  \*&---------------------------------------------------------------------\*  
\*& Report  Z004\_INHERIT  
\*&  
\*&---------------------------------------------------------------------\*  
\*&  
\*&  
\*&---------------------------------------------------------------------\*  
  
REPORT  z004\_inherit.  
  
\*----------------------------------------------------------------------\*  
\*       CLASS vehicle DEFINITION  
\*----------------------------------------------------------------------\*  
\*  
\*----------------------------------------------------------------------\*  
CLASS vehicle DEFINITION.  
  PUBLIC SECTION.  
    METHODS: gofaster,  
             writespeed.  
  
  PROTECTED SECTION.  
    DATA speed TYPE i.  
  
ENDCLASS.                    "vehicle DEFINITION  
  
\*----------------------------------------------------------------------\*  
\*       CLASS car DEFINITION  
\*----------------------------------------------------------------------\*  
\*  
\*----------------------------------------------------------------------\*  
CLASS car DEFINITION INHERITING FROM vehicle.  
  PUBLIC SECTION.  
    METHODS: refuel.  
  
  PROTECTED SECTION.  
    DATA fuellevel TYPE i.  
ENDCLASS.                    "car DEFINITION  
  
\*----------------------------------------------------------------------\*  
\*       CLASS boat DEFINITION  
\*----------------------------------------------------------------------\*  
\*  
\*----------------------------------------------------------------------\*  
CLASS boat DEFINITION INHERITING FROM vehicle.  
  
ENDCLASS.                    "boat DEFINITION  
  
\*----------------------------------------------------------------------\*  
\*       CLASS vehicle IMPLEMENTATION  
\*----------------------------------------------------------------------\*  
\*  
\*----------------------------------------------------------------------\*  
CLASS vehicle IMPLEMENTATION.  
  METHOD gofaster.  
    speed = speed + 1.  
  ENDMETHOD.                    "gofaster  
  
  METHOD writespeed.  
    WRITE: / 'The Vehicle Speed is: ', speed LEFT-JUSTIFIED.  
  
  ENDMETHOD.                    "writespeed  
ENDCLASS.                    "vehicle IMPLEMENTATION  
  
\*----------------------------------------------------------------------\*  
\*       CLASS car IMPLEMENTATION  
\*----------------------------------------------------------------------\*  
\*  
\*----------------------------------------------------------------------\*  
CLASS car IMPLEMENTATION.  
  METHOD refuel.  
    fuellevel = 60.  
    WRITE: / 'You have just filled up your fuel tank.'.  
  ENDMETHOD.                    "refuel  
  
ENDCLASS.                    "car IMPLEMENTATION  
  
START-OF-SELECTION.  
  
  DATA: car1 TYPE REF TO car.  
  DATA: boat1 TYPE REF TO boat.  
  
  CREATE OBJECT: car1,  
                 boat1.  
  
  car1->gofaster( ).  
  car1->writespeed( ).  
  car1->refuel( ).  
  
  boat1->gofaster( ).  
  boat1->writespeed( ).

In this lesson, we're going to go through a coding solution to the test that I set you in the last

lesson you had to create a vehicle class and then from the vehicle class, you were going to create

two subclasses, one for car and one for boats.

You were going to add different attributes and methods and then finally create the objects, instantiate

them from your calling program.

So let's go straight to the op ed and let's see how I would do this.

Here we go.

So I've created a blank program to start with, and then I'll just start creating the classes, I won't

talk too much.

I'll just get the coding done and then we can review everything.

So there we have the basic class defined, we have our definition and we have our implementation.

There's nothing new here.

You should be familiar with all this.

The next part was if I flip back to the slide.

We have to define a class A subclass for car and for boats, so we'll do the cowman first.

We have to define additional attributes called fuel level in the protected section and another method

called refuel.

So back at the top,

let's create iconoclasts.

And this time, we need to use the inheriting from unspecified the class we're inheriting from and it's

we're going to inherit from vehicle.

And just like normal, we define public sections, private sections, protected sections and so on.

Well, the first thing is we need a public section because we have to define a method in the public

section.

It refuel.

And then in the protected section for the circus

or some data, and it was fuel level.

Standee took.

OK, so that's the definition.

Now we'll go and do the implementation, so just as normal, we specify class and specify the class

we're working with and then implementation just the same as normal.

And just like any normal method, sorry, any normal object, we just declare the method and we put

in the code.

Easy stuff.

So we need to do is close the method out and then and close another tidy up.

Excellent.

Now, the next task was to define a boat class if you member define a boat class that inherits from

vehicle, but importantly, we don't need any additional attributes or methods.

So I don't know if you got a bit puzzled on this one, but I mean, it doesn't get any easier than redefining

a super class when you don't need any more attributes or methods, because all we have to do is go back

to the top, will go to the bottom of our definition area.

Create some space,

and just as with a car, with a boat

definition, inheriting from specified vehicle, close it down with costs and that is it, because we're

not adding any attributes or methods.

We don't need to declare any sections.

We don't need to count any methods or attributes.

So that is it.

That's the implementation side.

And then because sorry definitions side and then because again, we got no methods, what we've nothing

to implement, it's just going to inherit the code from the vehicle class so we don't even need to code

anything for the implementation side.

It's all done.

So it's actually quite easy to tidy it all up and do a check sample and there is no public protected

and private section.

Oh, of course, I didn't declare it, did.

I would just stick the public section in the.

Pumps full stop.

Pretty to save another Syntex check.

You may only define methods between costs, implementation and costs.

What have I done?

Let me just scoot up a little bit.

Oh, yes, look, I didn't close out the glass from the vehicle.

And the Closs.

Another go that's better syntax check in is now verifying everything is good.

Now, the final task for me to do is actually create some code that will create these objects.

So let me do that, go down to the bottom of the program and as usual, stay out of selection.

Multiclass, some objects.

Notice I'm not creating an object for vehicle here and let's actually instantiate the object.

So that will create both objects at the same time, which is changing our create object statement,

I just flip back to the slide just to refresh my memory.

OK, so first of all, let's call the car methods and the first one must go faster.

Then we have to right speed bumps.

Then we had a refuel, do believe, and we go.

And if you remember, for the car method that go faster and right speed inherited from the vehicle class,

but then we created a brand new method of refuel for our car, subclause them for the boat.

And we have a right speed, if you remember, we didn't add any other methods or attributes to the boats

so close.

So that is all we need.

Let's do a syntax check.

Everything looks good.

Let's give it a run.

Fantastic.

The way the vehicle.

I don't know what that means.

It should be vehicle.

Let me go change up.

But the speed is one.

You've just filled up your fuel tank and of course speed is one.

Let me change this.

Oh, boy.

There we go.

That's better.

So the vehicle speed is one you've just filled up your fuel tank and the vehicle speed is one.

And let's have a recap of exactly what we've done here that go to the top.

This vehicle class contains a protected attribute called speed and a couple of public methods.

Go, go faster and right speed and then created two new classes, car and boats.

And both of these classes are inherited from vehicle.

This means both classes inherit the attributes and methods from the vehicle class now because the speed

attribute was declared in the protected section.

Our subclasses have access to this variable, but if we were to put it in the private section, our

subclasses wouldn't be able to access this attribute at all.

And if it went in the public section, well, even though everything would work, we would be opening

up our program to security risks because any calling program will be able to change the attribute without

using the methods of the class.

Now for the car class, I then went on to specify if I can find it.

He would go specify an additional method called refuel.

And an additional attribute called fuel level.

For the boats, of course, this was a bit different, I didn't need to add any additional attributes

or methods, so I actually didn't need to implement any code for boats.

All I needed to do was define the definition for boats and make sure it was inheriting from vehicle.

And then I finish things off by creating the code to instantiate objects based on the car class and

the boat class, and then call the car methods and the boat methods.

Now, let's have a look at these in a bit more detail as well, because with car, I called to go faster

method.

So when we go up, we can see that the go faster method was actually defined in the vehicle class,

not in the car class.

But the syntax we use.

And we want to call a method for a subclass, we reference the subclass the object we created and then

call the method that it's inherited from its parent class vehicle.

For right speed, exactly the same thing, that method comes from vehicle as well.

It's somewhere up here, let me find you and then when we come down to this third Matkal refuel, well,

this is a bit different because in the sockless we actually define the refuel method.

The refuel method isn't part of the parent class at all.

It just belongs to the sockless.

And when we go down to boat, will be cost boats didn't introduce any new methods or any new attributes.

When we call the methods to go faster and right speed, they're the methods that are inherited from

the vehicle class.

And notice we didn't create a vehicle class at all.

We just created subclasses.

The carcass and the boat cluster inherited from vehicle.

So how did your code compare to this, what did you think of it?

I think it's pretty easy to understand.

It's pretty straightforward and hopefully you haven't had too much difficulty understanding the inheritance

part.

If you have go over the video again, play around with your code until it all makes sense.