Framework- Semi finished application(set of classes bundled together- jars) which we use and get rid of all redundant logic or to avoid boiler plate code and make our development easy.

**Unit Testing** – testing individual/small code units/snippets for their proper functioning with all possible inputs.

**Why we need unit testing :**

1. test all code together when some code changes and ensure everything works fine without any errors.
2. Test code in early stages and fix the defects in early stages
3. to automate testing
4. test code reusage
5. you can also use this test classes to think about actual test cases.

Unit testing f/w’s – Junit, TestNG

Jars,wars,ears - bundle of compiled source code(.class files)

Jar- core java application package

War- web application package

Ear- enterprise application package

Junit – developed by Erich gamma & Kent beck

Junit is a simple framework to write repeatable tests. It is instance of xunit architecture for unit testing frameworks.

**xUnit** is the collective name for several [unit testing](https://en.wikipedia.org/wiki/Unit_testing) [frameworks](https://en.wikipedia.org/wiki/Software_framework) that derive their structure and functionality from [Smalltalk](https://en.wikipedia.org/wiki/Smalltalk)'s [SUnit](https://en.wikipedia.org/wiki/SUnit). SUnit, designed by [Kent Beck](https://en.wikipedia.org/wiki/Kent_Beck) in 1998,

Junit-x.jar

Java+unit testing = Junit

**Setup**:

Method1: download junit jars from <http://junit.org/junit4/>... Go to down and install- download junit and hamcrest jar, add it to your project build path.

**Method2**: in eclipse – right click on project- go to properties- go to java build path- add library- select junit and click ok

Method3: create maven project in eclipse- add junit jar dependency in pom.xml

Junit3 – extend testcase, manually prefix test before evry test method, setup, teardown methods to implement common logic before and after each method, here u don’t have before and after class methods.

Junit 4- we use annotations, we need not extend TestCase class manually, need not manually prefix test before every test method, and we have different annotaions for different purpose like doing logic before method in class, before all the methods in the class…….

@Test – which says that method is junit test method

@Before- this method runs before every test method in a given class// junit 3 - setup

@After- this method runs after every test method in a given class// junit3- tearDown

@BeforeClass – runs once before all test methods are getting executed

@AfterClass – runs after all test methods are executed

@Ignore – whenever you have added some test methods but may be some of them

Are not fully implemented- you might need to run the testclass by ignoring that methods

When existing class got some implementaion logic changes and you your method should hange accordingly- this scenario we wil try to ignore existing test method until the logic in test method is according to main method.

@Test(timeout) - timeout attribute tests for the method execution within declared time,else it throws exception.

@expected – This helps to make sure that code is behaving as expected in excepional situations.

verify if the actual code is throwing the error that we have decalred in the test method

This attribute takes any subclass of the Throwable-checked, unchecked..

Paramaters

Parameterized tests - @parameterized,@parameters

{3, true}, {7,true}, {10, false}

Junit Suites – to combine the execution of tests from many classes

@RunWith(Suite.class), @Suite.SuiteClasses

Timeout – to check that the given method runs within the given time.

Different assert methods…

Junit – java

Nunit (wriiten in c#)- .net

TestNG – unit testing F/W based from junit and Nunit but it has many more added advantages.

1. More number of annotations – more flexibility
2. All the configurations can be controlled by XML – need not modify java classes or actual code all the time
3. It supports mutithreaded environments and code is thread safe – unlike junit we wil not see any static methods for any declarations.
4. Data driven testing – parameterised testing is supported.
5. It provides support for logging
6. Very good test report generations.
7. Easily group the combinations of methods and easily exclude…
8. It supports all testing from unit testing(I can test a small unit/method in a class) to integration testing(I can combine different classes, packages…..)

Testng jar

Different ways in eclipse to add the external jars:

1. Eclipse – help – marketplace- install testng plugin for eclipse and then right click project-properties-JavaBuildPath-libraries- add the

Library to the project build path.

* Using add library

1. Download the jar from main website or maven central repository and place it in your prject by creating a lib folder., and then right click project-properties-JavaBuildPath-libraries- add the

Library to the project build path – add jars

1. Create a maven project and simply add the dependency to your pom.xml…

@Test

Suites-Tests-Classes-testMethods

Before Suite – run once before all the tests tags in the suite are run

BeforeTest- run once before running any method inside the classes tag of test tag

BeforeClass- runs once before runnign the first test method in a given test class

BeforeMethod- run once before running each test method

beforeGroups

afterGroups

AfterMethod

AfterClass

AfterTest

AfterSuite

@Test-

timeout – to define execution time for amethod in milliseconds, it wil fail the test if method takes more than the specified time.

expectedExceptions- it verifies the exceptions to be thrown by the method- if that particular exception is not being thrown from the main method or if different exception occurs then this test will result in a failure

groups—is used for both Test and BeforeClass annotations , it is used to make special /required groups as per the given scenario.

We use include and exclude tags in test-ng.xmk to configure the groups.

dependsOnMethods- this method should execute only if the method on which it is depending wil succesfully execute- else, this method will be skipped from the execution

dependsOnGroups- this method should execute only if the group on which it is depending wil succesfully execute- else, this method will be skipped from the execution

harddependency- depending method wil be surely skipped in case of any exception in parent method on which it is depending upon

softDependency- alwaysRun=true – even though there is issue in parent method it will stil run the child/dependent method…this can be used in a scenario whre u want to follow some order of execution of mthods but still u don’t want the child method execution completely depend on parent execution

parameters-

pass using xml and parameters or from code using dataprovider

alwaysRun

parallelism in tesng:

other @Test attributes:

invocationCount : no of times a method should execute

threadPoolSize: size of thread pool i.e., no of threads to be used to complete the given invocation count,

this attribute will be ignored if invocationCount is not specified.

invocationTimeOut : max time (milliseconds) the given test method should take in order to complete the total invocationCounts.,this attribute will be ignored if invocationCount is not specified.

singleThreaded=true – if u want to make sure that a particular method should run in same thread…

priority- which runs test methods in the order u specify- it runs in ascending order of number…

parallelism at sute level:

<suite name="YelpAppSuite" parallel="tests" thread-count="10">

<suite name="YelpAppSuite" parallel="methods" thread-count="10">

<suite name="YelpAppSuite" parallel="classes" thread-count="10">

<suite name="YelpAppSuite" parallel="instances" thread-count="10">

Testng supports – custom logging

And it gives good reporting features

Also we can run junit classes from test ng using junit=true in testng xml

<suite name="YelpAppSuite" parallel="tests" thread-count="10">

Unit Testing – is testing the smallest unit(small piece) of code for proper operation.

If the unit of code you are testing have some dependencies on other classes or modules you will just mock the dependency(using features of testing fw) , skip all integrations and do unit testing alone.

TDD – test driven development

1.Connection- host,url, uname,pwd

2.DriverManager

3.Connection object

4.Statement object

JDBCConnection{

Returning conn or statement obj

}

5.Execute statement- select \* from emp

Framework: Semi finished application(in the form jars or bundle of classes) using which we can avoid boilerplate code /code redundancy and make development easy

JUnit – this is a unit testing framework

It is under xunit architecture – unit testing architeture

Nunit- dot net

TestNG

Unittesting:

Testclass- test method

execute some testcases- assertions

I need run the testcases

Calculator{

Addition add;

Subtraction sub;

Public double doCalc(double d1, double d2, char operator){

Switch(){

Case ‘+’: add.addition();

Case ‘-’: sub.subtraction();

}

}

}

Integeration Testing

PricingCalculator - calculator

EMICalculator- calculator

Expected- to test that the actual method throws a specific exception

Data driven testing – using @Runwith(parameterised)

@Parameters

Junit suite.class – helps to run set of test classes together instead of running them indivdually

TestNG - testing framework inspired from Junit, Nunit(dot net unit testing fw) and Xunit architectures.

Apart from unit testing it also helps to do functional,regression,integration test cases

It has wide number of annoatations.

It supports data driven testing

It supports parallel testing – TestNG internally suppports multi threading

It has features w.rto dependencies(test depending on other)

It supports xml configuarations rather than grouping classes and test in java class like junit

It provides wide variety of grouping like Suites,Tests,Classes,Methods.

Write a program to find fibonacii series and perform unit testing for the same program.

0 1 1 2 3 5 8 …………