close the Browser.
driver.close();
}
}

```

Output

Upon execution, the radio button is selected and the output of the commands are displayed in the console.

```

Problems @ Javadoc Declaration Results of running suite Console 
<terminated> webdriverdemo [Java Application] C:\Program Files\Java\jre8\bin\javaw.exe (5 Aug 2014 12:03:03 pm)
The Output of the IsSelected true
The Output of the IsEnabled true
The Output of the IsDisplayed true

```

Check BoxInteraction

In this section, we will understand how to interact with Check Box. We can select a check box using the 'click' method and uncheck using the same 'click' method.

Let us understand how to interact with a check box using <http://www.calculator.net/mortgage-calculator.html>. We can also check if a check box is selected/enabled/visible.

The screenshot shows a mortgage calculator page with various input fields and a summary table. On the left, there's a sidebar with a list of financial calculators. The main area has a heading 'Mortgage Calculator' and a sub-instruction 'Modify the values and click the Calculate button to use'. Below this are input fields for Home Price (\$300,000), Down Payment (20%), Loan Term (30 years), and Interest Rate (4.16%). A checkbox labeled 'Include Optionals Below' is checked. To its right is a summary table with columns for 'Monthly' and 'Total' payments. At the bottom, there's a 'Calculate' button and a table of additional costs like Property Taxes, Home Insurance, etc. A red arrow points from the 'Include Optionals Below' checkbox to the corresponding line in the DOM tree below.

```

<td>
    <br>
    <label for="caddoptional">
        <input id="caddoptional" type="checkbox" checked="" value="1" name="caddoptional">
        <b>Include Optionals Below</b>
    </label>
</td>

```

```

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;

```

```
public class webdriverdemo
{
    public static void main(String[] args) throws InterruptedException
    {
        WebDriver driver = new FirefoxDriver();

        // Puts an Implicit wait, Will wait for 10 seconds
        // before throwing exception
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

        // Launch website
        driver.navigate().to("http://www.calculator.net
        /mortgage-calculator.html");
        driver.manage().window().maximize();

        // Click on check box
        driver.findElement(By.id("caddoptional")).click();

        System.out.println("The Output of the IsSelected " +
        driver.findElement(By.id("caddoptional")).isSelected());

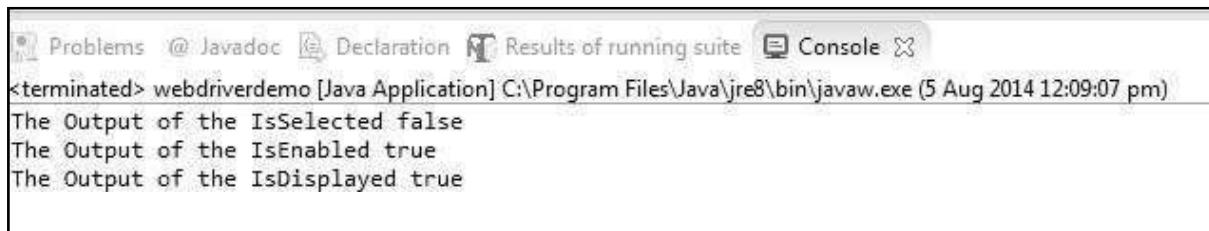
        System.out.println("The Output of the IsEnabled " +
        driver.findElement(By.id("caddoptional")).isEnabled());

        System.out.println("The Output of the IsDisplayed " +
        driver.findElement(By.id("caddoptional")).isDisplayed());

        driver.close();
    }
}
```

Output

Upon execution, the check box is unchecked after the click command (as it was checked by default) and the output of the commands are displayed in the console.



```

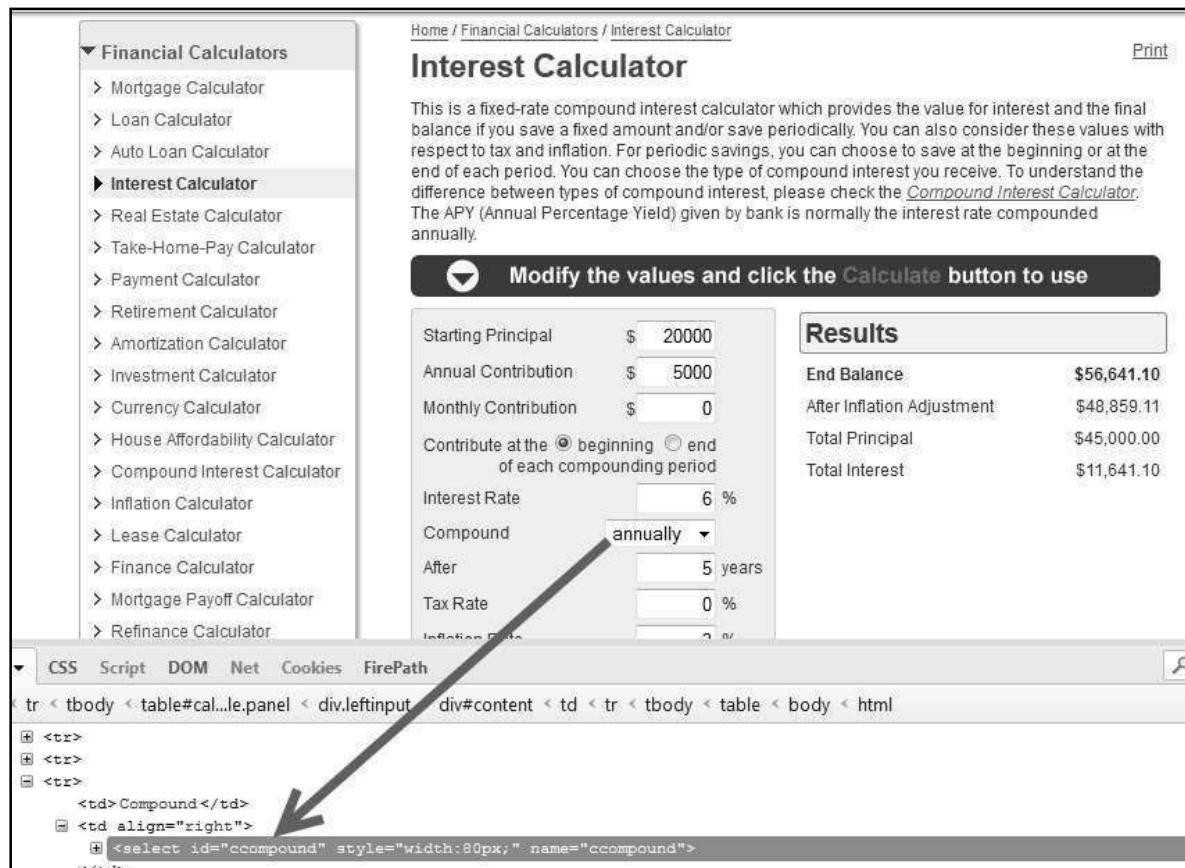
Problems @ Javadoc Declaration Results of running suite Console
<terminated> webdriverdemo [Java Application] C:\Program Files\Java\jre8\bin\javaw.exe (5 Aug 2014 12:09:07 pm)
The Output of the IsSelected false
The Output of the IsEnabled true
The Output of the IsDisplayed true

```

Dropdown Interaction

In this section, we will understand how to interact with Dropdown Boxes. We can select an option using 'selectByVisibleText' or 'selectByIndex' or 'selectByValue' methods.

Let us understand how to interact with a dropdown box using <http://www.calculator.net/interest-calculator.html>. We can also check if a dropdown box is selected/enabled/visible.



Interest Calculator

This is a fixed-rate compound interest calculator which provides the value for interest and the final balance if you save a fixed amount and/or save periodically. You can also consider these values with respect to tax and inflation. For periodic savings, you can choose to save at the beginning or at the end of each period. You can choose the type of compound interest you receive. To understand the difference between types of compound interest, please check the [Compound Interest Calculator](#). The APY (Annual Percentage Yield) given by bank is normally the interest rate compounded annually.

Modify the values and click the Calculate button to use

Results	
Starting Principal	\$ 20000
Annual Contribution	\$ 5000
Monthly Contribution	\$ 0
Contribute at the <input checked="" type="radio"/> beginning <input type="radio"/> end of each compounding period	
Interest Rate	6 %
Compound	annually ▾
After	5 years
Tax Rate	0 %
Inflation Rate	0 %

FirePath

```

tr < tbody < table#cal...le.panel < div.leftinput < div#content < td < tr < tbody < table < body < html
  ↗ <tr>
  ↗ <tr>
  ↗ <tr>
    ↗ <td>Compound</td>
    ↗ <td align="right">
      ↗ <select id="cccompound" style="width:80px;" name="cccompound">
        ↗ <option value="annually">annually</option>
        ↗ <option value="semi-annually">semi-annually</option>
        ↗ <option value="quarterly">quarterly</option>
        ↗ <option value="monthly">monthly</option>
        ↗ <option value="bi-monthly">bi-monthly</option>
        ↗ <option value="quarterly">quarterly</option>
        ↗ <option value="bi-annually">bi-annually</option>
        ↗ <option value="annual">annual</option>
      </select>
    </td>
  </tr>

```

```

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;

public class webdrivedemo
{
    public static void main(String[] args) throws InterruptedException
    {
        WebDriver driver = new FirefoxDriver();

        // Puts an Implicit wait, Will wait for 10 seconds
        // before throwing exception
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

        // Launch website
        driver.navigate().to("http://www.calculator.net
        /interest-calculator.html");
        driver.manage().window().maximize();

        // Selecting an item from Drop Down List Box
        Select dropdown =
        new Select(driver.findElement(By.id("ccom pound")));

        dropdown.selectByVisibleText("continuously");

        // you can also use dropdown.selectByIndex(1) to
        // select second element as index starts with 0.
        // You can also use dropdown.selectByValue("annually");

        System.out.println("The Output of the IsSelected " +
    
```

```

driver.findElement(By.id("ccompound")).isSelected();

System.out.println("The Output of the IsEnabled " +
driver.findElement(By.id("ccompound")).isEnabled();

System.out.println("The Output of the IsDisplayed " +
driver.findElement(By.id("ccompound")).isDisplayed();

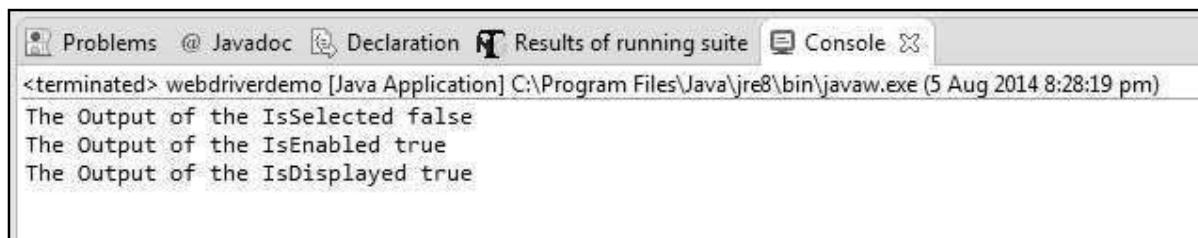
driver.close();

}
}

```

Output

Upon execution, the dropdown is set with the specified value and the output of the commands are displayed in the console.



The screenshot shows a Java application running in an IDE. The console tab displays the following output:

```

Problems @ Javadoc Declaration Results of running suite Console X
<terminated> webdriverdemo [Java Application] C:\Program Files\Java\jre8\bin\javaw.exe (5 Aug 2014 8:28:19 pm)
The Output of the IsSelected false
The Output of the IsEnabled true
The Output of the IsDisplayed true

```

Synchronization

To synchronize between script execution and application, we need to wait after performing appropriate actions. Let us look at the ways to achieve the same.

Thread.Sleep

Thread.Sleep is a static wait and it is not a good way to use in scripts, as it is sleep without condition.

```
Thread.Sleep(1000); //Will wait for 1 second.
```

Explicit Waits

An 'explicit wait' waits for a certain condition to occur before proceeding further. It is mainly used when we want to click or act on an object once it is visible.

```
WebDriver driver = new FirefoxDriver();
driver.get("Enter an URL");
WebElement DynamicElement = (new WebDriverWait(driver,
10)).until(ExpectedConditions.presenceOfElementLocated(By.id("DynamicElement")));

```

Implicit Wait

Implicit wait is used in cases where the WebDriver cannot locate an object immediately because of its unavailability. The WebDriver will wait for a specified implicit wait time and it will not try to find the element again during the specified time period.

Once the specified time limit is crossed, the WebDriver will try to search the element once again for one last time. Upon success, it proceeds with the execution; upon failure, it throws an exception.

It is a kind of global wait which means the wait is applicable for the entire driver. Hence, hardcoding this wait for longer time periods will hamper the execution time.

```
WebDriver driver = new FirefoxDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("Enter an URL");
WebElement DynamicElement = driver.findElement(By.id("DynamicElement"));

```

Fluent Wait

A FluentWait instance defines the maximum amount of time to wait for a condition to take place, as well as the frequency with which to check the existence of the object condition.

Let us say we will 60 seconds for an element to be available on the page, but we will check its availability once in every 10 seconds.

```
Wait wait = new FluentWait(driver)
    .withTimeout(60, SECONDS)
    .pollingEvery(10, SECONDS)
    .ignoring(NoSuchElementException.class);

WebElement dynamicElement = wait.until(new
Function<WebDriver, WebElement>()
{

```

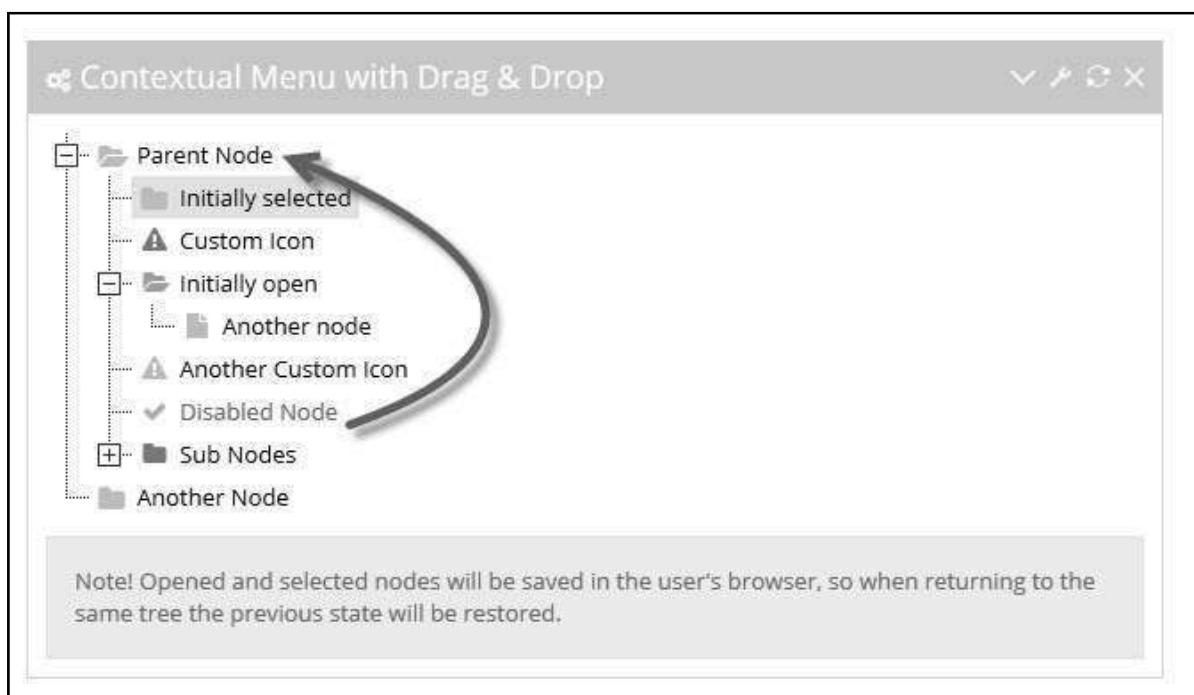
```
public WebElement apply(WebDriver driver)
{
    return driver.findElement(By.id("dynamicElement"));
}
```

Drag & Drop

As a tester, you might be in a situation to perform a 'Drag & drop' operation. We will perform a drag and drop operation by picking up a tree grid that is available for us on

http://www.keenthemes.com/preview/metronic/templates/admin/ui_tree.html.

In the example, we would like to drag an element 'Disable Node' from 'initially open' folder to 'Parent Node' Folder.



```
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.interactions.Action;
```

```

public class webdrivedemo
{
    public static void main(String[] args) throws InterruptedException
    {
        WebDriver driver = new FirefoxDriver();

        // Puts an Implicit wait, will wait for 10 seconds
        // before throwing exception
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

        // Launch website
        driver.navigate().to("http://www.keenthemes.com/preview/
metronic/templates/admin/ui_tree.html");
        driver.manage().window().maximize();

        WebElement From =
driver.findElement(By.xpath(".//*[@id='j3_7']/a"));

        WebElement To =
driver.findElement(By.xpath(".//*[@id='j3_1']/a"));

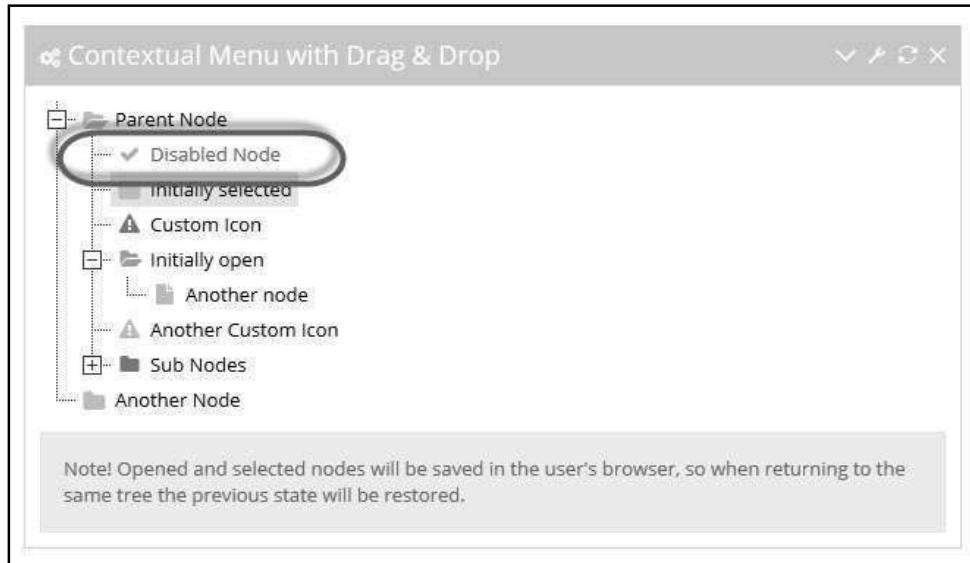
        Actions builder = new Actions(driver);
        Action dragAndDrop = builder.clickAndHold(From)
            .moveToElement(To)
            .release(To)
            .build();
        dragAndDrop.perform();

        driver.close();
    }
}

```

Output

After performing the drag-drop operation, the output would be as shown below.



Keyboard Actions

Given below are the methods to perform keyboard actions:

- **sendKeys** - Sends keys to the keyboard representation in the browser. Special keys that are not text, represented as Keys are recognized both as part of sequences of characters, or individually.
- **pressKey** - Press a key on the keyboard that is NOT text. The keys such as function keys "F1", "F2", "Tab", "Control", etc. If keyToPress is a sequence of characters, different driver implementations may choose to throw an exception or to read only the first character in the sequence.
- **releaseKey** - Release a key on the keyboard after executing the keypress event. It usually holds good for non-text characters.

Here are the syntax to call keyboard functions using Selenium WebDriver.

```
void sendKeys(java.lang.CharSequence keysToSend)
void pressKey(java.lang.CharSequence keyToPress)
void releaseKey(java.lang.CharSequence keyToRelease)
```

Mouse Actions

Listed below are some of the key mouse actions that one would come across in most of the applications:

- **Click** - Performs a Click. We can also perform a click based on coordinates.
- **contextClick** - Performs a context click/right-click on an element or based on the coordinates.
- **doubleClick** - Performs a double-click on the webelement or based on the coordinates. If left empty, it performs double-click on the current location.
- **mouseDown** - Performs a mouse-down action on an element or based on coordinates.
- **mouseMove** - Performs a mouse-move action on an element or based on coordinates.
- **mouseUp** - Releases the mouse usually followed by mouse-down and acts based on coordinates.

Here are the syntax to call mouse actions using Selenium WebDriver:

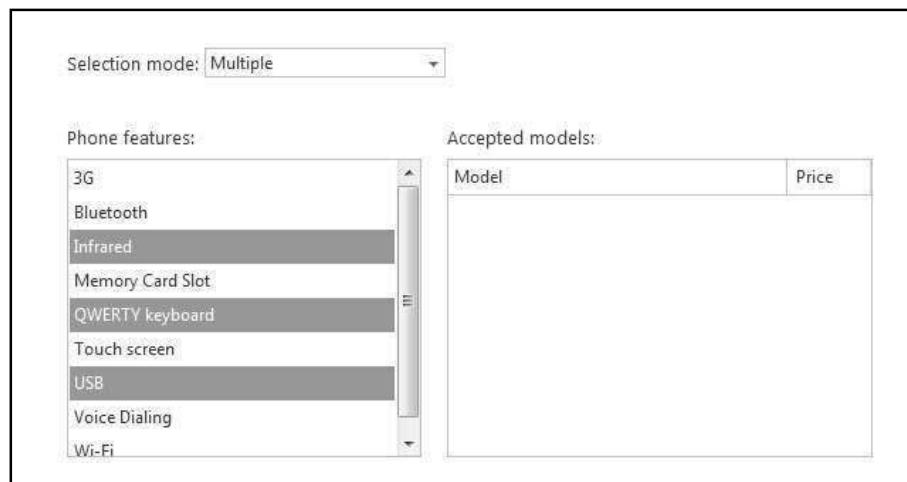
```
void click(WebElement onElement)
void contextClick(WebElement onElement)
void doubleClick(WebElement onElement)
void mouseDown(WebElement onElement)
void mouseUp(WebElement onElement)
void mouseMove(WebElement toElement)
void mouseMove(WebElement toElement, long xOffset, long yOffset)
```

Multi SelectAction

Sometimes we would be in a situation to select two or more items in a list box or text area. To understand the same, we would demonstrate multiple selection from the list using
['http://demos.devexpress.com/AspxEditorsDemos/ListEditors/MultiSelect.aspx'](http://demos.devexpress.com/AspxEditorsDemos/ListEditors/MultiSelect.aspx).

Example

Let us say, we want to select 3 items from this list as shown below:



Let us see how to code for this functionality:

```

import java.util.List;
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.interactions.Action;

public class webdrivedemo
{
    public static void main(String[] args) throws InterruptedException
    {
        WebDriver driver = new FirefoxDriver();

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

        driver.navigate().to("http://demos.devexpress.com
/aspxeditorsdemos/ListEditors/MultiSelect.aspx");

        //driver.manage().window().maximize();

        driver.findElement(By.id("ContentHolder_lbSelectionMode_I")).click();
        driver.findElement(By.id("ContentHolder_lbSelectionMode

```

```

    _DDD_L_LB1T0")).click();

    Thread.sleep(5000);

    // Perform Multiple Select
    Actions builder = new Actions(driver);
    WebElement select =
        driver.findElement(By.id("ContentHolder_lbFeatures_LBT"));
    List<WebElement> options = select.findElements(By.tagName("td"));
    System.out.println(options.size());
    Action multipleSelect = builder.keyDown(Keys.CONTROL)
        .click(options.get(2))
        .click(options.get(4))
        .click(options.get(6))
        .build();
    multipleSelect.perform();

    driver.close();

}

}

```

Output

Upon executing the script, the items would be selected as displayed above and the size of the list box would also be printed in the console.

The screenshot shows an IDE interface with several tabs at the top: Problems, @ Javadoc, Declaration, Results of running suite, and Console. The Console tab is active and displays the following text:

```

Problems: @ Javadoc Declaration Results of running suite Console
<terminated> webdriverdemo [Java Application] C:\Program Files\Java\jre8\bin\javaw.exe (6 Aug 2014 12:27:57 am)
Number of items in the List box is 9

```

Find All Links

Testers might be in a situation to find all the links on a website. We can easily do so by finding all elements with the Tag Name "a", as we know that for any link reference in HTML, we need to use "a" (anchor) tag.

Example

```
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
public class getalllinks
{
    public static void main(String[] args)
    {
        WebDriver driver = new FirefoxDriver();
        driver.navigate().to("http://www.calculator.net");
        java.util.List<WebElement> links =
        driver.findElements(By.tagName("a"));
        System.out.println("Number of Links in the Page is " +
        links.size());
        for (int i = 1; i<=links.size(); i=i+1)
        {
            System.out.println("Name of Link# " + i - +
            links.get(i).getText());
        }
    }
}
```

Output

The output of the script would be thrown to the console as shown below. Though there are 65 links, only partial output is shown below.

```
Problems @ Javadoc Declaration Console X
<terminated> webdriverdemo [Java Application] C:\Program Files\Java\jre8\bin\javaw.exe (30 Jul 2014 8:00:04 pm)
Number of Links in the Page is 65
Name of Link# 0 - calculator.net
Name of Link# 1 - put this calculator on your website
Name of Link# 2 - Financial
Name of Link# 3 - Mortgage
Name of Link# 4 - Loan
Name of Link# 5 - Auto Loan
Name of Link# 6 - Interest
Name of Link# 7 - Take-Home-Pay
Name of Link# 8 - Payment
Name of Link# 9 - Retirement
Name of Link# 10 - Amortization
Name of Link# 11 - Investment
Name of Link# 12 - House Affordability
Name of Link# 13 - Compound Interest
Name of Link# 14 - Inflation
Name of Link# 15 - Lease
Name of Link# 16 - Finance
Name of Link# 17 - Mortgage Payoff
Name of Link# 18 - Refinance
Name of Link# 19 - Interest Rate
Name of Link# 20 - Weight Loss
Name of Link# 21 - BMI
Name of Link# 22 - Calorie
Name of Link# 23 - Army Body Fat
Name of Link# 24 - Body Fat
Name of Link# 25 - BMR
Name of Link# 26 - Weight Watchers Points
Name of Link# 27 - Anorexic BMI
Name of Link# 28 - Carbohydrate
Name of Link# 29 - Ideal Weight
Name of Link# 30 - Body Type
```

9. TEST DESIGN TECHNIQUES

There are various components involved in designing the tests. Let us understand some of the important components involved in designing a framework as well. We will learn the following topics in this chapter:

- Page Object Model
- Parameterizing using Excel
- Log4j Logging
- Exception Handling
- Multi Browser Testing
- Capture Screenshots
- Capture Videos

Page Object Model

Selenium acts on webelements with the help of their properties such ID, name, XPath, etc. Unlike QTP which has an inbuilt object repository (OR), Selenium has no inbuilt ORs.

Hence we need to build an OR which should also be maintainable and accessible on demand. Page Object Model (POM) is a popular design pattern to create an Object Repository in which each one of those webelements properties are created using a class file.

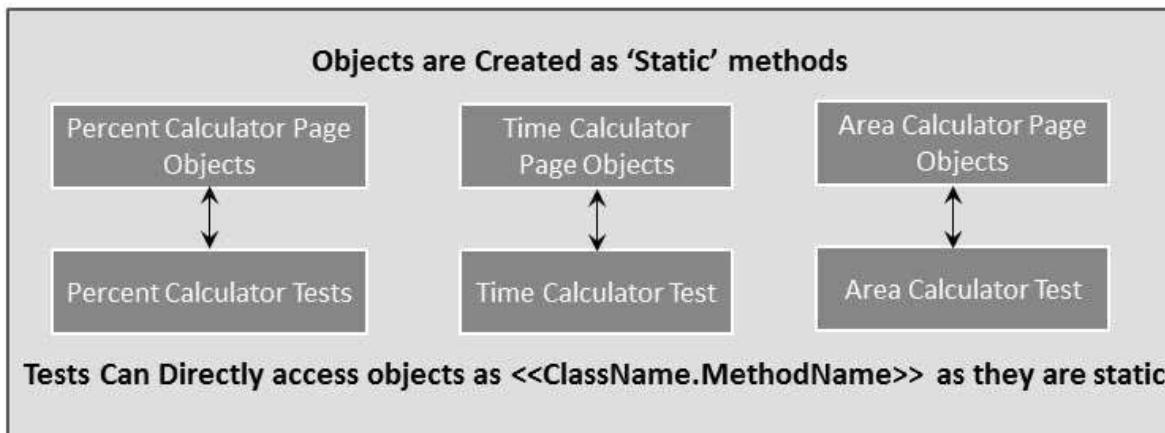
Advantages

- POM is an implementation where test objects and functions are separated from each other, thereby keeping the code clean.
- The objects are kept independent of test scripts. An object can be accessed by one or more test scripts, hence POM helps us to create objects once and use them multiple times.
- Since objects are created once, it is easy to access as well as update a particular property of an object.

POM FlowDiagram

Objects are created for each one of the pages and methods are developed exclusively to access to those objects. Let us use <http://calculator.net> for understanding the same.

There are various calculators associated with it and each one of those objects in a particular page is created in a separate class file as static methods and they all are accessed through the 'tests' class file in which a static method would be accessing the objects.



Example

Let us understand it by implementing POM for percent calculator test.

Step 1 : Create a simple class (page_objects_perc_calc.java) file within a package and create methods for each one of those object identifiers as shown below.

```

package PageObject;

import org.openqa.selenium.*;

public class page_objects_perc_calc
{
    private static WebElement element = null;

    // Math Calc Link
    public static WebElement lnk_math_calc(WebDriver driver)
    {
        element =
            driver.findElement(By.xpath(".//*[@id='menu']/div[3]/a"));
        return element;
    }
}

```

```
// Percentage Calc Link
public static WebElement lnk_percent_calc(WebDriver driver)
{
    element =
        driver.findElement(By.xpath(".//*[@id='menu']/div[4]/div[3]/a"));
    return element;
}

// Number 1 Text Box
public static WebElement txt_num_1(WebDriver driver)
{
    element = driver.findElement(By.id("cpar1"));
    return element;
}

// Number 2 Text Box
public static WebElement txt_num_2(WebDriver driver)
{
    element = driver.findElement(By.id("cpar2"));
    return element;
}

// Calculate Button
public static WebElement btn_calc(WebDriver driver)
{
    element =
        driver.findElement(By.xpath(".//*[@id='content']/table/tbody
/tr/td[2]/input"));
    return element;
}

// Result
public static WebElement web_result(WebDriver driver)
```

```
{
    element =
        driver.findElement(By.xpath("//*[@id='content']/p[2]/span/font/b"));
    return element;
}
}
```

Step 2 : Create a class with main and import the package and create methods for each one of those object identifiers as shown below.

```
package PageObject;

import java.util.concurrent.TimeUnit;

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

public class percent_calculator
{
    private static WebDriver driver = null;

    public static void main(String[] args)
    {
        driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("http://www.calculator.net");

        // Use page Object library now
        page_objects_perc_calc.lnk_math_calc(driver).click();
        page_objects_perc_calc.lnk_percent_calc(driver).click();

        page_objects_perc_calc.txt_num_1(driver).clear();
        page_objects_perc_calc.txt_num_1(driver).sendKeys("10");
        page_objects_perc_calc.txt_num_2(driver).clear();
        page_objects_perc_calc.txt_num_2(driver).sendKeys("50");
    }
}
```

```

page_objects_perc_calc.btn_calc(driver).click();

String result =
page_objects_perc_calc.web_result(driver).getText();

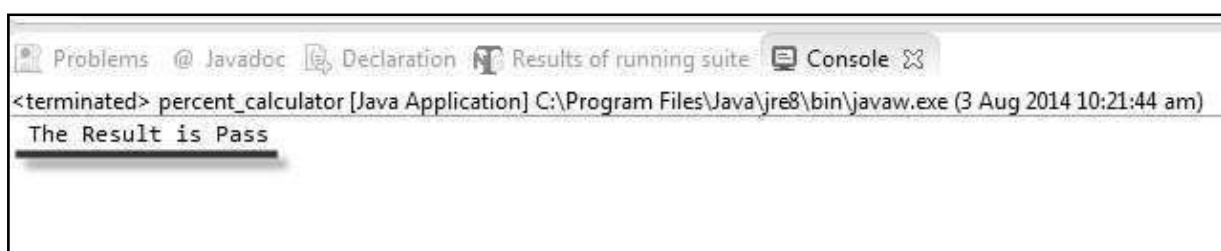
if(result.equals("5"))
{
    System.out.println(" The Result is Pass");
}
else
{
    System.out.println(" The Result is Fail");
}

driver.close();
}
}

```

Output

The test is executed and the result is printed in the console. Given below is the snapshot of the same.



The Result is Pass

Data Driven using Excel

While designing a test, parameterizing the tests is inevitable. We will make use of Apache POI - Excel JAR's to achieve the same. It helps us read and write into Excel.

Download JAR

Step 1 : Navigate to the URL – <http://poi.apache.org/download.html> and download the ZIP format.

The screenshot shows the Apache POI - Download page. On the left is a sidebar with links for Overview, Home, Download, Components, Help, Getting Involved, Component APIs, and Apache Wide. The main content area has a title "Apache POI - Download" and a section "Available Downloads" with instructions on how to download and verify the artifacts. It lists the latest stable release (Apache POI 3.10-FINAL) and archives of all prior releases. Below this, it mentions that Apache POI releases are available under the Apache License, Version 2.0, and provides a link to the NOTICE file. A section for "February 2014 - POI 3.10-FINAL available" announces the release of 3.10-FINAL, noting bug fixes and improvements. The "Binary Distribution" section contains two download links: "poi-bin-3.10-FINAL-20140208.tar.gz" and "poi-bin-3.10-FINAL-20140208.zip". Both links include MD5 and SHA1 checksums.

Step 2 : Click on the Mirror Link to download the JAR's.

The screenshot shows the Apache Software Foundation - Apache Download Mirrors page. It features the Apache feather logo and the text "The Apache Software Foundation". Below this is a section titled "Apache Download Mirrors" with the subtext "We suggest the following mirror site for your download:". A highlighted link provides the URL: <http://mirror.olnevhost.net/pub/apache/poi/release/bin/poi-bin-3.10-FINAL-20140208.zip>. A note at the bottom states: "Other mirror sites are suggested below. Please use the backup mirrors only to download PGP and MD5 signatures."

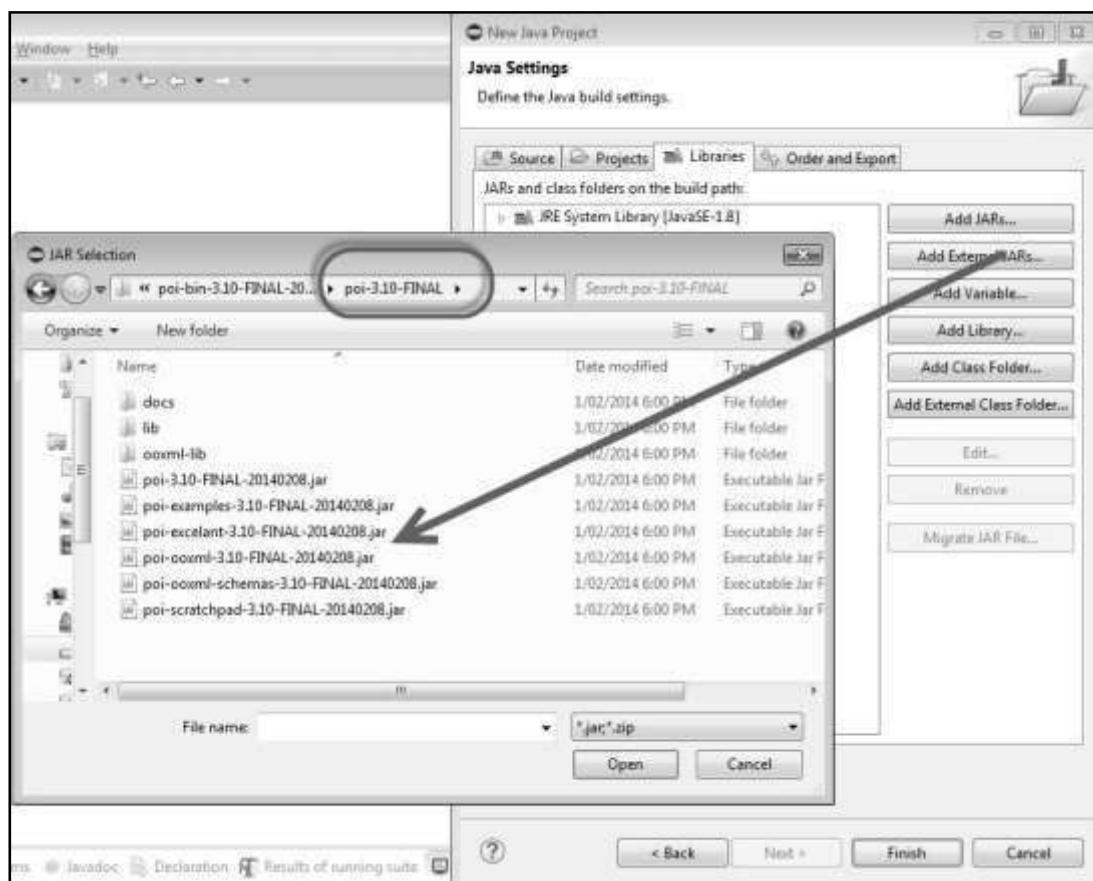
Step 3 : Unzip the contents to a folder.

Name	Date modified
poi-bin-3.10-FINAL-20140208	3/08/2014 10:30 AM
selenium-java-2.42.2	27/07/2014 8:18 AM
chromedriver.exe	1/05/2014 1:38 PM
chromedriver_win32 (1).zip	1/08/2014 12:24 AM
IEDriverServer.exe	19/02/2014 6:45 PM
poi-bin-3.10-FINAL-20140208.zip	3/08/2014 10:30 AM
selenium-java-2.42.2.zip	27/07/2014 8:17 AM
selenium-server-standalone-2.42.2.jar	27/07/2014 7:42 AM

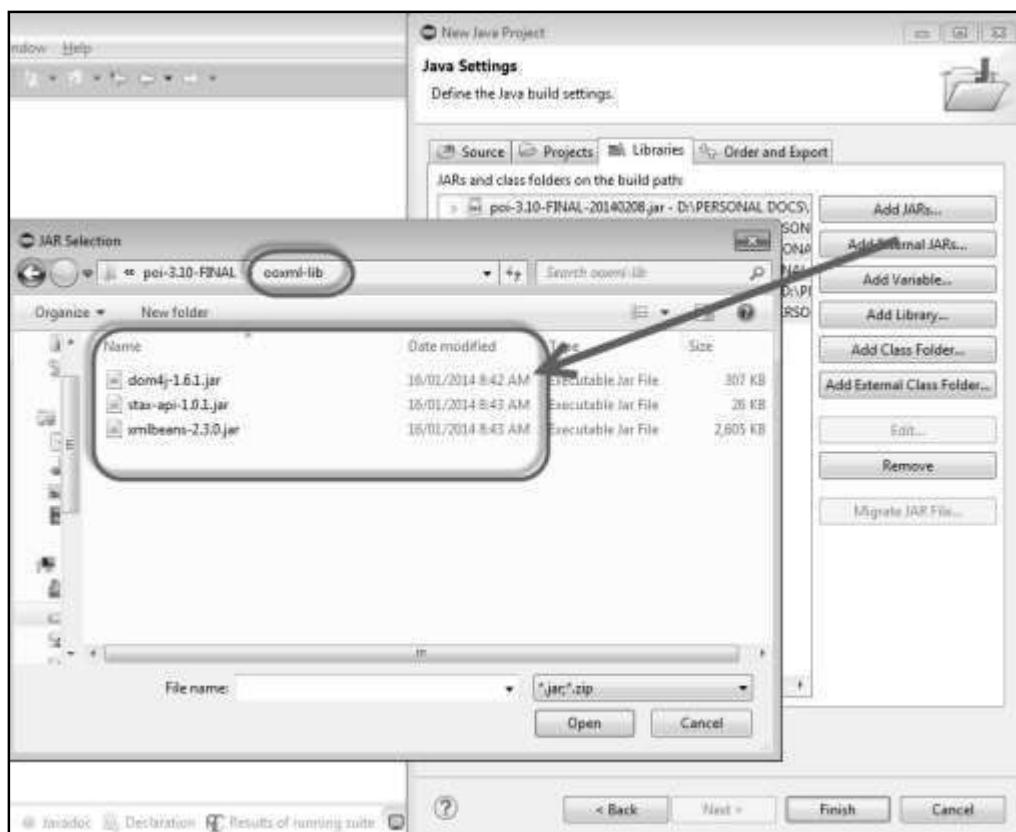
Step 4 : Unzipped contents would be displayed as shown below.

PERSONAL DOCS > Selenium Trials > JAR > poi-bin-3.10-FINAL-20140208 > poi-3.10-FINAL >				
with ▾ New folder				
Name	Date modified	Type	Size	
docs	1/02/2014 6:00 PM	File folder		
lib	1/02/2014 6:00 PM	File folder		
ooxml-lib	1/02/2014 6:00 PM	File folder		
LICENSE	16/01/2014 8:37 AM	File	27 KB	
NOTICE	16/01/2014 8:37 AM	File	1 KB	
poi-3.10-FINAL-20140208.jar	1/02/2014 6:00 PM	Executable Jar File	1,906 KB	
poi-examples-3.10-FINAL-20140208.jar	1/02/2014 6:00 PM	Executable Jar File	306 KB	
poi-excelant-3.10-FINAL-20140208.jar	1/02/2014 6:00 PM	Executable Jar File	30 KB	
poi-ooxml-3.10-FINAL-20140208.jar	1/02/2014 6:00 PM	Executable Jar File	1,008 KB	
poi-ooxml-schemas-3.10-FINAL-2014020...	1/02/2014 6:00 PM	Executable Jar File	4,831 KB	
poi-scratchpad-3.10-FINAL-20140208.jar	1/02/2014 6:00 PM	Executable Jar File	1,212 KB	

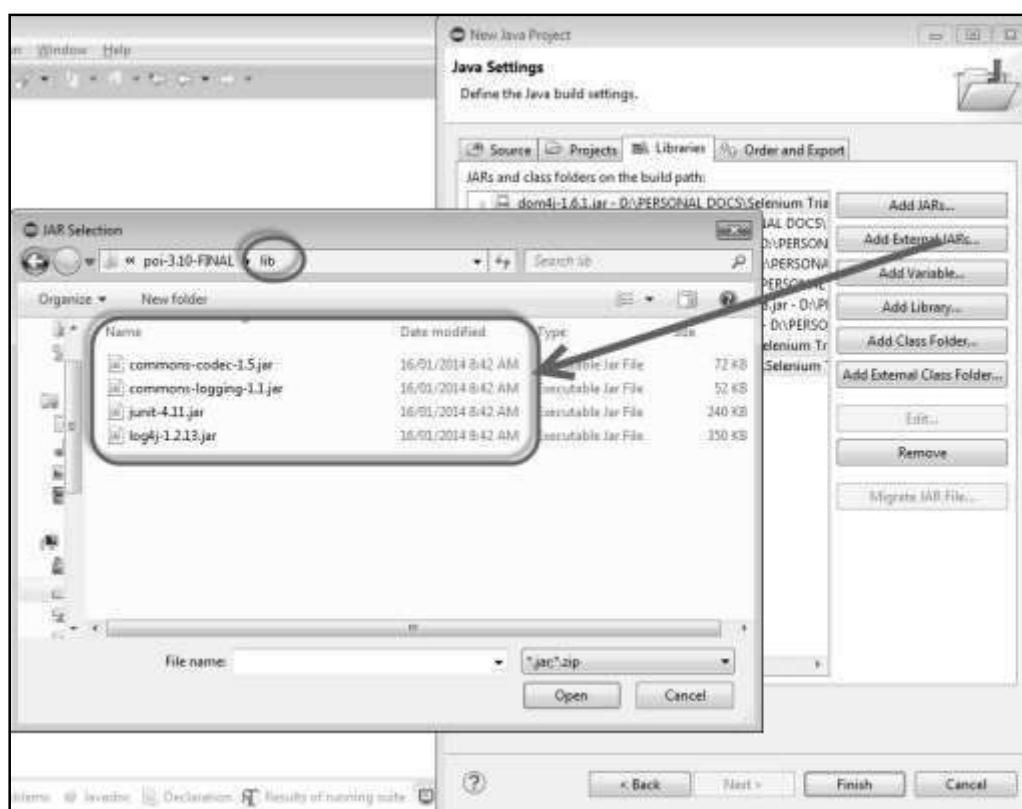
Step 5 : Now create a new project and add all the 'External JARs' under 'poi-3.10.FINAL' folder.



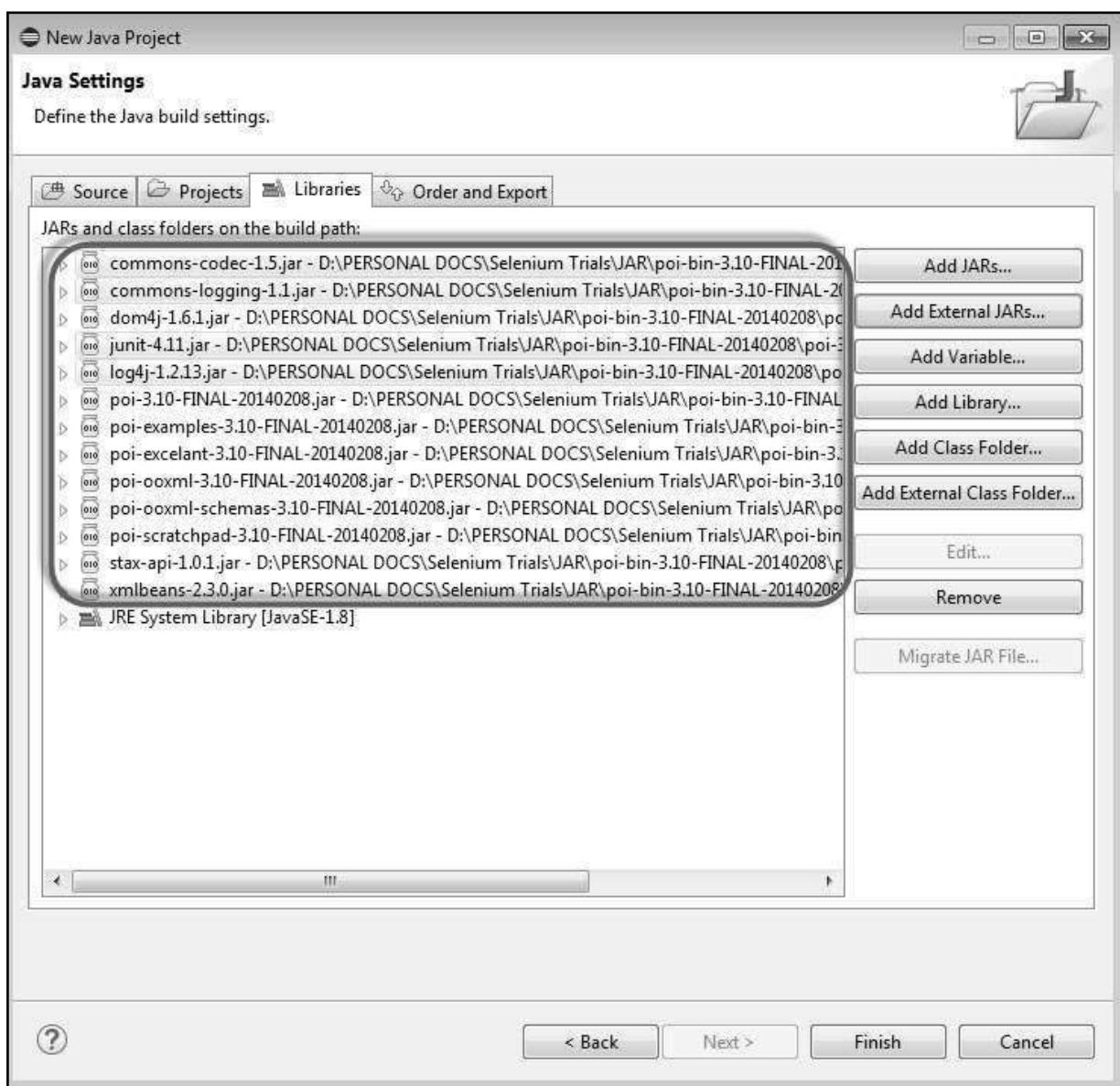
Step 6 : Now add all the 'External JARs' under the 'ooxml-lib' folder.



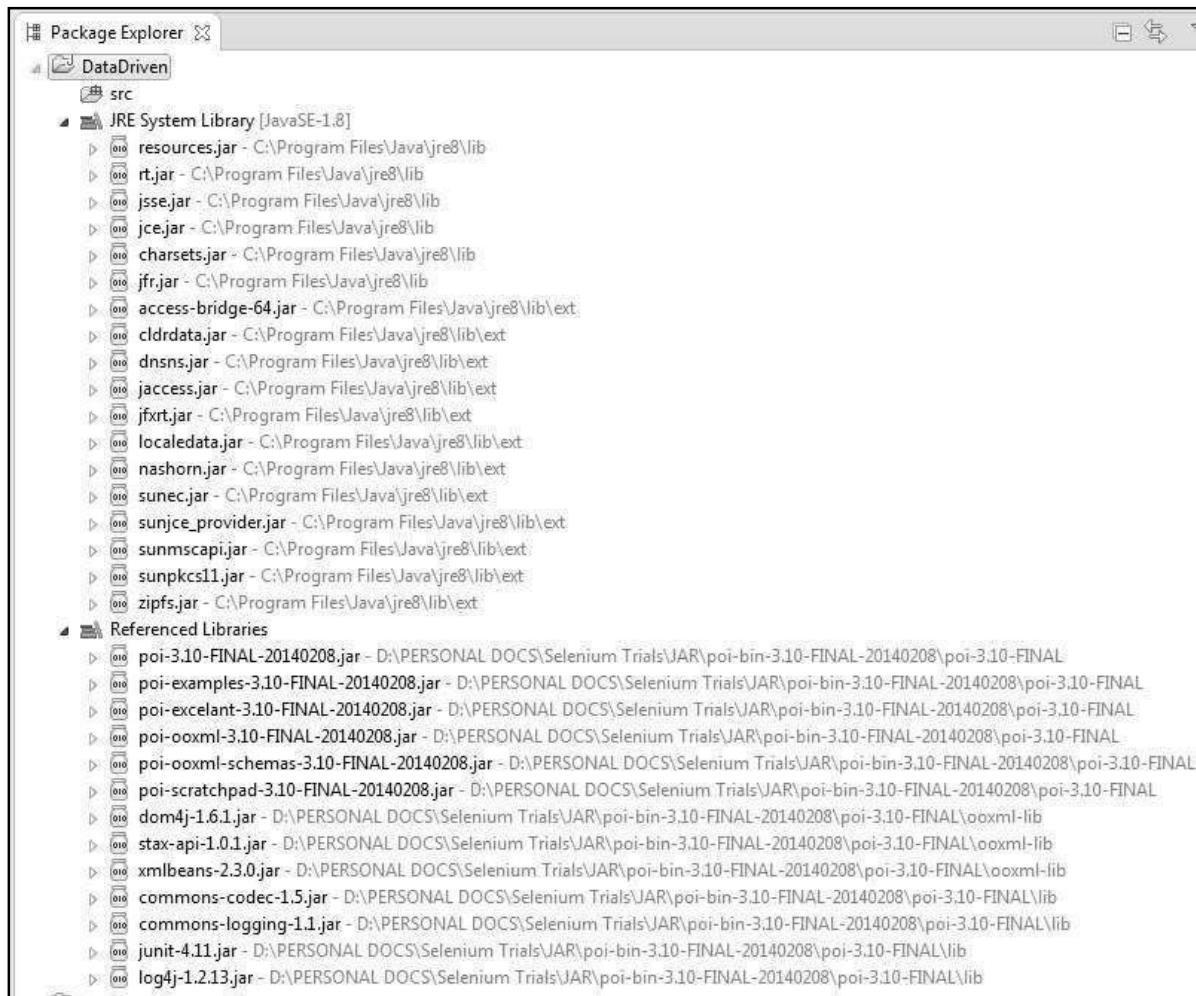
Step 7 : Now add all the 'External JARs' under the 'lib' folder.



Step 8 : The Added JAR is displayed as shown below.



Step 9 : The Package Explorer is displayed as shown below. Apart from that, add 'WebDriver' related JAR's



Parameterization

For demonstration, we will parameterize the percent calculator test.

Step 1 : We will parameterize all the inputs required for percent calculator using Excel. The designed Excel is shown below.

	A	B	C	D
1	Test Name	Num1	Num2	Expected Result
2	Percent Calculator	10.25	50.56	5.1824
3	Percent Calculator	20.55	40.66	8.35563
4	Percent Calculator	30	200.5	60.15
5	Percent Calculator	103.5	500	517.5
6				
7				

Step 2 : Execute all the percent calculator functions for all the specified parameters.

Step 3 : Let us create generic methods to access the Excel file using the imported JARs. These methods help us get a particular cell data or to set a particular cell data, etc.

```

import java.io.*;
import org.apache.poi.xssf.usermodel.*;

public class excelutils
{
    private XSSFSheet ExcelWSheet;
    private XSSFWorkbook ExcelWBook;

    //Constructor to connect to the Excel with sheetname and Path
    public excelutils(String Path, String SheetName) throws Exception
    {
        try
        {
            // Open the Excel file
            FileInputStream ExcelFile = new FileInputStream(Path);

            // Access the required test data sheet
            ExcelWBook = new XSSFWorkbook(ExcelFile);
            ExcelWSheet = ExcelWBook.getSheet(SheetName);
        }
        catch (Exception e)
        {
            throw (e);
        }
    }

    // This method is to set the rowcount of the excel.
    public int excel_get_rows() throws Exception
    {
        try
        {

```

```

        return ExcelWSheet.getPhysicalNumberOfRows();
    }
    catch (Exception e)
    {
        throw (e);
    }
}

// This method to get the data and get the value as strings.
public String getCellDataasString(int RowNum, int ColNum) throws Exception
{
    try
    {
        String CellData =
        ExcelWSheet.getRow(RowNum).getCell(ColNum).getStringCellValue();
        System.out.println("The value of CellData " + CellData);
        return CellData;
    }
    catch (Exception e)
    {
        return "Errors in Getting Cell Data";
    }
}

// This method to get the data and get the value as number.
public double getCellDataasnumber(int RowNum, int ColNum) throws Exception
{
    try
    {
        double CellData =
        ExcelWSheet.getRow(RowNum).getCell(ColNum).getNumericCellValue();
    }
}

```

```

        System.out.println("The value of CellData " + CellData);
        return CellData;
    }

    catch (Exception e)
    {
        return 000.00;
    }
}
}

```

Step 4 : Now add a main method which will access the Excel methods that we have developed.

```

import java.io.*;
import org.apache.poi.xssf.usermodel.*;

public class excelutils
{
    private XSSFSheet ExcelWSheet;
    private XSSFWorkbook ExcelWBook;

    //Constructor to connect to the Excel with sheetname and Path
    public excelutils(String Path, String SheetName) throws Exception
    {
        try
        {
            // Open the Excel file
            FileInputStream ExcelFile = new FileInputStream(Path);
            // Access the required test data sheet
            ExcelWBook = new XSSFWorkbook(ExcelFile);
            ExcelWSheet = ExcelWBook.getSheet(SheetName);

        }
        catch (Exception e)
    }
}

```

```

    {
        throw (e);
    }
}

// This method is to set the rowcount of the excel.
public int excel_get_rows() throws Exception
{
    try
    {
        return ExcelWSheet.getPhysicalNumberOfRows();
    }

    catch (Exception e)
    {
        throw (e);
    }
}

// This method to get the data and get the value as strings.
public String getCellDataasstring(int RowNum, int ColNum) throws Exception
{
    try
    {
        String CellData =
        ExcelWSheet.getRow(RowNum).getCell(ColNum).getStringCellValue();
        // Cell = ExcelWSheet.getRow(RowNum).getCell(ColNum);
        // String CellData = Cell.getStringCellValue();
        System.out.println("The value of CellData " + CellData);
        return CellData;
    }
    catch (Exception e)
    {
}
}

```

```

        return "Errors in Getting Cell Data";
    }

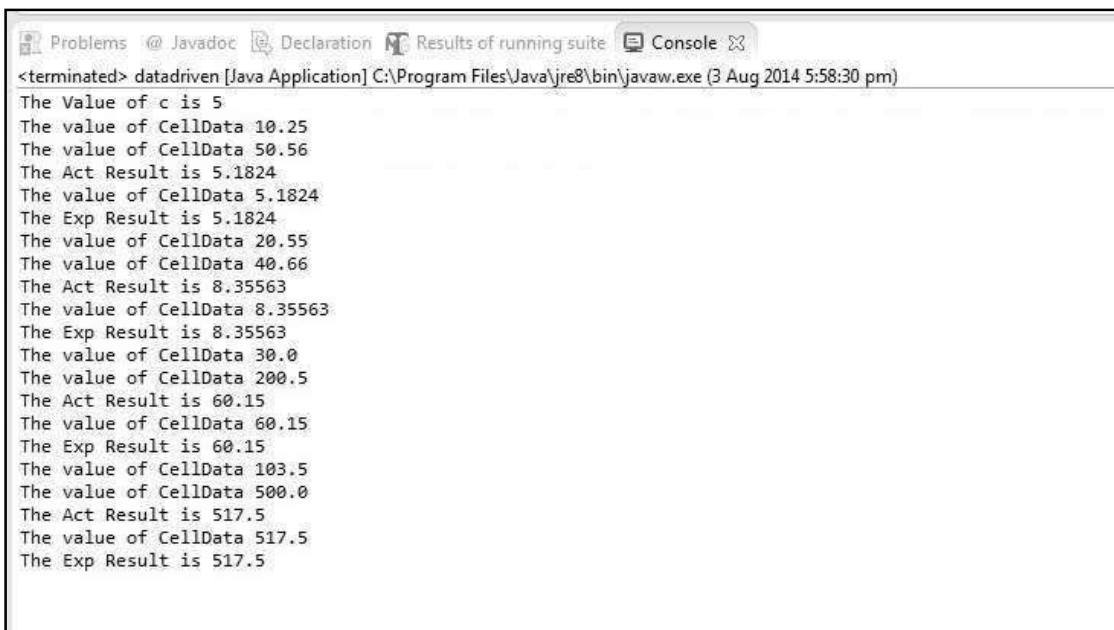
}

// This method to get the data and get the value as number.
public double getCellDataasnumber(int RowNum, int ColNum) throws Exception
{
    try
    {
        double CellData =
            ExcelWSheet.getRow(RowNum).getCell(ColNum).getNumericCellValue();
        // Cell = ExcelWSheet.getRow(RowNum).getCell(ColNum);
        // String CellData = Cell.getStringCellValue();
        System.out.println("The value of CellData " + CellData);
        return CellData;
    }
    catch (Exception e)
    {
        return 000.00;
    }
}
}

```

Output

Upon executing the script, the output is displayed in the console as shown below.



```

Problems @ Javadoc Declaration Results of running suite Console
<terminated> datadriven [Java Application] C:\Program Files\Java\jre8\bin\javaw.exe (3 Aug 2014 5:58:30 pm)
The Value of c is 5
The value of CellData 10.25
The value of CellData 50.56
The Act Result is 5.1824
The value of CellData 5.1824
The Exp Result is 5.1824
The value of CellData 20.55
The value of CellData 40.66
The Act Result is 8.35563
The value of CellData 8.35563
The Exp Result is 8.35563
The value of CellData 30.0
The value of CellData 200.5
The Act Result is 60.15
The value of CellData 60.15
The Exp Result is 60.15
The value of CellData 103.5
The value of CellData 500.0
The Act Result is 517.5
The value of CellData 517.5
The Exp Result is 517.5

```

Log4j Logging

Log4j is an audit logging framework that gives information about what has happened during execution. It offers the following advantages:

- Enables us to understand the application run.
- Log output can be saved that can be analyzed later.
- Helps in debugging, in case of test automation failures.
- Can also be used for auditing purposes to look at the application's health.

Components

1. Instance of Logger class.
2. Log level methods used for logging the messages as one of the following:
 - error
 - warn
 - info
 - debug
 - log

Example

Let us use the same percent calculator for this demo.

Step 1 : Download log4j JAR file from <https://logging.apache.org/log4j/1.2/download.html> and download the Zipped format of the JAR file.

Last Published: 2012-05-13 | Version: 1.2.17 | Apache | Log4j Services > log4j > 1.2

Download Apache log4j 1.2.17

Apache log4j 1.2.17 is distributed under the Apache License, version 2.0 ⓘ.

The link in the Mirrors column should display a list of available mirrors with a default selection based on signature. Links to the originals on the main distribution server.

	Mirrors
Apache log4j 1.2.17 (tar.gz)	log4j-1.2.17.tar.gz ⓘ
Apache log4j 1.2.17 (zip)	log4j-1.2.17.zip ⓘ

It is essential that you verify the integrity of the downloaded files using the PGP or MD5 signatures of our releases.

The PGP signatures can be verified using PGP or GPG. First download the KEYS ⓘ as well as the asc directory ⓘ, rather than from a mirror. Then verify the signatures using:

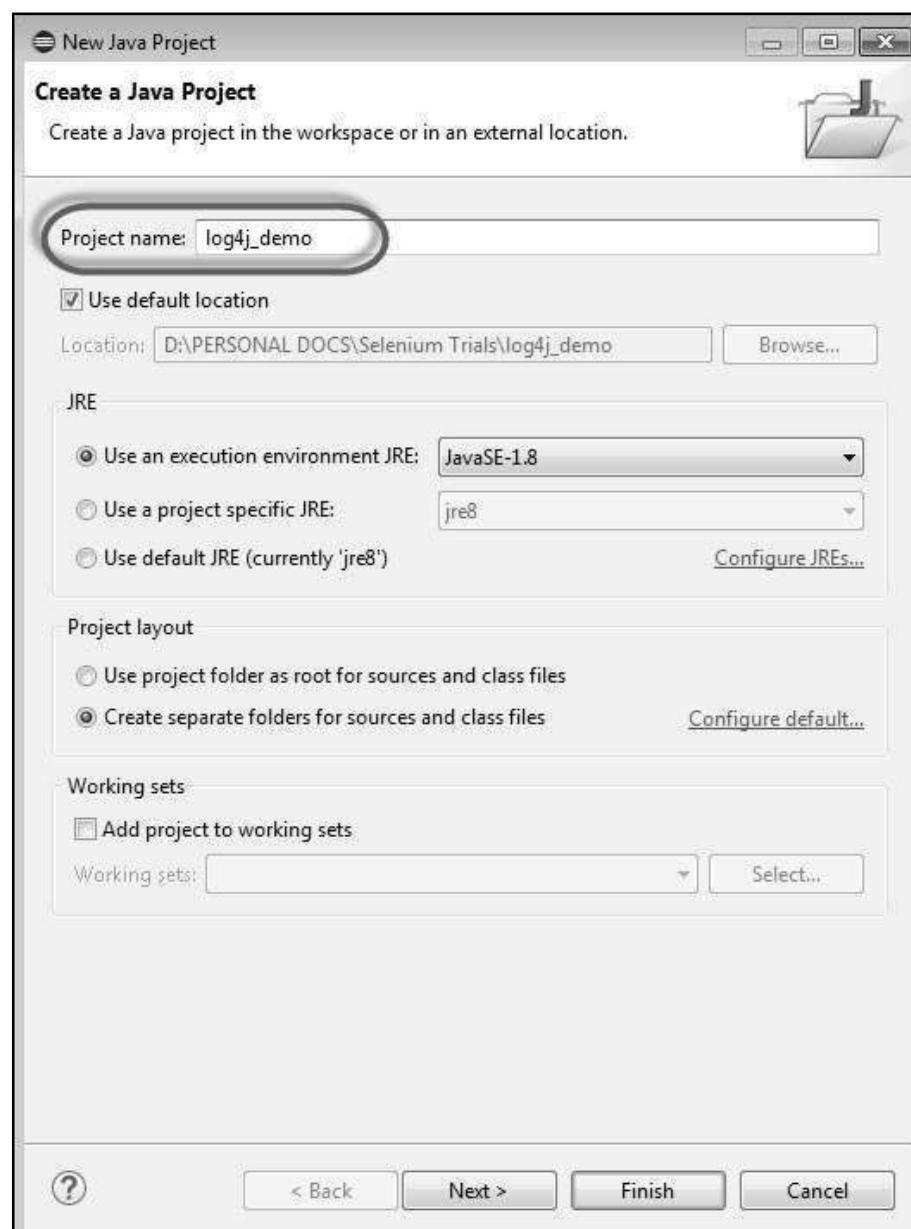
```
* gpg --import KEYS
* gpg --verify log4j-1.2.17.tar.gz.asc
```

Apache log4j 1.2.17 is signed by Christian Grobmeier 42196CAB

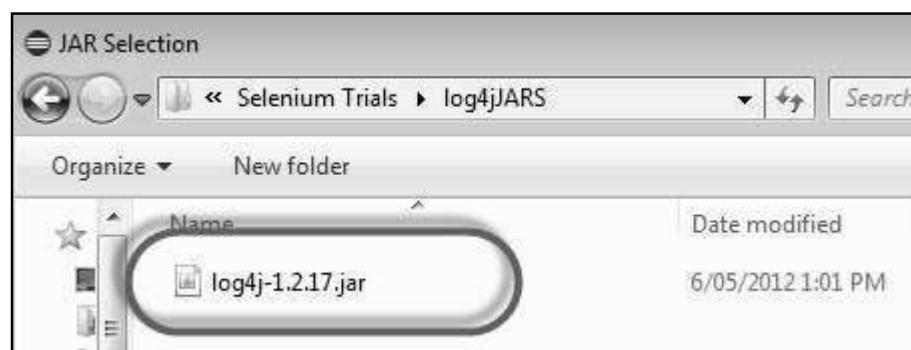
Step 2 : Create 'New Java Project' by navigating to the File menu.



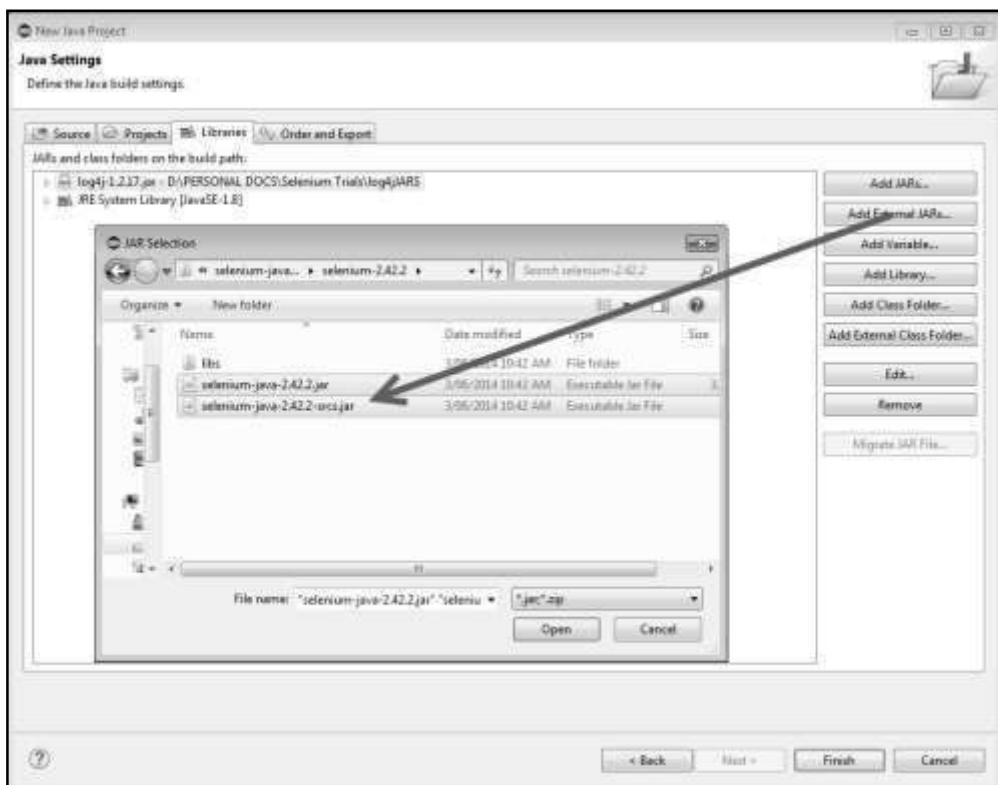
Step 3 : Enter the name of the project as 'log4j_demo' and click 'Next'.



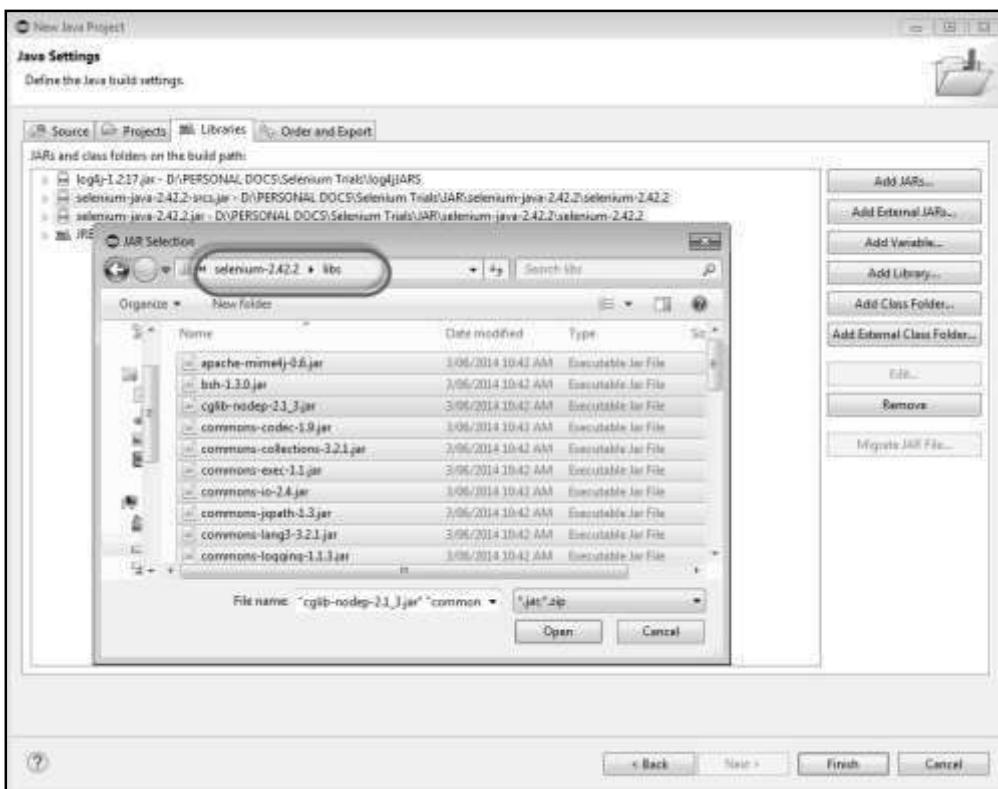
Step 4 : Click Add External Jar and add 'Log4j-1.2.17.jar'.



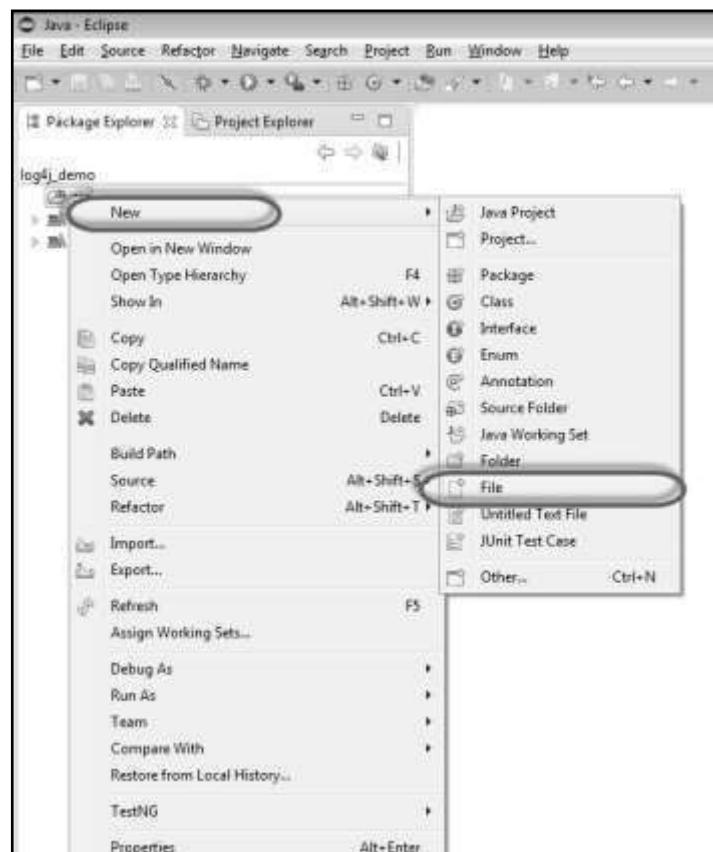
Step 5 : Click Add External Jar and add Selenium WebDriver Libraries.



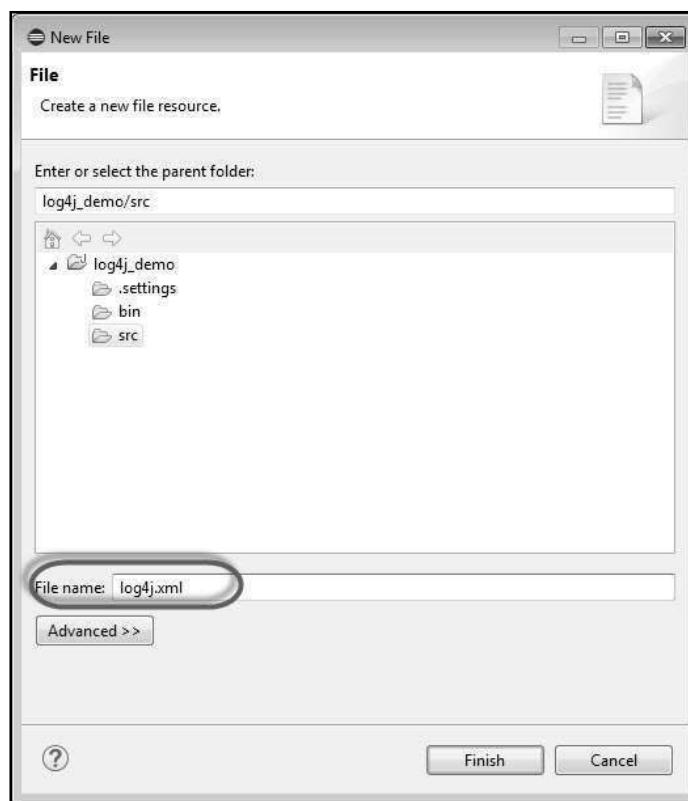
Step 6 : Click Add External Jar and add Selenium WebDriver JAR's located in the Libs folder.



Step 7 : Add a New XML file using which we can specify the Log4j properties.



Step 8 : Enter the Logfile name as 'Log4j.xml'.



122

Step 9 : The final folder structure is shown below.



Step 10 : Now add the properties of Log4j which would be picked up during execution.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/" debug="false">
    <appender name="fileAppender" class="org.apache.log4j.FileAppender">
        <param name="Threshold" value="INFO" />
        <param name="File" value="percent_calculator.log"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d{yyyy-MM-dd HH:mm:ss} [%c] (%t:%x) %m%n" />
        </layout>
    </appender>
    <root>
        <level value="INFO"/>
        <appender-ref ref="fileAppender"/>
    </root>
</log4j:configuration>
```

Step 11 : Now for demonstration purpose, we will incorporate log4j in the same test that we have been performing (percent calculator). Add a class file in the 'Main' function.

```
package log4j_demo;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import org.apache.log4j.xml.DOMConfigurator;
```

```

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;

public class log4j_demo
{

    static final Logger logger =
    LogManager.getLogger(log4j_demo.class.getName());

    public static void main(String[] args)
    {
        DOMConfigurator.configure("log4j.xml");

        logger.info("#####");
        logger.info("TEST Has Started");

        WebDriver driver = new FirefoxDriver();

        // Puts an Implicit wait, will wait for 10 seconds
        // before throwing exception
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

        // Launch website
        driver.navigate().to("http://www.calculator.net/");
        logger.info("Open Calc Application");

        // Maximize the browser
        driver.manage().window().maximize();

        // Click on Math Calculators
        driver.findElement(By.xpath(".//*[@id='menu']/div[3]/a")).click();
    }
}

```

```

logger.info("Clicked Math Calculator Link");

// Click on Percent Calculators
driver.findElement(By.xpath(".//*[@id='menu']/div[4]/div[3]/a"))
.click(); logger.info("Clicked Percent Calculator Link");

// Enter value 10 in the first number of the percent Calculator
driver.findElement(By.id("cpar1")).sendKeys("10");
logger.info("Entered Value into First Text Box");

// Enter value 50 in the second number of the percent Calculator
driver.findElement(By.id("cpar2")).sendKeys("50");
logger.info("Entered Value into Second Text Box");

// Click Calculate Button
driver.findElement(By.xpath(".//*[@id='content']/table
/tbody/tr/td[2]/input")).click();
logger.info("Click Calculate Button");

// Get the Result Text based on its xpath
String result =
driver.findElement(By.xpath(".//*[@id='content']
/p[2]/span/font/b")).getText();
logger.info("Get Text Value");

// Print a Log In message to the screen
logger.info(" The Result is " + result);

if(result.equals("5"))
{
    logger.info("The Result is Pass");
}

```

```

    }

else
{
    logger.error("TEST FAILED. NEEDS INVESTIGATION");
}

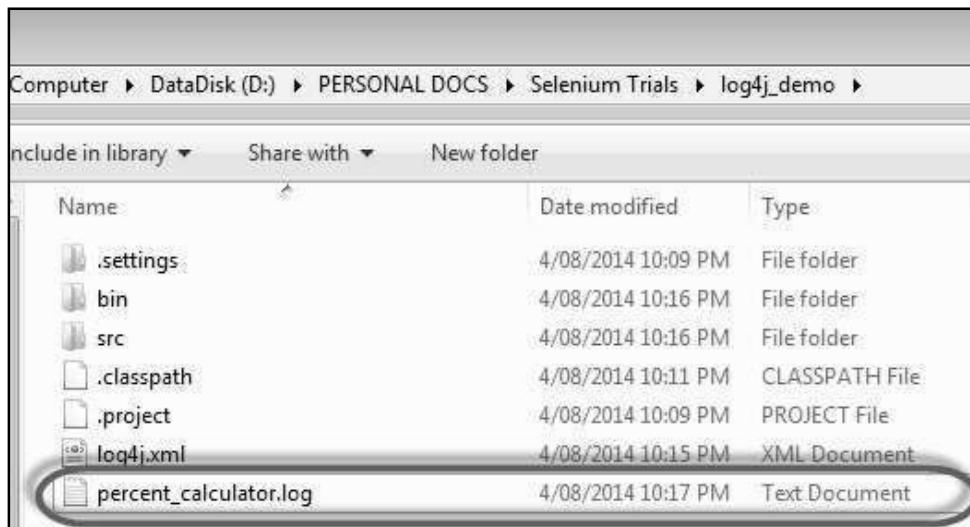
logger.info("#####");

// Close the Browser.
driver.close();
}
}

```

Execution

Upon execution, the log file is created on the root folder as shown below. You CANNOT locate the file in Eclipse. You should open 'Windows Explorer' to show the same.



The contents of the file is shown below.

```

1 2014-08-04 22:17:22 [log4j_demo.log4j_demo] (main:) #####
2 2014-08-04 22:17:22 [log4j_demo.log4j_demo] (main:) TEST Has Started
3 2014-08-04 22:17:31 [log4j_demo.log4j_demo] (main:) Open Calc Application
4 2014-08-04 22:17:33 [log4j_demo.log4j_demo] (main:) Clicked Math Calculator Link
5 2014-08-04 22:17:33 [log4j_demo.log4j_demo] (main:) Clicked Percent Calculator Link
6 2014-08-04 22:17:34 [log4j_demo.log4j_demo] (main:) Entered Value into First Text Box
7 2014-08-04 22:17:35 [log4j_demo.log4j_demo] (main:) Entered Value into Second Text Box
8 2014-08-04 22:17:35 [log4j_demo.log4j_demo] (main:) Click Calculate Button
9 2014-08-04 22:17:36 [log4j_demo.log4j_demo] (main:) Get Text Value
10 2014-08-04 22:17:36 [log4j_demo.log4j_demo] (main:) The Result is 5
11 2014-08-04 22:17:36 [log4j_demo.log4j_demo] (main:) The Result is Pass
12 2014-08-04 22:17:36 [log4j_demo.log4j_demo] (main:) #####

```

Exception Handling

When we are developing tests, we should ensure that the scripts can continue their execution even if the test fails. An unexpected exception would be thrown if the worst case scenarios are not handled properly.

If an exception occurs due to an element not found or if the expected result doesn't match with actuals, we should catch that exception and end the test in a logical way rather than terminating the script abruptly.

Syntax

The actual code should be placed in the try block and the action after exception should be placed in the catch block. Note that the 'finally' block executes regardless of whether the script had thrown an exception or NOT.

```

try
{
    // Perform Action
}
catch(ExceptionType1 exp1)
{
    // Catch block 1
}
catch(ExceptionType2 exp2)
{
    // Catch block 2
}
catch(ExceptionType3 exp3)
{
    // Catch block 3
}
finally
{
    // The finally block always executes.
}

```

Example

If an element is not found (due to some reason), we should step out of the function smoothly. So we always need to have a try-catch block if we want to exit smoothly from a function.

```
public static WebElement lnk_percent_calc(WebDriver driver) throws Exception
{
    try
    {
        element =
            driver.findElement(By.xpath(".//*[@id='menu']/div[4]/div[3]/a"));
        return element;
    }
    catch (Exception e1)
    {
        // Add a message to your Log File to capture the error
        Logger.error("Link is not found.");
        // Take a screenshot which will be helpful for analysis.
        File screenshot =
            ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

        FileUtils.copyFile(screenshot, new
            File("D:\\framework\\screenshots.jpg"));

        throw(e1);
    }
}
```

Multi Browser Testing

Users can execute scripts in multiple browsers simultaneously. For demonstration, we will use the same scenario that we had taken for Selenium Grid. In the Selenium Grid example, we had executed the scripts remotely; here we will execute the scripts locally.

First of all, ensure that you have appropriate drivers downloaded. Please refer the chapter "Selenium Grid" for downloading IE and Chrome drivers.

Example

For demonstration, we will perform percent calculator in all the browsers simultaneously.

```
package TestNG;

import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.testng.annotations.*;

public class TestNGClass
{
    private WebDriver driver;
    private String URL = "http://www.calculator.net";

    @Parameters("browser")
    @BeforeTest
    public void launchapp(String browser)
    {

        if (browser.equalsIgnoreCase("firefox"))
        {
            System.out.println(" Executing on FireFox");
            driver = new FirefoxDriver();
            driver.get(URL);
            driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
            driver.manage().window().maximize();
        }
        else if (browser.equalsIgnoreCase("chrome"))
    }
```

```

{
    System.out.println(" Executing on CHROME");
    System.out.println("Executing on IE");
    System.setProperty("webdriver.chrome.driver",
    "D:\\chromedriver.exe");
    driver = new ChromeDriver();
    driver.get(URL);
    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
    driver.manage().window().maximize();
}

else if (browser.equalsIgnoreCase("ie"))
{
    System.out.println("Executing on IE");
    System.setProperty("webdriver.ie.driver",
    "D:\\IEDriverServer.exe");
    driver = new InternetExplorerDriver();
    driver.get(URL);
    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
    driver.manage().window().maximize();
}

else
{
    throw new IllegalArgumentException("The Browser Type is Undefined");
}
}

@Test
public void calculatepercent()
{
    // Click on Math Calculators
    driver.findElement(By.xpath(".//*[@id='menu']/div[3]/a")).click();

    // Click on Percent Calculators
    driver.findElement(By.xpath(".//*[@id='menu']/div[4]/div[3]/a")).click();
}

```

```

// Enter value 10 in the first number of the percent Calculator
driver.findElement(By.id("cpar1")).sendKeys("10");

// Enter value 50 in the second number of the percent Calculator
driver.findElement(By.id("cpar2")).sendKeys("50");

// Click Calculate Button
driver.findElement(By.xpath(".//*[@id='content']/table/tbody/
tr/td[2]/input")).click();

// Get the Result Text based on its xpath
String result =
driver.findElement(By.xpath(".//*[@id='content']/p[2]/span
/font/b")).getText();

// Print a Log In message to the screen
System.out.println(" The Result is " + result);

if(result.equals("5"))
{
    System.out.println(" The Result is Pass");
}
else
{
    System.out.println(" The Result is Fail");
}

}

@AfterTest
public void closeBrowser()
{
    driver.close();
}

```

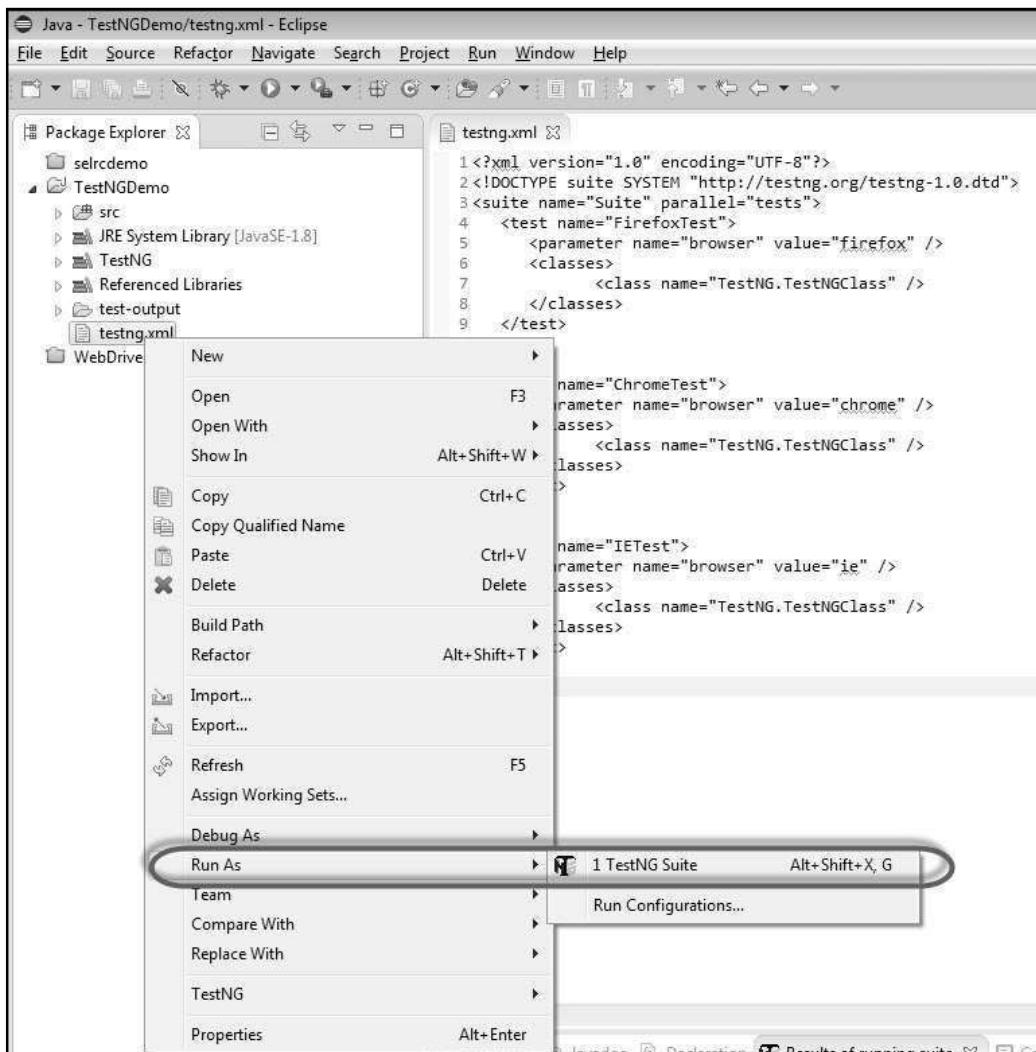
{
}

Create an XML which will help us in parameterizing the browser name and don't forget to mention parallel="tests" in order to execute in all the browsers simultaneously.

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer view displays a project named 'TestNGDemo' with a 'src' folder containing a 'TestNG' directory. Below these are 'JRE System Library [JavaSE-1.8]', 'TestNG', 'Referenced Libraries', and 'test-output'. A file named 'testing.xml' is selected and highlighted with a red oval. An arrow points from this oval to the right-hand code editor window. The code editor displays the 'testing.xml' file content:

```
1<?xml version="1.0" encoding="UTF-8"?>
2<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
3<suite name="Suite" parallel="tests">
4    <test name="FirefoxTest">
5        <parameter name="browser" value="firefox" />
6        <classes>
7            <class name="TestNG.TestNGClass" />
8        </classes>
9    </test>
10
11    <test name="ChromeTest">
12        <parameter name="browser" value="chrome" />
13        <classes>
14            <class name="TestNG.TestNGClass" />
15        </classes>
16    </test>
17
18
19    <test name="IETest">
20        <parameter name="browser" value="ie" />
21        <classes>
22            <class name="TestNG.TestNGClass" />
23        </classes>
24    </test>
25
26</suite>
```

Execute the script by performing right-click on the XML file and select 'Run As' >> 'TestNG' Suite as shown below.



Output

All the browsers would be launched simultaneously and the result would be printed in the console.

Note : To execute on IE successfully, ensure that the check box 'Enable Protected Mode' under the security tab of 'IE Option' is either checked or unchecked across all zones.

```
Problems @ Javadoc Declaration Results of running suite Console X
<terminated> testng.xml [TestNG] C:\Program Files\Java\jre8\bin\javaw.exe (5 Aug 2014 12:56:29 am)
[TestNG] Running:
D:\PERSONAL DOCS\Selenium Trials\TestNGDemo\testng.xml

Executing on CHROME
Executing on IE
Executing on IE
Executing on FireFox
Starting ChromeDriver (v2.10.267521) on port 37977
Only local connections are allowed.
Started InternetExplorerDriver server (64-bit)
2.40.0.0
Listening on port 15717
The Result is 5
The Result is Pass
The Result is 5
The Result is Pass
The Result is 5
The Result is Pass
=====
Suite
Total tests run: 3, Failures: 0, Skips: 0
=====
```

TestNG results can be viewed in HTML format for detailed analysis.

The screenshot shows the TestNG report interface. The main window title is "TestNG report". The left sidebar has a tree view under "All suites" with nodes for "Suite" (selected), "Info", and "Results". The "Tests for Suite" panel on the right lists three test classes: FirefoxTest (1 class), ChromeTest (1 class), and IETest (1 class). Below this, the "Results" panel shows a summary: 3 Passed, 0 Failed, and 0 Skipped. The bottom section displays a detailed tree view of all tests, including their names and execution times, such as "Suite (3/3/0.0) (0.276 s)" and "IETest (1/0.499 s)".

Capture Screenshots

This functionality helps to grab screenshots at runtime when required, in particularly when a failure happens. With the help of screenshots and log messages, we will be able to analyze the results better.

Screenshots are configured differently for local executions and Selenium Grid (remote) executions. Let us take a look at each one them with an example.

Localhost Execution

In the following example, we will take a screenshot after calculating the percentage. Ensure that you give a valid path to save the screenshot.

```
import java.io.File;
import java.io.IOException;
import java.util.concurrent.TimeUnit;

import org.apache.commons.io.FileUtils;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;

public class webdrivedemo
{
    public static void main(String[] args) throws IOException
    {
        WebDriver driver = new FirefoxDriver();

        // Puts an Implicit wait, will wait for 10 seconds
        // before throwing exception
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

        // Launch website
        driver.navigate().to("http://www.calculator.net/");

        // Maximize the browser
        driver.manage().window().maximize();
    }
}
```

```

// Click on Math Calculators
driver.findElement(By.xpath(".//*[@id='menu']/div[3]/a")).click();

// Click on Percent Calculators
driver.findElement(By.xpath(".//*[@id='menu']/div[4]/div[3]/a")).click();

// Enter value 10 in the first number of the percent Calculator
driver.findElement(By.id("cpar1")).sendKeys("10");

// Enter value 50 in the second number of the percent Calculator
driver.findElement(By.id("cpar2")).sendKeys("50");

// Click Calculate Button
driver.findElement(By.xpath(".//*[@id='content']/table
/tbody/tr/td[2]/input")).click();

// Get the Result Text based on its xpath
String result =
driver.findElement(By.xpath(".//*[@id='content']/p[2]
/span/font/b")).getText();

File screenshot =
((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(screenshot, new
File("D:\\screenshots\\screenshots1.jpg"));

// Print a Log In message to the screen
System.out.println(" The Result is " + result);

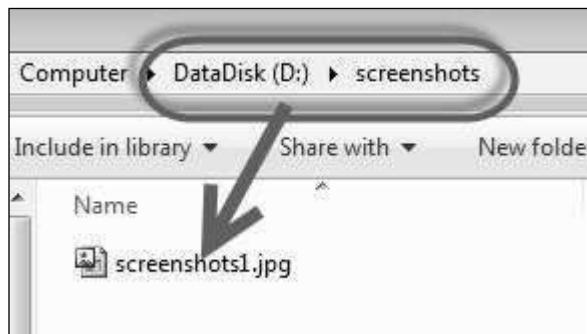
// Close the Browser.
driver.close();
}

```

{}

Output

Upon executing the script, the screenshot is saved in the 'D:\screenshots' folder with the name 'screenshots1.jpg' as shown below.



Selenium Grid – Screenshot Capture

While working with Selenium Grids, we should ensure that we are taking the screenshots correctly from the remote system. We will use augmented driver.

Example

We will execute the script on a Firefox node attached to a hub. For more on configuring hub and nodes, please refer the **Selenium Grids** chapter.

```
package TestNG;

import org.openqa.selenium.remote.Augmenter;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.TakesScreenshot;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;
import java.io.File;
import java.net.URL;
import java.net.MalformedURLException;
import org.apache.commons.io.FileUtils;
```

```

import org.openqa.selenium.remote.RemoteWebDriver;
import java.io.IOException;

public class TestNGClass
{
    public WebDriver driver;
    public String URL, Node;
    protected ThreadLocal<RemoteWebDriver> threadDriver = null;

    @Parameters("browser")
    @BeforeTest
    public void launchapp(String browser) throws MalformedURLException
    {
        String URL = "http://www.calculator.net";
        if (browser.equalsIgnoreCase("firefox"))
        {
            System.out.println(" Executing on FireFox");
            String Node = "http://10.112.66.52:5555/wd/hub";
            DesiredCapabilities cap = DesiredCapabilities.firefox();
            cap.setBrowserName("firefox");

            driver = new RemoteWebDriver(new URL(Node), cap);
            // Puts an Implicit wait, Will wait for 10 seconds
            // before throwing exception
            driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

            // Launch website
            driver.navigate().to(URL);
            driver.manage().window().maximize();
        }
        else
        {
            throw new IllegalArgumentException("The Browser Type is

```

```
Undefined");
}

}

@Test
public void calculatepercent() throws IOException
{
    // Click on Math Calculators
    driver.findElement(By.xpath(".//*[@id='menu']/div[3]/a")).click();

    // Click on Percent Calculators
    driver.findElement(By.xpath(".//*[@id='menu']/div[4]/div[3]/a")).click();

    // Make use of augmented Driver to capture Screenshots.
    WebDriver augmentedDriver = new Augmenter().augment(driver);
    File screenshot =
        ((TakesScreenshot)augmentedDriver).getScreenshotAs(OutputType.FILE);

    FileUtils.copyFile(screenshot, new
        File("D:\\screenshots\\remotescreenshot1.jpg"));

    // Screenshot would be saved on the system where the script is
    // executed and NOT on remote machine.

    // Enter value 10 in the first number of the percent Calculator
    driver.findElement(By.id("cpar1")).sendKeys("10");

    // Enter value 50 in the second number of the percent Calculator
    driver.findElement(By.id("cpar2")).sendKeys("50");

    // Click Calculate Button
    driver.findElement(By.xpath(".//*[@id='content']/table/tbody
        /tr/td[2]/input")).click();
}
```

```
// Get the Result Text based on its xpath
String result =
driver.findElement(By.xpath("//*[@id='content']/p[2]
/span/font/b")).getText();

// Print a Log In message to the screen
System.out.println(" The Result is " + result);

if(result.equals("5"))
{
    System.out.println(" The Result is Pass");
}
else
{
    System.out.println(" The Result is Fail");
}

@AfterTest
public void closeBrowser()
{
    driver.quit();
}
```

Output

Upon executing the script, the screenshot is captured and saved in the specified location as shown below.



Capturing Videos

Sometimes we may not be able to analyze the failures just with the help of a log file or a screenshot. At times, it helps to capture the complete execution as a video. Let us understand how to capture videos.

We will make use of Monte Media Library to perform this operation.

Configuration

Step 1 : Navigate to the URL - <http://www.randelshofer.ch/monte/index.html> and download the screen recorder JAR as shown below.

Monte Media Library

The Monte Media Library is a Java library for processing media data. Supported media formats include still images, video, audio, and meta-data.

This is an *experimental* library for my personal studies!

This library has some overlap in functionality with the Java Media Framework (JMF). For some codecs, the Monte Media Library provides wrappers, which allow to use them with JMF. However, in general, Monte Media is not compatible with JMF.

Demo Applications

Most demos consist of an executable JAR file which also includes the source code. On some platforms, you can double click the JAR file to execute the demo, on others, you can start it with the command `java -jar nameofdemo.jar`

To get the source code, rename the file to `nameofdemo.zip` and unzip it.

Screen Recorder



Creates a screen recording with audio and stores it in a movie file.

The demo code is located in the package `org.monte.screenrecorder`

`MonteScreenRecorder.jar`

Click the launch button to start the recorder:



Movie Converter



Converts a movie from one file format into another.

This advanced demo demonstrates the use of codec chains, and the ability for processing stereo videos created with the Fujifilm Real 3D W1 camera.

Click the launch button to start the demo:



Movie Maker

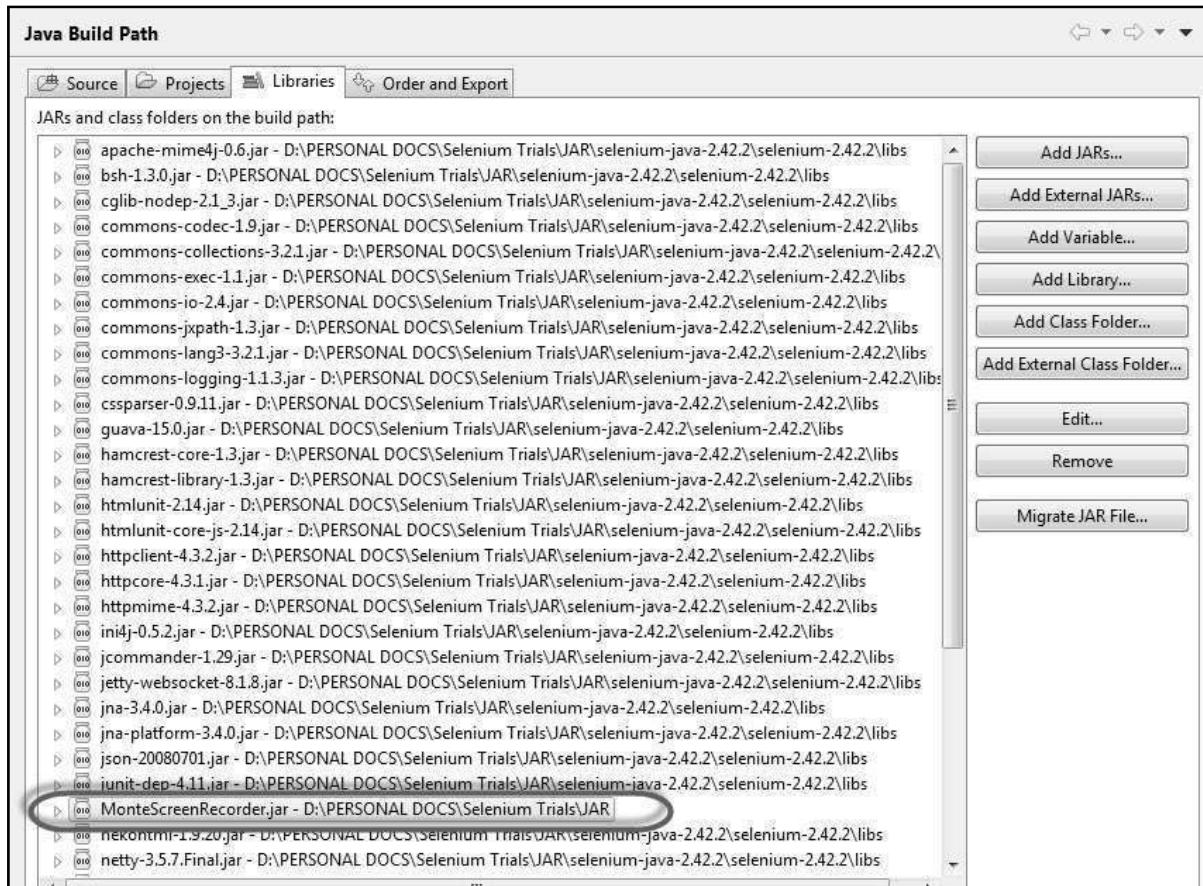


This demo creates a QuickTime movie from a folder with image files and an audio file.

The movie is written with the `QuickTimeWriter` class.

The demo code is located in the class `org.monte.moviemaker.MovieMakerMain`.
`MovieMaker.jar`

Step 2 : After downloading, add the JAR file to the Libraries of the current project.



Step 3 : We will use Java's AWT package to initialize the graphics configuration.

```
GraphicsConfiguration gc = GraphicsEnvironment
    .getLocalGraphicsEnvironment()
    .getDefaultScreenDevice()
    .getDefaultConfiguration();
```

Step 4 : An instance of ScreenRecorder is created which takes the following parameters.

Parameter	Description
GraphicsConfiguration	Provides information about the display screen such as size and resolution.

Video and compression format	The output format (AVI) of the movie with number of frames/sec.
Color of the mouse cursor and refresh rate	Specifies the mouse cursor color and refresh rate.
Audio format	If 'NULL', audio will NOT be recorded.

Example

We will capture a video of the simple test execution - percent calculation.

```

import java.io.File;
import java.io.IOException;
import java.util.concurrent.TimeUnit;

import org.apache.commons.io.FileUtils;

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

import org.monte.media.math.Rational;
import org.monte.media.Format;
import org.monte.screenrecorder.ScreenRecorder;

import static org.monte.media.AudioFormatKeys.*;
import static org.monte.media.VideoFormatKeys.*;
import java.awt.*;

public class webdriverdemo
{

```

```

private static ScreenRecorder screenRecorder;
public static void main(String[] args) throws IOException, AWTException
{
    GraphicsConfiguration gconfig = GraphicsEnvironment
        .getLocalGraphicsEnvironment()
        .getDefaultScreenDevice()
        .getDefaultConfiguration();

    screenRecorder = new ScreenRecorder(gconfig,
        new Format(MediaTypeKey, MediaType.FILE, MimeTypeKey,
        MIME_AVI),
        new Format(MediaTypeKey, MediaType.VIDEO, EncodingKey,
        ENCODING_AVI_TECHSMITH_SCREEN_CAPTURE,
        CompressorNameKey, ENCODING_AVI_TECHSMITH_SCREEN_CAPTURE,
        DepthKey, (int)24, FrameRateKey, Rational.valueOf(15),
        QualityKey, 1.0f,
        KeyFrameIntervalKey, (int) (15 * 60)),
        new Format(MediaTypeKey, MediaType.VIDEO,
        EncodingKey, "black",
        FrameRateKey, Rational.valueOf(30)), null);

    WebDriver driver = new FirefoxDriver();

    // Start Capturing the Video
    screenRecorder.start();

    // Puts an Implicit wait, will wait for 10 seconds
    // before throwing exception
    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

    // Launch website
    driver.navigate().to("http://www.calculator.net/");
}

```

```

// Maximize the browser
driver.manage().window().maximize();

// Click on Math Calculators
driver.findElement(By.xpath(".//*[@id='menu']/div[3]/a")).click();

// Click on Percent Calculators
driver.findElement(By.xpath(".//*[@id='menu']/div[4]/div[3]/a"))
.click();

// Enter value 10 in the first number of the percent Calculator
driver.findElement(By.id("cpar1")).sendKeys("10");

// Enter value 50 in the second number of the percent Calculator
driver.findElement(By.id("cpar2")).sendKeys("50");

// Click Calculate Button
driver.findElement(By.xpath(".//*[@id='content']/table
/tbody/tr/td[2]/input")).click();

// Get the Result Text based on its xpath
String result =
driver.findElement(By.xpath(".//*[@id='content']
/p[2]/span/font/b")).getText();

File screenshot =
((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

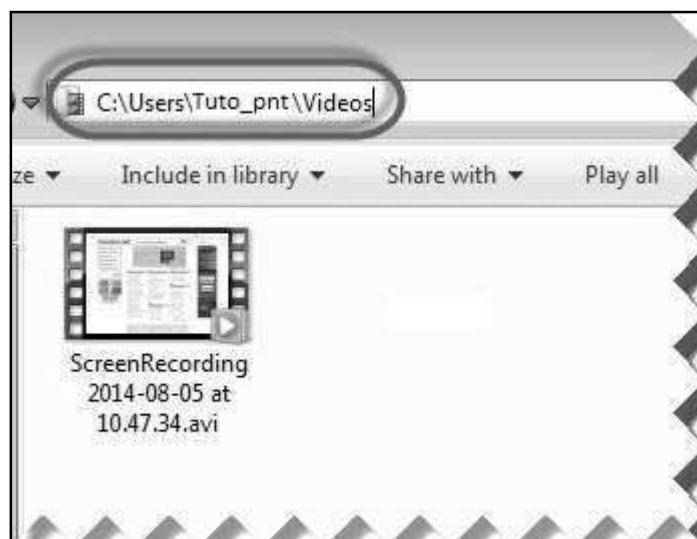
FileUtils.copyFile(screenshot, new
File("D:\\screenshots\\screenshots1.jpg"));

```

```
// Print a Log In message to the screen  
System.out.println(" The Result is " + result);  
  
// Close the Browser.  
driver.close();  
  
// Stop the ScreenRecorder  
screenRecorder.stop();  
}  
}
```

Output

The recorded video is saved in the "C:\users\<<UserName>>\Videos" folder as shown below.



10. TestNG

What is TestNG?

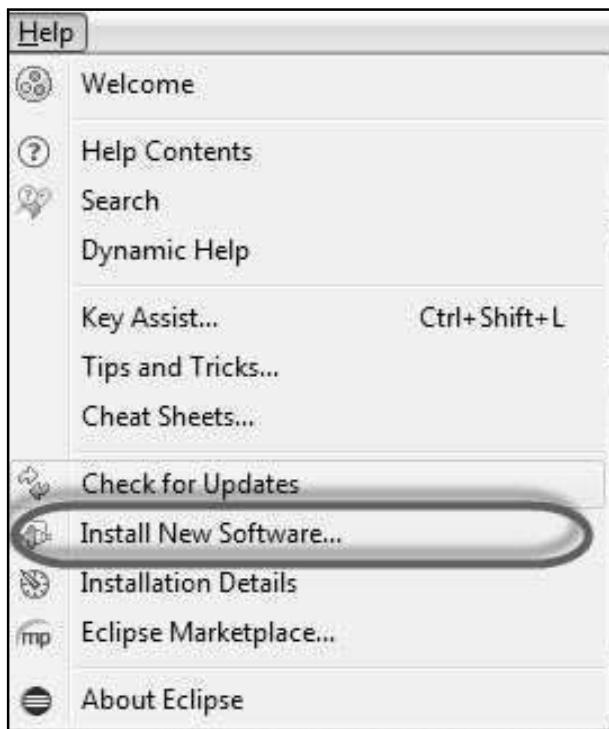
TestNG is a powerful testing framework, an enhanced version of JUnit which was in use for a long time before TestNG came into existence. NG stands for 'Next Generation'.

TestNG framework provides the following features:

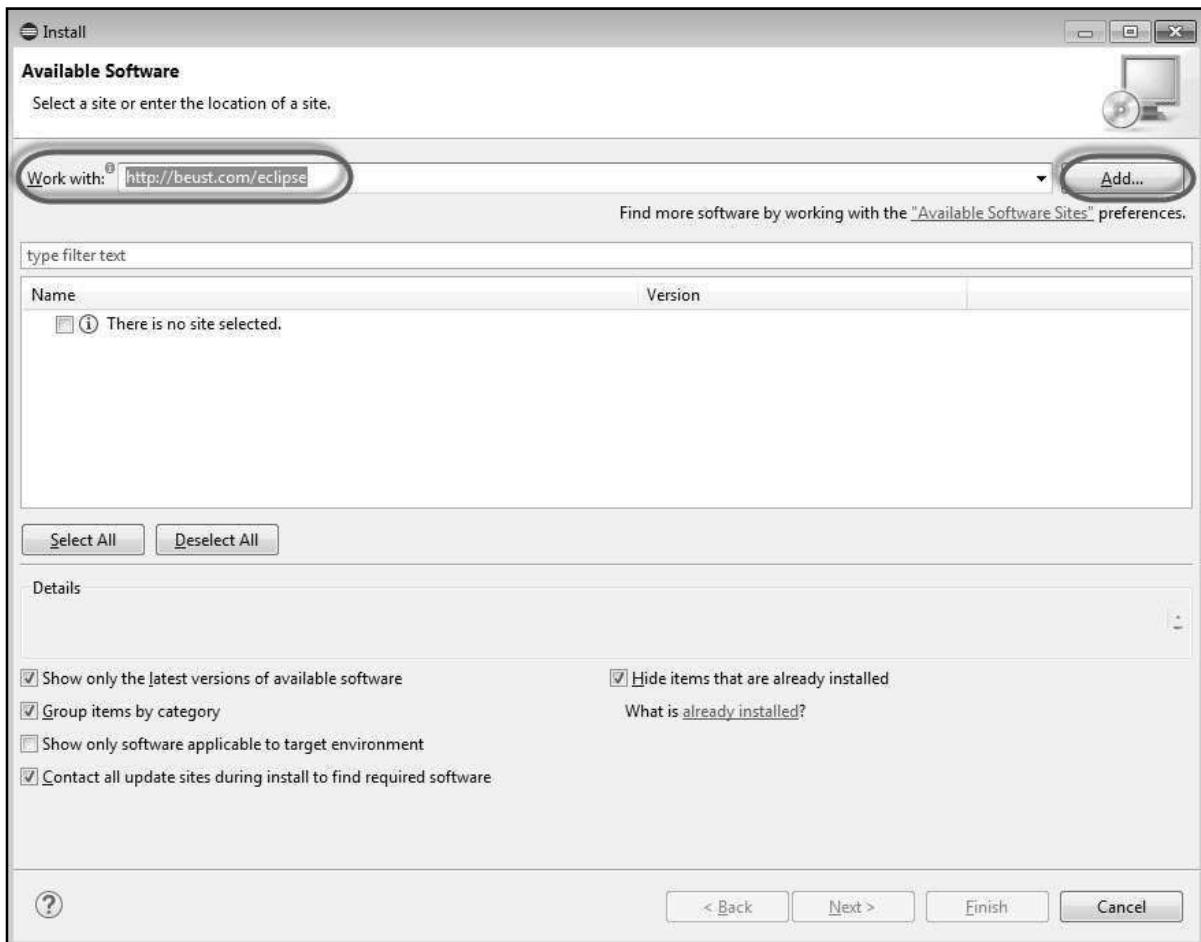
- Annotations help us organize the tests easily.
- Flexible test configuration.
- Test cases can be grouped more easily.
- Parallelization of tests can be achieved using TestNG.
- Support for data-driven testing.
- Inbuilt reporting

Installing TestNG for Eclipse

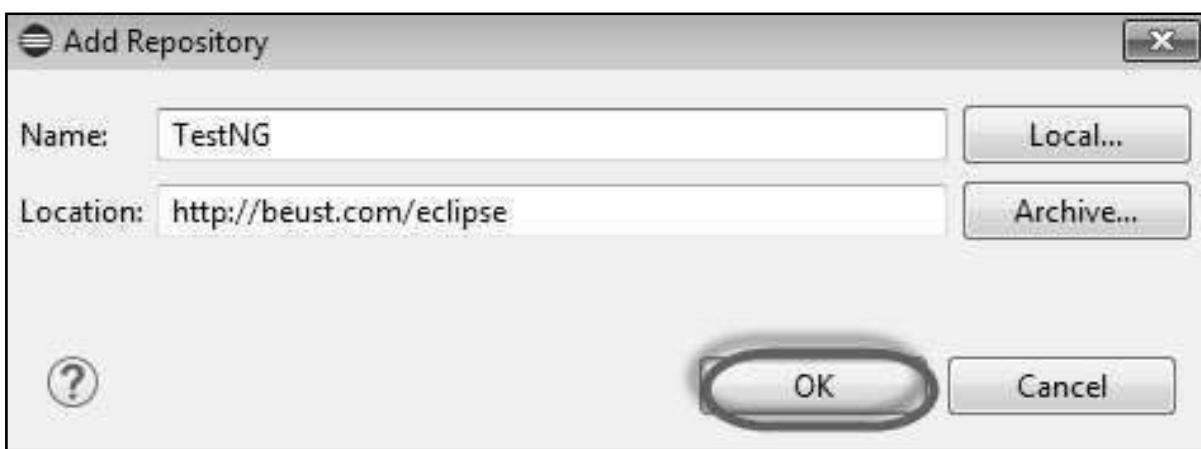
Step 1 : Launch Eclipse and select 'Install New Software'.



Step 2 : Enter the URL as 'http://beust.com/eclipse' and click 'Add'.



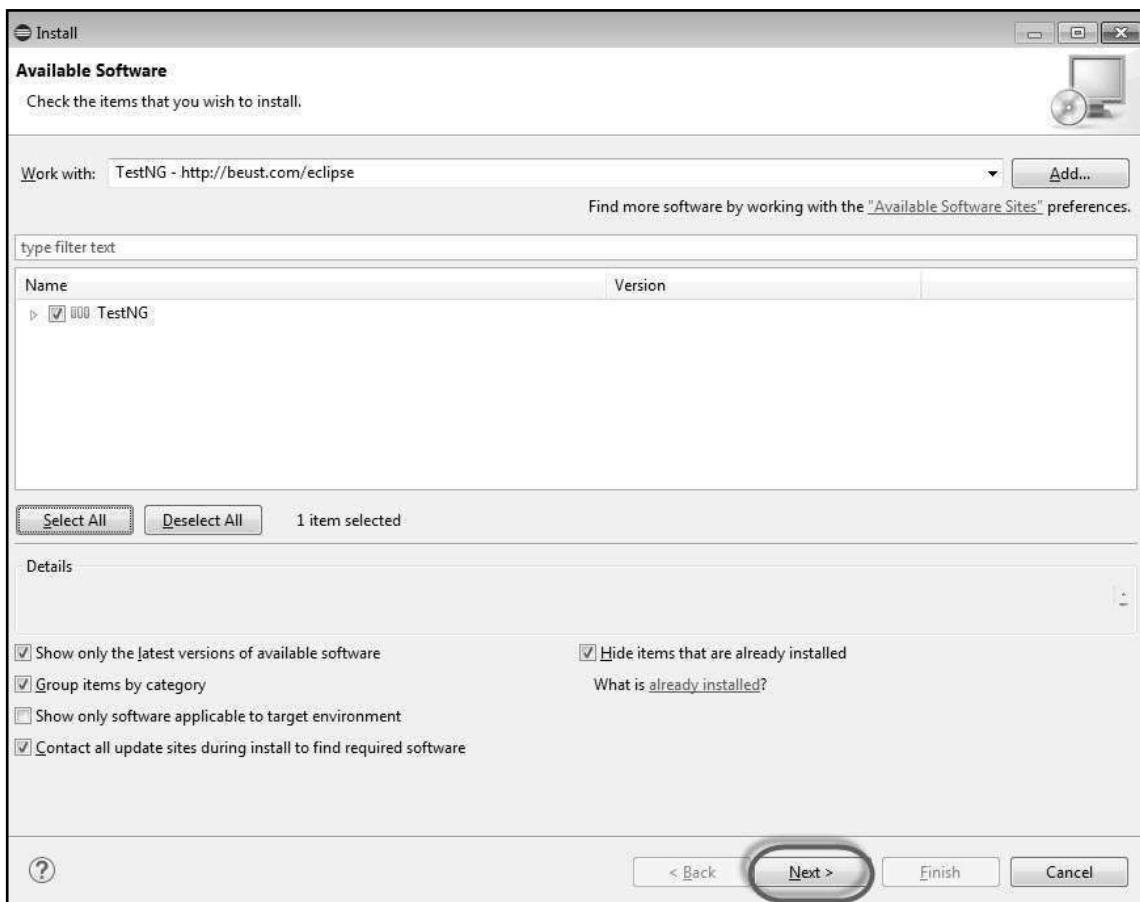
Step 3 : The dialog box 'Add Repository' opens. Enter the name as 'TestNG' and click 'OK'



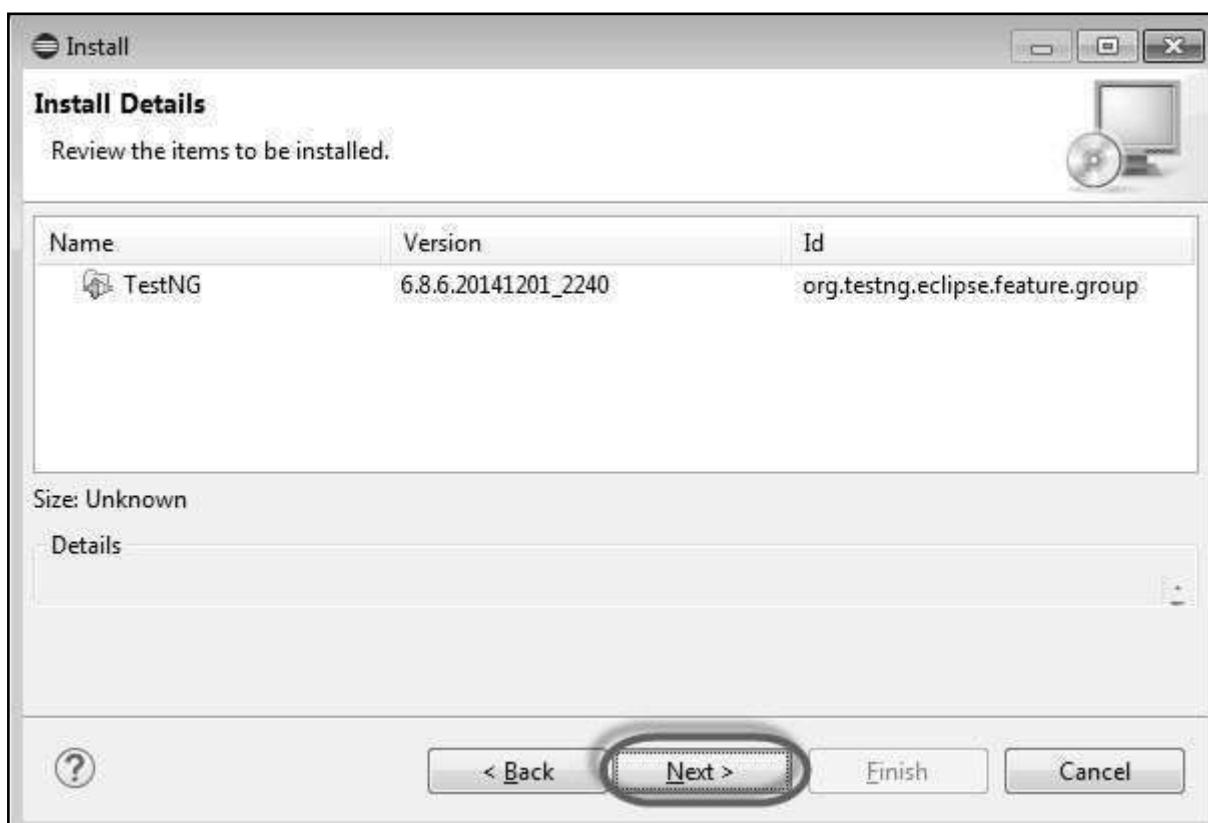
Step 4 : Click 'Select All' and 'TestNG' would be selected as shown in the figure.



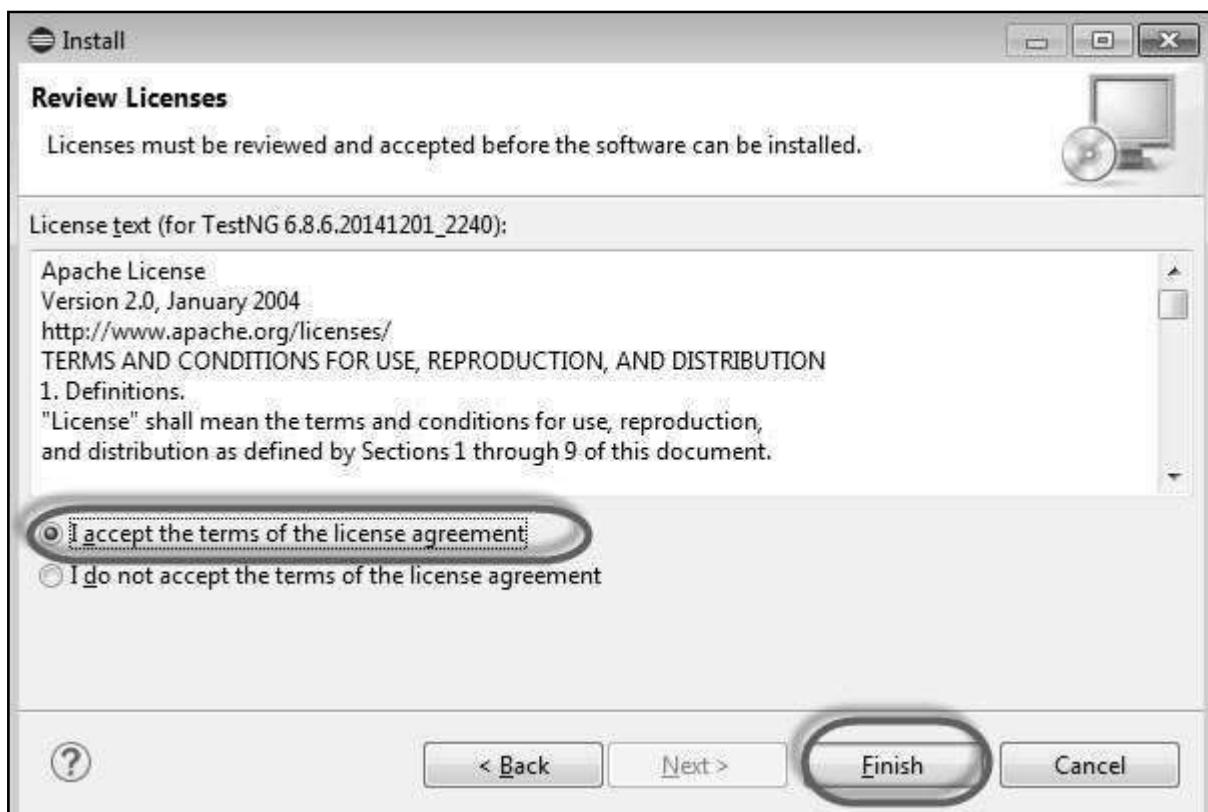
Step 5 : Click 'Next' to continue.



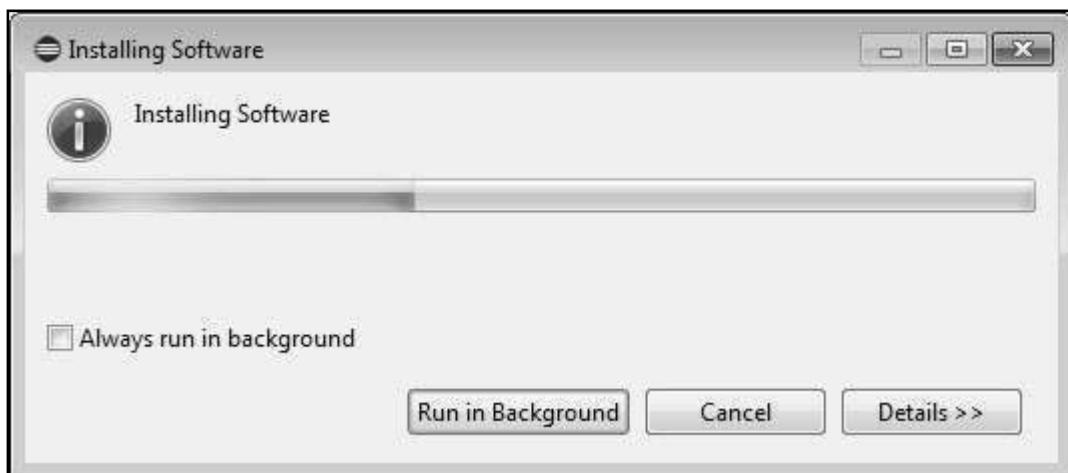
Step 6 : Review the items that are selected and click 'Next'.



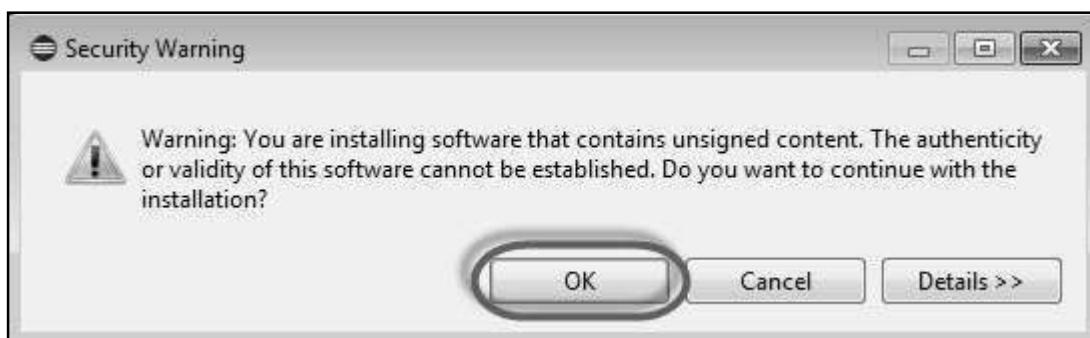
Step 7 : "Accept the License Agreement" and click 'Finish'.



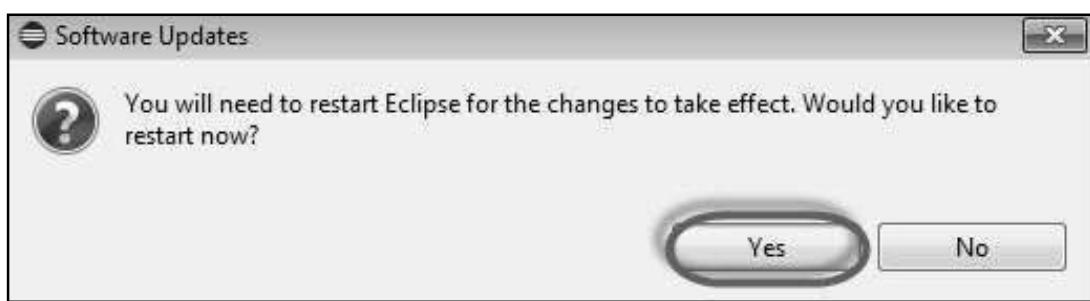
Step 8 : TestNG starts installing and the progress would be shown follows.



Step 9 : Security Warning pops up as the validity of the software cannot be established. Click 'Ok'.



Step 10 : The Installer prompts to restart Eclipse for the changes to take effect. Click 'Yes'.



Annotations in TestNG

Annotations were formally added to the Java language in JDK 5 and TestNG made the choice to use annotations to annotate test classes. Following are some of the benefits of using annotations. More about TestNG can be found [here](#).

- TestNG identifies the methods it is interested in by looking up annotations. Hence, method names are not restricted to any pattern or format.

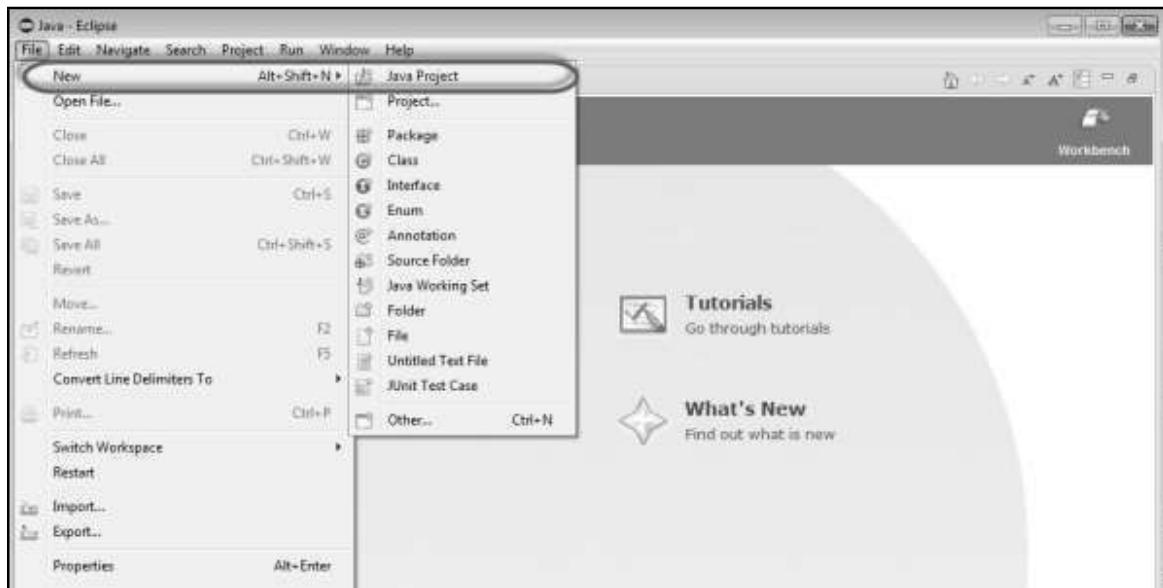
- We can pass additional parameters to annotations.
- Annotations are strongly typed, so the compiler will flag any mistakes right away.
- Test classes no longer need to extend anything (such as TestCase, for JUnit 3).

Annotation	Description
@BeforeSuite	The annotated method will be run only once before all the tests in this suite have run.
@AfterSuite	The annotated method will be run only once after all the tests in this suite have run.
@BeforeClass	The annotated method will be run only once before the first test method in the current class is invoked.
@AfterClass	The annotated method will be run only once after all the test methods in the current class have run.
@BeforeTest	The annotated method will be run before any test method belonging to the classes inside the <test> tag is run.
@AfterTest	The annotated method will be run after all the test methods belonging to the classes inside the <test> 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

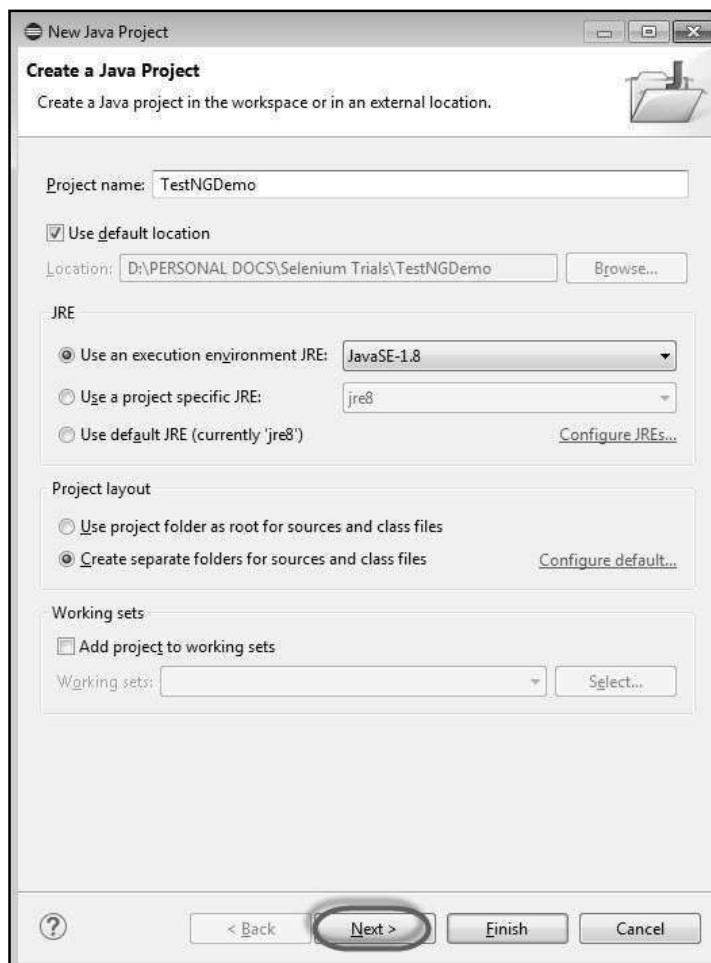
	after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.
@BeforeMethod	The annotated method will be run before each test method.
@AfterMethod	The annotated method will be run after each test method.
@DataProvider	Marks a method as supplying data for a test method. The annotated method must return an Object[][] where each Object[] can be assigned the parameter list of the test method. The @Test method that wants to receive data from this DataProvider needs to use a dataProvider name equals to the name of this annotation.
@Factory	Marks a method as a factory that returns objects that will be used by TestNG as Test classes. The method must return Object[].
@Listeners	Defines listeners on a test class.
@Parameters	Describes how to pass parameters to a @Test method.
@Test	Marks a class or a method as part of the test.

TestNG-Eclipse Setup

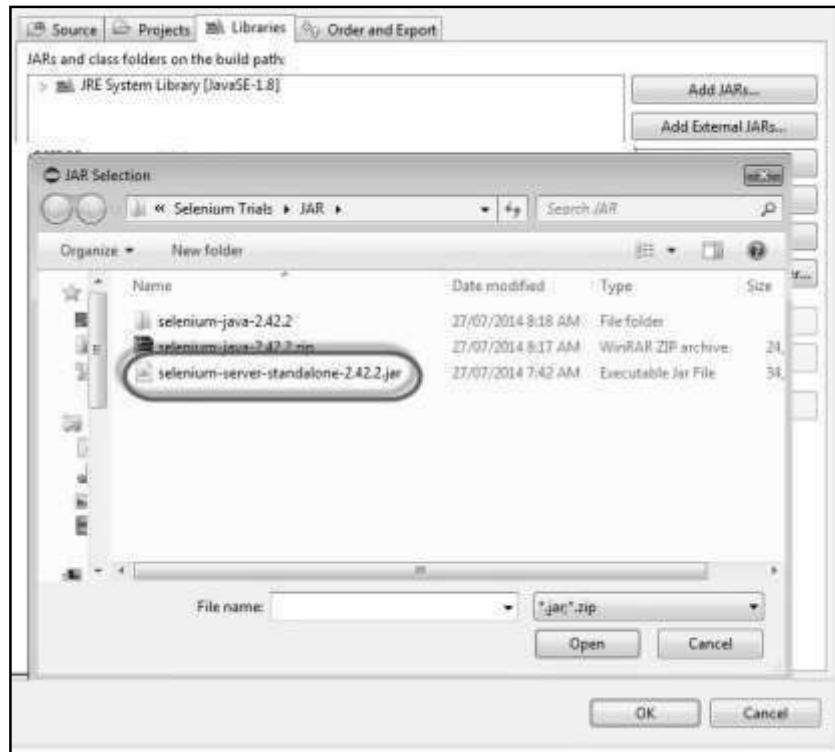
Step 1 : Launch Eclipse and create a 'New Java Project' as shown below.



Step 2 : Enter the project name and click 'Next'.



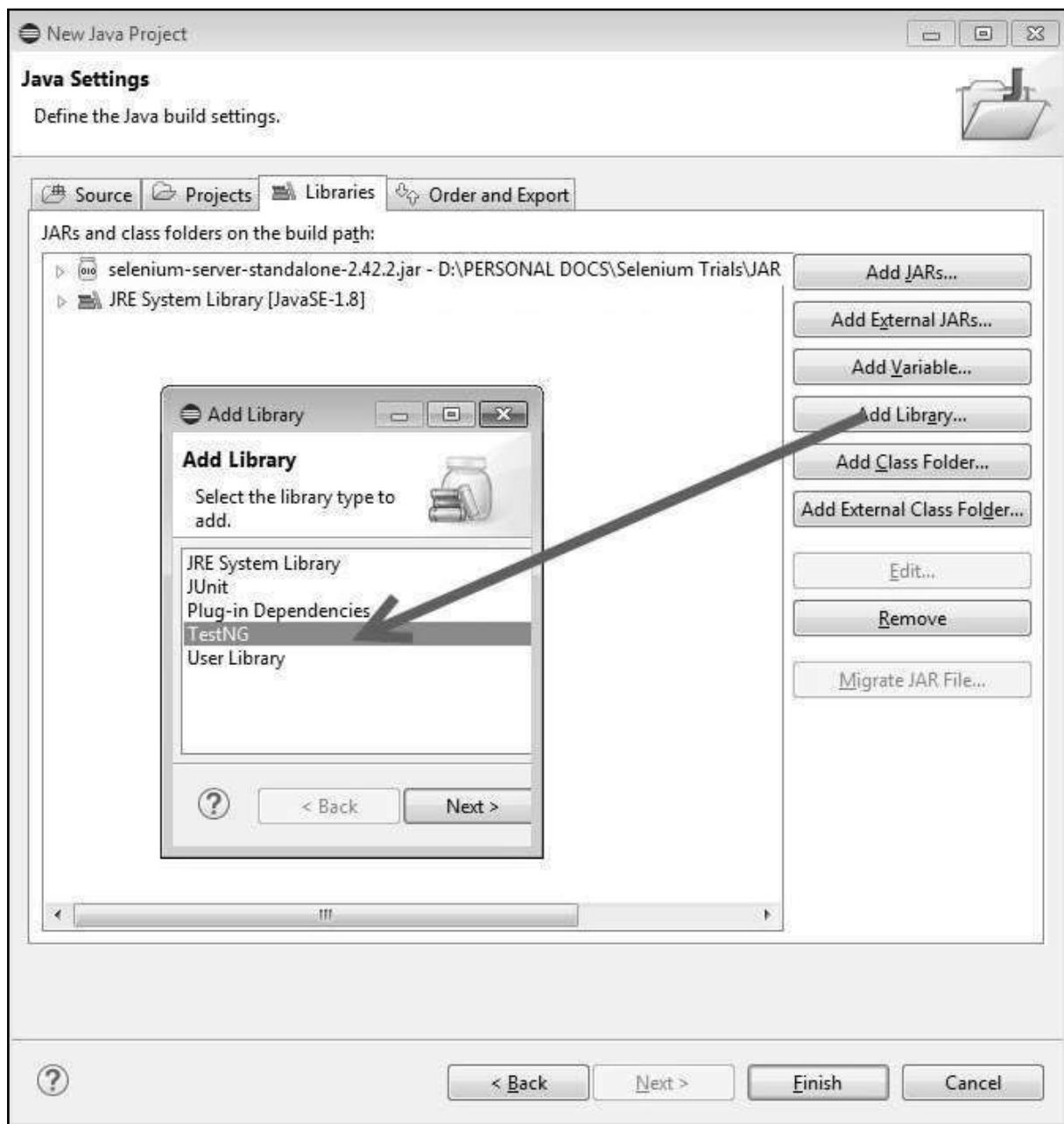
Step 3 : Navigate to "Libraries" Tab and add the Selenium Remote Control Server JAR file by clicking on "Add External JAR's" as shown below.



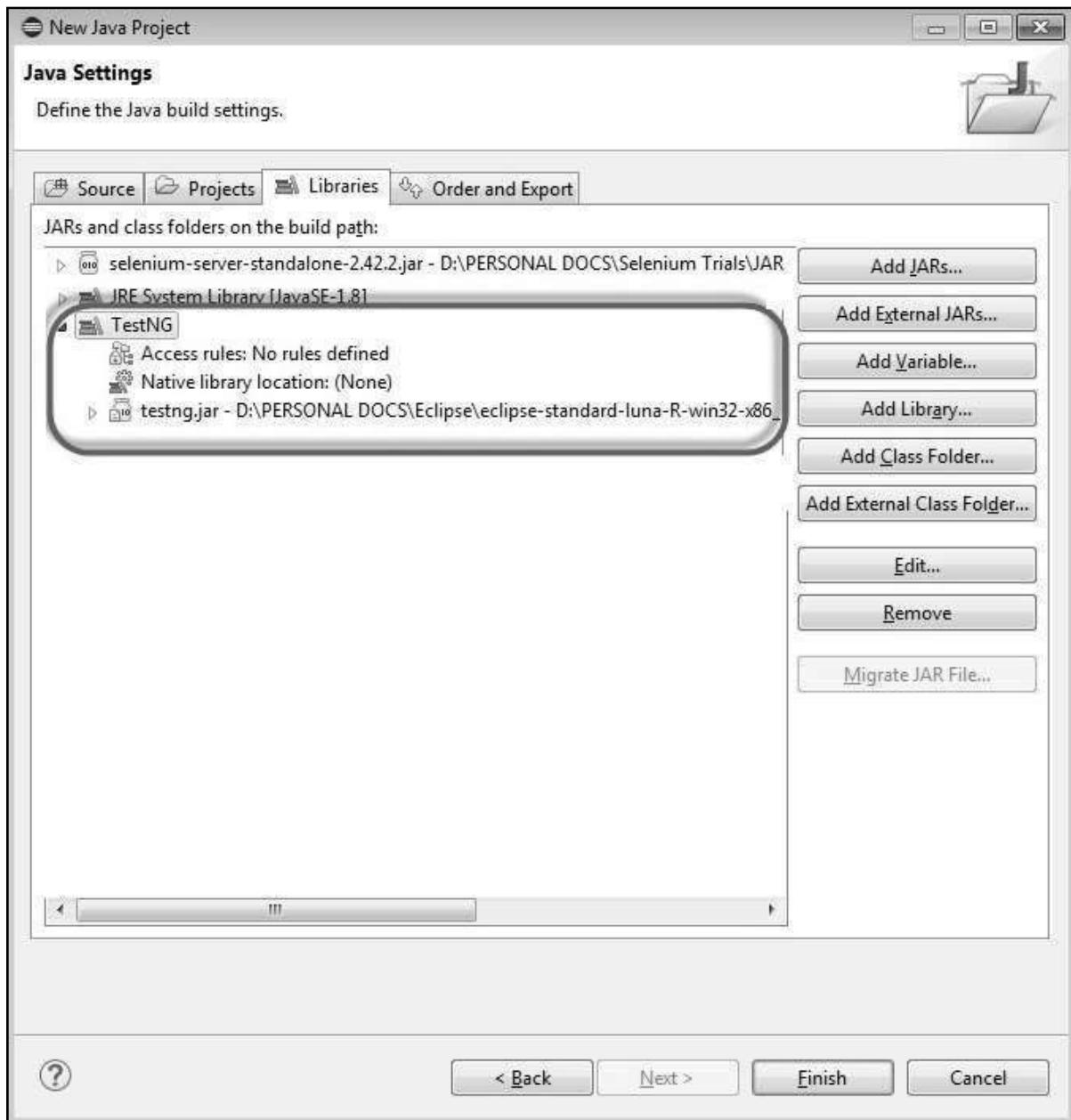
Step 4 : The added JAR file is shown here. Click 'Add Library'.



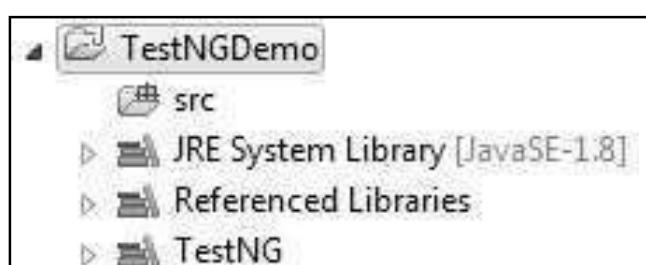
Step 5 : The 'Add Library' dialog opens. Select 'TestNG' and click 'Next' in the 'Add Library' dialog box.



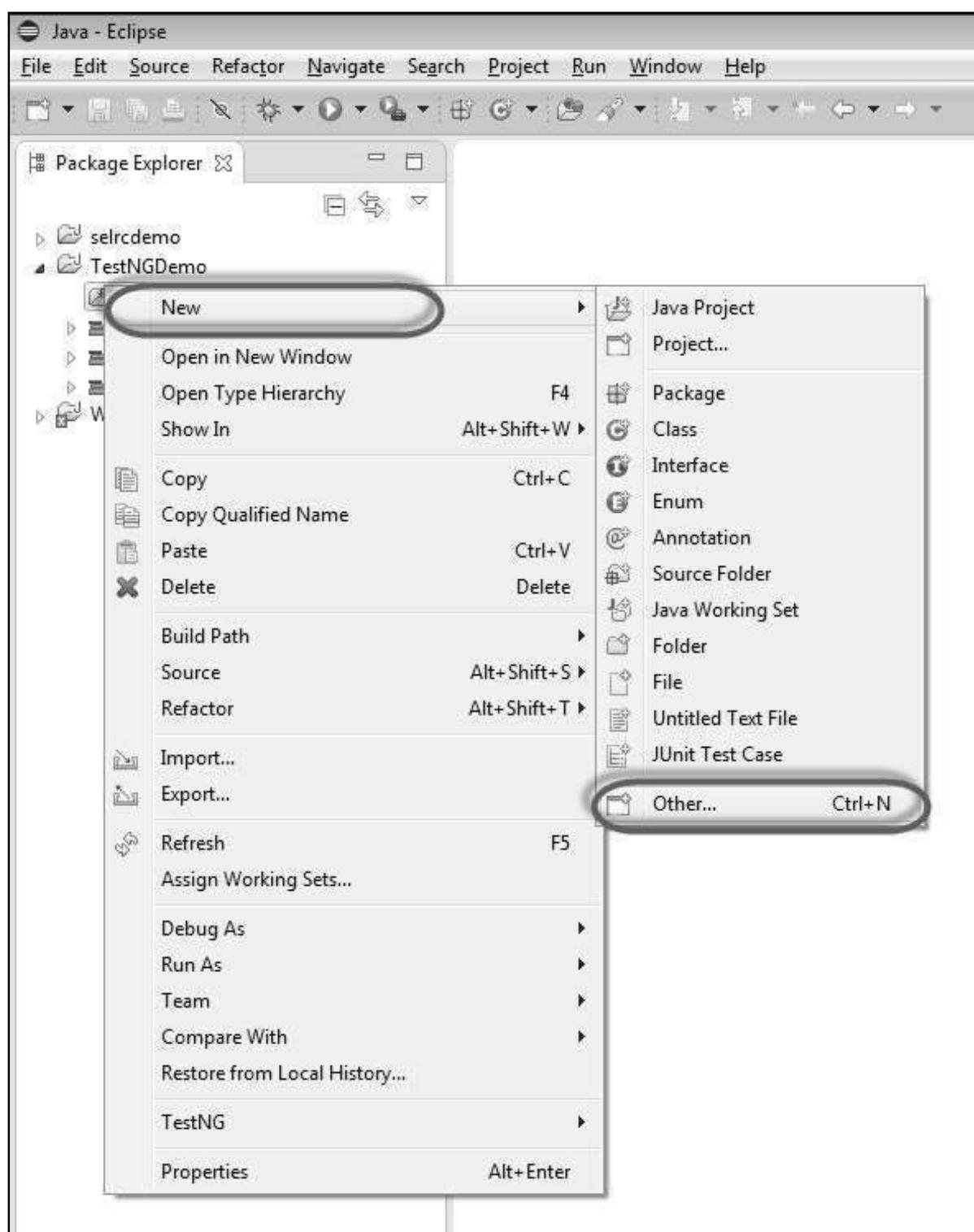
Step 6 : The added 'TestNG' Library is added and it is displayed as shown below.



Step 7 : Upon creating the project, the structure of the project would be as shown below.



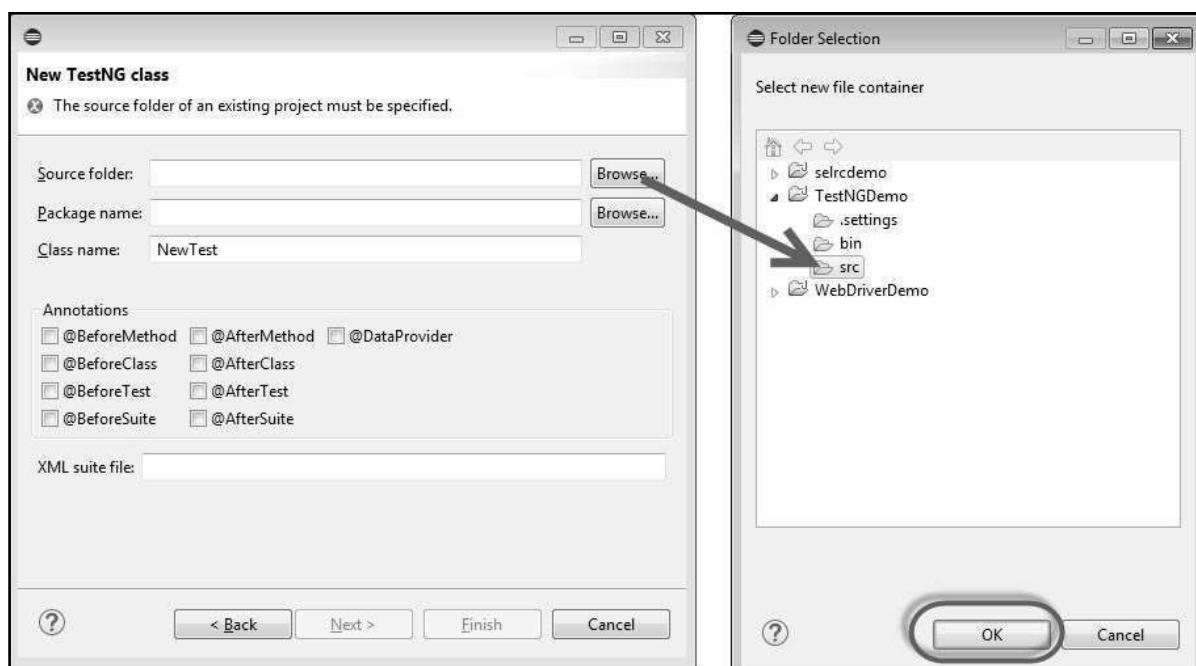
Step 8 : Right-click on 'src' folder and select New >> Other.



Step 9 : Select 'TestNG' and click 'Next'.



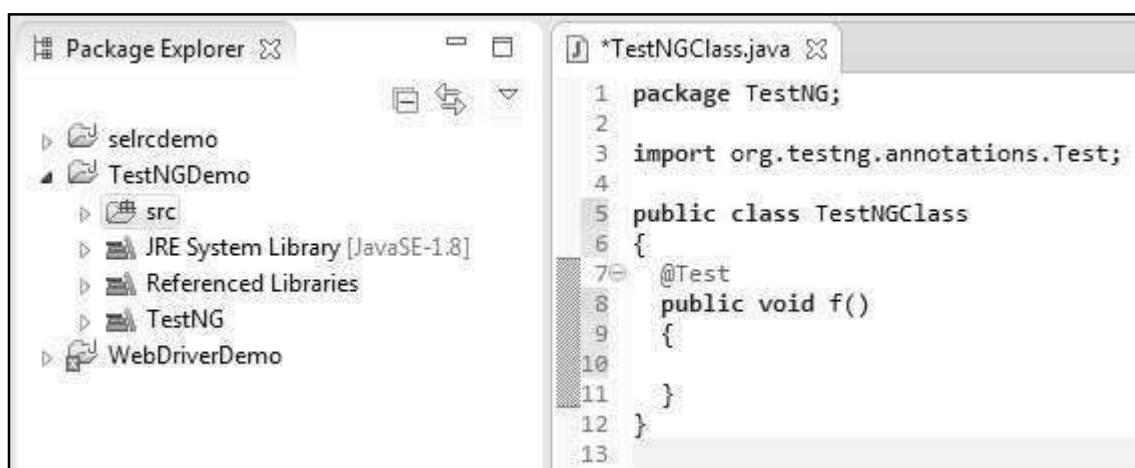
Step 10 : Select the 'Source Folder' name and click 'Ok'.



Step 11 : Select the 'Package name', the 'class name', and click 'Finish'.



Step 12 : The Package explorer and the created class would be displayed.



First Test in TestNG

Now let us start scripting using TestNG. Let us script for the same example that we used for understanding the WebDriver. We will use the demo application, www.calculator.net, and perform percent calculator.

In the following test, you will notice that there is NO main method, as testNG will drive the program execution flow. After initializing the driver, it will execute the '@BeforeTest' method followed by '@Test' and then '@AfterTest'. Please note that there can be any number of '@Test' annotation in a class but '@BeforeTest' and '@AfterTest' can appear only once.

```
package TestNG;

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class TestNGClass
{
    WebDriver driver = new FirefoxDriver();

    @BeforeTest
    public void launchapp()
    {
        //Puts an Implicit wait, Will wait for 10 seconds
        // before throwing exception
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        //Launch website
        driver.navigate().to("http://www.calculator.net");
        driver.manage().window().maximize();
    }

    @Test
    public void calculatepercent()
```

```

{

    // Click on Math Calculators
    driver.findElement(By.xpath("//*[@id='menu']/div[3]/a")).click();

    // Click on Percent Calculators
    driver.findElement(By.xpath("//*[@id='menu']/div[4]/div[3]/a")).click();

    // Enter value 10 in the first number of the percent Calculator
    driver.findElement(By.id("cpar1")).sendKeys("10");

    // Enter value 50 in the second number of the percent Calculator
    driver.findElement(By.id("cpar2")).sendKeys("50");

    // Click Calculate Button
    driver.findElement(By.xpath("//*[@id='content']/table
/tbody/tr/td[2]/input")).click();

    // Get the Result Text based on its xpath
    String result =
        driver.findElement(By.xpath("//*[@id='content']/p[2]
/span/font/b")).getText();

    // Print a Log In message to the screen
    System.out.println(" The Result is " + result);

    if(result.equals("5"))
    {
        System.out.println(" The Result is Pass");
    }
    else
    {
        System.out.println(" The Result is Fail");
    }
}

```

```

    }

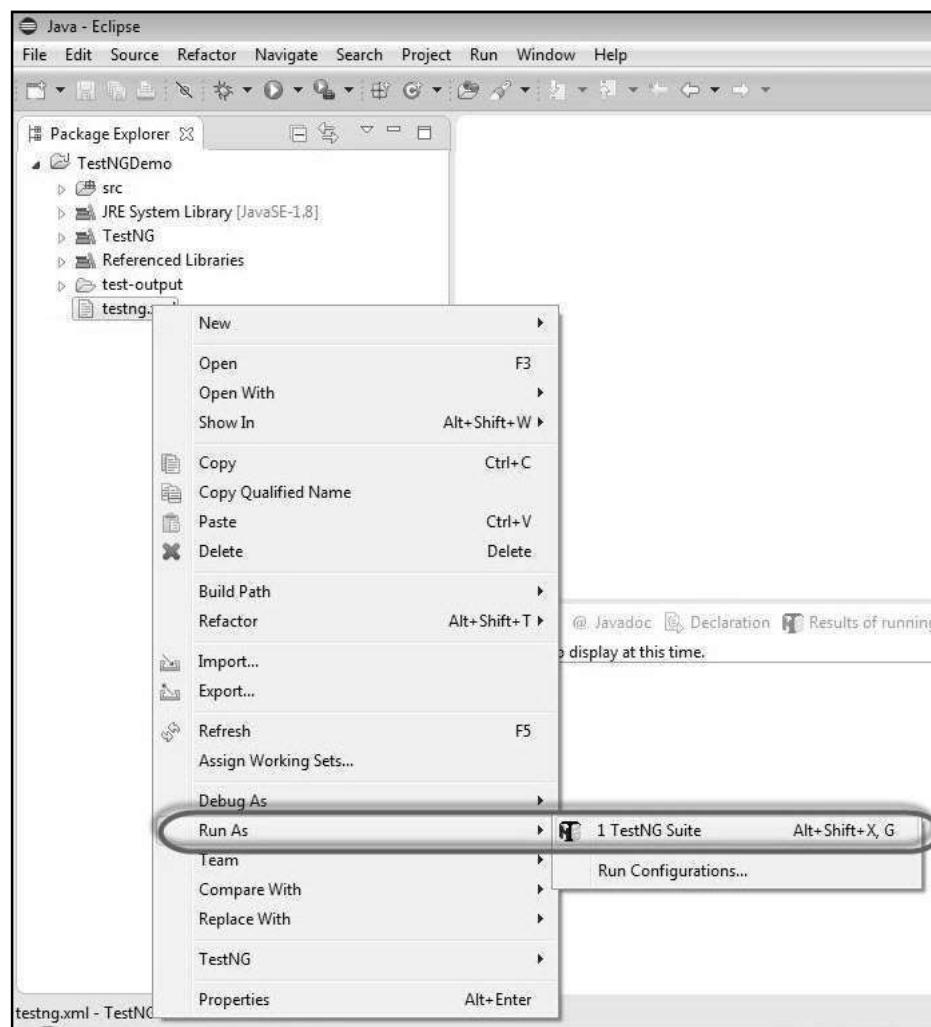
}

@AfterTest
public void terminateTest()
{
    driver.close();
}
}

```

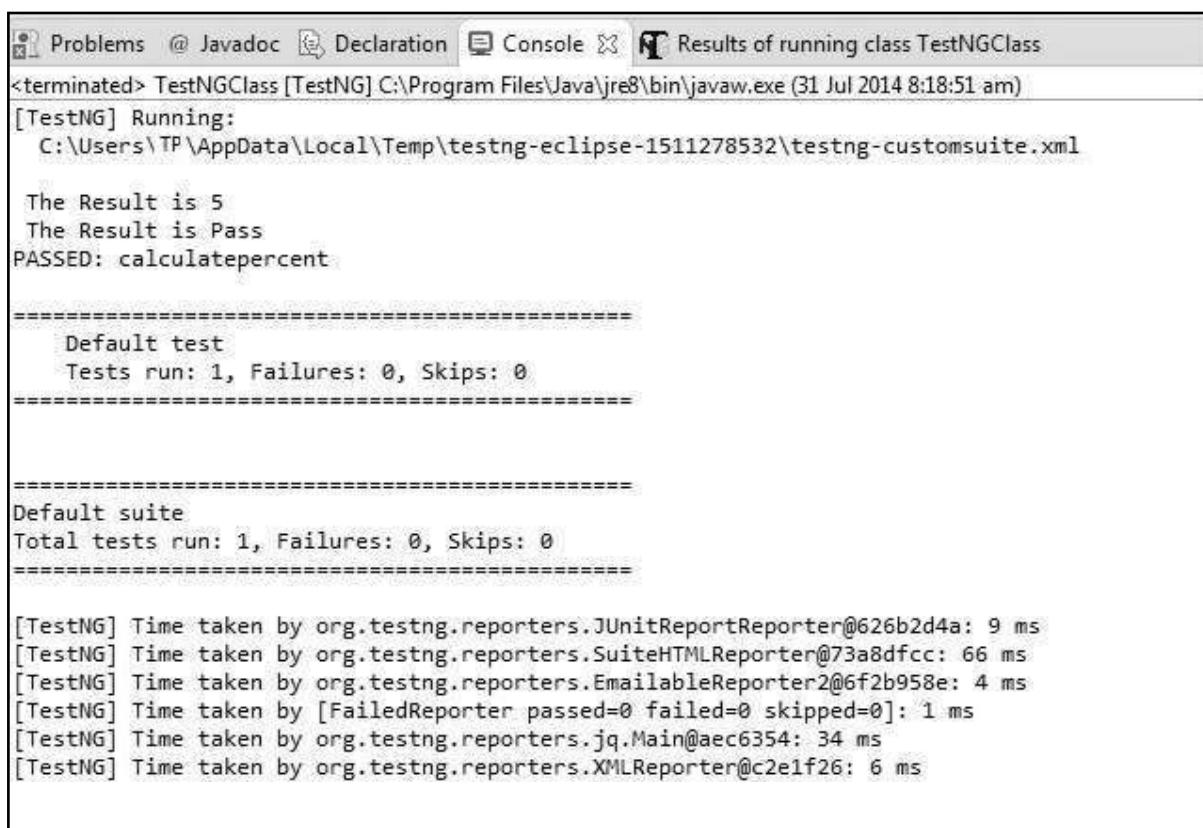
Execution

To execute, right-click on the created XML and select "Run As" >> "TestNG Suite"



Result Analysis

The output is thrown to the console and it would appear as shown below. The console output also has an execution summary.



```

Problems @ Javadoc Declaration Console Results of running class TestNGClass
<terminated> TestNGClass [TestNG] C:\Program Files\Java\jre8\bin\javaw.exe (31 Jul 2014 8:18:51 am)
[TestNG] Running:
C:\Users\TP\AppData\Local\Temp\testng-eclipse-1511278532\testng-customsuite.xml

The Result is 5
The Result is Pass
PASSED: calculatepercent

=====
Default test
Tests run: 1, Failures: 0, Skips: 0
=====

=====
Default suite
Total tests run: 1, Failures: 0, Skips: 0
=====

[TestNG] Time taken by org.testng.reporters.JUnitReportReporter@626b2d4a: 9 ms
[TestNG] Time taken by org.testng.reporters.SuiteHTMLReporter@73a8dfcc: 66 ms
[TestNG] Time taken by org.testng.reporters.EmailableReporter2@6f2b958e: 4 ms
[TestNG] Time taken by [FailedReporter passed=0 failed=0 skipped=0]: 1 ms
[TestNG] Time taken by org.testng.reporters.jq.Main@aec6354: 34 ms
[TestNG] Time taken by org.testng.reporters.XMLReporter@c2e1f26: 6 ms

```

The result of TestNG can also be seen in a different tab. Click on 'HTML Report View' button as shown below.



The HTML result would be displayed as shown below.



11. SELENIUM GRID

Selenium Grid is a tool that distributes the tests across multiple physical or virtual machines so that we can execute scripts in parallel (simultaneously). It dramatically accelerates the testing process across browsers and across platforms by giving us quick and accurate feedback.

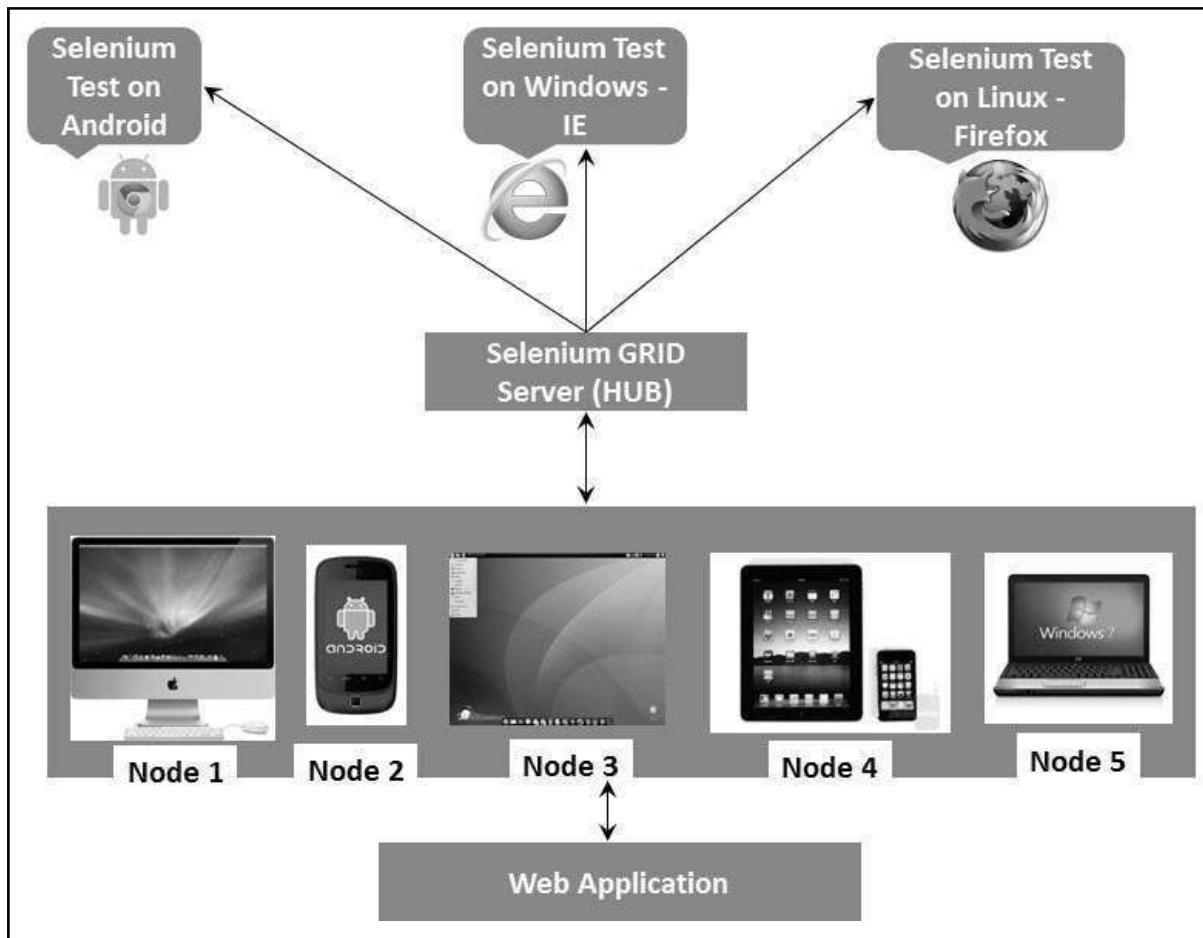
Selenium Grid allows us to execute multiple instances of WebDriver or Selenium Remote Control tests in parallel which uses the same code base, hence the code need NOT be present on the system they execute. The selenium-server-standalone package includes Hub, WebDriver, and Selenium RC to execute the scripts in grid.

Selenium Grid has a Hub and a Node.

- **Hub** - The hub can also be understood as a server which acts as the central point where the tests would be triggered. A Selenium Grid has only one Hub and it is launched on a single machine once.
- **Node** - Nodes are the Selenium instances that are attached to the Hub which execute the tests. There can be one or more nodes in a grid which can be of any OS and can contain any of the Selenium supported browsers.

Architecture

The following diagram shows the architecture of Selenium Grid.



Working with Grid

In order to work with the Grid, we need to follow certain protocols. Listed below are the major steps involved in this process:

- Configuring the Hub
- Configuring the Nodes
- Develop the Script and Prepare the XML File
- Test Execution
- Result Analysis

Let us discuss each of these steps in detail.

Configuring the Hub

Step 1 : Download the latest Selenium Server standalone JAR file from <http://docs.seleniumhq.org/download/>. Download it by clicking on the version as shown below.

SeleniumHQ
Browser Automation

edit this page search selenium: Go

Projects Download Documentation Support About

Downloads

Below is where you can find the latest releases of all the Selenium components. You can also find a list of [previous releases](#), [source code](#), and additional information for [Maven users](#) (Maven is a popular Java build tool).

Selenium IDE

Selenium IDE is a Firefox plugin which records and plays back user interactions with the browser. Use this to either create simple scripts or assist in exploratory testing. It can also export Remote Control or WebDriver scripts, though they tend to be somewhat brittle and should be overhauled into some sort of Page Object-y structure for any kind of resiliency.

Download latest released version [2.5.0](#) released on 01/Jan/2014 or view the [Release Notes](#) and then [install some plugins](#).

Selenium Server (formerly the Selenium RC Server)

The Selenium Server is needed in order to run either Selenium RC style scripts or Remote Selenium Webdriver ones. The 2.x server is a drop-in replacement for the old Selenium RC server and is designed to be backwards compatible with your existing infrastructure.

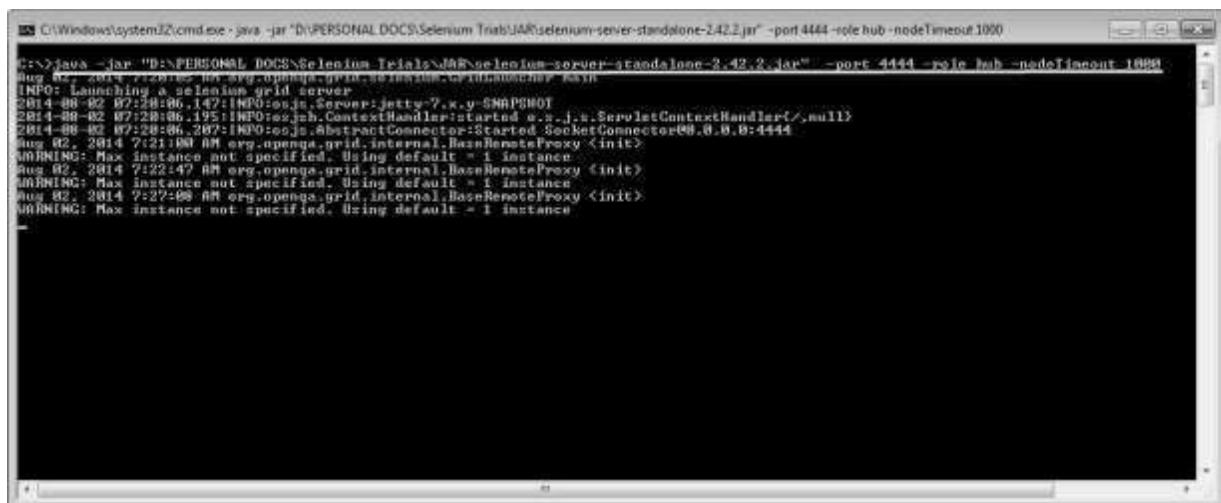
Download version [2.42.2](#)

To use the Selenium Server in a Grid configuration [see the wiki page](#).

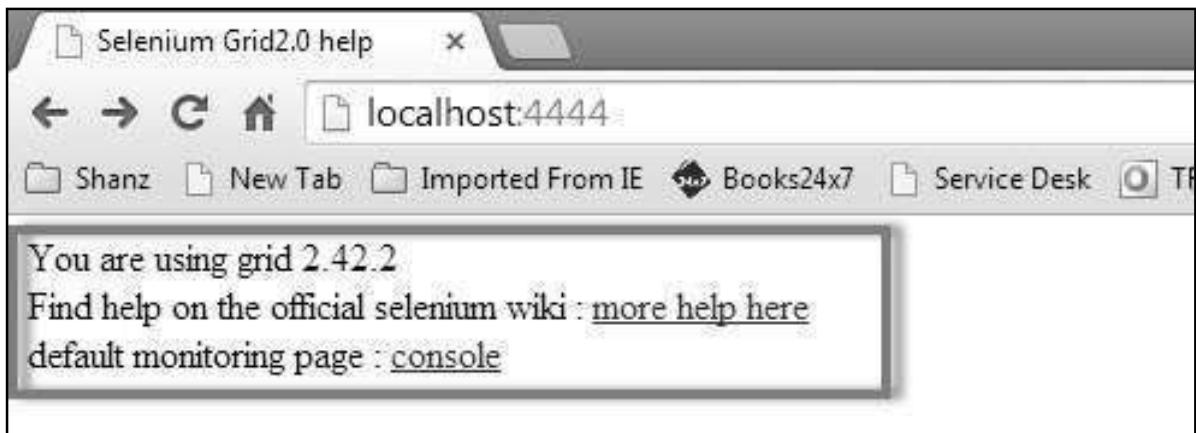
Step 2 : Start the Hub by launching the Selenium Server using the following command. Now we will use the port '4444' to start the hub.

Note : Ensure that there are no other applications that are running on port# 4444.

```
java -jar selenium-server-standalone-2.25.0.jar -port 4444 -role hub -nodeTimeout 1000
```



Step 3 : Now open the browser and navigate to the URL <http://localhost:4444> from the Hub (The system where you have executed Step#2).



Step 4 : Now click on the 'console' link and click 'view config'. The config of the hub would be displayed as follows. As of now, we haven't got any nodes, hence we will not be able to see the details.



Configuring the Nodes

Step 1 : Logon to the node (where you would like to execute the scripts) and place the 'selenium-server-standalone-2.42.2' in a folder. We need to point to the selenium-server-standalone JAR while launching the nodes.

Step 2 : Launch FireFox Node using the following command.

```
java -jar D:\JAR\selenium-server-standalone-2.42.2.jar -role node -hub http://10.30.217.157:4444/grid/register -browser browserName=firefox -port 5555
```

Where,

D:\JAR\selenium-server-standalone-2.42.2.jar = Location of the Selenium Server Standalone Jar File(on the Node Machine)

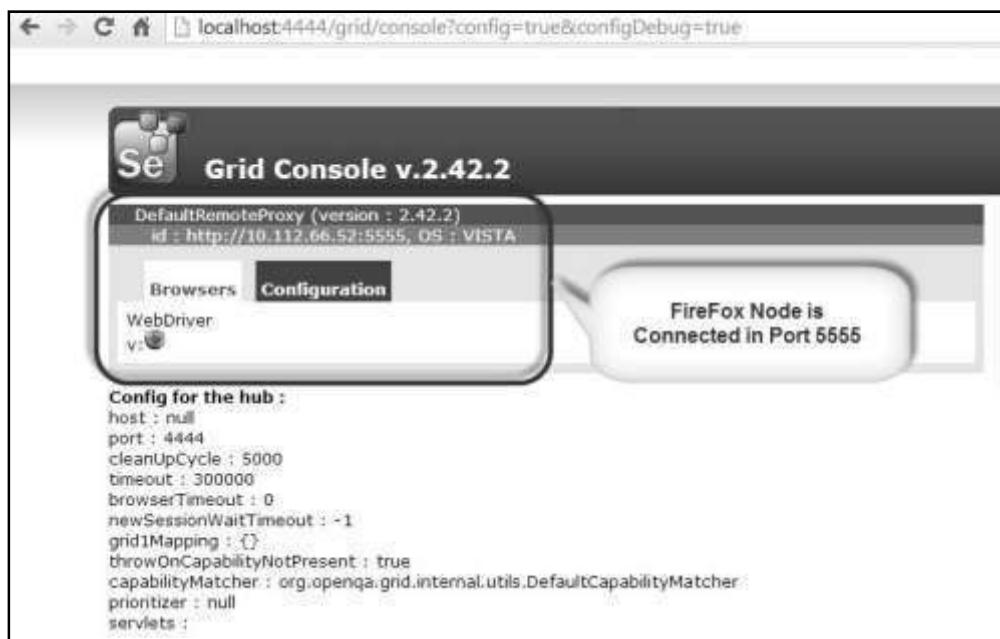
http://10.30.217.157:4444 = IP Address of the Hub and 4444 is the port of the Hub

browserName = firefox (Parameter to specify the Browser name on Nodes)

5555 = Port on which Firefox Node would be up and running.

```
Administrator: C:\Windows\system32\cmd.exe java -jar D:\JAR\selenium-server-standalone-2.42.2.jar -role node -hub http://10.30.217.157:4444/grid/register -browser browserName=firefox -port 5555
INFO: launching a selenium grid node
INFO: 02-28 2014 21:39:40 AM org.openqa.grid.common.RegistrationRequest addCapabilityFromString
INFO: Adding browserName=firefox
[7-29-14 21:39:41,054 INFO] [Grid]: Grid node started
[7-29-14 21:39:41,074 INFO] [Grid]: Grid node v2.42.2. Built from revision 6a6995d
[7-29-14 21:39:41,085 INFO] [Grid]: RemoteWebDriver instances should connect to: http://10.30.217.157:5555/wd/hub
[7-29-14 21:39:42,047 INFO] [Grid]: Version Jetty/5.1.x
[7-29-14 21:39:42,065 INFO] [Grid]: Started HttpContext{/ selenium-server/selenium-server}
[7-29-14 21:39:42,065 INFO] [Grid]: Started org.openqa.jetty.jetty.servlet.ServletHandler@653440f3
[7-29-14 21:39:42,065 INFO] [Grid]: Started ContextHandler@1324444{/selenium-server}
[7-29-14 21:39:42,069 INFO] [Grid]: Started HttpContext{/selenium-server}
[7-29-14 21:39:42,069 INFO] [Grid]: Started HttpContext{/}
[7-29-14 21:39:42,085 INFO] [Grid]: Started SocketListener on 0.0.0.0:5555
[7-29-14 21:39:42,085 INFO] [Grid]: Started org.openqa.jetty.jetty.Server@2e568d8
[7-29-14 21:39:42,085 INFO] [Grid]: Using the json response for capabilities:{"id": "selenium@elcom", "hubDriver": "firefox", "browserName": "firefox", "platform": "VISTA"}, {"configuration": "hub": "http://10.30.217.157:4444", "port": 5555, "proxy": "org.openqa.grid.selenium.proxy.DefaultRemoteProxy", "hub": "http://10.30.217.157:4444/grid/register", "port": "5555", "browser": "firefox", "browserName": "firefox", "host": "10.30.217.157", "maxSession": "15", "regUrl": "http://10.30.217.157:4444/grid/register?&version=2", "class": "org.openqa.grid.common.RegistrationRequest"}?
[7-29-14 21:39:42,085 INFO] [Grid]: Starting auto register thread. Will try to register every 5000 ms.
[7-29-14 21:39:42,085 INFO] [Grid]: Registering the node to hub :http://10.30.217.157:4444/grid/register
```

Step 3 : After executing the command, come back to the Hub. Navigate to the URL - http://10.30.217.157:4444 and the Hub would now display the node attached to it.



Step 4 : Now let us launch the Internet Explorer Node. For launching the IE Node, we need to have the Internet Explorer driver downloaded on the node machine.

Step 5 : To download the Internet Explorer driver, navigate to <http://docs.seleniumhq.org/download/> and download the appropriate file based on the architecture of your OS. After you have downloaded, unzip the exe file and place it in a folder which has to be referred while launching IE nodes.

The screenshot shows the SeleniumHQ website's 'Downloads' section. On the left sidebar, there are links for 'Selenium Downloads', 'Latest Releases', 'Previous Releases', 'Source Code', and 'Maven Information'. Below that is a 'Donate to Selenium' section with a 'Donate' button and a link to 'through sponsorship'. Under 'Selenium Sponsors', it lists 'See who supports the Selenium project'. At the bottom left is a 'BrowserStack' logo. The main content area has sections for 'Downloads', 'Selenium IDE', and 'Selenium Server (formerly the Selenium RC Server)'. The 'The Internet Explorer Driver Server' section is highlighted with a red rounded rectangle. It contains a brief description, a note about setting \$PATH, and a download link for 'Download version 2.42.2'.

Step 6 : Launch IE using the following command.

```
C:\>java -Dwebdriver.ie.driver=D:\IEDriverServer.exe -jar
D:\JAR\selenium-server-standalone-2.42.2.jar -role webdriver -hub
http://10.30.217.157:4444/grid/register -browser
browserName=ie,platform=WINDOWS -port 5558
```

Where,

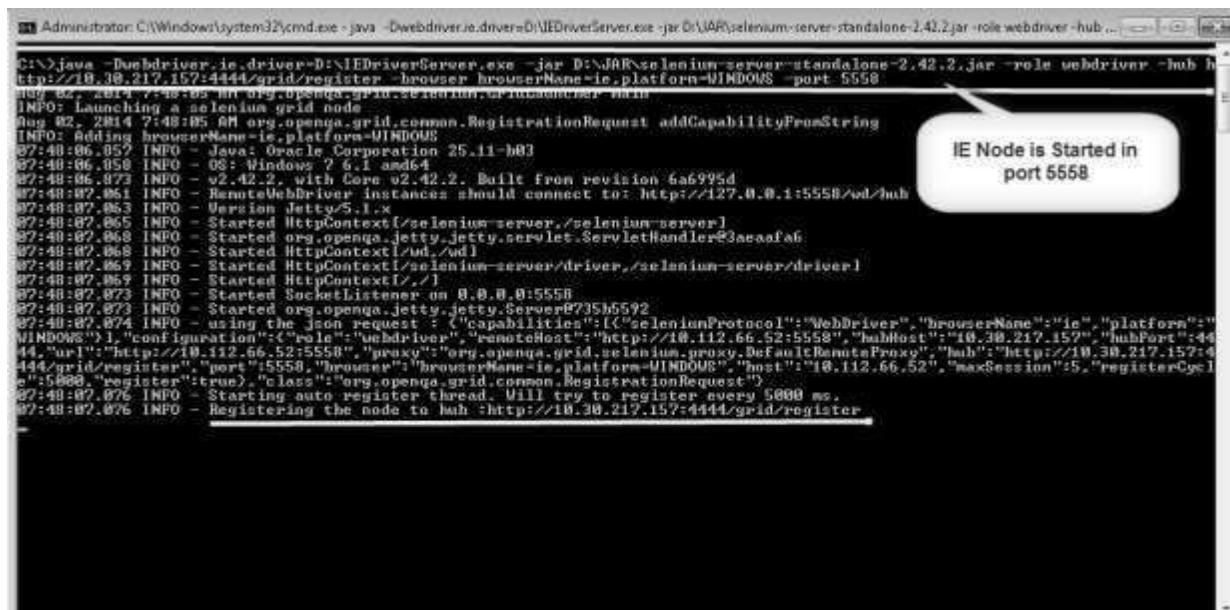
D:\IEDriverServer.exe = The **location of the downloaded the IE Driver**(on the Node Machine)

D:\JAR\selenium-server-standalone-2.42.2.jar = **Location of the Selenium Server Standalone Jar File**(on the Node Machine)

http://10.30.217.157:4444 = **IP Address of the Hub and 4444 is the port of the Hub**

browserName = **ie (Parameter to specify the Browser name on Nodes)**

5558 = **Port on which IE Node would be up and running.**



```
C:\>java -Dwebdriver.ie.driver=D:\IEDriverServer.exe -jar D:\JAR\selenium-server-standalone-2.42.2.jar -role webdriver -hub http://10.30.217.157:4444/grid/register -browser browserName=ie,platform=WINDOWS -port 5558
09:48:02, 2814 7:48:05 AM org.openqa.grid.selenium.GridLauncher main
INFO: Launching a selenium grid node
09:48:02, 2814 7:48:05 AM org.openqa.grid.common.RegistrationRequest addCapabilityFromString
INFO: Adding browserName=ie,platform=WINDOWS
09:48:06, 052 INFO - Java: Oracle Corporation 25.11-b03
09:48:06, 058 INFO - OS: Windows 7 6.1 amd64
09:48:06, 073 INFO - v2.42.2, with Core v2.42.2. Built from revision 6a6995d
09:48:07, 061 INFO - RemoteWebDriver instances should connect to: http://127.0.0.1:5558/wd/hub
09:48:07, 063 INFO - Version Jetty/5.1.x
09:48:07, 065 INFO - Started HttpContext{"/selenium-server{/selenium-server}"}
09:48:07, 068 INFO - Started org.openqa.jetty.jetty.servlet.ServletHandler@3aeaafab
09:48:07, 068 INFO - Started HttpContext{/selenium-server/WebDriver{/selenium-server/Driver}"}
09:48:07, 069 INFO - Started HttpContext{/selenium-server/Driver{/selenium-server/Driver}"}
09:48:07, 073 INFO - Started SocketListener on 0.0.0.0:5558
09:48:07, 073 INFO - Started org.openqa.jetty.jetty.Server@735b5592
09:48:07, 074 INFO - using the given request : "Capabilities": {"seleniumProtocol": "WebDriver", "browserName": "ie", "platform": "WINDOWS"}}, 1, "configuration": {"value": "remoteWebDriver", "remoteHost": "http://10.112.66.52:5558", "hubPort": "10.30.217.157:4444", "url": "http://10.112.66.52:5558", "proxy": "org.openqa.grid.selenium.proxy.DefaultRemoteProxy", "hub": "http://10.30.217.157:4444/grid/register", "port": "5558", "browser": "browserName=ie,platform=WINDOWS", "host": "10.112.66.52", "maxSession": 5, "registerCycle": 5000, "register": true}, "class": "org.openqa.grid.common.RegistrationRequest"}
09:48:07, 076 INFO - Starting auto register thread. Will try to register every 5000 ms.
09:48:07, 076 INFO - Registering the node to hub :http://10.30.217.157:4444/grid/register
```

IE Node is Started in port 5558

Step 7 : After executing the command, come back to the Hub. Navigate to the URL - <http://10.30.217.157:4444> and the Hub would now display the IE node attached to it.



Step 8 : Let us now launch the Chrome Node. For launching the Chrome Node, we need to have the Chrome driver downloaded on the node machine.

Step 9 : To download the Chrome Driver, navigate to <http://docs.seleniumhq.org/download/> and then navigate to Third Party Browser Drivers area and click on the version number '2.10' as shown below.

Third Party Drivers, Bindings, and Plugins

Selenium can be extended through the use of plugins. Here are a number of plugins created and maintained by third parties. For more information on how to create your own plugin or have it listed, consult the docs.

Please note that these plugins are not supported, maintained, hosted, or endorsed by the Selenium project. In addition, be advised that the plugins listed below are not necessarily licensed under the Apache License v.2.0. Some of the plugins are available under another free and open source software license; others are only available under a proprietary license. Any questions about plugins and their license of distribution need to be raised with their respective developer(s).

Third Party Browser Drivers NOT DEVELOPED by seleniumhq

Browser	Version	Change Log	Issue Tracker	Selenium Wiki Page	Released
Chrome	2.10	change log	issue tracker	selenium wiki page	2014-05-01
Opera	1.5	change log	issue tracker	selenium wiki page	2013-08-13
GhostDriver	(PhantomJS)		issue tracker	SeConf talk	
Windows Phone	4.14.028.10		issue tracker		2013-11-23
Selendroid - Selenium for Android			issue tracker		
ios-driver			issue tracker		
BlackBerry 10			issue tracker		2014-01-28
Appium			issue tracker		
Crosswalk			issue tracker		2014-05-05

Step 10 : Download the driver based on the type of your OS. We will execute it on Windows environment, hence we will download the Windows Chrome Driver. After you have downloaded, unzip the exe file and place it in a folder which has to be referred while launching chrome nodes.

Index of /2.10/

Name	Last modified	Size	ETag
Parent Directory		-	
chromedriver_linux32.zip	2014-05-01 20:46:22	2.33MB	4fecc99b066cb1a346035bf022607104
chromedriver_linux64.zip	2014-05-01 22:40:58	2.20MB	058cd8b7b4b9688507701b5e648fd821
chromedriver_mac32.zip	2014-05-01 22:36:30	3.93MB	f00daefc3ada3619edda2961f2beadr5c
chromedriver_win32.zip	2014-05-01 22:59:28	2.71MB	082e91e5c8994a7879710caeecd62e334
notes.txt	2014-05-01 20:46:23	0.00MB	2/c5/f1c84c22b442/f5/c74f246ce1a

Step 11 : Launch Chrome using the following command.

```
C:\>java -Dwebdriver.chrome.driver=D:\chromedriver.exe -jar
D:\JAR\selenium-server-standalone-2.42.2.jar -role webdriver -hub
http://10.30.217.157:4444/grid/register -browser
browserName=chrome,platform=WINDOWS -port 5557
```

Where,

D:\chromedriver.exe = The location of the downloaded the chrome Driver(on the Node Machine)

D:\JAR\selenium-server-standalone-2.42.2.jar = Location of the Selenium Server Standalone Jar File(on the Node Machine)

http://10.30.217.157:4444 = IP Address of the Hub and 4444 is the port of the Hub

browserName = chrome (Parameter to specify the Browser name on Nodes)

5557 = Port on which chrome Node would be up and running.

```
Administrator: C:\Windows\system32\cmd.exe - java -Dwebdriver.chrome.driver=D:\chromedriver.exe -jar D:\JAR\selenium-server-standalone-2.42.2.jar -role webdriver -hub http://10.30.217.157:4444/grid/register -browser browserName=chrome,platform=WINDOWS -port 5557
...
[INFO: Launching a selenium grid node
Aug 02, 2014 7:52:54 AM org.openqa.grid.common.RegistrationRequest addCapabilityFromString
[INFO: Adding browserName=chrome,platform=WINDOWS
...
[INFO: Java: Oracle Corporation 25.11-b63
[INFO: OS: Windows 7 6.1 and64
[INFO: v2.42.2, with Core v2.42.2. Built from revision 6a6995d
[INFO: RemoteWebDriver instances should connect to: http://127.0.0.1:5557/wd/hub
[INFO: Version Jetty/5.1.x
[INFO: Started HttpContext[/selenium-server,/selenium-server]
[INFO: Started org.openqa.jetty.servlet.ServletHandler@3aaeafab
[INFO: Started HttpContext[/wd,/wd]
[INFO: Started HttpContext[/selenium-server/driver,/selenium-server/driver]
[INFO: Started HttpContext[/selenium-server/WebDriver,/selenium-server/WebDriver]
[INFO: Started SocketListener on 0.0.0.0:5557
[INFO: Started org.openqa.jetty.Server@735b5592
[INFO: Using the json request: {"capabilities":[{"seleniumProtocol":{"Webdriver","browserName":"chrome","platform":"WINDOWS"}],"configuration":{"role":"webdriver","remoteHost":"http://10.112.66.52:5557","hubHost":"10.30.217.157","hubPort":4444,"url":"http://10.112.66.52:5557","proxy":org.openqa.grid.selenium.proxy.DefaultRemoteProxy,"hub":http://10.30.217.157:4444/grid/register,"port":5557,"browser":browserName=chrome,platform=WINDOWS,"host":10.112.66.52,"maxSession":5,"registerCycle":5000,"register":true,"class":org.openqa.grid.common.RegistrationRequest}}
[INFO: Starting auto register thread. Will try to register every 5000 ms.
[INFO: Registering the node to hub http://10.30.217.157:4444/grid/register
```

Step 12 : After executing the command, come back to the Hub. Navigate to the URL - <http://10.30.217.157:4444> and the Hub would now display the chrome node attached to it.



Develop the Script and Prepare the XML File

Step 1 : We will develop a test using TestNG. In the following example, we will launch each one of those browsers using remote WebDriver. It can pass on their capabilities to the driver so that the driver has all the information to execute on Nodes.

The Browser Parameter would be passed from the "XML" file.

```
package TestNG;

import org.openqa.selenium.remote.DesiredCapabilities;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;
import java.net.URL;
import java.net.MalformedURLException;

import org.openqa.selenium.remote.RemoteWebDriver;

public class TestNGClass
{
    public WebDriver driver;
    public String URL, Node;
    protected ThreadLocal<RemoteWebDriver> threadDriver = null;

    @Parameters("browser")
    @BeforeTest
    public void launchapp(String browser) throws MalformedURLException
    {
        String URL = "http://www.calculator.net";
        if (browser.equalsIgnoreCase("firefox"))
        {
            System.out.println(" Executing on FireFox");
            String Node = "http://10.112.66.52:5555/wd/hub";
            DesiredCapabilities cap = DesiredCapabilities.firefox();
            cap.setBrowserName("firefox");
        }
    }
}
```

```

driver = new RemoteWebDriver(new URL(Node), cap);
// Puts an Implicit wait, will wait for 10 seconds
// before throwing exception
driver.manage().timeouts()
    .implicitlyWait(10, TimeUnit.SECONDS);

// Launch website
driver.navigate().to(URL);
driver.manage().window().maximize();
}

else if (browser.equalsIgnoreCase("chrome"))
{
    System.out.println(" Executing on CHROME");
    DesiredCapabilities cap = DesiredCapabilities.chrome();
    cap.setBrowserName("chrome");
    String Node = "http://10.112.66.52:5557/wd/hub";
    driver = new RemoteWebDriver(new URL(Node), cap);
    driver.manage().timeouts()
        .implicitlyWait(10, TimeUnit.SECONDS);

    //Launch website
    driver.navigate().to(URL);
    driver.manage().window().maximize();
}

else if (browser.equalsIgnoreCase("ie"))
{
    System.out.println(" Executing on IE");
    DesiredCapabilities cap = DesiredCapabilities.chrome();
    cap.setBrowserName("ie");
    String Node = "http://10.112.66.52:5558/wd/hub";
    driver = new RemoteWebDriver(new URL(Node), cap);
    driver.manage().timeouts()
        .implicitlyWait(10, TimeUnit.SECONDS);
}

```

```

        //Launch website
        driver.navigate().to(URL);
        driver.manage().window().maximize();
    }
else
{
    throw new IllegalArgumentException
    ("The Browser Type is Undefined");
}
}

@Test
public void calculatepercent()
{
    // Click on Math Calculators
    driver.findElement(By.xpath(".//*[@id='menu']/div[3]/a")).click();

    // Click on Percent Calculators
    driver.findElement(By.xpath(".//*[@id='menu']/div[4]/div[3]/a")).click();

    // Enter value 10 in the first number of the percent Calculator
    driver.findElement(By.id("cpar1")).sendKeys("10");

    // Enter value 50 in the second number of the percent Calculator
    driver.findElement(By.id("cpar2")).sendKeys("50");

    // Click Calculate Button
    driver.findElement(By.xpath(".//*[@id='content']/table/tbody
/tr/td[2]/input")).click();

    // Get the Result Text based on its xpath
    String result =
    driver.findElement(By.xpath(".//*[@id='content']/p[2]")

```

```
/span/font/b")).getText();
// Print a Log In message to the screen
System.out.println(" The Result is " + result);

if(result.equals("5"))
{
    System.out.println(" The Result is Pass");
}
else
{
    System.out.println(" The Result is Fail");
}

@AfterTest
public void closeBrowser()
{
    driver.quit();
}
```

Step 2 : The Browser parameter will be passed using XML. Create an XML under the project folder.



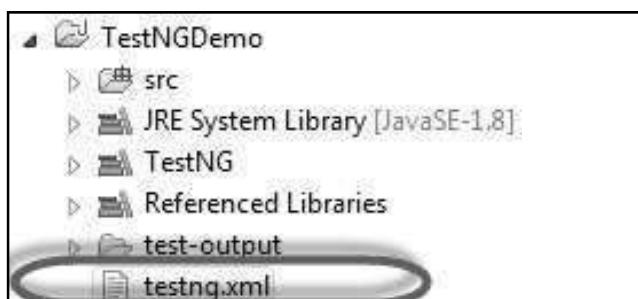
Step 3 : Select 'File' from 'General' and click 'Next'.



Step 4 : Enter the name of the file and click 'Finish'.



Step 5 : TestNg.XML is created under the project folder as shown below.



Step 6 : The contents of the XML file are shown below. We create 3 tests and put them in a suite and mention parallel="tests" so that all the tests would be executed in parallel.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite" parallel="tests">
    <test name="FirefoxTest">
        <parameter name="browser" value="firefox" />
        <classes>
            <class name="TestNG.TestNGClass" />
        </classes>
    </test>
</suite>
```

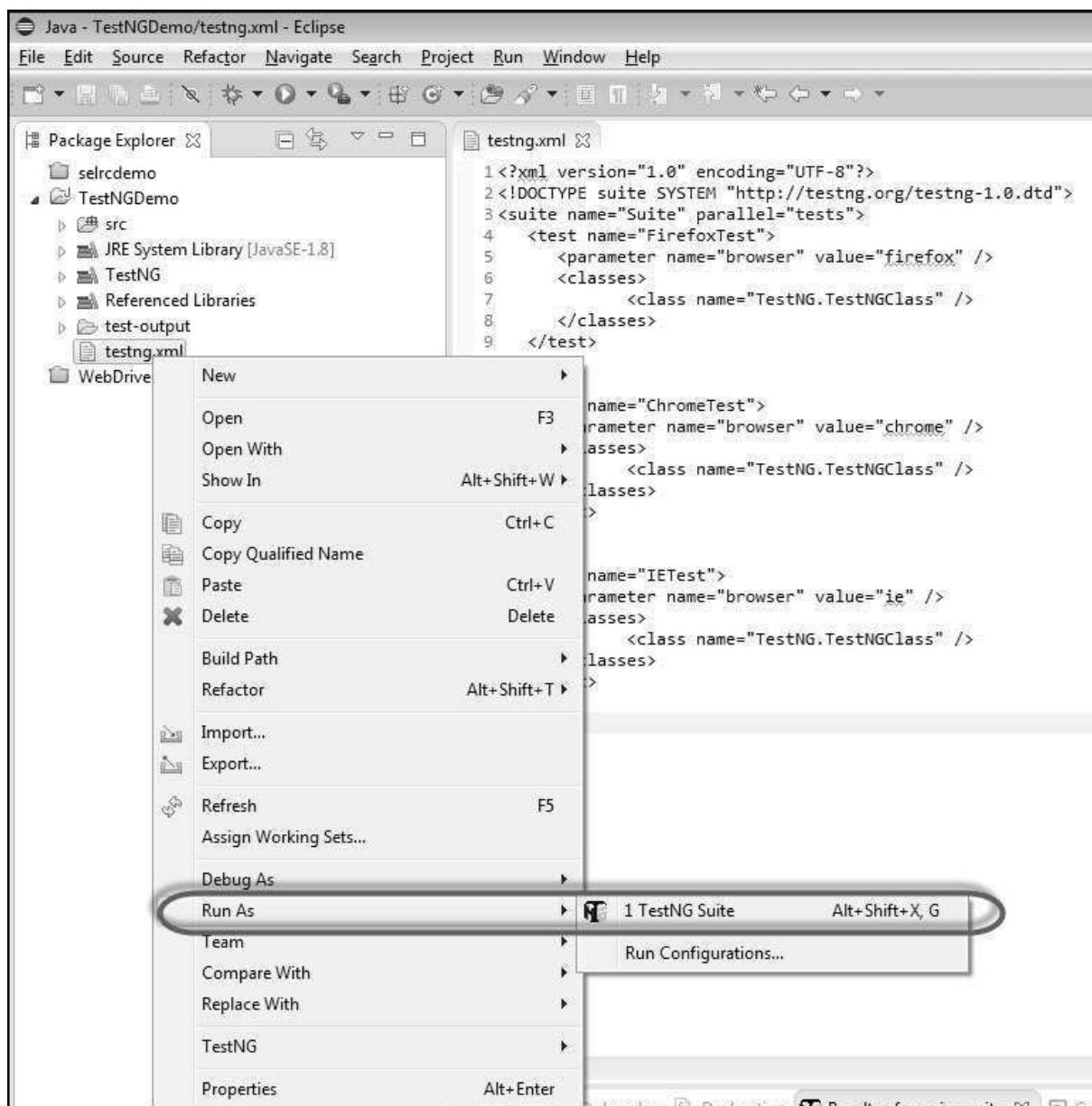
```
</classes>
</test>

<test name="ChromeTest">
    <parameter name="browser" value="chrome" />
    <classes>
        <class name="TestNG.TestNGClass" />
    </classes>
</test>

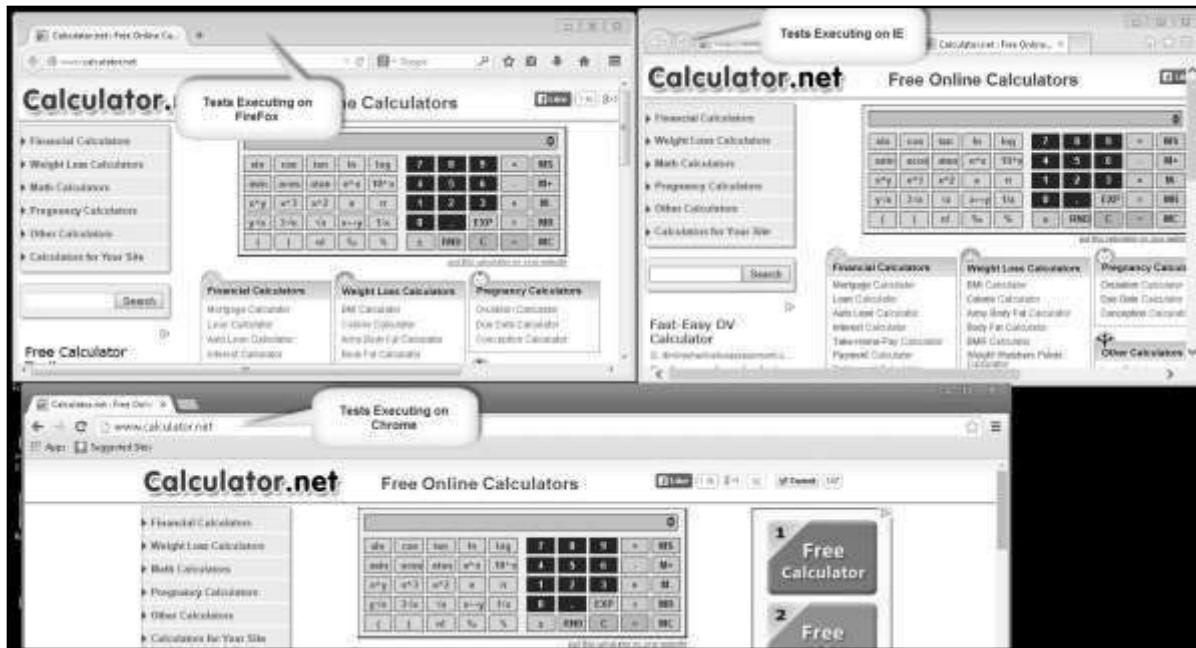
<test name="IETest">
    <parameter name="browser" value="ie" />
    <classes>
        <class name="TestNG.TestNGClass" />
    </classes>
</test>
</suite>
```

Test Execution

Step 1 : Select the created XML; right-click and choose 'Run As' >> 'TestNG Suite'.



Step 2 : Now open the Node, where we have launched all the browser nodes. You will see all the three browsers in execution simultaneously.



Result Analysis

Step 1 : Upon completing the execution, we can analyze the result like any other execution. The result summary is printed in the console as shown in the following snapshot.

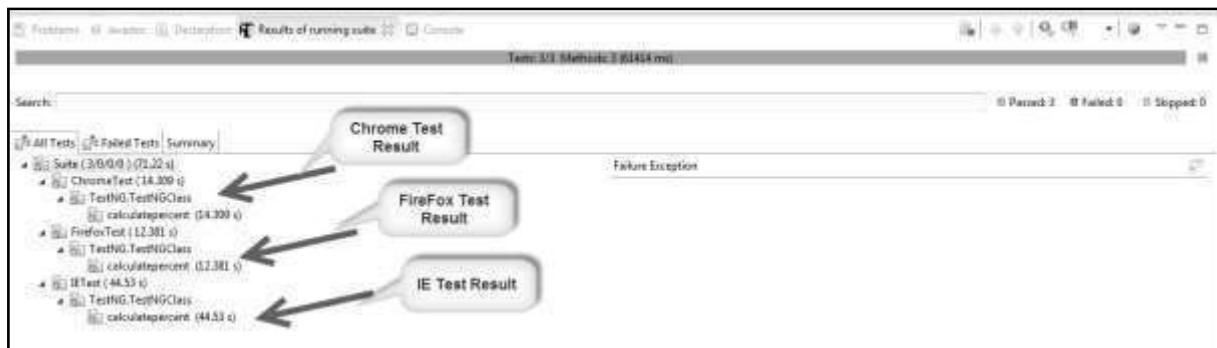
```

Problems @ Javadoc Declaration Results of running suite Console X
<terminated> testng.xml [TestNG] C:\Program Files\Java\jre8\bin\javaw.exe (2 Aug 2014 8:25:04 am)
[TestNG] Running:
D:\PERSONAL DOCS\Selenium Trials\TestNGDemo\testng.xml

Executing on FireFox
Executing on IE
Executing on CHROME
The Result is 5
The Result is Pass
The Result is 5
The Result is Pass
The Result is 5
The Result is Pass
The Result is Pass

=====
Suite
Total tests run: 3, Failures: 0, Skips: 0
=====
```

Step 2 : Navigate to the 'Results of Running Suite' Tab and TestNG would display the result summary as shown below.



Step 3 : Upon generating the HTML, we will be able to see the test results in HTML format.

