

SELENIUM FRAMEWORK TESTNG & BDD

What is TestNG

- **TestNG** is an automation testing framework in which NG stands for "Next Generation".
- Using TestNG, you can generate a proper report, and you can easily come to know how many test cases are passed, failed, and skipped. You can execute the failed test cases separately.
- Suppose, you have five test cases, one method is written for each test case (Assume that the program is written using the main method without using testNG). When you run this program first, three methods are executed successfully, and the fourth method is failed. Then correct the errors present in the fourth method, now you want to run only fourth method because first three methods are anyway executed successfully. This is not possible without using TestNG.
- The TestNG in Selenium provides an option, i.e., testng-failed.xml file in test-output folder. If you want to run only failed test cases means you run this XML file. It will execute only failed test cases.



testNG annotation

BeforeSuite

BeforeTest

BeforeClass

BeforeMethod

Test Case

AfterMethod

AfterClass

AfterTest

AfterSuite

TestNG annotation program

```
import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;
public class testngAnnotations {
    // Test Case 1
    @Test
    public void test1() {
        System.out.println("Test Case 1");
    } // Test Case 2
    @Test
    public void test2() {
        System.out.println("Test Case 2");
    }
    @BeforeMethod
    public void beforeMethod() {
        System.out.println("Before Method");
    }
}
```

```
@AfterMethod
public void afterMethod() {
    System.out.println("After Method");
}
@BeforeClass
public void beforeClass() {
    System.out.println("Before Class");
}
@AfterClass
public void afterClass() {
    System.out.println("After Class");
}
@BeforeTest
public void beforeTest() {
    System.out.println("Before Test");
}
@AfterTest
public void afterTest() {
    System.out.println("After Test");
}
@BeforeSuite
public void beforeSuite() {
    System.out.println("Before Suite");
}
@AfterSuite
public void afterSuite() {
    System.out.println("After Suite");
}
}
```

```
package com.Prac;

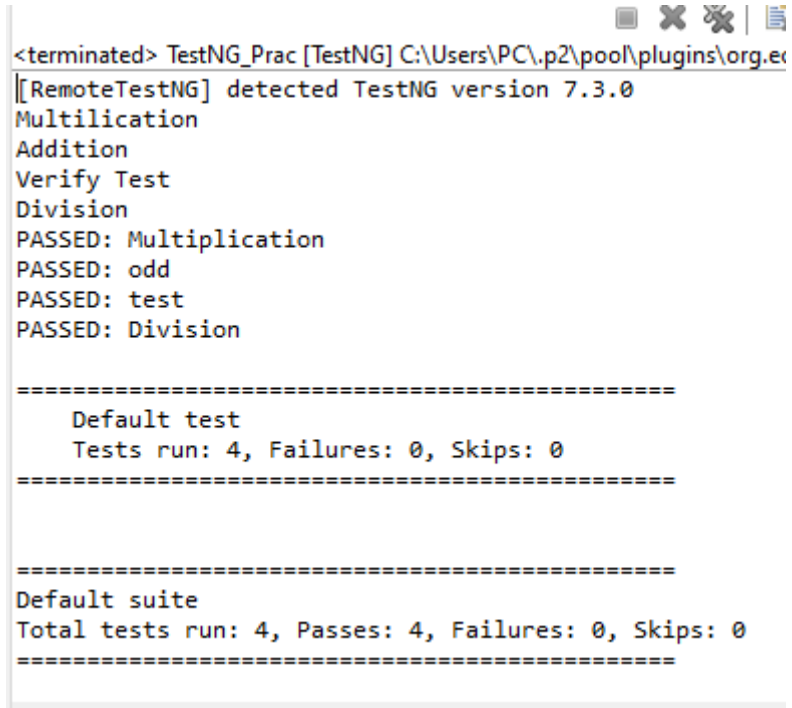
import org.testng.annotations.Test;

public class TestNG_Prac {
    @Test()
    public void test()
    {
        System.out.println("Verify Test");
    }

    @Test(priority=0)
    public void odd()
    {
        System.out.println("Addition");
    }

    @Test(priority=2)
    public void Division()
    {
        System.out.println("Division");
    }

    @Test()
    public void Multiplication()
    {
        System.out.println("Multilication");
    }
}
```



The screenshot shows the output of a TestNG test run in an IDE console. The window title is "<terminated> TestNG_Prac [TestNG] C:\Users\PC\.p2\pool\plugins\org.e". The output text is as follows:

```
<terminated> TestNG_Prac [TestNG] C:\Users\PC\.p2\pool\plugins\org.e
[[RemoteTestNG] detected TestNG version 7.3.0
Multilication
Addition
Verify Test
Division
PASSED: Multiplication
PASSED: odd
PASSED: test
PASSED: Division

=====
    Default test
    Tests run: 4, Failures: 0, Skips: 0
=====

=====
Default suite
Total tests run: 4, Passes: 4, Failures: 0, Skips: 0
=====
```

There are two ways to pass the parameters in TestNG:

- *TestNG Parameters*
- *TestNG DataProviders*

TestNG Parameters

```
package com.javatpoint;
import org.testng.annotations.Test;
import org.testng.annotations.Parameters;
public class Sum
{
    @Test
    @Parameters({"a", "b"})
    public void add(int c, int d)
    {
        int sum=c+d;
        System.out.println("Sum of two numbers : "+sum);
    }
}
```

TestNG xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
    <parameter name="a" value="5"/>
    <parameter name="b" value="3"/>
    <test name="Sum">
        <classes>
            <class name="com.javatpoint.Sum"/>
        </classes>
    </test>
</suite> <!-- Suite -->
```

Passing multiple parameters is just similar to the single parameters, but we will be providing multiple values in a single parameter. Observe the following test code, which checks if the sum of two integers is as expected or not.

```
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

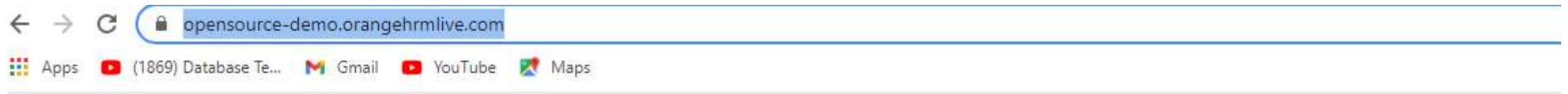
public class DProvider {
    @DataProvider (name = "data-provider")
    public Object[][] dpMethod(){
        return new Object[][] {{2, 3 , 5}, {5, 7, 9}};
    }

    @Test (dataProvider = "data-provider")
    public void myTest (int a, int b, int result) {
        int sum = a + b;
        Assert.assertEquals(result, sum);
    }
}
```

In the above code, I have passed three values a,b and result to check if the sum is equal to result or not

```
PASSED: Sum(2, 3, 5)
FAILED: Sum(5, 7, 9)
java.lang.AssertionError: expected [12] but found [9]
```

Sample Project URL : <https://opensource-demo.orangehrmlive.com/>



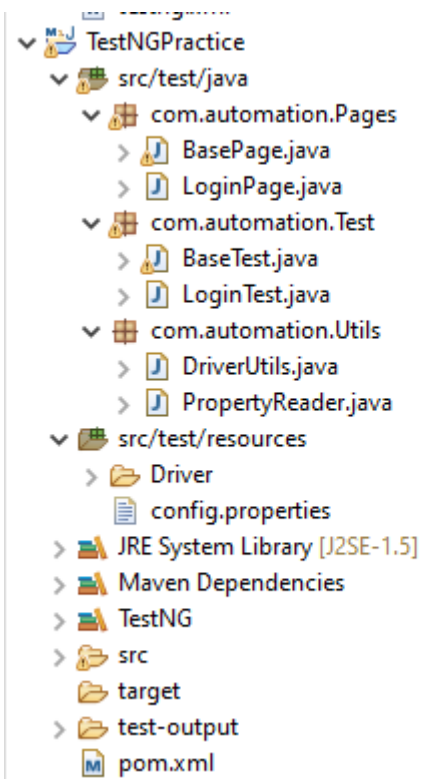
(Username : Admin | Password : admin123)



LOGIN Panel

LOGIN

[Forgot your password?](#)

- 
- The screenshot displays the project structure of a TestNG framework in an IDE. The project is named 'TestNGPractice'. It contains a 'src/test/java' directory with three sub-packages: 'com.automation.Pages' (containing 'BasePage.java' and 'LoginPage.java'), 'com.automation.Test' (containing 'BaseTest.java' and 'LoginTest.java'), and 'com.automation.Utils' (containing 'DriverUtils.java' and 'PropertyReader.java'). There is also a 'src/test/resources' directory containing a 'Driver' folder and a 'config.properties' file. The project also includes a 'pom.xml' file and is configured with 'JRE System Library [J2SE-1.5]', 'Maven Dependencies', and 'TestNG'.
- TestNGPractice
 - src/test/java
 - com.automation.Pages
 - BasePage.java
 - LoginPage.java
 - com.automation.Test
 - BaseTest.java
 - LoginTest.java
 - com.automation.Utils
 - DriverUtils.java
 - PropertyReader.java
 - src/test/resources
 - Driver
 - config.properties
 - JRE System Library [J2SE-1.5]
 - Maven Dependencies
 - TestNG
 - src
 - target
 - test-output
 - pom.xml

Code for DriverUtils.java

```
1 package com.automation.Utils;
2
3 import java.util.concurrent.TimeUnit;
4
5 public class DriverUtils {
6
7     public static WebDriver driver;
8
9     public static void CreateDriver()
10    {
11        System.setProperty("webdriver.chrome.driver",
12                            "C:\\Users\\PC\\eclipse-workspace\\TestNGPractice\\src\\test\\resources\\Driver\\chromedriver.exe");
13        driver = new ChromeDriver();
14        driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);
15        driver.manage().window().maximize();
16    }
17
18    public static WebDriver getDriver() {
19
20        if (driver==null)
21        {
22            CreateDriver();
23        }
24
25        return driver;
26    }
27
28 }
```

Code for PropertyReader.java

```
1 package com.automation.Utils;
2
3 import java.io.FileInputStream;
4
5 public class PropertyReader {
6
7     static Properties prop=new Properties();
8
9     public static void init() throws FileNotFoundException, IOException
10    {
11        prop.load(new FileInputStream
12                  ("C:\\Users\\PC\\eclipse-workspace\\TestNGPractice\\src\\test\\resources\\config.properties"));
13    }
14
15    public static String getproperty(String key) {
16        return prop.getProperty(key);
17    }
18
19 }
```

BASEPAGE.JAVA

```
1 package com.automation.Pages;
2
3+ import org.openqa.selenium.WebDriver;
4
5
6
7
8
9
10 public class BasePage {
11     WebDriver driver;
12
13-     public BasePage()
14     {
15
16         driver=DriverUtils.getDriver();
17         PageFactory.initElements(driver,this);
18     }
19
20 }
21
```

LoginPage.java

```
1 package com.automation.Pages;
2
3+ import org.openqa.selenium.WebElement;
4
5
6 public class LoginPage extends BasePage {
7-     @FindBy(id="txtUsername")
8     private WebElement txtUsername;
9
10-     @FindBy(id="txtPassword")
11     private WebElement txtPassword;
12
13-     @FindBy(id="btnLogin")
14     private WebElement btnLogin;
15
16-     @FindBy(xpath="//span[text()='Invalid credentials']")
17     private WebElement txtmsg;
18
19
20-     public void enterUsername(String un)
21     {
22         txtUsername.sendKeys(un);
23     }
24
25-     public void enterPassword(String pw)
26     {
27         txtPassword.sendKeys(pw);
28     }
29
30
31-     public void clickLogin()
32     {
33         btnLogin.click();
34     }
35
36 }
37
```

BaseTest.java

```
1 package com.automation.Test;
2
3+ import java.io.FileNotFoundException;
4
5-
6 public class BaseTest {
7
8     @BeforeTest
9     public void start() throws FileNotFoundException, IOException
10 {
11     DriverUtils.CreateDriver();
12     PropertyReader.init();
13     System.out.println("property created");
14 }
```

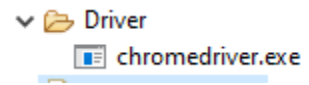
LoginTest.java

```
1 package com.automation.Test;
2
3+ import java.io.FileNotFoundException;
4
5- Run All
6 public class LoginTest extends BaseTest {
7-     @Test
8     Run | Debug
9     public void verifyLogin() throws InterruptedException, FileNotFoundException, IOException
10 {
11     LoginPage lp=new LoginPage();
12     DriverUtils.getDriver().get("https://opensource-demo.orangehrmlive.com/");
13     Thread.sleep(10000);
14     lp.enterUsername(PropertyReader.getproperty("login.username"));
15     lp.enterPassword(PropertyReader.getproperty("login.password"));
16     lp.clickLogin();
17 }
18 }
```

```
1 login.username=Admin  
2 login.password=admin123
```



DRIVER EXECUTABLE



Driver executable file need to download based on chrome version which you are using



CUCUMBER/BDD FRAMEWORK

Behaviour driven development is the software development process that cucumber was build

To support

BDD Supports Agile.

Cucumber is used to execute automated acceptance tests written in the “Gherkin” language. Gherkin is a domain-specific language for behavior de




















Cucumber test automation makes use of two important files –

Feature file – Contains code written in **Gherkin** (plain English text)

- **Given**(Example)-the user has entered invalid credentials
- **When**(Example)-the user clicks submit button
- **Then**(Example)-display the proper validation message

Step definition file – Contains the actual code



- ▼  src/test/java
 - >  com.automation.Pages
 - >  com.automation.runner
 - >  com.automation.Stepdef
 - >  com.automation.Test
 - >  com.automation.Util
- ▼  src/test/resources
 - ▼  config
 -  config.properties
 - ▼  Driver
 -  chromedriver.exe
 - ▼  Features
 - >  BackgroundFeature
 -  Data.feature
 -  Demooutline.feature
 -  Execution.feature
 -  Out.feature
 -  Out1.feature
 -  cucumber.properties



Login Feature

Feature: This feature we will gonna test login functionality

Author - Plabani

Developer - Plabani

@regression

Scenario: verify Login Successful

Given user open website

When user login with username "login.username" and password "login.password"

Then verify login successful

LoginSteps:

```
package com.automation.stepdef;
```

```
import org.junit.Assert;
```

```
import com.automation.pages.LoginPage;
```

```
import io.cucumber.java.en.Then;
```

```
import io.cucumber.java.en.When;
```

```
public class LoginSteps {
```

```
    LoginPage loginPage = new LoginPage();
```

```
        @When("user open website")
```

```
        public void user_open_website(){
```

```
            loginPage.openWebsite();
```

```
        }
```

```
        @When("user login with username {string} and password {string}")
```

```
        public void user_login_with_username_and_password(String username, String password) {
```

```
            loginPage.doLogin(username, password);
```

```
        }
```

LoginPage

```
package com.automation.pages;
```

```
import org.junit.Assert;
```

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

```
import org.openqa.selenium.support.FindBy;
```

```
import com.automation.utils.PropertyReader;
```

```
public class LoginPage extends BasePage {
```

```
    @FindBy(id = "txtUsername")
```

```
    private WebElement usernameInput;
```

```
    @FindBy(id = "txtPassword")
```

```
    private WebElement passwordInput;
```

```
    @FindBy(id = "btnLogin")
```

```
    private WebElement loginBtn;
```

```
    @FindBy(id = "spanMessage")
```

```
    private WebElement loginErrorMsg;
```

```
    public void doLogin(String usernameKey, String passwordKey) {
```

```
        usernameInput.sendKeys(PropertyReader.getProperty(usernameKey));
```

```
        passwordInput.sendKeys(PropertyReader.getProperty(passwordKey));
```

```
        loginBtn.click();
```

```
    }
```

```
    public void openWebsite() {
```

```
        driver.get(PropertyReader.getProperty("base.url"));
```

```
    }
```

```
    public void validateLoginPage() {
```

```
        Assert.assertTrue("Username not displayed on login page", usernameInput.isDisplayed());
```

```
        Assert.assertTrue("Password not displayed on login page", passwordInput.isDisplayed());
```

```
    }
```

```
    public boolean isLoginErrorDisplayed() {
```

```
        return loginErrorMsg.isDisplayed();
```

```
    }
```

HomeSteps:

```
@Then("verify login successful")
public void verify_login_successful() {
    Assert.fail();
    homePage.verifyHomePage();
}
```

HomePage

```
package com.automation.pages;

import java.util.List;

import org.junit.Assert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.support.FindBy;
import io.cucumber.datatable.DataTable;
public class HomePage extends BasePage {
    @FindBy(id = "branding")
    private WebElement logo;

    @FindBy(id = "menu_admin_viewAdminModule")
    private WebElement menuAdmin;

    @FindBy(id = "menu_admin_UserManagement")
    private WebElement userManagementMenu;

    @FindBy(id = "menu_admin_viewSystemUsers")
    private WebElement usersMenu;

    @FindBy(id = "btnAdd")
    private WebElement addBtn;

    @FindBy(id = "welcome")
    private WebElement userIcon;

    @FindBy(xpath = "//a[text()='Logout']")
    private WebElement logoutLink;

    public void verifyHomePage() {
        Assert.assertTrue("Homepage is not displayed", logo.isDisplayed());
    }
}
```



Hooks

```
package com.automation.stepdef;

import com.automation.utils.DriverUtils;
import com.automation.utils.PropertyReader;

import io.cucumber.java.After;
import io.cucumber.java.Before;

public class Hooks {

    @Before("@ui")
    public void setUpUITest() {
        DriverUtils.createDriver();
        PropertyReader.initProperty();
    }

    @After("@ui")
    public void cleanUp() {
        DriverUtils.getDriver().quit();
    }
}
```



Feature-Scenario OutLine-Concept

Scenario outline basically replaces variable/keywords with the value from the table. Each row in the table is considered to be a scenario

Login.feature

Feature: Verify Login

Author:Plabani

Scenario Outline: verify Login Unsuccessful

with valid user and invalid password

Given user open website

When user login with username "<username>"

When user login with password "<password>"

Then verify login error message is displayed

Examples:

username	password	
login.username	invalid.login.password	
invalid.login.user	login.password	

LoginSteps:

@When("user login with username {string}")

```
public void user_login_with_username(String string) {
```

```
lp.enterUsername(PropertyReader.getProperty(string));
```

```
}
```

@When("user login with password {string}")

```
public void user_login_with_password(String string) {
```

```
lp.enterPassword(PropertyReader.getProperty(string));
```

```
}
```



```
package com.automation.Pages;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
import com.automation.Util.DriverUtils;
public class LoginPage extends BasePage{

    @FindBy(id="txtUsername")
    private WebElement txtUsername;

    @FindBy(id="txtPassword")
    private WebElement txtPassword;

    @FindBy(id="btnLogin")
    private WebElement btnLogin;

    @FindBy(xpath="//span[text()='Invalid credentials']")

    private WebElement txtmsg;

    public void enterUsername(String un)
    {
        txtUsername.sendKeys(un);
    }

    public void enterPassword(String pw)
    {
        txtPassword.sendKeys(pw);
    }

    public void clickLogin()
    {
        btnLogin.click();
    }

    public boolean isLoginErrorDisplayed() {
        return txtmsg.isDisplayed();
    }
}
```



Feature: Verify Login

Author:Plabani

Scenario: verify Login with valid user and Valid password

Given user open website

When user login with Valid username "<username>"

|Admin|

And user login with Valid password "<password>"

|admin123|

Then Click Login

LoginSteps:

```
@When("user login with Valid username {string}")
public void user_login_with_valid_username(String string, io.cucumber.datatable.DataTable
dataTable) {
    List<List<String>> data=dataTable.asLists();
    DriverUtils.getDriver().findElement(By.id("txtUsername")).sendKeys(data.get(0).get(0));

}
```

```
@When("user login with Valid password {string}")
public void user_login_with_valid_password(String string, io.cucumber.datatable.DataTable
dataTable) {
    List<List<String>> data=dataTable.asLists();
    DriverUtils.getDriver().findElement(By.id("txtPassword")).sendKeys(data.get(0).get(0));
}
```

```
@Then("Click Login")
public void click_login() {
    lp.clickLogin();

}
```

Note: LoginPage can use previous slide programsa



Background in **Cucumber** is used to define a step or series of steps that are common to all the tests in the feature file.
A **Background** is much like a **scenario** containing a number of steps.

BACKGROUND.FEATURE

Feature: Verify Login Demo

Background: Verify Login

Given Open URL

Scenario Outline: Verfy username and password

When enter username and password <username> and <password>

Then Login click

Examples:

username	password	
login.username	invalid.login.password	
invalid.login.user	login.password	

Scenario: Verrify Login

When Login successful

Then click on welcome button

And click on Logout link

BACKGROUNDSTEPS

PACKAGE COM.AUTOMATION.STEPDEF;

IMPORT IO.CUCUMBER.JAVA.EN.GIVEN;

IMPORT IO.CUCUMBER.JAVA.EN.THEN;

IMPORT IO.CUCUMBER.JAVA.EN.WHEN;

PUBLIC CLASS BACKGROUDSTEPS {

 @GIVEN("OPEN WEBSITE")

 PUBLIC VOID OPEN_WEBSITE() {

 }

 @WHEN("ENTER VALID USERNAME AND PASSWORD")

 PUBLIC VOID ENTER_VALID_USERNAME_AND_PASSWORD() {

 }

 @THEN("CLICK ON LOGIN BUTTON")

 PUBLIC VOID CLICK_ON_LOGIN_BUTTON() {

 }

 @WHEN("USER CLICK ON WELCOME OPTION")

 PUBLIC VOID USER_CLICK_ON_WELCOME_OPTION() {

 }

 @THEN("CHECK LOGOUT LINK")

 PUBLIC VOID CHECK_LOGOUT_LINK() {

 }

 @WHEN("CLICKED ON DASHBOARD")

 PUBLIC VOID CLICKED_ON_DASHBOARD() {

 }

 @THEN("DASHBOARD FRAME DISPLAYED")

 PUBLIC VOID DASHBOARD_FRAME_DISPLAYED() {

 }

}

```
1 package com.automation.runner;
2
3 import org.junit.runner.RunWith;
4
5
6
7
8 @RunWith(Cucumber.class)
9 //@CucumberOptions(features="C:\\Users\\PC\\eclipse-workspace\\CucumebrAutomation\\src\\test\\resources\\features\\OneDat
10 glue = "com.automation.Stepdef", plugin = "json:target/cucumber.json"
11 //)
12 public class Ru {
13
14 }
15
```

```
1 login.username=admin
2 login.password=admin123
3 invalid.username=admin1
4 valid.password=admin123
5 valid.username=admin
6 invalid.password=123
7 empname=Alice Duval
8 username=Neha2
9 Password=Plabani1234!
10 confirm_password=Plabani1234!
11 |
```