Contents

(Introduction / Welcome)

Hello and welcome to the Virtual Training Company presentation of Microsoft C# 2010. My name is Mark Long and I'll be your instructor for this course and we're going to dive into Microsoft's flagship development language and we're going to look at C# 2010 top and bottom. And pay some attention to the new things that are taking place. Now while there's not a whole large long laundry list of things that are new to C# 2010, they've certainly improved a lot of things. They've worked on the Compiler quite a bit and they've added a new major feature called Dynamic Types, which is something that we've heard for years would never happen to C# 2010. So anyway, we're going to cover a lot of information here. I'm glad you're here. We're going to have some fun, we're going to learn a lot. Let's get started.

(Introduction / Course Overview)

Now before we jump into the course, let's talk about the Course Overview. I want you to have an idea of where you're headed and what we're going to talk about here. First of all, I'm going to start off and do just a little bit of C# history, so you understand where this came from and quite frankly it makes you look a little smarter around the water cooler. Then we're going to talk a little bit about what's new in 2010. There's a not a whole lot of things new but the things are new, are pretty spectacular and so we'll talk about those. Next we're going to talk about how to get the tools and by that, I mean the Programming Tools so that you can follow along with me here and do some hands on as I work. Then I'll talk about Visual Studio. We'll take a little tour of Visual C# 2010 Express tour. If you're not familiar with the Visual Studio products, I think this will help you feel a little more comfortable in there. Then I'm going to take a real short, kind of, swerve out of C# per se and talk about Object Orientated Programming. Because if you don't really understand this and some of the basics of it, it's going to be kind of hard to understand what we can do with C#. Then I'm going to dive into some C Sharp basics, there'll be quite a few videos in this section. We'll talk about variables, Data Types, the new Dynamic Type in, uh 2010. Do an example of the new Dynamic Type. We'll talk about Casting Strings, the String Builder, Expressions, Statements, Enumerations, If Statements, Conditional Operators, those sorts of things. From there we'll dive into classes and really start to talking about some of the Inheritance Functionalities. We'll talk about creating a class, Instantiating Objects, Using Constructors, Overloading Constructors. Partial Classes, Static Classes, all kinds of things out there. And then we will jump into Constructors, Overloading Constructors, just all sorts of things about the classes themselves. And then from there, we're going to jump into and talk about Inheritance. Now this is something that becomes the basis for how to design and implement applications in an Object Orientated environment and so we'll cover that. Then we'll go from there, Static Classes. Then we'll jump on Fields and Properties, these tend to confuse people. They can be confusing. so we'll go through those. We'll talk about Read or Write Properties, Property Access, Modifiers. We'll talk about Access Modifiers on our fields or properties or methods or classes. So we'll cover those quite a bit. Then we'll concentrate on methods. We'll talk about Creating Methods, using Out Parameters and Optionals and the Params and Method Modifiers. Overloading our methods, hiding our methods and then Extension Methods, kind of a funky piece of functionality if you haven't seen, you need to see this one. Then we'll go from there into Delegates and Events, this is starting to get a little more advanced. If you don't understand Delegates, I'm going to put some examples in here, very, very simple. You will be able to get your head around these Delegates and understand what's going on. Then we'll use that as a foundation to talk about Events and you'll learn about how to set up an Event. How to use an Event Publisher, an Event Subscriber. Then raise and handle that Event and I'll just kind of point you to some of the things. Then we'll talk about Generics a little bit. Now these things are really cool and if you've never used Generics, you've got to see this. You've got to get your head around Generics. It doesn't take much. Once you see one happen, then you've pretty much got it. And then from there, we'll kind of close the course out. We're going to talk just a little bit about Interfaces. Because I mention Abstract Classes in the course and there's always a big question in live classes, what's the big difference between an Abstract Class and an Interface? And when should I use one or the other, so we'll kind of discuss that. Take a look and we're going to do quite a few examples. There's probably going to be 20 to 30 or more example programs that you're going to have in your Work Files folder here. So having said all that, let's just jump right in and get started, taking a look at C# 2010.

(Introduction / Understanding the .NET Framework)

Now let's turn our attention toward understanding the .NET Framework in just a minute here. To understand the .NET Framework means to understand the underlying functionality, the platform that C# is running on and it will in turn make you a better C# Programmer. Now let's go back and do just a little history here. When the .NET Framework first appeared, it marked a fundamental changing or adjustment in how programming was carried out on the Microsoft platform. Now this created cat calls and howls from the developers at the time who had invested a lot of time and money in the old COM environment. And so this major departure from that COM environment, really scared a bunch of folks, made them mad. All kind of things were being said but the original concept, keep in mind, was really aimed at web development, interestingly enough. Because at the time, web development was just really taking off, the web servers had some irritating aspects about how they managed memory. On the Client Side we were writing CGI Script, everything was running as script, an interpreted language on the web server and it just wasn't very efficient. But the .NET Framework idea kind of grew and developed and it ended up being this massive collection of classes and that fundamentally is what the .NET Framework is. It is this massive collection of classes that exposes functionality for programmers. Good example, I want to add some functionality to an application that I'm writing in C# and I want that application to reach across a network and pass some data via TCP/IP to another computer. Well there's a class for that. I can go out into System.Network and I don't know the rest of it of the top of my head. But I can find a class that will allow me to, to basically package data into TCP/IP Packets and send it across a network. I don't have to know how to do that. I just have to know what class to grab, instantiate an object off, of that class and pass the proper parameters to the proper methods, and boom the magic happens. Now another advantage of this is that, that functionality is now consistent. Because if you're writing an application in VB.NET and you're trying to carry out the same functionality, you're going to grab that same class, and you're going to pass on information in the same way. Make it very easy for us to troubleshoot each others programs, add functionalities later, extend it and that sort of thing. Now speaking of Extended, most of the classes in the .NET Framework can be extended or Subclassed if you know what that is, if you don't, not to worry, we're going to talk about it later. But the entire .NET Framework is organized into Namespaces. I've already mentioned System.Network but just keep in mind that if I want to do something with a database, I'm going to look for classes in the System.Data Namespaces. If I want to work with Input Output, I'm going to look in System.IO. So the Namespace is simply collects the classes into organized groups so that we know where to go and how to start looking for the functionality we need. Now the NET Framework actually began development in the late 1990's and it was at that time, under the codename or they even mention this as the marketing name, Next Generation Window Services or NGWS. Apparently that wasn't cool and sexy enough for Microsoft, because by the time this was actually introduced in the late 2000, it had been named the .NET Framework. Now .NET 1.0 was the first version that was released. We are currently at .NET 4.0. There's already rumblings out there about .NET 4.5, don't worry about that right now, it's a long way off. But I'll spare you all the history between 1.0 and 4.0 just suffice to say that 4.0 is really cool, it's picked up a lot of really neat functionalities and you can do an awful, awful lot with this platform. Now the advantages that you get, along with all those classes in the .NET Framework are first of all, something called the Common Language Runtime. Now this is not a .NET Framework Class so I'm not going to go too deep here. But this is kind of like the policeman who oversees everybody that's running any language that's .NET compliant runs under the control of the Common Language Runtime. Now it's also language independence built into the .NET Framework. You can code in any kind of language that is .NET compliant and there's over 40 of them and they all adhere to a common type system. Which means any type or object that I build in any .NET compliant language can be easily shared and utilized by any other .NET language. This is really cool. So you might want to go out and read up on that a little bit. And then of course, Security, you have to mention Security. It is a fundamental part of the design and implementation of the .NET Framework and again, won't go too deep here, you can go out and read about that. Portability, this is something you will read about in all the marketing material. You'll hear it but just understand, kind of the rest of the story here. The .NET Framework is engineered to be platform agnostic, meaning you can run on multiple types of processors, multiple environments. However, Microsoft has not chosen to actually implement that functionality. Right now we're running on the Windows Intel Platform and that's the only place it's really been used. So, but do keep in mind, this is in Microsoft's pocket they can pull this out at any time, the .NET Framework's ready to go for this. Now what you need to do, is to go out to Microsoft.com, just do a quick search for the .NET Framework. If you're not familiar with the .NET Framework, I'd strongly, strongly encourage you to get more familiar with the .NET Framework. It is the foundation that C Sharp is running on, it is what makes C# function. So the more you understand about the .NET Framework, the better you're going to be at designing and actually implementing and developing your apps in C#. So that's just a little quick overview of understanding the .NET Framework.

(Introduction / C# History)

And now let's talk a little C# history. This will help you understand where the language has come from, how you got it, where it's going and it'll help you with some water cooler chatter around the office there. If you're in that environment. Okay. Well first of all, understand C# came to us through the NET Framework. Now this was introduced early 2000's, marked a major, major change in how we do programming on the Microsoft platform. We first saw it early 2000's. It was a major departure from the prior environment, which was the COM, the Component Object Model environment in Microsoft's early days and the original concept was aimed at cleaning up web development only. It kind of grew, became popular and it, spawned and introduced to us, the C# language. So understand, C# came to us as part of the early Microsoft .NET initiative. Now Microsoft had some goals. The first one was to create a simplified version of the C Languages. Now this has to do with the history of Windows. Most early Windows was written in C and then in C ++, and as a result, you had to use a lot of C stuff to do a lot of things, very high end, in Microsoft programming. Now the problem with this was, C, C ++ were difficult languages, they required the programmer to really be on top of the game or you experienced memory leaks, all other kind of issues could come in. So it was difficult to program and program efficiently in C and C ++ unless you had a lot of experience and really knew what you were doing. Then of course the Visual Tools made programming even easier. So we got a simplified version of the C Languages, then we got that put in to an environment that made programming even easier. And here's the trivia for you. C# marked the complete absolute separation of Microsoft from Java. And yes you read that right. If you've heard this before, it's old news, but if you haven't, it might surprise you. Microsoft and Java were working together for awhile, this marked the end of that, go Google that, read that, it's pretty fascinating. Now C Sharp is the flagship language of Microsoft development. Now for you VB.NET ters out there who are already firing up the email program to send me some mail about how VB.NET, you can do anything with it that you can with C# and all that. I don't want to get in that debate. Okay. But here's the facts. C# has become the flagship language because it attracted or pulled into the fold, all those old C and C ++ developers who were doing the really, really mad scientist stuff on the Windows platform. Well when they came into the language, they went to the language. When they came into the .NET Framework, they naturally gravitated toward the language that they were most familiar with and the environment, and that was C. So as a result, you have an awful a lot of experience and knowledge out there and as a result, C# has become the flagship language. Microsoft themselves will tell you, that if you code something in C#, then you code the same functionality in VB.NET. The intermediate language that's created by the .NET Framework will be almost identical. Okay. So don't want to feed the debate here about which is better, C Sharp or VB.NET. But just understand C# has become the flagship language. What were the design goals Microsoft had in mind when they created C#? They wanted it to be simple. Now keep it mind, when you look at it, you're going to be thinking, maybe, wow, I thought he said simple. Well simple in comparison to the old C and C ++ language. Modern, meaning at the time that they created it, they wanted it to have functionalities that made things easier based on what we were doing with programming at the time. Now this was early 2000's. The prior 5 years had seen a massive change, and what we wanted our programming languages to do, because of the Internet. They wanted it to be a general purpose language, not to tightly niche toward one or two functionalities. They also wanted it to be Object Orientated and it is, we'll talk about that a lot more later. Now the current version and this will confuse you sometimes is C# 4.0. It's also known as C# 2010. Now if you really dig into that, it's called 4.0 because it's introduced as part of the Version 4.0 of the .NET Framework. It's called 2010 because it's also simultaneously released as part of the Visual Studio 2010 package. Okay. So anyway, that's just to confuse you a little bit and Microsoft's always ready to go there. So the C# coding skills though are in high demand. So has C Sharp been a hit? You bet it's been a hit. There's a lot of surveys out there that continually turn up, that C# coding skills are among the top three technical skill requirements for all advertised jobs out there. And especially when you combine with SQL programming, then you really start to rise to the top of job candidates. So it's well worth your time to master C#, it's really going to help your career development and it's really going to help in the employment area of your career. Okay. So anyway, that's just a little C# history to kind of let you know, where we're coming from and where we're going.

(Introduction / New in 2010)

Probably the first question that comes to most people's minds when they see a new version of a programming language come out, they, they first say, well wait a minute what's new? What's the compelling feature that should make me want to stick my head into this and try to figure this out? And so in C# 2010, there are a couple of things that are new that will really change the way you do some things and I just want to highlight those. Then there's a lot of other new things that we'll talk about as we move through the course here. The first one and by far the biggest is Dynamic Support. Now C# 2010 provides support for late binding to Dynamic Types. And I'll talk about Dynamic Types specifically and show you an example of that as we move through the course. But they're introducing a new Data Type here and it's called Dynamic. Now in other languages like Visual Basic or whatever, you would know this as a Variant Data Type. The Data Type is basically selected automatically based on the type data that the Compiler sees there. The reason this happened is because it provides simplified access to COM. Remember COM, the old Component Object Model? All those APIs think Office, think HTML Document Object Model, those kinds of things. And so Microsoft, after kind of looking down on this kind of deal, like a, a Variant Data Type or Dynamic Support in a language like C# for so long, has now decided to go that way again because of that COM programmability. Now Office programmability interestingly enough, is considered one of the great new features in 2010 and guess what? That comes about via again, that new Dynamic Data Type. And so again, we'll talk about Dynamic Data Type as we move through here. There's also quite a few improvements in Visual Studio and how the Compiler works, the things that it will flag for you. One of the coolest things, you're going to notice right away is, anytime I click on a symbol, like a Class Name, a Method Name, anything like that. In Visual Studio, in the Code Window, it will highlight all instances of that symbol all throughout the text. So if I'm trying to thing about refracturing something or locating a particular symbol somewhere else in the code, this make it much easier to do. So anyway, that's just some of the highlights of what's new in 2010 for C#. And I'll point out some more things as we move through the course and look at various aspects of C# 2010.

(Introduction / DreamSpark)

DreamSpark is a program that Microsoft has out there that you need to be aware of. It's amazing how many people don't know this is out there, but you need to know it as a C# developer. DreamSpark is designed for students. If you're a registered student and you can prove it to Microsoft and it's very easy, just go to DreamSpark.com and look at the details and see what it takes to qualify. If you qualify as a student, you can get a whole truck load of Microsoft technologies, development technologies, database technologies, absolutely free. Microsoft is desperately trying to encourage development on their platform and they're going after the student market. Now a close cousin of that also is WebSiteSpark. This is a program for designers and developers, notice individuals or small businesses. Microsoft's not really drawing any lines here, you just simply have to register, stay in contact with them, provide a little feedback. But the idea here is to make these development technologies, and this includes C# 2010 available to users and developers out there, to encourage development on the Microsoft platform. So the probably easiest way to do this, is to go to Microsoft.com and do a search for DreamSpark if you're a student or WebSiteSpark, if you're an individual or small business. And you want to learn more and get your hands on some of the Microsoft technologies. Now what do you get in this Microsoft program? Well you get all kind of free technologies, because Microsoft realized a few years ago, as Apple began to inundate the market with the, the MacBooks and all the various Apple products. That Apple was providing their Xcode development environment absolutely free of charge as part of the operating system. If you didn't get it as part of the operating system, if you didn't get it as part of the operating system, you simply went to Apples website and downloaded the thing for free. Visual Studio, on the other hand, Microsoft's development product, for working with the .NET Framework, cost you, you know, 1,000 Dollars, 1,500 Dollars, 1,900 Dollars, depending on the version you got. And as a result, what do you think students have more money for? A free download or an 1,800 Dollar download? And you guessed it. A lot of people were just, by default, based on economics, moving to the Apple Xcode environment and so Microsoft began to offer theirs. Now what you can you get in these programs? You get a fully functional absolutely free version of Visual Studio Professional for, like 2 years. Then you can get Expression Studio if you're a designer, you get a copy of Windows Server to test it on. You get a full copy of SQL Server, the database server to test it on and many, many others. But the bottom line is, if you want to learn to Microsoft development and if you want to do Microsoft development, they will give you the tools free to get you started. You just have to go on the website, qualify for these programs, sign up, jump through a couple of hoops, tell them a little bit about you, what you're going to do with them and just be involved in the program, and you will get these things absolutely free. So just wanted you to know that. If you're a C# developer or if you're trying to learn C Sharp or the Microsoft platform, this is an excellent way to grab these tools. So go check out DreamSpark and WebSiteSpark out on Microsoft's website.

(Visual C# 2010 Express / Getting Visual C# 2010 Express)

In this course, I'm going to use Visual C# 2010 Express to build the code examples and to show you some demos as we move through here. And what I want to show you in this video is how to go out and find Visual C# 2010 Express. Now this is not a big deal unless you're totally brand new to this stuff. So if you just go to www.Microsoft.com and you see it right here and just do a search when you get there and the easiest thing to do is just Visual Studio Express, just like you see right here. And in the search results that comes up, notice you will see, Microsoft Visual Studio Express. Notice in mine, I can see the Visual Studio 2008 Express and 2010. Just go to the main one because what's going to happen inevitably, as, as time goes by and this course stays out there on VTC's site, you may have a little trouble getting Visual Studio 2010 Express, we may up to 2012 or whatever. Those will probably work for the backward compatibility, for example, if you're trying to play with C# 2008, you should be okay running this in the Visual Studio 2010 Express. So you should always be able to grab these notice if I just look under Visual Studio and get the free Visual Studio Express products. I can do that and so right now, notice right here, there's the different versions out there. There's Visual Web Developer 2010 Express, Visual Basic 2010, Visual C#, which is what we would get for this course, and then there's a C ++ 2010 Express. And then, if you don't want to be dependent on having an active Internet connection when you install this, you can click this and download ISO images, so that you can install them, you can burn that to a CD and so forth. But now this is like a 694 megabyte download, so I'm not going to do that here. Nor am I going to include it in your course, so you just have to go out and grab these, whatever's the latest. So what you want to do is, just go out and find the latest Visual Studio, Visual C# 2010 Express link and click on that and it will take you to the installation. Now keep in mind also, that you could use Visual Studio 2010 which you can get on Microsoft's website as well and notice if I just back up to here and I'll back up and notice, Visual Studio and that is the full functioning version of Visual Studio 2010. Notice you can get a trial download for this. Now I'm not going to use this in the course because it can be a little bit more challenging for some people to install. They may have had this before and they've expired on it and it won't reinstall. So I'm going to go with the full fledged free version of Visual C# 2010 Express. Now here's the reason I'm showing you all this in this video go to Microsoft.com, do a search, this will change, these pages change without warning from time to time but I just wanted you to see. Just go out there and search it, grab the latest 2010 C# Express product and you shouldn't have any problems following right along in the course.

(Visual C# 2010 Express / Installation and Use)

In this video, I want to show you just a little gotcha that may show up when you install Visual C# 2010 Express. Now where you will have the problem is not when you actually install but notice in, in a previous video that was entitled Getting Visual C# 2010 Express, I went through the steps of how to go get the Visual Studio Express versions and notice we are down to the Visual C# 2010 Express. Okay. I just went to Microsoft.com, did a search for Visual Studio Express. Okay. Now when I click on the Visual C# 2010 Express button, notice I have an Install Now and then I can come down here and choose English and Install Now. When I do that, I can just, this is a little confusing and you may see this if I close this, I'm just going to keep looping around. I can install the trial for the full Visual Studio 2010 Professional or I can install Visual C# 2010 Express. Now if I click that, it's going to take me to another link and it's going to give me a little window here, a box pops up and notice it says VCS underscore Web EXE. Don't worry about that name, just click Save File, once that comes down, it'll be pretty quick. Just double-click that executable, the VCS underscore web and then this will actually start to, first of all, it will install the product but what it's going to actually do is, examine your disk and see if you need the prerequisites. If you do not, then it's going to go grab those things, pull them down and it's going to gyrate for a little while. I think it took about 10, 15 minutes on my machine to actually pull everything down, install it and then Visual C# 2010 Express was on my machine, no problem. Now it's pretty straightforward on the install but what I want to show you here is, once I installed mine, then I had some other things on my computer from prior operations that I was doing. And if I go to All Programs, you will notice under Microsoft Visual Studio 2010 Express, I had already installed the Visual Web Developer 2010 Express. And also, a long time ago, had the Visual Studio 2010 Professional trial on here, and they're giving me another trial link here. But now the Visual C# 2010 Express went on just fine. But notice, when I try to open it, the little Splash page just flashes at me for just a second and then I get this, looks like an error message and it's warning me that only some of the Microsoft Visual Studio 2010 products on this computer, have been upgraded to Service Pack 1. None will work correctly until all have been upgraded, if you see that notice you can just download the Service Pack 1 and if it starts in Maintenance Mode, select Repair. Notice if I click this it's just going to close and I'm not going to open. This is not a big, huge, major deal just simply go open Visual 2010 Express for C#, click on that link, jump out there and then notice Visual Studio 2010, Service Pack 1 Installer. Follow that link, install that notice this is going to be another one of these links, 795 kilobytes. It'll download very quickly, then we're just going to double-click the EXE right here. We're going to let it come down, scan for viruses. I'll just double-click that and let it run and that will now start to pull things down and install them. You may well see that if you've had other Visual Studio products on your machine. And this'll happen over and over, if Microsoft has released Service Packs for whatever you have loaded on your machine so just wanted you to see this. Don't let it throw you, just follow it, go through the steps. Notice you can also give the Service Pack in ISO format if you would like and you don't have to depend on Internet connection for that, and you can do it somewhere without Internet. And notice it's giving me a, a question here, do I want to reapply it? Or do I want to remove it from this computer? I want to reapply that and what's happened is, something here has got it, but everything hasn't. Most likely that old Web Developer has it and my new Express does not. So I'll just reapply that by clicking Next and then we'll go from there. So anyway just keep an eye for that and don't let it throw you and then once you do all this, you should be up and running with Visual C# 2010 Express and ready to write some code.

(Visual C# 2010 Express / Visual C# 2010 Expre ss Tour)

Once you have successfully installed Visual Studio Express and specifically Visual C# 2010 Express, you will then be able to see it down here. If you click on the Start Button and go to All Programs, you will see Microsoft Visual Studio 2010 Express and then under there you will see C# 2010 Express. Now if you have any other Express packages on your machine, you will see them listed here as well and you may also see this little advertisement, if you will, a link for trying the full Visual Studio 2010 Professional version. But if I click on this link, now keep in mind, if you right-click on this link, I can pin it. Notice I have mine pinned to the start menu so it's saying unpin. But if I, I unpin it and then come back and look, right-click on it and notice I can pin it to the Start Menu. Now, when I click on the Start Button, I can see it right here and I can click it and it makes it a little easier. You'll get a Splash screen and then Visual C# 2010 Express will open up. Now for the most part, this is going to look identical to what you will see on the full Visual Studio 2010 project, with just a few minor cosmetic differences. There are some functionality differences that I won't go into here but they won't affect us in this course here. But notice when this comes up, it's giving you a couple of links, one to Create a New Project or to Open an Existing Project. And I'll do a separate video called Understanding Solutions and Projects and we'll talk a little bit more about those where to find them once you've closed them out and saved them and so forth. But notice on the Start page that you get right here, it's going to give you some links here. There's a Welcome screen and you can go to some Beginner Developer Learning Center and so if you click on that, it's actually going to go out. There's a browser built into the Visual Studio environment and so you can go out there and get some information and some links on the various things and notice they've put that in a Tab. Here's your Start page still on a tab and here's a Tab here and I can just hit the X and close that out. Then you'll notice there's a Latest News and I can set the RSS Feed and enable this and this will update the News Channel and it will always give me the latest that Microsoft's putting out in various places on their website. And then of course, I can just click on that, it'll jump me out to Microsoft's website and give me information, for example, like this is for Microsoft Expressions. It's a cool way to stay on top of what's going on. Now notice I can tell it to close this page after the project load which means the Start page will go away, it won't be a Tab and I can determine whether or not I want this page to show on start up. So that's where I can set that. Now to create a new project, notice I'll just select a New Project and I have some choices. Now throughout this course, we are going to choose the Windows Forms Application notice we're using Visual C#. We can give it a name down here, I'll just let this one be Windows Forms Application 1. Then I'll just click OK and then notice this is going to do a little gyrating down here. It's saying, notice right down here on the bottom left, creating project, Windows Forms Application 1. It brings up some information here in the Solution Explorer, we'll talk about that in a separate video and then this is my form. It creates my first form for me and I'm ready to go. Now over here on the left is the Toolbox and if I mouse over this and click this little Push Pin, notice this thing will stay up all the time. This one has the Push Pin that's a vertical, which means it's going to stay up and I can resize as needed on either one of these. But what's really neat about this interface and this is old news for almost everybody unless you're brand new but you do need to see this. This is called the Push Pin and if I click it, then Solution Explorer will go away and if I click this one, Toolbox will go away. But notice they're still here, right above the mouse right here, is Toolbox and if I mouse over it, that will pop out and I can choose what I would like and when I mouse over, off, of that it goes away. Same thing for the Solution Explorer. Okay. Now if I click on any particular object here, I want to see properties for that object. Now what I'm going to do is come out here and pin that up and if I go to the View section of the Menu up here, I can choose Other Windows. And then I can come down and choose Properties Window and this will show me the properties on my form 1. Okay. And I can scroll up and down through here and see the properties that I might want to see. Now if I go to Toolbox, make it come out and pin it and if I grab a button, just left-click it and drag it on and let it go. Notice as long as I've selected button, now it's going to show me the properties on my button. So for example, text is button 1. I can come over here at design time and change that to Mark. And as soon as I click anywhere else, I've changed the text on my button to Mark. Okay. So that's just kind of the way you can move around in here and anything else that I open. For example, if I open the Program or if I double-click my button, it takes me to the Code Behind page, where I can write my code. Notice that's the Form 1 CS for C#. That's where I can actually write my Button Click Handler and these are all going to stay up here in Tabs. Now granted, the Visual Studio environment can be a little intimidating if you're brand new to this. Because you're going to get Tabs across here you can get things going here. This is now kind of tabbed inside of this. Okay. And you can pull these up individually but again notice, if you pin this and then bring Properties out and pin it, they kind of stack. There's a lot of things you can do, I can grab this and move it around and put it in different places and so forth but you can play with all that and see what's going on. I just wanted to give you a quick little tour of what's happening in the Express Integrated Development Environment. And again, this is very, very, very similar to what you're going to see in the full fledged, Visual Studio product. Okay. So I'm not going too deep here, you'll see me doing a lot of things throughout the course. So if you're new to this, just kind of watch as we do demos and you'll see a lot of other things. And of course, I haven't said anything about all of our buttons up here, our little shortcut buttons. So anyway, that's a quick look at the Visual C# 2010 Express Integrated Development Environment.

(Visual C# 2010 Express / Understanding Solutions and Projects)

When you work with Visual C# 2010 Express, you're going to be working with a set of organizational functionalities called Solutions and Projects. And so what I'm going to do is open Visual C# 2010 Express here and I'm just going to create a New Project. So I'll click on New Project, we'll take all the defaults, Windows Forms Application, Visual C# is our only choice here. And I'm just going to leave the name that I could change right here as Windows Forms Application 1 and then I'll click on OK. Now notice this set up for me over here, a Solution and a Project and notice it's mentioning right here, that I have 1 project in my Solution. Now this gives you a hint that I can have multiple projects and in this first project here, Windows Forms Application 1, it gave me my first form and then I can go out and enter the Toolbox here, and just double-click on Button and I can drop a button onto that form. And then if I double-click that and jump to the Code Behind, you will notice that I'm working in Namespace, Windows Form Application 1, same as my project. Now I can also right-click, I'll left-click on Solution first and then I'll right-click and I can add a New Project to this solution and notice it's warning me here, the current project must be saved, before adding a new one. So okay, I'll take that, notice I'll name it Windows Forms Application 1. I could change it here if I wanted to, notice I can also change the name of my Solution name here if I would like, and notice this right here, Create a Directory for a Solution. It's generally better to leave that checked in most cases and I'll show you why in just a moment. So anyway, we'll click Save here, that saves the Windows Forms Application 1. It gives me the chance now to create the Forms Application 2 and I'll click OK here and notice it creates another Windows Form Application. Now I'm going to go again into Toolbox, double-click Button and drop a button onto this form. And notice if I double-click this one, I am now in Namespace Windows Forms Application 2 as you can see right here. So one of the things that'll I have to watch between moving between these forms, I can come back up and go into this form and notice I'm back in Application 1 now. Okay. So you'd have to watch for this but it's very easy to work with multiple projects in a single solution. Now one other thing I want to show you here, is each of these projects, notice I can expand and collapse these at will and I can see for example, properties on my various aspects of my project. I don't want to go too deep here, references to this project. Notice these are the Namespaces that have references to right now, System, System Core, System Data. So if I want to use another Microsoft Namespace, I just right-click Reference, Add a Reference and I come out into .NET and then you can see the various System.Directories Service. System.Drawing, those sorts of things. And I can even set references to other projects. Okay. So I can work with that and anyway, all these are out there, and you can do this collapsing thing and you can get to all these sort of things within each particular project. Now one other thing I want to show you, before I leave you here, is right-click on Solution and go to Properties and you can see, that I can set Dependencies. I can set all kinds of information, notice the single start up project, which project do I want to start up in this Solution? So I can choose which one is the start up application or the start up project in the Solution. Then I can go to each individual project and right-click and I can go to Properties, and I can work with various properties. Notice Application, my Assembly Name, the Output Type here, let me move this over, so you can see it all. Whether it's a Windows Application, a Console or a Class Library and then notice I can do all kinds of things, Builds, Build Events, Debug, Resources, all kinds of settings. Reference Path Signings, Security. So anything you need to do, you can get out here at the Properties level, on the individual projects as well. So anyway, I just wanted you to see these. Now the last thing I want to show you here is, once I've created these projects and I'm done with them, notice I can close these Tabs out at up here. But if I come down to, in Windows 7, I have the Libraries Folder here but if I go into this and go into Documents now, you will see that there is a Visual Studio 2010 out here. So if I double-click that and go into Projects, you will see, there is my Windows Forms Application. And you remember the little checkbox back there a few minutes ago, where it said, you know, do you want to save this in it's own directory? If I go inside there, there is my Solution file, notice Microsoft Studio Solution and then this is the collection of all the individual files within the project. For example, if I want to see my code, I can just right-click this and I can even open it with something like Notepad, and I can get a glance at my code inside there. Okay. So lots of ways to manage this and work with it. I hope this will help you understand where are all your files are going. What's in there, how to move around, how to adjust things if you need to.

(Visual C# 2010 Express / Your First C# Program)

For those of you who are brand new to this whole Microsoft development world and especially with Microsoft's Development Tools, I'm going to do a very short video here. A very simple, your first C# program. So what we're going to do is, come down and click on Start, we'll open Microsoft Visual C# 2010 Express. So I'll just click on that and open it and then I'm going to create your first program. Now this one will not be out there in the Work Files, you just need to follow along and do this. Okay. So notice New Project, that's what we'll do, and Windows Forms Application. We'll just change the name down here to, First Program. And then we'll take OK, click on OK. Notice it has created for me, over here on the right and will do for you as well, the Solution called First Program. And it contains 1 project and that project's called First Program. Well, here's our form, I'm going to mouse over Toolbox, let that fly out and then I'm going to pin it up so I can see these things. Now there are a number of ways, you can put your controls onto the form. I can just double-click it and notice it puts it in the upper left and notice I've highlighted that, I'm going to hit my Delete key and take it away. Or I can drag it on and wherever I drop it, the upper left corner will start right there. Okay. So I could have easily dragged it down here and then I can, of course, very easily drag it later. Now what's really cool is if I drag another button on, you'll notice these little blue lines pop up. So that if I want to put these things, you know, in places where they're lined up and this is taking care of alignment for me. Okay. So I'm just going to delete Button 2. Now if I double-click Button 1 I'm going to come out here and write some code. Then I'm going to make this Solution Explorer and the Properties go away so that I have some room and what I'm going to do is, just put a nice irritating Message Box. So Message Box, now notice, how intellisense is trying to help me write this? It's showing Message Box here. If I just hit the dot, notice I get Message Box Show and then when I open this, you can see all my Overloads and you will see how to do Overloads later. These are the various options I have for the parameters. Alright. Well the first one, notice, I just put a, a string text in here, and so I'll just put my first program. Then I will close that and then at the end of all our C# lines, we need a semicolon. Notice if I don't put the semicolon and I go to the next line, I'll get a little red squiggly, and when I mouse over it, it'll usually throw it, tell me, give me a hint on the Error Message. If it don't I can look right down here on the Error List that comes up and double-click, and it'll take me to where the problem is and there it is. So I need to put, it's telling me semicolon is expected so I can just hit it and I press Return and boom, there's my semicolon. Now to test this or to run it, I will go up here onto the, the Toolbar up here and just click the Play Button or actually, Start Debugging. And this is actually starting me in Debug Mode. I don't have any Break Points in here, so it's not going to stop and I click on the Button and boom there is my little Message Box and my first program. Okay. You'll notice there actually wasn't a boom but notice since I'm starting debugging, I have a Locals Window and a Call Stack to help me debug what's going on in my application. And if I'd had any errors here, then notice if I leave this off and I've got an error out here and I try to build it, it's going to tell me there were Build Errors, do you want to continue and run the last successful build? Well it's probably not going to help you, so I'm just going to say no and go figure out what my error is and take care of it. Alright. Notice this Error List down here is just like all other windows. I can determine how much of the screen it takes up, by clicking and dragging or I can unpin this and then it will show up later when I need it. And notice I tell it No and boom it popped up and notice it popped up to the, the last place where I had dragged it and sized it. Okay. So anyway, that's as simple as it is. Up here when I run my program, this will kick it off. This will give me my Run List and let me show you one more thing here before I leave you. Let's kick this off again and if it's a large project, you will see an Output Window down here let me go to View, Other Windows. Well I'm not going to have it here for right now, I'll show you that a little bit later on. Okay. But there's lots of things that are going to happen in this interface, just get comfortable with it. Notice right down here, here's my Error List Tab, I can always bring this up, I can see what's going on, I can see messages out here. So anyway lots of things happening here, that's a very simple first program, that's the basics for running programs in Visual C# 2010. Now there are some other things happening in the background and we'll get to those later. But if you can get your head around that, you're ready to go.

(Visual C# 2010 Express / Commenting Your Code)

If you want to see a bunch of nerds get into a debate, just get 2 or 3 in a room and ask them about Comments and when should you Comment? How much should you Comment? Now everybody agrees that there should be Commenting in code but nobody agrees it seems on exactly how much Commenting and when to Comment. But what I want to show you in this video is how we do Comments in C#. This is pretty straightforward, it's not a huge departure from, from anybody else out there in the development world for the most part but what I'm going to do is, drag a button onto a form. Notice I did a little different way that time and, and let me delete this and go back and show you. I clicked on the button, came over here and notice I've got this little Plus sign and the little A B. Now I can drag and build the button, what size I want and when I drop it. So if you ever want a button that large, that's where you can do it or you can just drop it on there and then resize it with the handles. Right. But anyway, we'll leave our button like this and just double-click it. Now what I want to do is, when I click on this button, I'm going to create an Integer. I'll do an Integer A and I'll set it equal to 5 and then I will do a Message Box.show that says you know, Variable created and I find, don't know why I would do something this useless. But hey, I've written a lot of useless code in my time. Okay. Now notice it's giving me a little squiggly here that says that Variable A is assigned but never used. And that's okay, we'll live with that for now but, so I've written some useless code here that looks useless. But why would I want this Message Box here? I can put some Comments in here and the easy way to do this is, 2 forward slashes and notice it's turned some things green. This Message Box is totally useless and notice this is going to set everything on this line. Now something else that you can do and I'll take it away here and you can put it at the end of a line out here. Okay. And notice, this line will still execute but I have a Comment at the end. So the 2 forward slashes will put a Comment just on that line, create an Assigned Variable. Now if I want to Comment a bunch of things and this is kind of a trick a lot of people use qnd that is I want to Comment both of these lines out. Well I could come right here and put the 2 slashes on the front and that just Commented those 2 lines out. Then I have to go manually, take these things back off again alright, to undo them. I can just highlight both of these lines, let me get this one off the end down here, so I don't confuse you anymore than I probably already am. Notice if I highlight the 2 lines and I come up here on the little Toolbar and if you don't see this Toolbar, you can go up here to View Toolbars and you can see there's Standard and Text Editor. I'm working with Text Editor here but highlight these 2 lines, come up here and just click Comment. It will put 2 slashes in front of every one of those lines that I have highlighted. Then when I'm done with it, I can come highlight these lines again, click this one and Uncomment. Now this little trick with these two buttons, it's amazing how many people either forget about these or don't realize they're up there. It's a neat little trick, I noticed that if I want to Comment out a bunch of stuff, I can highlight everything and it just Commented out everything or I can undo it. Now what's interesting is, I can just keep doing this and then I have to keep undoing it to get it back. Okay. But that's one way. Then if I want to Comment a, a whole section out and not use the Toolbars, I can use the forward slash and an asterisk and notice everything after this is green. So it is Commented from here, all the way down to here. The way I stop this Commenting is, I can come right here and I can put the asterisk and the forward slash. And notice now everything between these, notice I started it with a forward slash asterisk, I close it with an asterisk forward slash and this one is now Uncommented. So I can use the asterisk forward slash, I'll delete it here. I'll grab this one, I drag it up here and notice everything is green. I'll come down here and put it and notice it's Commented everything from here to here and now this is open. So you got a lot of options here with Comments. Use Comments, first of all, to document anything in your code that's going to be hard to figure out when you come back, a week, 2 weeks, 2 months, 6 months later and try to make changes to this code, and update it, enhance it whatever but also for other developers. Now there's always this debate about how many Comments? How much should I write? To what level of detail? And trust me on this, you'll never satisfy other developers. Write yourself enough in Comments so that you can understand what's going on, so that they can understand what's going on. And just keep in mind, you're always going to hear with your Comments, this person doesn't believe in Comments or the ones they wrote were too cryptic, I couldn't understand them. Or they wrote their autobiography in Comments and I don't understand that. So you'll always hear one or the other but I would fail on the side of writing too many Comments maybe making them a little too detailed. Because it's amazing how many times you come back to some code a few months later and you're thinking, was I on some real like, really strong sinus medicine that morning, because none of this code seems to be making any sense. Why would I write it that way? As it always usually is in life, there are reasons for doing what you're doing; they just don't make sense later. Okay. So anyway that's a quick look at Comments in C#.

(Visual C# 2010 Express / Collapse and Expand Code)

The Visual Studio environment gives you a functionality that you need to be aware of. This is not really a C# functionality, it comes to us through the Visual Studio Integrated Development Environment. But it's a very neat tool that you can use especially when you're trying to troubleshoot code or when you're trying to work with certain parts of code and that is and you'll notice these little boxes out here on the left side of the code. We can expand and collapse this code and this stuff is done for us automatically. For example, you'll notice that this entire Namespace, if I click this little button, it collapses all that code up and these little three buttons out here, if I mouse over it, it'll show me what it looks like. How cool is that? And then I can expand it back and so this particular class, I can click that and shrink that entire class. But I can come out here and look and see what's in the class, and then for example, this particular, let's do this one. This Button Click Handler which is really the same as a method in most cases. I can compress just that or collapse just that, and then I can mouse over the Ellipses out here, the 3 periods and I can see the code that's inside there. Now this is huge because as I create my own later, you know, Private Void Add this. So I create my own method out here and you'll notice that it's automatically given this thing okay? And then I'll just put some Comments, add code later right here. Then I can collapse this, come over this and I can see that all I've got is Add code later. So you can see that I could collapse all of my methods, so that I just see the names here, and then just by mousing over, I can see what code I'm looking for. Something else you can do when you're troubleshooting and you're trying to look for problems in your code, you're trying to debug. Once I know that this is working properly I can simply collapse it and then I can come back and check this later and say, okay, now this was working right, I can collapse it. Then of course, anytime I want to, I can get it back. Now notice I can right-click here and I can work with some of this stuff on Outlining. You notice, I can toggle Outlining, toggle All Outlining, and this will get it all back open for me. So lots you can do with this and again this is not really a C Sharp feature, it's a feature of the Visual Studio Integrated Development Environment. But it can really be handy when your code gets a little more complex than what I've got right here.

(Visual C# 2010 Express / Using Code Snippets)

Now there's another feature that's kind of a marriage of the Visual Studio Express environment and the C# programming language and that's a functionality called Code Snippets and a lot of developers just simply don't know these were there. They've never noticed them but I want you to see these things that can really save you a lot of time. For example, let's say that in this little method that I've added here called Add This, and we'll talk more about methods later. So if this doesn't make sense to you, don't worry about it and I want to put a Do Loop. Okay. A Do, Do While. Well, so now I start to think, well wait a minute now. How I exactly do I build that? And I have to go look up an example if I don't do these every day but notice if I just type Do, you will notice, right here, there is a Code Snippet for the Do While Loop. Now if I double-click this, nothing happens. Okay. And that's a little confusing from what I'm used to seeing. But I'm going to go back and I'm going to type Do and then just Tab twice and notice it has set it up for me. Now there's lots of these Code Snippets out there. For example, if I'm going to create a class, I type Class Tab twice and notice it has set this right here, up for me. So now in yellow, I just have to go back and put in the name that I want to build my class on okay? So I'll go delete that, that's not a good place. Let's see, I can create a, let me go down here and let's do this again. I'll do Class notice there's Class, double-click and I'll just call this Mark and then I want to create a Constructor, so I'm just going to do C Tour double-click and notice there's my Constructor for the class. Okay. If you know what Constructors are, a light just came on, if you don't, don't worry about it, we'll just take that right back off. So there's a lot of things I can do here for example for loops, for, let me get over here, let's see, For Loop, For 2 Tabs, there it is, very, very easy to get things happening. There's a lot of these out there, if you just go to Google and just do a search on C# Code Snippets, S N I P P E T S, and you can see these. Another popular one we like to use a lot of, Switch Statements in our coding and so if I just come up here and type Switch, 2 Tabs, there it is and it starts to lay this out for me, makes it very easy to do things. So anyway, Code Snippets are really cool, you can recognize them out there. For Example, I can type Class, you'll notice it's got this little icon here and all I have to do when I get that, is just Tab once, Tab again and there it is. So anyway, use Code Snippets, watch for them in Intellisense as you're out there looking around, and you can actually learn these as you go through coding something that's not even related. But you'll notice those little icons. So anyway that's Code Snippets. It can really save you a lot of time in your coding.

(Visual C# 2010 Express / Refactoring Code)

In this video, I want to show you an example of a real powerful feature in C# called Refactoring Code. Now again this is a functionality of the Visual Studio environment, but it's a great tool you need to know about. Let's take a look at something here where, let me, let me do something here, just don't want to confuse you, but, anywhere in here that you've got some code and let me you show how I got where I'm at. I've just got a form and I dropped a button on it and I double-clicked and I actually had already written some code here. But in my Button 1 Click Event a couple of things I want to show you about Refactoring. Okay. I can Refactor or change aspects of my code automatically. What I'm going to do is, jump down here, outside the Namespace, notice where I'm at here and I'm going to create a class. I'm going to call it Public Class Mark and then I will just put a little Public Void Add This, put a little method in here. Okay. And then we would later add some code here. Alright. Well notice when I come up here, I can create an instance of Mark. If you don't know what I'm doing, don't worry, I just want to show you something kind of quick here and set that equal to, not a new instance, but a new instance of Mark and then notice when I hit M and a period. I can see my Add This Method down here and so I will tell it to call that and that's pretty straightforward. Alright. Not a lot's going on here. There's absolutely no functionality here, I'm real good at writing useless code. So what I'm going to do now, is show you how I can Refactor. Notice if I get on the Class Name Mark and right-click, I can choose Refactor and notice I can rename this. And let's say, instead of Mark, I want to call it, VTC Mark and notice I can preview my changes, I can search in Comments, search in Strings. I'll just say OK. There's a preview of the changes that are going to happen, this is what it's going to look like here and I just apply it. Now notice everywhere that it was Mark, it has been changed to VTC Mark. A really cool way to make changes to your code. It will get all of them for you, most of the time a lot of neat stuff can happen there, well let's look at something else. Within my class here, in Mark, I start to write some things inside my Add This Method. Okay. And what I do is, I create a String called A and I set it equal to Hello World. Right. And then I create a Message Box and I call the Show Method and I'm going to print my string out. Then I realize, hey you know what? That really should be a method. Okay. That shouldn't be inside here. I'm going to be calling this Hello World a lot and doing a Message Box, don't ask me why. Okay. And so what I can do, is highlight the code, right-click, go to Refactor and say Extract Method. Left-click on that, what do you want to call this? Well I'm going to call this the Useless Message Box Method. Okay. And I hit OK and notice it created a Private Useless Message Box Method for me and in place of my code where it was, it put the Useless Message Box call here. Alright. So that's Refactoring, a couple of things you can do very easily here. You will find yourself all the time, within one method, writing 8, 10, 12 lines of code, thinking, man I might have to call this some other places. This really should be a method on it's own. Fine, just highlight it all, right-click, Refactor, make it a method or Extract a Method. So anyway that's Refactoring so keep that out there in your little developer toolbox in your brain and pull it out when you need it, a lot of cool functionality there.

(Object Oriented Programming / Object Oriented Development pt. 1)

One of the things you'll always see about C# if you're especially new to this whole world of C# programming is the fact that C# is touted as an Object Orientated language. Now most languages are these days but if you're new to this term Object Orientated, I want to spend just a little time in this two part video explaining what Object Orientated Development is. What it looks like and then in Part 2, I'm going to draw you a little animation so that you can kind of get a visual in your brain of what's going on. First of all, understand, this whole idea of Object Orientated Development is programming that's based on the concept of objects. Now an object is a Memory Resident Copy of a class. So you'll hear a lot about classes and the basic idea is, that we are going to build classes that are blueprints for objects. Now the class contains whatever functionalities that we want all of our objects to have. That single class that we create is the blueprint for multiple objects that we can use at one time in memory. And each object is said to be an instance of a class alright? Now that you're thoroughly confused with all that, let's go on. Each object that I create from the class will have it's own state or it's own set of properties. Keep in mind I have my own state, I have a certain height, a certain weight, a certain age, a certain hair color. You have your own state, that is your height, weight, hair color, eye color and these can change from time to time. Especially you know, you can get a haircut. Ladies and a few men nowadays color their hair differently, so you may see on, you know, one day, they look one way, another day they look another way. They may get new glasses, may get color contacts, so our state can change. But each one of us individually has our own state, our own set of properties that make a unique us. But yet we're all built on the same class which is human beings. Now the properties and methods of each object come from the class. The properties are those things that I was just referring to. Hair color, eye color, height, weight, something that describes that object. A method is something that object can do. Some people can play a guitar, that's a method for that particular object. So I might be able to write very well. So that might be another method. So property describes aspects of the particular object Methods or actions the object can take. Now let's kind of put this in real world. Objects really represent real world things. Good example, let's say that we're hired to create an office application for some veterinarians to operate their office. Well think about the objects involved in a veterinarian's office. He has patients, they're animals, but still they're patients. And then really in the end game here, the real patients are not necessarily the animals but the owners of the animals who are paying the bills and making the decisions, whether to go through with treatments for the animals, or maybe to take them to another vet. And then there are the doctors themselves, the veterinarians and then the other object is the owners of the patients or the owners of the animals. Then there's the actual animals themselves. Then after animals in general, we may see dogs, we may see cats and we may see snakes, anything else and some of the other weird pets that people have. Not that they're weird but, I've held a snake before and it wasn't pleasant. If you like snakes, good for you. I'm not going to have a snake for a pet anytime soon though, let me tell you. But anyway, then we also have an account. So every client here, every owner has an account where we handle their billing. And so if we were going to write an application here, we would probably build objects that take care of all the things surrounded by this. Okay. Treatments the animals get, billings to the account, patients name, address, phone numbers, emails, payments they make. Any kind of thing like that. Now here's what happens a lot of times. And there's just one example, there's a lot of different ways to do this but a lot of developers will, in the process, of what's called gathering requirements or creating a requirements document. Is where we try to determine, now wait a minute, what exactly do we want this application to do? Because if we don't spell that out on the front end, how we will ever know that we've reached the end and we've completed it? Because we never really know if we've got it doing everything that we wanted it to do. So a lot of times you'll ask the person that's hiring you, just give me a good concise brief description of what you want this application to do functionally, when it's completed and when I deliver it to you. So describe what you guys do here. Describe what you want this application to do, and you'll see something like this. Provide data maintenance for an office of 4 veterinary doctors who provide scheduled and emergency exams and treatments to household pets. These are dogs, cats, snakes, et cetera and pet owners in good financial standing are mailed a bill at the end of each month for services rendered. Now that's the good overview of what's going on in this office that we're going to write an application for. And what a lot of developers will do, is locate the nouns in this description and these become excellent candidates for classes. And notice I didn't say they would become classes, but just look at some of the nouns here. Office, doctor, pets or animals and then we have dogs, cats, snakes. So animals in general all have certain things, they have a height, weight, hair color whatever a color, that sort of thing, an age. But then dogs, cats, snakes are a little different obviously and they might require some different treatments. Then there are owners of the pets, notice right here is our owners. Then they're mailed a bill. So look for the nouns. Okay. There are some more nouns in here that we might want to deal with. But this gives us a good suggestion of what might be a class in our application and then if you locate the verbs, these are great candidates for methods. Now if you don't understand methods just yet, don't worry about it, you'll see them later on in the course. So if we locate the verbs in this little paragraph, notice scheduled, emergency, treatments, standingness in the green boxes here. Those are Actions that we may want to perform or have performed in some of our classes. For example, a good financial standing. Well how we do determine that? We're going to have to check it against something in our database to see what's going on there. Scheduling, for the scheduled portion. Our clients or our patients, the pets have to be scheduled. Some of them were emergency situations and that might be, you know, additional billing. So anyway, that's how we're going to determine a lot of this and understand there's going to be a fight. If you get 2 developers looking at one of these same paragraphs, they're going to come up with a different final design. That's okay. But this will give you an idea of how we kind of get started with Object Orientated Development, join me in Part 2 and we'll do a little animation on what comes from this.

(Object Oriented Programming / Object Oriented Development pt. 2)

Welcome to Part Two of Object Orientated Development. And in this video, I want to show you just a real brief animation that really should kind of make it click about the relationship between classes and objects and how you instantiate those objects. What we're going to do, with say our fictional veterinary program here, is notice we're going to first code up a class. Now notice we do Public Class Animal, you'll see this a little bit later on in the course. But we're going to set up some Strings here for some variables. These would actually be Properties, we'd do them a lot differently than this, but to make it simple here. Notice for every Animal Object that we create, we want to store a name, the type of animal, whether it's canine, feline, whatever the other ones are, I don't know. The weight of the animal and then some, something about the bill here. I don't know what that would be but anyway. And then we're going to have office visit as a method for this particular animal, and this also might even better put in an Exam Class or whatever. But this is just a little function if you will that runs and this becomes a method on the object. So it just takes the amount of, the difference between the time the patient came in to the time they leave and we subtract that out and set it equal to bill. Then we return bill and then we'll multiply that by whatever hourly rate or whatever, you know, metric that the doctor wants to use. Okay. But anyway, that's a class. Okay. So we code that up and we have it out there. And then we can execute this command. We can put this command in our code and notice animal is a Type and it's talking about this class which is a Type or a class. And then we're going to create an Instantiated Object off, of this class or this Type called Animal. We're going to call it A and we're going to set A equal to a new instance of this Animal Class. Now when we execute this command in our code, what happens is, an object called A, notice right here, A, gets built on the class Animal now exists in memory. And so we can set the String Name, the type, the weight, the integer of bill and we can set those and those will apply only to object A. Now if you're hard core C Sharper, don't worry about how I've got these set up. Okay. But these would be our Properties. So object A has a distinct name, type and weight and this is in memory. Then we can do another command, Animal B equals new animal. So we're creating a variable or an object called B based on the animal type and then we're creating that in memory by passing the new keyword and telling it a new instance of the Animal Class. And that creates another object that starts out identical to where A started out. And this one is called B and B can have it's own unique name, type, weight, bill and it can execute this method. So we create this object, say for you know, a dog named Spot that comes in. This is for a cat named Fluffy that comes in. And Object A for Spot can write this stuff back and forth to a database, store information, change things. The method can just record in the database what we did to Spot. What treatment Spot received, what should be billed and all that sort of thing. Then down here, Fluffy the kitten has the same things working. So basically we have just simply built 2 objects in memory that represent this real world Instantiation of an animal. Okay. Spot the dog, Fluffy the cat. If you can get your head around that, if you can get that visual in your head, that will go a long way to understanding this whole idea of Object Orientated Development. We're going to create objects in memory, work with those. And then when we're done, we'll simply destroy that object, take it out of memory and we've already written our information to the database or whatever we want to do with it. And then the object goes away and we can continue. It's a very efficient use of memory and if we ever want to change how all of our objects work. We simply go back to our class, make a change in the code. The next time somebody creates a New Object, it has all that functionality and any future objects that get created. So that's a real quick visual overview of Object Orientated Development.

(Object Oriented Programming / OOP Basics)

And now let's talk about some Object Orientated Programming basics. Now anytime we're talking about Object Orientated Programming, we're talking about a certain way of setting up a program and the benefits that it gets us. Now to understand this really, take a look at a couple of videos I did in a separate part of the course, called Object Orientated Development part 1 and 2. And this will kind of give you a little chalk talk and then a visual and then in this video I want to talk about what all that type of programming actually gets us. Well, when you think about the old way that we used to program, we used to write sequential lines of code that ran from beginning to end and we numbered them, 10, 20, 30, 40, 50, 60. And when we started our program it ran from top to bottom and it branched to other lines of code when necessary and it was often quite necessary, quite often to branch around. And so this stuff became known as Spaghetti Code and it was kind of hard to follow, hard to troubleshoot, hard to update, and code that was repeatedly run, later got grouped into Subroutines. That helped the organization a little bit but it was still very difficult to maintain, very difficult to update and Object Orientated Programming came along and it began to break the program into individual objects, similar to real world operation. And so if it was a Real World Object, we made it a Software Object. And each object in our program is totally separate from the rest of the program. These are discreet little sections of code, that come into being and take up spot in memory as a little collected set of functionalities and then they go away. Objects are created in memory when needed and then we discard them when we're done with them. Objects contain the various code functionalities that are unique to the operation and the current state of that object. So we can change properties on the object and again, if you've already watched the videos on Object Orientated Development, you know what I'm talking about here about maintaining state and code functionalities. Let's take an example here. We want to bake a pizza. If I go into my kitchen and begin to bake a pizza, what am I going to do? Well I'm going to work with some objects. There's going to be an oven that I have to work with, a pizza that I have to work with and then a person, which is me that I'm going to have to work with. Now each object is independent of the others. The pizza and the oven are not related except that the oven takes actions on the pizza and causes changes to happen. I'm a person, I make changes on the oven and the pizza, that makes changes happen and so forth. Each object can be created when needed. In this instance, we created an oven when we needed it and we installed it in the house and then I can carry on functionalities when I need it. The pizza was created when needed. Somebody created it because they needed to sell it. I bought it because I needed to eat it and then I'm going to use the oven that was created when I'm ready to prepare it. The person was created, we won't get into all that. Alright. Now each object can be updated or changed without effecting the others. If I call a repair man out and he makes a change to my oven, he does not effect either me or the pizza. He will effect the operation of the oven that could effect me, where he has to say now look. This is the way you used to turn it on. This is the way you do it now. Same thing with the pizza. If they change the pizzas, we may get a new pizza that requires a different cooking time. Now each object in Object Orientated Programming is going to have certain characteristics and these are fundamental to Object Orientated Programming and it's why we got into this business and this is the real beauty and power of Object Orientated Programming. And that is first of all, Abstraction. Now there is a lot more characteristics I'm just kind of hitting the high points here. The user does not need to understand how actions are carried out. I don't know how an oven works, don't know what makes it work. I just know that if I turn certain knobs, it gets hot in there. And if I turn another knob I can determine how hot it gets and so then I can use it. Classification, objects are classified, grouped and related for code reuse and for organization. My kitchen's very well organized. There's an oven and anything that has to do with cooking, has to do with the oven, there's the pizza. Anything that has to do with eating pizza, enjoying pizza, storing pizza, has to do with the pizza. And then there's me. And there's a whole Classification there that we won't get into a usable interface. Objects have to expose functionality in a simple manner. I know how to operate an oven. I could probably come to your house and operate your oven with just a little bit of thought, because ovens all expose a similar, simple interface. Usually knobs that cause things to get hot. The interface is all the end user should understand. It's all they should be required to understand. I should not have to understand how they make pizzas, where cheese comes from. How to manufacture cheese, tomato sauce, any of that stuff. I should just know the interface on a pizza which is, cut the plastic open, put it in the oven for the set amount of time. On the oven, the interface is pretty simple. It's those knobs on the top. I should be able to figure out, turn certain knobs, make certain things get hot. That's what we want to accomplish with Object Orientated Programming. Each one of our real world objects is recreated in a software model and it has these characteristics and actually a few more. But this is what we're headed toward and this makes your program much, much easier to update, much, much easier to maintain and much, much easier to troubleshoot. And so in the next few videos, we're going to delve into some of these things and talk about some other aspects of Object Orientated Programming.

(Object Oriented Programming / Understanding Classification)

In this video, I want to turn our attention briefly to Classification and understanding Classification and how it comes into play in our software design. Now this is not an Object Orientated Programming class per se, but understand that Object Orientated Programming is precisely what you're going to be doing with C#. That's what it was designed to do. So the more you can understand about this, especially if you're new to this technology, the better off you're going to be and the faster you're going to get up to speed with C#. So we're going to do, just 3 or 4 videos here on different aspects of Object Orientated Programming. It's not wasted time, I know it's not C#, but we will dive into C# in just a few videos here and you will see exactly how this stuff fits in, and it will make your C# learning and implementation much easier. So first of all, let's get a definition of Classification. It is, notice, the systematic arrangement of information and behavior into a meaningful entity. Now what exactly does that mean? Well let's just talk about it the way we live it. Classification is a part of every day life. Think about cars. How many different makes, models, years, colors, functionalities of cars are there? But immediately, when I say the word car, you know what we're talking about, some sort of object that we use for transportation, has 4 wheels, is a certain size, not too big, not too small, even though there's a myriad of sizes under that single Classification of cars. Yet when I say car, you know what I'm talking about. Trucks, if I were to ask you in one sentence, what's the main difference between a car and a truck? Some people can nail that, others have a little bit of trouble, saying, well you know, it's kind of hard to say. Generally, trucks haul things. Well cars can haul things but trucks can haul bigger things. And then homes. How different are homes in size? In design? In location? And layout? But yet we classify living spaces as homes. Businesses, we classify buildings operations as businesses. So we do this all the time. Now in software development we find out, that our functionalities are grouped into classes. And so that is the first, that's kind of the seed for Classification and that is these classes that we program. Now classes are then grouped into Namespaces. So in the .NET Framework we have System.Data. And in the System.Date Namespace we'll see all the classes that have to do with database functionalities, grouped together in that Namespace and so we're classifying our classes and then our classes classify functionalities. Now Classification happens almost naturally in real life. When we say shopping, when we say cars, when we say trucks. However, it doesn't seem to happen naturally in software development. There seems to always be a tendency for our software development to wonder off into these Single Objects that do 46 different things. It acts like a car, it acts like a truck, it acts like a home. It acts like a chainsaw, it acts like all kinds of things and then when it breaks or we try to correct something in it, we break a whole lot of other things and that's what we want to avoid. Each class should provide functionality, absolutely specific to the particular object that, that class is replicating from real life. Alright. Let's take a look at an example at here. The Animal Object in our little program, example program that we're using about the veterinarians office. We don't want to put any functionality in that Animal Class that doesn't directly relate to that animal. We don't want to put stuff about the doctor in the Animal Class. We don't want to put too much stuff about the owner in the Animal Class and we don't want to put anything about the doctor's educational requirements or any kind of educational updates in the Animal Object. Only things about an animal goes in that Animal Object. The accounting, the billing or the Bill Object, whatever you want to call it. We don't want to put anything about treatment n there other than how it relates directly to the billing aspects of the business. So a Bill Class that's going to create objects about billing, only needs to contain functionalities. We need to classify the functionalities. So whenever we're thinking about, let's see, I need to think about, where would I go look at whatever is updating our ledger? or updating our past due? Well obviously that's going to be in the Bill Class so I'm going to go into the Bill Class, in that Classification, look for things that effect billing. Okay. So that's what Classification is, when you read about it and when you hear about it, that's what they're talking about. Now you should do this normally. You're doing it normally everywhere else in life, you should develop the habit of doing it normally in software development. Separate your functionalities by class, then the Classes create objects and then one class always stays true to that particular Classification. That makes it very easy to update your project, very easy to troubleshoot the project and very easy for another developer to take it on and take it to higher levels of functionality.

(Object Oriented Programming / Understanding Encapsulation)

Encapsulation is another aspect of Object Orientated Programming that you're going to read and hear about, even when you start to delve into programming languages like C# or VB.NET or any kind of programming language actually. That is involved with Object Orientated Development. And so in this video, I want to talk about Encapsulation, it's a very simple concept but again these concepts while they're simple to understand and you kind of give it a nod. Yeah okay, I got that. When you begin to write code, it's amazing how easy it is to slip away from these things and kind of fall victim to some very common nerd mistakes if you will. Okay, so Encapsulation is one of the most important aspects of Object Orientated Programming or OOP as you'll see it and hear it out there. And that is simply this, that a class should hide the complexity of it's operation from the user and I could add here, as much as possible. Let's think about an oven. When I use an oven, do I know how it makes it work? Well obviously not I'm not an oven engineer. Okay. If those people exist and obviously there's some out there somewhere. All I need to know how to do, is turn the oven on and open the door. Everything else is time. Right. If I leave it too long, it burns. If I don't leave it long enough, then it doesn't get done enough. Okay. So that's all I should understand about an oven. Now operation of the oven is Encapsulated away from you. I turn a little a knob to tell it to take it to 450 degrees, it happens. I don't know anything about thermostats. I don't want to know anything about thermostats. I don't want to have to study thermostats, I just want my pizza cooked. Right. And so that is Encapsulation. The functionalities of an oven, the operation of the oven is Encapsulated away from me, but the operation of the oven is exposed to me by the interface that's offered and in this case, it's the knobs. Now Encapsulation also involves the accessibility of the internal functionality. We'll talk about this later, when we talk about Modifiers on Properties, on Methods, on Classes. They can be public, they can be private, they can be protected. We'll get into that later. But that's part of the whole Encapsulation scheme. There's a lot of things that might happen, if I'm trying to transfer data across the Internet. But do I want the end user to be concerned? No, I want them to type an URL or an IP Address and everything gets taken care of for them. Now it may take me a couple of weeks of banging out some code, you know and almost losing my sanity and finally figuring it out and getting it to work. But in the end, when a user when uses my class, they simply, they pass some information, they call a method and boom, it's just happening. Now what does Encapsulation give us? Well the main thing is, it supports the concept of code reuse. Now we've all kind of been led to believe that operating in an Object Orientated Programming Environment is going to let us continue to build objects and build objects and build objects, until the day comes that the boss comes in and says hey, this is the application I want and we say, no problem. I'll take this object that I've already built and this object that I've already built and this object that the other people built in the other department and this object that this lady over here in Human Resources built and I'm just going to tie them together with a few lines of code and boom it's done. It's all working, I'm out of here, gone by 1 O'clock. Playing golf by 1.30. Okay. I've never actually seen that happen but it is possible that we can reuse a lot of different code. The Oven Class could easily be used in somebody else's project. Why? It's written in the .NET Framework. As long as they're writing in any .NET compliant language, they could take the Oven Class that we wrote. They could place it into their particular application of their project, Set References to it and boom they should be able to use it. And that user does not need to understand how the oven works. They don't need to understand on the inside, how our Oven Class is working. They just need to know how to call the functionality to make it happen. That's what Encapsulation is and that's the way we should be constructing our classes.

(Object Oriented Programming / Understanding Inheritance)

In this video, we're going to continue our kind of a worldwind tour through Object Orientated Programming and Development and talk about a term called Inheritance. Now this is one you're going to see and you're going to use quite a bit. This will be a fundamental part of C# development, C# functionality and C Sharp program design. Inheritance is a functionality that happens between our classes that is a direct result of that Classification or breaking functionality into individual classes. Object Inheritance is very, very similar to Human Inheritance. That is the ability of one class to access the functionality of another class and that would be the Parent Class. Think about this, if you have children or if you've ever been a child, let's do it that way okay? If you've ever been a child and if you're listening to me, then you have, then your parent's had certain functionalities that you automatically had access to. Now maybe you didn't know how to drive when you were 8 years old but you inherited kind of that ability to travel around from your parent's. Right. If they had a car and you were their child, it meant you had a car. So if they could drive 60 miles an hour down the highway, to grandma's house. That meant you could travel 60 miles an hour down the highway to grandma's house. The same thing can happen with our classes and our objects, we can create that kind of relationship. So one class can be the Parent Class, another class can be the Child Class and the Child Class retains it's own functionality and it retains the functionality of the Parent. But it also has the ability to extend or even replace some of the functionalities in the Parent Class. So we can take all the characteristics, the methods, the properties, the functionalities of the Parent Class, automatically use those in the Child Class without having to write any code, and then, we can have the Child Class do a little bit extra. Alright. Now Inheritance creates a special relationship between objects called Specialization. Now let's look at an example of Specialization through Inheritance using the classes in our Veterinary Application that we've talked about in a number of videos in the course so far. You remember our Animal Class? Well think about, if we're writing an application about a veterinary office, they're going to deal with a lot of animals. Now an animal, whether it's a dog, a cat, a snake, a lizard, whatever, it's going to have certain characteristics, a name, a color, an owner. It's going to have a lot of different characteristics right, that's going to be general to every animal. So we can put all those in an Animal Class. Then we can create a Dog Class, because there are certain things about dogs that are unique to dogs. They're common to animals but there not unique to cats or spiders or those kinds of things. So we can have the Dog Class inherit from the Animal Class. So any properties, methods the Animal Class has, we can access through the Dog Class, and then we can add special things, for example, dogs have to be treated for Heartworms. Well not all animals do, so we inherit from Animal, we get the name, the weight, the color, the owner's name. Then in the Dog Class we add a method for dealing with Heartworms. Alright. Then we have a Collie Class, which is a Specialization of the Dog Class, now notice the Collie Class is a Specialization of the Dog Class. The Dog Class is a Specialization of the Animal Class. Now the Collie Class has access to all the functionalities and properties of the Animal Class and the Dog Class, but it can also add functionalities that are unique to the Collie. The Collie maybe susceptible to certain types of diseases and so it needs certain treatments that other dogs may not need, and so we can specialize that way. Notice we can specialize on the feline side, on the cats; cats are obviously animals so we can inherit from Animal, they have names, they have colors, they have weights. But certain animals, for example Persians long haired cats, have difficulties with respiratory issues because of bathing themselves and hair balls and all that. And so you can see what's happening here with Specialization. Since we have classified functionalities as far as animal and then more specifically dogs and then more specifically collie, we can use Inheritance to give us a neat interlocking, if you will, of these different functionalities. Getting more specific as we move down the chain of our classes here, and this is called an Object Model. And so if you've ever heard the term, Object Model, this is what they're talking about. Now keep in mind this is exactly how the .NET Framework is built. Okay. You have certain Base Classes that we then, for example, there's a Controls Class and then that Controls Class is inherited by a Button Class okay? So a button is a control but has different functionalities, different aspects say from a List Box or a Drop-down Box and so we can specialize that way. That happens in the .NET Framework and when you start to think about coding in C# in an Object Orientated Environment, you want to do the same thing with your classes there. Build the most general classes, let everybody inherit from that and then specialize on down and what you're doing here, keep in mind, you're encapsulating, you're abstracting things away from everyone. You're proving for Inheritance here and then we'll talk about Overloading which is another aspect of Inheritance in a separate video. But as long as you can get your head around what Inheritance is, when we actually implement it in code later, the light will come on. And you'll say ah, that, that clown talked about that earlier in the course, I'm right there with him. So that's a quick look at Inheritance.

(Object Oriented Programming / Understanding Overloading)

Overloading is a functionality that you can utilize in a number of different places in C#, especially as an outgrowth of the Object Orientated aspects of C Sharp. And you really need to understand this because it's going to come into play in a lot of different ways that you're going to see later on when we get into some code examples. Now Inheritance can produce a lot of options in the number of available methods in our classes or functions in our classes that become methods in our objects when we instantiate them. Think about this in this little model we were using before in some different videos. We had the Animal Class and the dog inherited from animal and collie inherited from dog, then we did the same similar thing on the cat side. We could have, say if we had, 10 methods in each one of these classes that's 50 methods. Well think about this, there maybe instances where for example a Feed Method, we could have a generic method for feeding the animal but then feeding a dog could be a little different from feeding just generic animal. So we want to a Feed Method so that when we need to call it, it would only apply to the dog and then a Feed Method that when we call it would only apply to the collie, same thing for cats. Well how could we pull that off? Well the way we do that is to simply name the methods the same thing and I know what you're thinking, now wait a minute how can you distinguish them? I'll show you in just a second. So we can have Method Names that are duplicated in Child Classes and each method may be required in different locations in our program based on the functionality that is needed. And so while we have Multiple Methods with the same name, we may want to call, say the method from the Collie Class instead of the method from the Dog Class or the method from the Dog Class instead of the Animal or the Collie Class. Alright. So we already know methods can share names but here's the catch, only if they have different Signatures. So let's go look at what we're talking about here. Overloading is where we use different Signatures for the same Method Name. Now a Signature is the name of the method, this is a Do This Method and the number of parameters that get passed in and the type of parameters. Okay. So this particular method called Do This is a public and it's going to return an integer, but notice it's Do This and it takes an integer of A. Now I can have another method called Do This, notice the same name as this but this one takes 2 parameters, one's an integer A, one's an integer B. So here's the deal, I'm overloading the Do This Function here or the Do This Method. If I call Do This and pass it 1 single number it's going to run this one. Now obviously there would be some code under here. Right. If I call Do This and pass it 2 integers, it's going to run this one. Now this one could be in one Class, this one could be another Class somewhere else and it would automatically know to go pick that one up and run it. This gives us the ability to produce a lot of different options and the number of available methods and yet make it very, very easy to pinpoint exactly what functionality we want to run, just based on what we pass to the method. And we'll look at some, a little bit more complicated examples of this a little bit later on in the course, but that is functionally what Overloading is. It is creating Multiple Methods and Multiple Functions with the same name, in this instance, its Do This. But the Signature is different in that they take different numbers of and different Data Types of parameters that are passed in. And of course, also known as arguments. Alright. So that's the basics on Overloading.

(C# Variables / C# Variables)

Now let's turn our attention to C# Variables. Now a variable is a little portion of memory, think of it as a small box where you're going to store things and variables are fundamental to any programming language that you use. But the more you can understand about them in C#, the more you can do with the language and the easier it is to understand what's going on. So if our variable is a small box that we're going to use to store information, then you can think about how you store things in your attic or in your garage. Three questions that need to be answered when we get ready to store something. First of all, what kind of data do I intend to store in the box? How small is small? Now I said it's a small box used to store information, well I don't want to store a giant box with just one pair of shoes in it and I don't want to store a box that's too small where the shoes are sticking out. And then I have to make a decision, where exactly do I want to store the box, so that I can manage it efficiently? If it's a box I'm going to be getting down a lot and getting into and changing the contents, I need to put it in, you know, a nice convenient place. If it's a box, then I'm going to store this for extended periods of time and never go back into the box, I might want to put it high on a shelf or behind something else, that sort of thing. Now declaring a variable in C# answers these questions, or even if the way I declare the variable doesn't answer the questions, it allows C# to make some assumptions based on the way I declared the variable. And these assumptions are things such as, where to store the variable, how much memory to use to store the variable, how to destroy the variable when you're done with it and those sorts of things. Now let's look at a simple example here. I'm creating a Variable called A, it's going to be a Data Type of Integer and I'm going to initialize it with a value of 546. So that's the way you would see it in the code with a semicolon on the end, you'll see lots of these throughout the course. Now this tells C#, first of all, what type of box to use, an Integer Box because I'm going to store Integers in there. It also tells C# what value to store in the box, 546, it tells C# the name of the box, A, it tells C# how much memory to use. Now you're saying wait a minute, Mark, you didn't tell it how much memory to use. Well I did indirectly, I told it to use the Integer or the Int Data Type and that always uses 4 bytes or 16 bits of memory. Now just for your information, when I use an Integer, an Int Type that utilizes 16 bits, then that means in English, I can store values in a range. Integers meaning whole numbers, no decimals from roughly negative 2 billion all the way up through positive 2 billion or roughly 4 billion, a little over 4 billion in values. C Sharp then assumes where the box should be stored based on the way I declared it and it's going to store it on the heap, which is very, a little bit more involved, a little bit more complex or as a Value Type. And this is the same as saying I'm going to put it somewhere where I can change it really quickly, very efficiently and throw it away when I'm done it very efficiently and we'll talk more about that later on in the course. Now here's a question to ask yourself, do I really need to use enough memory to store up to positive 2 billion and down to negative 2 billion? Think about this number, if I have 1.5 billion inches, then I can put them end to end and reach around the earth at the Equator. Now think about this, if I'm going to use this variable to store, say the number of the employees we have at our company, well right now we have 546. Do you think we'll ever have 2 billion employees? Okay, so that's probably not efficient use of that particular Data Type for that particular variable. Now we can provide C# with a little more accurate information based on the data that we know we're going to be storing in that variable and so we can instead of calling it an Integer, A equals 546. We can choose Short and we'll talk about other Data Types in separate videos but for now, just trust me here. We'll redo our Variable Declaration and we'll say Short A equals 546. That Short Data Type uses 2 bytes of memory and that means we can store from negative 32,768 through positive 32,767. Now you're going to be okay here with this variable until you try to store 32,768 or negative 32,769. Once you do that, you're going to get an error from C# saying, whoa wait a minute, this overflow on your box is too full, I can't store everything in there. So that's the low down on C# Variables, we will build on this information, as we move through the course.

(C# Variables / Data Types)

Anytime you create a Variable especially in C#, you're going to be concerned with a Data Type, you have to tell C# what Data Type to use when you create the variable. Now let's take this back to the attic picture, why can't you find things in your attic? Well if your attic is anything like mine, you may have a box in the attic, that's you know, in a big pile with other boxes fallen over on it and all kind of stuff. And that box that you see in the stack there is labeled Shoes and so I dig that box out and guess what I find? I find that it contains shoes, old photos, clothes, some computer disks that I have no idea what's on them, so my box says Shoes, but it's got a lot more stuff in there. So if I came up here looking for some old clothes, I would probably not look in that box because it says it has Shoes, then I realize that the box is half empty. So not only am I not storing the proper stuff in the box that it's labeled as, I'm also wasting space with that box. The bottom line is, is I have myself a Data Type issue with this particular container. So Data Types establish storage rules and limits on the variables we create. Now what's neat about C# is it's going to force us to only put the Data Types into a variable that we told it we want to store there and it won't let us put too much in there. Now it will let us waste space and I talked about that a little bit in the video entitled C# Variables. But the bottom line is, Data Types determine what kind of data can be stored in the variable. Now there's a little bit of playing that we can do with this as far as putting different flavors of certain Data Types in variables and we'll talk about that a little bit later on and of course, the Data Type determines how much data can be stored in the variable. Now C# provides three basic types that you can use for data. First is the Value Type, then there's the Reference Type and then there's the Pointer Type, I don't want to go too far into this right now but I do want to show you a great resource. Okay. Now this will help you understand, all the types that we can use for storing things in C# are called Predefined Types. They were already written by Microsoft, they're already there in the NET Framework, we just simply use them. Those are defined or basically divided into two areas, Value Types and Reference Types. Now Reference Types are our Objects or our Strings, now Reference Types don't store the data directly in our variable. The variable is a pointer to where our data is actually stored okay? Then on our Value Types, you'll notice we have Structs and Enumerations and Enumerations are where we want to store values like 0,1, 2, 3,4,5,6 but we want to give them English names like Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday. Now the simple types are what we use the most and notice that's subdivided into Bool for Boolean, true or false, 0 or 1 and then the Numeric Types that we use most of all and that's split into two sections there. There's Integral Types and then there's Decimal and Floating Point Types. Now these, are numbers that have decimals or fractional values and then the Integral Types are what we usually use the most. Notice sbyte, byte, short, US, ushort, integer, uinteger, long, ulong and character. Now what does this S mean? S means that's signed by, Signed, S I G N ED. That means that we can store positive and negative values. So notice, we have a short and ushort. Now short means we can store negative and positive, ushort is Unsigned, which means only positive, same thing for integer, long and character. So this little chart here can really help you keep straight the relationship with the different Data Types and I would strongly encourage you to look at this chart, maybe even refer back to it from time to time. This will really help you kind of keep in mind where your Data Types are. But Data Types are very important and the more you can master these, the easier it is, especially when we start talking about, what happens when we need to now change a variable's Data Type and store different types of data in there on purpose? And that's going to be Casting and we'll talk about that in a separate video. But it's really going to help if you can understand, kind of the relationship where you don't have to memorize this but just understand the different types that are out there and kind of how they relate to each other.

(C# Variables / Built-in Value Types)

Now let's take a little closer look at the Built-in Value Types and what I want to do here is just kind of list them out for you and show you the keyword, the C Sharp Keyword that you will use when you're actually programming with these. The type is the actual underlying .NET Type that's used and the keyword that we're seeing for C# is really just an alias. And then we'll see the size of it, how many bits or bytes it takes in memory to store data in that Value Type and then the actual range and numbers as humans see them that we can store in those bits. So let's start off with the sbyte. The sbyte is actually a, a type for the .NET Framework of System.Sbyte. It uses 8 bits to store the data and this a Signed, notice the S. So it can store from negative 128 to positive 127. The byte is a System.Byte type, it uses 8 bits and goes from 0 to 255. The Short Data Type is actually, according to the .NET Framework, a System.Int16. It uses 16 bits and you can store from negative 32,768 through positive 32,767. The ushort and notice there's a System.Int16 and this an unsigned short and you can go from 0 to 65,535. The Integer, which is kind of what we all use just without even thinking about it most of the time, is System.Int32 type. It uses 32 bits and it goes from just over negative 2 billion to just over positive 2 billion and this is why it's so popular, because most of the numbers that most of us deal with on a daily basis, don't generally exceed 2 billion unless you're looking at bonus's and that sort of thing. Right. So anyway a UN, an unsigned integer is a System.Unsignedinteger 32. UInt32. It uses 32 bits and notice since we don't have to go negative to positive 2 billion, you can basically double that almost and you, we can go form 0 to 4 billion, just over 4 billion, 4 billion and some change as they say on television. And in this case, the change is 294 million plus. The Long Type is a System.Int64. It uses 64 bits and I'm not even going to read this number to you, just notice this is a Bill Gates, Steve Jobs kind of a number. A little tribute to Steve Jobs there and notice this is a huge number. So if you want to store really, really large numbers, you're keeping up, you know, some sort of accounting application for a large corporation, this is probably what you want to use right here. So anyway Long will give you this and Unsigned Long will take you from 0 to 18 quazgabillion or whatever this is, don't know what this is but if you need that size number, there's your Data Type. Then we got just a couple more of the double. Notice this is the System.Double at 64 bits, that's an exponential number. I'll refer you back to your seventh grade, eighth grade math book to figure out what that means in English or you can use a scientific calculator. But trust me, plenty of room here and notice on these doubles, decimals floats, what we're really looking for, is not the actual size of the total size of the number but the levels of accuracy we can get in the decimal places and they certainly gives us that. The decimal is the System. Decimal, takes 128 bits and notice I didn't even have room to put commas in here. Okay. But again, we're not worried about using a number this big, we're worried about using these as decimal places, especially for scientific applications and very high tolerancy situations. So those are the Value Types that are built-in. Again most of the time, we just use Int and never think about it. This will give you a chance to kind of pay attention to these, think about which one am I using? And the real thing you want to watch is, the range that you really need to store a number in and then think about how many bits of memory. Now we live in an age where memory is just everywhere and its gigabytes, and terabytes, but still the more your computer has to manage, you are beginning to slow it down. You're not going to see a significant performance hit on just one or two variables. But the more you start consuming more and more memory in more and more places; it's going to start to effect your application at some point. Then when you put a bunch of simultaneous users on it, you're now starting to multiply it exponentially. So anyway, just keep this in mind, use this as a reference and really educate yourself as to the value types and the choices that you have as you program and utilize memory.

(C# Variables / New Dynamic Type)

One of the new additions to C# 2010 that you're going to hear a lot about is this idea of a New Dynamic Type, that we can use when we create variables. Now the Dynamic Type when you think about that term, Dynamic and you want to think, well wait a minute, what do you mean? What is Dynamic? And you'll hear them talk about Dynamic Programming, what is that? Well, is it this awesome, amazing programming at a whole new level? Well, not quite. What it is, for a simple definition, is that the program simply has more abilities to change types at runtime, not limited just to design time. Now what's kind of interesting is, the Dynamic Type is new to C# but it is certainly not new to programming in general. So what are the pros of the New Dynamic Type? Why do we have it in C#? It gives us a lot of options and flexibility and the big con is there's lots of options and flexibility. There's so many options and so many flexibilities that you can actually lose control of this thing and introduce really hard to find errors into your code if you're not careful. Now the Dynamic Type takes the Var Type or the Variable Type to a whole new level. If I create a Variable in C# and I instead of calling it an Integer, an Int or a long or a short. If I call it a Var, if I create it of the type Var, that means the Compiler will infer or it will figure out and set the Variables Types based on whatever input I'm putting on that variable at design time. So if I say Var A equals 3, then it's going to figure out, hey wait a minute, dude's trying to save integers here, so I'll make the type Integer. So that's all it's doing, it's choosing the Data Type for us. These have been everywhere, a lot of the scripting languages, early Visual Basic, all had these things. Okay. Now Dynamic, the Dynamic Type takes it a step further, it works the very same as the Var Type but you can also change the type dynamically at runtime. Now as Yogi Bear used to say, well this is deja vu, all over again. If you don't know who Yogi Bear is, if you're a little too young for that, go out there and look him up on Google, look at some of his quotes, they're hilarious. Okay. So anyway, what we're looking at is deja vu all over again. Remember scripting language programming, you could set those Data Types up, you didn't even have to mention a type, it would figure it out and it would even change them as your program runs which is exactly what this New Dynamic Type will do. But that sounds dangerous doesn't it? Why do we want to do this? Simple answer, COM, the old Component Object Model expects the languages to use Dynamic Types. Well why do we care about COM? Because we left COM a long time ago when we went to the NET Framework. Well here's why, C# doesn't support Dynamic Types and there was a lot of crowing and everything for a few years there from Microsoft wheels, talking about C# would never support Dynamic Types. Well guess what? Microsoft Office programming requires interaction with, guess what? COM because Microsoft Office is still underlying in there, got a lot of COM functionality in it. Okay. So C# programmers have found Office programming to be somewhat difficult. Dynamic Types is the fix to that difficulty with Office programming and so in trying to basically sew up all these neat functionalities and inter operations with Office, it's been somewhat problematic from a platform standpoint. Because your C# people couldn't do some of the things that would have made it a lot easier in the Office programming environment and so now we have the fix. And so that's the real reason you're seeing this New Dynamic Type and just a little intro into what it actually does for you in C# 2010.

(C# Variables / Dynamic Type Example)

Now let's take a look at a very simple example of using the Dynamic Data Type. This New Dynamic Type is something, you, for a long time, never though you'd see in C# and I want you to just notice here that I've just got a Basic Windows Form Application. Doesn't matter, this is not going to be in your Work Files Folder, it's just too simple. You need to type this, run the code yourself and get your head around what's going on here, this is just pretty cool. I'm going to create a Variable of an Integer Type called A and set it equal to 3, then I'm going to set A equal to Hello. Sounded simple didn't it? Then I am going to and you'll notice I've already got a squiggly line, it's going to say hey wait a minute, whoa, whoa, whoa, dude. A is an Integer. Now it looks like you're trying to stick a String into an Integer Data Type and then what I'm going to do is set the Text Box 1's Text Property equal to A. Well, obviously this is not going to run. If, even if I try to compile this and run it, it's saying wait a minute dude, you got build errors, I can't do this. First of all, you're trying to convert, you declared an Integer Data Type, now you're trying to stick String into it and this is just not going to work, then you're trying to print it out down here. It has to be a String to go in the Text Box but it's still, it's still an Integer because I'm not even playing right here. Okay. So to use these, I would have to do a Cast here. We'll talk about that in a separate video, there's some other things you're going to have to do. But notice what happens when I just make this a Dynamic Data Type. Notice that you're going to see a lot of the problems simply go away because when I run this now and I click the button, notice, Hello shows up there. What has happened? The Dynamic Data Type, notice the description right there, it represents an object whose operations will resolve at runtime. It looks like what is the Type A here? Well when I put the 3 into it, it was an Integer. But then when I dropped a String into it, it changed it to a String and it reported it out to the Text Box, pretty straightforward. Now if I go the other way, let me just show you this. In case you try this, there's a little aspect here that I don't want to confuse you about. If I set it back to 3 here and run it. Okay. And click this, it's going to sit here for a minute and it's going to say, whoa, wait a minute. Now I have a problem, I can't convert the type Int to String. That's not the problem of the Dynamic Type; it's the problem of the Text Box. The Text Box, let me get out of my Debug here, the Text Box has to see String Values. And so we're going to call the Two String Method on the A because at this point, the A is an Integer and a Text Box can't deal with an Integer, it has to be a converted to a String. So when we run this, you'll see that it now pops up, 3 is in there as a String not as an Integer. Okay. But that simple example shows you what's going on with the Dynamic Type. Now this is huge because when you're trying to use C#, in different instances, where you're trying to work say, in the COM environment, where those old Variant Data Types were out there, it was very hard to maintain these things and keep things, all the plates spinning and balanced. The Dynamic Data Type let's us mimic that and let's us work with those Data Types out there. And so anyway, that's a very simple example there of just exactly what the Dynamic Data Type brings to the table for the C# Developer.

(C# Variables / C# Operators)

In this short video, I want to go over just a few of the C# Operators. Now at first, you're going to think I've lost my mind, that you know what these things mean but it is surprising how many people get confused by these and don't really understand what's going on. And ultimately, could create havoc with their programs or chase down some bizarre calculations and only later figure out that they didn't really understand what was going on. Now these first few shouldn't be a problem. The Asterisk is what we use for multiplication in C Sharp, just like a lot of different languages, it'll multiply values together. The Forward Slash is the Division Operator. The Percent Sign is the modulus and this divides one value by another but it only returns the remainder, it doesn't return the result of the division, only the remainder if there is one. And then the Plus Sign of course, adds values together. Subtraction subtracts one value from another. You've been knowing that probably since before you could read, so we'll move on. Now the Bitwise and Logical Operators are where a lot of people can get confused. The Ampersand Sign is a Logical and of two values the straight up and down pipe, if you're going to call it that is a Logical or of two values. So anytime we're working in a Bitwise environment or a Logical, that's what you're going to use. The little house top or the carrot is a Logical XOR. Now XOR you'll have to jump into the binary world to understand that, if you don't know what it is, I would strongly suggest you go out and Google it, take a look at it. But the bottom line is, an XOR function will always produce a 1, it's only going to deal in 0's and 1's. It will produce a 1 if either of it's inputs is 1 and it will produce a 0 output if both of it's inputs are 0 or 1. So if there's a match, two 0's, you'll get a 0, if there's two inputs or one, you'll get a 0, anything else, will return a 1. So if that's confusing it's only because it's confusing. So just go out, look that up but if you're working in the binary world, there's a lot of power out there, but you'll just have to make sure you understand what these are. The little squiggly that a lot people call that, is the Logical Not and you will see that from time to time. The two left brackets, the two left angled brackets, right left, Shift Operator and in binary terms if you're familiar with binary, that is a promotion, we just made it larger. Right Shift Operator and Bitwise and Logical is a demotion, we just basically made the numbers smaller. Now the Assignment Operators are where most of us live, most of the time and we all know these first few equal is a Logical and of two values. Two equals is where we set something equal to something logically and then you have the Plus Equal, the Minus Equal, the Multiplication Equal and Division Equal. Each one of these we'll, we'll start with the Plus Equal. It will take whatever value is there, add whatever is on the right side and then it will just keep tacking that on. This is a shortcut way of saying for example, total equals total plus A. So all we would do is total plus equals A and it will take care of it for us. Most people are already aware of this, if not, just go out and look at that in the documentation. Now the Modulus Equal takes the remainder and adds it to the value that's there, this is one you don't see every so often but there's a lot of power here, keep this one in mind. It's amazing how many times you'll see developers write, you know, a pretty interesting little script or a little routine to call, a little method to call to do just this, when all they have to do is call Modulo equal. Then there's the Ampersand Equal, this is a Logical and and assigns a value. The Logical or and assigns a value, that's the pipe and the equal. The Carrot and the equal is a Logical XOR and assigns a value, and then you have the angle Brackets Left, this is a left shift. This is a binary operation, left shift and assigns a value and then there's the right shift and assigns a value. These are the ones that I see really kind of surprise people when you see them. You're normally going to use these in, in more, hate to say the word sophisticated, but more complex scientific type applications. But one thing that I'm noticing, as the Internet and information technology continues to explode and expand, it's amazing how the complexity of what our programs do and how they compare data and how they search data continues to grow as well. So anyway, I just wanted to give you a little heads up on some of these Operators that tend to trip people up and if you know these, no big deal. But if you don't know them, any of them, that when you first saw them, you're like whoa, what is that? What exactly does that do? I've given you a little explanation here. I would strongly suggest you go out into the C# Reference and, or just go to Google and type it in and look at what it does but make sure when you do that, you're looking at what it does in C# because they can be somewhat different from one language to another occasionally. So anyway, that's the C Sharp Operators that I wanted you to take a look at.

(C# Variables / Casting pt. 1)

Casting is a functionality that you need to be familiar with in C#. This is a functionality you're going to find in a lot of programming languages and in this video which is Part 1 of Casting, I'm going to kind of take you through the textual explanation of Casting and why we need it. What we use it for and then in the second part, we'll actually jump out into the coding environment and do a little example to show you. So first off, let's talk about the issue here and the bottom line is, is that humans and then C# see data a little bit differently. As a human being, we simply see values for example, 546. Is that a 5, a 4 and a 6? Yes but we see it and comprehend it as a combination of those numbers as 546, we know that from the context that the data is being used in and then of course there's the concept of negative numbers. We just put a negative sign in front and it changes the concept. And then there's the large numbers, 5,687,892,344, the commas tell us that this data is related and we just keep pushing it out into larger subdivisions of thousands, millions, billions. Now C# sees data that requires different memory requirements. Now when you looked at 546 or negative 546 or even the large number there, you didn't even think in terms of, wait a minute, how much brain space is it going to take to remember this? That's just kind of hard wired into you and it's happening, if you think about that too much, it'll make your head hurt. But anyway, C# has to deal with these things. C# simply sees an Integer that needs to be stored in memory and then C# also then is going to do one extra piece of service for us, and that is, it is going to expect to see a Data Type that's going to be stored in that memory. Now when we create this type, not only do we need to tell C# how much memory that it needs to store, be used to store this particular data, it's also when we give it that type, it's going to help us protect ourselves from ourselves. By restricting further uses of that variable to that same or very similar types. Okay. A good example, if we create a variable using the Variable or the VAR or the VAR Data Type. C# will look at 546 for example and say, okay that looks like an Integer, so I'll just default this out to Integer and then they can only store Integers or other types. Now when it told it to make it an Integer, what it's really doing is giving itself the capacity to store from roughly negative 2 billion to positive 2 billion in that memory space. Now let's go back and take a look a little closer at what goes on in the background with C#. When we decide that we want to store 546, we just simply see that positive number 546. C# sees a positive number that needs to be stored in memory and it also knows really quickly that it needs more than one byte of memory for storage. Now how do I get that? If you convert 546 to a binary number, you can use your scientific calculator in your operating system to do this, notice this is what 546 looks like in binary. Alright. Well notice if you count the 0's and 1's here 1, 2, 3,4,5,6,7,8,9,10, it takes 10 bits to do the trick. Now there are 8 bits in one byte. So one byte's not going to get it, so we have to go to two bytes or 16 bits to be able to store 546. Alright. Now the Short Data Type provides two bytes of storage and it can store from negative 32,768 through positive 32,767. Now let me just kind of give you a little, another piece of information here. The Int or the Integer uses 4 bytes or 32 bits and that's now we can get from negative 2 billion to positive 2 billion but notice what's happening to C# here. What if I later decide to change that Data Type. Alright. If I'm going to store a larger number than what I already have, then I'm going to need more bits. I'm going to need either a larger box to store more numbers or smaller box to store less numbers. So what's really got to happen here is I've got to take data from one size box and put it into another size box and Casting is the way I do that. Casting will tell C#, okay look, I know the dude created the, the Short Type to store 546 but now he needs to store, you know, a billion, 300,000. So that's not going to fit into a Short, so we're going to have to Cast this to a Long or to an Integer. So if I put a smaller type into a larger one, no problem because if I take a pair of shoes out of a shoebox and put them into a refrigerator box, no problem. The shoes are going to fit perfectly, not going to have any issues with losing a shoe. But if I try to take a bunch of shoes, say 100 pairs of shoes out of a refrigerator box and try to put them into a smaller box, say a single shoe box, I now have a possible big problem. Actually I do have a big problem and that is, some of these shoes simply are not going to fit into that smaller box. Now this is why in some instances, when we move data from one Data Type to another, C# doesn't seem to care. It actually does care, but it says, I'll tell you what, you're taking a pair of shoes out of a shoebox, putting them in a refrigerator box, I can handle that, no data's going anywhere, no problem. However if I'm trying to move from the refrigerator box down to the shoebox. Okay. Say from, from an Integer down to a Short, C#'s going to say, whoa wait a minute dude, you need to think this through because you could possibly be trying to put too many shoes into too smaller box. Some of them aren't going to fit and as a result, you're going to lose some shoes here and that is why we want to use Casting. Casting is what we use, it's the way we tell C#, look I know I'm going from a larger box to a smaller one, but trust me on this, I know what I'm doing and you know what usually happens when you say things like that. But anyway we won't get into that here. So that's what Casting is. Casting is the ability to inform C#, yes I want to change the Data Type and yes I understand the ramifications. Now we'll look at some code in a separate video and see how to actually implement this.

(C# Variables / Casting pt. 2)

In this video, I want to continue in Part Two of Casting, what we talked about in Part One. But here I want to jump out into C# 2010 Express and actually write some code and show you this in action and show you the danger of Casting and kind of the ability that it gives you. Well, first of all, let's start off with a variable that's a Short Type. Now notice I've got a little key up here, a Short will store from negative 32,768 to positive 32,767. An Integer stores from negative 2 billion and 147 million and plus whatever up to 2 billion 147,43,767. Okay. Never mind the typo. So what I want to do is if I create an Integer that's a Short and it's a Short and I call it S and I set it equal to 15,000. Pretty straightforward. Right. Then if I create an Integer called I and I set it to equal to S, oh scrolled up, no big deal. Okay. Now what I'm going to do is, out here on my form and I'm not going to include this in the Work Files, you can just build this and play with it. I've got a form with a Text Box and a button. What I'm going to do is, I am going to set my Text Box 1.Text field equal to the, the value of I right here, my Integer. Okay. Now let me show you something, when I try to run this, the Text Box expects it to be a String. So I'm going to take the value of I that will be an Integer and I'm just going to basically convert that to a String. Now this is not doing a Cast, it really is making a copy of it and basically more or less arraying it out into characters. Don't worry about that right now but this will show you the actual number value that it ended up with. So I started with 15,000 in the Short and then I copied the Short into an Integer, no big deal, 15,000 came across. Alright. Well, let's switch this up, what if the Integer I is worth 15,000. Okay. And the Short S is now set equal to I. Okay. And I want to make sure that I report out my S, out to the Text Box. Notice that it's going to say, I can't implicitly convert a Type Integer to a Short. Okay. Because you could be losing something. Well we know that a Short will store up to negative 32,768 and we're only trying to put 15,000 in it, this shouldn't be a problem. So what I'll do is, I'll just Cast it, so I'll tell it, okay look come on now, I know what it is okay and I am just going to explicitly take care of this. Okay. And so I am going to, go the other way here; I am going to explicitly convert I into a Short because that's what it would have done before. Notice that when I didn't have this Short there, what it was actually doing, it was actually saying okay, I'll tell you what, I'm going to take I, they want to put it in a Short. So I'm going to have to convert I to a Short for it to work and then it's going to think, well wait a minute, whoa, whoa, whoa. Going from a possible of 2 billion down to storing only 32,000, dude could be in trouble here and so I am going to put the Short back in and notice that when I run this, it will report out the 15,000. So we basically did a Down Cast there if you want to think of it that way. We went from a longer Data Type to a shorter one, but we were okay, because we knew we weren't violating any limits here and so we explicitly said, look dude, quit worrying about it. I got it. Okay. Well let's go another way here. What if we do 32,000, 32,767? Notice that's the limit of a Short, and so I can run that and everything's okay. 32,767, but now let's exceed that. Now I've told it that I know what I'm doing, shut up, don't give me any more squigglies, don't tell me I can't do it. So I want you to just kind of be quiet, convert this Integer into a Short. The Integer is actually larger than a Short can handle and I want you to see what happens, we're passing in 32,768. It shows up as negative 32,768. And what's happened? Well I won't go into a lot of details here, but suffice it to say, that the Short doesn't have enough room in binary to store that large of a number and so it starts to shift things around and it truncates things. And let's just go ahead and make this 328,784, that's the number I started with, I explicitly shoved it into a Short. And there's a saying out there that I'm not going to say in the video about putting 5 pounds of a certain substance or putting 10 pounds of a certain substance into a 5 pound bag and that's exactly what we're doing here. When I run this, notice 328,784, it gets put into a variable that will only go as high as 32,767, it really gets confused. Right. And it comes back to 1104. So this is what you want to be careful with, with Casting. Now if you go look at a chart of your Data Types and I actually did a video on the built-in Value Types and if you go back look at that, you will see for example that a Byte's smaller than a Short. A Short's smaller than an Integer; an Integer's smaller than a Long. And so anytime we're going backwards from those things, it's going to squawk at us and it's going to say, hey wait a minute here, wait a minute, you're trying to go from a large one down to a smaller one and I can't let you do that. If you want to tell me that you want to do it, I'll let it go. Okay. So that's the simplicity of Casting and you saw the syntax of how I did it. If I is an Integer and I want it to be a Short, I just say, I put the Data Type that I want it to be, in front of it, in parenthesis and the Compiler says okay, whatever you say that's what I'm going to do with it. So anyway, that's Casting.

(C# Variables / Strings)

In this video, I want to introduce the concept of Strings and how Strings are managed within C# and it's going to be a short video because I want to kind of lay out the basics here. And then in a closely related video that's going to be entitled Strings Example, I'll actually jump out into the Visual Studio Express Tool and we'll play with Strings just a little bit to show you what I'm talking about here. Now we got an issue, when I see a group of characters, I see them as words that's thse way humans see characters grouped together. We see them as words and we simply know that if there's a space before and after, then all those other letters together are words and we should try to use our phrenetic skills to turn those into a word that has a meaning. Now computers see characters grouped together as simply a string of characters grouped together, and spaces may or may not mean things. And so what we're going to have to do with Strings when we're dealing with computers, is figure out some way to implement these things in such a way that they can be stored together in memory and represented the correct way. Now many languages simply store Strings as just an array of characters, and if you know what I'm talking about with an array, then you're, you're right here with me. If you don't know what an array is, it is simply a variable that can store more than one value that uses an Index Number to keep up with them. So a word that has 5 letters would be a variable that is an array that stores 5 values, 0 through 4 and each letter would be 0 then 1 then 2 then 3 then 4, that would be 5. Now .NET stores Strings a little it differently, it simply puts whatever String we give it into or, or converts into an Unchangeable Object and I know what you're thinking, wait a minute, didn't you just say Unchangeable? Because I do Strings all the time and, and first I make it Mark and then I make that same String, Mark Long. So then I create a Variable called String Name and I set it equal to Mark, then I come back and set name equal to Mark Long, then I come back and set it to equal to Mark T Long. Well, it looks like its being changed but in reality that String is actually unchangeable. When you change a String, .NET simply deletes the old String and creates a new one with the same variable name and this is fundamental to understanding. I'm going to do a demo, a little example in a separate video, but I didn't want to try to put the two together because I don't want to rush the example for you. Okay. So just understand that Strings are handled in a very interesting way in .NET and it's actually very cool. Because once I create a word or I create some data as a String, automatically since it's an object in .NET can deduce the length of it very easily. And I can do all kinds of things with it and I'll show you that in the example. So look for the video, Strings Example and we'll take a little closer look at Strings in C#.

(C# Variables / Strings Example)

Now let's take a look at an example of a String out in the development environment and you'll notice I've opened Visual C# 2010 Express. And I'm just going to create a New Project here and I will call this the String Example and you'll see this out there in the Work Files folder, although it's not very complex, you could probably just as easily follow along with me. Now notice I've pinned my Toolbox, yours could look like that, just mouse over Toolbox, pin it up out there and I'm going to double-click and put a button on my form. I'll put it kind of in the middle. Then I will go out and grab a Text Box, double-click that, put it just above the button, try to center it up just a little bit for neatness sake, then I'm going to drop a few labels out there. Okay. I'll put, I'll just double-click here, I'm going to put 4 out there. So I'll drag the fourth one down here, we'll put it equal to the label, I'll drag three down here, line it up with the button, two line it up with the button and then one and don't you love those handles? The way they drag down. Okay, now what I've got here is I'm going to manipulate a String and I'll input the String into the Text Box and when I click the Button, I'll show the various manipulations in the labels. So double-click the button to get to the Button Click Handler and the first thing we want to do is, what we always do. We want to create a String called A and we're going to load that String or set it equal to whatever is input into the Text Box 1, into the Text Field okay or the text property if you will. And then just for fun, I'll show you a little, what I'm going to do about this later. I'm going to set up another String called B and I'm just going to go ahead and set it, hard wire it to the letters A R. Okay. Now let's start off with a simple thing that we always do with Strings and that is we're going to set Label1.Text equal to our String, very simple, so let's go run that. Now we do this all the time without really thinking about what's going on. So if I put the, the String Mark in the Text Box and click Button, notice it puts Mark down here because we said take whatever is in A, our String, which was, we put in the Text Box. Load it into A and then put that into the Label Text and there it is. Well, here's what you may not have realized and that is, that when you created this A right here, this Variable A of the Type String, you actually instantiated an object. Now C# did this for you in the background but you now have an object out there and you can see this very easily because we're going to load into Label Text 2, A and notice if I hit the period, this looks very much like an object, for a simple reason, it actually is. Notice the methods and properties and some of these were generics. These things with the little less than, greater than signs out here are generics. We'll talk about generics later but I'm going to go down to, notice there is a linked property. So I can check the length of my String, but length is going to be an Integer and I'm trying to drop it into a String Value for the text and so I'm going to have to convert this to a String to make that happen. Alright. And that'll take care of that and so now when I run this and I type Mark in my Text Box right here, you'll notice that the, this right here, dumps the String into Label 1 and then Label 2 notice, it's getting the linked property. Converting it to a String you put in there, but notice there's 4 characters. If I put an S on this and redo it, there's 5 characters. Right. And if you count them, you will see there's 5. Well, what else can we do with this object? Well we can do all kinds of things. Let's come down here and in Label3.Text let's do a little check. We can take our Object A and call the Contains Method and say, does it contain what's in the String B? And if you remember I created B right up here and let's convert because it's going to basically return a Boolean Value and let me correct my spelling here. And so now, when we run this, you will notice again, I'll put in Mark, click the button. Notice there's the String, there's the count or the length of the characters in the string and it's true. Now if I change this to Bob and Print Button, now notice false, because there is no AR in the String now because that's what we're checking for on the Contains right here. We're checking to see if A contains B and B is AR. So we got a false on that one. One more, let's just do here, Label4.Text. Let's set that equal to something and this is really cool. You use this a lot and it's already there, you may not have realized it. Notice we can just call Two Upper and you know what that's going to do. It's going to take whatever is in the Text Box, convert it to all upper case and show it to me. You use that a lot for passwords, for all kinds of situations, part numbers and those sorts of things. Click button, notice it's converting it to upper case Mark. Now it's doing this all because when we create a String, it's actually an emutable object. It can't be changed and so any time that I'm adding letters to this, I'm actually destroying an object, creating a new one. We will talk about dealing with that and we'll actually kind of expand on what we're talking about here when we look at the StringBuilder Class a little bit later on in the course. But anyway, I just wanted you to see, these Strings are really pretty neat, pretty complex objects. We kind of take them for granted but I wanted you to see what's going on in the background and we will build on this knowledge a little bit later on.

(C# Variables / Understanding Scope)

Scope is a term that you're going to hear and read about in documentation and in C Sharp I just want to explain very briefly here in this video, what exactly is Scope? Now usually when we're talking about Scope on a project, we're talking about, you know, how big the project should be, what all's it going to involve. When we're talking about a development environment like C# or programming, we're usually talking about the variables. And when we talk about the Scope of a Variable, we mean how much of the code that's written here can actually read and understand and make reference to a variable that I have created. Now there's a very simple rule on Scope in C# and that is variables are available within the Code Block in which they are defined. Now the Code Block are those little squiggly braces and let's just look at a demo of that. So what I'm going to do is just jump out into Visual C# 2010 Express and if I just drag a button onto a form and double-click that Button and you'll notice I'm just in a standard old Windows Forms Application. I just opened it up, did a new project, took the default. Okay. Notice here that if I create an Integer called A and set it equal to 54, I can very easily grab my Button 1 Text and set it equal to A and convert it to String, so they don't give me any problems. Alright. And so when I run that, no problem, I click the button, notice it turns to 54, no big deal. Okay. But if I come down here and try to create my own, private void Mark and I try to create an Integer B and set it equal to A, I'm going to have a bit of a problem. Notice there's a little red squiggly under A and if I mouse over it, it says A doesn't exist in the current context, why? Because A is scoped right here within this Block. Notice this curly brace or squiggly brace, whichever you want to call it, technical terms both. It's defined, it's only alive from this curly brace down to this one. Well once I jumped into these, it doesn't know what A is, so A is within Scope inside here, it is now out of Scope. Okay. And so we're getting the curly brace. Now if I take A and I'm going to come up here and give myself some room, so you can see this. If I take this line and highlight it and drag it up here and put it there, notice it's now in Scope for this brace down to this brace so it's in Scope for all of this. Notice my little line went away right here, I can now use A. So what I've done here is, at this point, this variable is really at the Class Level and it's available to any method within my class and if I put inside either one of these, the Scope is for that particular method only. Okay. Now you want to use this to your advantage and at this point, you're actually beginning to kind of manage memory if you will because if I want this to live. Now this is not going to be a big, huge thing and I'm not going to be able to detect on the, you know, the performance of my computer, whether or not I'm storing A at this level or not. But just understand that when I put it inside here, then once this method has been used and I jump out of this method, this thing, the Garbage Collector will destroy that and hand the memory back to the system okay? So there is a, a truth to the fact that by putting it up here, I am going to consume more memory because this is going to have to be stored in memory throughout the execution of this class. Until we kind of leave this class right down here, until we get past this class, then we're going to be consuming memory for this Integer A. Okay.

So anyway, that's what they're talking about when they're talking about Scope in these things. Very simple concept, just remember the variables are in Scope or they're available for use and manipulation within the curly braces in which they're defined.

(C# Variables / Statements)

Statements is one of those terms that you read about out there when you're looking at various documentation, it's in all the books but a lot of times we just kind of never really get a good grasp of exactly what a Statement is and this can be confusing. And so I want to kind of help you with this because this is something that I see students doing all the time. They'll think they kind of understand what a Statement is, we all kind of know what it is but then when you say, what exactly is a Statement? We all kind of tend to say well it's kind of, I think, well what I've always thought it is and when you're reading the documentation on C# and you're trying to do something more complex; those kinds of things will really cause problems for you. So in this video, I want to very quickly go over the different types of Statements that are available in C# so that this doesn't confuse you when you read them. And it is confusing because they will interchange the word Statement for about six different types of Statements. So understand that a Statement is simply a code instruction, terminated by a semicolon. Now in it's simplest term, it is a one sentence thing, okay a simple statement. Notice this Int, I N T A for creating a Variable called A, it's the type Integer and we're setting equal to 53 and we put a semicolon, that's a simple Statement. The easiest way to remember Statement is, it's a line of code that ends with a semicolon in C#, that's a Statement. Now what really confuses you as you read through the various pieces of documentation is, there are 6 other places where they will call things Statements but they don't look like this simple thing right here. And so it kind of makes you think, well I thought I knew what a Statement was, but I guess I don't. So let's go over those real quickly to make sure that you're not confused at all when you read documentation. First of all, a simple Statement, we've already talked about it, we use this to define Variables, Constants and that sort of thing. Now a Code Block is a Statement, but keep in mind in Code Blocks, like If Statements and some other things, where the code is expecting to see one line and evaluate that. We can put Multiple Statements grouped together between curly braces and it treats everything within those curly braces, as, you guessed it, as a Statement and so we get in our head that a Statement is that one line of code that ends with the semicolon. And then we see these multiple lines and curly braces and in documentation, you know, they'll always say, well the next Statement and you're thinking, this is the top line of the information. Those curly braces know it's all the information in those curly braces. Now an Expression Statement, you'll see that term, they're talking about a Statement that again ends in a semicolon but the Statement is evaluated to produce a result. You might see Integer A equals and then a method called, Add This and you pass in two parameters but the thing you're looking for there, is that something's evaluated to produce a result. It's loading it into something else and usually it's a method called and that sort of thing so that's an Expression Statement. We're going out and running an Expression, we're bringing it back and that sort of thing. Then a Selection Statement, this chooses a value based on a condition. You see this most of the time in your If Statements or your Switch Statements and those pieces in there are called Selection Statements. The next one is an Iteration Statement and this is where we're doing, repeating a Code Block for each item in a set and of course this is your classic For, For Each, While, Do While, those sorts of things and again, it's called a Statement. It's usually that very top or the very bottom, depending on which one you're doing, line where it says, you know, if J equals 5 then, that's your Iteration Statement. Then your Control Statement, this transfers execution from one set of Statements to another. This is things like Break, Continue, Go To, your return in your methods or your Functions, so that's a Control Statement. Now these things can confuse you, so understand really, a Statement is a complete thought for a complete line of code or a group of code that's being evaluated as if it's a single line of code. This is very fundamental I know, it seems very elementary but I see constantly in live classes, where this confuses people about, wait a minute, where's my Statement? Because what you're doing, a lot of people will look at code examples and it will refer them to a Statement and they get a little confused about, wait a minute, what exactly is the statement here? Because I thought I knew what a Statement was but this doesn't seem to look like it. Okay. So understand there are 6 types, they each have to do with the functionality that they're being used in and it's kind of, for lack of a technical term, kind of a loosy goosy term sometimes. So just understand, Statements can exist in different forms.

(C# Variables / Expressions)

Expressions are a fundamental part of C#, they are a fundamental part of any language. It's another one of these things that I continually see students a little bit confused about exactly what an Expression is. It's kind of like, if you've ever had a small child, say, 3, 4, 5 years old ask you, something like, what does that mean? You know, what does the word that, T H A T, what does that mean? How do you define that? And when you try to define it to them, you always end up trying to use that word and you're like, well that means, when you're talking about something, you're telling it that and, and they're just sitting there looking with this puzzled look and you're confusing them more than you're helping them and often I see the same issue with programmers with Expressions. Now we all kind of understand what an Expression is and if you've got this nailed down and you're okay with this, then just skip this video, it's going to be short anyway. But I want to make sure we're all on the same page here. Because as you inevitably try to deepen your understanding and your programming skills with C#, the documentation that you're going to be reading out there will refer to an Expression and if you're not really sure what it is, you kind of get lost there and then you kind of lose it on a grander scale from there. Okay. An Expression is a String of Operators and Operands, that's the definition you'll see all over the place, that brings up another question. Okay, well what exactly is an Operator and an Operand? Now you learned this in the third grade. Now depending on your age, the third grade has been longer ago for some of us than for others, for myself, it's been an amazingly long period of time now. Okay. So let's talk about Operators and Operands. An Operator performs mathematical operations and when we talk about an Operator in an

Expression, we're talking about the Plus Sign, the Negative Sign, the Asterisk that we use for multiplication, the Slash that we use for division, the Remainder sign which is the Percent sign here. Greater than, less than, greater than or equal to, less than or equal to, not equal, all that stuff, those are Operators. Alright. An Operand is the item, usually the number that's being affected by the Operator okay? So if you look at 3 plus 4, you're looking at an Expression, it is expressing a value, 3 plus 4. The plus is the Operator. The 3 and the 4 are the Operands, they're what's being operated on. Okay. So just wanted to make sure that you're on the page here. Now this looks incredibly simple, however again, I keep finding especially in live classes, places where people are confused on this because in instructions on examples or on little projects that students have to work on, people are getting confused on such simple things as what's a Statement? What's an Expression? What's meant by those terms? Well that's it. So this has been a, a review of the third grade math class. Okay. But I just wanted to make sure that you, we're all on the same page with this, trust me, this will help you to understand what you're reading in documentation as you dive farther into C#.

(C# Variables / Using StringBuilder)

The StringBuilder Class is something that you need to be aware of, you need to understand it and you need to have this guy out there or this girl, I don't know what it is, don't want to know. You need to have this one out there in your programmer, mad scientist, developer toolbox to pull it out when you need it in certain situations. And it can be very, very valuable in those places in code where you're trying to manipulate strings, you know, very rapidly. So let's talk about StringBuilder here. What exactly is StringBuilder? Well it is an alternative to the String Type. Now in a separate video in this course, I talked about Strings and the name of that video was just Strings. Then I did a little example about the String Object that's created when we use that String's Data Type or that String Data Type and so StringBuilder is an alternative to that. Now to understand StringBuilder, you need to understand String. When I create a Variable of the type String, understand that I'm creating an object that is immutable. Immutable in English means, it cannot be changed. So basically I've created more or less, a read only variable, read only memory, I can write to it when I create it and then I can't anymore. So every time you change that String Value, in the background, C# is deleting that old object and creating a new object with the same name, with the changes that you put out there. Now obviously, anytime you use one of the methods in the String Class, a New Object gets created and so this is not very efficient. Some of the least efficient things you can do in an Object Orientated Programming Language, is Instantiate New Objects and then destroy old ones. So this is okay most of the time but this is not a good solution at all for Strings that change very often or repeatedly or let's say we're looping through some things and we're building or concatenating onto a String and so StringBuilder comes to the rescue here. So how is it different? Well StringBuilder is itself a class, however that class creates an array of characters. Now interestingly enough, this is how a lot of other programming languages actually deal with Strings right out of the box. C# gives you this choice of using either or and using the String Type can be very simple, it can be very easy and a lot of times, quick and easy, is a good trade off for efficient until you're really straining it on performance and then you need the efficiency. So with StringBuilder, we're actually building an array of characters but what's cool about it is, the methods that we get from this StringBuilder Class allow us to deal with it as if it's a String. And in the background, it takes care of the Array Functionalities to continue to represent this to us as a String. It's allowing us to modify a String without having to create a New Object. It is modifying the underlying array if you will. This is a great choice for Repeated String Concatenation because we're simply only adding values into our array if you will. Now the StringBuilder is a member of the System.Text Namespace so you'll have to refer to that Namespace and set a Reference to it if you're going to use StringBuilder. Fortunately, most of the projects in C#, when you create them, will automatically include that and you don't have to worry about it. Now the best way to understand this, is with an example of StringBuilder and I don't think I have time to do that justice in this video. So we'll just call this Part 1 and then we'll jump over into a StringBuilder example video and actually show what using StringBuilder looks like and feels like. Okay. So join me in the StringBuilder Example Video and we'll get our hands on this, in a code environment.

(C# Variables / StringBuilder Example pt. 1)

Now let's take a look at an example of the StringBuilder and using this StringBuilder Class and Type, out in some code. So you'll notice I've opened a C Sharp 2010 Express and this is Part 1 by the way. This will be a two parter. No way I'll ever get through this in one video without putting you to sleep. So notice I'm going to create this and you will find this example out there in your Work Files folder, I'm just going to call this StringBuilder Example, we'll click OK on this. What we're going to do to make this example and to demonstrate exactly how the String Builders working, I'm going to drop a few objects out here, I'm going to put a couple of buttons. I'm just going to double-click button twice and I'll put my Button 1 and 2 right about here and then I'll put a couple of labels out here. So I'll go double-click on Label a couple of times and I'll drop those down here, line those up together and then I'll give us a Text Box to use to put things in to add to our particular StringBuilder Object okay? So that's what it looks like, we've got a Text Box, two buttons, two labels. Now what I'm going to do is, just jump to the code by double-clicking on Button1 and let's start to create the code to demonstrate this thing. Now the first thing that I'm going to do, if I'm going to create a String called A but I'm going to use the StringBuilder, I've got to create a new instance of that StringBuilder Class. And I'm going to do that up here at the Class Level, so it's easier to write some of this code and notice I'm going to create an A. It's going to be of a type StringBuilder and so there's A and I'll set that equal to a new instance of StringBuilder and I'm going to pass it a value here of 5. Now what I'm setting is, is the initial capacity for this because what StringBuilder Class is actually doing for me, is really creating a collection of characters in the background. An array if you will, called A and it has five items in it or five, a capacity of five. And we'll talk about length and capacity because that tends to confuse people, first time they use this thing. Now let's go down to our Button1 Click Event. The first thing that I want to do is take whatever is put in the Text Box and I want to load that into my new String Character or my new String Type here that's built off of StringBuilder as oppose to the standard System.String Type. Okay. And so what I'm going to tell it is, I want to append whatever is in Text Box 1 Text Property, onto my StringBuilder Object here A. Okay. So that's my String. Now to see this, this will take whatever I put in the Text Box when I click the Button, it'll take whatever is in the Text Box and append it onto my String. It won't create a New Object, it won't destroy an object, it's just going to append it onto the object that's already created for me here when my class for the Form 1 is built. Now to be able to see that, we are going to load Label1.Text and we're going to set it equal to A. Okay. Now if I spell Label 1 right you would be surprised in how much easier all this works. Now notice right here, it's saying it can't implicitly convert the Type StringBuilder to String so I'm going to do that here. I'm going to say to String because again, this looks like a String but it's actually a StringBuilder Class Object. Okay. So if I run this, let's look at this, kind of simply here first of all. Here's my form and if I type the word Mark in here and click Button 1 notice it just simply drops my String that I created right here into here. Now if I put S on this, notice it, it appended the entire thing. Okay. Now if I wanted to just add an A, I'll put just an A and notice it appended just an A down here. Okay. If I put a large X or capital X I guess I should say, it will append that and every time I'm clicking this button, it's appending these things, but it's not recreating an object. It's, it's doing it very efficiently and everything's going along just like I need it to go. Alright. So as you can see to append things onto Strings or to concatenate Strings, gets very, very efficient with this Model. Alright. Because notice I can put a bunch of things out here and just keep clicking this and clicking this and it's appending them right off the form. Okay. Now there's another cool thing I can see here with using the StringBuilder Class and I will put that in Label 2 and I'll put that in the Text Property of 2 and that is, I'll, let's make this a little easier to read. We'll set the length or we'll read the length and I'll concatenate that to this and we'll just call A and we'll get that property off of that A StringBuilder Object and notice there it is. Length, we'll convert that String so that we can see it, now when we run it and I put an M up here and click the Button and notice, that is, my String now contains an M or my String Object, my StringBuilder Object, I should say, A and the length is 1. If I click it again, it added another M because it was in the Text Box and it's 2 and if I add an A in here, it'll add it, might know it's just appending it on to the end of that array if you will and notice I'm up to 3 now. Okay. So that's Length. Now what I'm going to do is stop here, this will be the end of StringBuilder Example Part One. Join me in StringBuilder Example Part Two and we're going to talk about capacity, which is different from length but it confuses folks and then we'll talk about how to clear our String and that sort of thing. So join me in Part Two.

(C# Variables / StringBuilder Example pt. 2)

Welcome to Part Two of the StringBuilder Example Video. And as a quick review in Part One we built this simple example of using a StringBuilder and whatever I put in the Text Box goes into my String and it shows me the Length and I can click it again and add it. And I can add some more things to it and notice it's counting the Length for me. Alright. What I want to do in this video, is also add a property to this, that's called Capacity and explain that. Because it tends to confuse people and I'm going to make my Toolbox go away so that you can see all this. I'll take my semicolon away and I'm going to continue to concatenate here and I want this to be a Multi-line Label. So I'm going to call the Environment New Line Property and what I'm going to do is append onto new line. Notice, Capacity and then I will add A and I'll call the Capacity Property and then I'll do the Two String Functionality to convert that to a String. Now I want to explain the difference. If you remember, way back up here when I created the StringBuilder Variable A as an object of a StringBuilder Class I passed it as part of the Constructor, 5. And that was telling it, create this array in the background and give it a capacity of 5 and then I want you to manage it from there. So if I run this, you will notice that when I load the variable, it's already created, it's already been instantiated and when the form opened up. But when I start to load or append stuff into it, notice when I put an M in the length is 1, the Capacity is 5. That is saying I have 1 item in there, I have a Capacity of 5. Now if I go put something else in, notice it's 2 and still 5. I put something else in there, there's 3 and 5, I'll put another T in there, that's 4 and 5. I'll add another one, that's 5 and 5. But notice when I add something else and I'll put a W in there, notice my length went to 6 and if you count these letters, they're kind of small and it goes to 6 and the Capacity is 10. Alright. And so what's the capacity is taking care of is, again this is back to like creating variables. If I know that I'm going to store 5 characters or less, that's a good number to put in here. But if I'm going to store more than that, go ahead and put it at 25 because there is a little overhead in resizing this array if you will. So if I set it at 25 to start with now I can go back in here and I can put 3 characters at a time and notice how many times I can just append to this without having to go through the overhead of increasing Capacity. Once I do that, notice I just increased it to 50. So it doubles whatever I set it up as right here on the front, whatever the Capacity is, whenever it runs out, it doubles that. Now there's another thing I want to be able to do and let's go back out to my design, that's Button 2. When I've had enough of this, how do I clear this thing out? I don't want to destroy it but I do want to clear it out. Well I just call my, my Instantiated Object A and I'm going to call the Clear Method on this and then just by clicking Button 2 I will clear that out. How easy is that? So now I'll go back and run this again. I'll, I'll add some stuff to it, there it goes, Capacity of 6, length of 6, Capacity of 25. If I click Button 2 okay and come back, notice my length is 0 and 25 and I can start over again. So I'll just keep hitting F, there's 4, if I hit Button 2, it clears it and when I come back it had 1 in there, so I'm 1 and my Capacity is still 25. Okay. So you can see how quickly this thing can append things or manipulate strings without having to destroy the object and create a new one every time and incur all that overhead. So that's the big difference between the StringBuilder Class and a true or a what we call a simple String Object in C#. So keep this kind of out there in your Toolbox. When you need to manipulate strings repeatedly and quickly, this is a great way to do that without incurring a lot of overhead for creating and destroying objects.

(C# Variables / Enumerations)

Enumerations have been around a long time in C#, there's nothing new about them in C# 2010. But I want to show them to you here in case you're not comfortable with these things. Because in the example, I'm going to show you something really cool that you can do with Enumerations and you can get a lot of mileage out of these things and do some really cool stuff, really quick with them. First of all, what is an Enumeration? Well it's a Value Type, it's a variable that we're going to create that defines a set of named Constant Values. It's a variable that has multiple values stored inside it. Not only does it have multiple values but it also automatically stores Integer Values or Indexes if you will for each of those values. Now when we create an Enumeration, you seldom see a type mentioned. The Default Type is Int and that is the System.Int32. That's 4 bytes, 32 bits, it'll store a lot, we won't go into all that right now, but the variables are automatically enumerated or indexed if you will, starting at 0 unless you change that. I'll show you a little bit more about that in just a second. So why do we want to use Enumerations? What's the big deal with these things? Well the big thing is, they improve the readability of your code and that makes troubleshooting much, much more easier. And trust me, you will come back and look at your code in a few weeks, few months and you will wonder what kind of controlled or uncontrolled substance was I on when I wrote this mess. So here's the bottom line with Enumerations, instead of a situation where we want to store and manipulate values based on, for example, the days of the week and so we create an Integer called Sunday set it equal to 0. Then we create a Variable of a type Integer, call it Monday, we set it equal to 1 and then we create one called Tuesday, set it 2 and I would have semicolons on the ends of all those. But to try to make it clear here, notice we just set those up, and we just keep going all the way through Saturday and we'd end up at 7. Now if I create an Enumeration, notice I will use the Syntax Public if I want it public, enum Days and then I will just list inside curly braces, all the different variables that I want to have out there. Okay. Or the different values that I want to have out there. Now the Values can then be read as Text or the Enumeration Value and you're asking yourself what Enumeration Value? You didn't mention one. I didn't and so Sunday will be 0, Monday will be 1, Tuesday will be 2, Wednesday will be 3, you see the pattern here and then I can use those in various ways. Now we'll look at using these in code in a separate video entitled Enumerations Example but bear with me here and let's talk about kind of chalk talk in Enumerations just a little bit here. Now I've already mentioned that the Default Data Type is Integer and you could actually use any type you want except the Char Type okay or the Character Type. So notice I used public enum Days, Sunday, Monday, Tuesday and this is Sunday 0, Monday 1, Tuesday 2 and this is a type of Integer. And so what's, what I can store in Sunday, Monday or Tuesday as far as my Enumeration Value can be anything that will fit in Integer and that can be quite large. If I want it to be even larger, notice I can use public enum and then I put the word, I mean size is the name of my enum. So instead of Days, this ones called Size and notice I put a Colon and the Data Type that I want it to be. Notice if I leave the Colon and the Data Type off like I did up here, it defaults to Integer. This will create Sunday, Monday, Tuesday out there, 0, 1 and 2 but it will be of the Long Data Type, so it can store larger numbers in each one of those for the Enumeration. Now here's where that comes into play, I can assign Enumeration Values if I would like. So I can create it as public enum Days and if for whatever reason, I'm using some sort of factor, maybe this is a discount we apply, maybe it's additional shipping charge, maybe it's how many hours that we're going to commit to getting it out. It, who knows what we're doing with this programmatically. But I know that Sunday is equal to 4 and Monday is equal to 6, Tuesday is equal to 12. These do not have to be sequential as a matter of fact, two of the Enumeration Values here, could be the same, Monday could be 12 and Tuesday could be 12. That's pretty much the chalk talk skinny, that's a technical term for you on Enumerations. The best way to understand these is to actually go out and play with these things in code and that's what we'll do in a separate video. So go look up Enumations Example and we'll just code up and show you in code what we've all been discussing right here, so join me there.

(Classes / Understanding Classes)

Now for the next few videos, we're going to dive in to the topic of classes and we're going to talk about a lot of different things about classes but let's start here with understanding the basics of what a class is. Now I'm going to just kind of chalk talk about a class here in this short video and then we will jump out into some code and look at actually creating a class and then Instantiating Objects off of that class. But to talk about it from a textual standpoint here and to help you understand, a class is a User-Defined Construct. It is a collection of code that we create and we can create as the programmer and what it's going to do is group together variables, methods and events and give those things meaning as a group. For example, a Class Name might be dog, cat, animal okay, database. So class is going to allow us to create variables, methods and events that are all related in some way and they provide some sort of unique yet related functionality to each other. And usually we use a class to build objects that represent real world functionalities. And as we go through the course here, we'll talk about some examples of using classes and stuff for a, a veterinarians office and so you'll see classes like dog, cat, animal, snake, those sorts of things. Because when you bring a dog into the vet, the dog's going to have certain characteristics, certain properties, name, weight and owners name. But then it needs to have certain actions taken. We need to weigh the animal, we need to feed the animal, we need to innoculate the dog. We need to treat it for Heartworms, those sorts of things and we can program all those into a class and then the class becomes a blueprint or a template if you will and we use that to Create Objects. Now off of a single class, we can create multiple objects that exist out there in memory and these objects are each identified uniquely as variables and each of these objects have their own state. Their properties can be certain characteristics and then these classes create objects that reside in memory during their lifetime. So I can take a class, use it as a blueprint, create an object off of it that has the properties, the methods, the events and all that were built into that underlying class. And then this object can live in memory and be used to gather and maintain state about whatever the object is. And then use some of the properties and so forth and the methods of that object to write information in the databases, to read information from databases, to interact with other classes, interact with other programs, do all kind of cool things. So there's a lot there about classes and this is all basic stuff but the best way to understand it is to see one in action. So in the next two or three videos, we'll go out into the code and actually create and use some classes so that you'll really understand what's going on here.

(Classes / Creating a Class)

Now let's take a look at actually Building a Class in the code. And so I've opened Visual C# 2010 Express, I'm just going to create a New Project and I will just call this Class Example. You won't see this one on your Work Files because there's not going to be a lot going on here, I'm just going to show you a couple of things and we will build classes as these examples build in some future tutorials here. So I'll just take the name there, take all the defaults, you notice it brings me to a Form. And to create a class I'm just going to drop a Button on the form to give us a little bit of control with our code and I'll double-click to get back there. Now notice one thing though, since the class is the foundational piece of functionality or the foundational building block for an Object Orientated Program and an Object Orientated Environment like C#, notice that we start off with a class here. And our form is itself a class and that's where our button went and we'll get into some more about that stuff a little bit later on. But if I want to create a New Class that I can use here, all I have to do is, come down here, I can stay within this Namespace or I can create a New Namespace. But for now notice this class, right here, this is a Partial. We'll talk about Partials later, starts here and ends right here. So I can come down here anywhere and I can create another class and I'll just say Public Class Mark. Come down to the next line, open my Curly Braces and then go a couple and close them and everything I put inside here will be part of my class. And so I can create Integer A and set it equal to 54 and for the most part I have created a class. It's very simple, very easy to do and notice I put public on here. If I didn't put public and I just created a class then it created that as a Private Class and we'll talk about these Access Modifiers, Public, Private and all that in a separate video here. Okay. So that created a class within this Namespace, Class Example. Then I can come up here and instantiate an object from that Class and use it and I'll show you that in a separate video as well. But I want to show you another way that I can create a class in C# and a lot of people like to do this because it makes it a little easier to read the code. Now I'm going to take this class outright here, if I go into the Solution Explorer, right-click on Project Name and choose Add. I can add, notice, New Item. Now this is kind of unique, if I click on New Item here, you'll notice there's a class right up here at the top and I can take that or they've also got a little shortcut in here. I can right-click here and say Add and it shows me Class right down here and I can also Shift-Alt-C to do it but that brings me to the same place. Okay. And then notice my class is going to have this CS Extension right down here for C# and I can change this to Mark.CS and then click Add. And notice the difference this time though, is Mark CS is it's own separate file and if I open the Solution Explorer and I go back to my Form 1. I've still got Form 1 here, here's my Form 1 Class. Right. My Namespace Class Example, just like before. Okay. If I highlight all that. But now I also now have out here in a separate file, Mark CS and if I double-click that it brings it up and notice the Namespace here, notice in Form 1, the Namespace was Class Example. In my New Class file, Mark.CS Namespace is still Class Example and there's Class Mark and I can come out here and start to create, you know, whatever I want to build in my class. Now again, a lot of developers like this because it puts the classes out here in separate files and I can see them and there's some other more advanced things I can do with them out here but we won't get into that right now. But anyway, this is just kind of the simple basics on creating a class, not hard to do at all. We can also, if you'll notice, if we start class, we can do Code Snippets. Just start spelling class and as soon as it appears here, hit the Tab twice and it sets it up. This is a little better because as many times as I type these little Curly Braces, I just simply, I don't know why, can't hit them every time. Okay. And so using the properties thing and then, you know, I can change that and now I've created another class. So I've got a Class Mark and a Class Bob. This by the way is bad design because the name of the class file is Mark and now I've put a class called Bob in here and that's just going to confuse everybody. Okay. So the way I should do that, go out here to Class Example, right-click, Add, Add a New Class, double-click right here, call it Bob and now I have a class out here called Bob to go with the one Mark. Okay. So anyway that's the very simple way to build a class here and as we go through the next few videos, we're going to jump into things like instantiating an object of these classes, our Class Access Modifiers, Constructors and so forth. So join me in the other videos and we'll build on what we've learned here.

(Classes / Instantiating an Object)

Now let's take a look at Instantiating an Object off of a Class or turning a class into an object. Now you will hear those terms interchanged, class and object and they're not really the same thing. Okay. But you will hear this all over the place, don't let it confuse you and I promise you before you get to the end of this course, you're going to be using those terms interchangeably as well. So you'll be just as guilty as the rest of us. Alright. So I'm going to just double-click, I've got a form with a button on here, I just opened, just a Generic Windows Forms Application. I'll double-click my Button to get back here to the Code Behind and I'm just going to create a class and then I'm going to instantiate it. Okay. So if I type the word Class, double-click the Tab button or hit the Tab button twice, it will set it up for me and I will call my class Mark and we're going to talk a little bit more later on in this course in separate videos, about fields and properties and methods and so forth. But for now, I'm just going to create a field called A and I'll set it equal to 34 and then I'll create one called B and I'll set it equal to 56 and then I'll get real creative and create one called C and set it to 67. And then I'm going to create a little method here and notice I'm just going to call Public Void, meaning it doesn't return anything and I'll just call MTL. I won't pass it any parameters and then I will put my two squiggly braces, curly braces in and then I'll just call them Message Box. Message Box.Show and I'll just say Hello from MTL. Okay. Pretty straightforward, useless program but I specialize in those sometimes. Now to instantiate this, this is our class and notice the functionality that it has. It has three fields that we can set. Alright. We can change the value of these variables, A, B and C, at Class Level and we have one function. Anytime we call MTL. Right. It is just going to throw up a Message Box that says Hello from MTL. Alright. Very, very useful program we're building here. Now to use this class, what I'm going to have to do is instantiate it and the way I do that is I tell it, okay, the Type Mark and notice, Intellisense sees my type Mark here. So I'll say Mark and then I'll just call it M is the name of my variable that is of the type Mark and I set it to equal to a new instance of Mark. And I have to tell it that it's my class that I'm looking at and at that point, I just created in memory a variable called M of type Mark that is exposing this functionality to me. Now notice if I hit M and then a period, Intellisense can see those things and there is my MTL right there and notice I can't see these integers. What's up? Shouldn't they be up here? Well yeah they should. Okay. I am going to go ahead and grab MTL and call that. So you notice the squiggly goes away, why didn't these happen? Well it's because of those dastardly Access Modifiers. So I'll have to call these things Public and I'll just grab this and copy it on the front of each one and notice now, when I come back up here and hit M, you'll notice A, B and C show up. Now that's a real good example of what these Public and Private and Friend and Protect Modifiers and all that stuff do. Not Friend, but, I'm thinking VB now, but we'll talk about those a little bit later on. Okay. But notice I can set A equal to 103. Right. And it's very simple to do that. Now let's go look at this when we run it. So I run my application and when I click on my button1, I'm going to create this New Object in memory, this M Object that's of a type Mark and that it now exposes all that functionality. So I will click this, click the button and notice Hello from MTL, it happened and since I didn't really list my variable right here, which we could do that very easily, by setting our Button1.Text equal to M.A and we'll probably have to send that to a String to make it behave. Okay. And if I put my little dill out here I should be okay. Now let's do it again and notice that changed to our 103 that we set right here okay? Now here's the cool thing about objects. Okay. Notice I can do Mark R equals a new instance of Mark and I can call M.MTL. And this is going to create a memory variable called M that's of the type Mark and this one's a type R. This was an R Variable of type Mark and so I have M and R, two separate objects off of the same class. And so when I run this, you'll notice that there's the first one, Hello from MTL and there's the second one, when I clicked OK, Hello from MTL and then this 103 here. Okay. Play with this and start to do some, some different things and you'll see that you can use a class as a blueprint for multiple objects and then we will build on this functionality as we proceed on through the course. This is very basic stuff here. Okay. But if you're new to this, if you're kind of new to Object Orientated Programming, this is going to help you tremendously until you get this part down, until you have this very solid in your brain, you're not going to be able to go very far with Object Orientated Programming. Okay. So that's Instantiating a Class.

(Classes / Class Access Modifiers)

Class Access Modifiers are a bit of a functionality that we can use to start to really design Object Models and different levels of encapsulation and functionality in our programs. Now these are also known just as Access Modifiers and sometimes you'll just hear the word Modifiers used by itself. Now these kind of rear their little head in a demo that I was doing in a different place in the course. And if you go look at the video entitled Instantiating an Object, I ran into an Access Modifier issue and some of the Class Fields and so you'll see that, in that video. And keep in mind now that these Access Modifiers show up in a lot of different places and so you really need to kind of get these things straight in your brain about how they operate and what they do. So at the absolute bottom line a Class Access Modifier is simply a keyword that controls the accessibility of various types and members. Now these Modifiers can be used on classes themselves, they can be used structs, if you don't know what a struct is, not to worry, we're going to talk about structs later on in the course. Delegates, we're going to talk about delegates a little bit later on in the course and then methods themselves and we will touch on them when we dig into methods a little bit, again later on in the course. So what are we going to talk about in this video? Well we're going to talk about these Class Access Modifiers. Now there are five Class Access Modifiers that you can choose from when you create your classes and really what you're trying to do here is determine, okay, what's the scope of this class? Who can Instantiate Objects off of this class? And really we determine that by where they are. Now the first one is Private, which by the way let me back up here. Private is the default, if I just say, Class Mark and I don't mention anything about a Modifier, C# is going to default it to Private. Alright. Then there's Public, there's Protected, there's Internal and then there's Protected Internal. Now let's talk a little bit more about each one of these. When I mark a class with the Public Modifier that means this class can be accessed by any other code in the same assembly or another assembly that References. Now what that means in English is that you can Instantiate Objects off of this class that's marked Public from anywhere. So if you need to access a class on this one, you can, you just go through the Instantiation Procedure and create a Type, a Variable of a Type, whatever the class is that's public and you can create a new one, a new object off of that. Now if your class is marked Private, then the only people who can create objects or instantiate objects off this Class is code that's in the same class or struct. Okay. And you'll see that a little bit later as we move through here in some separate videos. So if it's Public, you can instantiate an object off of it from anywhere in code. If it has that Private Modifier, you can only instantiate an object off of this class, if you're code is within the same class or struct as the class is marked Private. Okay. Now Protected, this means that I can access the class or in other words, I can Instantiate Objects from, from that class, only by code in the same class or struct or in a class that is derived from that class. Now by Derived, we're talking about Inheritance here and we will play with Inheritance and give you a little heads up on Inheritance a little bit later on. We talked about Inheritance briefly when we talked about the Object Orientated aspects, understanding the basics of Object Orientated Design, but we will talk about Inheritance a little bit later. So keep that in mind, when we talk about that, most of the time we use Protected to simply limit that objects can only be instantiated from the same class or especially from those derived from that class. Now the other Modifiers are Internal and this is where a class can only be instantiated by any code in the same assembly but not from another assembly. Now this brings up a whole new term and when we're talking about assembly, when we compile our project, it's going to go into assembly of the Namespace of the project that we built or our Solution. Now we can use the Assembly Builder Tool to basically break different pieces of code up into different assemblies. Okay. And an assembly, the best way to think of an assembly is what the old DLL used to be. This becomes the freestanding piece of code that we move around from machine to machine to machine to make our executable run. Okay. And then our last one is Protected Internal, now notice the wording here. A class that's marked Protected Internal can be utilized in the same assembly or in any class in any assembly that derives from the containing class. And we're getting into some Inheritance speak here, so just kind of put this one on the back burner for now but that's your five Modifiers. Keep in mind when you build a class, you will need to think through and put these Modifiers on there. By default, if you don't mention a Modifier, C#'s going to assign it Private. So keep that in mind, just kind of stick this back there in your brain, you'll see these as we move through here in some future examples.

(Classes / Understanding Constructors)

In this video, we're going to start to wade just a little bit deeper out there in the Class Construction World and I want to talk to you about something called Constructors. Now if you've never seen these things, they're a bit funky the first time you work with the, but they make sense, you can do a lot with them and so let's talk about Constructors now. A Constructor is a method that is executed when an object of a class is instantiated. Let me say that another way. If I build a class called Mark and then I create an object from that class, I instantiate an object, that class called Mark is going to have a method that's named Mark and that is the Constructor. That Constructor is there automatically whether I put it there or not. Okay. But it's always going to be there, a Constructor is just a fancy name, it's a designating name for the method that has the same name as the class and gets run immediately when the class is first created or when the object is created, I just did it. I just switched them. Okay. Notice how I used object and class interchangeably there but you're going to hear that everywhere, get used to it. What we use this for is, it allows us to initialize any aspect of New Objects, when the object comes into being and this can save us a lot of time and trouble. Now a Constructor is created by default, make sure you got this, make sure you don't miss this. If I create a class and don't mention a Constructor, don't even think about a Constructor, a Constructor was created for me. Where I'm going to get into some really neat functionality here is, is when I create my own Custom Constructor and I start to put functionality in it. Now let's go the next level, if you know what Overloading is, this statement's going to really make the light start to flicker for you and that is that the Constructor can be overloaded. If you don't know what Overloaded is, we'll talk about it when we get down to methods and we talk about methods in some separate videos. Okay. Now ultimately the best way to understand Constructors is to see one in action. So if you'll look for the video called Using Constructors. I'll go out and work with some Constructors, build a Constructor, instantiate an object using a Constructor and you'll see what the magic is all about here. But this is something you really need to grasp; this is something that's used all over the place in Object Orientated Programming especially in C#. So join me in Constructor Example and we'll take a look at these things in real life.

(Classes / Using Constructors)

Now let's take a look at using a Constructor out here in code and this will really help you figure out what's going on with these things and you're really going to get some lights flashing about some great functionalities you can add to your C# programs. Notice that I have created a very simple application here; it's a Windows Forms Application in the Visual C# 2010 Express Tool. I just named it Constructor Example; you will see a copy of this in your Work Files Folder and so it will be very similar to this, if not identical and you can very easily just follow along with me right here to get the same thing. Now what I'm going to do is, double-click on the Button here that I've dropped onto the form and we're just going to write some code. I am going to stay within my Namespace but I'm going to create a New Class here. Alright. And the way to do that, type the word class and notice I'm working with a Code Snippet here. I will just hit the Tab key once, hit it again and it puts all my curly braces and everything for me and I'll just change the name. So the class is Mark and I'm going to go down here and just put one quick method in here. Public Void, it's not going to return anything, test this and we'll Build Methods a little bit later on and we'll talk about some of the nuances of that. But in this method I'm going to put a very simple, irritating Message Box and we'll Show and we'll just say, Test This has been called. A very useful program as you can see and so there's my class. The class is Mark, it has a method called Test This and so now all that's left to do is to go up here and write some code to instantiate an object off of this class Mark and start to use functionality which in this case, is very helpful. We'll call Test This and have a Message Box pop up. So the Type is Mark, the Variable name will be M and we'll set M equal to a new instance of Mark and just like that we have our object and notice we're going to call our Test This Method, pretty straightforward. Right. And so when I run this you can guess what's going to happen, I'm going to click on Button 1 and boom, Test This has been called. You didn't actually hear a boom or see a boom but I've got that irritating habit, don't know why and I apologize. So anyway, all that happened, I created my new instance of an object and my instance of this Mark Class was M. I called the Test This, it reached into here, it ran my Message Box and that popped up. Now something happened there, there is a Constructor here, you just can't see it. So if I want to use the Constructor for this class, let's say that when I call or when I instantiate an object off of the Mark Class. I want it to set some fields or some properties to certain values. I want to connect to databases, I want to verify something, I want to make sure that there isn't already an instance of this class that has been instantiated, the singleton model or method. But without getting into all that, let's just talk about building a Constructor here. Well if I just use Public Mark and this looks a little confusing the first time you see it. I just created the Constructor, notice it has the exact same name as the class and I will do another Message Box here to show you the functionality. Message Box Show and you love watching people type the same thing over and over and over but I apologize. Constructor Mark runs, now let's close that. Now notice I haven't changed anything about what I'm calling here but this time when I instantiate my New Object called M off of the Mark Class, as soon as it comes into being, right here, we're going to run this Constructor okay? So anything inside this Constructor executes and so we will jump out here, run this thing. Notice when I click button this time, the Constructor ran and now everything that I have coded up to run, starts to be called. Okay. So that's your Constructor, now you can do an awful lot of things with these Constructors and you can get very creative with them and you can make all kind of neat things happen. Now here's the next big news with these things. You can actually overload these. Now this is a case of the chicken and the egg because we haven't talked about Method Overloading or any kind of Overloading yet in the course, so in a separate video, if you go look it up, it'll, should be right after this one in the course, depends on how we organize things. There's a video out here somewhere called Overloading Constructors. I would strongly suggest you watch that one, you will see this very same idea of a Constructor but we're going to add a new twist to it. But keep in mind that just by putting whatever kind of code we want in a method, the same name as the class, we are working with the Constructor and we can basically, notice nomenclature, construct objects with certain characteristics at the time they are created. So that's what a Constructor looks like and we will dig a little deeper in future videos.

(Classes / Overloading Constructors)

Welcome to Overloading Constructors. Now Overloading is a functionality that we can use on things like Constructors and really on methods, which is actually what a Constructor is here. Notice this is just a method with the same name as the class. We can Overload it by changing the Signature, by creating the same method with different Signatures and we can call the same method but simply by the parameters that we pass, we will call different instances of that method. Now if it looks, sounds confusing, it's only because it is the first time you see it but I'm going to make it clear for you real quick here. So we're going to overload the Mark Constructor. Notice, a little quick review, we've got a class called Mark; we've got one method in it, called Test This. It just does a little Message Box that says Test This has been called, let us know we got there. Then we created a Constructor which is a method with the same name as the class and what I'm going to do now is Overload it. Okay. So I'm going to create another one, Public Mark but this time I am going to tell it that this one is expecting a parameter to be passed in an argument and I'm going to take a String and then I will put my little curly braces in. Alright. So notice now I basically have two Constructors, I have Public Mark that doesn't take anything and this is how we would instantiate here and then I have Public Mark and I'm expecting a String Value to be passed in at the time the object is created off of the class. Now so, let's write some code to do something here and we'll do another Message Box Show and in this one, I'm just going to make it very easy to see what's going on and I'll just concatenate whatever is passed into this, into our Message Box. And so you can see that it's called Mark and then we'll same the name out there. Alright. And with that you'll notice that if I run this, nothing's going to happen. This will not be run if I call this why? Because this is instantiating an object called M off of Mark and I'm not passing anything in, so it's going to read this and it's going to bring up that Constructor. So let's run this, watch it happen. Constructor Mark runs, there it is, because this Constructor right here ran and then my Test This has been called and I'll close this program and notice here's my Test This. Okay. But now what if I pass a String in here. Okay. What if I pass Bob in here? Notice now, I'm telling it, let's create an object called M of the Type Mark, which is my class. Except now when I create the new Mark, let's pass Bob into it. What's going to happen is, the Constructor's going to say, oh wait a minute, dude's passing in a single String so you want me to run this Constructor. Okay. So you notice this time this Constructor will not run, this one will, simply based on the fact that we're passing a String in and it matches the Signature here alright? So we will run this, boom, Mark, Bob ran, that Constructor ran and then Test This got called. Now notice, Test This got called because nothing's changed here, once we overloaded and we got Bob. Now here's something else we can do and I'll just kind of give you a hint here and you can play with this later. Notice I could put Mark, I could pass in a String called Name, then I could pass in an Integer called Weight. Okay. And now every time I instantiate an object I could now pass in, you know, Bob 220 and this would call this particular Constructor. Right. So I will leave it to you to write some code to work with that but notice I now have my Standard Constructor Mark, so if you don't pass anything in up here when you create it, this one will fire. I have a Constructor that's overloaded that takes one String and if you call it just with this String, it'll run this one. Then I have a Constructor, another Overload, I'm taking a String and an Integer and so as many times as I have different ones here, I can overload it that many ways. So you can see, I can get a lot of power here on how I create my objects, what comes into being with each one of those objects and I can do all kinds of cool stuff. Now let me show you one more thing here that I can do and I don't know if you need this or not, but notice I can create another object called Mark and I'll call this one J and set it equal to a new, a new instance of Mark and I won't call anything except Test This on this one. And you will notice that my J, the, the first one, I'm going to get two objects here, so you're going to see a lot of things fly around and so the first one that happens is the Mark Bob. So it called, let me move this out of the way, so that we can see it. Okay. So it called Mark Bob first and this Constructor ran. Alright. So let me get my form back. And soon as I go out of the way, Test This got called and then the Constructor Mark ran and Test This got called because we created this one. Notice Constructor Mark ran okay? So you can see I've created two separate objects and I'm using two different overloads on the creation of that object to determine what happens when it comes into being. Okay. Hope I haven't confused you, hope that helps, go through this a couple of times if you're lost, but Constructors add a lot of power and control on how our objects come into being when we instantiate them.

(Classes / Partial Classes)

Yet another tool that you have in your Developer Toolbox when it comes to dealing with classes in C# is the idea of Partial Classes. Now Partial Classes have been a part of C# since about 2005, I think C# Version 2.0 brought this out with the .NET Framework 2.0. But Partial Classes, let us build our classes in multiple files. So this gives us a lot of functionalities and this is also includes Structs and Interfaces, they can also be built this way as well. What we're getting at here is that I can have two or more source files, the first time you see this it looks a little weird and then it will click on you and see, you see the amount of power and functionality and flexibility that you have. So if I've got two Class Definitions and they both have the same Class Name and if they use both, use the Partial Keyword, then I can use as many as source files as I would like, give them the same Class Name, use the Partial Keyword in all of them and they all have to have the same Modifier accessibility. They all have to be Public, they all have to be Private or you'll get a Compilation Error when you try to use them. But as long as I meet all of those standards, then the Partial Class is going to be combined at compile time into one single class. Now if any parts of my Partial Class are declared Abstract then the finished class will be Abstract. Now we haven't talked about Abstract in this course yet, we're going to, so you can kind of look around, there will be a, a video out there somewhere called Abstract and Sealed Classes and we'll talk about that. Because again, just like Abstract, any parts of your Partial Class are declared as Sealed then the finished class is going to be Sealed. Now what are the advantages of Partial Classes? Why would I want to get into this business? Well think about this, I can have multiple programmers working on a Single Class simultaneously. Now in the old days we, you know, we resorted to all kinds of checking in, checking out files and you still do that a lot, but with this, I can be working on a class called Mark as a Partial, you can be working on a class called Mark as a Partial, somebody else can be working on a class called Mark as a Partial. And then when our application gets compiled, all of that magically becomes one class. I can also, this is a huge one, notice this one. I can add functionality to a class without recreating the source file. Now if I created a class called Mark and put it out there in my application and you come behind me, 4,5,6 months later and you want to add more functionality to that class, then you've got to go into my original file, change it and then put it back out there. However, if I built my original file, my original class as a Partial Class, then there's a file out there for that, then you could just create your own Partial Class with the name Mark and guess what? When it gets compiled, it looks as if they were all put together when it was originally done anyway. A lot of power, a lot of functionality, now this is being used all over the place, the .NET Framework and C# to allow us to extend functionality, to add additional functionalities, to overload things, do all kinds of really neat things, without having to actually go back and hack, if you will or change the original class. So Partial Classes are something you're going to see all over the place, as a matter of fact. If you just open any Windows Forms Application in C#, you'll notice your form itself is a Partial Class and again you'll see this everywhere. So I'll do an example of this in a separate video, you can look for a video entitled Partial Classes Example and you'll see this actually coded up and we'll work with it just a little bit. Very easy to get your head around this and very easy to use this to cause all kind of great, great functionalities in your applications.

(Classes / Partial Classes Example)

Let's take a look now at an example of using a Partial Class and I want to show you just what these things will do for you and the kind of power they will lend to your development. Now keep in mind, I didn't mention this before but Partial Classes are one of those things that really draws the ire of a lot of the Object Orientated purists. Object Orientated purists will say that this is not true Object Orientated Programming, that we're beginning to separate things back out and we're going to create problems. I don't want to get into that debate but just understand that it's out there and you will run into groups of people, other developers, that you when you mention Partial Classes, they won't like them. Okay. So just kind of watch for that sensitivity and notice that I have created a very simple Windows Form Application in Visual C# 2010 Express. I named this one Partial Class Example and you will see an example of this out there in your Work Files folder, there's not a whole lot of example here. What we're going to do, is we're just going to come down here within this Namespace and create a Public Class and we'll call it Mark. And we'll put our curly braces in and then I'm just going to put a Public Void Test This Method out here and I'll just put here that we would add code here. Whatever our implementation would be. Now notice to really make this happen, I can put the word Partial in here. Now this is telling this that there's other pieces of this elsewhere and I need to look for them and put them all together to make the class. Now this is Test This, now what I'm going to do is come over here to my Solution or to my Project, right-click and Add a Class, and notice this class is going to come up and I'll just double-click here. I'm going to call it Mark as well, I will add it here and in the name of the class here where I'm creating it, I will create Public Partial Class Mark, just like I did before and on this one, I am going to put a Public Void Test That. And then of course I would add code here later on. Okay. So I've got a Partial here, out here in this Free-standing Class and then I've got a Partial Class Mark here in my actual form code and so what I need to do now is just instantiate an object off of this class. And so I will create a Variable called B off the Mark Class of, of a Type Mark. Set it equal to a new instance of the Mark Class and notice that when I call up, not D, not N, but B, you'll notice that I see Test That and Test This. So they're coming from both individual, Test This is coming from this Partial Class out here in my event code and then Test That is coming from this Partial Class, that out here, that has a Free-standing Class in my Solution okay? So that's as simple as it is and again, be very careful. When you're starting to work with these Partial Classes you may, for example, wonder where they are. Well if I just right click on the Class Name and say Go to Definition, you'll notice it'll pop up, there are two matches found. And just by looking at the locations here, if I double-click, notice that one took me to Mark. And if I double-click here, this one took me into Form1 CES. So very easy to find where these pieces are and there can be a lot more than two pieces in various places out there. Alright. So anyway just be again, aware, some developers don't like this kind of thing, they think we're breaking the true Object Orientated nature but avoid those kinds of little skirmishes if you can. This is another one of those tools to put in your Toolbox and notice this let's developers work on this Mark Class all they want to out here and then at the same time it allows me to work on. I didn't mean double-click that. It allows me to work back here in my code on the Code Behind, on the Code Handling on my design surface to be able to provide functionalities and then both of these magically just work when it comes to compiling and running the application. So anyway, that's a real brief look at Partial Classes.

(Classes / Testing an Object Type)

Now let's talk about testing an Objects Type. It's amazing how many times in code you need to do this, it's also amazing how simple it is and so I want to save you some time here and show you how to do this in C#. What I've got here is a very simple solution called Testing Object Type. I will put this out there in the Work Files folder for you, although there's not a lot going on here. But I'm going to double-click the Button and while I say there's not a lot going on, it is amazing how often you need to do this but how irritating it is to go out and chase this down and then to see it was this easy. Well, let's first of all, I am going to just make up some classes here so we can check for the Types on those and I'll just do Public Mark and I'll just leave myself a note here to add some methods later. And I am just going to cheat and copy this thing and come down here and we'll do a Public Class. We'll do a dog and then we'll come down here and cheat some more and we'll do a Struct and we'll make this a cat okay? Nothing against cats or Structs. Okay. Then let's come up here and give ourselves some things to check. Now I'm just going to put this in our Button Click Event, just like all the examples that I'll do here. I'll put a form and a button and by the way this is a just a Basic C# Windows Forms Application and I'm using C# 2010 Express. So if I go back to the code in my Button Click Event, let's create some things. Let's create an Integer called X and set it equal to 5, let's create a String called A and set it equal to something irritating, like Hello. Then let's do, let's create an instance of Mark. Then let's create an instance of the dog and then let's do an instance of the Struct, Cat C equal new Xat. Okay so there they all are, let's talk about, how do I check these things? Well the first thing I want to do is I can check to see if something is a particular type and I use very simply the Is Keyword okay? And so I can say, If, If D Is Dog. Okay. And there you can see it, then come down here and give me a Message Box and just say, The Type Is. Then I'm going to do another thing to show you kind of a way, to go the other way. I can call the Get Type Method, I can call the Get Type Method and it will just report the Type as it sees. So this will check to see is it a dog and then I can use Get Type to, without even checking, it'll just tell me what it is alright? And then this is going to be a little bit clunky but I want you to see how we can do this with these things. Let me drop down here and I will just put another Message Box that says, Incorrect Type. Alright, so if D is in fact Dog then it'll report this out and it will show me what type it is and verify what I'm looking for, if not it will show Incorrect Type. So let's just run this and you'll see the Type is Mark. Okay. And that is because, notice what I did. Okay. My Get Type here doesn't match, I told you this was clunky. Alright. So I've got to match D. Okay. If D is Dog then D Get Type, so let's go back and run this and you'll see the Type is Testing Object Type Dog. So this was true, D is Dog and it caused the Message Box here to fire up and it's reporting it back to me. So again, hope you see the significance here, if there is any significance I guess. This Get Type called on your object or, or whatever your variable, whatever, will tell you what it is. This will check the condition to see if it is that. For example, if I want to check to see if X is an Integer, then I can run that test and notice that when I execute this, it'll come back and say is the Type is System Int 32. So it was an Integer and it returned this to me and it reported the Type back to me. Same thing, you see where we're headed here, you see how this is going to work. If A is String then let's call A and you see where we're going here. Now let me show you something here, we checked for the types on the dog and cat and so forth but I can check to see if D is Mark. Okay. So if D is of a Type Mark then report that to me. Okay. And you're going to notice here that this one's going to kick back Incorrect Type because it checked D is not a Type of Mark, D is a Type of Dog. So if I'll put dog in here and I don't how many have gone to sleep yet but this is deceptively simple but it is amazing how many times you need to verify this stuff in a program and it's very, very easy to do. The Is Keyword is how we check to see what Type a Variable is and the Get Type is what will just report it back to us in String form. So anyway, hope this will help you and save you a couple of minutes one afternoon.

(Classes / Abstract and Sealed Classes)

Abstract and Sealed Classes are yet another option you have when you're building Classes and I just want to mention them to you briefly in this video. I'm not going to go too deep here but you may see them later on in some examples. But at the, the worst case scenario here, go out and read about Abstract and Sealed Classes and take a look at them. Because when we get into Inheritance and I'm kind of steering away from Inheritance a whole lot in this course, to delve into some other aspects of C# but with Abstract and Sealed Classes it gives us the ability to construct classes, to use them in an Inheritance Model to give us different functionalities. If we build an Abstract Class what we're really doing is, we're creating a class that's intended only to be a Base Class for other classes. In other words, other classes are going to inherit this class. Alright. And it's going to pull whatever we put in this Abstract Class, it's going to pull that information down into the Inheriting Class. Now Abstract Classes cannot be instantiated, now that's very important. If I build an Abstract Class, I never intend for it to be instantiated as an Object, really what I'm doing is building a model if you will that I want other classes to implement. Now this, I may have Methods in there and by Abstract, I really mean there is no code, there's no functionality built in, it's only naming the Class and it's naming the Methods and so forth inside that class. Now I can also mark an Abstract Class as Sealed which means it cannot be inherited itself alright? In other words, other things can inherit from it and it's used as a Base Class but it can't be inherited by other classes if I mark it as Sealed. Now any Derived Classes, any classes that inherit from the Abstract Class that I create must include functionalities and implementations of all Abstract Methods within my Abstract Based Class. In other words, if I have three Methods in my Abstract Class and then I inherit that class then my Derived Class must implement all 3 of those Abstract Methods in that Abstract Class. Until I do it's not going to compile, it's going to throw errors and so we're going to have to override those Base Class Methods using the Overriding Keyword and put functionality in there before we can get a good compile. Okay. So an Abstract Class is something we're going to use to force you to make sure that you get all the functionalities that we want in there and if you inherit from our Abstract Class, you got to do it. Now a Sealed Class, I made mention to it a few seconds ago, earlier in this video. A Sealed Class is simply a class that cannot be inherited by other classes. Alright. If I've got a class and I want to build a class that has to be used as I've built it, you can't inherit it, you can't add functionality to this class, I don't want you changing anything, other than maybe through overrides or something, but you cannot inherit this class and just pick up the functionalities. Now these are out there in the .NET Framework, there are times you'll use them, I don't want to go into a lot here. Just know these are out there, know you can use them and this might be a great place for you to go do a little more reading if you're not already familiar with these concepts but I did want to touch on them because you're going to see these things but they're Abstract and Sealed Classes. Two more designations we can make to our classes to help us build the Object Formation that we want and the Class Structure that we want to provide the functionalities that we require in our applications.

(Classes / Inheritance Example pt. 1)

Now let's take a look at a real simple example of Inheritance. Now I talked about Inheritance way back toward the first of the course when I talked about some of the Object Orientated Programming basics and in that I explained generally what Inheritance is. And in this video I want you to see a good example of Inheritance using 3 classes and some of the examples that I was talking about back in the beginning of the course. So let's take off with this and this is going to be a two parter. So in Part 1 here, I'm going to kind of explain the rationale and show you how we're going to set this up and then actually demo some of the functionalities as we get on toward Part 2. Well first of all, the idea of Inheritance is one of the fundamental aspects of Object Orientated Programming and what we do when we start to design an application using Object Orientated Programming Techniques, is we want to determine what real world objects do we want to represent in code. Well let's go back early in the course, I was talking about this fictitious veterinarians office. A bunch of animal doctors if you will and we want to start to think through the process of building an application for these veterinarians. Well when you think about what they do all day long, they treat animals. They may treat animals that are sick, that are injured, just standard check-ups, physicals, whatever you call those for animals. I guess you would call it a physical and so we want to start to think through, okay, what objects would they need? Well the first one that comes to mind is just a general Animal Object. So I would create a class, a Public Class Animal and if you think about it, any animal that comes into their office is going to need certain things. Okay. One of the things we're going to need to know is who owns this animal. Okay. Who's going to be responsible for paying the bill, is the better way to say that and I'm just going to put Public Fields here. Now we would do Properties in a real world application but I'm going to make it as simple and easy to read as possible here. Okay. So what I'm going to do is, we're going to say Public String Owner First Name, then Public String Owner Last Name. Alright. And that's not a string that is a String. So that's going to help us, every animal's going to have an owner. Okay. Then we're going to think about, okay, every animal is going to need a Void, we're going to put a method in here for Physical, Physical Exam. So first thing you're going to do when you bring your animal in is do a physical exam on it so that we can determine if the animal is okay, what's the problem? If it's injured, if it's acting strange, if it won't eat, that sort of thing. And so I would put me a Message Box or something in here for now and I would come back and write some more code a little bit later on as I determine exactly what that code should be okay? And so we'll just kind of stop it right there but anything that has to do with animals in general, we're going to put into the Animal Class. Now let's think through this thing just a little more and think, okay, also people will be bringing their dogs in here. So let's create a Public Class called Dog, not Doug and let's think about the things that are unique to dogs. Now if I'm not using Inheritance then I need to copy these fields and I need to put those in Dog as well, if I can copy these, I can place them down here. Then when I instantiate an animal, let's say if I create a Type of Animal called A and set it to a new instance of an Animal. Don't you love to watch people type who are trying to talk at the same time. Alright. I apologize for my clunky typing. Notice on the Animal Object A and when I instantiate it, intellisense shows me, there's my first name, owner last name and physical exam and their are all those three things in Animal. Well, when I get ready to do a dog, when I'm wanting to work with somebody's dog, I can go to the Dog Object and notice now I'm instantiating A to make this not confusing, I'll call dog a D and notice I get owner first name and last name and that's coming from down here. Well with Inheritance, I can avoid having to type all this information over. Okay. So let me take this away up here and now instead of adding those same things in Dog, let's just think about what's unique to a dog as oppose to an animal. Well, one thing, a dog is going to have a particular breed. Alright. So we'll create, this would be a property, but we'll create a field for a dog and we will call it Public String Breed and then what else happens to a dog? Let's do something very appetizing here; I hope you're not eating as you're watching the video here, if you are, warning,. Okay. You might want to pause this, come back later after you finish the sandwich. Public, we won't do any returns here; we will do a Heartworm Treatment. Okay. So Heartworm Treatment on the Dog and we'll just put a Message Box here that says, you know, Heartworm Treatment so that we know that we're hitting the right place and then let's think. One more thing that people could have is a cat and so let's stop right here at Part 1 and we'll come back in Part 2, we'll add the Cat Class and then show you how we're going to use Inheritance to make magic happen here. So join me in part two.

(Classes / Inheritance Example pt. 2)

Welcome to Part Two of the Inheritance Example video and I'm just going to continue right on where we stopped in Part One. And if you remember in Part 1, we had created the Animal Class with the owner first name, last name, physical exam Method. Notice it starts off as a, function here that doesn't return anything, becomes a method when an object is instantiated off of this class. Then I created the Class Dog and it deals with things that are unique to dogs when they come into our veterinary practice. We want to keep up with the breed of the dog and then of course we want to do things like Heartworm Treatments and it would be more in our simple example here, we're going to try to keep as simple as we can. Okay. And so let me take off some of these lines that we don't need here so it's a little easier to read for you and now let's go down, let's think about cats. Okay. When we cat in, there's going to be some things about cats that are going to be unique from animals in general and especially from things like dogs and the obvious one here is for a cat, we want to put a method in here, we won't return anything for now to keep it simple but we might want to declaw a cat. And I understand the strong feelings that a lot of people have about declawing cats and it is something to think about, you're taking away their defenses and if you declaw one, you need to be in position to provide a safe home for it forever. So anyway, all that aside, can you tell I'm just kind of talking while I'm trying to code here. So Declaw Cat. Okay. But it is important if your a cat lover. Okay, now so I've created these 3 objects or these 3 classes and so if I create an Animal Object called A and set it equal to a new instance of Animal. Right. I get everything I see in animal. Alright. If I do that with Dog, I'm just going to change all these to Dog and I'll change this to D to make it a little easier on you, notice I get everything I see on Dog. Well, let's do something totally different, I'm going to kill that and now I want to tell Dog to inherit Animal. Okay. And I probably need to spell that correctly, I'll double-click and it will correct it and so now notice this colon here, is saying the class is Dog and we are inheriting from Animal. So Animal is the Base Class and Dog is going to be the Subclass. Now if I go back up here and if I create a type of dog, Variable Name is D and I set it equal to a new instance of a Dog Class and I hit D, notice I'm getting the owner name, the first name and the physical exam from the Animal Class. Alright. Then from the Dog Class I'm picking up breed and Heartworm Treatment. So by creating one Object here, Inheritance is giving me the functionality from both Dog and Animal. Okay. I can come down to Cat and say Inherit Dog. Now this is kind of weird that I would do this. Okay. Because I'm picking up things from Dog that I might not want but I just want you to see something here. Let's change this to C and I have to change this but it makes it a little easier. I can't stand examples where they keep using the same variable names over and over. So notice now I'm getting breed from Dog, I'm getting physical exam from the Animal, I'm getting owner first and last name from Animal, Heartworm Treatment from Dog and Declaw from Cat. Okay. Notice if I mouse over these and just click them, notice this is telling me this is the Void Cat Declaw, this is the Dog Heart Worm Treatment, this is coming from Animal Owner Last Name. And so generally what you're probably going to do in this situation is just inherit from Animal and so what you can do now, obviously, when you start to think through building an application, you want to think in terms of, okay, what can I put most generally in the Base Classes? And then Inherit functionalities to add more specifics but then inherit these base functionalities into my more specific class and it saves a lot of code. It makes your Object Structure very logical and makes it very easy to update this application and so forth. So that is the simplest possible example of Inheritance that you're ever going to see. Now Inheritance opens a whole new world of possibilities for doing some really good things and for getting really confused. So you'll need to dig into Inheritance a little bit to start to see the power there but this is something you really need to get comfortable with. So that is a Two Part Inheritance Example.

(Classes / Static Classes)

Next up on our tour of classes and class functionalities is the subject of a Static Class. Now Static Classes do some very, very convenient things for us but they have to be used in the correct way or they can really create some issues for you. First of all, what is a Static Class? A Static Class is a class that you can access and you can use all of it's methods but you don't have to create an instance of the class, you don't have to create an object to use it. You don't have to go through all this, you know, Animal A equals new instance of Animal, you can just put the name of the class, a period and whatever Property Field Method you want to run and it's there, it's available for you. Now the way you do this is when you create the class, one way that you do it and I'll talk about the other one later in this video, just put the word Static in front of the Class Definition. Now I will do an example of creating and using Static Classes in a separate video so you can look for that video, it will be entitled, very cleverly, I might add, the Static Class Example but we'll get to that a little bit later on. So to create one, I'm just going to say Static Class Mark and then follow it with all my implementation and that will give me a Static Class. Now when I declare a Class as Static when I create it, it means that I can only put members or methods in there that are marked Static themselves. If I try to add a Non-Static or just a regular Member to it, it's not going to compile. Visual Studio's going to throw me a Compilation Error and say wait a minute dude, you got a Static Class here, all your methods have to be Static as well. This also means that this class cannot be instantiated as an object and it cannot use Instance Constructors but we will talk about Static Constructors in a separate video, just hold onto that one. Now when I mark a class as Static, it also makes the Class Sealed meaning it cannot be inherited. You can't inherit it, you can't instantiate it, you just have to use it in it's form that you created it. Now there are two ways to achieve Static Class functionality, let's take a look at those. The first one is to just simply declare the class as Static, just like this, Public Static Class Mark, that does the trick. Then I'm going to add my curly braces and add my functionalities. I have to declare every method as Static, I've already mentioned this, if I try to do a method that's not Static I get a Compilation Error. Now there's another way to do this and this one is somewhat more dangerous. I can just declare my class normally, Public Class Mark and then I have to make an effort to realize and remember to declare every method that I add in that class as Static. Okay. So Public Class Mark, then Public Static Void Test This, Public Static Void Test That. Here's the problem with this, if I leave this Static Keyword or Declaration off of this method, this method now allows me to instantiate an object of this class to get to this method. So now I've got a class that can be used as a Static to get to this method and then has to be instantiated to get to this method, that might be what you want, it might not be what you want. You just have to think this through and make sure that if you this kind of stuff, you document it in your code so that somebody coming back later can figure this out very easily. It is possible to do that, is it what you want? That's kind of up to you, generally you want to limit your class functionalities either Static or Non Static, but again there's plenty of developers who will fight with you on that one as well. So anyway, that's the quick run through on Static Classes and the way to really get these things is to see them in example and so in the next video we'll do a Static Class Example for you.

(Classes / Static Class Example)

Now let's take a look at an example of building and using a Static Class. What I'm going to do is I've got my same old easy to follow example environment here, I just created a Static Class Example Project as a Windows Forms Application in Visual C# 2010 Express. Dropped a button on my form and I'll use that Button Click Event to kick off my actions back here. So what I'm going to do is go down, stay within the Namespace and I am going to create a Static Class. So I'll say Public Static Class Mark and then I'll put my curly braces and what I want to do now is create a Method. Okay. So a, actually let's do a Field here. I'll do Public Static Integer Weight. Okay. That's always a popular topic around my house and then let's do a method here, I will choose Call VTC. So there's a method here to Call VTC and so whatever I need to have happen when I Call VTC, I shouldn't put that semicolon up there, don't pay attention to that when I'm talking. Okay. So Call VTC and I'll just put a Message Box out here, Show Called VTC. Alright, those should have been periods and that should have been a semicolon, other than that, it was pretty close, you got to admit. Now notice this is a Public Static Class so how do I use this? I don't have to go through any instantiation or anything, I just say Mark hit the period and notice there is Weight. I can set Weight equal to 145, yeah right, I think I weighed 145 in the fourth grade. Then there's Mark.Call VTC and then notice if I run this, you won't see anything happen with the Weight but you will see Call to VTC, it will execute that. Okay. So that's Static. Now what happens here if I try to create a Method, Public Void Bob, don't know what Bob's going to do here, put in my squirrly braces, my curly braces not squirrly braces. But notice what's going to happen here, if I try, since I've marked my Class Static, if I try to compile this, it's going to say, whoa dude, you got some errors here, what is it? Notice Bob, you can't declare an Instance Member in a Static Class and that is exactly what that is right now. Okay. And so now I am going to just copy a Message Box down here and change this to Call Bob and I want to show you what I was talking about in the video that was entitled Static Classes. If I take the Static designation off of here. Alright. And compile that, it will run. Okay. Everything's fine, notice my code up here works because Weight is Static and Call VTC is Static. Okay. But notice that when I hit Mark and hit the period, it doesn't see Bob because Bob is not Static, you have to instantiate this. Okay. So I will do Mark M equals new instance of Mark and notice on M now, I can only see Bob. Okay. Now both of these will run but you got to be careful so I can, I can either do a Static Class a couple of ways. I can make sure that all of my Fields, Properties and Methods are Static or I can name it as Static and it will keep me from putting any normal standard instance type members in here. If I want to mix these up for any reason, I can leave Static out of the Class Declaration and then mix these as I would like. Just be careful that you don't lose control of this and you're really going to start to make your programming very confusing because you have Mark here and Mark here. But Statics are a great way to handle a lot of issues that come up, do keep in mind when it's a Static Field Property like that, since there are no instances, any time I change Weight here and look at it anywhere else in the code, I've changed it anywhere I'm going to look at it. Okay. So anyway, that's kind of the low down on a very simple example of using Static on your classes and on your methods. And in the next video or two, I want to dig into Constructors on Static Classes because this is even more interesting and it can also give you a functionality that you can use during those special weird times that you always seem to bump into in the course of developing applications. So anyway this is a good, quick, simple example of using a Static Class.

(Classes / Static Class Constructor)

And now let's talk about a Static Constructor. Now this is one of those things that's kind of out there, you don't hear about it a whole lot but it's very unique and the best way I can describe a Static Constructor is kind of like one of those really bizarre tools that an auto mechanic or a truck mechanic has way down in the bottom of their toolbox. So they're working on something really kind of unique and all of a sudden they pull out this really strange looking tool and it does just exactly what they need. They'll put it back in the box and it could be weeks or months before they ever need that thing again but it's a lifesaver when they need it. That is kind of what a Static Constructor can turn into. Now a class can have a Static Constructor now notice I didn't say a Static Class, it certainly can, but any class can utilize both a Static and a Standard Class Constructor and I can hear some of you saying, okay, what and why? And let's answer those. First of all, what is the characteristics of a Static Constructor? Well a Static Constructor is called only once. Now I did a video somewhere else in the course, you can go look it up, called Using Constructors and Understanding Constructors. If Constructors are not a kind of a natural thing to you at this point, I would recommend you may want to go watch those two videos again before you try to dig too far into Static Constructors. But if you remember an Instance Constructor gets called every time a instance or an Object is instantiated off of a class. A Static Constructor is called only once, it's only called when the very first instance is created from a class or when the first Static Member is referenced. Doesn't matter if it's a Field, a Property, a Method, so whenever that first Static Member gets referenced, boom the Static Constructor's called and it will not be called again to stop the program and restart it. Okay. Now you cannot directly call a Static Constructor, it has to be called as a result of accessing a Static Member for the first time. You can't give your Static Constructor any Access Modifiers, you can't throw Parameters at it, which means in English, we can't overload a Static Constructor. But again this capability of having a Constructor, that's called only once, it's called automatically. However let me add one caveat here, I didn't put it on the screen but you have no control over when exactly that Constructor gets called. If other things are going on, it could be called in a little bit different order than what you had hoped and you can't control that. Okay. But anyway a Static Constructor is a unique item and the best way to understand them is to look at it in a code example and I'll do that in a separate video. So you might want to go out and look for the video Static Constructor Example and you'll see in this code and see it in action and I think it will make the light come on for you, so join me there.

(Classes / C# Structs)

The first time you encounter a Struct in C#, it's going to kind of take you off guard and you'll kind of wonder, like everybody does, wait a minute, just exactly why do I want to use this thing? And so let's talk about Structs here and then we'll take a look at them in an example code in a separate video. First of all, a Struct is a close cousin to the class and as a matter of fact it's a very close cousin to the class and the main difference between a Struct and a Class is a Struct is a Value Type, a Class is a Reference Type. Now if you understand what a Value Type is and a Reference Type, I've told you just about all you need to know about Structs. If you don't understand then let's keep going. Structs share most of the same syntax and functionalities of the class. In other words, the way you set up a class, the way you manipulate a class, the way you use a class is going to be very, very similar to what you do with a struct with some minor differences. So what are those differences, what is different about a Struct as compared to a Class? Well there is no Default Constructor in a Struct and let me just make, maybe this will help you understand it better than anything I could say. Think of a Struct as a Class that behaves like a Standard Variable. When we create a Struct it's a Value Type, it just exists, all the data stored right there in it and it doesn't come into being like an Object, it just already is there. Okay. So there's no Default Constructor, you can have a Constructor but it has to be Parameterized, it has to look like what we would call Overloaded Constructor in a class and it can be instantiated without using a New Operator. Now that's kind of the weird thing about Structs, you can either do them just like a class and use the New Operator or you can use them without the New Operator. If you use it without using the New Keyword or the New Operator in instantiating the Struct, then if you don't use that New Keyword, it will prevent the Constructor from being called at all. So this is kind of interesting, I can use this like a class, it looks and feels like a class but depending on how I bring it into action determines whether or not that Constructor gets called. Now you cannot inherit with Structs so you cannot inherit from another Struct or a class and now the question, okay, so I got a little hint of an idea about what a Struct is, why would I use a Struct instead of a class? Again, a Struct is a Value Type. Generally we can create Value Types and destroy them with a lot less overhead than Reference Types. Keep in mind we're talking milliseconds, it's not a huge deal with today's machines and memory. However, it's still there. So generally, you'll see things like creating a Lightweight Objects and these really aren't true Objects that we think of when we instantiate an object off a class, but it is a Lightweight Object. It acts like an Object, it feels like an Object, it smells like an Object but it's a Value Type, not a Reference Type. Then they can, I've already mentioned this a couple of hundred times I think in this video, they can be more efficiently created and destroyed because they're a Value Type. Now here's the general rule according to Microsoft documentation. Now a lot of people will disagree with this and rightly so. The time to use a Struct is when it's best. Okay. How do you determine that? Well you'll just have to be there. Okay. General rule, you want to use a class when you're trying to manipulate data that's intended to be modified after an Object is created. So if you're going to create the data and modify a lot, probably you want to use a Class. Use a Struct for data that's not intended to be modified after the Struct is created. Also a class will let you do things because it's a Reference Object, if you copy it, if you make a copy of it to another Object Type, you're only making a copy of the Reference Pointer. Okay. Not the underlying data and it, with a Struct if you make a copy of it, you're actually copying the Object and the data and everything. Now generally people use Structs when they're working with things like circles, rectangles, those sorts of things, generally things that you create one time and then just use but you don't manipulate them a whole lot. Okay. Again, those are very general rules, if you're not familiar with Structs, you need to get familiar with Structs. Okay. So this is kind of the real short version, chalk talk, to kind of give you some information in your brain about what these are, how they're different from classes and we will do a Struct Example in a separate video to let you see one of these things come alive in code.

(Classes / Struct Example)

Now let's build a simple Struct and just take a look at it. I want you to see what it looks like in code and then in a separate video, we're going to talk about a bit of an oddity about Structs and that is Constructors and the New Keyword and so forth but for now we're just going to make it simple. You'll notice I've created a, a Windows Forms Application, I named it Struct Example and probably won't put this one out there in the Work Files because it's very straightforward just like a class. So we want to create a Struct, notice I'll create a Public Struct Mark, notice this is the exact same thing as creating a class. I just put the word Struct instead of Class and then I am going to put Integer Weight, I'll do a Field here, won't spend a lot of time with the Property and then I'll put a Public Void Test This Method here and I'll just put a Message Box.Show Hello. Pretty straightforward. Now notice when I instantiate this, I just call Mark M equals New, Mark, everything looks just like a Class. When I call M there's Test This and everything's pretty straightforward. Okay. And so when I run it, everything happens, now what's the big difference? Well I would refer you back to the video entitled C# Structs to start to lay out some of the things that we can do with Structs, why we want to use them, they're a Value Type, instead of a Reference Type and in the next video. I split these up because I didn't want to go over on time and bore you to death with too much information in one video. But in the next video, we're going to talk about Struct Constructors, so go look for that video and watch that one and we're going to dig just a little bit deeper into Structs and how they behave.

(Classes / Struct Constructors)

Now let's concentrate on Struct Constructors and this is where Structs really differ from classes in a pretty major way and you'll notice that I've, in a Solution here in a project name, Struct Constructor Example. You will see this in your Work Files folder and I've got a Form and a Button and a, just a straight up Windows Forms Application project in Visual C# 2010 Express. Now what I want to do is go back here and create a Struct and you'll notice this looks just like a Class, Public Struct, S Mark and notice no real difference in creating a Struct and a Class. Now here's the first thing that I want to show you, I cannot create a Constructor that does not take Parameters in a Struct and I'll just copy a Message Box and drop it in here for you. Okay. Because and I'll even take this X out so that it doesn't confuse anybody and I'll try to get this thing working. Okay. Now if I try to instantiate this S Mark SM equals new instance of S Mark, if I try to execute that, it's going to start to squawk and it's going to say, wait a minute dude, Structs cannot contain explicit Parameter List Constructors. Okay. So the first thing is, that's a no no, I can't have a Constructor that doesn't have a Parameter. Okay. So what can I have? Well as it turns out, I can jump straight into a Constructor that does have Parameters because I can call Public S Mark and pass in a String that's X and now this Message Box that I had appear, let me go grab it again, hang on, I made some changes to it there. I can put it right here and then I can get that. Notice if I call SM New Mark and pass Hello, then as soon as this thing instantiates this, it's going to, see this Public Constructor, it will pass in Hello right here and I'll get my Message Box. Okay. So I will run this and as soon as I click this button, there it is, S Mark Struct initialized with Hello and there is where that came from. So I can use a Parameterized Constructor and that's not going to cause me any problems whatsoever. Now there is something else that's interesting about these Structs and that is, I want you to notice that this will work. I'm going to take this out and put something a little different in there. S Mark space SM and then just call SM and I will need to give myself a Method out here, so let me just put in a Public Void Exercise, forgot to do that and out here I'm just going to drop a Message Box in here to show me that this is working and so I will just drop this in here, paste it in and so now, notice I can call S Mark Exercise and this will work. Now notice a Struct, this is not Static or anything but I can just call it straight without using the New Keyword and that will work, that will call that Exercise, notice Struct Mark S Mark running. So this is kind of some of the oddities, if you try this with a class, if I change this Struct name here to Class. Okay. And try to run this, notice immediately I get a squiggly up saying, wait a minute that's a use of an Unassigned Local Variable SM but if I change this back to Struct not a big deal. That has to do with the Value and Reference Types and all kinds of other things and we won't go into that right now but that's just kind of some of the intricacies about using Structs and Constructors. You can't use a Parameterized Constructor like you can with a class, you can however, use a Constructor as long as it has at least one Parameter in it and you can create an Object of a Construct using the New Keyword or leaving the New Keyword out. So lots of things that are a little bit funky about Structs, I would strongly encourage you to read these and play with them and determine when you would like to use them, when it would better to use them. And this is one of those things that you can get a bunch of nerds in a parking lot fighting really quick about and that is when you should use Structs, when you should not use them and so forth. So anyway, that's a quick look at the topic of Struct Constructors.

(Fields and Properties / Understanding Fields)

Fields are a very important part of the Class Definition because a field as you can see in the simple definition here is a variable that is declared directly in a class or a struct. Now usually you'll concern yourself with these in classes but understand they work for structs just as well. Now another way to think of these fields is that they are also known as Class Level Variables. These are variables that we just declare in the, at the Class Level, they're good in every method throughout the class. Now when I create these they become what's called Instance Fields because if my class is not static, then that means I'm going to create multiple instances of that class. And if I create one of these fields at the Class Level or a variable at the Class Level, then that variable is going to store unique data, specific to every separate instance or object that I create using that particular class. Alright. Now keep in mind if we have a Static Field up there, it's shared among all instances of a Class and changing it one instance will also change it in all the other instances. So the best way to understand fields is that I can create at them at the Class Level, they're available to everybody, they will be unique to every instance. If I make them Static, they're going to be shared everywhere. Now having said that, they're are specific ways that you want to use fields because they can introduce some pretty serious problems and challenges to your coding and we're going to look at those through the next few videos here as we dig a little deeper into fields, properties and that whole world. So the best practice on fields, fields should be used for variables that have Private or Protected accessibility. The problem is, if I just put a variable out there, that anybody can set without any kind of checks on my part, they can introduce errors. For example, if I have a field out there that's Public called Age and I'll allow you to put an age in there and instead of putting 39, you fat finger that and put 93. Well that could create problems in my database and it could cause that particular person to not show up in searches, it could cause all kind of problems. So a field doesn't let me check the values or verify them, so I don't want to do Public Fields, I want to do fields privately so that I can only access these things from within my code and then I will expose that Private Field out as a property and then I can write code. I'll explain all that and demonstrate it a little bit later on, but most of the time, we're going to use these fields in Private or Protected Accessibility Levels and in that instance they're called Backing Fields a lot of times. You'll hear them called Backing Fields because they backup Public Properties but they give us a chance to verify what's going into our properties. Public Fields generally are not recommended, you should always get a little nervous when you start to use Public Fields in your classes. Alright. Now I've used them in examples in a couple of places here to simply make my code very clean and easy but in a real world example, you don't use them, but as we step through these next few videos about fields and then we get into properties, you will see the correct way to use fields. Okay. So the first thing, understand what a field is and then as we move through here you'll understand the relationship between fields and properties.

(Fields and Properties / Defining Fields)

Now let's talk about actually defining our fields and some of the options that you have. Now as a quick review the field's going to be the foundational piece of information often times within classes. Then we're going to use these fields to build and provide access to properties and we'll talk about those a little bit later on in some separate videos and then show you some coding samples. But notice now, I've just got the Basic Windows Forms Application 1 out in the Visual C# 2010 Express, got a button on my form and I'll just double-click that button and jump back into the code. Now we're going to use these fields within a class and the best way to think of these things is they're a variable that belongs to a class and so if I create a class here called Mark. The first thing that I want to do is, I want to build a variable that can be unique to every object that is created from this class or every object that is instantiated from this class. So I'm going to create a Public Variable, we'll talk about Access Modifiers a little bit later but I, I will call this, this variable that's going to be an Integer Data Type and I will call it Weight. Now I could have called it X, Y, whatever word I want to as long as it's not a C Sharp Keyword and it'll warn me about that if I do and it's as simple as that. Now when I create an instance of this class, the magic of correct typing, notice I can see Weight here and so at this point, I can set Weight equal to 5 and then I can come down here and I can create Mark N equals New Mark and I can set N Weight equal to 6. Now what I've done here is I have created two individual, wait a minute, I've got, this is case sensitive and I've got to get that right. Now I've got two individual Objects, M and N. Now M sees the Weight as 5 and N sees the Weight at 6. Alright. I could do a little Message Box to show you that but take my word for it. Okay. And so that is going to give me basically what's called Instance Level Variables or Instance Level Fields. Now like all variables, these fields that I've created, just like you see here with Weight, these can be written to and they can be read from. Now what you're looking at right here is really kind of the minimum syntax, actually the minimum I can leave Public off and now I'm getting squigglies because it can't see this, because it now sees this as Private because it defaults out to Private. Alright. And so, that's, let me take those off and let's talk about what we can do to actually define our fields and there's a bunch of ways to do this. There are some things that you might see out there that will tend to confuse you but let's talk about some nomenclature here. The Field Initializer, what you've done here is, you've basically declared a field but the Field Initializer is anything that follows an Equals sign and let's say that I can assume that the Default Value of this field should be 100. And so I say Integer Weight equals 100, now what I've really told the Compiler, I've got a Private Level Field, it's the Integer Data Type, it's called Weight and it's equal to 100. This equal 100 is the Initializer, that is the Field Initializer. So if you ever read that, just know that, that's what we're talking about. Now what I'm doing here is I'm exercising or, or carrying out what's called an Explicit Field Initialization because I told it what it would be. Alright. And I've told it exactly what value it should be. If I do it this way and don't give it an, an initialized value then I am doing an, Implicit Field Initialization. Now what the Compiler is going to do here is, it's going to say, wait a minute, this clown didn't exactly tell me what Integer should be stored in Weight so I'm going to default that to 0. And if I do a Boolean, it's going to default that to false, the default the any Reference Types is always going to be null. Okay. And you can find out some more about that a little bit later on in some different places in the class here but anyway that's just kind of a short run up to that. Now I can also, let's say that I want to create two Integers. I want to create an Integer A. Alright. And then I want to create Integer B and I want to set it equal to 25. So I've got an Implicit and an Explicit right there. Now I could have done that this way, I could have said Integer A comma B equal 25. Now in some languages out there, this would set A and B both equal to 25 but in this instance, B is equal to 25, A is implicit and it's value is 0. So just watch for that, now you generally won't see it, a lot of people won't write it this way, because it tends to get confusing for people. You will normally see them written out like this, line by line but that's pretty much the bottom line on Field Declarations and again these are at the Class Level. You'll see them defined outside of any methods or anything and these are normally used, we'll create these things as Private and use properties to read and write to that private value but you'll see that when we get to properties later in a separate video. And I think that video is going to be, well you can take a look at Understanding Properties and Creating a Property to see that. But that's kind of the bottom line on Declaring Fields in your classes.

(Fields and Properties / Field Modifiers)

Fields accept Modifiers just like classes do, just like methods do and each one of these, I'm going to go over these Access Modifiers and it's going to feel like I'm repeating myself but you will notice there's little nuances. Just a couple of little different options on each one of these and you need to be aware of these because this is where you really get your power and your control when you use these. Now when you're Declaring Fields for your classes, you need to understand that there are Eight Access Modifiers that you have at your disposal. There's New, Public, Private, Protected, Internal, Static, Read Only and then Volatile. Okay. Some people may call it volatile, however you want it, potato, potato. Okay. But when you're dealing with these Modifiers, you'll notice that we continue to see the same ones over and over, Public, Private, Protected, Internal and these kind of become the, the general stable that we use a lot and also Static probably. But being able to utilize these things will help you not only write safer, better code but will also help you troubleshoot code when you come back. Now let's go through these for our Access Modifiers at the Field Level. The New Modifier indicates that a field in the Base Class is being hidden. Now when we get into the world of Inheritance, we may have fields in a Base Class with the same name as a field in a Derived Class and the New just simply tells the Compiler, look I know this is going on and I'm using the correct one, so don't worry about it. Okay. Now the Public Field Modifier or Access Modifier simply means just like on everywhere else you see Public as a Modifier, whether it's the Class or the Method Level, the field can be accessed without restriction from any other class. Just it's Public, it's out there, and this by the way is considered dangerous programming most of the time with classes. A Private Modifier means it can be accessed only from within the Containing Class so if I've got a Private Modifier on a field and that means I can only read or write to that field from code within that class itself. Now a Protected Modifier means that I can access it either from within the class that it's in, the Containing Class or any Derived Classes or any classes that inherit from this particular Containing Class. Next up is Internal, Internal can only be accessed from the current assembly but keep in mind we could have multiple classes within the current assembly. So we're talking about having build our application, it's all been packaged up into an assembly, anybody in that assembly can access that particular field. Static, this is, we talked about Static in a set of videos entitled Static Classes. We did a Static Class Example, Static Class Constructors and then we'll talk about Static and Methods later but this just simply means it's not instance based, it belongs to the class itself, which means, that you're not going to have unique values for this for every object because this never leaves. It stays right there with the class and you can access it by just using the Class Name. and the name of the field. Now Read Only is kind of unique, this one can only be initialized when it's first declared or you can use the Constructor. Now that's something that's kind of unique with the fields, I can also initialize my Read Only Fields via the Constructor. So when my object comes into being I can set those fields through the Constructor if I'm set on Read Only. Now Volatile, this is just indicating that a field might be modified by multiple threads that are executing at the same time. So these are your 8 choices, you will see these, you will normally see your Public, Private, your Internal, Static and Read Only. Volatile and New you tend to not see that much but you could see them out there and I just wanted you to be aware of them.

(Fields and Properties / Understanding Properties)

When it comes to Properties you're going to find out that they're very close cousins to fields and so I'm going to start of with a, a really old joke, you know, I crossed a field with a method. Okay. That's not the joke but that's what we're doing here to get to the properties. The old joke is, I crossed an elephant with a rhino and of course what did you get? Hell if I know. Okay. So it has nothing to do with C# but I hope you enjoy it. I crossed a field with a method, what did you get? Well I got a Property, that is exactly what a Property is. A Property simply mediates access to fields, so when you're looking at a Property in a class, you are looking at fields, you're just looking at a piece of functionality that's forward facing to the user, to the, anybody outside the class that let's the programmer control what happens to the fields that are used. Now what we're actually going to do is we're going to set Private Fields that actually hold the data and then we're going to expose Public Fields out to the people who consume and use the class and so the Property is going to look like a field to out there, to that end user. Now it's going to act just like a method, anytime the Property gets accessed, some code's going to fly into place and it's going to do whatever we want it to do. So the Property gets accessed just like a field, the Compiler then does some magic for you and the magic comes through something called Accessor Methods and we'll talk about those when we jump out and do these in a code environment for you in some separate videos. Now a Property is going to contain up two blocks of code, now most of the time, you're going to see two blocks of code and you'll see these by default when you work with properties. The first one is a Get Block and the Get Block contains any code that you want to run when the Property is read. In other words, when I'm writing some code and I've instantiated an object and I want to read, I've got a New Object called A and I want to read the Weight Property, I want to see the value that's in the Weight Property on Object A. Well, when I call that, it's actually going to go through this Get Block and if I've got any code there, it wants to refresh the value of A or apply any kind of mathematics to the value of A before it sends it back, it can do that. Okay. So we can always check and run any kind of code we want to when somebody asks for a Property. The Set Block is usually where we do most of the functionality with our properties and that is we have a Private Field that actually stores the data that the end user sees but in the Set Block, we load that Private Field with whatever the end user passes in, but this is the beauty of a Property here. When the end user passes data in and tries to write data to our Property that's going to be stored in this hidden Private Field back there, we can run whatever code we want to, to verify that data that's coming in, to check it for ranges, check it for restrictions, check it for number of characters. We can run regular Expressions on it, we can query databases and do whatever we want to. So the bottom line is, is we're not at the mercy of the end user to push, you know, potentially harmful code or erroneous code into our application. We can always check it, now that's the big advantage of a Property as oppose to a field. The programmer has control, the programmer can verify any data coming in or going out and ensure that it's in the format they want, that it meets whatever requirements they have with the application and it just makes things a whole lot safer and a whole lot neater. So that's what properties are, that's kind of the description and in some separate videos here, we'll jump out into C# 2010 Express and actually code up some of these and let you see them in action.

(Fields and Properties / Creating a Property)

Now let's take a look at creating a Property and as we create a Property, I want to explain some of the things that's going on here to help you understand and if you're new to this, this can be real confusing and I hope this will help. Notice I've got a form and a button, my same old simple example in Visual C# 2010 Express. I just created a Windows Forms Application, so if I double-click on the Button here, you'll notice that here's my Button Click Event. Notice here's the Class for the Partial Form Class and I've got too many in here and what I want to do is, I'm going to create my own class down here. Public Class Mark, feel free to use your name, I think Mark's a little better class, but we won't get into that. Now let's take a look at what we actually want to do. I want to create a field if you will called Weight for this class. Now if I do it this way, it works, they'll be a, a Weight Field for every object instantiated off this class, however this is dangerous because the end user can set Weight equal to whatever they want to and I have no control over it. And so they could set Weight equal to 15,000 that could cause problems in my program when I'm trying to deal with weights of products or people or whatever. Alright. So Weight at 15,000 probably not correct, I've never met anybody that weighs 15,000 pounds, although I'm well on my way to that mark if I don't start running again soon, but that has nothing to do with C# so we'll just drop it. Okay. So what I want to do here is, to get around that, I want to make Weight a Property where I have some control. Now the way we do that, is first of all, let's make the real value that I want to store Private and the standard way to do this is to set it lower caps. Okay. So I'm going to create an Integer, a Private Integer that I can only set from inside the class as Weight and then I'm going to create a Public Property. Okay. An Integer of Weight with capital letters, keep in mind C# is case sensitive, so Weight Lower Case and Weight Upper Case is two different variables. Alright. Now what I'm going to do is, open and close some curly braces on this and then I'm going to put the Set Accessor in, go to the next line, open and close some curly braces, go down a couple of lines and I'm going to call the Get Accessor and put a couple of curly braces. Okay. Now here's what's going to happen, when someone tries to set the value of Weight, capital Weight, that they're going to see because it's Public and they say, Weight equals 6. Well what's going to happen is, whatever they set Weight to, the Weight equals 6, it's as if they're calling a method. Okay. That returns Void but passes in an Integer called Value and we're going to set my real Weight, which is Weight equal to Value or whatever value they pass in and I'll explain that in just a minute. Whenever someone wants to know what the current value of the Public Weight is, they will try to read this, write it out into a Text Box, a Label or use it in a calculation and when they read Weight, we are just going to return the value of Weight, lower case. Because we will have set this through this or through an Accessor somewhere else. Okay. Now let's think about what's going on right here on the Set. If you want to think about Set right here, when someone says Weight equals 5, what's really happening, this Set Accessor is in the background, running a function that looks something like this or a method if you will. It's as if you called Public Void capital Weight and you passed it an Integer called Value. And when you did that, we simply took our Private Variable and we set it equal to whatever you passed in. So this is actually what's running you whenever you set Weight equal to 5 and this Set Accessor right here is running this in the background. You don't have to code this, C# does it for you. When you try to read the value and you say what is weight equal to? Then what it's going to do is it's as if you just called a Public Void Weight, you didn't pass it anything and it simply returned Weight to you. So that's what that looks like when you call Get, this is really kind of what's happening in the background, we're getting a problem here because I'm, I've got a scope issue or something, so don't worry about that. But that's what's going on. Okay. Now an easy way to think about this is if I jump out here I'm going to copy and paste this so that you don't have to watch me fumble through but these particular items are what's being called. Now if you can get that image in your head I would strongly encourage you to kind of code this out and look at it and then it will make sense. Now notice the power this gives us, when someone tries to set Weight out there in their code. Okay. They're going to write something, you know, let's, let's just do one here. Mark M equals New, Mark Class and see, they're going to see Weight here and so they call M Weight equals you know, 150. Okay. I think I weighed 150 in third grade. Anyway, so I set Weight equal 150, what it's going to do, it's saying Weight equals 150, it's going to call Set, it's going to pass 150 in as the value and so 150 equals Weight and I just set my internal Weight. I could before I set that, I could write here, start to write code to check or verify the value what's being passed in. If it's greater than a certain value and I can change it, I can throw an exception, I can do whatever I want to. So this gives me control right here, this Set Accessor is what now protects my real value which is Private, my real variable which is Private and it allows a Public Variable for people to read and write. Now this Equals sign, when they set something equal to 150, that is what this caused this Set Accessor and then when they just tried to read this, for example, if I set Button1.Text equal to M Weight right? I'm trying to read that, no, that's, I'm going to have a two String issue there. Okay. So that would fix that, but notice that's going to read M Weight and it's going to call the Get Accessor, it's just going to return whatever I've already got stored in here safely. That's the skinny on properties, make sure that makes sense to you because you're going to see this over and over and over in more advanced ways as you develop your C# skills.

(Fields and Properties / Automatically Implemented Properties)

Now let's take a look at automatically implementing our properties. Now in a video that was entitled Creating a Property, I created a property called Weight and I went through this exercise of creating a Private Backing Property and using my Set and Get Accessors but if you'll notice this. If I clear out all of the extraneous stuff that I put in here trying to explain this thing, you'll notice that we're just setting the Weight equal to the value of whatever they pass in and then we're, we're just returning whatever we had out there. Now this is kind of cool but it's really not doing anything, so let's back this out and let's talk about exactly what we have to do. Well if we're not going to implement any code on any of this stuff right now at this point, then what we can do is, just automatically implement a property automatically. And let me show you the full way of typing it out and then I'll show you a great shortcut on this but all I had to do was say Public Int Weight and then open and close my curly braces and then just type Get semicolon Set semicolon and that's it. That Weight Property will now work, notice I didn't have to do a Private Backing Property, I didn't have to worry about any code whatsoever, this just happens. Now I can hear some of you yelling, dude, you're not doing anything there, why would you go through that exercise? Well it does act the same as a Public Field, however this is considered good practice, number one and this also reminds me that this is just out there, they can Set it and Get it with no restrictions and I can come back later. I could add a Backing Field and start to expand my code and write it in here but this is just considered good practice. Okay. And you'll see this all over the place out there. Now there's a very quick way to do the same thing. I can type the word Prop and this is a Code Snippet, hit the Tab twice and notice, there it is. Just double-click on this, My Property, call it Weight and I'm done. Okay. So that's an Automatically Implemented Property and what you want to do is automatically implement your properties as you build them and this just gives you a good practice to follow. And then if you ever want to come back and start adding your Set Accessor Values or anything like that or any kind of checks or validations in the Set Accessor, you can very easily do it. But this is considered good practice, you will see this all over the place but just understand to set this up, you don't have to type forever. Alright. Prop two Tabs, Set your Property and also if you want to come out and change your Type you can do it right here as well and with that, you're done.

(Fields and Properties / Read or Write Only Properties)

Making our properties read or write only in C# 2010 is very easy, it's been this way for awhile in C# but notice if I've got a property out here and I create a Public Integer called Weight and then I'm going to come down here and do it real quickly. Get Set now if I try to use that and I'll create a new instance of the Mark Class as a Variable M of the Mark Type, then I can get to Weight and I can set it equal to 5, no problem. Now if I want to make this a read only that means I can only get the value, I can't set it and so I will take Set out and you'll notice instantly I get a red squiggly line and if I mouse over that, it's telling me the Property or Indexer Mark.Weight cannot be assigned to, it is read only. So just by taking Set out, just don't put a Set Accessor in there, only put a Get, it makes it read only. Now what if I change this to Set. Alright. And notice it will let me set it, I can now set it to 5 but if I try to read this value. Okay. So let's set our Button1.Text Value equal to M, M.Weight. Okay. I'm going to get a Conversion Error anyway but notice that it's saying that I can't do this because it lacks the Get Accessor. Now what that means is, I can only set this but I can't get it. That means it's write only. Now Write Only Variables are rare out there in the real world, but just know that you can do them because there's that, you know, one in a couple of million times whn you're writing code that you need a Write Only and so if you have just Set, you're Write Only. If you have just Get, you're Read Only and if you back to Get and Set, you have full functionality. This is giving me a squiggly now because I need to convert it to a String to make that happen up there. And so now there's magic. But that's the way to do Read Only or Write Only Properties in C#. Bottom line just remove whichever Accessor that you don't want to support and you make it happen.

(Fields and Properties / Property Access Modifiers)

Just like our classes, our methods, our fields, Properties have Access Modifiers as well and there's always a default on each one of these. And regardless of whether you're talking about classes, methods, fields, properties, if you don't mention an Access Modifier it's going to Default to Private. And that can sometimes surprise you if you don't think about it because you're trying to access something from outside the class and it's not even there, you can't see it and then you'll finally realize, oh wait a minute, I didn't declare any kind of Modifier and so it has Defaulted to Private. Now the Access Modifiers that are available for a property are very familiar, the old standbys, Public, Private, Protected, Internal and Protected Internal. Now let's talk about these and you'll notice there's just enough difference here between properties, fields, methods, classes to kind of drive you nuts but let's just step through these. First up, the Public Access Modifier just like with classes, just like 00:01:00] with fields, means you can access this property without restriction. This is a full Public Interface to the outside world and you can get there. Private means you can only access this property, meaning you can only write to it or read from it, from code within the Containing Class where the property lives. Protected, that means that I can access that from within that Containing Class or from any class that's inheriting from the Containing Class. Okay. Then I've got the Internal, I can access it from anywhere in the current assembly and then last but not least, is Protected Internal. I can access it from anywhere in the current assembly or any class, even if they're in other assemblies that derives from this class. In other words, if they're using this class as a base and they're inheriting it, then they will be able to access this particular field, anybody else won't be able to. There's your choices on Property Access Modifiers, again the more your familiar with these then what you'll be able to do with these and what you're really trying to do is, provide absolutely no more exposure to your properties than you actually have to, to allow program functionality for security reasons.

(Methods / Understanding Methods)

Now let's turn out attention to Methods and let's talk about just some basics here in a video entitled Understanding Methods. And the reason I'm concentrating on understanding Methods here, these are some of the functionalities that are the basics for what your objects and your classes can do for you with your program and a lot of developers kind of paint themselves in a corner and they always build their Methods the same way. They tend to forget about certain functionalities and so I would strongly encourage you to take a look at the videos in the course here that had to do with Methods because there's a lot going on here and a lot of really cool stuff. Now first of all let's start off with a definition so everybody's on the same page. A Method is a Class Member that implements an action, notice that one word there, an action that can be performed by an object. So we're going to write functionality into our class, then when this class is instantiated into an object, this functionality becomes a Method. It becomes the actions that, that particular object can take. Now another way to think of a Method, if you ever did any procedural programming back before the world of Object Orientated Programming appeared, this was what looked like Subroutines there. Now Methods provide the magic and the functionality for our classes and our objects, there's a couple of things about Methods you need to always make sure that you do. First of all, you need to make sure that you choose a name for the Method that makes the Methods action very, very clear. Just one quick glance, one quick read of the name of the Method and you know what it does, it checks the customer account, it has to do with treating an animal with our veterinarian example that we've been using in a number of places in the class here. So number one, name the Method what it does. The second important aspect is that a Method should perform one task and one task only. I don't know why but all developers seem to have this fascination and they all call you over to their desk and they're all excited and they say man, watch this, watch this. You pass in these two parameters and boom, you know, this thing balances the company check book, it hires three new people, you know, it checks the air filters in the air conditioning system. You know, it does filing, it does all kind of stuff, that's going to be a problem later on when you need to update this application, troubleshoot the application, make your Methods perform one task and name the Methods well. The Methods, the functionality of the Methods should be directly related to the Method's name and it shouldn't wonder outside the fence there. Now the best way to think about this and remember it and this is also good when you're designing your applications, your class names are usually nouns, again in the context of our veterinary example that we've been working with a little bit in the course here, class names are usually names, Animal, Dog, Cat, Snake, Bird. Methods names are usually verbs, things like Add, Copy To, Verify, Count, Heartworm Treatments, Feed the Animal, Board the Animal, that sort of thing. So verbs tend to be our Methods Names, nouns tend to be our Class Names. Now this is very important right here, each Method in a class must have a unique Signature. Now if you don't know what this is, you're thinking I didn't know these things could write, they can't. A Signature is basically the description that we would make of this Method, it includes the name of the Methods that you give it and then the parameters for that Method. Now when we're talking about parameters, we're meaning things like the number of parameters, the types of these parameters, you know, Integer, Decimal and so forth and then the actual order of those parameters. All these things taken together are the Signature of the Method and this becomes huge in functionalities that you're going to see that we're going to build on as we move through the course here. Now the Method also needs to designate the type of data that it returns, if it returns a result. Now Methods sometimes will send you something back but when it stops running, it finishes running it's code, it has a value, a calculated value, a result, a true or a false and it needs to pass that back to whoever called it. And you need to determine and designate what that it is, if it's an Integer, a String, a Long, a Decimal, those sorts of things and then your Methods don't have to return anything. If they don't instead of putting a Data Type there, we're just going to put the word Void. Now you'll notice here I haven't done any Methods in this video, I'm going to actually jump into the C# environment in a separate video called Creating Methods and we're going to build a couple and talk about some of the aspects of building them, what the syntax looks like and all that. But this video I just wanted to lay the basic groundwork, make sure we're all on the same page and that's the Basics of understanding Methods.

(Methods / Creating Methods)

Now let's jump out into Visual C# 2010 Express and let's just build a very simple method and talk about some of the ins and outs of methods and I'm also going to lay the foundation and point for some very clever segways into other aspects of methods that we'll cover in later videos. I'm just going to double-click on the Form, you'll notice I just created a New Project, a Windows Forms Application Project. And I didn't even put a button there, just double-clicked on the form, that dropped us into the Form1 Load Event and we'll talk about Events later. But for now let's just create a very simple method. I will use a Modifier, Public Int Calc and I'll put Int A, Int B close my parenthesis, go down to the next line and open and close curly braces. Now what is going on here, Public is the Access Modifier for methods, we'll talk a lot more about Access Modifiers for your methods in a separate video that will entitled Methods Modifiers, so I know you'll be anxious to watch that movie. Then we're going to tell the, in the definition of the method, we're going to indicate rather than tell, we're going to indicate what the return Data Type will be and I'll talk about that in just a second. There's the name of the method, again we want to make sure we name it and what it does. This is calculating something, probably in the real world, I would want to say Calc Interest or Calc Weight, whatever I, you know, whatever I need to do. Then I'm passing in, I am naming two parameters that means when I call this, I'm going to have to pass in two arguments and I am expecting an Integer in the first position and this is A. Then I'm expecting an Integer in the second position and this one will go in as B. Now what I'm going to do here is jump down into the method, make sure I stay within my curly braces there and I'm just going to write Return A times B. Close the statement with a semicolon just like always, notice there is no semicolon right here. Now what's going to happen when we call Calc Weight and we pass in two arguments into our parameters here, it's going to take those two arguments, multiply them together and return them back out as an Integer Data Type. So let's run this and let's just look and see what happens. So if I call Calc Weight, open parenthesis and notice it's telling me, hey wait a minute, you're looking for an Integer here, so I'll give it an Integer, hit the Comment. It's saying okay, you need a second Integer here, I'll put that in, close the parenthesis, give it a semicolon and I'm ready to go. Now Calc Weight is going to pass 3 and 4 in, 3 into here, 4 into here, these are positional and it will return it. Now where's it going to return it? Well, it's going to return it right back up here to where I called it but I don't have anything to display up there and so what I will do is, to make this a little easier to see, I will create an

Integer called C and I will just set it equal to and I'll pull Calc Weight up to right there. And that will go call Calc Weight, load the result into Integer C and then I'll do a nice little Message Box to show C. Now I have to convert that to a String because it's coming back as an Integer and I want it to show on a Message Box which is expecting Strings. Alright. So I will execute this, run it and you'll see there's 12, now this popped up before the form because my actions are in the Form Load Event. So as the form's loading, my code runs, I click OK and there's my form. Now if I didn't want to return anything here, I could just take the Integer our and put the word Void, that simply means hey nothing's coming back and then what I'm going to have to do here is take my, I'm going to have to change it since I'm not returning anything. I'll create an Int C down here and then I will set C equal to A times B and then I will put my Message Box down here. So I will just cut it out of here, paste it in here and we've got the same situation. Then I will take this off of here and so now what we're going to do is just call Calc Weight from here. When we call it here, it will execute this and instead of returning anything, it is simply going to just run the Message Box right here. And so I can run this either way, notice it ran it, there's my 12 and then it opens my form. So you always have a question about, do I want to just pass my data in and manage it here and have it do whatever I need to do or do I want to return it back and do something? That is going to depend on how you're doing it and what you're actually trying to accomplish and so forth. Now I want to show you a common error. First of all, a tip then an error. Okay. First of the tip, instead of calling this Integer A, to make it easier for you to understand, you could call this, you know, Package Weight, PKG WGT or even spell it out. Okay. P Weight and then on Integer B, whatever this is, this could be Ship. Alright. Notice that's going to work and it's going to pass them in, I'll have to match them here. Okay. So I have to copy here and match here. Okay. So that's one tip, that way when you come back and look at this 3 months from now, you can understand what you're passing in here because this is usually not right here, it's way off in some other part of the program and it'll save you a lot of time. The next thing, when I try to call this, I want you to see, I'm going to take these parameters off these arguments and notice I'm getting a red squiggly line and if I mouse over that, notice the Error Message says, No Overload for Method Calc Weight takes 0 arguments. And it's saying, I don't see a Calc Weight out there with a Signature that doesn't have any arguments in it and that is telling you that you're trying to call it without passing it something that it's looking for. Okay. I'll have the same problem if I just do 1, 3 or if I do 3, 4, if I give it 3 arguments and mouse over, it says, there's no overload that takes 3 arguments. Alright. And so once I match those up, my squiggly will go away and I'm okay. So anyway, that's kind of the basics on methods, we will go a lot more in depth with methods in the next few videos in the course but for now, that's a good starting point with just Basic Methods Construction and Operation.

(Methods / Understanding Parameters)

Now let's focus just a little bit on Parameters. If you've already seen the video Creating Methods you've already seen me using Parameters and if you've ever worked with C# Methods, you've used Parameters yourself and in this video I want to talk about the basics of Parameters and understanding them because we're going to build on this as we go through. Now let's start off with the purpose of Parameters and let's deal with the very confusing aspect of Parameters right off the bat here. What Parameters do is they allow the caller of the methods to supply values or objects for processing by that methods or we're going to pass these values into the code that makes up the methods and allow it to use it. Now here's a good example, notice I have a methods called Test This. It's a Public Method, meaning everybody outside this class can see it and utilize it when they create objects. It's Void, means it doesn't return anything, if I could see the code here, it does some functionality but it doesn't return anything back and it's taking two Parameters or it's expecting two Parameters. The first on is an Integer Data Type and it's called Weight, we can tell just by looking at this variable, it's an Integer and we're going to be passing in some sort of Weight value. The second one here is object and it's going to be passed in through the Variable B. Now sometimes, often times actually, we confuse Parameters with Arguments and usually these things are used interchangeably but technically that's not correct and so let's go over this here. And this can be important because sometimes when you're reading documentation and you're trying to figure out some other more advanced aspect of C#, these little definition type things can really confuse you and hinder your ability to figure some things out. So first of all, a Parameter is the value that a methods expects you to pass when you call the methods. Alright. So this, up here this Integer Weight and Object B, these are Parameters. This is the methods saying, okay here's the Parameters I want you to use, I need an Integer and I need an object and I'm going to take whatever Integer give me and I'm going to load into Weight and I'm going take whatever object you give me and load it into B and pass it in. So the Parameter is the value that a methods wants to see. An argument is the actual value that is passed to the Parameter or through the Parameter. For example, if I call Test This and put 100 right here, then that's the argument. The argument 100 goes into the Parameter Weight, then I have an object, let's say I have a class called Bob and I put Bob in here, Bob gets loaded into B and passed in. So the Paramter is Object or B and the argument is Bob, make sure that makes sense to you. Think of a Parameter as a parking space, thing about argument as a car, we can multiple different cars into that parking space as we need to. It's not a big huge deal but again it can save you some time, sometimes when you're trying to figure things out. Now here's the big thing that you need to get your head around and I'm going to tell you about it here and then I'm going to show it to you in some code in a separate video. When you pass these Parameters or you pass an argument into a methods, using a Parameter, they can be passed as one of two types and you've already seen these. They can be passed into the methods as a Value Type which is the default, if you don't mention the Type, it'll go in as a Value and here's an example of one. Again, same example, Public Void Test This Integer A. So we're going to call Test This, we've got to provide it an Integer, 36, 45, 87. Now since we don't mention anything besides Integer in A, this is going in as a Value Type. So this variable is going to be a Value Type, now that means it's going to be stored on the heap. If we copy it, it makes a copy of it, we have two separate copies. Then we can pass our Paramters as a Reference. You don't see this very often, it's explicitly or it's implicitly set for us when we pass in objects and that sort of thing but we can also designate it and you don't see this a whole lot, but you will see it and I'm going to demonstrate it for you in code. When we want to pass our Parameters and arguments as References, you will see it written this way. Notice Public Void Test This Ref Integer A. Now notice we're calling Test This, we're going to pass in an Integer A, 32, 46, 85 that sort of thing so when we pass this Integer in, this A, so we pass it 86. It gets loaded into A, it's going to be passed into the methods as a Reference Type not a Value Type. Now that's going to be huge based on how we use that particular variable here, A and the value that's coming in later on in our program and I'm going to show you an example specifically showing you what's going to happen when we pass the same value into the same methods using it as a Value and a Reference. So you can join me By Ref and By Val and in that video, I'm going to go through this thing of Value and Reference passing to see what it will do. So anyway that's the basics on Parameters and we'll dig a little deeper on these as we move through the next few videos.

(Methods / By Val and By Ref pt. 1)

Welcome to Part One of By Value and By Reference. Now this is going to be a two part video because I want to talk to you about something that's fundamental that you get your head wrapped around, that's a southern technical term for understand, how these parameters and arguments are passed into these methods and how it can really confuse you. It can steal an afternoon from you sometime trying to figure out why things are changing the way they are. Now we're talking about the standard thing in C# about Value Types or Reference Types. Value Types store the data within the variable, Reference Types store Pointers to the value and so if we change any one of the Pointers, the value changes for every object. So let's go take a look at what's going on here and I'm going to build a little example to show you and notice I just had there a Basic Windows Forms Application in C#. I named it Method Parameters Example, you will see this project in your Work Files folder because there's going to be a fair amount of irritating keying here. But what I want to do before I even start calling anything, is, let's go down here and let's create, stay within our class Level here and let's create a methods called Public Void Test Val and let's set a Parameter of Integer on this. Now what I want to do is just take say, A equals A minus 4. Alright. Pretty simply and then I want to create a Message Box or just pop up a Message Box to see what's happened. Okay. And I'm actually just going to copy this in there so you don't have to watch me type this otherwise we could here for about a 2 hour video, watching me try to type this stuff in here. Alright. And let me make this go away so that we have the entire screen and we can see. Notice what I'm going to do, I'm going to call this, I'm going to pass a value in right here as an Integer. It's going to go into here and whatever Integer's there, I'm going to set it equal to itself, minus 4. Then I'm going to throw up a Message Box that says, My Int, which is the value we're going to pass in, passed in via a Value Parameter A and inside the methods A is equal to and it's going to show us what A is equal to here. Alright. So let's go up here and put some magic code. We're going to create in the Button Click Event an Integer called My Int which is what I'm referring to here and we're going to set that equal to 20. Now what we're going to do is we're going to put a Message Box up here, you see why I'm just going to put this out there for you, by Value before the Method, My Int equals and then I'm just going to concatenate and I'll put the value of My, My Int Variable. Okay. And then I will call my Test Val and I'll pass it My Int and then I'm going to put another Message Box, so I'll just copy this, paste it right there and then I will say, By Value after the Method. So we're going to get three Message Boxes when I run this. We're first going to get this Message Box that shows you the value of this Integer before I run the method and it will be 20. Then we will pass that variable equal to 20 into the Test Val Method and the Test Val Method's going to take it, load it into A, it's going to make a copy of it, put it into A, pass it in, do some work on it. It's going to tell us what the value of A is after this happens, 20 minus 4 should be 16 and then it's going to return back up here, it's not going to report the data anywhere else. Then we're just going to run a Message Box and we're going to see My Int at this particular scope and see what the value of it is. Alright. So let's run this and take a look. So I will run this, I'll click the button, the first one, By Value before Method, My Int equals 20, then My Int's passed in via Value Parameter inside the method, A is equal to 16, we guessed that because A equals A minus 4. A equals 20 minus 4. So it's 16, then we click OK again and then when we get back up here to this second Message Box up here in the Click Event, By Value after the Method, My Int is 20. So notice what happened, when we passed My Int right here into the Test Val Method, it made a copy of it. Okay. It made a copy of it and it said okay, this is a copy of My Int, so come in here do whatever you want to with it. A became a copy of My Int, total separate copy and it manipulated A inside here but it did not effect this particular, My Integer. So now let me run that one more time, so that you can see it, let me get this out of the way, we'll click the button first. Okay. That is this line right here. Okay. So before the value, before the method, My Int equals 20 and there it is. Then we pass it into the method as A, so we pass My Int right here into A, it does it, it becomes 16 and then when it comes back out of the method and we run it up here again and check My Int, it's equal to 20 why? Because the copy of My Int that was passed in here was changed but not this particular one right here. Okay. My Int got copied into A, we worked on A and reported A. Alright. That's passing it By Value, that is the default, notice I didn't mention anything right here about how to pass this in. You don't even have to put By Val in there or anything like that, just don't mention the type or how you want it to be treated and it will automatically default to By Value. Now in Part Two, we're going to do the same example, we're can actually add into here so you can see I'm running side by side and we're going to look at passing A in as a Reference. So join me in Part Two of By Value and By Reference.

(Methods / By Val and By Ref pt. 2)

Welcome to By Value and By Reference Part Two and in Part Two we're going to continue with the example that we were building in Part One. And as a quick refresher, if you remember in Part 1, we created a simple method and we were looking at passing our arguments through this Parameter as a Value Type. And in this video, in Part Two, we're going to go the Reference Route and we're going to look at the differences in what happens to our value up here that we're passing in, into our variable. Alright. So let's come down here to start with in this example and let's create a method that will take a Reference and so I'm going to do Public Void Test Ref and this time I'm going to pass in By Reference an Integer called A. Then I'll put my little braces and now I'm going to copy this very same information from Test Val and I'm going to put it right here and then I'm just going to make a, a quick change here. Int passed in via a Reference Parameter, A inside the method and now notice this will confuse you. Understand what I'm saying here, some of you maybe thinking, wait a minute dude, an Integer is a Value Type, are you like converting that or casting that into a Reference Type? No I'm not, I want to pass in an

Integer Data Type but I want to treat it as a Reference when I pass it in. Alright. I don't want to go too deep here but just trust me. Now what's going to happen is, we are going to come up here now and write some magic code to actually make this happen and so we're going to use the same variable right here. So I'm just going to copy this Message Box and come down here and I'm just going to change this to By Ref before the Method, My Integer or My Int equals whatever My Int equals which should be 20. Alright. Then we're going to call our Test Ref Method and we're going to pass it as a Reference, My Int. Now really what we're saying here is, I want you to pass My Int, I want you to pass this Integer with a value of 20 into this method down here, Test Ref but keep in mind when we pass it as a Reference, we're not actually passing the variable. We're not making a copy of the variable, we're making a copy of the Address Pointer to where that data is. Now what that means, that's very similar to creating two desktop shortcuts on your desktop pointing to the same item. Alright. If you change one of those items, then you basically, you've messed up both of your desktop shortcuts. If you were to make a change to both the desktop shortcuts, it, they, they would be correct then but with this, it's just the opposite. If I have My Integer pointing to 20 and an A pointing to 20, then if I change A to 19 then I also changed My Integer to 19 because each of these will just be pointers to this and so if this changes then the data returned to both pointers changes. Okay. So there's, I'm making my Reference call here and then I am going to put another Message Box, I'll just copy the one I used up here to write here and I will say, By Ref after the Method. So I'm going to run this, the first three that we see will be By Value, the second three will be By Reference. So if I haven't fat fingered anything, let's see if it will crank up and run. Now here's the first one, By Value, let me keep going here, By Value, notice it went in as 20. Inside the method became 16, after the method it became 20. Now here we go By Reference. By Reference before the Method it was 20, inside the Reference, inside the Method, it became 16 and after the method it's 16 and notice the difference there. Pretty big deal, let me run that again and move everything out of the way and let's talk through this. So I'm going to click it, before the Value right here, before the Method passing it By Value, notice My Int is equal to 20. We pass it into the Method right here into to Test Val and this is that line right here, A equals A minus 4, 20 minus 4 equals A, that's 16 and that is what we printed here, that's what you see. Then we come back out and check it right here and By Value after the method, My Int, right here equals 20, why? Because when we passed My Int in through right here, it simply made a copy of My Int, this one stayed okay as 20 out here, A became it's own variable worth 20 in here and when we changed A, we were changing a different variable than My Int. Then we did the Reference route, so By Reference, My Integer is equal 20 and here's the line that shows us that, no big deal. Then we pass it into here, 20 minus 4 equals 20, so A is equal to 20 but now since we passed By Reference A points to the same place that my Integer points. So when I change 20 to 16 it also changed it for My Integer and when I get back out and read it after the Method, sure enough that's exactly what happened. Because when I read the box right here after the Method My Int equals 16. Now this will drive you nuts, if you ever try to do this or if you ever try to do this, you don't have to write a bunch of code, you know, to make sure that both of these match and set one and so forth. If you'll just set it By Reference then you don't just make copies here, you will actually make this one point to the same place or the same value as whatever you're passing in. That's fundamental, you can use that to your advantage in a big way, it can also keep you there for an extra day and the weekend trying to figure out where the bugs coming from okay. So that's the difference between passing your arguments, parameters, as Value or as Reference.