



REPORT ZYNY\_CLASS\_09.  
\* Interfaces  
  
INTERFACE intf\_speed.  
  METHODS: writespeed.  
ENDINTERFACE.                    "intf\_speed  
  
\*----------------------------------------------------------------------\*  
\*       CLASS train DEFINITION  
\*----------------------------------------------------------------------\*  
\*  
\*----------------------------------------------------------------------\*  
CLASS train DEFINITION.  
  PUBLIC SECTION.  
    INTERFACES intf\_speed.  
    METHODS: gofaster.  
  PROTECTED SECTION.  
    DATA: speed TYPE i.  
ENDCLASS.                    "train DEFINITION  
  
\*----------------------------------------------------------------------\*  
\*       CLASS train IMPLEMENTATION  
\*----------------------------------------------------------------------\*  
\*  
\*----------------------------------------------------------------------\*  
CLASS train IMPLEMENTATION.  
  METHOD gofaster.  
    speed = speed + 5.  
  ENDMETHOD.                    "gofaster  
  METHOD intf\_speed~writespeed.  
    WRITE: / 'The train Speed is: ', speed LEFT-JUSTIFIED.  
  ENDMETHOD.                    "intf\_speed~writespeed  
ENDCLASS.                    "train IMPLEMENTATION  
  
\* Ou prgram starts here  
  
DATA mytrain TYPE REF TO train.  
  
START-OF-SELECTION.  
  
  CREATE OBJECT mytrain.  
  mytrain->gofaster( ).  
  mytrain->intf\_speed~writespeed( ).  
    
\*  mytrain->writespeed( ). error, can't use this syntax  
  mytrain->gofaster( ).  
  mytrain->intf\_speed~writespeed( ).

In this lesson, we're going to turn our attention to focusing on interfaces.

Now, we have seen how polymorphism is used through inheritance, using a single interface that is common

between a super class and its subclasses.

This is made possible because of the single inheritance model introduced in APAP objects.

Throughout this course, we have used the term interface to refer to how our programs interact with

methods of a class.

The method interface defines how our programs interact with the method itself.

Without understanding, we can now move on to discussing how we can define interfaces independently

of any specific class.

What I am talking about is something that is created very similar to a class, but with no actual functionality,

no actual code, no methods.

It's really just like a method signature or from a technical about perspective, like a super class

that cannot be instantiated.

The job of an interface in a rape object is primarily used to define common interfaces or protocols

for classes.

This means many different classes can use interfaces to fully or partially create the definition of

a class.

Let's have a look at the syntax.

First of all, you can see here on the right.

We start with the word interface, then we give it an interface name and then at the bottom we have

the end interface.

This is very similar to a class we can define data, class data, methods, class methods and so on,

even events and class events.

We will come onto events later on.

I know we haven't covered them, but they're coming up in a future video.

Even though it looks very similar to a class.

There are a few differences.

Notice we haven't used any visibility section, this is because components of an interface are always

considered public.

The reason interfaces exist is to expand the public interfaces of any implementing class.

So this makes perfect sense.

Also, interfaces do not have an implementation plan and there is no need for the definition part to

be defined either.

The most common use for interfaces is to add additional methods to the public interface of a class,

but it isn't just limited to that.

You can, if you wish to create interfaces that contain all the same components that you define in a

class.

So let's have a look at how we actually implement an interface.

Each class can implement one or more interfaces in its class definition.

And as you can see in this example, the interface has been included in the public section of this class.

The public section is expanded by the components of the interface.

To reference an interface component within a class, we must use the interface component selector.

And as you can see, this consists of the interface name and then we have the tilde symbol.

And then the interface component itself, by referencing components this way, this allows for a class

to contain interface components of the same name because they are differentiated by the prefix they

interface.

Tilde.

Let's have a look at a program and see how this works.

So let me load up the op ed.

OK, let's start coding this.

So to begin with, we start off with the word interface and then we specify the interface name, so

we'll do one I n t f underscore speed, we will stick with our normal sort of vehicle and car scenarios.

What we could, if we want to tell you what, let's change it, we'll use train instead and be adventurously.

Then, just like with a cloth, I'll define a method.

Remember, it needs to be method's because it's going in the implementation part of the class expanded

it and let's do the old right speed method.

Then all we need to do is end the interface.

So let's just define a nice, simple class.

I'm not going to use a train scenario.

A move defying the public sanction, remember, said that the interface expands the public section of

a class definition.

So now here, this is the point where we can actually bring in the interface that we've defined.

We can include it in the class definition itself.

It's all we do is specify interfaces.

Now we go.

And then we specify the interface name, oops, not that one.

That's it.

And then if we want, we can specify additional methods for the class that you remember from some previous

examples, we had a lousy go faster method.

Didn't we go faster?

And as usual, would define a protected section, puts an attribute in their.

And then we'll just end the class.

So it's pretty simple stuff, all we've done is define the interface where including just a method in

this interface, and then we're going to include it in this class definition.

The rest of the coding is more or less exactly what you've been doing when we've been learning about

other elements of APAP objects.

So we need to do is create an implementation for this class.

Modifying the method, this is to go faster.

I'd say plus five

and the method.

And this time, what we have to do is use our component selector, so we have to specify the interface.

Which is A.F. and disco speed.

And then we use the tilde.

And then we specify the method name.

Then we can write the code for the method.

And we'll close the method off and then close the class.

Let's turn to YouTube and a pretty princess.

Here we go.

So now all we have to do is go to our program.

And then the second one.

My train, then we go, we will call our right speed method, which, as you remember, is defined first

of all in the interface.

So we have to use the component selector again to our interface speed.

Tilde, and then the method name.

Save the code, let's do a syntax check, everything is good, so I'll activate the program.

And then let's give it a test.

We go to speed is five, so you can see it's very similar to creating just a normal class.

But we've extended things by introducing an interface which allows us to break out the definition of

our class into its own separate.

I won't call it object.

Well, let's just say interface.

OK, so we can declare interfaces completely separate to our class and then make use of that interface

in many different class definitions.

Let's move on to the next lesson.