

This lesson is focused on static constructors with regards to inheritance.

Now, the same principles apply for static constructors as they do, for instance, constructors, when you declare a static constructor in a subclass, it redefines the static constructor of its superclass.

And just as with an instance, constructor redefinition.

We do not use the redefinition addition when creating our code.

There is a slight difference with the procedures involved when redefining static constructs as though in that you do not have to call the static constructor of the superclass.

This is all taken care of by the runtime environment automatically to ensure that the static constructors are called in the correct sequence.

If you recall, when we discussed static constructors earlier in this course, we stated that the first time a class is addressed in our program, the static constructor is executed.

Well, exactly the same applies with our subclasses.

Our static constructor is only called once during the program execution, which is triggered the first

time our subclass is referenced by the system.

And because of inheritance, the system will automatically search through our inheritance hierarchy

to find the next highest superclass whose static constructor has not been executed and then executes

them.

So we take care of it for us.

Let's have a look at an example program and see how this works in action.

So let me switch to the ABA.

Repetiteur.

Here we go.

And let's hand the code.

First thing is I'm going to go to the top.

What I want to do is come down to the Ford class definition.

And I want to ask the class method in here because one doesn't exist.

So within the public section, I just create a little bit of space.

And then this is where I'm going to define our class method.

Remember, we have to call it class constructor.

And then what I'm going to do with this method is actually come down here to the protected section and

that's some class date, remember?

A class constructor can only work with class data.

Oh, I'm going to do is create a new attribute that holds some little log.

So just some kind of log information type character with a length of 40 next.

I'll go down to our Mercedes definition, and as you can see, it's empty because he's just inheriting

from Ford.

So what I'm going to do is create some code to redefine the cost structure within this Mercedes class.

Start with the public section.

And once again, I want a little bit of in fact, what I'm going to do is just copy this whole line

and I can use it again in the Atlanta class.

Now, note here, I'm not going to want a.

Another data item within the Mercedes class.

I'm just going to inherit.

This piece of cloth data from the fourth class itself, it's now down to the Audy.

Define the class method again.

So in this case, this is our third level, so we're redefining the cost method in the Mercedes class,

which redefines the method in the fourth class.

And once we've done that, we need to come down to the implementation section and start adding the code

in their.

So method class constructor, the logic.

Or just a normal statement, should I say so we have the catalog attribute is going to equal.

The fourth class constructor has been used, so just a bit of text that will be stored in the attribute

and then we will write it out to the screen.

Nice and simple, and we'll just close out that method and let me copy that to the clipboard, I'm always

a fan of saving my fingers extra typing.

So how now?

I was going to say we will now move down to the Mercedes class implementation, but of course we've

got no code that Upwey because we didn't implement anything.

So let me copy this whole class.

Here we go, and then we'll rename it to Mercedes.

Plus, constructor will change this to Mercedes to indicate that this has been used by the code.

We don't need this method constructor.

That's all we need in there, then we'll go down to the county and pop the same sorts of coding.

And I will add just some comments.

Did the same here.

Super.

So I think that's about it.

Let me save the code.

Check my code.

Excellent, and everything seems OK.

I'm surprised I didn't make a syntax error there, but good, let's test it out and see what it does.

So in theory, when we create this object, Ford, remember, is going to execute the class constructor

as the first thing.

So we will do that.

And then then we will go on and execute the Mercedes class, create object on my output, because these

are class methods.

They shouldn't really do anything with regards to inheritance, class method calling, going back up

the hierarchy, I will stick a breakpoints on and then you'll see what I mean.

So the first line we're going to create the object, the fall costs, let's step into the code and the

first time we do that, it runs the class constructor.

So as we step through that, it's going to write that to the screen, then that finishes.

But we're still in the create object for Ford now.

So when we hit next, it then has to execute the normal construct and not the class.

So it goes through that code and we saw that in the previous videos.

Now it's done that we can go into the mine Mercedes class.

So when we trying to execute that.

It's done two things at once here.

The first thing he did was to see if, well, it it did a check to see if the fourth class constructor

had already been executed and they had so there was no need for it to execute.

So now it just comes to the Mercedes class constructor and executes that.

And then because the Mercedes class inherits normal, constructive from fault, it then executes that

code, then we come down to my Audi.

Oh, look, I've left a spelling mistake in here.

I should have changed this to Audi, first of all, so we could see printing out on the screen.

But it's going to print Mercedes twice.

But never mind.

As you can see in the code, it is calling the clutch constructive for Audi.

So does that.

Ben is calling the superclass than it is this one and then finishes.

So what do we have?

Well, the code is executed each class constructor when the various objects were called in turn.

So first of all, default was cold and it executed default constructor, then Mercedes's, and it executed

the Mercedes class constructor, then the Audi and so on.

But what they didn't show us was the inheritance tree working in the system, even though it was in

the background checking to make sure the next higher level class class constructor had been executed.

We can actually see evidence of that, could we?

Apart from the fact that it actually didn't go to execute it.

So let's change things around.

First of all, let me change this to Audi.

I would go and then we're going to change these round.

So instead of calling the Ford class first, let's call the Audi one first.

I mean, line things up again.

We go now activated, who's got some matters?

OK, so I've got some full stops and things somewhere.

Here we go.

Colma footsteps OK, what active now.

Now this time we're going in reverse order.

So we're going to create the cost for Audi first.

Now this is where the inheritance tree kicks in with our cost constructor's let's see in action and

we will discuss it as we go along.

So we come down to the Audi first.

Now as I step into this, the system is going to check to make sure the higher level class constructors

all let me say that differently.

The higher level classes in our inheritance tree have their class constructors executed first, and

that's what it's done.

It's gone all the way to the top of our tree because he knows that.

Well, first of all, the Mercedes cost structure wasn't executed, but then it did a check to say,

well, Mercedes is inherited from Ford, so I need to check to see if the Ford constructor has been

executed as well because he hadn't.

This is the first time anything in our super class and subclasses has been touched in the program.

It then has to execute this highest level class in the hierarchy, three to four class constructor.

So that's what he's doing.

So the first thing is this foaled one, he will execute this.

Then he steps back down to the Mercedes.

The next level in our inheritance hierarchy executes them.

And then finally it will get to the Audi class constructor.

And once it's done all the constructor's, it will go about its normal business of calling the normal

method constructor's, the instant constructor's.

So here is going to execute the superclass constructor and would just choose half six, then this one.

And then he goes on to Ford.

So now when it goes on to Ford, it will check to make sure the Ford constructor has been executed and

the house because my Audi class did that.

So when we do it, it doesn't execute the class constructor.

It goes straight to the instance constructor.

Then we can just execute.

And the same thing for Mercedes, then everything's written out.

So this shows our full constructor was executed, first of all, because the way inheritance works,

it goes to the top of the tree, looks for that Coskun structure and executes it all in sequence.

Let's move on.