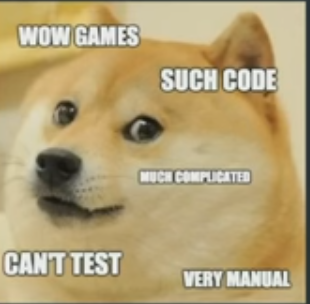
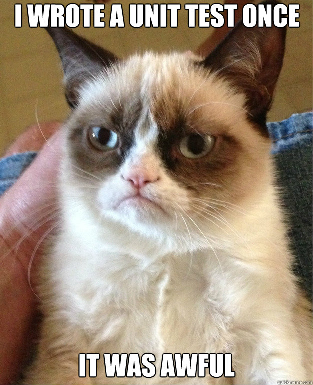
# Introduction

The unit tasting is useful and cool feature. That I should use so I explore and study it deeper and deeper

But



# Unity Test tools

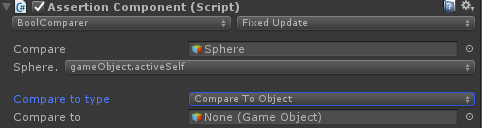
## Resources

1. [Video](https://unity3d.com/learn/tutorials/topics/production/unity-test-tools) with introduction (My review is 7/10 good introduction but a lot of talking about nothing)
2. [Video](https://www.youtube.com/watch?v=_OYojVTaqxY) from Unity tech(Just talking on conference my review 6/10)
3. [Article 1 basics](https://blogs.unity3d.com/2014/05/21/unit-testing-part-1-unit-tests-by-the-book/) (8/10 good introduction and interesting point of view)
4. [Article 2 Mock included](https://blogs.unity3d.com/2014/06/03/unit-testing-part-2-unit-testing-monobehaviours/) (9/10 Mind blowing shit)
5. [Video Unity and TDD](https://www.youtube.com/watch?v=GIJptHunxow)
6. [Documentation](https://bitbucket.org/Unity-Technologies/unitytesttools/wiki/Home)

## Assert Component



Simple way to check is everything OKAY. Way of testing that event monkey can handle.



Assert is simple check that allow me to assert(Утверждать) something.

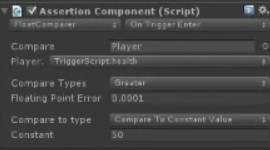
It works in play mode. If some of my assertion fails It will drop an exception.

***Tip:***

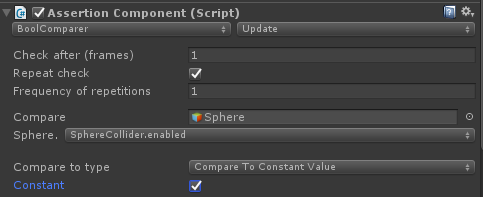
***If you pause game at moment the problem come.***

### Example

Simple example to check will palyer HP be grater then 50 after hit. If not we have an exception.



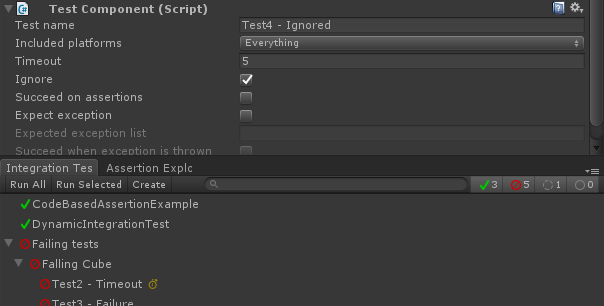
Another example you can check when some component is turned off



## Integrations Tests



Is way to test is everything feed together



So our test is 

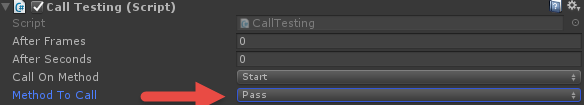
And some object from inside hierarchy should call



Or

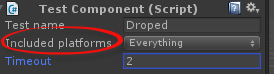


Or it can be done by using Call testing . You can call pass or fail after using components



### Platforms

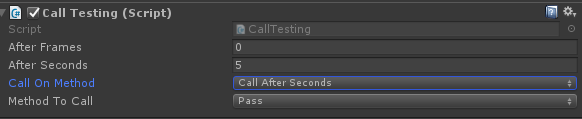
The big plus is that you can run test and make sure it works on different platforms.



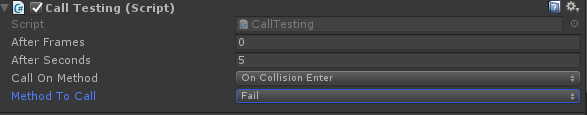
### Testing Ways

#### Call Testing

Using call testing. And combining it to fail and pass.



In this particular scenario the test will be passed in 5 seconds

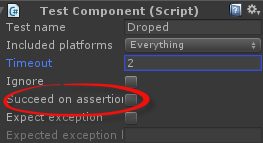


And in this case test will fail when collistion will happen.

If you will combine them the test will be passed after 5 seconds. And failed if object will hit something

#### Assertion

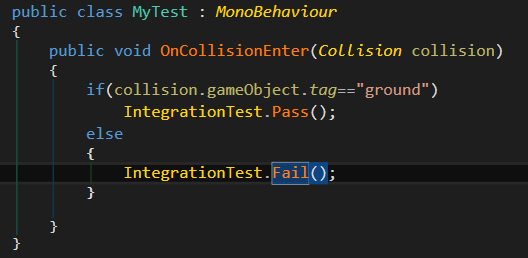
You can easy combine it with [assert component](#_Assert_Component).



It is way if all assets are true we success and if some of them wrong we fail.

#### Custom Script

Or you can use any custom script. To test any scenario you need.



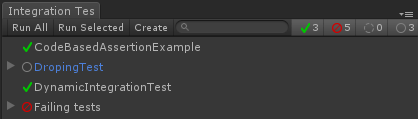
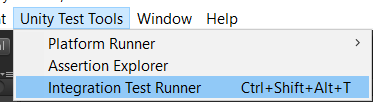
For example in this case the test will be passed if first thing you hit is ground and fail if we hit something else.

### Example

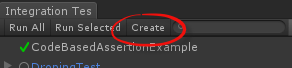
Simple example of creating the assertion test creation. The test after 5 seconds test is passed. But if we hit something test is failed



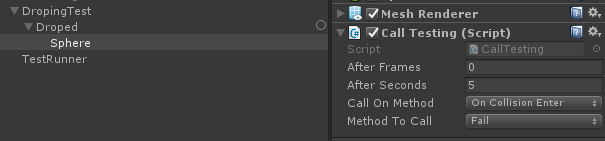
1. Open Integration test runner panel



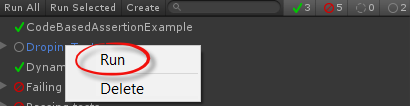
1. Click Create button



1. Add Sphere to the test 3d object sphere. And add call methods that represent your logic.(Or you can write your own look [Custom Script](#_Custom_Script) for more details )

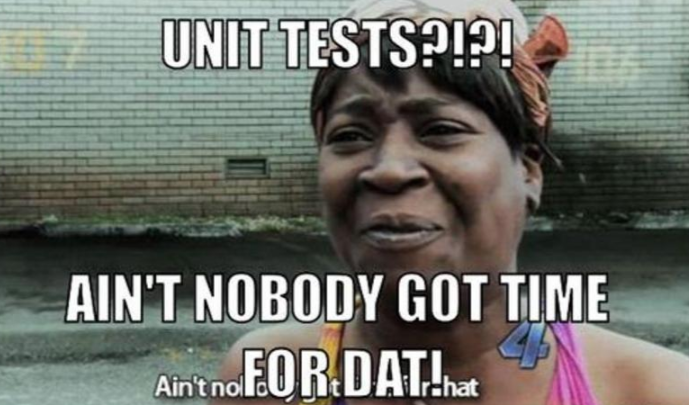


1. Click RMB on the test and run it.



1. Well Done!!!

## Unit Tests



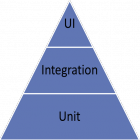


To prevent that use unit tests.

After some time it will become a habit quickly and you will write pro-automation designs without even thinking about it.

Normal C# NUnit tests that you can run from Unity ~~but much more easy to run it from Visual Studio~~. And in Unity you can test it with GameObject. And in Visual Studio you will get security error.

### Rolls and tips



**1Unit tests should be ground base of all types of test!**

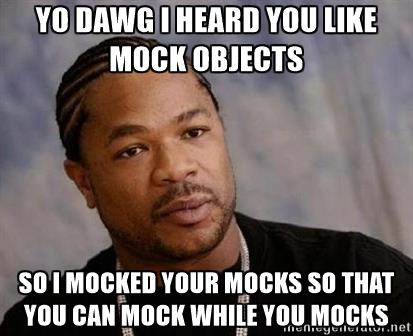
**2 Never make your tests dependent on each other.**

**3 Use** NSubstitute for mocks creation

**Tip**

**A true unit test needs to be fully automated.**

### **Mocks in C#**



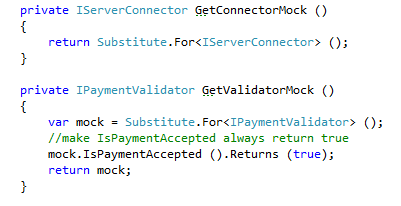
If we need to test



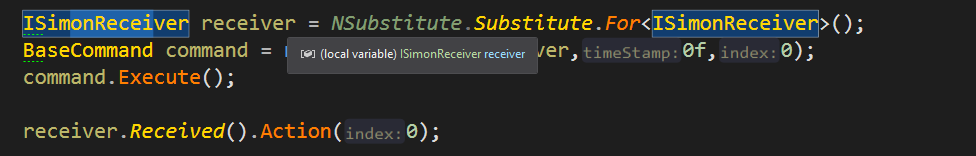
To made test simple program that order pizza from external resources.

You will have problems when you are testing with dependency injection. To prevent such problems use interfaces for testing. And development.

**Using Interface is Cool with ~~NUnit~~ NSubstitute!!!**



### NSubstitue check for execution

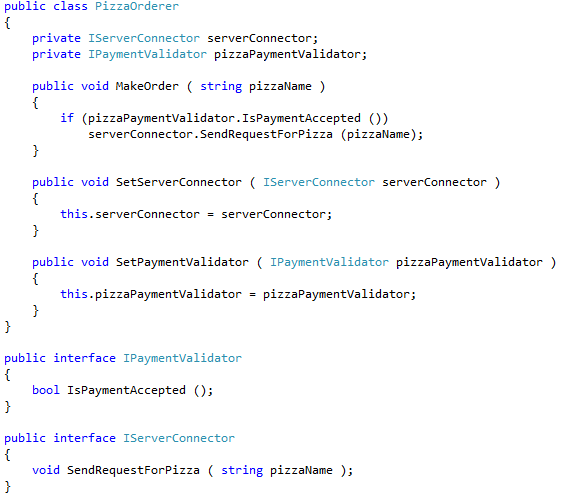


NSubstitue onEvent



#### **Mock Example**

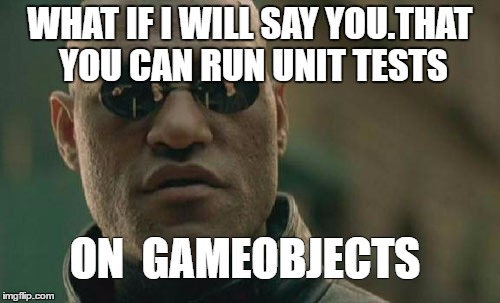
To decouple your modules use interfaces!

[](https://blogs.unity3d.com/wp-content/uploads/2014/05/pizza-example2.png)



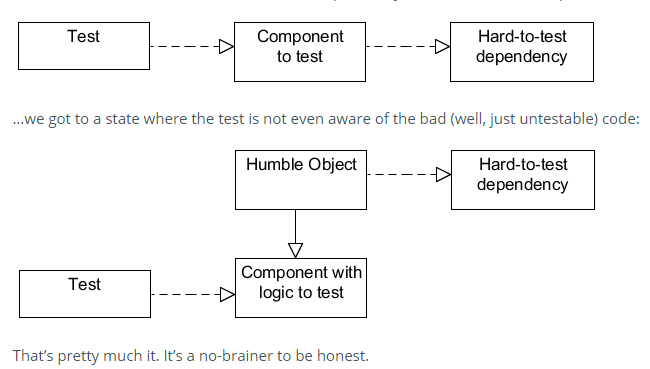
### Mocking in Unity or (THE HUMBLE OBJECT PATTERN)

The Humble Object Pattern



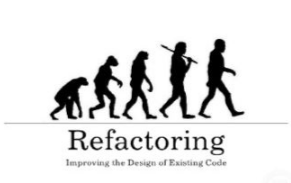
[Link to article I take it from](https://blogs.unity3d.com/2014/06/03/unit-testing-part-2-unit-testing-monobehaviours/)

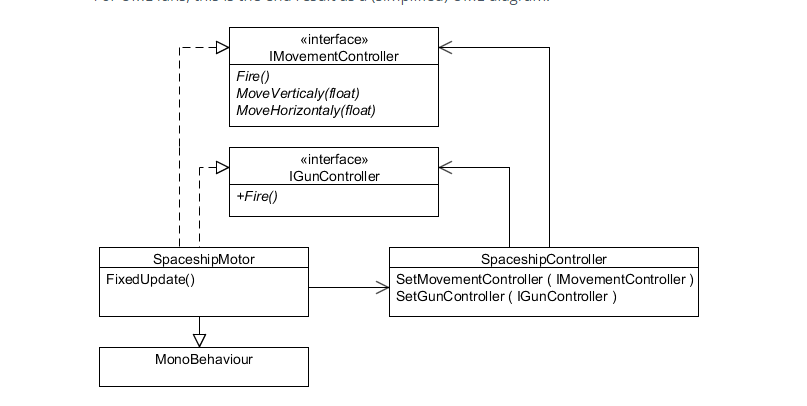
[Pattern](https://d.docs.live.net/1ec9d46a4910ba05/Documents/0%20Study/Notes/GameDev(Algoritms%20and%20Pattersn).docx) description



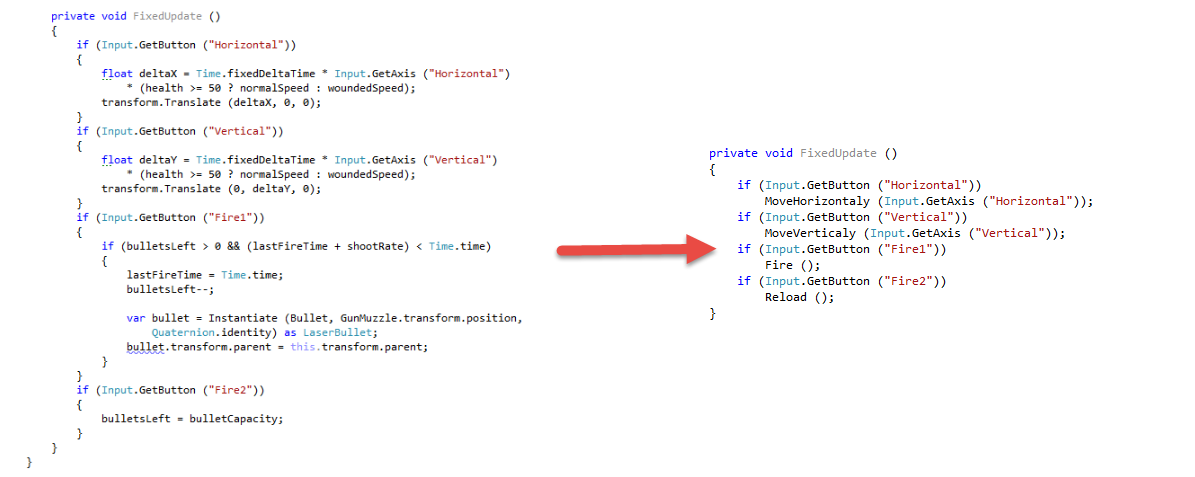
#### Example

To be able test MonoBehaviour. First thing you should do is.



IN simple word what we gone do is 

1. We should separate our logic from referencing Unity API.

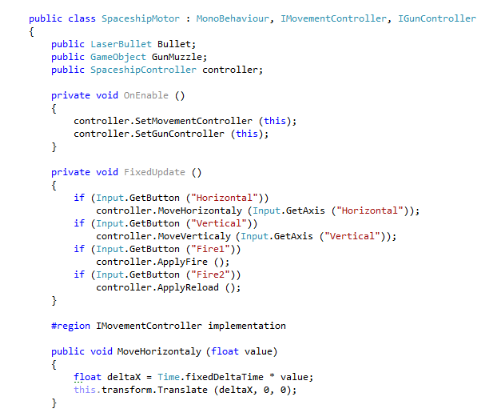


So now those methods can be call not only from MonoBehaviour Itself but also by our test

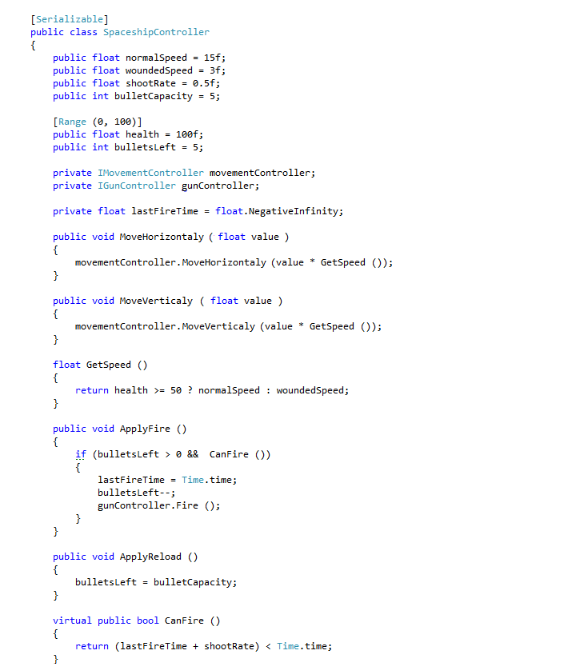
Next step is to separate logic and Unity API Refrences into different classes.

And for dependmency injection we will use interfaces

Unity API Refrencing class



Logic class



# C# NUnit language

Ctrl-T,R - to run one test

gT – to run all tests.

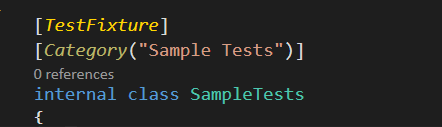


## Basics

First of all you need to add NUnit framework

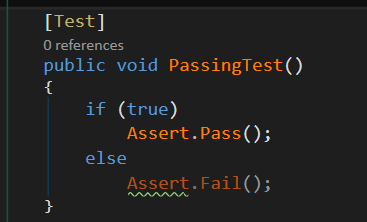
 - to made Unit tests

To test something you will need to create separate class with tag 



Also you can add category for naming and simplifying.

Then you create method with tag 



And inicide of it you test all functionality you need.

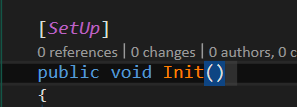
That simple even monkey can do it



## Attributes

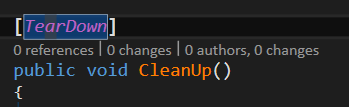
### SetUp

Will be run before each test to made setup for each test. You can connect to the DB open file system check internet connection e.t.c …



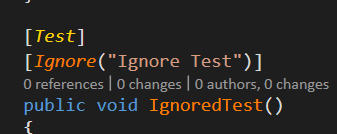
### TearDown

This method is opposite to the SetUp It is run in the end of each test. So you can clenup after your self. Close DB connection .Remove Temp files e.t.c…



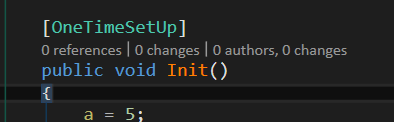
### Ignore

If test hurt your fillings just ignore him.



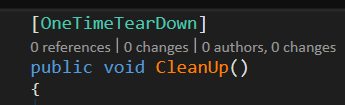
### OneTimeSetUp

Is one for whole fixture It will run once before all other test.



### OneTimeTearDown

It opposite to OneTimeSetUp it will run after all tests. So you can clean up shit after youself.



## Assert

Assert is the static class that contains a collection of methods for testing the functionality

 - to pass the test

 - to fail the test

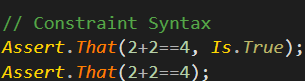
 - if you got no result from test

 - you can compare you actuall result to your expectation. Warning!!! It can hurt your fillings.

### That

Basically is the same as Assert but you can write longer check.

But you can write down spaghetti code to check everything you need at once.



#### IS

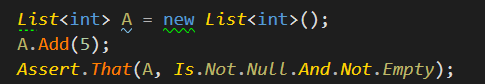
Is Null or empty check



Is Null



Is Not null and not Empty



Is Equal to



Is True



Is InstanceOf

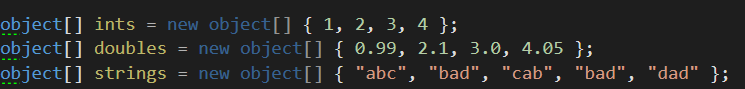


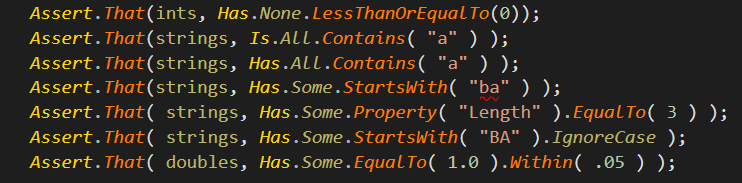
Is Grater Then and Less then



#### Has

Is for testing Collecitons

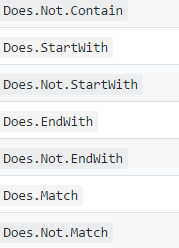




#### Does

To work with strings you can simply use does.



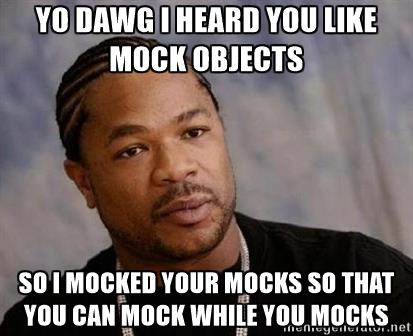


And you can use Not to check opposite

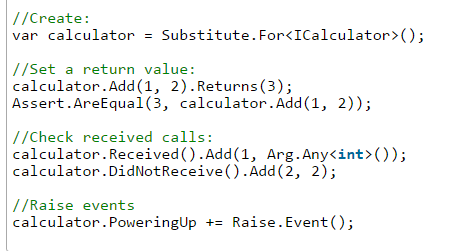
# NSubstitute language

[NSubstitute](http://nsubstitute.github.io/)

So NSubtitue is simple way to create mock object from your interface instead of actually write it.

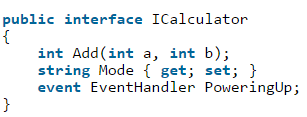


So it makes life a bit batter



## Example

Okay so we have this ICalculator example





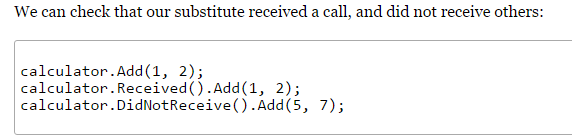
So step one on our road to success is to create Substitute



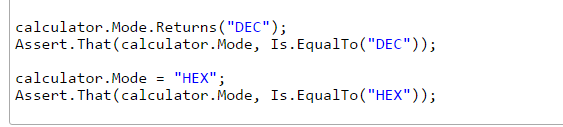
And now we wan’t to test our fake calculator so we say him what to do



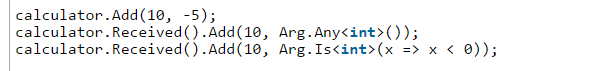
Now we wan’t to know is our substituation recived a command or not



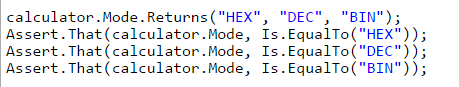
Also we can say what our properties are returning



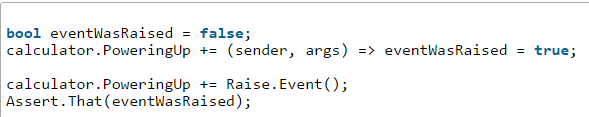
Specifie input parameters



Or you can even specify return



And even raise up our events.

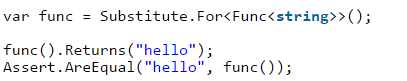


## Creating a substitute

For some inerface



For some function or delegate.



## Set return values

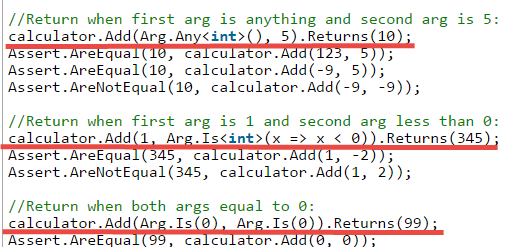
### For methods



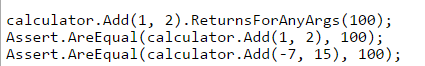
### For properties



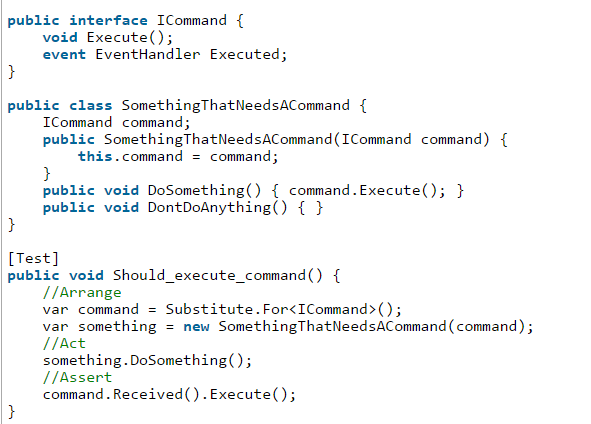
### Specific args



### For any args



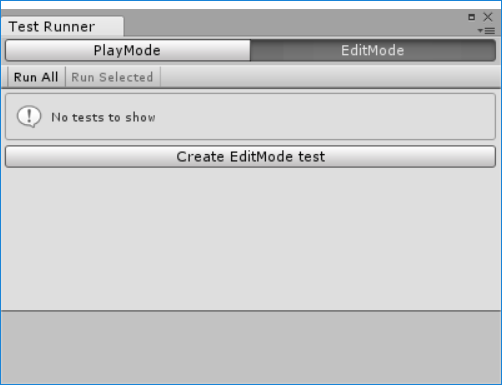
## Check is command was recived



# Practice

1. Check and take look on examples in Unity Test Tools pack and other types of example.
2. Read documentation about Unity Test Tools.
3. Thing ab out 3 different for each one
   1. Test Component
   2. Integration Test
   3. Unit Test
4. Create 2 different of each one for “Smash the Snack”.
5. Create 2 different of each one for “Pain of Teeth”.

# New Unity Test runner in Unity 2017



[Official Documentation](https://docs.unity3d.com/560/Documentation/Manual/testing-editortestsrunner.html)