
LEARNING SELENIUM 3.0

eBook

Chapter 1 –Before You Start	6
Java Basics	6
Java Virtual Machine	6
Java Development Kit	8
Class Concepts	10
Object Oriented Programming System Concepts	15
String Class	24
Summary	25
Chapter 2 – Introduction to Selenium	26
What is Automation Testing?	26
Use of Automation Testing	26
Tools for Automation Testing	27
Why Automation is Important for Your Career?	28
What is Selenium?	29
History of Selenium	30
Component of Selenium	31
Difference between API's and GUI's	33
Summary	35
Chapter 3 – Setup Eclipse	36
Download and Install Java	36
Download and Start Eclipse	40
Download and Configure WebDriver Java Client	44
Setup a Project	46
Create Packages	47
Create a First Java Test Case	51
Import WebDriver Source File	55
Summary	57
Chapter 4– WebDriver	58
WebDriver Interface	58
WebElement Interface	64
Launching Firefox Browser	66
Summary	73
Chapter 5 – Browser and Navigation Commands	74
How to Open a URL	74
Verify Page Title	75
Strategy to Get the Page Source	78
Difference between Close and Quit	80

Ways to Navigate Back and Forward	81
How to Refresh a Page	83
Another Way of Navigating to a Specific Page	84
Summary	86
Chapter 6 – Locators	87
What are Locators?	87
Difference Between Absolute and Complete Xpath	90
Finding Your First Element	91
WebElement Commands	93
Summary	97
Chapter 7 – Element Identification	98
Element Inspector in Mozilla, Chrome and IE	98
Element Locator Tool for Firefox Locator	101
Firebug and Firepath Add-ons in Mozilla	102
Various HTML Locator Strategies	109
Xpath Helper Plug-in for Chrome	112
Selection of Effective Xpath	113
Handling Dynamic Objects/Ids on the Page	115
Summary	117
Chapter 8 –Tables, Checkboxes and Radio Buttons	118
Identify Table Rows and Columns	118
Extracting Values from a Cell	120
Dynamically Identify Tables Data	121
Select Class in Selenium	128
Drop Down Handle	130
Select Multiple Values from the List	131
Select and Deselect Operations By Index, Value and Visible Text	133
Summary	139
Chapter 9 –Selenium Waits, Alerts and Switch Windows	140
Wait	140
Implicit Wait	140
Explicit Wait	140
Fluent Wait	142
How to Use Expected Conditions with Waits	142
WebDriverWait and its Uses	143
Different Wait Until Conditions	144
Alerts	144

Switch Windows	147
Different Between Window Handle and Handles	147
Switching and Closing Windows, Tabs and Popups	148
Concepts of Set Interface in Java	151
Concept of Window ID	152
Javascript and Selenium	152
Summary	154
Chapter 10 –Action Class	155
What is Action Class	155
What Can We Do With Action Class	156
Mouse Hover and Mouse Movement with Action	163
Finding Coordinates of a Web Object	166
Drag and Drop Action	167
Robot Class	170
Summary	173
Chapter 11–TestNG Framework	174
What is TestNG?	174
Annotations in TestNG	187
How to Run Test Suite in TestNg	189
Groups in TestNG	193
Depend-on in TestNG	197
Test Case Sequencing in TestNG	198
TestNG Asserts	200
TestNG Parameters	201
Skipping Test Cases	203
Multi Browser Testing in TestNG	205
Parallel Testing in TestNG	208
TestNG Listeners	209
Summary	219
Chapter 12 – Log4j Logging	220
Introduction	220
Creating Logger	220
Download Log4j	222
Add Log4j Jar	226
Test Case with Log4j	228
Log4j Log Manager	231
Log4j Appenders	234

Log4j Layouts	238
Summary	241
Chapter 13 – Database Connections	242
Database Connection	242
Database Testing in Selenium Using MySql Server	244
Configuring SQL	248
JDBC Driver	249
Understanding Connection Interface	253
Statement Interface	254
Prepared Statement Interface.	255
ResultSet Interface	256
Firing Queries Using Java on Eclipse	256
Summary	258
Chapter 14 – Data Driven Automation Framework	259
What is Automation Framework?	259
Features of Automation Framework	259
Benefits of Using Automation Framework	260
Different Types of Automation Framework	261
What is Data Driven Framework?	266
What is Modular Driven Framework?	269
What is Keyword Driven Framework?	273
What is Hybrid Driven Framework?	274
Test Data Files	275
TestCore Class	278
Object.properties File to Store Xpaths	281
Configuration Files	284
Emailing Test Results	286
Generating Reports	289
Generating Application and Selenium Logs	291
Summary	294
Chapter 15 – JUnit Java Framework	295
About JUnit and TestNG	295
What is JAVA Framework	313
Test Annotations	313
Executing Tests in Sequence	317
Assertions	320
Error Collectors	320

How to Parameterize Test Cases	322
Summary	325
Chapter 16 – Maven	326
Introduction to Maven	326
Install Maven on Eclipse IDE	327
Install Maven on Mac	337
How to Create a New Maven Project	339
How to Create a New Maven Project in Eclipse	347
Configure Selenium Continuous Integration with Maven	353
Summary	369
Chapter 17 – Jenkins	370
Introduction of Jenkins	370
Jenkins Setup	375
Selenium Integration with Jenkins	383
Summary	387
Chapter 18 – Hybrid Framework	388
Hybrid Framework	388
Combining Multiple Frameworks for Test Execution	392
Summary	395
Chapter 19 –Page Object Model (POM)	396
Page Object Model	396
Page Factory	396
Set up Page Object Model (POM) in Selenium	397
Summary	402
Chapter 20 –Miscellaneous	403
Handling Cookies	403
Hash Testing Using Selenium	408
Screenshot Capturing	410
Screen Reading Through Selenium	412
Highlighting Elements through JavaScript in Selenium	413
Katalon Studio	415
Summary	416

Chapter 1 –Before You Start

Java Basics

Java is a high level, robust, secured and object-oriented programming (OOP) language. It was developed by James Gosling, Mike Sheridan, Patrick Naughton in 1995 for Sun Microsystems, later acquired by Oracle Corporation. Java technology is used to develop applications such as Standalone Application, Web Application, Enterprise Application and Mobile Application. Games, Smart Card, Embedded System, Robotics for a wide range of environments, from consumer devices to heterogeneous enterprise systems. Java language, a C-language derivative has its own structure, syntax rules, and programming paradigm. The Java language helps to create modular programs and reusable code which can run on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

With the advancement of Java multiple configurations were built to suit various types of platforms such as J2EE for Enterprise Applications, J2ME for Mobile Applications, Java SE, Java EE, and Java ME.

JAVA Components

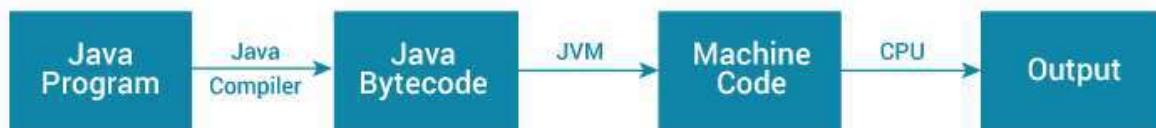
The Java language's programming paradigm is based on the concept of OOP, which the language's features support. Structurally, the Java language starts with packages. A package is the Java language's namespace mechanism. Within packages are classes, and within classes are methods, variables, constants. The Java Runtime Environment (JRE) includes the Java Virtual Machine (JVM), code libraries, and components that are necessary for running programs that are written in the Java language. The JRE is available for multiple platforms. The JRE can be freely redistributed with applications, according to the terms of the JRE license, to give the application's users a platform on which to run your software. The JRE is included in the Java development Kit (JDK).

Java Virtual Machine

A JVM is an abstract computing machine, or virtual machine because it doesn't physically exist. It is a platform-independent execution environment that converts Java bytecode into machine language and executes it.

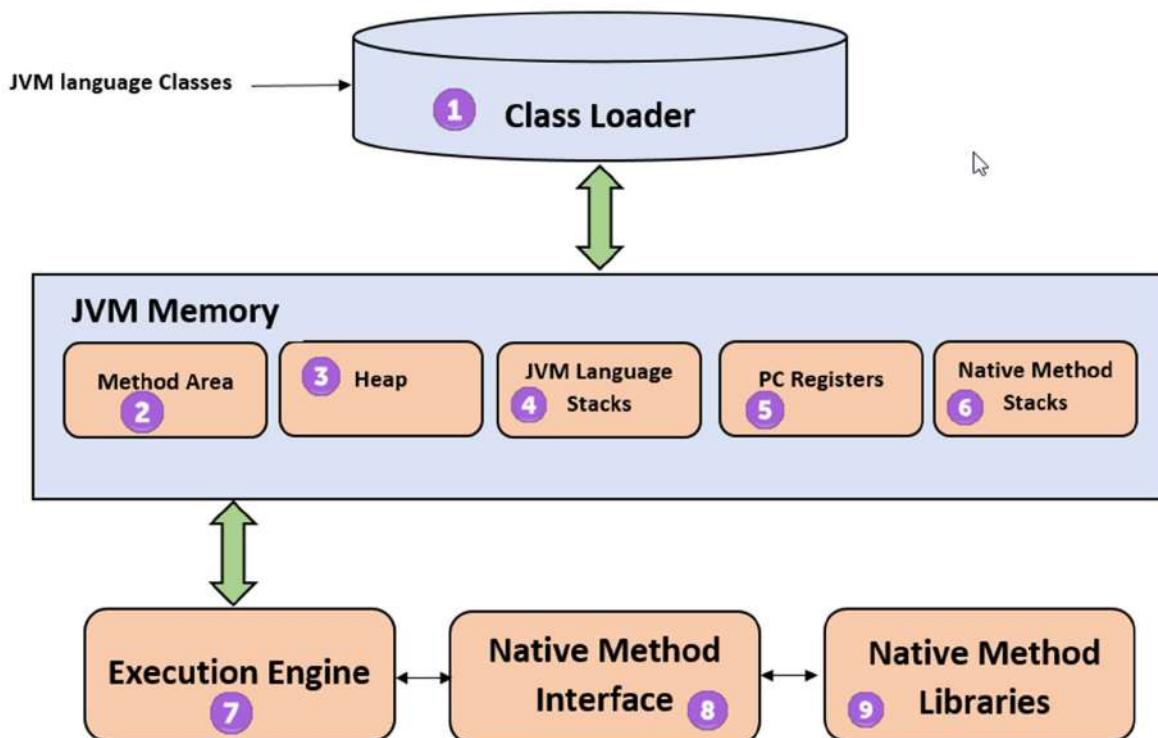
How JVM works?

A Java program is written and saved as .java extension. The compiler checks the code against the language's syntax rules and then writes out bytecode in .class files. Bytecode is a set of instructions targeted to run on a JVM. The byte code can run on any platform such as Windows, Linux, Mac OS. Each operating system has different JVM, however the output they produce after execution of bytecode is same across all operating systems. The pictorial representation of compilation and execution of a Java program is:



The Compiler (javac) converts source code (.java file) to the byte code (.class file). The JVM executes the bytecode produced by compiler. At runtime, the JVM reads and interprets .class files and executes the program's instructions on the native hardware platform for which the JVM was written. The JVM interprets the bytecode just as a CPU would interpret assembly-language instructions.

JVM Architecture



The components of JVM are explained below:

- **ClassLoader**: The class loader is a subsystem used for loading class files. It performs three major functions as Loading, Linking, and Initialization.
- **Method Area**: JVM Method Area stores class structures like metadata, the constant runtime pool, and the code for methods.
- **Heap**: All the Objects, their related instance variables, and arrays are stored in the heap. This memory is common and shared across multiple threads.
- **JVM language Stacks**: Java language Stacks store local variables, and it's partial results. Each thread has its own JVM stack, created simultaneously as the thread is

created. A new frame is created whenever a method is invoked, and it is deleted when method invocation process is complete.

- **PC Registers:** PC register store the address of the Java virtual machine instruction which is currently executing. In Java, each thread has its separate PC register.
- **Native Method Stacks:** Native method stacks hold the instruction of native code depends on the native library. It is written in another language instead of Java.
- **Execution Engine:** It is a type of software used to test hardware, software, or complete systems. The test execution engine never carries any information about the tested product.
- **Native Method interface:** The Native Method Interface is a programming framework. It allows Java code which is running in a JVM to call by libraries and native applications.
- **Native Method Libraries:** Native Libraries is a collection of the Native Libraries(C, C++) which are needed by the Execution Engine.

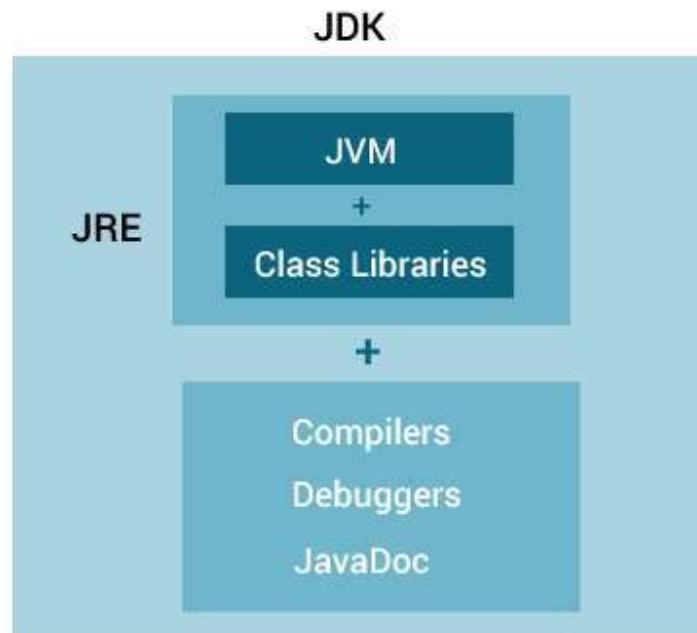
JVM Operations

The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

Java Development Kit

The Java Development Kit (JDK) is a software development environment which is used to develop Java applications, Java applets. The JDK contains JVM, interpreter/loader, compiler, an archiver, a documentation generator required to complete the development of a Java Application. The pictorial representation of a JDK is:



JRE Components

JRE consists of the following components:

- Deployment technologies, including deployment, Java Web Start and Java Plug-in.
- User interface toolkits, including Abstract Window Toolkit (AWT), Swing, Java 2D, Accessibility, Image I/O, Print Service, Sound, drag and drop (DnD) and input methods.
- Integration libraries, including Interface Definition Language (IDL), Java Database Connectivity (JDBC), Java Naming and Directory Interface (JNDI), Remote Method Invocation (RMI), Remote Method Invocation Over Internet Inter-Orb Protocol (RMI-IIOP) and scripting.
- Other base libraries, including international support, input/output (I/O), extension mechanism, Beans, Java Management Extensions (JMX), Java Native Interface (JNI), Math, Networking, Override Mechanism, Security, Serialization and Java for XML Processing (XML JAXP).
- Lang and util base libraries, including lang and util, management, versioning, zip, instrument, reflection, Collections, Concurrency Utilities, Java Archive (JAR), Logging, Preferences API, Ref Objects and Regular Expressions.
- Java Virtual Machine (JVM), including Java HotSpot Client and Server Virtual Machines.

Eclipse is a popular open source Integrated Development Environment (IDE) for Java development. Eclipse handles basic tasks, such as code compilation and debugging apart from providing an interface for writing and testing code. In addition, Eclipse can be used to organize source code files into projects, compile and test those projects, and store project files in any number of source repositories.

Difference between JVM, JRE and JDK

JRE: JRE is the environment within which the java virtual machine runs. JRE contains Java virtual Machine(JVM), class libraries, and other files excluding development tools such as compiler and

debugger. That means you can run the code in JRE but you can't develop and compile the code in JRE.

JVM: JVM runs the program by using class, libraries and files provided by JRE.

JDK: JDK is a superset of JRE, it contains everything that JRE has along with development tools such as compiler, debugger.

Class Concepts

A class is a user defined entity from which objects are created. An object can be defined as an instance of a class. An object contains an address and takes up some space in memory whereas class doesn't store any space. In other words, class is a blueprint or a set of instruction to build a group of object which has common properties.

For Example, we can think of class as a sketch of a building. It contains all the details about the floors, doors and windows. Based on these descriptions we build the building. Building is the object. Since, many buildings can be made from the same description, we can create many objects from a class.

Object

It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of:

State: It is represented by attributes of an object. It also reflects the properties of an object.

Behavior: It is represented by methods of an object. It also reflects the response of an object with other objects.

Identity: It gives a unique name to an object and enables one object to interact with other objects.

If we consider the real-world, we can find many objects around us, cars, dogs, humans. All these objects have a state and a behavior. If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging the tail, running. If you compare the software object with a real-world object, they have very similar characteristics. Software objects also have a state and a behavior. A software object's state is stored in fields and behavior is shown via methods. So in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods

Class Declaration

A Java class declarations can include these components, in order:

- **Modifiers:** A class can be public or has default access.
- **Class name:** The name should begin with a initial letter (capitalized by convention).
- **Superclass:** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
- **Interfaces:** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
- **Body:** The class body surrounded by braces, { }.

Syntax of a class is:

```
class <class_name>{
    field;
    method;
}
```

Constructors

Constructors are used for initializing new objects. Fields are variables that provides the state of the class and its objects, and methods are used to implement the behavior of the class and its objects. There are various types of classes that are used in real time applications such as nested classes, anonymous classes, lambda expressions. Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Example

```
public class Puppy {
    public Puppy() {
    }
    public Puppy(String name) {
        // This constructor has one parameter, name.
    }
}
```

Class Variables

A class can contain any of the following variable types.

- **Local variables:** Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables:** Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables:** Class variables are variables declared within a class, outside any method, with the static keyword.

Creating an Object

An object is created from a class. In Java, the new keyword is used to create new objects. There are three steps when creating an object from a class.

- Declaration: A variable declaration with a variable name with an object type
- Instantiation: The 'new' keyword is used to create the object
- Initialization: The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Following is an example of creating an object

```

public class Puppy {
    public Puppy(String name) {
        // This constructor has one parameter, name.
        System.out.println("Passed Name is :" + name );
    }

    public static void main(String []args) {
        // Following statement would create an object myPuppy
        Puppy myPuppy = new Puppy( "tommy" );
    }
}

```

Output

Passed Name is :tommy

Initializing an object

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor.Example:

```

// Class Declaration

public class Dog
{
    // Instance Variables
    String name;
    String breed;
    int age;
    String color;

    // Constructor Declaration of Class
    public Dog(String name, String breed,int age, String color)
    {
        this.name = name;
        this.breed = breed;
        this.age = age;
        this.color = color;
    }

    // method 1
    public String getName()
    {

```

```
        return name;  
    }  
  
    // method 2  
    public String getBreed()  
    {  
        return breed;  
    }  
  
    // method 3  
    public int getAge()  
    {  
        return age;  
    }  
  
    // method 4  
    public String getColor()  
    {  
        return color;  
    }  
  
    @Override  
    public String toString()  
    {  
        return("Hi my name is "+ this.getName() +  
              ".\nMy breed,age and color are " +  
              this.getBreed()," + this.getAge() +  
              ","+ this.getColor());  
    }  
  
    public static void main(String[] args)  
    {  
        Dog tuffy = new Dog("tuffy","papillon", 5, "white");  
        System.out.println(tuffy.toString());  
    }  
}
```

Output:

```
Hi my name is tuffy.  
My breed,age and color are papillon,5,white
```

Code Explanation:

This class contains a single constructor. We can recognize a constructor because its declaration uses the same name as the class and it has no return type. The Java compiler differentiates the constructors based on the number and the type of the arguments. The constructor in the Dog class takes four arguments. The following statement provides "tuffy","papillon",5,"white" as values for those arguments:

```
Dog tuffy = new Dog("tuffy","papillon",5, "white");
```

Note: All classes have at least one constructor. If a class does not explicitly declare any, the Java compiler automatically provides a no-argument constructor, also called the default constructor. This default constructor calls the class parent's no-argument constructor (as it contain only one statement i.e super());, or the Object class constructor if the class has no other parent.

Ways to create object of a class

There are four ways to create objects in java.

Using new keyword: It is the most common and general way to create object in java.

Example:

```
// creating object of class Test  
Test t = new Test();
```

Using Class.forName(String className) method : There is a pre-defined class in java.lang package with name Class. The forName(String className) method returns the Class object associated with the class with the given string name. We have to give the fully qualified name for a class. On calling new Instance() method on this Class object returns new instance of the class with the given string name.

```
// creating object of public class Test  
// consider class Test present in com.p1 package  
Test obj = (Test)Class.forName("com.p1.Test").newInstance();
```

Using clone() method: clone() method is present in Object class. It creates and returns a copy of the object.

```
// creating object of class Test  
Test t1 = new Test();
```

```
// creating clone of above object
Test t2 = (Test)t1.clone();
```

- Deserialization : De-serialization is technique of reading an object from the saved state in a file. Refer Serialization/De-Serialization in java

```
FileInputStream file = new FileInputStream(filename);
ObjectInputStream in = new ObjectInputStream(file);
Object obj = in.readObject();
```

Object Oriented Programming System Concepts

Object Oriented Programming is a methodology or paradigm to design a program using classes and objects. It is a programming style which is associated with the concepts like class, object, Inheritance, Encapsulation, Abstraction, Polymorphism. Object-oriented languages follow a different programming pattern from structured programming languages like C and COBOL. The Object-oriented languages combine data and program instructions into objects.

Object

An object is a self-contained entity that contains attributes and behavior. Instead of having a data structure with fields (attributes) and passing that structure around to all of the program logic that acts on it (behavior), in an object-oriented language, data and program logic are combined. This combination can occur at vastly different levels of granularity, from fine-grained objects such as a Number, to coarse-grained objects, such as a FundsTransfer service in a large banking application. An object-based application in Java is based on declaring classes, creating objects from them and interacting between these objects.

Principles of OOPs

The object-oriented paradigm supports four major principles

Inheritance

In OOP, computer programs are designed in such a way where everything is an object that interacts with one another. Inheritance is one such concept where the properties of one class can be inherited by the other. It helps to reuse the code and establish a relationship between different classes. In Java, there are two classes:

- Parent class (Super or Base class)
- Child class (Subclass or Derived class)

A class which inherits the properties is known as Child Class whereas a class whose properties are inherited is known as Parent class. The biggest advantage of Inheritance is that the code in

base class need not be rewritten in the child class. That is the variables and methods of the base class can be used in the child class as well.

Syntax:

```
class A extends B
{
}
```

Here class A is child class and class B is parent class.

Example:

```
class Faculty {
    String designation = "Faculty";
    String college = "BookWorld";
    void does() {
        System.out.println("Teaching");
    }
}
public class ScienceFaculty extends Faculty
{
    String mainSubject = "Science";
    public static void main(String args[]){
        ScienceFaculty obj = new ScienceFaculty();
        System.out.println(obj.college);
        System.out.println(obj.designation);
        System.out.println(obj.mainSubject);
        obj.does();
    }
}
```

Output:

```
BookWorld
Faculty
Science
Teaching
```

Code Explanation:

- There is a parent class Faculty and a child class ScienceFaculty.
- In the ScienceFaculty class there is no need to write the same code which is already present in the present class.
- The college name, designation and does() method is common for all the faculties, thus ScienceFaculty class does not need to write this code
- The common data members and methods can be inherited from the Faculty class.

Polymorphism

Polymorphism in java is a concept by which we can perform a single action by different ways. In other words, Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object. Polymorphism in java can be performed by method overloading and method overriding.

There are two types of polymorphism in java

- compile time polymorphism is achieved by method overloading
- runtime polymorphism is achieved by method overriding

For example, let's say we have a class Animal that has a method `sound()`. Since this is a generic class so we can't give it a implementation like: Roar, Meow, Oink.

```
public class Animal{
    ...
    public void sound(){
        System.out.println("Animal is making a sound");
    }
}
```

Now lets say we two subclasses of Animal class: Horse and Cat that extends Animal class. We can provide the implementation to the same method

```
public class Horse extends Animal{
    ...
    @Override
    public void sound(){
        System.out.println("Neigh");
    }
}
```

And

```
public class Cat extends Animal{
    ...
    @Override
    public void sound(){
        System.out.println("Meow");
    }
}
```

As you can see that although we had the common action for all subclasses `sound()` but there were different ways to do the same action. It would not make any sense to just call the generic `sound()` method as each Animal has a different sound. Thus we can say that the action this method performs is based on the type of object.

Example 1:Runtime Polymorphism example:

Code of Animal.java

```
public class Animal{
    public void sound(){
        System.out.println("Animal is making a sound");
    }
}
```

Code of Horse.java

```
class Horse extends Animal{
    @Override
    public void sound(){
        System.out.println("Neigh");
    }
    public static void main(String args[]){
        Animal obj = new Horse();
        obj.sound();
    }
}
```

Output:

Neigh
Cat.java

```
public class Cat extends Animal{
    @Override
    public void sound(){
        System.out.println("Meow");
    }
    public static void main(String args[]){
        Animal obj = new Cat();
        obj.sound();
    }
}
```

Output:

Meow

Example 2: Compile time Polymorphism

```
class Overload
```

```

{
    void demo (int a)
    {
        System.out.println ("a: " + a);
    }
    void demo (int a, int b)
    {
        System.out.println ("a and b: " + a + "," + b);
    }
    double demo(double a) {
        System.out.println("double a: " + a);
        return a*a;
    }
}
class MethodOverloading
{
    public static void main (String args [])
    {
        Overload Obj = new Overload();
        double result;
        Obj .demo(10);
        Obj .demo(10, 20);
        result = Obj .demo(5.5);
        System.out.println("O/P : " + result);
    }
}

```

Output:

```

a: 10
a and b: 10,20
double a: 5.5
O/P : 30.25

```

Code Explanation:

Here the method *demo()* is overloaded 3 times: first method has 1 int parameter, second method has 2 int parameters and third one is having double parameter. Which method is to be called is

determined by the arguments we pass while calling methods. This happens at runtime so this type of polymorphism is known as compile time polymorphism.

Abstraction

Abstraction is a process where only relevant data is shown to the user and unnecessary details of an object are hidden. For example, when the user login to his bank account online, he enter his user_id and password and press login, The after process of to and fro information transfer and verification is all abstracted away from the user.

The abstraction in object oriented programming can be achieved by various ways such as encapsulation and inheritance.

A Java program is also a great example of abstraction. Here java takes care of converting simple statements to machine language and hides the inner implementation details from outer world.

Abstract Keyword (Abstract Classes and Methods)

An abstract class is never instantiated. When a class contains an abstract method, then it is declared as abstract class. It is used to provide abstraction. Note that an abstract class does not provide 100% abstraction because it may contain a concrete method as well

Syntax:

```
abstract class class_name { }
```

Example of Abstract class

```
abstract class A
{
    abstract void callme();
}

class B extends A
{
    void callme()
    {
        System.out.println("this is callme.");
    }

    public static void main(String[] args)
    {
        B b = new B();
        b.callme();
    }
}
```

Output:

this is callme.

Note the key points about abstract classes:

- Abstract classes are not Interfaces. They are different, we will study this when we will study Interfaces.
- An abstract class may or may not have an abstract method. But, if any class has even a single abstract method, then it must be declared abstract.
- Abstract classes can have constructors, member variables and normal methods.
- Abstract classes are never instantiated.
- When you extend an abstract class with abstract method, you must define the abstract method in the child class or make the child class abstract.

Abstract class cannot be instantiated. Below is an example to demonstrate same.

```
abstract class Student
{
    public void name()                      // concrete (non-abstract) method
    {
        System.out.println("Name is Adam");
    }
    public void marks()                     // concrete (non-abstract) method
    {
        System.out.println("Marks scored are 80");
    }
    public static void main(String args[])
    {
        Student s1 = new Student();      // Error raised, see the error in
        screenshot
    }
}
```

Output:

```
Student.java:13: error: Student is abstract; cannot be instantiated
    Student s1 = new Student();      // Error raised, see the error in
    screenshot
                                ^
1 error
```

Abstract Method

The methods that are declared without any body within an abstract class are called abstract methods. The method's body, in this case, is defined by its subclass. An abstract method can never be final and static. Any class that extends an abstract class must implement all the abstract methods declared by the super class.

Syntax:

```
abstract return_type function_name (); //No definition
```

Abstract method in an abstract class

```
//abstract class
abstract class Sum{
    /* These two are abstract methods, the child class
     * must implement these methods
     */
    public abstract int sumOfTwo(int n1, int n2);
    public abstract int sumOfThree(int n1, int n2, int n3);

    //Regular method
    public void disp(){
        System.out.println("Method of class Sum");
    }
}

//Regular class extends abstract class
class Demo extends Sum{

    /* If I don't provide the implementation of these two methods,
    the
     * program will throw compilation error.
     */
    public int sumOfTwo(int num1, int num2){
        return num1+num2;
    }

    public int sumOfThree(int num1, int num2, int num3){
        return num1+num2+num3;
    }

    public static void main(String args[]){

```

```

        Sum obj = new Demo();
        System.out.println(obj.sumOfTwo(3, 7));
        System.out.println(obj.sumOfThree(4, 3, 19));
        obj.disp();
    }
}

```

Output:

```

10
26
Method of class Sum

```

Note the key points about abstract method:

- Abstract methods don't have body, they just have method signature.
- If a class has an abstract method it should be declared abstract, the vice versa is not true, which means an abstract class doesn't need to have an abstract method compulsory.
- If a regular class extends an abstract class, then the class must have to implement all the abstract methods of abstract parent class or it has to be declared abstract as well.

Encapsulation

Encapsulation means putting together all the variables and the methods into a single unit called Class. The idea behind is to hide how things work and just exposing the requests a user can do. Encapsulation provides the security that keeps data and methods safe from inadvertent changes. Encapsulation can be achieved in Java by:

- Declaring the variables of a class as private.
- Providing public setter and getter methods to modify and view the variables values.

Example:

```

public class Student {
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public static void main(String[] args) {
    }
}

```

Code Explanation:

- There is a class Student which has a private variable name and a getter and setter methods through which the name of a student can get and set.
- Through these methods, any class which wishes to access the name variable has to do it using these getter and setter methods.

String Class

String is a sequence of characters, for e.g. “World” is a string of 5 characters. In Java, string is a constant and cannot be changed once it has been created. The class String includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase. Case mapping is based on the Unicode Standard version specified by the Character class.

The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings. String concatenation is implemented through the StringBuilder(or StringBuffer) class and its append method. String conversions are implemented through the method `toString`, defined by Object and inherited by all classes in Java.

There are two ways to create a String object:

- By string literal: Java String literal is created by using double quotes.
For Example: `String str="Hello";`
- By new keyword: Java String is created by using a keyword “new”.
For Example: `String str=new String("Hello");`

Java String pool refers to collection of Strings which are stored in heap memory. In this, whenever a new object is created, String pool first checks whether the object is already present in the pool or not. If it is present, then same reference is returned to the variable else new object will be created in the String pool and the respective reference will be returned.

Example:

```
public class StringDemo {
    public static void main(String args[]) {
        char[] helloArray = { 'w', 'o', 'r', 'l', 'd', '.' };
        String helloString = new String(helloArray);
        System.out.println( helloString );
    }
}
```

Output:

World.

Summary

- Java is a high level, robust, secured and object-oriented programming (OOP) language
- The Java Runtime Environment (JRE) includes the Java Virtual Machine (JVM), code libraries, and components that are necessary for running programs that are written in the Java language
- The Java Development Kit (JDK) is a software development environment which is used to develop Java applications, Java applets
- Abstraction is a process where only relevant data is shown to the user and unnecessary details of an object are hidden

DUCAT®
www.ducatindia.com

Chapter 2 – Introduction to Selenium

What is Automation Testing?

Automation Testing uses an automation software to execute test case suite. The automation software enters test data into the system under test, compare expected and actual results and generate detailed test reports. Successive development cycles of the system will require execution of same test suite repeatedly. Using a test automation software, it's possible to record this test suite and re-play it as required. Once the test suite is automated, no human intervention is required. Automated software testing is important due to following reasons:

- Manual Testing of all workflows, all fields, all negative scenarios is time and money consuming
- It is difficult to test for multilingual sites manually
- Automation does not require Human intervention
- Automation increases the speed of test execution
- Automation helps increase Test Coverage

There are several situations, which cannot be tested manually and require automation testing such as,

- Comparing two images pixel by pixel.
- Comparing two spreadsheets containing thousands of rows and columns.
- Testing an application under the load of 100,000 users.
- Performance Benchmarks.
- Testing the application on different browsers and on different operating systems in parallel.

Use of Automation Testing

The goal of Automation is to reduce the number of test cases to be run manually and not to eliminate Manual testing altogether. The benefits of automated testing are:

- 70% faster than the manual testing
- Increase in test coverage of application features
- Reliable in results
- Ensure Consistency
- Saves Time and Money
- Improves test accuracy
- Human Intervention is not required while execution
- Increases Efficiency
- Complex testing made easy
- Better speed in executing tests
- Re-usable test scripts
- Test Frequently and thoroughly
- More cycle of execution can be achieved through automation

Tools for Automation Testing

Automated software testing is becoming more and more important for many software projects in order to automatically verify key functionality, test for regressions and help teams run a large number of tests in a short period of time. There are various tools that help software teams build and execute automated tests. Many teams are actively using unit tests as part of their development efforts to verify critical parts of their projects such as libraries, models and methods.

With the growing number of web-based applications this is changing, however, as verifying and testing web-based interfaces is easier and there are various tools that help with this, including open source tools. Below are few of the widely used automation testing tools:

Selenium

Selenium is a software testing tool used for Regression Testing. It is an open source testing tool that provides playback and recording facility for Regression Testing. The various features of selenium are:

- It provides the provision to export recorded script in other languages like Java, Ruby, RSpec, Python, C#, etc
- It can be used with frameworks like JUnit and TestNG
- It can execute multiple tests at a time
- Autocomplete for Selenium commands that are common
- Store tests as Ruby Script, HTML, and any other format
- It allows to insert comments in the middle of the script for better understanding and debugging

QuickTest Professional

QuickTest Professional (QTP) is widely used for functional and regression testing as it addresses every major software application and environment. To simplify test creation and maintenance, it uses the concept of keyword driven testing. It allows the tester to build test cases directly from the application. The various features of QTP are:

- It fix defects faster by thoroughly documenting and replicating defects for developer
- Collapse test creation and test documentation at a single site
- Parameterization is easy than WinRunner
- QTP supports .NET development environment
- It has better object identification mechanism
- It can enhance existing QTP scripts without "Application Under Test" being available, by using the ActiveScreen

Rational Functional Tester

Rational functional tester is an Object-Oriented automated Functional Testing tool that is capable of performing automated functional, regression, data-driven testing and GUI testing. The various features of Rational Functional Tester are:

- It supports a wide range of protocols and applications like Java, HTML, NET, Windows, SAP, Visual Basic, etc.
- It can record and replay the actions on demand
- It integrates well with source control management tools such as Rational Clear Case and Rational Team Concert integration
- It allows developers to create keyword associated script so that it can be re-used
- Eclipse Java Developer Toolkit editor facilitates the team to code test scripts in Java with Eclipse
- It supports custom controls through proxy SDK (Java/.Net)
- It supports version control to enable parallel development of test scripts and concurrent usage by geographically distributed team

WATIR

Watir is an open source testing software for regression testing. It enables to write tests that are easy to read and maintain. Watir supports only internet explorer on windows while Watir webdriver supports Chrome, Firefox, IE, Opera. The various features of Watir are:

- It supports multiple browsers on different platforms
- It uses a fully-featured modern scripting language Ruby
- It supports web application regardless of its development platform.

SilkTest

Silk Test is designed for doing functional and regression testing. For e-business application, silk test is the leading functional testing product. It is a product of Segue Software takeover by Borland in 2006. It is an object-oriented language just like C++. It uses the concept of an object, classes, and inheritance. The various features of Skilltest are:

- It consists of all the source script files
- It converts the script commands into GUI commands. On the same machine, commands can be run on a remote or host machine
- To identify the movement of the mouse along with keystrokes, Silktest can be executed. It can avail both playback and record method or descriptive programming methods to get the dialogs
- It identifies all controls and windows of the application under test as objects and determines all of the attributes and properties of each window

Why Automation is Important for Your Career?

Testing is an integral part of any successful software project. The type of testing (manual or automated) depends on various factors, including project requirements, budget, timeline, expertise, and suitability. Three vital factors of any project are of course time, cost, and quality. The goal of any successful project is to reduce the cost and time required to complete it successfully while maintaining quality output.

Many organization have developed test automation frameworks based on the above mention test automation tools. Automation frameworks designed and developed in future will be more autonomous in future and would require minimal intervention by the testing professional.

Implementing automated testing can be challenging initially but the key benefits you are going to gain make it very important for your career.

Time-saving

By implementing automated testing, it can help improve code quality and development velocity. When code changes are not causing errors, developers will have more time to focus on each sprint goal. For example: with automation testing, you can save time with data-driven testing.

Higher test coverage

Automated software testing can help you save time and increase test coverage. Lengthy tests are usually time-consuming and laborious to perform manually; this can be run with automated testing unattended on different computers with different configurations. Automated testing also provides testers more time and effort to focus on more futures, thus leading to a higher quality of the application.

Higher accuracy

For the manual testing, human errors are inevitable during repetitive and monotonous manual tests. When it is done with automated testing, it can help avoid the risks of human errors, increase accuracy and save time.

More fascinating

You can never get bored while conducting automation testing as the setting up test cases take a coding brain and thought, which keeps the best of your technical minds engrossed and dedicated to the process.

The results are public

In the cases of manual testing, the rest of the team involved is not able to see the results while the tests are being run. With automated tests, though, people can simply sign into the testing system and see the yielded results. This puts a good base for superior team cooperation and eventually a better final product.

What is Selenium?

Selenium is an open-source automated testing suite which has been designed for web applications spanning across different browsers and platforms. While being similar to Quick Test Pro (QTP), Selenium concentrates on automation of web-based applications. It is a collection of different tools, and it also has different developers as well.

Advantage of Selenium

Selenium is an open-source tool and support for the same exists for all the browsers that exist currently. It also supports popular programming languages such as Perl, Ruby, Java and .Net allowing the user to pick and choose the one that suits them the best. The look and feel of a web

application is required to be tested across many web browsers to deliver user experience that is flawless and perfect. There are many test automation tools which are relied upon by testing professionals to test web applications most effectively. Such tools most often fail to test the application when changes and enhancements are made to the code on a frequent basis. Selenium makes it easier for testers to evaluate the respective web application without investing any extra time and effort.

The some major advantages of Selenium are as below:

- Use of one test script across multiple web browsers: Selenium permits testers to write the test script a single time and run the same on several web browsers.
- Option to choose programming language: In Selenium users can write test scripts using any programming language.
- Supports Multiple Testing Frameworks: Selenium being a web application testing tool, it can also be used as a GUI driving library. Therefore the users have the choice to use Selenium for wrapping test scripts in other test frameworks. Testers also have option to integrate Selenium while utilizing any framework of their choice.
- Smooth Integration with existing ecosystem: Selenium can be integrated with various tools that are widely used like QMetry and Hudson. Therefore, testers are able to integrate the testing tools with the existing solutions and suite of tools. The simple and easy integration process associated with Selenium does not need the users to additionally invest in commercial tools.

History of Selenium

Selenium was created by Jason Huggins in 2004. He was an engineer at ThoughtWorks and while working on a web application that needed repetitive testing. To increase the efficiency of the repetitive testing and control browser actions he invented a JavaScript program called "JavaScriptTestRunner". The potential for this was quickly realized and it was made into open-source and renamed as Selenium Core. The testers using Selenium Core had to install the whole application under test and the web server on their own local computers because of the restrictions imposed by the same origin policy. So another ThoughtWork's engineer, Paul Hammant, decided to create a server that will act as an HTTP proxy to "trick" the browser into believing that Selenium Core and the web application being tested come from the same domain. This system became known as the Selenium Remote Control or Selenium 1.

The same origin policy prohibits JavaScript code from accessing elements from a domain that is different from where it was launched.

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

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

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

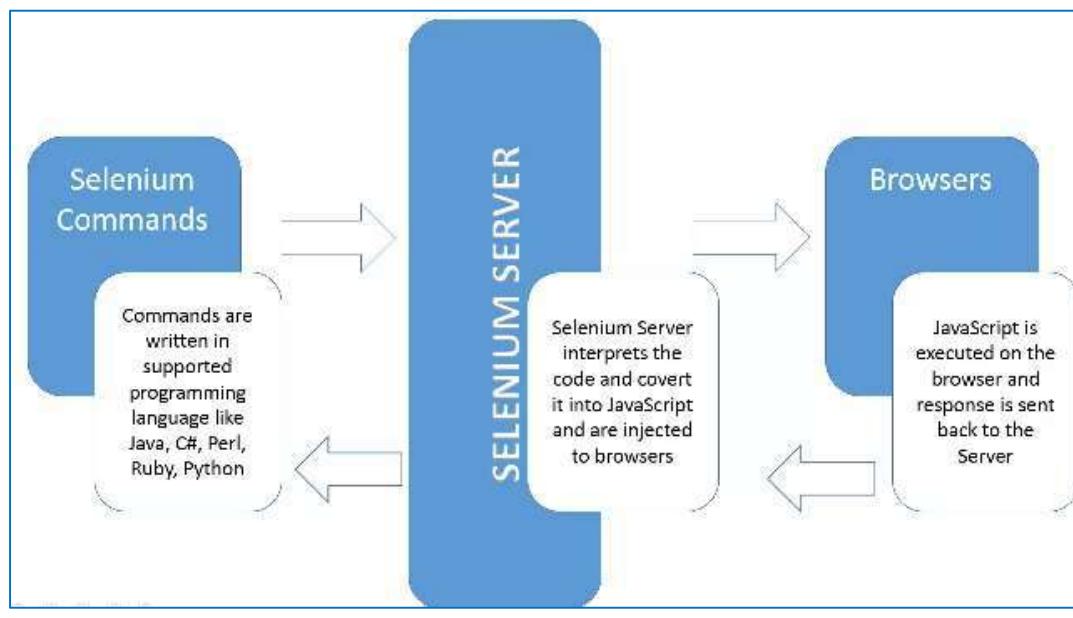
In 2008, the whole Selenium Team decided to merge WebDriver and Selenium RC to form a more powerful tool called Selenium 2, with WebDriver being the core. Currently, Selenium RC is still being developed but only in maintenance mode. Most of the Selenium Project's efforts are now focused on Selenium 2.

Component of Selenium

Architecture of Remote Control

Selenium Remote Control (RC) is a test tool that allows to write automated web application UI tests in many programming language. As of version 2.25.0, RC can supports Java, C#, PHP, Python, Perl, Ruby. Selenium Remote Control is great for testing complex AJAX-based web applications. It can be used for any java script enabled browser.

In Selenium RC, there is a manual process called Selenium Server and is mandatory to start before execution, which acts as a middleman between the code and the browser. The commands (API's) are sent to Server. It interprets the command and converts it into JavaScript and then JavaScript is injected to the browser. Now the browser executes the javascript and responds to a server, which again interprets the command and returns to code in the respective language.



Architecture of Remote Control

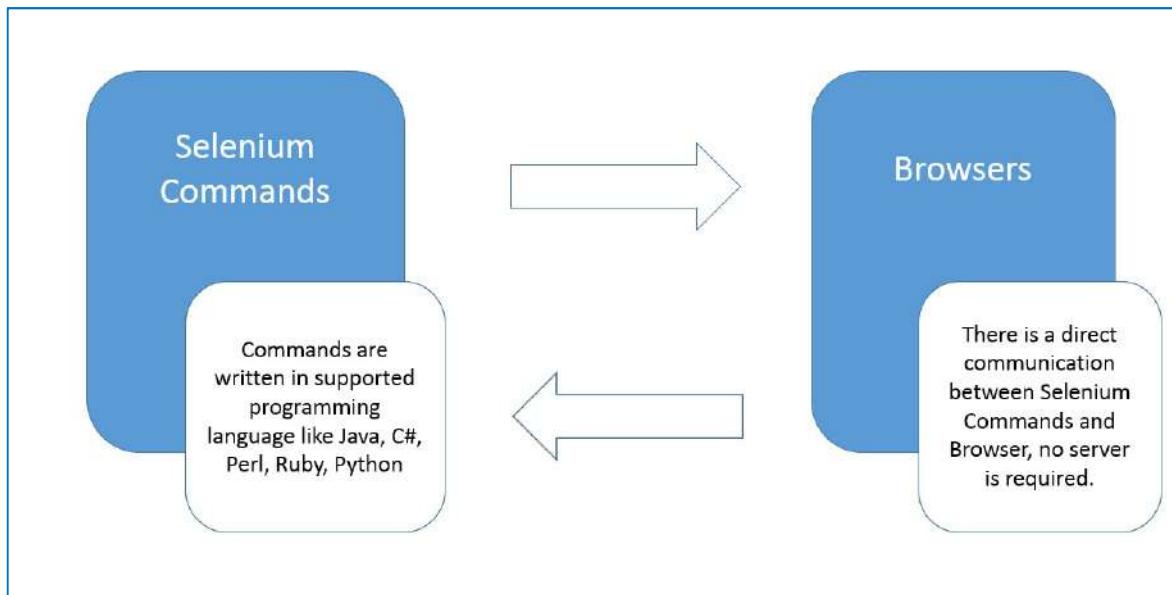
In RC server has to be restart again and again i.e. RC completely depends upon server. Switching between the multiple instances of the same browser is not possible. Switching between the multiple instances of the different browsers is not possible.

Architecture of WebDriver

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

- Open source
- Supports all the key vendors of the browser like Mozilla Firefox, Internet Explorer, Google Chrome, Safari, etc.
- Support Multiple languages like C#, JAVA, Ruby, Perl, Python, and PHP.
- Supports multiple platforms like Linux, Windows, MAC, etc.
- No middleman like Selenium RC server is required.
- Easy to remember API's.
- Easy to integrate with Testing frameworks.
- Framework Development.
- Parallel Testing capabilities.
- The API's written in WebDriver can directly interact with browsers.

The architecture of WebDriver is as below:



Architecture of GRID

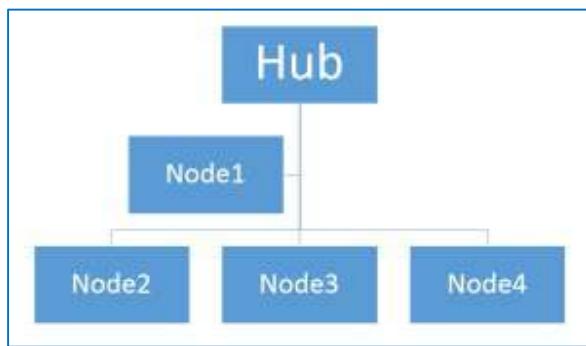
Selenium Grid is a tool is the last component of the selenium suite and is used together with Selenium RC to run parallel tests across different machines and different browsers all at the same time. Parallel execution means running multiple tests at once. The various features of GRDI are:

- Enables simultaneous running of tests in multiple browsers and environments.

- Fast Execution and Saves time enormously by reduces the execution time as test cases are executed parallel.
- Platform Independent, support almost all Operating System
- Language Independent.
- Utilizes the hub-and-nodes concept. The hub acts as a central source of Selenium commands to each node connected to it.

In Grid one of the systems is created as Hub. Hub works as a central point controlling all the nodes. Nodes are an actual machine on which execution is done.

The architecture of GRID is as below:



Difference between API's and GUI's

There are many approaches to test automation. The general approaches used widely are:

- Graphical user interface testing. A testing framework that generates user interface events such as keystrokes and mouse clicks, and observes the changes that result in the user interface, to validate that the observable behavior of the program is correct. Many test automation tools provide record and playback features that allow users to interactively record user actions and replay them back any number of times, comparing actual results to those expected. The advantage of this approach is that it requires little or no software development. This approach can be applied to any application that has a graphical user interface. However, reliance on these features poses major reliability and maintainability problems. Relabeling a button or moving it to another part of the window may require the test to be re-recorded. Record and playback also often adds irrelevant activities or incorrectly records some activities.
- API driven testing. A testing framework that uses a programming interface to the application to validate the behavior under test. Typically API driven testing bypasses application user interface altogether. It can also be testing public (usually) interfaces to classes, modules or libraries are tested with a variety of input arguments to validate that the results that are returned are correct. API testing is widely used by software testers due to the difficulty of creating and maintaining GUI-based automation testing. It involves directly testing APIs as part of integration testing, to determine if they meet expectations for functionality, reliability, performance, and security. API testing is

considered critical when an API serves as the primary interface to application logic since GUI tests can be difficult to maintain with the short release cycles and frequent changes commonly used with agile software development and DevOps.

DUCAT®
www.ducatindia.com

Summary

- Automation Testing uses an automation software to execute test case suite
- Selenium is a software testing tool used for Regression Testing
- QuickTest Professional (QTP) is widely used for functional and regression testing as it addresses every major software application and environment
- Rational functional tester is an Object-Oriented automated Functional Testing tool that is capable of performing automated functional, regression, data-driven testing and GUI testing
- Watir is an open source testing software for regression testing
- Selenium Grid is a tool is the last component of the selenium suite and is used together with Selenium RC to run parallel tests across different machines and different browsers all at the same time

DUCAT®
www.ducatindia.com

Chapter 3 – Setup Eclipse

Download and Install Java

The following are steps to install Java SE 9.0.1 in windows environment

Step 1: Go to [link: http://www.oracle.com/technetwork/java/javase/downloads/index.html](http://www.oracle.com/technetwork/java/javase/downloads/index.html). Click on Download JDK. For java latest version.



Java SE Downloads

Java Platform (JDK) 9

NetBeans with JDK 8

Java Platform, Standard Edition

Java SE 9.0.1
Java SE 9.0.1 includes important bug fixes. Oracle strongly recommends that all Java SE 9 users upgrade to this release.
[Learn more](#)

▪ Installation Instructions
▪ Release Notes
▪ Oracle License
▪ Java SE Licensing Information User Manual
▪ Third Party Licenses
▪ Certified System Configurations
▪ Readme

JDK
[DOWNLOAD](#)

Server JRE
[DOWNLOAD](#)

JRE
[DOWNLOAD](#)

Step 2: Next,

1. Accept License Agreement
2. Download Java JDK according to the version (32 or 64 bit) of java for Windows.

Java SE Development Kit 9 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, 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: From your Oracle account, select **Subscriptions**, expand **Technology**, and subscribe to Java.
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

[JDK 9.0.1 checksum](#)

Java SE Development Kit 9.0.1

You must accept the [Oracle Binary Code License Agreement for Java SE](#) to download this software.

1

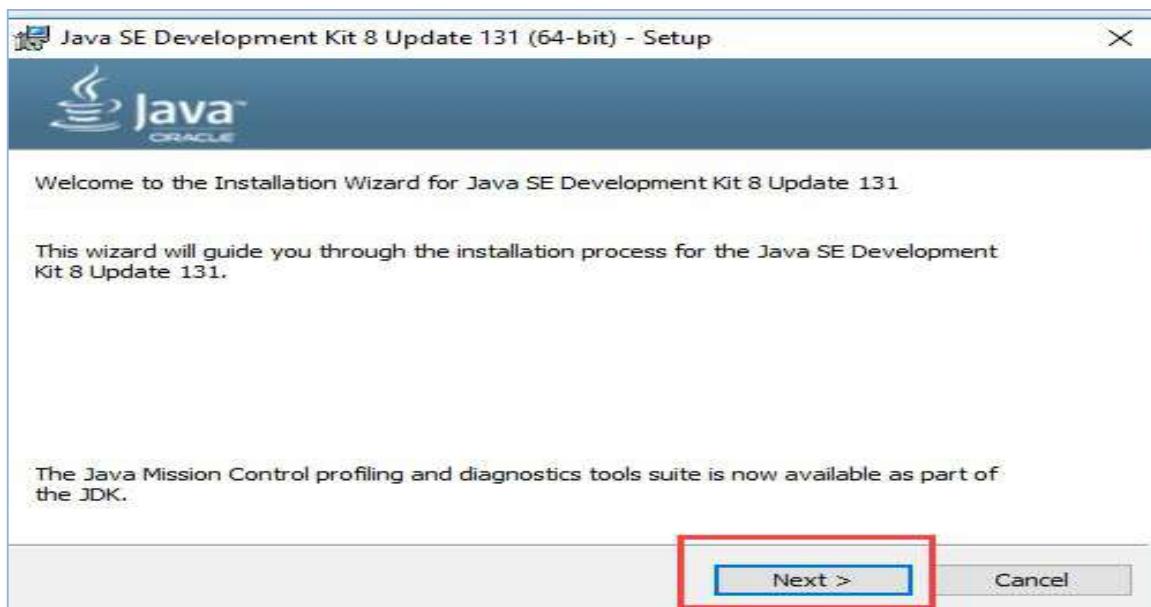
Accept License Agreement

Decline License Agreement

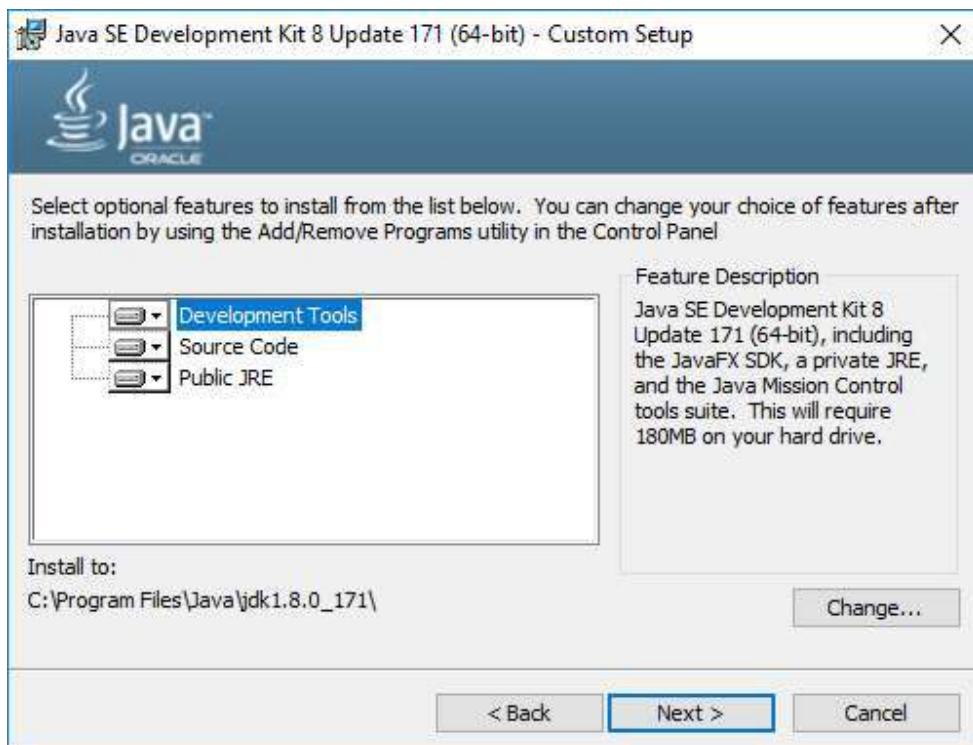
Product / File Description	File Size	Download
Linux	304.99 MB	jdk-9.0.1_linux-x64_bin.rpm
Linux	338.11 MB	jdk-9.0.1_linux-x64_bin.tar.gz
macOS	382.11 MB	jdk-9.0.1_osx-x64_bin.dmg
Windows	375.51 MB	jdk-9.0.1_windows-x64_bin.exe
Solaris SPARC	200.85 MB	jdk-9.0.1_solaris-sparcv9_bin.tar.gz

2

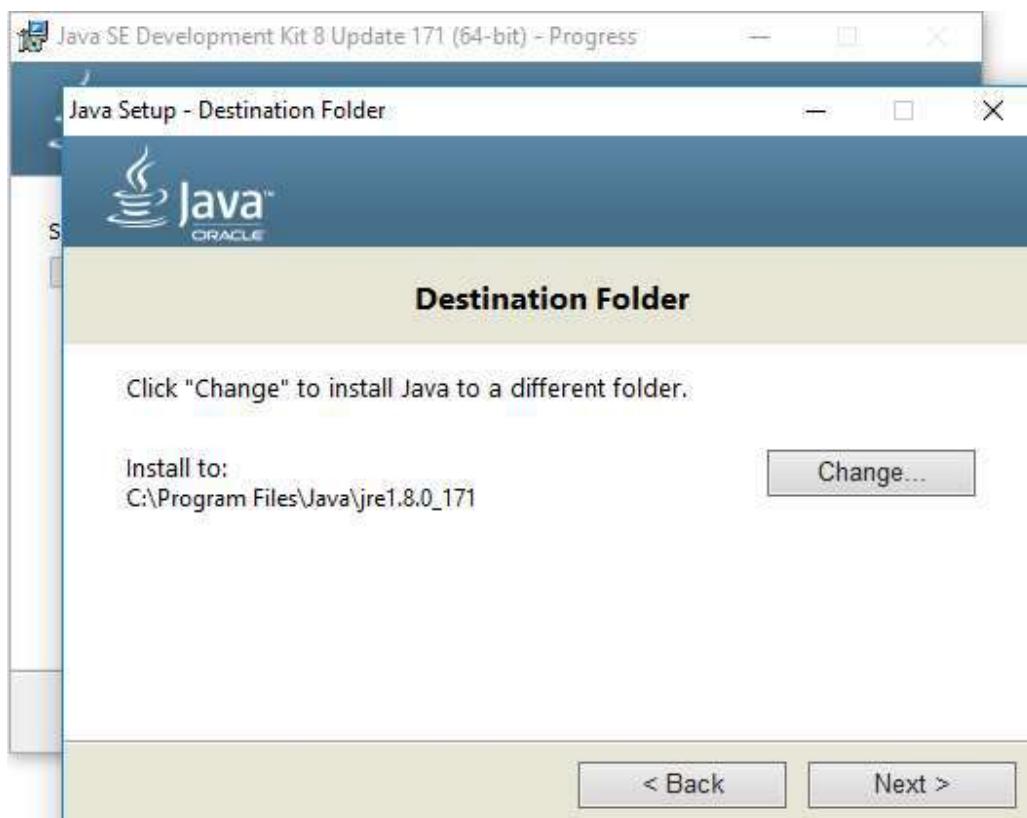
Step 3: once the download is complete, run the exe for install JDK. Click Next



Step 4: On the Custom Setup screen, leave the installation folder as it is and click on Next button.



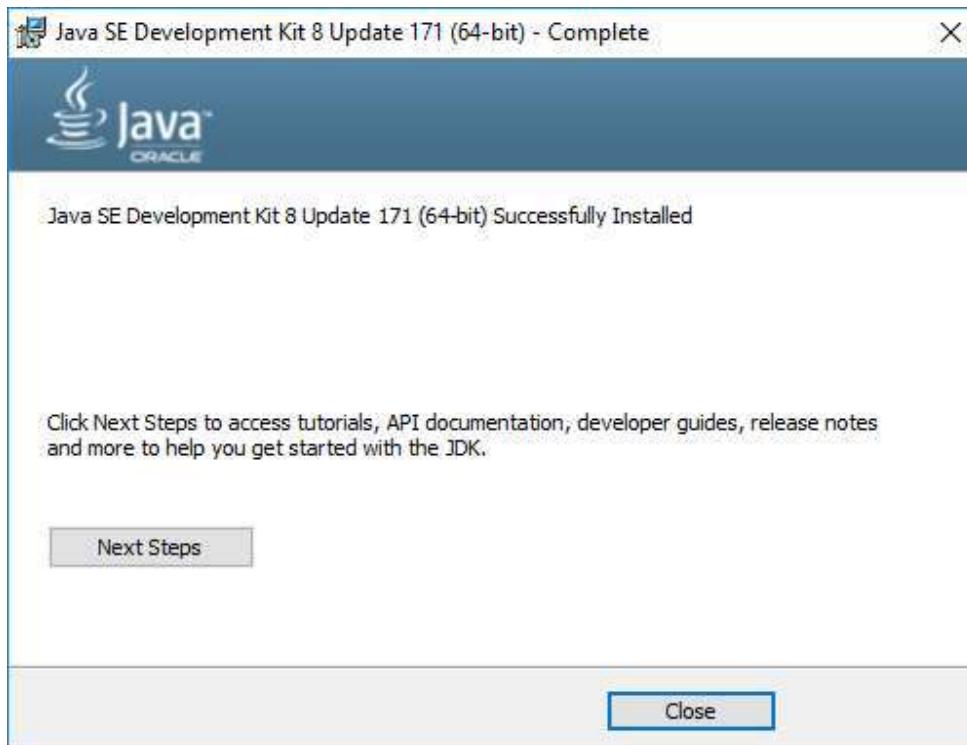
Step 5: Wait for few seconds for Java to install some files. After that, specify the destination folder of JRE. Leave the default location as it is and click on Next button



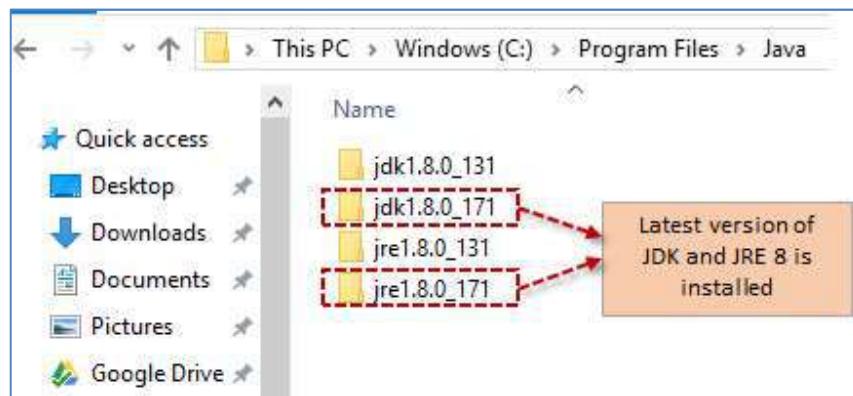
Step 6: JDK installation would now begin



Step 7: Once the installation is completed, the window will be shown as the below screenshot.
Click on Close button to close the Window



Step 8: To verify that the installation is successful, open Java folder in Program files. New JRE & JDK folders will be there with the latest version numbers. May be the previous versions of JRE or JDK folders will also be there, so don't worry as multiple versions of JDKs & JREs can co-exist.



Download and Start Eclipse

Eclipse is an integrated development environment (IDE) for Java and other programming languages like C, C++, PHP, and Ruby. Development environment provided by Eclipse includes

the Eclipse Java development tools (JDT) for Java, Eclipse CDT for C/C++, and Eclipse PDT for PHP, among others. The steps to download Eclipse for Java Developers, extract and save it in any drive are:

Step 1: Eclipse can be downloaded from <http://www.eclipse.org/downloads/>. The download page lists a number of versions of eclipse.

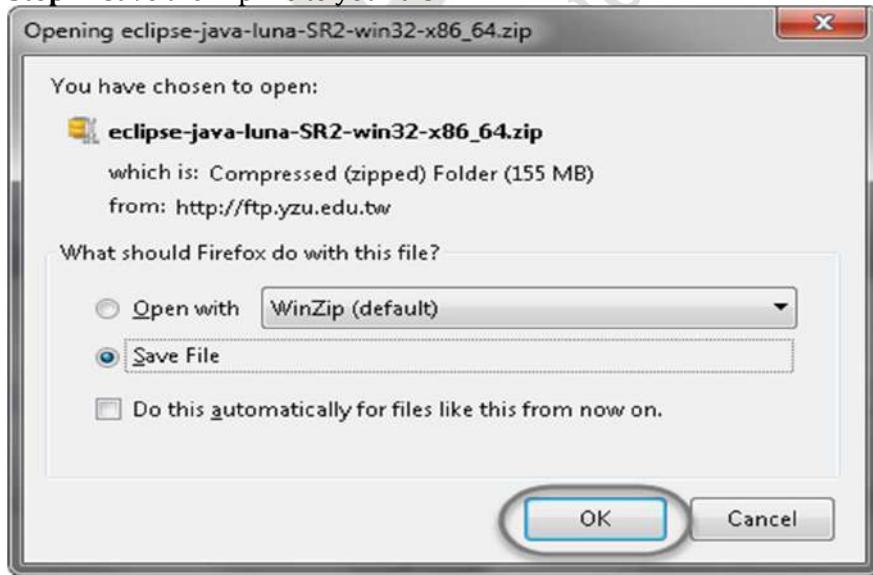


Packaging	Description	Download Count	Details	Windows 32 Bit	Windows 64 Bit
Eclipse IDE for Java EE Developers	221 MB	Downloaded 498,420 Times	Details		Windows 32 Bit Windows 64 Bit
Eclipse Classic 4.2	182 MB	Downloaded 369,565 Times	Details Other Downloads		Windows 32 Bit Windows 64 Bit
Eclipse IDE for Java Developers	149 MB	Downloaded 199,905 Times	Details		Windows 32 Bit Windows 64 Bit

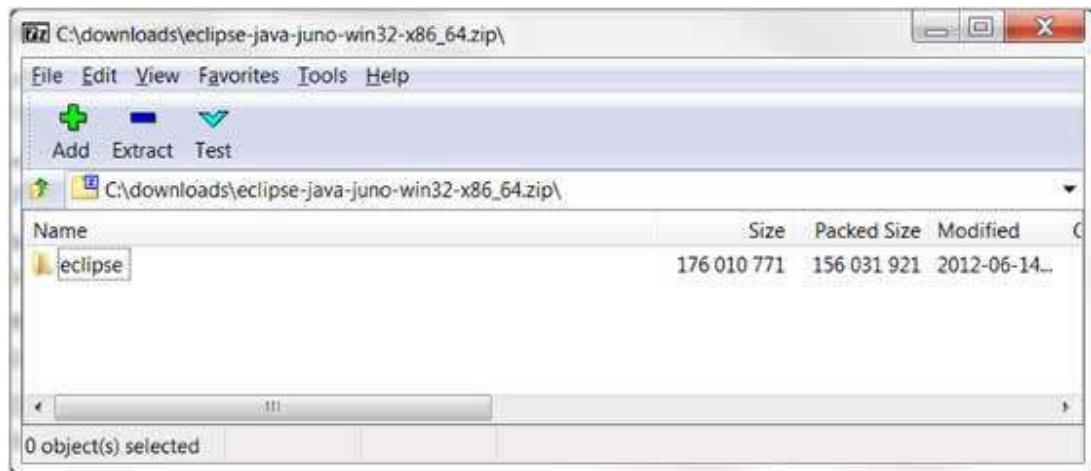
The capabilities of each packaging of eclipse are different. Java developers typically use Eclipse Classic or Eclipse IDE for developing Java applications.

The drop down box in the right corner of the download page allows to choose the operating system (Windows, Linux and Mac) on which eclipse is to be installed. Eclipse is packaged as a zip file.

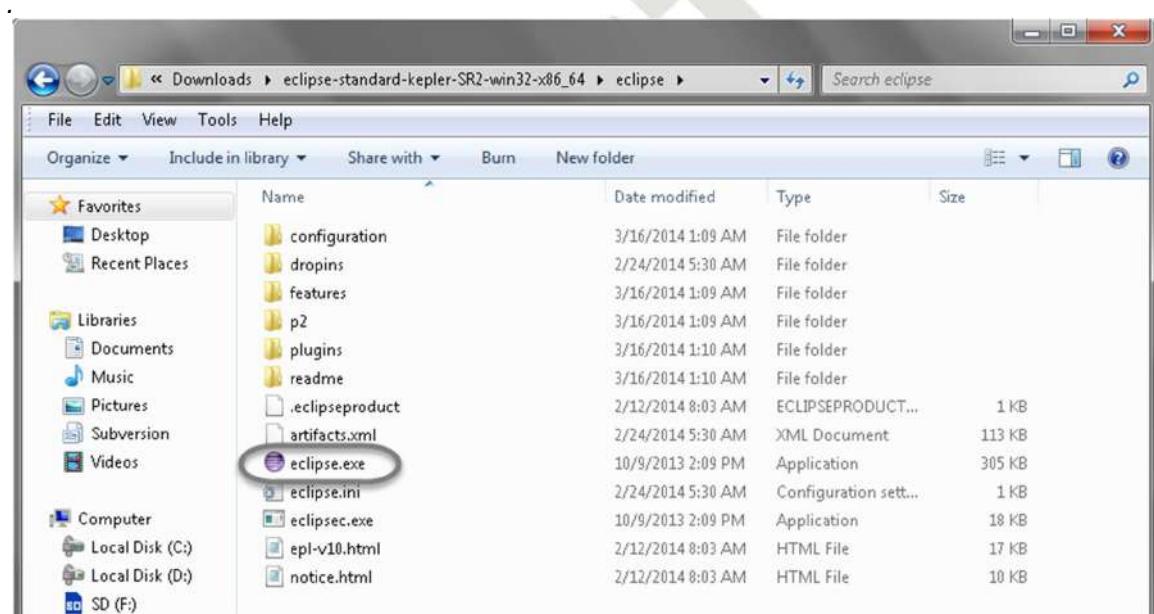
Step 2: Save the .zip file to your disk.

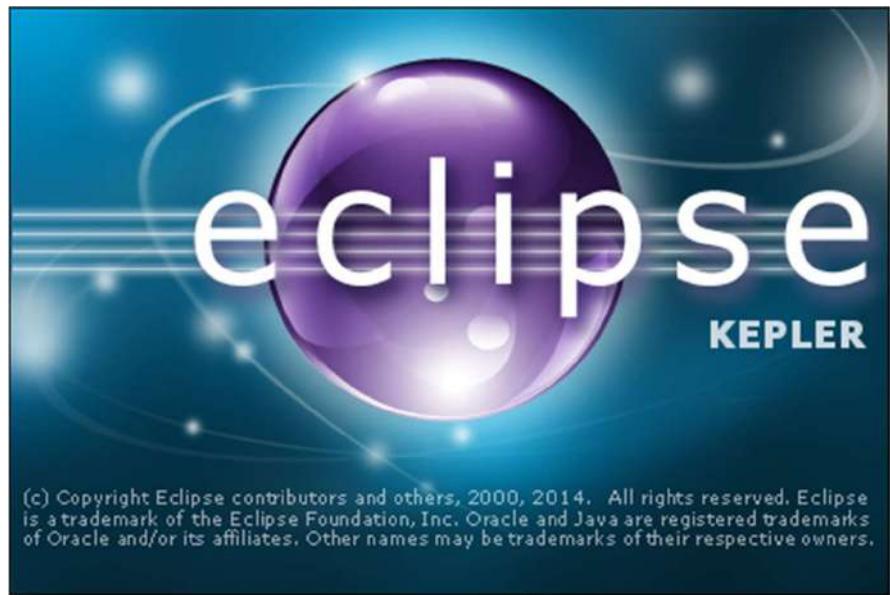


Step 3: After downloading the *Eclipse* archive, a tool (7-zip, PeaZip, IZArc) is needed to extract the contents of a zip file which will create the unzipped Eclipse folder. Open the eclipse folder.

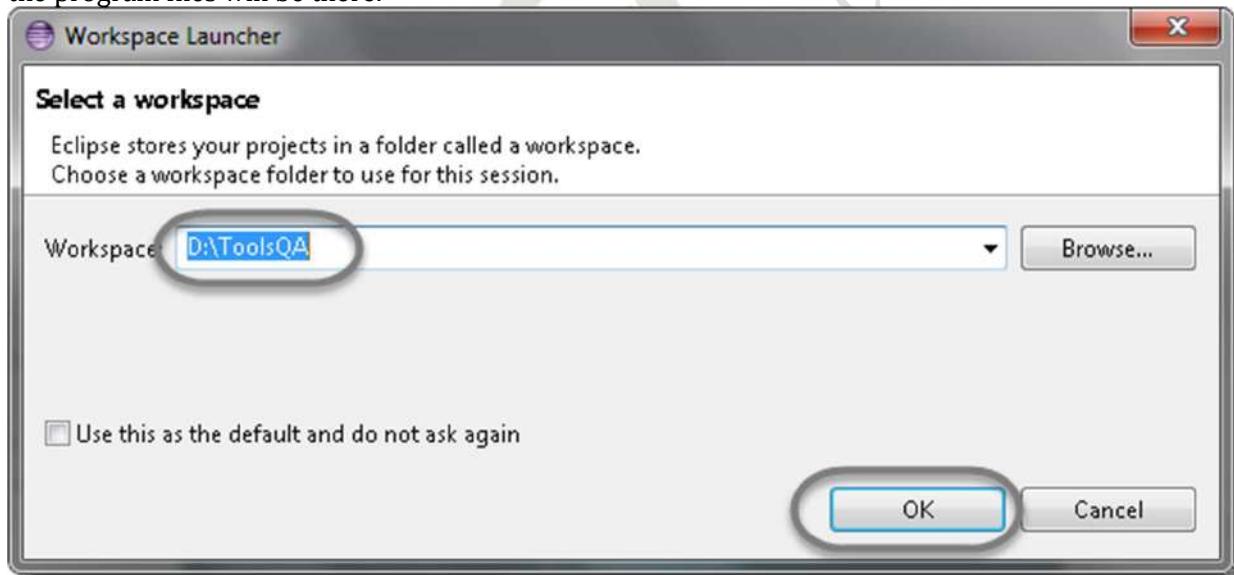


Step 4: Eclipse IDE does not have any installer, there will be a file inside the Eclipse folder named `eclipse.exe`. Just double click on the file to run Eclipse.

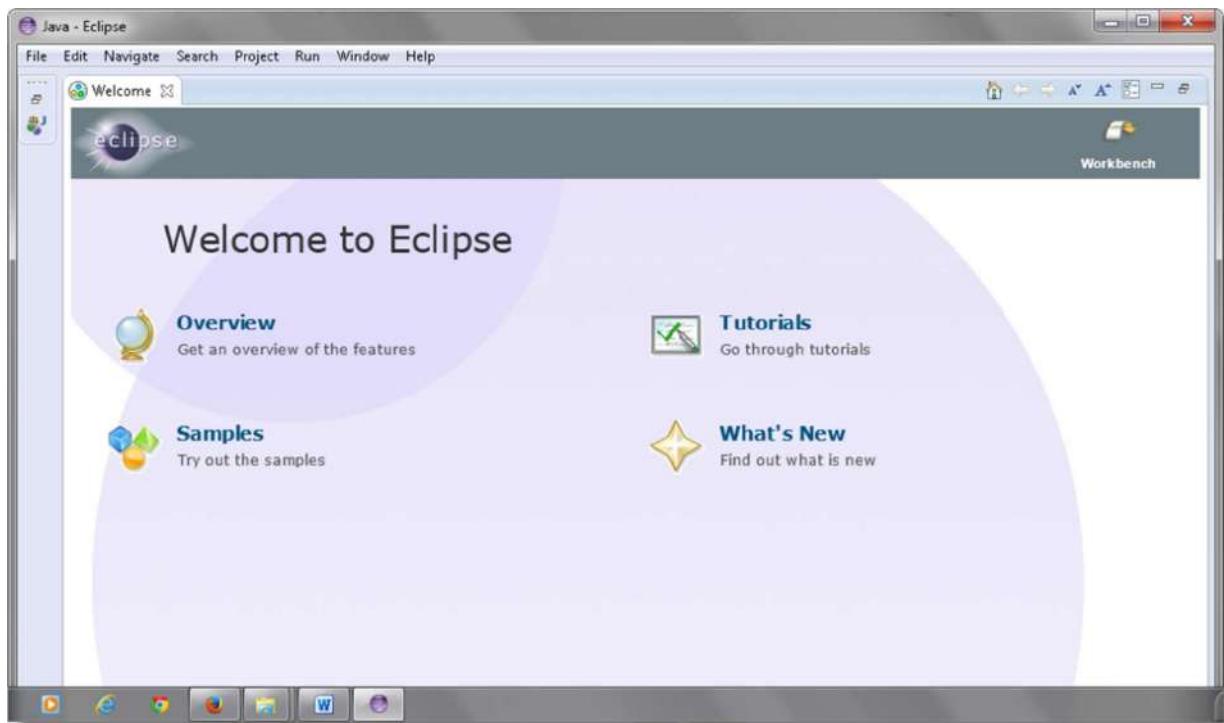




Step 5: After eclipse has been fully installed and extracted, create a workspace folder where all the program files will be there.



Step 6: After that the Welcome to Eclipse window will be shown. After completion of installation of eclipse restart the computer so that it refreshes system memory and allows changes made by installers and uninstallers to take effect.



Download and Configure WebDriver Java Client

The steps to download Selenium WebDriver are:

Step 1: Open Selenium download section using this link

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

Step 2: A section called **Selenium Client & WebDriver Language Bindings will be shown**. There will be different download links respective to different languages such as Java, C#, Ruby. For using Java with Selenium, download the Java specific drivers. To do so, click on Download from the Java section.

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 scripts, 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	3.12.0	2018-05-08	Download	Change log	Javadoc
C#	3.12.0	2018-05-08	Download	Change log	API docs
Ruby	3.12.0	2018-05-08	Download	Change log	API docs
Python	3.12.0	2018-05-08	Download	Change log	API docs
Javascript (Node)	4.0.0-alpha.1	2018-01-13	Download	Change log	API docs

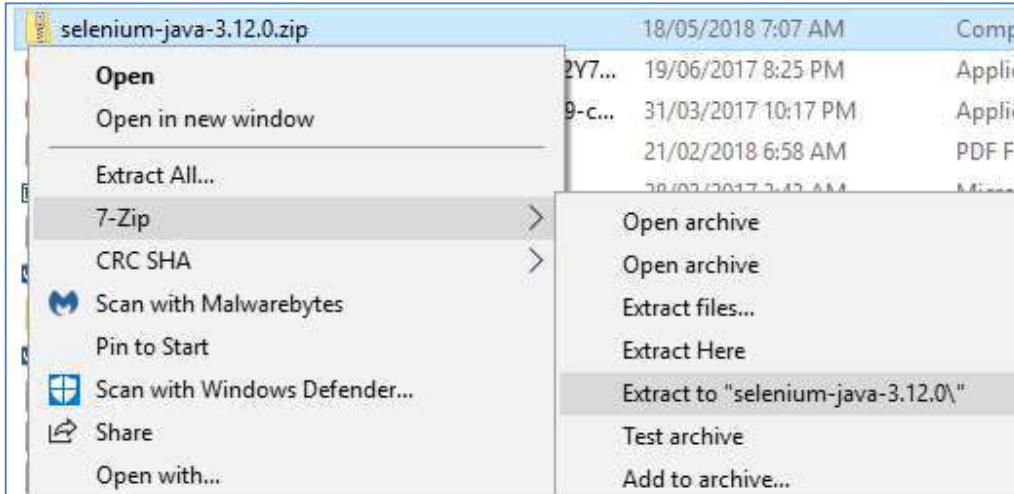
C# NuGet

NuGet latest release is 3.12.0, Released on 2018-05-08

- [WebDriver](#)
- [WebDriverBackedSelenium](#)

Step 3: Selenium Webdriver JARs would start downloading.

Step 4: Once the file is downloaded, unzip it by using any extractor like 7-zip.



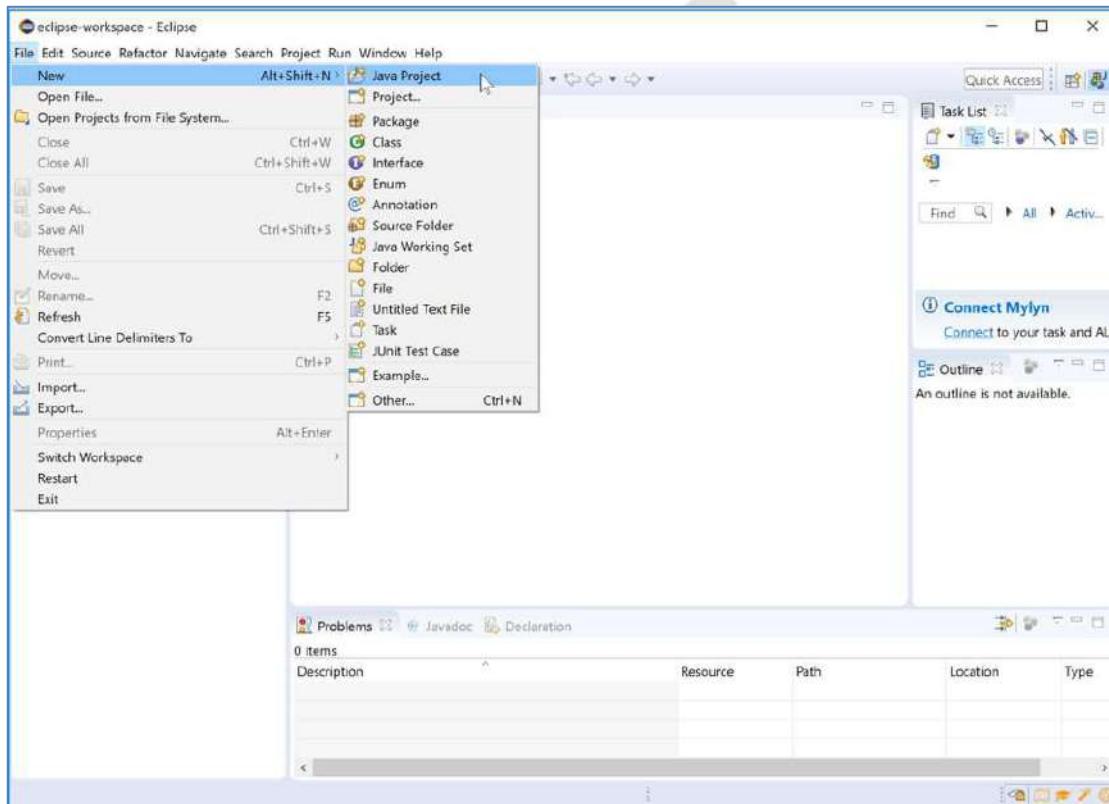
Step 5: Open the unzipped folder. It would look something like this. The JAR file names might change depending on the version. A libs folder, one or two other JAR files and some other files such as LICENSE, NOTICE will be there in the Selenium folder.

	Name	Date modified	Type
Quick access			
Desktop	libs	1/02/1985 1:00 AM	File folder
Downloads	CHANGELOG	1/02/1985 1:00 AM	File
Documents	client-combined-3.12.0.jar	1/02/1985 1:00 AM	Executable Jar File
Pictures	client-combined-3.12.0-sources.jar	1/02/1985 1:00 AM	Executable Jar File
Google Drive	LICENSE	1/02/1985 1:00 AM	File
	NOTICE	1/02/1985 1:00 AM	File

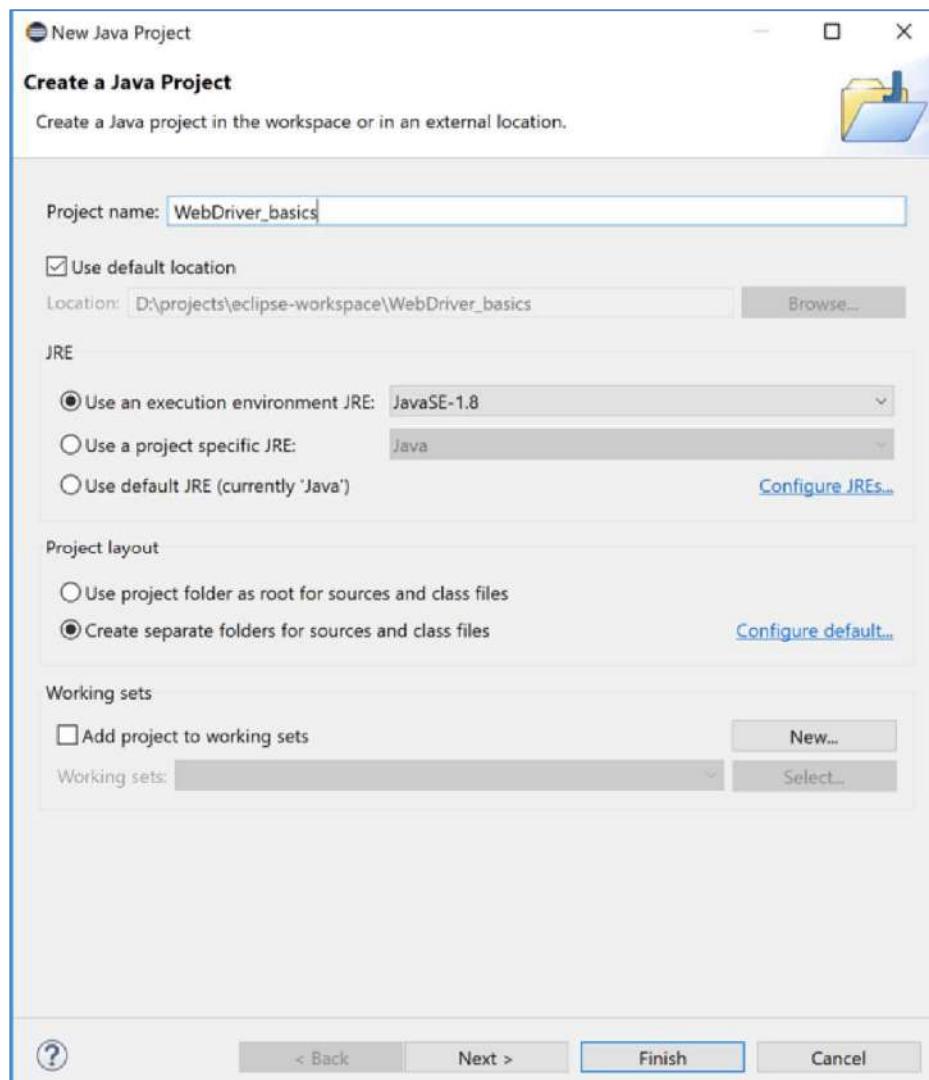
With this, the download process of the Selenium Webdriver is completed.

Setup a Project

Eclipse works with the concepts of workspaces—folders and spaces where projects can be created. A simple Java Project created by using the menu **File => New => Java Project**.



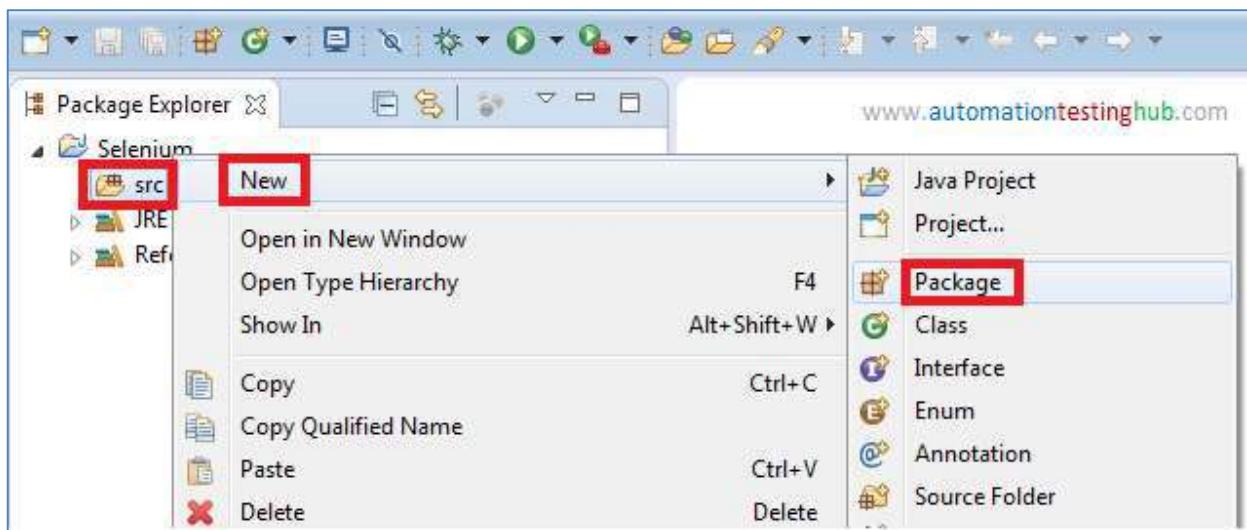
Fill out the basic information on the New Java Project dialog, and then click Finish to proceed.



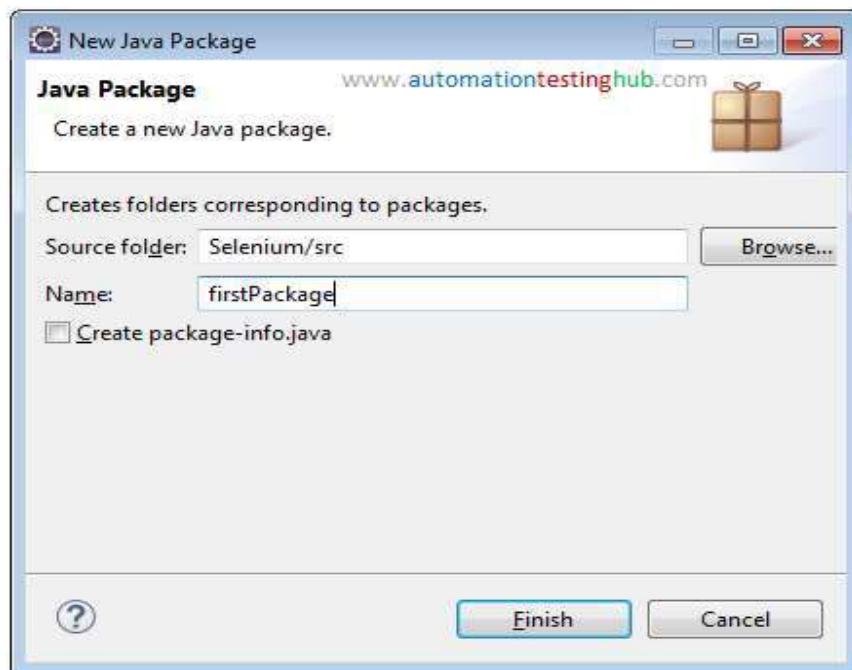
Create Packages

In Java, a new class is needed to write test scripts and it can be club with similar classes in packages. The steps to create a package and write a sample script is as below:

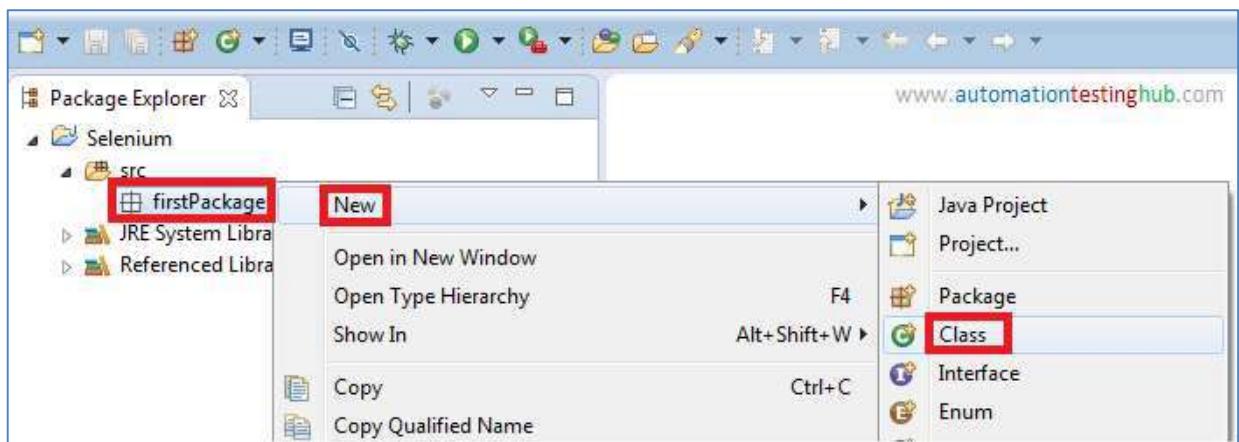
Step 1: Right click on src folder. Then select New > Package



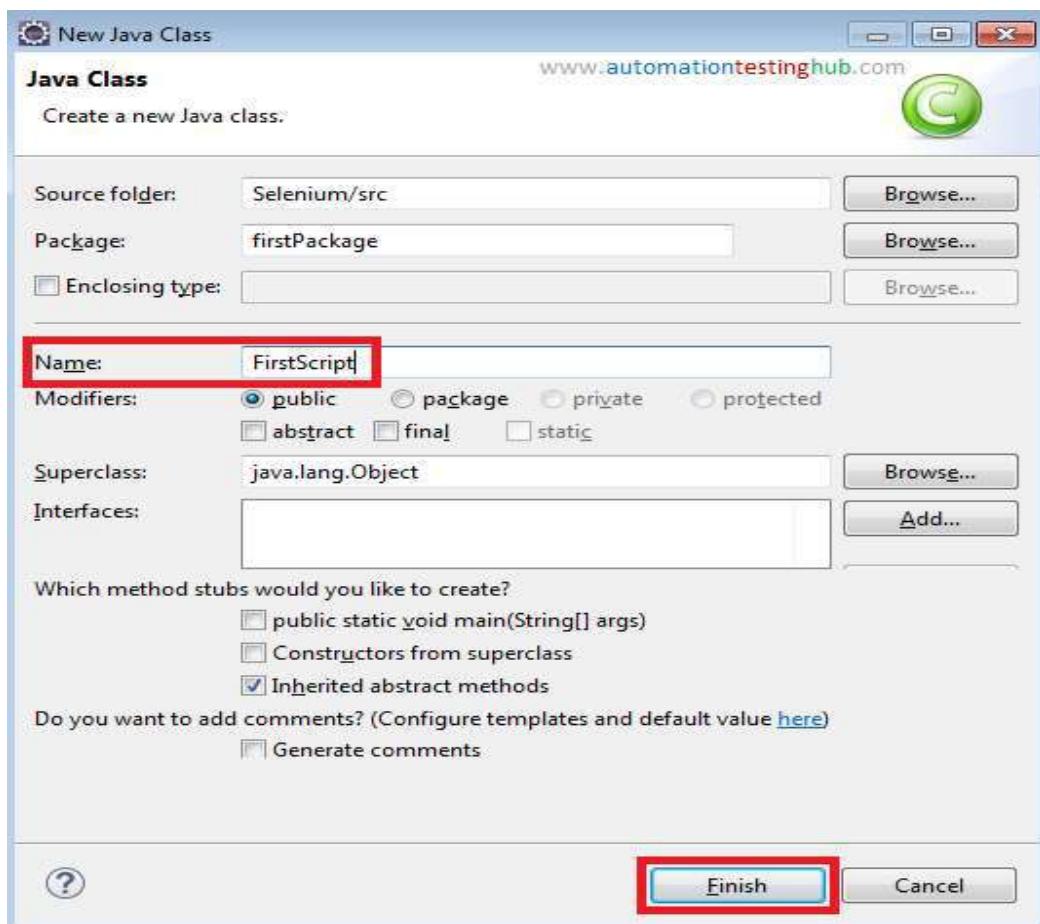
Step 2: In the New Java Package popup window, enter name as firstPackage and click on Finish button.



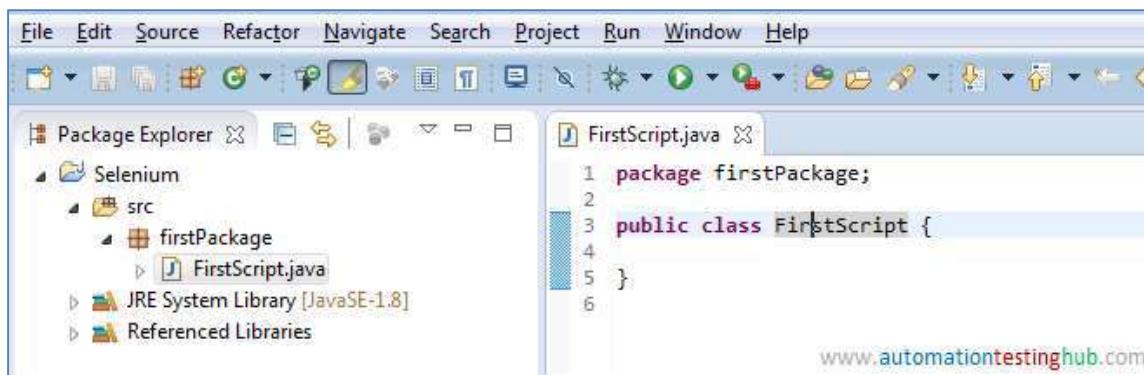
Step 3: The package will be displayed under src folder. Right click on the package, and then Select New > Class



Step 4: In the New Java Class popup window, write the name as FirstScript and click on Finish button



Step 5: The new class is now created and it looks as shown below. Here the script can be written.

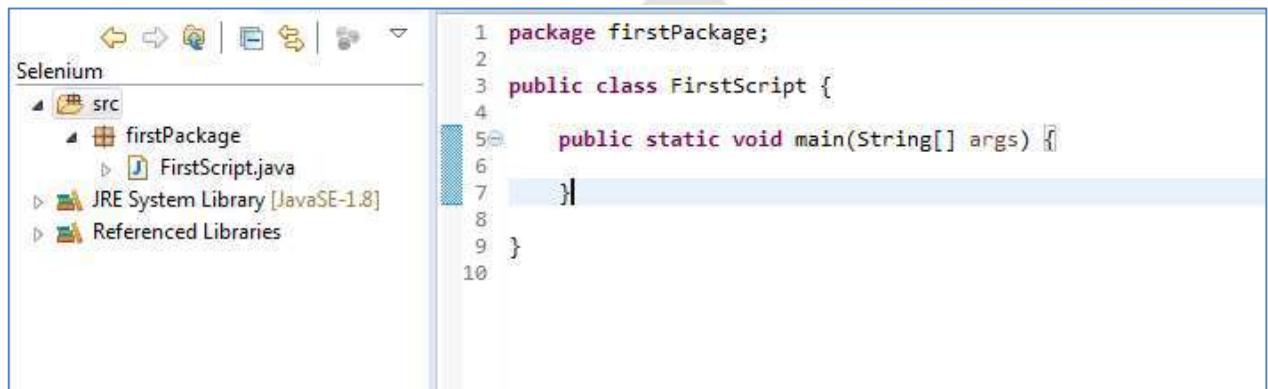


Step 6: Inside this class, A main() method has to be create. Main method acts as a starting point for Java, and is necessary to run the script . The code for main() method is below and should be added inside the class.

```

public static void main(String[] args) {
}

```



Step 7: Once the browser setup is complete, write the code snippet which launches the browser. The example below shows how the code would look like if Firefox is use as browser

Example:

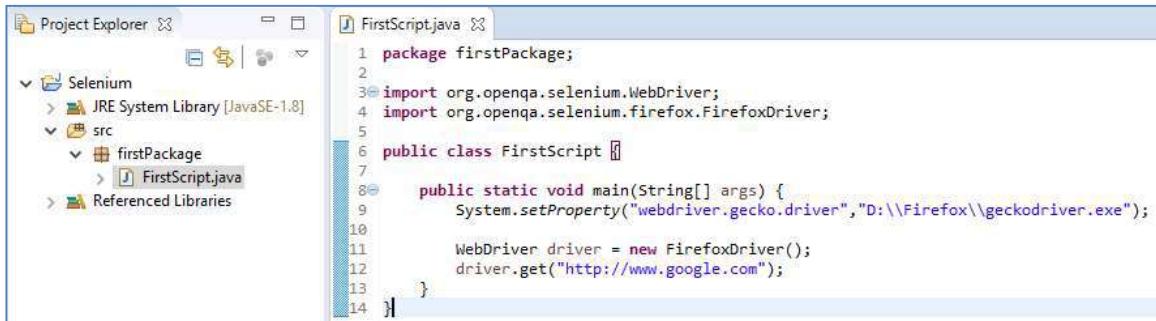
```

package firstPackage;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class FirstScript {
    public static void main(String[] args) {
        System.setProperty("webdriver.gecko.driver","D:\\Firefox\\geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        driver.get("http://www.google.com");
    }
}

```

}

Step 8: The eclipse code should look similar to the screenshot given below. Now run the code to see if it works fine.

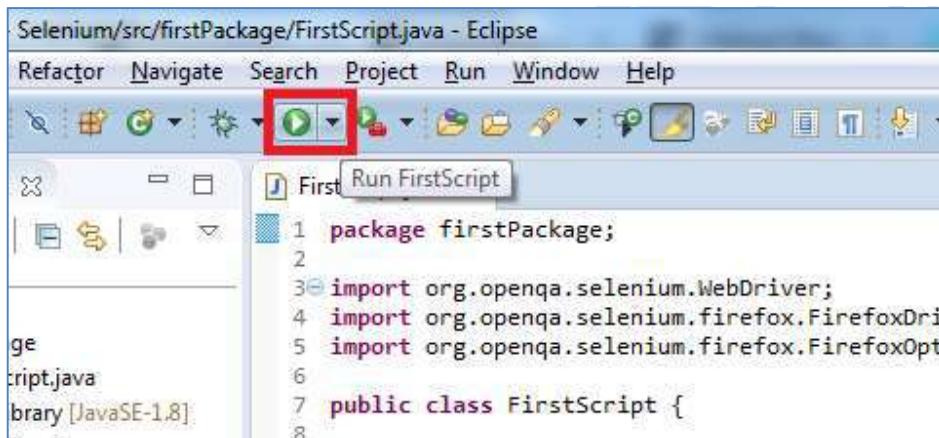


```

Project Explorer
Selenium
  JRE System Library [JavaSE-1.8]
  src
    firstPackage
      FirstScript.java
Referenced Libraries

FirstScript.java
1 package firstPackage;
2
3 import org.openqa.selenium.WebDriver;
4 import org.openqa.selenium.firefox.FirefoxDriver;
5
6 public class FirstScript {
7
8     public static void main(String[] args) {
9         System.setProperty("webdriver.gecko.driver","D:\\Firefox\\geckodriver.exe");
10
11         WebDriver driver = new FirefoxDriver();
12         driver.get("http://www.google.com");
13     }
14 }
```

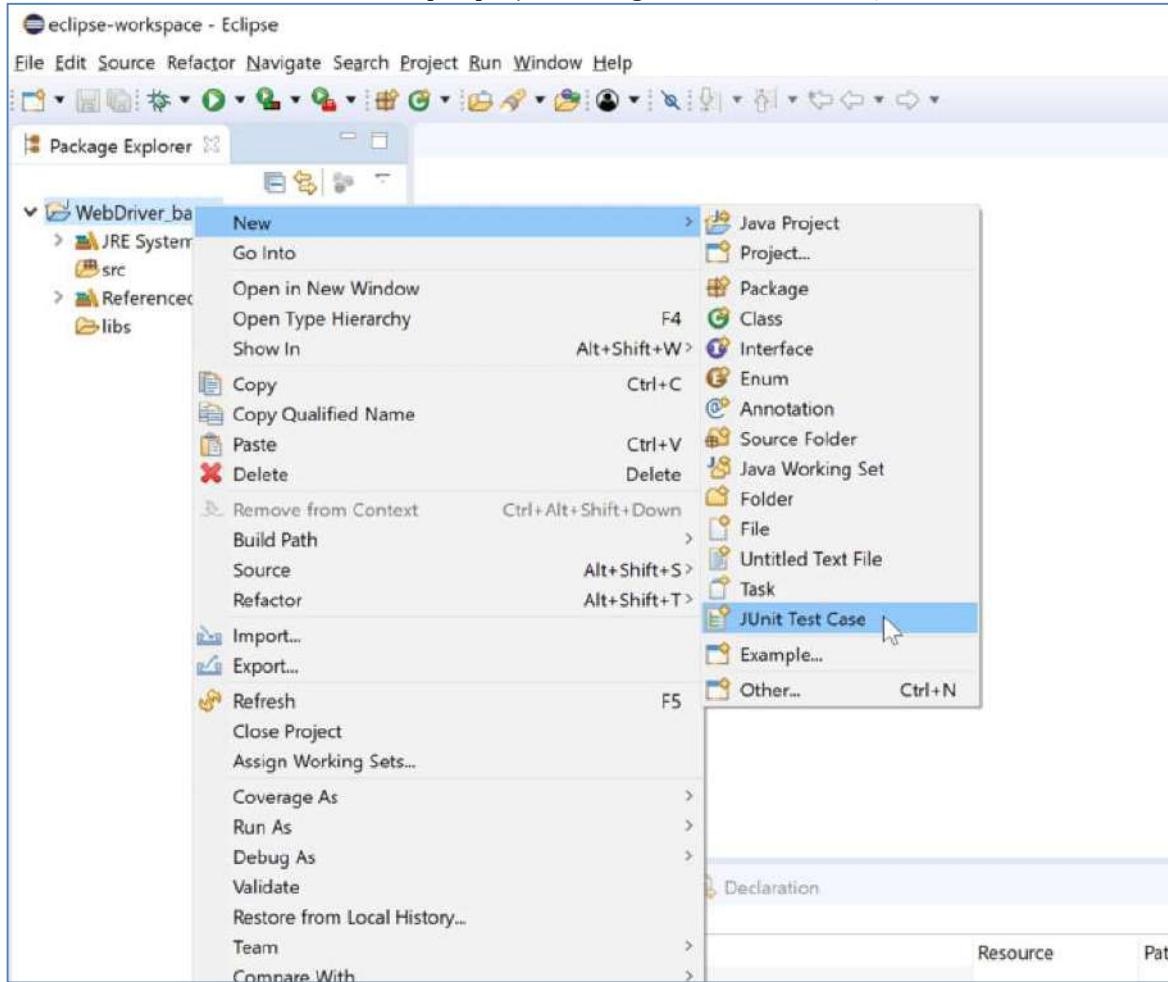
Step 9: To run the script, click on the run icon from the toolbar. If everything works fine, a new browser window will open.



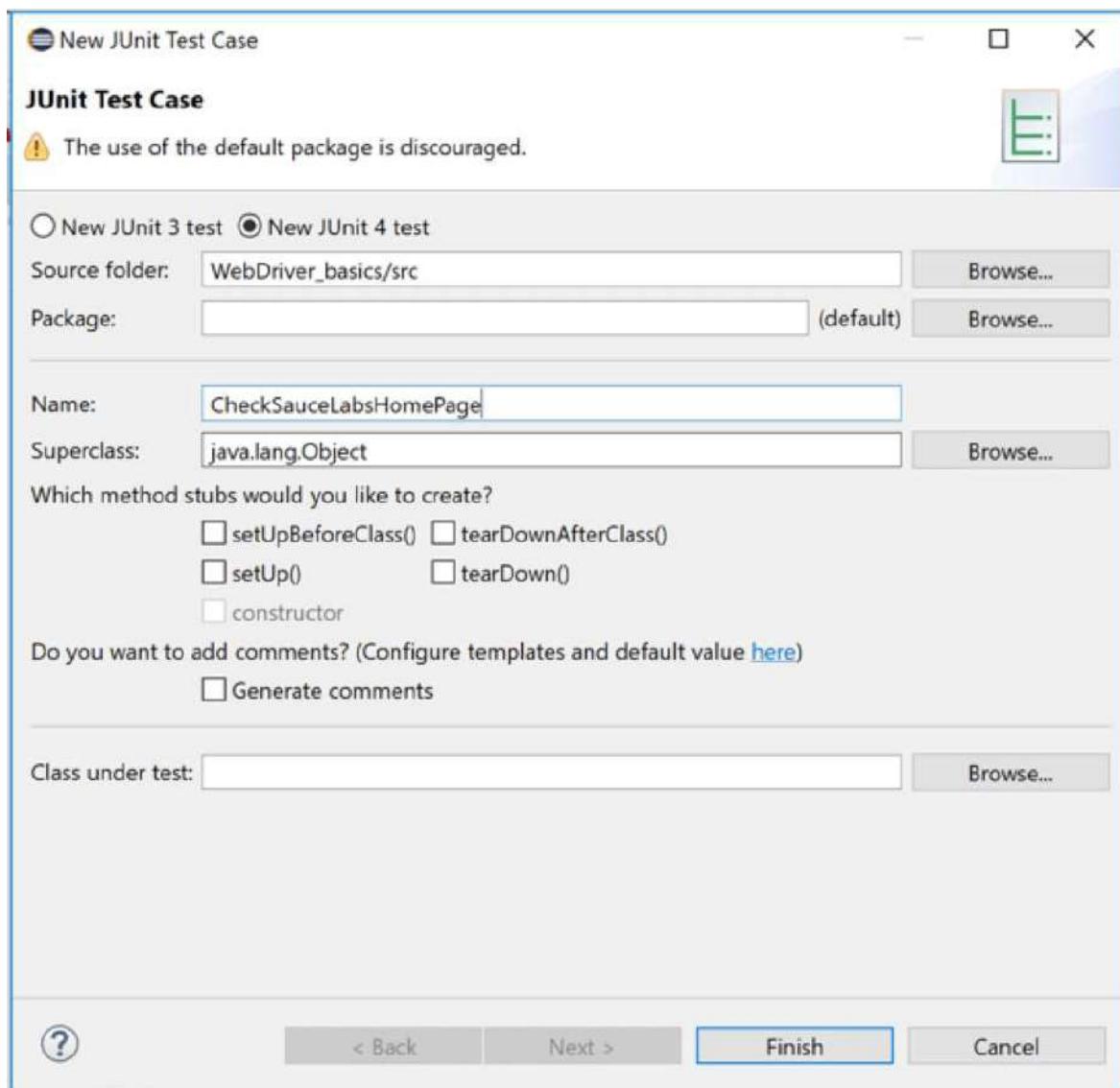
Create a First Java Test Case

WebDriver doesn't know how to do anything other than talk to the browser driver. As a result to execute the tests, some sort of test framework is needed. Here JUnit is used because it's very popular and is included in Eclipse's default installation.

Add a test case to the Eclipse project via right click => New => JUnit Test Case.



Give the test case a good name in the resulting dialog and click Finish to create the file.



A SIMPLE TEST

Below is a complete test case that starts a browser locally, executes a very simple test, then closes out the browser instance.

```

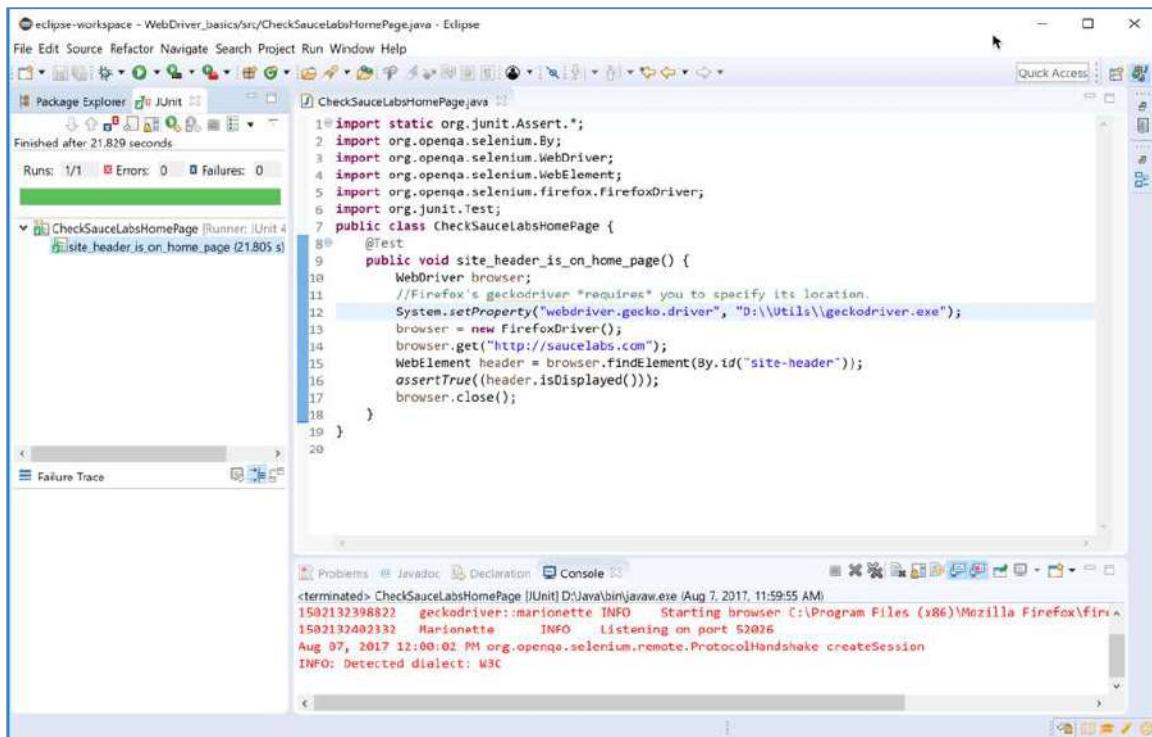
import static org.junit.Assert.*;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.junit.Test;

```

```
public class CheckSauceLabsHomePage {  
  
    @Test public void site_header_is_on_home_page() {  
  
        WebDriver browser;  
  
        //Firefox's geckodriver requires you to specify its  
        location. System.setProperty("webdriver.gecko.driver", "/User  
s/jimholmes/Utils/geckodriver");  
  
        browser = new  
        FirefoxDriver(); browser.get("http://saucelabs.com");  
  
        WebElement header = browser.findElement(By.id("site-  
header"));  
  
        assertTrue(header.isDisplayed());  
  
        browser.close();  
    }  
}
```

RUNNING THE TEST

Execute the test by right-clicking in the test body and select Run As => JUnit Test Case. Alternatively, use the shortcut combo Alt-Shift-X, T. The test will run and see the results in the JUnit Test Explorer pane.

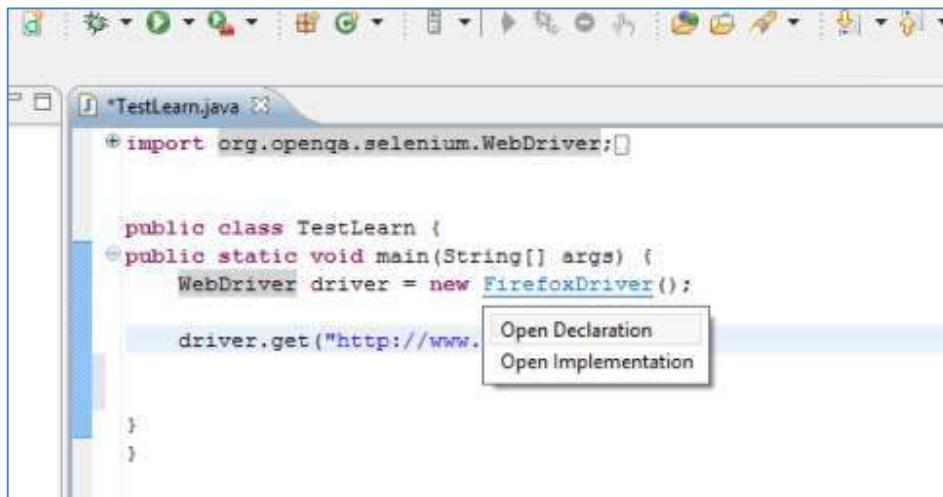


Import WebDriver Source File

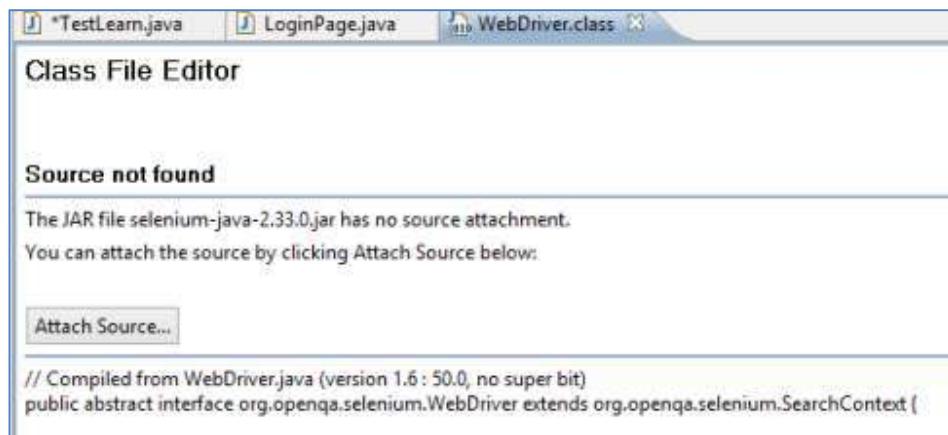
An example of import of webdriver source file using the structure of FirefoxDriver class is below:

Step 1: Move mouse over to the FirefoxDriver class with “Ctrl” button

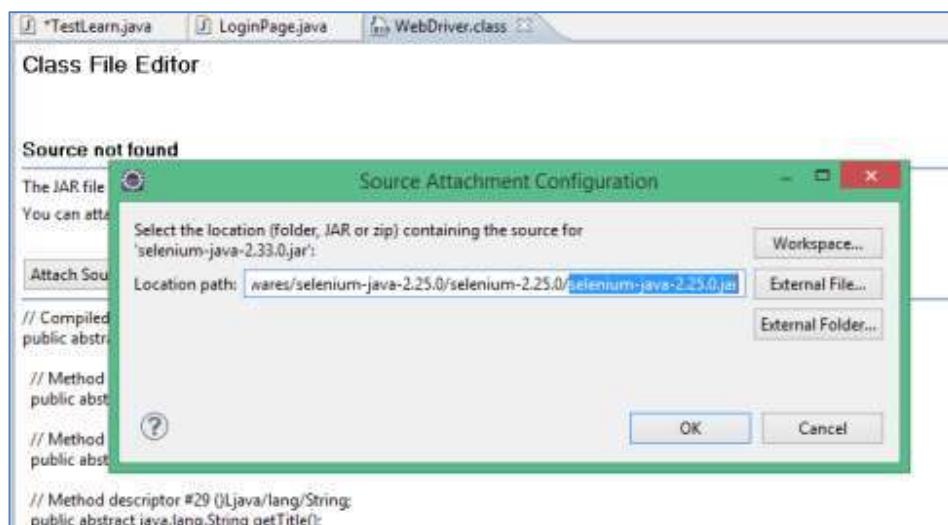
The two options will be shown in the pop up. Select the “Open Declaration” option.



Step 2: Eclipse opens Class File Editor. There if no source code is attached (like in this case), will get one “Attach Source” button. Click on this button.



Step 3: One file browser pop-up window will be appear, where either source file or source folder or Work space can be select. In this case select the source file.



Step 4: Click on 'Ok' button. Now the source implementation for FirefoxDriver class can be seen.

Summary

- Eclipse is an integrated development environment (IDE) for Java and other programming languages like C, C++, PHP, and Ruby
- Eclipse can be downloaded from <http://www.eclipse.org/downloads/>
- Selenium can be downloaded from <http://www.seleniumhq.org/download/>
- Eclipse works with the concepts of workspaces—folders and spaces where projects can be created

DUCAT®
www.ducatindia.com

Chapter 4- WebDriver

WebDriver Interface

Introduction

Interface is like a blueprint of Class. It contains variables and body less methods i.e. Abstract methods. An object to interface can't be created but classes can be created where the abstract methods of the interface can be implemented. The classes which implement the abstract methods of interface are known as implementation classes.

The primary new feature in Selenium 2.0 is the integration of the *WebDriver API*. *WebDriver* is designed to provide a simpler, more concise programming interface in addition to addressing some limitations in the Selenium-RC API. Selenium-WebDriver was developed to better support dynamic web pages where elements of a page may change without the page itself being reloaded. *WebDriver*'s goal is to supply a well-designed object-oriented API that provides improved support for modern advanced web-app testing problems. Selenium-WebDriver makes direct calls to the browser using each browser's native support for automation. How these direct calls are made, and the features they support depends on the browser you are using. Information on each 'browser driver' is provided later in this chapter.

For those familiar with Selenium-RC, this is quite different from what you are used to. Selenium-RC worked the same way for each supported browser. It 'injected' javascript functions into the browser when the browser was loaded and then used its javascript to drive the AUT within the browser. *WebDriver* does not use this technique. Again, it drives the browser directly using the browser's built in support for automation.

WebDriver Architecture

Selenium WebDriver's architecture is designed in a way that it talks to the browser in its native language. In order to write WebDriver code to work with Firefox, code for Firefox need to be written. If an Interface with abstract methods can be written then a message can be send to all the browser companies i.e. the third party companies to provide their implementation classes for an Interface. To implement the abstract methods of WebDriver interface in their way, there are separate class files for *FirefoxDriver*,*ChromeDriver*.

Name
internal
ExtensionConnection.java
FirefoxBinary.java
FirefoxDriver.java
FirefoxProfile.java
GeckoDriverService.java
MarionetteDriver.java
NotConnectedException.java
Preferences.java
UnableToCreateProfileException.java
ChromeDriver.java
ChromeDriverCommand.java
ChromeDriverCommandExecutor.java
ChromeDriverService.java
ChromeOptions.java

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

Above will be implementing rules of interface WebDriver over the third party browser class files Firefox and Chrome. *FirefoxDriver()* and *ChromeDriver()* methods are defined in the class files *FirefoxDriver* and *ChromeDriver* class files respectively.

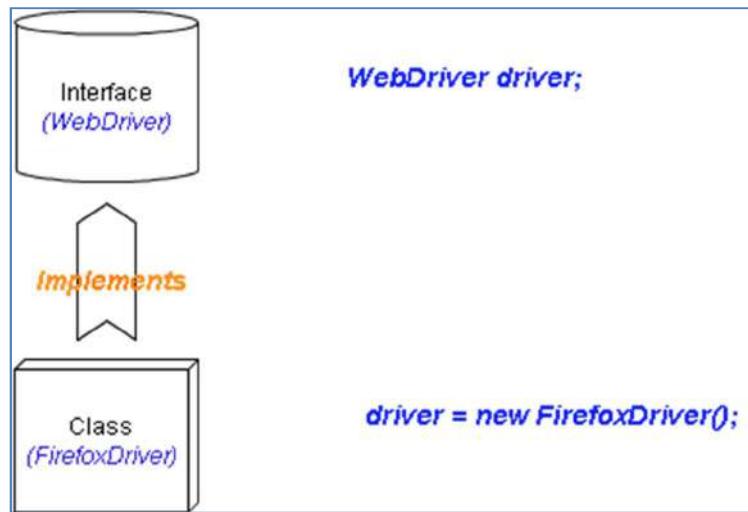
Example of the structure of a get method

```
Public Interface WebDriver()
{
    int iNum = 10;
    public void get(String URL);
}
```

Example of a class which would implement the interface

```
Public class FirefoxDriver implements WebDriver()
{
    public void get(String sURL)
    {
        System.out.println("URL="+sURL);
    }
}
```

In above example a class is *FireFoxDriver* and Interface is *WebDriver*.

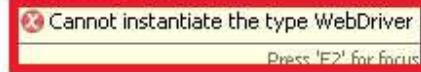


Interface instance

A reference variable can be created of an interface but any interface can't be instantiate since it is just a contract to be implemented in a Class. For example

`WebDriver driver = New WebDriver...` Not allowed,

```
public static void main(String[] args) {
    WebDriver selenium = new WebDriver();
```

 Cannot instantiate the type WebDriver
Press 'F2' for focus

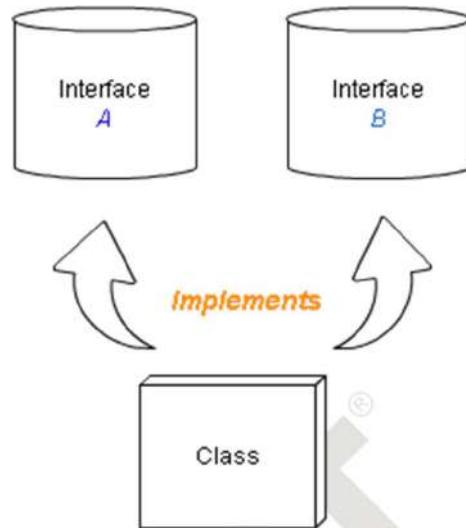
The correct implementation is

```
WebDriver driver = New FireFoxDriver();
driver.get(testUrl);
```

Important points

- Class that implements an interface must implement all the methods declared in the interface. For example, *FirefoxDriver* class should implement all the methods declared inside a *WebDriver* interface, same is the case with *ChromeDriver* or *IEDriver* classes.
- While implementing methods, you must follow the exact same signature (name + parameters) as declared in the interface
- Object of an Interface can't be instantiate/create.
- All the variable inside an interface are by default Final
- Class cannot Extend Interface only Implements it
- Interface can Extend another Interface but then the class which implements the interface need to implemented the methods of both interface

- Class can implement multiple Interface(Remember class cannot extend multiple classes, multiple inheritance in class is not possible)



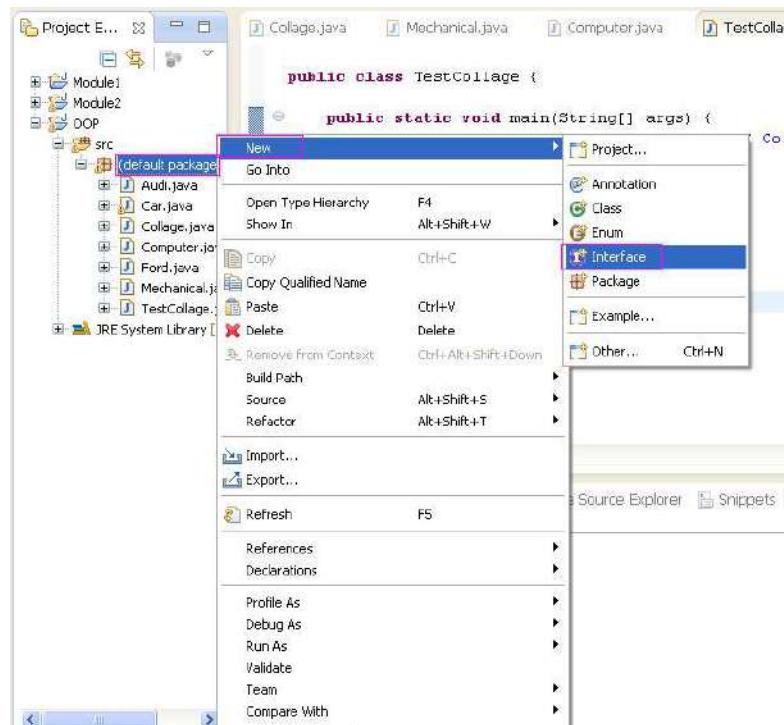
- Interface cannot hold constructor.
- Interface cannot hold instance fields/variables.
- By default all the methods of Interface are public so no need to provide access modifiers.
- An interface can have another interface i.e. known as nested interface.

Abstract Class v/s Interface

Abstract class is similar to interface and contain a mix of methods declared with or without an implementation. But in abstract class, we can declare fields that are not static and final, and define public, protected, and private concrete methods. With interfaces, all fields are automatically public, static, and final, and all methods that are declared or defined (as default methods) are public. Any number of interfaces can be implemented unlike abstract class which can extend only one class.

How to Add Interface In the Project

To add Interface In the project, Right click on package and go to -> New -> Interface. It will open New java Interface popup. Give any name to Interface (example: College) and click on save button. It will add *College.java* Interface file under the package.



Now let us see at simple example of Interface.

1. Create One Interface file with name = College.java as shown in below example.
2. Create 3 class file with name = Computer.java, Mechanical.java and TestCollege.java as shown in below example.

College.java

```
public interface College { //Interface file
    //Initialized static variable.
    //No need to write static because It Is by default static.
    String Collegename = "XYZ";

    //Created non static methods without body.
    void StudentDetails();
    void StudentResult();
}
```

Computer.java

```
//Class file Implemented with Interface file using implements
// keyword.

public class Computer implements College {
```

```

//@Override annotation describes that methods are overridden on
//interface method.

//Methods name, return type are same as methods of an Interface.

@Override
public void StudentDetails() {
    System.out.println("Computer Dept. Student Detail Part");
}

@Override
public void StudentResult() {
    System.out.println("Computer Dept. Student Result Part");
}

```

Mechanical.java

```

//Class file Implemented with Interface file using implements
//keyword.

public class Mechanical implements College{

    @Override
    public void StudentDetails() {
        System.out.println("Mechanical Dept. Student Detail Part");
    }

    @Override
    public void StudentResult() {
        System.out.println("Mechanical Dept. Student Result Part");
    }
}

```

TestCollege.java

```

public class TestCollege {//Class file. No need to implement
Interface.

public static void main(String[] args) {

```

```

        //Can access Interface variable directly using
Interface name.

    System.out.println(College.Collegename+" Collage student
details."); 

    //Created Computer class object with reference of interface
//College.

    College compdept = new Computer();
    //Methods will be called from Computer class.

    compdept.StudentDetails();
    compdept.StudentResult();

    //Created Mechanical class object with reference of interface
//College.

    College mecdept = new Mechanical();
    //Methods will be called from Mechanical class.

    mecdept.StudentDetails();
    mecdept.StudentResult();
}

}

```

3. Now if you will run TestCollege.java file, Output will looks like below.

```

XYZ Collage student details.

Computer Dept. Student Detail Part
Computer Dept. Student Result Part
Mechanical Dept. Student Detail Part
Mechanical Dept. Student Result Part

```

WebElement Interface

WebElement represents an HTML element. HTML documents are made up by HTML elements. HTML elements are written with a start tag, with an end tag, with the content in between. For example

```
<tagname> content </tagname>
```

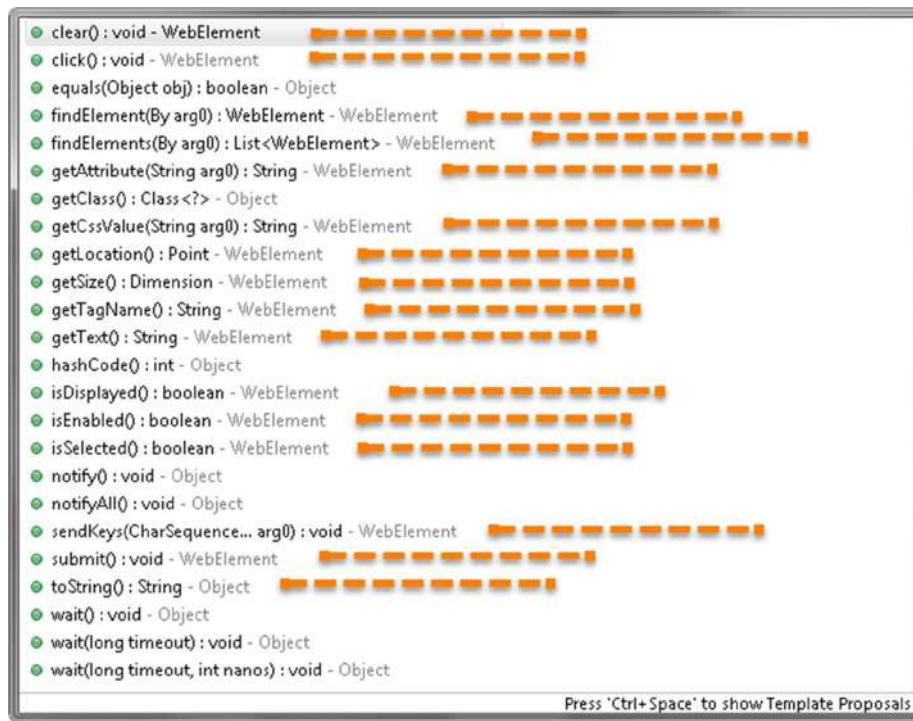
The HTML element is everything from the start tag to the end tag. For example:

```
<p> My first HTML paragraph. </p>
```

HTML elements can be nested (elements can contain elements). All HTML documents consist of nested HTML elements.

```
<html>
    <body>
        <h1> My First Heading </h1>
        <p>My first paragraph. </p>
    </body>
</html>
```

List of WebElement Commands



The methods followed by *Object* keyword are the generic methods gets from Object Class in Java. The *findElement* command of WebDriver always returns *WebElement*. To get the *WebElement* write object below statement

```
WebElement element = driver.findElement(By.id("UserName"));
```

And now *on type element dot*, Eclipse's intellisence will populate the complete list of actions just like the above image. *WebElement* can be of any type, such as ***Text, Link, Radio Button, Drop Down, WebTable*** or any *HTML element*. But all the actions will always populate against the any element irrespective of whether the action is valid on the *WebElement* or not. For example ***clear()* command**, even for a link element still there is the option to choose *clear()* command on it, which on choose may result in some error or may not does anything.

Launching Firefox Browser

The Selenium WebDriver calls the native methods of the different browsers to automate them. In Selenium there are different WebDrivers for different browsers such as *FirefoxDriver* for Firefox browser, *ChromeDriver* for Google Chrome, *InternetExplorerDriver* for Internet Explorer.

Firefox is one of the most traditional widely used browsers in automation. Thus Selenium WebDriver do not require any additional utility to be set before launching the browser. The Selenium package automatically references towards the default location of the firefox.exe, thus the user need not to set any other property. FirefoxDriver comes as a part of Selenium package and is present in xpi format. Now in order to launch Firefox with Selenium 3, *GeckoDriver* will be needed.

GeckoDriver

Gecko is the proprietary web browser engine developed by Mozilla. *GeckoDriver* is a proxy which is used to run Selenium 3 tests in Firefox. *GeckoDriver* is compatible with all versions of Mozilla Firefox, unlike Selenium 2. This essentially means that Mozilla will have to introduce a new version (or update) of *GeckoDriver* with every new release of Firefox.

The command to launch Firefox browser is:

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

This is the java implementation of launching a browser in Selenium. Here, 'WebDriver' is an interface and are creating a reference variable 'driver' of type WebDriver, instantiated using 'FireFoxDriver' class.

For those who are not very proficient in Java, an interface is like a contract that classes implementing it must follow. An interface contains a set of variables and methods without any body (no implementation, only method name and signature). As objects cannot be instantiate from interfaces. Hence, the below line of code is incorrect and throws compile time error saying "Cannot instantiate the type WebDriver".

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

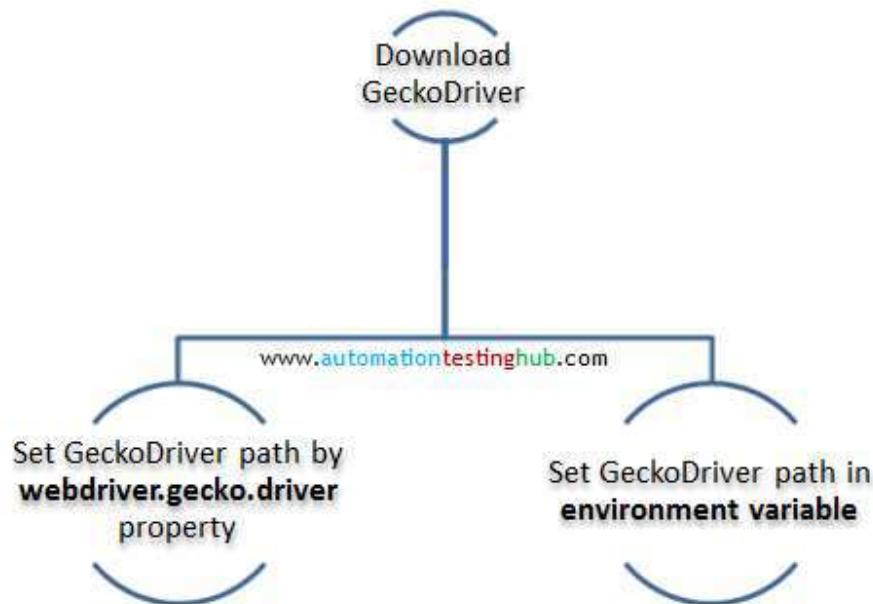
For instantiation of driver object, the classes like *FirefoxDriver* or *ChromeDriver* which have implemented the WebDriver interface are use. In other words, these driver classes have followed the contract of WebDriver by implementing all the methods of the WebDriver interface.

A reference variable of type *FirefoxDriver* can be create like this-

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

But having a WebDriver reference object helps in multi-browser testing as the same driver object can be used to assign to any of the desired browser specific driver. To launch Firefox with Selenium *Geckodriver*, first download *Geckodriver* and then set its path. This can be done in two ways as depicted in the below diagram.

Process to use GeckoDriver



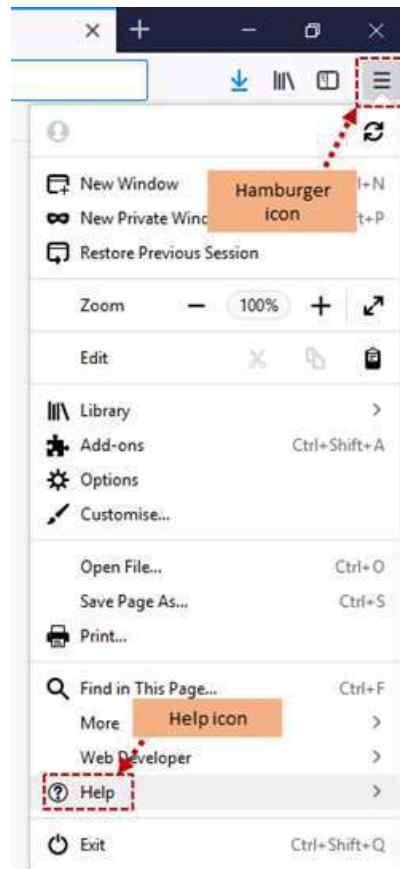
Install Geckodriver

The steps to install and configure *Geckodriver* are:

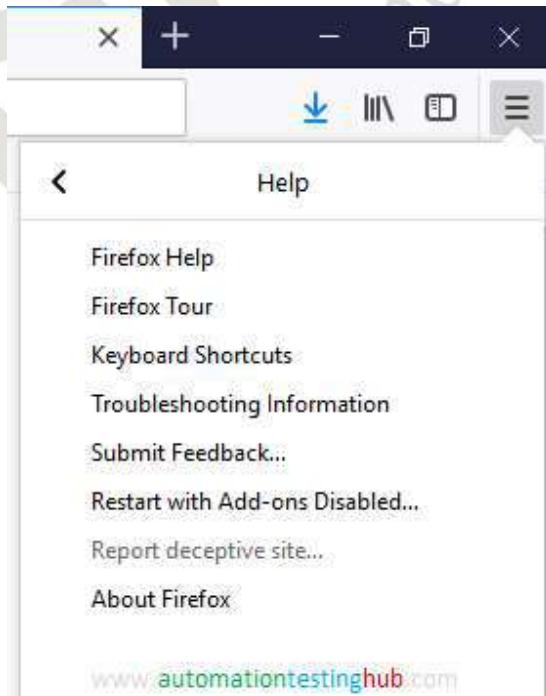
1. Check if Firefox is 32-bit or 64-bit

There are two versions of *Geckodriver* for Windows: 32-bit and 64-bit. Based on whether the Firefox is 32-bit or 64-bit, download the corresponding *Geckodriver* exe

2. Open Firefox on the machine. Click on Hamburger icon from the right corner to open the menu as shown below



3. From this menu, click on Help icon (Help icon is marked in red box in the above image)
4. Once click on Help icon, the Help Menu would be displayed



5. Click on About Firefox from the Help menu. About Mozilla Firefox popup would be displayed



6. Note down whether Firefox is 32 or 64 bit. Here, Firefox is 64-bit as shown in the above image. Now close this popup and close Firefox as well.
7. Download the latest version of Selenium *Geckodriver*
8. Follow the steps given below to download *Geckodriver*
9. Open this Github page – <https://github.com/mozilla/geckodriver/releases>
10. Download the latest release (windows version) based on whether your Firefox is 32-bit or 64-bit. We are downloading geckodriver-v0.20.1-win64.zip, as we have 64-bit Firefox



11. Once the zip file is downloaded, unzip it to retrieve the driver – geckodriver.exe

There are 2 methods using which this driver can be configured in the project.

- **Launch Firefox Method 1 :** webdriver.gecko.driver system property
Follow the steps given below to use this method

1. Copy the entire path of unzipped geckodriver.exe. Let us assume that the location is – D:\\Firefox\\geckodriver.exe. System.setProperty will be need to add with the driver location to the code.

The code to launch Firefox browser would look like this. In the folder paths in the below code, double backslash (\\) is used. This is because Java treats single back slash () as an escape character. So use double back slash, where it add some folder path.

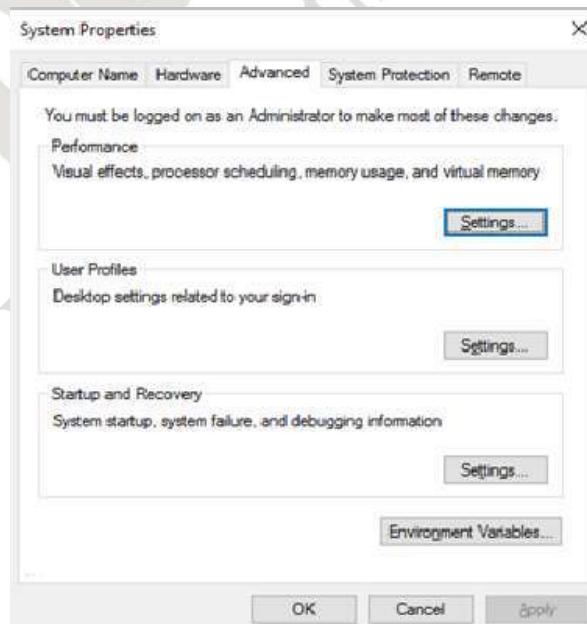
```
public class FirefoxTest {
    public static void main(String[] args) {

        System.setProperty("webdriver.gecko.driver", "D:\\Firefox\\
\\geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        driver.get("http://www.google.com");
    }
}
```

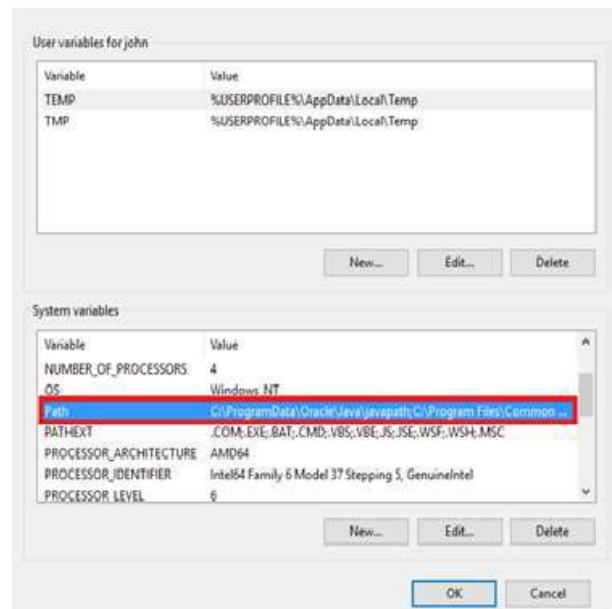
2. Run the code to verify that everything is working fine. The google.com gets opened in new Firefox window

- **Launch Firefox Method 2 : Set property in Environment Variables**

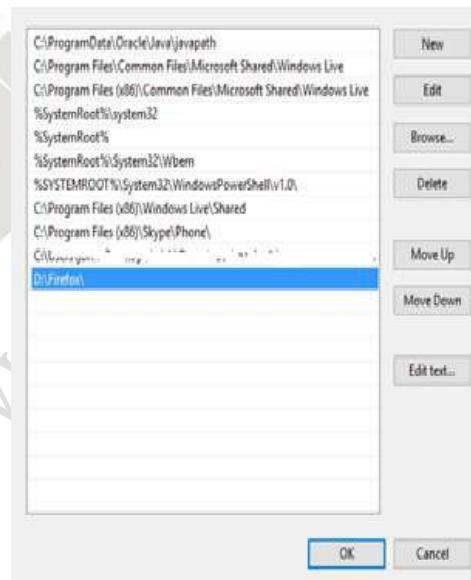
1. Copy the entire folder location where geckodriver.exe is saved. If the entire path is D:\\Firefox\\geckodriver.exe, then the folder location would be D:\\Firefox\\
2. Open advanced tab in System Properties window as shown in below image.



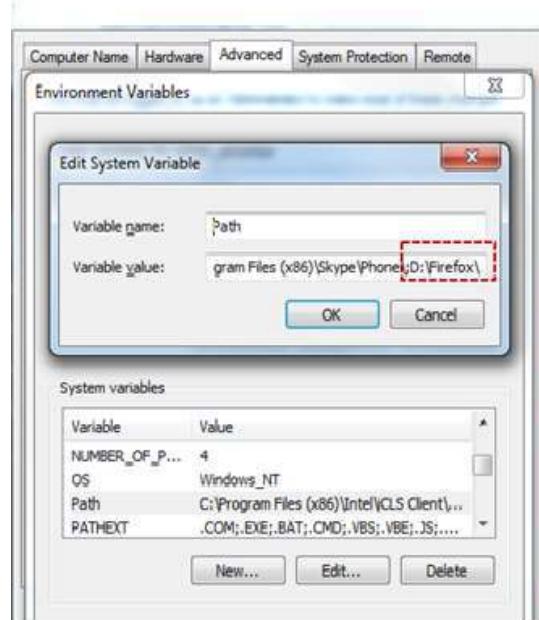
3. Open Environment Variables window.



- In System variables section, select the Path variable (highlighted in the above image) and click on Edit button. Then add the location of *Geckodriver* (D:\Firefox\), to path variable (below image shows UI for Windows 10)



- If Windows 7 is using, then move to the end of the Variable value field, then add a semi-colon (;) and then add the folder location as shown below (Semicolon acts as a separator between multiple values in the field)



6. Click on Ok button to close the windows. Once the path is set, the System property is not need to set every time in the test script. The test script would simply look like this

```
public class FirefoxTest {  
  
    public static void main(String[] args) {  
        WebDriver driver = new FirefoxDriver();  
        driver.get("http://www.google.com");  
    }  
}
```

7. Run the code to check that it works fine.

Summary

- Selenium WebDriver's architecture is designed in a way that it talks to browser in its native language
- WebElement represents an HTML element. HTML documents are made up by HTML elements
- The Methods followed by Object keyword are the generic methods gets from Object Class in Java
- The Selenium WebDriver calls the native methods of the different browsers to automate them

DUCAT®
www.ducatindia.com

Chapter 5 – Browser and Navigation Commands

Introduction

The Browser commands are generally the ones which we intuitively feel that a browser should control. Examples are “type url and hit enter”, “getTitle”, “back”, “forward” and so on. Video of the execution and walk through code is explained in the video at the bottom of this page.

Navigation commands are some which enables the user to navigate to some webpages and also to work with history like back, forth, refresh, to. We can access the these method by accessing the *navigate()* method

Selenium Navigation Commands are used to open a web page URL and navigation can be done between different web pages by clicking any element such as back, forward or refresh. Using the navigating interface back and forward can be move in browser history.

How to Open a URL

There are few methods that can be used to open an URL. The various methods are:

- **Navigate.To(URL)**

Syntax:

```
driver.navigate().to(URL);
```

This methods load a new web page in the current browser window. This is done using an HTTP GET operation, and the method will block until the load is complete. The parameters URL is a fully qualified URL.

Example:

```
public void navigationToURLExample()
{
    driver= new FirefoxDriver();
    driver.navigate().to("http://www.google.com");
}
```

- **Navigate.To(String)**

Syntax:

```
driver.navigate().to(String);
```

This methods load a new web page in the current browser window. It is an overloaded version of to (String) that makes it easy to pass in a URL.

Example:

```
public void navigationToStringExample()
{
    driver= new FirefoxDriver();
    String URL="http://www.facebook.com";
    driver.navigate().to(URL);
}
```

Verify Page Title

Verification of the current page title can be done by checking it with Get Title Command.

`String getTitle()`: This method returns the title string of the current page in the browser. It doesn't allow any parameter and its return type is a String. This method returns a String value. As the title is stored in the string and then it uses Assert selenium command to return true or false. It can also use If-statement to compare actual and expected web page title. It entirely depends on the discretion of the test engineer and requirement of the project.

Syntax:

```
driver.getTitle();
```

To compare a piece of the word(s) in the actual title use contains () method.

Syntax:

```
driver.getTitle().contains("Character sequence");
```

Here are the steps in the code to verify title of the page:

1. Use `getTitle()` and store the value in String

Example: `String actualTitle = driver.getTitle();`

2. Declare expected title in string

Example: `String expectedTitle ="Your expected title";`

3. Now compare expected and actual

Method-1 (Using If-Statement)

```
if(actualTitle.equalsIgnoreCase(expectedTitle))
System.out.println("Title Matched");
else
System.out.println("Title didn't match");
```

```

    package InviulTest;

    import java.util.concurrent.TimeUnit;
    import org.junit.Assert;
    import org.openqa.selenium.WebDriver;
    import org.openqa.selenium.chrome.ChromeDriver;
    import org.openqa.selenium.firefox.FirefoxDriver;
    public class Test {
        public static void main(String[] args) {
            System.setProperty("webdriver.chrome.driver",
"F:\\chromedriver.exe");
            WebDriver driver = new ChromeDriver();
            driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
            driver.navigate().to("http://www.inviul.com");
            String actualTitle = driver.getTitle();
            driver.manage().window().maximize();
            String expectedTitle = "Avinash Mishra Blogger: Learn Selenium
WebDriver, Automation Framework, QA, SEO & Digital Marketing | Inviul Blog | Inviul";
            if(actualTitle.equalsIgnoreCase(expectedTitle))
                System.out.println("Title Matched");
            else
                System.out.println("Title didn't match");
            driver.close();
            driver.quit();
        }
    }
}

```

Method-2 (Using Assert Command)

```

Assert.assertEquals(actualTitle, expectedTitle, "Title matched");
    package InviulTest;

    import java.util.concurrent.TimeUnit;
    import org.junit.Assert;
    import org.openqa.selenium.WebDriver;
    import org.openqa.selenium.chrome.ChromeDriver;
    import org.openqa.selenium.firefox.FirefoxDriver;

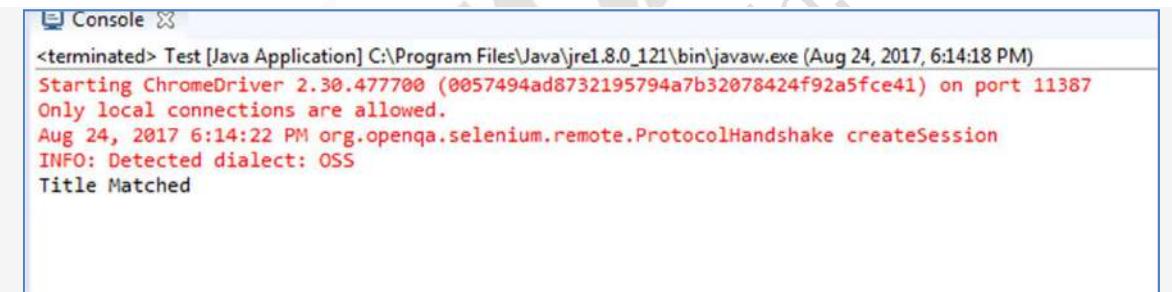
```

```

public class Test {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver",
        "F:\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.navigate().to("http://www.inviul.com");
        String actualTitle = driver.getTitle();
        driver.manage().window().maximize();
        String expectedTitle = " Learn Selenium WebDriver, Automation
Framework";
        Assert.assertEquals("Condition true", actualTitle, expectedTitle);
        driver.close();
        driver.quit();
    }
}

```

Output



```

Console ×
<terminated> Test [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (Aug 24, 2017, 6:14:18 PM)
Starting ChromeDriver 2.30.477700 (0057494ad8732195794a7b32078424f92a5fce41) on port 11387
Only local connections are allowed.
Aug 24, 2017 6:14:22 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: OSS
Title Matched

```

Example:

```

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

public class GetTitle
{
    public static void main(String[] args) throws Exception
    {
        WebDriver driver=new FirefoxDriver();
        driver.get("https://chercher.tech");
        System.out.println("Page title is : "+driver.getTitle());
    }
}

```

```

    }
}

}

```

Output

Page title is Selenium Webdriver

Example to Open Google.com and verify whether the title is Google or not.

```

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class Refresh
{
    public static void main(String[] args) throws Exception
    {
        WebDriver driver=new FirefoxDriver();
        driver.get("http://google.com");
        if (driver.getTitle().equals("Google"))
        {
            System.out.println("Title is Google ");
        }
        else
        {
            System.out.println("Title is not Google");
        }
    }
}

```

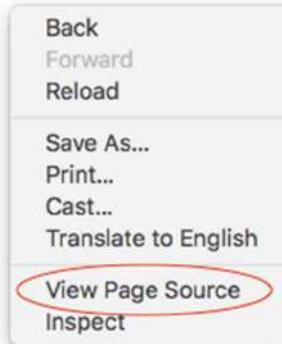
Likewise it can also be compare whether title has Google as a part of title or not

Strategy to Get the Page Source

String getPageSource(): This method retrieves the Source Code of a web page. It doesn't allow any parameter and its return value is a String. This method returns a String value
Syntax:

```
driver.getPageSource();
```

While doing test automation using Selenium WebDriver, one might have needs to scan through the HTML source of a webpage. Here it is shown how to get the HTML page source using Selenium WebDriver. One of the most common needs of using page source is to find out any hidden elements or some code snippet of javascript is present as part of the source code.



Lets first consider the below HTML source code from a page named "selenium-pagesource.html". Note that the page has not only HTML code, but javascript functions as well.

```
<!DOCTYPE html>
<html>
<head>
    <script>
        function buttonClicked()
        {
            alert("Hello")
        }
    </script>
</head>
<body>
    <div>
        This is an example on how to get html source code using
        webdiver.<br>
        <button value="Click Me" onclick="buttonClicked();">Click
        Me</button>
    </div>
</body>
</html>
```

Now let's see how we can get the entire source code of the above page using Selenium WebDriver and Java.

Selenium – Get Page Source Code

```

//Initialize the webdriver.
WebDriver driver = new SafariDriver();
//Invoke the web page.
driver.get("http://localhost:8080/selenium-pagessource.html");
//Get the page source and print the same to the console.
System.out.println(driver.getPageSource());
driver.quit();

```

The above example code will invoke the page and get the page source using `getPageSource()` function and print it to the console. `getPageSource` function returns a String which has the entire source code including any client side scripts. This will commonly be used to extract any text or content which can't directly be accessed using Selenium WebDriver find element methods.

Difference between Close and Quit

Close(): This method kills the current window which the WebDriver has the control over. It doesn't allow any parameter and its return type is void. It'll terminate the browser if it's the only window currently open

Syntax:

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

Example:

```

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

public class WebDriverCommands_2 {
    public static void main(String[] args) {
        WebDriver driver = new FirefoxDriver();
        driver.get("http://demoqa.com/frames-and-windows/");
        driver.findElement(By.xpath("//*[@id='tabs-1']/div/p/a")).click();
        driver.close();
    }
}

```

Quit (): This method closes all the associated windows opened by the WebDriver. It doesn't allow any parameter and its return type is void. It'll automatically close all of the opened windows and terminate the browser.

Syntax:

```
driver.quit();
```

Example

On running the code below, the all windows will be closed i.e. two pop-ups will automatically be closed as well.

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

public class PG3 {
    public static void main(String[] args) {
        System.setProperty("webdriver.firefox.marionette", "C:\\\\geckodriver.exe");
        WebDriver driver = new FirefoxDriver();
        driver.get("http://www.popuptest.com/popuptest2.html");
        driver.quit(); // using QUIT all windows will close
    }
}
```

Ways to Navigate Back and Forward

The Navigation commands are very useful and allow to control the navigation flows in the web browser.

Forward(): This method simulates the browser's forward button action. It doesn't allow any parameter and its return type is a void. It moves to the forward by a single page into the browser's history data.

Syntax:

```
driver.navigate().forward();
```

Example

```
public void navigationForwardExample()
{
    driver= new FirefoxDriver();
    String URL="http://www.twitter.com";
    driver.navigate().to(URL);
    driver.findElement(By.linkText("Forgot your
password?")).click();
    driver.navigate().back();
    Thread.sleep(1000);
    driver.navigate().forward();
}
```

- In the above example, to work with navigate forward method, there is need to travel to at least one or multiple pages in the browser history.
- The first page which has opened is twitter.com page and then traveled to forget password page. Now the browser has two pages in the history. If navigate.back() will use, it will redirect the user to twitter.com page whereas on using navigate.forward(), it will redirect the user to forgot password page.

Back():This method simulates the browser's back button action. It doesn't allow any parameter and its return type is a void. It moves you back by a single page into the browser's history data.

Syntax:

```
driver.navigate().back();
```

Example:

```
public void navigationBackExample()
{
    driver= new FirefoxDriver();
    String URL="http://www.twitter.com";
    driver.navigate().to(URL);
    driver.findElement(By.linkText("Forgot your password?")).click();
    driver.navigate().back();
}
```

- In the above example, to work with navigate back method, there is need to travel to at least one or multiple pages in the browser history.

- The first page which has opened is twitter.com page and then traveled to forget password page. Now the browser has two pages in the history. If `navigate.back()` will use, it will redirect the user to twitter.com page.

How to Refresh a Page

A Web page can be refresh with the help of the Refresh Command.

refresh(): This method simulates the browser's refresh button action. It doesn't allow any parameter and its return type is a void. It triggers the same action as does the F5 shortcut in the browser.

There are many ways to refresh the page .Here are some of the ways:

- Using `refresh()` method: Selenium webdriver has a method called `refresh()`. This is widely used command. It refreshes the current page after executing.

Syntax

```
driver.navigate().refresh();
```

- Using `sendKeys()` method: `sendKeys()` method is used like we do page refresh pressing F5 key through our keyboard. This manual task is executed by code using the `sendKeys()` command over an element.

Syntax:

```
driver.findElement(By.id(locator)).sendKeys(F5 key);
```

Example

```
driver.findElement(By.id("gbqfq")).sendKeys(Keys.F5);
```

- Using `navigate().to()` method: Actually browsing the same URL using `navigate().to()` function. It call `getCurrentUrl()` function to get the current URL of the page.

Syntax:

```
driver.navigate().to(driver.getCurrentUrl());
```

- Using `get()` method: If we know the URL then we can load the same URL again to reload the page.

Syntax:

```
driver.get("https://www.google.com.bd/");
```

- Using `sendKeys()` method with ASCII code: ASCII code can be used as argument on `sendKeys()` method that is equivalent to F5 key command of the keyboard.

Example:

```
driver.findElement(By.id("gbqfq")).sendKeys("\uE035");
```

Here,

`\uE035` is the ASCII code of F5 key.

- Using `executeScript()` method: Using this command it can execute any JavaScript for the need. If it execute `location.reload()` JavaScript function then current page will be reloaded which meets our purpose.

Syntax:

```
driver.executeScript("location.reload()");
```

Example:

```
public void testRefreshPage() {
    driver= new FirefoxDriver();
    driver.navigate().to("http://google.com");
    Actions actions = new Actions(driver);
    actions.keyDown(Keys.CONTROL).sendKeys(Keys.F5).perform();
}
```

In the above example, web page is refreshed after entering the values using `sendKeys()`. There are few scenarios where it is required to check if the data is getting cleared once when the page refreshes. And for few websites, it will display an alert when the user tries to refresh the page without saving the form which user has entered data.

Another Way of Navigating to a Specific Page

The Get Command is the way to load a Specific page.

get(String arg0) : This method **loads** a new web page in the current browser window. But it doesn't maintain the browser History and cookies so forward and backward button can't be used and on click page will not get scheduled. This method accepts String as a parameter and returns nothing.

Syntax:

```
driver.get(appUrl);
```

Where *appUrl* is the website address to load. It is best to use a fully qualified URL.

Example:

```
driver.get("http://www.google.com");  
//Or can be written as  
String URL = "http://www.google.com";  
driver.get(URL);
```

We can get url of page by using *getCurrentUrl()*, it returns Url as string

Example:

```
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.firefox.FirefoxDriver;  
  
public class GetUrl  
{  
    public static void main(String[] args) throws Exception  
    {  
        WebDriver driver=new FirefoxDriver();  
        driver.get("https://chercher.tech");  
        // below line gets the url of a website  
        driver.getCurrentUrl();  
    }  
}
```

Summary

- Selenium Navigation Commands are used to open a web page URL
- Verification of the current page title can be done by checking it with Get Title Command
- The Navigation commands are very useful and allow to control the navigation flows in the web browser
- The Browser commands are generally the ones which we intuitively feel that a browser should control
- Navigation commands are some which enables the user to navigate to webpages
- A Web page can be refresh with the help of the Refresh Command

Chapter 6 – Locators

What are Locators?

Locator is a command to identify the GUI elements for operation by Selenium IDE. The GUI elements can be text box, buttons or checkboxes. Identification of correct GUI elements is a prerequisite to creating an automation script. In other words Locators provide a way to access the HTML elements from a web page. They are the basic building blocks of a web page. A web developer must use a proper and consistent locator scheme for a website. Also, a test engineer must choose the correct locator strategy to automate the online workflows. Selenium names different types of locators to find the elements on a web page. Webdriver references the Web elements by using the `findElement(By.<locator()>)` method. The `findElement` method uses a locator/query object known as <`By`>. There are various kinds of "By" strategies which can utilize depending on the requirements as below:

- a) By ID.

Syntax

```
driver.findElement(By.id(<element ID>))
```

According to this strategy, the `By` method will return the first element matching the `id` attribute value. If it doesn't find any matching element then, it'll raise a `NoSuchElementException`. Using element Id is the most preferred way to locate an element, as usually Ids have unique values.

Example:

```
<input id="TechBeamers">
// Java example code to find the input element by id.
WebElement user = driver.findElement(By.id("TechBeamers"));
```

- b) By Name.

Syntax

```
driver.findElement(By.name(<element-name>))
```

`By.Name()` is another useful way to locate an element. This strategy states that the `<By.Name()>` method will return the first element matching the `name` attribute. If it's not able to locate it then, it'll throw a `NoSuchElementException`.

Example:

```
<input name="TechBeamers">
// Java example code to find the input element by name.
WebElement user = driver.findElement(By.name("TechBeamers"));
```

c) By Class Name.

Syntax:

```
driver.findElement(By.className(<element-class>))
```

This method gives the element which matches the values specified in the “class” attribute. If the element has more than one class, then this method will match against each one of them.

Example:

```
<input class="TechBeamers">
// Java example code to find the input element by className.
WebElement user = driver.findElement(By.className("TechBeamers"));
```

d) By TagName.

Syntax:

```
driver.findElement(By.tagName(<html tag name>))
```

This method is used to find the elements matching the specified tag name. This method can be called to extract the content within a tag or wish to perform any action on the tag element.

Example:

```
<form action="viewuser.asp" method="get" id="userform">
    Name: <input type="text" name="name"><br>
    Age: <input type="text" name="age"><br>
</form>

<button type="submit" form="userform" value="Submit">Submit</button>
WebElement form = driver.findElement(By.tagName("button"));
// Any action can be performed on the form button.
form.submit();
```

e) By CssSelector.

Syntax:

```
driver.findElement(By.cssSelector(<css-selector>))
```

With this method, a CSS selector can be used to locate the element. The CSS is cascading style sheet; it sets the styles for a web page and its elements.

Example:

```

<input class="email" id="email" type="text"
placeholder="your@email.com">

<input class="btn btn-small" type="submit" value="Subscribe to our
blog">

// Java code example for using cssSelector.

WebElement emailText =
driver.findElement(By.cssSelector("input#email"));

//Or

WebElement emailText =
driver.findElement(By.cssSelector("input.email"));

WebElement subscribe =
driver.findElement(By.cssSelector("input[type='submit'][value='Subsc
ribe to our blog']]"));

```

f) By XPath.

Syntax:

```
driver.findElement(By.xpath(<xpath>))
```

This method is used to locate an element using the XPath query. XPath is a way to traverse through the document object model, gives you the ability to select specific elements, attributes, or a section of an XML document.

Example:

```

// Java code example for XPath.

// Absolute path.

WebElement item =
driver.findElement(By.xpath("html/head/body/table/tr/td"));

// Relative path.

WebElement item = driver.findElement(By.xpath("//input"));

// Finding elements using indexes.

WebElement item = driver.findElement(By.xpath("//input[5]"));

// Finding elements using attributes values.

WebElement item =
driver.findElement(By.xpath("img[@alt='banner']"));

// XPath starts-with() example.

// => input[starts-with(@id, 'User')]

```

```
// XPath ends-with() example.  
// => input[ends-with(@id, '_name')]  
// XPath contains() example.  
// => input[contains(@id, 'email')]
```

HTML Basics

HTML (**Hypertext Markup Language**) is a *markup language* that defines the structure of content. It is the language in which most websites are written. It is used to create pages and make them functional. The code used to make them visually appealing is known as CSS.

HTML consists of a series of elements, which can be used to enclose, or wrap, different parts of the content to make it appear a certain way, or act a certain way. The enclosing tags can make a word or image hyperlink to somewhere else, can italicize words, and can make font bigger or smaller, and so on.

HTML Language Tags and Attributes

A **Markup Language** can be define as a way that computers speak to each other to control how text is processed and presented. To do this HTML uses two things: **tags and attributes**.

HTML Tags

Tags are used to mark up the start of an HTML element and they are usually enclosed in angle brackets. HTML elements are written with a start tag, with an end tag, with the content in between: <tagname> content </tagname>. The HTML element is everything from the start tag to the end tag: <p> My first HTML paragraph. </p>

HTML elements can be nested (elements can contain elements). All HTML documents consist of nested HTML elements. An example of a tag is: <h1>. Most tags must be opened <h1> and closed </h1> in order to function

HTML Attributes

Attributes contain additional pieces of information. Attributes take the form of an opening tag and additional info is placed inside. An example of an attribute is:

```

```

In this instance, the image source (src) and the alt text (alt) are attributes of the tag. The other attributes are:

Difference Between Absolute and Complete Xpath

XPath is a technique for walking through the DOM structure of the web page. XPath locators are robust and reliable. It is one method which guarantees to locate any element on the page using the XPath expression. The XPaths can be classified in the following two groups.

a) Absolute XPath.

It starts from the root element within the web page or part of the page and goes to identify the target element.

Example:

```
HTML/head/body/table/tr/td
```

To use locators like the XPath is easy as you give the direct element path. But the XPath would break when the element structure changes.

b) Complete XPath or Relative XPath

The complete XPath are easy to manage as they are short and concise. It is also better than the previous XPath style as it may survive the changes in the Page HTML to a certain degree. Though, building a complete XPath is time-consuming and quite difficult as you need to check all the nodes to form the path.

Example:

```
//table/tr/td
```

Finding Your First Element

Everything that is present on the web page is an element. An element is equivalent to a tag in HTML. One of the most important skills of a test automation engineer working with Selenium WebDriver is to be able to use appropriate methods to locate elements on a page. WebDriver users the SearchContext interface to locate elements on any page. The SearchContext interface has two methods:

- List `findElements(By by)`: This method finds all the elements that match an instance of locator.
- WebElement `findElement(By by)`: This method finds the first element that matches or throws a NoSuchElementException if it is not found.

There are multiple ways to uniquely identify a web element within the web page such as ID, Name, Class Name, Link Text, Partial Link Text, Tag Name and XPATH. Find Element command is used to uniquely identify a (one) web element within the web page. Whereas, Find Elements command is used to uniquely identify the list of web elements within the web page.

FindElement

Find Element command takes in the `By` object as the parameter and returns an object of type `WebElement`. `By` object in turn can be used with various locator strategies such as ID, Name, Class Name, XPATH etc. The syntax of FindElement command in Selenium web driver.

```
WebElement elementName =
driver.findElement(By.LocatorStrategy("LocatorValue"));
```

A Locator Value is the unique value using which a web element can be identified. Locator Strategy can be any of the following values.

- ID
- Name
- Class Name
- Tag Name
- Link Text
- Partial Link Text
- XPATH

Example:

```
WebElement loginLink = driver.findElement(By.linkText("Login"));
```

FindElements

Find Elements command takes in *By* object as the parameter and returns a list of web elements. It returns an empty list if there are no elements found using the given locator strategy and locator value. The syntax of find elements command.

```
List<WebElement> elementName =
driver.findElements(By.LocatorStrategy("LocatorValue"));
```

Example:

```
List<WebElement> listOfElements =
driver.findElements(By.xpath("//div"));
```

Example:

```
package com.sample.stepdefinitions;

import java.util.List;

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

public class NameDemo {

    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver",
        "X://chromedriver.exe");

        WebDriver driver = new ChromeDriver();
        driver.get("http://www.ducatindia.com/test/ajax.html");
        List < WebElement >;
        elements = driver.findElements(By.name("name"));
    }
}
```

```

        System.out.println("Number of elements:" + elements.size());
        for (int i = 0; i < elements.size(); i++) {
            System.out.println("Radio button text:" +
elements.get(i).getAttribute("value"));
        }
    }
}

```

- Find Element command returns the web element that matches the first most element within the web page.
- Find Elements command returns a list of web elements that match the criteria.
- Find Element command throws NoSuchElementException exception if it does not find the element matching the criteria.
- Find Elements command returns an empty list if there are no elements matching the criteria

WebElement Commands

WebElement is the smallest possible HTML entity that helps to form a fully functional web page. In other words a WebElement represents an HTML element. In Webdriver terminology, it is a Java class which represents an object that holds the reference to an HTML element. For example, In the below HTML code, the whole **div** including **opening and closing tag** is forming an HTML element.

<div id='employee'>...</div>

Every method of the WebDriver either returns something or return void(means return nothing). The same way findElement command of WebDriver returns WebElement. In order to get the WebElement object the below statement is used

```
WebElement element = driver.findElement(By.id("UserName"));
```

WebElement can be of any type, such as a Text, Link, Radio Button, Drop Down, WebTable or any HTML element. Above command will list all the actions against the any element irrespective of whether the action is valid on the WebElement or not. The Selenium Webdriver action commands associated with the web elements are:

clear() : This method will clear the value of any text type element. It doesn't allow any parameter and its return type is also void. This method doesn't impact any HTML element other than the field of text type.

Syntax:

```
element.clear();
```

Example:

```
WebElement user = driver.findElement(By.id("User"));
user.clear();
```

sendKeys(CharSequence... keysToSend): This method types in the values passed in the argument into the associated web element object. It allows a CharSequence as a parameter, and its return type is void. This method supports elements like INPUT and TEXTAREA elements.

Syntax:

```
element.sendKeys("enter text");
```

Example:

```
WebElement user = driver.findElement(By.id("User"));
user.sendKeys ("TechBeamers");
```

click(): This method performs the click action at the web element. It doesn't accept any parameter, and its return type is void. It is one of the most common methods which interacts with the web elements like links, checkboxes, buttons, etc.

Syntax:

```
element.click();
```

Example:

```
WebElement link = driver.findElement(By.linkText ("TechBeamers"));
link.click();
```

boolean isDisplayed(): This method checks the visibility of a web element. It doesn't allow any parameters but its return type is a boolean value. This method will return true if the element is present and visible on the page. It'll throw a *NoSuchElementFound* exception if the element is not available. If it is available but kept as hidden, then the method will return false.

Syntax:

```
element.isDisplayed();
```

Example:

```
WebElement user = driver.findElement(By.id("User"));
boolean is_displayed = user.isDisplayed();
```

boolean isEnabled(): This method return true/false depending on the state of the element (Enabled or not). This method will usually return true for all items except those which are intentionally disabled.

Syntax:

```
element.isEnabled();
```

Example:

```
WebElement user = driver.findElement(By.id("User"));
boolean is_enabled = user.isEnabled();
```

boolean isSelected(): This method tests if a element is active or selected. It doesn't allow any parameter, but it does return a boolean status. This method is only applicable for input elements like Checkboxes, Radio Button, and Select Options. It'll return true when it finds the element is currently selected or checked.

Syntax:

```
element.isSelected();
```

Example:

```
WebElement lang = driver.findElement(By.id("Language"));
boolean is_selected = lang.isSelected();
```

void submit(): This method initiates the submission of an HTML form. It doesn't allow any parameter and its return type is void. If the method is leading the current page to change, then it would wait until the new page gets loaded.

Syntax:

```
element.submit();
```

Example:

```
webElement submit_form = driver.findElement(By.id("Submit"));
submit_form.submit();
```

String getText(): This method will give you the innerText value of the element. You must note that the innerText property is CSS aware and the getText() method would return a blank if the text is hidden. This method provides the innerText of the associated element. The output includes the sub-elements and leaves any leading or trailing whitespace aside.

Syntax:

```
element.getText();
```

Example:

```
WebElement link = driver.findElement(By.xpath("MyLink"));
String strlink = link.getText();
```

String getTagName(): This method provides the tag name of the associated element. It doesn't allow to pass any parameter and returns a String object. It'll return the tag name i.e. "input" of the element. e.g. <input name="Password"/>.

Syntax:

```
element.getTagName();
```

Example:

```
WebElement pass = driver.findElement(By.id("Password"));
String strTag = pass.getTagName();
```

String getCssValue(String propertyName): This method provides the value of the CSS property belonging to the given element. This method may return an unpredictable value in a cross-browser setup. Like, you would have set the “background-color” property to “green” but the method could return “#008000” as its value.

Syntax:

```
element.getCssValue("font-size");
```

Example:

```
webElement name = driver.findElement(By.id("Name"));
String strAlign = name.getCssValue("text-align");
```

getSize() – This method returns the height and width of the given element. It doesn’t allow to pass any parameter, but its return type is the Dimension object. This methods provides the size of the element on the web page.

Syntax:

```
element.getSize();
```

Example:

```
WebElement user = driver.findElement(By.id("User"));
Dimension dim = user.getSize();
System.out.println("Height # " + dim.height + "Width # "+ dim.width);
```

getLocation(): This method locate the location of the element on the page. It doesn’t allow to pass any parameter, but its return type is the Point object. This method provides the Point class object.

Syntax:

```
element.getLocation();
```

Example:

```
WebElement name = driver.findElement(By.id("Name"));
Point point = name.getLocation();
String strLine = System.getProperty("line.separator");
System.out.println("X coordinate# " + point.x + strLine + "Y coordinate# " +
point.y);
```

Summary

- Locator is a command to identify the GUI elements for operation by Selenium IDE
- HTML (Hypertext Markup Language) is a markup language that defines the structure of content
- HTML consists of a series of elements, which can be used to enclose, or wrap, different parts of the content to make it appear a certain way, or act a certain way
- HTML Tags are used to mark up the start of an HTML element
- XPath is a technique for walking through the DOM structure of the web page

DUCAT®
www.ducatindia.com

Chapter 7 – Element Identification

Element Inspector in Mozilla, Chrome and IE

Inspecting elements in the browser helps web developers and designers to understand the process which occurs during the http request and response to and fro from the server. The inspection of elements is also used by persons working in automation testing.

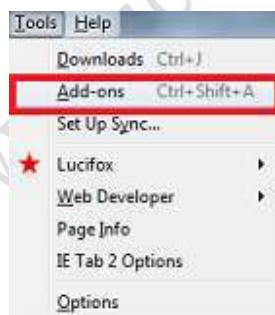
The information about the features of a web page can be found by inspecting the page or any particular element available in the page, such as

- Web page view feature. For example:
 - BG color
 - Font style
 - Font size
 - height of the page
 - width of the page
 - Alignment of any element in the page
 - Padding of the page
- Html information or url
- Network information
- Source of the web page
- Error which occurs during page load in the browser.

Inspection of element in Firefox browser

Inspection of element in firfox browser is done using a plugin called “Firebug”. The steps to install firebug and inspect element in the Firefox browser are:

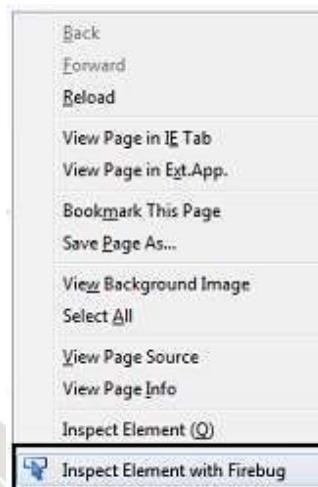
1. Download and install Firefox.
2. Once Firefox get installed successfully go to tools and select Add-ons



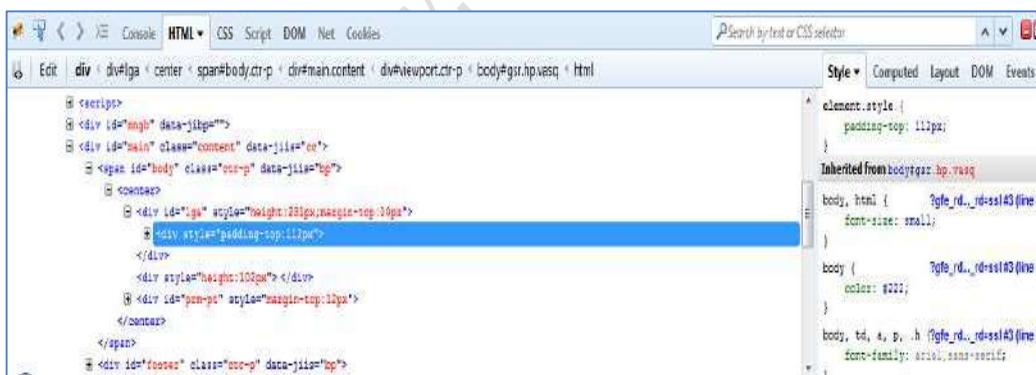
3. After selecting Add-ons search for “Firebug”



4. Now click on Install button will install the firebug in your Firefox browser.
5. Once the firebug installed successfully, any page can be load by right click on it.
6. After right click select the option "Inspect Element with Firebug"



7. Clicking on "Inspect Element with Firebug" will inspect all the elements available in the page or if you want to inspect any particular element, mouse over on that element and right click and select the same option will inspect the particular element.
8. Once inspected you can view all the information at the bottom of the page.

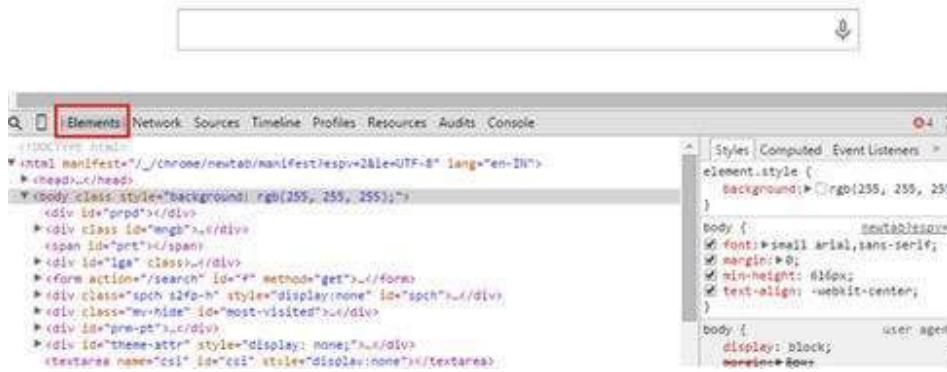


Inspection of element in Chrome browser:

Chrome browser comes with default "Inspect Element" Option. The steps to install firebug and inspect element in the chrome browser are:

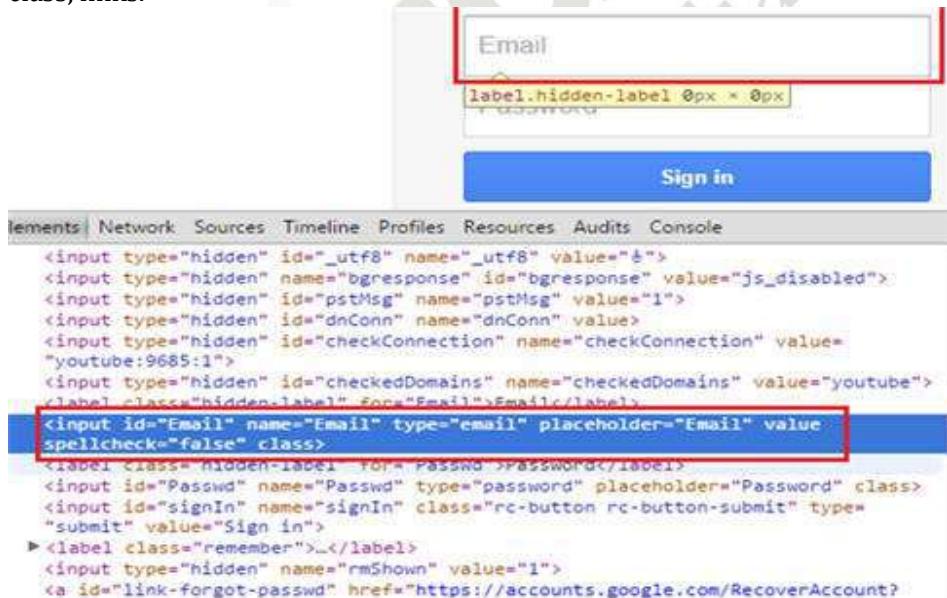
1. Launch the Google Chrome's Developer tool. Press F12 to launch the tool. You would be able to see like the below screen.

Google



Take a note that “Element” tab is highlighted in the above screenshot. Thus, element tab is the one which displays all the HTML properties belonging to the current web page. Navigate to the “Element” tab if it is not opened by default on the launch.
 You can also launch developer tool by right-clicking anywhere within the web page and by selecting “Inspect element” which is very similar to that of firebug’s inspection.

- Locate the desired object within the web page. One way to do the same is to right click on the desired web element and inspect. The HTML property belonging to that web element would be highlighted in the developer tool. Another way is to hover through the HTML properties and the matching web element would be highlighted. Thus, in this way user can locate ids, class, links.

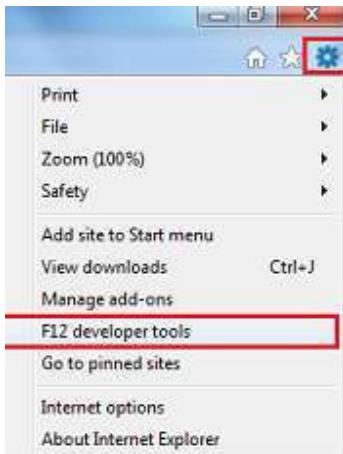


Inspection of element in Internet Explorer browser:

The steps to install firebug and inspect element in the Internet Explorer browser are:

- Open internet explorer and load the web page in the browser.
- Press F12 button in the key board will open the inspect window at the bottom. In IE it is known as “Developer Tool”.

3. Pressing F12 again will disable the inspection of the page.
4. If F12 does not work, then follow the other option which as below:
5. Open internet explorer and load the web page in the browser.
6. Click on Tools icon shown at right top and click on the option F12 developer tools.



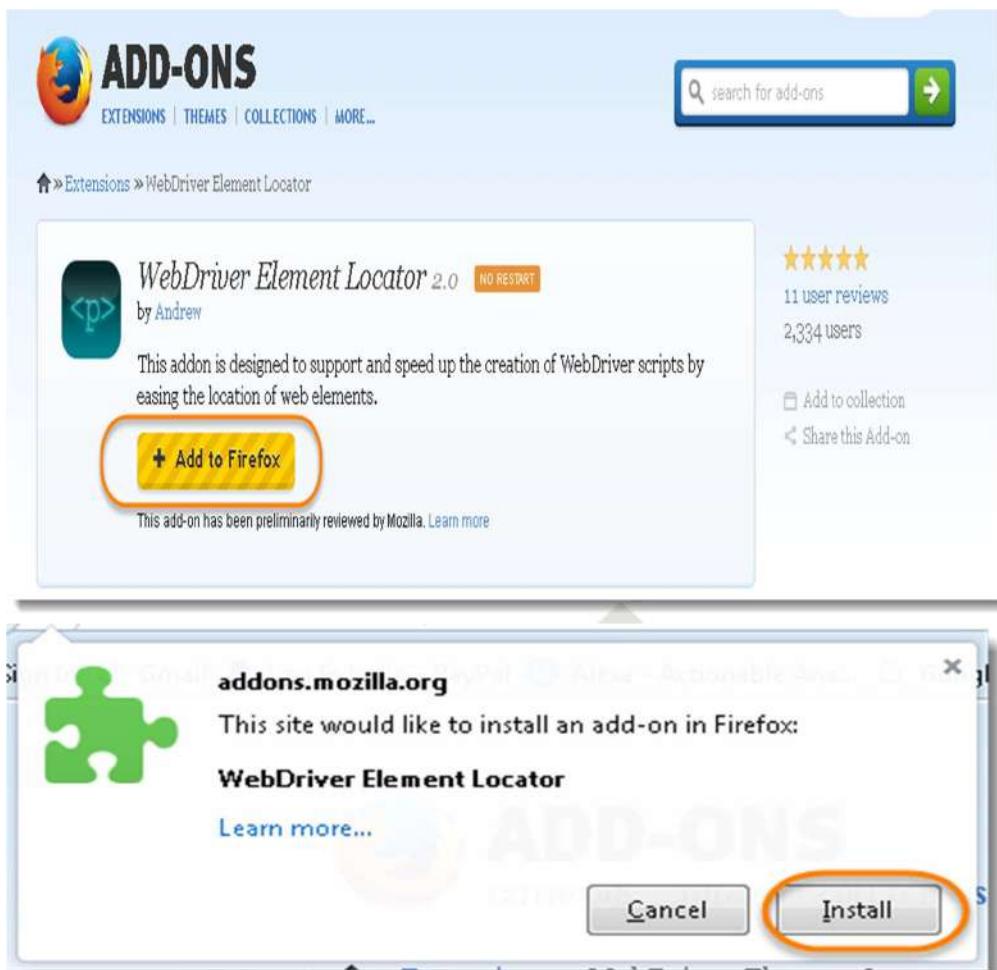
7. Clicking on F12 developer tools will inspect the element and will show the result at the bottom.
8. Same process can be follow to disable the developer tool or to inspect element.

Element Locator Tool for Firefox Locator

WebDriver Element Locator is an add-on for Firefox browser. As this is a add on to Firebox, it is easy to use and to use this by just right click on the web element which is to be locate, selecting an appropriate locator string and copying to clipboard. It would show multiple options of element locators in the browser's context menu. It displays the element locator with the complete Selenium script in different languages like C#, Java, Python and Ruby.

This addon can be downloaded as Firefox Add-On from the browser as below.

1. Go to Tools > Add-Ons, Search for WebDriver Element Locator, Click on Install,
2. Restart the Firefox Browser Click on Add to Firefox



The add-on provides below features:

- This add-on will check the locators for uniqueness, signified by red crosses and green ticks.
- If elements have long, fragile, auto-generated attributes such as id="ctl00_ElementContainer_Inputs_txtForename", the add-on will attempt to locate based on the final (and most significant) part of the value only.
- If locating via attributes is difficult, this add-on will attempt to locate via text value.

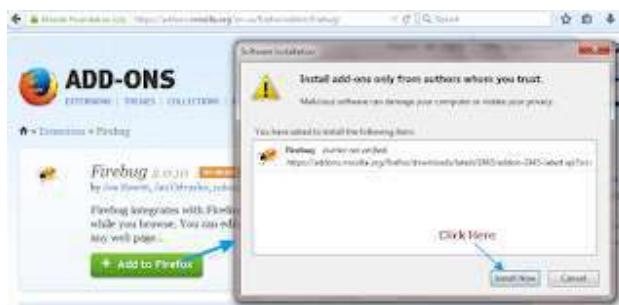
Firebug and Firepath Add-ons in Mozilla

Firepath and Firebug are very useful and powerful Firefox add-ons that could be used to inspect and generate XPath 1.0 expressions. Firepath and Firebug are used to find out the XPath or CSS Locators of the required elements. Below are the steps to install firebug

1. Go to firebug add-on Installation page using URL :- <https://addons.mozilla.org/en-us/firefox/addon/firebug/>. It show firebug add-on Installation button "Add to Firefox" as bellow.



2. Click On "Add to Firefox" as shown In above Image. It will open Software Installation dialog as bellow.



3. Click on "Install Now" button as shown In above Image.
4. It will start Installing Firebug In Firefox browser. At the end of Installation, It will show message like "Firebug has been Installed successfully." and show Firebug Icon In toolbar section as shown In bellow Image.



5. Now Firebug Is Installed In Firefox browser.

Steps to Install FirePath Add-On in Firefox Browser are as follow:

1. Open Mozilla Firefox browser.

Go to Fire path add on Installation page using URL: <https://addons.mozilla.org/en-us/firefox/addon/firepath/>. It will open the fire path Installation page as shown In bellow Image. Click on "Add to Firefox" button as shown In bellow given Image.



2. It will show Software Installation dialog as shown In bellow Image. Click on "Install Now" button. It will start Installing fire path add-on.



3. It will ask to restart Firefox browser at the end of Installation process as shown In bellow Image. Click on "Restart Now" as shown In bellow Image. It will restart Firefox browser.

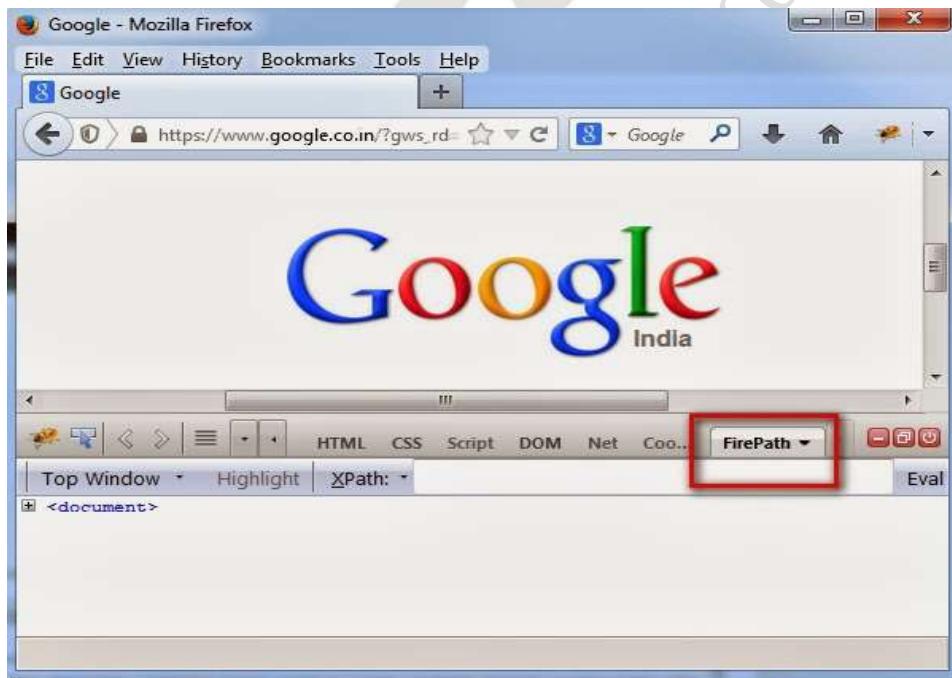


Finding out the 'Xpath' value of the inspected UI element using Firepath

1. Launch Firefox and navigate to any site say www.google.com
2. Click on the FireBug icon on the top right side of the page as shown below:



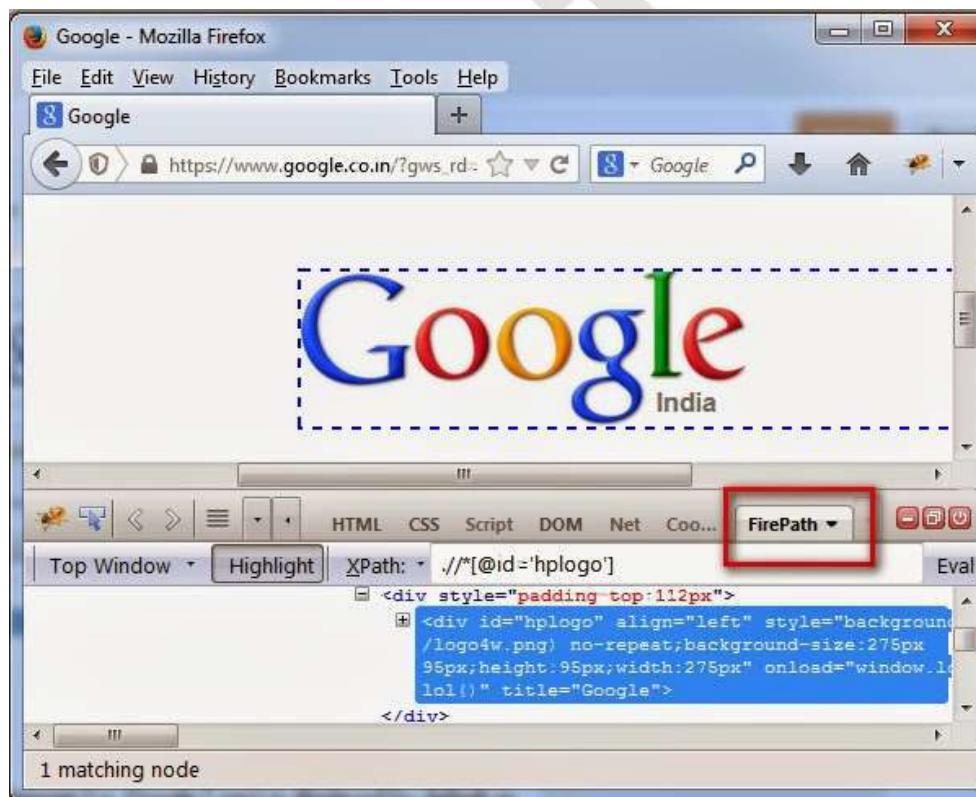
3. Ensure that FireBug interface along with 'FirePath' tab is displayed on the bottom of the page as shown below:



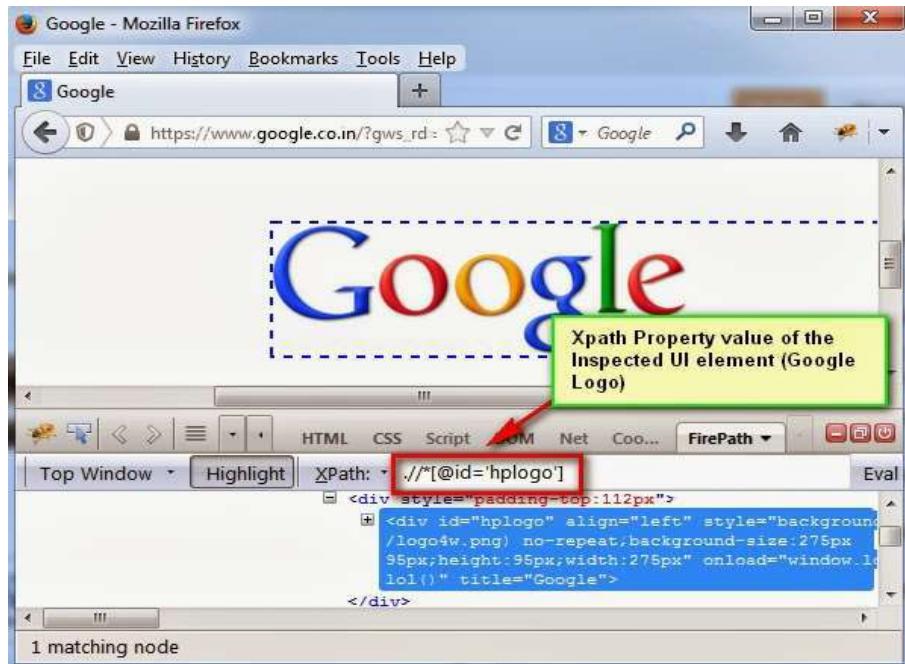
4. Click on the 'Inspect Element' FireBug option and select any UI element say 'Google Logo' as shown below:



5. Click on the 'Firepath' tab to find out the Xpath value of the inspected element (i.e. Google Logo in this example) as shown below:



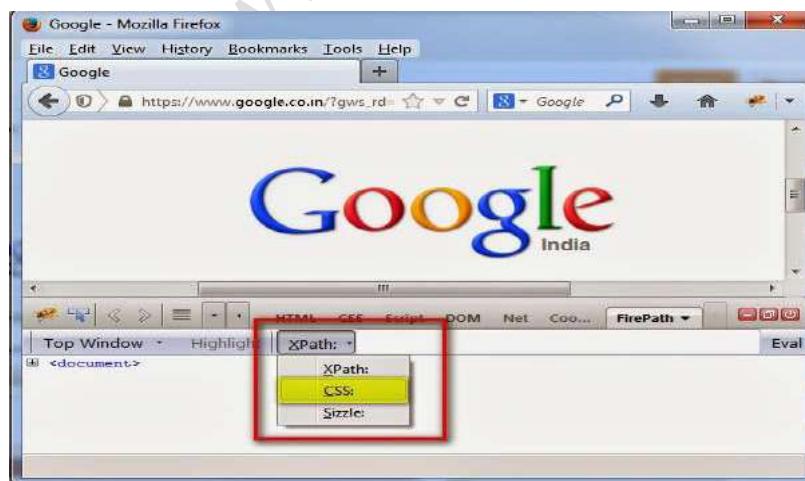
6. Ensure that 'Xpath' property value of the selected UI element (i.e. Google Logo) is displayed by default as shown below:



7. Copy the 'Xpath' property value of the inspected element

Finding out 'CSS' Selector value of the inspected UI element using Firepath:

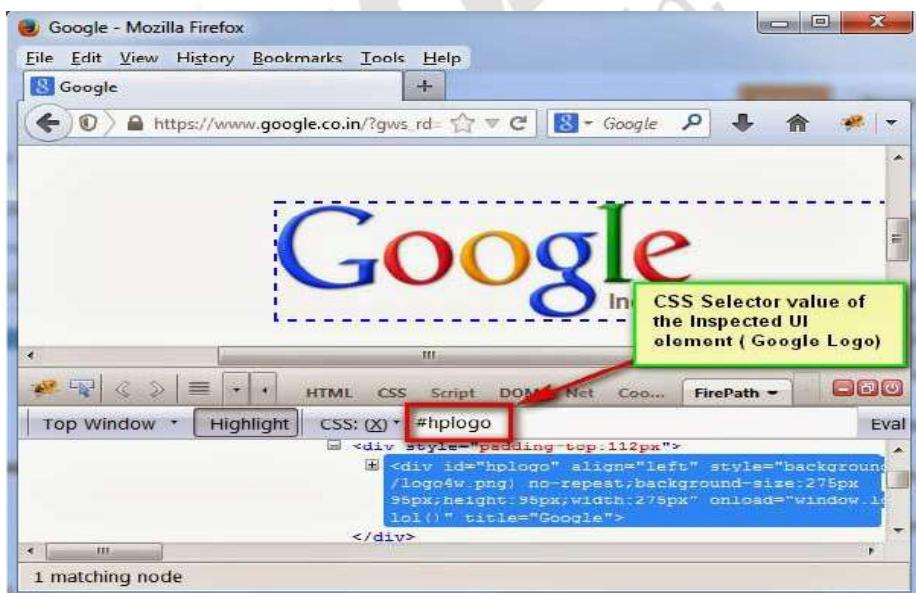
1. Repeat steps 1-3 of previous post.
2. Click on the Dropdown field as shown below and select 'CSS'



3. Click on the 'Inspect Element' firebug option and select any UI element say 'Google Logo' to inspect as shown below:



4. Ensure that 'CSS' property value of the inspected UI element (i.e. Google Logo) is displayed as shown below



5. Copy the 'CSS' Selector property value of the inspected element

Various HTML Locator Strategies

CSS selectors are string representation of HTML tags, attributes, Id, Class. CSS selectors can be used to locate elements which don't have Id or name or class. The different combinations of attributes which can be used to locate an element using CSS selector are:

- **Tag and ID:**

Syntax

```
css=tag#id
```

Example:

```
<input id="txtName" class="textboxcss" tabindex="1"
type="text">

css=input#txtName

WebElement cssele =
driver.findElements(By.cssSelector("input#txtName"));
```

Here input is tag name and Id is txtName with # sign.

- **Tag and Class:**

Syntax

```
css=tag.classname.
```

Example 1:

```
<input id="txtName" class="textboxcss" tabindex="1"
type="text">

css=input.textboxcss

WebElement cssele =
driver.findElements(By.cssSelector("input.textboxcss"));
```

Here input is tag name followed by dot and class name textboxcss

Example 2:

```
<input id="txtName" class="textboxcss top" tabindex="1"
type="text">

css=input.textboxcss.top

WebElement cssele =
driver.findElements(By.cssSelector("input.textboxcss.top"));
```

Here class name is textboxcss<space>top in that case the dot is put in between textboxcss and top.

- **Tag and Attribute:**

Syntax

```
css=tag[attribute='value']
```

Example:

```
<input value="Reading" type="checkbox">
css=input[type='checkbox']
```

Here in first case type is attribute and checkbox is its value, in other case value is an attribute and its value is Reading.

- **Tag, ID and Attribute:**

Syntax

```
css=tag#id[attribute='value']
```

Example:

```
<input id="txtName" class="textboxcss" tabindex="1"
name="taComment" type="text">
<input id="txtName" class="textboxcss" tabindex="1"
name="tbComment" type="text">
css=input#txtName[name='taComment']
WebElement cssele =
driver.findElements(By.cssSelector("input#txtName[name='taComment']"));
```

Here for both textboxes id is same ever class name is also same, only name is different for both. So if to locate first text box go with the locator given in above example. For second text box value of name attribute need to be change in same combination.

- **Tag, Class and Attribute:**

Syntax

```
css=tag.classname[attribute='value']
```

Example:

```

<input class="textboxcss" tabindex="1" name="taComment"
type="text">
<input class="textboxcss" tabindex="1" name="tbComment"
type="text">
css=input.textboxcss [name='taComment']
WebElement cssele =
driver.findElements(By.cssSelector("input.textboxcss
[name='taComment']"));

```

- **`nth-child()`:**

Syntax

`css=tag:nth-child(n)`

n represent child number

Example:

```

<ul>
<li>C</li>
<li>C++</li>
<li>C#</li>
<li>Java</li>
<li>Python</li>
<li>Ruby</li>
</ul>
css= li:nth-child(n)
WebElement cssele = driver.findElements(By.cssSelector("li:nth-
child(n)"));

```

Here ul li parent child structure is shown. Only tag names are common to everyone. Here to locate sat Java, put n=4 in above command.

- **`Inner Text:`**

Syntax

`css= tag:contains('inner text')`

Example:

```

<span>Upload you pic :</span>
css= span:contains('Upload you pic ')

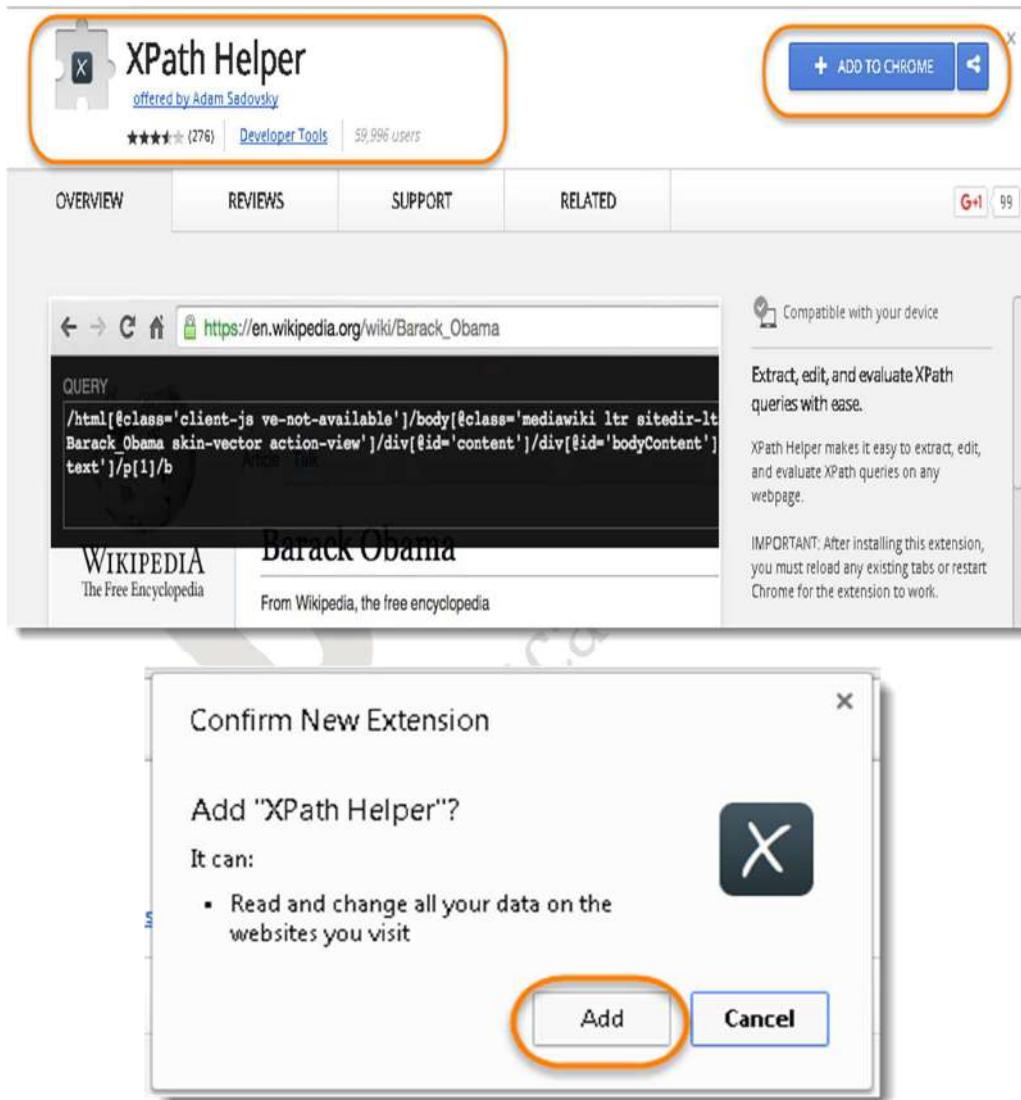
```

```
WebElement cssele =
driver.findElements(By.cssSelector("span:contains('Upload you
pic')"));
```

Xpath Helper Plug-in for Chrome

XPath Helper Plug-in is the chrome browser plug-in for locating the GUI elements on the web pages. The steps to Install XPath Helper Plugin for Chrome Browser is as below:

1. Go to <https://chrome.google.com/webstore/detail/xpath-helper/hgimnogillphhhkhlmebbmlgjoejdjpjl?hl=en>
2. Click on Add to Chrome and then click on Add button.



3. Once it is installed, it will display a pop up message for Successful installation.
4. Now the icon for XPath Helper will appear on the top right most side of the Chrome browser. Just click on it to open the helper.



The steps provide insight for using XPath Helper Tool.

1. Open a new tab and navigate to any webpage. Here www.DemoQA.com is used for demo.
2. Hit Ctrl-Shift-X (or Command-Shift-X on OS X), or click the XPath Helperbutton in the toolbar, to open the XPath Helper console.
3. Hold down Shift on moving mouse over elements on the page. The query box will continuously update to show the XPath query for the element below the mouse pointer, and the results box will show the results for the current query.
4. If desired, edit the XPath query directly in the console. The results box will immediately reflect your changes.
5. Repeat step (2) to close the console.

Note: If the console gets in the desired way, hold down Shift and then move the mouse over it; it will move to the opposite side of the page.

Selection of Effective Xpath

XPath can be viewed as a way to navigate round XML documents. Thus XPath has similarities to a set of street directions ie when one need to search for an address, he should know what is starting point to reach destination.

There are different ways of choosing xpaths and choosing the most effective xpaths as below:

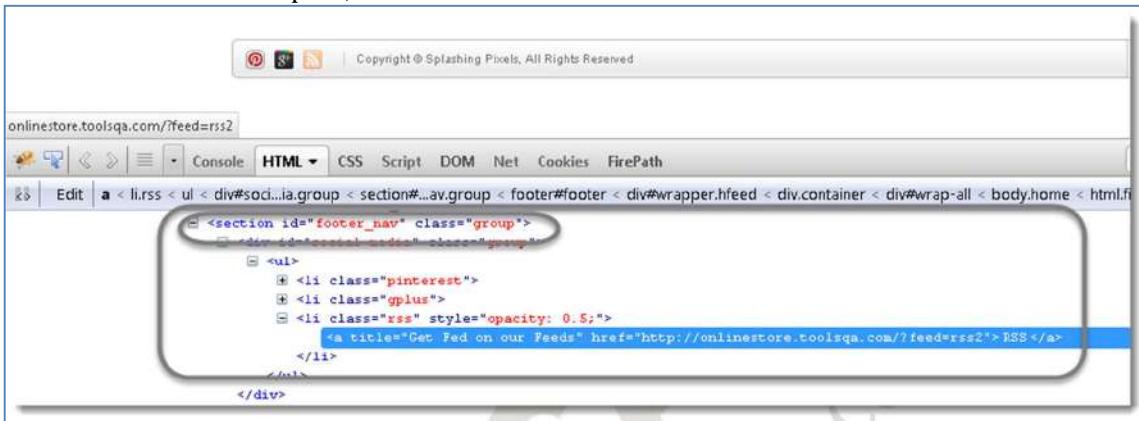
Example: Let's take an example of RSS button at the bottom of the page in the footer section of www.store.demoqa.com.

Technique 1 | Absolute XPath: The easiest way of finding the xpath is to use the Browser Inspector tool to locate an element and get the xpath of it.

XPath Generated by the tool is : /html/body/div[2]/div/div/footer/section[3]/div/ul/li[3]/a

Technique 2 | Relative XPath: At times XPath generated by Firebug are too lengthy and there is a possibility of getting a shorter XPath. Above xpath will technically work, but each of those nested relationships will need to be present 100% of the time, or the locator will not function. Above chosen xpath is known as Absolute xpath. There is a good chance that xpath will vary in every release. It is always better to choose Relative xpath, as it helps us to reduce the chance of element not found exception.

To choose the relative xpath, look for the recent Id attribute as in below screen shot.



The recent or last Id produced is 'footer_nav'. This id would be appropriate in this case, so a quality xpath will look like this: //*[@id='social-media']/ul/li[3]/a

The difference between the Absolute and Relative xpaths is:

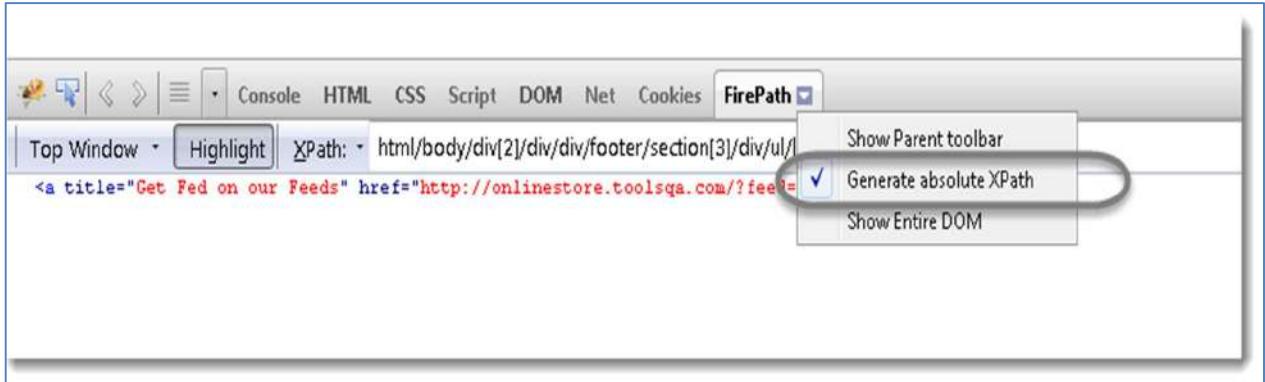
Absolute xpath: /html/body/div[2]/div/div/footer/section[3]/div/ul/li[3]/a

Relative xpath: //*[@id='social-media']/ul/li[3]/a

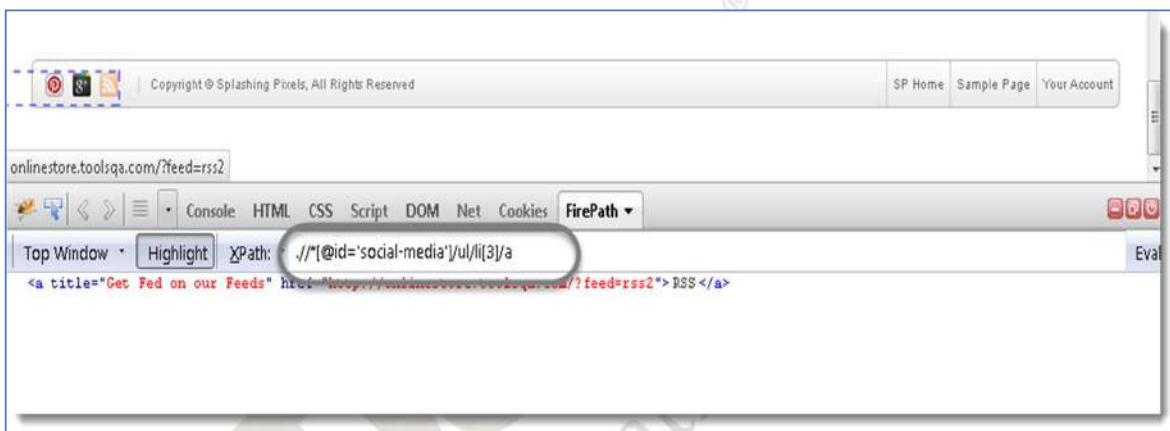
Please note that Absolute xpath is using single slash at the start of the xpath and relative is using double slash.

The difference between single '/' or double '//' is that a single slash at the start of Xpath instructs XPath engine to look for element starting from root node while a double slash at the start of Xpath instructs XPath engine to search look for matching element anywhere in the XML document.

There is an alternate way to get the relative xpath with help of the FirePath tool. Click on the drop down menu on the Firepath button and Unselect 'Generate absolute XPath'.



Now click on the same element with the Inspector, the new xpath will look like this:



If something gets changed above the id social-media, the xpath will still work.

Handling Dynamic Objects/Ids on the Page

In some applications, there are dynamic elements whose ids/xPaths change every time page reloads. A common example is listing of products on an E-commerce application. In this example, locators associated with different products can change if page is reloaded after use of sort, filter or add product function. For example, in order to interact with "Purchase buttons" associated with different Products. These buttons could have IDs in following format.

Example:

```
// Dynamic Element Locators
<button id="Submit-901" />
<button id="Submit-902" />
<button id="Submit-903" />
```

In this test scenario as element IDs can change on page reload, an id for each button has a common text "submit-" and a number that increments by value of 1. So, it need to understand

this scenario requirements and also understand the behavior of these elements as the page is reloaded. Once it understand, a strategy can be devise to interact with these elements.

Few common techniques used to handle dynamic elements are as below:

- **Absolute XPath**

Xpath Position or Absolute Xpath are most frequently used to resolve the dynamic element issues. Reason is, it does not change with page reloads. However, problem with using Absolute XPath locators is that they are very fragile. They are most prone to breakage in case of change in web page. This could create a big trouble for us in maintenance, if we start using these frequently, especially in large test suites. Therefore they should only be used as last option. Below is an example of Absolute XPath and XPath Position.

```
web_element_name=html/body/div[30]/div[2]/div[2]/div/div/div[1]/
table/tbody/tr/td[2]/table/tbody/tr/td[1]/table/tbody/tr/td[1]/table
/tbody/tr[2]/td[2]/em/button
//p[6]/label[2]/div/ins
```

- **Identify Element by starting Text**

If the dynamic elements have a definite pattern to them, here JavaScript functions can be use like “starts-with” or “contains” in element locators to separate the dynamic part of locator from static part.

For example, in case of dynamic submit button Id example which was discussed earlier, It can apply ‘starts-with’ function to access this locator irrespective of its dynamic part.

XPath: //button[starts-with(@id, 'Submit-')]

- **Identify Element by containing Text**

Similarly, in some scenarios where dynamic element is surrounded by a static value, ‘contains’ function can be use. For example element locators are as follow:

```
<input class="new-userfield-001">
<input class="old-userfield-002">
```

As ‘usefield’ part of element is static, so ‘contains’ function can apply to access this element locator as shown below...

XPath: //input[contains(@class, '-userfield-')].

- **Identify Element by Index**

If there are multiple elements present on page with same locator then use following Java code in the selenium WebDriver script to interact with element found at particular index.

```
driver.findElements(By.xpath("//*submit")).get(0).click();
```

- **Identify Element with reference of a closest stable element**

Use the DOM structure to find the closest stable element first and then this stable element can be used as a reference element to find the required element.

XPATH: //span1/.../following-sibling::div//button1

DOM structure could be found using Firefox extension like Firebug and FirePath. But in complex and large applications this approach is difficult to use because of large DOM structure.

- **Identify Element by stable preceding Text**

For web elements like text field and text areas we can identify them by using the stable text labels nearby. This approach might not be possible in all scenarios but it does resolve the dynamic element issues where possible.

Summary

- Inspecting elements in the browser helps web developers and designers to understand the process which occurs during the http request and response to and fro from the server
- WebDriver Element Locator is an add-on for Firefox browser
- Firepath and Firebug are Firefox add-ons used to inspect and generate XPath 1.0 expressions
- CSS selectors are string representation of HTML tags, attributes, Id, Class
- XPath Helper Plug-in is the chrome browser plug-in for locating the GUI elements on the web pages
- XPath can be viewed as a way to navigate round XML documents

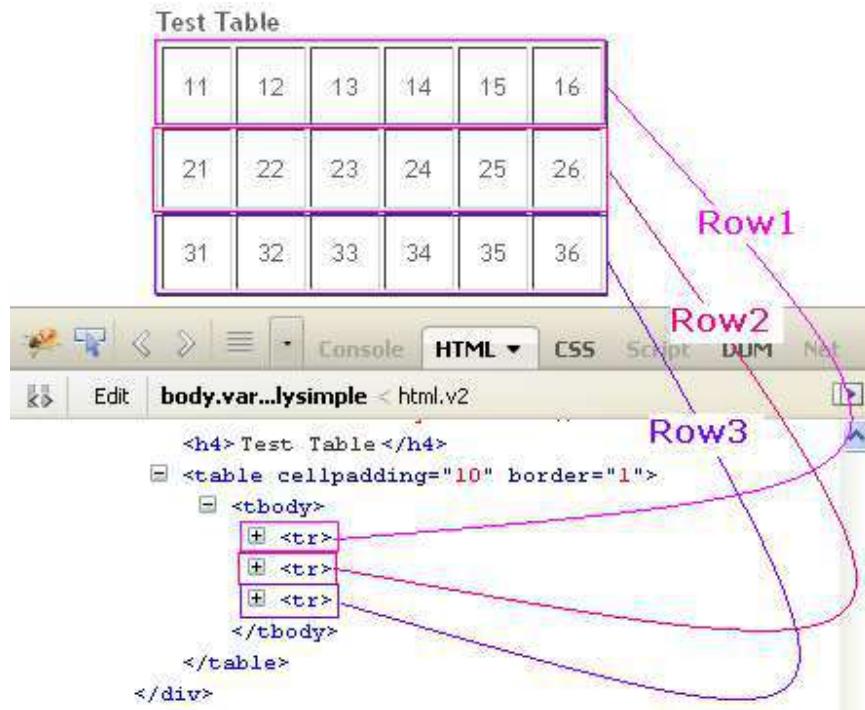
Chapter 8 –Tables, Checkboxes and Radio Buttons

Identify Table Rows and Columns

Web tables are a group of elements that are logically stored in a row and column format. It is used to organize similar information on a web page. The various tags defined in HTML table are:

- 'table' tag defines HTML table.
- 'tbody' tag defines a container for rows and columns.
- 'tr' defines rows in an HTML table.
- 'td'/'th' define the column of an HTML table.
- <th> tag for headings

The diagram below shows the mapping of Table with HTML tags



Excel sheet is a simple example of table structures. Each cell in the Excel sheet can be represented as <td> in the HTML table. The <td> elements are the data containers and these can contain all sorts of HTML elements like text, images, lists, other tables.

Example of HTML table (first row is defined as header and later two rows are containing data).

```

<table>
  <tbody>
    <tr>
      <th>Automation Tool</th>
      <th>Licensing</th>
      <th>Market response</th>
    </tr>
    <tr>
      <td>Apache Ant</td>
      <td>Apache Maven</td>
      <td>Apache Ivy</td>
    </tr>
  </tbody>
</table>

```

```

<tr>
    <td>Selenium</td>
    <td>Free</td>
    <td>In</td>
</tr>
<tr>
    <td>QTP</td>
    <td>Paid</td>
    <td>Out</td>
</tr>
</tbody>
</table>

```

Process to Identify Table rows and columns

The process to identify the Table rows and columns in selenium are:

- Get the entire HTML table and store this in a variable '*htmlTable*' of type web element.
- Get all the rows with tag name 'tr' and store all the elements in a list of web elements.
Now all the elements with tag 'tr' are stored in 'rows' list.
- Loop through each row and get the list of elements with tag 'th'. '*rows.get(0)*' will give first row and '*findElements(By.tagName("th"))*' will give list of columns for the row.
- Iterate using '*columns.getsize()*' and get the details of each cell.

In Selenium there are two ways of Identifying Table Rows and Columns:

- For Static tables: If a number of rows and columns are always constant,

Below is the xpath of one of the cell in html table. Let's say "firstname"

```
//div[@id='main']/table[1]/tbody/tr[1]/th[1]
```

Where, tr[1] defines first row and th[1] defines first column.

Example:

```

for(int numberOfRows=1; numberOfRows<=5; numberOfRows++)
{
    for(int numberOfCol=1; numberOfCol<=3; numberOfCol++)
    {

```

```

        System.out.println(driver.findElement(By.xpath
        ("//div[@id='main']/table[1]/tbody/tr
        ["+numberOfRows+"]/th["+numberOfCol+"]")));
    }
}
}

```

- For Dynamic Tables: If Number of rows and columns are NOT fixed.

Example:

```

WebElement htmltable=driver.findElement(By.xpath("//*[@id='main']/
table[1]/tbody"));
List<WebElement> rows=htmltable.findElements(By.tagName("tr"));

```

```

for(int rnum=0;rnum<rows.size();rnum++)
{
List<WebElement>
columns=rows.get(rnum).findElements(By.tagName("th"));
System.out.println("Number of columns:"+columns.size());
for(int cnum=0;cnum<columns.size();cnum++)
{
System.out.println(columns.get(cnum).getText());
}
}

```

Extracting Values from a Cell

Selenium WebDriver provides method `getText()` which can be used to read the inner text of any web element. For example below line of code returns the data displayed in cell c in selenium WebDriver.

Syntax

```
String actualValue = c.getText();
```

Example

```

package temp;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

```

```

import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.Select;
public class first {
public static void main(String[] args) {
    // TODO Auto-generated method stub
System.setProperty("webdriver.chrome.driver", "C:\\Selenuim\\chromedriver2.3.exe");
WebDriver driver = new ChromeDriver();
try{
driver.get("http://register.rediff.com/register/register.php");
Thread.sleep(2000);
WebElement e = driver.findElement(By.tagName("td"));
String actualValue = e.getText();
System.out.println("Text displayed in the first td ->
" + actualValue);
Thread.sleep(2000);
}
catch(Exception ex){
    System.out.println("Exception " + ex.getMessage());
}
finally{
    driver.close();
    driver.quit();
}
}
}

```

Dynamically Identify Tables Data

For identifying tables data dynamically follow the below mention sequence of steps:

- Fetch no of rows in a table

Once the table is located the next step is to iterate through it and count the number of rows & columns. Below code identifies all the rows & store it in the 'rows' list. Next we calculate the number of rows using `size()` method & store it in the '`rows_count`' variable.

```
List rows = Table.findElements(By.tagName("tr"));
```

```
rows_count = rows.size();
```

- Fetch # of columns in a row

Below code identifies all the columns in a row & store it in the 'columns' list. Next we calculate the number of columns using `size()` method & store it in the '`col_count`' variable.

```
List columns = rows.get(i).findElements(By.tagName("td"));
col_count = columns.size();
```

- Iterate through the Web Table

Outer loop iterates through all the rows one at a time and inner loop through all the columns (cells) within each row. `getText()` method is used to fetch a cell's value.

```
for(int i= 0; i<rows_count; i++) {
    List columns = rows.get(i).findElements(By.tagName("td"));
    col_count = columns.size();
    for(int j=0; j< col_count; j++) {
        cellText = columns.get(j).getText();
        System.out.print(cellText+" ");
    }
    System.out.println("");
}
```

Example

```
import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
public class WebTable {

    public static void main(String args[]) {
        int rows_count, col_count;
        String cellText = null;
        //Navigate to the Web page
        System.setProperty("webdriver.chrome.driver","Path to
your chromedriver.exe");
```

```

        WebDriver driver = new ChromeDriver();
        driver.get("http://money.rediff.com/gainers/bsc/daily/gro
upa?");
        driver.manage().window().maximize();
        //Identify the table
        WebElement Table =
driver.findElement(By.ByClassName.className("dataTable"));
        //Read specific cell value
        WebElement cell =
driver.findElement(By.xpath("//table/tbody/tr[2]/td[3]"));
        System.out.println("Data for Row 2 and Column 3 is
"+cell.getText()+".");
        //Fetch # of rows in a table
        List rows = Table.findElements(By.tagName("tr"));
        rows_count = rows.size();
        //Iterate through the rows
        for(int i= 0; i<rows_count; i++){
            //Fetch # of columns in a row
            List columns =
rows.get(i).findElements(By.tagName("td"));
            col_count = columns.size();
            //Iterate through the columns within particular row
            for(int j=0; j< col_count; j++){
                cellText = columns.get(j).getText();
                System.out.print(cellText+"   ");
            }
            System.out.println("");
        }
    }
}
    
```

CheckBox & Radio Button

CheckBox & Radio Button Operations are easy to perform and most of the times the simple ID attributes work fine for both of these. But selection and d-selection is not the only thing which need with Check Boxes and Radio Buttons. It might like to check that if the Check Box is already checked or if the Radio Button is selected by default or anything. Check Boxes and Radio

Button deals exactly the same way and it can perform below mentioned operations on either of them.

Methods to Select Checkbox And Radio Button

- **Use ID For Selecting Checkbox/Radio Button.**

The ID attribute can be used to select a Radio Button or a *CheckBox*. The Webdriver command is here to click which can be applied to both types of elements

```
// Java code example to select checkbox/radio button.
WebElement target = driver.findElement(By.id("checkbox1"));

target.click();
```

- Use *IsSelected* Method To Check The State Of Checkbox/Radio Button.

If a Checkbox/Radio Button has selected/deselected and just check its final state. Then, use the <*IsSelected*> command to know that the correct status of the element.

```
import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
public class findElementsTest {
    public static void main(String[] args) throws Exception {
        // Launch browser
        WebDriver driver = new FirefoxDriver();
        // Maximize window
        driver.manage().window().maximize();
        // Navigate to the URL
        driver.get("http://www.techbeamers.com");
        // Sleep for 5 seconds
        Thread.sleep(5000);
        // Store all the elements of the same category in the
        //list of WebElements.
        List<WebElement> list =
        driver.findElements(By.name("radioButton"));
```

```

// Create a boolean variable to store true/false.
Boolean is_selected = list.get(0).isSelected();

// If 'is_selected' is true that means the first radio
//button is selected.
if (is_selected == true) {

    // If the first radio button is selected by default
    //then, select the second radio button.
    list.get(1).click();

} else {

    // If the first radio button is not selected then,
    //click the first radio button.
    list.get(0).click();
}
}
}

```

Note: When there is a group of Radio Buttons/Check Boxes on the page then, it is possible that their names are same, but values are different. That's why it has used the Webdriver *findElements* command to get the list of web elements.

- Use Element Value For Selecting Checkbox/Radio Button.

One of the intuitive ways to select the Radio Buttons/Check Boxes is by toggling their Values. Please follow the below code example for more clarity.

```

import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
public class findElementsTest {

    public static void main(String[] args) throws Exception {
        // Launch browser

```

```

WebDriver driver = new FirefoxDriver();
// Maximize window
driver.manage().window().maximize();
// Navigate to the URL
driver.get("http://www.techbeamers.com");
// Sleep for 5 seconds
Thread.sleep(5000);
// Find the checkbox or radio button element by its name.
List<WebElement> list =
driver.findElements(By.name("checkbox"));

// Get the number of checkboxes available.
int count = list.size();

// Now, iterate the loop from first checkbox to last c
//checkbox.
for (int i = 0; i < count; i++) {

    // Store the checkbox name to the string variable,
    //using 'Value' attribute
    String sValue = list.get(i).getAttribute("value");

    // Select the checkbox if its value is the same that
    //you want.
    if (sValue.equalsIgnoreCase("checkbox")) {

        list.get(i).click();

        // This statement will get you out of the for
        //loop.
        break;
    }
}
}

```

- Use CssSelector For Selecting Checkbox/Radio Button.

Another easy way to select/deselect a checkbox or a radio button is by using the *cssSelector*. Please refer the below code snippet to bring more clarity.

```
// Java example code to select a checkbox using the cssSelector.
WebElement checkBox =
driver.findElement(By.cssSelector("input[value='TechBeamers']"));
checkBox.click();
```

Example

Below is the HTML code of the input form which you can save as an HTML file locally to practice the checkbox and radio button operations.

```
<html>
<head>
<title>Perform Checkbox and Radio Button Operations.
</title>
</head>
<body>
<form name="testform" action="" method="POST">
<div align="center">
<br>
<input type="checkbox" name="option-1" value="Java">Java
<input type="checkbox" name="option-2" value="C++">C++
<input type="checkbox" name="option-3" value="Python"
checked>Python
<input type="checkbox" name="option-4" value="PHP">PHP
<input type="checkbox" name="option-5" value="CSharp">CSharp
<input type="checkbox" name="option-6" value="Ruby">Ruby
<input type="checkbox" name="option-7" value="Perl">Perl
<br>
<input type="radio" name="group-1" value="Programming">
Programming
<input type="radio" name="group-1" value="Testing"> Testing
<input type="radio" name="group-1" value="Test Automation"
checked> Test Automation
</div>
```

```

</form>
</body>
</html>

```

Select Class in Selenium

Select is a class which is provided by Selenium to perform multiple operations on DropDown object and Multiple Select object. This class can be found under the Selenium's *Support.UI.Select* package. Its object is created by a *New* keyword with regular class creation. Select class only works for elements with <select> tags.

Syntax

```
Select oSelect = new Select();
```

Example

```
WebElement element = driver.findElement(By.id("Country"));
Select oSelect = new Select(element);
```

WebDriver Code using Selenium Select Class

Create a new java class named as "HandlingDropDown" under any running project.

Copy and paste the below code in the "HandlingDropDown.java" class.

Below is the test script that is equivalent to the above-mentioned scenario

```

import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;

/**
 * class description

```

```

*/



public class HandlingDropDown {
    WebDriver driver;

    /**
     * Set up browser settings and open the application
     */
    @Before
    public void setUp() {
        driver=new FirefoxDriver();

        // Opened the application
        driver.get("file:///F:/Work/Blogs/testingstuff/DemoWeb
Alert.html");
        driver.manage().window().maximize();
    }

    /**
     * Test to select the dropdown values
     * @throws InterruptedException
     */
    @Test
    public void testSelectFunctionality() throws
InterruptedException {

        // Go to google
        driver.findElement(By.linkText("Google")).click();

        // navigate back to previous webpage
        driver.navigate().back();
        Thread.sleep(5000);

        // select the first operator using "select by value"
    }
}

```

```

        Select selectByValue = new
Select(driver.findElement(By.id("SelectID_One")));
        selectByValue.selectByValue("greenvalue");
        Thread.sleep(5000);

// select the second dropdown using "select by visible text"
        Select selectByVisibleText = new Select
(driver.findElement(By.id("SelectID_Two")));
        selectByVisibleText.selectByVisibleText("Lime");
        Thread.sleep(5000);

// select the third dropdown using "select by index"
        Select selectByIndex = new
Select(driver.findElement(By.id("SelectID_Three")));
        selectByIndex.selectByIndex(2);
        Thread.sleep(5000);
    }

/**
 * Tear down the setup after test completes
 */
}

@After
public void tearDown() {
    driver.quit();
}
}

```

Drop Down Handle

A drop-down list is a graphical control element, similar to a list box which allows the user to choose one value from a list. When a drop-down list is inactive it displays a single value. When activated it displays a list of values from which the user may select one. When the user selects a new value the control reverts to its inactive state displaying the selected value. For handling dropdowns Selenium provides Select class that has some predefined method. There are three different ways by which dropdown values can be selected.

- Select by index
- Select by value
- Select by visible

Select Multiple Values from the List

WebDriver provides a Select class to Select Multiple Values from the List. Lets understand with an example of below dropdown with four values.

Maruti

Honda

BMW

Toyota

Option 1) if user wants to select the first two options from above drop down, then the code will be as below. The code will Select Maruti and Honda from a list.

```
WebElement selectList=
driver.findElement(By.xpath("//select[@name='cars']"));
Select select = new Select(selectList);
select.selectByVisibleText("Maruti");
select.selectByVisibleText("Honda");
```

Option 2) if user wants to select multiple options from drop down manually, he would press CTRL and click on all the options which he wants to select.In order to replicate the above method using automation webdriver provides Actions class to interact with Keyboard actions.

```
WebElement option1=
driver.findElement(By.xpath("//option[@value='Maruti']"));
WebElement option2=
driver.findElement(By.xpath("//option[@value='Honda']"));
Actions action = new Actions(driver);
action.keyDown(Keys.CONTROL).click(option1).click(option2).build().perform();
```

Above code will Select Maruti and Honda from a list. Here first we have put all the options which user wants to select into WebElement object and then used Actions Class to press down CTRL key and perform clicks on both options.

Example :Print and select all the options for the selected Multiple selection list.

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

import org.openqa.selenium.By;
```

```

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;

public class MultiSelectCommands {

    public static void main(String[] args) throws
InterruptedException {
        // Create a new instance of the FireFox driver
        WebDriver driver = new FirefoxDriver();

        // Put an Implicit wait
        driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);

        // Launch the URL
        driver.get("http://toolsqa.wpengine.com/automation-
practice-form");

        // Step 3: Select 'Selenium Commands' Multiple select box
        ( Use Name locator to identify the element )
        Select oSelect = new
Select(driver.findElement(By.name("selenium_commands")));

        // Step 4: Select option 'Browser Commands' and then
deselect it (Use selectByIndex and deselectByIndex)
        oSelect.selectByIndex(0);
        Thread.sleep(2000);
        oSelect.deselectByIndex(0);

        // Step 5: Select option 'Navigation Commands' and then
deselect it (Use selectByVisibleText and deselectByVisibleText)
        oSelect.selectByVisibleText("Navigation Commands");
        Thread.sleep(2000);
        oSelect.deselectByVisibleText("Navigation Commands");

        // Step 6: Print and select all the options for the
selected Multiple selection list.
        List<WebElement> oSize = oSelect.getOptions();
        int iListSize = oSize.size();

        // Setting up the loop to print all the options
        for(int i =0; i < iListSize ; i++){
            // Storing the value of the option
            String sValue =
oSelect.getOptions().get(i).getText();

            // Printing the stored value
            System.out.println(sValue);

            // Selecting all the elements one by one
            oSelect.selectByIndex(i);
}

```

```

        Thread.sleep(2000);
    }

    // Step 7: Deselect all
    oSelect.deselectAll();

    // Kill the browser
    driver.close();
}
}

```

Select and Deselect Operations By Index, Value and Visible Text

SelectByVisibleText: This method is used to choose or select an option given under any dropdowns and multiple selection boxes. It takes a parameter of String which is one of the Value of Select element and it returns nothing.

Syntax:

```
oSelect.selectByVisibleText("text");
```

Example: To select the value 2010.

```
Select oSelect=newSelect(driver.findElement(By.id("yy_date_8")));
oSelect.selectByVisibleText("2010");
```

Example: Any of the below html code can be taken as example.

Webdriver example code to select the value by visible text.

```
public class selectExamples {
    WebDriver driver;
    @Test
    public void selectSamples()
    {
        driver = new FirefoxDriver();
        driver.get("C:\\\\Users\\\\dropdown-select.html");
        WebElement
element=driver.findElement(By.name("Mobiles"));
        Select se=new Select(element);
        se.selectByVisibleText("HTC");
    }
}
```

```

        }
    }
}
```

SelectByIndex: It is same as selectByVisibleText but the only difference is that in this method we provide the index number of the option rather than the option text. It takes a parameter of int which is the index value of Select element and it returns nothing.

Syntax:

```
oSelect.selectByIndex(int);
```

Example: To select the value 2010 using index.

```
Select oSelect=newSelect(driver.findElement(By.id("yy_date_8")));
oSelect.selectByIndex(4);
```

Index starts from Zero, so the fifth position value will be at index 4.

Example

```
<html>
<head>
<title>Select Example by Index value</title>
</head>
<body>
<select name="Mobiles"><option value="0" selected> Please
select</option>
<option value="1">iPhone</option>
<option value="2">Nokia</option>
<option value="3">Samsung</option>
<option value="4">HTC</option>
<option value="5">BlackBerry</option>
</select>
</body>
</html>
```

Example: Webdriver code for Selecting a Value using select.selectByValue(Value);

```

public class selectByIndexExample {
    WebDriver driver;
    @Test
    public void selectSamples()
    {
        driver = new FirefoxDriver();
        driver.get("C:\\\\Users\\\\DEV\\\\Desktop\\\\html-and-css-code-samples\\\\Final Code\\\\chapter-07\\\\dropdown-select.html");
        WebElement
        element=driver.findElement(By.name("Mobiles"));
        Select se=new Select(element);
        se.selectByIndex(1);
    }
}

```

deselectByIndex: Deselect the option at the given index.

Syntax:

```
oSelect.deselectByIndex;
```

selectByValue: This method takes the value of the option rather the option text or index. It takes a parameter of String which is one of the value of Select element and it returns nothing.

Syntax:

```
oSelect.selectByValue("text");
```

Example: To select the value 2014

```
Select oSelect=newSelect(driver.findElement(By.id("yy_date_8")));
oSelect.selectByValue("2014");
```

Example:

```
<html>
```

```

<head>
<title>Select Example by Value</title>
</head>
<body>
<p>Which mobile device do you like most?</p>
<select name="Mobiles"><option selected> Please select</option>
<option value="iphone">iPhone</option>
<option value="nokia">Nokia</option>
<option value="samsung">Samsung</option>
<option value="htc">HTC</option>
<option value="blackberry">BlackBerry</option>
</select>
</body>
</html>

```

Webdriver code for Selecting a Value using select.selectByValue(Value);

```

public class selectExamples {
    WebDriver driver;
    @Test
    public void selectSamples()
    {
        driver = new FirefoxDriver();
        driver.get("C:\\\\Users\\\\DEV\\\\Desktop\\\\html-and-css-code-
samples\\\\Final Code\\\\chapter-07\\\\dropdown-select.html");
        WebElement
element=driver.findElement(By.name("Mobiles"));
        Select se=new Select(element);
        se.selectByValue("nokia");
    }
}

```

deselectByValue: Deselect all options that have a value matching the argument.

Syntax:

- oSelect.deselectByValue;

deselectByVisibleText: Deselect all options that display text matching the argument.

Syntax:

```
oSelect.deselectByVisibleText
```

deselectAll

Syntax:

```
select.deselectAll();
```

Example: To clear all selected entries. This works only when the SELECT supports multiple selections. It throws NotImplementedException if the "SELECT" does not support multiple selections. In select it mandatory to have an attribute multiple="multiple"

```
<html>
<head>
<title>Multi select Drop Down List Box</title>
</head>
<body>
<p>What all devices do you listen to music on?</p>
<select name="Mobdevices" multiple="multiple"><option value="0" selected> Please select</option>
<option value="1">iPhone</option>
<option value="2">Nokia</option>
<option value="3">Samsung</option>
<option value="4">HTC</option>
<option value="5">BlackBerry</option>
</select>
</body>
</html>
```

Webdriver code to show all the above deselect methods.

```
public class selectExamples {
    WebDriver driver;
    @Test
    public void selectSamples() throws InterruptedException
    {
        driver = new FirefoxDriver();
```

```

driver.get("C:\\\\Users\\\\DEV\\\\Desktop\\\\html-and-css-code-
samples\\\\Final Code\\\\chapter-07\\\\dropdown-select.html");

WebElement
element=driver.findElement(By.name("Mobdevices"));

Select se=new Select(element);

//Here we will take multi select dropdown to show you the
difference

se.selectByVisibleText("HTC");
se.selectByValue("nokia");

//From the above two commands, in the dropdown two values
will be selected.

//Now we will try to deselect any of the One
//Im using thread to see the difference when selecting
and selecting

Thread.sleep(3000);
se.deselectByValue("nokia");

//You can deselect the value by specifying the index,
value and VisibleText

//It will work if you the index is already selected
se.deselectByIndex(1);

//It will deselect if the visible text HTC is in selected
mode

se.deselectByVisibleText("HTC");

//It will de-select all the values which are selected
se.deselectAll();

}

}

```

Summary

- Web tables are a group of elements that are logically stored in a row and column format
- Selenium WebDriver provides method getText() which can be used to read the innertext of any web element
- Select is a class which is provided by Selenium to perform multiple operations on DropDown object and Multiple Select object
- A drop-down list is a graphical control element, similar to a list box which allows the user to choose one value from a list

DUCAT®
www.ducatindia.com

Chapter 9 –Selenium Waits, Alerts and Switch Windows

Wait

Waiting is having the automated task execution elapse a certain amount of time before continuing with the next step. It provides some slack between actions performed, in selenium slack between locating an element or any other operation with the element. The wait is mostly used to handle the *ElementNotVisibleException* exception during load of web page. Waits help a user to troubleshoot issues while re-directing to different web pages by refreshing the entire web page and re-loading the new web elements. At times there can be Ajax calls as well. Thus, a time lag can be seen while reloading the web pages and reflecting the web elements.

Selenium WebDriver provides the user with two types of waits in order to handle the recurring page loads, web element loads, the appearance of windows, pop ups and error messages and reflection of web elements on the web page.

- Implicit wait: An implicit wait makes WebDriver poll the DOM for a certain amount of time when trying to locate an element.
- Explicit wait: An explicit wait makes WebDriver wait for a certain condition to occur before proceeding further with execution.

Implicit Wait

Implicit waits are used to provide a default waiting time between each consecutive test step/command across the entire test script. Thus, subsequent test step would only execute when the 30 seconds have elapsed after executing the previous test step/command.

The implicit wait is a single line of a code and can be declared in the setup method of the test script. However, Implicit wait time increases test script execution time as each of the commands would be ceased to wait for a stipulated amount of time before resuming the execution.

The Syntax of Implicit wait is:

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

The above line of code shall be included into test script soon after instantiation of WebDriver instance variable. The first argument indicates the time in the numeric digits that the system needs to wait. The second argument indicates the time measurement scale in terms of SECONDS, MINUTES, MILISECOND, MICROSECONDS, NANOSECONDS, DAYS, HOURS.

For using implicit waits in the test scripts, it's mandatory to import the following package.

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

Explicit Wait

Explicit waits are used to halt the execution till the time a particular condition is met or the maximum time has elapsed. Unlike Implicit waits, Explicit waits are applied for a particular instance only. The explicit wait is used to tell the Web Driver to wait for certain conditions

(*Expected Conditions*) or the maximum time exceeded before throwing an "ElementNotVisibleException" exception.

The following are the *Expected Conditions* that can be used in Explicit Wait:

- alertIsPresent()
- elementSelectionStateToBe()
- elementToBeClickable()
- elementToBeSelected()
- frameToBeAvailableAndSwitchToIt()
- invisibilityOfTheElementLocated()
- invisibilityOfElementWithText()
- presenceOfAllElementsLocatedBy()
- presenceOfElementLocated()
- textToBePresentInElement()
- textToBePresentInElementLocated()
- textToBePresentInElementValue()
- titleIs()
- titleContains()
- visibilityOf()
- visibilityOfAllElements()
- visibilityOfAllElementsLocatedBy()
- visibilityOfElementLocated()

The Syntax of Explicit wait is:

```
WebDriverWait wait = new WebDriverWait(WebDriverReference, TimeOut);
WebDriverWait wait=new WebDriverWait(driver, 20);
WebElement element =
wait.until(ExpectedConditions.elementToBeClickable(By.id("someid")));
;
```

Example:

```
WebElement Testseleniumlink;
Testseleniumlink=
wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(
"/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/
/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i")));
Testseleniumlink.click();
```

In the above example, wait for the amount of time defined in either the "WebDriverWait" class or the "ExpectedConditions" to occur, whichever occurs first.

The above code states that to wait for an element for the time frame of 20 seconds as defined in the "WebDriverWait" class on the webpage until the "ExpectedConditions" are met and the condition is "visibilityofElementLocated".

Fluent Wait

The fluent wait is used to tell the web driver to wait for a condition, as well as the frequency with which we want to check the condition before throwing an "ElementNotVisibleException" exception. Each FluentWait instance defines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition. A user may configure the wait to ignore specific types of exceptions whilst waiting, such as 'NoSuchElementExceptions' when searching for an element on the page.

The Syntax of Fluent wait is:

```
Wait wait = new FluentWait(WebDriver reference)
    .withTimeout(timeout, SECONDS)
    .pollingEvery(timeout, SECONDS)
    .ignoring(Exception.class);
```

Example:

```
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
    .withTimeout(30, TimeUnit.SECONDS)
    .pollingEvery(5, TimeUnit.SECONDS)
    .ignoring(NoSuchElementException.class);

public WebElement apply(WebDriver driver) {
    return
        driver.findElement(By.xpath("//html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div[1]/div/div/a/i"));
}
```

In the above example, we are declaring a fluent wait with the timeout of 30 seconds and the frequency is set to 5 seconds by ignoring "NoSuchElementException"

A new function is created to identify the Web Element on the page.

Frequency is set to 5 seconds and the maximum time is set to 30 seconds. Thus this means that it will check for the element on the web page at every 5 seconds for the maximum time of 30 seconds. If the element is located within this time frame it will perform the operations else it will throw an "ElementNotVisibleException".

How to Use Expected Conditions with Waits

ExpectedConditions class provides a great help to deal with scenarios where it is ascertain for a condition to occur before executing the actual test step. The explicit wait is used to tell the Web Driver to wait for certain conditions (Expected Conditions) or the maximum time exceeded before throwing an "ElementNotVisibleException" exception.

The following are the Expected Conditions that can be used in Explicit Wait:

- alertIsPresent()
- elementSelectionStateToBe()
- elementToBeClickable()

- elementToBeSelected()
- frameToBeAvailableAndSwitchToIt()
- invisibilityOfTheElementLocated()
- invisibilityOfElementWithText()
- presenceOfAllElementsLocatedBy()
- presenceOfElementLocated()
- textToBePresentInElement()
- textToBePresentInElementLocated()
- textToBePresentInElementValue()
- titleIs()
- titleContains()
- visibilityOf()
- visibilityOfAllElements()
- visibilityOfAllElementsLocatedBy()
- visibilityOfElementLocated()

The Syntax of Explicit wait is:

```
WebDriverWait wait = new WebDriverWait(WebDriverReference, TimeOut);
WebDriverWait wait=new WebDriverWait(driver, 20);
WebElement element =
wait.until(ExpectedConditions.elementToBeClickable(By.id("someid")));
;
```

Example:

```
WebElement Testseleniumlink;
Testseleniumlink=
wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(
"/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/
div[2]/div[2]/div/div/div/div[1]/div/div/a/i")));
Testseleniumlink.click();
```

In the above example, wait for the amount of time defined in either the "WebDriverWait" class or the "ExpectedConditions" to occur, whichever occurs first.

The above code states that to wait for an element for the time frame of 20 seconds as defined in the "WebDriverWait" class on the webpage until the "ExpectedConditions" are met and the condition is "visibilityofElementLocated".

WebDriverWait and its Uses

Waits help the user to troubleshoot issues while re-directing to different web pages by refreshing the entire web page and re-loading the new web elements. WebDriver equips the user to handle the recurring page loads, web element loads, the appearance of windows, pop ups and error messages and reflection of web elements on the web page.

WebDriverWait class can be used in many different cases. Whenever there is a need to perform any operation on element, webdriver wait can be used to check if the element is present or visible or enabled or disabled or clickable.

The various scenarios where to use WebDriverWait are:

- **isElementPresent:** It is used to check for the element presence on the DOM of a page. ExpectedConditions will return true once the element is found in the DOM.
- **isElementClickable:** It is used to check if an element is visible and enabled such that element can be click.
- **isElementVisible:** It is used to check if the element is present and visible on the DOM of a page and visible. Visibility means that the element is not just displayed but also should have a height and width that is greater than 0.
- **isElementInVisible:** It is used to check if an element is either invisible or not present on the DOM.
- **isElementEnabled:** It is used to check if the element is enabled.
- **isElementDisplayed:** It is used to check if the element is displayed or not. It returns false when the element is not present in DOM.

Different Wait Until Conditions

ExpectedConditions class provides a great help to deal with scenarios where it is ascertain for a condition to occur before executing the actual test step. The explicit wait is used to tell the Web Driver to wait for certain conditions (Expected Conditions) or the maximum time exceeded before throwing an "ElementNotVisibleException" exception.

The following are the Expected Conditions that can be used in Explicit Wait:

- alertIsPresent()
- elementSelectionStateToBe()
- elementToBeClickable()
- elementToBeSelected()
- frameToBeAvailableAndSwitchToIt()
- invisibilityOfTheElementLocated()
- invisibilityOfElementWithText()
- presenceOfAllElementsLocatedBy()
- presenceOfElementLocated()
- textToBePresentInElement()
- textToBePresentInElementLocated()
- textToBePresentInElementValue()
- titleIs()
- titleContains()
- visibilityOf()
- visibilityOfAllElements()
- visibilityOfAllElementsLocatedBy()
- visibilityOfElementLocated()

Alerts

Alert is a small message box which displays on-screen notification to give the user information or ask for permission to perform certain kind of operation. It may be also used for warning

purpose. For e.g. a alert pop up window can be displayed if a user clicked on a button that displayed a message or may be when something is entered a form and HTML page requires some extra information.

Alerts are blocking in nature and will not allow any action on the underlying webpage if present. So if an alert is present on the webpage and access any of the element in the underlying page will get UnhandledAlertException exception ie. "Modal dialog present".

There are three broad category of alerts as below:

- **Simple Alert:** The simple alert displays some information or warning on the screen.
- **Prompt Alert:** The prompt Alert asks some input from the user
- **Confirmation Alert:** The confirmation alert asks permission to do some type of operation.

Ways to Handle Simple, Confirmation and Prompt Alert

Selenium provides an interface called Alert and is present in the org.openqa.selenium.Alert package. Alert interface provides following methods to handle alerts:

- void dismiss(): The dismiss() method clicks on the "Cancel" button as soon as the pop up window appears.
- void accept(): The accept() method clicks on the "Ok" button as soon as the pop up window appears.
- String getText(): The getText() method returns the text displayed on the alert box.
- void sendKeys(String stringToSend): The sendKeys() method enters the specified string pattern into the alert box.

Simple alert

Simple alerts have an OK button on them. They are used to display some information to the user. The first alert on our test page is a simple alert.

Example:

Below code will read the text from the Alert and then accept the alert. The driver.switchTo().alert() is used to switch from main window to an alert.

```
public static void main(String[] args) {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://toolsqa.wpengine.com/handling-alerts-using-
    selenium-webdriver/");
    driver.manage().window().maximize();
    // This step will result in an alert on screen
    driver.findElement(By.xpath("//*[@id='content']/p[4]/button")).click();
    Alert simpleAlert = driver.switchTo().alert();
    String alertText = simpleAlert.getText();
    System.out.println("Alert text is " + alertText);
```

```

        simpleAlert.accept();
    }
}

```

Confirmation Alert

Confirmation alert provides an option to accept or dismiss the alert to a user.

Example:

```

public static void main(String[] args) {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://toolsqa.wpengine.com/handling-alerts-using-
selenium-webdriver/");
    driver.manage().window().maximize();
    // This step will result in an alert on screen
    WebElement element =
    driver.findElement(By.xpath("//*[@id='content']/p[11]/button"));
    ((JavascriptExecutor)
    driver).executeScript("arguments[0].click()", element);
    Alert confirmationAlert = driver.switchTo().alert();
    String alertText = confirmationAlert.getText();
    System.out.println("Alert text is " + alertText);
    confirmationAlert.dismiss();
}

```

Prompt Alerts

The prompt alerts provides an option to add text to the alert box to get some input.

Example:

```

public static void main(String[] args) throws
InterruptedException {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://toolsqa.wpengine.com/handling-alerts-using-
selenium-webdriver/");
    driver.manage().window().maximize();
    // This step will result in an alert on screen
    WebElement element =
    driver.findElement(By.xpath("//*[@id='content']/p[16]/button"));
    ((JavascriptExecutor)
    driver).executeScript("arguments[0].click()", element);
}

```

```

Alert promptAlert = driver.switchTo().alert();
String alertText = promptAlert.getText();
System.out.println("Alert text is " + alertText);
//Send some text to the alert
promptAlert.sendKeys("Accepting the alert");
Thread.sleep(4000); //This sleep is not necessary, just for
demonstration
promptAlert.accept();
}

```

Switch Windows

Selenium WebDriver provides `switchTo()` method for switching between windows and frames. A switching to a window refers to an interaction with a new windows that open up when a user clicks on a certain link or button. For example, a new window can open up for login, sign-up button. Alternatively, the switching method is used to handle multiple windows. The Syntax of `switchTo()` method is

```
driver.switchTo().window();
```

Different Between Window Handle and Handles

The Selenium Web Driver assigns an alphanumeric id to each window as soon as the Web Driver object is instantiated. This unique alphanumeric id is called **window handle**. Selenium uses this unique id to switch control among several windows.

- **GetWindowHandle()**: This method is used to get the window handle of the current window. It returns a string of alphanumeric window handle

Example:

```
String handle= driver.getWindowHandle();
```

- **GetWindowHandles()**: This method is used to get the window handle of all the current windows. It returns a set of window handle

Example:

```
Set<String> handle= driver.getWindowHandles();
```

Selenium WebDrives use `switchTo()` method to handle switch between windows by using `GetWindowHandle()` and `GetWindowHandles()` method.

Example of usage of `GetWindowHandle()` to support moving between named windows.

```
driver.switchTo().window("windowName");
```

Alternatively, the a “window handle” can be passed to the “`switchTo().window()`” method to iterate over every open window as below

```
for (String handle : driver.getWindowHandles()) {
    driver.switchTo().window(handle);}
```

Selenium WebDrives use Iterators with windows to iterate over every open window as below:

Example:

```
driver.findElement(By.id("id of the link which opens new
window")).click();

//wait till two windows are not opened
waitForNumberofWindowstoEqual(2); //this method is for wait

Set handles = driver.getWindowHandles();
firstWinHandle = driver.getWindowHandle();
handles.remove(firstWinHandle);

String winHandle=handles.iterator().next();
if (winHandle!=firstWinHandle){

    //To retrieve the handle of second window, extracting the handle
    //which does not match to first window handle
    secondWinHandle=winHandle; //Storing handle of second window handle
    //Switch control to new window
    driver.switchTo().window(secondWinHandle);
```

Switching and Closing Windows, Tabs and Popups

Switching between windows

In order to shift focus from Parent Window to any child window, WebDriver use command WebDriver.SwitchTo().window(String windowHandle). This command takes in a window handle and switches the driver context on other window. Once the Switch happens all the driver commands will go to the newly focused window.

Example:

```
public static void main(String[] args) throws
InterruptedException
{
    WebDriver driver = new FirefoxDriver();
    driver.get("http://toolsqa.wpengine.com/automation-practice-
switch-windows/");
    String parentWindowHandle = driver.getWindowHandle();
    System.out.println("Parent window's handle -> " +
parentWindowHandle);
    WebElement clickElement = driver.findElement(By.id("button1"));
    for(int i = 0; i < 3; i++)
```

```

    }

    clickElement.click();
    Thread.sleep(3000);
}

Set<String> allWindowHandles = driver.getWindowHandles();
for(String handle : allWindowHandles)
{
    System.out.println("Switching to window - > " + handle);
    System.out.println("Navigating to google.com");
    driver.switchTo().window(handle); //Switch to the desired window first and then execute commands using driver
    driver.get("http://google.com");
}
}
}

```

Switching between Tabs

`switchTo()` method is used to switch between frames.

Example:

```

ArrayList<string> tabs = new ArrayList<string>(driver.getWindowHandles());
//Switch to new window
driver.switchTo().window(tabs.get(1));
driver.close(); //do some action in new window(2nd tab)
//Switch to main/parent window
driver.switchTo().window(tabs.get(0));
driver.getTitle(); //do some action in main window(1st tab)
</string></string>

```

Switching between Popups

`switchTo()` method is used to support moving between named PopUps. After action is triggered that opens a popup, the alert can be accessed of the currently open alert object. This object is used to accept, dismiss, read its contents or even type into a prompt.

Example:

```
Alert alert = driver.switchTo().alert();
```

Closing all the Windows

WebDriver use `WebDriver.close()` and `WebDriver.quit()` commands to close the opened browser windows. These commands close the current window on which the focus is present. One can use `switchTo()` command to select the correct windows and then call the `WebDriver.close` command.

The `WebDriver.quit()` closes all the windows opened in the session. This command shuts down the driver instance and any further commands to WebDriver results in exception.

Example:

In the code below the parent window will be close and then explicitly move focus to the last window in the list.

```

public static void main(String[] args) throws
InterruptedException
{
    WebDriver driver = new FirefoxDriver();
    driver.get("http://toolsqa.wpengine.com/automation-
practice-switch-windows/");
    String parentWindowHandle = driver.getWindowHandle();
    System.out.println("Parent window's handle -> " +
parentWindowHandle);
    WebElement clickElement =
driver.findElement(By.id("button1"));
    for(int i = 0; i < 3; i++)
    {
        clickElement.click();
        Thread.sleep(3000);
    }
    Set<String> allWindowHandles = driver.getWindowHandles();
    String lastWindowHandle = "";
    for(String handle : allWindowHandles)
    {
        System.out.println("Switching to window -> " +
handle);
        System.out.println("Navigating to google.com");
        driver.switchTo().window(handle); //Switch to the
desired window first and then execute commands using
driver
        driver.get("http://google.com");
        lastWindowHandle = handle;
    }
    //Switch to the parent window
    driver.switchTo().window(parentWindowHandle);
}

```

```

        //close the parent window
        driver.close();

        //at this point there is no focused window, it have
        //to explicitly switch back to some window.
        driver.switchTo().window(lastWindowHandle);
        driver.get("http://toolsqa.wpengine.com");

    }
}

```

Concepts of Set Interface in Java

A Set is a Collection that can't contain duplicate elements. It models the mathematical set abstraction. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.

The various methods declared by Set are as below:

- `add()`: It adds an object to the collection.
- `clear()`: It removes all objects from the collection.
- `contains()`: It returns true if a specified object is an element within the collection.
- `isEmpty()`: It returns true if the collection has no elements.
- `iterator()`: It returns an Iterator object for the collection, which may be used to retrieve an object.
- `remove()`: It removes a specified object from the collection.
- `size()`: It returns the number of elements in the collection.

Example:

```

import java.util.*;
public class SetDemo {
    public static void main(String args[]) {
        int count[] = {34, 22, 10, 60, 30, 22};
        Set<Integer> set = new HashSet<Integer>();
        try {
            for(int i = 0; i < 5; i++) {
                set.add(count[i]);
            }
            System.out.println(set);

            TreeSet sortedSet = new TreeSet<Integer>(set);
            System.out.println("The sorted list is:");
            System.out.println(sortedSet);
        }
    }
}

```

```

        System.out.println("The First element of the set is: "+
(Integer)sortedSet.first());
        System.out.println("The last element of the set is: "+
(Integer)sortedSet.last());
    }
    catch(Exception e) {}
}
}

```

Output

```

[34, 22, 10, 60, 30]
The sorted list is:
[10, 22, 30, 34, 60]
The First element of the set is: 10
The last element of the set is: 60

```

Concept of Window ID

Selenium WebDriver assigns an alphanumeric id to each window as soon as the WebDriver object is instantiated. This unique alphanumeric id is called window handle. Selenium uses this unique id to switch control among several windows. In simple terms, each unique window has a unique ID, so that Selenium can differentiate when it is switching controls from one window to the other.

Javascript and Selenium

JavaScript is the preferred language inside the browser to interact with HTML. There is no separate code required to execute the Java Script in WebDriver Script as Selenium provides predefined interface *JavascriptExecutor*. Inside this Interface the *executeScript()* method executes the Java Script which is passed as a String. The return values can be Boolean, Long, String, List, WebElement

Syntax:

```

JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript( Script, Arguments );

```

Example:

To click in Selenium if button or radio button is disable:

```

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

```

```
public class TestFirefox {  
    public static void main(String[] args) throws  
        InterruptedException {  
        // Open Firefox browser  
        FirefoxDriver driver=new FirefoxDriver();  
        // Maximize the window  
        driver.manage().window().maximize();  
        // Open application  
        driver.get("enter your application URL");  
        // This will execute JavaScript in your script  
        ((JavascriptExecutor)driver).executeScript("document.getElementById('enter your element id').click();");  
    }  
}
```

DUCAT
www.ducatindia.com

Summary

- Waiting is having the automated task execution elapse a certain amount of time before continuing with the next step
- Implicit waits are used to provide a default waiting time between each consecutive test step/command across the entire test script
- Explicit waits are used to halt the execution till the time a particular condition is met or the maximum time has elapsed
- Waits help the user to troubleshoot issues while re-directing to different web pages by refreshing the entire web page and re-loading the new web elements
- Alert is a small message box which displays on-screen notification to give the user information or ask for permission to perform certain kind of operation
- Selenium WebDriver provides switchto() method for switching between windows and frames

Chapter 10 –Action Class

What is Action Class

The Action class is user-facing API for emulating complex user action events. The Webdriver use the Action Class to perform keyboard events and mouse events such as drag and drop or clicking multiple elements while holding down the Control key. The Action class Implements builder pattern, in which a complex object is constructed that can be further used to create different representation of same object. The action events are found in org.openqa.selenium.interactions.Actions class.

Here is the code snippet for using actions:

- Configure the Action
Actions builder = new Actions(driver);
- To focus on element using webdriver
builder.moveToElement(element).perform();
- To click on the element to focus
builder.moveToElement(element).click().perform();

Perform method is used here to execute the action. The sequence of actions should be minimal.
Example:

```
package automationFramework;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.openqa.selenium.interactions.Actions;
public class mouseHover{
    public static WebDriver driver;
    public static void main(String[] args) {
        driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);

        driver.get("http://www.onlinestore.toolsqa.wpengine.com");

        WebElement element = driver.findElement(By.linkText("Product
Category"));

        Actions action = new Actions(driver);

        action.moveToElement(element).build().perform();

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

```
}
```

What Can We Do With Action Class

Selenium action classes enable to perform advanced activities such as Right-click, Mouse Hover, Pressing any keys on the keyboard, Double click, Click and Hold, Releasing the pressed keys on the keyboard, Scrolling, Drag and Drop. These advanced activities lower the burden of complex codes of the programming languages and allow to just call the methods for the respective activities. To use Actions class, we just need to import the `org.openqa.selenium.interactions.Actions` class

Keyboard Interactions using Actions API

In keyboard interaction, normally we consider events such as scrolling the element and focusing on it. With the help of Action API, keyboard interactions are easier now. In Advanced User Interactions API, absolute interaction with element is possible either by clicking on element or sending a Keys.TAB before sending text.

Actions Class Method for Keyboard Interaction

- `keyDown(Keys modifierKey)`: The `keyDown(Keys modifierKey)` method takes the modifier Keys as parameter (Shift, Alt and Control Keys - that modifies the purpose of other keys, hence the name). It is used to simulate the action of pressing a modifier key, without releasing. The expected values for the `keyDown()` method are - `Keys.SHIFT`, `Keys.ALT` and `Keys.CONTROL` only, passing key other than these results in `IllegalArgumentException`.
- `keyDown(WebElement element, Keys modifierKey)`: This another implementation of `keyDown()` method in which the modifier key press action is performed on a `WebElement`.
- `keyUp(Keys modifierKey)`: The `keyUp()` method is used to simulate the modifier key-up or key-release action. This method follows a preceeding key press action.
- `keyUp(WebElement element, Keys modifierKey)`: This implementation of `keyUp()` method performs the key-release action on a web element.
- `sendKeys(CharSequence KeysToSend)`: The `sendKeys(CharSequence KeysToSend)` method is used to send a sequence of keys to a currently focussed web element. Here, we need to note that it is different from the `webElement.sendKeys()` method. The `Actions.sendKeys(CharSequence KeysToSend)` is particularly helpful when dealing with modifier keys as it doesn't release those keys when passed(resulting in correct

behaviour) unlike the `webElement.sendKeys()` method.

- `sendKeys(WebElement element, CharSequence KeysToSend)`: This implementation of `sendKeys()` method is used to send a sequence of keys to a web element.

Code snippet for Keyboard Actions

- WebElement to which the keyboard actions are performed

```
WebElement textBoxElement = driver.findElement(By Locator of
textBoxElement);
```

- Creating object of Actions class

```
Actions builder = new Actions(driver);
```

- Generating an action to type a text in CAPS

```
Action typeInCAPS = builder.keyDown(textBoxElement, Keys.SHIFT)
    .sendKeys(textBoxElement, "artOfTesting")
    .keyUp(textBoxElement, Keys.SHIFT)
    .build();
```

- Performing the typeInCAPS action

```
typeInCAPS.perform();
```

Mouse Interactions using Actions API

In mouse actions we use current location of the element as a standard. First action will be relative to the location of the standard element, the next action will be relative to the location of the mouse at the end of the last action, etc.

Actions Class Method for Mouse Interactions

- `click()`: This method is used to click at the current mouse pointer position. It is particularly useful when used with other mouse and keyboard events, generating composite actions.
- `click(WebElement webElement)`: This method is used to click at the middle of a web element passed as parameter to the `click()` method.

- `clickAndHold()`: The `clickAndHold()` method is used to perform the `click` method without releasing the mouse button.
- `clickAndHold(WebElement onElement)`: This method performs the `click` method without releasing the mouse button over a web element.
- `contextClick()`: This method is used to perform the right click operation(context-click) at the current mouse position.
- `contextClick(WebElement onElement)`: This method performs the right click operation at a particular web element.
- `doubleClick()`: This method performs double click operation at a current mouse position.
- `doubleClick(WebElement onElement)`: Performs the double click operation at a particular web element.
- `dragAndDrop(WebElement fromElement, WebElement toElement)`: This is a utility method to perform the `dragAndDrop` operation directly wherein, we can pass the source element and the target element as parameter.
- `dragAndDropBy(WebElement fromElement, int xOffset, int yOffset)`: This method is a variation of `dragAndDrop(fromElement, toElement)` in which instead of passing the target element as parameter, we pass the x and y offsets. The method clicks the source web element and then releases at the x and y offsets.
- `moveByOffset(int xOffset, int yOffset)`: This method is used to move the mouse pointer to a particular position based on the x and y offsets passed as parameter.
- `moveToElement(WebElement toElement)`: This method is used to move the mouse pointer to a web element passed as parameter.
- `moveToElement(WebElement toElement, int xOffset, int yOffset)`: This method moves the mouse pointer by the given x and y offsets from the top-left corner of the specified web element.
- `release()`: This method releases the pressed left mouse button at the current mouse pointer position.
- `release(WebElement onElement)`: This method release the pressed left mouse button at a particular web element.

Code snippet for a Mouse Action

- `WebElement` which needs to be right clicked

```
WebElement rtClickElement = driver.findElement(By Locator of
rtClickElement);
```

- Generating an Action to perform context click or right click

```
Actions rightClickAction = new
Actions(driver).contextClick(rtClickElement);
```

- Performing the right click Action generated

```
rightClickAction.build().perform();
```

- If you want to handle drag and drop event, then you can use following code

```
Actions builder = new Actions(driver);
WebElement dragElement=driver.findElement(By.id("draggable"));
WebElement dropElement=driver.findElement(By.id("droppable"));
```

- To hold the drag element and then move it to drop element location

```
Action dragDrop =
builder.clickAndHold(dragElement).moveToElement(dropElement).bu
ild();
```

- To Execute the drag and drop Action

```
dragDrop.perform();
```

- If you want to click on menu option in mouse hover menu, then follow this code:

```
Actions builder = new Actions(driver);
WebElement menu = driver.findElement(By.id("menu"));
WebElement menuoption =
driver.findElement(By.id("menuoption"));
```

- Move to the main menu option and then sub-option

```
Action element=
builder.moveToElement(menu).moveToElement(menuoption).build();
```

- To Execute the Action

```
element.perform();
```

It's evident from the above examples that using "Actions API" in Webdriver simplifies input interactions. In fact some events can be only handled by Action API e.g. Drag and drop.

Code snippet to right click an element

```
Actions action = new Actions(driver);
WebElement element = driver.findElement(By.id("elementId"));
action.contextClick(element).perform();
```

Here, it is instantiating an object of Actions class. After that, it pass the WebElement to be right clicked as parameter to the *contextClick()* method present in the Actions class. Then, it call the *perform()* method to perform the generated action.

Sample code to right click an element

For the demonstration of the right click action, it will be launching Sample Site for Selenium Learning. Then on right-click a textbox, its context menu will get displayed, asserting that right click action is successfully performed.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;

public class RightClick {

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

    //Launching Sample site

    driver.get("http://artoftesting.com/sampleSiteForSelenium.html");
}

    //Right click in the TextBox
    Actions action = new Actions(driver);
    WebElement searchBox =
driver.findElement(By.id("fname"));
    action.contextClick(searchBox).perform();

    //Thread.sleep just for user to notice the event
    Thread.sleep(3000);
```

```

        //closing the driver instance
        driver.quit();
    }
}

```

Code snippet to double click an element

```

Actions action = new Actions(driver);
WebElement element = driver.findElement(By.id("elementId"));
action.doubleClick(element).perform();

```

Here, it is instantiating an object of Actions class. After that, it pass the WebElement to be double clicked as parameter to the doubleClick() method present in the Actions class. Then, call the perform() method to perform the generated action.

Sample code to double click an element

For the demonstration of the double click action, it will be launching Sample Site for Selenium Learning. Then it will double click the button on which the text "Double-click to generate alert box" is written. After that an alet box will appear, asserting that double-click action is successfully performed.

```

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;

public class DoubleClick {

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

    //Launching sample website
    driver.get("http://artoftesting.com/sampleSiteForSelenium.
html");
    driver.manage().window().maximize();
}

```

```

        //Double click the button to launch an alertbox
        Actions action = new Actions(driver);
        WebElement btn = driver.findElement(By.id("dblClkBtn"));
        action.doubleClick(btn).perform();

        //Thread.sleep just for user to notice the event
        Thread.sleep(3000);

        //Closing the driver instance
        driver.quit();
    }

}

```

Example to show how to use Actions class with doubleClick() and perform() methods to double click on element.

```

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

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class DoubleClick {
    WebDriver driver;
    @BeforeTest
    public void setup() throws Exception {
        driver =new FirefoxDriver();
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("http://only-testing-
blog.blogspot.in/2014/09/selectable.html");
    }
}

```

```

}

@Test
public void doubleClick_Button() throws IOException,
InterruptedException {
    WebElement ele =
driver.findElement(By.xpath("//button[contains(., 'Double-Click Me To
See Alert')]"));
    //To generate double click action on "Double-Click Me To See
Alert" button.
    Actions action = new Actions(driver);
    action.doubleClick(ele);
    action.perform();

    Thread.sleep(3000);
    String alert_message = driver.switchTo().alert().getText();
    driver.switchTo().alert().accept();
    System.out.println(alert_message);
}
}

```

On run this code, It will double click on "Double-Click Me To See Alert". Rest of code Is to handle alert generated by button double click. So this Is the way to double click on any element In selenium webdriver test.

Mouse Hover and Mouse Movement with Action

In order to perform mouse hover actions, all of the actions need to be chain which to achieve in one go. The WebDriver need to be move to the parent element that has child elements and click on the child element. With the object of the Actions class, Webdriver moves to the main menu and then to the sub menu and click on it.

Few of the scenarios are:

- **Mouse hover actions on an element using Actions Class:**

Example:

```
WebElementele = driver.findElement(By.xpath("xpath"));
```

```
//Create object 'action' of an Actions class
Actions action = new Actions(driver);
//Mouseover on an element
action.moveToElement(ele).perform();
```

- **Mouse hover actions on a sub-element using Actions Class:**

On click on the sub-element, first there is need to mouse hover on the parent-element and then sub-element and click on it.

Example:

```
//Main Menu
WebElementmainMenu =
driver.findElement(By.linkText("main_menu_link"));
//Create object 'action' of an Actions class
Actions actions = new Actions(driver);
//To mouseover on main menu
actions.moveToElement(mainMenu);
//Sub Menu
WebElementsubMenu =
driver.findElement(By.linkText("sub_menu_link"));
//To mouseover on sub menu
actions.moveToElement(subMenu);
//build() method is used to compile all the actions into a single
step
actions.click().build().perform();
```

Example to Handle Mouseover Action Using Selenium Actions Class:

```
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;

public class MouseHoverExample {

    public static void main(String[] args) throws Exception {
```

```

// Initialize WebDriver
WebDriver driver = new FirefoxDriver();

// Wait For Page To Load
driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);

// Go to URL
driver.get("http://www.myntra.com/");

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

// Mouse Over On " Men link "
Actions act = new Actions(driver);
By testlink = By.linkText("Men");
WebElement test = driver.findElement(testlink);
act.moveToElement(test).build().perform();

// Click on " bags & backpacks " link
driver.findElement(By.linkText("Bags & Backpacks")).click();

// Click on the categories - Bag-packs

driver.findElement(By.xpath("//*[text()='Categories']/following::li[1]/label")).click();

// Mouse Hover on the 1st bag
Actions sel = new Actions(driver);

sel.moveToElement(driver.findElement(By.xpath("//ul[@class='results small']/li[1]"))).build().perform();

// Click on the "Add to Bag" icon of the 1st bag
driver.findElement(By.xpath("//ul[@class='results small']/li[1]/div[1]//div")).click();

```

```

        // Hover over the shopping bag icon present on the top
navigation bar

        Actions mov = new Actions(driver);

mov.moveToElement(driver.findElement(By.xpath("//a[contains(@class,
'cart')]//div"))).click().build().perform();

        // Click on the remove icon
        driver.findElement(By.xpath("//span[@data-hint='REMOVE FROM
BAG'])[1]").click();

        // Closing current driver window
        driver.close();
    }
}

```

Finding Coordinates of a Web Object

Any software web element has its own position on page and generally it is measured in x and y pixels and known as x y coordinates of element. x pixels means horizontal position on page from left side and y pixels means vertical position on software web page from top side. In selenium WebDriver software testing tool, x y coordinates of element can be get by using Point class.

Example:

```

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.Point;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;
public class xyCoordinates {
    WebDriver driver;
    public void setup() throws Exception {
        driver =new FirefoxDriver();

```

```

driver.manage().window().maximize();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("http://only-testing-
blog.blogspot.in/2014/09/selectable.html");
}

public void getCoordinates() throws Exception {
    //Locate element for which you wants to retrieve x y coordinates.
    WebElement Image =
    driver.findElement(By.xpath("//img[@border='0']"));

    //Used points class to get x and y coordinates of element.
    Point point = Image.getLocation();®
    int xcord = point.getX();
    System.out.println("Element's Position from left side Is "+xcord +" pixels.");
    int ycord = point.getY();
    System.out.println("Element's Position from top side Is "+ycord +" pixels.");
}
}

```

Output:

```

Element's Position from left side Is 76 pixels.
Element's Position from top side Is 740 pixels.

```

Drag and Drop Action

In some applications, a situation may be face to automate drag and drop an item from one location to another location. These could not be achieve by using basic elements. The Actions class has two methods that support Drag and Drop.

- **dragAndDrop:**

Syntax:

```
Actions.dragAndDrop(Source locator, Destination locator)
```

First parameter "Source locator" is the element which need to drag
Second parameter "Destination locator" is the element on which need to drop the first element

- **dragAndDropBy:**

Syntax:

Actions.dragAndDropBy(Sourcelocator, x-axis pixel of Destinationlocator, y-axis pixel of Destinationlocator)

First parameter "Sourcelocator" is the element which need to drag
 The second parameter is x-axis pixel value of the 2nd element on which need to drop the first element.
 The third parameter is y-axis pixel value of the 2nd element on which need to drop the first element

Example:

```

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.interactions.Actions;
import org.testng.annotations.Test;

public class DragAndDrop {

    WebDriver driver;

    @Test
    public void DragnDrop()
    {
        System.setProperty("webdriver.chrome.driver",
        "E://Selenium//Selenium_Jars//chromedriver.exe ");

        driver= new ChromeDriver();
        driver.get("http://www.ducatindia.com/test/drag_drop.html");

        //Element which needs to drag.
        WebElement From=driver.findElement(By.xpath("//*[@id='credit2']/a"));

        //Element on which need to drop.
        WebElement To=driver.findElement(By.xpath("//*[@id='bank']/li"));

        //Using Action class for drag and drop.
        Actions act=new Actions(driver);

        //Dragged and dropped.
        act.dragAndDrop(From, To).build().perform();
    }
}

```

In the above code we launch the given URL in Firefox browser and then drag the BANK element and drop on the DEBIT SIDE block through dragAndDropmethod as explained below

- First, we capture the 1st element which we need to drag in variable "From."

```
WebElement
```

```
From=driver.findElement(By.xpath("//*[@id='credit2']/a"));
```

- Second, we capture the 2nd element on which we need to drop the 1st element in variable "To".

```
WebElement
```

```
To=driver.findElement(By.xpath("//*[@id='bank']/li"));
```

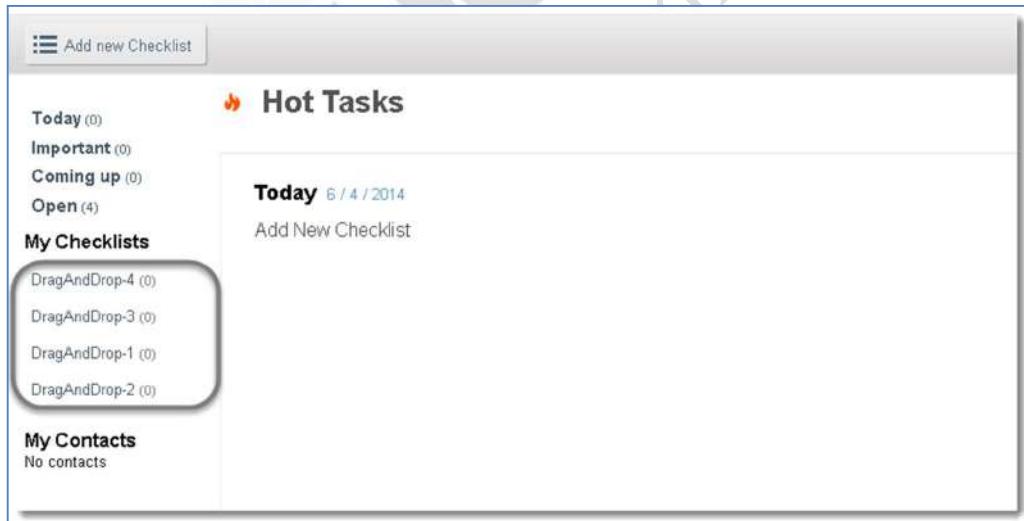
- Third, we create object of Actions class as we use methods of Actions class.

```
Actions act=new Actions(driver);
```

- For drag and drop element dragAndDrop method of Actions class is used and passes the parameters as the first element(Sourcelocator) "From" and the second element(Destinationlocator) "To". Below line will drag the 1st element and drop it on the 2nd element.

```
act.dragAndDrop(From, To).build().perform();
```

Example: There is a Checklist on the left side and there are four sub menus with the name of DragAndDrop-[1-3]. We will login to this website and then drag DragAndDrop-1 to DragAndDrop-4 of the left side sub menu.



```
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
```

```

import org.openqa.selenium.interactions.Action;
import org.openqa.selenium.interactions.Actions;
public class DragAndDrop {
    public static void main(String[] args) throws
InterruptedException {
        WebDriver driver = new FirefoxDriver();
        String URL = "http://sandbox.checklist.com/account/";
        driver.get(URL);
        driver.findElement(By.name("j_username")).sendKeys("Username");
        driver.findElement(By.name("j_password")).sendKeys("Password");
        driver.findElement(By.name("login")).click();
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(10000,
        TimeUnit.MILLISECONDS);
        WebElement From =
        driver.findElement(By.xpath(".//*[@id='userChecklists']"
        "/li[1]/a/span"));
        WebElement To =
        driver.findElement(By.xpath(".//*[@id='userChecklists']/li[4]/a/span"));
        Actions builder = new Actions(driver);
        Action dragAndDrop = builder.clickAndHold(From)
            .moveToElement(To)
            .release(To)
            .build();
        dragAndDrop.perform();
    }
}

```

Robot Class

Sometime in testing process it is required to control mouse and keyboard to connect with windows. For example to handle pop-ups, file upload and download. These windows applications are not handled by Selenium Webdriver.

To handle these pop-ups there is a Robot Class which interacts with OS pop/ups.

Role of Robot Class

- To simulate Mouse and Keyboard events
- Support selenium Webdriver to download or upload a file.

Initialization in scripts

Robot Class is part of java.awt.package.

There are two packages required to import for Robot Class.

```
import java.awt.AWTException;
import java.awt.Robot;
```

Create object of Robot Class

```
Robot robot= new Robot();
```

Popular methods used in Robot class are

- .keyPress();- This is used to press any key.

For example-`robot.keyPress(KeyEvent.VK_UP)` : It will press UP key in the keyboard.

- .keyRelease();- This is used to release the press key of keyboard.

For example-`robot.keyRelease(KeyEvent.VK_CAPS_LOCK)` ; :It will release the pressed capslock key in the keyboard.

- .mousePress();-This is used to press the left button of mouse.

For Example-`robot.mousePress(InputEvent.BUTTON1_MASK)` ;

- .mouseRelease();- This is used to release the pressed button of mouse.

For Example-`robot.mouseRelease(InputEvent.BUTTON1_MASK)` ;

- .mouseMove(); This will move the mouse pointer to the X and Y co-ordinates. Co-ordinates of elements are passed in mouseMove();.

```
robot.mouseMove(coordinates.getX(), coordinates.getY());
```

For example, if the task is to type 'A' (caps+a) using keyboard event.

The code will be like

```
Robot robot= new Robot();
```

```
//press CTRL+a  
  
robot.keyPress(KeyEvent.VK_CONTROL);  
  
Robot.keyPress(KeyEvent.VK_a);  
  
//release CTRL+a  
  
robot.keyPress(KeyEvent.VK_CONTROL);  
  
Robot.keyPress(KeyEvent.VK_a);
```

File upload using Robot Class

Common steps to upload the file using Robot class are

1. Copy the path of file which is to be uploaded.
2. Click on upload button.
3. Paste the path using Control+V and press enter.

Disadvantages of Robot Class

There are few disadvantages of Robot framework

- Mouse or keyboard event will only works on current instance of window.it is difficult to switch among different screen or windows. For example- A code is executing any robot event but the code execution is moved to other window, in this case Mouse or keyboard event will still remain on previous window.
- Some methods like mouseMove() is also depends on screen resolutions. So it might be happen that code will not work on other machine.

Summary

- The Webdriver use the Action Class to perform keyboard events and mouse events such as drag and drop or clicking multiple elements while holding down the Control key
- Selenium action classes enable to perform advanced activities such as Right-click, Mouse Hover, Pressing any keys on the keyboard, Double click, Click and Hold, Releasing the pressed keys on the keyboard, Scrolling, Drag and Drop
- Any software web element has its own position on page and generally it is measured in x and y pixels and known as x y coordinates of element

DUCAT®
www.ducatindia.com

Chapter 11–TestNG Framework

What is TestNG?

TestNG is an open source automated testing framework; where NG of TestNG means Next Generation. TestNG is similar to JUnit (a unit testing framework for Java programming language) but it is much more powerful than JUnit but still it's inspired by JUnit. It is designed to be better than JUnit, especially when testing integrated classes. Cedric Beust is the creator of TestNG.

TestNG eliminates most of the limitations of the older framework and gives the developer the ability to write more flexible and powerful tests with help of easy annotations, grouping, sequencing & parameterizing.

Features of TestNG

The features of TestNG are:

- Supports annotations.
- TestNG uses more Java and object oriented features.
- Supports testing integrated classes
- Separates compile-time test code from run-time configuration/data info.
- Flexible runtime configuration.
- Supports Dependent test methods, parallel testing, load testing, and partial failure.
- Flexible plug-in API.
- Support for multi-threaded testing.

Benefits of TestNG

There are number of benefits but from Selenium perspective, major advantages of TestNG are :

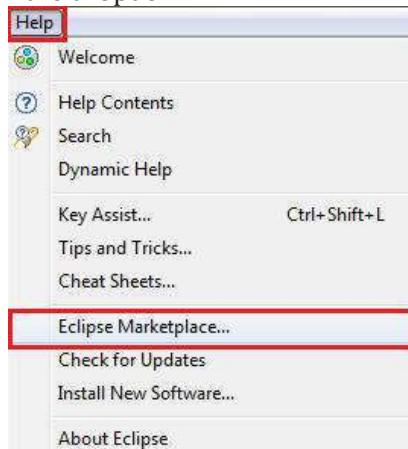
- **Simplified annotations:** Annotations have been simplified, making it easier for testers to understand them.
- **Results in HTML:** It generates reports in HTML.
- **Multiple test-cases at once:** Using TestNG Suite (testing.xml), which is essentially an xml file, testers can execute a number of test cases at a time. In this xml file, users need to mention the number of classes and test-cases they wish to execute.
- **Runs failed test-cases:** TestNG can also be used to run failed test-cases. This is one of most important advantages of TestNG over JUnit.
- **Allows grouping:** TestNG allows supports grouping. Using this feature, testers can group test-cases without too much effort, which was not possible with JUnit.
- **Allows running on multiple browsers:** TestNG allows testers to execute and run one script in multiple browsers.

- **Parametric testing:** Most of the time, testers have to execute a large number of varied tests, mainly due to the nature of their business logic. However, this lengthy process can be eliminated as parametric testing allows to run the same test a number of times by simply changing the values. It allows to pass parameters to the test methods in two ways: parameter and @dataProvider.
- **Bypassing or ignoring test-cases:** This feature is useful if a particular testcase(s) is not execute. In such instances, TestNG, with the help of annotation @Test(enabled = false), allows to disable or bypass the particular test-case(s).
- **Reporter class (it generates logs):** In TestNG, with the help of the reporter class, users can log messages for the test. This could be just surface or in-depth information, depending on the need
- **Expected exceptions:** It allows to trace the exception handling of the code. While writing a code, there may be situations where testers want to verify if an exception is being presented when executed. This method will give the details of the exceptions that are expected to be presented by that particular method. Use this method with @Test annotation.
- **Dependent on Method :** TestNG supports the ‘dependence’ method. It can have dependence as an attribute of a method, for example, if one method is dependent on another. This is not available in JUnit.
- **TestCase priority:** The test cases can be execute in a particular order. To accomplish this, define the order in @Test Annotation.

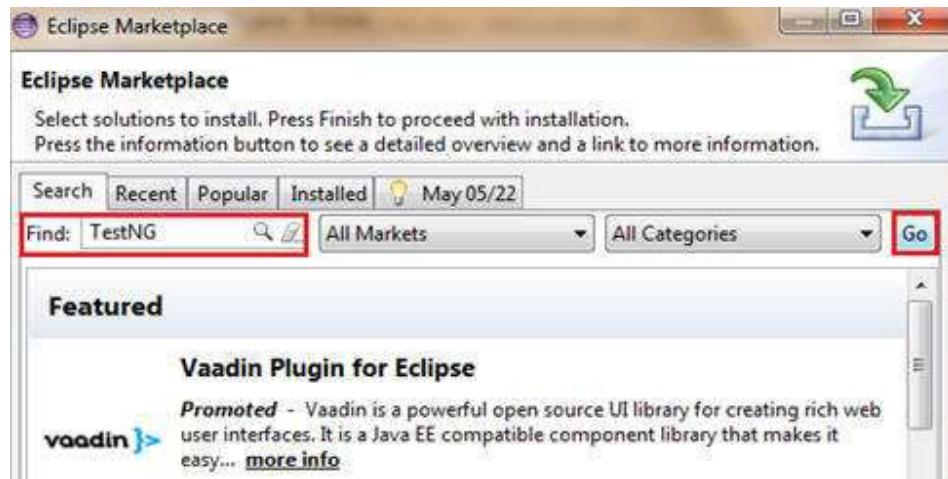
TestNG Installation in Eclipse

The steps to Download and install TestNG on eclipse are:

1. Launch eclipse IDE -> Click on the Help option within the menu -> Select “Eclipse Marketplace..” option within the dropdown.



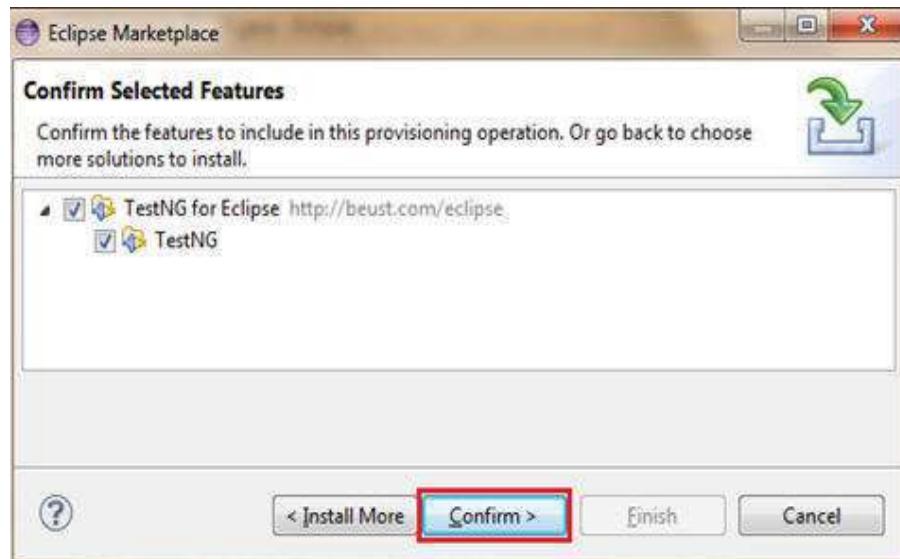
2. Enter the keyword “TestNG” in the search textbox and click on “Go” button as shown below.



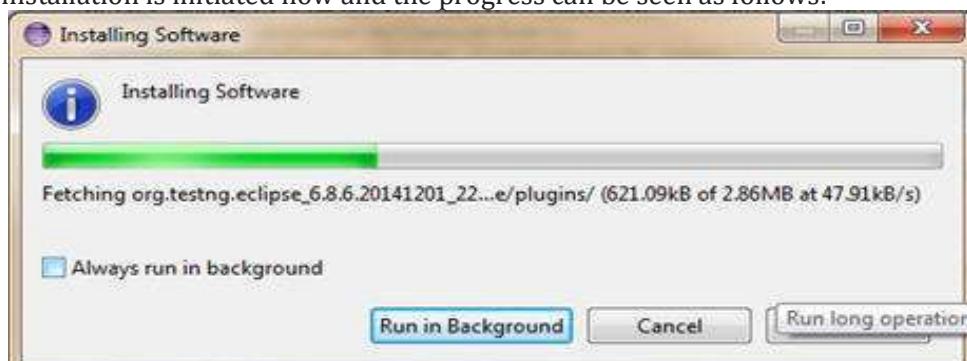
- Click on the “Go” button, the results matching to the search string would be displayed. Now click on the Install button to install TestNG.



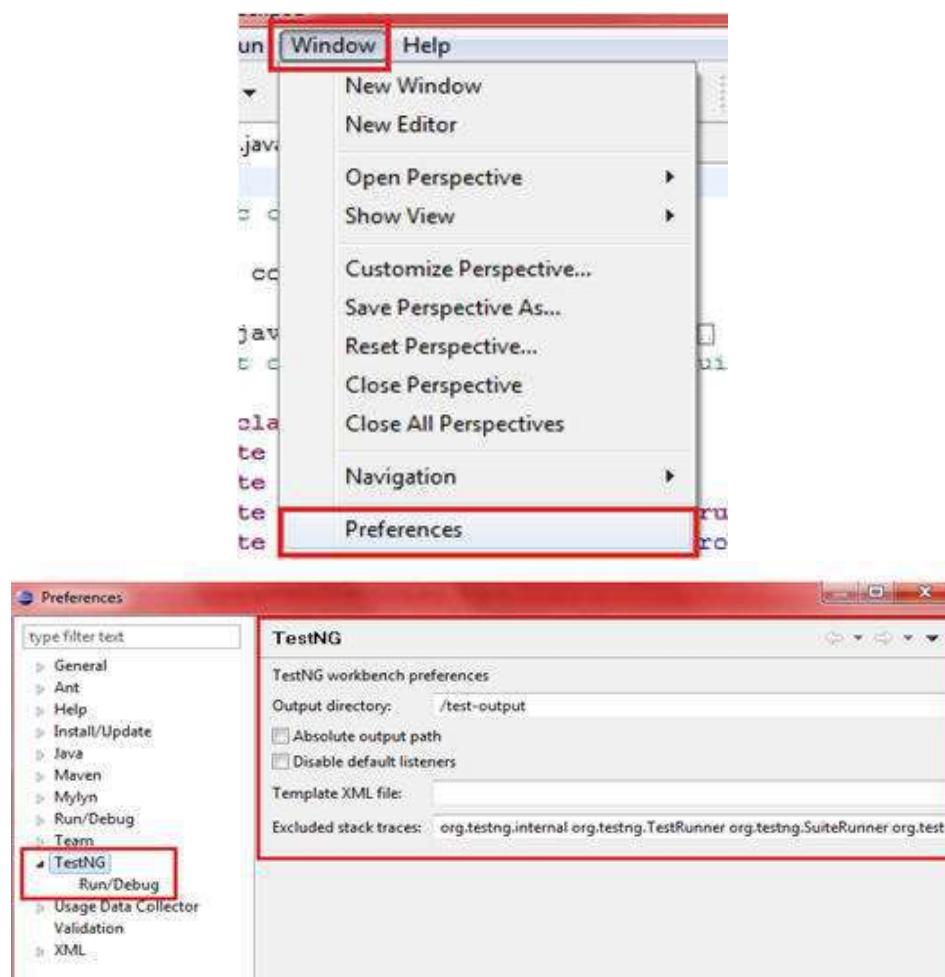
- Click on the Install button, you will get prompt with a window to confirm the installation. Click on “Confirm” button.



5. In the next step, the application would prompt you to accept the license and then click on the "Finish" button.
6. The installation is initiated now and the progress can be seen as follows:

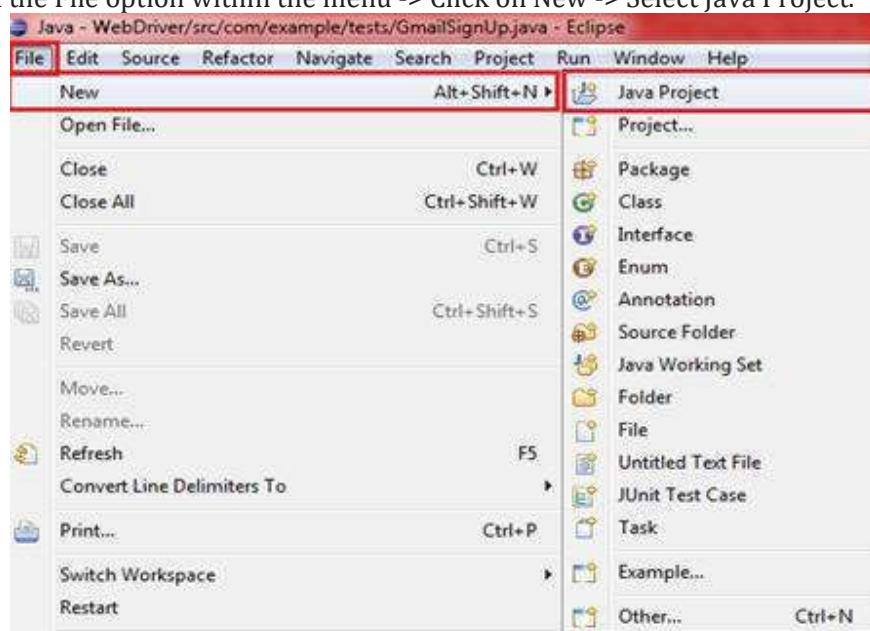


7. Restart eclipse so as to reflect the changes made.
8. Upon restart, verify the TestNG installation by navigating to "Preferences" from "Window" option in the menu bar. Refer the following figure for the same.

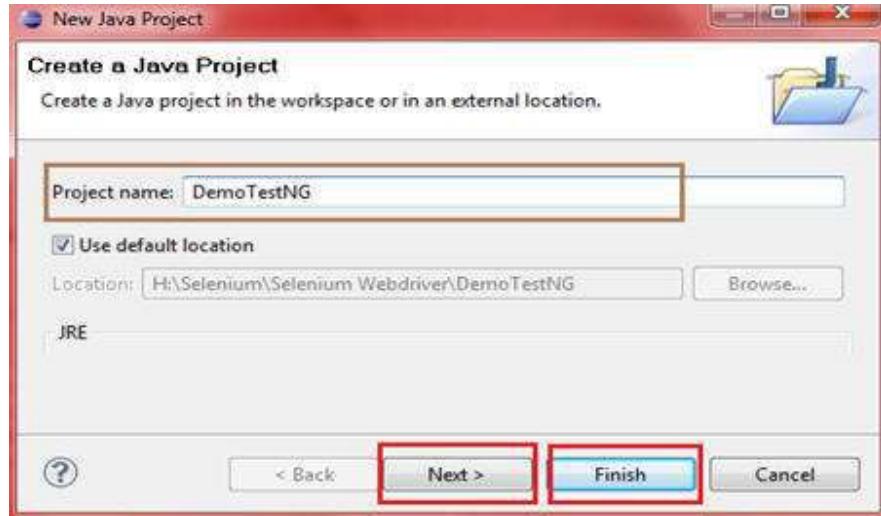


Creation of Sample TestNG project

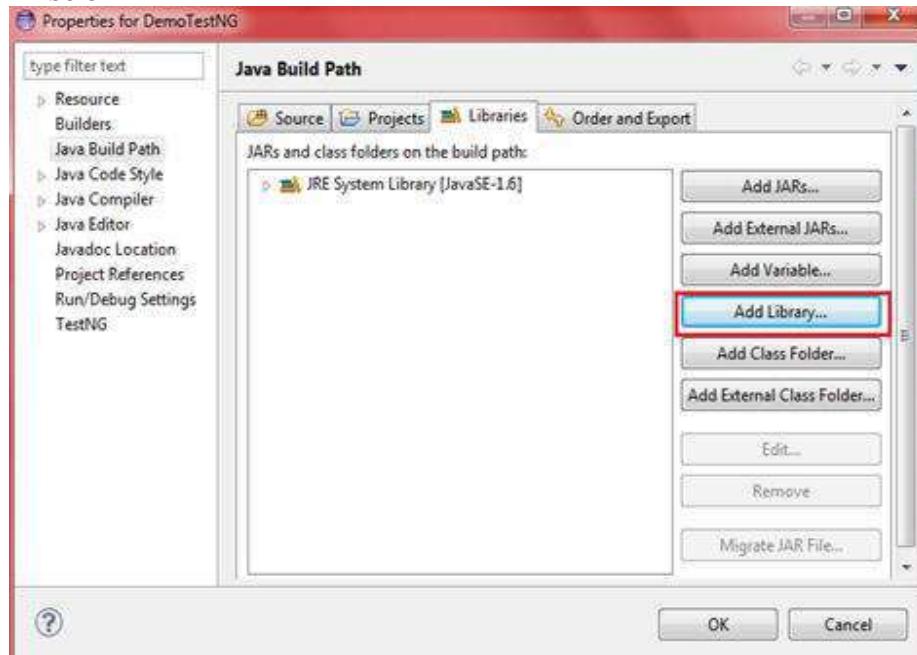
1. Click on the File option within the menu -> Click on New -> Select Java Project.



- Enter the project name as “DemoTestNG” and click on “Next” button. As a concluding step, click on the “Finish” button and your Java project is ready.



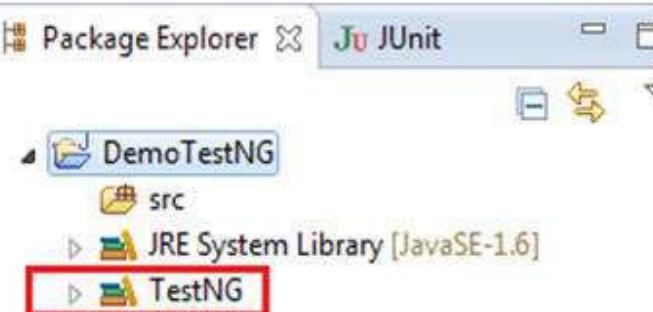
- The next step is to configure the TestNG library into the newly created Java project. For the same, click on the “Libraries” tab under Configure Build Path. Click on “Add library” as shown below.



- Select TestNG and click on the “Next” button as shown below in the image. In the end, click on the “Finish” button.

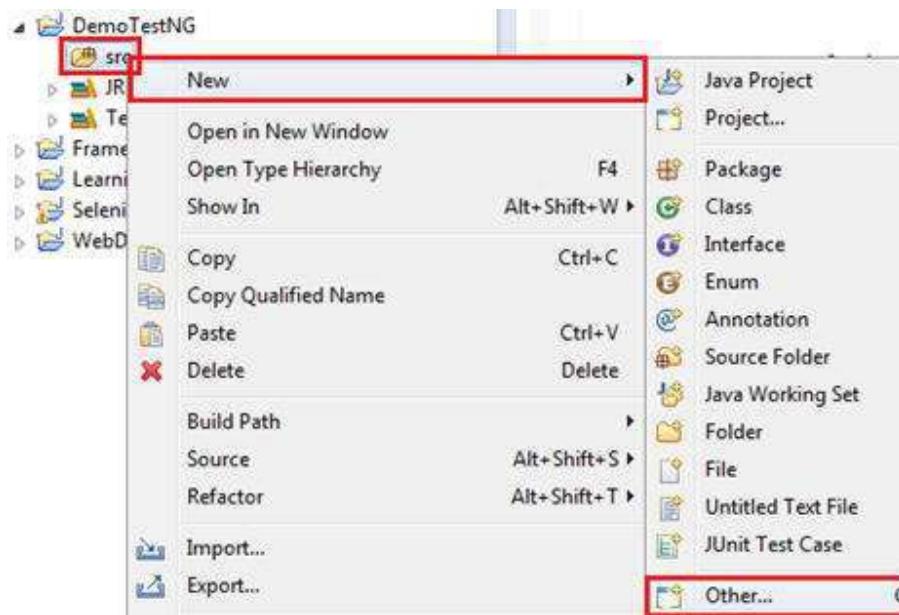


- The TestNG is now added to the Java project and the required libraries can be seen in the package explorer upon expanding the project.

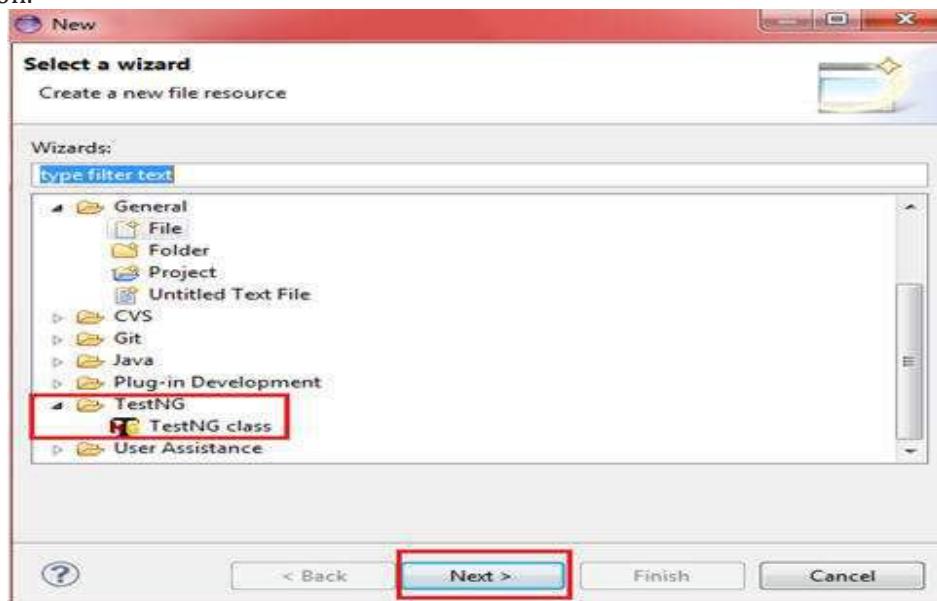


Creating TestNG class

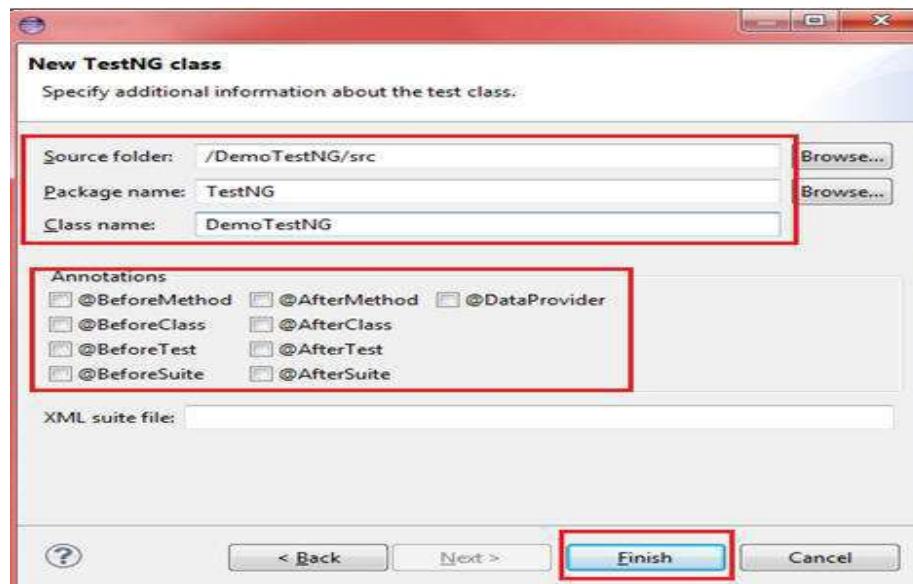
- Expand the "DemoTestNG" project and traverse to "src" folder. Right-click on the "src" package and navigate to New -> Other.



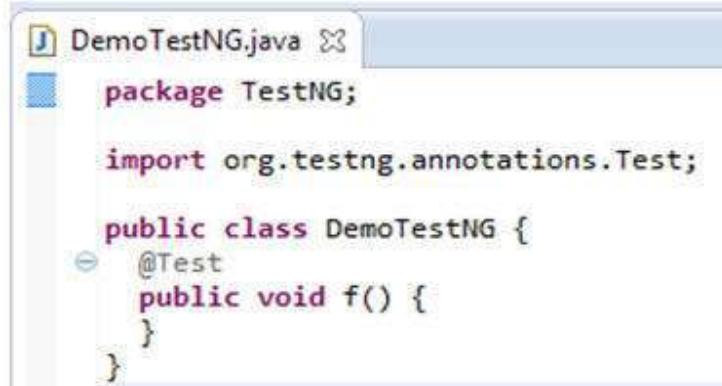
2. Expand TestNG option and select “TestNG” class option and click on the “Next” button.



3. Furnish the required details as following. Specify the Source folder, package name and the TestNG class name and click on the Finish button.



4. The above mentioned TestNG class would be created with the default schema.



```

package TestNG;

import org.testng.annotations.Test;

public class DemoTestNG {
    @Test
    public void f() {
    }
}

```

Test Scenario

Consider a test scenario where we:

- Launch the browser and open “gmail.com”.
- Verify the title of the page and print the verification result.
- Enter the username and Password.
- Click on the Sign in button.
- Close the web browser.

Code:

```

package TestNG;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

```

```

import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.Test;

public class DemoTestNG {

    public WebDriver driver = new FirefoxDriver();

    String appUrl = "https://accounts.google.com";

    @Test

    public void gmailLogin() {

        // launch the firefox browser and open the application url

        driver.get("https://gmail.com");

        // maximize the browser window

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

        // declare and initialize the variable to store the expected
        // title //of the webpage.

        String expectedTitle = "Sign in - Google
        Accounts ";

        // fetch the title of the web page and save it into a string
        // variable

        String actualTitle = driver.getTitle();

        Assert.assertEquals(expectedTitle,actualTitle);

        // enter a valid username in the email textbox

        WebElement
        username=driver.findElement(By.id("Email"));

        username.clear();

        username.sendKeys("TestSelenium");

        // enter a valid password in the password textbox
    }
}

```

```

        WebElement password =
        driver.findElement(By.id("Passwd"));

        password.clear();

        password.sendKeys("password123");

    // click on the Sign in button

        WebElement SignInButton =
        driver.findElement(By.id("signIn"));

        SignInButton.click();

    // close the web browser

        driver.close();

    }

}

```

Code Explanation with respect to TestNG

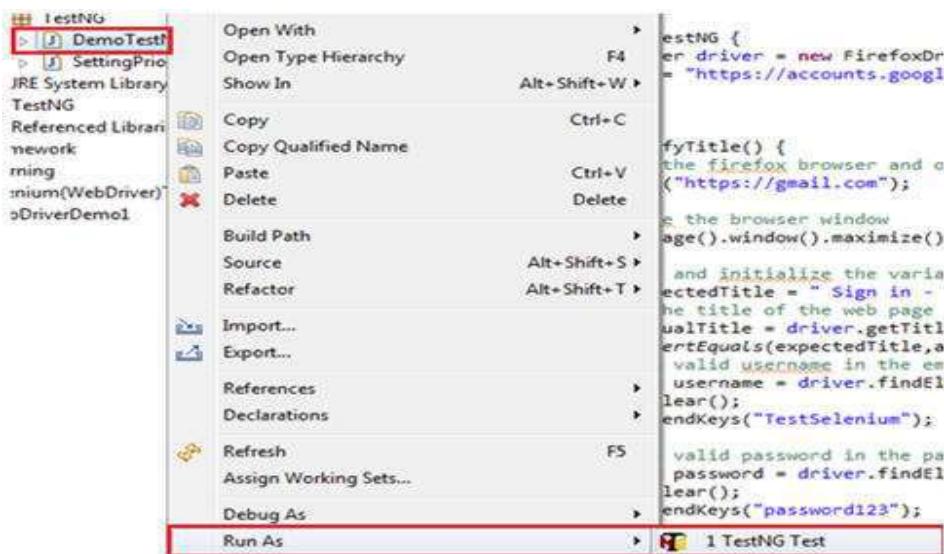
@Test: @Test is one of the **TestNG annotations**. This annotation lets the program execution to know that method annotated as @Test is a test method. To be able to use different TestNG annotations, we need to import the package “**import org.testng.annotations.***”.

There is no need of main() method while creating test scripts using TestNG. The program execution is done on the basis of annotations.

In a statement, we used Assert class while comparing expected and the actual value. Assert class is used to perform various verifications. To be able to use different assertions, we are required to import “**import org.testng.Assert**”.

Executing the TestNG script

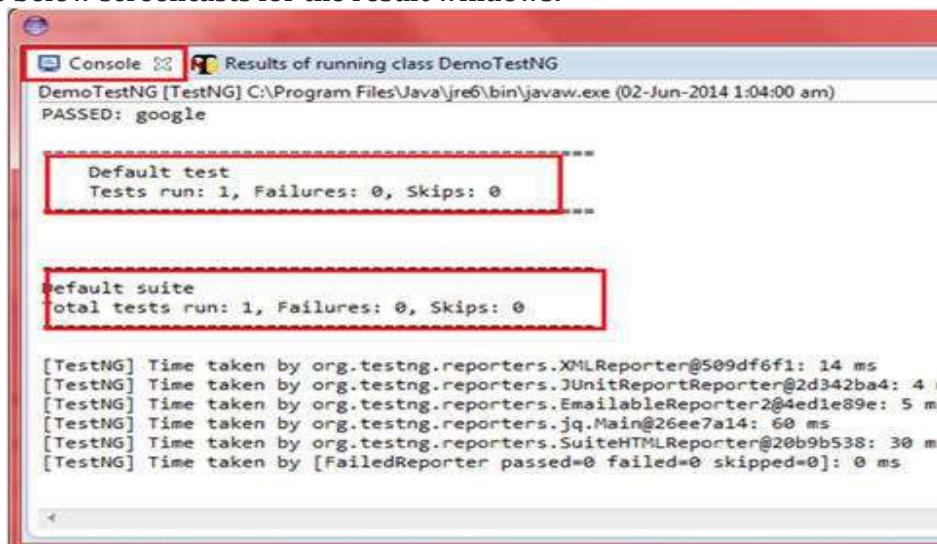
1. Right click anywhere inside the class within the editor or the java class within the package explorer, select “Run As” option and click on the “TestNG Test”.

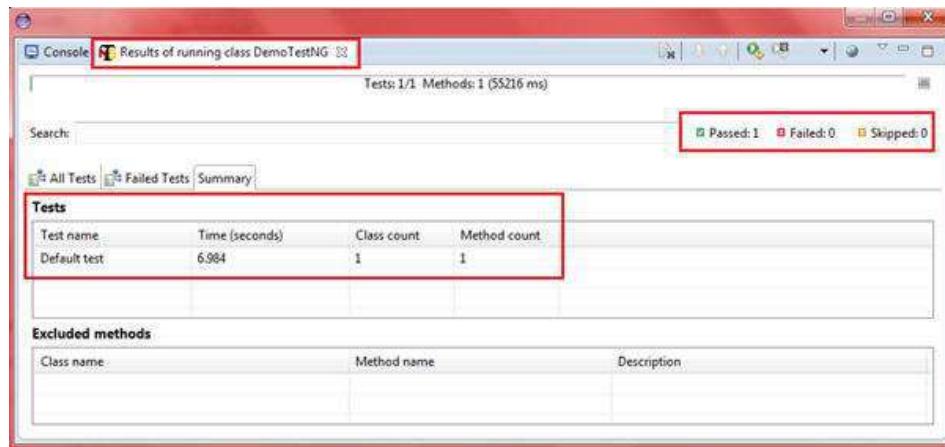


TestNG result is displayed into two windows:

- Console Window
- TestNG Result Window

Refer the below screencasts for the result windows:



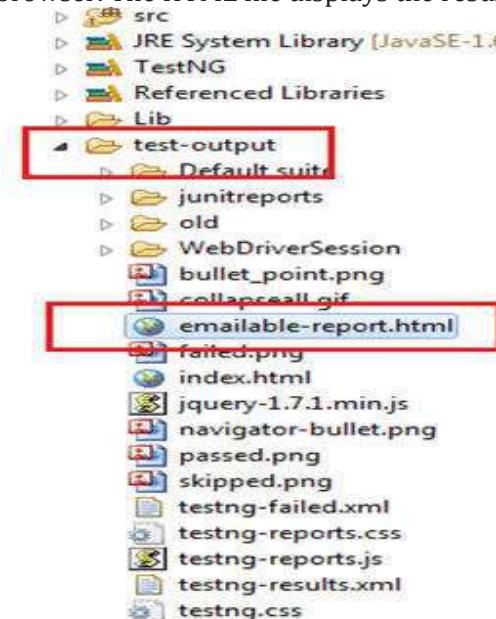


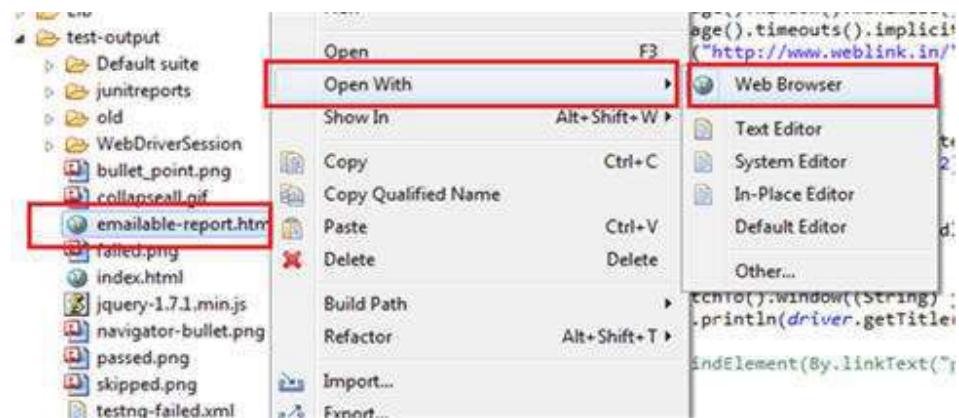
HTML Reports

TestNG comes with a great capability of generating user readable and comprehensible HTML reports for the test executions. These reports can be viewed in any of the browsers and it can also be viewed using Eclipse's build -in browser support.

To generate the HTML report, follow the below steps:

1. Execute the newly created TestNG class. Refresh the project containing the TestNG class by right-clicking on it and selecting "Refresh" option.
2. A folder named as "test-output" shall be generated in the project at the "src" folder level. Expand the "test-output" folder and open on the "emailable-report.html" file with the Eclipse browser. The HTML file displays the result of the recent execution.





3. The HTML report shall be opened within the eclipse environment. Refer the below image for the same.



The screenshot shows the Eclipse IDE interface with the 'TestNG Report' perspective selected. The title bar includes tabs for 'DemoTestNG.java', 'SettingPriority.java', 'DemoTestNG.java', and 'TestNG Report'. The 'TestNG Report' tab is active, displaying a URL 'file:///H:/WebDriver/Selenium(WebDriver)Training/test-output/emailable-report.html'. The main content area shows two tables:

Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups	Excluded
Default suite						
Default test	1	0	0	55,060		

Class	Method	Start	Time (ms)
Default suite			
Default test — passed			
advanced.day3.DemoTestNG	google	1401651289219	6984

Default test

advanced.day3.DemoTestNG#google

[back to summary](#)

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.

Here is the list of annotations that TestNG supports

Annotation & Description

`@BeforeSuite`

The annotated method will be run only once before all tests in this suite have run.

@AfterSuite

The annotated method will be run only once after all 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 a part of the test.

Following are some of the benefits of using annotations

- TestNG identifies the methods it is interested in, by looking up annotations. Hence, method names are not restricted to any pattern or format.
- Additional parameters can be pass 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).

How to Run Test Suite in TestNg

A test suite is a collection of test cases intended to test a behavior or a set of behaviors of software program. In TestNG, we cannot define a suite in testing source code, but it is represented by one XML file, as suite is the feature of execution. It also allows flexible configuration of the tests to be run. A suite can contain one or more tests and is defined by the <suite> tag. <suite> is the root tag of your testng.xml. It describes a test suite, which in turn is made of several <test> sections.

The following table lists all the legal attributes that <suite> accepts.

Attribute & Description
name The name of this suite. It is a mandatory attribute.
verbose The level or verbosity for this run.

parallel

Whether TestNG should run different threads to run this suite.

thread-count

The number of threads to use, if parallel mode is enabled (ignored other-wise).

annotations

The type of annotations you are using in your tests.

time-out

The default timeout that will be used on all the test methods found in this test.

Let's take an example having two test classes, Test1 & Test2, to run together using Test Suite.

Create a Class

Create a java class to be tested, say, MessageUtil.java in C:\>JUNIT_WORKSPACE.

```
/*
 * This class prints the given message on console.
 */
public class MessageUtil {
    private String message;

    // Constructor
    // @param message to be printed
    public MessageUtil(String message) {
        this.message = message;
    }

    // prints the message
    public String printMessage() {
        System.out.println(message);
        return message;
    }

    // add "Hi!" to the message
    public String salutationMessage() {
        message = "Hi!" + message;
    }
}
```

```

        System.out.println(message);
        return message;
    }
}

```

Create Test Case Classes

- Create a java class file named Test1.java in C:\>TestNG_WORKSPACE.

```

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

public class Test1 {
    String message = "Manisha";
    MessageUtil messageUtil = new MessageUtil(message);

    @Test
    public void testPrintMessage() {
        System.out.println("Inside testPrintMessage()");
        Assert.assertEquals(message, messageUtil.printMessage());
    }
}

```

- Create a java class file named Test2.java in C:\>TestNG_WORKSPACE.

```

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

public class Test2 {
    String message = "Manisha";
    MessageUtil messageUtil = new MessageUtil(message);

    @Test
    public void testSalutationMessage() {
        System.out.println("Inside testSalutationMessage()");
        message = "Hi!" + "Manisha";

        Assert.assertEquals(message, messageUtil.salutationMessage());
    }
}

```

```

    }
}

```

- Now, let's write the testng.xml in C:\>TestNG_WORKSPACE, which would contain the <suite> tag as follows

```

<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name = "Suite1">

    <test name = "exampletest1">
        <classes>
            <class name = "Test1" />
        </classes>
    </test>

    <test name = "exampletest2">
        <classes>
            <class name = "Test2" />
        </classes>
    </test>

</suite>

```

- Suite1 includes exampletest1 and exampletest2.

Compile all java classes using javac.

```
C:\>javac MessageUtil.java Test1.java
Test2.java
```

- Now, run the testng.xml, which will run the test case defined in the provided Test Case class.

```
C:\>java -cp "C:\TestNG_WORKSPACE"
org.testng.TestNG testng.xml
```

Output

```

Inside testPrintMessage()
Manisha

```

```
Inside testSalutationMessage()
```

```
Hi!Manisha
```

```
=====
```

```
Suite1
```

```
Total tests run: 2, Failures: 0, Skips: 0
```

```
=====
```

The test-output folder can also be checked. Under the Suite1 folder, two html files can be seen which are created, exampletest1.html and exampletest2.html, which would look as follows

exampletest1			
<small>(Hover the method name to see the test class name)</small>			
PASSED TESTS			
Test method	Exception	Time (seconds)	Instance
testPrintMessage		0	TestNGTest1
testBasicTest		0	TestNGTest1

exampletest2			
<small>(Hover the method name to see the test class name)</small>			
PASSED TESTS			
Test method	Exception	Time (seconds)	Instance
testSalutationMessage		0	TestNGTest2
testBasicTest		0	TestNGTest2

Groups in TestNG

Group test is a new innovative feature in TestNG, which doesn't exist in JUnit framework. It permits to dispatch methods into proper portions and perform sophisticated groupings of test methods. Group tests provide maximum flexibility in how the tests are partitioned, and doesn't require to recompile anything if it wants to run two different sets of tests back to back.

Groups are specified in the testng.xml file using the <groups> tag. It can be found either under the <test> or <suite> tag. Groups specified in the <suite> tag apply to all the <test> tags underneath.

Now, let's take an example to see how group test works.

Create a Class

- Create a java class to be tested, say, MessageUtil.java in C:\> TestNG_WORKSPACE.

```
/*
 * This class prints the given message on console.
 */
public class MessageUtil {
    private String message;

    // Constructor
    // @param message to be printed
    public MessageUtil(String message) {
        this.message = message;
    }

    // prints the message
    public String printMessage() {
        System.out.println(message);
        return message;
    }

    // add "tutorialspoint" to the message
    public String salutationMessage() {
        message = "tutorialspoint" + message;
        System.out.println(message);
        return message;
    }

    // add "www." to the message
    public String exitMessage() {
        message = "www." + message;
        System.out.println(message);
        return message;
    }
}
```

- Create Test Case Class, `GroupTestExample.java`.
- Add test methods, `testPrintMessage()` and `testSalutationMessage()`, to your test class.
- Group the test method in two categories
 - Check-in tests (`checkintest`): These tests should be run before you submit new code. They should typically be fast and just make sure no basic functionality is broken.
 - Functional tests (`functest`): These tests should cover all the functionalities of your software and be run at least once a day, although ideally you would want to run them continuously.
- Create the java class file named `GroupTestExample.java` in `C:\>TestNG_WORKSPACE`.

```

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

public class GroupTestExample {
    String message = ".com";
    MessageUtil messageUtil = new MessageUtil(message);

    @Test(groups = { "functest", "checkintest" })

    public void testPrintMessage() {
        System.out.println("Inside testPrintMessage()");
        message = ".com";
        Assert.assertEquals(message, messageUtil.printMessage());
    }

    @Test(groups = { "checkintest" })

    public void testSalutationMessage() {
        System.out.println("Inside testSalutationMessage()");
        message = "tutorialspoint" + ".com";
        Assert.assertEquals(message,
        messageUtil.salutationMessage());
    }

    @Test(groups = { "functest" })
  
```

```

public void testingExitMessage() {
    System.out.println("Inside testExitMessage()");
    message = "www." + "tutorialspoint"+".com";
    Assert.assertEquals(message, messageUtil.exitMessage());
}
}

```

- Create testng.xml

Create testng.xml in C:\> TestNG_WORKSPACE, to execute test case(s). Here, we would be executing only those tests that belong to the group functest.

```

<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name = "Suite1">
    <test name = "test1">

        <groups>
            <run>
                <include name = "functest" />
            </run>
        </groups>

        <classes>
            <class name = "GroupTestExample" />
        </classes>

    </test>
</suite>

```

- Compile the MessageUtil, Test case classes using javac.

```
C:\TestNG_WORKSPACE>javac MessageUtil.java
GroupTestExample.java
```

- Now, run the testng.xml, which will run only the method testPrintMessage(), as it belongs to the group functest.

```
C:\TestNG_WORKSPACE>java -cp "C:\TestNG_WORKSPACE"
org.testng.TestNG testng.xml
```

- Verify the output. Only the method testPrintMessage() is executed.

```
Inside testPrintMessage()
```

```
.com
Inside testExitMessage()
www..com

=====
Suite1
Total tests run: 2, Failures: 1, Skips: 0
=====
```

Depend-on in TestNG

Sometimes, you may need to invoke methods in a Test case in a particular order or you want to share some data and state between methods. This kind of dependency is supported by TestNG as it supports the declaration of explicit dependencies between test methods.

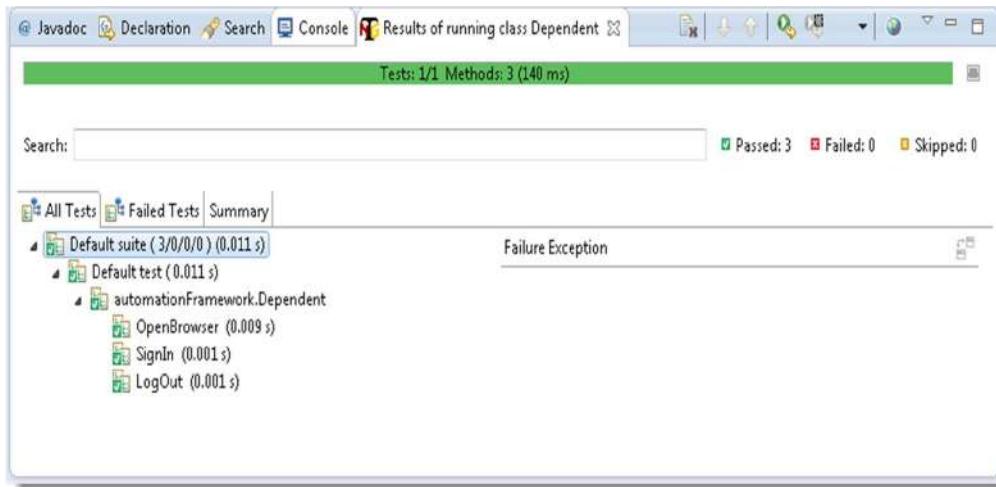
TestNG allows you to specify dependencies either with:

Using attributes dependsOnMethods in @Test annotations OR
 Using attributes dependsOnGroups in @Test annotations.

Take a look over the below example:

```
import org.testng.annotations.Test;
public class Dependent {
    @Test (dependsOnMethods = { "OpenBrowser" })
    public void SignIn() {
        System.out.println("This will execute second (SignIn)");
    }
    @Test
    public void OpenBrowser() {
        System.out.println("This will execute first (Open Browser)");
    }
    @Test (dependsOnMethods = { "SignIn" })
    public void LogOut() {
        System.out.println("This will execute third (Log Out)");
    }
}
```

The output will be like this:



Test Case Sequencing in TestNG

After building & validating the testing models several test cases are generated. The next biggest task is to decide the priority for executing them by using some systematic procedure.

The process begins with identification of "Static Test Cases" and "Dynamic Test Runs", brief introduction of which is as under.

Test case: It is a collection of several items and corresponding information, which enables a test to be executed or performing a test run.

Test Run: It is a dynamic part of the specific testing activities in the overall sequence of testing on some specific testing object.

Every time we invoke a static test case, we in-turn perform an individual dynamic test run. Hence we can say that, every test case can correspond to several test runs.

Why & how do we prioritize?

Out of a large cluster of test cases in our hand, we need to scientifically decide their priorities of execution based upon some rational, non-arbitrary, criteria. We carry out the prioritization activity with an objective to reduce the overall number of test cases in the total testing feat.

If test priority is not defined while, running multiple test cases, TestNG assigns all @Test a priority as zero(0).

Example:

We have assigned the Priority to each test case means test case will the lower priority value will be executed first.

Priority in testNG in action

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

```

import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.Test;

public class Priority_In_testNG {
    WebDriver driver;

    // Method 1: Open Browser say Firefox
    @Test (priority=1)
    public void openBrowser() {
        driver = new FirefoxDriver();
    }

    // Method 2: Launch Google.com
    @Test (priority=2)
    public void launchGoogle() {
        driver.get("http://www.google.co.in");
    }

    // Method 3: Perform a search using "Facebook"
    @Test (priority=3)
    public void performSearchAndClick1stLink() {

        driver.findElement(By.xpath(".//*[@title='Search']")).sendKeys("Facebook");
    }

    // Method 4: Verify Google search page title.
    @Test (priority=4)
    public void FaceBookPageTitleVerification() throws Exception {

        driver.findElement(By.xpath(".//*[@value='Search']")).click();

        Thread.sleep(3000);
        Assert.assertEquals(driver.getTitle().contains("Facebook - Google Search"), true);
    }
}

```

```

    }
}

```

Code Explanation

After assigning priority to each testcases, run the above code using testNG as shown in Video-2 mentioned below.

Here, you can see that test cases are prioritized. Test case having lower priority are executed first i.e. now there is a sequential execution according to priority in the test cases. Hence, all test cases are passing now.

Output

```

PASSED: openBrowser
PASSED: launchGoogle
PASSED: peformSearchAndClick1stLink
PASSED: FaceBookPageTitleVerification

```

TestNG Asserts

Asserts helps to verify the conditions of the test and decide whether test has failed or passed. A test is considered successful ONLY if it is completed without throwing any exception.

Here it is verifying if the page title is equal to 'Google' or not. If the page title is not matching with the text / title that we provided, it will fail the test case.

To explain what assertion is, lets us look into below code sample

```

@Test
public void testCaseVerifyHomePage() {
    driver= new FirefoxDriver();
    driver.navigate().to("http://google.com");
    Assert.assertEquals("Google", driver.getTitle());
}

```

First create a Firefox driver, next navigate to the Google page and then the third line here is called Assertion.

It can also be written as below `Assert.assertEquals("Goooole", driver.getTitle(), "Title not matching");`

TestNG supports assertion of a test using the Assert class and Assertion plays an important role when testing an application.

The page title will change in the below code and see the result after executing it.

```

@Test
public void testCaseVerifyHomePage() {
    driver= new FirefoxDriver();
}

```

```

        driver.navigate().to("http://google.com");
        Assert.assertEquals("Gooooogle", driver.getTitle());
    }
}

```

The above code will throw you an Assertion error as below: java.lang.AssertionError: expected [Google] but found [Gooooogle]

Like wise there are many Assertions provided by the TestNG. The below are the few which are used commonly.

- *assertEqual(String actual, String expected)*: It takes two string arguments and checks whether both are equal, if not it will fail the test.
- *assertEqual(String actual, String expected, String message)*: It takes three string arguments and checks whether both are equal, if not it will fail the test and throws the message which we provide.
- *assertEquals(boolean actual, boolean expected)*: It takes two boolean arguments and checks whether both are equal, if not it will fail the test.
- *assertEquals(java.util.Collection actual, java.util.Collection expected, java.lang.String message)*: Takes two collection objects and verifies both collections contain the same elements and with the same order. if not it will fail the test with the given message.
- *assert.assertTrue(condition)*: It takes one boolean arguments and checks that a condition is true, If it isn't, an AssertionError is thrown.
- *assert.assertTrue(condition, message)*: It takes one boolean argument and String message. It asserts that a condition is true. If it isn't, an AssertionError, with the given message, is thrown.
- *assert.assertFalse(condition)*: It takes one boolean arguments and checks that a condition is false, If it isn't, an AssertionError is thrown.
- *assert.assertFalse(condition, message)*: It takes one boolean argument and String message. It Asserts that a condition is false. If it isn't, an AssertionError, with the given message, is thrown.

TestNG Parameters

Everybody knows the importance of Parameterization in testing and in automation testing. It allows to automatically run a test case multiple times with different input and validation values. As Selenium *Webdriver* is more an automated testing framework than a ready-to-use tool, it will have to put in some effort to support data driven testing in your automated tests. It usually prefer to use Microsoft Excel as the format for storing my parameters but so many of my followers have requested to write an article on TestNG Data Provider.

TestNG again gives us another interesting feature called TestNG Parameters. TestNG usually pass parameters directly to the test methods with the testng.xml.

Let me take a very simple example of LogIn application, where the username and password is required to clear the authentication.

1. Create a test on my demo OnlineStore application to perform LogIn which takes the two string argument as username & password.
2. Provide Username & Password as parameter using TestNG Annotation.

```

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.Test;
import org.testng.annotations.Parameters;
public class TestngParameters {
    private static WebDriver driver;
    @Parameters({ "sUsername", "sPassword" })
    public void test(String sUsername, String sPassword) {
        driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
        driver.get("http://www.store.demoqa.com");

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

        driver.findElement(By.id("log")).sendKeys(sUsername);
        driver.findElement(By.id("pwd")).sendKeys(sPassword);
        driver.findElement(By.id("login")).click();

        driver.findElement(By.xpath(".//*[@id='account_logout']/a")).click();
        driver.quit();
    }
}

```

3. The parameter would be passed values from testng.xml which will see in the next step.

```

<suite name="Suite">

<test name="ToolsQA">

<parameter name="sUsername" value="testuser_1"/>
<parameter name="sPassword" value="Test@123"/>

<classes>

<class
name="automationFramework.TestngParameters" />

</classes>

```

```
</test>

</suite>
```

4. Now, run the testng.xml, which will run the parameterTest method. TestNG will try to find a parameter named sUsername & sPassword.

Skipping Test Cases

Let's see how to skip TestNG test deliberately. Sometimes we may face a situation where our test cases might not be ready and it need to skip those tests from running. One way of skipping a test method is by using `throw new SkipException()` exception.

Scenario 1: Skip TestNG Test, if a condition met else continue execution.

Let see a sample WebDriver test case example where `SkipException()` is placed inside if condition to Intentionally skip that test. Once `SkipException()` thrown, remaining part of that test method will not be executed and control will goes directly to next test method execution.

```
import org.testng.annotations.Test;
import org.testng.SkipException;
public class SkipTestCase {
    public void aSkipTest() {
        String condition = "Skip Test";
        if(condition.equals("Skip Test")){
            // throw new SkipException("Skipping - This is not ready
            // for testing ");
        }else{
            System.out.println("I am in else condition");
        }
        System.out.println("I am out of the if else condition");
    }
    public void nonSkipTest(){
        System.out.println("No need to skip this test");
    }
}
```

Output:

[TestNG] Running:

```
I am out of the if else condition
No need to skip this test
```

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

Scenario 2: On uncomment the “throw new SkipException()” in the if condition.

```
import org.testng.annotations.Test;
import org.testng.SkipException;

public class SkipTestCase {
    public void aSkipTest(){
        String a ="Skip Test";
        if(a.equals("Skip Test")){
            throw new SkipException("Skipping - This is not ready for
testing ");
        }else{
            System.out.println("I am in else condition");
        }
        System.out.println("I am out of the if else condition");
    }

    public void nonSkipTest(){
        System.out.println("No need to skip this test");
    }
}
```

Output

[TestNG] Running:

```
No need to skip this test
PASSED: nonSkipTest
SKIPPED: aSkipTest
```

```
=====
Default suite
=====
```

```
Total tests run: 2, Failures: 0, Skips: 1
=====
```

Skip exception thrown and the remaining part of the first test method “aSkipTest” not executed and control reached to second test method “nonSkipTest” and printed the value as “No need to skip this test”.

Multi Browser Testing in TestNG

In every project it is required to perform multi-browser testing to make sure that the functionality is working as expected with every browser to give equal user experience to all of the wide range of audience. It takes a considerable time to test everything on every browser, when used automation to reduce the testing efforts then do the multi-browser testing using automation. TestNG gives us functionality to perform same test on different browsers in a simple and easy way.

1. Create your Script to test a LogIn application using TestNG class.
2. Pass ‘Browser Type’ as parameters using TestNG annotations to the before method of the TestNG class. This method will launch only the browser, which will be provided as parameter.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class MultiBrowser {
    public WebDriver driver;
    @Parameters("browser")
    @BeforeClass
    // Passing Browser parameter from TestNG xml
    public void beforeTest(String browser) {
```

```

// If the browser is Firefox, then do this

if(browser.equalsIgnoreCase("firefox")) {

    driver = new FirefoxDriver();

// If browser is IE, then do this

} else if (browser.equalsIgnoreCase("ie")) {

    // Here I am setting up the path for my IEDriver

    System.setProperty("webdriver.ie.driver",
"D:\ToolsQA\OnlineStore\drivers\IEDriverServer.exe");

    driver = new InternetExplorerDriver();

}

// Doesn't the browser type, lauch the Website

driver.get("http://www.store.demoqa.com");

}

// Once Before method is completed, Test method will start

@Test public void login() throws InterruptedException {

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

    driver.findElement(By.id("log")).sendKeys("testuser_1");

    driver.findElement(By.id("pwd")).sendKeys("Test@123");

    driver.findElement(By.id("login")).click();

}

@AfterClass public void afterTest() {

    driver.quit();

}

```

3. Create a TestNG XML for running your test. Configure the TestNG XML for passing parameters i.e. to tell which browser should be used for running the Test.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

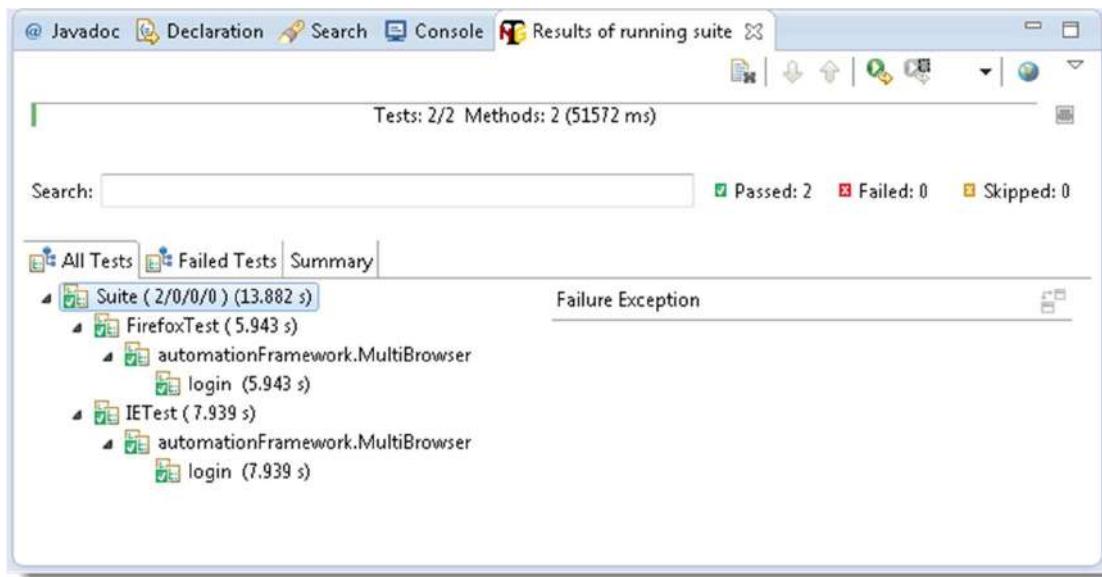
<suite name="Suite" parallel="none">

    <test name="FirefoxTest">
        <parameter name="browser" value="firefox" />
        <classes>
            <class name="automationFramework.MultiBrowser" />
        </classes>
    </test>

    <test name="IETest">
        <parameter name="browser" value="ie" />
        <classes>
            <class name="automationFramework.MultiBrowser" />
        </classes>
    </test>
</suite>
```

Note: It can set any number of Browsers here and just for the example purpose it has set up only two main browsers.

4. Now it's time to run the xml. Run the test by right click on the testng.xml file and select Run As > TestNG Suite.



Parallel Testing in TestNG

Using the feature provided by TestNG for Parallel Executions, just take the above example for Sign In application with two different browsers. This time all it is to execute test in both browsers simultaneously.

Now just set the 'parallel' attribute to 'tests' in the above used xml and give a run again. This time it will notice that the both browsers will open almost simultaneously and the test will run in parallel.

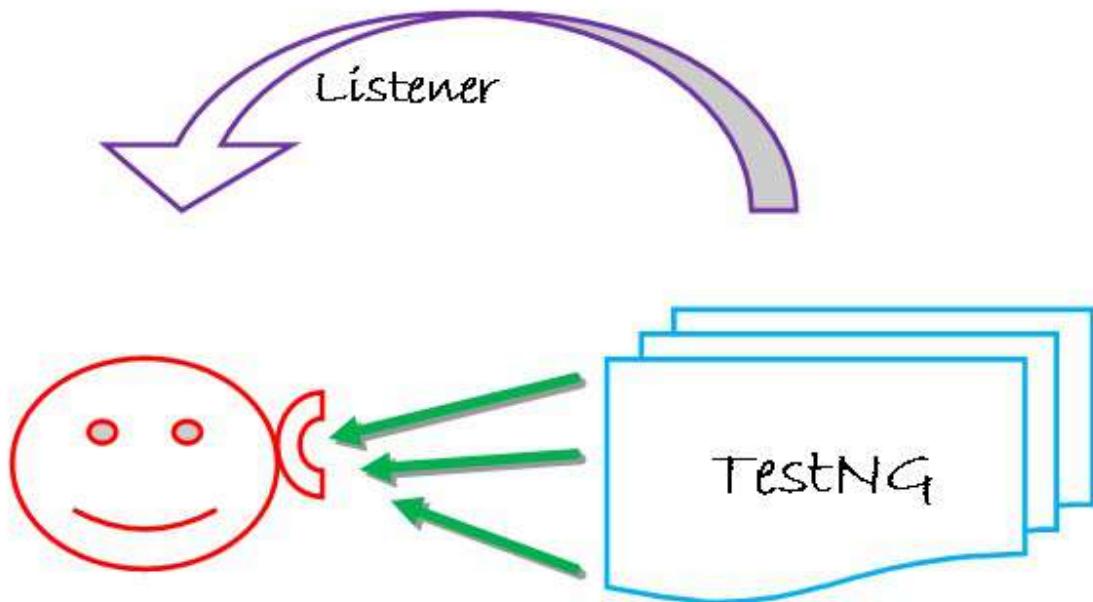
XHTML

```
<?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="automationFramework.MultiBrowser" />
        </classes>
    </test>
    <test name="IETest">
        <parameter name="browser" value="ie" />
        <classes>
            <class name="automationFramework.MultiBrowser" />
        </classes>
    </test>
</suite>
```

```
</test>
</suite>
```

TestNG Listeners

Listener is defined as interface that modifies the default TestNG's behavior. As the name suggests Listeners "listen" to the event defined in the selenium script and behave accordingly. It is used in selenium by implementing Listeners Interface. It allows customizing TestNG reports or logs. There are many types of TestNG listeners available.



Types of Listeners in TestNG

There are many types of listeners which allows you to change the TestNG's behavior.

Below are the few TestNG listeners:

- IAnnotationTransformer
- IAnnotationTransformer2
- IConfigurable
- IConfigurationListener
- IExecutionListener
- IHookable
- IInvokedMethodListener
- IInvokedMethodListener2

- IMethodInterceptor
- IReporter
- ISuiteListener
- ITestListener

Above Interface are called TestNG Listeners. These interfaces are used in selenium to generate logs or customize the Testing reports.

ITestListener has following methods

- OnStart- OnStart method is called when any Test starts.
- onTestSuccess- onTestSuccess method is called on the success of any Test.
- onTestFailure- onTestFailure method is called on the failure of any Test.
- onTestSkipped- onTestSkipped method is called on skipped of any Test.
- onTestFailedButWithinSuccessPercentage- method is called each time Test fails but is within success percentage.
- onFinish- onFinish method is called after all Tests are executed.

Example

Steps used

- Create a 'New Class' file and give it a name 'Listener', by right click on the Package and select New > Class.
- Now Implements ISuiteListener, ITestListener and IInvokedMethodListener to this newly created class. For implementing a listener class, the class has to implement the org.testng.ITestListener interface. These classes are notified at runtime by TestNG when the test starts, finishes, fails, skips, or passes.

```
package utility;

import org.testng.IInvokedMethod;
import org.testng.IInvokedMethodListener;
import org.testng.ISuite;
import org.testng.ISuiteListener;
import org.testng.ITestContext;
import org.testng.ITestListener;
```

```
import org.testng.ITestNGMethod;  
  
import org.testng.ITestResult;  
  
import org.testng.Reporter;  
  
public class Listener implements ITestListener, ISuiteListener,  
IInvokedMethodListener {  
  
    // This belongs to ISuiteListener and will execute before the  
    Suite start  
  
    @Override  
  
    public void onStart(ISuite arg0) {  
  
        Reporter.log("About to begin executing Suite " +  
        arg0.getName(), true);  
  
    }  
  
    // This belongs to ISuiteListener and will execute, once the  
    Suite is finished  
  
    @Override  
  
    public void onFinish(ISuite arg0) {  
  
        Reporter.log("About to end executing Suite " +  
        arg0.getName(), true);  
  
    }  
  
    // This belongs to ITestListener and will execute before  
    starting of Test set/batch  
  
    public void onStart(ITestContext arg0) {
```

```
        Reporter.log("About to begin executing Test " +
arg0.getName(), true);

    }

    // This belongs to ITestListener and will execute, once the
    Test set/batch is finished

    public void onFinish(ITestContext arg0) {

        Reporter.log("Completed executing test " +
arg0.getName(), true);

    }

    // This belongs to ITestListener and will execute only when the
    test is pass

    public void onTestSuccess(ITestResult arg0) {

        // This is calling the printTestResults method

        printTestResults(arg0);

    }

    // This belongs to ITestListener and will execute only on the
    event of fail test

    public void onTestFailure(ITestResult arg0) {

        // This is calling the printTestResults method

        printTestResults(arg0);

    }
}
```

```
// This belongs to ITestListener and will execute before the
main test start (@Test)

public void onTestStart(ITestResult arg0) {

    System.out.println("The execution of the main test starts
now");

}

// This belongs to ITestListener and will execute only if any
of the main test(@Test) get skipped

public void onTestSkipped(ITestResult arg0) {

    printTestResults(arg0);

}

public void onTestFailedButWithinSuccessPercentage(ITestResult
arg0) {

}

// This is the method which will be executed in case of test
pass or fail

// This will provide the information on the test

private void printTestResults(ITestResult result) {

    Reporter.log("Test Method resides in " +
result.getTestClass().getName(), true);

    if (result.getParameters().length != 0) {

        String params = null;
```

```
for (Object parameter : result.getParameters()) {  
  
    params += parameter.toString() + ",";  
  
}  
  
Reportер.log("Test Method had the following  
parameters : " + params, true);  
  
}  
  
String status = null;  
  
switch (result.getStatus()) {  
  
case ITestResult.SUCCESS:  
  
    status = "Pass";  
  
    break;  
  
case ITestResult.FAILURE:  
  
    status = "Failed";  
  
    break;  
  
case ITestResult.SKIP:  
  
    status = "Skipped";  
  
}  
  
Reportер.log("Test Status: " + status, true);
```

```

}

// This belongs to IInvokedMethodListener and will execute
before every method including @Before @After @Test

public void beforeInvocation(IInvokedMethod arg0, ITestResult
arg1) {

    String textMsg = "About to begin executing following
method : " + returnMethodName(arg0.getTestMethod());

    Reporter.log(textMsg, true);

}

// This belongs to IInvokedMethodListener and will execute
after every method including @Before @After @Test

public void afterInvocation(IInvokedMethod arg0, ITestResult
arg1) {

    String textMsg = "Completed executing following method :
" + returnMethodName(arg0.getTestMethod());

    Reporter.log(textMsg, true);

}

// This will return method names to the calling function

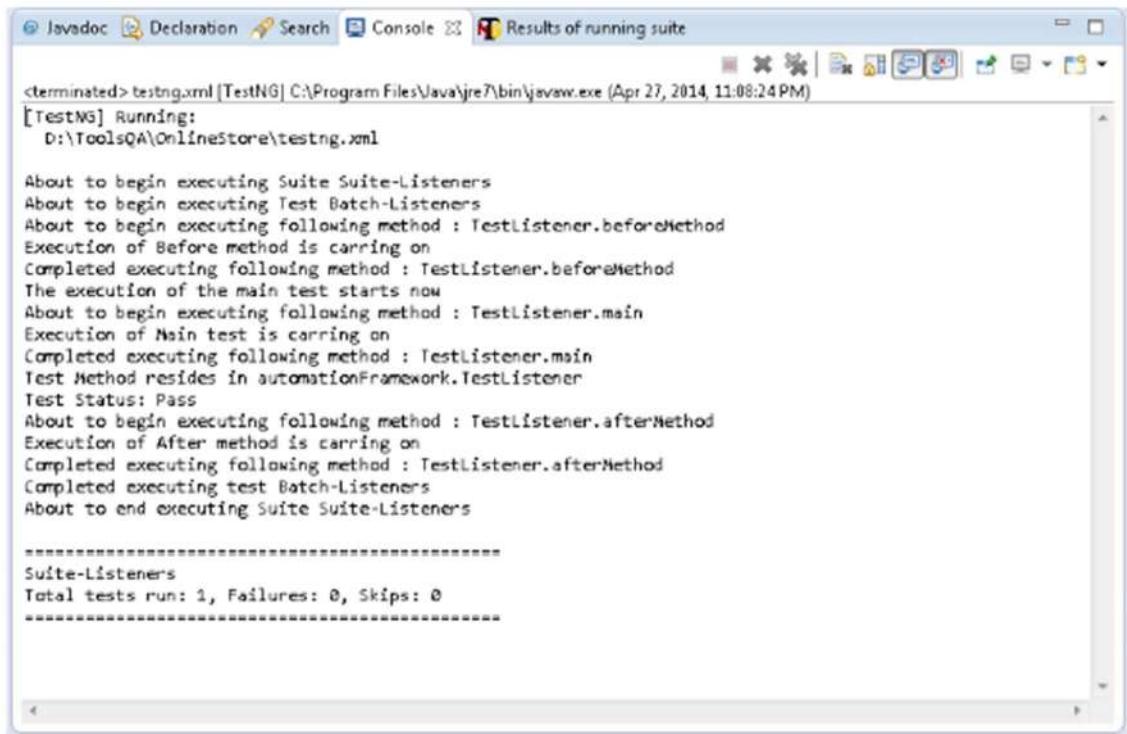
private String returnMethodName(ITestNGMethod method) {

    return method.getRealClass().getSimpleName() + "." +
method.getMethodName();

}

}

```



```

<terminated> testng.xml [TestNG] C:\Program Files\Java\jre7\bin\javaw.exe (Apr 27, 2014, 11:08:24 PM)
[TestNG] Running:
D:\ToolsQA\OnlineStore\testng.xml

About to begin executing Suite Suite-Listeners
About to begin executing Test Batch-Listeners
About to begin executing following method : TestListener.beforeMethod
Execution of Before method is carrying on
Completed executing following method : TestListener.beforeMethod
The execution of the main test starts now
About to begin executing following method : TestListener.main
Execution of Main test is carrying on
Completed executing following method : TestListener.main
Test Method resides in automationFramework.TestListener
Test Status: Pass
About to begin executing following method : TestListener.afterMethod
Execution of After method is carrying on
Completed executing following method : TestListener.afterMethod
Completed executing test Batch-Listeners
About to end executing Suite Suite-Listeners

=====
Suite-Listeners
Total tests run: 1, Failures: 0, Skips: 0
=====
```

```

package automationFramework;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.Test;

// This code will implement TestNG listeners

@Listeners(PackageName.ListenerClassName)

// For e.g. @Listeners(utility.Listener.class)

public class TestListener {

    @Test

    public void main() {
```

{

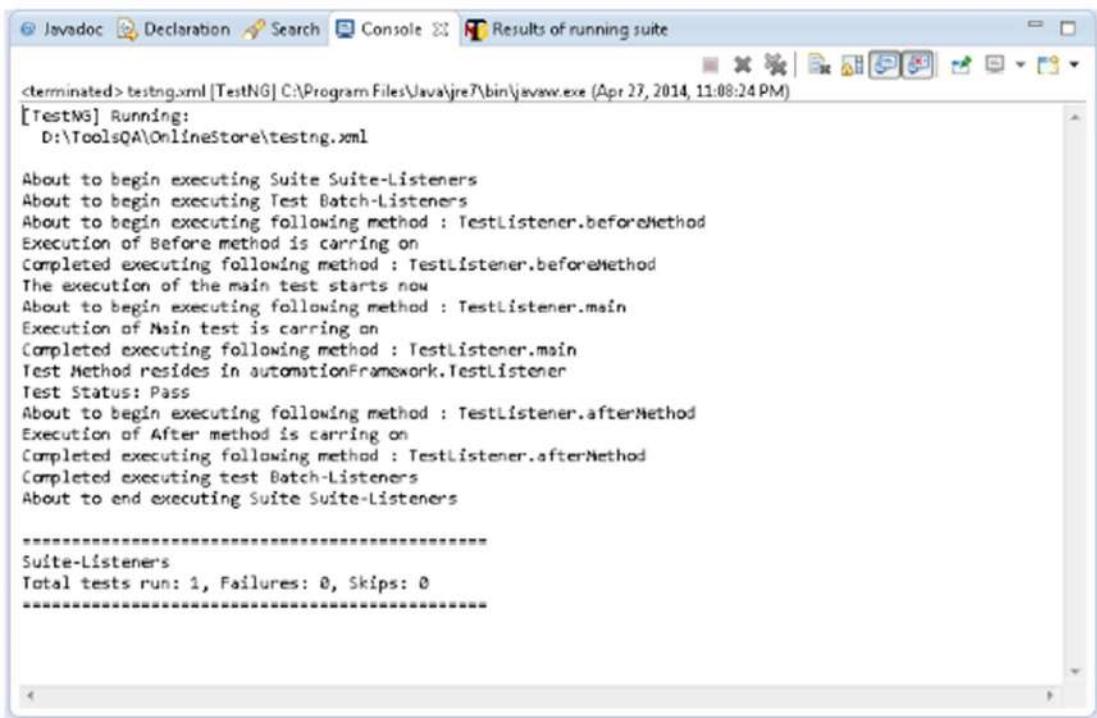
}

Listener tag in TestNG xml: Although approach 1 is more than enough to get you started, it's not an "elegant" way of using Listeners, because you are forced to add this @Listeners section to each of your classes, which you perhaps won't want. So what you do is, you create a TestNG Suite xml and then add up the listeners section to this suite xml file. That way, all of your tests would essentially leverage the listener that you wrote.

```
<suite name="Suite-Listeners" parallel="none">

    <listeners>
        <listener class-name="utility.Listener" />
    </listeners>
    <test name="Batch-Listeners">
        <classes>
            <class name="automationFramework.TestListener" />
        </classes>
    </test>
</suite>
```

Output of the test case will look like this:



The screenshot shows a Java console window titled "Results of running suite". The output text is as follows:

```
<terminated> testng.xml [TestNG] C:\Program Files\Java\jre7\bin\javaw.exe (Apr 27, 2014, 11:08:24 PM)
[TestNG] Running:
D:\ToolsQA\OnlineStore\testng.xml

About to begin executing Suite Suite-Listeners
About to begin executing Test Batch-Listeners
About to begin executing following method : TestListener.beforeMethod
Execution of Before method is carrying on
Completed executing following method : TestListener.beforeMethod
The execution of the main test starts now
About to begin executing following method : TestListener.main
Execution of Main test is carrying on
Completed executing following method : TestListener.main
Test Method resides in automationFramework.TestListener
Test Status: Pass
About to begin executing following method : TestListener.afterMethod
Execution of After method is carrying on
Completed executing following method : TestListener.afterMethod
Completed executing test Batch-Listeners
About to end executing Suite Suite-Listeners

=====
Suite-Listeners
Total tests run: 1, Failures: 0, Skips: 0
=====
```

Summary

- TestNG is an open source automated testing framework
- TestNG eliminates most of the limitations of the older framework and gives the developer the ability to write more flexible and powerful tests with help of easy annotations, grouping, sequencing & parameterizing
- TestNG can also be used to run failed test-cases
- TestNG allows testers to execute and run one script in multiple browsers
- In TestNG, we cannot define a suite in testing source code, but it is represented by one XML file, as suite is the feature of execution
- Asserts helps to verify the conditions of the test and decide whether test has failed or passed

DUCAT®
www.ducatindia.com

Chapter 12 – Log4j Logging

Introduction

Log4j is a reliable, fast and flexible logging framework (APIs) written in Java, which is distributed under the Apache Software License. log4j is a popular logging package written in Java. Log4j has been ported to the C, C++, C#, Perl, Python, Ruby, and Eiffel languages.

Log4j is highly configurable through external configuration files at runtime. It views the logging process in terms of levels of priorities and offers mechanisms to direct logging information to a great variety of destinations, such as a database, file, console, UNIX Syslog.

Advantages of Log4j

The advantages of Log4j are:

- Log4j allows a very good logging infrastructure without putting in any efforts.
- Log4j gives the ability to categorize logs at different levels (Trace, Debug, Info, Warn, Error and Fatal).
- Log4j gives the ability to direct logs to different outputs. For e.g. to a file, Console or a Database.
- Log4j gives the ability to define the format of output logs.
- Log4j gives the ability to write Asynchronous logs which helps to increase the performance of the application.
- Loggers in Log4j follow a class hierarchy which may come handy to the applications.

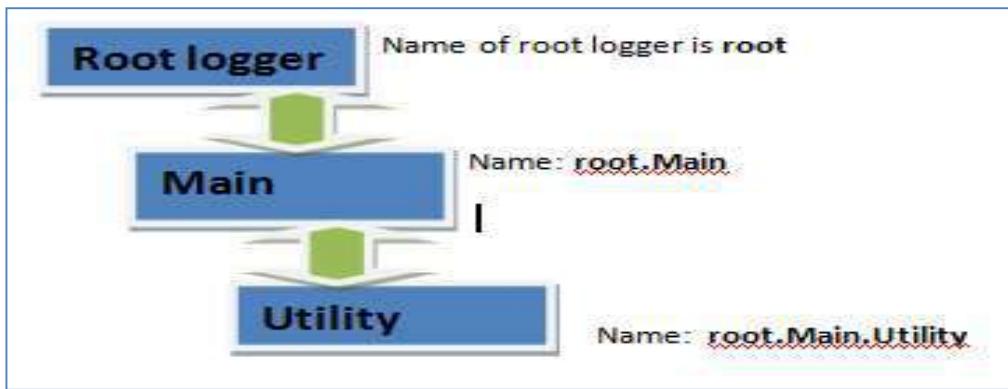
Components of Log4j

log4j has three main components:

- **Loggers:** It is responsible for capturing logging information.
- **Appenders:** It is responsible for publishing logging information to various preferred destinations.
- **Layouts:** It is responsible for formatting logging information in different styles.

Creating Logger

Logger is the object which lets the log information to the required Log location, be it console or a file or even a database. Logger objects follow hierarchy similar to class hierarchy in any OOP language. Naming convention of Logger hierarchy is in the name. Each objects name decide which hierarchy it follows. For example take a logger named “Main.Utility”. So Utility is the child of Main and Main is the father of Utility. Also, all Loggers are derived from root Logger. The actual hierarchy will be root. Main.Utility with root being ancestor of Utility and Father of Main. This can be shown in a diagram below:



These relationships are managed by the LogManager class. Lets illustrate it using an example

```

import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
public class SampleEntry {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

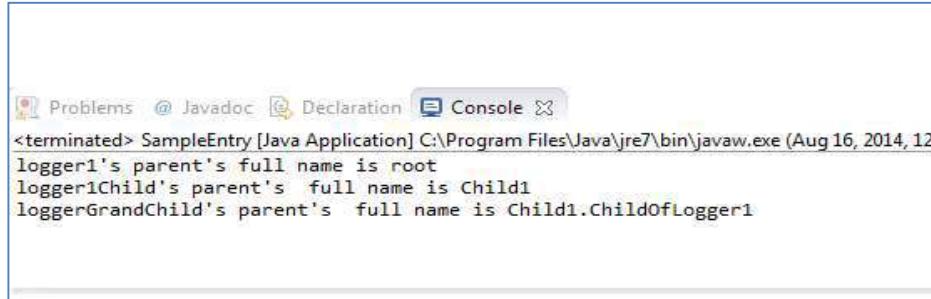
        Logger chance =
LogManager.getLogger(SampleEntry.class.getName());
        Logger logger1 = LogManager.getLogger("Child1");
        Logger logger1Child =
logger1.getLogger("Child1.ChildOfLogger1");
        Logger loggerGrandChild =
LogManager.getLogger("Child1.ChildOfLogger1.GrandChild");

        System.out.println("logger1's full name is " +
logger1.getParent().getName());
        System.out.println("logger1Child's full name is " +
logger1Child.getParent().getName());
        System.out.println("loggerGrandChild's full name is " +
loggerGrandChild.getParent().getName());

    }
}

```

Output:



```
Problems @ Javadoc Declaration Console
<terminated> SampleEntry [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Aug 16, 2014, 12
logger1's parent's full name is root
logger1Child's parent's full name is Child1
loggerGrandChild's parent's full name is Child1.childOfLogger1
```

As it can see that logger1 is the parent of logger1Child and grandparent of logger GrandChild's.

Logging levels

Logger class have following print methods that help you to log information.

- Trace
- Debug
- Info
- Warn
- Error
- Fatal

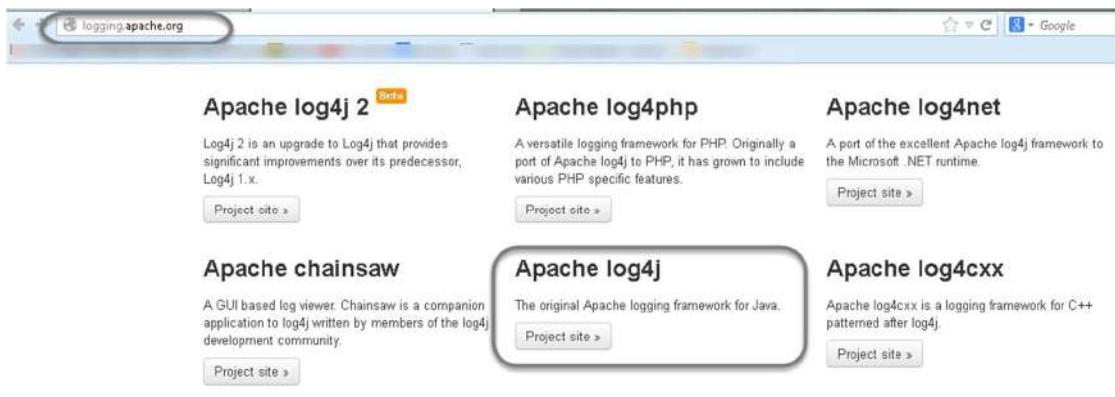
For example a debug log can be print by just writing `Logger.Debug("This is a debug log")`. Any other overloaded Logger can be used to choose `Debug()` method. All these print statements are called Levels. Each log level expects a certain type of information for example debug level expects logging of that information which may help a programmer debug the application in case of failures. Similarly Error Level expects all the Errors to be logged using this level. Log level of a logger can be set by using the `Logger.setLevel` method. Log levels have following order:

TRACE < DEBUG < INFO < WARN < ERROR < FATAL

Download Log4j

Follow the given below Steps to download Log4j

1. Go to link <http://logging.apache.org/> and click Apache log4j.



The screenshot shows the Apache Logging Services website at logging.apache.org. It displays several projects:

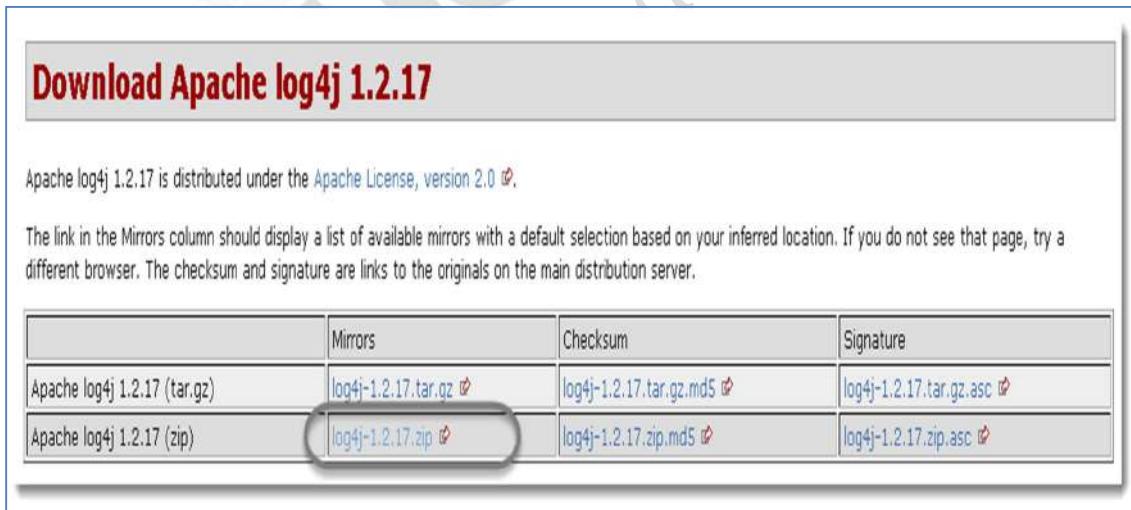
- Apache log4j 2**: Log4j 2 is an upgrade to Log4j that provides significant improvements over its predecessor, Log4j 1.x. [Project site >](#)
- Apache log4php**: A versatile logging framework for PHP. Originally a port of Apache log4j to PHP, it has grown to include various PHP specific features. [Project site >](#)
- Apache log4net**: A port of the excellent Apache log4j framework to the Microsoft .NET runtime. [Project site >](#)
- Apache chainsaw**: A GUI based log viewer. Chainsaw is a companion application to log4j written by members of the log4j development community. [Project site >](#)
- Apache log4j**: The original Apache logging framework for Java. [Project site >](#)
- Apache log4cxx**: Apache log4cxx is a logging framework for C++ patterned after log4j. [Project site >](#)

- Click on “Download” on the left side menu.



The screenshot shows the Apache log4j 1.2.17 download page at logging.apache.org/log4j/1.2/download.html. The page title is "Apache log4j™ 1.2". On the left, there's a sidebar with links: "About log4j 1.2", "What's New?", "Download", "FAQ", "Roadmap", and "Documentation". The main content area has a heading "Download Apache log4j 1.2.17". Below it, it says "Apache log4j 1.2.17 is distributed under the Apache License, version 2.0". The text continues: "The link in the Mirrors column should display a list of available mirrors with a default selection based on your inferred location. If you do not see that page, try a different browser. The checksum and signature are links to the originals on the main distribution server." A table below lists download links for tar.gz and zip formats.

- Click on the ZIP file under Mirrors column.



The screenshot shows the same Apache log4j 1.2.17 download page. The ZIP file link under the "Mirrors" column for the "Apache log4j 1.2.17 (zip)" row is circled in red. The table columns are "Mirrors", "Checksum", and "Signature".

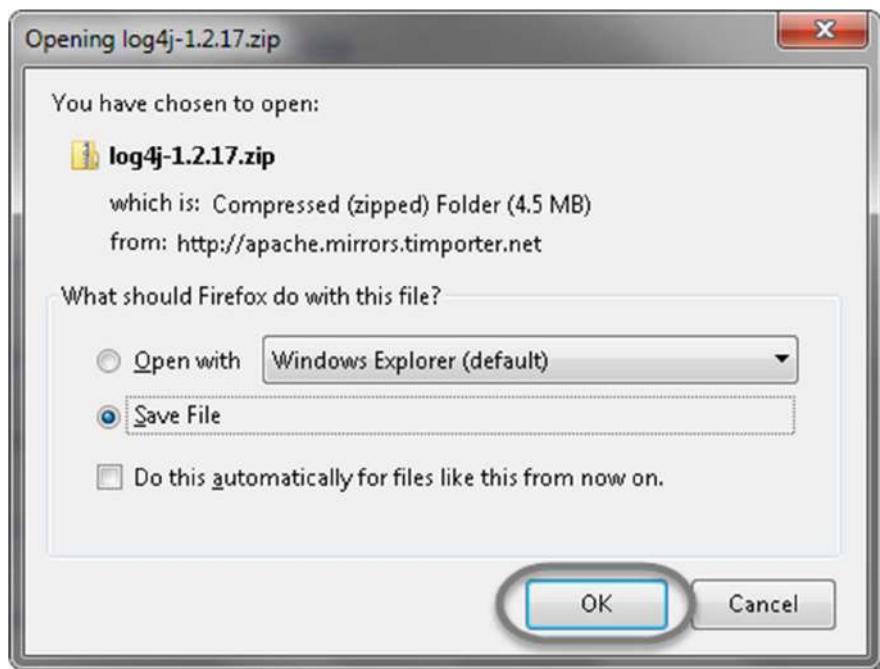
	Mirrors	Checksum	Signature
Apache log4j 1.2.17 (tar.gz)	log4j-1.2.17.tar.gz	log4j-1.2.17.tar.gz.md5	log4j-1.2.17.tar.gz.asc
Apache log4j 1.2.17 (zip)	log4j-1.2.17.zip	log4j-1.2.17.zip.md5	log4j-1.2.17.zip.asc

- Click on the highlighted link at the top of the page.

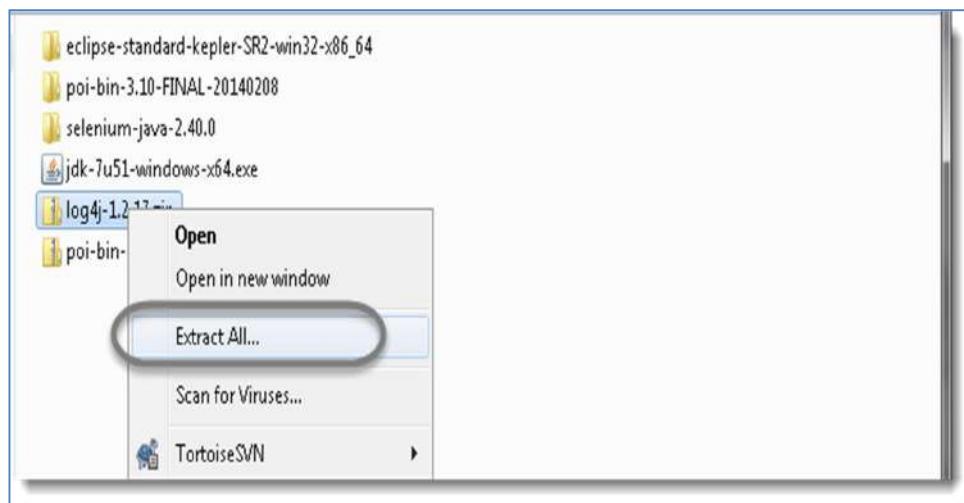


The screenshot shows the Apache Software Foundation's download mirrors page. At the top, there is a logo of a feather and the text "The Apache Software Foundation". Below this, a search bar has "Apache Download Mirrors" typed into it. A link to "http://apache.mirrors.timporter.net/logging/log4j/1.2.17/log4j-1.2.17.zip" is highlighted with a blue border. Below the link, a note says "Other mirror sites are suggested below. Please use the backup mirrors only to download PGP and MD5 signatures to verify your downloads or if no other mirrors are working." There is also a "Search" button.

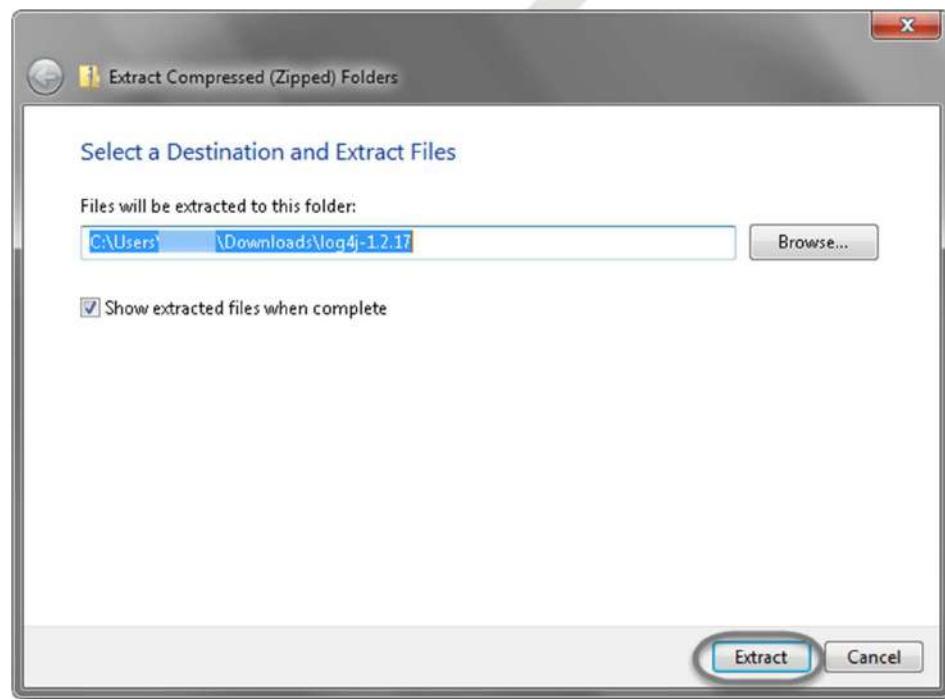
5. Select the radio button for “Save File” and click OK. Zip file will be saved on the system with in few seconds.



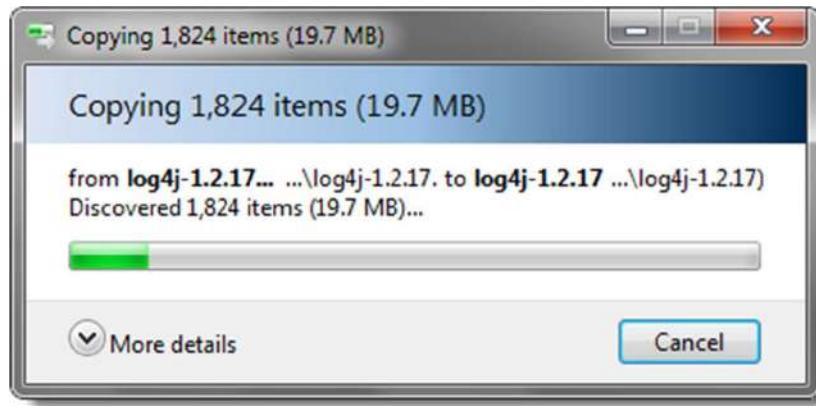
6. Right click on the Zip file and select “Extract All”.



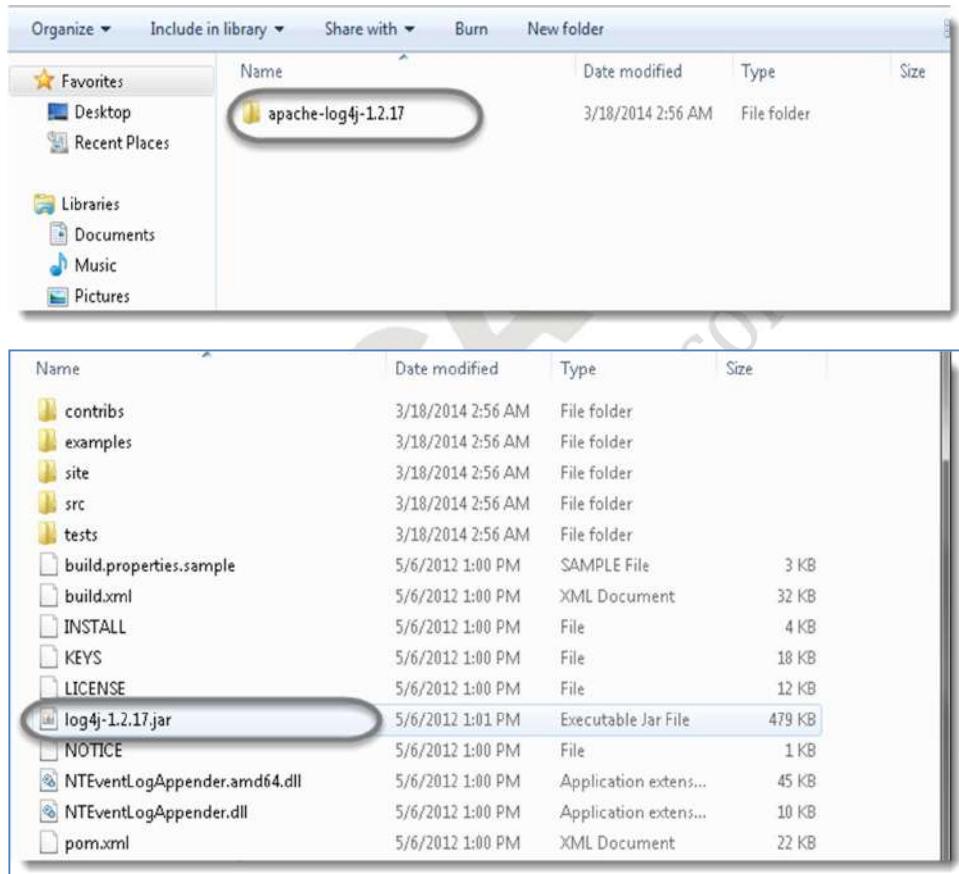
7. Specify the location.



8. Wait for the Extraction to finish.



9. Open the Log4j extracted folder from the saved location.

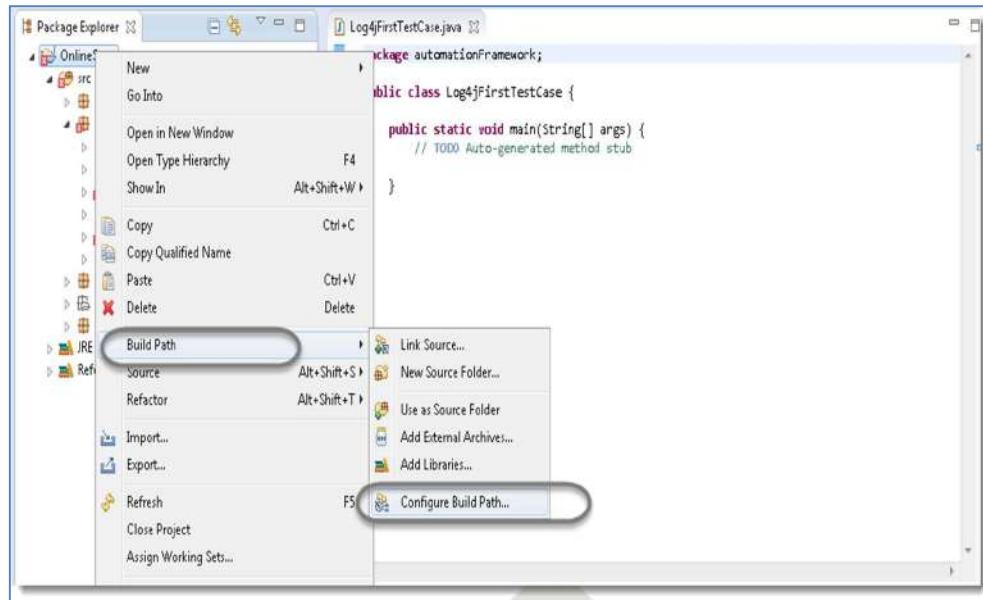


10. The Log4j JAR file has downloaded, now all is need to do is to add this JAR file to the project and write the first Log4j logging enabled test script.

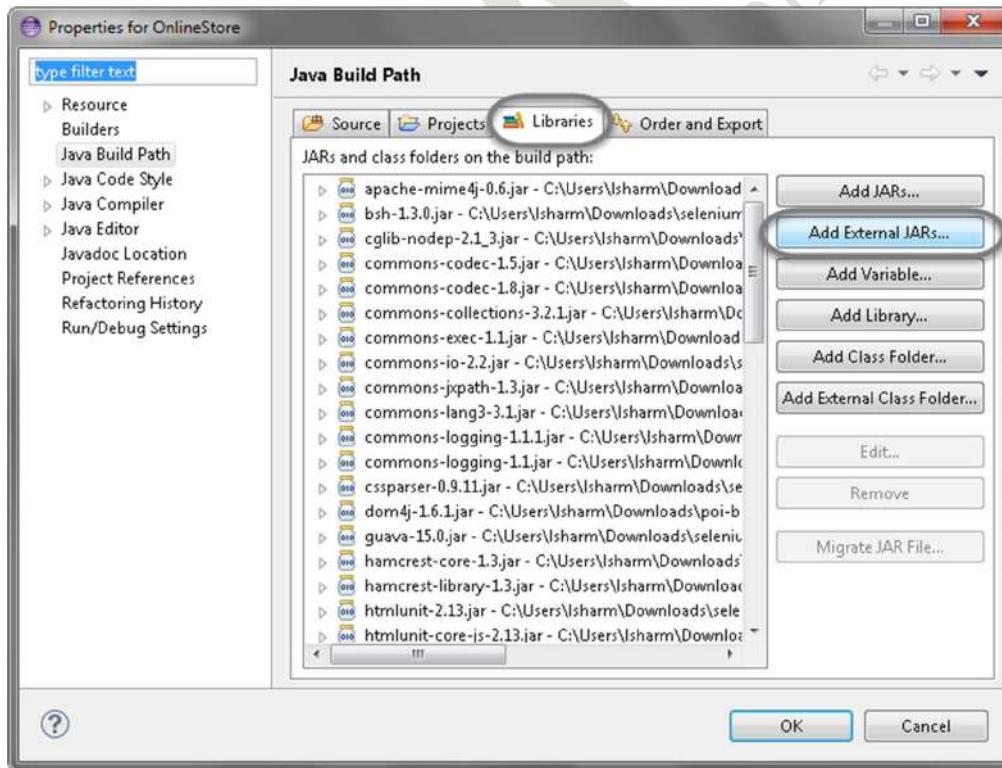
Add Log4j Jar

The steps to follow to add Log4j Jar are:

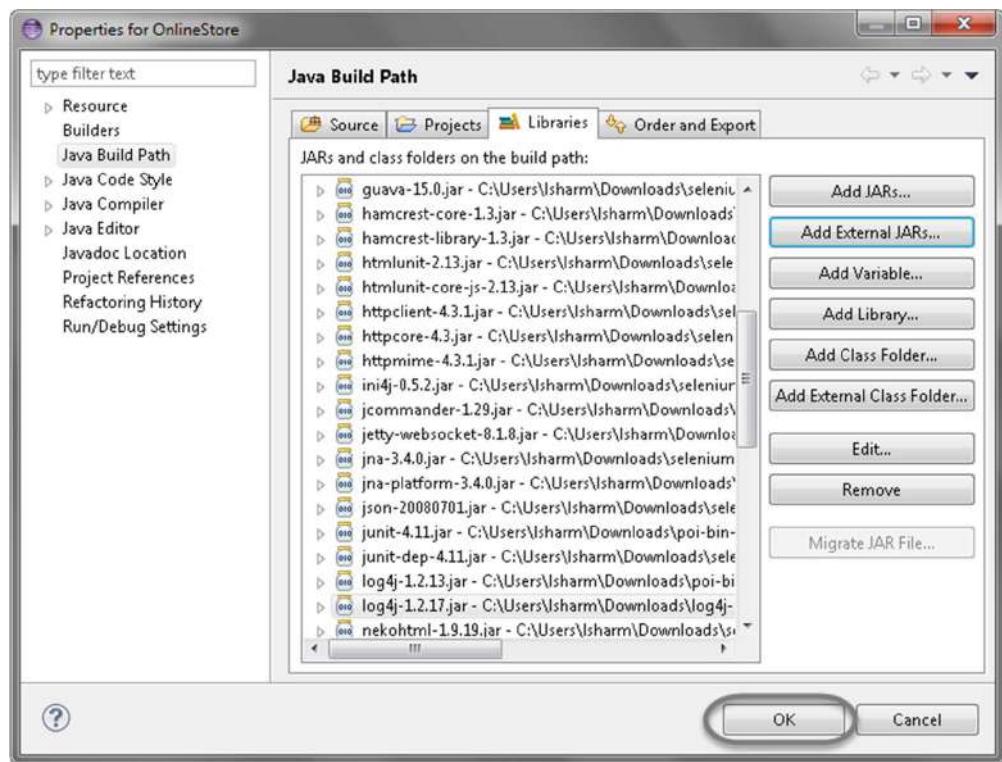
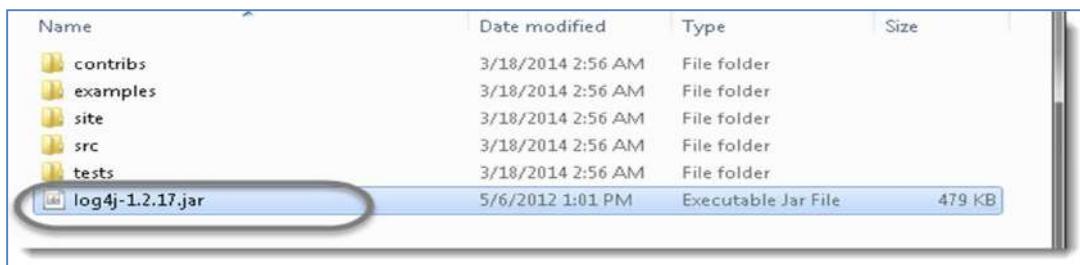
1. Right click the project name and navigate to Build Path and select "Configure Build Path".



- Click on Add External JARS and navigate to the folder where the Log4j jar files are kept.



- Select the Executable Jar File and click Open.



Test Case with Log4j

Following steps are used to create test case with Log4j:

1. Create a new XML file, log4j.xml and place it under the Project root folder and Paste the following code in the log4j.xml file

```
<?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="logfile.log"/>

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="%d %-5p [%c{1}] %m %n"
/>

</layout>

</appender>

<root>

<level value="INFO"/>

<appender-ref ref="fileAppender"/>

</root>

</log4j:configuration>

```

Note: After pasting the code make sure that the code is exactly same, as copying from HTML may change some symbols(“) to (?).

2. Now include logging code in to the test script

```

import java.util.concurrent.TimeUnit;
import org.apache.log4j.Logger;
import org.apache.log4j.xml.DOMConfigurator;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class Log4j {

    private static WebDriver driver;
    private static Logger Log =
        Logger.getLogger(Log4j.class.getName());
    public static void main(String[] args) {
        DOMConfigurator.configure("log4j.xml");
        // Create a new instance of the Firefox driver
        driver = new FirefoxDriver();
        Log.info("New driver instantiated");
    }
}

```

```

//Put a Implicit wait, this means that any search for
//elements on the page could take the time the implicit
//wait is set for before throwing exception

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

Log.info("Implicit wait applied on the driver for 10
seconds");

//Launch the Online Store Website

driver.get("http://www.onlinestore.toolsqa.wpengine.com");
Log.info("Web application launched");

// Find the element that's ID attribute is 'account' (My
//Account)

driver.findElement(By.id("account")).click();
Log.info("Click action performed on My Account link");

// Find the element that's ID attribute is 'log'
Username

// Enter Username on the element found by above desc.

driver.findElement(By.id("log")).sendKeys("testuser_1");
Log.info("Username entered in the Username text box");

// Find the element that's ID attribute is 'pwd'
Password

// Enter Password on the element found by the above
desc.

driver.findElement(By.id("pwd")).sendKeys("Test@123");
Log.info("Password entered in the Password text box");

// Now submit the form. WebDriver will find the form
for //us //from the element

driver.findElement(By.id("login")).click();
Log.info("Click action performed on Submit button");

// Print a Log In message to the screen

System.out.println(" Login Successfully, now it is the
time to Log Off buddy.");

// Find the element that's ID attribute is
// 'account_logout' (Log Out)

driver.findElement(By.id("account_logout"));
Log.info("Click action performed on Log out link");

// Close the driver

driver.quit();

```

```

        Log.info("Browser closed");
    }
}

```

3. Check the output file "logfile.txt". The output will look like below:

Name	Date modified	Type	Size
.settings	3/16/2014 5:26 AM	File folder	
bin	3/20/2014 4:16 AM	File folder	
src	3/20/2014 4:16 AM	File folder	
test-output	3/21/2014 1:28 AM	File folder	
.classpath	3/18/2014 3:20 AM	CLASSPATH File	7 KB
.project	3/16/2014 5:26 AM	PROJECT File	1 KB
log4j.xml	3/20/2014 4:05 AM	XML Document	1 KB
logfile.log	3/21/2014 2:24 AM	Text Document	1 KB



```

logfile.log - Notepad
File Edit Format View Help

2014-03-21 02:23:00,691 INFO [Log] New driver instantiated
2014-03-21 02:23:00,721 INFO [Log] Implicit wait applied on the driver for 10 seconds
2014-03-21 02:23:14,464 INFO [Log] Web application launched
2014-03-21 02:23:14,634 INFO [Log] Click action performed on My Account link
2014-03-21 02:23:21,246 INFO [Log] Username entered in the Username text box
2014-03-21 02:23:21,636 INFO [Log] Password entered in the Password text box
2014-03-21 02:23:21,726 INFO [Log] Click action performed on Submit button
2014-03-21 02:23:25,748 INFO [Log] Click action performed on Log out link
2014-03-21 02:23:25,988 INFO [Log] Browser closed

```

Log4j Log Manager

LogManager, as the name suggests, is the Manager of all logger objects. This is the static class that you refer to for creating Logger objects. LogManager also keeps a list of all the loggers being created by the application. If I were to summarize, LogManager does following work

- Create instances of Logger objects.
- Store references of all the created logger objects.
- Allow reuse of same logger object in different parts of the code.

Let's look at this example code. This shows how we can create different logger instances out of a LogManager.

```

import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

```

```

public class SampleEntry {

    public static void main(String[] args) {

        //This is how we create a logger object
        Logger logger1 = LogManager.getLogger("Logger1");
        Logger logger2 = LogManager.getLogger("Logger2");
        Logger logger3 = LogManager.getLogger("Logger3");
        BasicConfigurator.configure();
        logger1.info("This is logger 1");
        logger2.info("This is logger 2");
        logger3.info("This is logger 3");
    }
}

```

All the logger objects can be retrieve inside LogManager at a particular instance by using the getCurrentLoggers() method. Here is the code sample

```

import java.util.Enumeration;
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

public class SampleEntry {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Logger logger1 = LogManager.getLogger("Logger1");
        Logger logger2 = LogManager.getLogger("Logger2");
        Logger logger3 = LogManager.getLogger("Logger3");
        BasicConfigurator.configure();
        logger1.info("This is logger 1");
        logger2.info("This is logger 2");
        logger3.info("This is logger 3");
    }
}

```

```

Enumeration loggers = LogManager.getCurrentLoggers();
while(loggers.hasMoreElements())
{
    logger3.info("Logger name is " +
loggers.nextElement().getName());
}
}
}

```

One very important property of LogManager it can also retrieve an existing logger object by name. Also, if a logger object will create with the same name as an existing logger object, it will pass on the reference of the existing logger object instead of creating one. This can be shown in the code below

```

import java.util.Enumeration;
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

public class SampleEntry {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Logger logger1 = LogManager.getLogger("Logger1");
        Logger logger2 = LogManager.getLogger("Logger2");
        Logger logger3 = LogManager.getLogger("Logger3");
        BasicConfigurator.configure();
        logger1.info("This is logger 1");
        logger2.info("This is logger 2");
        logger3.info("This is logger 3");

        Logger logger1_2 = LogManager.getLogger("Logger1");
        Logger logger1_3 = LogManager.getLogger("Logger1");
        //You will see that LogManager doesnt create a new
        //instance of logger
    }
}

```

```

//Object with name Logger1, instead passes on the
//reference to the
//existing Logger1 object. We can confirm this with
//following lines

logger1_2.info("The name of this logger is " +
logger1_2.getName();

if(logger1_3.equals(logger1))
{
    logger1_3.info("Both objects are same");
}
else
{
    logger1_3.info("Logger1 and logger1_2 are different
objects");
}

}
}

```

Output

```

1 [main] INFO Logger1 - This is logger 1
2 [main] INFO Logger2 - This is logger 2
2 [main] INFO Logger3 - This is logger 3
2 [main] INFO Logger1 - The name of this logger is Logger1
2 [main] INFO Logger1 - Both objects are same

```

As it can see from the last two lines of output both logger1_2 and logger1_3 are pointing to same logger object which was created at the start of the program. This way same logger can be reuse across different parts of the test code.

Log4j Appenders

Appenders are the Log4j objects which deliver logs to the required destinations. For example a ConsoleAppender will deliver the logs to the console and a FileAppender to the log file. Some types of Appenders are as below:

- **File Appenders:** Almost all the time there is logging of data occur in a file instead of printing it on the console. This is for obvious reasons, as to copy the logs so that it can be keep for reference and browse through it to find problems. Whenever there is a need

to log in a file, a FileAppender will be used. This code sample explains to create a FileAppender object and then set it to the required logger.

```

import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.FileAppender;
import org.apache.log4j.Layout;
import org.apache.log4j.Level;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import org.apache.log4j.SimpleLayout;

public class SampleEntry {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        BasicConfigurator.configure();
        Logger OurLogger =
LogManager.getLogger("OurLogger");

        //create a FileAppender object and
        //associate the file name to which you want the logs
        //to be directed to.
        //You will also have to set a layout also, here
        //We have chosen a SimpleLayout
        FileAppender fileAppender = new FileAppender();
        fileAppender.setFile("logfile.txt");
        fileAppender.setLayout(new SimpleLayout());

        //Add the appender to our Logger Object.
        //from now onwards all the logs will be directed
        //to file mentioned by FileAppender
        OurLogger.addAppender(fileAppender);
        fileAppender.activateOptions();

        //lets print some log 10 times
        int x = 0;
        while(x < 11) {
            OurLogger.debug("This is just a log that I want
to print " + x);
            x++;
        }
    }
}

```

Output

```

DEBUG - This is just a log that I want to print 0
DEBUG - This is just a log that I want to print 1
DEBUG - This is just a log that I want to print 2

```

```

DEBUG - This is just a log that I want to print 3
DEBUG - This is just a log that I want to print 4
DEBUG - This is just a log that I want to print 5
DEBUG - This is just a log that I want to print 6
DEBUG - This is just a log that I want to print 7
DEBUG - This is just a log that I want to print 8
DEBUG - This is just a log that I want to print 9
DEBUG - This is just a log that I want to print 10

```

While creating an appender the Layout is to be added which have been chosen. In this case SimpleLayout() has chosen. Also, whenever there will be a change in Appender object for example adding a file path or adding the Layout you have to call *activateOptions()*. The *activateOptions()* will activate the options set previously. This is important because the changes to Appender object won't take place until *activateOptions()*. The log file will be found in the project folder in the eclipse workspace. Also, this is how the logs look in the log file:

- **Console Appenders:** For testing purpose it may need to redirect the output logs to the console. Actually ConsoleAppender directs the logs to System.err and System.out streams. These streams are also read by Console and hence the output is displayed at the console as well. Let's see with a code sample on how to use ConsoleAppender Object.

```

import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.ConsoleAppender;
import org.apache.log4j.FileAppender;
import org.apache.log4j.Layout;
import org.apache.log4j.Level;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import org.apache.log4j.SimpleLayout;

public class SampleEntry {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        BasicConfigurator.configure();
        Logger OurLogger =
        LogManager.getLogger("OurLogger");

        //create a ConsoleAppender object
        //You will also have to set a layout also, here
        //We have chosen a SimpleLayout
        ConsoleAppender ConsoleAppender = new
        ConsoleAppender();
        ConsoleAppender.setLayout(new SimpleLayout());

        //Add the appender to our Logger Object.
    }
}

```

```

//from now onwards all the logs will be directed
//to file mentioned by FileAppender
OurLogger.addAppender(ConsoleAppender);
ConsoleAppender.activateOptions();

//lets print some log 10 times
int x = 0;
while(x < 11){
    OurLogger.debug("This is just a log that I want to
print " + x);
    x++;
}
}
}

```

This way it will redirect all the logs to the console and can see the output in the console.

- JDBC Appender: JDBCAppenders are used to write logs to a Data base. These appenders accept data base connection credentials to connect to DB. Let's see a code sample to understand.

```

import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.ConsoleAppender;
import org.apache.log4j.FileAppender;
import org.apache.log4j.Layout;
import org.apache.log4j.Level;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import org.apache.log4j.SimpleLayout;
import org.apache.log4j.jdbc.JDBCAppender;

public class SampleEntry {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        BasicConfigurator.configure();
        Logger OurLogger =
        LogManager.getLogger("OurLogger");

        //create a JDBCAppender class instance
        JDBCAppender DataBaseAppender = new JDBCAppender();
        //Provide connection details to the
        //Database appender

        DataBaseAppender.setURL("jdbc:mysql://localhost/test");
        //Connection url
        DataBaseAppender.setUser("User1"); //Username for the
        DB connection
        DataBaseAppender.setPassword("ThisPassword");
        //Password for the DB connection
    }
}

```

```

dataBaseAppender.setDriver("com.mysql.jdbc.Driver"); //  

Drivers to use to connect to DB

//set the sql insert statement that you want to use
dataBaseAppender.setSql("INSERT INTO LOGS VALUES ('%x',  

now() ,'%C','%p','%m')");

//activate the new options
dataBaseAppender.activateOptions();

//Attach the appender to the Logger
OurLogger.addAppender(dataBaseAppender);

int x = 0;
while(x < 11) {
    OurLogger.debug("This is just a log that I want to
print " + x);
    x++;
}
}
}

```

This code explains how to set up a JDBCAppender object and use it for logging. In the code the insert statement has to be given to the JDBCAppender. This statement is used to insert logs in the desired database table. Here the statement INSERT INTO LOGS VALUES ('%x', now(), '%C', '%p', '%m') is used. It says that logs are inserted in the table named LOGS.

Log4j Layouts

At times, the user wishes certain information to be pre - pended or appended with each log statement. For example, to print a timestamp along with the log statement. Thus, such requirements can be accomplished by “Layouts”.

Layouts are a utility that allows the user to opt for the desired format in which the logs would be rendered. Appenders and Layout have a tight coupling between them. Thus, it required to map each of the appenders with a specific layout. A user is leveraged to define multiple appenders, each map with a distinct layout. The user is leveraged to specify the desired pattern in the value of the ConversionPattern parameter.

```

<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%d{dd-MM-yyyy HH:mm:ss}%-5p[%c{1}]: %m%n"/>
</layout>

```

Layout class help us define how the log information should appear in the outputs. Here is a sample code which uses PatternLayout Class to change the formatting of logs:

```

import java.util.Enumeration;
import org.apache.log4j.Appender;

```

```
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.ConsoleAppender;
import org.apache.log4j.Layout;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import org.apache.log4j.PatternLayout;

public class SampleEntry {

    //All the loggers that can be used
    static Logger mainLogger =
    LogManager.getLogger(SampleEntry.class.getName());

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        BasicConfigurator.configure();
        ConsoleAppender appender = new ConsoleAppender();
        appender.activateOptions();
        PatternLayout layoutHelper = new PatternLayout();
        layoutHelper.setConversionPattern("%-5p [%t]: %m%n");
        layoutHelper.activateOptions();

        mainLogger.getAppender("ConsoleAppender").setLayout(layoutHelper);
        appender.setLayout(layoutHelper);
        mainLogger.addAppender(appender);
        //Create a console appender and attach it to our
        //mainLogger
        mainLogger.info("Pattern 1 is displayed like this");
        layoutHelper.setConversionPattern("%C %m%n");
        mainLogger.info("Pattern 2 is displayed like this");

    }

}
```

Expected output on the console will be

```
INFO [main]: Pattern 1 is displayed like this  
Log4jSample.SampleEntry
```

```
Pattern 2 is displayed like this  
Manual Layout  
Selenium Layout
```

DUCAT®
www.ducatindia.com

Summary

- Log4j is a reliable, fast and flexible logging framework (APIs) written in Java, which is distributed under the Apache Software License
- Log4j is highly configurable through external configuration files at runtime
- Log4j allows a very good logging infrastructure without putting in any efforts
- Log4j gives the ability to categorize logs at different levels
- Log4j gives the ability to direct logs to different outputs. For e.g. to a file, Console or a Database
- Log4j gives the ability to define the format of output logs
- Log4j gives the ability to write Asynchronous logs which helps to increase the performance of the application

DUCAT®
www.ducatindia.com

Chapter 13 – Database Connections

Database Connection

The connectivity to an application database is required by Selenium Webdriver for testing web applications and perform many operations such as information retrieval, submission and validation. For example the database connection is required to verify that the UI data is exact copy of Database table. The reasons to connect to a database are as:

- To get test data: If we automate the database, we can directly fetch the test data from database and then work on them in test automation script
- To verify result: In automation we can verify the front end result with backend entry in the database
- To delete test data created: In automation it is good practice to delete the test data created, using database automation, we directly fire the delete query to delete the test data created
- To update certain data: As per the need of test script, the test data can be updated using update query

The selenium don't provide facility to directly connect to database but instead use database driver APIs to connect to database. One such database driver is Java Database Connectivity (JDBC) which provides APIs to connect and interact with Database. Other such platform independent driver is ODBC. Since the connectivity to a database is through database drivers, the selenium webdriver can connect to various database such as MSSQL, MYSQL, ORACLE, SYBASE.

Database connectivity

The steps to connect to a database using database drivers are as below. The detail of each step will discussed in later sections of this chapter.

1. Load and Registering the Driver: This step requires loading the database driver into the selenium script such as JDBC, ODBC.
2. Establishing Connection: The traditional way to establish a connection with a database is to call the driver registered with the project. For example, JDBC provides DriverManager class to connect to database as seen below:

```
DriverManager.getConnection(URL, "username", "password")
```

Here,
 URL syntax is : jdbc:<subprotocol>:<subname>
 Where,
 <subprotocol> is the name of the driver or the name of a database connectivity mechanism.
 <subname> is the point of a subname is to give enough information to locate the data source .It may includes IP address , Port number and exact name of DataSource.
 Example of URL is
`jdbc:mysql://localhost:3306/TestMySQL`
3. Creating Statement Object: A Statement object is used to send SQL statements to a database over the created connection.
4. Execute the Statement: The SQL statements are executed and results are stored in a ResultSet object. This Java object contains the results of executing an SQL query.

5. Closing the connection: The connection to database are closed using the methods provide the database driver. For example, JDBC provide *close()* method to close the database connection made by *getConnection()* method.

Example: Below program provide an insight into a JAVA program which connects to a mysql database using a JDBC driver.

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class connectingToMySQLDBExample {
    private static String userName;
    private static String password;
    private static String dbURL;
    private static Connection connection;
    public static void main(String[] args) {
        try {
            userName = "username";
            password = "password";
            dbURL = "jdbc:mysql://artoftesting.com/testDB";
            try {
                Class.forName("com.mysql.jdbc.Driver");
            }
            catch (ClassNotFoundException e) {
                System.out.println("MySQL JDBC driver not found.");
                e.printStackTrace();
            }
            try {
                connection = DriverManager.getConnection(dbURL,
                    userName, password);
                Statement st = connection.createStatement();
                String sqlStr = "select * from testTable";
                ResultSet rs = st.executeQuery(sqlStr);
                while (rs.next()) {
                    System.out.println(rs.getString("name"));
                }
            }
        }
    }
}

```

```
        Connection.close();

    } catch (SQLException e) {
        System.out.println("Connection to MySQL db
                           failed");
        e.printStackTrace();
    }

} catch (Exception e) {
    e.printStackTrace();
}

}
```

The output of the program depends upon the connectivity to the database and the value of the name column in the database table 'testTable'. Hence the output is not explicitly written based on the assumption.

Code Explanation:

The first bold line of the code establishes the connection to the database. Here it is trying to connect to a mysql database using JDBC driver, The JDBC driver is loaded in the second bold statement. The third and fourth bold statement create statement object. The fifth statement executes the statement while the last bold statement closes a database connection.

Note: The same example may be referred in rets of sections of the chapter.

Database Testing in Selenium Using MySql Server

In rest of chapter you will see use of JDBC driver in examples to connect to databases. The JDBC will be explained in later part of this chapter however a brief about JDBC is written in order to explain against the context of this section.

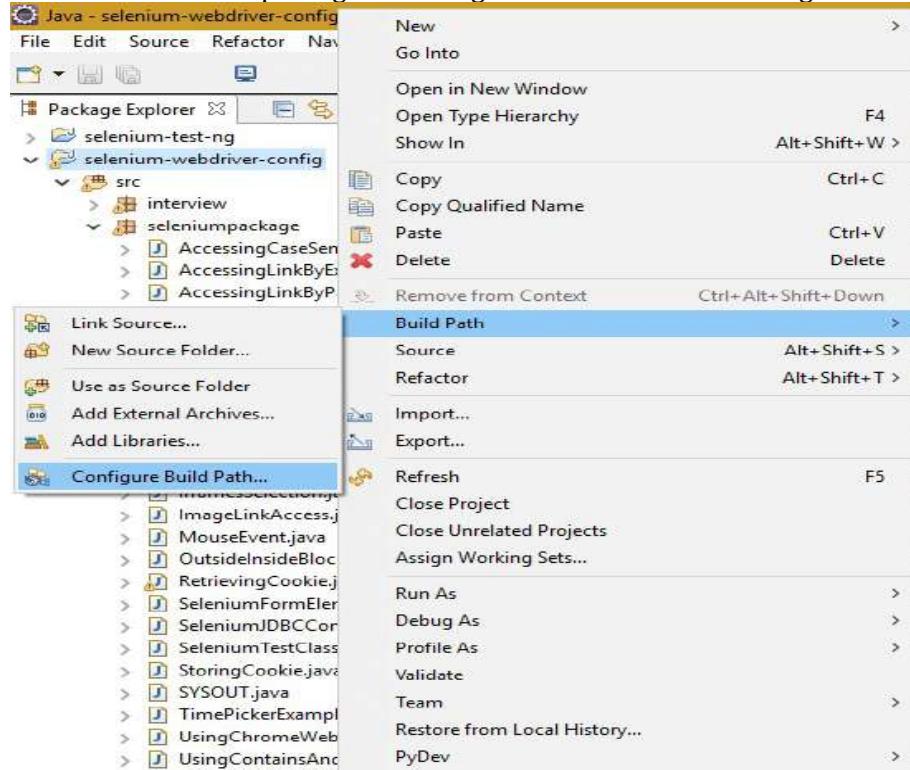
The Java Database Connectivity (JDBC) API provides universal data access from the Java programming language. Using the JDBC API, you can access virtually any data source, from relational databases to spreadsheets and flat files. It lets the user connect and interact with the database and fetch the data based on the queries you use in the automation script. In other words JDBC Driver facilitates:

1. Establishing a Database connection
 2. Sending SQL Queries to the Database
 3. Processing the results

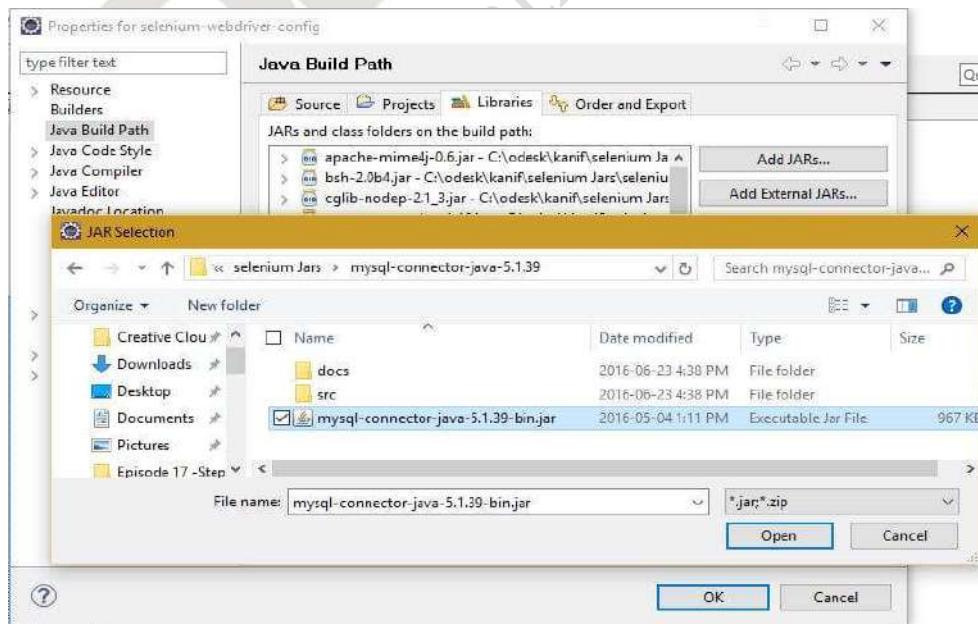
JDBC requires drivers for database it connect to. You need to download the MySQL driver for JDBC from the MySQL site. This MySQL driver contains the required class files and Java source files to be used by JDBC driver. The download link is:
(<https://dev.mysql.com/downloads/connector/j/>)

Once you download the zip file ‘mysql-connector-java-5.1.39.zip’, unzip to see the jar inside which is actual driver file. The further steps to add this driver in Eclipse are as below:

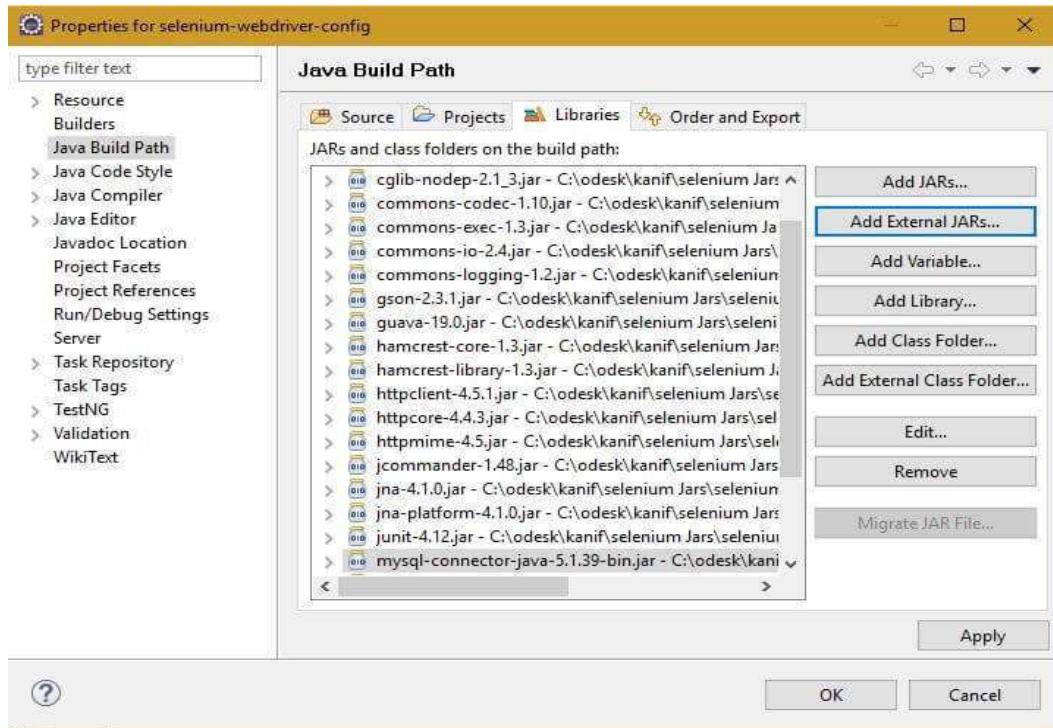
1. Right-click on the selenium package and navigate as 'Build Path' > 'Configure Build Path'.



2. Click on the 'Libraries' tab and click on the 'Add External JARs' button. Select the path to the unzipped 'mysql-connector-java-5.1.39.jar' file. Click on the Open button.



3. The build path set up will show up as below



Now, that MySQL driver is added in the project, next step is to make a sample program to connect to MySql Database using JDBC.

```
package seleniumpackage;
import java.sql.Connection;
import java.sql.Statement;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import java.sql.ResultSet;
import java.sql.DriverManager;
import java.sql.SQLException;
public class SeleniumJDBCConnection {
    public static final String QUERY = "select * from WEB_TESTING;";
    public static void main(String[] args) throws
    ClassNotFoundException, SQLException {
        String baseWebUrl = "";
        String expectedWebsiteTitle = "";
        //Load MySQL JDBC driver
        Class.forName("com.mysql.jdbc.Driver");
        //Create Connection to DB
    }
}
```

```

Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/webtesting"
, "MySQLDatabase", "softwaretestingclass");
Statement stmt = con.createStatement(); //Create Statement Object.
ResultSet rs= stmt.executeQuery(QUERY); // Execute the SQL Query.
Store results in ResultSet.

while (rs.next()) {
// While Loop iterates through all data present in a table.

    baseWebUrl = rs.getString(1);
    expectedWebsiteTitle = rs.getString(2);
}

con.close();
System.out.println("baseWebUrl: "+baseWebUrl);
System.out.println("expectedWebsiteTitle: "+expectedWebsiteTitle);
WebDriver driver = new FirefoxDriver();
String actualWebsiteTitle = "";
driver.get(baseWebUrl);
actualWebsiteTitle = driver.getTitle();
if (actualWebsiteTitle.contentEquals(expectedWebsiteTitle)){
System.out.println("Test has Passed!");
} else {
System.out.println("Test has Failed!");
}
driver.close();
}
}

```

Output:

```

baseWebURL: https://www.facebook.com/
expectedWebsiteTitle: Facebook - Log In or Sign Up
Test has passed!

```

Code Explanation:

First the JDBC driver for MYSQL is loaded by importing `import java.sql.DriverManager` and making reference to class "`com.mysql.jdbc.Driver`". For registering the Driver we have loaded the Driver class using `forName()` method. The `forName()` is the static factory method which is present in predefined class called "Class". This method loads the class which is mentioned as parameter, here it is `com.mysql.jdbc.Driver`. Internally this Driver class will register the driver by using static method called `registerDriver()`. The database connection is made in below line of code by calling all static method called `getConneciton()` present in `DriverManager` Class.

```
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/webtesting",
"MySQLDatabase","softwaretestingclass");
```

This *DriveManager* class requires URL, username, and password. Once the database connection is established, SQL queries can be executed via JDBC connection. Here, we have created a SQL to select a record (here URL and title) from the database. With the help of Statement interface and '*createStatement*' method of Connection interface which is then executed with the help of '*executeQuery*' method. This method returns a result set that contains URL and title.

Configuring SQL

The various SQL statements which can be used to interact with the Database are listed below. Please note that SQL statements are universal statements which can be used to interact with any kind of database. The syntax of SQL statements remains same everywhere. Below list provides a basic overview of making SQL statements.

- **Creating Statement Object:**

For creating statement object we need to call a method called *createStatement()* which is present in Connection Interface. This method returns Statement object and it is no argument method.

Example:

```
con.createStatement();
```

- **Executing Queries:**

For executing queries there are different methods present in Statement Interface for selecting and updating records.

- Selecting records: for executing select queries, we call a method called *executeQuery()*. This method takes string as parameter and returns *ResultSet* object.

```
Resultset rs= st.executeQuery("Select * from Employee");
```

**We will read about ResultSet object in later part of this chapter.*

- Updating records: To update records in a table we use a method called *executeUpdate()*. This method takes string as parameter and returns integer value as how many records have been updated.

Example:

```
int result = st.executeUpdate("update Employee set
EmpName='abc' where EmpId=2");
```

- **Closing the Connection:**

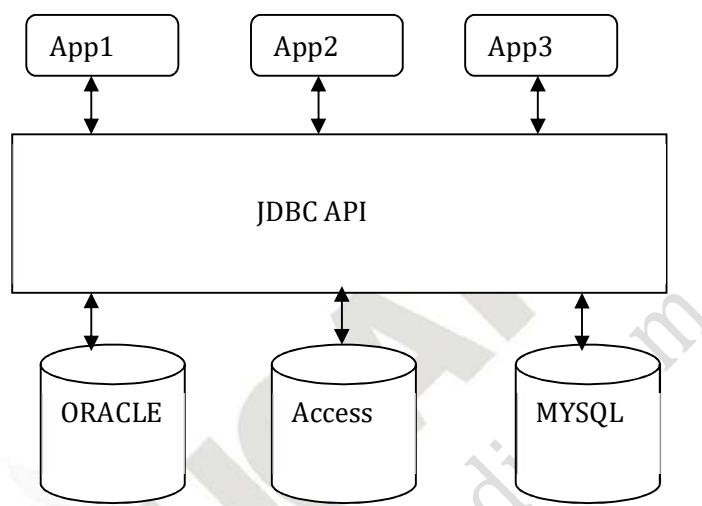
Once execution of all statements were completed we need to close all the connections made to the Database by using method called *close()* present in Connection interface.

Example:

```
con.close();
```

JDBC Driver

Java Database Connectivity (JDBC) is a Java API to interact with the data sources. Here word interaction implies to the connection and retrieving and updating data in data sources. The data sources can be anything kind of tabular data such as relational database, spreadsheets and flat files. JDBC provides APIs to connect with the data sources. The APIs uses the drivers to connect to different type of data sources such as EXCEL, MySQL, MSSQL, ORACLE. You are required to download drivers for each type of data sources and include in your project to have the APIs interact with selected data sources. The pictorial representation of JDBC driver is :



JDBC APIs

The JDBC APIs provides universal data access from the Java programming language. Using the JDBC API, you can access various data source, from relational databases to spreadsheets and flat files. The JDBC API consists of a set of classes and interfaces written in the Java programming language. The JDBC API is comprised of two packages, `java.sql` and `javax.sql`.

JDBC API using database drivers provides below functionality to a developer.

1. APIs to connect to the data source.
2. APIs to execute queries in form of `SELECT`, `INSERT` and `UPDATE` statements.
3. APIs to retrieve the result received from the data source.

JDBC drivers

JDBC Driver is a software component that enables JAVA application to interact with the data source. JDBC drivers are client-side adapters that convert requests from Java programs to a protocol that the data source can understand. The data sources can be anything kind of tabular data such as relational database, spreadsheets and flat files. There are four types of JDBC drivers:

- **JDBC-ODBC bridge driver:** The JDBC-ODBC bridge driver uses Open Database Connectivity (ODBC) driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. Note that ODBC is also an API for accessing DBMS. Before JDBC, ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language

and thus is platform dependent and unsecured. Due to this reason this type of driver is not used and is unsupported Oracle. The advantages and disadvantages of JDBC-ODBC bridge driver.

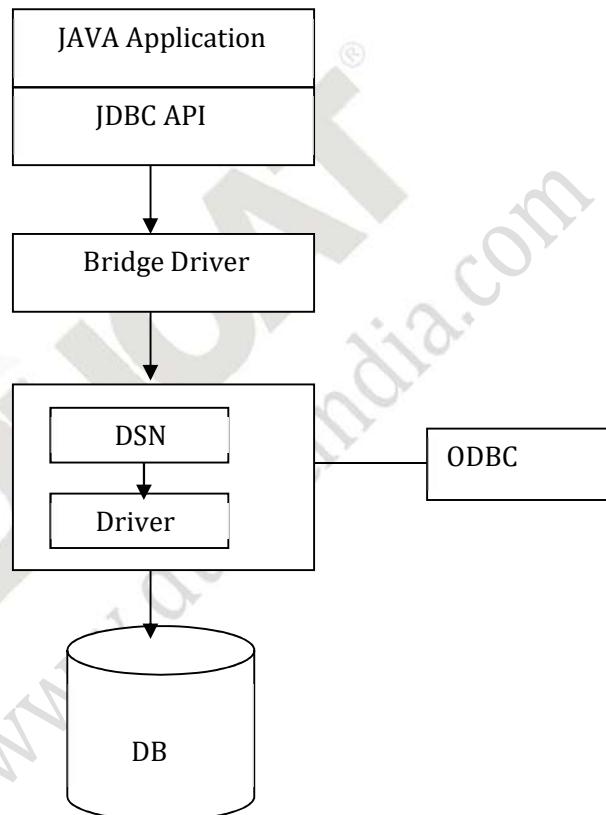
Advantages:

- It is easy to use.
- It can easily connect to any database.

Disadvantages:

- It offers low performance as JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

The pictorial representation is:



- **Native-API driver:** The Native API driver uses the client-side libraries of the database. This driver need local installation of native database binary code and configured to work. The driver then converts JDBC method calls into native calls of the database API. For example, in order to use MYSQL the driver would require MYSQL binaries installation locally to have APIs available to use by the driver. The advantages and disadvantages Native-API driver are:

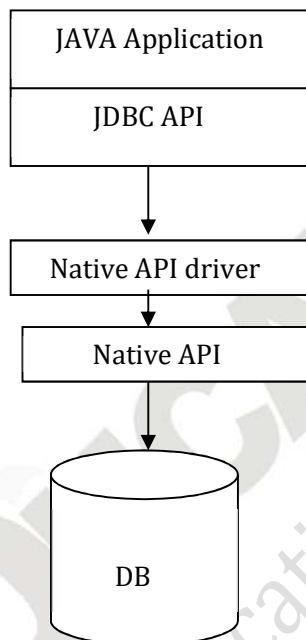
Advantages:

- It gives higher performance than JDBC-ODBC bridge driver.

Disadvantages:

- The Native database needs to be installed on each client machine.
- The Vendor client library needs to be installed on client machine.

The pictorial representation is:



- **Network Protocol driver:** The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. Thus the calls are passed through the network to the middle-tier server. The middle-tier then translates the request to the database. This driver is most flexible JDBC solution as they do not require any native binary code on the client. The advantages and disadvantages of Network Protocol driver are:

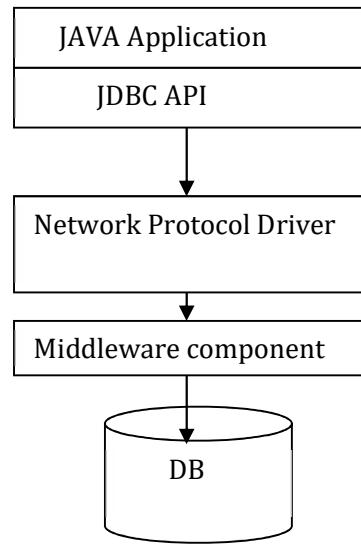
Advantages:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

The pictorial representation is:



- **Thin driver:** The thin driver converts JDBC calls directly into the vendor-specific database protocol. It uses JAVA networking libraries to communicate directly with the database server and thus require no client installation or configuration. Since the protocols are vendor-specific and proprietary, DBMS vendors are generally the only companies providing this driver. The advantages and disadvantages Thin driver are:

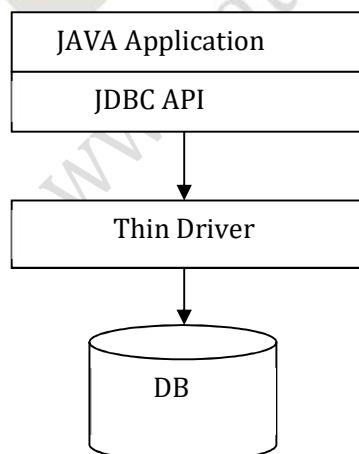
Advantages:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantages:

- Drivers depend on the Database.

The pictorial representation is:



JDBC API Classes and Interfaces

The Java API interacts with the database after procuring a data source connection and executing SQL statements in case of database. This API connects the Java programming languages with various databases with the help of their database drivers. Following are the classes and interfaces that are provided by the JDBC API.

- **Driver Manager:** This class returns a database connection object after accepting DB connection URL, username and password.
- **Driver:** This is database specific driver that helps in creating JDBC connection with the database.
- **Connection:** It is an interface that is able to provide information describing its table, its supported SQL grammar, its stored procedures, the capabilities of this connection.
- **Statement:** It is an interface to the pre-compiled object that is used to efficiently execute SQL statements into the database.
- **ResultSet:** It is an interface to the object that maintains a cursor pointing to its current row of data.
- **SQLException:** It is an exception class that defines the various SQL exceptions. It is mandatory to catch SQL exception through this class when you execute any SQL statement via JDBC connection.

Understanding Connection Interface

A connection interface provides a standard abstraction to access the session established with a database server. This interface represents a session between a JAVA application and a database. All the SQL statements which are executed, the results are returned within the context of a Connection object. Connection interface is mainly used to create *java.sql.Statement*, *java.sql.PreparedStatement* and *java.sql.CallableStatement* objects. You can also use it to retrieve the metadata of a database like name of the database product, name of the JDBC driver, major and minor version of the database. A connection object can be obtained by using the *getConnection()* method of the *DriverManager* class. The syntax of *getConnection()* method is

```
Connection con=DriverManager.getConnection(url,username,password);
```

The commonly used methods of Connection Interface are:

- public Statement createStatement(): It creates a statement object that can be used to execute SQL queries.
- public Statement createStatement(int resultSetType,int resultSetConcurrency): It creates a Statement object that will generate ResultSet objects with the given type and concurrency.
- public void setAutoCommit(boolean status): It is used to set the commit status.By default it is true.
- public void commit(): It saves the changes made since the previous commit/rollback permanent.
- public void rollback(): It drops all changes made since the previous commit/rollback.
- public void close(): It closes the connection and Releases a JDBC resources immediately.

It is required explicitly to close all the connections associated with connection object with database made during the program using the `close()` method. The syntax of `close()` method is:

```
conn.close();
```

Statement Interface

The Statement interface provides a standard abstraction to execute SQL statements and return the results using the `ResultSet` objects. It provides methods to execute queries with the database.

Type of Statements

The JDBC API provides three different interfaces to execute the different types of SQL queries

- **Statement:** This type is used to execute normal SQL queries.
- **PreparedStatement:** This type is used to execute dynamic or parameterized SQL queries.
- **CallableStatement:** This type is used to execute the stored procedures.

Statement Interface

Statement interface is used to execute normal SQL queries. You can't pass the parameters to SQL query at run time using this interface. A Statement object contains a single `ResultSet` object at a time. The `execute` method of a Statement implicitly close its current `ResultSet` if it is already open. Statement object can be created by using the `createStatement()` method of the Connection interface. Lets take one small example to understand `createStatement()`.

```
Statement st = con.createStatement();
```

The above statement defines 'st' object of type Statement. The 'con' is a connection object which establishes connection to the required data source. At this time only object is created. This object need to be executed along with SQL statement. You can see below example as how the Statement object is executed and result assigned to `ResultSet`.

```
ResultSet rs = st.executeQuery("select * from employeeList");
while (rs.next()) {
    System.out.println(rs.getString("empname"));
}
```

In above example, it can be seen that 'st' object has a method as `executeQuery()` which has SQL statement as an argument. This result set after the execution is assigned to the `ResultSet` object 'rs'. The The `ResultSet` provided methods to iterate through the result set and perform required actions.

Type of execute methods

The Statement object has three types of execute methods as below:

- **boolean execute (String SQL):** Returns a boolean value of true if a `ResultSet` object can be retrieved; otherwise, it returns false. This method to execute SQL DDL statements or when it need to use truly dynamic SQL.
- **int executeUpdate (String SQL):** Returns the number of rows affected by the execution of the SQL statement. This method is used to execute SQL statements to get a number of rows affected.

- **ResultSet executeQuery (String SQL):** Returns a ResultSet object. This method is used when it expect to get a result set, with a SELECT statement.

Just as a Connection object can be closed, the Statement object should also be closed by using the *close()* method. The syntax of *close()* method is same as that explained in connection interface section of this chapter.

Prepared Statement Interface.

PreparedStatement is used to execute dynamic or parameterized SQL queries.

PreparedStatement extends Statement interface. PreparedStatements are pre-compiled and hence their execution is much faster than that of Statements. In *PreparedStatement*, the SQL is passed in the *prepareStatement()* method while creating the *PreparedStatement* and leave the execute method empty. However, the SQL statement can be overridden which is passed in *prepareStatement()* by passing another SQL in the *executeQuery()* method. Let's take one small example to understand *PreparedStatement()*.

```
PreparedStatement st=con.prepareStatement("insert into employeeList
values ('Heartin',2)");
st.executeUpdate();
```

The above statement defines 'st' object of type *PreparedStatement*. The 'con' is a connection object which establishes connection to the required data source. At this time not only object is created but the SQL statement is also written that need to be executed. In next line the 'st' object is executed using *executeQuery()* method.

PreparedStatement also has a set of methods such as *setString()*, *setInt()*, which can be used to parameterize a *PreparedStatement*. Lets see below example:

```
PreparedStatement ps1=con.prepareStatement("insert into employeeList
values (?,?)");
ps1.setString(1, "Heartin4");
ps1.setInt(2, 7);
ps1.executeUpdate();
```

In this example, the SQL Insert query requires two values which are passed to the SQL using *setString()* and *setInt()* methods. Also note that set methods should be called before calling the *executeUpdate()* method.

Just as you close a Statement object, you should also close the *PreparedStatement* object with the *close()* method.

ResultSet Interface

By now you know that the SQL SELECT statements read data from a database query and return the data in an object. This object is called as '*ResultSet*'. The *java.sql.ResultSet* interface represents the *ResultSet* of a database query. This *ResultSet* maintains data in row and column format. In order to access row and columns the object maintains a cursor that points to the current row in the *ResultSet*. This cursor is movable forward and backwards based on the properties of the *ResultSet*.

Type of ResultSet

There are three types of Result set:

1. `ResultSet.TYPE_FORWARD_ONLY`: The cursor can only move forward in the result set. This is default *ResultSet* type.
2. `ResultSet.TYPE_SCROLL_INSENSITIVE`: The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
3. `ResultSet.TYPE_SCROLL_SENSITIVE`: The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

Example:

```
Statement stmt =
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);
```

Methods of ResultSet

The methods of the *ResultSet* interface can be defined into three categories:

1. Navigational methods: These methods are used to move the cursor around.
2. Get methods: These methods are used to view the data in the columns of the current row being pointed by the cursor.
3. Update methods: These methods are used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well

Based on above categories, JDBC provides the following connection methods to create statements with desired *ResultSet*:

1. `createStatement(int RSType, int RSConcurrency)`;
2. `prepareStatement(String SQL, int RSType, int RSConcurrency)`;
3. `prepareCall(String sql, int RSType, int RSConcurrency)`;

Here, the first argument indicates the type of a *ResultSet* object and the second argument is one of two *ResultSet* constants for specifying whether a result set is read-only or updatable.

Firing Queries Using Java on Eclipse

The steps of Firing Queries using Java on Eclipse are as below:

1. Open Eclipse IDE and Select Database Perspective (Windows >> Open Perspective >> Other).
2. Create Connection Profile: Here profile can be create for connecting to DB.
Method 1: Right click on “Database Connections” and click New.
Method 2: Click on “New Connection Profile” button in “Data Source Explorer” view.
3. Select DB to connect from popup window. And provide connection profile name (JBT in this case).
4. Click Next. In next screen provide DB connection details.
5. Once provide it can check if connection to DB is successful or not by clicking “Test Connection” button. If everything is fine it should show Ping Succeed popup window. If not then check if database is responding and is available to connect.
6. After providing the connection related details click Finish. If everything is fine then a new Connection profile named JBT will be created in Data Source Explorer view
7. To execute a query “SQL Scrapbook” is used. To open SQL scrapbook click scrapbook button on top.
8. It will open new SQL scrap book, from there it can execute sql query. To view the result of the query fired it will require “SQL Results” view which can be opened from Windows >> Show View >> Other.
9. Execution plan of the query can also be checked via Eclipse IDE. To view Execution plan “Execution Plan” view needs to be opened. Eclipse IDE can show execution plan in two ways.

Summary

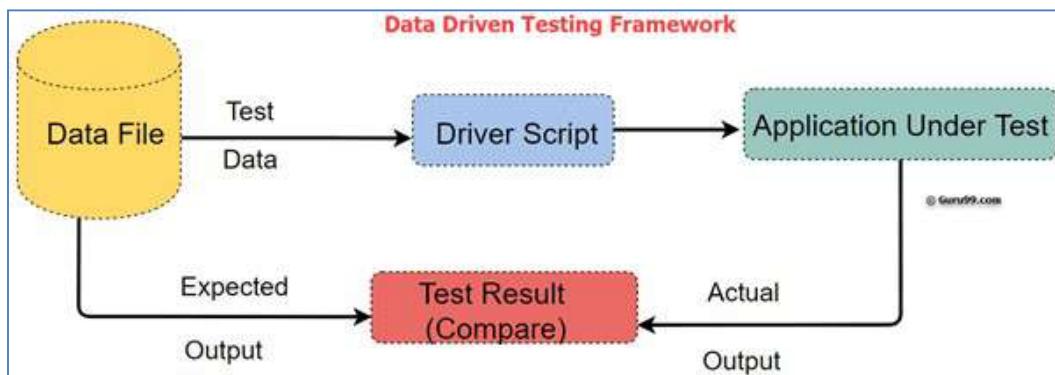
- The selenium don't provide facility to directly connect to database but instead use database driver APIs to connect to database
- The Java Database Connectivity (JDBC) API provides universal data access from the Java programming language
- JDBC Driver is a software component that enables JAVA application to interact with the data source
- A connection interface provides a standard abstraction to access the session established with a database server
- The Statement interface provides a standard abstraction to execute SQL statements and return the results using the ResultSet objects

DUCAT®
www.ducatindia.com

Chapter 14 – Data Driven Automation Framework

Data-driven is a test automation framework which stores test data in a table or spreadsheet format. This allows automation engineers to have a single test script which can execute tests for all the test data in the table.

In this framework, input values are read from data files and are stored into a variable in test scripts. DDT (Data Driven testing) enables building both positive and negative test cases into a single test. In Data-driven test automation framework, input data can be stored in single or multiple data sources like xls, XML, csv, and databases.



What is Automation Framework?

A framework is considered to be a combination of set protocols, rules, standards and guidelines that can be incorporated or followed as a whole so as to leverage the benefits of the scaffolding provided by the Framework. An “Automation Framework” is a set of guidelines like coding standards, test-data handling which provide an execution environment for the automation test scripts. The framework provides the user with various benefits that help them to develop, execute and report the automation test scripts efficiently.

In other words, a framework is a constructive blend of various guidelines, coding standards, concepts, processes, practices, project hierarchies, modularity, reporting mechanism, test data injections to pillar automation testing. The framework enable a user to follow these guidelines while automating application to take advantages of various productive results such as ease of scripting, scalability, modularity, understandability, process definition, re-usability, cost, maintenance.

Features of Automation Framework

The various features of Automation Framework can be categorized as below:

General features

- Re-use of existing code by writing code once and invoking it elsewhere
- Consistency of test automation implementation by defining standards
- Tool independence by implementing the framework as separate software
- Ease of framework installation and maintenance
- Ease of upgrading the deployed framework

- Supported operating systems and browsers

Features related to test design

- Simple to learn and use
- Common operations already built-in the framework
- Minimal changes to the test automation in case the application changes
- Integration with version control systems and test management systems

Features related to test execution

- Ability of the framework to run on multiple systems
- Support for test configuration e.g. application name, server name/ URL and test cases to be executed
- Support for scheduled execution
- Supported invocations of the framework e.g. by GUI or by command-line
- Support for invoking and closing/ shutting down the application(s) under test
- Ability to connect to myriad test data sources and collect test data
- Error recovery e.g. by dismissing unknown windows, retrying a failed operation with/ without configurable delays, proceeding to the next step/ test case and even resetting or re-starting the application(s) under test
- Integration with defect tracking systems
- Clean up test data in the application(s) under test after completing each test

Features related to test reporting

- Logging states and results
- Reporting test results (in one or multiple formats e.g. Excel or HTML) with metrics such as number of test cases in the test plan, test cases executed, test cases passed, test cases failed and the time taken to execute the test cases
- Distributing test results e.g. by publishing to web or email

Benefits of Using Automation Framework

The benefits of automation framework are:

- Ease of scripting: An automation framework in place ensures consistent coding and that best practices are followed. Standard scripting will result in team consistency during test library design and prevent individuals from following their own coding standards, thus avoiding duplicate coding. It helps to streamline testing projects that involve a group of testers working on multiple features of an application.
- Scalable: Whether multiple web pages are being added or Objects or data, a good automation framework design is scalable. A framework should be much easier to extend to larger projects.
- Modularity: Automation framework ensures that the complex task is broken down to smaller manageable tasks which can then be re-used whenever required.
- Understandability: An automation framework enables quick transition (or understand) the overall architecture & bring people up-to-speed.

- Re-usability: Common library files can be reused when required, no need to develop them every time. Modularity allows testers to re-use common modules in different scripts to avoid unnecessary & redundant tasks.
- Cost & Maintenance: A well designed automation framework helps in maintaining the code in light of common changes like Test data, Page Objects, Reporting structure, etc.
- Better error handling: A good automation framework helps us catch different recovery scenarios and handle them properly.
- Minimal manual intervention: Automation framework can take care of mundane & redundant manual tasks.
- Easy Reporting: The reporting module within framework can handle all the report requirements. Any changes, if required, can be incorporated in the same module for easy & efficient reporting. A framework allows for user friendly interface and reporting options.
- Segregation: A framework helps segregate the test script logic and the corresponding test data. The Test data can be stored into an external database like property files, xml files, excel files, text files, CSV files, ODBC repositories etc.
- Test configuration: Test suites covering different application modules (or functionalities) can be configured easily using an automation framework.
- Continuous integration: An automation framework helps in today's time of continuous integration.

Different Types of Automation Framework

There is a divergent range of Automation Frameworks available. These frameworks differ from each other based on their support to different key factors to do automation like reusability, ease of maintenance. The few most popularly used Test Automation Frameworks are as below:

- **Module Based Testing Framework:** In the modular testing framework, testers create test scripts on module wise by breaking down the complete application under test into smaller, independent tests. These individual test scripts can be combined to make larger test scripts by using a master script to achieve required scenarios. This master script is used to invoke the individual modules to run end to end test scenarios. In this framework, testers write function libraries to use it whenever required.
- **Linear Architecture Testing Framework:** Linear Architecture Framework is a basic level test automation framework which is in the form of 'Record and Playback' in a linear fashion. This framework is also known as 'Record and Playback' framework. This type of framework is used to test small sized applications. In this type, creation, and execution of test script are done individually for each test case individually.
- **Data Driven Testing Framework:** Data driven test automation framework is focused on separating the test scripts logic and the test data from each other. It allows to create test

automation scripts by passing different sets of test data. The test data set is kept in the external files or resources such as MS Excel Sheets, MS Access Tables, SQL Database, XML files. The test scripts connect to the external resources to get the test data. This framework gives more test coverage with reusable tests and flexibility in execution of tests only when required and by changing only the input test data.

- **Keyword Driven Testing Framework:** It is also known as table-driven testing or action word based testing. In Keyword-driven testing, a table format is used to define keywords or action words for each function or method. It performs automation test scripts based on the keywords specified in the excel sheet. By using this Framework, testers can work with keywords to develop any test automation script, testers with less programming knowledge would also be able to work on the test scripts. The logic to read keywords and call the required action mentioned in the external excel sheet is placed in the main class.
- **Hybrid Testing Framework:** Hybrid Test automation framework is the combination of two or more frameworks mentioned above. It attempts to leverage the strengths and benefits of other frameworks for the particular test environment it manages.
- **Behavior Driven Development Framework:** The purpose of this Behavior Driven Development framework is to create a platform which allows everyone (such as Business Analysts, Developers, Testers) to participate actively. It requires increased collaboration between Development and Test Teams.

What is the Page Object Model?

The Page Object Model is a design pattern of testing, derived from the Object Oriented Programming concepts. The POM describes the web application into the number of web pages being used, and contains the elements as properties, and actions as methods. This offers low maintenance on the tests developed.

Creating the test automation framework

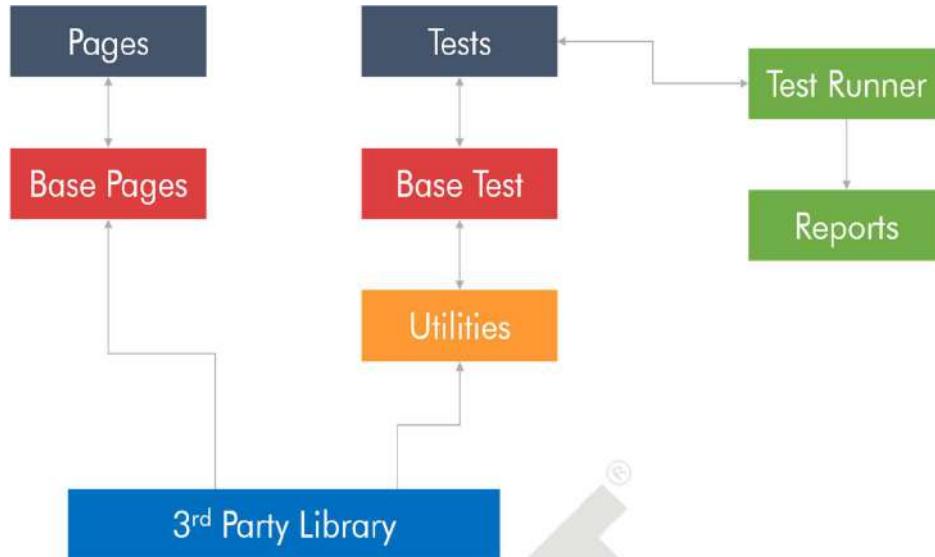
The creation and designing a simple test automation framework for running tests on web applications, using Selenium webdriver, TestNG and Maven, using Java programming language is shown here. This framework should help to run the tests from an Eclipse IDE or an IntelliJ IDE.

The various tools used are:

- Apache Maven: It defines the project structure
- Selenium Webdriver: It provides Test automation tool/library
- Selenium Grid: It provides a feature to run tests on multiple test environments on a network
- TestNG: It provides test runner and report engine

Test suite Structure

The test suite structure is:

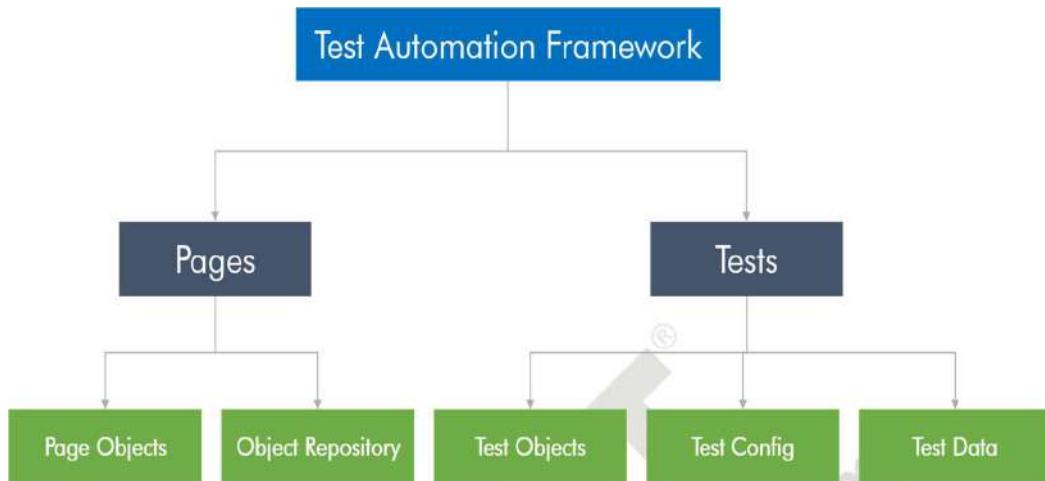


- **BasePage:** BasePage is a class which contains the reusable methods that will be used by the various pages. For e.g., the BasePage class contains “getWebElement” method which will return a web-element when it pass the locator and the driver instances.
- **Pages:** Pages are a set of classes, named based on the name of the webpage. For e.g., Login-Page, Home-Page, are so named because of the webpage they represent. The Login-Page will contain the public methods to perform login, and private attributes for username and password fields. The private attributes for username and password fields will help maintain encapsulation of the unwanted information for the tests.
- **BaseTest:** BaseTest is a class which contains the reusable methods that will be used by the various tests. The BaseTest is expected to contain the “@BeforeTest”, “@AfterTest”, “@BeforeClass”, “@AfterClass” annotated methods to help manage the test sequence. The BaseTest will also contain the reference to the public web driver variable which will be instantiated and initialized in the @BeforeTest or @BeforeClass method when the test initiates and continue as a static variable till the @AfterTest or @AfterClass methods is called to end the test.
- **Tests:** Tests are a set of classes, named based on the test to be conducted for each webpage. For e.g., LoginTest, is so named because of the test method that would be used to test the Login page/action. The LoginTest will contain the public methods annotated with “@Test” to test the login functionality. The LoginTest will instantiate the LoginPage class, using the PageFactory.initElements method, and then will invoke the LoginPage log in method with data parameters for username and password which will set the values in the username and password fields in the web application, and click on the submit button. Validations need to be inserted post the submit button click so that the login can be validated. Boolean values are returned from the Login-Page, and assertions are performed at the test level for validations.

- Utilities: Utilities contain the various reusable classes and methods for overall run of the test suite. An example of a utility class could be a TestListener implementation to capture screenshots in case of failures.

Project Structure of the Test Automation Framework

The project structure in Eclipse or IntelliJ could be as follows:



Steps to create the framework in Eclipse

1. Create a new Maven Project
2. Provide a suitable package name for the project :: com... is a good naming convention it can follow

The folder structure would look like this: src/main/java, src/main/resources, src/test/java and src/test/resources

3. All the code would code into two parts.

Test Objects and Data Providers would go into src/test/java :: Meaning all testing related code should go into src/test/java

Page Objects should go into /src/main/java

Named the package when creating the maven package structure

Within the src/main/java and src/test/java see the defined package name and a sample java file will be found. Remove these files.

4. The project structure should look something like below.

Under src/main/java it can have com....pages

Under src/test/java it can have com....tests and com....datastore

It can have sub packages within these main packages to identify with modules

5. Based on the choice of browsers you can use the TestNG structure to set up the base test.

6. TestNG setup: Give the browser name, version and OS platform in the TestNG file to distinguish between classes, tests or suite. Since here tests are used for differentiation of browsers, so use the following structure for testng.xml

```

1 <suite>
2   <test1>
3     <parameter1> </parameter1>
4     <parameter2> </parameter2>
5     <classes>
6       <class1></class1>
7       <class2></class2>
8     </classes>
9   </test1>
10  <test2>
11    <parameter1> </parameter1>
12    <classes>
13      <class3></class3>
14      <class4></class4>
15    </classes>
16  </test2>
17 </suite>
```

7. PageObjects

For the page objects, you can have differences in how you identify and interact with elements. This is a sample code which is given here

```

public class LoginPage {
By txtUserName = By.id("uid");
By txtPassword = By.id("password");
By pageTitle = By.xpath("html_title");
By btnLogin = By.id("btnLogin");
```

8. TestObjects

For the tests, it would be using @Test annotations to define the tests.

```

@Test
public void TestClass1() {
// Test Code here
}
```

For the base test, it would need a @BeforeTest method which defines the browser and driver to be used. This set up will be based on the parameters passed, or any initial configuration which have set up. From the code above, in the TestNG.xml, it has parameters defined for every test. Generally these have to be very similar. The parameter in the test2 is superflous (not recommended)

```

@BeforeTest
public void beforeTest(String parameter1) {
// Here you define how you can set the driver
WebDriver driver = null;
If (parameter == "chrome")
driver = new ChromeDriver();
```

```

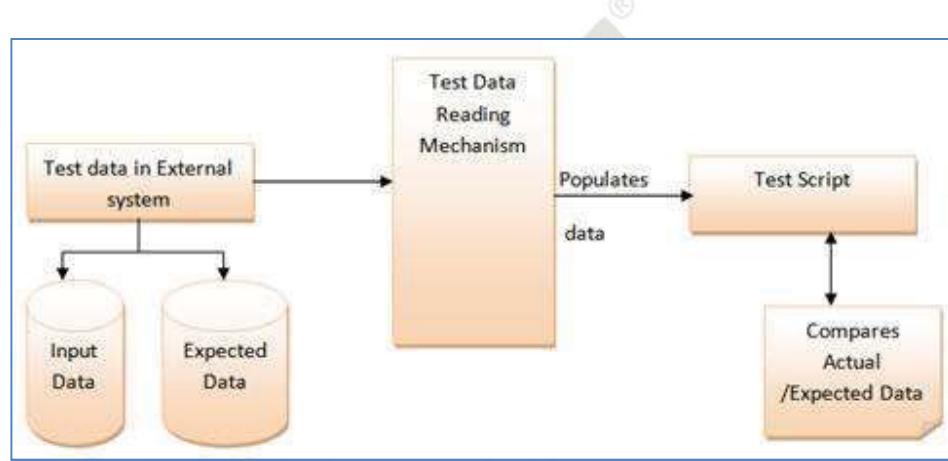
else if (parameter == "firefox")
driver = new FirefoxDriver();
}

```

What is Data Driven Framework?

While automating or testing any application, at times it may be required to test the same functionality multiple times with the different set of input data. Thus, in such cases, the test data can't be let embedded in the test script and need to be retained in data into some external database outside the test scripts. Data Driven Testing Framework helps a user to segregate the test script logic and the test data from each other. It lets the user store the test data into an external database. The external databases can be property files, xml files, excel files, text files, CSV files, ODBC repositories. The data is conventionally stored in "Key-Value" pairs. The key can be used to access and populate the data within the test scripts.

The Data driven testing framework can be represented as below:



Steps to create a Data driven Framework are:

Example: Consider to Test Login functionality of an application

1. Identify the Test Cases as below
 - Input correct username and password – Login Success
 - Input incorrect username and correct password – Login Failure
 - Input correct username and incorrect password - Login Failure
2. Create detailed test Steps for above 3 Test Cases
3. Create Test Script
4. Create an excel/csv with the Input Test Data
5. Modify the Script to Loop over Input Test Data. The input commands should also be parameterized

This framework purely depends on data and data source can be anything like Excel file, CSV File, database.

In data driven framework script will be separated from Data part, it means so if any changes happen we do not have to modify all the test cases.

Example

To create 50 Gmail accounts so two approaches are there to do this

First- Create 50 scripts and run them.

Second- Keep all the data separate in the file and changes the data only that is required for script and script will be only one. In future, any changes in the application then it has to modify one script only not the scripts.

In simple words when it has to execute the same script with multiple sets of data then adopt data driven framework

In this post, taking data from 2D Array and passing the data into script.

Open Facebook and type username and password and login

This test case should run 2 times with different set of data(data we have provided in the 2D array)

Let's implement the same

```
<span style="font-size: 14pt; font-family: arial, helvetica, sans-serif;">package DataDrivenTesting;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class TestDDT {

    // this will take data from dataprovider which we created
    @Test(dataProvider="testdata")
    public void TestFireFox(String uname, String password) {

        // Open browsre
        WebDriver driver=new FirefoxDriver();

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

```

// Load application
driver.get("http://www.facebook.com");

// clear email field

driver.findElement(By.id("email")).clear();

// Enter username
driver.findElement(By.id("email")).sendKeys(uname);

// Clear password field
driver.findElement(By.id("pass")).clear();

// Enter password
driver.findElement(By.id("pass")).sendKeys(password);
}

// this is DataProvider which actually feed data to our test cases
here I have taken 2 D //array with 2 rows and 2 column it means. It
will run our test case two times because we //have taken 2 rows.
While first iteration this will pass username and password to test
//case and in second iteration perform the same for second rows

@DataProvider(name="testdata")
public Object[][] TestDataFeed(){

    // Create object array with 2 rows and 2 column- first parameter is
    //row and second is //column
    Object [][] facebookdata=new Object[2][2];

    // Enter data to row 0 column 0
    facebookdata[0][0]="Selenium1@gmail.com";

    // Enter data to row 0 column 1
    facebookdata[0][1]="Password1";

    // Enter data to row 1 column 0
    facebookdata[1][0]="Selenium2@gmail.com";
}

```

```

// Enter data to row 1 column 0
facebookdata[1][1]="Password2";

// return arrayobject to testscript
return facebookdata;
}

} </span>
```

Advantages:

- The most important feature of this framework is that it considerably reduces the total number of scripts required to cover all the possible combinations of test scenarios. Thus lesser amount of code is required to test a complete set of scenarios.
- Any change in the test data matrix would not hamper the test script code.
- Increases flexibility and maintainability
- A single test scenario can be executed altering the test data values.

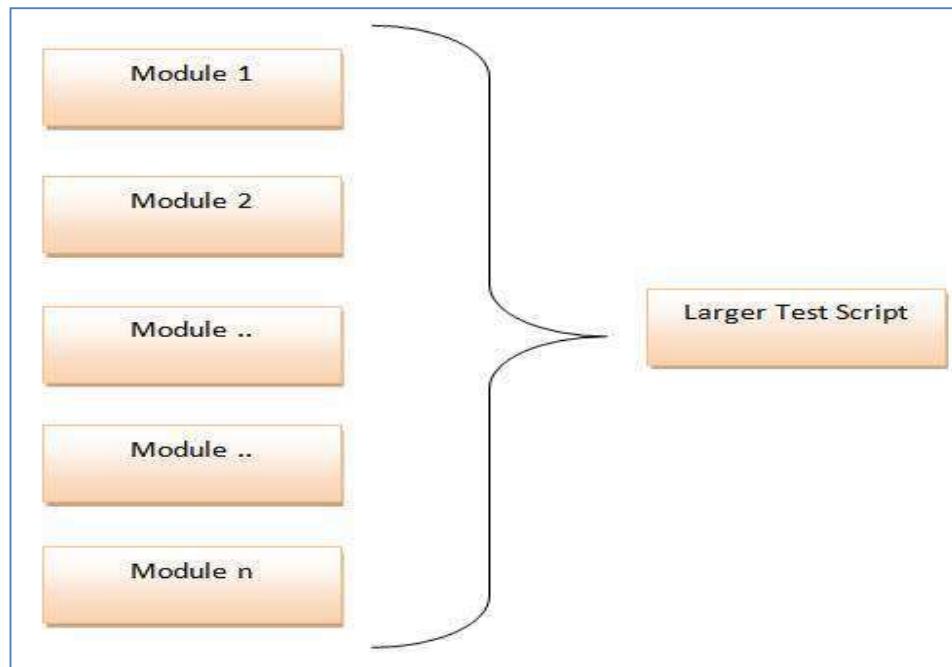
Disadvantages:

- The process is complex and requires an extra effort to come up with the test data sources and reading mechanisms.
- Requires proficiency in a programming language that is being used to develop test scripts.

What is Modular Driven Framework?

Module based Testing Framework is based on one of the popularly known OOPs concept, Abstraction. The framework divides the entire “Application under Test” into a number of logical and isolated modules. For each module, a separate and independent test script is created. The modules are separated by an abstraction layer in such a way that the changes made in the sections of the application doesn’t yield effects on this module. The representation of framework is as below:





Types of Modular driven Framework

- Test Script Modularity Framework: Enables creation of Small, Independent Scripts representing Modules& Functions of the Application under Test (AUT).
- Test Library Architecture Framework: Enables creation of Library Files representing Modules & Functions of the Application under Test (AUT)

How to do Modular driven Framework

Look for repeated functionality in the application for example the 'login' functionality. It can simple wrap this functionality in a method and can give it a sensible name.

1. Create a 'New Package' file and name it as 'appModule', by right click on the Project and select New > Package. We will be creating different packages for Page Objects, Utilities, Test Data, Test Cases and Modular actions. It is always recommended to use this structure, as it is easy to understand, easy to use and easy to maintain.
2. Create 'New Class' and name it as SignIn_Action by right click on package 'appModule' and select New > Class. It will add new class 'SignIn_Action' under package 'appModule'.
3. Now create a Public Static Void Method and name it as Execute and club the following steps in to it:

Click on the My Account link.

Enter Username

Enter Password

Click on the Submit button

This method will not have any Argument (driver) and Return value as it is a void method.

```

import org.openqa.selenium.WebDriver;
import framework.pageObject.Home_Page;
import framework.pageObject.LogIn_Page;
public class SignIn_Action{
    public static void Execute(WebDriver driver){
        Home_Page.lnk_SignIn(driver).click();
        LogIn_Page.txtbx_UserName(driver).sendKeys("testuser_1");
        LogIn_Page.txtbx_Password(driver).sendKeys("Test@123");
        LogIn_Page.btn_LogIn(driver).click();
    }
}

```

4. Create a New Class and name it as Module_TC by right click on the 'automationFramework' Package and select New > Class. It will be creating all the test cases under this package.

Now convert the old POM_TC in to the new Module based test case.

```

package automationFramework;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
// Import package appModule./*
import appModules.SignIn_Action;
import pageObjects.Home_Page;
public class Module_TC {
    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.store.demoqa.com");
        // Use your Module SignIn now
        SignIn_Action.Execute(driver);
        System.out.println("Login Successfully, now it is the time
to Log Off buddy.");
        Home_Page.lnk_LogOut(driver).click();
    }
}

```

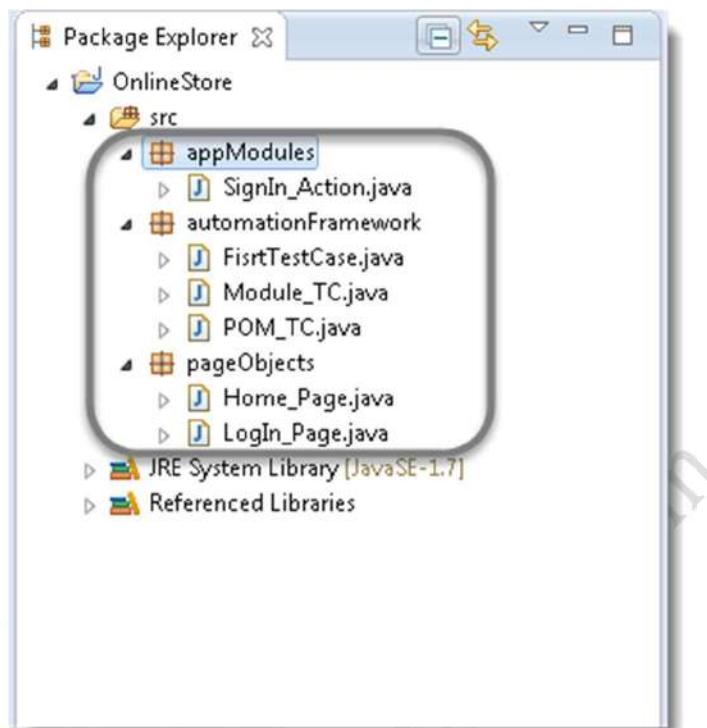
```

        driver.quit();
    }
}

```

Call to SignIn_Action will automatically execute all the steps mentioned under it.

Project explorer window will look like this now.



Advantages:

- The framework introduces the high level of modularization which leads to easier and cost-efficient maintenance.
- The framework is pretty much scalable
- If the changes are implemented in one part of the application, only the test script representing that part of the application needs to be fixed to leave all the other parts untouched.

Disadvantages:

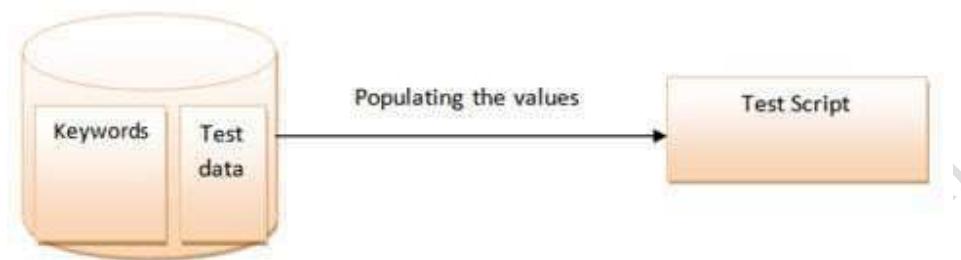
- While implementing test scripts for each module separately, the test data is embed (Data which supposed to perform testing) into the test scripts. Thus, whenever a test is being supposed with a different set of test data, it requires the manipulations to be made in the test scripts.

What is Keyword Driven Framework?

The Keyword driven testing framework is an extension to Data driven Testing Framework in a sense that it not only segregates the test data from the scripts, it also keeps the certain set of code belonging to the test script into an external data file.

These set of code are known as Keywords and hence the framework is so named. Keywords are self-guiding as to what actions need to be performed on the application.

The keywords and the test data are stored in a tabular like structure and thus it is also popularly regarded as Table driven Framework. Take a notice that keywords and test data are entities independent of the automation tool being used.



Example Test case of Keyword Driven Test Framework

STEP NO	DESCRIPTION	KEYWORD	LOCATOR/DATA
1	Login to application	login	
2	Click on homepage	clickLink	//*[@id='homepage']
3	Verify logged in user	verifyLink	//*[@id='link']

In the above example, keywords like login, clicking and verify Link are defined within the code.

Depending upon the nature of application keywords can be derived. And all the keywords can be reused multiple times in a single test case. Locator column contains the locator value that is used to identify the web elements on the screen or the test data that needs to be supplied.

All the required keywords are designed and placed in the base code of the framework.

Advantages

- In addition to advantages provided by Data Driven testing, the Keyword driven framework doesn't require the user to possess scripting knowledge, unlike Data Driven Testing.
- A single keyword can be used across multiple test scripts.

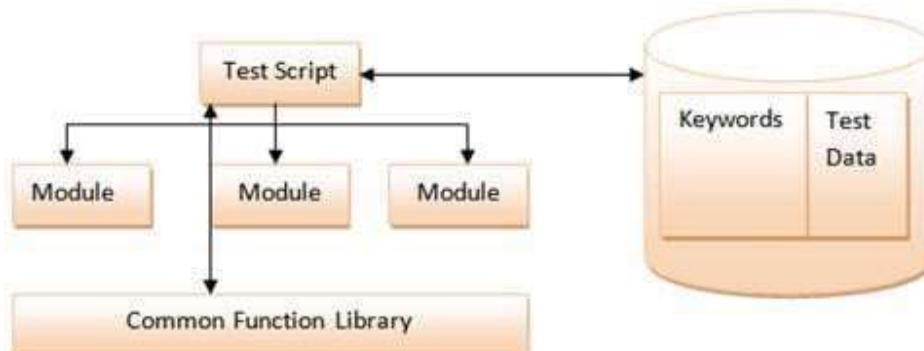
Disadvantages

- The user should be well versed with the Keyword creation mechanism to be able to efficiently leverage the benefits provided by the framework.

- The framework becomes complicated gradually as it grows and a number of new keywords are introduced.

What is Hybrid Driven Framework?

As the name suggests, the Hybrid Testing Framework is a combination of more than one above mentioned frameworks. The best thing about such a setup is that it leverages the benefits of all kinds of associated frameworks.



Test sheet would contain both the keywords and the Data.

Step	Description	Keyword	Locator	Data
Step1	Navigate to login page	navigate		
Step2	Enter User Name	input	//*[@id='username']	userA
Step3	Enter Password	input	//*[@id='password']	password
Step4	Verify Home page	verifyUser	//*[@id='User']	
Step5	Verify User link	verifyLink	link='UserLink'	userA
Step6	Logout from the application	clickLink	//*[@id='logout']	

In the above example, keyword column contains all the required keywords used in the particular test case and data column drives all the data required in the test scenario. If any step does not need any input then it can be left empty.

Test Data Files

Why Data Driven Testing?

There are number of data sets that have to be used to test a feature of an application. Running the same test with different data manually is time-consuming, error prone and a boring task.

Let us understand this scenario with an example.

Suppose there is a need to test the login/Register/ any form with multiple input fields with 100 different data sets. To test this there are three different approaches:

- Create 100 scripts one for each dataset and execute each test one by one.
- Change the data in the script and execute it multiple times.
- Import the data from the excel sheet and execute the script multiple times with different data.

First two scenarios are laborious, time-consuming implying low ROI. Hence, we must follow the third approach. In the third approach, are implementing the Data Driven framework, where all our data resides in an excel sheet, where it is imported from and used to test the features of the application.

How to create a Data Driven Automation Framework

The steps to test Login functionality of an application are below

1. Identify the Test Cases as below
 - Input Correct username and password – Login Success
 - Input incorrect username and correct password – Login Failure
 - Input correct username and incorrect password - Login Failure
 -
2. Create detailed test Steps for above 3 Test Cases

Description	Test Steps	Test Data	Expected Results
Check Login for valid credentials	1. Launch the application 2. Enter Username password 3. Click Okay 4. Check Results	Username: valid password: valid	Login Success
Check Login for invalid credentials	1. Launch the application 2. Enter Username password	Username: invalid password: valid	Login Fail

	3. Click Okay 4. Check Results		
Check Login for invalid credentials	1. Launch the application 2. Enter Username password 3. Click Okay 4. Check Results	Username: valid password: invalid	Login Fail

3. Create Test Script

If you observe the Test Steps remain common through the 3 Test Steps. You need to create a Test Script to execute these steps

```
// This is Pseudo Code
// Test Step 1: Launch Application
driver.get("URL of the Application");
// Test Step 2: Enter Password
textbox_username.sendKeys("valid");
// Test Step 3: Enter Password
textbox_password.sendKeys("invalid");
// Test Step 4: Check Results
If (Next Screen) print success else Fail
```

4. Create an excel/csv with the Input Test Data

5. Step Modify the Script to Loop over Input Test Data. The input commands should also be parameterized

```
/ This is Pseudo Code
// Loop 3 Times
for (i = 0; i <= 3; i++) {
    // Read data from Excel and store into variables
    int input_1 = ReadExcel(i, 0);
    int input_2 = ReadExcel(i, 1);

    // Test Step 1: Launch Application
    driver.get("URL of the Application");

    // Test Step 2: Enter Password
    textbox_username.sendKeys(input_1);
    // Test Step 3: Enter Password

    textbox_password.sendKeys(input_2);
    // Test Step 4: Check Results
    If(Next Screen) print success
    else Fail
}
```

Above are just 3 test cases. The test script can be used to loop over following test cases just by appending test data values to Excel

Input incorrect username and incorrect password – Login Fail

Input correct username and password blank – Login Fail

Input blank username and blank password- Login Fail

Best practices of Data Driven testing:

Below given are Best testing practices for Data-Driven testing:

- It is ideal to use realistic information during the data-driven testing process
- Test flow navigation should be coded inside the test script
- Drive virtual APIs with meaningful data
- Use Data to Drive Dynamic Assertions
- Test positive as well as negative outcomes
- Repurpose Data Driven Functional Tests for Security and Performance

Advantages of Data-Driven testing

- Allows to test application with multiple sets of data values during Regression testing
- Test data and verification data can be organized in just one file, and it is separate from the test case logic.
- Base on the tool, it is possible to have the test scripts in a single repository. This makes the texts easy to understand, maintain and manage.
- Actions and Functions can be reused in different tests.
- Some tools generate test data automatically. This is useful when large volumes of random test data are necessary, which helps to save the time.
- Data-driven testing can perform any phase of the development. A data-driven test cases are generally merged in the single process. However, it can be used in multiple test cases.
- Allows developers and testers to have clear separation for the logic of their test cases/scripts from the test data.
- The same test cases can be executed several times which helps to reduce test case and scripts.
- Any changes in the test script do not affect the test data

Disadvantages of Data Driven testing

- Quality of the test is depended on the automation skills of the Implementing team
- Data validation is a time-consuming task when testing large amount of data.
- Maintenance is a big issue as large amount of coding needed for Data-Driven testing.
- High-level technical skills are required. A tester may have to learn an entirely new scripting language.
- There will be more documentation. Mostly related to scripts management tests infrastructure and testing results.

- A text editor like Notepad is required to create and maintain data files.

TestCore Class

What is Apache POI

Apache POI is an open source library developed and distributed by Apache Software Foundation to design or modify Microsoft Office files using Java program. It is a popular API that allows working around excel files using Java Programs. In short, you can read and write MS Excel files using Java. Apache POI is your Java Excel solution. You use HSSF if you needed to read or write an Excel file using Java (XLS) and you use XSSF if you need to read or write an OOXML Excel file using Java (XLSX). It has many predefined methods, classes, and interfaces.

Loads xls file

The excel files are considered to be the data source/data providers for test script execution. These files store the test data into key-value pairs. Each test script can have its own test data file. The name of the test script and the corresponding test data files/ excel sheet has been kept same for the traceability perspective.

Each of the columns represents a key and each of the rows represents a test data/value. Note that the test data formats are solely user defined. Thus based on the requirements, the test data files can be customized.

Follow the below steps to read excel files using Apache POI in Selenium WebDriver:

1. Download Apache POI jar file
<https://poi.apache.org/download.html>
2. Add download jar files
3. Select Project and Right click on the Project – Go to ‘Build path’ – Go to ‘Configure build path’ – Click on ‘lib’ section – Add external jar

Precondition

4. Create an xlsx file and save it at particular location. Enter some data to read using Selenium. Close the created excel file before executing the script. (Here excel file is used as ‘Test.xlsx’ in the D Driver and the sheet name is used as TestData.)
5. Go to option “Format Cells” and under Number Tab option, select Text and click on OK. By default it will be general, you need to make it as Number. If it doesn’t make it as text, there is a chance of NullPointerException error.

Below mentioned script shows to read excel files in Selenium using Apache POI:

```

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;

```

```

import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

//to read excel files using Apache POI
public class ReadExcel {
    public static void main (String [] args) throws IOException{
        FileInputStream fis = new FileInputStream("D:\\Test.xlsx");
        XSSSSFWorkbook workbook = new XSSSSFWorkbook(fis);
        XSSFSheet sheet = workbook.getSheetAt(0);
        //Cell A1 = row 0 and column 0. It reads first row as 0 and Column A
        //as 0.
        Row row = sheet.getRow(0);
        Cell cell = row.getCell(0);
        System.out.println(cell);
        System.out.println(sheet.getRow(0).getCell(0));
    }
}

```

Output

SoftwareTestingMaterial.com

Run Selenium Server Through code

When test automating a web application using Selenium, it have to start the Selenium server first, so that a new Selenium session is created to talk to the web browser. This can be either done manually, i.e. user running a command line to start the Selenium server, or to get the pure automation effect of Selenium, it is best to start the Selenium server via a program code. The code below is written in Java and starts the Selenium server when called. Normally this would be the first action within your main() function.

```

import org.openqa.selenium.server.RemoteControlConfiguration;
import org.openqa.selenium.server.SeleniumServer;
import com.thoughtworks.selenium.Selenium;
import java.net.BindException;
import java.net.ServerSocket;

```

```

public class Server {

    public static SeleniumServer server;

    public static void startSeleniumServer() throws Exception {

        try {
            ServerSocket serverSocket = new
            ServerSocket(RemoteControlConfiguration.DEFAULT_PORT);
            serverSocket.close();
        //Server not up, start it
            try {
                RemoteControlConfiguration rcc = new
                RemoteControlConfiguration();

                rcc.setPort(RemoteControlConfiguration.DEFAULT_PORT)
                ;
                server = new SeleniumServer(false, rcc);

            } catch (Exception e) {
                System.err.println("Could not create Selenium Server
because of: " +
e.getMessage());
                e.printStackTrace();
            }
            try {
                server.start();
                System.out.println("Server started");
            } catch (Exception e) {
                System.err.println("Could not start Selenium Server
because of: "
                + e.getMessage());
                e.printStackTrace();
            }
        } catch (BindException e) {
            System.out.println("Selenium server already up, will
reuse...");
        }
    }
}

```

```

public static void stopSeleniumServer(Selenium selenium) {
    selenium.stop();
    if (server != null)
    {
        try
        {
            selenium.shutDownSeleniumServer();
            server.stop();

            server = null;
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
}

```

Another very simple and easy way to start selenium server inside the program, and if it is not bothered about setting specific port, just call these few lines in your program (before starting selenium session):

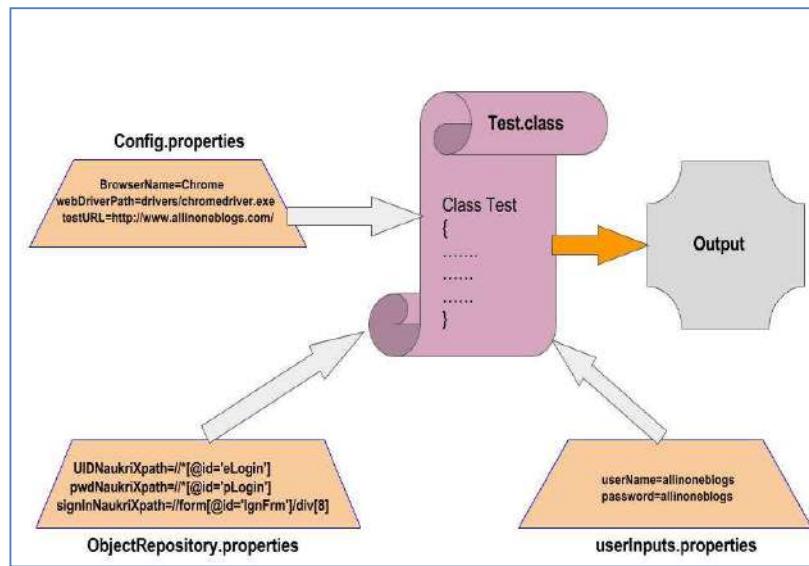
```

SeleniumServer _server = new SeleniumServer();
_server.boot();
_server.start();
_server.stop(); and once it has finished with the tests.

```

Object.properties File to Store Xpaths

According to name, a Properties file is used to define properties of keys which would have been used in our Selenium project. A Property file contains data in Key=Value format. It helps external users to communicate with the code and generate output on the basis of inputs. The properties file in Selenium Webdriver requires a custom Property class to provide a method to read locators. Or it can also achieve it by writing a Static method to access properties file and get locators.



Features of a Property File

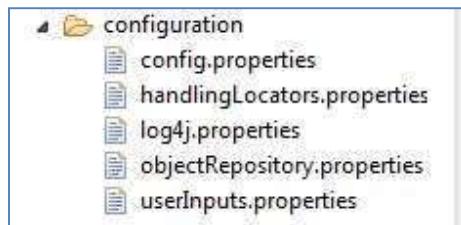
- Can be used to access user input
- Can be used to store XPath values
- Changes can be made easily without accessing class files
- A user can create as many as properties file as per requirement of Project.
- Easy to maintain test data or user input
- Used in the framework to keep the similar types of variable in separate files.
- It stores data in the Key=Value format without any special characters like "" or ;
- By default property file returns key values in String format.
- Can store values of any data type in the properties file but we have to parse the values in required data type.
- We can also set multiple values for a single key using any separator
- Comments can be included in the file using double hash(#) symbols.

Creating Properties File

The Steps to create properties file are:

1. Create Project
2. Now right-click on a folder, Select New -> File
3. Give File name as "testdata.properties"
4. Click Finish
5. Testdata.properties file would be added to your selected folder.
6. Now open the properties file to add key=values in following format:

```
testURL=http://www.allinoneblogs.com/
searchXpath=//*[@id="search-3"]/form/label/input
```



Accessing Properties file in Class files

1. Locating properties file in project:

```
File file=new File(filePath);
```
2. Convert file in byte form using FileInputStream

```
FileInputStream fis=new FileInputStream(file);
```
3. Create object of Properties Class

```
Properties prop=new Properties();
```
4. Load data from input stream to properties object using load()

```
prop.load(fis);
```
5. Accessing values from Property file to a variable in Class file

```
String testData=prop.getProperty("testURL");
driver.get(testData);
```

Example:

```
Properties File: (objectRepository.properties)
testURL=http://www.allinoneblogs.com/
searchXpath=//*[@id="search-3"]]/form/label/input
```

Sample.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class Sample {
    public static void main(String[] args) throws IOException
    {
        // Step #1: Locate Properties File
        File file=new
        File("configuration//objectRepository.properties");

        // Step #2: Convert data into bytes
```

```

FileInputStream fis=new FileInputStream(file);

// Step #3: Creating object of Properties Class
Properties prop=new Properties();

// Step #4: Load data from inputstream to Properties
//object
prop.load(fis);

// Step #5: Store values from Properties file to local
//variable.
String URL=prop.getProperty("testURL");
String Xpath=prop.getProperty("searchXpath");

System.out.println("URL: "+URL);
System.out.println("Xpath: "+Xpath);
}

}

```

Configuration Files

Create a configuration properties file to store parameters such as log file path, test data path, results path, selenium execution speed, wait period, etc, then make changes to the parameters whenever needed. Java provides a way to create configuration file for storing configurable parameters of an application. Selenium Test Suite also needs such a file to keep global configuration settings.

Example to create and to access values from the properties file.

- Create a file text file (using any text editor such as notepad) Selconfig.properties (as in the file – view file) and save the file as “Selconfig.properties”
- Create classes as in the following code snippets:

```

package com.selftechy.readdata;
import com.selftechy.library.*;
import java.io.IOException;
public class TestRun {
    public static void main(String[] args) throws IOException {
        // TODO Auto-generated method stub
    }
}

```

```

config cfg = new config("F:\\Helios-
Workspace\\Config\\SelConfig.properties");
System.out.println(cfg.ReadProperty("ConfigPath"));

}

import java.io.*;
import java.util.*;
public class config {
    String str, key;
    private String filepath;

    public config(String filepath){
        this.filepath=filepath;
    }

    public String ReadProperty(String propkey) {
        String propval="";
        try{
            int check = 0;
            while(check == 0){
                check = 1;
                File cfgfile = new File(filepath);
                if(cfgfile.exists()){
                    Properties props = new Properties();
                    FileInputStream propin = new
                    FileInputStream(cfgfile);
                    props.load(propin);
                    propval=props.getProperty(propkey);
                }
                else{
                    check = 0;
                }
            }
        }
    }
}

```

```

        catch (IOException e) {
            e.printStackTrace();
        }

        return propval;
    }
}

```

Save both the classes and then execute the first one (TestRun.java) as Java application. In the Selconfig.properties file, TestData & ConfigPath are the keys and "F:\\Helios-Workspace\\TestData & F:\\Helios-Workspace\\Config" are the respective values. In .properties file comments can put using "#". Any sentence that starts with # will be considered as comment.

To access the values of any properties, use the "key" parameter. For example, to access the value of "TestData" and "ConfigPath" write the code as follows:

```

config cfg = new config("F:\\Helios-
Workspace\\Config\\SelConfig.properties");
cfg.ReadProperty("ConfigPath")
cfg.ReadProperty("TestData")

```

Emailing Test Results

There might be situations, where it might be tasked by the management people or the clients to send email after every test execution.

The program which is need include in Selenium Framework is,

//The jar files which has used here are activation.jar and mail.jar Can be downloaded from Internet.

```

import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.activation.FileDataSource;
import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;

public class SendMail
{
//reportFileName = TestExecutionResultFileName
public static void execute(String reportFileName) throws Exception
{
String path=<Report file path>;

```

```

String[] to={"<stakeholder1>","<stakeholder2>"};
String[] cc={ };
String[] bcc={"<AutomationTester>"};

SendMail.sendMail("<AutomationTesterUserName>",
"<AutomationTesterPassword>",
"smtp.gmail.com",
"465",
"true",
"true",
true,
"javax.net.ssl.SSLSocketFactory",
"false",
to,
cc,
bcc,
"<Subject line>",
"<Contents if any>",
path,
reportFileName);
}

public static boolean sendMail(String userName,
String passWord,
String host,
String port,
String starttls,
String auth,
boolean debug,
String socketFactoryClass,
String fallback,
String[] to,
String[] cc,
String[] bcc,
String subject,
String text,
String attachmentPath,
String attachmentName) {

//Object Instantiation of a properties file.
Properties props = new Properties();

props.put("mail.smtp.user", userName);
props.put("mail.smtp.host", host);
if(!"".equals(port)){
props.put("mail.smtp.port", port);
}

if(!"".equals(starttls)){
props.put("mail.smtp.starttls.enable",starttls);
props.put("mail.smtp.auth", auth);
}

if(debug) {
}
}

```

```

props.put("mail.smtp.debug", "true");
else{
props.put("mail.smtp.debug", "false");
}

if(!"".equals(port)){
props.put("mail.smtp.socketFactory.port", port);
}
if(!"".equals(socketFactoryClass)){
props.put("mail.smtp.socketFactory.class",socketFactoryClass);
}
if(!"".equals(fallback)){
props.put("mail.smtp.socketFactory.fallback", fallback);
}

try{
Session session = Session.getDefaultInstance(props, null);
session.setDebug(debug);

MimeMessage msg = new MimeMessage(session);
msg.setText(text);
msg.setSubject(subject);

Multipart multipart = new MimeMultipart();
MimeBodyPart messageBodyPart = new MimeBodyPart();
DataSource source = new FileDataSource(attachmentPath);
messageBodyPart.setDataHandler(new DataHandler(source));
messageBodyPart.setFileName(attachmentName);
multipart.addBodyPart(messageBodyPart);

msg.setContent(multipart);
msg.setFrom(new InternetAddress(userName));
for(int i=0;i<to.length;i++){
msg.addRecipient(Message.RecipientType.TO, new
InternetAddress(to[i]));
}
for(int i=0;i<cc.length;i++){
msg.addRecipient(Message.RecipientType.CC, new
InternetAddress(cc[i]));
}
for(int i=0;i<bcc.length;i++){
msg.addRecipient(Message.RecipientType.BCC, new
InternetAddress(bcc[i]));
}
msg.saveChanges();

Transport transport = session.getTransport("smtp");
transport.connect(host, userName, passWord);
transport.sendMessage(msg, msg.getAllRecipients());
transport.close();
}

```

```

        return true;
    } catch (Exception mex) {
        mex.printStackTrace();
        return false;
    }
}
}
}

```

The above Source code will do the job of triggering email after every execution.

Add the below snippet at the end of the test execution report creation.

```
[sourcecode language="java"]
SendMail.execute(ExecutionFileName); [/sourcecode]
```

With this the whole setup of automatic email triggering functionality is integrated with the framework.

Generating Reports

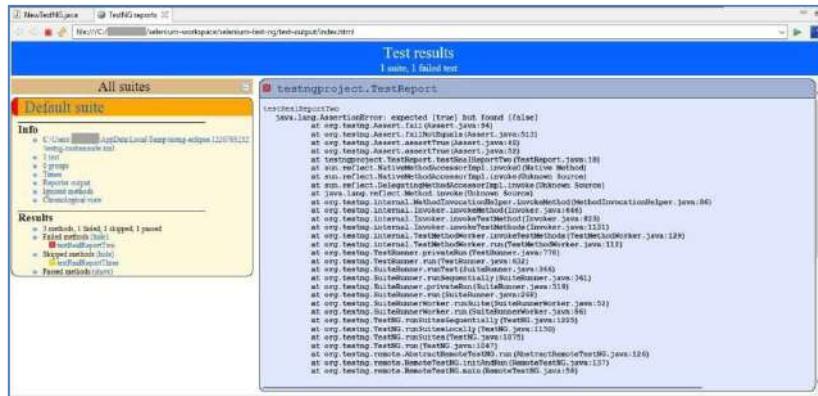
A perfect test automation tool is what it need for the successful execution of the testing requirements in the Agile process. And there are multiple factors which play a critical role in the formation of a robust automation framework. One such element is reporting which not only make it aware of the status of the success or failure but helps in finding out the potential bugs. The essential qualities of a good test report are:

- **Brevity:** A report should be short and concise. It should reveal the total no. of successes as well as failures.
- **Trackability:** Captures all the footprints that could lead to the root-cause of a failure.
- **Traceability:** It must provide the ability to review the following.
 - Historical data for test cases and failures
 - Age of a particular defect
- **Sharable:** It should support a format which you can share through email or integrate with CI tools like Jenkins/Bamboo.
- **Test coverage:** It should highlight the test coverage for the following.
 - Test coverage of the module under test.
 - Test coverage of the application under test.

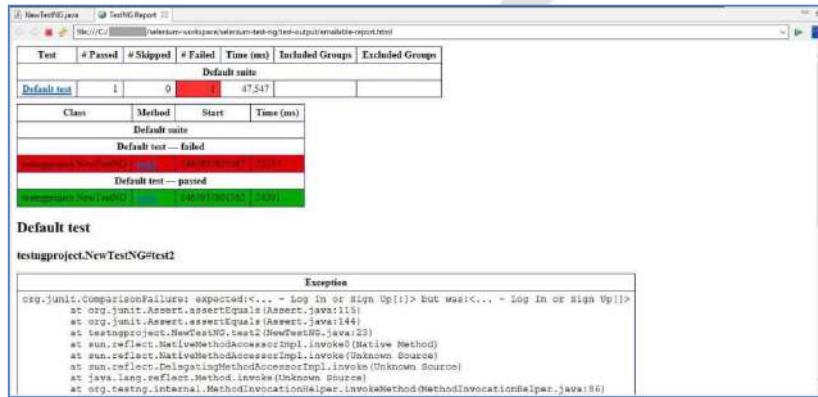
TestNG Framework Test report

TestNG framework library provides a test report in HTML format every time the execution of the test suite is completed. It creates a test-output folder that is present at the root of the test project. This folder contains two types of test reports which are as follows:

HTML: This html report file contains the test report of a recently executed test suite. It contains details like an error, reporter logs, groups, time, TestNG xml files, etc. as shown below.



Emailable-report.html: This html report file contains the summary of current test execution that has test cases message in color codes. The test cases that are passed are in green color and the test cases which are failed are in red color as shown below.



Customization of TestNG Report

TestNG default reports are very detailed and handy. But still sometimes, it need to customize the report as per the team requirements such as remove or add one or more fields from the report, change the report layout, display the report in different format such as PDF, html, excel for all of such requirement TestNG framework has two interfaces that help in report customization as follows:

ITestListener Interface

This interface is used to customize real-time test report i.e. when it execute a handful of test cases using TestNG framework and we desire a report for an individual test case. In this scenario, for each test case, it have to implement ITestListener interface which will override onTestFailure, onTestSuccess, onTestStart, onTestSkipped, onFinish methods to convey the correct status of the currently executing test case into the report.

IReporter Interface

This interface is implemented to customize the final test report that is generated by TestNG framework. It has only one method to override i.e. generateReport. This method possesses all the required information to complete the test execution in the List<ISuite> with which report can be generated.

Generating Application and Selenium Logs

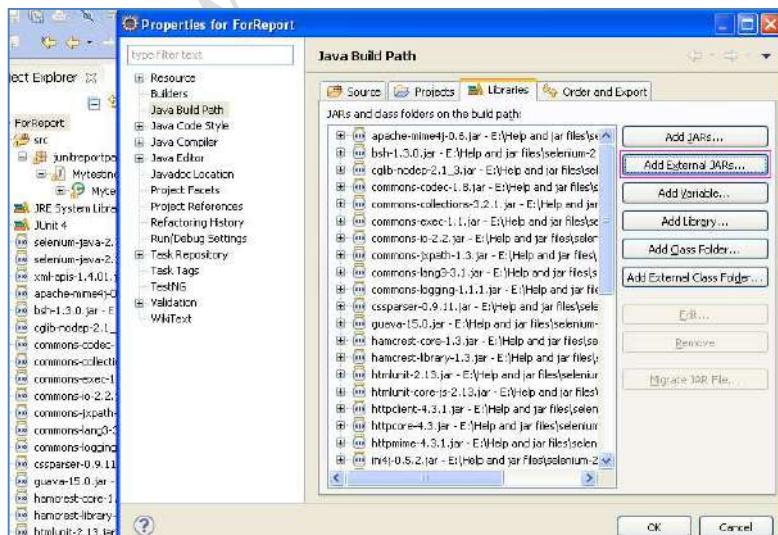
Below are the steps to generate log file in selenium webdriver test for software web application.

1. Download log4j jar file from logging.apache.org
2. Click on Zip folder link as shown in bellow image. Current version of log4j jar file is log4j-1.2.17. Version of log4j jar file may be different in future due to the frequent version change.



3. On click the zip folder link, it will show you zip mirror links to download folder. Download and save zip folder by clicking on mirror link.
4. Now Extract that zip folder and look inside that extracted folder. log4j-1.2.17.jar file will be there.
5. Import log4j-1.2.17.jar file in the eclipse project

Right click on your project folder and go to Build Path -> Configure Build path. It will open java build path window as shown in bellow image.



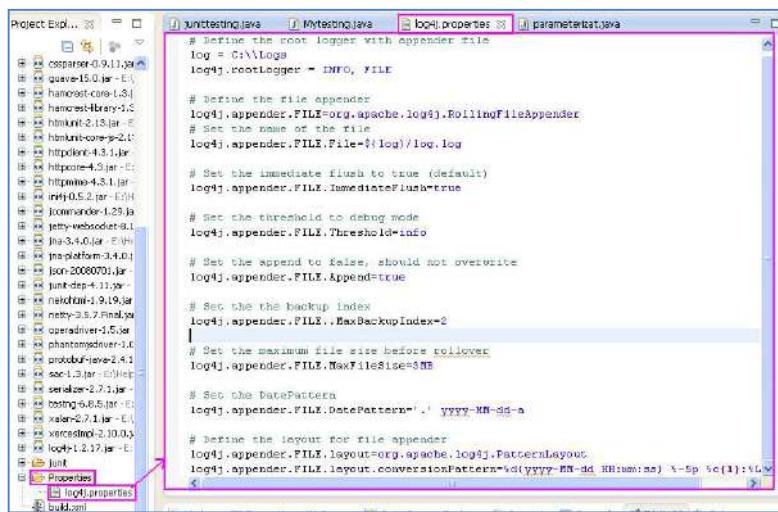
6. Click on Add External JARs button and select log4j-1.2.17.jar file. It will add log4j-1.2.17.jar file in the build path.

7. Click on OK button to close java build path window.

8. Create Properties folder under the project folder and add log4j.properties file in it.

Create folder with name = Properties under your project by right clicking on the project folder.

Now the properties folder under project folder will looks like bellow.



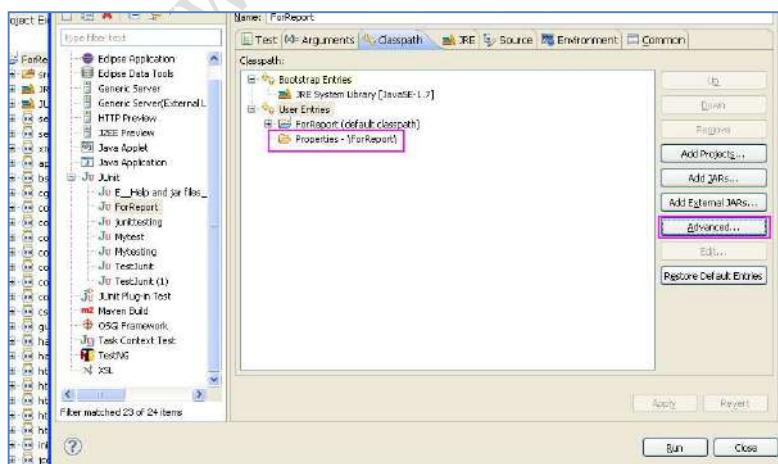
9. Set Run Configuration

This is optional step. It need to set and Run configure if log file is not generated and display bellow given warning message in the console when the test case will run.

log4j:WARN No appenders could be found for logger (dao.hsqlmanager).

Go to Run -> Run Configurations -> Classpath

Select User Entries -> Click on 'Advanced' button -> Select Add Folder -> And then select Properties folder from the project and click on OK. Now the Class tab will looks like bellow.



10. Create example (Sample) test case under the project.

11. Import log4j header file in the project

```
import org.apache.log4j.*;
```

12. Run the test case.

```
public void test () throws InterruptedException
{
    Logger log;
    driver.findElement(By.id("text1")).sendKeys("My First Name");
    log = Logger.getLogger(Mytesting.class);
    log.info("Type In Text field.");
    Select mydrpdwn = new
        Select(driver.findElement(By.id("Carlist")));
    mydrpdwn.selectByVisibleText("Audi");
    log = Logger.getLogger(Mytesting.class);
    log.info("Select value from drop down.");
    WebDriverWait wait = new WebDriverWait(driver, 15);

    wait.until(ExpectedConditions.elementToBeClickable(By.id("te
xt2")));
}
```

When this test case for software web application is executed, Go to folder = C:\Logs. log.log file will be created over there. Open that file and look in to it. Log written in the software test case will be inserted in the log file. In above example, Syntax marked with Pink color are written for inserting log. Now log can be write at any place in the test case of software web application as shown in above example.

Summary

- An Automation Framework is a set of guidelines like coding standards, test-data handling which provide an execution environment for the automation test scripts
- An automation framework in place ensures consistent coding and that best practices are followed
- Automation framework ensures that the complex task is broken down to smaller manageable tasks which can then be re-used whenever required.
- In the modular testing framework, testers create test scripts on module wise by breaking down the complete application under test into smaller, independent tests
- Data driven test automation framework is focused on separating the test scripts logic and the test data from each other
- Data-driven is a test automation framework which stores test data in a table or spread spreadsheet format
- In Data-driven test automation framework, input data can be stored in single or multiple data sources like xls, XML, csv, and databases
- When test automating a web application using Selenium, it have to start the Selenium server first, so that a new Selenium session is created to talk to the web browser
- Properties file is used to define properties of keys which would have been used in our Selenium project
- TestNG framework library provides a test report in HTML format every time the execution of the test suite is completed
- ITestListener Interface and IReporter Interface are used to customize TestNG Report
- There are series of steps that needs to be followed to generate application and selenium logs

Chapter 15 – JUnit Java Framework

About JUnit and TestNG

JUnit

JUnit is a unit testing framework for Java programming language. A Unit Test Case is a part of code, which ensures that another part of code (method) works as expected. A formal written unit test case is characterized by a known input and an expected output, which is worked out before the test is executed. The known input should test a precondition and the expected output should test a post-condition.

JUnit promotes the idea of "first testing then coding", which emphasizes on setting up the test data for a piece of code that can be tested first and then implemented. This approach is like "test a little, code a little, test a little, code a little." It increases the productivity of the programmer and the stability of program code, which in turn reduces the stress on the programmer and the time spent on debugging.

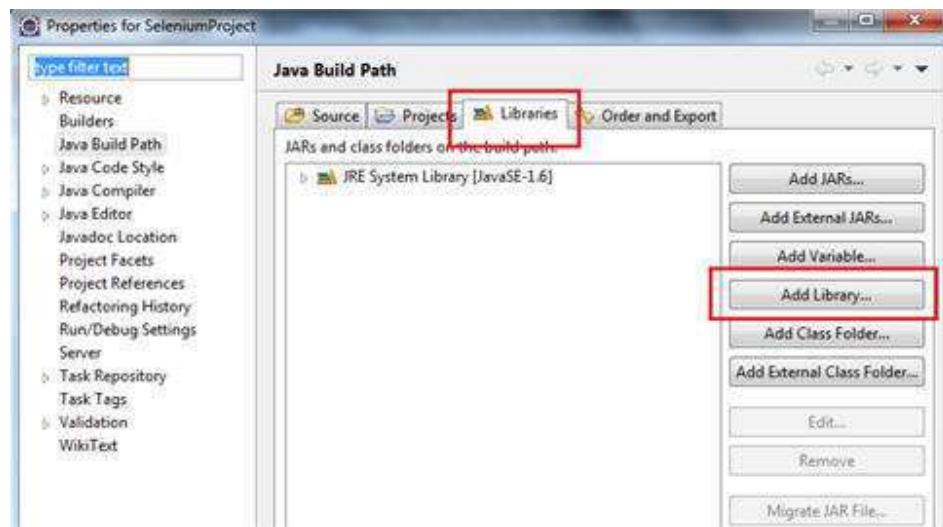
Features of JUnit

The Features of JUnit are:

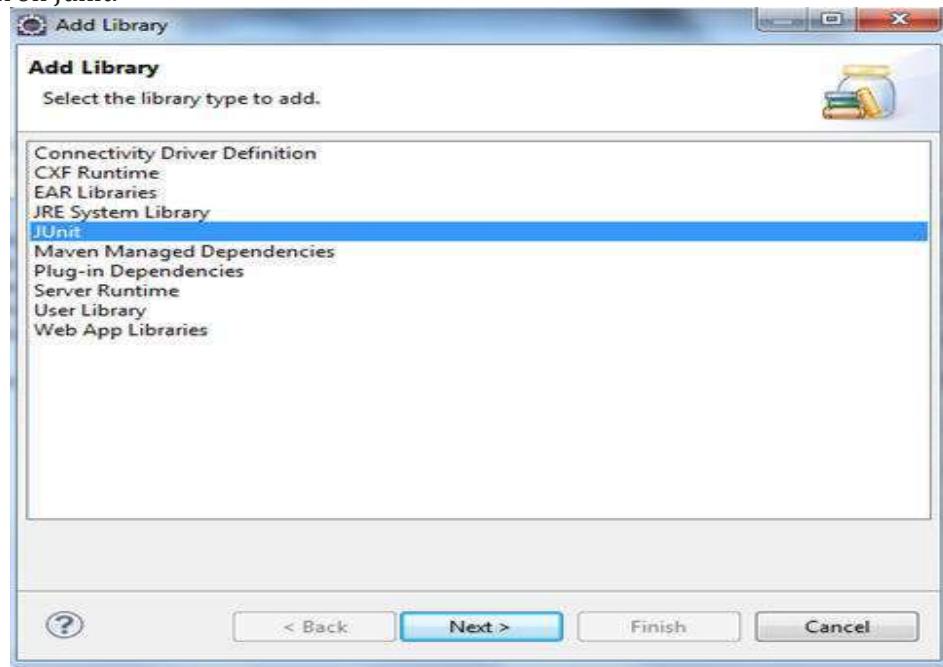
- JUnit is an open source framework, which is used for writing and running tests.
- It provides annotations to identify test methods.
- It provides assertions for testing expected results.
- It provides test runners for running tests.
- JUnit tests allow you to write codes faster, which increases quality.
- JUnit tests can be run automatically and they check their own results and provide immediate feedback. There's no need to manually comb through a report of test results.
- JUnit tests can be organized into test suites containing test cases and even other test suites.
- JUnit shows test progress in a bar that is green if the test is running smoothly, and it turns red when a test fails.

Adding JUnit library in Java project

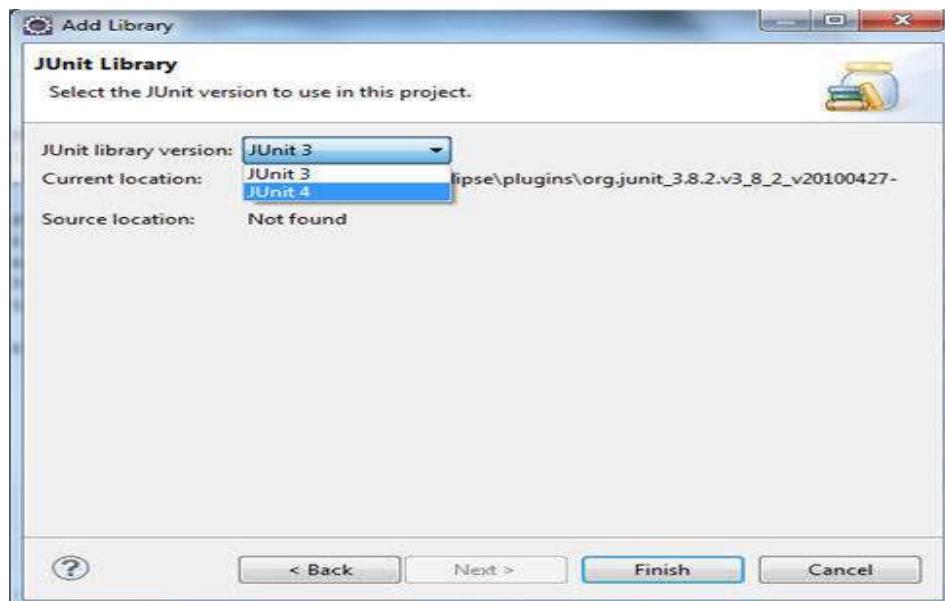
1. Right click on Java project->Build Path->Configure Build path
2. Click Libraries->Add Library



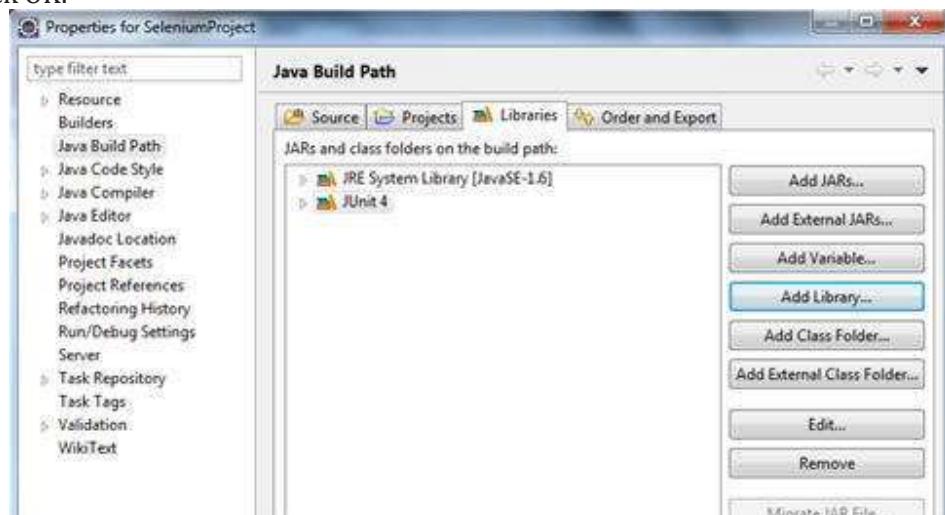
3. Click on Junit.



4. Select Junit4->Finish



5. Click OK.



JUnit Annotations Used in Selenium scripts

There are many annotations available in Junit. Here we have described few annotations which are used very frequently in Selenium scripts and framework.

- `@Test`

`@Test` annotation is used to run a Junit test.

Example:

```
@Test
public void junitTest()
```

```
{
System.out.println("Running Junit test");
Assert.assertEquals(1,1);
}
```

- **@Before:**

`@Before` annotation is used to run any specific test before each test.

```
public class Junittest {
@Before
public void beforeTest() {
System.out.println("Running before test");
}

@Test
public void junitTest() {
System.out.println("Running Junit test");
}
}
```

Output:

```
Running before test
Running Junit test
```

- **@BeforeClass**

This method executes once before running all test. The method has to be a static method. Initialization of properties files, databases etc are done in the `beforeClass` method.

```
public class Junittest {
@BeforeClass
public static void beforeClassTest() {
System.out.println("Executed before class method");
}

@Test
public void junitTest() {
System.out.println("Running Junit test");
}
```

}

```

@Test
public void secondJunitTest() {
    System.out.println("Running second Junit test");
}
}

```

Output:

```

Executed before class method
Running JUnit test
Running second JUnit test

```

- **@After**

This method executes after each test.

```

public class Junittest {
    @Test
    public void junitTest() {
        System.out.println("Running Junit test");
    }
}

```

@After

```

public void afterTest() {
    System.out.println("Running after method");
}
}

```

Output:

```

Running JUnit test
Running after method

```

- **@AfterClass**

Like **@BeforeClass**, **@AfterClass** executes once after executing all test methods. Like a **@BeforeClass** method, **@AfterClass** method has to be a static method.

```
public class Junttest {  
  
    @Test  
    public void junitTest(){  
        System.out.println("Running Junit test");  
    }  
  
    @Test  
    public void secondJunitTest(){  
        System.out.println("Running second Junit test");  
    }  
  
    @AfterClass  
    Public static void afterClassTest(){  
        System.out.println("Running afterclass method");  
    }  
}  
  
Output:  
Running JUnit test  
Running second JUnit test  
Running afterclass method
```

TestNG

TestNG is a testing framework inspired from JUnit and NUnit, but introducing some new functionalities that make it more powerful and easier to use. TestNG is an open source automated testing framework; where NG means NextGeneration. TestNG gives the developer the ability to write more flexible and powerful tests.

Features of TestNG

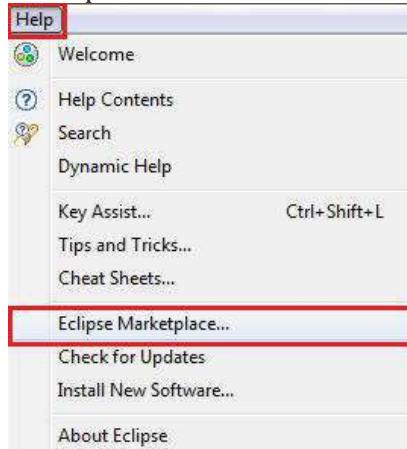
The features of TestNG are:

- Supports annotations.
- TestNG uses more Java and object oriented features.
- Supports testing integrated classes
- Separates compile-time test code from run-time configuration/data info.
- Flexible runtime configuration.
- Supports Dependent test methods, parallel testing, load testing, and partial failure.
- Flexible plug-in API.
- Support for multi threaded testing.

TestNG Installation in Eclipse

The steps to Download and install TestNG on eclipse are:

9. Launch eclipse IDE -> Click on the Help option within the menu -> Select “Eclipse Marketplace..” option within the dropdown.



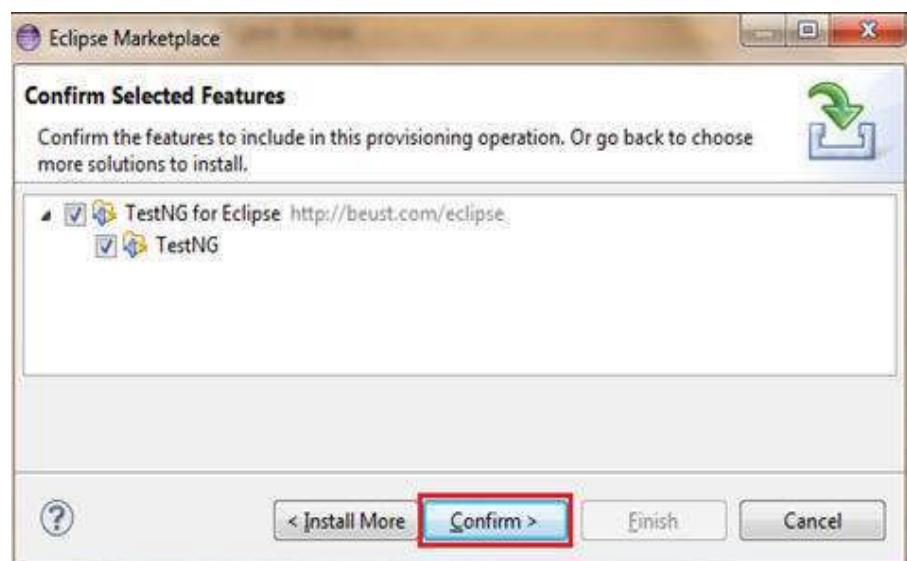
10. Enter the keyword “TestNG” in the search textbox and click on “Go” button as shown below.



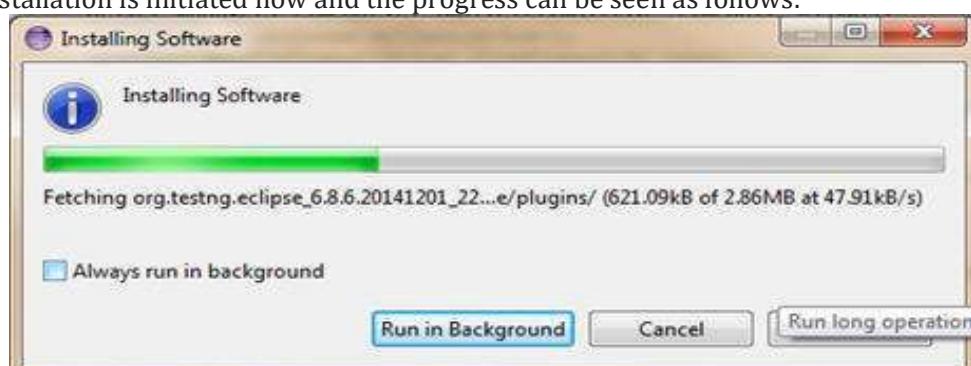
11. Click on the “Go” button, the results matching to the search string would be displayed. Now click on the Install button to install TestNG.



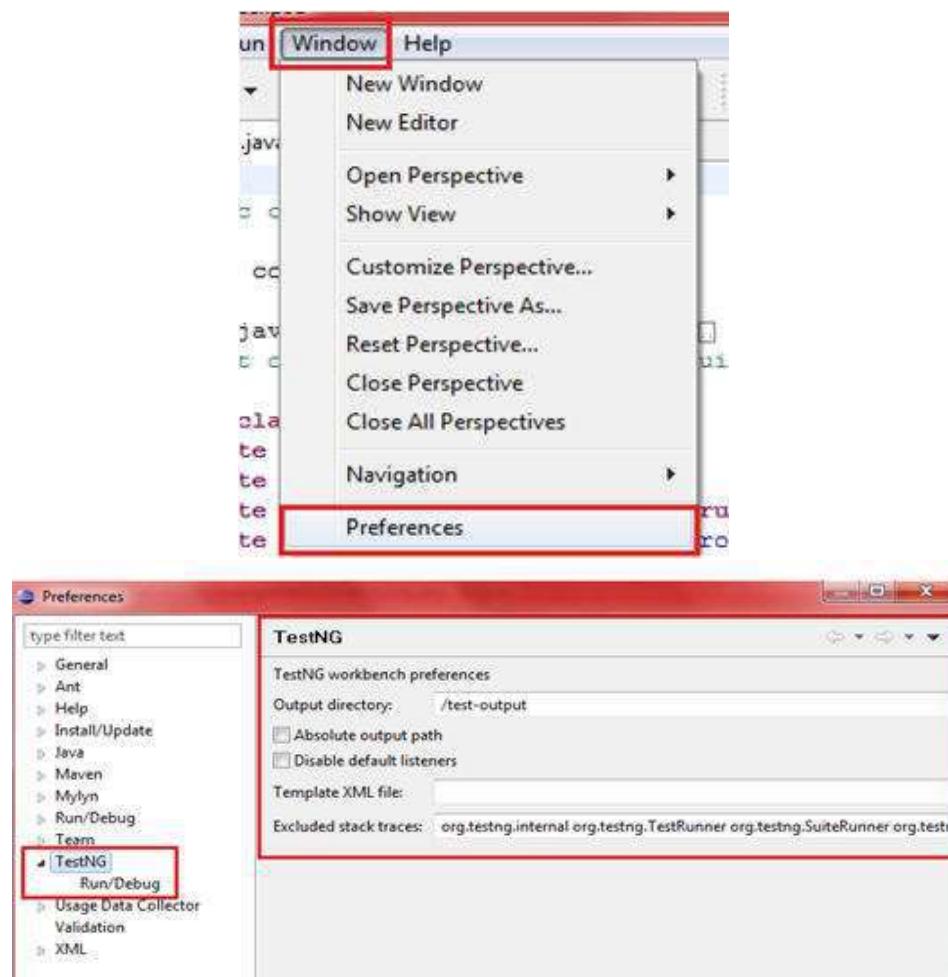
12. Click on the Install button, you will get prompt with a window to confirm the installation.
Click on "Confirm" button.



13. In the next step, the application would prompt you to accept the license and then click on the "Finish" button.
14. The installation is initiated now and the progress can be seen as follows:

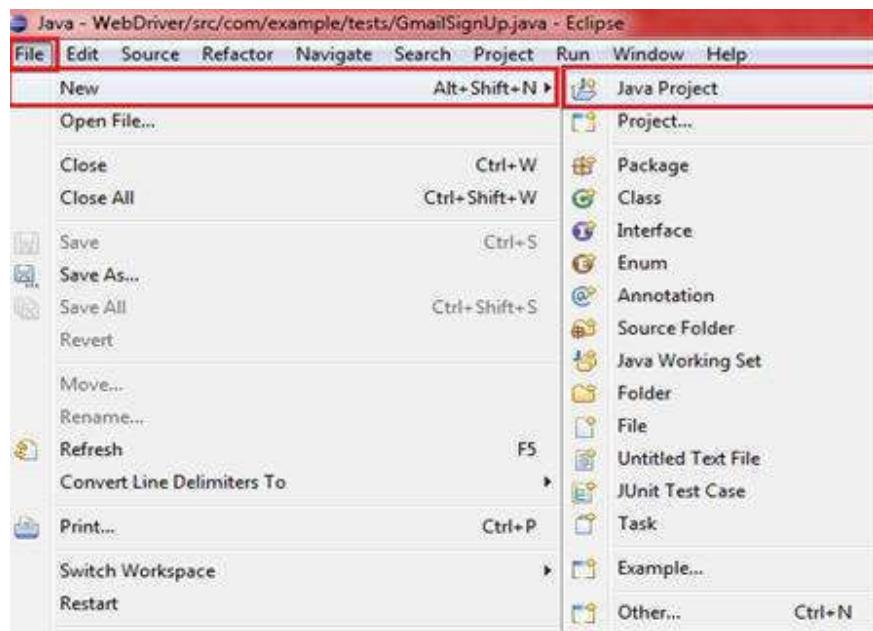


15. Restart eclipse so as to reflect the changes made.
16. Upon restart, verify the TestNG installation by navigating to “Preferences” from “Window” option in the menu bar. Refer the following figure for the same.

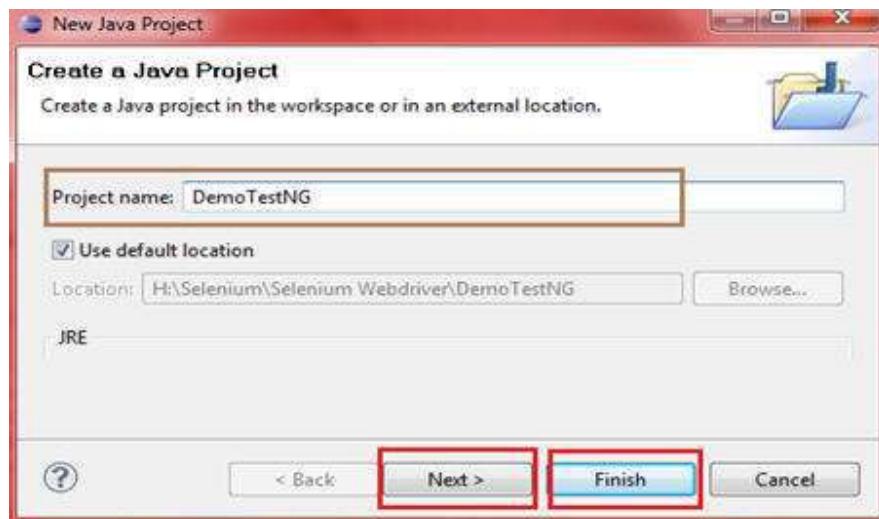


Creation of Sample TestNG project

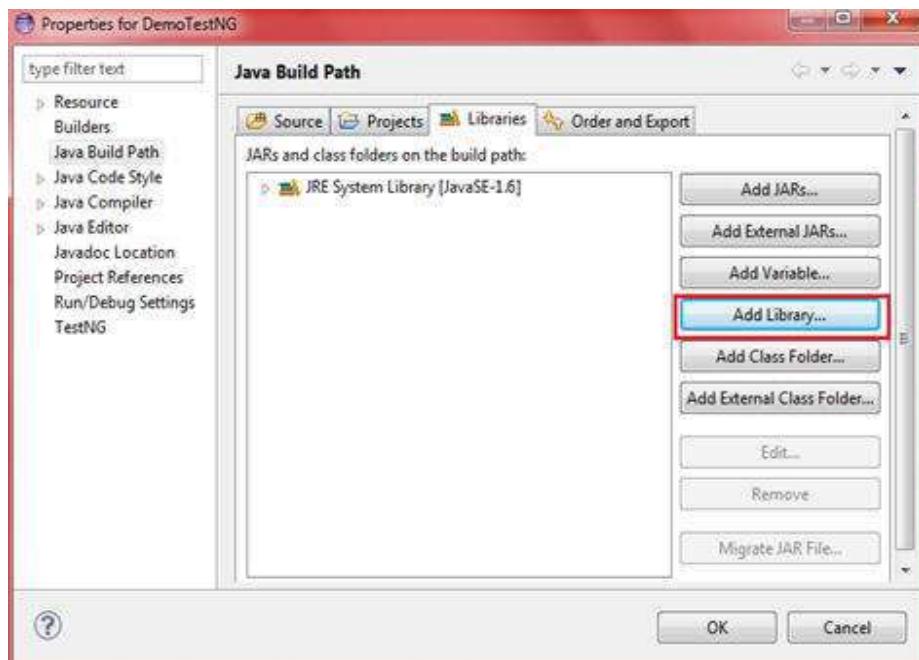
6. Click on the File option within the menu -> Click on New -> Select Java Project.



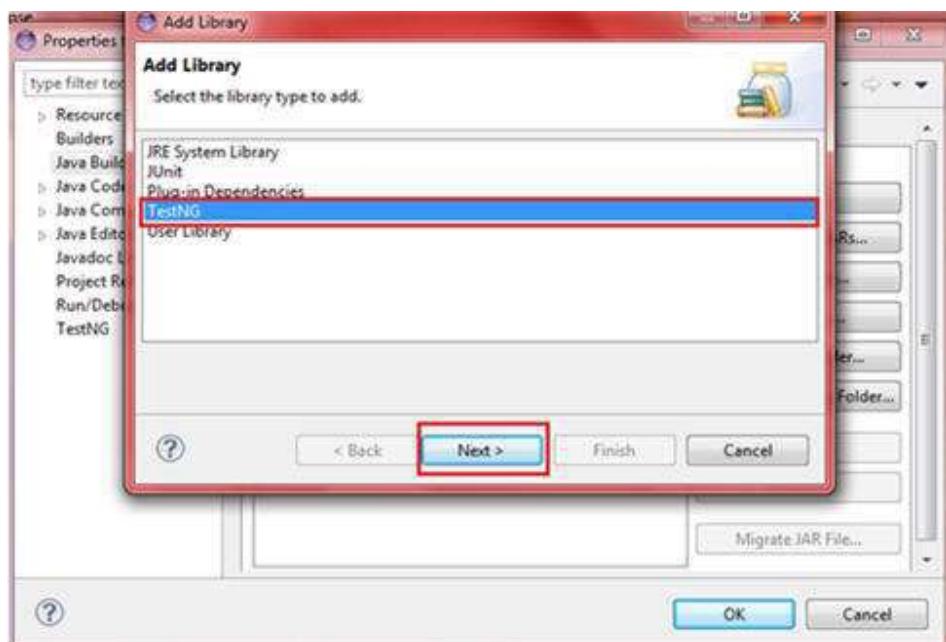
- Enter the project name as "DemoTestNG" and click on "Next" button. As a concluding step, click on the "Finish" button and your Java project is ready.



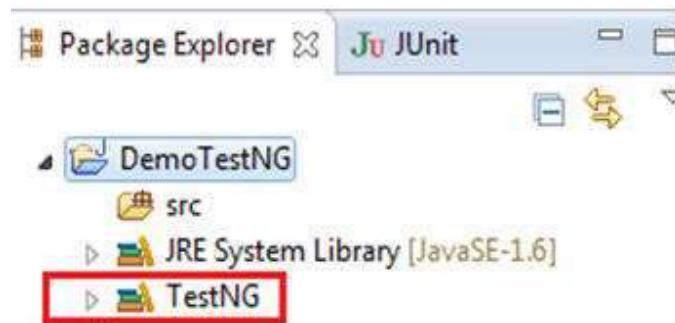
- The next step is to configure the TestNG library into the newly created Java project. For the same, click on the "Libraries" tab under Configure Build Path. Click on "Add library" as shown below.



9. Select TestNG and click on the “Next” button as shown below in the image. In the end, click on the “Finish” button.

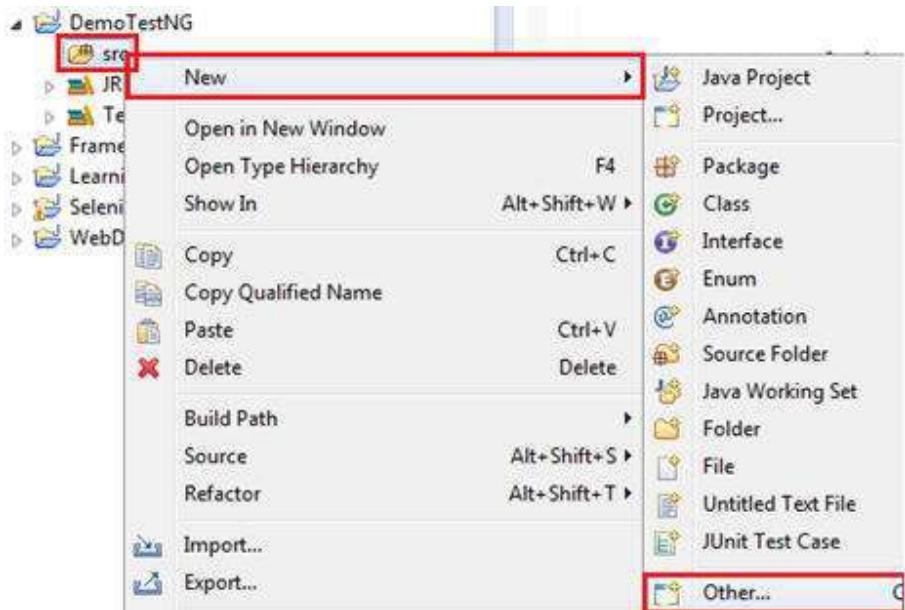


10. The TestNG is now added to the Java project and the required libraries can be seen in the package explorer upon expanding the project.

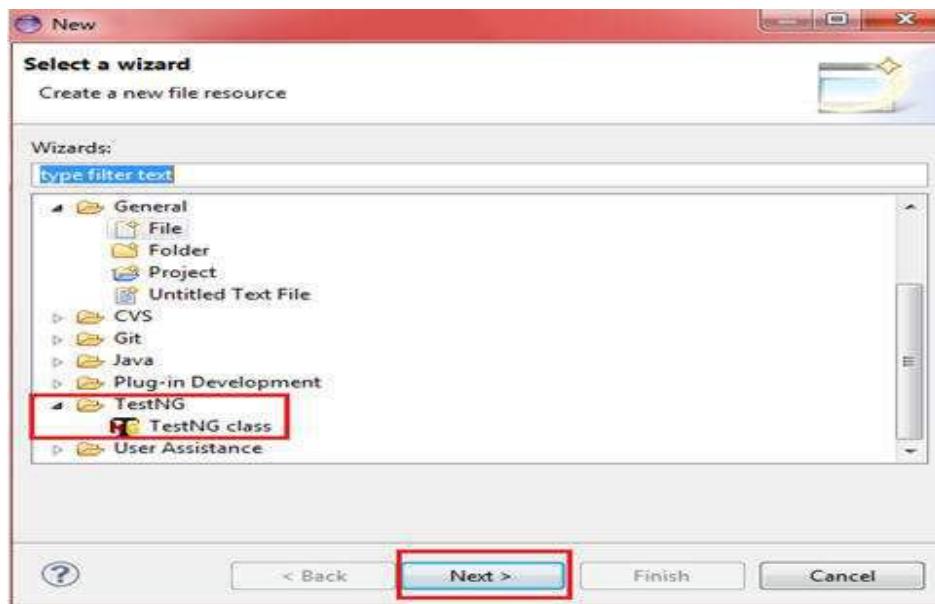


Creating TestNG class

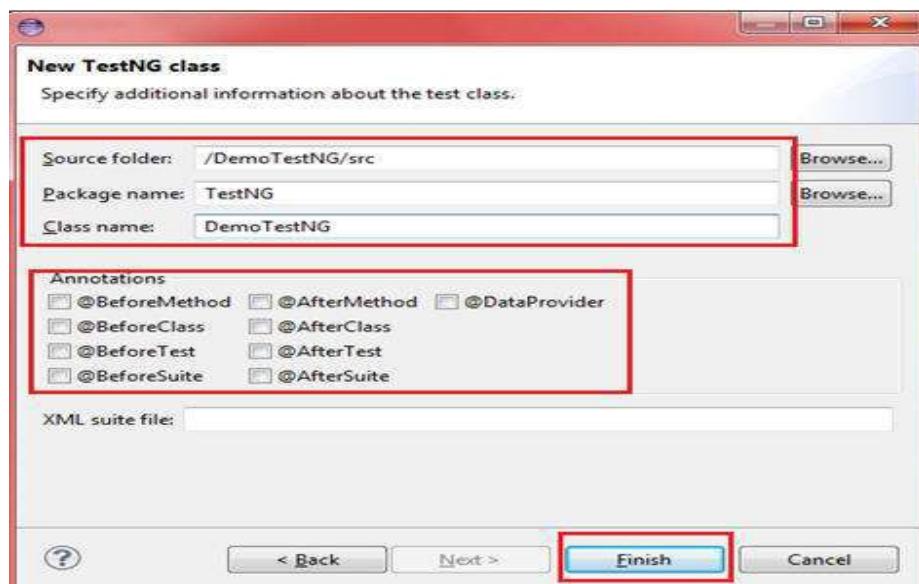
5. Expand the "DemoTestNG" project and traverse to "src" folder. Right-click on the "src" package and navigate to New -> Other.



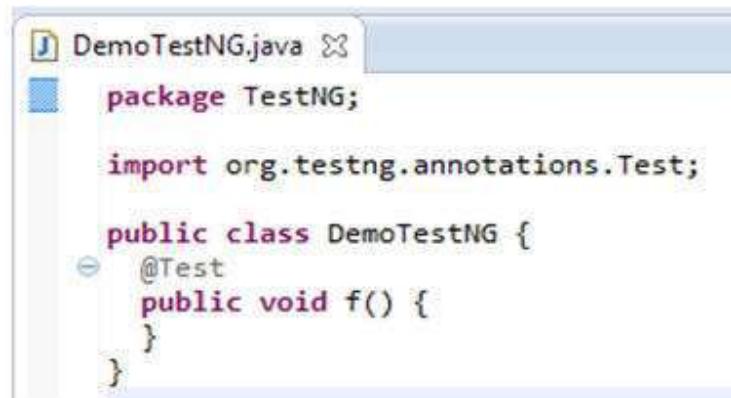
6. Expand TestNG option and select "TestNG" class option and click on the "Next" button.



7. Furnish the required details as following. Specify the Source folder, package name and the TestNG class name and click on the Finish button.



8. The above mentioned TestNG class would be created with the default schema.



```

J DemoTestNG.java ✘
package TestNG;

import org.testng.annotations.Test;

public class DemoTestNG {
    @Test
    public void f() {
    }
}

```

Test Scenario

- Launch the browser and open “gmail.com”.
- Verify the title of the page and print the verification result.
- Enter the username and Password.
- Click on the Sign in button.
- Close the web browser.

Code:

```

package TestNG;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.Assert;

import org.testng.annotations.Test;

public class DemoTestNG {

    public WebDriver driver = new FirefoxDriver();

    String appUrl = "https://accounts.google.com";;

    @Test

    public void gmailLogin() {

        // launch the firefox browser and open the application url

        driver.get("https://gmail.com");
    }
}

```

```
// maximize the browser window

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

// declare and initialize the variable to store the expected title
//of the webpage.

        String expectedTitle = "Sign in - Google
Accounts";

// fetch the title of the web page and save it into a string
//variable

        String actualTitle = driver.getTitle();

        Assert.assertEquals(expectedTitle,actualTitle);

// enter a valid username in the email textbox

        WebElement
username=driver.findElement(By.id("Email"));
);

username.clear();

username.sendKeys("TestSelenium");

// enter a valid password in the password textbox

        WebElement password =
driver.findElement(By.id("Passwd"));

password.clear();

password.sendKeys("password123");

// click on the Sign in button

        WebElement SignInButton =
driver.findElement(By.id("signIn"));

SignInButton.click();

// close the web browser

        driver.close();

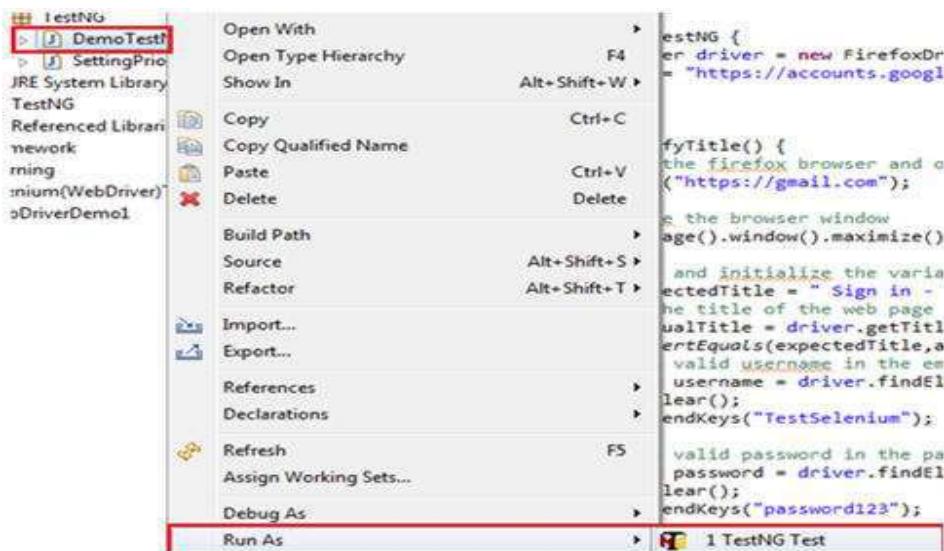
}
```

Code Explanation with respect to TestNG

@Test – @Test is one of the **TestNG annotations**. This annotation lets the program execution to know that method annotated as @Test is a test method. To be able to use different TestNG annotations, we need to import the package “**import org.testng.annotations.***”. There is no need of main() method while creating test scripts using TestNG. The program execution is done on the basis of annotations. In a statement, we used Assert class while comparing expected and the actual value. Assert class is used to perform various verifications. To be able to use different assertions, we are required to import “**import org.testng.Assert**”.

Executing the TestNG script

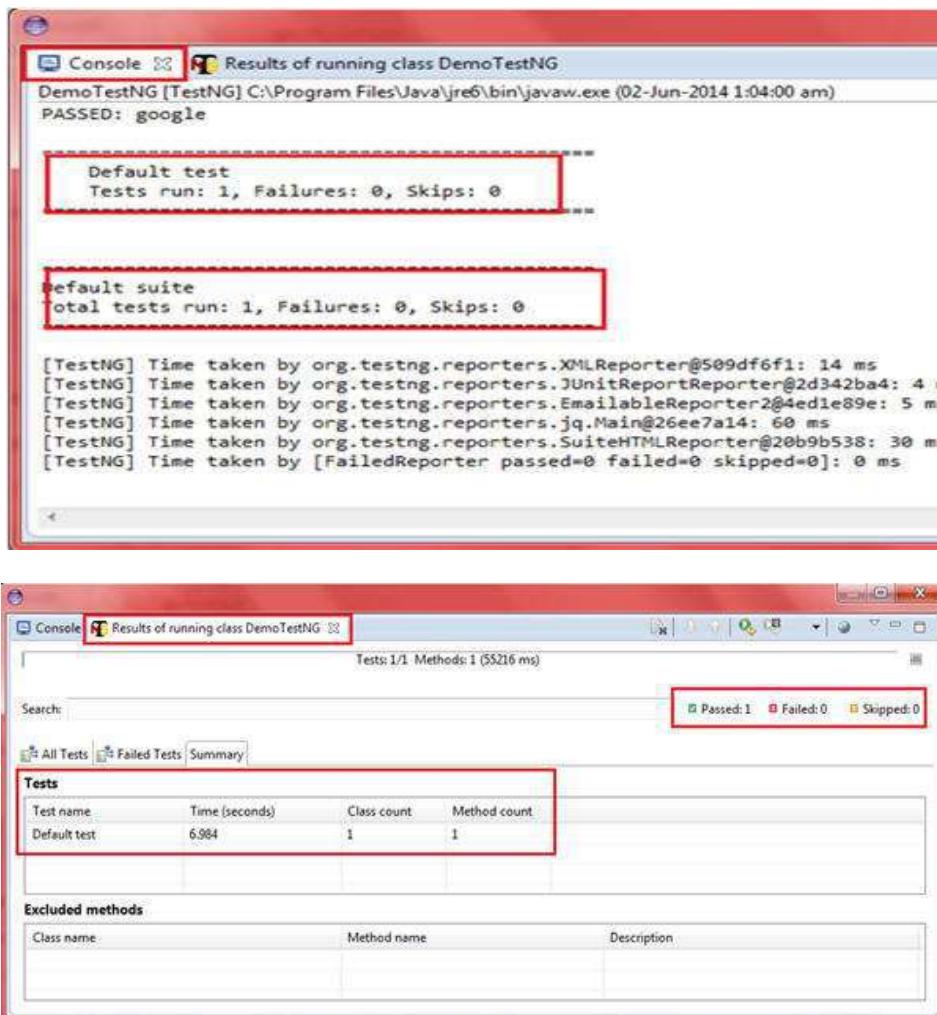
2. Right click anywhere inside the class within the editor or the java class within the package explorer, select “Run As” option and click on the “TestNG Test”.



TestNG result is displayed into two windows:

- Console Window
- TestNG Result Window

Refer the below screencasts for the result windows:

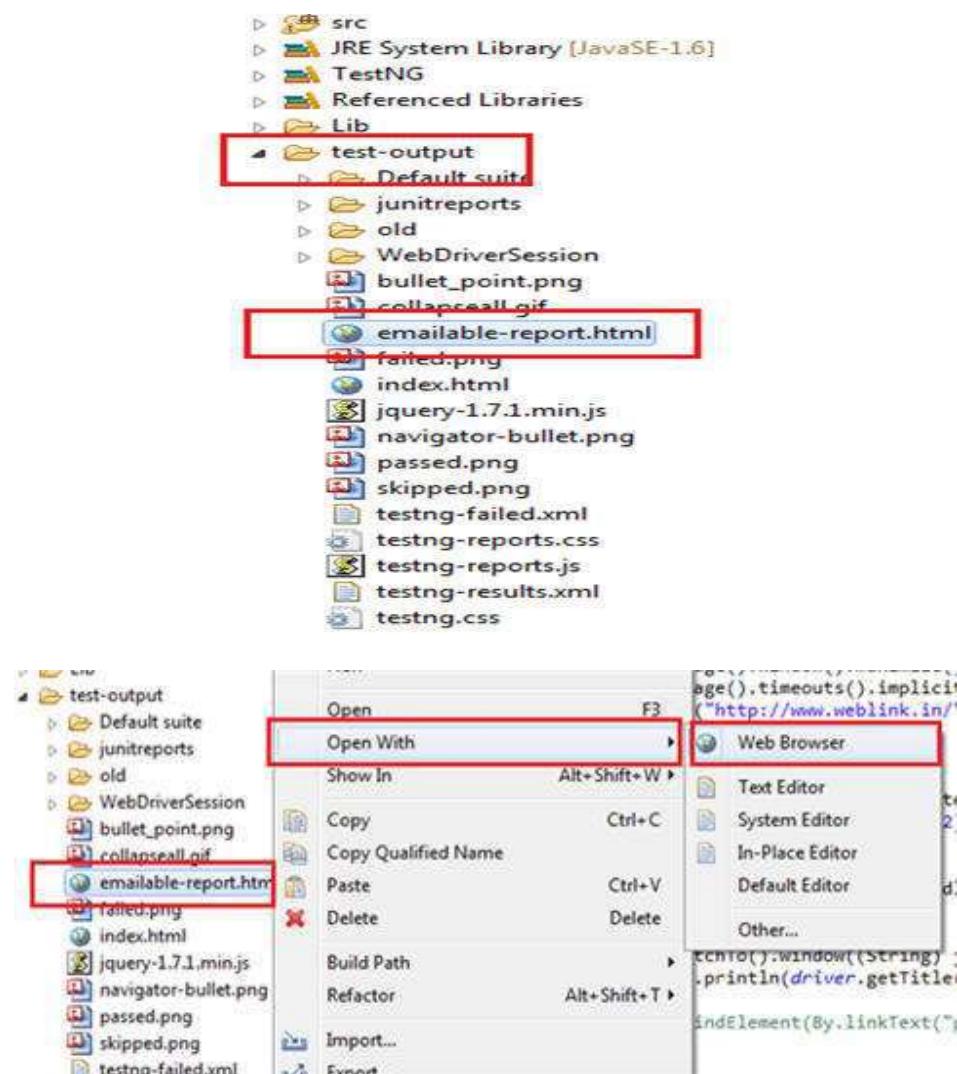


HTML Reports

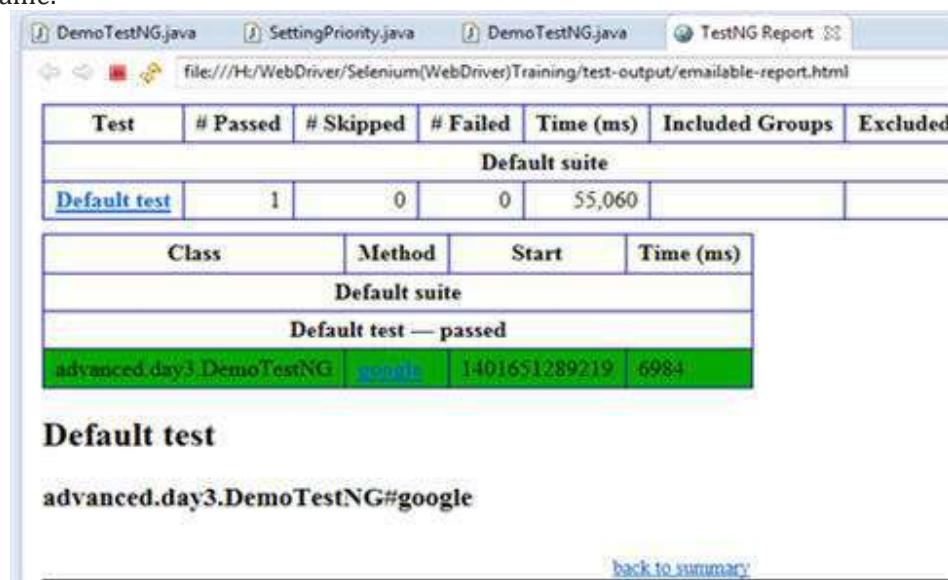
TestNG comes with a great capability of generating user readable and comprehensible HTML reports for the test executions. These reports can be viewed in any of the browsers and it can also be viewed using Eclipse's build -in browser support.

To generate the HTML report, follow the below steps:

4. Execute the newly created TestNG class. Refresh the project containing the TestNG class by right-clicking on it and selecting "Refresh" option.
5. A folder named as "test-output" shall be generated in the project at the "src" folder level. Expand the "test-output" folder and open on the "emailable-report.html" file with the Eclipse browser. The HTML file displays the result of the recent execution.



6. The HTML report shall be opened within the eclipse environment. Refer the below image for the same.



Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups	Excluded
Default suite						
Default test	1	0	0	55,060		
Default suite						
Default test — passed						
advanced.day3.DemoTestNG	google		1401651289219	6984		

Default test

advanced.day3.DemoTestNG#google

[back to summary](#)

What is JAVA Framework

Frameworks are large bodies (usually classes) of prewritten code to which new custom code is added to solve a problem in a specific domain. Thus the framework uses custom code because it is usually the framework that is in control. Technically, a framework is a collection of software libraries or components which provide a defined application programming interface (API). The framework is used by calling its methods, inheritance, and supplying "callbacks", listeners, or other implementations of the Observer pattern.

A very common example of JAVA framework are GUI frameworks, eg Java's Swing and AWT classes. These framework have a huge amount of code to manage the user interface, and there is inversion of control because one starting the GUI framework, then wait for it to call listeners.

The JAVA collections classes are sometimes called a framework due of the size and complexity.

Test Annotations

The Annotations are lines of code that are inserted in the program/ business logic to control how the methods below them are to be run. They are always preceded by the @ symbol. The annotations differ depending on your project requirements. However, the flow of execution will be the same. Both JUnit and TestNG uses annotations. For example TestNG uses @BeforeMethod ,@AfterMethod similar to @Before ,@After in JUnit4.

The list of annotations are:

Description	TestNG	JUnit 4
Test annotation	@Test	@Test
Executes before the first test method is invoked in the current class	@BeforeClass	@BeforeClass
Executes after all the test methods in the current class	@AfterClass	@AfterClass
Executes before each test method	@BeforeMethod	@Before
Executes after each test method	@AfterMethod	@After
annotation to ignore a test	@Test(enable=false)	@ignore
annotation for exception	@Test(expectedExceptions = ArithmeticException.class)	@Test(expected = ArithmeticException.class)

timeout	@Test(timeout = 1000)	@Test(timeout = 1000)
Executes before all tests in the suite	@BeforeSuite	n/a
Executes after all tests in the suite	@AfterSuite	n/a
Executes before a test runs	@BeforeTest	n/a
Executes after a test runs	@AfterTest	n/a
Executes before the first test method is invoked that belongs to any of these groups is invoked	@BeforeGroups	n/a
run after the last test method that belongs to any of the groups here	@AfterGroups	n/a

Example:

1. Create a java class file name TestngAnnotation.java in C:\>TestNG_WORKSPACE to test annotations.

```

import org.testng.annotations.Test;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.AfterSuite;

```

```

public class TestngAnnotation {
    // test case 1
    @Test
    public void testCase1() {
        System.out.println("in test case 1");
    }
}

```

```
// test case 2

@Test
public void testCase2() {
    System.out.println("in test case 2");
}

@BeforeMethod
public void beforeMethod() {
    System.out.println("in beforeMethod");
}

@AfterMethod
public void afterMethod() {
    System.out.println("in afterMethod");
}

@BeforeClass
public void beforeClass() {
    System.out.println("in beforeClass");
}

@AfterClass
public void afterClass() {
    System.out.println("in afterClass");
}

@BeforeTest
public void beforeTest() {
    System.out.println("in beforeTest");
}

@AfterTest
public void afterTest() {
    System.out.println("in afterTest");
}
```

```

@BeforeSuite
public void beforeSuite() {
    System.out.println("in beforeSuite");
}

@AfterSuite
public void afterSuite() {
    System.out.println("in afterSuite");
}

}

```

2. Create the file testng.xml in C:\>TestNG_WORKSPACE to execute annotations.

```

<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

```

```

<suite name = "Suite1">
    <test name = "test1">
        <classes>
            <class name = "TestngAnnotation"/>
        </classes>
    </test>
</suite>

```

3. Compile the Test case class using javac.

```
C:\TestNG_WORKSPACE>javac TestngAnnotation.java
```

4. Run the testing.xml, which will run the test case defined in the provided Test Case class.

```
C:\TestNG_WORKSPACE>java org.testng.TestNG testng.xml
```

Output:

```

in beforeSuite
in beforeTest
in beforeClass
in beforeMethod
in test case 1
in afterMethod
in beforeMethod
in test case 2
in afterMethod
in afterClass
in afterTest
in afterSuite
=====

```

Suite

Total tests run: 2, Failures: 0, Skips: 0

- First of all, beforeSuite() method is executed only once.
- Lastly, the afterSuite() method executes only once.
- Even the methods beforeTest(), beforeClass(), afterClass(), and afterTest() methods are executed only once.
- beforeMethod() method executes for each test case but before executing the test case.
- afterMethod() method executes for each test case but after executing the test case.
- In between beforeMethod() and afterMethod(), each test case executes.

Executing Tests in Sequence

JUnit

In JUnit @FixMethodOrder(MethodSorters.NAME_ASCENDING) is used to run the test methods by method name, in lexicographic order.

Example:

```
package com.mkyong;
import org.junit.FixMethodOrder;
import org.junit.Test;
import org.junit.runners.MethodSorters;
import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;
//Sorts by method name
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class ExecutionOrderTest {
    @Test
    public void testB() {
        assertThat(1 + 1, is(2));
    }
    @Test
    public void test1() {
        assertThat(1 + 1, is(2));
    }
    @Test
    public void testA() {
        assertThat(1 + 1, is(2));
    }
}
```

```

    @Test
    public void test2() {

        assertEquals(1 + 1, 2);
    }

    @Test
    public void testC() {
        assertEquals(1 + 1, 2);
    }
}

```

Output

```

test1
test2
testA
testB
testC

```

TestNG

TestNG provides three ways to set ordering for tests:

- Using preserve-order in the testng.xml file

The preserve-order attribute in the testng.xml file is used to have TestNG run the tests in the order they appear in the XML file:

Example:

```

<test name="OrderedTestNGTests" preserve-order="true">
    <classes>
        <class name="TestNGTestClass">
            <methods>
                <include name="testOne" />
                <include name="testTwo" />
            </methods>
        </class>
    </classes>
</test>

```

- Using the priority attribute

The priority attribute in @Test annotation can be used to prioritize test methods and determine the order in which they are run

Example:

```

public class TestNGPrioritized {
    @Test(priority = 3)
    public void testThree() {
    }

    @Test(priority = 1)
    public void testOne() {
    }

    @Test(priority = 2)
    public void testTwo() {
    }
}

```

- Using dependencies

In TestNG, we can have tests and test suites depend on other tests or test suites. This also implicitly defines the order in which the tests are executed. For example, when test A depends on test B, test B will automatically be run before test A. These dependencies can be defined in code.

Example:

```

public class TestNGOrderedTests {
    @Test(dependsOnMethods = {"parentTest"})
    public void childTest() {
    }

    @Test
    public void parentTest() {
    }
}

```

Alternatively, we can define dependencies on group level in the testng.xml file:

```

<test name="TestNGOrderedTests">
    <groups>
        <dependencies>
            <group name="parenttests" />
            <group name="childtests" depends-
on="parenttests" />
        </dependencies>
    </groups>
</test>

```

Assertions

Selenium commands are of three types:

- Actions: generally manipulate the state of the application like “click this link” and “select that option”. If an Action fails, or has an error, the execution of the current test is stops.
- Accessors: examine the state of the application and store the results in variables, e.g. “Title”.
- Assertions verify that the state of the application is same to what we are expecting..

Assertions provide a means to validate any kind of test. The Assertion results are based on the comparison of actual & expected results. A test is considered passed only when the Assertions show no Exceptions. Selenium Assertions can be of three types: “assert”, “verify”, and “waitFor”.

When an “assert” fails, the test is aborted.

When a “verify” fails, the test will continue execution, logging the failure.

A “waitFor” command waits for some condition to become true. They will fail and halt the test if the condition does not become true within the current timeout setting.

The various used assertions are:

- **Assert Equals:** Assert Equals works by comparing the expected condition with that of the actual condition based on which the test results are displayed.
- **Assert Not Equals:** “Assert Not Equals” serves the purpose of negation testing for Testers. If the actual and expected **do not match, then the test Script passes else it fails.**
- **Assert True:** This Assertion passes the Test Step only when the boolean value returned is “True”
- **Assert False:** “Assert False” passes the Test Step only when the boolean value returned is “False”
- **Assert Null:** This Assertion verifies if the object under test is null, and the passes the same if the result is so.
- **Assert Not Null:** This Assertion functions opposite to that of “Assert Null”. Thus this Assertion verifies if the object under test is not null, and the passes the same if the result is so.
- **Assert Same:** This Assertion checks that two objects refer to the same object, if it does, then the Assertion passes else it fails the same.
- **Assert Not Same:** This Assertion checks that two objects do not refer to the same object, if it does than the Assertion passes else it fails the same.

Error Collectors

ErrorCollector in JUnit is a rule that allows execution of a test to continue after the first problem is found. Error collector objective is to collect all the error which comes up with script execution and report it only at the end. To give more understanding consider q case while testing script any line of code fails due to network failure, assertion failure, or any other reason. In that situation, executing test script can continue by using “error collector.”

JUnit uses @Rule annotation which is used to create an object of error collector. Once the object for error collector is created the errors can be added into the object using method addError. Throwable is the super class of Exception and Error class in Java. The benefit of adding all errors in an Error Collector is that one can verify all the errors at once. Also, if the script fails in the middle, it can still continue executing it.

Example:

- Create a class and a rule to collect all the errors and add all the errors using `addError(Throwable)`.

```

import org.junit.Assert;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.ErrorCollector;
public class ErrorCollectorExample {
    @Rule
    public ErrorCollector collector = new ErrorCollector();

    @Test
    public void example() {
        collector.addError(new Throwable("There is an error in
first line"));
        collector.addError(new Throwable("There is an error in
second line"));

        System.out.println("Hello");
        try {
            Assert.assertTrue("A " == "B");

        } catch (Throwable t) {
            collector.addError(t);
        }
        System.out.println("World!!!!");
    }
}

```

- Add above test class in a test runner and execute it to collect all errors.

```

import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

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

        Result result =
JUnitCore.runClasses(ErrorCollectorExample.class);

        for (Failure failure : result.getFailures()) {

            System.out.println(failure.toString());
        }
        System.out.println("Result=="+result.wasSuccessful());
    }
}

```

}

How to Parameterize Test Cases

Parameterized tests allow a developer to run the same test over and over again using different values. There are five steps to create a parameterized test.

- Annotate test class with `@RunWith(Parameterized.class)`.
- Create a public static method annotated with `@Parameters` that returns a Collection of Objects (as Array) as test data set.
- Create a public constructor that takes in what is equivalent to one "row" of test data.
- Create an instance variable for each "column" of test data.
- Create your test case(s) using the instance variables as the source of the test data.

Example:

1. Create a java class to be tested, say, PrimeNumberChecker.java in C:\>JUNIT_WORKSPACE.

```
public class PrimeNumberChecker {
    public Boolean validate(final Integer primeNumber) {
        for (int i = 2; i < (primeNumber / 2); i++) {
            if (primeNumber % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

2. Create a java test class, say, PrimeNumberCheckerTest.java. Create a java class file named PrimeNumberCheckerTest.java in C:\>JUNIT_WORKSPACE.

```
import java.util.Arrays;
import java.util.Collection;
import org.junit.Test;
import org.junit.Before;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;
import org.junit.runner.RunWith;
import static org.junit.Assert.assertEquals;
@RunWith(Parameterized.class)
public class PrimeNumberCheckerTest {
    private Integer inputNumber;
    private Boolean expectedResult;
```

```

private PrimeNumberChecker primeNumberChecker;
@Before
public void initialize() {
    primeNumberChecker = new PrimeNumberChecker();
}
// Each parameter should be placed as an argument here
// Every time runner triggers, it will pass the arguments
// from parameters we defined in primeNumbers() method
public PrimeNumberCheckerTest(Integer inputNumber, Boolean expectedResult) {
    this.inputNumber = inputNumber;
    this.expectedResult = expectedResult;
}
@Parameterized.Parameters
public static Collection primeNumbers() {
    return Arrays.asList(new Object[][] {
        { 2, true },
        { 6, false },
        { 19, true },
        { 22, false },
        { 23, true }
    });
}
// This test will run 4 times since we have 5 parameters
defined
@Test
public void testPrimeNumberChecker() {
    System.out.println("Parameterized Number is : " +
inputNumber);
    assertEquals(expectedResult,
primeNumberChecker.validate(inputNumber));
}
}

```

3. Create a java class file named TestRunner.java in C:\>JUNIT_WORKSPACE to execute test case(s).

```

import org.junit.runner.JUnitCore;
import org.junit.runner.Result;

```

```

import org.junit.runner.notification.Failure;
public class TestRunner {
    public static void main(String[] args) {
        Result result =
JUnitCore.runClasses(PrimeNumberCheckerTest.class);
        for (Failure failure : result.getFailures()) {
            System.out.println(failure.toString());
        }

        System.out.println(result.wasSuccessful());
    }
}

```

4. Compile the PrimeNumberChecker, PrimeNumberCheckerTest and Test Runner classes using javac.

C:\JUNIT_WORKSPACE>javac PrimeNumberChecker.java

PrimeNumberCheckerTest.java

TestRunner.java

Now run the Test Runner, which will run the test cases defined in the provided Test Case class.

C:\JUNIT_WORKSPACE>java TestRunner

Output:

```

Parameterized Number is : 2
Parameterized Number is : 6
Parameterized Number is : 19
Parameterized Number is : 22
Parameterized Number is : 23
True

```

Summary

- JUnit is a unit testing framework for Java programming language
- A Unit Test Case is a part of code, which ensures that another part of code (method) works as expected
- JUnit promotes the idea of "first testing then coding"
- Frameworks are large bodies (usually classes) of prewritten code to which new custom code is added to solve a problem in a specific domain
- The Annotations are lines of code that are inserted in the program/ business logic to control how the methods below them are to be run
- Assertions provide a means to validate any kind of test

DUCAT®
www.ducatindia.com

Chapter 16 – Maven

Introduction to Maven

Maven is a simple build automation tool which is basically used for enterprise Java projects, designed to take much of the hard work out of the build process. Maven uses a declarative approach, where the project structure and contents are described, rather than the task-based approach used in Ant or in traditional make files, for example. This helps enforce company-wide development standards and reduces the time needed to write and maintain build scripts.

The ease of source code compilation, distribution, documentation, collaboration with different teams and other vital tasks are seamless by using maven. Maven aims to describe 2 important things as how a software is built and the dependencies, plug-ins & profiles that the project is associated in a standalone or a distributed environment.

The maven can also be used in building & managing the projects written using C#, ruby and other programming languages as well.

Maven History

Maven was initially designed and developed by the Jakarta turbine project. At the later point of time, the Apache group developed the Maven to such an extent to support developing & building multiple projects together, publishing those projects, deploying them and generating the reports. The JARs/WARs of any maven project can be shared across any distributed environments.

Advantages of Using Maven over Ant

- Managing dependencies.
- Convention over configuration - configuration is very minimal
- Multiple/Repeated builds can be achieved.
- Focus on automation.
- Plugin management.
- Testing - ability to run JUnit and other integration test suites.
- Making the development process transparent.
- Provision to check the status of each build.
- Avoiding inconsistent setups.
- Standard and uniformed infrastructure among projects.
- Few Glossaries around Maven

Maven Repository

Local Repository

This is the place where Maven stores all the project jars files or libraries or dependencies. By default the folder name is '.m2' and by default the location in windows 7 is 'Libraries\Documents\.m2'.

Central Repository

Maven central repository is the default location 'http://mvnrepository.com/' for Maven to download all the project dependency libraries. For any library required in the project, Maven first look in to the .m2 folder of Local Repository, if it does not find the required library then it looks in Central Repository and download the library in to local repository.

Dependency Keyword

Dependencies are the libraries, which are required by the project. For example Log4j jars, Apache Poi jars, Selenium Jars. Dependencies are mentioned in the Maven pom.xml like this:

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>2.43.1</version>
</dependency>
```

Surefire Plugin

The Surefire Plugin is used during the test phase of the build lifecycle to execute the unit tests of an application. It generates reports in 2 different file formats like plain text file, xml files and html files as well. Even if you are using TestNG or Junits framework for reporting, this plugin is must to use, as it helps Maven to identify tests.

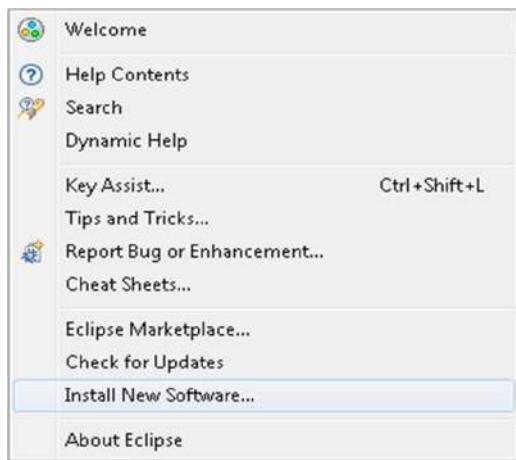
Maven POM

POM is Project Object Model XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects. Some of the configuration that can be specified in the POM are the project dependencies, the plugins or goals that can be executed, the build profiles.

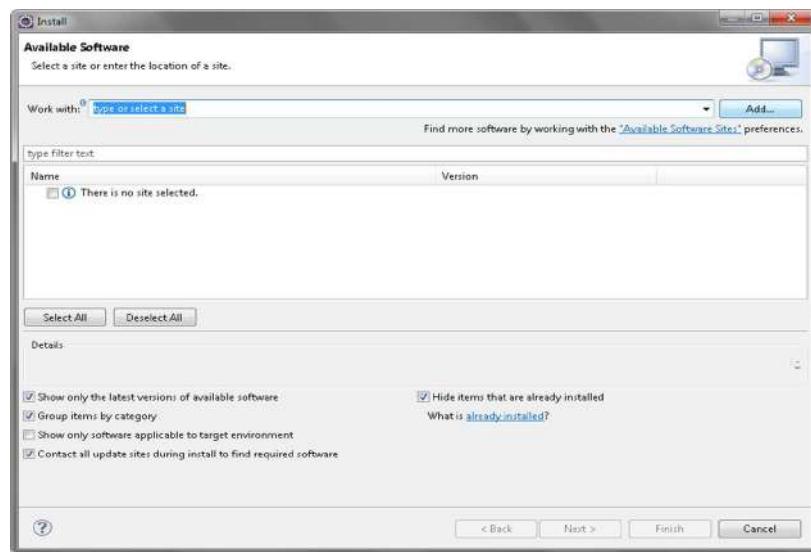
Install Maven on Eclipse IDE

The steps to Install Maven in Eclipse IDE are as below:

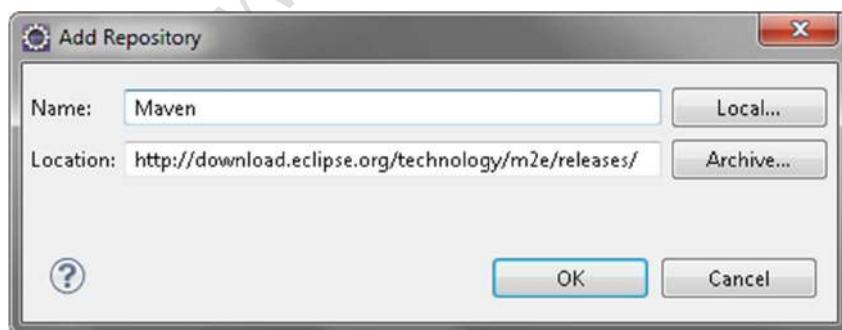
1. Click on the Help from the top menu in Eclipse and select 'Install New Software'



2. Click on the Add button on the newly opened window.

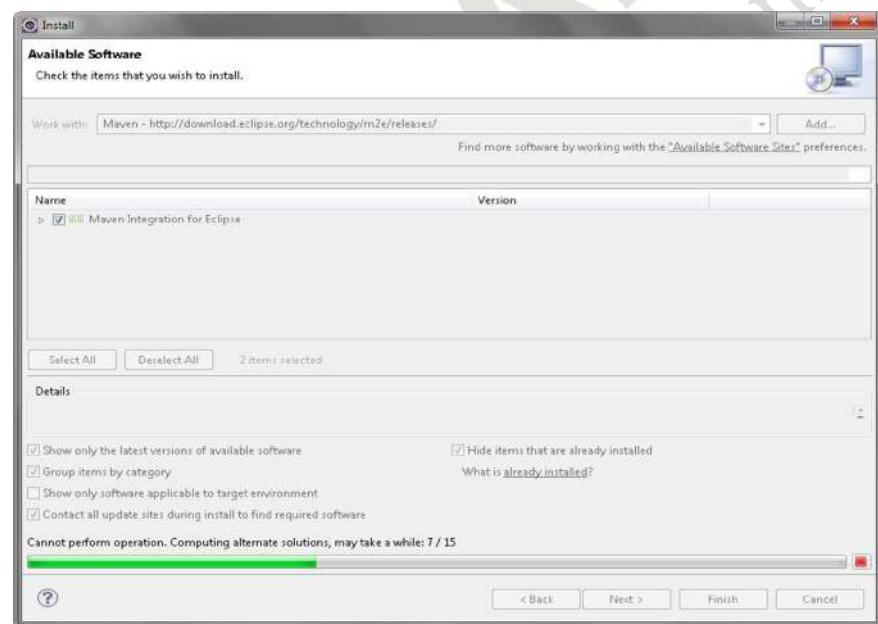
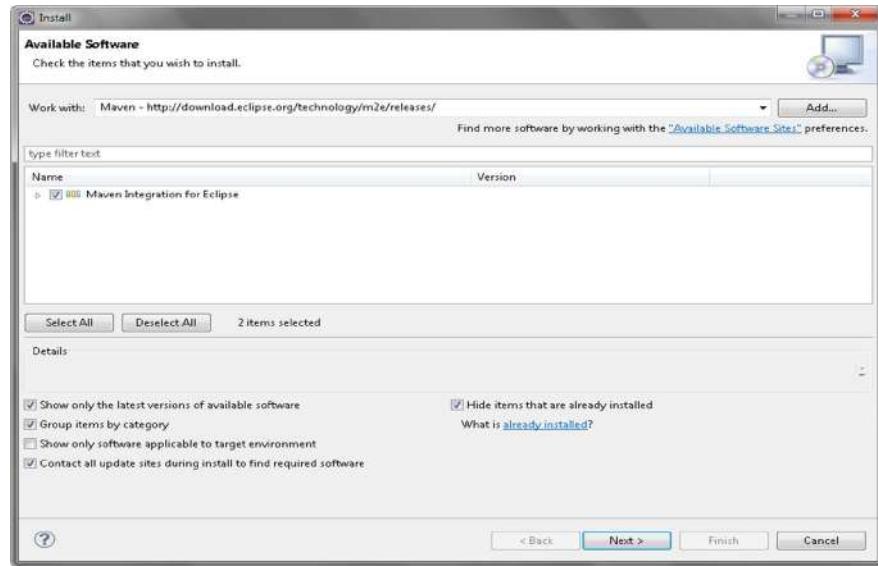


3. In the Name box, type 'Maven' and in the Location box, type '<http://download.eclipse.org/technology/m2e/releases/>'

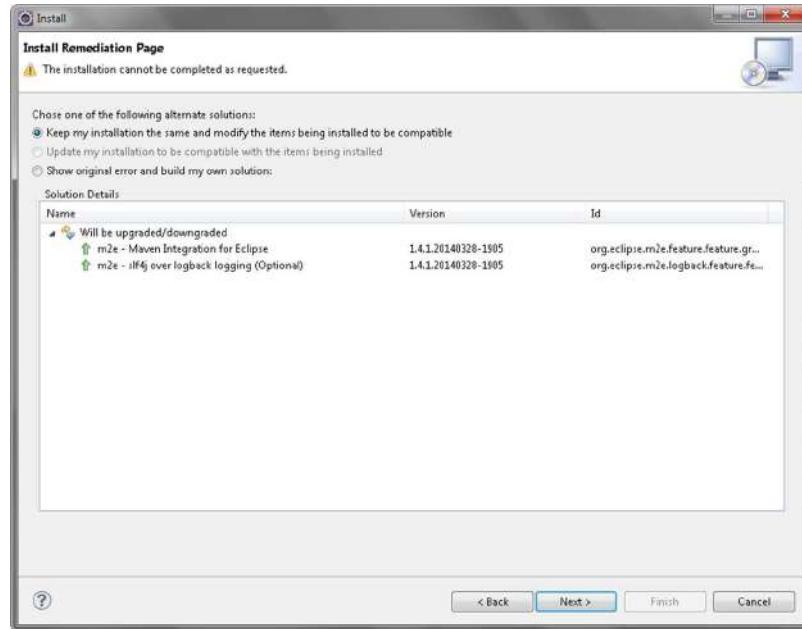


Note: The URL is the location where you can download the Maven for Eclipse.

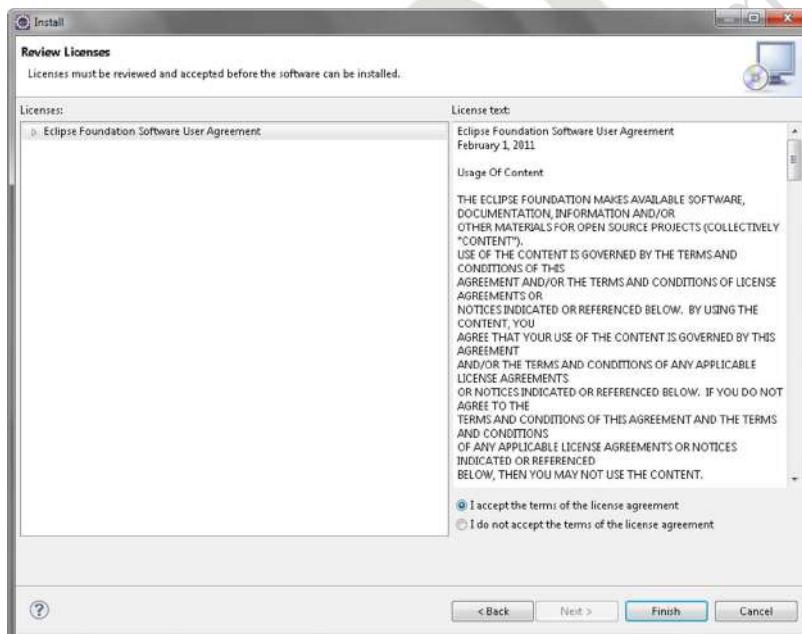
4. A check-box will appear in the pop window, Check the check-box and click on Next button.



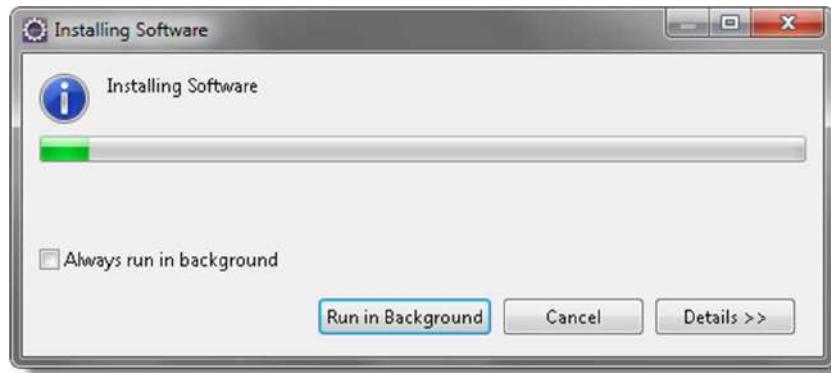
5. Keep the default settings and click on Next button.



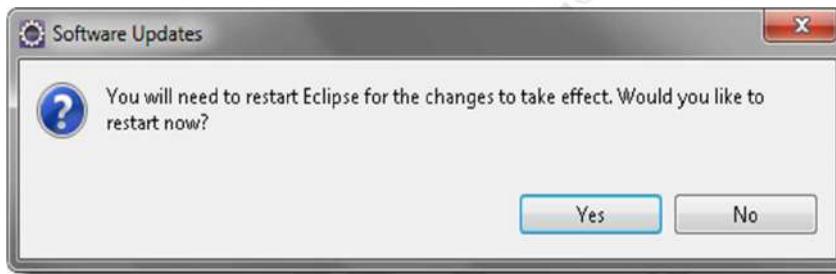
- Accept the 'Terms and Conditions' and move forward by clicking on Finish button.



- Wait while it finish the installation.



8. Once the installation is finished, it will ask to restart the Eclipse. Please click on Yes, so that changes can be reflected properly.



Install Maven on Windows

Maven is a build and dependency management tool for java based application development. Just like other java based development tools, it is not installed as windows service, rather it is configured using windows environment variables. These variables can be accessed from below location:

All Control Panel Items > System > Advanced system settings > Environment Variables

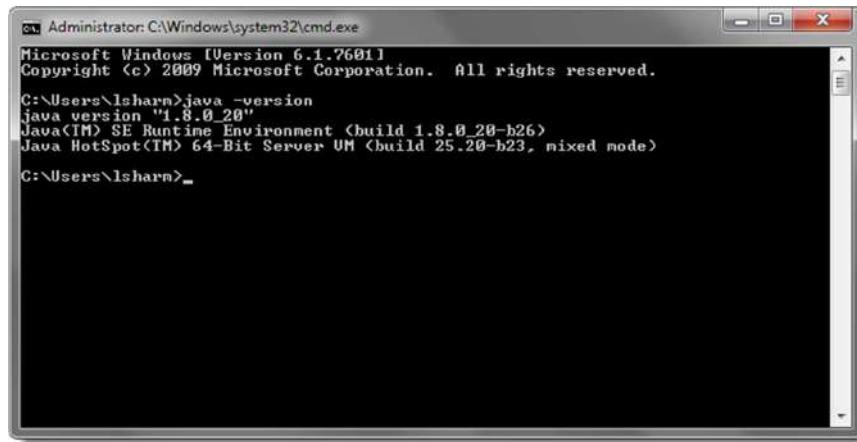
There are certain steps which are need to perform to set up Maven, such as installing Java on the system, setting up environment variable and downloading Eclipse IDE.

Follow the below steps one by one to set Maven in Windows:

1. Download and Install Java

Install Java, if it is not already installed. Java latest version can be installed from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

- To check the Java version installed on the machine, please go to Run and type 'cmd' to launch the command prompt.
- Now type 'Java -version' and press Enter.



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

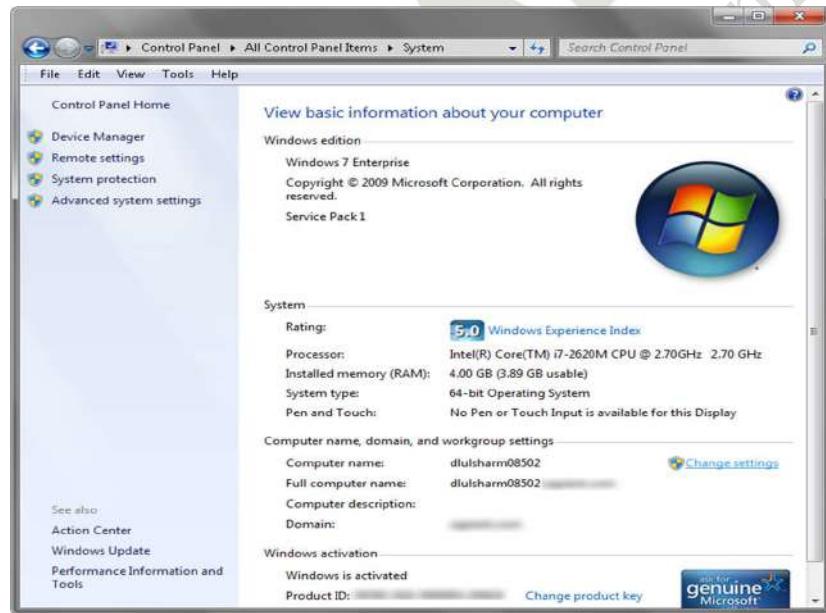
C:\Users\lsharm>java -version
java version "1.8.0_20"
Java(TM) SE Runtime Environment (build 1.8.0_20-b26)
Java HotSpot(TM) 64-Bit Server VM (build 25.20-b23, mixed mode)

C:\Users\lsharm>
```

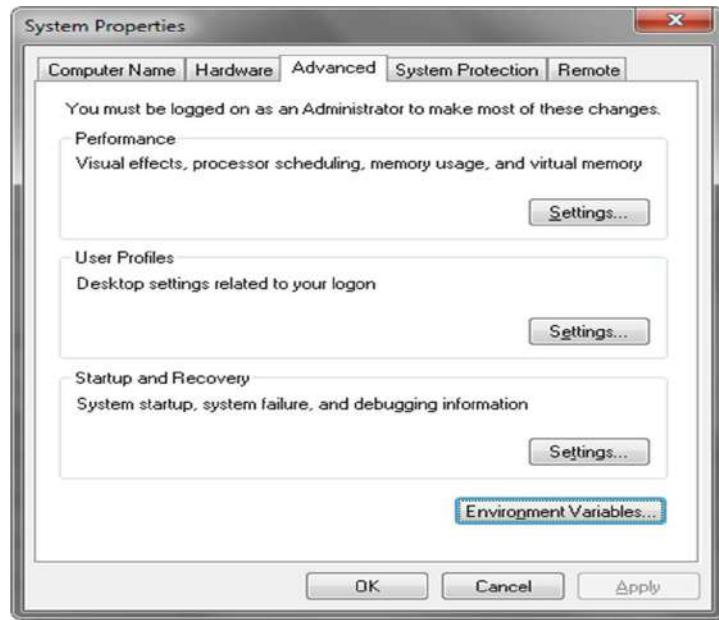
2. Set Up Java Environment Variable

Once the Java installation is done, set up the Java Environment Variable. To set the Java environment variable, open System Settings.

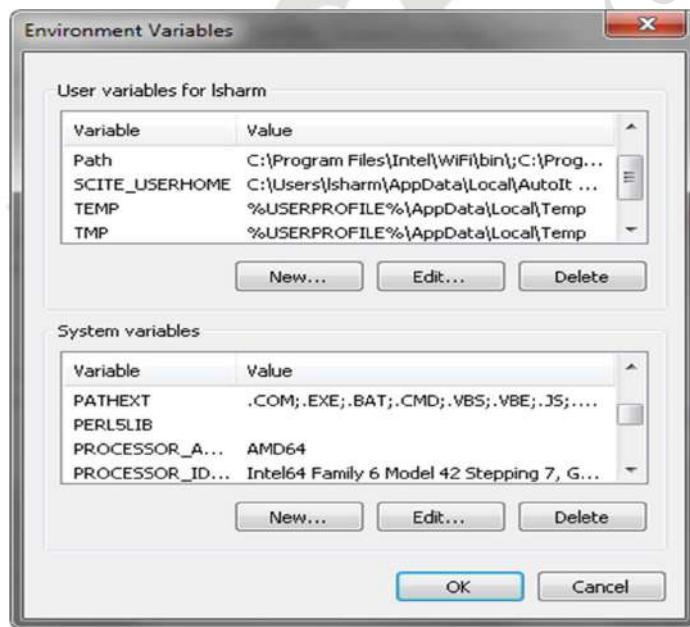
- Go to My Computer and select Properties by Right Clicking on empty space in My Computer.
- Click on 'Change settings'.



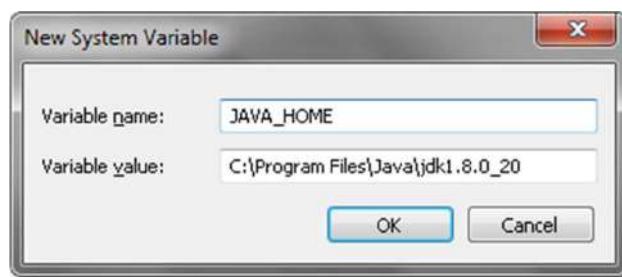
- A pop up window will display, click on 'Advanced' tab and then click on 'Environment Variables..'.



- Click on New button under 'System Variables.'



- Write 'JAVA_HOME' in the Variable name box and enter 'C:\Program Files\Java\jdk1.8.0_20' JDK path in the Variable value box and click OK.



- Entry for newly created Java variable will be displayed under 'System Variables..'



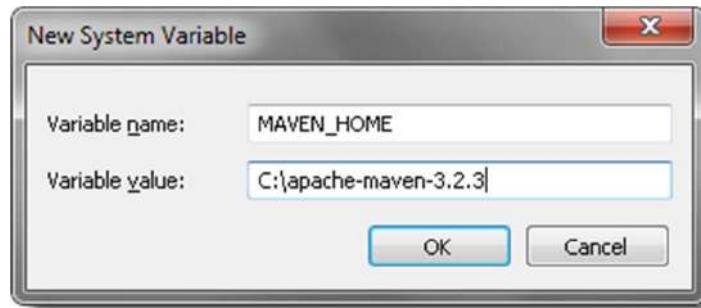
3. Download Maven and Set up Maven Environment Variable

- Next step is to download the Maven and it can be downloaded from below link :
<http://maven.apache.org/download.cgi>.

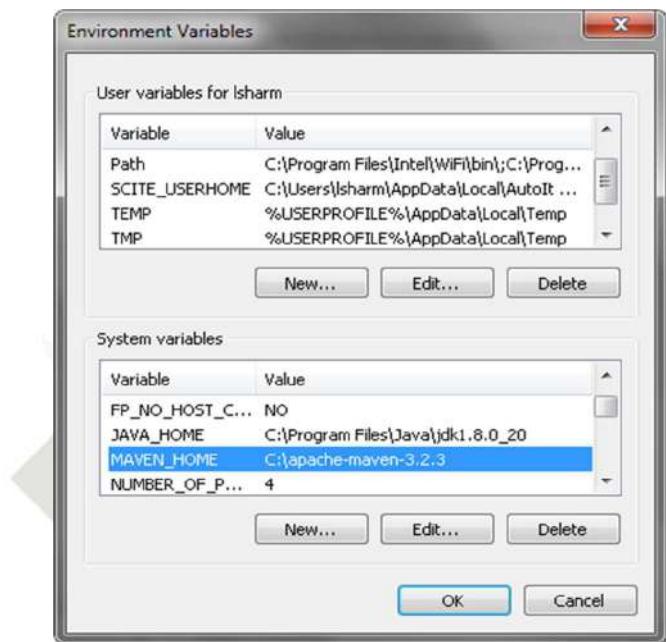
	Link	checksum	signature
Maven 3.2.3 (Binary tar.gz)	apache-maven-3.2.3-bin.tar.gz	apache-maven-3.2.3-bin.tar.gz.md5	apache-maven-3.2.3-bin.tar.gz.asc
Maven 3.2.3 (Binary zip)	apache-maven-3.2.3-bin.zip	apache-maven-3.2.3-bin.zip.md5	apache-maven-3.2.3-bin.zip.asc
Maven 3.2.3 (Source tar.gz)	apache-maven-3.2.3-src.tar.gz	apache-maven-3.2.3-src.tar.gz.md5	apache-maven-3.2.3-src.tar.gz.asc
Maven 3.2.3 (Source zip)	apache-maven-3.2.3-src.zip	apache-maven-3.2.3-src.zip.md5	apache-maven-3.2.3-src.zip.asc
Release Notes	3.2.3		
Release Reference Documentation	3.2.3		

- Extract it to some location. Here it is extracted to 'C:/apache-maven-3.2.3'.
- Set up the Maven Environment Variable the same way we set up the Java Environment Variable above.

Write 'MAVEN_HOME' in the Variable name box then enter 'C:\apache-maven-3.2.3' Maven path in the Variable value box and click OK.



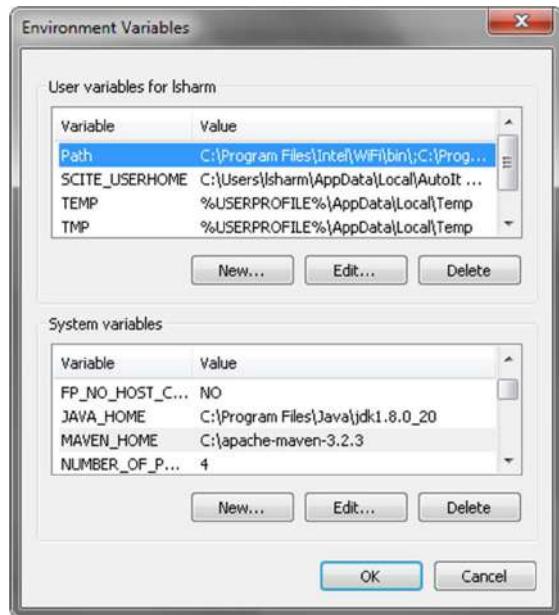
- Entry for newly created Maven variable will be displayed under 'System Variables..'



4. Update the Path Variable

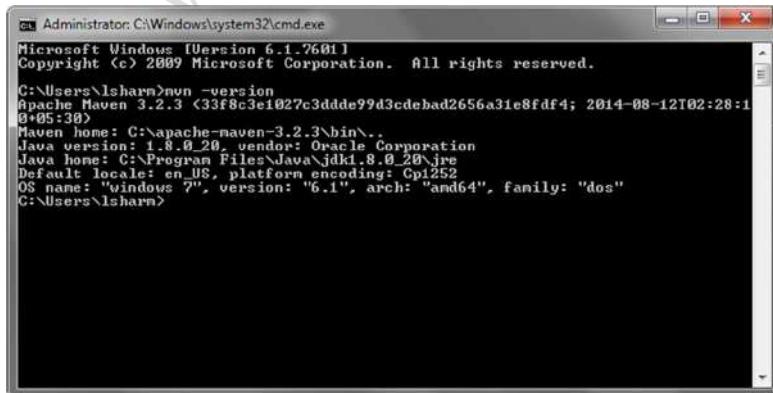
To run Maven from command prompt, this is necessary to update the Path Variable with Maven's installation 'bin' directory.

- Go to My Computer and select Properties by Right Clicking on empty space in My Computer.
- Click on 'Change Settings'.
- A pop up window will display, click on 'Advanced' tab and then click on 'Environment Variables..'.
- Click on Edit button under 'User Variables for UserName', where UserName is your machine name.
- Write 'PATH' in the Variable name box then enter 'C:\apache-maven-3.2.3\bin' Maven path in the Variable value box and click OK.



5. Test the Maven Installation

Maven installation is complete. Now lets test it from windows command prompt. Go to Run and type 'cmd' in application location search box. A new command prompt will be opened. Type mvn -version in command prompt.



Install Maven on Mac

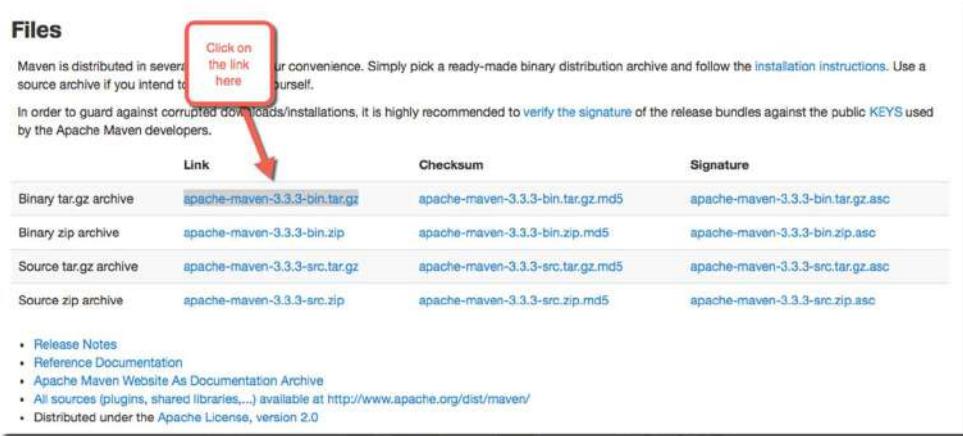
Maven is a tool that is also available on Mac machine. Maven basics remain same across Windows and Mac machines.

Prerequisite: Java should be installed on the Mac machine.

1. Download Maven binaries

First download to Maven binaries. To do so click on the following link:

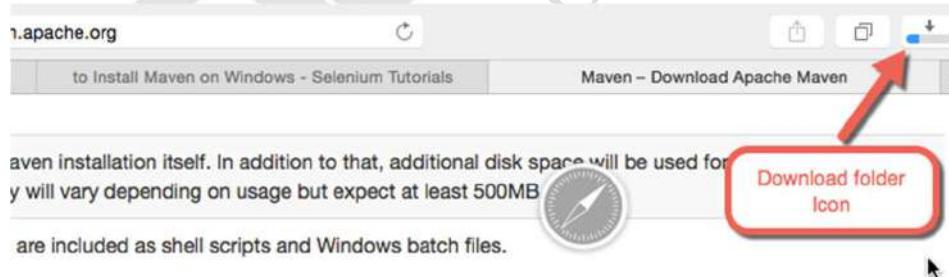
<https://maven.apache.org/download.cgi> On the downloads page above click on the link apache-maven-3.3.3-bin.tar.gz link. Here is a small screenshot:



The screenshot shows the Apache Maven Downloads page. A red box highlights the link "apache-maven-3.3.3-bin.tar.gz" under the "Link" column for the "Binary tar.gz archive". An arrow points from the text "Click on the link here" to this highlighted link. Below the table, there is a list of links:

- Release Notes
- Reference Documentation
- Apache Maven Website As Documentation Archive
- All sources (plugins, shared libraries,...) available at <http://www.apache.org/dist/maven/>
- Distributed under the Apache License, version 2.0

On click on the link a .tar.gz file will be downloaded in the downloads folder. Download folder icon can be found on the upper right corner of the Safari browser window. As shown in the image below.



2. Unzip the files

Now go to the downloads folder and unzip the files there by double clicking on the downloaded file. Here the .tag.gz file that we have got is apache-maven-3.3.3-bin-2.tar. Here is what it will get when you will unzip/double click the folder



3. Set the Path variable

Like on a Windows system here also it has to set the path variable on a Mac machine too. Path variable will have to be updated to include the path on which you will copy all your Maven libraries. After unzipping the downloaded file just open the unzipped folder. You will find a few folders inside it. There will be a folder called bin, this folder will have a file called mvn. This is the file which is used to run Maven commands. File path should be mention in the Path variable of Mac system. Open the terminal window on the Mac machine. Once the terminal window up type in following command to see all the environment variable

\$ printenv

```
Virenders-MBP:~ viren$ printenv
TERM_PROGRAM=Apple_Terminal
SHELL=/bin/bash
TERM=xterm-256color
TMPDIR=/var/folders/xb/myqngxs51m13_g69nlvkyj100000gn/T/
Apple_PubSub_Socket_Render=/private/tmp/com.apple.launchd.j3iapHN2tN/Render
TERM_PROGRAM_VERSION=343.7
TERM_SESSION_ID=45A085D4-3DE7-4B02-BBC8-9B0E561024A4
USER=viren
SSH_AUTH_SOCK=/tmp/com.apple.launchd.BNqg04MQp1/Listeners
__CF_USER_TEXT_ENCODING=0x1F5:0x8:0x0
PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/git/bin
PWD=/Users/viren
XPC_FLAGS=0x0
XPC_SERVICE_NAME=@
SHLVL=1
HOME=/Users/viren
LOGNAME=viren
LC_CTYPE=UTF-8
SECURITYSESSIONID=186a7
_=~/usr/bin/printenv
Virenders-MBP:~ viren$
```

To see only the path variable values type in command

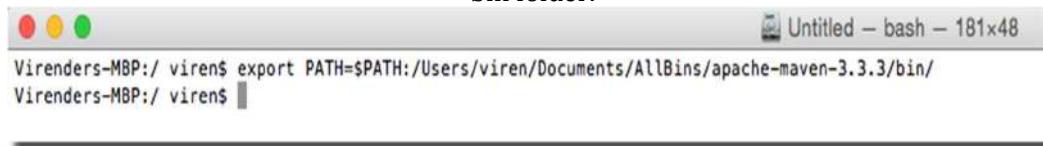
\$ echo \$PATH

```
Virenders-MBP:~ viren$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/git/bin
Virenders-MBP:~ viren$
```

Update the path variable to point to Maven's bin folder.

\$ export PATH=\$PATH:/Users/viren/Documents/AllBins/apache-maven-3.3.3/bin/

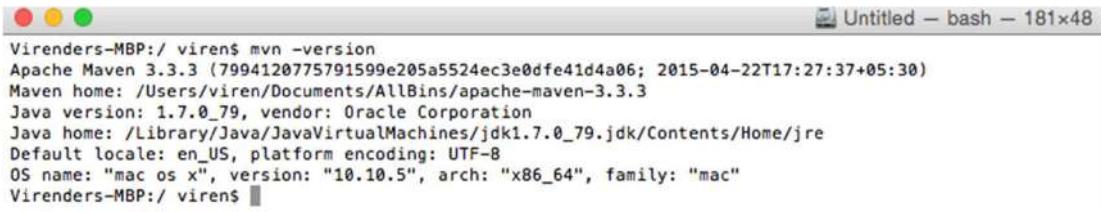
Here the PATH variable has updated with existing value of path + the new value of of Maven's bin folder.



```
Virenders-MBP:/ viren$ export PATH=$PATH:/Users/viren/Documents/AllBins/apache-maven-3.3.3/bin/
Virenders-MBP:/ viren$
```

4. Check the Installation

\$ mvn -version



```
Virenders-MBP:/ viren$ mvn -version
Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T17:27:37+05:30)
Maven home: /Users/viren/Documents/AllBins/apache-maven-3.3.3
Java version: 1.7.0_79, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "10.10.5", arch: "x86_64", family: "mac"
Virenders-MBP:/ viren$
```

How to Create a New Maven Project

There are two ways of creating Maven project. One by using command prompt and other from within the eclipse IDE.

The steps to create a New Maven Project from Command Prompt.

1. Go to Run and type 'cmd' to open Command Prompt.



2. Browse to the folder where the project is set up and then type the below command:

```
mvn archetype:generate -DgroupId=ToolsQA -DartifactId=DemoMavenProject -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\lsharm>cd..
C:\Users>cd..

C:\>mvn archetype:generate -DgroupId=ToolsQA -DartifactId=DemoMavenProject -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Note: 'cd..' is used for go back to previous folder and 'cd foldername' used for go in to the folder.

Here 'DartifactId' is the project name and 'DarchetypeArtifactId' is the type of Maven project.
There are different types of maven projects like web project, java project etc.

Once the Enter is press after typing the above command, it will start creating the Maven project.

```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\lsharm>cd..
C:\Users>cd..

C:\>mvn archetype:generate -DgroupId=ToolsQA -DartifactId=DemoMavenProject -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
[INFO] Scanning for projects...
[INFO]
[INFO] Building Maven Stub Project <No POM> 1
[INFO]
[INFO] --- maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom ---
[INFO] >>> maven-archetype-plugin:2.2:generate <default-cli> > generate-sources
@ standalone-pom >>>
[INFO] <<< maven-archetype-plugin:2.2:generate <default-cli> < generate-sources
@ standalone-pom <<<
[INFO] --- maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
[INFO]
[INFO] Using following parameters for creating project from Old <1.x> Archetype:
maven-archetype-quickstart:1.0
[INFO]
[INFO] Parameter: basedir, Value: C:\
[INFO] Parameter: package, Value: ToolsQA
[INFO] Parameter: groupId, Value: ToolsQA
[INFO] Parameter: artifactId, Value: DemoMavenProject
[INFO] Parameter: packageName, Value: ToolsQA
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old <1.x> Archetype in dir: C:\DemoMavenProject
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 4.108 s
[INFO] Finished at: 2014-10-01T23:30:17+05:30
[INFO] Final Memory: 14M/114M
[INFO]
```

Note: In case of build failure, please check the Maven version number in the pom.xml file. It should match the version of Maven installed on the machine. Look for this in the POM

'<http://maven.apache.org/POM/3.2.3>'. Correct the version from all the places it is mentioned in the POM.

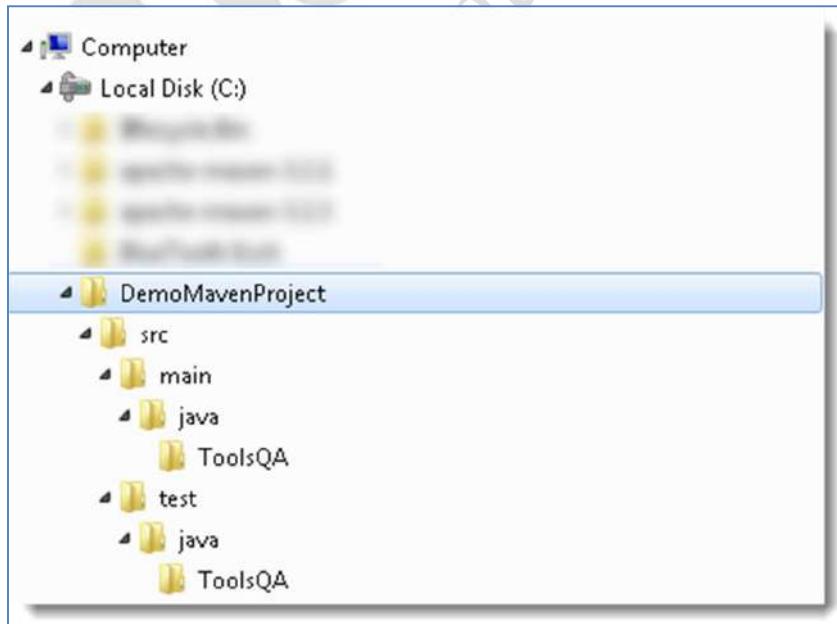
3. Go to the project location to see the newly created maven project. Now open the pom.xml file, which reside in project folder. By default the POM is generated like this:

```

<project xmlns="http://maven.apache.org/POM/3.2.3"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/3.2.3
  http://maven.apache.org/maven-v3_2_3.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>ToolsQA</groupId>
  <artifactId>DemoMavenProject</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>DemoMavenProject</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>

```

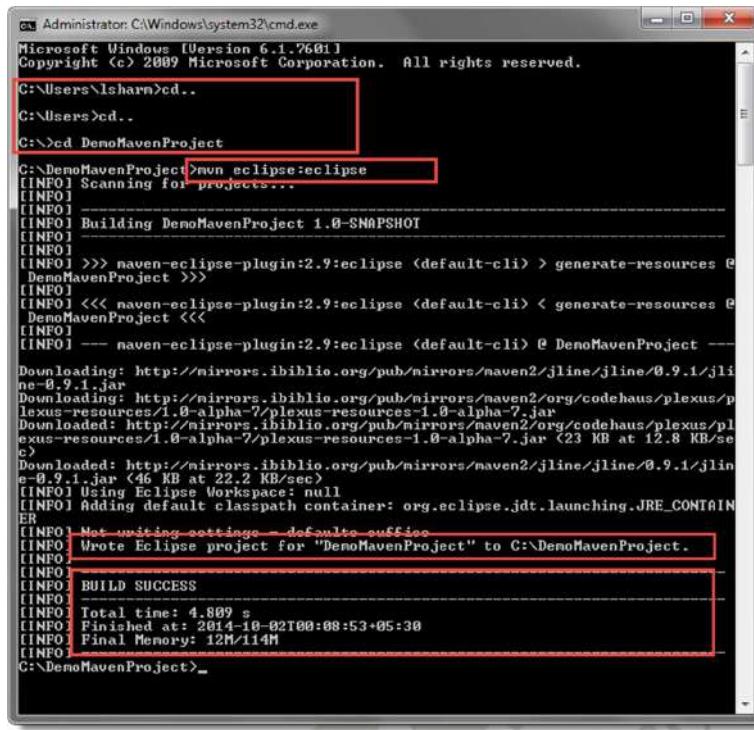
4. The default folder structure of the Maven project.



Note: Test cases resides under the src > test > java > ToolsQA will only be considered as a test by Maven, rest will be ignored if test cases are put in some other folder.

The steps to create a New Maven Project from Eclipse

1. Open command prompt and browse to the Maven project and type this 'mvn eclipse:eclipse'



```

Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

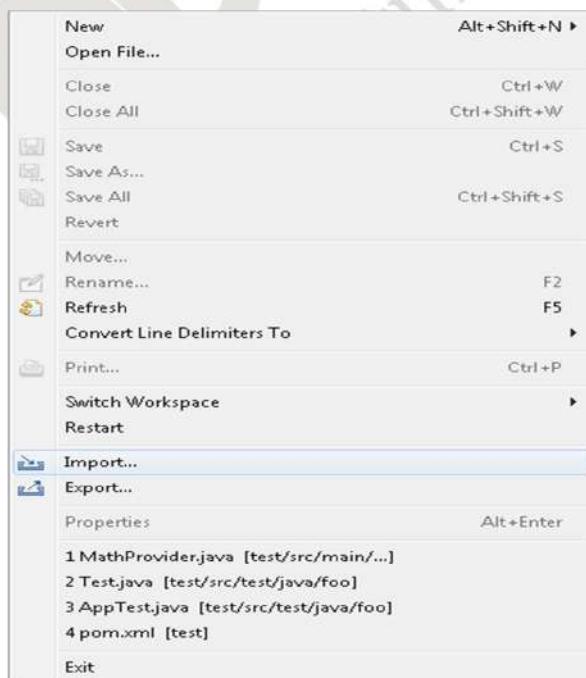
C:>cd..
C:>cd..
C:>cd DemoMavenProject
C:>mvn eclipse:eclipse
[INFO] Scanning for projects...
[INFO]
[INFO] Building DemoMavenProject 1.0-SNAPSHOT
[INFO]
[INFO] --- naven-eclipse-plugin:2.9:eclipse <default-cli> > generate-resources @ DemoMavenProject >>>
[INFO]
[INFO] <<< naven-eclipse-plugin:2.9:eclipse <default-cli> < generate-resources @ DemoMavenProject <<<
[INFO]
[INFO] --- naven-eclipse-plugin:2.9:eclipse <default-cli> @ DemoMavenProject ---
[INFO]
[INFO] Downloading: http://mirrors.ibiblio.org/pub/mirrors/maven2/jline/jline/0.9.1/jline-0.9.1.jar
[INFO] Downloading: http://mirrors.ibiblio.org/pub/mirrors/maven2/org/codehaus/plexus/plexus-resources/1.0-alpha-7/plexus-resources-1.0-alpha-7.jar
[INFO] Downloaded: http://mirrors.ibiblio.org/pub/mirrors/maven2/org/codehaus/plexus/plexus-resources/1.0-alpha-7/plexus-resources-1.0-alpha-7.jar (23 KB at 12.8 KB/sec)
[INFO] Downloaded: http://mirrors.ibiblio.org/pub/mirrors/maven2/jline/jline/0.9.1/jline-0.9.1.jar (46 KB at 22.2 KB/sec)
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.launching.JRE_CONTAINER
[INFO] Met existing settings - defaultсу suffice
[INFO] Wrote Eclipse project for "DemoMavenProject" to C:\DemoMavenProject.
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 4.809 s
[INFO] Finished at: 2014-10-02T00:08:53+05:30
[INFO] Final Memory: 12M/114M
[INFO]
C:>mvn eclipse:eclipse

```

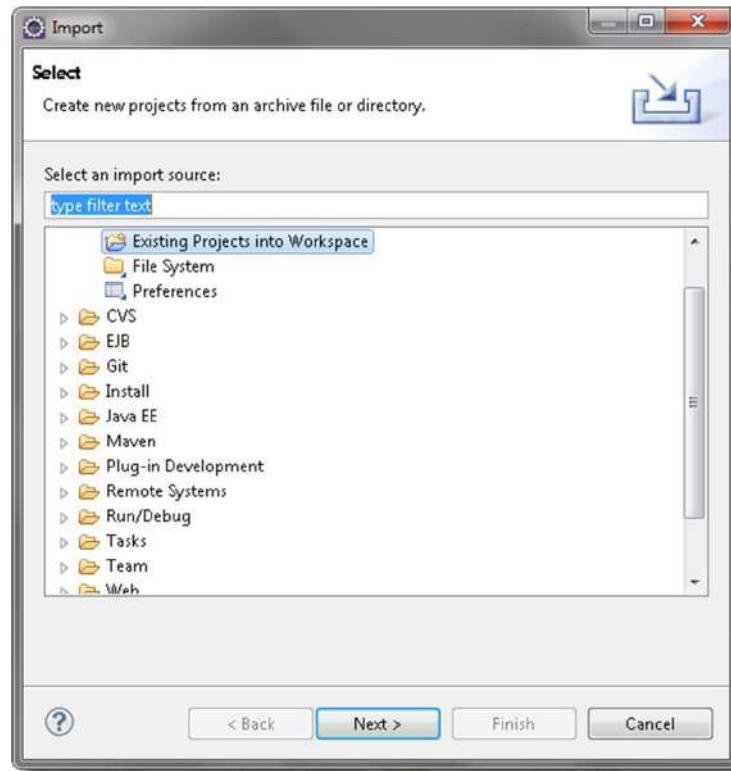
The project is now compatible for Eclipse IDE.

Import Maven Project in to Eclipse

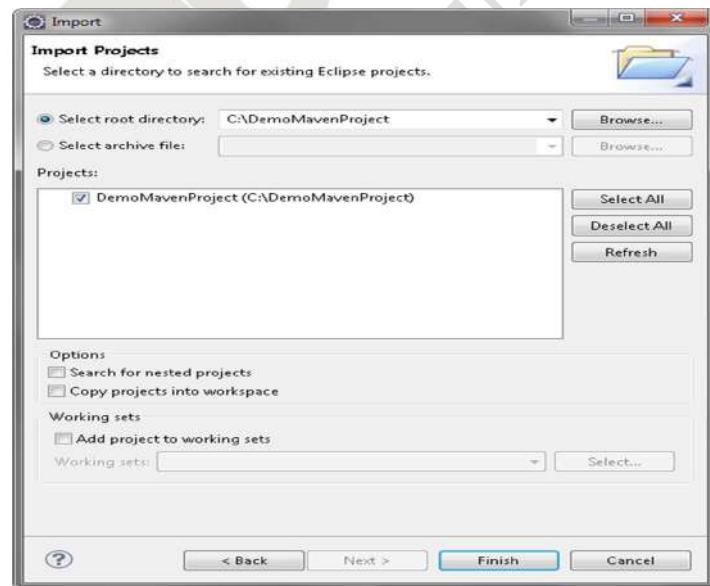
- Open Eclipse and go to File > Import.



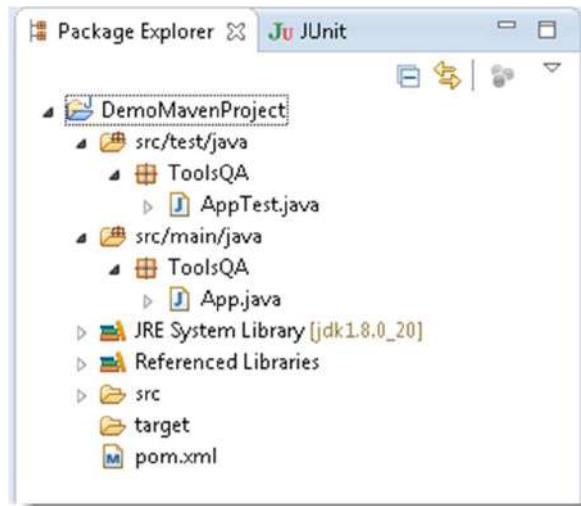
- Click on Existing Projects in to Workspace.



- Browse to Maven project folder and click on Finish.



- Project Explorer will now look like this:



- Modify the POM. The default pom.xml is too simple, often times it need to add the compiler plugin to tell Maven which JDK version to compile the project.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>2.3.2</version>
    <configuration>
        <source>1.6</source>
        <target>1.6</target>
    </configuration>
</plugin>
```

- Update the jUnit from 3.8.1 to latest 4.11

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
</dependency>
```

- Now the POM will look like this:

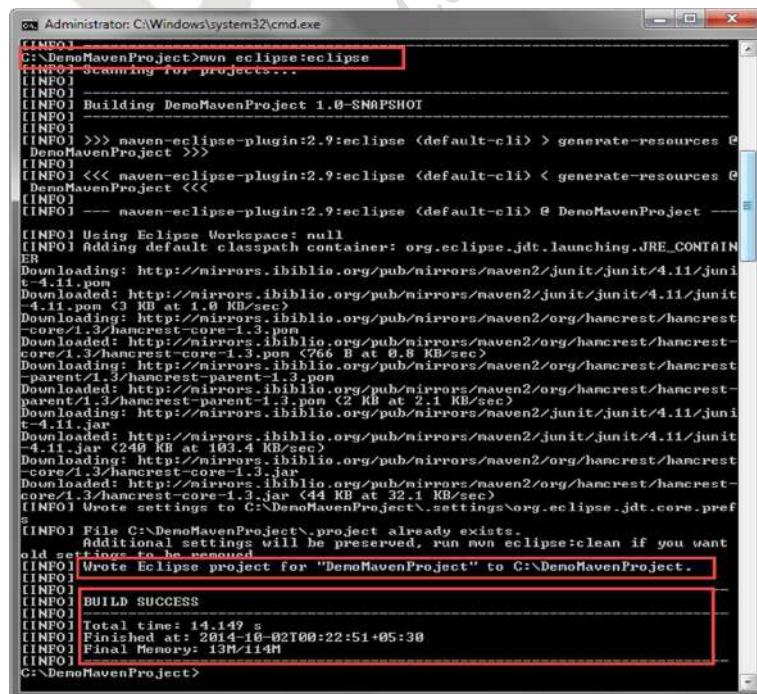
```
<project xmlns="http://maven.apache.org/POM/3.2.3"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/3.2.3
    http://maven.apache.org/maven-v3_2_3.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>ToolsQA</groupId>
    <artifactId>DemoMavenProject</artifactId>
    <packaging>jar</packaging>
```

```

<version>1.0-SNAPSHOT</version>
<name>DemoMavenProject</name>
<url>http://maven.apache.org</url>
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.2</version>
            <configuration>
                <source>1.6</source>
                <target>1.6</target>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>

```

- Once the POM has modified, do the Maven Project to work with Eclipse step again. Open command prompt and browse to the Maven project and type this ‘mvn eclipse:eclipse’



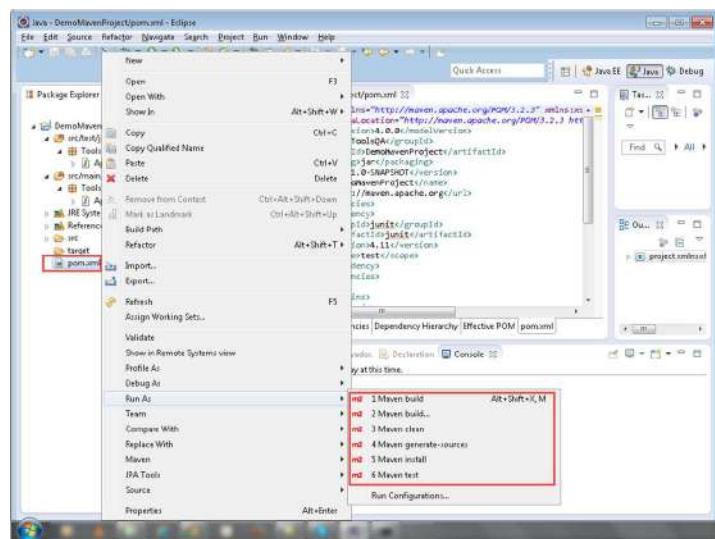
```

Administrator: C:\Windows\system32\cmd.exe
[INFO] Scanning for projects...
[INFO]
[INFO] Building DemoMavenProject 1.0-SNAPSHOT
[INFO]
[INFO] --- maven-eclipse-plugin:2.9:eclipse (default-cli) @ DemoMavenProject ---
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.launching.JRE_CONTAINER
[INFO] Downloading: http://mirrors.ibiblio.org/pub/mirrors/maven2/junit/junit/4.11/junit-4.11.pom
[INFO] Downloaded: http://mirrors.ibiblio.org/pub/mirrors/maven2/junit/junit/4.11/junit-4.11.pom (3 KB at 1.0 KB/sec)
[INFO] Downloading: http://mirrors.ibiblio.org/pub/mirrors/maven2/org/hamcrest/hamcrest-core/1.3/hamcrest-core-1.3.pom
[INFO] Downloaded: http://mirrors.ibiblio.org/pub/mirrors/maven2/org/hamcrest/hamcrest-core/1.3/hamcrest-core-1.3.pom (766 B at 0.8 KB/sec)
[INFO] Downloading: http://mirrors.ibiblio.org/pub/mirrors/maven2/org/hamcrest/hamcrest-parent/1.3/hamcrest-parent-1.3.pom
[INFO] Downloaded: http://mirrors.ibiblio.org/pub/mirrors/maven2/org/hamcrest/hamcrest-parent/1.3/hamcrest-parent-1.3.pom (2 KB at 2.1 KB/sec)
[INFO] Downloading: http://mirrors.ibiblio.org/pub/mirrors/maven2/junit/junit/4.11/junit-4.11.jar
[INFO] Downloaded: http://mirrors.ibiblio.org/pub/mirrors/maven2/junit/junit/4.11/junit-4.11.jar (249 KB at 103.4 KB/sec)
[INFO] Downloading: http://mirrors.ibiblio.org/pub/mirrors/maven2/org/hamcrest/hamcrest-core/1.3/hamcrest-core-1.3.pom
[INFO] Downloaded: http://mirrors.ibiblio.org/pub/mirrors/maven2/org/hamcrest/hamcrest-core/1.3/hamcrest-core-1.3.pom (44 KB at 32.1 KB/sec)
[INFO] Wrote settings to C:\DemoMavenProject\.settings\org.eclipse.jdt.core.preferences
[INFO] File C:\DemoMavenProject\.project already exists.
[INFO] Additional settings will be preserved, run mvn eclipse:clean if you want old settings to be removed.
[INFO] Wrote Eclipse project for "DemoMavenProject" to C:\DemoMavenProject.
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 14.147 s
[INFO] Finished at: 2014-10-02T00:22:51+05:30
[INFO] Final Memory: 13M/114M
[INFO]
C:\DemoMavenProject>

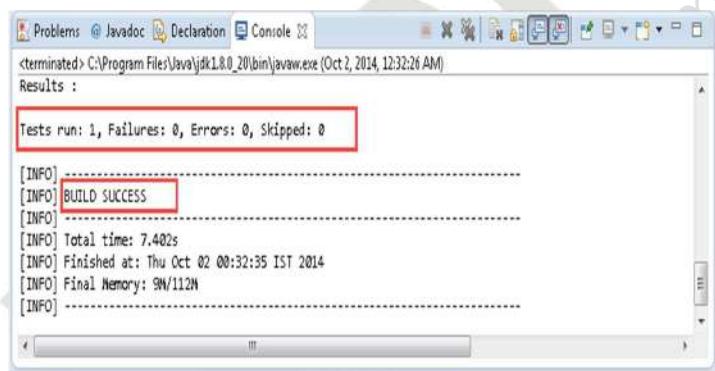
```

Run the first Maven Test

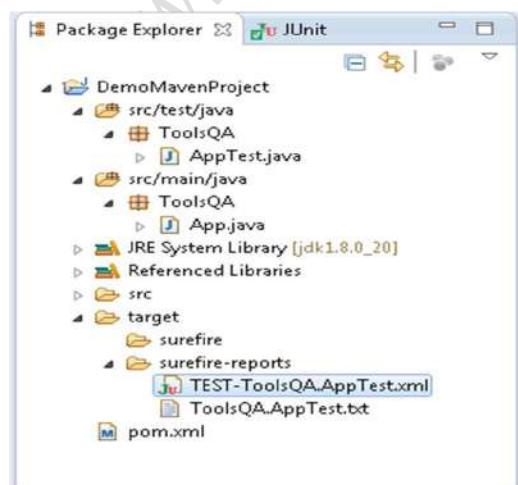
- Right click on the pom.xml and go to Run As > Maven test.



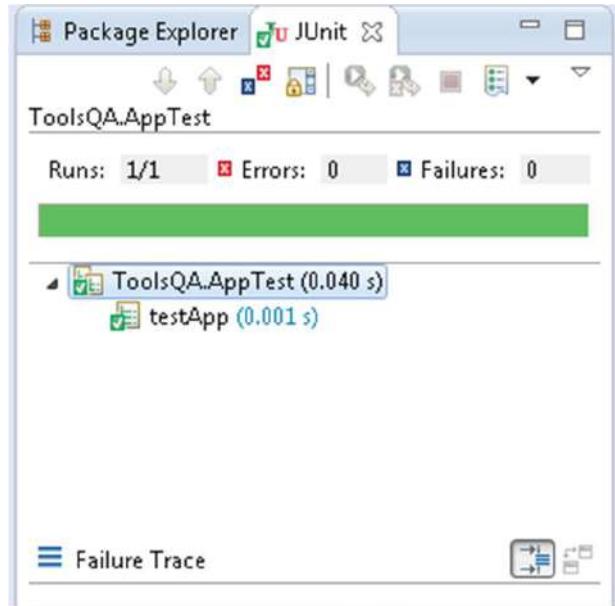
- In the console window of Eclipse, see the information like this:



- Go to 'SureFire Reports' folder and open the xml file.



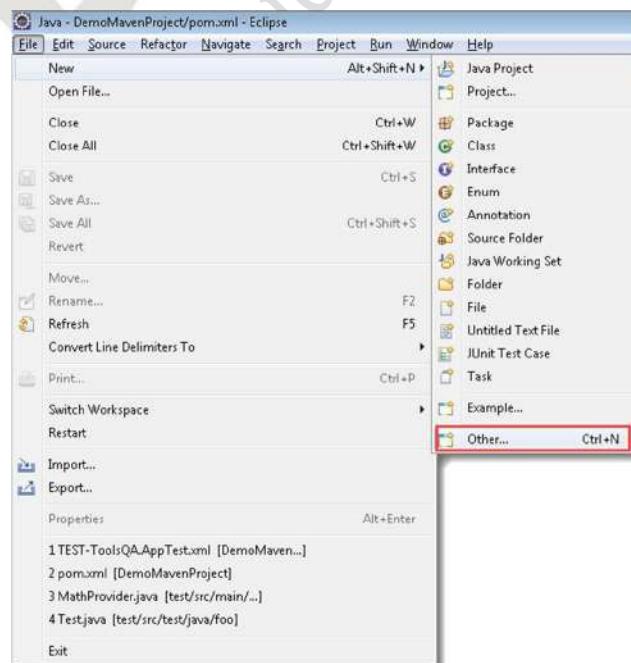
4. It will display the result of the JUnit test.



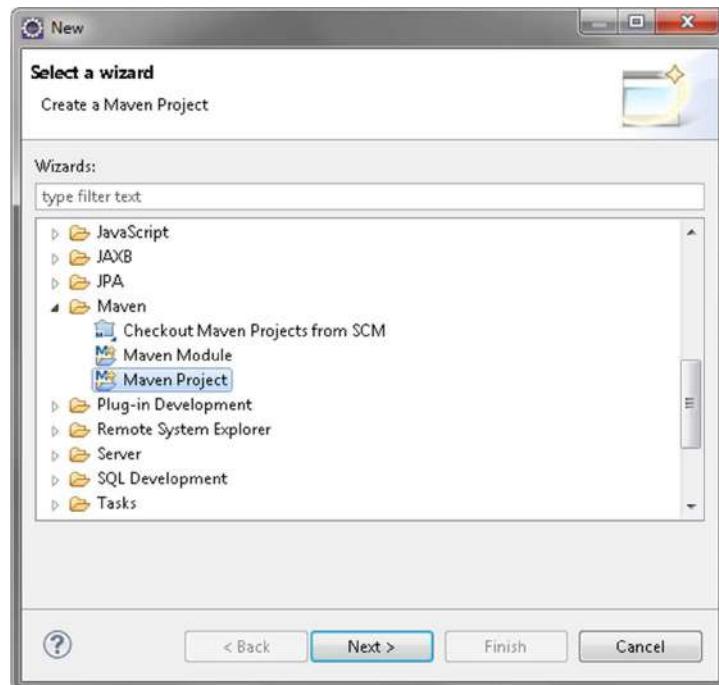
How to Create a New Maven Project in Eclipse

This is the second method to create a Maven project. But in this rather than creating a project outside eclipse and then import in to eclipse, it will directly create a maven project in to eclipse.

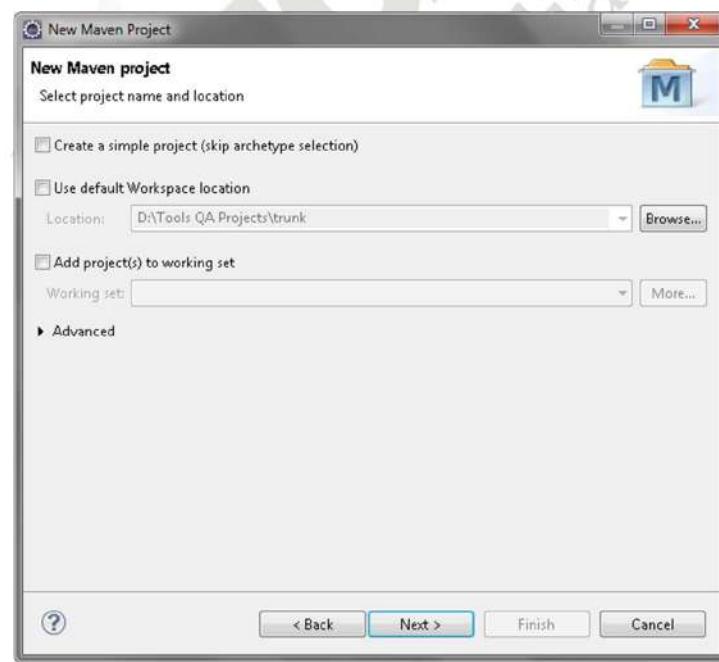
1. Open your eclipse and Go to File > New > Others.



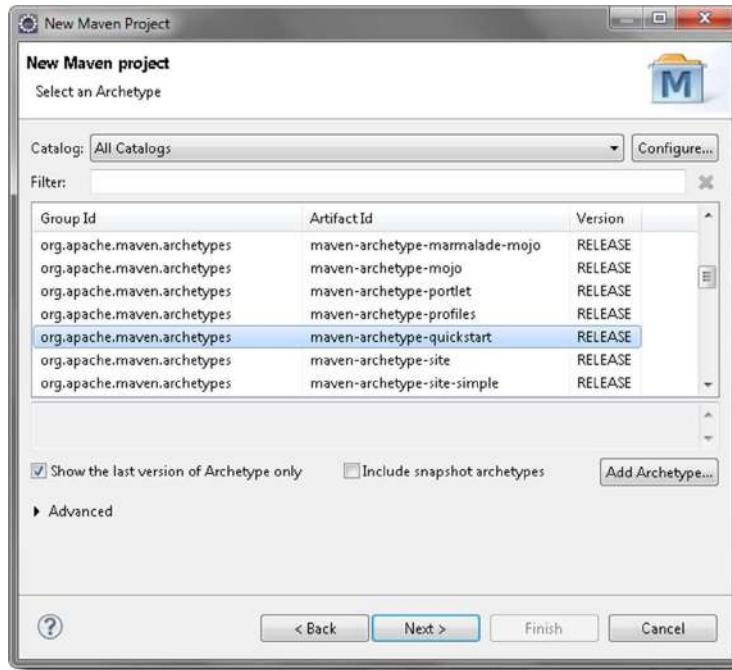
2. Select Maven Project and click on Next.



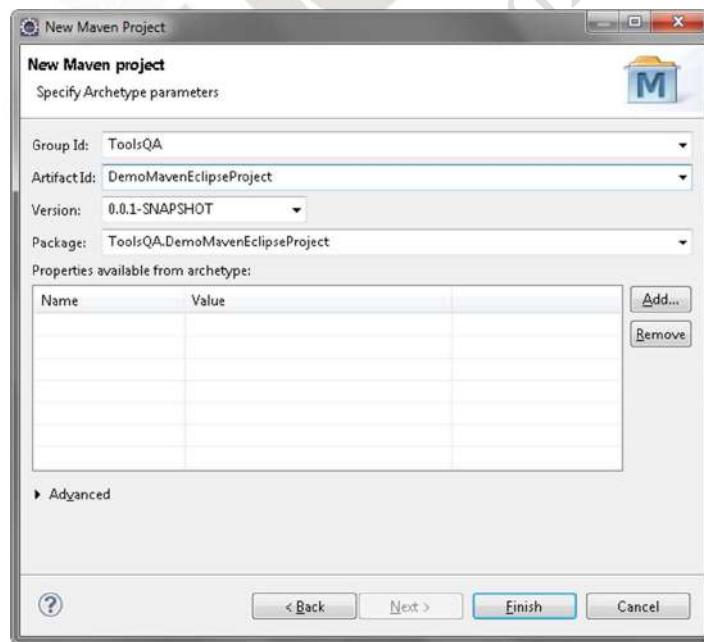
3. Un-check the 'Use default Workspace location' and with the help of Browse button choose the workspace where it would like to set up the Maven project.



4. Select the archetype, for now just select the 'maven-archetype-quickstart' and click on Next.



5. Specify the Group Id & Artifact Id and click on Finish.



Note: Here the ‘artifactId’ is the project name.

6. Go to the project location to see the newly created maven project. Now open the pom.xml file, which reside in project folder. By default the POM is generated like this:

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>ToolsQA</groupId>
  <artifactId>DemoMavenEclipseProject</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

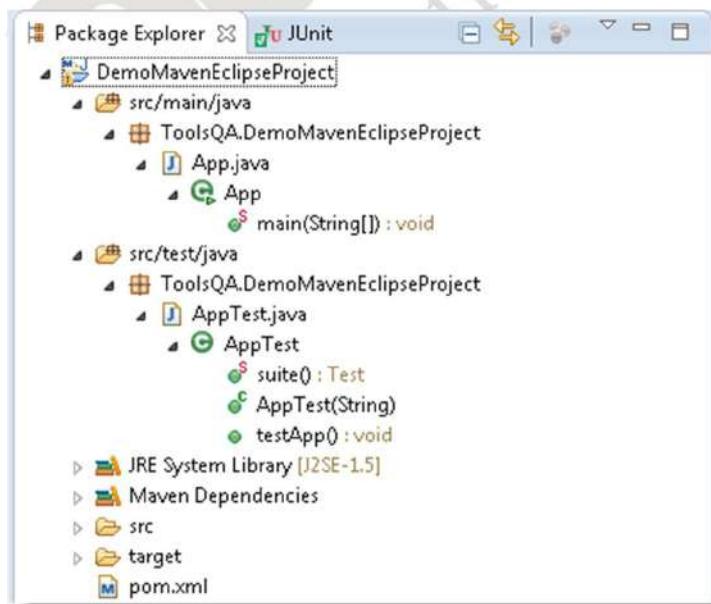
  <name>DemoMavenEclipseProject</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>

```

7. Look at the default folder structure of the Maven project.



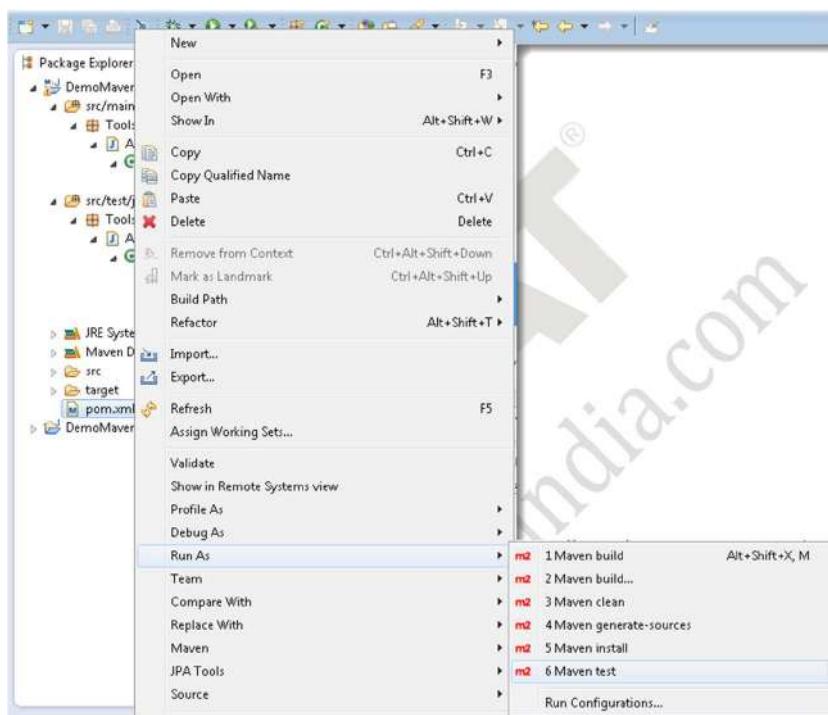
Note: Test cases resides under the src > test > java > PackageName will only be considered as a test by Maven, rest will be ignored if you put the test cases in some other folder.

8. Modify xml with latest Junit and save the xml.

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
</dependency>
```

Run the first Maven Test

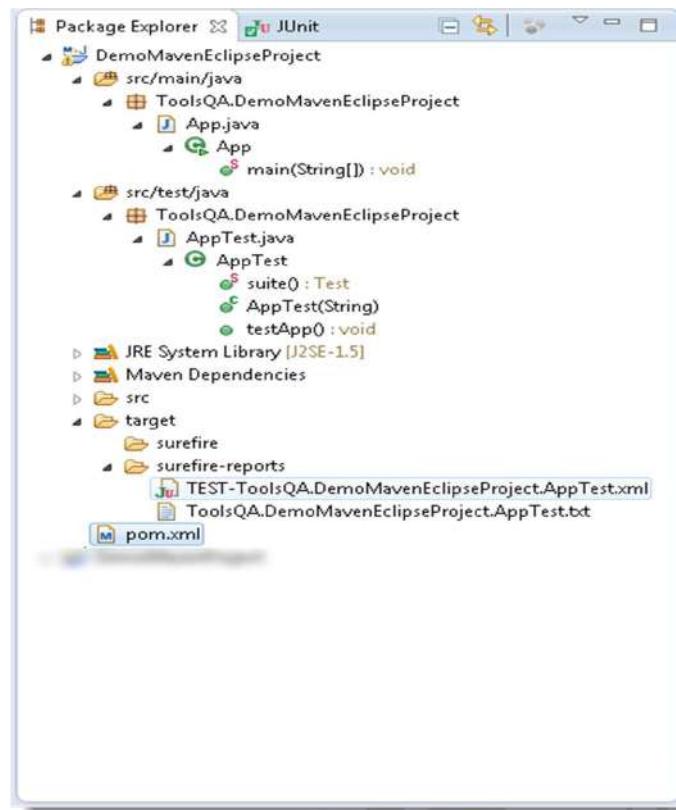
- Right click on the pom.xml and go to Run As > Maven test.



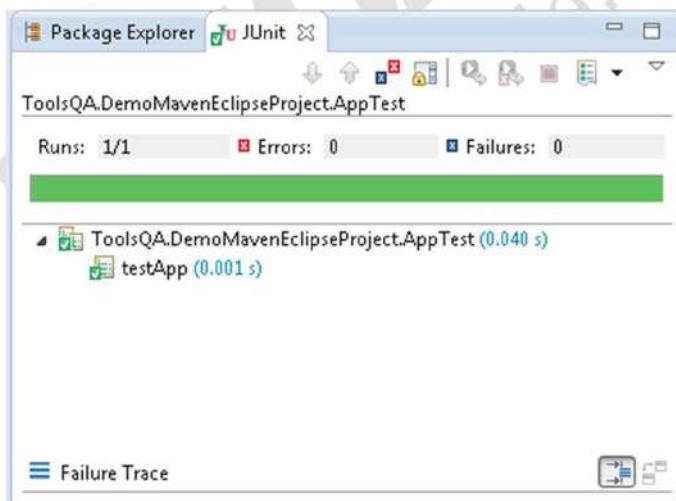
- In the console window of Eclipse, the information will be seen like this:



- Go to 'SureFire Reports' folder and open the xml file.



4. It will display the result of the JUnit test.



Configure Selenium Continuous Integration with Maven

Configure Selenium Continuous Integration with Maven project is not a big task. But making that project work with Maven is little tricky and that requires few configurations around the project. Once it done with that only then the test will be recognized by the Maven build system, else Maven will always ignore the test and perform the build. The complete list of steps are as followed:

1. Convert Selenium project in to Maven Project
2. Remove Errors from the Project
3. Remove Associated Libraries from the Project Build Path
4. Add Dependencies to the Maven Repository
5. Add JDK to the Project Build Path
6. Add Maven Dependencies to the Project Build Path
7. Resolve Apache POI errors
8. Create Maven Test folder structure
9. Create a new TestNg Test
10. Run the Test with Maven Build

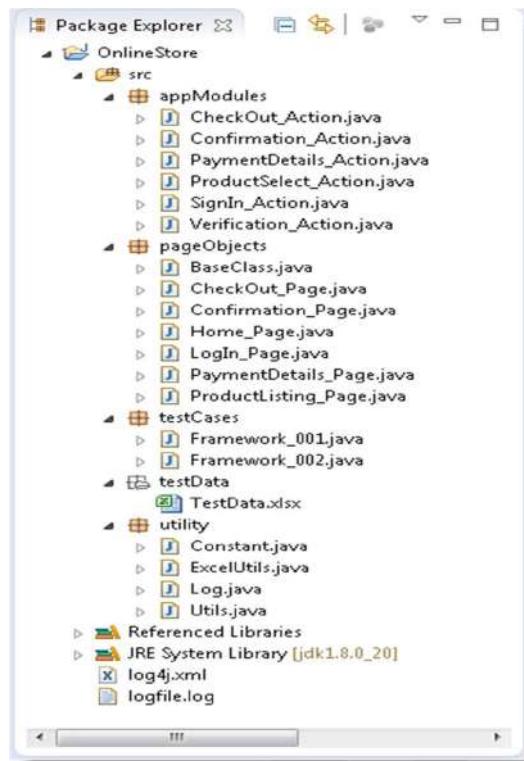
Prerequisite for Converting a Selenium Project in to Maven Project

- Java is Installed
- Eclipse IDE is Installed
- Maven is Installed
- Dummy Selenium Project is Imported

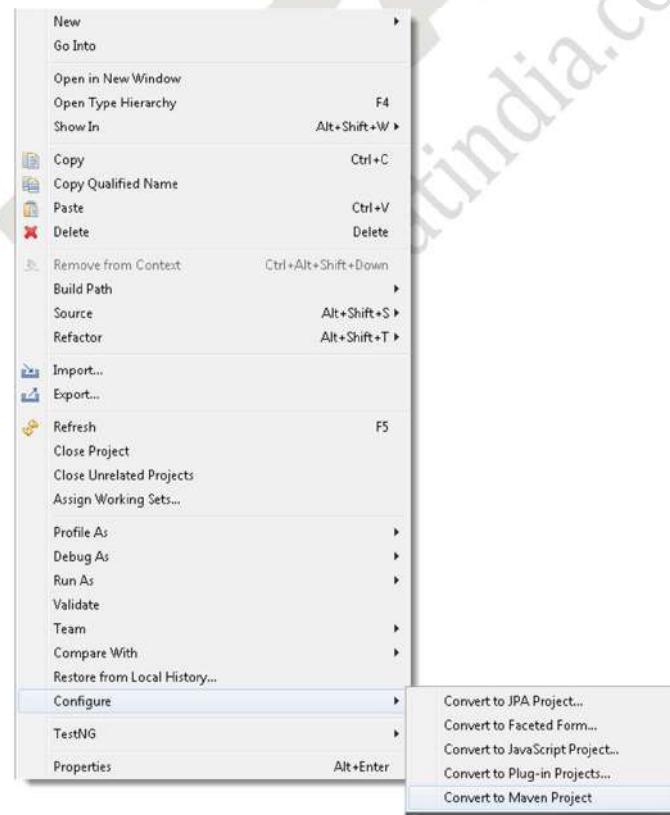
1. Convert a Selenium Project in to Maven Project

Before moving on to this step, make sure that the dummy Selenium project has been imported. Importing a project in Eclipses simple, it just need to go on File menu > Import.... Then on the new window select General > Existing Projects in to Workspace and press Next. Browse the downloaded Selenium project.

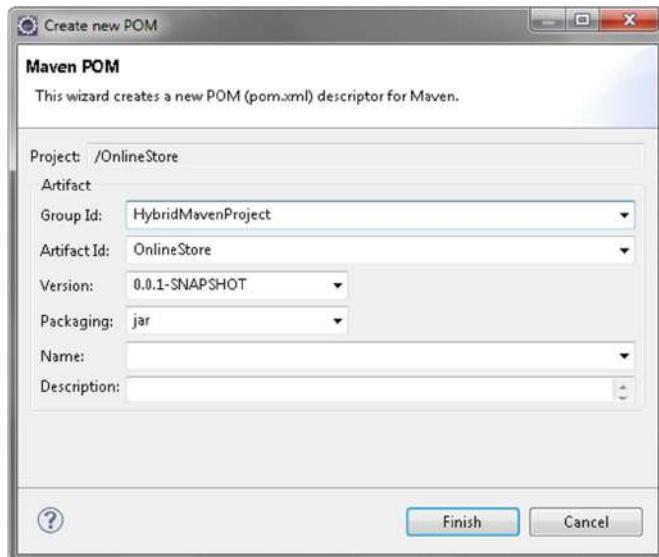
2. Once it is done with the import of the dummy project, the project explorer window will look like this.



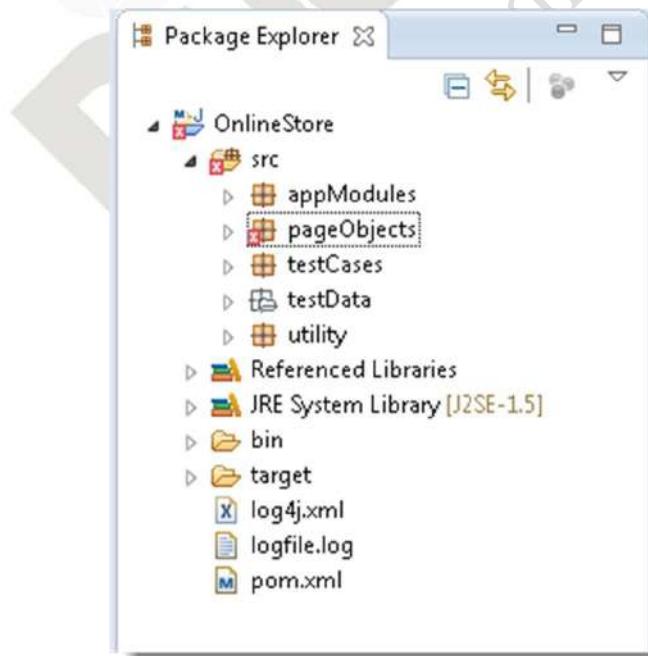
3. Right click on the Online Store folder and select Configure > Convert to Maven Project.



4. It then ask to fill the Group ID and Artifact ID. Fill appropriate name and click on Finish button. Do not change the rest of the default settings.



5. Eclipse will take few seconds to convert the project and once it is completed with the process, the project explorer window will look like this.



6. Remove Errors from the Project

It will be notice that the project now have quite a good number of errors, These errors are on throws statement because the try catch block is used in the methods but Exception is not

declared in the method declaration. To resolve the errors add throws declaration in all the methods where ever the try catch statement is used.

7. Remove Associated Libraries from the Project Build Path

During the build, Maven only consider its own libs/repository and it will not consider the libraries available in the project build path. So the next step is to add all those libraries and dependencies which is required in the project. To figure out what all is required, just remove all the libraries from the build path and the project will end up with many more errors. Then it can be browse to every single error and one by one add

8. Add Dependencies to the Maven Repository

Dependencies are the libararies, which are required by the project. For example Log4j jars, Apache Poi jars, Selenium Jars.

- Maven Local Repository

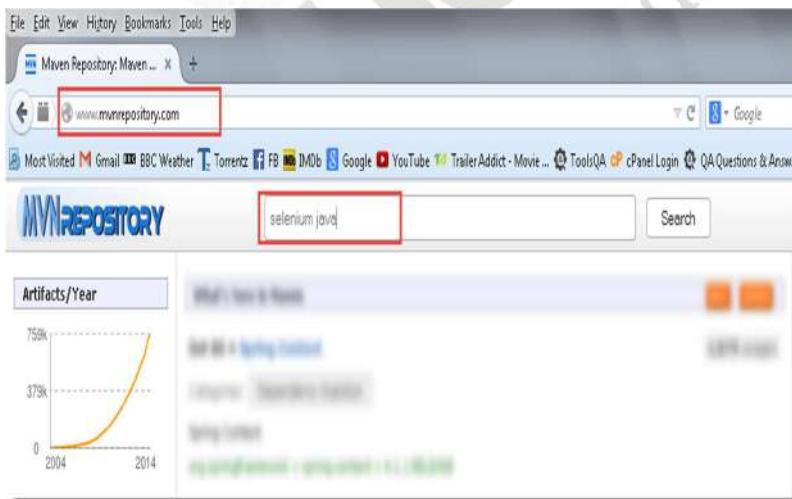
This is the place where Maven stores all the project jars files or libraries or dependencies. By default the folder name is '.m2' and by default the location in windows 7 is 'Libraries\Documents\.m2' or 'C:\Users\yourusername\.m2'.

- Maven Central Repository

Maven central repository is the default location 'http://mvnrepository.com/' for Maven to download all the project dependency libraries. For any library required in the project, Maven first look in to the .m2 folderof Local Repository, if it does not find the required libarary then it looks in Central Repository and download the libabry in to local repository.

- Maven POM

Go to <http://www.mvnrepository.com/> and search for Selenium Java.



Click on the Selenium Java from the search results.



Found 7405 results

1. Selenium Java

Categories: Web Testing

Selenium Java

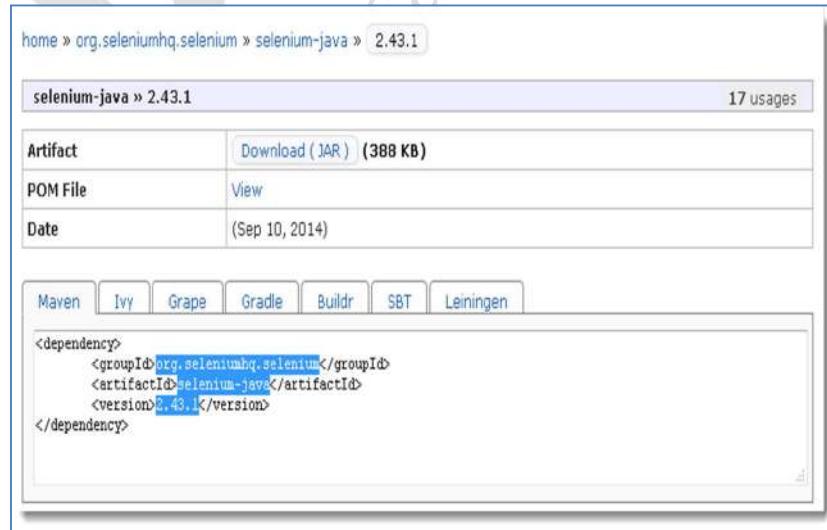
org.seleniumhq.selenium » selenium-java

252 usages

Click on any version of the Selenium. it is allowed to choose any version here if in case it is specific to any project.

	Version	Usages	Type	Date
2.43.x	2.43.1	17	release	(Sep, 2014)
	2.43.0	6	release	(Sep, 2014)
2.42.x	2.42.2	35	release	(Jun, 2014)
	2.42.1	5	release	(May, 2014)
2.41.x	2.42.0	13	release	(May, 2014)
	2.41.0	46	release	(Mar, 2014)
2.40.x	2.40.0	27	release	(Feb, 2014)

Take a look at the highlighted area on the below image. This highlighted details needs to be entered in the Maven pom.xml.



home » org.seleniumhq.selenium » selenium-java » 2.43.1

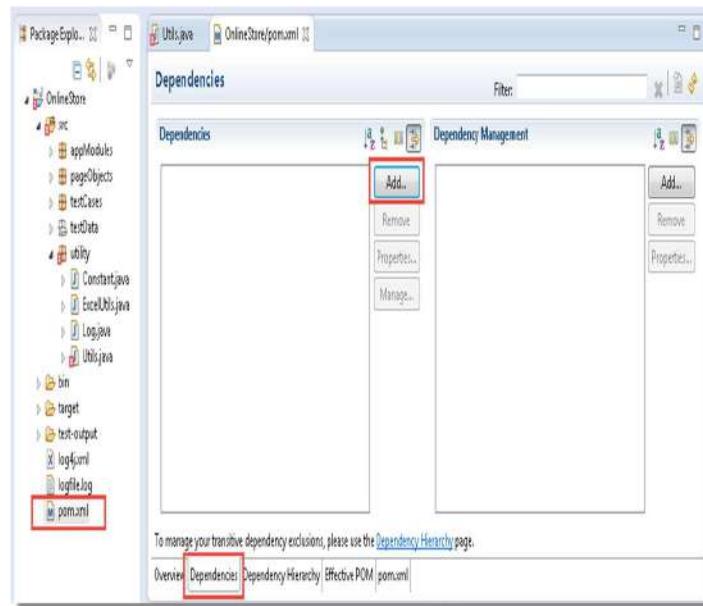
selenium-java » 2.43.1 17 usages

Artifact	Download (JAR) (388 KB)
POM File	View
Date	(Sep 10, 2014)

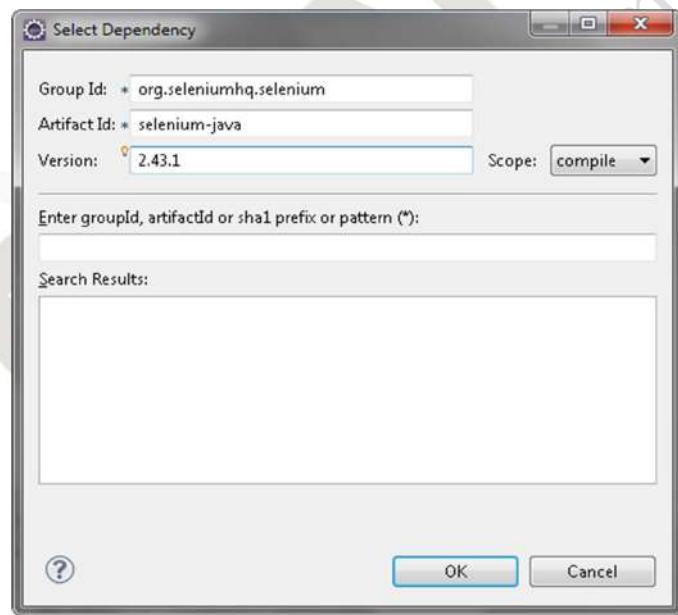
Maven Ivy Graal Gradle Buildr SBT Leiningen

```
<dependency>
<groupId>org.seleniumhq.selenium</groupId>
<artifactId>selenium-java</artifactId>
<version>2.43.1</version>
</dependency>
```

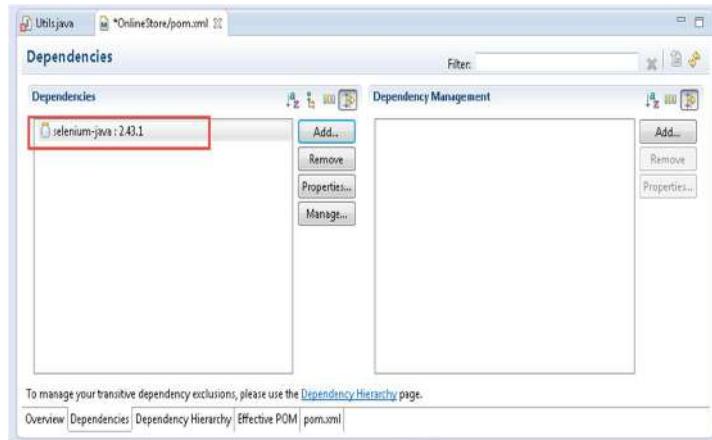
Double click on the pom.xml file which is at the bottom of the project explorer window.
Select the Dependencies tab and then click on Add button.



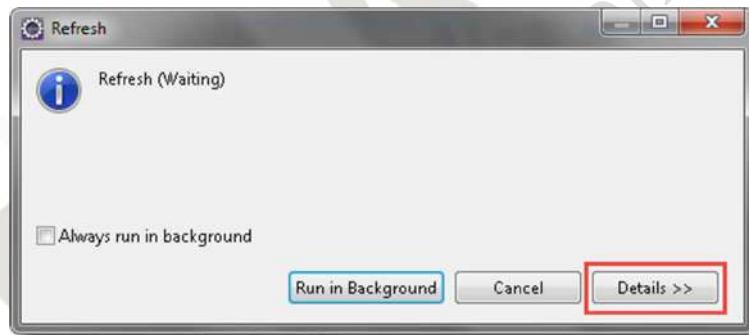
A pop window will display. Enter the highlighted detail here which we copied from maven repository website, as per the below screen shot. Make sure that Group ID, Artifact ID & Version are exactly the same.



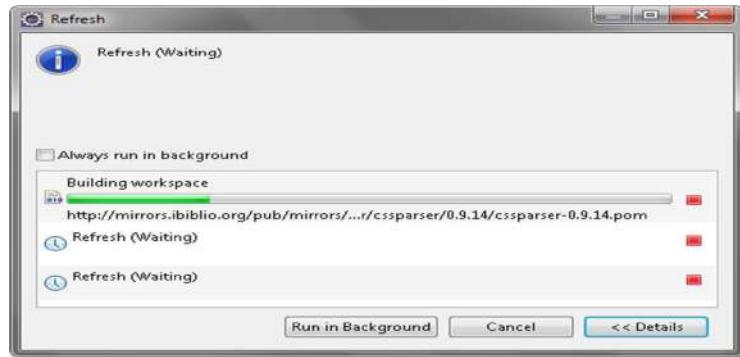
Now the Dependencies tab will look like this with Selenium Java added on it.



Now once the changes has been saved, right click at the empty space of the project explorer and select Refresh.



If it is choose 'Run in Background', it will download the Selenium Java jar files in to the Maven dependencies and the downloading process will take place at the background. If you choose 'Details>>', it will display the downloading status bar and the process.



The same way it is required to add all the dependencies which are required in the project. Now add the Log4j dependency in the POM file.



log4j > 1.2.17 1,372 usages

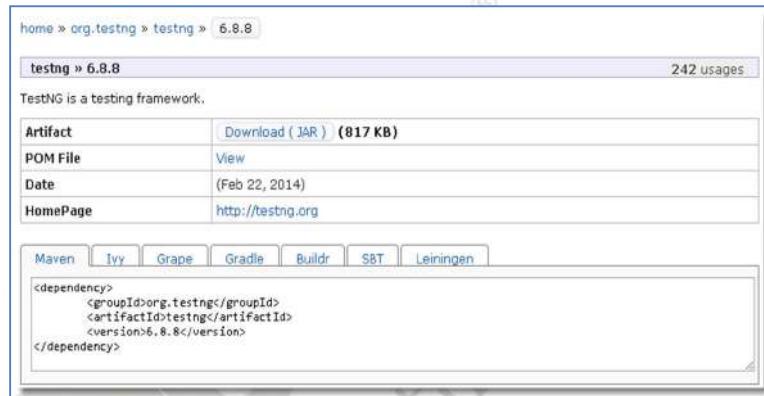
Apache Log4j 1.2

Artifact	Download (BUNDLE) (479 KB)
POM File	View
Date	(May 06, 2012)
HomePage	http://logging.apache.org/log4j/1.2/
Organization	Apache Software Foundation
Issue Tracker	https://issues.apache.org/bugzilla/describecomponents.cgi?product=Log4j

Maven Ivy Grape Gradle Buildr SBT Leiningen

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

Add TestNG dependency to the POM file.



home > org.testng > testng > 6.8.8

testng > 6.8.8 242 usages

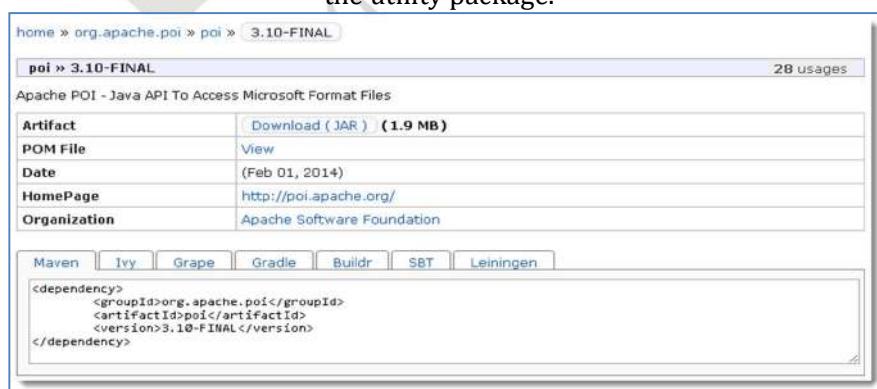
TestNG is a testing framework.

Artifact	Download (JAR) (817 KB)
POM File	View
Date	(Feb 22, 2014)
HomePage	http://testng.org

Maven Ivy Grape Gradle Buildr SBT Leiningen

```
<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>6.8.8</version>
</dependency>
```

Go to ExcelUtils class under the utility package.



home > org.apache.poi > poi > 3.10-FINAL

poi > 3.10-FINAL 28 usages

Apache POI - Java API To Access Microsoft Format Files

Artifact	Download (JAR) (1.9 MB)
POM File	View
Date	(Feb 01, 2014)
HomePage	http://poi.apache.org/
Organization	Apache Software Foundation

Maven Ivy Grape Gradle Buildr SBT Leiningen

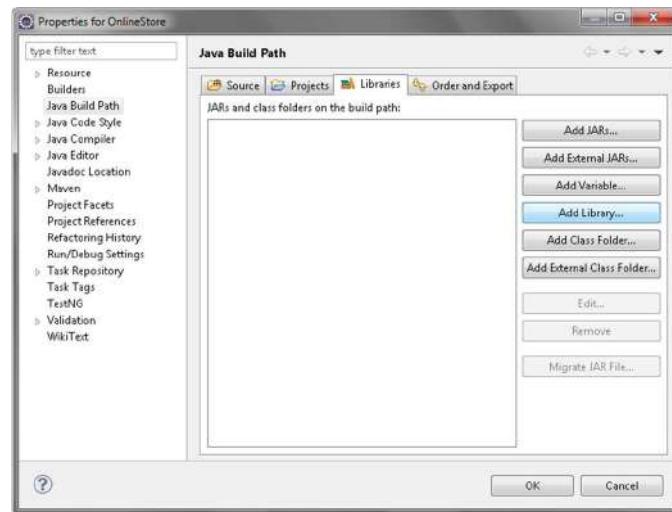
```
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>3.10-FINAL</version>
</dependency>
```

The SurefirePlugin is used during the test phase of the build life cycle to execute the unit tests of an application. It generates reports in 2 different file formats like plain text file, xml files and html files as well. Even if TestNG or Junit framework is used for reporting, this plugin is must to use, as it helps Maven to identify tests.

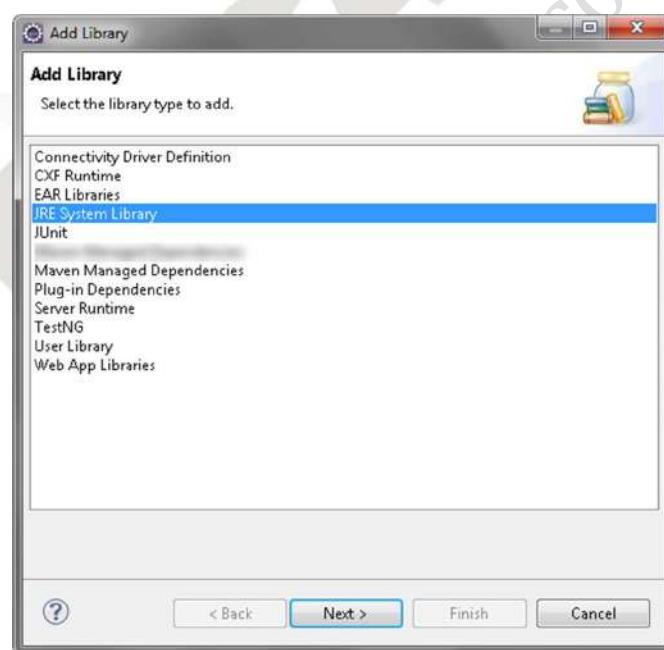
5. Add JDK to the Project Build Path

Right Click on the project explorer and select Build Path > Configure Build Path.

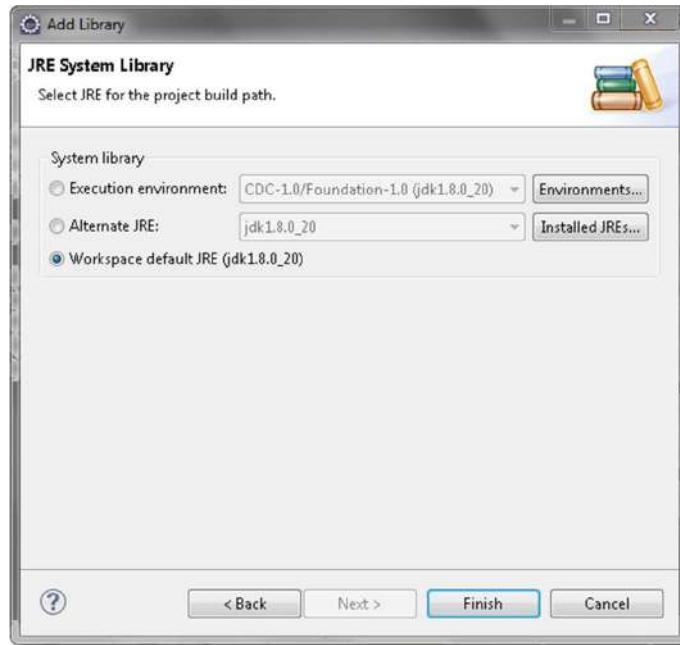
Select the Libraries tab and click on Add Library



Select JRE System Library and click on Next button.



Select 'Workspace default JRE(jdk XXXXX)' radio button and click on Finish button.



6. Add Maven Dependencies to the Project Build Path

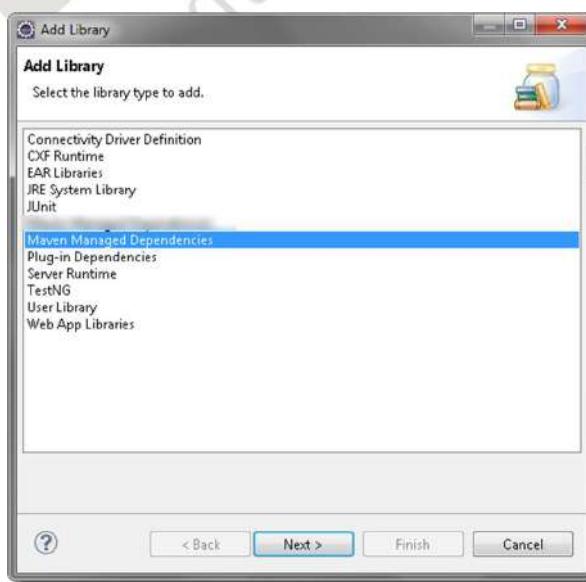
When a new Maven project has created, dependencies are added by default, so Maven Dependencies can be seen under the project explorer window. But when the Selenium project is convert to the Maven Project, maven dependencies needs to be added on to the project build path. So far it has just added the dependencies in the maven repository, now it need to link that repository to the project.

Right Click on the project explorer and select Build Path > Configure Build Path.

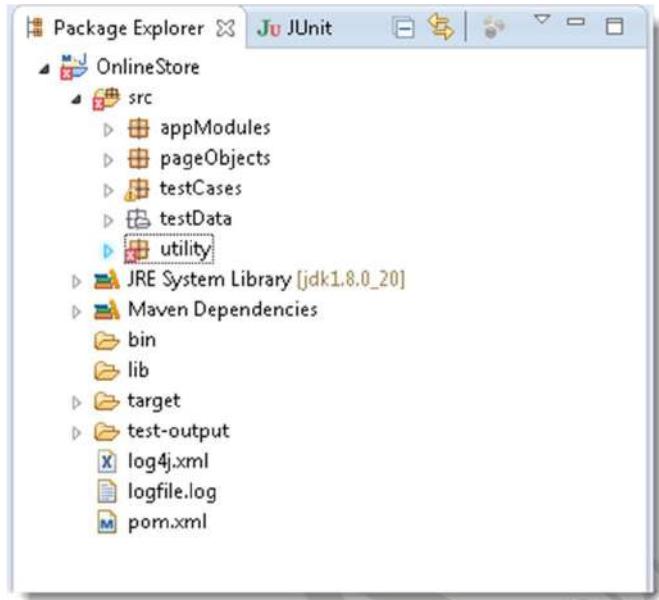
Select the Libraries tab.

Click on Add Library...

Select Maven Managed Dependencies and click on Next.



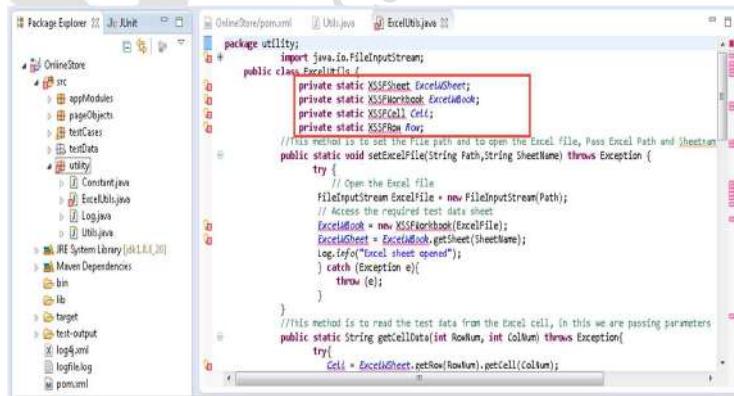
Take a look over the project explorer window, Maven Dependencies can be seen.



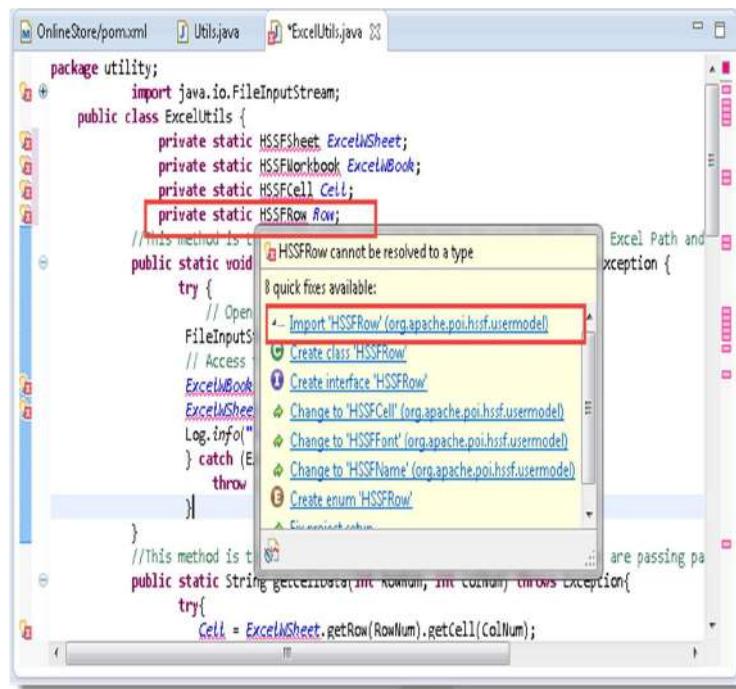
7. Resolve Apache POI Errors

There is a little change in the current version of Apache POI which is that they have changed the XSSF in to HSSF. If the ExcelUtils class of utility package will be seen, then many errors will be there around it.

Just change all the XSSF entries in to HSSF for the sheet, workbook, cell and row.



Now by bringing the cursor over the each of the keyword, import the relevant packages.



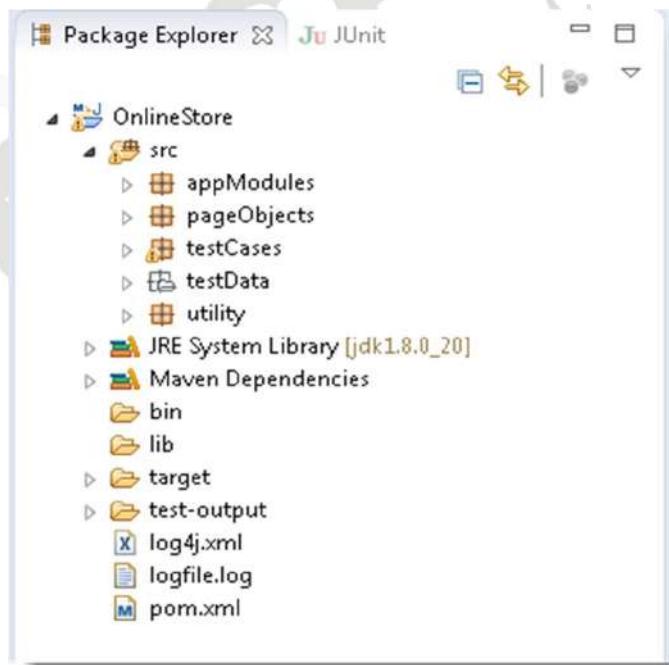
The screenshot shows the Eclipse IDE interface with the Java editor open. A code completion dropdown is displayed over the line of code: `private static HSSFR Row;`. The dropdown title is "HSSFR cannot be resolved to a type". It contains several quick fix options, with the first one, "Import 'HSSFR' (org.apache.poi.hssf.usermodel)", highlighted with a red box.

```

package utility;
import java.io.FileInputStream;
public class ExcelUtils {
    private static HSSFSheet ExcelSheet;
    private static HSSFWorkbook ExcelBook;
    private static HSSFCell Cell;
    private static HSSFR Row;
    //This method is to open excel file
    public static void main(String[] args) throws Exception {
        try {
            // Open FileInputS
            // Access
            FileInputStream FileInputStream = new FileInputStream("C:\\Users\\Administrator\\Desktop\\Book1.xls");
            ExcelBook = new HSSFWorkbook(FileInputStream);
            ExcelSheet = ExcelBook.getSheetAt(0);
            Log.info("File opened successfully");
        } catch (Exception e) {
            throw e;
        }
    }
    //This method is to get cell value
    public static String getCellData(int rowNum, int column) throws Exception {
        try{
            Cell = ExcelSheet.getRow(rowNum).getCell(column);
        }
    }
}

```

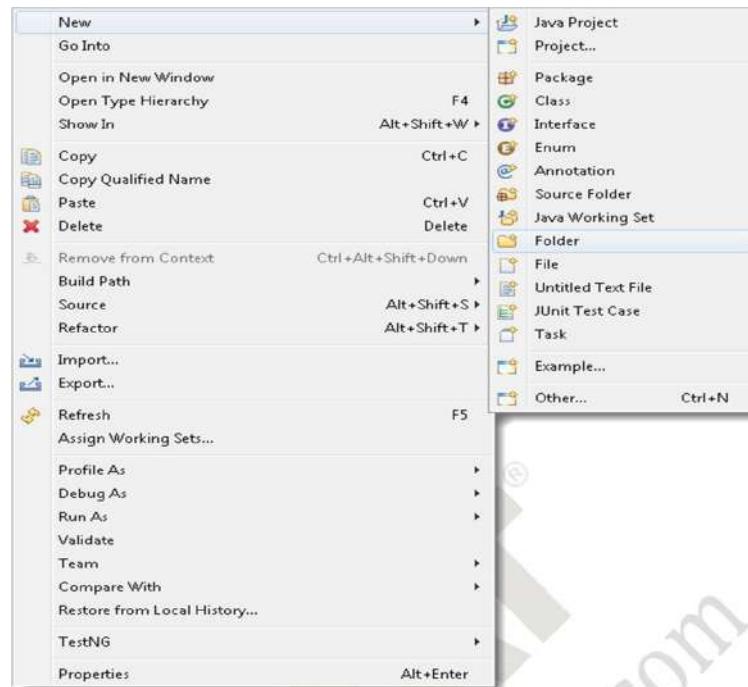
Project explorer window will now look like this, as now the Selenium project is completely error free.



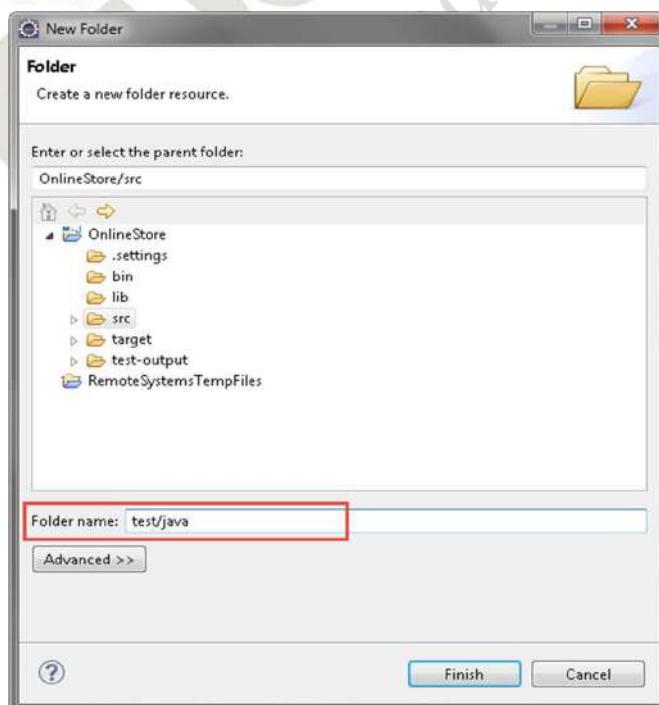
8. Create the Maven Test folder structure

Maven has its own project folder structure and it is advisable to follow its default folder structure. For example all the test of the project should be in src > Test > Java folder. Lets just create the same for this project now.

Right click on the src folder and select New > Folder.



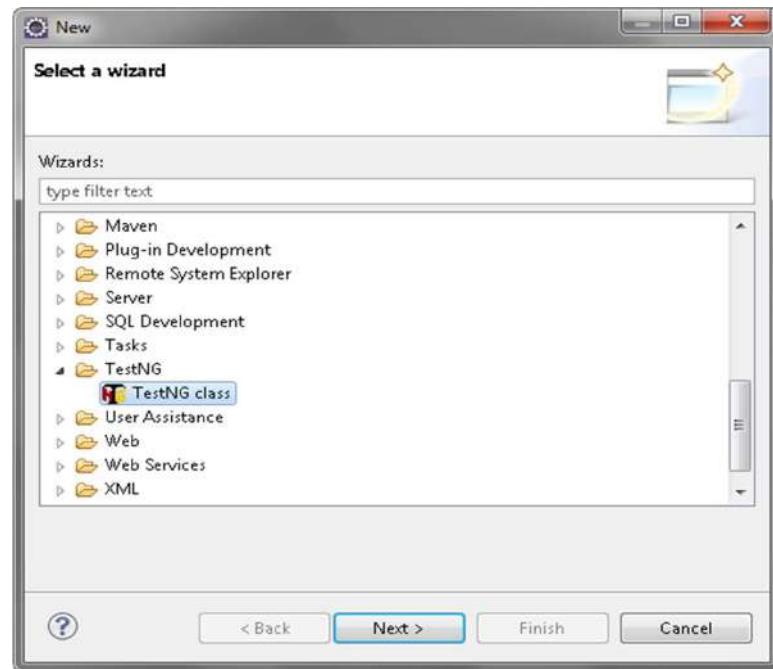
Give the folder name as 'test/java' and click on Finish button.



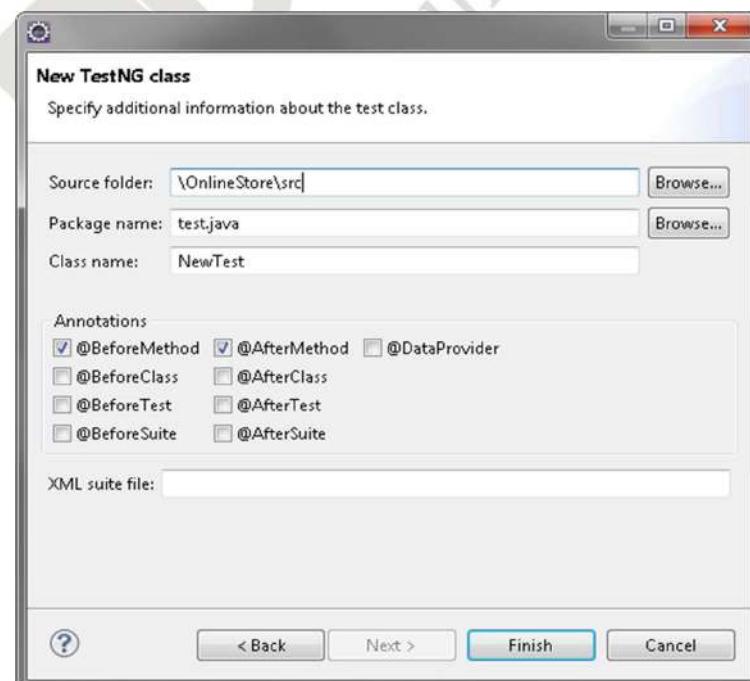
9. Create a TestNG test for the Selenium Maven Project

Either a new test can be created or cut/paste the existing tests to this src/test/java folder. The prerequisite for this step is that the TestNG should be installed in the Eclipse.

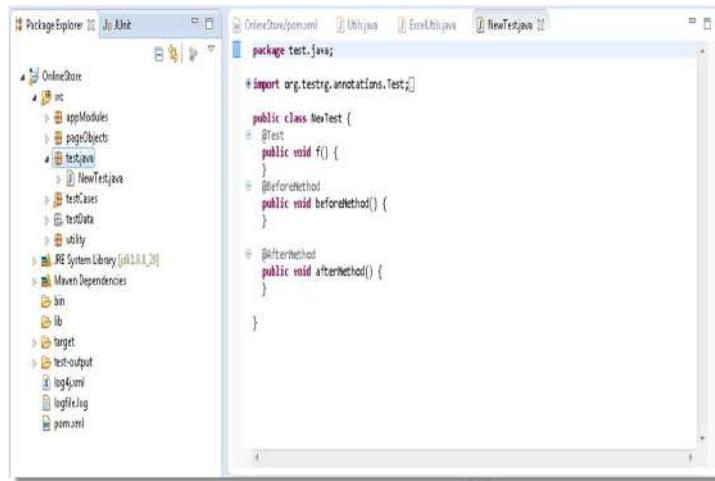
Right click on the test/java folder and Select New > Other. A new window will display and then select TestNG Class from it.



Enter Class name of any choice, but for now take it as NewTest. Now select any of the Annotations as well. Click on Finish button.



Project explorer and the Eclipse window will look like this now.



10. Run the Selenium Test with Maven Build

Right click on the pom.xml and Select Run As > Maven Test.



Once the run is complete, the result will be displayed in the console window of the Eclipse IDE.

The screenshot shows an IDE interface with several tabs at the top: Problems, Javadoc, Declaration, Console, and TestNG. The TestNG tab is active, displaying the output of a test run. The output window shows the following text:

```
-----  
TESTS  
-----  
Running test.java.NewTest  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.429 sec  
Results:  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0  
-----  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 25.721s  
[INFO] Finished at: Fri Oct 10 03:42:14 IST 2014  
[INFO] Final Memory: 18N/83M  
[INFO] -----
```

DUCAT[®]
www.ducatindia.com

Summary

- Maven is a simple build automation tool which is basically used for enterprise Java projects, designed to take much of the hard work out of the build process
- The JARs/WARs of any maven project can be shared across any distributed environments
- The Surefire Plugin is used during the test phase of the build lifecycle to execute the unit tests of an application
- POM is Project Object Model XML file that contains information about the project and configuration details used by Maven to build the project
- Maven is a build and dependency management tool for java based application development. Just like other java based development tools, it is not installed as windows service, rather it is configured using windows environment variables

DUCAT®
www.ducatindia.com

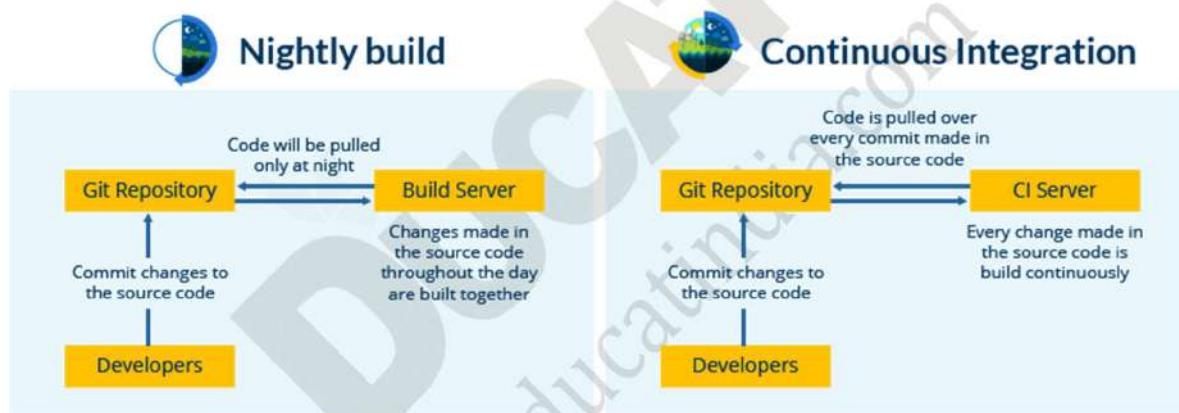
Chapter 17 – Jenkins

Introduction of Jenkins

Jenkins is an open source automation server. With Jenkins, organizations can accelerate the software development process by automating it. Jenkins manages and controls software delivery processes throughout the entire lifecycle, including build, document, test, package, stage, deployment, static code analysis. It is a server-based system that runs in servlet containers such as Apache Tomcat. It supports version control tools, including AccuRev, CVS, Subversion, Git, Mercurial, Perforce, ClearCase and RTC, and can execute Apache Ant, Apache Maven and sbt based projects as well as arbitrary shell scripts and Windows batch commands.

What is Continuous Integration?

Continuous Integration is a development practice in which the developers are required to commit changes to the source code in a shared repository several times a day or more frequently. Every commit made in the repository is then built. This allows the teams to detect the problems early. Apart from this, depending on the Continuous Integration tool, there are several other functions like deploying the build application on the test server, providing the concerned teams with the build and test results.



Jenkins and CI

Jenkins is a software that allows continuous integration. A continuous integration (CI) is a process in which all the development activities are integrated at a given point of time by compiling and building the project. In this process the developer's activities are collaborated and merged at the central system. Jenkins is one of the most popular open-source continuous integration and continuous delivery servers.

Jenkins Workflow

The workflow steps of Jenkins are:

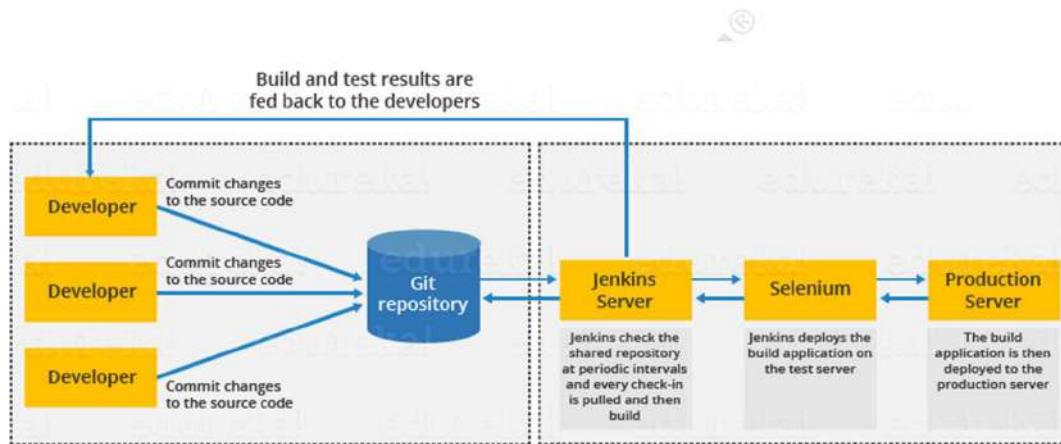
1. The developers check their source code
2. Jenkins picks the changed source code and trigger a build and run any tests if required.
3. The build output will be available in the Jenkins dashboards. Automatic notifications can also be sent back to the developer.

Features of Jenkins

The features of Jenkins are:

- It is easy to install & configure.
- It provides permanent link i.e. readable URLs for the build details.
- It facilitates email integration for notifications.
- It provides JUnit tests reporting.
- It provides tagging support for every successful builds.
- It facilitates distributed builds.
- It creates file fingerprinting for managing dependencies
- It supports variety of plugins.

Jenkins Architecture



The above diagram depicts the following functions:

- Developer commits the code to the source code repository.
- The Jenkins server checks the repository at regular intervals for changes.
- After a commit occurs, the Jenkins server detects the changes that have occurred in the source code repository. Jenkins pulls up those changes and starts preparing a new build.
- If the build fails, the concerned team is notified.
- If built is successful, then Jenkins deploys the built in the test server.
- After testing, Jenkins generates a feedback and notifies the developers about the build and test results.
- Jenkins continues to check the source code repository for changes made in the source code.

Jenkins Distributed Architecture

Jenkins uses a Master-Slave architecture to manage distributed builds. In this architecture, Master and Slave communicate through TCP/IP protocol.

Jenkins Master

The master server handles below tasks.

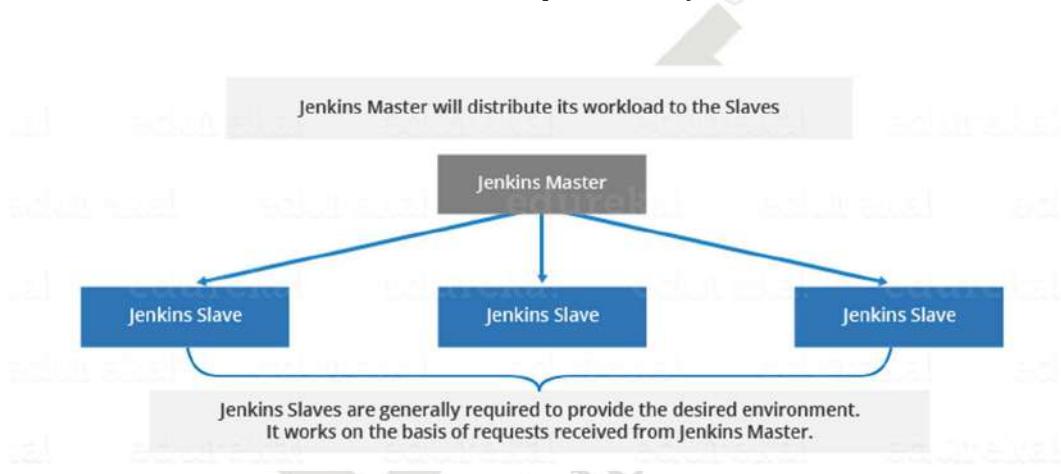
- Scheduling build jobs.
- Dispatching builds to the slaves for the actual execution.
- Monitor the slaves (possibly taking them online and offline as required).
- Recording and presenting the build results.
- A Master instance of Jenkins can also execute build jobs directly.

Jenkins Slave

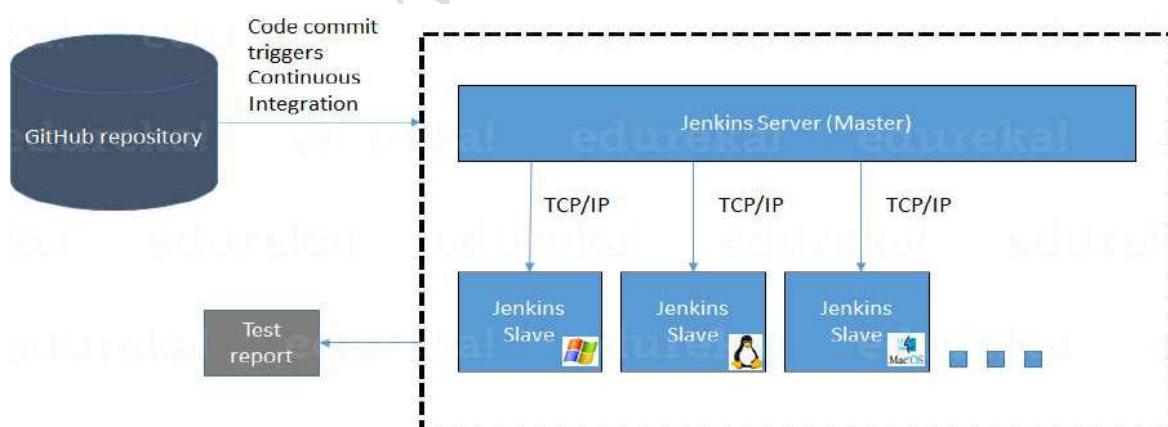
A Slave is a Java executable that runs on a remote machine. The slave server handles below tasks:

- It hears requests from the Jenkins Master instance.
- It executes build jobs dispatched by the Master.

The master and slave architecture can be represented by below flowchart.



A practical diagrammatic representation in which Jenkins is used for testing in different environments such as Ubuntu, MAC, and Windows is as below:



Jenkins Build Pipeline

The build pipeline is used to know which task Jenkins is currently executing. The Jenkins Pipeline gives an overview of where tests are up to. In build pipeline the build as a whole is broken down into sections, such as the unit test, acceptance test, and packaging, reporting and deployment phases. The pipeline phases can be executed in series or parallel, and if one phase is successful, it automatically moves on to the next phase.

Jenkins Plugins

Plugins have been released for Jenkins that extend its use to projects written in languages. Plugins are available for integrating Jenkins with most version control systems and bug databases. Many build tools are supported via their respective plugins. There are a set of plugins dedicated for the purpose of unit testing that generate test reports in various formats and automated testing which supports automated tests. Builds can generate test reports in various formats supported by plugins and Jenkins can display the reports and generate trends and render them in the GUI. The various plugins are:

- Mailer: Allows configuring email notifications for build results. Jenkins will send emails to the specified recipients whenever a certain important event occurs, such as:
 - Failed build.
 - Unstable build.
 - Successful build after a failed build, indicating that a crisis is over
 - Unstable build after a successful one, indicating that there's a regression
- Credentials: Allows storing credentials in Jenkins. Provides a standardized API for other plugins to store and retrieve different types of credentials.
- Monitoring external jobs: Adds the ability to monitor the result of externally executed jobs.
- SSH Agents: This plugin allows managing agents running on *nix machines over SSH. It adds a new type of agent launch method. This launch method will
 - Open a SSH connection to the specified host as the specified username,
 - Check the default version of java for that user,
 - If the default version is not compatible with Jenkins's agent.jar, try to find a version of java that is,
 - Once it has a suitable version of java, copy the latest agent.jar via SFTP,
 - Start the agent process.
- Javadoc: This plugin adds Javadoc support to Jenkins. The plugin enables the selection of "Publish Javadoc" as Post-build action, specifying the directory where the Javadoc is to be gathered and if retention is expected for each successful build.

Install Jenkins

The steps to install Jenkins is as below:

1. Install Java Version 8:

The steps to install Java depends upon the operating system. Here example is based on Linux OS. To install in Linux run below command
`sudo yum install java-1.8.0-openjdk`

2. Install Apache Tomcat Version 9

The steps to install Apache Tomcat depends upon the operating system. Here example is based on Linux OS.

- Install `wget` by running `sudo yum install wget`
 - Download the Tar file for Tomcat 9 using the below command using `wget`
`wget https://archive.apache.org/dist/tomcat/tomcat-9/v9.0.0.M10/bin/apache-tomcat-9.0.0.M10.tar.gz`
 - Extract the contents from this downloaded Tomcat 9 tar file
`tar xzf apache-tomcat-9.0.0.M10.tar.gz`
 - Move this extracted file to a new directory Tomcat9 using the `mv` command
`mv apache-tomcat-9.0.0.M10 tomcat9`
 - Provide a username and password for Apache Tomcat.
`gedit /home/edureka/tomcat9/conf/tomcat-users.xml`
 - Delete the content of the `tomcat-users.xml` file. Copy the below block and paste it in `tomcat-users.xml` file.
- ```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
 <role rolename="manager-gui"/>
 <role rolename="manager-script"/>
 <role rolename="manager-jmx"/>
 <role rolename="manager-jmx"/>
 <role rolename="admin-gui"/>
 <role rolename="admin-script"/>
 <user username="edureka" password="edureka"
 roles="manager-gui,manager-script,manager-jmx,manager-
 status,admin-gui,admin-script"/>
</tomcat-users>
```
- Start Apache Tomcat  
`./bin/startup.sh`

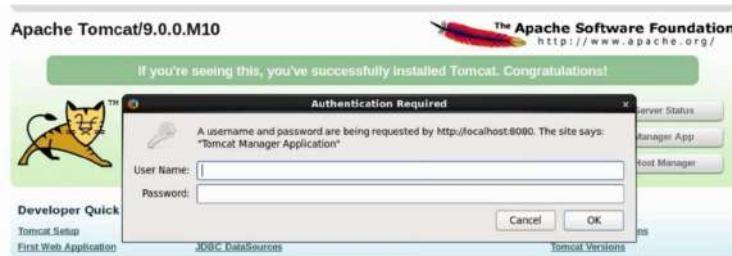
## 3. Download Jenkins war File

- Download Jenkins war file by using `wget` command:

```
wget http://updates.jenkins-
ci.org/download/war/2.7.3/jenkins.war
```

## 4. Deploy Jenkins war File

- To deploy Jenkins war file that you have downloaded in the previous step, open your browser and access `localhost:8080` again. Now click on the Manager App and give user name and password



- You will be directed to Tomcat web application manager page. When you scroll down you will find an option called Deploy. Give the context path, i.e. /jenkins and directory URL, i.e. location of the Jenkins war file in your system and click on Deploy.

Deploy

Deploy directory or WAR file located on server

Context Path (required):

XML Configuration file URL:

WAR or Directory URL:

**Deploy**

- Now in the Tomcat web application manager page you can find Jenkins listed under Applications along with other web apps.

Applications					
Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes
jenkins	None specified	jenkins v29.3	true	0	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes
docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes

## Jenkins Setup

Follow the below step-by-step procedure to use Jenkins with Selenium

### STEP 1

Download Jenkins from the official website of Jenkins – Jenkins. Download the latest .war file. Jenkins can be started via the command line or can run in a web application server.

Refer to the below steps for the execution through the command line:

Open the command prompt and type java -jar and enter the path of a .war file

Drag and drop the jenkins.war file

```

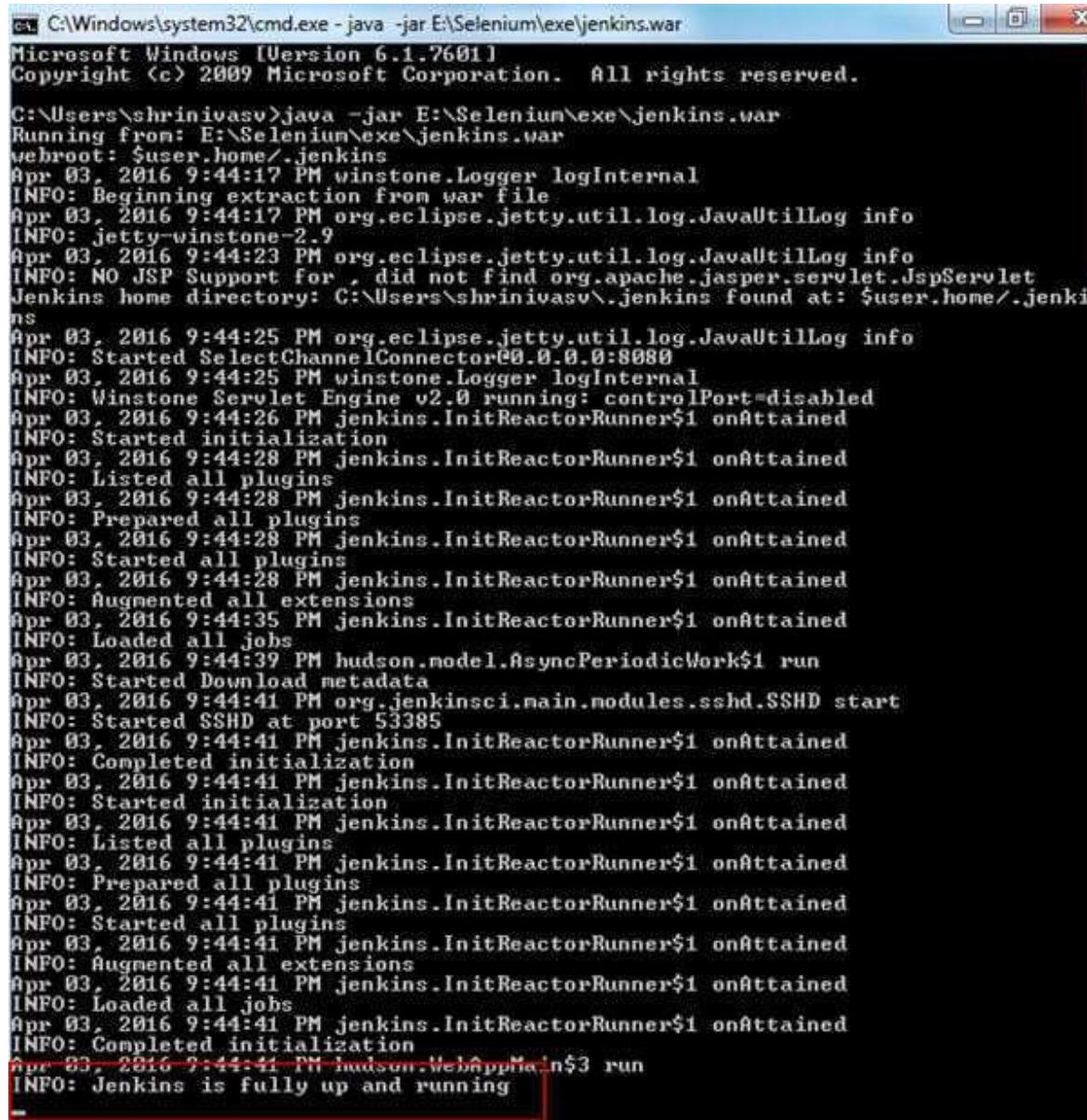
jenkins.war 3/8/2016 5:04 PM WAR File 63,104 KB
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\Users\shrinivasu>java -jar E:\Selenium\exe\jenkins.war

```

Press enter and check if your Jenkins.war file started to run and check the status information on the command prompt console.

It should show – Jenkins is fully up and running



```
C:\Windows\system32\cmd.exe - java -jar E:\Selenium\exe\jenkins.war
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\Users\shrinivasv>java -jar E:\Selenium\exe\jenkins.war
Running from: E:\Selenium\exe\jenkins.war
webroot: $user.home/.jenkins
Apr 03, 2016 9:44:17 PM winstone.Logger logInternal
INFO: Beginning extraction from war file
Apr 03, 2016 9:44:17 PM org.eclipse.jetty.util.log.JavaUtilLog info
INFO: jetty-winstone-2.9
Apr 03, 2016 9:44:23 PM org.eclipse.jetty.util.log.JavaUtilLog info
INFO: NO JSP Support for , did not find org.apache.jasper.servlet.JspServlet
Jenkins home directory: C:\Users\shrinivasv\.jenkins found at: $user.home/.jenki
ns
Apr 03, 2016 9:44:25 PM org.eclipse.jetty.util.log.JavaUtilLog info
INFO: Started SelectChannelConnector@0.0.0:8080
Apr 03, 2016 9:44:25 PM winstone.Logger logInternal
INFO: Winstone Servlet Engine v2.0 running: controlPort=disabled
Apr 03, 2016 9:44:26 PM jenkins.InitReactorRunner$1 onAttained
INFO: Started initialization
Apr 03, 2016 9:44:28 PM jenkins.InitReactorRunner$1 onAttained
INFO: Listed all plugins
Apr 03, 2016 9:44:28 PM jenkins.InitReactorRunner$1 onAttained
INFO: Prepared all plugins
Apr 03, 2016 9:44:28 PM jenkins.InitReactorRunner$1 onAttained
INFO: Started all plugins
Apr 03, 2016 9:44:28 PM jenkins.InitReactorRunner$1 onAttained
INFO: Augmented all extensions
Apr 03, 2016 9:44:35 PM jenkins.InitReactorRunner$1 onAttained
INFO: Loaded all jobs
Apr 03, 2016 9:44:39 PM hudson.model.AsyncPeriodicWork$1 run
INFO: Started Download metadata
Apr 03, 2016 9:44:41 PM org.jenkinsci.main.modules.sshd.SSHD start
INFO: Started SSHD at port 53385
Apr 03, 2016 9:44:41 PM jenkins.InitReactorRunner$1 onAttained
INFO: Completed initialization
Apr 03, 2016 9:44:41 PM jenkins.InitReactorRunner$1 onAttained
INFO: Started initialization
Apr 03, 2016 9:44:41 PM jenkins.InitReactorRunner$1 onAttained
INFO: Listed all plugins
Apr 03, 2016 9:44:41 PM jenkins.InitReactorRunner$1 onAttained
INFO: Prepared all plugins
Apr 03, 2016 9:44:41 PM jenkins.InitReactorRunner$1 onAttained
INFO: Started all plugins
Apr 03, 2016 9:44:41 PM jenkins.InitReactorRunner$1 onAttained
INFO: Augmented all extensions
Apr 03, 2016 9:44:41 PM jenkins.InitReactorRunner$1 onAttained
INFO: Loaded all jobs
Apr 03, 2016 9:44:41 PM jenkins.InitReactorRunner$1 onAttained
INFO: Completed initialization
Apr 03, 2016 9:44:41 PM hudson.WebAppMain$3 run
INFO: Jenkins is fully up and running
```

Now check whether your Jenkins is ready to use; by default, it uses port 8080.

Type “<http://localhost:8080>” in the browser and press enter. It will show you Jenkins UI.



The screenshot shows the Jenkins dashboard at localhost:8080. The left sidebar has links for New Item, People, Build History, Manage Jenkins, and Credentials. Under Build Queue, it says 'No builds in the queue.' Below that is a table for Build Executor Status with two idle entries. The main area displays three Jenkins jobs in a table:

S	W	Name	Last Success	Last Failure	Last Duration
		Demo-Srinivas-Jenkins	24 days · #9	N/A	9.5 sec
		QA_Metrics_Kill_Production_Report	17 days · #1	N/A	4 min 0 sec
		Test	N/A	N/A	N/A

Legend: RSS for all, RSS for failures, RSS for just test builds.

It will load the Jenkins dashboard empty by default. I created some Jenkins job in the above screenshot as an example and hence, it did not load empty.

## STEP 2

To use Selenium with Jenkins you need to configure Jenkins with Selenium.

Follow the below steps:

- Go to Jenkins dashboard
- Click on manage Jenkins
- Click on configure Jenkins
- Click on JDK installation – In JDK name section enter the name, under Java Home section – give your java path



JDK Name	JAVA_HOME
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_66

Install automatically

**Delete JDK**

**Add JDK**

List of JDK installations on this system

The radio button, Install automatically is checked by default. You need to uncheck it because it will automatically update with the new Java version and there might be a possibility that Selenium doesn't support the new Java version. It is better to uncheck it. Now click on apply and save.

Your Jenkins is configured with Selenium and is now ready to be used with Selenium. Both Jenkins and Selenium code is written in Java. Hence, if you give the Java path then internally it will communicate and process your job.

### STEP 3

Now, create a Selenium script and a TestNG XML file. This TestNG XML file will be required for creating a batch file and we will use that batch file in Jenkins. Refer below TestNG code:

Refer below TestNG code:

```

package session_2;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.Test;

public class jenkins_demo
{

 @Test
 public void testgooglsearch(){

 WebDriver driver = new FirefoxDriver();

 driver.get("http://google.in");
 String Expectedtitle = "Google";
 String Actualtitle = driver.getTitle();
 System.out.println("Before Assertion " + Expectedtitle + Actualtitle);
 Assert.assertEquals(Actualtitle, Expectedtitle);
 System.out.println("After Assertion " + Expectedtitle + Actualtitle + " Title matched ");

 }
}

package session_2;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.Test;

public class jenkins_demo
{

 @Test
 public void testgooglsearch(){

 WebDriver driver = new FirefoxDriver();
 //it will open the google page
 driver.get("http://google.in");
 //we expect the title "Google " should be present
 }
}

```

```

String Expectedtitle = "Google";
//it will fetch the actual title
String Actualtitle = driver.getTitle();
System.out.println("Before Assertion " + Expectedtitle +
Actualtitle);
//it will compare actual title and expected title
Assert.assertEquals(Actualtitle, Expectedtitle);
//print out the result
System.out.println("After Assertion " + Expectedtitle + Actualtitle
+ " Title matched ");
}
}

```

## Output

Before Assertion GoogleGoogle  
 After Assertion, GoogleGoogle Title matched  
 PASSED: testgooglrsearch

## Create a TestNG xml file, refer below code:

```

<suite name="Sample Suite">
 <test name="Learning Jenkins" >
 <classes>
 <class name="session_2.jenkins_demo"></class>
 </classes>
 </test>
</suite>

```

## STEP 4

Go to your project root directory and create a library folder.

Refer the screenshot below:

	Name	Date modified	Type	Size
ites	.settings	12/10/2015 12:26 ...	File folder	
ktop	bin	3/31/2016 10:48 AM	File folder	
vnloads	lib	3/8/2016 6:06 PM	File folder	
pbox	src	3/31/2016 10:46 AM	File folder	
ent Places	test-output	4/3/2016 2:10 PM	File folder	
ies	.classpath	3/12/2016 12:57 PM	CLASSPATH File	1 KB
uments	.project	9/27/2015 2:58 PM	PROJECT File	1 KB
sic	run	3/9/2016 11:35 AM	Windows Batch File	1 KB
ures	testng	4/3/2016 2:19 PM	XML Document	1 KB

Now, add all your jar files which are required for running your Selenium script:

	Name	Date modified	Type	Size
tes	apache-log4j-1.2.15	10/5/2011 12:38 PM	Executable Jar File	383 KB
stop	commons-codec-1.9	11/8/2014 12:52 PM	Executable Jar File	258 KB
vnloads	commons-logging-1.1.3	11/8/2014 12:52 PM	Executable Jar File	61 KB
pbox	javassist-api-5.0.3	6/2/2013 10:08 PM	Executable Jar File	2,046 KB
ent Places	junit-4.11	11/8/2014 12:52 PM	Executable Jar File	240 KB
es	jxl-2.6	8/2/2013 12:07 PM	Executable Jar File	645 KB
uments	mail	3/6/2013 4:16 PM	Executable Jar File	509 KB
ic	poi-3.11-20141221	12/17/2014 1:08 AM	Executable Jar File	2,032 KB
ures	poi-examples-3.11-20141221	12/17/2014 1:08 AM	Executable Jar File	321 KB
os	poi-excelant-3.11-20141221	12/17/2014 1:08 AM	Executable Jar File	30 KB
uter	poi-ooxml-3.11-20141221	12/17/2014 1:08 AM	Executable Jar File	1,183 KB
el Disk (C:)	poi-ooxml-schemas-3.11-20141221	12/17/2014 1:08 AM	Executable Jar File	5,465 KB
Volume (E:)	poi-scratchpad-3.11-20141221	12/17/2014 1:08 AM	Executable Jar File	1,260 KB
Drive (F:)	selenium-java-2.52.0	2/11/2016 11:07 AM	Executable Jar File	1,843 KB
rk	selenium-java-2.52.0-srcs	2/11/2016 11:07 AM	Executable Jar File	668 KB
	selenium-server-standalone-2.52.0	2/16/2016 2:53 PM	Executable Jar File	30,227 KB
	testng-6.8.21	4/10/2015 6:16 PM	Executable Jar File	818 KB
	xmlbeans-2.6.0	11/8/2014 12:53 PM	Executable Jar File	2,667 KB

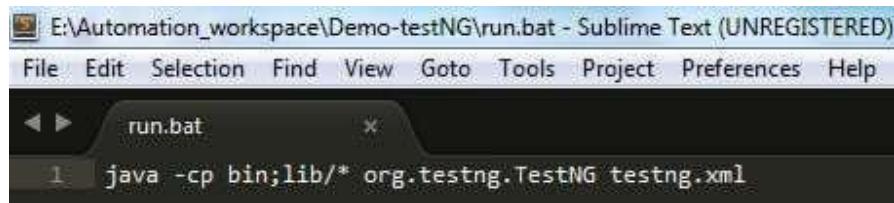
## STEP 5

Creating a batch file and using it in Jenkins

Create a batch file by following the below steps:

Open the notepad and type:- Java -cp bin;lib/\* org.testng.TestNG testng.xml

- By doing this, Java -cp will compile and execute a .class file which is located at bin directory and all our executable jar file is located at lib directory and we are using a TestNG framework so specify org.testng.TestNG. Also, specify the name of xml file which will trigger the expected TestNG script.
- Save the file with .bat extension and check the type of file. It should be “windows batch file”. To cross-check whether the batch file is created properly, double-click the batch file and it will execute the code. Refer the below code of batch file:



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

## STEP 6

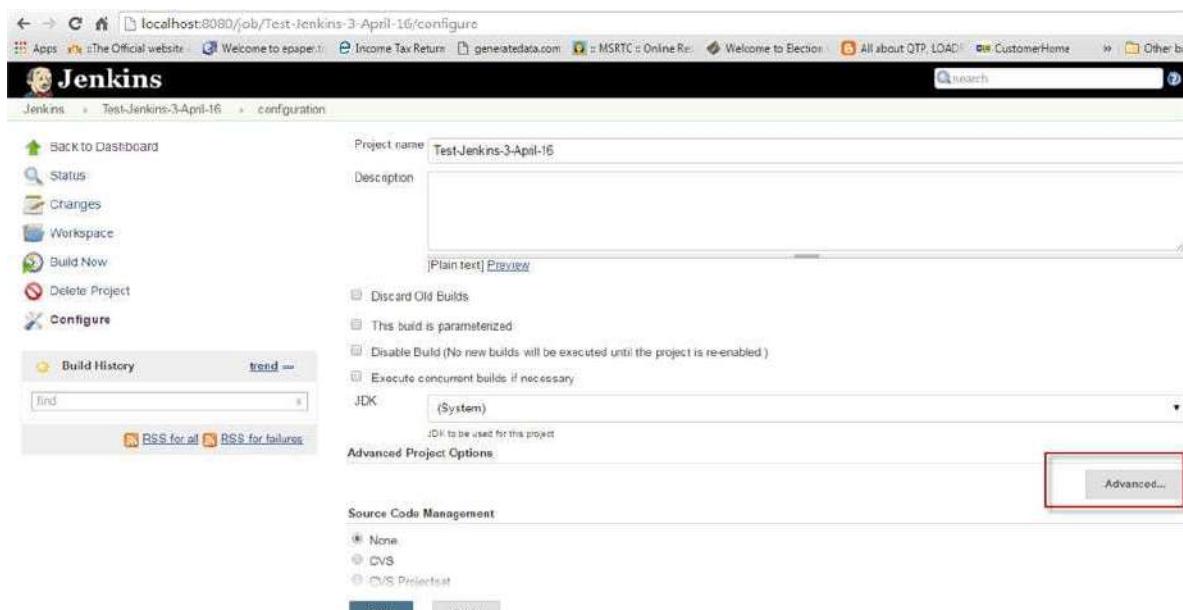
Next, we need to add a batch file in Jenkins.

For adding the batch file follow the below steps:

- Go to the Jenkins dashboard, create a new job in Jenkins
- Click on a new item and enter the item name and check the freestyle project radio button

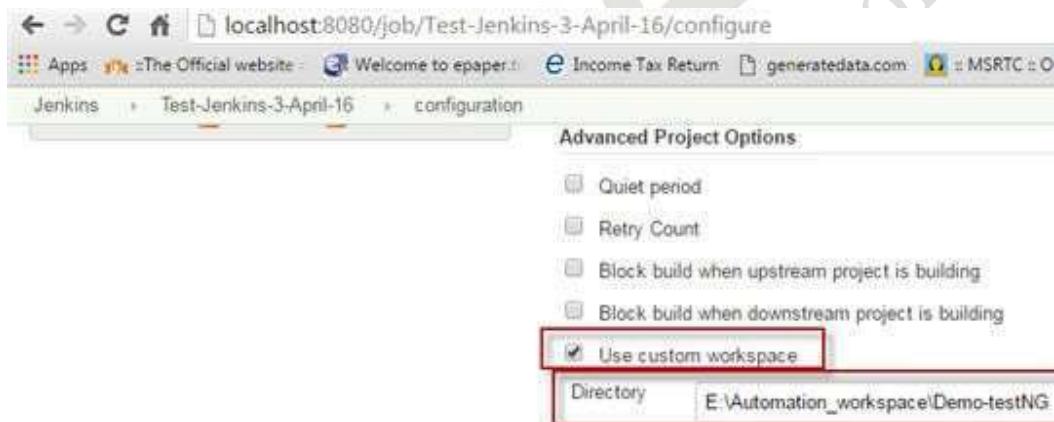


- Click Advanced options



The screenshot shows the Jenkins configuration page for the job 'Test-Jenkins-3-April-16'. The 'Project name' is set to 'Test-Jenkins-3-April-16'. The 'Description' field is empty. Under 'Build Triggers', there are four options: 'Discard Old Builds', 'This build is parameterized', 'Disable Build (No new builds will be executed until the project is re-enabled)', and 'Execute concurrent builds if necessary'. The 'JDK' dropdown is set to '(System)'. At the bottom right, there is a red-bordered 'Advanced...' button.

- Click on use custom workspace and give your Selenium script project workspace path: "E:\Automation\_workspace\Demo-testNG"



The screenshot shows the 'Advanced Project Options' section of the Jenkins configuration page. It includes options for 'Quiet period', 'Retry Count', 'Block build when upstream project is building', 'Block build when downstream project is building', and 'Use custom workspace'. The 'Use custom workspace' checkbox is checked and highlighted with a red border. Below it, the 'Directory' field contains the path 'E:\Automation\_workspace\Demo-testNG'.

- Then go to Build and Select an option from the drop-down box, execute your build through Windows batch command
- And give your batch file name here – "run.bat"

Build

Execute Windows batch command

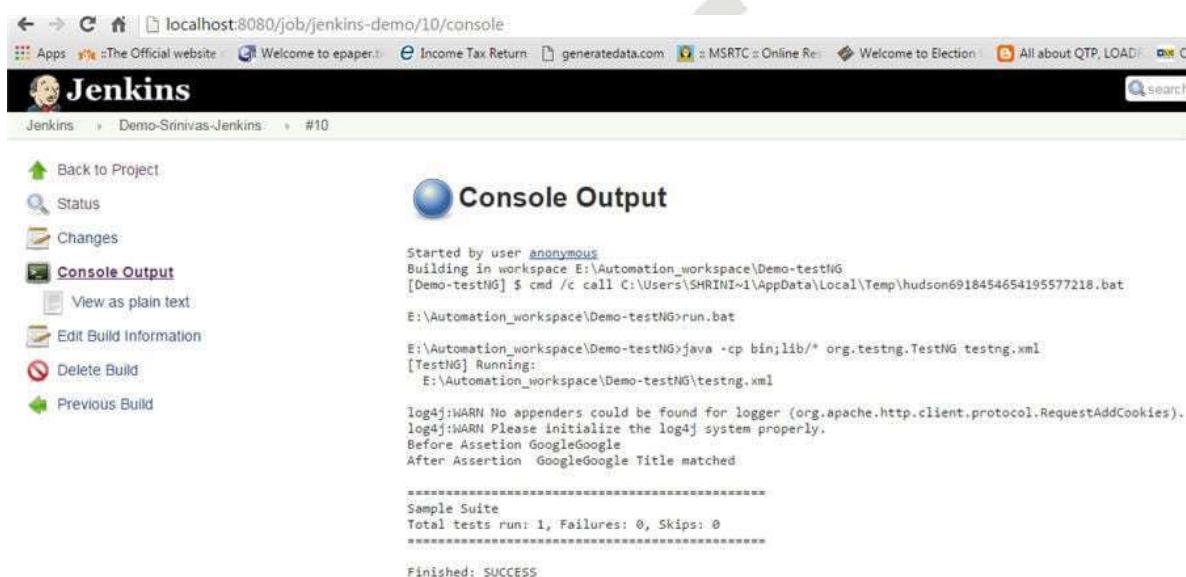
Command: run.bat

[See the list of available environment variables](#)

[Delete](#)

[Save](#) [Apply](#)

- Click on apply and save
- Click on the build now and see the build result on console output



The screenshot shows the Jenkins interface with the following details:

- URL:** localhost:8080/job/jenkins-demo/10/console
- Job Name:** Jenkins
- Build Number:** #10
- Console Output:**

```

Started by user anonymous
Building in workspace E:\Automation_workspace\Demo-testNG
[Demo-testNG] $ cmd /c call C:/Users/SHRINI~1/AppData/Local/Temp/hudson6918454654195577218.bat
E:\Automation_workspace\Demo-testNG>java -cp bin;lib/* org.testng.TestNG testng.xml
[TestNG] Running:
E:\Automation_workspace\Demo-testNG\testng.xml

log4j:WARN No appenders could be found for logger (org.apache.http.client.protocol.RequestAddCookies).
log4j:WARN Please initialize the log4j system properly.
Before Assertion GoogleGoogle
After Assertion GoogleGoogle Title matched

=====
Sample Suite
Total tests run: 1, Failures: 0, Skips: 0
=====

Finished: SUCCESS

```

## Selenium Integration with Jenkins

Jenkins is an open source tool written in Java and provides continuous delivery and continuous integration service for software development. It automates your manual task of code deployment process from development box -> QA -> Stage -> Production. Jenkins supports plugins which can integrate with various tools such as Git, SVN, build pipeline. The basic functionality of Jenkins is to execute a predefined list of steps on the basis of time and some event.

### Example to use Jenkins with Selenium

1. Create a Selenium script and a TestNG XML file. This TestNG XML file will be required for creating a batch file and we will use that batch file in Jenkins.

Create a TestNG code like below example:

```

package session_2;
import org.openqa.selenium.WebDriver;

```

```

import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.Test;
public class jenkins_demo
{
 @Test
 public void testgooglrsearch()
 {
 WebDriver driver = new FirefoxDriver();
 //it will open the goggle page
 driver.get("http://google.in");
 //we expect the title "Google " should be present
 String Expectedtitle = "Google";
 //it will fetch the actual title
 String Actualtitle = driver.getTitle();
 System.out.println("Before Assertion " + Expectedtitle +
 Actualtitle);
 //it will compare actual title and expected title
 Assert.assertEquals(Actualtitle, Expectedtitle);
 //print out the result
 System.out.println("After Assertion " + Expectedtitle +
 Actualtitle + " Title matched ");
 }
}

```

#### Output:

```

Before Assertion GoogleGoogle
After Assertion, GoogleGoogle Title matched
PASSED: testgooglrsearch

```

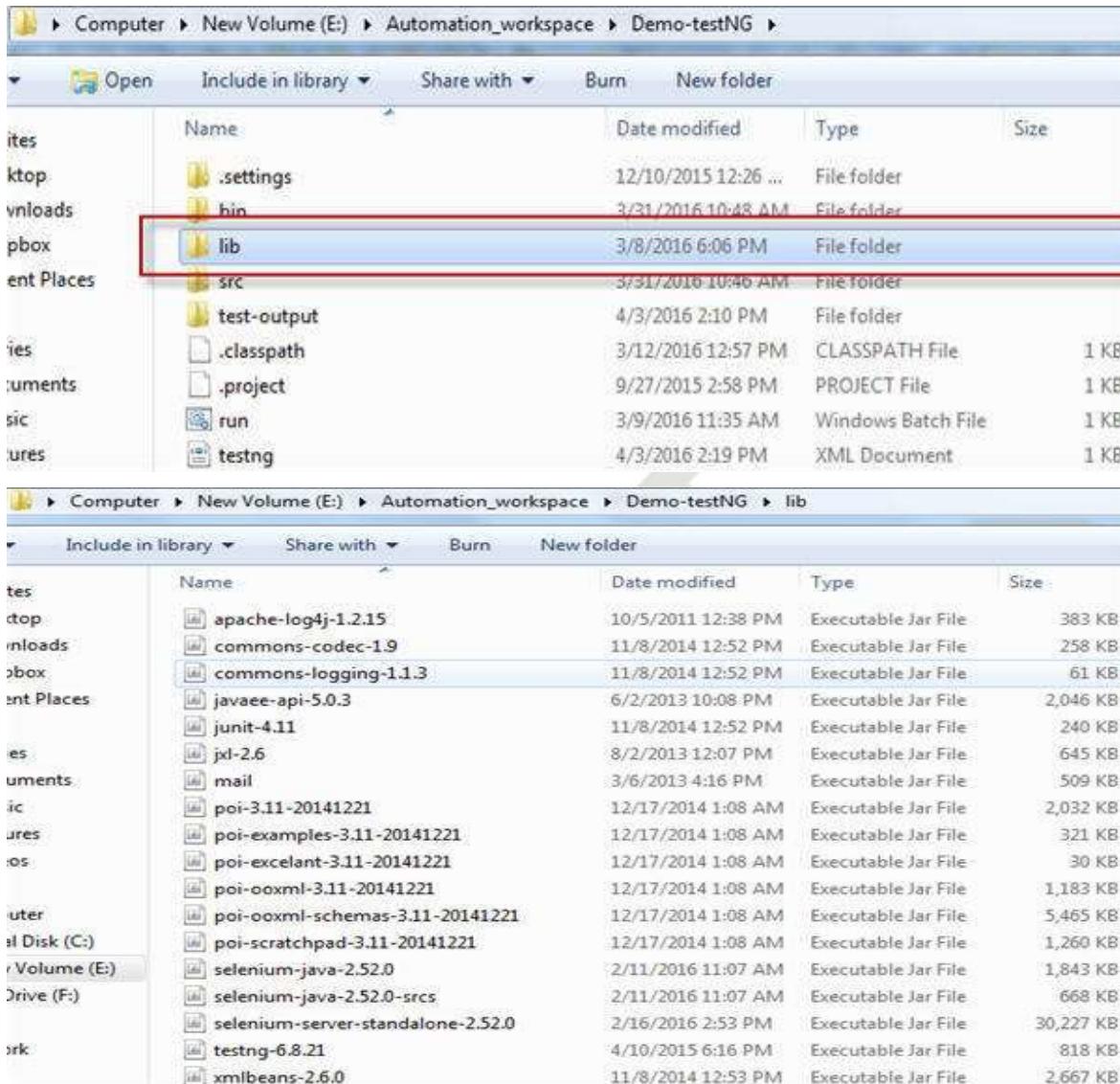
#### Create a TestNG xml file:

```

<suite name="Sample Suite">
 <test name="Test Jenkins">
 <classes>
 <class name="session_2.jenkins_demo"></class>
 </classes>
 </test>
</suite>

```

2. Go to your project root directory and create a library folder and add all your jar files which are required for running your Selenium script.



	Name	Date modified	Type	Size
ites	.settings	12/10/2015 12:26 ...	File folder	
ktop	bin	3/31/2016 10:48 AM	File folder	
vloads	lib	3/8/2016 6:06 PM	File folder	
pbox	src	3/31/2016 10:46 AM	File folder	
ent Places	test-output	4/3/2016 2:10 PM	File folder	
ies	.classpath	3/12/2016 12:57 PM	CLASSPATH File	1 KB
uments	.project	9/27/2015 2:58 PM	PROJECT File	1 KB
sic	run	3/9/2016 11:35 AM	Windows Batch File	1 KB
ures	testng	4/3/2016 2:19 PM	XML Document	1 KB

	Name	Date modified	Type	Size
tes	apache-log4j-1.2.15	10/5/2011 12:38 PM	Executable Jar File	383 KB
stop	commons-codec-1.9	11/8/2014 12:52 PM	Executable Jar File	258 KB
inloads	commons-logging-1.1.3	11/8/2014 12:52 PM	Executable Jar File	61 KB
pbox	javassist-api-5.0.3	6/2/2013 10:08 PM	Executable Jar File	2,046 KB
ent Places	junit-4.11	11/8/2014 12:52 PM	Executable Jar File	240 KB
es	jxl-2.6	8/2/2013 12:07 PM	Executable Jar File	645 KB
uments	mail	3/6/2013 4:16 PM	Executable Jar File	509 KB
ic	poi-3.11-20141221	12/17/2014 1:08 AM	Executable Jar File	2,032 KB
ures	poi-examples-3.11-20141221	12/17/2014 1:08 AM	Executable Jar File	321 KB
ios	poi-excelant-3.11-20141221	12/17/2014 1:08 AM	Executable Jar File	30 KB
uter	poi-ooxml-3.11-20141221	12/17/2014 1:08 AM	Executable Jar File	1,183 KB
l Disk (C:)	poi-ooxml-schemas-3.11-20141221	12/17/2014 1:08 AM	Executable Jar File	5,465 KB
r Volume (E:)	poi-scratchpad-3.11-20141221	12/17/2014 1:08 AM	Executable Jar File	1,260 KB
Drive (F:)	selenium-java-2.52.0	2/11/2016 11:07 AM	Executable Jar File	1,843 KB
rk	selenium-java-2.52.0-srcs	2/11/2016 11:07 AM	Executable Jar File	668 KB
	selenium-server-standalone-2.52.0	2/16/2016 2:53 PM	Executable Jar File	30,227 KB
	testng-6.8.21	4/10/2015 6:16 PM	Executable Jar File	818 KB
	xmlbeans-2.6.0	11/8/2014 12:53 PM	Executable Jar File	2,667 KB

3. Create a batch file and using it in Jenkins

- Open the notepad and type:- Java -cp bin;lib/\* org.testng.TestNG testng.xml

By doing this, Java -cp will compile and execute .class file which is located at bin directory and all our executable jar file are located at lib directory. Specify org.testng.TestNG. Also, specify the name of xml file which will trigger the expected TestNG script.

- Save the file with .bat extension and check the type of file. It should be "windows batch file". To cross check whether the batch file is created properly, double click the batch file and it will execute the code.

4. To add a batch file in Jenkins. For adding the batch file follow the below steps:

- Go to the Jenkins dashboard, create a new job in Jenkins

- Click on new item and enter the item name and check the freestyle project radio button
- Click Advance options
- Click on use custom workspace and give your Selenium script project workspace path: “E:\Automation\_workspace\Demo-testNG”
- Then go to Build and Select option from drop down box, execute your build through Windows batch command
- Give batch file a name. Click on apply and save
- Click on build now and see the build result on console output

DUCAT®  
[www.ducatindia.com](http://www.ducatindia.com)

## Summary

- Jenkins is an open source tool written in Java and provides continuous delivery and continuous integration service for software development
- The basic functionality of Jenkins is to execute a predefined list of steps on the basis of time and some event
- A continuous integration is a process in which all the development activities are integrated at a given point of time by compiling and building the project
- Jenkins supports plugins which can integrate with various tools such as Git, SVN, build pipeline

DUCAT®  
[www.ducatindia.com](http://www.ducatindia.com)

# Chapter 18 – Hybrid Framework

## Hybrid Framework

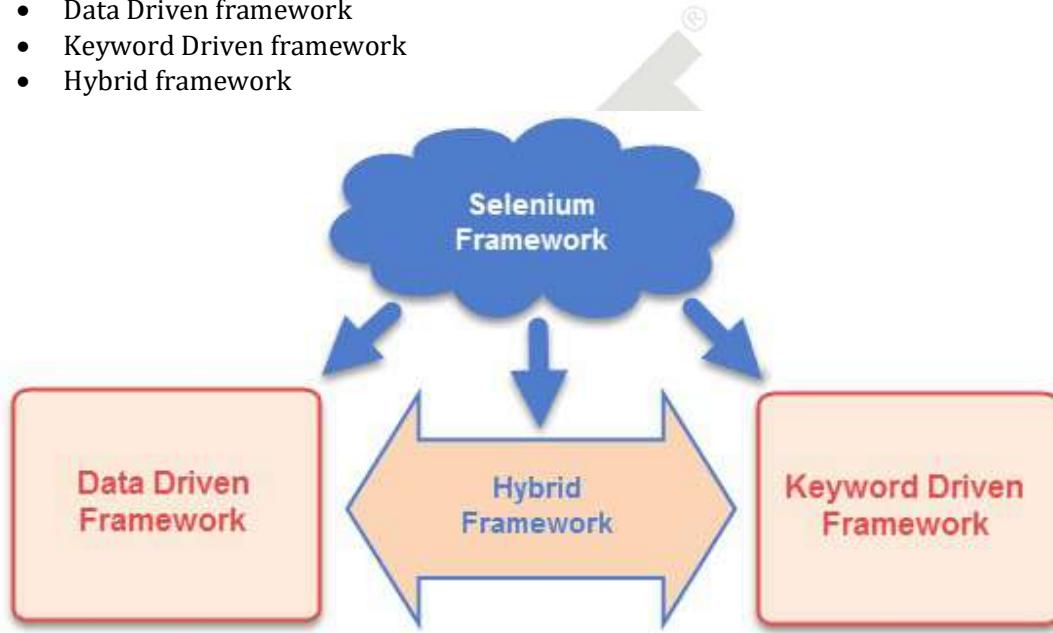
### Introduction

Selenium framework is a code structure for making code maintenance simpler, and code readability better. A framework involves breaking the entire code into smaller pieces of code, which test a particular functionality.

The code is structured such that, the “data set” is separated from the actual *test case* which will test the functionality of the web application. It can also be structured in a way wherein, the test cases which need to be executed are called (invoked) from an external application (like a .csv).

There are a number of frameworks out there, but 3 commonly used Selenium framework (s) are:

- Data Driven framework
- Keyword Driven framework
- Hybrid framework



Without a framework in place, there will be one test case which will comprise the entire test functionality. The scary part is, this single test case has the capability to rise up to a million lines of code. So it's pretty obvious that a test case so huge will be tough to read. Even if it wants to modify any functionality later, then it will have a tough time modifying the code.

Since the implementation of a framework, will result in smaller but multiple code pieces, there are various benefits.

### Benefits of Selenium framework

- Increased code re-usage
- Improved code readability

- Higher portability
- Reduced script maintenance

## Hybrid framework

A combination of the *Data-Driven* and *Keyword-Driven* (or Modular-Driven) frameworks is commonly said to be a Hybrid-Driven framework. In general, a Hybrid-Driven framework is a collection of two or more frameworks that can be customized and accessed by any user. For example, a combination of Page Objects, a *Keyword-Driven framework*, a Data-Driven framework, an *object repository*, and reporting listeners provides a powerful Hybrid framework. Hybrid Test framework is a concept where we are using the advantage of both Keyword and Data driven framework.

### Advantages of Hybrid Framework

The Hybrid framework is built with a number of reusable modules / function libraries that are developed with the following features in mind:

- **Maintainability:** Hybrid framework significantly reduces maintenance effort
- **Re-usability:** It allows to reuse test cases and library functions
- **Manageability:** effective test design, execution, and traceability
- **Accessibility:** easy to design, develop, modify and debug test cases while executing
- **Availability:** Allows to schedule automation execution
- **Reliability:** due to advanced error handling and scenario recovery
- **Flexibility:** framework independent of system or environment under test
- **Measurability:** customizable reporting of test results ensure the quality output

A Hybrid framework can be built by storing the methods to execute in an excel file (keyword driven approach) and passing these method names to the Java Reflection Class (data driven approach) instead of creating an If/Else loop in the *DriverScript* class.

#### Example:

```

import java.lang.reflect.Method;
public class DriverScriptJava
{
 //This is a class object, declared as 'public static'
 //So that it can be used outside the scope of main[] method
 public static Actions actionKeywords;
}

```

```

public static String sActions;

//This is reflection class object, declared as 'public static'
//So that it can be used outside the scope of main[] method
public static Method method[];

public static void main(String[] args) throws Exception
{
 //Declaring the path of the Excel file with the name of the Excel
 //file
 String sPath = "C:\\\\Users\\\\Vardhan\\\\workspace\\\\Selenium Frameworks
Demo\\\\
dataEngine.xlsx";

 //Here we are passing the Excel path and SheetName to connect with
 //the
 Excel file
 //This method was created previously
 ReadExcelData.setExcelFile(sPath, "Sheet1");

 //Hard coded values are used for Excel row & columns for now
 //Later on, we will use these hard coded value much more
 //efficiently
 //This is the loop for reading the values of the column (Action
 //Keyword)
 row by row
 //It means this loop will execute all the steps mentioned for the
 test
 case in
 Test Steps sheet
 for (int iRow=1;iRow<=7;iRow++)
 {
 sActions = ReadExcelData.getCellData(iRow, 1);
 //A new separate method is created with the name 'execute_Actions'
 // this method can be find below of the this test
 //So this statement is doing nothing but calling that piece of code
 to
 execute
}

```

```

execute_Actions();
}

}

//This method contains the code to perform some action
//As it is completely different set of logic, which revolves around
the
action only, it makes sense to keep it separate from the main
driver
script
//This is to execute test step (Action)
private static void execute_Actions() throws Exception
{
//Here we are instantiating a new object of class 'Actions'
actionKeywords = new Actions();

//This will load all the methods of the class 'Actions' in it.
//It will be like array of method, use the break point here and do
the
watch
method = actionKeywords.getClass().getMethods();

//This is a loop which will run for the number of actions in the
Action
Keyword class
//method variable contain all the method and method.length returns
the
Total number of methods
for(int i = 0;i<method.length;i++)
{
//This is now comparing the method name with the ActionKeyword
value
received from the excel
if(method[i].getName().equals(sActions))
{ //In case of match found, it will execute the matched method
method[i].invoke(actionKeywords);
//Once any method is executed, this break statement will
}
}

```

```
take the flow outside of for loop
 break;
}
}
}
}
```

Take a look at the modified “DriverScript” class in the below code snippet. Here, instead of using multiple If/ Else loops, data driven approach is used to read the method names from the excel file.

# Combining Multiple Frameworks for Test Execution

The Keywords and the scripts for all the test cases are same as in KDF (Keyword Driven Framework). However, TC3: Open the order has been parameterized. Hence the script for this test case is written to receive the order number from an Excel file and to write the customer name into the excel file

```
Test Case1(TC1): Login into the application
Keyword:Login ()

Test Case2(TC2): Insert the Order
Keyword:InsertOrder()

Test Case3(TC3): Open the Order for several order numbers
Keyword: OpenOrder()
```

Description: Here the same script used to develop a DDF (Data Driven Framework) is used, thereby achieving the test case for several iterations.

## Script:

Test Case4(TC4): Delete the Order  
Keyword:DeleteOrder()  
Test Case5(TC5): Close the application  
Keyword:CloseApp()

By following this simple method, parameterization of TC3 is achieved. If applicable, all the other test cases can also be parameterized in the same test.

Above example, is a very simple way of designing hybrid framework. The same framework can also be achieved with descriptive programming.

## Advantages

- The time taken to run the test designed with hybrid framework is relatively less compared to other frameworks
- This can be used when we need all the test cases and inputs that are associated with particular test case, in the same test suite.

## Disadvantages

- Clear knowledge on the combining different framework is required.

**Example code:** Below is a simple class for Hybrid (Modular and Data Driven) framework:-

*SearchData.java* has the code for data to be used in the test.

```
package com.data;

public class SearchData {
 private String url;
 private String searchWord;

 public String getUrl() {
 return url;
 }

 public void setUrl(String url) {
 this.url = url;
 }

 public String getSearchWord() {
 return searchWord;
 }

 public void setSearchWord(String searchWord) {
 this.searchWord = searchWord;
 }
}
```

```

}

SearchPage.java
Contains modules of code.

package com.page;

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

import com.data.SearchData;

public class SearchPage {
 WebDriver driver;

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

 public void launchGoogle(SearchData searchData) {
 driver.get(searchData.getUrl());
 }

 public void search(SearchData searchData) {

 driver.findElement(By.name("q")).sendKeys(searchData.getSearchWord());
 driver.findElement(By.name("btnG")).click();
 }
}

```

*GoogleTest.java* contains the actual *Junit Test*.

```

package com.test;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.WebDriver;

```

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

import com.data.SearchData;
import com.page.SearchPage;

public class GoogleTest {
 protected WebDriver driver;

 @Before
 public void setUp() {
 driver = new FirefoxDriver();
 }

 @After
 public void tearDown() {
 driver.close();
 driver.quit();
 }

 @Test
 public void searchTest() throws InterruptedException {
 // set the data
 searchData searchData = new searchData();
 searchData.setUrl("https://www.google.com");
 searchData.setSearchWord("Selenium");

 // call the methods
 SearchPage searchPage = new SearchPage(driver);
 searchPage.launchGoogle(searchData);
 searchPage.search(searchData);
 Thread.sleep(10000);
 }
}
```

## Summary

- We can create three types of test framework using Selenium WebDriver.

- These are Data Driven, Keyword Driven, and Hybrid test framework.
- We can achieve Data-driven framework using TestNG's data provider.
- In Keyword driven framework, keywords are written in some external files like excel file and java code will call this file and execute test cases.
- The hybrid framework is a mix of keyword driven and data driven framework.

## Chapter 19 –Page Object Model (POM)

### Page Object Model

Page Object Model is a Design Pattern which has become popular in Selenium Test Automation. It is widely used design pattern in Selenium for enhancing test maintenance and reducing code duplication. Page object model (POM) can be used in any kind of framework such as modular, data-driven, keyword driven, hybrid framework etc. A page object is an object-oriented class that serves as an interface to a page of your Application Under Test(AUT). The tests then use the methods of this page object class whenever they need to interact with the User Interface (UI) of that page. The benefit is that if the UI changes for the page, the tests themselves don't need to change, only the code within the page object needs to change. Subsequently, all changes to support that new UI is located in one place.

### Page Factory

The *PageFactory* Class in Selenium is an extension to the Page Object design pattern. It is used to initialize the elements of the Page Object or instantiate the Page Objects itself. Annotations for elements can also be created (and recommended) as the describing properties may not always be descriptive enough to tell one object from the other.

It is used to initialize elements of a Page class without having to use '*FindElement*' or '*FindElements*'. Annotations can be used to supply descriptive names of target objects to improve code readability. There is however a few differences between C# and Java implementation – Java provide greater flexibility with *PageFactory*.

### Difference between Page Object Model (POM) and Page Factory

Page Object is a class that represents a web page and hold the functionality and members.

Page Factory is a way to initialize the web elements you want to interact with within the page object when you create an instance of it.

### Advantages of Page Object Model Framework

- Code reusability: We could achieve code reusability by writing the code once and use it in different tests.
- Code maintainability: There is a clean separation between test code and page specific code such as locators and layout which becomes very easy to maintain code. Code changes only on Page Object Classes when a UI change occurs. It enhances test maintenance and reduces code duplication.
- Object Repository: Each page will be defined as a java class. All the fields in the page will be defined in an interface as members. The class will then implement the interface.
- Readability: Improves readability due to clean separation between test code and page specific code

## Set up Page Object Model (POM) in Selenium

Let's assume below program our base test case and implement the Page Object Model (POM) in it. The steps include

- Create a 'New Package' file and name it as 'pageObjects', by right click on the Project and select New > Package. We will be creating different packages for Page Objects, Utilities, Test Data, Test Cases and Modular actions. It is always recommended to use this structure, as it is easy to understand, easy to use and easy to maintain.
- Create a 'New Class' file and refer the name to the actual page from the test object, by right click on the above created Package and select New > Class. In our case it is Home Page and LogIn Page.
- Now create a Static Method for each Element (Object) in the Home Page. Each method will have an Argument (driver) and a Return value (element).

```

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

public class Home_Page {

 private static WebElement element = null;

 public static WebElement lnk_MyAccount(WebDriver driver) {

 element = driver.findElement(By.id("account"));
 }
}

```

```

 return element;

 }

public static WebElement lnk_LogOut(WebDriver driver) {

 element = driver.findElement(By.id("account_logout"));

 return element;

}

```

Driver is being passed as an Argument so that Selenium is able to locate the element on the browser (driver).

Element is returned, so that an Action can be performed on it.

Method is declared as Public Static, so that it can be called in any other method without instantiate the class.

Follow the same rule for creating LogIn Page class.

```

import org.openqa.selenium.*;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

public class LogIn_Page {

 private static WebElement element = null;

 public static WebElement txtbx_UserName(WebDriver driver) {

 element = driver.findElement(By.id("log"));

 return element;
 }
}

```

```

 }

public static WebElement txtbx_Password(WebDriver driver) {

 element = driver.findElement(By.id("pwd"));

 return element;

}

public static WebElement btn_LogIn(WebDriver driver) {

 element = driver.findElement(By.id("login"));

 return element;

}

```

- Create a ‘New Class’ and name it as POM\_TC by right click on the ‘automationFramework’ Package and select New > Class. We will be creating all our test cases under this package.

Now convert your old First Test Case in to the new Page Object Model test case.

```

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

// Import package pageObject.*

import pageObjects.Home_Page;

import pageObjects.LogIn_Page;

```

```

public class PageObjectModel {

 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.store.demoqa.com");

 // Use page Object library now

 Home_Page.lnk_MyAccount(driver).click();

 LogIn_Page.txtbx_UserName(driver).sendKeys("testuser_1");

 LogIn_Page.txtbx_Password(driver).sendKeys("Test@123");

 LogIn_Page.btn_LogIn(driver).click();

 System.out.println(" Login Successfully, now it is the
 time to Log Off buddy.");

 Home_Page.lnk_LogOut(driver).click();

 driver.quit();

 }

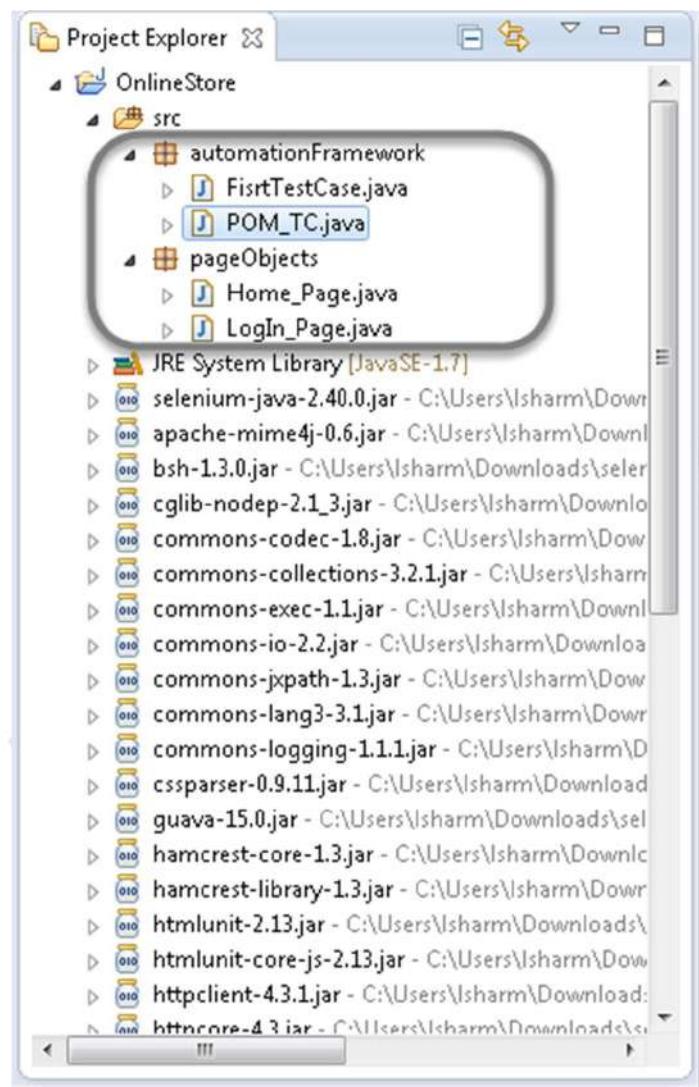
}

```

You will notice that once you type Home\_Page in your test script and the moment you press dot, all the methods in the Home Page will display. We can expose methods in order to reduce

duplicated code. We are able to call these method multiple times. This will ensure a better maintainable test code, because we only have to make adjustments and improvements in one particular place.

Your Project explorer window will look like this now.



## Summary

- Page Object Model is an Object Repository design pattern in Selenium WebDriver.
- Under this model, for each web page in the application, there should be corresponding page class.
- This Page class will find the WebElements of that web page and also contains Page methods which perform operations on those WebElements.
- Code becomes less and optimized because of the reusable page methods in the POM classes.
- Selenium Page Factory Pattern is like an extension to Page Object Model , but Page Factory is much enhanced model.
- Page Factory class can be used to make using Page Objects simpler and easier

DUCAT®  
[www.ducatindia.com](http://www.ducatindia.com)

# Chapter 20 –Miscellaneous

## Handling Cookies

### What Is An HTTP Cookie And How It Works?

HTTP Cookie is also called as a web cookie, a browser cookie or an Internet cookie. It is nothing but a text file present in the web browser of the client machine or PC that stores a small piece of information received from a website as a key-value pair when the user does the web browsing. When a user loads that website again, the browser sends back the stored cookie to the server that notifies it about the user's past activities. HTTP Cookies store not only the user's browsing history but also the shopping cart information for an online store, login user ids and passwords and other related information.

### Why Handle Cookies in Selenium?

Each cookie is associated with a name, value, domain, path, expiry, and the status of whether it is secure or not. In order to validate a client, a server parses all of these values in a cookie.

When testing a web application using selenium web driver, you may need to create, update or delete a cookie.

For example, when automating Online Shopping Application, you many need to automate test scenarios like place order, View Cart, Payment Information, order confirmation, etc.

If cookies are not stored, you will need to perform login action every time before you execute above listed test scenarios. This will increase your coding effort and execution time.

The solution is to store cookies in a File. Later, retrieve the values of cookie from this file and add to it your current browser session. As a result, you can skip the login steps in every Test Case because your driver session has this information in it.

The application server now treats your browser session as authenticated and directly takes you to your requested URL.

## Introducing Web Driver

The primary new feature in Selenium 2.0 is the integration of the WebDriver API. WebDriver is designed to provide a simpler, more concise programming interface in addition to addressing some limitations in the Selenium-RC API. Selenium-WebDriver was developed to better support dynamic web pages where elements of a page may change without the page itself being reloaded. WebDriver's goal is to supply a well-designed object-oriented API that provides improved support for modern advanced web-app testing problems

## Selenium WebDriver Query Commands For Cookies

Using Selenium WebDriver API, we can query and interact with browser cookies with the help of following WebDriver's built-in methods.

**Get Cookies:** This statement is used to return the list of all Cookies stored in web browser.

```
manage().getCookies();
```

**Get Cookies by name:** This statement is used to return the specific cookie according to its name.

```
manage().getCookieNamed(arg0);
```

**Add Cookies:** This statement is used to create and add the cookie.

```
manage().addCookie(arg0)
```

**Delete Cookies:** This statement is used to delete a specific cookie.

```
manage().deleteCookie(arg0);
```

**Delete Cookies by name:** This statement is used to delete a cookie according to its name.

```
manage().deleteCookieNamed(arg0);
```

**Delete All Cookies:** This statement is used to delete all cookies.

```
manage().deleteAllCookies();
```

**Example: Cookie handling in Selenium.**

We will use <http://demo.avactis.com> for our demo purpose.

This will be a 2 step process.

- Step 1) Login into application and store the authentication cookie generated.
- Step 2) Used the stored cookie, to again login into application without using userid and password.

Step 1) Storing cookie information.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileWriter;
import java.util.Set;
```

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

public class cookieRead{

 public static void main(String[] args)
 {
 WebDriver driver;
 System.setProperty("webdriver.chrome.driver","G://chromedriver.exe");
 driver=new ChromeDriver();
 driver.get("http://www.ducatindia.com/test/cookie/selenium_aut.php");
 // Input Email id and Password If you are already Register
 driver.findElement(By.name("username")).sendKeys("abc123");
 driver.findElement(By.name("password")).sendKeys("123xyz");
 driver.findElement(By.name("submit")).click();
 // create file named Cookies to store Login Information
 File file = new File("Cookies.data");
 try
 {
 // Delete old file if exists
 file.delete();
 file.createNewFile();
 }
 }
}
```

```

FileWriter fileWrite = new FileWriter(file);

BufferedWriter Bwrite = new BufferedWriter(fileWrite);

// loop for getting the cookie information

// loop for getting the cookie information
for(Cookie ck : driver.manage().getCookies())

{

Bwrite.write((ck.getName()+"."+ck.getValue()+"."+ck.getDomain()+"."+
ck.getPath()+"."+ck.getExpiry()+"."+ck.isSecure()));

Bwrite.newLine();

}

Bwrite.close();

fileWrite.close();

}

catch(Exception ex)

{

ex.printStackTrace();

}

}

}

```

### **Using stored cookie to login into the application.**

Now, we will access the cookie generated in step 1 and use the cookie generated to authenticate our session in the application

```

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.Date;
import java.util StringTokenizer;
import org.openqa.selenium.Cookie;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

```

```
public class CookieWrite
{
 public static void main(String[] args) {
 WebDriver driver;

 System.setProperty("webdriver.chrome.driver", "G://chromedriver.exe");
 driver=new ChromeDriver();
 try{
 File file = new File("Cookies.data");

 FileReader fileReader = new FileReader(file);

 BufferedReader Buffreader = new BufferedReader(fileReader);

 String strline;
 while((strline=Buffreader.readLine())!=null){

 StringTokenizer token = new StringTokenizer(strline,";");

 while(token.hasMoreTokens()){

 String name = token.nextToken();
 String value = token.nextToken();
 String domain = token.nextToken();
 String path = token.nextToken();
 Date expiry = null;

 String val;
 if(! (val=token.nextToken()).equals("null"))
 {
 expiry = new Date(val);
 }
 Boolean isSecure = new Boolean(token.nextToken()).
 booleanValue();
 }
 }
 }
 }
}
```

```

Cookie ck = new
Cookie(name,value,domain,path,expiry,isSecure);
System.out.println(ck);
driver.manage().addCookie(ck); // This will add the stored
cookie to your current session
}
}
}catch(Exception ex) {
ex.printStackTrace();
}

driver.get("http://www.ducatindia.com/test/cookie/selenium_aut.php")
;
}
}

```

**OUTPUT:** You are taken directly to the login success screen without entering the input user id and password

## Hash Testing Using Selenium

HashMap is a interface in java and can implements various classes in java like Hashmap, Hashtable & many other. They are the part of collections framework and are used to handle. store data. Map interface represent the mapping in between key and value. What that mean it, in Map or hashmap, value is stored in the form of Key & Value pair. In this article i will try to demonstrate the same.

We can use HashMap to with TestNG Dataprovider in Data Driven Testing. It can be Hash testing as we are using HashMap tp test data in selenium.

Here, we read the data from the excel/csv file using Hashmap. Please go through below code example to read data from the external source using hashmap/dictionary or data driven approach.

```

import java.io.File;
import java.io.IOException;
import java.util.HashMap;
import jxl.Cell;
import jxl.Sheet;
import jxl.Workbook;
import jxl.read.biff.BiffException;

class testclass {

```

```

static String[][] getexceldatainArray;
static HashMap<String, String> dictionary;

public static void main(String[] args) throws IOException,
BiffException {
 // Create an array based on data in excel file
 excelarrayclass tc = new excelarrayclass();
 getexceldatainArray =
tc.CreateArrayfromExcel("D:\\Testing.xls",
 "testing");
 for (int i = 0; i < getexceldatainArray.length; i++) {
 dictionary = DataDictionaryFromArray(getexceldatainArray,
i);
 // the keys will be value in the header column
 String username = dictionary.get("UserName");
 /*
 * Perform the required operation using the dictionary
values
 */
 System.out.println("UserName " + username);
 }
}

public String[][] CreateArrayfromExcel(String WorkbookName,
String SheetName)
throws BiffException, IOException {
String[][] arrTestDta;

Workbook workbk = Workbook.getWorkbook(new
File(WorkbookName));
Sheet wrkSheet = workbk.getSheet(SheetName);
arrTestDta = new
String[wrkSheet.getRows()][wrkSheet.getColumns()];

// Loop to get data from excel file into an array

```

```

for (int i = 0; i < wrkSheet.getRows(); i++) {
 for (int j = 0; j < wrkSheet.getColumns(); j++) {
 Cell cell = wrkSheet.getCell(j, i);
 if (cell.getContents() != null) {
 arrTestDta[i][j] = cell.getContents();
 }
 }
}
return arrTestDta;
}

public static HashMap<String, String>
DataDictionaryFromArray(
 String[][] arrayforDictionary, int iRow) {
 // Create a hashmap to store the data as dictionary in key
 // value form
 HashMap<String, String> dictionary = new HashMap<String,
 String>();
 // get the size of the array, assuming being used in data
 // driven test,
 // get the number of columns in the data file
 int iColCnt = arrayforDictionary[0].length;
 for (int i = 0; i < iColCnt; i++) {
 // mark the header column values as key and current column
 // value as
 // key value pair
 dictionary.put(arrayforDictionary[0][i],
 arrayforDictionary[iRow][i]);
 }
 return dictionary;
}
}

```

## Screenshot Capturing

While running the automation test suites using Selenium WebDriver, if defects are found then they are momentary and does not explain much of the details. In such case, it is desirable to take

the screenshot which will capture all of the details of the defect. This helps the developer in bug analysis. Selenium WebDriver can take screenshots automatically with the help of class TakesScreenshot that is present in Selenium WebDriver API. In a similar way, we can generate PDF files as well.

Following is the demonstration of a test script in Java to take test screenshot of a website using WebDriver API.



```

1 package testingproject;
2
3 import java.io.File;
4 import org.apache.commons.io.FileUtils;
5 import org.openqa.selenium.OutputType;
6 import org.openqa.selenium.TakesScreenshot;
7 import org.openqa.selenium.WebDriver;
8 import org.openqa.selenium.firefox.FirefoxDriver;
9 import org.testng.annotations.Test;
10
11 public class TestScreenshot {
12 @Test
13 public void testDemo00TakeScreenShot() throws Exception{
14 WebDriver driver = new FirefoxDriver();
15 driver.get("http://www.softwaretestingclass.com/");
16 //Call takesScreenshot function
17 this.takeSnapshot(driver, "c://selenium_demo//screenshot.png");
18 }
19 public void takeSnapshot(WebDriver webDriver, String fileLocalPath) throws Exception{
20 //Convert web driver object to TakesScreenshot
21 TakesScreenshot screenShot = ((TakesScreenshot) webDriver);
22 //Call getScreenshotAs method to create image file
23 File sourceFile=screenShot.getScreenshotAs(OutputType.FILE);
24 //Move image file to new destination
25 File DestinationFile=new File(fileLocalPath);
26 //Copy file as destination
27 FileUtils.copyFile(sourceFile, DestinationFile);
28 }
29 }
30
31

```

## Explanation of the test script

In the test script, the first and foremost step is to instantiate the Firefox WebDriver. After instantiation, such an object has the reference as 'driver'. With the help of this driver, object invoke and load the website URL. Here the URL under test is <https://www.softwaretestingclass.com/>. Here a method *TakesSnapShot* has created in the same class that accepts two arguments WebDriver instance and file path.

Inside this method, web driver object is converting to *TakesScreenshot* with the following statement.

```
TakesScreenshot screenShot = ((TakesScreenshot) webdriver);
```

We are using the *getScreenshotAs* method of converted driver object in the last step. This accepts a single argument which is given as type FILE as shown below. It returns a file object.

```
File SourceFile=screenShot.getScreenshotAs(OutputType.FILE);
```

In the next step, we are instantiating a file that accepts the local file path (here `c://selenium_demo//screenshot.png`). This file object we are passing as an argument along with the file object returned in the last step to copy the screenshot file as shown below.

```
File DestinationFile=new File(fileLocalPath);
FileUtils.copyFile(SourceFile, DestinationFile);
```

## Output

When the above test script will run, it will capture a full screenshot of the website home page located at URL <https://www.softwaretestingclass.com/> and will copy that screenshot PNG file to local drive (Path as "`c://selenium_demo//screenshot.png`"). Shown below is the captured screenshot.



## Screen Reading Through Selenium

Screen reading is also known as accessibility assessment automation. Accessibility assessment automation is a piece of code that can evaluate the web controls on a web page for accessibility. Accessibility assessment automation tools are useful because they can save a significant amount of time. The larger the site or the more complex the layout, the higher the chance of missing errors by manual testing. So accessibility automation can be used as an excellent way to generate a list of accessibility issues. There are various errors that can be caught by automation tools and do not need human inspection. A good example is in a case where images do not have any alt text, an accessibility assessment is an efficient way to document each instance one by one when an automatic means of finding is available.

### Tools used in accessibility assessment automation

- Selenium-WEBDRIVER

Selenium is a popular open-source web-based automation tool. It provides a more stable approach to automating the browser. More about Selenium can be found on the Selenium website:

[http://docs.seleniumhq.org/docs/03\\_webdriver.jsp](http://docs.seleniumhq.org/docs/03_webdriver.jsp). We used Selenium

WebDriver to login to the application and to invoke the HTML\_Code Sniffer on Web Page.

- Testing Framework

TestNG is a testing framework that is used with Selenium WebDriver. We used the TestNG framework for different types of annotations and to generate various type of reports (e.g. test output report)

- Java

Java is used as a scripting language. It is used with Selenium WebDriver to login to the application and inspect the web controls on the web page.

- Eclipse

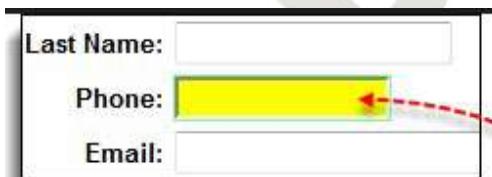
We used Eclipse IDE (an editor) to write the code using the scripting language Java.

- HTML Codesniffer

The backbone of our accessibility assessment automation process. You can simply drag it from '[http://squizlabs.github.io/HTML\\_CodeSniffer/](http://squizlabs.github.io/HTML_CodeSniffer/)' and it will be added to your Bookmark. HTML\_CodeSniffer is a client-side JavaScript that can check HTML code and detect WCAG 2.0 A/AA/AAA or Section 508 violations. For more information, please visit [HTML\\_CodeSniffer](#). We did not invoke it from the bookmark; instead, we called the HTML Code\_Sniffer JavaScript in Selenium code.

## Highlighting Elements through JavaScript in Selenium

In Selenium, we can use JavascriptExecutor (interface) to execute Javascript code into webdriver.



Here, we will execute Javascript which will highlight the element

Let's implement the same this will highlight the user name field.

```
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
```

```

public class Highlight {

 public static void main(String []args) {

 WebDriver driver=new FirefoxDriver();

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

 driver.get("http://www.facebook.com");

 // Create the JavascriptExecutor object
 JavascriptExecutor js=(JavascriptExecutor)driver;

 // find element using id attribute
 WebElement username= driver.findElement(By.id("email"));

 // call the executeScript method
 js.executeScript("arguments[0].setAttribute('style','border:
solid 2px red');", username);
 }

}

```

In above program it, will highlight username field. Now we can not write every time the same code to highlight element so we will create reuse method and when we have to highlight element we will call the method.

### **Create highlight element method for reuse**

```

public static void highLightElement(WebDriver driver,
WebElement element)

{
 JavascriptExecutor js=(JavascriptExecutor)driver;

 js.executeScript("arguments[0].setAttribute('style',
'background: yellow; border: 2px solid red;')", element);

```

```
try
{
 Thread.sleep(500);
}

catch (InterruptedException e) {

 System.out.println(e.getMessage());
}

js.executeScript("arguments[0].setAttribute('style','border:
solid 2px white');", element);
```

## Katalon Studio

Katalon Studio is a free automation testing solution developed by Katalon LLC. The software is built on top of open-source automation frameworks Selenium, Appium with a specialized IDE interface for API, Web and Mobile testing .

Katalon Studio is a comprehensive toolset for web and mobile app automation testing. This tool includes a full package of powerful features that help overcome common challenges in web UI test automation, for example, pop-up, iFrame, and wait-time. This user-friendly and versatile solution help tester test better, work faster, and launch high quality software thank to the intelligence it provides to the entire test automation process.

### Features

- The test automation framework provided within Katalon Studio was developed with the keyword-driven approach as the primary test authoring method with data-driven functionality for test execution.
- The user interface is a complete integrated development environment (IDE) implemented on Eclipse rich client platform (RCP).
- The keyword libraries are a composition of common actions for web, API, and mobile testings. External libraries written in Java can be imported into a project to be used as native functions.
- The main scripting language is Groovy, Java, and JavaScript and can be executed against all modern browsers, iOS, and Android applications supported by Selenium and Appium.
- Katalon Studio monthly release includes the up-to-date open-source drivers matching the latest web and mobile environments to reduce the project maintenance cost and configuration efforts.

## Summary

- Cookie is a text file present in the web browser of the client machine or PC that stores a small piece of information
- HashMap is a interface used in Hash testing.
- Selenium WebDriver can take screenshots automatically with the help of class TakesScreenshot that is present in Selenium WebDriver API
- Screen reading automation is a piece of code that can evaluate the web controls on a web page for accessibility.
- JavascriptExecutor interface can be used to execute Javascript code into webdriver and to highlight element.
- Katalon Studio is a comprehensive toolset for web and mobile app automation testing.

DUCAT®  
[www.ducatindia.com](http://www.ducatindia.com)