# TestNG Framework (Testing New Generation Framework)

TDD – Test Driven Development Framework

It’s a unit testing framework, for developers. Testers also can use to design test cases in systematic way.

Features available:

* 1. HTML reports
  2. Different annotations
  3. Priorities/sequences
  4. Dependencies
  5. Grouping
  6. Paremeters
  7. Data provider
  8. Testng.xml
  9. Can generate proper format of test results
  10. Testcases can be grouped and execute by testing.xml
  11. Same testcase can be execute multiple times (invocation count)
  12. Make it run in certain order by prioritizing them
  13. Skip particular test
  14. Annotations available
  15. No static main method necessary
  16. Uncaught exceptions are automatically handled by TestNG, by considering that test case as failed.
  17. Parallel testing is possible

Annotations

|  |  |  |
| --- | --- | --- |
| @BeforeSuite | Set system property | Pre conditions |
| @BeforeTest | Launch chrome browser |
| @BeforeClass | Login to app |
| @BeforeMethod | Enter URL (repeat b4 all @Test) |
|  |  |  |
| @Test | Google title test | Test cases |
|  |  |  |
| @AfterMethod | Logout (repeat after all @Test) | Post conditions |
| @AfterClass | Close browser |
| @AfterTest | Delete all cookies |
| @AfterSuite | Generate test report |

1. @BeforeMethod and @AfterMethod will execute before and after each and every @Test methods
2. @Test => Execute in alphabetical manner
3. @Test(enabled = false) => Skip the testcase
4. @Test(Priority = 1) => Make them execute in particular order/sequence
5. @Test(Priority = 1, groups = “cars”) => make test cases grouping
6. @Test(dependsOnMethods= “loginTest”) => if loginTest got failed, it’s all dependent methods will get skipped
7. @Test(dependsOnMethods= “loginTest”, alwaysRun = true)=> this method will always run even its dependent method fails
8. @Test(invocationCount= 5) => Same test case will be executed 5 times
9. @Test(timeOut = 2000) => Entire test method should be terminated in 2000milli seconds
10. @Test(expectedExceptions = ArithmeticException.class) => instruct JVM not to consider failed, if this exception arises.
11. @DataProvider and @Test(dataprovider = “**getTestData**”) => provide data from external source
12. @Parameters({“url”}, {“username”})

## TestNG html report

Test-Output folder -> Index.html

* 1. Info
     + Testing.xml
     + 1 Tests
     + 4 Groups
     + Times
     + Reporter output
     + Ignored methods
     + Chronological view
  2. Results
     + Passed
     + failed

## Data Driven Framework with TestNG – Data Provider (@dataprovider)

**[Test case will execute multiple times for each and every set of data]**

1. public class TestUtil

{

Public static ArrayList<Object[]> getDataFromExcel()

{

//assume 4 rows of data

//read all data from excel by apache POI or XLs\_Reader

//Store every row in Object[] array

//add that array in an ArrayList<Object[]> in every iteration

//finally return the ArrayList<Object[]>

}

}

1. @Dataprovider

Public Iterator<Object[]> **getTestData**()

{

ArrayList<Object[]> testData = TestUtil.getDataFromExcel();

return testData.iterator();

}

1. @Test(dataprovider = “**getTestData**”)

Public void myTestPage(String firstName, String lastName, String email, String state)

{

//use the parameters

}

Result:

PASSED: myTestPage(String firstName1, String lastName1, String email1, String state1)

PASSED: myTestPage(String firstName2, String lastName2, String email2, String state2)

PASSED: myTestPage(String firstName3, String lastName3, String email3, String state3)

PASSED: myTestPage(String firstName4, String lastName4, String email4, String state4)

**Parameters (keep environment variables – generic variables throughout the program)**

Passing parameters from testing.xml file

**Testmg.xml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="Suite">

<test name="Test">

<parameter name=“url” value=“https://www.gmail.com”>

<parameter name=“username” value=“abcd.gmail.com”>

<classes>

<class name="com.paramenter.ParameterTest" />

</classes>

</test>

</suite>

**TestCase**

@test

@parameters({“url”, “username”})

Public void gmailLoginTest(String url, String userName)

{

//code

}

**Parallel run of tests/classes/suites in TestNG**

**Parallel – methods**

<suite name="Suite">

<test name="Test" parallel= “methods”>

<classes>

<class name="com.paramenter.ParameterTest" />

</classes>

</test>

</suite>

**Parallel – Classes**

<suite name="Suite">

<test name="Test" parallel= “classes”>

<classes>

<class name="com.paramenter.ChromeTest" />

<class name="com.paramenter.FirefoxTest" />

</classes>

</test>

</suite>

**Parallel – Tests**

<suite name="Suite" parallel="tests" thread-count="1">

<test name="Test">

<classes>

<class name="com.paramenter.ChromeTest" />

<class name="com.paramenter.FirefoxTest" />

</classes>

</test>

<test name="Test">

<classes>

<class name="com.paramenter.ChromeTest" />

<class name="com.paramenter.FirefoxTest" />

</classes>

</test>

</suite>

# POM (Page Object Model)

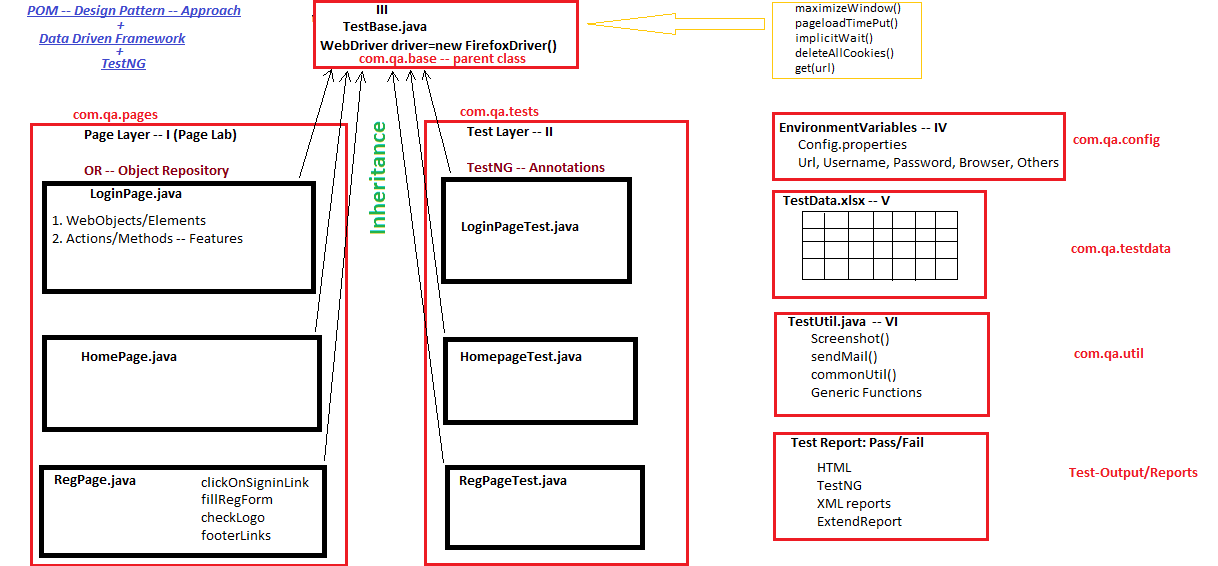
Design pattern to maintain object repository for WebElements

Easier to maintain the code

Objects can be reused, avoid duplicates

Layers:

* 1. Page layer – WebElements and their corresponding methods
  2. Test Layer – Test cases and its verifications



### POM without PageFactory

**In page layer:**

public class HomePage {

    By emailAddress = By.xpath("//div[contains(@id,'Emaild')]");

    public void typeEmailId(String id){

        driver.findElement(emailAddress).sendKeys(id)

    }

}

### POM with PageFactory

**In Page Layer:**

public class HomePage {

public HomePage()

{

PageFactory.initElements(driver, this);

}

@FindBy(xpath="//button[text()='email']")

WebElement emailAddress;

Public void typeEmailId(String id)

{

emailAddress.sendKeys(id);

}

}