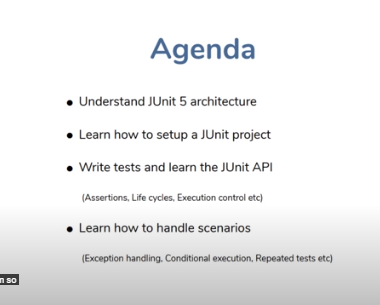
JUnit 5-( Java brain youtube channel on junit)

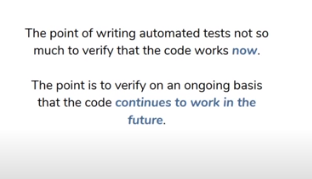
Link: <https://www.youtube.com/watch?v=2E3WqYupx7c&list=PLqq-6Pq4lTTa4ad5JISViSb2FVG8Vwa4o>

Git: <https://github.com/koushikkothagal/junit-5-basics-course/blob/master/pom.xml>



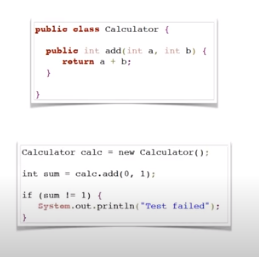


Difference between manual test and test using code: when you write a test case someone else makes a change to your code they're going to be running your test case and if they break the functionality your test case is going to catch it and this is not something you can do with manual testing

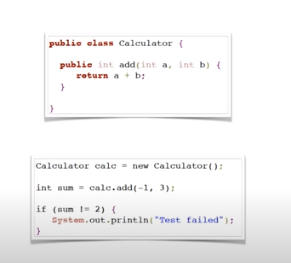


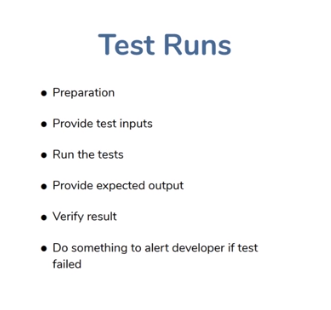


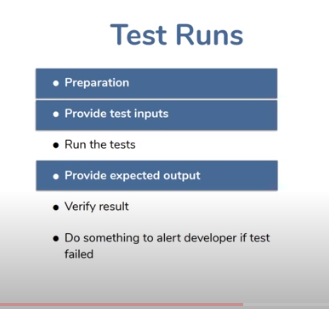
Code+test code



Another test case with –ve number

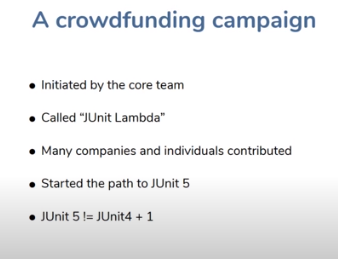




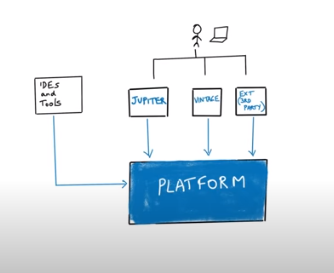


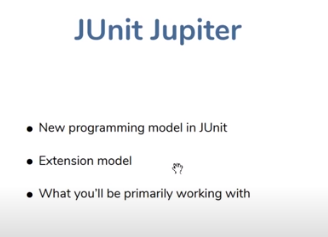
Why JUnit 5



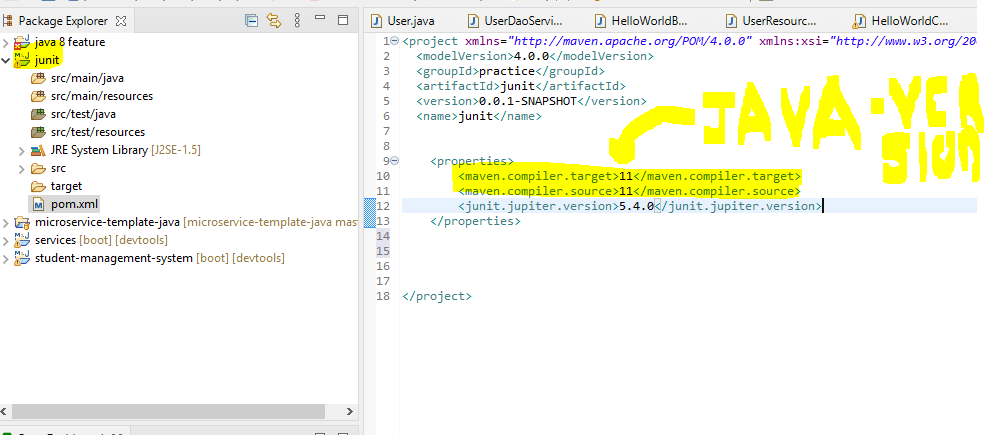


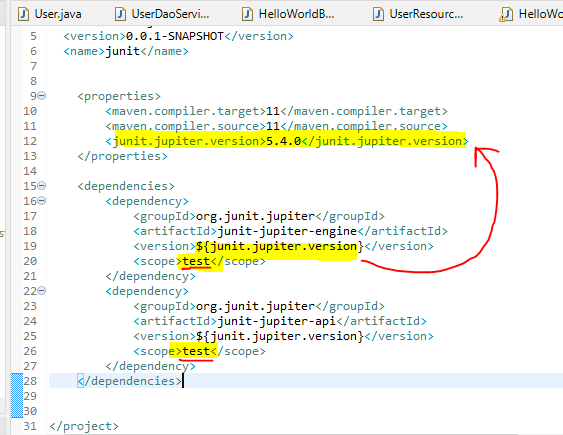
JUnit 5 Architecture



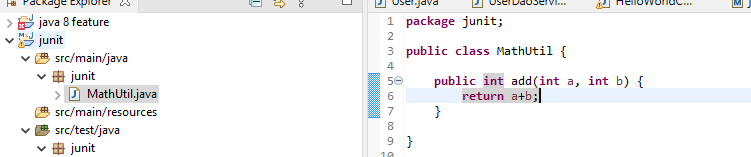


Created a simple maven project

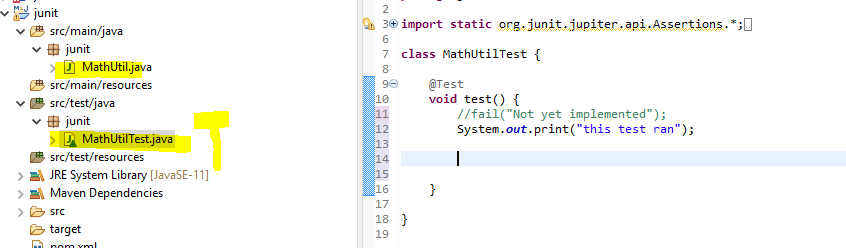


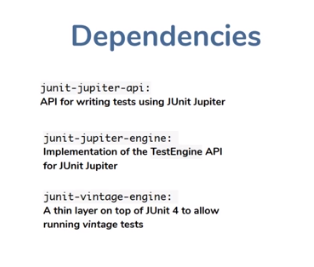


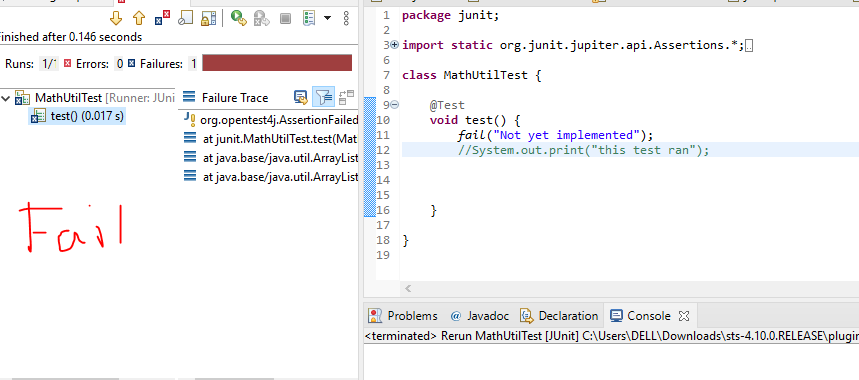
Why <scope> : <https://stackoverflow.com/questions/26975818/what-is-scope-under-dependency-in-pom-xml-for>

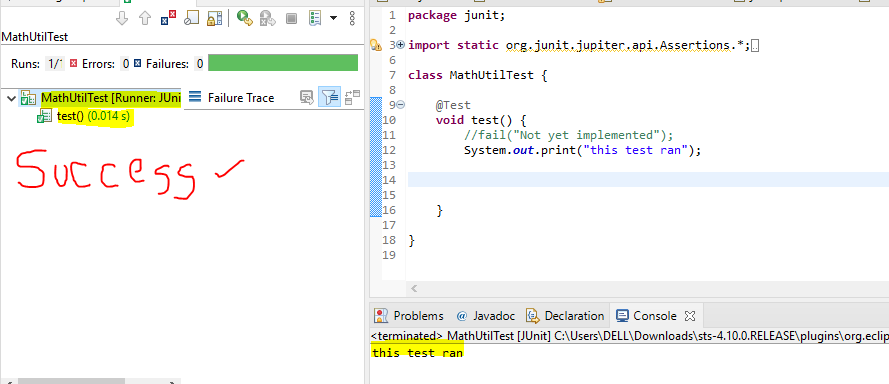


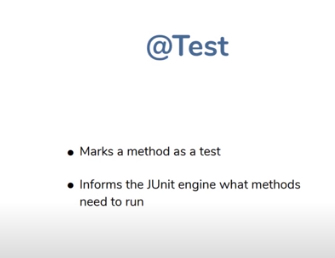
Right click on the class -> new ->junit test. We can create test class like this.



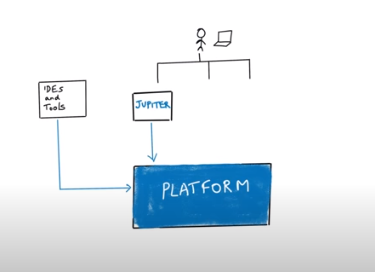


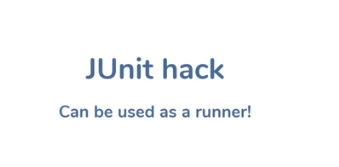


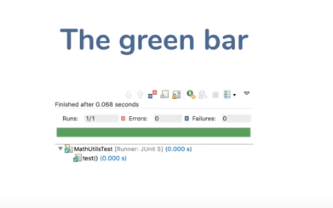




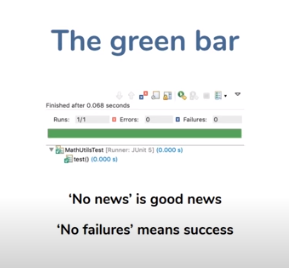
If we right click on project -> run as-> Junit: juniy engine will look all the classes for the @test annotated methods.



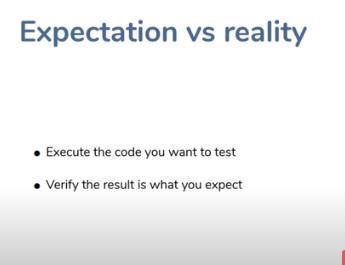


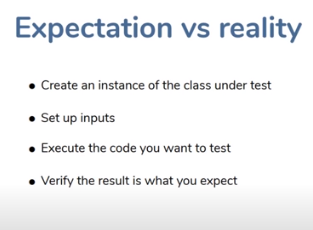


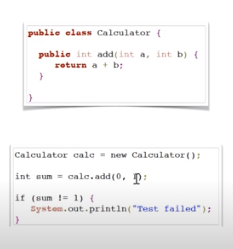


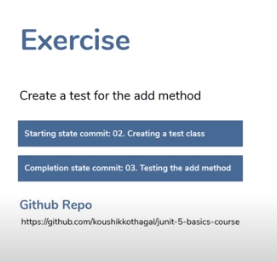


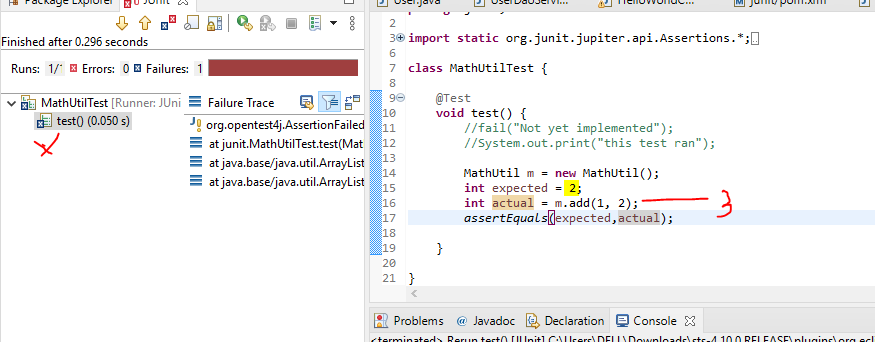
Using Assertions

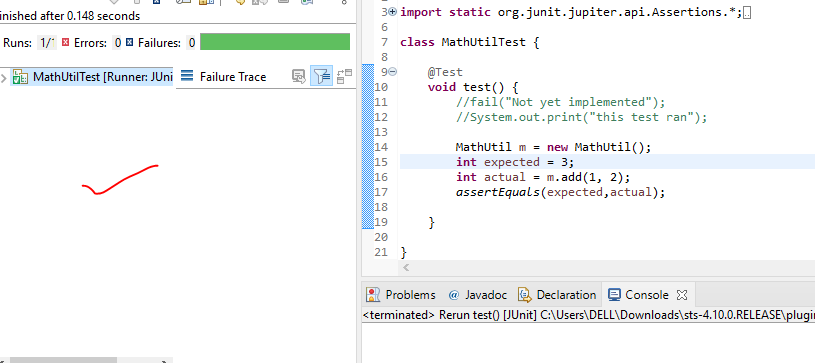




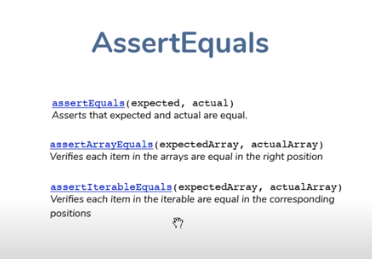


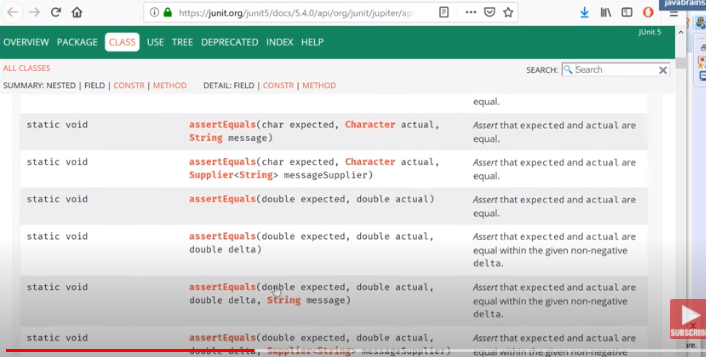


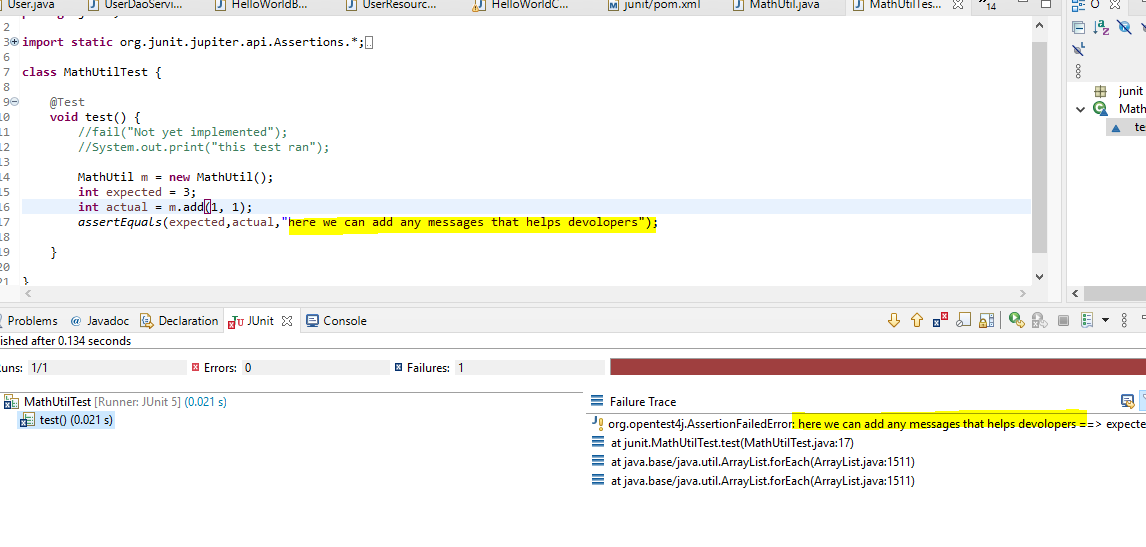


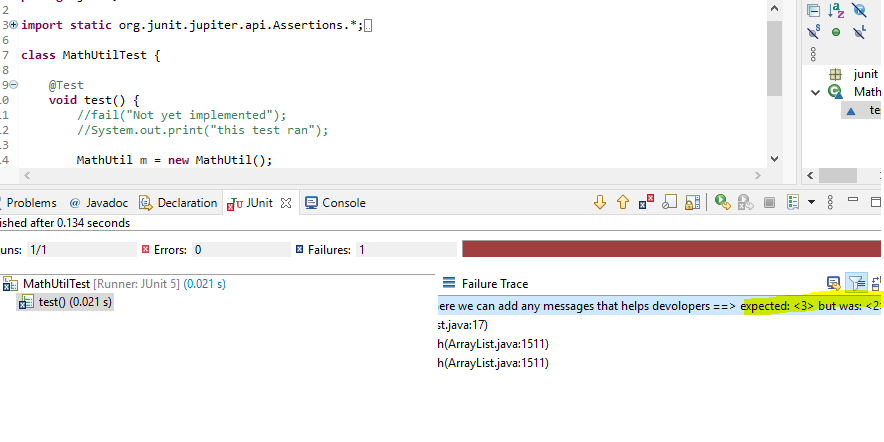


Assert methods

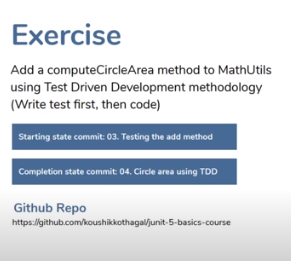




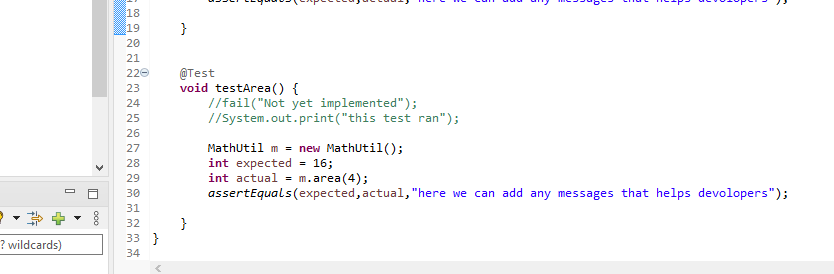


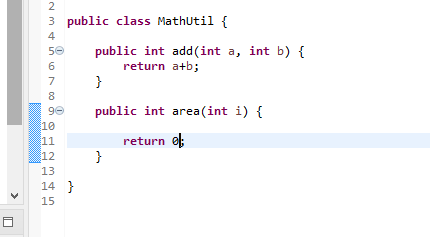


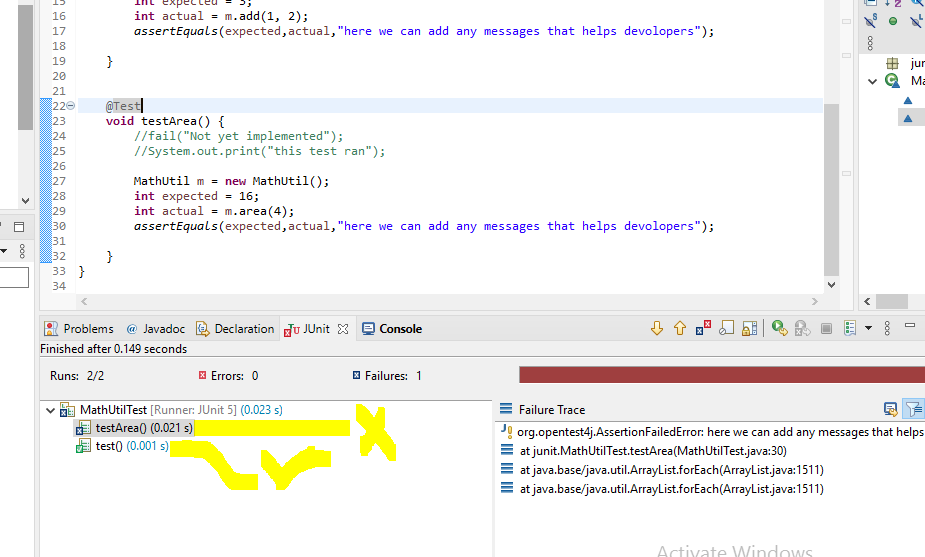
Test driven development with JUnit



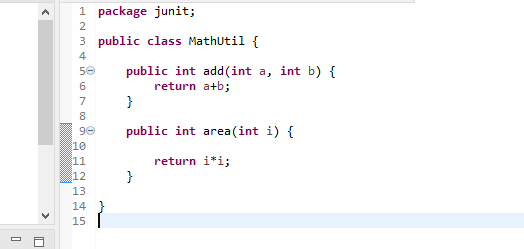
**Write test first then actual code, so the test turns green**







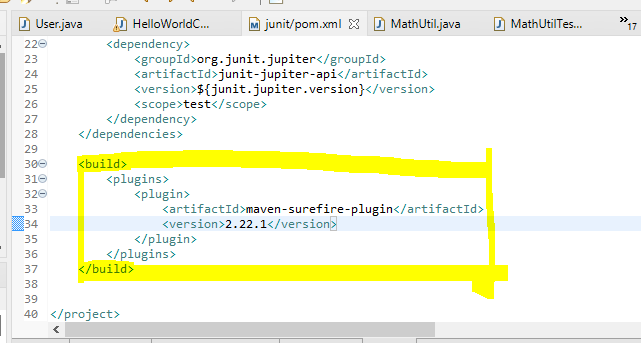
**Now we edit the are function**





Maven surefire plugin integration

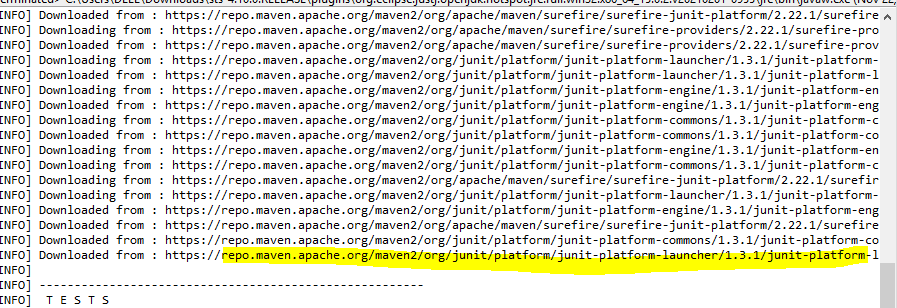
**So if we are running CI/CD pipieline and all we can’t run as junit test . that time we need to add some plugins into the pom.xml. that plugin is called maven surefir plugin.(so this is the plugin used in maven to run Junit tests. So here we ara not take any help from Junit integerated with IDE)**



**How to run : right click on project -> run as -> maven test -> just put test and run**

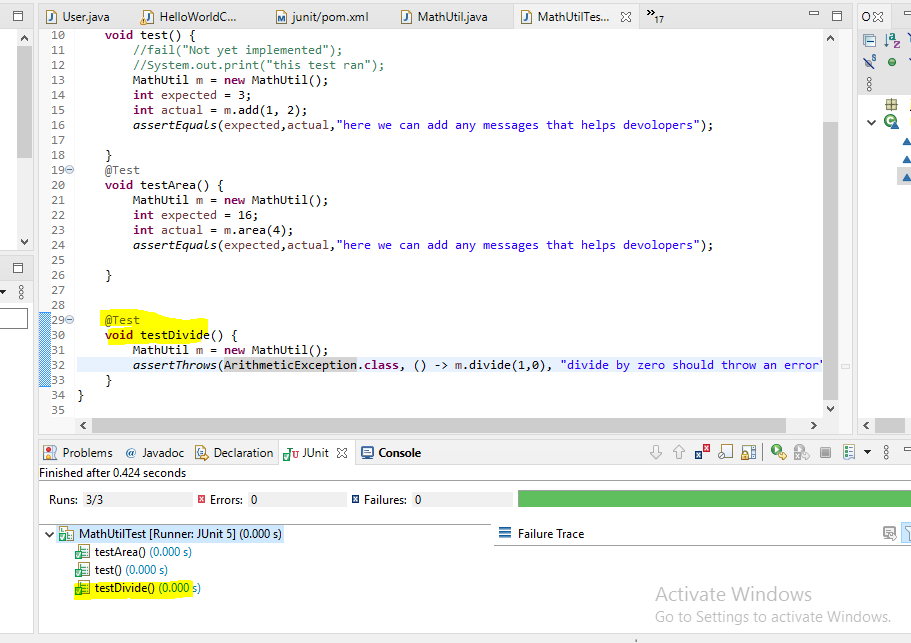


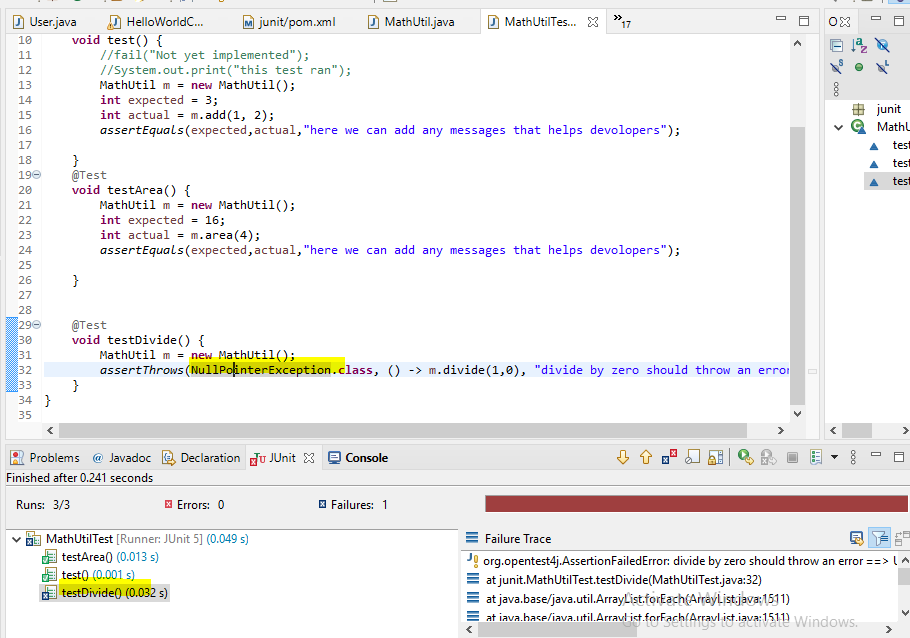
**HERE WE can see lot of junit telated stuff is downloading as the part of build**



**Like this we are running in CI/CD pipeline.**

Asserting exceptions with assertThrows





Life cycle and test antipatterns to avoid

**life cycle is the process in which the test instance is created it's managed and destroyed**

**so unlike other classes in your code base when you run this thing when you right-click and say run as and run as JUnit tests whose initiate whose initializing this class? this needs to be an instance right, mathsutilstest needs to be an instance somebody has to say new mathutilstest who's doing that? J unit is managing the lifecycle of this class.**

**There shoudn’t be any order to execute the method (if order is there dependency is there )**

**But for order we have an annotation @order**

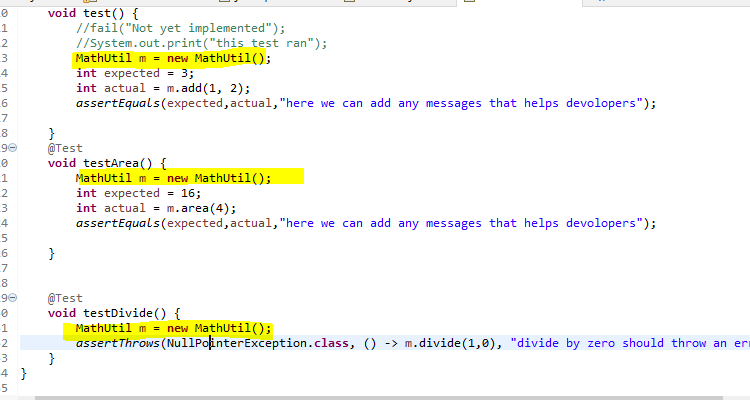
**J unit actually creates a new class instance for every test run for every method run rather so here's what happens and before I tell that let me tell you one other thing I was talking about the order right one very common trap that people fall into when they're writing tests is they create the state outside the method which is very very bad**

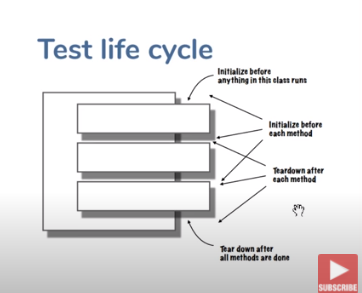
**Junit 5 create new instance of test class (here muthutiltest) for every test method.**

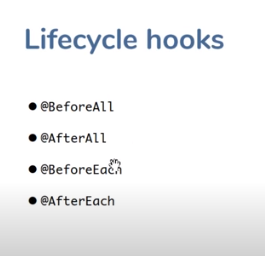
**And don’t use variable also in test class, (in function we can have). If add instance variable it is sort of dependencies**

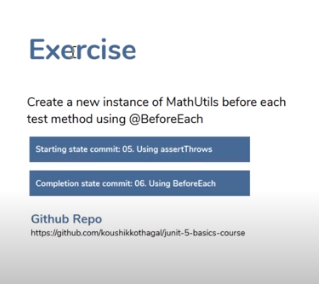
**j-unit creates a new test instance every time a test method runs tests add runs well there is a new math utils test test divide runs there's a new math utils test all right so it's a new instance for every method so this wouldn't even work anytime you're checking this you are going to get the default value right so it helps in that sense so that's another thing that you shouldn't be doing all rig**

**that’s why we are creatin new instance of MathUtil.**

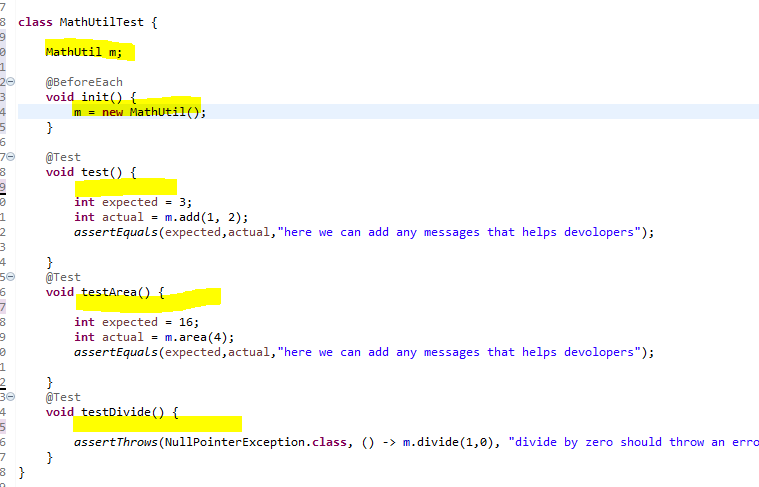


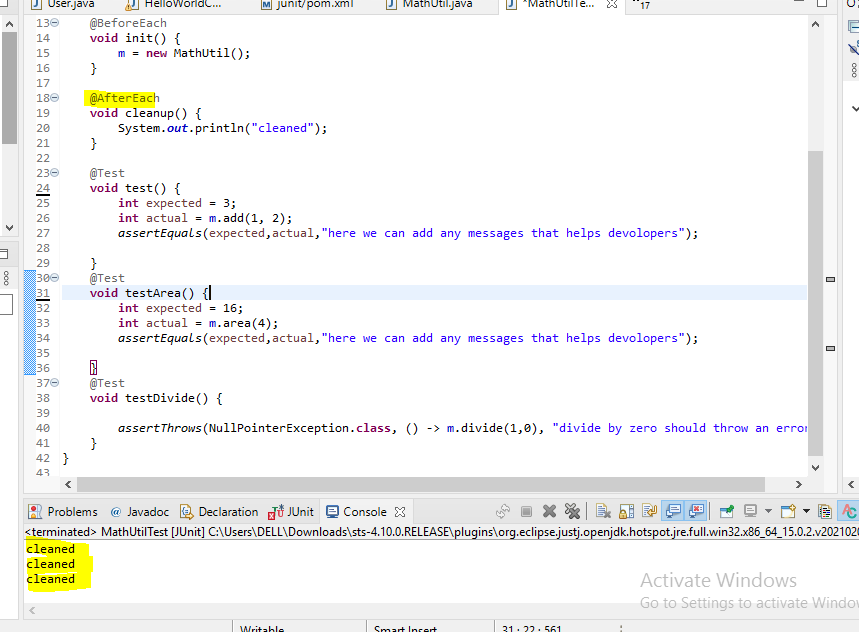


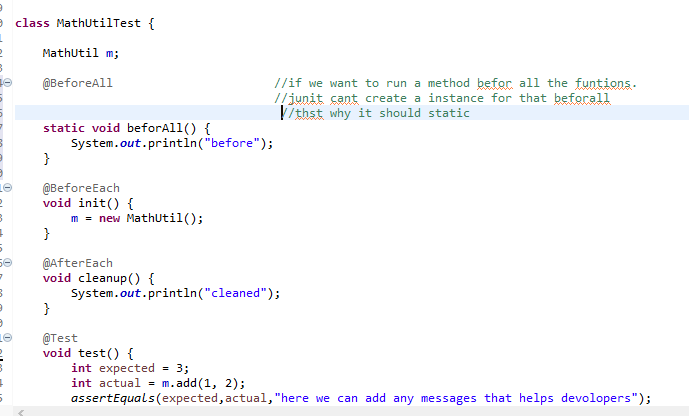


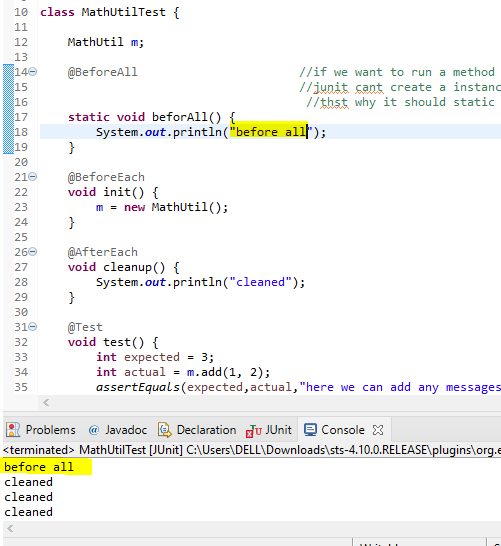


**We removed object initialization(MathUtil m = new MathUtil(); from every function, and added in the init(), so init() method will run before every mehod, so init() wil run 3 times, because we have 3 test methods)**







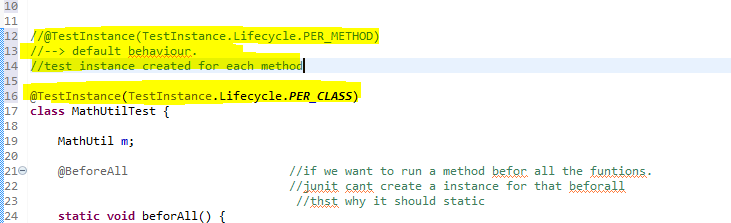


Changing default TestInstance behavior

**Junit creating separate instances for all the test methods. We need to change that behavior. We can change this behavior**





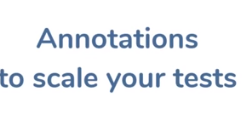


**In this case we can have member variable and all.**

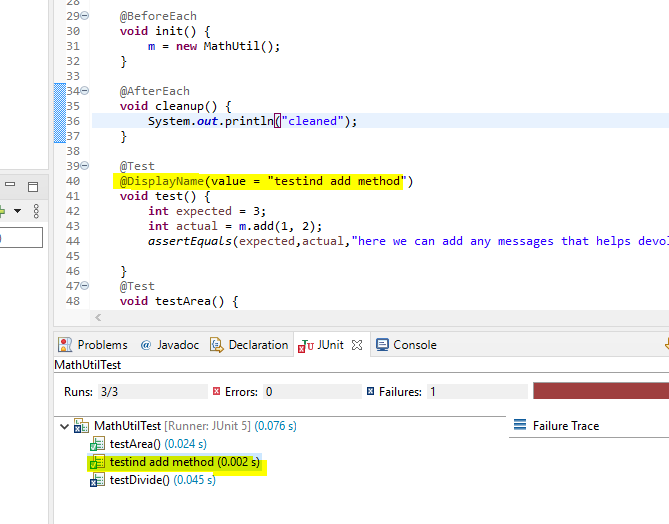
**Generally this is bad practice**

**In this case beforeAll not need static modifier also**

Using DisplayName and Disabled annotations

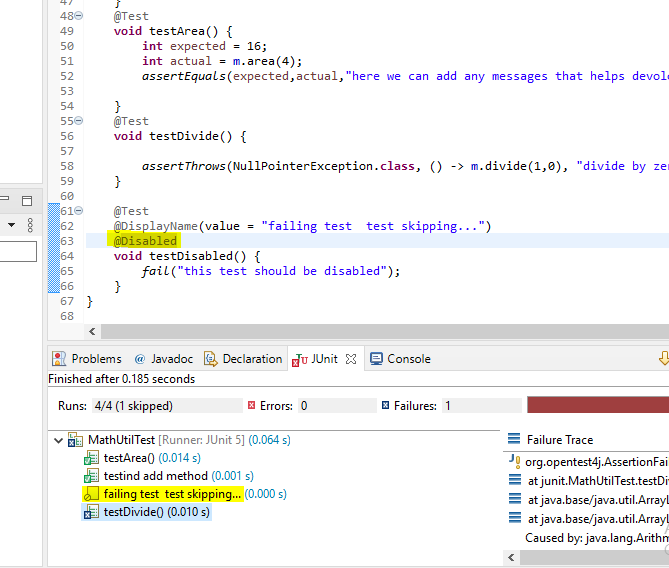




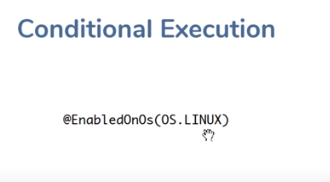


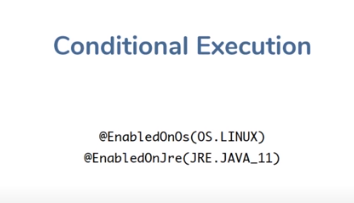


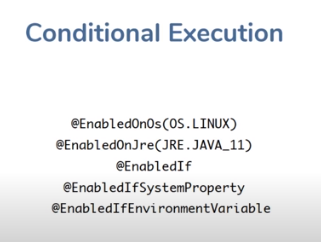
**So we can skip the test (the test it fails everytime) using disabled**



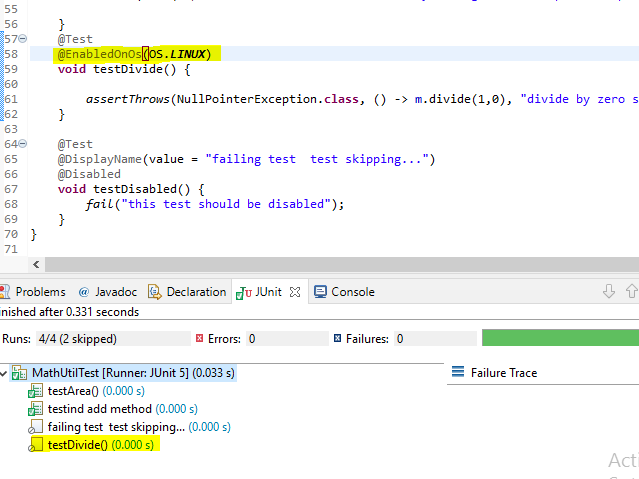
Conditional executions and assumptions







**In below example, the testcase(testdivide) will only run at LINUX machines.**

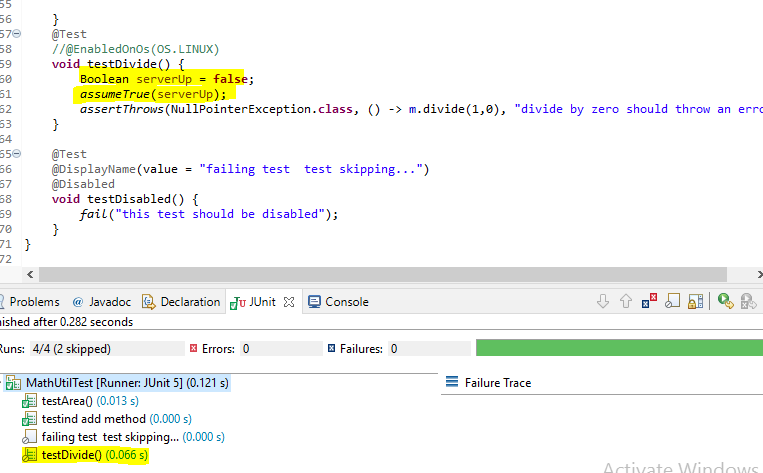


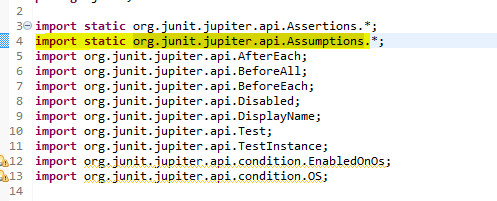
**AssumeTrue(boolean) Function**

**It used disable or enable the test function**

**If the boolean value is true then only it will run**

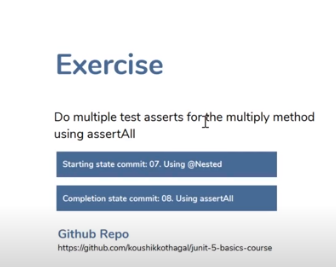
**So some case the the test case depend upon the server up or down if the server is up then run otherwise don’t run**

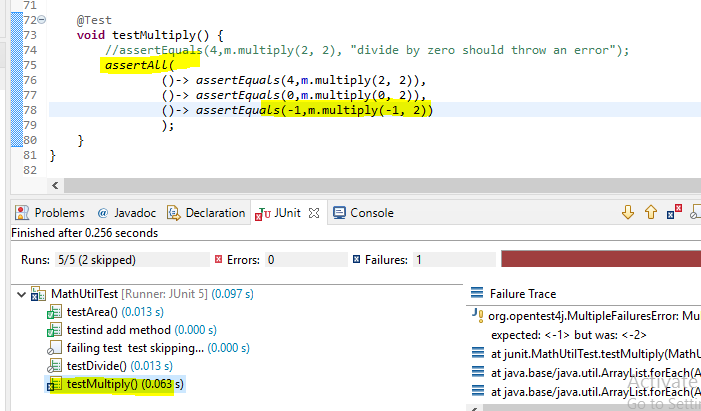


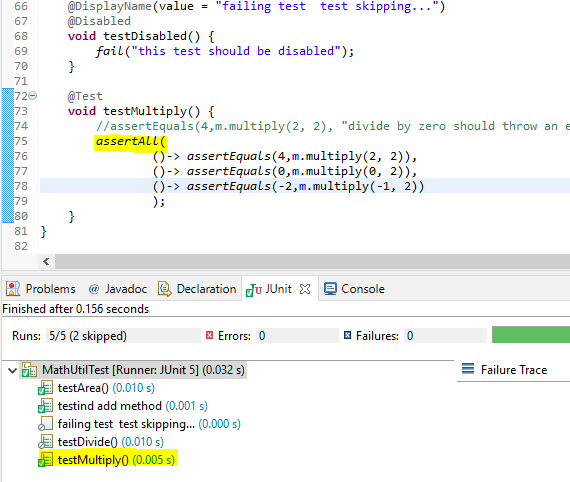


Using AssertAll



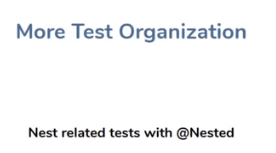






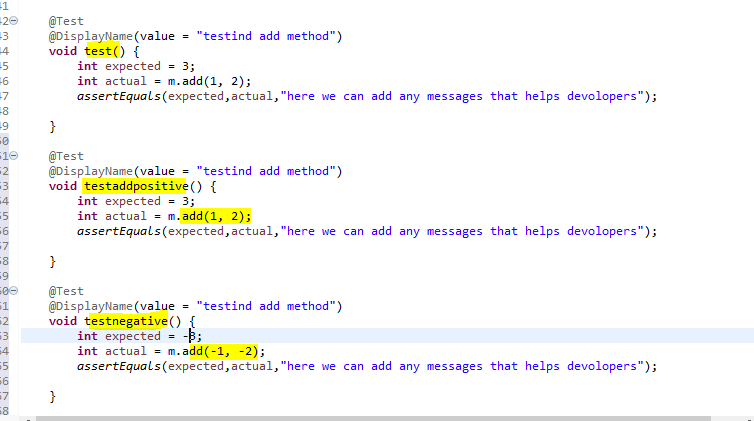
Writing nested test classes

**somehow group those things together and say these are test a single thing but then each of those tests are testing for different scenarios, using @Nested**





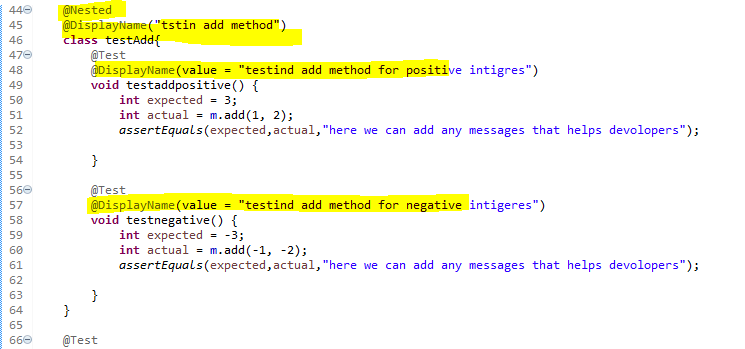
**We can do like below, but no.of testcase will increase**



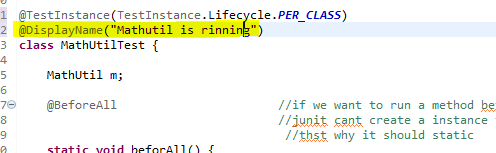
**So,**

**We gruped those test cases in a nested class**

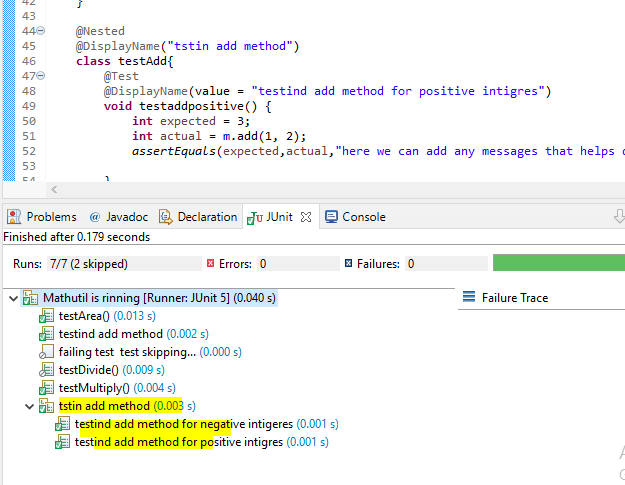
**Like below**



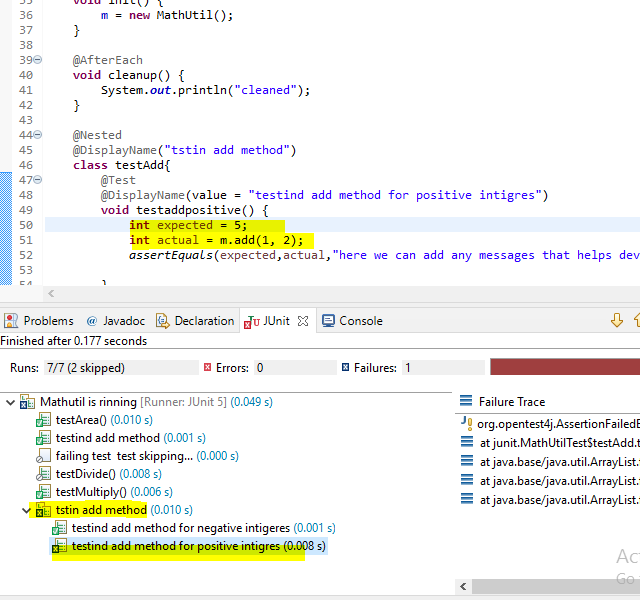
**And we added @Displayname on the main class also, si it become quite readable**



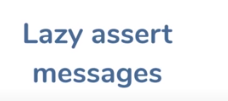
**So when we running the result looks like**



**If one fails,its lokklike this**



Using supplier for assert messages

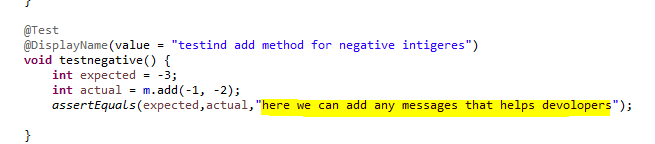


**not only can you pass lambdas to assertions in assert all you can actually pass lambdas into any assertions for the statement for the message that the test displays if the test were to fail.**

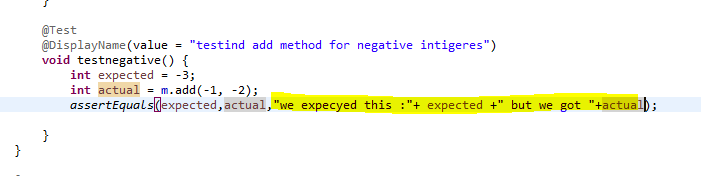
**what do I mean by that now take a look at this message that we show when a test fails this particular thing assert equals I'm running a particular test and then I'm putting this message over here it says hey Junit if this test were to fail show that message right**

**when is that message gonna be even applicable the message is gonna be applicable when the test fails if creating that message involves consuming some resources right**

**so let's say it's a bit expensive to create that string message to show then it's kind of counterproductive to create that message every time for every test even if the test completely succeeds and there's no need to show that message right**



**Changes to**



**between an example let's say you're doing a bunch of computations here let's say expected equals -2 sure because this I'm kind of computing this message so that I can provide a more detailed message it doesn't make sense in this context because j-unit is going to give you that anyway but let's say let's go with me here let's say I do this expected and then actual should return some expected but you're done actually**

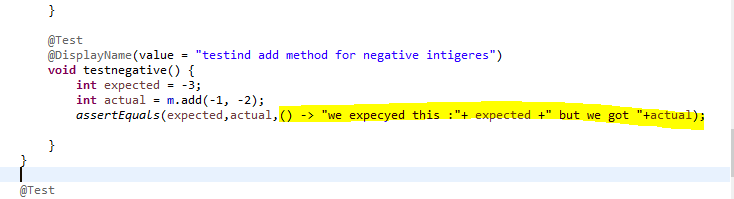
**all right let's say this is my was my test this is what I'm trying to do now every time this test runs this string is being created right so there is a string should return some and expected is appended to it and then but return is appended to it and then actual is appended to it there are optimising optimizations there but let's assume for the sake of this discussion that this is a an expensive piece of string to compute right this string is gonna get computed then this class runs no matter whether the test passes or fails because this is getting computed even before the value is being compared this string is being passed to the assertequals method the assertequals method gets these three arguments and then it asserts and then uses the string only when the assertion is failed**

**so you are computing it irrespective of whether the test passes or fails now what if this is an expensive piece of string to calculate but if that needs to be deferred what if you want to calculate this lazy**

**but guess what you can create a lambda and pass it over here the lambda is a supplier so basically it's something that returns a single value as long as you pass in a lambda which returns a string what Junit is gonna do is it's gonna say hey I I know you haven't given me a string but you've given me a method to execute which is gonna give me a string I'm fine with that I'm only gonna execute that method if this test were to fail if the test passes**

**I'm not gonna give that method so guess what this string computation if you put it inside a lambda it's not gonna get call it the computation is not gonna happen until and unless the test fails right so it's very simple to change this all I need to do is convert this to a lambda all right now this there's going to be a function that that assertequals is gonna take keep it with it and it's going to execute it promises to execute that function only when the test fails right small optimization but this is also something that our gin unit can do for you and that's something that's important to remember when you have when you're dealing with when you're dealing with stuff that is a little bit complicated to calculate you don't want to do that every time JUnit 5 Basics**

**So this is the more optimized code**



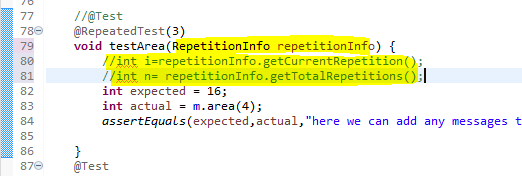
Using RepeatedTest





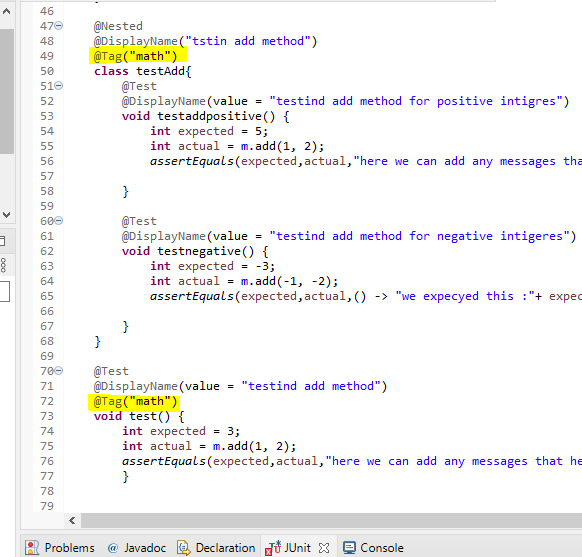
**If one fails the mail node will also show the failed status**

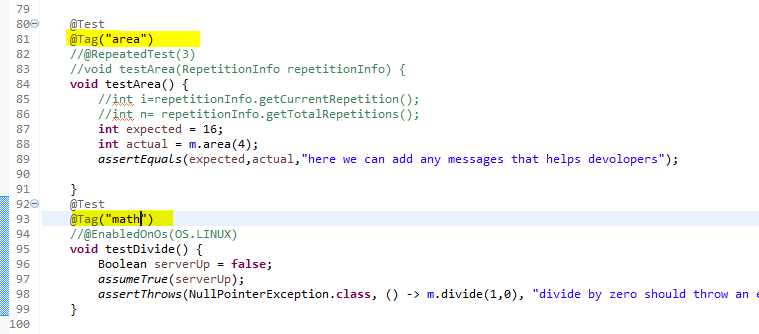
**And we can do some on the basis besis of repeatation info like below(means we can costamize each repetation)**



Tagging tests with @Tag

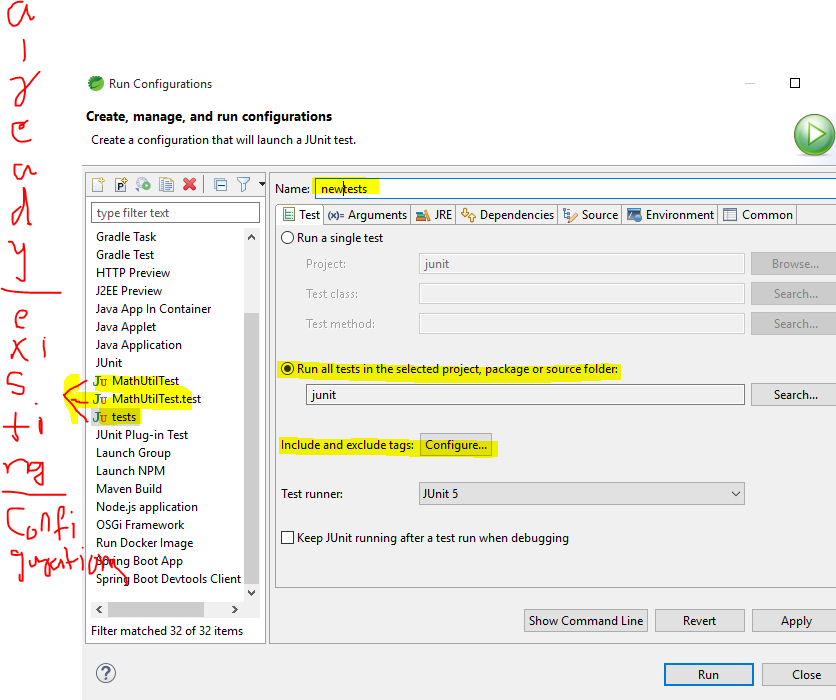
**We are grouping using tag, so we ren selefctively using tags, so I added tags like below**





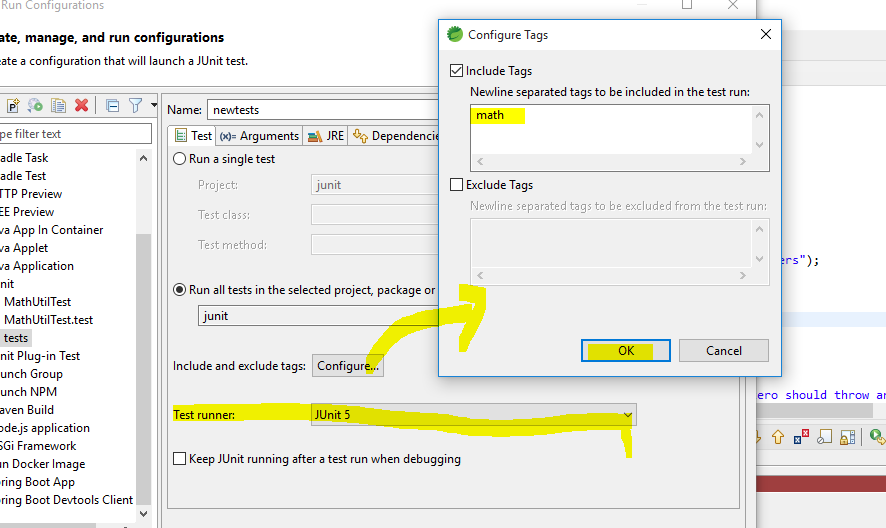
**How to run selectively in eclipse**

**Gp torun as configuration - > new configuration - > give a name**

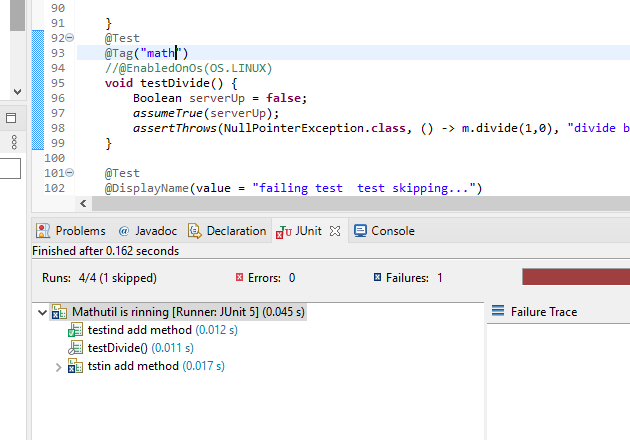


**Select “Run all tests in the selected project”**

**Include tags (click on configure)**



**o/p: only math tag test case ran.**



Using TestInfo and TestReporter

**In previous demo, we implemented @repeatedtest, so there we used a parameter “RepeatedInfo” that actually we are saying ti Junit framework that I need this dependency. So there we are using dependency injection**

**Junit providing suchtypes of classes, it will inject you the dependencies if we declare those classes like we declared the RepeatedtestInfo**

**So like that we have TestInfo and TestReporter(these are dependency injected classes)**

**TestInfo contains the information abot the tests**

**TestReport gives an access to the filnal reports where Junit reporting the results**

