



This lesson is all about nesting interfaces.

So just without lesson description, I'm actually telling you that we have the ability to nest interfaces

inside other interfaces.

And the interface we have created so far is what we call an elementary interface, and this means it

only contains basic interface components, we haven't nested it in any other interface and we haven't

tested any other interface inside of it.

Now, when an interface nest's another interface inside of it, we actually call it a compound interface.

And an interface which is nested inside another interface is called a component interface.

They all sound very similar, don't they just remember a compound contains other interfaces, a component

means it's a component of another interface.

As you might be able to work out, a component interface can also be a compound interface, which tells

us we can test interfaces within interfaces, within interfaces and so on.

It can just keep going on and on.

Now, let's move down to the bottom left of the slide here.

To create an interface that is made up of one or more existing interfaces, you have to use the interfaces

statement so you can see here in the middle of the text, we have the interfaces statement.

And it appears in the middle of our interface definition, no, it can only appear within an interface

definition and not a class definition.

So the basic structure is interface without an interface name.

We have our normal interface components, such as data and methods and so on.

And then we specify interfaces and then we list out any interfaces we want to include inside.

And we closed it off with the end interface statement.

Now let's move over to the right of the screen and consider this example.

And in this scenario here, I've declared interface one and you can see that I've labeled it up as a

component interface, and that's because Interface One is a component of interface to.

And when you look at interface two, I've also labeled this one as a component interface as well, because

Interface two is a component of interface three.

But let me step back.

Interface two is also a compound interface because we can say interface to includes interface one.

So this one can have two names are exactly the same time.

When we come down to interface three, we can see it's declared with the interface, his statement,

and it's including Interface one and interface two.

Now, it's important to realize that when interfaces are nested inside of each other like this example,

you may think that we have built a hierarchy here, but it doesn't work that way.

All the component interfaces are all on the same level.

So here, interface one actually becomes just a component of interface three.

This means whenever we see a compound interface, it can only contain each component interface once.

And I've tried to show this down below with this syntax, when we refer to Interface one through interface

three, we actually say interface three, then the components selector, interface one.

You see, we're not specifying interface to at all.

Interface one belongs solely within interface three.

Here, the wrong syntax, we don't have this hierarchy being built, so we don't say we can refer to

interface one by going into phase three to interface to to interface one.

That's incorrect.

An interface, I'll say again, only exists once as a component within a compound interface.

So now you understand about component interfaces and compound interfaces.

Let's have a look at how we can implement them in our code.

And when we write code to implement compound interfaces in a class, each component interface expands

the public section of the class with their original names.

And we will never have to worry about any naming conflicts, because, like I mentioned previously,

each interface occurs only once.

Let's have a look at this code in detail and we'll follow it through and see how we can call the methods

of the different interfaces.

So first of all, we're declaring an interface a an interface A contains a method called a one.

Then we have interface B, an interface B is a compound interface because it includes interface.

But notice also it declares its own method called a one.

Then Interface C.

It's pretty much the same as Interface B, and that includes interface and has its own methods, one,

then interface D includes interface B and interface C.

But notice that doesn't declare it so method it just including interface B and C.

So now let's go to the right and have a look at the code on how we implement this.

We declare a class and in the definition of the class we have the public section, which is where we

place our interface.

And you can see the interface we're declaring here or using, should I say, is interface D.

So just because of that, that will tell you where actually including also interface C, B and A would

include DNA because A is part of B and C, and we're including B because it's specified in interface

D and C, because it's specified in interface as well.

So we're including all of them here.

But that rule says an interface can exist only once with inside a compound interface.

So when we come down to the implementation of our class and we try and call these individual methods,

you can see that we can call interface a method, a one just by specifying interface and using the component

selector and specifying the method name.

Then we have interface B component selector and the method name again, exactly the same with interface

C, which issues a component selector specify the method and that's it.

So it's quite easy and that shows you the components of a compound interface only exist once.

And it also demonstrates how we access the methods of each of those interfaces when they're brought

in or when we just bring in the one compound interface.