***5/21/2018 Notes:***

*For jobs: I was a proofreader for 2 years and one of the best they had…I will be able to read through code and find syntax and logic errors better than most people.*

**GIT & GITHUB**

Git and github are 2 technologies that perform one function

***GIT*** = source control management(SCM) / backs up to cloud

Push code to repository (cloud) at least 1x per day

***REPO*** = repository

Keep repositories organized

Store all REPOs in c:\repos

DO NOT DO A GIT INIT IN REPOS!!!

~ = sitting in your home directory

Create all of our code inside c:\repos with subfolders



!!!NOTE that /c/repos is not a repository!!! --- The real repositories will be in subfolders

***SOURCE CONTROL*** = manages multiple versions of documents and keeps it in order

***GITHUB*** = same thing as GIT but in the cloud

Topics for GIT AND GITHUB today include: (1) Bash shell (see image above); (2) Command line interface (CLI); (3) File structure; and (4) Commands

***Bash shell:***

Similar to Linux command line

Commands are in book (pg.14 or so in book)

Also Greg will be getting us the slides for this presentation

***Command Line Interface (CLI):***

Git commands entered at command line (git cmd [opts}

In book on page 14

git: program name

cmd: what to do

[opts]: options based on command

All commands are lower case

>Git status –s

***File structure:***

How to tell if it is a file repository

Add –a to see hidden files

If it has a .git folder inside then it is a repository

You don’t want repositories inside of other repositories

Removing the .git folder removes git control

***Commands:***

All commands begin with git (lower case)

See GIT/GITHUB slide printouts for other commands such as init, remote, etc.

If you forget –m on message commits then add comments to top and “:X” in place of insert on text editor that pops up

Steps to add/commit things on github:(slide 17)

1. git init adds folder to start with git
2. git add to start tracking
3. git commit to commit (don’t forget to use -m for message)

***5/22/2018 Notes:***

***4 Announcements:***

1. Don’t leave electronics behind or other things behind each night
2. .NET meetup tonight @ 6PM
3. Party tomorrow @ 4:30PM
4. Fill out Day 1 feedback form

***SQL Servers:***

* ***SSMS*** = SQL Server Management Studio (interacts with the SQL table to manipulate it)
* Primary key is a unique identifier (at least one column but can be more than one) when taken together makes the value of that row unique – can use integers and for the bootcamp we will always use “Id” incremented 1 integer at a time (i.e., 1,2,3,4…)
* In SQL strings are bound by single quotes
* To insert a single quote (‘) in a string you must insert two single quotes next to each other (‘’); note that this is not a quotation mark but two single quotes
* Ordering can be different in SQL if your table is in different languages
* can't have any columns in the list that do not have a correspoding "group by" when using a sum
* when grouping data you must either group by or be aggregated or errors will be returned
* "having" works just like where but is used on aggregated data

***GitHub Upload of SQL Lesson:***

|  |
| --- |
| Max-Student@Student03 MINGW64 /c/repos  $ cd sqllesson  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git status  On branch master  Your branch is up to date with 'origin/master'.  Untracked files:  (use "git add <file>..." to include in what will be committed)  SqlLesson Queries/  nothing added to commit but untracked files present (use "git add" to track)  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git remote -v  origin https://github.com/gpdoud/SqlLesson.git (fetch)  origin https://github.com/gpdoud/SqlLesson.git (push)  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git remote remove origin  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git remote -v  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git remote add origin https://github.com/michaelbuchanan23/sqllesson.git  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git remote -v  origin https://github.com/michaelbuchanan23/sqllesson.git (fetch)  origin https://github.com/michaelbuchanan23/sqllesson.git (push)  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git status  On branch master  Untracked files:  (use "git add <file>..." to include in what will be committed)  SqlLesson Queries/  nothing added to commit but untracked files present (use "git add" to track)  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git add .  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git status  On branch master  Changes to be committed:  (use "git reset HEAD <file>..." to unstage)  new file: SqlLesson Queries/Assignment(TotalByState).sql  new file: SqlLesson Queries/Average.sql  new file: SqlLesson Queries/FromJupiterToSun.sql  new file: SqlLesson Queries/JoinSelectingOnlySpecificColumns.sql  new file: SqlLesson Queries/JoinViews.sql  new file: SqlLesson Queries/Name-City-State.sql  new file: SqlLesson Queries/OuterJoinWithNullValue.sql  new file: SqlLesson Queries/SelectAll.sql  new file: SqlLesson Queries/TotalSalesByCustomer.sql  new file: SqlLesson Queries/Union.sql  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git commit -m "Updated Sql Lesson with Queries from class on 5/22/2018"  [master 41f7d08] Updated Sql Lesson with Queries from class on 5/22/2018  10 files changed, 73 insertions(+)  create mode 100644 SqlLesson Queries/Assignment(TotalByState).sql  create mode 100644 SqlLesson Queries/Average.sql  create mode 100644 SqlLesson Queries/FromJupiterToSun.sql  create mode 100644 SqlLesson Queries/JoinSelectingOnlySpecificColumns.sql  create mode 100644 SqlLesson Queries/JoinViews.sql  create mode 100644 SqlLesson Queries/Name-City-State.sql  create mode 100644 SqlLesson Queries/OuterJoinWithNullValue.sql  create mode 100644 SqlLesson Queries/SelectAll.sql  create mode 100644 SqlLesson Queries/TotalSalesByCustomer.sql  create mode 100644 SqlLesson Queries/Union.sql  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git push origin master  Counting objects: 43, done.  Delta compression using up to 4 threads.  Compressing objects: 100% (39/39), done.  Writing objects: 100% (43/43), 9.06 KiB | 927.00 KiB/s, done.  Total 43 (delta 6), reused 0 (delta 0)  remote: Resolving deltas: 100% (6/6), done.  To https://github.com/michaelbuchanan23/sqllesson.git  \* [new branch] master -> master |

***5/23/2018:***

* SQL server 2017 google search will yield lots of results on the topic as will searches for C#
* Coding from scratch helps ingrain things better than copy and paste so code from scratch where possible
* Create apps in 2 steps:

1. Make it work/***FUNCTIONAL*** no matter how ugly it looks; and
2. Make it ***ELEGANT*** meaning:
   1. Robust meaning it doesn’t break;
   2. It is easily enhanced by someone else; and
   3. It’s efficient

* How do we keep learning:
  + Meetups
  + CLEs
  + Pro-bono work
* Make sure to verify your understanding of what an individual wants in their program
  + Verify by reflection – repeat back the request in your own words
* Join customer and order, sort by order amount biggest first so desc, no id fields but all other columns in
* Functions page for strings on Microsoft website:
* 
* When doing a delete make sure there is a where clause or it will delete everything…use where clause even if you intend to delete everything
* You can highlighted certain portions of code in Sql Server if you only want to run that particular portion of the code
* Comment group of lines shortcut key is ctrl+k+c
* Uncomment group of lines shortcut key is ctrl+k+u
* Read the data you want to change first via UPDATE with a where clause then copy that WHERE clause to your update in order to accurately update the data
* Don’t ever update your primary key
* The only time (usually) you should use null with numbers is when using foreign keys
* nvarchar is now the new way to define a string because of international languages where you need a double byte
* zip codes should always be strings because it doesn’t represent a mathematical number

***Notes 5/24/2018:***

* *Interviews*: Be succinct during interviews in your answer unless you are asked to elaborate

***SQL Server:***

* When changing the table definition the syntax is ALTER TABLE [name of table]
  + Usually used to add/drop column or change the type (e.g., change zip code from int to string)
* Purchase Request System (PRS) Tables:
  + User: list of users authorized to log in. A user is the owner of a purchase request.
  + Vendor: company that sells products
  + Product: an item that can be requested by a user on a purchase request
  + Purchase Request: grouping of all items a user requests (i.e., instance of a request containing user and total)
  + Purchase Request Line Item: a single product and quantity on a purchase request
* To create a unique inex on a table:
  + Create (unique) index [name] on [tablename](col1,col2,…)
* Stored Procedures:
  + Basically creating functions for sql
  + You may also just do queries in stored procedures (in cases like where variables are needed like different users production per week which will change with each query meaning users and the week will change with each query)
  + Stored procedures vary wildly across different database systems (i.e., MySQL vs. SQL Server)
  + Code block is the same syntactically as one line of code – between begin and end of stored procedure is a code block
* Create stored procedure to load data into tables for each of our new tables called “Add” and “Table Name”

***Notes 5-25-2018:***

***TO DO: COMBINE NOTES INTO ONE DOCUMENT***

***GIT/GITHUB Assignment Information:***

* We will push our assignments to greg via <https://github.com/gpdoud/dotnet4-assignments.git>
* “git pull origin master” is how we pull
* If we are having issues with pushing then we should pull down the most recent version of the folder via “git pull origin master” then attempt to “git push origin master” again

***Code blocks:***

* Is either a single statement or
* Begin…End

***T-SQL:***

* Allows one to code procedures in sql server

***General Coding Note:***

* If you are nesting your if statements more than 3 levels deep then you need to reevaluate how you’ve done it and do it more efficiently and easier to read
* When you need to know whether a certain kind of data exists then you use an Exist – 1 statement such as in “Select1-LookFor1InstanceAndReturnTwoDifferingSentences(UseForTransactions).sql”
* Looping does a group of statements potentially more than once until some Boolean condition is true or false
* Default values in SQL are all null

***Notes 5/29/2018:***

* Ask Greg how to go about solving a problem like Practice>SQLBasic Join>Challenges on [HackerRank](http://www.hackerrank.com/) – basically, how to solve difficult problems on hacker rank without looking up an answer and attempting to reverse engineer it?
  + Reverse engineer it then attempt to solve it again a few days later
* How to implement row\_number() in sql?
* How to implement pivot in sql
* Look into SQL diagramming techniques which will help us understand how to put things together

***C#:***

* C# is the way to program for windows
* We need to be as good in the IDE(i.e., visual studio) as we are with C# the language
* To familiarize ourselves with Visual studio make sure to right click on everything in visual studio
* SQL is declarative meaning you tell it what you want and it does it for you
  + In C#, it is a compiled language meaning you have to tell it how to do it unlike in SQL
* It is IMPORTANT that we focus on the concept of how to program and don’t focus on the syntax type of things
* Compiled languages
  + *Source:* Start off with your source (i.e., lines of text that you enter in your program)
  + *Compiler:*
    - The source goes into a compiler to compile the code into intermediate language (aka byte code—it is also close to assembly language)
    - The compiler shows syntax errors in the source code
    - Note that the intermediate language is not output if there are errors
  + *Runtime Environment:*
    - Takes the byte code/intermediary language and interprets it to run
    - Semantic errors/logic errors are errors between intermediary language and runtime environment
      * There errors are like bugs in that they may be sporadic – need to recreate the error to know what is wrong and how to fix it
* *Classes:* classes are allowed to have 2 things in them: (1) properties; and (2) methods
  + *Properties:*
    - Properties are things like fields, variables, etc. (thing of them like nouns—person,places,things)
    - These properties holds data like strings, integers, etc.
  + *Methods:*
    - Methods are functions, procedures, etc.
    - Code that does something…actions
    - Typically called the actions on the properties
* Put everything in main before it closes because after main runs it will shut down the program
* Capital letters are typically used for programs, methods, etc. in C#
* When creating classes think about the thing that you are modeling…what do I need to know about it in order to do this modeling…so step 1 = add properties to the class
* “void” means the method does not return a value
* Creating classes:
  + Properties remain private
  + Methods are public
* Data types:
  + Scalar (int, bool, DateTime,Decimal…)
    - Cannot be null by default – recently they have allowed null to help with sql issues where nulls are allowed in these data types
  + Reference (everything else) includes strings
    - Can be null
* “int[]” is an array of integers whereas “int” is just an integer
* “struct” creates a new type that is a collection of other types
  + Note that struct does not have to contain the same type while arrays must all be the same type

***5/30/2018 Notes (DAY 7):***

***Git -> Github in Visual Studio:***

* In visual studio to add a repository go to Team Explorer -> Settings -> Repository Settings -> Remotes -> Add and add as the window below shows



* Then to commit go to Changes and enter a message for your commit, do a commit all then sync then push it remotely
* Sourcetree = gitbash but with a GUI
  + Greg stated some people he knows use this instead of gitbash

***Commenting code:***

* Comment your code so that someone else can pick up and figure out what you are doing
  + Can do so by “//”, “/\*…\*/”, and ///
  + The three slashes can turn your documentation into a webpage and according to Greg is “the way to go” use <summary> …. </summary> for purposes of tagging the summary of your page for a website
  + Also use <param name = “args”>Command line parameters as strings</param> -- didn’t catch what this was for
* To comment out lines you can highlight them then click then “ctrl+e,c”
* To uncomment lines you can highlight the block of code then “ctrl+e,u”

***C#:***

* Console.ReadLine() returns a string
* If statements:

If (Boolean expression)

Statement | statement block

Else

Statement | Statement Block

* Always use curly braces for if statements/statement block even if it is only one line of statement
* Ctrl+shift+b in visual studio does a build
* Don’t nest more than 3 if statements --- if you need more than that then use
* While statements:

While (Boolean expression)

Statement | statement block

* While statements keep looping through until the Boolean expression returns false
* Using “break;” in a while statement will break the while statement even if its Boolean expression is still true

***C# Methods:***

* Blocks of code that execute anywhere in your program to perform a function such as changing string to integer
* “Main” is a method in “static void Main(string[] args)”
  + The method signature includes the name of the method, whether it returns anything, and the parameters
* It is generally bad practice to put one method inside of another method
* Put created methods above or below the “Main” method
* Methods may share the same name (called overloading) so you can pass differing parameter types to the method and get the result you want (e.g., a method called “Area” that can calculate areas of squares, rectangles, and circles depending on the type of input it receives)
* ***Scope:***
  + When you define something it is only available in that scope (e.g., variables declared in the main method are not available in other methods)
  + If we create variables in things like nested if statements then they will not exist outside of the statements they are in so it is better to create those variables before running the if statements
  + Wherever your variables are declared that is the scope so whatever that code block is
* Random #’s:
  + Random rnd = new Random();
  + > int nbr = rnd.Next(1,7) – give me a random number between 1 and 6
    - [Random.Next Method (Int32, Int32)](https://msdn.microsoft.com/en-us/library/2dx6wyd4(v=vs.110).aspx)
* Method – breaking up
  + You should break up methods
  + A method should only do one thing
  + You should be able to see every line of code of method on your screen (roughly 20 lines)…break up into smaller pieces if they get any bigger than this
  + The best programming right now is being “loosely coupled” (kind of like modular)
* To find bugs in code you can “step through” the code using visual studio:
  + Debug -> Start debugging (F5)
  + Debug -> Step Into (F11)
    - steps into a method call
  + Debug -> Step Over (F10)
    - steps over the method (i.e., execute the method and go to the next statement)
  + highlight a line of code and right click and run to cursor will run the program to that line and then stop

***5/31/2018 (Day 8):***

***Class:***

* A ***CLASS*** is a user defined type such as int, string, bool, etc.
  + So if we create a class named “Student” then “Student s” would create a variable “s” of class type “Student”
* The class name has to be unique in the namespace
* There are only two types of things that can be in a class:
  + Properties (data/fields/variables); and
  + Methods
* The Class functionality was created because:
  + When something breaks you only have to go one place to fix it versus going everywhere that the code has been copy-pasted --- in other words it isolated pieces of code
  + it also helps simplify complex systems which would then help with debugging
* Put all your class files in its own source file…the name of the source file should match the class
  + To create a new source file:
    - Go to solutions explorer, right click it, go to add, and then class
    - Shift+Alt+C also works
    - Either way you do it above, after the window opens to add a class for this class we want “Class --- Visual C# Items”
* Process to create classes:
  + What data do we need?
    - For example, if a square then:
      * We need the area (length of side2) or perimeter (4x length of side) which can be found by getting the length of one side
* [Classname] [variable] = new [classname] ();
* Interpolation ($”{a}”) – allows you to use variables in the quotes by putting dollar sign at beginning of Console.WriteLine
* ***ENCAPSULATION***– bringing together the data and the methods that operate on that data into an object called a class --- Greg said to know this for interviews
* Use verbs when creating methods telling others what they do
* Coding challenge:
  + Let’s Make A Deal 1,2,3 --- behind either curtain 1, 2, or 3 is a grand prize
  + You pick a number 1, 2, or 3
  + A number that you didn’t pick is revealed as not being the grand prize
  + You are now given the option to change your guess…should you change it?
  + Now write a program to simulate it
* For debugging Greg uses “quick watch” in visual studio a lot to check the state of variables – you can get there by putting your cursor after ending “}” for “static void Main(string[] args)” then doing “Run to Cursor” then going back and right clicking on a given variable and going to “quick watch” via right click
  + Great for checking error conditions to make sure it works when it’s supposed to
  + Get familiar with the debugging tools in visual studio so we can quickly fix problems with our code
* ***Arrays:***
  + Use List and Dictionary for arrays – usually list – list is the same as a fixed array except it is dynamic
  + In a dictionary you assign each item in the array a key and a key value (e.g., key = OH and value = Ohio)
  + List <Student> says we are going to have a dynamic array of student instances—is an index of data
    - public List<Student> Students = new List<Student>();
  + to add things to a list use:
    - dotnetbc4.Students.Add(foster);
  + to remove things from a list use:
    - dotnetbc4.Students.Remove(foster);
* ***Breakpoint:***
  + Click on the grey line at a line of code
  + When debugging it will stop the program running on that particular point

***Notes 6/1/2018:***

***Classes:***

* List<int> -- tell it what data type will be in the list, in this case it’s an int but can also be a user created class like Student
* Dictionary <string, string> --dictionary has two parameters
  + Can’t change it once you define it
* Note that there is a generic class for stack – he says he rarely uses this in his career
  + It is a last in, first out (LIFO)
* Scalar variables/types cannot be null in C# as they can be in SQL
  + However, recent changes have allowed us to make scalar variables nullable but you have to declare it as such:
    - “int? k = null;” //this allows the variable k to be null and thus the following is ok
* For lists:
  + List <T>
    - Add: Add(value)
    - Update: [0] = value
    - Delete: Remove(value)
* If you are unsure if a method will return something just make it “void”…you can always go change it later
* Key mapping (keymapping) in Visual Studio: Tools -> Options -> Environment -> Keyboard

***Exceptions:***

* In the exception class Greg said to pay attention to the “Message” because that will give you a hint as to what is wrong
* When the message doesn’t tell you what you need sometimes the “Inner Exception” in quick watch will tell you what you need to know
* You get there by clicking on “View Details” after doing our “Run to Cursor” at the last “}” in the program.cs
* Try {

[statement]

} catch (Exception ex) {

[statement]

}

* For example/throwing an exception:

try {

int x = 0;

int y = 1 / x;

} catch(Exception ex) { //can also use the type of DivideByZeroException

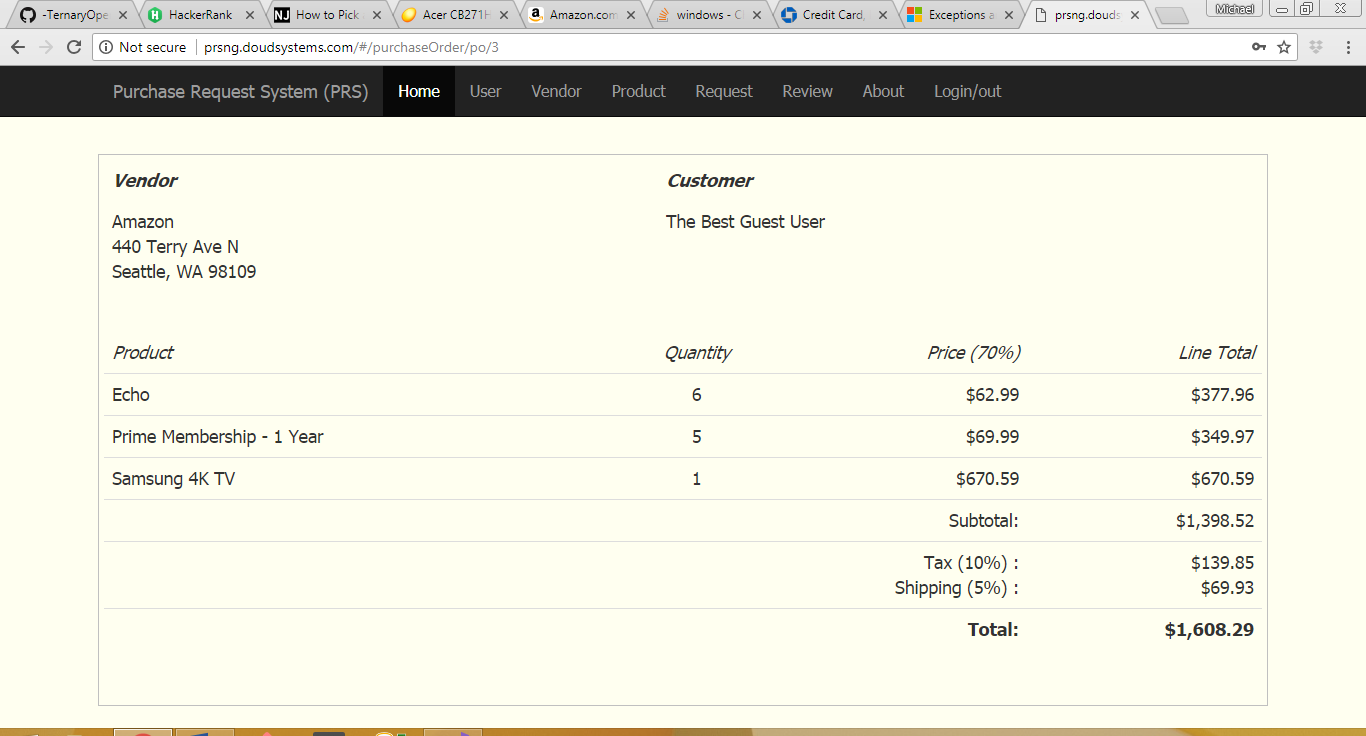
Console.WriteLine("Can't divide by zero bro");

}

//outputs “Can’t divide by zero bro”

* There is a generic application exception you can put in your own program called “Application Exception”
* When to use exceptions: often in else statements after everything else has been looked at…in other words, use exceptions for error checking in an “else” statement with an “if” statement where other error checking is happening – useful because we can’t always think of everything that could go wrong and good exception handling can help you track down where the errors are happening

OPTION PIECE OF WORK FOR CAPSTONE THAT NO ONE HAS EVER DONE: IN THE VENDOR LIST THERE IS A SHOPPING CART WHICH CREATES A PURCHASE ORDER THAT WOULD BE SENT TO AMAZON FOR ALL APPROVED PURCHASE REQUESTS – MOST OF THIS WORK HAS TO BE DONE IN C# -- IT IS LIKE THE PICTURE BELOW -- [LINK](http://prsng.doudsystems.com/#/purchaseOrder/po/3) – UN: user PW: user



***6/4/2018 (Day 10):***

* If a product has multiple vendors then you need a product table, a vendor table, and a relational/relationship table in order to house all the relationships for product 🡪 vendor and vendor 🡪 product – these relationship tables would contain a foreign key for product, foreign key for vendor, and possibly quantity and price
* Use a password manager for passwords to avoid getting hacked

***Switch statements:***

* Pg. 16 of *A Detailed Approach to Practical Coding* by Nathan Clark
* Cleaner way to write a load of if-else statements
* Switch(v){

Case “a”: //same as saying if variable(v) is equal to “a”

[statement]

break;

Case “b”:

[statement]

break;

default: //this is essentially the “else” clause of switch-case

[statement]

break;

}

* If you need to look for >, < then you will need to use if, else statements…the switch-cases are geared toward when you have a single piece of data and you want to compare to
* Switch is also fast and easier to comprehend/read then a large if-else

***Number Functions:***

* Pg.48 of *A Detailed Approach to Practical Coding* by Nathan Clark - math functions reference

***String functions:***

* Pg. 62 of *A Detailed Approach to Practical Coding* by Nathan Clark – good reference

***Enum:***

* We can create these data types ourselves
* There are not many circumstances where we use this anymore
* We will likely run into this in legacy software/code
* One example would be something like “enum TransType” where 1 = send and 2 = receive
* enum TransType {

Send, //system assigns a value of 1 to Send

Receive //system assigns a value of 2 to Receive

}; //this statement defines Send and Receive

* the difference it makes is that we can now use an if statement such as “if (TransId == TransType.Send)” in our if statement instead of “if (TransId ==1)” –this documents our code a heck of a lot better – also this assumes that we are receiving TransId with a value of 1 or 2 where 1==send or 2==receive

***Stoplight program:***

* Stoplight: we build a solution called “Stoplight” -- this used something called workflow which Greg said is used often in programming
* This used enum types and switch - case

***Static:***

* Static means relating to the type itself, rather than an instance of the type
* You access a static member using the type name instead of a reference or a value, e.g., Guid.NewGuid().
* In addition to methods and variables, you can also declare a class to be static (since C# 2.0).
* A static class cannot be instantiated and can only contain static members
* Constants are a good use of static data (e.g., Pi)
* Static variables are like global data (meaning there is only one of them
* Use a static variable to generate an Id

***Constructors:***

* Constructors are methods that are defined in a class and get called automatically when you create a new instance --- they have a special syntax
* In our case we used them to implement a counter and set an Id for each instance
  + See StaticExampleSolution for example that we did in class
* Can make more than one constructor that take different parameters as long as the parameters that they take are unique

***DECIMAL*** is a fixed type, precise data point (ideal for currency)

Vs

***DOUBLE*** is not ideal for currency because you have to round to the decimal place you want whereas decimal will do that for you

***Inheritance (Derived Class: Base Class):***

* The example Greg gave is how a German Shepard different from a dog – he stated a German Shepard has a long snout
  + Class GermanShepard : Dog {

[statement block]

}

* “Class Square : Rectangle, Quadrilateral”
  + This is simply saying that Square inherits the properties of both Rectangle and Quadrilateral
* When one class inherits from another the inheriting class receives virtually anything that is not private from the inherited class – if it’s private then the inheriting class doesn’t get it

***If you click on something like “integer” and hit F1 then it will take you right to the Microsoft page for that data type***

***Types of Classes:***

* ***PUBLIC*** - it is accessible outside of the class that it is defined in
* ***PRIVATE*** – accessible inside the class only
* ***PROTECTED*** – accessible inside the class and all inherited classes (i.e., same as private but public to inherited classes

***Polymorphism:***

* This is where the virtual—override stuff comes in to choose the right print method, for instance, based on what the type really is vs. what variable we place it into – see the “InheritanceSolution” for anexample of this
* Calls different methods based on what it is at runtime
* The 3 tenets of object oriented programming (OOP):
  + ***ENCAPSULATION*** - bringing together the data and the methods that operate on that data into an object called a class --- Greg said to know this for interviews
  + ***INHERITANCE*** - When one class inherits from another the inheriting class receives virtually anything that is not private from the inherited class – if it’s private then the inheriting class doesn’t get it
  + ***POLYMORPHISM*** - Calls different methods based on what it is at runtime

***Banking:***

* Account
  + Savings : Account
  + Checking : Account
* Account has:
  + nbr (number) //string –account# won’t change so static?
  + owner //don’t worry about adding the owner at this time
  + balance //decimal data type -- private
  + description //string
* Account needs methods for:
  + Deposit //return decimal of updated balance
  + Withdrawal //return decimal of updated balance
  + GetBalance //return balance so decimal return type
* All Savings: Account have:
  + IntRate (double type since more than 2 decimals may be needed)j
  + Method for CalcInterest() //do simple interest
* All Checking : Account have:

***6/5/2018 (Day 11):***

* When you create a set of classes for a customer to use don’t use Console.WriteLine, rather use exceptions because they may be running on different platforms
* Always create a default constructor first when creating a new class – default meaning a constructor with no parameters