I could see many familiar faces over here.. Good Morning...Good to see you all here

As you all know, Today's topic is JUnit

Before starting, I'm curious to understand what all you know about JUnit

Anybody what do you understand about Junit, what your understanding about Junit??

Good to know that some of you already know Junit, probably its going to make my job easier

Okay lets gets started...

Junit - J in Junit stands for Java and Unit is nothing but Unit =

Unit testing framwwork for Java.. will see as we go through session

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **Agenda**

Agenda - This is what I'm planning to cove it today

Before getting into actual details about Junit - Will quickly cover what is Unit Testing. Where it falls into Software Tesing

What is Junit, some background who invented it when it was invented,

What all configuration that we need to use Junit, what all features its provided

How to write Junit test cases, How to run, how to see results and all...

With Junit we do Mocking, what is it, how to do it, what all popular mocking frameworks

At last - We will go through benefits that we get out of Junit, and some best practices that we should follow while working on Junit

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **Software Testing?**

Because Junit is Unit Testing framework I though to start with some basic...

I know you all know about it very well but it would probably will help you to relate where Junit fits

Software testing is nothing but checking functionalities of application to ensure it meets business requirements and works as per expected

At border level Functional testing is categorized into primarily 3 - Unit / Integration / System - I know there are other like Performance, Regression,, UAT and all but these are primary 3 categories to test software functionality

In Unit testing - we primarily test smallest unit software, Smallest unit can be methods and class. Generally we test these units independently and in isolation

In Integration testing - ??

System Testing - Complete, end-to-end testing of software is system testing

Software testing - traditionally we used to do Manual testing but since last couple of years we have automation in place.

Which actually benefits in many ways..

We all know benefits so won't go into details... Let get into actual topic

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **What is Junit?**

Junit - the definition

In Short, Its a framework which helps us to do unit testing automated way.

This was originally designed by in mid 90s its not something new its way old framework

Its entirely written in Java. So it has set classes which helps us to write test case. Will get into details of thoe. Thease class are in org.junit packages. This bundled into jar named Junit.jar

So if you want to use Junit you can simply downalod and copy Junit jar in you java project and if you have maven project then you can add artifcate

So its very simple to get started with it

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **Automation Frameworks**

This is one slide that i want to quickly cover before geeting into actual Junit

The whole point i wanted to highlight here is that Junit is only to test Java code and that to only this part

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **Unit Test**

Junit framework has various version, current or latest version is 5 but so far we were using mostly v4 that why i'm going to cover v4 here

There was considerable change in v3 to v4 will quickly discuss that as well

With v4 - now any class can be test case earlier in v3 if we want to extend class with some class

To indicate method in class we just need to annote it with @Test

This is input and this is what my expected output - to indicate output we use Asserts

**\*\*\*\*\*\*\*\*\*\*\*\*\*\* Annotation**

**@Ignore**

In JUnit, to ignore a test, just add a @Ignore annotation before or after the @Test method.

**To ignore a test, why not just comment the test methods or @Test annotation?**

A : The test runner will not report the test. In IDE, the test runner will display the ignored tests with different icon or color, and highlight it, so that you know what tests are ignored.

**2. Why make a test that doesn’t test?**

A : For large project, many developers are handling different modules, the failed test may caused by other teams, you can add @Ignore on the test method to avoid the test to break the entire build process.

A : Or you want someone to help to create the test, like @Ignore ("help for this method!"), the optional parameter(String) will be displayed in the test runner.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\* Assert**

Assert is a method useful in determining Pass or Fail status of a test case

There are various types of assertions like Boolean, Null, Identical etc.

Junit provides a class named Assert, which provides a bunch of assertion methods useful in writing test cases and to detect test failure

Otionally the first parameter can be String message that is output on failure

**Boolean -** If you want to test the boolean conditions (true or false), you can use following assert methods

**Null object -** If you want to check the initial value of an object/variable, you have the following methods:

**Identical -** If you want to check whether the objects are identical (i.e. comparing two references to the same java object), or different.

It will return true if **expected == actual**

**assertEquals -** You have **assertEquals(a,b)**which relies on the**equals()**method of the Object class.

* Here it will be evaluated as **a.equals( b ).**
* Here the class under test is used to determine a suitable equality relation.
* If a class does not override the **equals()** method of **Object**class, itwill get the default behaviour of **equals()**method, i.e. object identity.

If **a** and **b** are primitives such as **byte**, **int**, **boolean**, etc. then the following will be done for assertEquals(a,b) :

**a** and **b** will be converted to their equivalent wrapper object type (**Byte,Integer**, **Boolean**, etc.), and then **a.equals( b )** will be evaluated.

Exception -

1. **@Test expected attribute** - Use this if you only want to test the exception type

2. **Try-catch and always fail**() - This is a bit old school, widely used in JUnit 3. Test the exception type and also the exception detail.

Always remember the fail()! If the line you want to test didn’t throw any exception, and you forgot to put the fail(), the test will be passed (false positive).

3. **@Rule ExpectedException -** This ExpectedException rule (since JUnit 4.7) let you test both the exception type and also the exception detail, same like “2. Try-catch and always fail()” method, but in a more elegant way :

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Mocking**

However, in the real world, software has dependencies. We have action classes that depend on services and services that depend on data access objects (DAOs) and the list goes on.

You may call Web Service, you may call MF programs, you may have data base interaction, you may have Asynch communiction - posting message in Q, or listing to Q.

The idea of unit testing is that we want to test our code without testing the dependencies.

This test allows you to verify that the code being tested works, regardless of it's dependencies.

The theory is that - if the code I write works as designed and my dependencies work as designed, then they should work together as designed.

This mock object will expect a certain method to be called with certain parameters a

Mockito facilitates creating mock objects seamlessly. It uses Java Reflection in order to create mock objects for a given interface. Mock objects are nothing but proxy for actual implementations.

nd when that happens, *it will return an expected result.*

**\*\*\*\*\*\*\*\*\*\*\*\*\*\* Mocito**

Let’s look at the role of the annotations in the above example.

* @Mock will create a mock implementation for the StockService
* @InjectMocks will inject the mocks marked with @Mock to this instance when it is created.
* So **when** or **where** are these instances created? Well, it is done in this line, which resides in the setUp method:

MockitoAnnotations.initMocks(this);

* So these instances would be created at the start of every test method of this test class.

Great! Now we have successfully created and injected the mock, and now we should tell the mock how to behave when certain methods are called on it.

The when then pattern:

* We do this in each of the test methods. The following line of code tells the Mockito framework that we want the getPrice() method of the mock service instance to return certain value based on input instance.

when(stockServiceMock.getPrice(googleStock)).thenReturn(50)

* when is a static method of the Mockito class, and it returns an OngoingStubbing<T> (T is the return type of the method that we are mocking — in this case, it is boolean).
* The following are some of the methods that we can call on this stub
  + thenReturn(returnValue)
  + thenThrow(exception)
  + thenCallRealMethod()
  + thenAnswer() - this could be used to set up smarter stubs and also mock the behavior of void methods as well ([see How to mock void method behavior](https://javacodehouse.com/mockito-tutorial#void)).
* Simply putting this all in one line again: when(dao.save(customer)).thenReturn(true);
* Do we really need to pass in an actual customer object to the save method here? No, we could use matchers like the following:when(dao.save(any(Customer.class))).thenReturn(true);

When() - Mockito adds a functionality to a mock object using the methods when()

Verify () - Mockito can ensure whether a mock method is being called with reequired arguments or not. It is done using

Mockito provides the following additional methods to vary the expected call counts.

* atLeast (int min) − expects min calls.
* atLeastOnce () − expects at least one call.
* atMost (int max) − expects max calls.

Mockito provides Inorder class which takes care of the order of method calls that the mock is going to make in due course of its action.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Coverage**

Junit we are testing individual units and target is to write test case for all paths, all logical conditions and possible exceptions.

So in enterprise project its bit difficult to determine just simply looking at Junit whether we have covered all scenarios its because we have many classes, each class can may have multiple methods which inturn can have multiple logic paths and logical conditions...

There are some plugins that are available in market which you can use within IDE which would tell us coverage.

Coverage is noting but when Junit test case runs what all code line it went through.

In general our goal is to have 70% of coverage.

That means 70 % of your should be covered by Junit

There are some situation where we can't write test case -

There are some situation we should not target to write test cases - like get/set methods

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TDD**

Traditionally we used to write test case after we coded the functionality.

So take a example where in we want to implement portfolio - we used to write all code and once that comp

But the issues with that approach was that we used to write test cases to support whatever functionality that is coded.

Because same developer we have coded functionality generally used to write test case

And it general tendency that you see implementation and to match that you write test case

So if we have coded functionality wrongly then we used to write test case to support that

Which is not right...

That's where concept of TDD - Test Driven development was coined

In this approach we write test case first and then write code to pass that test cases

So test case are considered requirement and when we write initially all should fail because we have yet not implemented functionality.

Once we start implement functionality test cases should start pass one by one.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Benefits**

Junit is very simple, Its installation, configuration and writting test case is simple

Its open source and can be used free of cost

We can write test cases, which can check their own resulst and provide use feedback in pass/fail

We can setup them to run automatically. Hae you header build tools like Jenkins, Bamboo which are used to build you code and deploy to test/pilot/prod. We can do setting to run test case automatically there.

So as soon as you commit code, Bamboo job will trigger, it will first compile all your code, then will try to run all you test case if all passes then deploy code to shared environment. If any test case fails then it can fail your build send email to developer and will not deploy your code.

**Saves defect cost -** Its preventing mechanism. It helps to detect defects early in cycle

The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels. Compare the cost (time, effort, destruction, humiliation) of a defect detected during acceptance testing or when the software is live.

Unit tests are also great for **preventing regressions** – bugs that occur repeatedly. Many times there’s been a particularly troublesome piece of code which just keeps breaking no matter how many times I fix it. By adding unit tests to check for those specific bugs, you can easily prevent situations like that. You can also use integration tests or functional tests for regression testing, but unit tests are much more useful because they are very specific, which makes it easy to pinpoint and then fix the problem.

Unit testing **increases confidence** in changing/ maintaining code. If good unit tests are written and if they are run every time any code is changed, we will be able to promptly catch any defects introduced due to the change.

**Development faster** - If you do not have unit testing in place, you write your code and perform that fuzzy ‘developer test’ (You set some breakpoints, fire up the GUI, provide a few inputs that hopefully hit your code and hope that you are all set.) But, if you have unit testing in place, you write the test, write the code and run the test. Writing tests takes time but the time is compensated by the less amount of time it takes to run the tests;

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Best Practices**