Steps involved in installing TestNG:

1. Open eclipse, go to help and click on eclipsemarketplace.
2. In the find textbox, type TestNG and click on search.
3. In the 'TestNG for eclipse' section, click on install.
4. After waiting for sometime, TestNG for eclipse checkbox will be displayed, enable the checkbox and then click on confirm.
5. Click on 'I accept the terms' radio button and click on 'finish.'

TestNG Framework (Test Next Generation)

1. It is a framework which is used to test the software under test and obtain report.

Steps involved in downloading TestNG Jars

1. Open the browser and browse for MVNrepository
2. Click on the first link
3. In the search box, type testng and click on search.
4. Click on TestNG link
5. Click on 6.14.3 link
6. Click on jar link (820KB)
7. Place the downloaded jar file in the selenium components folder.

Steps involved in creating the project and adding the testNG jars.

1. Go to eclipse and create a new Java Project
2. Right click on the newly created java project. Go to build path and click on configure buildpath.
3. Click on add external jars button.
4. Select the TestNG jarfile and click on open.
5. Click on Apply and Close
6. Add selenium jars and driver softwares.

Program:

```
public class MagentoDemo
{
 @Test
 public void positiveCredential2()
 {
  ChromeDriver driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.manage().timeOuts().implicitlyWait(20, TimeUnit.seconds);
  driver.get("https://www.magento.com");
  driver.findElement(By.linkText("My Account")).click();
  driver.findElement(By.id("email")).sendKeys("sucheadra.abc@gmail.com");
  driver.findElement(By.id("pass")).sendKeys("welcome123");
  driver.findElement(By.id("send2")).click();
  driver.findElement(By.linkText("Log Out")).click();
 }
 public void positiveCredential1()
 {
  ChromeDriver driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.manage().timeOuts().implicitlyWait(20, TimeUnit.seconds);
  driver.get("https://www.magento.com");
  driver.findElement(By.linkText("My Account")).click();
```

```
driver.findElement(By.id("email")).sendKeys(
driver.findElement(By.id("pass")).sendKeys("Welcome123");
driver.findElement(By.id("send2")).click();
driver.findElement(By.LinkText("Log Out")).click();
}
}
}
```

Note: A testNG class consists of tests methods

→ Any Method which is annotated using @test is referred to as a test method.

→ A testNG class can contain any number of test methods

→ test method is also referred to as a test case.

→ When multiple test() are present, ascii value of the method name is compared. the one with less ascii value, is executed first.

reports generated in testNG

1. Once the testNG class is executed, select project, right click and click on refresh. Once we refresh the project, one folder by name 'testOutput' would appear.

→ Inside the testOutput folder we will have two types of reports.

(i) index.html

(ii) emailablereport.html

@BeforeMethod & @AfterMethod

@BeforeMethod:

→ Such methods which are annotated using @BefreMethod, they would get executed before the execution of each testcase or test method

@AfterMethod : Such methods which are annotated using @AfterMethod, they will get excecuted after the excecution of each test case or test method.

```
BeforeMethod
@Test
public void positiveCredential1() {  ——→ TestMethod
}
Aftermethod
Before method
@Test
public void positiveCredential2() {  ——→ TestMethod
}
AfterMethod
```

```
public class Demo{
  @BeforeMethod
  public void beforeMethod(){
    System.out.println("Before method executed");
  }
  @AfterMethod
  public void afterMethod(){
    System.out.println("After method executed");
  }
  @Test
  public void positiveCredential1(){
    System.out.println("Inside positive Credential1");
  }
```

```
@Test
public void positiveCredential2() {
    System.out.println("Inside positive Credential2");
}
}
```

output:

```
Before method executed
inside positive Credential1
after method executed
Before method executed
inside positive Credential2
after method executed
```

Program:

```
public class BeforeAfterMethodDemo
{
ChromeDriver driver;
@BeforeMethod
public void openBrowser()
{
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.manage().timeouts().implicitlyWait(20,TimeUnit.SECONDS);
  driver.get("https://www.magento.com");
}
@AfterMethod
public void terminate()
{
  driver.quit();
}
@Test
public void loginValidation()
{
  driver.findElement(By.linkText("My Account")).click();
  driver.findElement(By.id("email")).sendKeys("suchandra.abc
                                              @gmail.com");
  driver.findElement(By.id("pass")).sendKeys("Welcome123");
  driver.findElement(By.id("send2")).click();
  driver.findElement(By.linkText("Log out")).click();
}
@Test
public void clickRegister(){
    driver.findElement(By.linkText("My Account")).click();
    driver.findElement(By.xpath("//span[text()='Register']")).
                                              click();
}
}
```

@BeforeClass & @AfterClass

@BeforeClass : Such methods which are created using @BeforeClass would get executed only once before the execution of before() of the current class.

@AfterClass : Such methods which are created using @AfterClass would get executed only once after the execution of after() in the current class.

program:

```
public class Demo{

  @BeforeClass
  public void beforeClass(){
    s.o.p("Before class executed");
  }
@AfterClass
public void afterClass()
{
  system.out.println("After class executed");
}
@BeforeMethod
public void beforeMethod()
{
  s.o.p("Before method executed");
}
@AfterMethod
public void afterMethod()
{
  s.o.p("After method executed");
}
@Test
public void positiveCredential1()
{
  s.o.p("Inside positive credential 1");
}
@Test
public void positiveCredential2()
{
  s.o.p("Inside positive credential 2");
}
```

Output:

```
BeforeClass executed
Before method executed
Inside positive credential 1
After Method executed
Before Method executed
Inside positive Credential 2
After method executed
After Class executed.
```

Priorities in the TestNG

The order of execution of the test methods or test cases inside the TestNG class is based on the alphabetical order or the ascii values. However we can change the order of execution of the test cases depending on the requirements using the concept of priority.

Syntax: Test(priority = priority_num)

Test method without priority

program:

```
public class Demo {

@Test
public void a_method(){
  s.o.p("a-method executed");
}
```

```
@Test
public void b_method()
{
  s.o.p("b_method executed");
}
@Test
public void c_method()
{
  s.o.p("c_method executed");
}
@Test
public void d_method()
{
  s.o.p("d_method executed");
}
@Test
public void e_method()
{
  s.o.p("e_method executed");
}
}
```

output

```
a_method executed
b_method executed
c_method executed
d_method executed
e_method executed
```

with priority :

```
class Demo{
@Test(priority = 33)
public void a_method()
{ s.o.p("a_method executed");
}
@Test(priority = 0)
public void b_method()
{ s.o.p("b_method executed");
}
@Test(priority = 42)
public void c_method()
{ s.o.p("c_method executed");
}
@Test(priority = 6)
public void d_method()
{
  s.o.p("d_method executed");
}
@Test(priority = 99)
public void e_method()
{
  s.o.p("e_method executed");
}
}
```

O/P:

```
b_method executed
d_method executed
a_method executed
c_method executed
e_method executed
```

Multiple test methods with the same priority

When multiple test methods have same priority then the test case execution would be based on the alphabetical order or the ascii values.

Ex:-
```
public class Demo{

    @Test(priority=6)
    public void a_method()
    {
      s.o.p("a-method executed");
    }
    @Test(priority=2)
    public void b_method()
    {
      s.o.p("b_method executed");
    }
    @Test(priority=2)
    public void b_method()
    {
      s.o.p("b_method executed");
    }
```

O/P:

b_method executed
c_method executed
a_method executed

test methods with priority and without priority

If a testNG class contains the test methods with priority and the test methods without priority then during execution, then highest priority would be given to such test(), without priority.

Ex:-
```
public class Demo{

    @Test(priority=8)
    public void a_method()
    {
      s.o.p("a_method executed");
    }
    @Test
    public void b_method()
    {
      s.o.p("b_method executed");
    }
    @Test(priority=6)
    public void c_method()
    {
      s.o.p("c_method executed");
    }
    @Test
    public void d_method()
    {
      s.o.p("d_method executed");
    }
```

O/P:

b_method executed
d_method executed
c_method executed
a_method executed

TestNG Suite

Steps involved in creating the ~~xml~~ testNG.xml file

1. After creating the testNG class, select the class, right click, go to testNG and click on convert to testng.
2. Select the name for the xml file and click on finish.

3. program:

```
package com.abc.demo;
import org.testng.annotations.Test;
public class Demo5
{
 @Test
 public void positiveCredential1()
 {
  System.out.println("inside positive credential1");
 }
 @Test
 public void positiveCredential2()
 {
  s.o.p ("inside positive credential2");
 }
}
```

xml code

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
<test thread-count="5" name="Test">
<classes>
 <class name="com.abc.demo.Demo5"/>
</classes>
</test><!--Test-->
</suite><!--Suite-->
```

xml structure

```
<suite>
    <test>
        <classes>
            <class>....</class>
            :
        </classes>
        :
    </test>
    :
</suite>
```

include and exclude

include: is used to specify within the testNG class whatall testcases should be included in the execution.

exclude: It is used to exclude a particular test case from getting executed.

Ex:- program

```
public class Demo{
{
 @Test
 public void testCase1(){
   s.o.p("testCase1 executed");
 }
```

```
@Test
public void testCase2()
{ s.o.p ("testCase2 executed");
}
@Test
public void testCase3()
{
 s.o.p("testCase3 executed");
}
```

xml code

```
<classes>
  <class name = "com.abc.demo.Demo4" >
   <methods>
      <include name = "testCase2"></include>
   </methods>
  </class>
```

o/p:

testcase2 executed

xml code:

```
<classes>
   <class name = "com.abc.demo.Demo4">
   <methods>
      <exclude name = "testCase3" ></exclude>
   </methods>
 </class>
```

o/p:

testCase1 executed

Groups

program:

```
public class DemoGroup
{
 @Test (groups = "inbox")
  public void testCase1()
  { s.o.p("Test Case 1 executed);
  }
 @Test (groups = "compose").
 public void testCase2()
  { s.o.p (" TestCase 2 executed");
  }
@Test (groups = "compose").
public void testCase3()
  { s.o.p ("TestCase3 executed");
  }
@Test (groups = "logout").
public void testCase4()
  { s.o.p (" TestCase 4 executed");
  }
@Test (groups = "inbox")
public void testCase5()
  { s.o.p ("testCase 5 executed");
  }
}
```

xml code

```
<test thread-count ="5" name = "Test">
<groups>
  <run>
    <include name = "inbox"></include>
  </run>
</groups>
<classes>
    <class name = "com.abc.demoGroup"/>
</classes>
```

output:

testCase1 executed
testCase5 executed

@BeforeTest & @AfterTest

@BeforeTest & @AfterTest is not related to the test cases or test methods, rather it is related to the test tag present in the xml. Such methods which are annotated using BeforeTest annotation would get executed, before the execution of each test tag in the xml. Such test methods which are annotated using AfterTest annotation would get executed, after the execution of all the methods in each test class.

program:

```
public class Demo5
{
 @Test
 public void positiveCredential1()
 { s.o.p("positive credential 1 executed");
 }
 @Test
 public void positiveCredential2()
 { s.o.p ("positive credential 2 executed");
 }
 @BeforeMethod
 public void beforeMethod()
 { s.o.p ("before method executed");
 }
 @AfterMethod
 public void afteryMethod()
 { s.o.p ("after method executed");
 }
 @BeforeClass
 public void beforeClass()
 { s.o.p ("before class executed");
 }
 @AfterClass
 public void afterClass()
 { s.o.p ("after class executed");
 }
```

```
@BeforeTest
public void beforeTest()
{
  s.o.p("before test executed");
}
@AfterTest
public void afterTest()
{
  s.o.p("after test executed");
}
}
```

xml

```
<suite name="Suite">
  <test thread-count="5" name="login">
  <classes>
     <class name="com.abc.demo.Demo5"/>
  </classes>
 </test><!--...Test-->
 </suite><!--...Suite-->
```

Note:

BeforeSuite
<suite>
BeforeTest
<test>
BeforeClass
<classes>
BeforeMethod
Testcases
AfterMethod
</classes>
AfterClass
</test>
AfterTest
</suite>
AfterSuite

Various attributes associated with the test annotation

dependsOnMethods
Assert
AlwaysRun
Enabled
InvocationCount

dependsOnMethods : Whenever we have to make a testcase dependent on another testcase, then we should be using dependsOnMethods.

Assert.fail : when we want to intentionally fail a testcase

AlwaysRun : Whenever we want to execute a testcase irrespective of whether the dependent testcase is pass or fail, then we should take the help of AlwaysRun = true

Enabled : Whenever we have to hide a testcase, we should use Enabled = false

InvocationCount : Whenever we want to invoke a test case multiple number of times, we should use the invocationCount

Program:

```
public class Demo6 {
  @Test
```

```
public void login()
{
  //Assert.fail();
  Assert.assertEquals(true, False);
  s.o.p("login succesful");
}
@Test(dependsOnMethods = "login", alwaysRun = true)
public void logout(){
  s.o.p("logout succesful");
}
@Test(enabled = false)
public void inbox()
{ s.o.p("inbox testcase executed");
}
@Test(invocationCount = 5)
public void compose()
{
  s.o.p("Compose test case executed");
 }
}
```

Parametarization

→ In case parameterization, the data would be stored in the xml file. From the xml file, we should fetch the data and we should use it in the test code and we should be executing the test code on the s/w under test.

→ The parameters can be specified within the test level. If the parameters are present within the test, then the parameters would be available only for that particular test. If the parameters have to be available for all the tests, then place the parameters at the suite level.

```
public class ParameterDemo
{@Test
@Parameters({"url","email","password"})
public void positiveCredential(String url,String email,String pw)
{ s.o.p(url);
  s.o.p(email);
  s.o.p(pw);
ChromeDriver driver = new ChromeDriver();
driver.manage().window().maximize();
driver.manage().timeouts().implicitlyWait(20,TimeUnit.SECONDS);
driver.get(url);
driver.findElement(By.LinkText("My Account")).click();
driver.findElement(By.Id("email")).sendKeys(email);
```

```
driver.findElement(By.id("pass")).sendKeys(pw);
driver.findElement(By.id("send2")).click();
driver.findElement(By.linkText("logOut")).click();
driver.quit();
}
}
```

Xml:

```
<suite name="Suite">
    <parameter name="url" value="https://www.magento.com"></parameter>
    <parameter name="email" value="sucheendra.abc@gmail.com"></parameter>
    <parameter name="password" value="Welcome123"></parameter>
  <test thread-count="5" name="Inbox">
    <classes>
      <class name="com.abc.demo.ParameterDemo"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

Data Providers

invocation count

new Object[2][2]

Total no. of parameters

```
public class DataProvidersDemo
{
@DataProvider(name = "authentication")
public Object[][] dataProvider() {
    Object[][] obj = new Object[2][2];
    obj[0][0] = "sucheendra.abc@gmail.com";
    obj[0][1] = "Welcome123";
    obj[1][0] = "nithinmanjunath1991@gmail.com";
    obj[1][1] = "Welcome123";
    return obj;
}
@Test(dataProvider = "authentication")
public void positiveCredential(String email, String pw) {
    ChromeDriver driver = new ChromeDriver();
    driver.manage().window().maximize();
    driver.manage().timeouts().implicitlyWait(20, TimeUnit.seconds);
    driver.get("https://www.magento.com");
    driver.findElement(By.linkText("My Account")).click();
    driver.findElement(By.id("email")).sendKeys(email);
    driver.findElement(By.id("pass")).sendKeys(pw);
    driver.findElement(By.id("send2")).click();
    driver.findElement(By.linkText("logOut")).click();
    driver.quit();
}
}
```

Cross Browser Testing

Whenever the slw under test has to be tested on multiple browsers parallelly, we use the concept of cross browser testing.

```
public class CrossBrowserDemo
{
  WebDriver driver;
  @BeforeTest
  @Parameters("browser")
  public void init(String browser)
  {
    if(browser.equals("chrome"))
    {
      driver = new ChromeDriver();
    }
    else if(browser.equals("firefox"))
    {
      driver = new FirefoxDriver();
    }
  }
  @Test
```

xml

```
<suite name="Suite" parallel="tests">
  <test thread-count="5" name="chrome">
  <parameter name="browser" value="chrome"></parameter>
  <classes>
   <class name="com.abc.demo.CrossBrowserDemo"/>
  </classes>
  </test>
  <test thread-count="5" name="firefox">
  <parameter name="browser" value="firefox"></parameter>
  <classes>
    <class name="com.abc.demo.CrossBrowserDemo"/>
  </classes>
  </test>
  </suite>
```

MAVEN

Maven is a project management tool which is used to download the jar files automatically to the project.

Steps involved in downloading MAVEN:

1. Open the browser and search for Apache Maven
2. Click on the first link
3. Click on download link
4. Scroll down to files section and click on apache-maven-3.6.0-bin.zip
5. Extract the zipfile and place the Apachemaven in the selenium components folder.
6. Select My PC, right click, click on properties, click on advanced system settings and click on environment variables.
7. Under System variables, create one variable by name M2_HOME and selec set the value with the path of the apache maven home directory.
8. D:\ABC\July2\seleniumcomponents\apache-maven-3.6.0-bin\apache-maven-3.6.0
9. Create a new variables under settings MAVEN_HOME and select the path of MAVEN_Home Directory and set it as a value.
9. Select Path variable under System variables and paste the following path.

D:\ABC\July2\selenium components\apache-maven-3.6.0-...bin

10. To check whether MAVEN is configured or not, go to cmd prompt & type mvn -v

Maven project consists of 3 phases

(1) Creation of the project
(2) Adding the dependencies
(3) Executing the project

Creation of the maven project

1. Go to eclipse, click on file, select new and click on project
2. Search for maven project. Click on Maven Project
3. Enable the check box, create a simple project and click on next
4. Under the group id, give any package name (com.abc.magento)
5. Under the artifact id, give the name of project (MagentoTesting) and click on finish
6. From the pageObjectModel project, copy all the page objects i.e loginPage.java, mainPage.java, welcomePage.java
7. Go to the Maven project, select src/main/java folder

and paste it

8. From the pageObjectModel object, copy magentoTest which contains the test case and paste it in src/test/java folder

Adding the dependencies

1. Open any browser & search for mvn repository.
2. Click on the first link. In the search textbox, type selenium and click on search button.
3. Click on the first link (Selenium Java)
4. Click on 3.14.0
5. Under maven section, copy the code present and paste it in pom.xml file (project object model)
6. Inside the pom.xml file, create <dependencies> tag, within <dependies> tag, paste the code which is copied from mvn repository.
7. Again from the mvn repository, search for testNG
8. Click on the first link (testNG). Click on 6.14.3 link.
9. From the maven section, copy the dependency and paste the dependency in pom.xml file.
10. Once adding all the dependencies, select the project and then go to maven and click on update project.

Execution of the MAVEN Project

1. Select the maven project, right click, go to run as and select MAVEN test

MagentoTest.xml

```
<version>0.0.1-SNAPSHOT</version>
<dependencies>
  <!... https://mvnrepository.com/artifact/org.selenium hq.selenium/selenium-java ->
  <dependency>
    <groupId>org.seleniumhq.org.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.14.0</version>
  </dependency>
  <!... https://mvnrepository.com/artifact/org.testng/testng ->
  <dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>6.14.3</version>
    <scope>test</scope>
  </dependency>
```

MagentoTest.java

```
public class MagentoTest {
  @Test
```

```
public void positiveCredential1(){
}
{
FirefoxDriver driver = new FirefoxDriver();
driver.manage().window().maximize();
driver.manage().timeouts().implicitlyWait(20,TimeUnit.
                                          seconds
driver.get("https://www.magento.com");
Welcome w = new Welcome(driver);
w.clickOnMyAcct();
login l = new login(driver);
l.typeEmail("nitinmanjunath1991@gmail.com");
l.typePass("Welcome123");
l.clickOnLogin();
Main m = new Main(driver);
m.clickOnLogout();
driver.quit();
}
}
```

Version control tool or source management tool or revision control tool

⇒ Version control tools are used to manage the code which is developed by various developers in a single repository. It can also be used to monitor the revisions that are done to the code

⇒ The VCT that we are using is Github

Steps involved in downloading Git

1. Go to google and type git
2. Click on the first link, click on 'download 2.19.2 for windows'
3. Place the downloaded file in the selenium Component folder

Installing git

1. Select git.exe file, right click and select run as administrator.
2. Perform the installation by following the steps given in the software.

Steps involved in creating github account

1. Goto www.github.com
2. Click on signup button
3. Enter the username, email address
4. Set Enter the password
5. Solve the puzzle, given in the verify account section
6. Click on create an account button
7. Click on continue button, click on skip this step link
8. Click on start a project button

creating a new repository in Github and uploading the project from Eclipse

→ In github, after email is verified, click on new repository button (MagentoTesting)
→ select a name on repository and click on create repository button.
→ Go to eclipse and in the 'Quick Access' textbox, type git repositories and select Git repositories
→ In the Git repositories, click on 'clone a Git repository' option
→ Copy the link present in (url of Git hut repository) and paste it in url section present in source Git repository dialogue box and click on next, again click on next in the branch selection dialogue box.
→ In the local destination dialogue box, select the path of local repository. Click on finish
→ Select the project, right click, go to Team and click on ShareProject Option
→ In the configure git repository dialogue box, select the path of local repository. (C:\Users\Sucheendra\git\MagentoTesting\.git and click on finish.
→ Select the project, right click, go to Team and click on commit.
→ Move all the files which has to be uploaded to the github repository from unstaged changes combo box to staged changes combo box.
→ Enter a suitable commit msg and click on commit and push, click on next and enter the github user name & password whenever required.

Jenkins

Jenkins is a tool which is used to install and execute the build automatically.

Q) Different phases in Jenkins:

(i) Download Jenkins
(ii) Configure Jenkins
(iii) Install & Execute the build

(i) Downloading Jenkins:

1) Open any browser & search for jenkins.io
2) Click on documentation button
3) Click on getting started link
4) Click on the link download jenkins
5) Place the downloaded file in the selenium components folder
6) Whereever jenkins.war file is present, open up the command prompt & paste the following link

java -jar jenkins.war --httpPort=8080

7) Once jenkins is up and running msg is displayed minimize the cmd prompt, open the browser and navigate to the following link.

http://localhost:8080

8) After navigating to the above link, find the admin password from the path given by jenkins & paste it in administrator password text field & click on continue.
9) Click on select plugins to install button, click on install button.
10) In the top left corner, click on the jenkins dropdown. Go to manage jenkins and click on manage plugins.
11) Click on available tab & search for unlish maven.
12) Enable the checkbox unlish maven, and click on the button, install without restart.
13) Click on (In the jenkins dashboard) manage jenkins
14) Click on Global Tool Configuration
15) Under jdk section, click on the button jdk installation
16) If install automatically checkbox is enabled, disable it.
17) Give the name of jdk as JAVA_HOME and for it give the path as C:\Program files\Java\jdk1.8.0_171
18) Under git section, disable install automatically checkbox and give the path of git.exe as follows

C:\Program files\Git\bin\git.exe

19) Go to maven section and click on maven installation button, uncheck install automatically checkbox and for the name variable, set it as maven_home and for

for maven_home select path of apache-maven-directory

20) Click on save button

21) Go back to dashboard and click on the link new item

22) Enter the item name as MagentoTesting, after ente the item name, click on maven project and click ok.

23) Enable the checkbox, github project and paste the link of github repository where the project is present.

24) Under the source code management section, click on git radio button, paste the url of the github repository where the project is present.

25) Under the build triggers section uncheck the checkbox build whenever and check the poltscm checkbox and in the schedule textbox type

H/2 *****

H/2 **** → every 2 min automatically project will be bui

H/1 **** → every 1 hour

26) Under the build section, for root pom, give the path of pom file of the project from local repository and in the goals type test as the goal

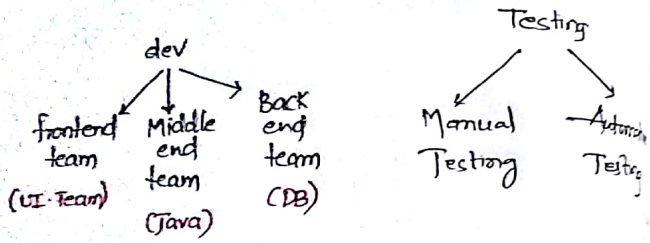
27) Click on save

28) Click on the link build now

LeanFT

LeanFT is a powerful functioning tool which is used to perform functional testing in Ajile & devops methodology

LeanFT - LeanFunctional Testing



Installation of LeanFT

1. Before installing LeanFT, first we should install Node.js
2. From the LeanFT folder, double click on setup.exe file, click on LeanFT Setup, click on ok.
3. Once visiual c++ is installed, click on next. Accept the terms and click on next, click on next.
4. Click on full installation radio button, check the eclipse checkbox and provide the path of eclipse directory
5. Click on install. Click on Finish.
6. Find the gson jar from the LeanFT folder given & paste it in plgins folder of eclipse.

Automate the following scenario

1. Open the chromebrowser
2. Navigate to magento Appln
3. Click on myaccount
4. Enter email, password, and click on login.
5. Click on logout
6. Close the browser.

steps involved in creation of leanFT project

1. In eclipse, create a new workspace specifically for writing the leanFT programs
2. Right Click on project explorer. Go to new & click on project
3. search for LeanFT JUnit Project & double click on it.
4. Give any name to the project (MagentoAutomation).
5. Click on finish.

(1)

Code:

```
@Test
public void test() throws GeneralLeanFTException{
 Browser browser= BrowserFactory.Launch(BrowserType.Chrome);
 WebElement myacct= ..... .....
 myacct.click();
 EditField email= ..... .....
 email.setValue("sucheendra.abc@gmail.com");
 EditField pass = ..... .....
 pass.setValue("Welcome123");
 Button login = ..... .....
 login.click();
```

```
Link logout = .....
logout.click();
browser.close();
}
}
```

(2)

ASSIGNMENT

* Opent the chrome browser, navigate to facebook. Enter the first name, surname, mobile number, new password and click on signout button.

PROGRAM-2

Automating tables using LeanFT

Manual Testing	200
Selenium	400
LeanFT	600

Html:

```
<html>
<body>
  <table id="tab" name="tab" border="1">
    <tbody>
      <tr>
        <td> ManualTesting </td>
        <td> 200 </td>
      </tr>
      <tr>
        <td>Selenium </td>
        <td>400</td>
      </tr>
      <tr>
        <td> LeanFT</td>
        <td>600</td>
      </tr>
    </tbody>
  </table>
</body>
</html>
```

(3)

Program:

```
@Test
public void test() throws GeneralLeanFTException
{
Browser browser = BrowserFactory.Launch(BrowserType.INTERNET_EXPLORER);
browser.navigate("D:\\ABC\\July2\\WebPages\\LeanFT\\Table.html");
Table table = browser.describe(Table.class, new TableDescription
                .Builder().tag("Name").build());
for (TableRow row : table.getRows())
{
  for (TableCell cell: row.getCells())
  {
    S.O.P(cell.getText());
  }
}
```

Automating Notepad Application:

1. Open the Notepad Application.
2. From the Menu, select "format" & click on "font".
3. One dialogue box would be open, highlight all the combo boxes present in the dialogue box & also print the text associated with the dialogue boxes.
4. From the font combo box, select 'Arial' font. Once Arial font is selected, click on 'OK'
5. In the notepad, type 'Welcome to ABC'.

(4)

Program:

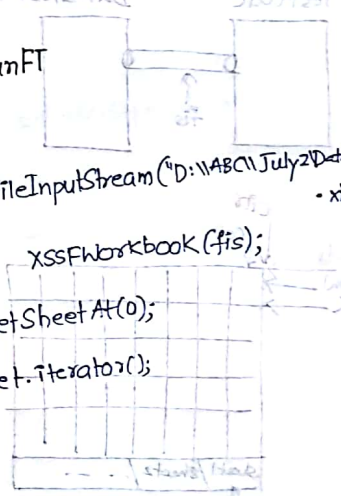
First create a new project.

```
@Test
public void test() throws GeneralLeanException, IOException
{
//to open notepad
ProcessBuilder pb = new ProcessBuilder("C:\\Windows\\notepad.exe");
pb.start();
//select notepad
Window fullNotepad = ....
// selecting format → font menu (Format → fnt)
Menu menu = ....
String path = menu.buildMenuPath("Format", 2); //access font
MenuItem item = menu.getItem(path);          // get & store path of font
menu.select(item);                           // selecting font
Dialog dg = .....                            //accessing dialog box
ComboBox[] children = dg.findChildren(ComboBox.class, null);
for (ComboBox cb : children)
{
 cb.highlight();
 System.out.println(cb.getAttachedText());
}
ComboBox element = .....
element.select("Arial");
String actual = element.getSelectedItem();
Assert.assertEquals("Arial", actual);
```

(5)

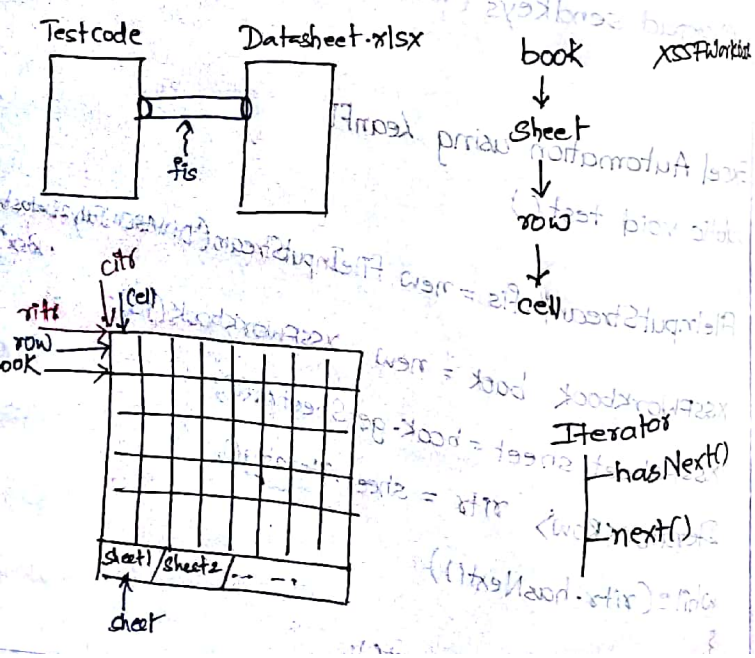
```
Button OK = .....
OK.click();
fullNotepad.sendKeys("Welcome to ABC");
}
```

Excel Automation using LeanFT



```
public void test()
{
FileInputStream fis = new FileInputStream("D:\\ABC\\July2\\Datasheet.xlsx");
XSSFWorkbook book = new XSSFWorkbook(fis);
XSSFSheet sheet = book.getSheetAt(0);
Iterator<Row> ritr = sheet.iterator();
while(ritr.hasNext())
{
 Row row = ritr.next();
```

```
Iterator<Cell> citr = row.iterator();
while(citr.hasNext())
{
 Cell cell =citr.next();
switch (cell.getCellType())
{
 case Cell.CELL_TYPE_STRING:
      S.o.p (cell.getStringCellValue());
      break;
 case Cell.CELL_TYPE_NUMERIC:
      S.o.p (cell.getNumericCellValue());
 }
}
}
```




```
c/p:
dhoni
2000.0
virat
1000.0
pending
3000.0
david
4500.0
```

Application Model FrameWork

```
@Test
p.v. test()
{
 Browser browser = BrowserFactory.Launch (BrowserType.
                                          CHROME);
 browser.navigate ("https://www.magento.com");
 WebElement myacct = ....
 myacct.click();
  EditField
 WebElement email = .....
 email.setValue ("sucheendha.abc@gmail.com");
  EditField
 WebElement password = ....
 password.setValue ("Welcome123");
  Button
 WebElement login = .....
 login.click();
 Link logout = ......
 logout.click();
browser.close();
```

Steps involved in application Model Framework

1. Select the project, right click, go to new & click on other
2. Select LeanFT Application Model item & click on next.
3. Give a name to the application Model Item (AppModel Demo)
4. Click on Finish.

Note: To generate screenshot for all activities (i) after appl model of framework model, inside it, go to resources folder & open leanFT.properties file, scroll down to end & change #SnapshotLevel=off to 'SnapshotLevel=ALL, & save

@Test
p.s.v.test()
{
}

reexecute the pgm. Go to eclipse, click on leanFT, click on view last run results.