07:00:20 Good morning everybody.

07:00:23 Good morning.

07:00:27 Morning.

07:00:31 Looks like we got some people lagging behind today Oh man.

07:00:37 So, I got on the shirt working.

07:00:39 Did you. Yeah, apparently. I had to update, not just, I was trying in Visual Studio code. I ended up thinking, well, what if I open up Visual Studio, because I have that on my computer to and see if that's working right.

07:00:55 And apparently I haven't updated it in a long time. and once it updated on the sharp and Visual Studio started working code.

07:01:05 What a joke.

07:01:07 So smartly something in Visual Studio was interfering with Bredius code.

07:01:14 Well, all right, that makes sense because they do use the same resources to provide ID and you're editing environment.

07:01:22 One not update and break the other seems like a really really bad design choice was so funny to me I was like what it makes sense. And that like that, but I was still like what.

07:01:35 So for anybody who missed it. We had a small issue with VS code last night that every statement wound up getting like the red squiggly underline that says hey you have an error.

07:01:47 But the actual code ran perfectly.

07:01:50 So, I mean, everything wound up with an error, where we used using system using system IO using system.

07:02:00 Like, all of it, all of it heads quickly, everything was highlighting red with errors.

07:02:06 And like every trick with every trick we could find wasn't working, but updating Visual Studio got it.

07:02:13 That's infuriating hilarious but infuriating

07:02:26 team. We're getting there. We're getting there people are waking up

07:02:36 Megan's back, Megan We missed you.

07:02:41 I missed y'all to definitely would prefer being training. vs.

07:02:47 vs me

07:02:52 on a plane for 12 hours.

07:03:07 There you go.

07:03:07 Oh, Mike has the trick with the Magic Touch when it comes to fixing editors.

07:03:15 No, he's saying no, like saying he doesn't want any part of this nonsense. No one time.

07:03:21 That was me.

07:03:24 It was Travis the magically fixed it. No, it was me that said, Don't remind me. Wow. Well I'm sitting here watching Mike shake his head that he doesn't want to know.

07:03:36 Um, ya know, so the thing that wasn't letting me put bash into the terminal and VS code is because it was new or old, Oh yeah that'll do it.

07:03:47 I think it was.

07:03:48 Was it Kyla we were looking at that trying to figure out how we can get back, get back as a terminal option in VS code. Yeah.

07:03:56 Did you ever get that working. Nope, I gave up. Oh no, come on. I tried I tried for 30 minutes. Did you delete it and then get, like, download the newest.

07:04:08 version of VS code because that's what fixed it for me. I did, I I'll play around with it later, but it just, I don't know last night, I was not having any of it so I gave up.

07:04:18 do you have Visual Studio.

07:04:21 Yeah, that's my new question, because that's, I had an uptick Visual Studio for the school to start working right.

07:04:30 Tyler was doing most of his editing with Visual Studio rather than in the US code.

07:04:36 So the reason we were looking at getting the terminal was so that we could be consistent, so that when I did something in VS code or when he was sharing.

07:04:42 Sharing a screen with everybody. It was familiar so that we could tell what was going on, which I appreciate. That's a good mindset to have when you're working with a group.

07:04:52 All right, well get 22 says Me and 20, we're getting close.

07:05:00 Well, I hope everybody got some rest. Last night I hope we have a better understanding. After playing around with inheritance, a little bit yesterday.

07:05:11 As far as understanding object oriented principles and we talked about those four pillars.

07:05:18 If we can get inheritance nailed down.

07:05:24 Everything else will just come with it.

07:05:26 If we can get the idea of building a class, and then having a subclass extended.

07:05:34 Everything else will be easy.

07:05:35 Everything else will be just ways that we tweak that property, things like when we were using access modifier is and we were running into Well, can we access a field, because it's protected or private or is it public and we can use it.

07:05:52 If we understand inheritance.

07:05:54 Then we're going to understand access modifier.

07:05:57 If we understand, having multiple constructors.

07:06:01 Then we understand method overloading, which is part of polymorphous.

07:06:14 And we're going to talk about overriding and overloading today, specifically, that's our topic. So, I know, if you if you ran into a roadblock yesterday if you were trying to write classes and it just wasn't working, and you kept running into problems.

07:06:25 When I open up the office and we go into breakout rooms. Come see me, come talk about it. Because if we can get this nail down, we're set.

07:06:43 Everything else is just going to fall right behind it.

07:06:38 All right. So before we jump on anything new.

07:06:42 Does anybody have questions, run into problems issues with their editor, anything that we can address. Before we get started today.

07:06:53 It also gives me a minute to down the rest of my coffee

07:07:04 I don't know you guys are uncharacteristically quiet when I asked that question now. Are we moving too slowly, am I good to just like hit this into warp speed and we'll just know Samantha's like no, no, no.

07:07:16 All right.

07:07:17 But we're doing okay, nobody had any massive issues working with the classes we were playing with last night.

07:07:24 Awesome.

07:07:27 Okay, let me go ahead and get set up. I'm going to start a screen share on this machine.

07:07:41 Alrighty.

07:07:44 And I want to go ahead and start our recording for the day.

07:07:57 Okay.

07:07:59 So our topic for today is going to be method, overriding and method overloading.

07:08:06 And I want to show everybody how I do a new project right from the top, because I think this is one of those steps that I just kind of gloss over and I haven't been showing you guys what I'm doing.

07:08:21 So I started by creating a new folder. We're going to call it over, writing and overloading.

07:08:27 And I can open this.

07:08:29 When I right click, I get an option that says open with code. Or I could open, get back here.

07:08:37 Either of those is going to work. I'm going to start with get bash.

07:08:42 And by using that we can make this a little bigger for us.

07:08:49 By using that open get bashed or over the here, it already targets the correct folder.

07:08:55 So I can just go ahead and

07:09:00 create a new.net project.

07:09:05 And now I'm done with get back, I did the one command that I wanted to run.

07:09:10 I now have

07:09:22 a.csj.cs PROJ C sharp project file, and a program that CS.

07:09:26 The next trick that I usually use is inside of the folder that I want to be working inside of.

07:09:33 I usually just do open with code.

07:09:41 And once it finishes loading I'm still getting that hey wait. There we go.

07:09:46 We get our been, we get our Aaj when you see this pop up.

07:09:51 I usually ignore it because I don't really care.

07:09:55 I don't need to build those assets that it's trying to tell me to build. If you click Yes.

07:10:03 All it does is create this VS code folder. I know a couple of people saw that they had one and I didn't. That's why I usually ignore that pop up.

07:10:13 What it's doing is creating a launch dot JSON, which only runs, when we use the built in debugger on Visual Studio code, which we haven't used we've been using.net run and doing it ourselves.

07:10:30 So I usually ignore the pop up but if you click it, it's not gonna hurt you.

07:10:36 And we get ourselves a new console project.

07:10:42 When I open that terminal that's at the bottom of my window, you have this terminal button up top, gives us the menu here.

07:10:52 And it's going to default to whatever your default terminal is mine is still set for power shell because I haven't changed it.

07:10:59 But this is where I go over and say yeah I need to get bashed terminal, just because that's what I like that's what I'm familiar with.

07:11:06 Now I have to move, everybody smiling faces back over here, his name is in my way.

07:11:12 Alright, so now I can once again just double check what we've got.

07:11:21 I'm expecting to see a Hello World if everything works properly.

07:11:27 hey presto, we are good to go.

07:11:29 All right.

07:11:32 For us to start investigating overloading, and overriding.

07:11:37 What I want to do first, is create a main method.

07:11:43 A main piece of program.

07:11:45 So I'm going to start by creating a couple of things I need my using statements.

07:11:53 And I think I just want system for the time being, so that I can use console right line and get stuff out to Mike, my terminal.

07:12:02 I'm going to need a namespace.

07:12:16 We'll call it override overload.

07:12:20 For the first thing in my namespace I need a class.

07:12:33 And then inside this very first class. This is where we're going to have our public static void Main.

07:12:49 And now we're back to.

07:13:01 back to the beginning.

07:13:03 And if we've done everything right,

07:13:07 we still get a Hello World. Excellent.

07:13:13 But we've gone through this process of declaring are using declaring our namespace declaring our class, so that we can make use of the namespace. Later on, so that we can make use of the classes, later on.

07:13:35 So let's keep moving from here. The next thing I want to do is make a new class.

07:13:41 So I'm going to jump over and just create a new file. And we're going to call it base.

07:13:49 I'm going to need the namespace

07:14:08 override overload.

07:14:17 Oh, you know what I did.

07:14:21 I forgot to make this a Nazi Yes.

07:14:25 If it's not a Nazi Yes.

07:14:28 It doesn't know what the heck I'm doing so it doesn't interpret the namespace it doesn't give me the linking the nice pretty colors.

07:14:37 Alright, so I've got my namespace. Now we're going to need to create a class.

07:14:47 Let's start with this, I know missing command, hang on.

07:15:00 Now I tried to tell me Hey something's missing.

07:15:04 What could I add

07:15:08 before class.

07:15:14 Some kind of access modifier.

07:15:17 Public Works.

07:15:26 Now, let's come up with some simple things that our class can do,

07:15:36 I think, to make this a little easier to start with.

07:15:41 I want to just create

07:15:46 a new string.

07:15:48 And I'm willing to set it to public to start.

07:15:59 What do we want to do here.

07:16:03 Get a name did get an email, let's call this.

07:16:08 Something really easy, so that we can tell it apart later.

07:16:14 Yeah, let's do it like that.

07:16:19 And then we're going to create a constructor

07:16:26 constructor is going to be public.

07:16:31 And I don't want to have to send parameters.

07:16:34 All I want is for

07:16:45 bass string to be set to base.

07:16:55 Pretty easy class.

07:16:58 Let's get another one that we can work with. let's go ahead and save that before I lose it.

07:17:03 And we're going to create a new one will call this a sub

07:17:15 No, you know what I want derived will keep the same nomenclature

07:17:23 derived got CS because we have base classes and derived classes, parent or child, Super or sub.

07:17:32 So if we're going to use base let's use derived.

07:17:35 We need to put our derived class in the same namespace

07:17:43 will need to create some kind of access modifier, so that we can create a class and name it derived

07:17:58 will give it some string as well.

07:18:06 What are they named the other one.

07:18:07 Was it just based string so this will be derived string.

07:18:15 And then we'll have some kind of constructor.

07:18:25 We're going to do the same thing.

07:18:39 And we'll save it.

07:18:44 Each of these is currently its own class.

07:18:48 What can I do to derived, to make it extend my base class

07:19:00 is the colon and base, like you were base.

07:19:04 That sounds right. Where should I put it right next to your derived.

07:19:11 Let's try and use a line number because I've got derived on about four different places on line three line three there we go, let's be specific when we say these things.

07:19:21 And we want to say what the base class will be in this, in this case, base.

07:19:29 So now derived is extending base.

07:19:35 And we are inheriting the properties of the base class.

07:19:39 Let's go ahead and try to just create one of those.

07:19:46 I can create a.

07:20:08 A new bass object called my base.

07:20:19 And it helps if you spell it correctly.

07:20:30 In a new derived object called my derived.

07:20:49 Alright so now that we have these two objects.

07:20:55 We can use the fields from those objects.

07:21:00 And I should be able to do a console.

07:21:04 right line

07:21:08 and output.

07:21:11 A

07:21:14 my bass dot.

07:21:16 Yep, bass string.

07:21:23 And I should be able to do a

07:21:29 can help myself a console right line.

07:21:35 My derived

07:21:38 and use the derived string.

07:21:45 And if we save that and run it, what do we expect the output to be.

07:22:03 is then derived.

07:22:11 Samantha agrees.

07:22:16 Do we get a third of it and let me get another vote on this one, come on somebody jumping.

07:22:24 Ryan says yep Mike says Yep. I think so too. Let's see what happens.

07:22:28 Unless we really messed up

07:22:36 hey presto our output is base and derived.

07:22:41 Cool.

07:22:45 Let's do one more. Let's try a console.

07:22:52 So Ellie.

07:22:54 When you get lost in the middle of the word.

07:22:58 Because our derived class extends the base class.

07:23:06 A derived object should also contain

07:23:12 the same things the same fields

07:23:17 as the base object from the base class.

07:23:24 So let's see if we can reach through and touch the my derived bass string, we can.

07:23:39 So we're getting back to the idea of having multiple, multiple classes inheriting fields from one another.

07:23:54 At this point, this is what we played with yesterday, pretty familiar shouldn't be surprising to anybody at this point right.

07:24:04 So let's go ahead and

07:24:09 look at what makes overloading possible.

07:24:14 Let's start with our base class.

07:24:20 When we create a base class and we call our constructor.

07:24:26 That wasn't what I wanted to do. Go away.

07:24:43 Do it get the little exit menu, it's just under everybody in zoom.

07:24:40 Sorry about that. When we call our constructor.

07:24:52 We're sending it no parameters

07:24:57 which matches what we did here.

07:24:59 We didn't send any parameters when we created this new my base object.

07:25:08 When we decide that we want the ability to send parameters. And we create a second

07:25:19 spelling. Good lord.

07:25:22 A second constructor statement.

07:25:25 And we can send a string.

07:25:51 What we've done is overload.

07:25:55 We have overloaded.

07:25:57 The constructor.

07:26:00 We now have two different ones that we send different numbers of parameters, different parameters entirely.

07:26:10 That they both accomplish the same goal.

07:26:16 That's what makes this special.

07:26:19 We can say

07:26:22 that over loading.

07:26:26 If I could spell it properly.

07:26:31 is creating multiple versions of a method or constructor

07:26:54 that except different numbers of parameters or different parameters.

07:27:20 and accomplish the same task.

07:27:33 So if I create a method for my base class.

07:27:41 That is,

07:27:44 let's go ahead and just create a thing. Let's make a public method.

07:28:01 We'll call it a no return and partners for my keyboard.

07:28:31 punctuation, it's important.

07:28:34 Yeah, we'll make it proper English and capitalize things to,

07:28:45 if I create a single method that accomplishes a task, and then have another method

07:28:57 that I'm going to send a parameter.

07:29:21 too far back. There we go.

07:29:24 And we can have it use

07:29:29 that parameter that we're sending.

07:29:53 We have overloaded.

07:29:56 The speak method.

07:30:00 That's all it takes.

07:30:05 So we can

07:30:08 go back to our program, and use those two methods now.

07:30:18 So I should be able to call

07:30:23 the my base object and hit the speak method.

07:30:30 I can also do the same for the my base object and hit the same.

07:30:39 The same one and send it a string.

07:30:55 If we run those let's see if I can stop myself from being too embarrassed. Hey, Look at that.

07:31:02 Our first call, where we use the method with no parameters.

07:31:09 We get our output of Hello I am a base class object.

07:31:14 And in the second one where we do send a string, we have a different result we are obviously calling the other function. The other method.

07:31:25 And we are able to access whatever's going on inside that second method

07:31:37 compiler. The language is just smart enough to recognize that hey we're sending a string.

07:31:43 Let's use this other method to do that.

07:31:47 Some things to keep in mind when you overload.

07:31:54 A method.

07:31:55 There's some best practices.

07:32:01 We want to use descriptive names

07:32:11 for the methods.

07:32:14 They all need to be the same if they are to be overloaded.

07:32:39 We want to use descriptive names for the parameters.

07:32:59 we could,

07:33:05 we could create multiple versions of the same thing here I'm going to do it just for the sake of example.

07:33:33 we could use the same name for all of them.

07:33:38 For every instance of that thing of that parameter being sent into the method.

07:33:49 And you should try to use the same name for the same parameter.

07:34:25 We want to avoid using a different name.

07:34:38 We don't want to do that.

07:34:40 We want to keep.

07:34:43 If it's the same string being sent to the same method.

07:34:47 Even though it's overloaded.

07:34:50 Use the same name.

07:34:56 This was a point that Travis brought up last night.

07:35:00 We need to use or not use we need to send parameters

07:35:10 in a consistent order.

07:35:19 If we have

07:35:24 you know what let's just do this. If we have multiple versions of the same method. Being overloaded.

07:35:55 Give me a second here to make my example.

07:36:15 We would not change the order of the two parameters that we've already established.

07:36:22 We wouldn't switch the into in the string type, like this.

07:36:28 We need to keep our orders, consistent.

07:36:33 So, if you can imagine.

07:36:37 Your first step to be just write out all of the possible parameters.

07:36:44 It's okay to remove parameters from that order. But we don't switch the order of parameters. when we come up with our overloaded methods.

07:37:23 Every once in a while we'll have a method that we overload, that has a slightly different use case.

07:37:37 So what we want to try to do is

07:37:50 do not.

07:38:14 do not have overloaded with parameters at the same position, and similar types,

07:38:23 but with different semantics different outcomes different results different actions.

07:38:30 All right. If we have

07:38:34 our two functions here are two methods that we are sending similar similar parameters, what they're doing with those parameters should be similar.

07:38:50 If we want them to, you know, right line.

07:39:02 Well, interesting.

07:39:03 If we want them to write out something.

07:39:20 We want both to have a similar

07:39:30 functionality.

07:39:34 I think is the best word.

07:39:40 This becomes useful to us, when different parameters may need to be pre determined may need to have some pre assigned value, or may not have been collected.

07:39:57 If you can imagine that filling out a form online, and some fields are required and some are not that they may have a default value that you want to use.

07:40:09 It's that kind of a thing where it can just figure out and say, Oh, I know that value, I can send that in.

07:40:17 And we don't need to.

07:40:21 To assign all of those values every time.

07:40:24 We don't need our user or our code to assign all of those values every time.

07:40:32 I don't think I'm going to use those right now, but we'll have them there for reference.

07:40:41 questions so far on overriding a method.

07:40:46 And the return type will be different.

07:40:52 Yes.

07:40:55 And we can prove it here, we can return a string.

07:41:09 I think you can.

07:41:10 We're gonna find out.

07:41:20 Now that I opened my fat mouth I really hope so.

07:41:40 That's right one.

07:41:42 No problem.

07:41:52 That's the right one.

07:41:54 And then we can prove it with

07:42:06 that.

07:42:09 If we hit it again we would expect to see

07:42:15 it prints out dumb, which is the value that was returned by calling our overloaded method in base.

07:42:23 So, yes, we can have different returns.

07:42:34 Sam says Why didn't you add other number two the second method,

07:42:41 which one here, then and this guy.

07:42:46 Just because I hadn't gotten that far, I mean we could throw it in down here.

07:42:53 I wasn't planning on on doing anything with those methods, they were purely for the sake of example so that we could look at the order of parameters being sent in.

07:43:04 Because I'm a visual guy and I like to see that stuff when I say it,

07:43:11 but very good question. Yes we can have a different return type, with the same name of a method.

07:43:25 Any other questions before we move on to looking at overloading, or excuse me, overriding, We did overloading, we're going to look at overriding next.

07:43:48 So far so good.

07:43:52 All right.

07:43:56 I'm going to go ahead and save this.

07:43:59 And we're going to go look at are derived class.

07:44:05 We're going to need a method

07:44:12 that we're going to just create a quick different method.

07:44:29 Let's not having to do all that much.

07:44:31 I don't want to have to send it anything. I don't really want to have to return anything.

07:44:38 I just want to know that I am running.

07:44:55 This particular method from derived class.

07:45:05 So if we do that, we should be able to call our are derived class object.

07:45:19 And go ahead and stop that from printing out with unnecessarily. And we can use my derived.

07:45:34 And we should get just a simple printout that says printing from there we go running new method from derived. Perfect.

07:45:46 Method overriding is having a super and subclass have the same method with different functionality, or different parameters, or something like that.

07:46:08 Let's go ahead and start by getting a definition for method overriding.

07:46:19 We're going to say, it will put this in are derived class.

07:47:11 Yeah, we're implying that from another class method overriding is a technique that allows the invoking of functions from another class, the base class.

07:47:26 In the derived class.

07:47:35 creating a method in the derived.

07:47:55 Creating a method in the derived class with the same name, or the same signature, as a method in the base class.

07:48:13 So what we're going to do in our base class.

07:48:23 We can create

07:48:26 a very, very similar method to the one in our derived class.

07:48:37 It's going to have the same name.

07:48:40 It's going to be new method, and the only difference is that it's going to run from the base class.

07:48:58 But you may notice that know we're starting to get some really weird squiggle ease.

07:49:04 Because our compiler has started to recognize that Wait a minute.

07:49:09 You have new method in your derived class, and you have new method in your base class.

07:49:17 You can't do that you can't have the same thing in two places.

07:49:23 We need to tell the compiler we need to tell the program that yes, this is okay, I'm doing this on purpose.

07:49:33 The way that we're going to do this

07:49:37 is by using two keywords

07:49:44 in the base class.

07:49:50 We're going to use a modifier, similar to an access modifier, that it sets, some conditions on a field.

07:49:59 These modifier.

07:50:02 That's just what they're called are going to set a condition on the method that we are going to override.

07:50:10 There are a couple of different model fires.

07:50:13 The first one that we're going to use his virtual.

07:50:20 And just the same way that we can have our access modifier. Here, We're going to use our generic modifier

07:50:32 There.

07:50:35 So we can say that our declaration is our access modifier.

07:50:46 Then our bottom five modifier, the return type,

07:51:00 the method name.

07:51:05 And then parameters.

07:51:21 So the virtual keyword.

07:51:39 identifies or is used to modify a method, a property in index or to allow it to be overwritten.

07:52:06 I know that line got a little bit long.

07:52:09 But, there we go.

07:52:16 So when we decide that. Yes, I would like to have a method in my base class that will be overridden later on.

07:52:37 It needs to have the virtual keyword as a modifier.

07:52:35 If we go to are derived class where it says oh man there's still something weird going on.

07:52:42 We need to tell our system, we need to tell the compiler once again that this is the method that we're going to work with.

07:52:51 Let me get my note here.

07:52:55 I keep clicking that and I don't want to click that.

07:52:58 Zoom keeps hiding the thing that I need to click to close it.

07:53:11 So we have an access modifier public. We have a return type void.

07:53:16 We have our method name, new method.

07:53:20 We even have parameters, we're just not sending any parameters, still counts.

07:53:25 What we need is a modifier.

07:53:28 It's going to go between our access modifier, and our return type.

07:53:39 And the one that we're going to use is override

07:53:59 and override is required to do this.

07:54:48 if we don't include override, we still get that, that error, we still get that weird squiggly say a pop back up again.

07:55:00 It says, it looks like this is what you want to do. It looks like you might want to override this, but you haven't told me that yet.

07:55:08 So I don't know what the heck is going on.

07:55:11 We have to explicitly tell

07:55:15 that this is override.

07:55:30 When we use an override keyword, and when we create an overriding method.

07:55:41 There are some things that we need to do.

07:56:10 Give me a second here.

07:56:31 must be.

07:56:35 Hey, come on, he. There you go.

07:57:24 Just throw in some punctuation. So, when we use an override when we override a method.

07:57:34 The member that we are overriding, whether it's a method or an index or or a property, or an event.

07:57:46 So far we're just doing it with a method.

07:57:50 It must be in both classes that we're trying to do this with.

07:57:57 Otherwise, what the heck are we trying to override.

07:58:01 If we use the override keyword.

07:58:03 We're telling the compiler, look for something to override, if we just use it and there's no other method no other member that needs to be overridden.

07:58:16 Then we're going to confuse the heck out of our system.

07:58:22 The override members must have the same signature. As the inherited method.

07:58:28 This signature void new method needs to be the same as the version that we are overriding whatever our target is. That's what we need to be using the same signature.

07:58:52 Both methods must be virtual abstract or override.

07:59:00 When we have a base method.

07:59:07 If we do not include the virtual abstract or override key word here on our base method.

07:59:20 The use of override will throw an error.

07:59:30 It's not marked virtual abstract, or override has to be marked we have to declare we have to say, This is the method that I want to override.

07:59:44 We put it back and magically new method is ready to be overwritten again.

07:59:50 Finally, we must not use the static or virtual modifier to override a method.

07:59:58 This comes into play when we use multi level inheritance.

08:00:05 We only have two classes right now, with one with derived extending base.

08:00:13 If I had a second or a third level of inheritance.

08:00:18 I would not be able to use static or virtual to override my new method again.

08:00:28 Let's see if we can really quickly build one that does that.

08:00:40 I want to copy.

08:00:43 Just about everything from my derived.

08:00:58 will say that this is second derived, we are extending derived.

08:01:06 We can still have you know what let's do a second derived string.

08:01:41 It should be okay.

08:01:42 Did I spell something one

08:01:47 second derived slide number five. Did I forget an M. Yeah, see od up there it is. See, my spelling it gets ahead of me. That's good.

08:01:59 Thank you. I know Rio saw it, I appreciate you guys keeping an eye out for me like that.

08:02:04 I'm good I'm not that good.

08:02:11 And then we'll close out our class like that.

08:02:17 Okay.

08:02:15 We now have a base class with new method.

08:02:21 We have a derived class, also with a new method.

08:02:26 And now we have our second derived class.

08:02:32 Also with a new method

08:02:36 that last rule that we talked about must not use the static or virtual modifier to override a method.

08:02:46 What we're saying is on this latest and greatest.

08:02:51 I can't

08:02:54 do that.

08:02:58 See how it automatically just threw up this new error. Oh man, there's a thing. You can't do that.

08:03:07 This is the problem.

08:03:10 Once we've started to override. I can continue to override.

08:03:20 But I cannot switch from overriding a method.

08:03:24 And then in the child class in the derived class, go back to it being virtual or abstract.

08:03:31 This would have to either continue to override,

08:03:38 or I would have to not use it.

08:03:47 So in this case would override the new method in base, or the new method in derived since both.

08:04:00 We are all the way down we are overriding the derived new method.

08:04:07 The derived new method is overriding the base, new method.

08:04:14 Because, second derived extends derived.

08:04:21 That's the method that we are overriding.

08:04:27 It's that chain of inheritance thing again.

08:04:30 Yeah, but so suppose that in derived, you had additional fields or additional variables of any kind.

08:04:44 And those were called in the into new method would second derives new method also have those extra ones.

08:04:59 I guess it like I understand. Yeah, I see what you're saying it overrides, but I'm wondering if it overrides based on the actual keyword virtual versus whatever is last in line.

08:05:14 So, it's going to depend on the object that we create and how we call the method.

08:05:22 Give me just a second and we can demonstrate that.

08:05:28 So most of the time this is Edward asking so most of the time virtual keyword will use base class and then the rest of the derived classes will use override.

08:05:37 Yes, There are other options.

08:05:41 Then using virtual, we can also use abstract and override an abstract class.

08:05:50 But we haven't introduced that modifier yet, but yes, we start with the virtual say this is the method I want to override.

08:05:58 And then every time we use the override modifier after that.

08:06:03 We're just changing it down.

08:06:19 Alright, let's see if we can get a good demonstration of the derived.

08:06:34 Get the derived methods running. So now that we have them.

08:06:38 We've got everything saved, we can go back to our program.

08:06:45 And I think what I want to do is let's see we're using my derived let's start with base.

08:07:08 Yep. Okay, so let's double check.

08:07:11 We should be able to create a base object and run new method from base.

08:07:18 So far so good.

08:07:22 Let's go ahead and use

08:07:32 the my derived.

08:07:40 Now we're able to hit from my derived.

08:07:49 And we should still. Now that we have another one.

08:07:52 We can create a second

08:07:58 derived.

08:08:22 control us to save Thank you.

08:08:30 So now we can hit that from the second derived as well.

08:08:48 Let's try something a little different than declaring each of these separately.

08:09:01 I think what I want to do next is will simply create

08:09:11 an object.

08:09:15 Jay Darn you know, just give me OBJ. Thank you.

08:09:20 And we will make it a new base

08:09:24 will then call.

08:09:29 It doesn't like LBJ.

08:09:39 We just let me create the thing.

08:09:55 you need to put base before thing.

08:09:59 I've got to do one of these.

08:10:11 There we go.

08:10:14 So we create a base class object named thing. We then say that thing is of type new base. So it's a new one.

08:10:23 Then we can start implementing it, then we can do thing dot new method.

08:10:35 Then we can change it.

08:10:44 We can say that instead of being a base class it's going to be derived.

08:10:50 And we can

08:10:53 hit that same version of new method from the derived class.

08:11:02 Yep, there we go, then we can say that it's going to be a new second derived.

08:11:20 And we should see each of these different new methods, being called in our output.

08:11:31 There we go.

08:11:40 So, even though base has a new method derived can have a new method, and the second derived can have a new method as well.

08:11:51 This is over, writing the method.

08:12:03 Any questions on overriding, as opposed to overloading on the virtual or the override keywords.

08:12:30 So far so good.

08:12:40 All right.

08:12:42 Then what I'd like to do is take a short break. I've got 1112. So let's take 15 minutes. We will reconvene at 1130.

08:12:54 Get up, stretch, walk around, grab some more coffee because I am unfortunately out, and it's time to make more.

08:13:01 When we come back, we'll talk about the other keywords that go along with virtual, and override I mentioned there's abstract and one or two others. So we'll touch on those.

08:13:15 And then we'll try and exercise using the code that you guys were working on yesterday to just keep building with what you guys already have sound good.

08:13:26 All right, let me go ahead and stop the recording here.

08:13:36 Alright, so take a bit.

08:13:38 I'll see you guys in a few minutes.

08:29:33 He's doing but he's upset with me.

08:30:07 I think he's mad because I won't let him jump into the turtle tank.

08:30:10 Look, if you could get into a shower and clean yourself up afterwards I would let you, but I would have to take you in the shower and you don't like it when that happens.

08:30:22 Cats.

08:30:23 Why do we bother.

08:30:27 I don't know, I was deceived my dog would come with me but he wouldn't know what he's doing.

08:30:36 You know he's harassing one of the other cats fenders leave Tabitha alone.

08:30:42 He just wanted the spot that she was curled up on, that's, that's what it was.

08:30:47 He, he just wanted the nice comfy spot where he decided that he wanted to hang out and he scared Tabitha off of the good spot.

08:31:03 Screen lid for the tank I actually need to get a new lead for it Samantha, because it's, it's a new tank. I actually got the tank like the day before Thanksgiving, and I got the turtle into it.

08:31:14 And it had two nice big glass lids. And I figured okay I'm set, I can put my heat lamp on top of the glass lid, and it'll slowly warm it up and they'll be enough really to keep going through the lid and the turtle will be fine.

08:31:28 Turns out that the heat lamp I have is too powerful for the glass, and it created a thermal shock and cracked the glass lid.

08:31:36 So the turtle now does not have a heat lamp because I can't suspend the heat lamp up above the tank any good way yet.

08:31:44 Then risk it down.

08:31:46 But he has like his, his day light bulb so he has all the light he needs, and the environment is warm enough, it's just not like the heat lamp he's supposed to have for basking.

08:32:00 But I do need to get some kind of lid so that the cat can't jump in and some kind of suspension for the heat lamp so that I can actually have his warm set up properly.

08:32:16 This is what I get for having multiple pets Would you stop climbing on stuff, nope, nope, nope, nope, nope, nope, not playing that game.

08:32:25 You can say hello.

08:32:29 You gonna say hello or No, I guess not.

08:32:43 We've actually covered the the big concept pretty well.

08:32:48 We've covered what it does and how we can use it to our advantage.

08:32:54 Right, we can use over loading, to have the same function, except different sets of parameters.

08:33:03 We can use overriding to use the same name, but accomplish different goals.

08:33:09 We did it with a really simple example where we just printed out whatever was in that function, whatever was in that method.

08:33:18 But the principle stands that you can change whatever is going on inside that method and access the one that you need.

08:33:28 And I mentioned that there are other modifier.

08:33:32 We've used virtual and we've used override.

08:33:36 But there are a couple of others that we have at our disposal, that we're going to us in different situations.

08:33:44 So let me go ahead and start up a whiteboard. And then once I have the whiteboard we will start up a recording.

08:33:56 And I will boot my cat from the furniture again.

08:34:00 He's being feisty today.

08:34:06 Okay.

08:34:07 So our objective now is to look at the different modifier is that we use.

08:34:16 We said that modifier is our C sharp keywords that we use to modify a declaration.

08:34:41 We're gonna say specifically of a type, like a class

08:34:48 struct

08:34:55 or interface.

08:35:03 But that modifier, can also be used on Type members.

08:35:37 Let's see if we can clean that up a little bit so it's not quite so painful to look at.

08:35:51 And we've already run into that where I wanted to do that.

08:35:58 That was not where I wanted to do that. We've already run into the virtual, and the override keywords.

08:36:13 So let's get some quick notes on this.

08:36:16 We said that virtual is used to modify a method

08:36:30 and allow it to be overwritten.

08:36:43 Get the squirt bottle out because Tennessee has decided to keep jumping on to the, the launching point to get into the turtle.

08:36:53 Okay, so we've got virtual. We've also got override

08:37:03 override is required

08:37:09 to extend or modify

08:37:15 the abstract. Pardon me.

08:37:19 Good deal.

08:37:31 override is required to extend or modify the abstract or virtual.

08:37:51 implementation of an inherited method.

08:38:08 The others that we're going to touch on are read only.

08:38:17 We're going to talk about static.

08:38:26 We can do a little with context.

08:38:35 And then the real one that we're going to spend a little more time on

08:38:44 is abstract.

08:38:46 You missed the and cons.

08:38:51 Did I do I did.

08:38:58 Thank you.

08:38:59 Okay.

08:39:00 Let's start from the bottom up, actually, just because then we'll be able to spend some time on abstract,

08:39:11 a constant field.

08:39:17 We're going to apply a constant.

08:39:21 And we are going to modify a field or a local

08:39:29 it modified modified mud.

08:39:33 If eyes, a field

08:39:40 to not be changed.

08:39:46 Making it.

08:39:51 Not a variable.

08:39:58 It becomes a constant.

08:40:17 To be perfectly honest when I don't remember what final in Java does, I'd have to look up the the definition of final, but probably pretty close. See has a couple of different ways to do very similar things like this constant word using on a field.

08:40:42 So,

08:40:46 constant, constant fields can be

08:40:54 no numbers.

08:40:56 So any number type Boolean string or No.

08:41:05 Whoa, not what I want to just get bigger.

08:41:09 Thank you.

08:41:11 Give it a second. There we go.

08:41:24 We haven't really talked about Reference Types versus value types.

08:41:32 I promise we will.

08:41:33 But I want you guys to know that only string, and no

08:41:46 reference types can be convinced.

08:41:57 And when we use context.

08:42:01 We cannot use the static

08:42:10 modifier.

08:42:28 So, a real quick dirty example of this.

08:42:33 We could have something that we want to create an integer, named x with the value of zero.

08:42:43 To make this constant.

08:42:45 All we would have to do is say

08:42:50 cost.

08:42:53 x. If we were doing this with a double or we wanted to use some other access modifier.

08:43:27 That's right.

08:43:35 Or we could use it with a strike.

08:43:57 All right, all of those become constants.

08:44:15 When we look at static

08:44:19 static we can use on the class.

08:44:27 If we use it on a specific member. It makes only that member.

08:44:38 So static

08:44:47 a static class, cannot be instantiated or extended.

08:45:03 If it's static

08:45:10 all of its members

08:45:16 must be static.

08:45:25 So basically, it just becomes a container for static members.

08:45:50 When we use the static modifier on a member.

08:46:00 We cannot use

08:46:07 this keyword.

08:46:37 All static members belong to the class type, rather than the specific object.

08:47:14 And we can still reference that static member.

08:47:19 But we do it through the type name, rather than the object name.

08:47:50 So we can call.

08:47:56 like that.

08:48:00 We're calling the whole class, rather than the specific object.

08:48:16 I'm realizing that I'm going to run out of room on this page so I'm going to save so that we have this information.

08:48:23 And then as things get pushed off the bottom of the page will still have it.

08:48:29 And I'll make sure that all of the different images get uploaded to the trainer code repo so we have them.

08:48:44 Alright, so a read only a read only

08:48:52 its initial ization can occur as part of the Declaration, or in a constructor in the same class.

08:49:39 And like a contest.

08:49:45 But it's initialization can be deferred until the constructor finishes.

08:50:13 So property wise, it's a lot of the same mods, a lot of the same restraints as a contest.

08:50:23 It's just, where it happens when it happens that will change.

08:50:39 Alright, that's going to bring us to abstract, which is one of the important ones like virtual, and to override. We're going to use abstract, a lot.

08:50:54 **Abstract is going to give us the ability to start an idea in a base class and finish the idea in a derived class**.

08:51:15 It lets me declare that I'm going to do this, but not yet. Whenever I use this parent class.

08:51:19 I want to make sure I do this other thing later on. but it's going to change so much that I don't have the specifics in my parent class the specifics have to come in my derived class in my child class.

08:51:34 So we can say that an abstract.

08:51:39 It implies that the thing being modified

08:51:49 has a missing or incomplete

08:51:55 implementation.

08:52:03 It's not complete yet.

08:52:05 We want it to be.

08:52:07 But it's not quite ready

08:52:13 abstract classes are intended to be a base class.

08:52:39 It's not meant to be instantiated on its own.

08:52:53 So if we declare an abstract class.

08:52:59 We need to declare we have to declare a derived class to extend it

08:53:09 classes, methods, properties indexers and events.

08:53:18 So, Members, except a field can be abstract.

08:53:48 And finally, to finish that up.

08:53:52 A member

08:53:58 marked as abstract. Well hello Come on, where's my spacebar when I needed.

08:54:06 muscle must be implemented

08:54:14 by a nun abstract.

08:54:28 My fingers are all over the place.

08:54:31 Non abstract classes that are derived from the base class.

08:54:59 Alright, I'm going to go ahead and save this screenshot as well.

08:55:04 And then I'm going to grab this information on abstract because I want to keep looking at abstracts for a few minutes.

08:55:14 But let's go ahead and kind of start fresh.

08:55:25 There we go.

08:55:34 So if we try and break our understanding of abstract into three China chunks.

08:55:53 Because we're going to deal with abstract things in different ways.

08:56:00 And we said that we could have an abstract class,

08:56:10 we could create an abstract method.

08:56:20 Or we can create

08:56:24 an abstract property.

08:56:39 So we're abstract class.

08:56:43 It can't be instantiated we can't create an object of abstract class type.

08:56:52 It's not complete, it's not a full thought it's not a complete thing.

08:56:57 It's like saying I want to create half of an orange.

08:57:02 They're not made that way.

08:57:04 They come as a complete unit. You can't grow half of an orange, you can grow a smaller orange, but you're never going to look at the tree and find an orange perfectly cut in half, just growing there, it just doesn't happen.

08:57:20 Alright, so our abstract classes are the same.

08:57:23 We use them

08:57:26 to help us grow other things, but it cannot be instantiated on its own.

08:57:43 In abstract class may contain

08:57:47 abstract methods and accessories.

08:58:11 But it must provide an implementation.

08:58:18 For all interface, members.

08:58:30 And I know we haven't really talked about interface members.

08:58:34 It's on our to do list.

08:58:36 For now we'll just accept that an abstract class has to contain all of its interfaces that it needs.

08:58:44 Finally, it cannot include the sealed modifier.

08:59:00 We haven't talked about the sealed modifier.

08:59:06 We haven't talked about it yet, because we're still working through inheritance and sealed is going to modify inheritance. But we're not there yet.

08:59:23 An abstract method.

08:59:34 An abstract method is a virtual method, implicitly.

08:59:47 We've said that a virtual method is one that we plan on overriding later on in our code.

08:59:57 An abstract method is just that it's one that we plan on overriding later on, for it to be abstract, it has to be virtual.

09:00:12 We can only use an abstract method.

09:00:21 In an abstract class.

09:00:27 If you have a single abstract method.

09:00:31 The class must be abstract.

09:00:51 An abstract method does not carry a body.

09:00:56 It doesn't carry any action doesn't carry any commands.

09:00:59 It says simply.

09:01:01 I want to fill this in later.

09:01:04 So it carries know implementation.

09:01:08 So we don't need the curly braces. You could decide to have the parameters.

09:01:15 But we're not going to give it any set of commands to go with it yet.

09:01:26 Right, getting down to it, an abstract method can only have implementation in a derived method.

09:01:52 using the override keyword, which makes sense, right, we have to use the override keyword to denote that we are overriding a virtual method.

09:02:13 An abstract method is implicitly a virtual method.

09:02:16 So we have to use the override keyword to continue that abstraction, and that extension.

09:02:30 Finally, it cannot

09:02:34 have the static

09:02:41 or virtual keywords.

09:02:52 So, if something is static.

09:02:55 We're not going to be able to change it defeats the purpose of it being abstract, it already is virtual by being an abstract method.

09:03:06 So we're avoiding that kind of redundancy by saying, don't use virtual or static.

09:03:19 Finally abstract properties.

09:03:23 This one's easy.

09:03:35 Everything about the methods, is the same.

09:04:09 And the only thing that we're going to see different is that abstract properties are going to be written with their curly braces, but they still don't have an implementation.

09:04:31 Let me go ahead and save this for us.

09:04:46 Excuse me. You're absolutely right.

09:04:49 Rue.

09:04:51 Thank you, Jacqueline.

09:04:59 Okay.

09:05:01 I want to try to do a really simple example.

09:05:07 And it's going to be so simple that we're not going to do it in VS code.

09:05:12 We're just going to kind of do some pseudo code here in our whiteboard.

09:05:19 So I'm going to clear this for us, so we have a blank slate to start with.

09:05:26 And if I want to start by creating a class. Let's assume that I'm inside of the right name space.

09:05:36 I need a class declaration.

09:05:50 my base class, I want to be abstract.

09:05:54 What I'm going to do is expect to extend it and expect to inherit it.

09:06:00 So I'm going to declare that this whole class is abstract.

09:06:09 And when I do that, I can still create fields.

09:06:15 I can create

09:06:21 values.

09:06:44 So we would recognize that those are fields.

09:06:51 And then I can try to create a abstract method.

09:06:57 I still need an access modifier.

09:07:06 So I'm going to use public.

09:07:13 I'm still going to need the modifier I am using now.

09:07:19 Just my general modifier so this is going to be an abstract method.

09:07:30 We're still going to declare a return type.

09:07:42 So here we could say it's void.

09:07:45 And then we still need to name.

09:07:59 And if we had any parameters.

09:08:01 So here I could just say,

09:08:05 abstract method, no parameters.

09:08:08 And because I'm declaring it as an abstract method.

09:08:13 It does not get a body of text to follow it, it does not get a body of code.

09:08:18 So we end the line with our semi colon and line character.

09:08:28 Then we can create some getters really quick.

09:09:00 And that's it. That's our abstract class.

09:09:03 That's our base class.

09:09:09 So now we can move on to a derived class where we actually implement all of these things where we make use of them and where we actually say what the heck is going to happen when we call our method of abstract method, and where we use it.

09:09:27 So let's come down a little bit, and we need a new class.

09:09:37 We know it's going to inherit and extend the base class.

09:09:53 And the first thing that I feel like doing is handling the abstract method that we started to define, but we never gave a body. So let's do that.

09:10:07 It's going to be a public abstract void.

09:10:13 I can even copy most of this to start, drop it down here.

09:10:18 But I'm no longer going to use the abstract modifier keyword.

09:10:26 Who's got the right keyword for me. What do I need

09:10:35 override. That's all I gotta go with override.

09:10:40 All right.

09:10:42 We can get rid of that end line character because now I do want to put a body.

09:10:50 Let's keep chugging along.

09:10:56 And I want to deal with my protected variables. I have an int x, that let's just add a little bit to it.

09:11:10 And I've got a why that let's do the same thing will just add a little bit to it.

09:11:16 And I can come and close my body there.

09:11:19 That's all it takes. I'm doing something.

09:11:24 We have extended and overwritten. The abstract class.

09:11:31 An Overview overridden that abstract method.

09:11:35 Let's keep going because we have two more methods that we can create a override method for.

09:11:47 We did the abstract, we still have an inch wide and an inch x, we can do something with those getters.

09:11:57 So if we come and do a public.

09:12:01 And again, it needs to be an override.

09:12:16 I just need to get

09:12:21 in return.

09:12:25 My value of x.

09:12:29 But we can have fun with it I mean we can return whatever you want, you can do calculations here if we wanted to do we could do an x plus 10, just for funds these.

09:12:41 Why would you. I don't know why not.

09:12:47 And we can do the same thing with our

09:12:52 into why.

09:13:26 See if we can get that visible.

09:13:35 We have to delete some of my white space. Oh man.

09:13:37 There we go.

09:13:42 So we, In our base class,

09:13:48 we say we are going to have an abstract method.

09:13:55 But I don't know what it's going to be yet.

09:13:57 I know I want one, I just, I don't know what it is yet.

09:14:02 So we can say that by using abstract modifier.

09:14:06 I know I'm going to have an int x. Get her method.

09:14:13 But I don't know exactly what yet.

09:14:15 I know I'm going to have an into why, but I'm not sure.

09:14:32 And if we tried to create just a base class object.

09:14:38 It wouldn't work it would just fail.

09:14:41 He would come out with an error that says, can't, can't do that. it's an abstract class we can't just create one.

09:14:50 We have to use the derived class here.

09:14:59 I'm going to go ahead and save that image. I'm also going to copy the body of code into one of our files,

09:15:11 so that it is like text editable.

09:15:14 For everybody as well.

09:15:19 While I do that.

09:15:21 Does anybody have questions comments concerns.

09:15:27 Rude things to say about the instruction so far as anybody's cat decided to take over their bedroom, I don't know what do you got.

09:15:58 nothing yet.

09:16:00 Okay, I'm gonna go ahead and stop our recording them.

09:16:10 And we've got about 45 minutes, until lunch.

09:16:16 I'm going to push the code that we wrote this morning.

09:16:21 Our base derived. And the second derived classes, as well as the abstract class that we just wrote on that whiteboard.

09:16:33 It's all going to get pushed into a new folder on trainer code that will just be called overloading overriding.

09:16:42 It's going to be there for you guys to play with, and experiment with and try and implement new things, see if you can break it and then fix it.

09:16:53 all that good stuff.

09:16:56 Give me just a second to go through my end here.

09:17:06 Yep, that's fine.

09:17:25 Okay. If you had to your trainer code repos and do a get pull, you will see that the overloading overriding folder. Now exists in Wk three.

09:17:40 I'm going to open up some breakout rooms and give us room to spread out and test things at your own speed.

09:17:54 What I'd like to try. Everybody tried to do what I would like to see everybody attempt.

09:17:57 Yesterday we made our own classes that extended employee, an extended person.

09:18:03 I'd like to see everybody try to create an abstract class and extended and define the things that we've abstracted.

09:18:15 It doesn't have to be pretty. It doesn't have to be anything in particular.

09:18:23 I just want you guys to get a chance to play with these modifier.

09:18:28 With abstract with virtual with override so that we can get comfortable with using them.

09:18:42 Sound like a plan to everybody.

09:18:45 All right, it, let's see at about 1250.

09:18:51 I'm going to pull us all back into the main room. We can reconvene, see if anybody has any questions. If anybody ran into trouble.

09:18:59 But other than that. The rooms are open.

09:19:02 If you guys will give me another five minutes to get our whiteboards up into the trainer code repo. It'll be in the whiteboard folder with titles.

09:19:15 Right.

09:19:15 Megan if you want to jump down to the office I can give you the rundown of what we covered yesterday, and we'll see if we can get up to speed.