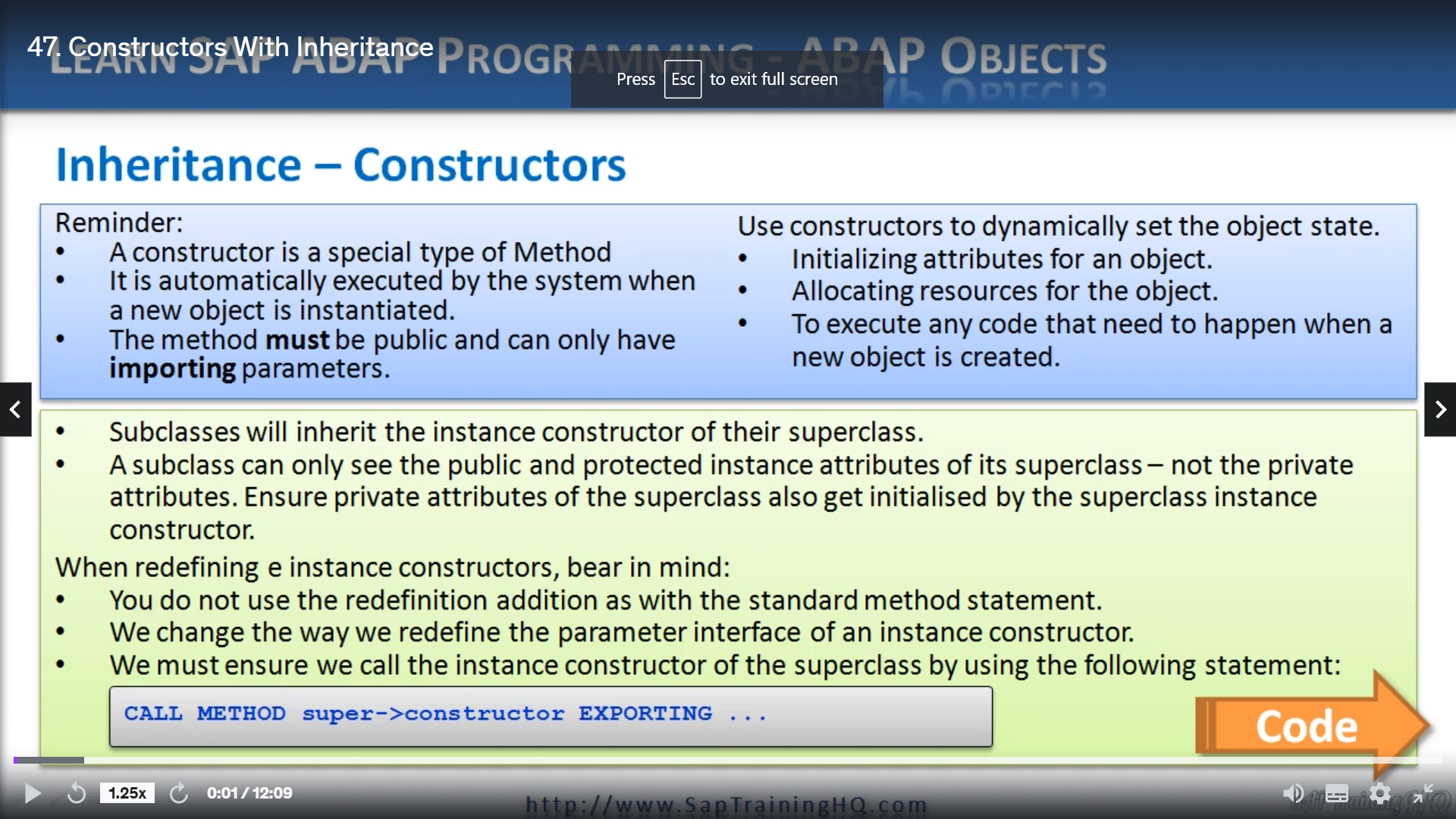
REPORT ZYNY\_CLASS\_05.  
\*----------------------------------------------------------------------\*  
\*       CLASS ford DEFINITION  
\*----------------------------------------------------------------------\*  
\*  
\*----------------------------------------------------------------------\*  
CLASS ford DEFINITION.  
  PUBLIC SECTION.  
    METHODS constructor  
      IMPORTING  
        p\_model TYPE string.  
  
  PROTECTED SECTION.  
    DATA: model TYPE string.  
  
ENDCLASS.                    "ford DEFINITION  
  
\*----------------------------------------------------------------------\*  
\*       CLASS mercedes DEFINITION  
\*----------------------------------------------------------------------\*  
\*  
\*----------------------------------------------------------------------\*  
CLASS mercedes DEFINITION INHERITING FROM ford.  
  
ENDCLASS.                    "mercedes DEFINITION  
  
\*----------------------------------------------------------------------\*  
\*       CLASS audi DEFINITION  
\*----------------------------------------------------------------------\*  
\*  
\*----------------------------------------------------------------------\*  
CLASS audi DEFINITION INHERITING FROM mercedes.  
  PUBLIC SECTION.  
    METHODS constructor           " No REDEFINITION addition is used.  
      IMPORTING  
        p\_model TYPE string  
        p\_wheels TYPE i.  
  
  PROTECTED SECTION.  
    DATA: wheels TYPE i.  
  
ENDCLASS.                    "audi DEFINITION  
  
\* IMPLEMENTATION  
CLASS ford IMPLEMENTATION.  
  METHOD constructor.  
    model = p\_model.  
  ENDMETHOD.                    "constructor  
ENDCLASS.                    "ford IMPLEMENTATION  
  
\*----------------------------------------------------------------------\*  
\*       CLASS audi IMPLEMENTATION  
\*----------------------------------------------------------------------\*  
\*  
\*----------------------------------------------------------------------\*  
CLASS audi IMPLEMENTATION.  
  METHOD constructor.  
    CALL METHOD super->constructor  
      EXPORTING  
        p\_model = p\_model.  
        wheels = p\_wheels.  
  ENDMETHOD.                    "constructor  
ENDCLASS.                    "audi IMPLEMENTATION  
  
\* Our program starts here.  
  
DATA: my\_ford       TYPE REF TO ford,  
      my\_mercedes   TYPE REF TO mercedes,  
      my\_audi       TYPE REF TO audi.  
  
START-OF-SELECTION.  
  
  CREATE OBJECT: my\_ford      EXPORTING p\_model = 'FOCUS',    " SUPERCLASS  
                 my\_mercedes  EXPORTING p\_model = 'C-CLASS',  " SUBCLASS - NO REDEFINITION  
                 my\_audi      EXPORTING p\_model = 'A4'        " SUBCLASS - INTERFACE REDIFINED  
                                        p\_wheels = 4.  
  
  uline. 

So now we get onto the topic of constructor's with inheritance.

Now, I will say, if you followed me so far with the inheritance up to now, you're doing a great job

because it's not easy if you're not familiar with object oriented programming.

Now, this topic may stretch your brain a little bit more, but I will try my very best to keep it as

plain and simple as possible.

So earlier in the course, we discussed constructor's and does a reminder, a constructor is just a

special type of method that is automatically executed by the system when a new object is instantiated.

Now, there are two types of constructor's instance, constrictors and static constrictors, remember,

static constrictors I may only use just to initialize static attributes of a class.

Now, constructors are really useful because they allow us to initialize attributes of an object as

soon as an object is instantiated.

When we create an object, our instant constructor will immediately carry out the instructions coded

within it.

Now, static constructors are a little bit different.

They do not need an object to be instantiated.

Therefore, instructions within a static constructor are carried out when a class is referenced for

the very first time or when an object is instantiated from that class.

The very first time.

So now comes the question.

So what happens with constructors in the case of inheritance?

Well, it's not too dissimilar from normal methods within a class ALDE, so classes will inherit the

instance constructor of their supercluster.

This means that when a subclass is instantiated, all the logic from the superclass will be inherited

in the subclass to carry out any instance, attribute initialization tasks as required.

Now, one thing to bear in mind is that a subclass can only see the public and protected instance attributes

of its superclass.

It doesn't have any access to any of the private attributes.

So when the subclass is instantiated, we have to ensure any private attributes of the superclass also

get initialized by the superclass instance constructor.

Now, to make sure the constructor of the superclass is executed, which in turn will initialised a

superclusters private instance, attributes, we need to make sure the redefinition of the constructor

itself is coded appropriately.

Now, before we take a look at doing this, there are a few rules we must follow when we redefine instance

constructors that differ from the standard redefinition of normal methods.

Now, these are we do not use the redefinition additions like we do with the standard method statement.

And we change the way we redefine the parameter interface of an instance constructor, and then finally

we must ensure we call the instance constructor of the superclass by using this statement here.

Call method super dash greater than constructor exporting, and then we list of parameters that.

And we have to bear in mind that if I subclause does not redefine the construct of the superclass,

we must fill in the non optional input parameters defined within the superclass itself.

And also, if we choose to define a completely new construct for our subclass, we must fill in the

non optional input parameters for the subclass and then if needed.

And sure, we call the Supercluster constructor filling in its non optional input parameters also.

Now, it sounds a bit confusing, doesn't it?

So let's have a look at an example program to show the sequence of processing carried out by inherited

instance constructor's.

So let me swap to the guey and I'll load up a brand new program.

So let me try to bring you one.

I've got some ideas about this, but I'll tell you what, I will pause the video here.

There's no need to watch me coat this because it might get quite lengthy.

So I'll pause the video, put the codeine in, and then we can review it line by line.

OK, so here we are.

I put a program together and we'll go through it and then we'll see how the program rooms in debug mode

so you can see the sequence of events.

So first of all, I've created a brand new class called Foord, so I'm using the car scenario again.

Now, the full class is one of three classes that I've defined in this program.

We have one super class and two subclasses.

And if we go down and have a look, you'll see here we have Difford, which is the super class.

Then we have a subclass called Mercedes which inherits from Ford, but then we have an Audi which inherits

from Mercedes.

So we actually have three levels of class inheritance here.

And the fault class as an instance, constructor with an input parameter of model.

And from the forecasts, we derive the Mercedes class and as you can see, it just inherits the standard

construct from the Ford class.

Then we have the Audi class, which inherits from Mercedes subclause.

But the Audy class redefines the instant constructor and contains two input parameters.

And just before we go into Tuesday about debugger, I want to step through this code line by line to

explain what's going to happen, to explain the different steps the system will take when creating objects.

So first of all, if I scroll down, I will show you at the bottom where we got the class ending.

I'm creating three objects.

And when we run this code to create the fold object, the constructor for the fourth class is called

during the create object statement.

And initialize is the model attribute using the model parameter passed in.

So we're going to pass in focus.

If we go to the top, that's going to set P model to the word focus.

Now this is our parameter, so we must go down and actually see the implementation of the constructor

and show that our attribute model will be set to the value stored in that parameter.

Focus now, next, when we run the next line.

And the Mercedes object is instantiated in exactly the same way as it just uses the basic inheritance

and there's no redefinition of the instance constructor, the instance constructor of a super superclass,

the Ford class is going to be executed and the model attribute is initialized from this model parameter

pastie.

So it's going to contain C dash class.

Then when we execute the code for my Audi and create the my Audi object, the Audi class has its own

instance constructor.

Let's take a look at that as Ford, Mercedes and Audi.

So the Audi class has its own instanced constructor and we have two parameters.

So this has a different interface.

We pass in two parameters, the model and wheels.

And when doing so, the constructor of a super class is called first before the attributes of the Audi

class are initialized with these parameters passed.

So if we go down to the implementation for the Audi, we can see that another special line here, we

can see that when the constructor is called, the first thing we have to do is make sure we call the

constructor of the Audi superclass.

So we use call Method Superdog greater than constructor.

Once the Supercross constructor has executed, we then continue processing the rest of the code within

the hour, the constructor, and pay special attention here when we're calling this super constructor,

we have to pass in the required parameters.

So here a model equals pay model.

So that's an overview of what's going to happen.

OK, so you should have this code to be able to download and study.

I'm going to go through it in debug mode now so we can step through it step by step.

But please take a look at it and just make sure you understand the flow.

OK, so the first thing is I'm going to set to break point.

Tell you what I need on the line, don't I, just so it doesn't run off the end.

Yeah, we get.

So we're going to test so the first time I run, this creates object statement to create the misfold

object passing in focus.

The first thing it's going to do.

Is run the constructor for the fourth class, all basic stuff.

Next, we're going to create the object from my Mercedes.

Now, remember, Mercedes is inherited from Ford, so because there is no redefinition of the construct

there, it calls the constructor of its super class.

So as we can see, each run in default constructor.

Next, we're going to create the Audi object when this runs.

This is a little bit different.

What it does to begin with is because it steps into the Audi constructor, which is a redefinition of

its, what shall we say, super super class.

So remember, the super class of the Audi class is the Mercedes class, but because there's no constructor

there, it then goes up a further level of the inheritance hierarchy.

So it has to go to the Ford super class.

So if I execute, we're going to call the method super constructor.

And then it's popped up, so it executes the code for the fall constructor.

And then pass it back down into the Audi constructor to finish the processing their.

And then we come out and he does the underline to the screen.

So that demonstrates how the constructor's of the various level of classes in the hierarchy are executed,

so I suggest you load this upon your own system, have a play around with it and make sure you understand

the actual flow.

Let's move on to the next lesson.