So in the last lecture, you learned how to inject dependencies using properties. This approach works in a lot of scenarios, but some dependency injection frameworks can't inject dependencies via properties. So an alternative way is to inject dependencies via constructor

parameters.

So here in our constructor, I'm going to add a parameter IFileReader and then I'm going to change this public property to a private field, because we're going to inject a dependency at the time of creating a video service object. So, I'm going to delete the getter and the setter.

Change the public to private, and finally to follow the naming convention .net we name FileReader to underline lower case f FileReader

and one last change, here we are always initializing this file reader to a new FileReader which is the new implementation. We want to have flexibility here. So, we should replace this with the argument that we get in the constructor. fileReader. So we call this approach constructor injection. However, there are a couple of other problems with this approach, and I'm going to show you work arounds. One problem is that because we have changed the signature of this constructor, chances are, we have broken some code somewhere else. So if you look at the program class, you can see this slide is broken, because now we have to pass an argument to this constructor. So, one way is to pass a new FileReader object. But I'm going to show you a better way. So, let's delete this.

Back in our video service, increase a separate constructor that doesn't take any parameters. So, a default constructor. In this constructor, you can set a file reader to a new FileReader object. So in our production code, we're going to use this constructor, but in our test code, we're going to use this constructor so we can pass a fake FileReader. With this approach, we can see our code is no longer broken. Now we can make this even a little bit better. So, we can combine these two constructors into one constructor. So, I'm going to delete the default constructor, and make this parameter optional. So we set this to null, and that means wherever we create a video service object we don't necessarily have to pass the file reader object as an argument. So if you look at the program class, you can see the code is still working, we don't have a compilation error, now back in the video service, I'm going to change this line to something like this. So fileReader, ?? new FileReader. What does this mean?

That means the fileReader is not null, you're going to use that to set this private fried, otherwise, if it's null, you're going to new up, real FileReader object here. So in our production code, in this case here, we're creating a new video service without any arguments to the constructor. In this case our argument is null. So, with this implementation,

you're going to set fileReader.new. FileReader object. In our unit test, we're going to pass a fake fileReader object. I'm going to use that to set this fileReader field.

Okay? So back to our unit test, we no longer have these properties, so delete, instead when creating a video service object we pass a new FakeFileReader. Let's run the test one more time, okay so still passing, excellent. So this approach is what we call constructor injection. And as far as I know, most dependency injections out there do support constructor injection. In the next lecture, we're going to have a quick overview of dependency injection framework.