**CSCD-371 .Net Programming Class notes:**

**Day 2**

\*\*Mark Capka taking notes feel free to make suggested edits

*NOTE: In class code is pretty fragmented to note particular pieces, all code is available under todays lecture in the class git i’ll add as I can and will remove this message once I add in the code. I will also upload everything to my github tomorrow for versioning control*

***Action Items from Thursday (today)***

* ***Assignment 1 (by Monday EOD)***
* ***Reading chapters 1-6 (by Tuesday)***
  + ***Pay attention to***
    - ***Pattern matching***
    - ***Properties (chapter 6, the whole thing and it is foundational) + coding guidelines***
    - ***Nullibility: Null values and not null values (chapter 2) (hint at end of lecture)***
* ***Submit google form if you haven’t already***
* ***Reach out via GitterChat if needed***
* ***ATTEMPT HOMEWORK BEFORE THURSDAYS TO ENSURE YOU HAVE TIME TO ASK QUESTIONS***

***Action items From Tuesday*** *Clone to your fork first*

* *Submit the google form for submitting your github id*
* *start reading chapters 1-6*
* *confirm your tools*
* *Start looking at/working on Assignment1*

**Instructors:**

Email address: **EWU-Instructors@IntelliTect.com**

**Mark Michaelis**

**Michael Stokesbury (Mike)**

**Kevin Bost**

**Github Link:** [**https://github.com/IntelliTect-Samples/EWU-CSCD371-2021-Winter\**](https://github.com/IntelliTect-Samples/EWU-CSCD371-2021-Winter%5C)

**Reason for 1-6 chapter upfront reading:** to cover more material in class for our benefit. We should be relatively familiar with the basics at this point, should be more skimming the first 6 chapters since much of it we should be familiar withbased on the fundamentals of other languages like Java.

**Testing:** Advantage of automated testing/unit testing is that it saves us from manually verifying every feature by giving the computer an expectation of the results that your code should have.

**\*Regression testing/retesting:** even after modifying code later, if your original unit test is correct, you will still get verification that your results haven’t changed despite alteration of later code.   
 **\*\*\*\*that makes knowing how to write and utilize these tests very useful and time saving. We will set up our system to keep our quality high.**

Often it is the subtle mistakes that trip us up. These little subtleties may escape us between the expectation and result but will likely be caught by a properly written test.

**HOW MUCH UNIT TESTING?:** 100% code coverage is difficult, expensive, and unlikely.

0% code coverage is very easy, but potentially very expensive

**Correct:** between 0-100% (lol). Ideally 75%-80% BUT it does depend on the project you are working on, the languages, how intent revealing your code is… **There is no magic answer**

**(HOWEVER) we should be at about 80-90%.**

**TEST DRIVEN DEVELOPMENT Design:**

1. *Write a test to show that you need to write production code.* 
   1. *Next write the simplest implementation that lets the test pass*
      1. *Refactor*
         1. *Repeat*

*End Result: all qualities are fulfilled, code coverage is high, minimizes duplication of code*

***Writing with unit tests is like driving with a seat belt,*** *there is an absolute correct way to do it (wearing your seatbelt/writing unit tests) and* ***once you become familiar with the correct way of doing it and become practiced in doing so, it becomes habit and will feel weird to go without*** *code without tests, just as it feels weird to drive without a seatbelt after a while.*

**Risk is increased drastically as project size increases without unit testing.** This is reflected in industry with additional costs as well to cover for lacks of unit testing, whether testing manually.

**Can you have 100% code coverage and still have bugs?? YES - we can have bad tests or incomplete code.**

Just because a test is written, doesn’t mean that you are writing tests that achieve what you desire, or that fully cover all of the potential bugs.

*I.e*

*[2:26 PM] Kevin Bost (Guest)*

*public void Divide(int a, int b) => a / b;*

*[TestMethod]*

*public void RunTest() => Assert.AreEqual(2, Divide(4, 2));*

**Exercise following Mark:**

Create Class library project in visual studio:

Project name: LoginStuff

Check box for place solution and project in the same directory.(AT END OF CLASS WE WILL MOVE SOLUTION)

*\*inside the lectures, there will be a folder for the class which will hold the code from the class, video, etc.*

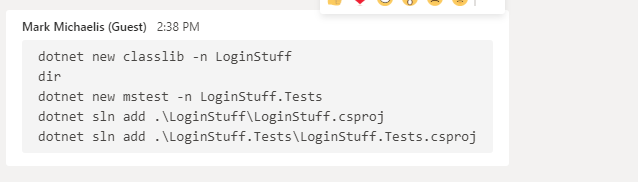
*Then right click on “LoginStuff” in solution Explorer and create new Project to house the tests and create Project: LoginStuff.tests*

*[2:34 PM] Kevin Bost (Guest)*

*Naming convention: <Name Of Project being Tested>.Tests*

*​[2:36 PM] Kevin Bost (Guest)*

*Solution files just provide organization of projects and files within Visual Studio. VSCode just uses your folder structure.*

***CommandLine:*** *find directory you would like the project created in: *

*dotnet new classlib -n LoginStuff*

*dir*

*dotnet new mstest -n LoginStuff.Tests*

*dotnet sln add .\LoginStuff\LoginStuff.csproj*

*dotnet sln add .\loginStuff.Tests\LoginStuff.Tests.csproj*

**VIEW:** Test Explorer to run tests (similar to eclipse and other IDE)

**GenerateType:** put in projectLoginStuff: DO THROUGH CODE GEN -follow up on this later for clarity in notes Can use autocode to automatically add dependency

**CommandLine:** go into test project directory:

Dotnet add reference ..\LoginStuff.csproj

h | select -last 6

Code . (will open up VS or your default IDE)

**Powershell is dotnet in command line.**

This means that you can hit . and it will pull up a bunch of options of commands you can do.

I.e. if you type; “this is a test”. Which will give all methods that are available after the dot. Allows us to easily work with a CLI as well while we learn C

**Right click on LoginStuff, add, new item, class, svm /\*for static void main\*/**

Marks’ preference is to do the solution directory in the **parent** of directory.

\*create the solution, then add it

\*If we don’t do this, then we may not end up with our solution linked to our other files.

\*solution files are ugly. They give the path and the solution project you are pointing at. If you don’t have it at the parent level, when you move the solution you also have to move the project.

Generally don’t use an abbreviation in C# unless it is an agreed upon ALWAYS used abbreviation. For example: HTML is an accepted abbreviation

\*Because of intellisense, we should just have to declare the name once and intellisense should help suggest the longer names anyway, shouldn’t really increase your typing much.

CONVENTIONS ARE EXTREMELY IMPORTANT \*\*\*\* THEY WILL BE STICKLERS

Classes and Methods: PascalCase:

Local Variables: camelCase

Fields: \_PascalCase (will talk more about later, called "unsderscore pascal case". )

\*\*\*\*\*\*\*READ AND MEMORIZE THE CODING GUIDELINES\*\*\*\*\*

BEST WAY TO UNDERMINE YOUR EXPERTISE IS TO NOT FOLLOW THE CONVENTIONS

They will tell us how to turn analyzers on so the compiler will tell us if we are violating any of the conventions.

\*\*\*\*PAY ATTENTION TO NAMING AND STYLE

SHORTCUTS: \* MUST MAKE SURE C# shortcuts are installed or selected)

Ctrl + r, t Only works in the context of tests

Ctrl + shift + B Build

Ctrl + f5 runs the program if there is one selected

**code coverage:** can tell us how any of our tests are passing and in which methods they are testing. *To find code coverage -> go to test view, then drop down and bottom option*

*CODE WITH COVERAGE: might be tested*

*CODE WITHOUT COVERAGE: is NOT tested*

***Visual studio enterprise:*** *can have* ***live unit testing*** *as an option. Likely EWU can probably help you get a login (see 1st day notes on enterprise edition info)*

*ApplicationTests.cs FOR FULL CODE: see the github for todays lecture for the in class code*

*Can be created very quickly thanks to intellisense suggestions.*

***F12 is your friend: ctrl+.*** *Will generate the method, hit enter, should now compile.*

*Assert.IsTrue(success)*

***\*\*\*\*This generated method does NOT transfer the name of the variables.***

***//incomplete code - will return and try to clarify***

*\*\*SO another option is to the generation using defined local variables instead.*

*I.e. string userName = “Inigo.Montoya”;*

*String password = “OpenSaysMe”*

*Bool success = application.Login(userName, password)*

*Assert.IsTrue(success);*

*Doesn’t matter which type you use as long as you remember to change the name of the parameters. If it isn’t done for you.*

***Generally -*** *don’t write code unless it helps make things more clear.*

*I.e. the “this” keyword isn’t really needed in C#*

***Alt + shift + enter:*** *maximizes source code real estate on screen*

***Alt + arrowKey :*** *moves your line of code*

*public bool Login(string username, string password)*

*{*

*return password == “OpenSaysMe”;*

*} //this should pass*

*Test fails*

*Write test that passes*

*Refactor*

*[TestMethod]*

*Public void ValidLogin()*

*{*

***//Arrange***

*Application application = new Application();*

***//Act***

*Bool result = application.Login(*

*Assert.IsFalse(application.Login(userName: “Inigo.Montoya”, password: “Bad Password”)))*

***//Assert ONLY HAVE ONE ASSERT OTHERWISE IF ONE TEST FAILS THE REST WONT RUN***

*Assert.IsFalse(result)*

*}*

***Then refactor: Since all of our previous tests all create an application it is doing alot of duplicate work so we can refactor to reuse.***

***OK TO NAME THINGS THE SAME AS THE DATA TYPE***

*I.e. Application Application; as a variable*

*Generally don’t use fields, just use properties.*

***In C# there are properties:***

***Property:*** *gives the ability to intercept to validate what you have is going to be set correctly.*

*Application Application; //field of data type application*

***Ctrl + . :*** *Enter can be used to rename all of our applications within scope. Really helps with refactoring*

*Public Application Application {*

*Get => Application;*

*Set => Application = value;*

*}*

***Ctrl + alt + click*** *on the author links that you would like to edit at the same time. There will belinks for places that are suggested to make the changes. This will place the 3 cursors and we can edit those all 3 at once since we are editing the same word, this will save us lots of time and energy.*

***Properties:*** *C# supports the ability to have validation on your getters and setters..*

***NOTE:*** *Assert methods are just fancy ways of throwing exceptions with useful error message. Any unit test that does not throw an exception is a success, and any unit test that throws an exception is a failure.*

***DEBUGGING HOTKEY: f9 for breakpoint***

*[TestMethod]*

*Public void CheckApplicationIsNull()*

*{*

*Application = null;*

*Assert.IsNull(Application);*

*} // ctrl + r, t*

*If we change the above test method to a property: we can right click and select “encapsulate as property” which will establish Application as:   
Property:*  ***Application Application {get; set;}***

**ALWAYS USE PROPERTIES INSTEAD OF FIELDS UNLESS YOU HAVE TO**

Access from property instead of field

**Properties:** Underscore\_Pascal\_Case

\*allows us to do interception

**\*\*properties start with underscore**

*So we will change our previously listed getters and setterS:*

*Public Application Application:*

*{*

*get => \_Application;*

*set => \_Application = value ??*

*Throw new ArgumentNullException(nameof(value));*

*}*

*Ultimately, getters and setters are first class properties that allow us to verify the getters and setters in C#, unlike Java where we had to have fields, then getters and setters to fetch them.*

*In C# we can check our getters to determine if they are retrieving something valid.*

*So for example, we could set it so that \_Application cannot be null.*

***F11: Debugging: Step In***

*IF WE RUN OUR TESTS IN A DIFFERENT ORDER IT MAY AFFECT THE RESULTS.*

*UNIT TESTS SHOULDN’T AFFECT EACH OTHER. IF A TEST FAILS OR SUCCEEDS ANY PREVIOUS OR FUTURE TESTS SHOULDN’T BE IMPACTED.*

*[TestInitialize]*

*Public void Setup()*

*{*

*Console.WriteLine(“Inside Setup”);*

*Application = new Application();*

*}*

*//gives every single test a new instance of the application*