1)what is the use of framework in automation testing

Framework is nothing but it is separate project contains packages/classes/methods.

generally framework contains reusable components/code

Ex: in framework i created classes/methods to handle exceptions/custom reports, reading data from database, excel file

It will be a **jar** file in java

It will be **DLL** in C#

in automation code we add this jar/dll’s (using buildpath->add external jar files option) and call appropriate classes/method Whenever it is required

Ex: selenium jar file is framework and we add this file jar file in our project

We use classes/methods of selenium framework (WebDriver, Webelement classes, get method)

2) what are things u consider/keep in framework

Need to Create classes/methods to handle exceptions/custom reports, reading data from database, excel file

3) different types of frameworks we have in selenium/qtp

Ans: Data driven, hybrid, keyword driven, BDD

4) what is data driven frame work and when do we use it?

Ans: Data Driven Framework is a framework which is driven by the different combinations of input and output data.

to test same functionality more than once with different set of data/input data

We store the test data in external files (excel/csv) or table.

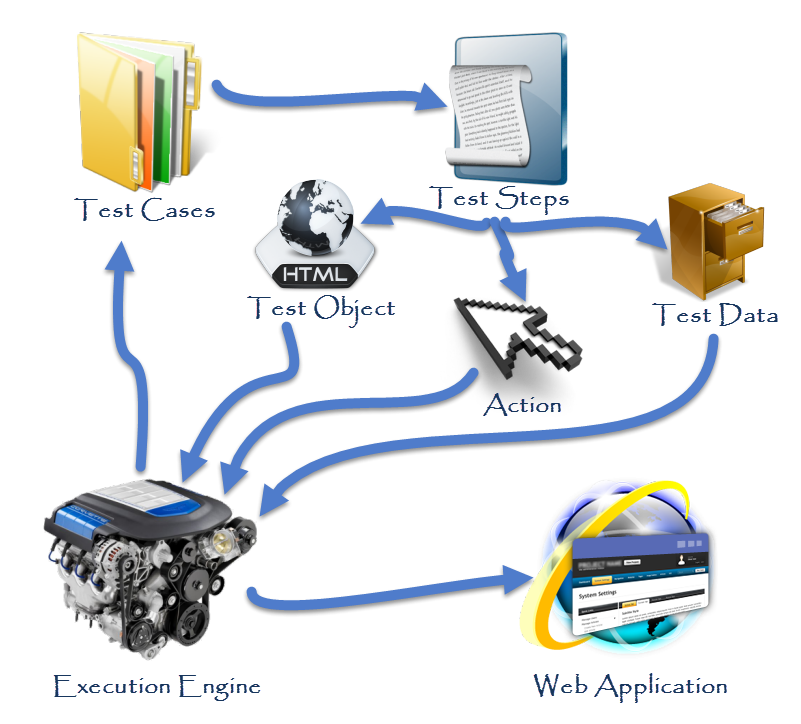
5) what is keyword frame work and when do we use it?

Ans:

Keyword Driven Framework is a type of**Functional Automation Testing Framework** which is also known as **Table-Driven** testing or **Action Word based** testing. The basic working of the *Keyword Driven Framework* is to divide the Test Case in to four different parts. First is called as *Test Step*, second is *Object* of Test Step, third is *Action* on Test Object and fourth is *Data* for Test Object.

The above categorization can be done and maintained with the help of Excel spread sheet:

**Test Step:** It is a very small description of the *Test Step* or the description of the *Action* going to perform on *Test Object*.  
**Test Object:** It is the name of the Web *Page object/element*, like Username & Password.  
**Action:** It is the name of the *action*, which is going to perform on any Object such as *click*, *open* browser, *input*etc.  
**Test Data:** Data can be any value which is needed by the Object to perform any action, like Username value for Username field.



6)what is hybrid framework and when do we use it

This framework is the combination of both data-driven and keyword driven testing frameworks. It allows data driven scripts to take advantage of the powerful libraries and utilities in a keyword based approach.

**Advantages:**

The Hybrid framework is build 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

1. what is POM and modular framework

# Modular Driven Framework

In most of the web application we have few set of actions which are always executed in the series of actions. Rather than writing those actions again and again in our test, we can club those actions in to a method and then calling that method in our test script. Modularity avoids duplicacy of code. In future if there is any change in the series of action, all you have to do is to make changes in your main modular method script. No test case will be impacted with the change.

**What is POM?**

* **Page Object Model** is a design pattern to create **Object Repository** for web UI elements.
* 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.
* Name of these methods should be given as per the task they are performing, i.e., if a loader is waiting for the payment gateway to appear, POM method name can be waitForPaymentScreenDisplay().

**Advantages of POM**

1. Page Object Patten says operations and flows in the UI should be separated from verification. This concept makes our code cleaner and easy to understand.
2. The Second benefit is the **object repository is independent of test cases**, so we can use the same object repository for a different purpose with different tools. For example, we can integrate POM with TestNG/JUnit for functional[Testing](http://www.guru99.com/software-testing.html)and at the same time with JBehave/Cucumber for acceptance testing.
3. Code becomes less and optimized because of the reusable page methods in the POM classes.
4. **Methods** get **more realistic names** which can be easily mapped with the operation happening in UI. i.e. if after clicking on the button we land on the home page, the method name will be like 'gotoHomePage()'.

**How to implement POM?**

Simple POM:

It's the basic structure of Page object model (POM) where all Web Elements of the **AUT** and the method that operate on these Web Elements are maintained inside a class file.A task like **verification** should be **separate** as part of Test methods.

### What is Page Factory?

Page Factory is an inbuilt Page Object Model concept for Selenium WebDriver but it is very optimized.

Here as well, we follow the concept of separation of Page Object Repository and Test Methods. Additionally, with the help of PageFactory class, we use annotations **@FindBy** to find WebElement. We use initElements method to initialize web elements

**@FindBy** can accept **tagName, partialLinkText, name, linkText, id, css, className, xpath**as attributes.

Let's look at the same example as above using Page Factory

### AjaxElementLocatorFactory

    One of the key advantages of using Page Factory pattern is AjaxElementLocatorFactory Class.

It is working on lazy loading concept, i.e. a timeout for a WebElement will be assigned to the Object page class with the help of AjaxElementLocatorFactory .

Here, when an operation is performed on an element the wait for its visibility starts from that moment only. If the element is not found in the given time interval, test case execution will throw 'NoSuchElementException' exception

**Summary**

1. Page Object Model is an Object Repository design pattern in Selenium WebDriver.
2. POM creates our testing code maintainable, reusable.
3. Page Factory is an optimized way to create object repository in POM concept.
4. AjaxElementLocatorFactory is a lazy load concept in Page Factory pattern to identify WebElements only when they are used in any operation.

**5)what is testng and junit?**

# Ans: JUnit Vs TestNG

Both TestNG and Junit are[Testing](http://www.guru99.com/software-testing.html)framework used for unit testing. TestNG is similar to JUnit. Few more functionalities are added to it that makes TestNG more powerful than JUnit.

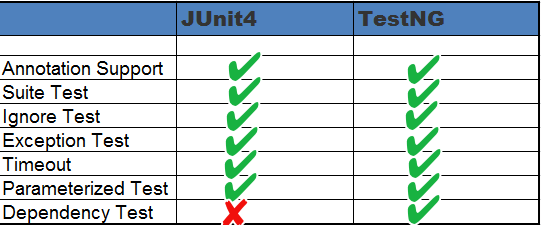
This tutorial is mainly focused to analyse features of JUnit and TestNG. It help developers to decide which framework should be used for unit testing. Let's analyse similarities between TestNG and JUnit4 first.

TestNG is a testing framework inspired by JUnit and NUnit.

Here is the table that shows the features supported by JUnit and TestNG.

### JUnit4 and TestNG Feature comparison

Both TestNG and JUnit4 looks similar, except one or two features. Let's have a comparison between the two to quickly decide, which technology is more favourable for unit testing. Below table highlights the features supported by both:

[](http://cdn.guru99.com/images/junit/052416_1236_JUnitVsTest1.png)

### Annotations

Both JUnit and TestNG uses annotations and almost all the annotations looks similar.

TestNG uses @BeforeMethod ,@AfterMethod similar to @Before ,@After in JUnit4.

Both TestNG and Junit4 uses @Test(timeout = 1000) for timeout .Check the table below for more details-

|  |  |  |  |
| --- | --- | --- | --- |
| **S.N.** | **Description** | **TestNG** | **JUnit 4** |
| 1 | Test annotation | @Test | @Test |
| 2 | Executes before the first test method is invoked in the current class | @BeforeClass | @BeforeClass |
| 3 | Executes after all the test methods in the current class | @AfterClass | @AfterClass |
| 4 | Executes before each test method | @BeforeMethod | @Before |
| 5 | Executes after each test method | @AfterMethod | @After |
| 6 | annotation to ignore a test | @Test(enbale=false) | @ignore |
| 7 | annotation for exception | @Test(expectedExceptions = ArithmeticException.class) | @Test(expected = ArithmeticException.class) |
| 8 | Timeout | @Test(timeout = 1000) | @Test(timeout = 1000) |
| 9 | Executes before all tests in the suite | @BeforeSuite | n/a |
| 10 | Executes after all tests in the suite | @AfterSuite | n/a |
| 11 | Executes before a test runs | @BeforeTest | n/a |
| 12 | Executes after a test runs | @AfterTest | n/a |
| 13 | Executes before the first test method is invoked that belongs to any of these groups is invoked | @BeforeGroups | n/a |
| 14 | run after the last test method that belongs to any of the groups here | @AfterGroups | n/a |

### Suite Test

Suites are used to execute multiple tests together. Suites can be created using both TestNG and JUnit4. However, suites are more powerful in TestNG as it uses very different method for execution of tests. Let's understand it using code snippet as given below:

**Using JUnit4**

Below class describes use of suite while working with JUnit4:

package guru99.junit;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

@RunWith(Suite.class)

@Suite.SuiteClasses({

SuiteTest1.class,

SuiteTest2.class,

})

public class JunitTest {

// This class remains empty,it is used only as a holder for the above annotations

}

**Using TestNG**

TestNG uses xml to bundle all tests at one place.Below xml describes use of suite while working with TestNG:

<!DOCTYPE suite SYSTEM "http://beust.com/testng/testng-1.0.dtd" >

<suite name="My test suite">

<test name="testing">

<classes>

<class name="com.guru99.SuiteTest1" />

<class name="com.guru99.SuiteTest2" />

</classes>

</test>

</suite>

### Ignore Test

Using both we can skip a test .Let's see it using code example as given below:

**Using JUnit4**

Below code snippet describes use of @ignore annotation while working with JUnit4:

@Ignore

public void method1()

{

System.out.println("Using @Ignore , this execution is ignored");

}

**Using TestNG**

Below code snippet describes use of @Test(enabled=false) annotation while working with TestNG:

@Test(enabled=false)

public void TestWithException()

{

System.out.println("Method should be ignored as it's not ready yet");

}

### Exception Test

Exception testing is available both in TestNG and JUnit4. It is used to check, which exception is thrown from the test?

**Using JUnit4**

Below code snippet describes use of exception test while working with JUnit4:

@Test(expected = ArithmeticException.class)

public void divideByZero()

{

Int i = 1/0;

}

**Using TestNG**

Below code snippet describes use of exception test while working with TestNG:

@Test(expectedExceptions = ArithmeticException.class)

public void divideByZero()

{

Int i = 1/0;

}

**Timeout**

This feature is implemented both in TestNg and JUnit4.Timeout is used to terminate a test which takes longer than specified time (in milliseconds).

**Using JUnit4**

Below code snippet describes use of timeout test while working with JUnit4:

@Test(timeout = 1000)

public void method1()

{

while (true);

}

**Using TestNG**

Below code snippet describes use of timeout test while working with TestNG:

@Test(timeOut = 1000)

public void method1()

{ while (true);

}

### Parameterized Test

JUnit provides an easier and readable approach to test known as Parameterized test. Both TestNG and JUnit supports parameterized test but differ in the way they define the parameter value. Let see this one by one.

**Using JUnit4**

The "@RunWith" and "@Parameter" annotations are used to provide parameter value for the unit test. The annotation @Parameters have to return List[] .This parameter will be passed into the class constructor as an argument.

@RunWith(value = Parameterized.class)

public class JunitTest{

privateint number;

public JunitTest6(int number)

{

this.number = number;

}

@Parameters

public static Collection<Object[]> data()

{

Object[][] data = new Object[][] { { 1 }, { 2 }, { 3 }, { 4 } };

returnArrays.asList(data);

}

@Test

public void parameterTest()

{

System.out.println("Parameterized Number is : " + number);

}

}

**Using TestNG**

In TestNG, XML file or "@DataProvider" is used to provide a parameter for testing.

Here @Parameters annotation declared in the method, needs a parameter for testing. The data used as the parameter will provide in TestNG's XML configuration files. By doing this, we can reuse a single test case with different data sets, and we can get different results.

public class Test1 {

@Test

@Parameters(value="number")

public void parameterTest(int number)

{

System.out.println("Parameterized Number is : " + number);

}

}

See below xml file to be used for above class:

<!DOCTYPE suite SYSTEM "http://beust.com/testng/testng-1.0.dtd" >

<suite name="My test suite">

<test name="testing">

<parameter name="number" value="2"/>

<classes>

<class name="com.guru99.Test1" />

</classes>

</test>

</suite>

### Summary :

We saw JUnit4 and TestNG comparison in details. We also saw both are similar except parameterized test and dependency test. In short we can say, based on flexibility and requirement we can choose any one of them for unit testing.

**9)writing unit test cases with testng**

Types of Software Development Tests

**Unit tests** are used to test individual code components and ensure that code works the way it was intended to. Unit tests are written and executed by developers. Most of the time a testing framework like JUnit or TestNG is used. Test cases are typically written at a method level and executed via automation.

**Integration Tests** check if the system as a whole works. Integration testing is also done by developers, but rather than testing individual components, it aims to test across components. A system consists of many separate components like code, database, web servers, etc. Integration tests are able to spot issues like wiring of components, network access, database issues, etc.

**Functional tests** check that each feature is implemented correctly by comparing the results for a given input against the specification. Typically, this is not done at a developer level. Functional tests are executed by a separate testing team. Test cases are written based on the specification and the actual results are compared with the expected results. Several tools are available for automated functional testing like [Selenium](http://www.seleniumhq.org/) and [QTP](http://www.tutorialspoint.com/qtp/).

As mentioned earlier, unit testing helps developers to determine whether the code works correctly. In this blog post, I will provide helpful tips for unit testing in Java.

## 1. Use a framework for unit testing

Java provides several frameworks that for unit testing. [TestNG](http://testng.org/doc/index.html) and [JUnit](http://junit.org/junit4/) are the most popular testing frameworks. Some important features of JUnit and TestNG:

 Easy to setup and run

 Supports annotations

 Allows certain tests to be ignored or grouped and executed together

 Supports parameterized testing, i.e. running a unit test by specifying different values at run time

 Supports automated test execution by integrating with build tools like Ant, Maven, and Gradle

## 2. Use Test Driven Development – Judiciously!

Test-driven development (TDD) is a software development process in which tests are written based on the requirements before any coding begins. Since there is no code yet, the test will initially fail. The minimum amount of code is then written to pass the test. The code is then refactored until it is optimized.

The goal is to write tests that cover all the requirements as against simply writing code first that may not even meet the requirements. TDD is great as it leads to simple modular code that is easy to maintain. Overall development speeds up and defects are easily identified. Also, unit tests get created as a by-product of the TDD approach.

However, TDD may not be suitable in all situations. In projects where the design is complicated, focusing on the simplest design to pass the test cases and not thinking ahead can result in huge code changes. Also the TDD approach is difficult to use for systems which interact with legacy systems, GUI applications or applications that work with databases. Also, the tests need to be updated as the code changes.

So before deciding on TDD approach, the above factors should be kept in mind and a call should be taken based on the nature of the project.

## 3. Measure code coverage

Code coverage measures (in percentage) how much of the code is executed when the unit tests are run. Normally, code with high coverage has a decreased chance of containing undetected bugs, as more of its source code has been executed in the course of testing. Some best practices for measuring code coverage include:

 Use a code coverage tool like Clover, Corbetura, JaCoCo, or Sonar. Using a tool can improve testing quality, as these tools can point out areas of the code that are untested, allowing you to develop additional tests to cover these areas.

 Whenever new functionality is written, immediately write new tests to cover.

 Ensure that there are test cases that cover all the branches of the code, i.e. if/else statements.

High code coverage does not guarantee the tests are perfect, so beware!

## 4. Externalize test data wherever possible

Prior to JUnit4, the data for which the test case was to be run has to be hardcoded into the test case. This created a restriction that in order to run the test with different data, the test case code had to be modified. However, JUnit4 as well as TestNG support externalizing the test data so that the test cases can be run for different datasets without having to change the source code.

### TestNG

The following is the testng.xml (the configuration file for TestNG) that has the data for which the test is to be executed:

|  |  |
| --- | --- |
|  |  |

As can be seen, in this case the test will be executed twice, once each for the values 3 and 7. In addition to specifying the test data via the XML configuration file, it can also be provided in a class via the [DataProvider](http://testng.org/javadocs/org/testng/annotations/DataProvider.html) annotation.

### JUnit

Similar to TestNG, test data can also be externalized for JUnit. The following is a JUnit test case for the same MathChecker class as above:

|  |  |
| --- | --- |
|  |  |

As can be seen, the test data for which the test is to be executed is specified by the **getTestData()** method. This method can easily be modified to read the data from an external file instead of having hardcoded data.

## 5. Use assertions instead of print statements

Many new developers are in the habit of writing a System.out.println statement after each line of code to verify the code executed correctly. This practice often extended to unit tests, leading to cluttered test code. Along with the clutter, this requires manual intervention by developers to verify the output printed on the console to check if the test ran successfully or not. A better approach is to use assertions which automatically indicate test results.

The following StringUtil class is a simple class with one method that concatenates two input strings and returns the result:

The testStringUtil\\_Bad will always pass as it has no assertions. A developer manually needs to verify the output of the test at the console. The testStringUtil\\_Good will fail if the method returns a wrong result and does not require developer intervention.

## 6. Build tests that have deterministic results

Some methods do not have a deterministic result, i.e. the output of that method is not known beforehand and can vary each time. For example, consider the following code that has a complex function and a method that calculates the time required (in milliseconds) for executing the complex function

|  |  |
| --- | --- |
|  |  |

In this case, each time the calculateTime method is executed, it will return a different value. Writing a test case for this method would not be of any use as the output of the method is variable. Thus, the test method will not be able to verify the output for any particular execution.

## 7. Test negative scenarios and borderline cases, in addition to positive scenarios

Often, developers spend a huge amount of time and effort in writing test cases that ensure the application works as expected. However, it is important to test negative test cases as well. A negative test case is a test case that tests if a system can handle invalid data. For example, consider a simple function which reads an alphanumeric value of length 8, typed by a user. In addition to alphanumeric values, the following negative test cases should be tested:

 User specifies non – alphanumeric values like special characters

 User specifies blank value

 User specifies a value which is larger or smaller than 8 characters

Similarly, a borderline test case tests if the system works well for extreme values. For example, if a user is expected to enter a numeric value from 1 to 100, 1 and 100 are the borderline values and it is very important to test the system for these values.