In this lesson, we're going to continue looking at events and specifically we're going to focus on

events, handlers.

So now that we have defined events in a program, we need to implement some more code to handle these

events, so we must create events, handlers.

And you can create event handler methods for the class in which the event is defined or any other class.

Declaring the event a message is a very similar process to defining any other type of method, but we

just have to introduce a new addition.

So to declare the methods we can use, either the normal methods or methods statement are shown in the

example here.

As you can see, we give the method statement with a method name, but then we have the additions for

events, and that's when we specify the event that our method is going to handle.

And this method declaration statement statically assigns the hundredth method to a class or interface.

Notice here that the parameter interface this time can only be an important parameter and has to be

defined with the same name as the exporting parameter that was declared in the corresponding event.

And you'll notice also at the end of this instance method declaration, we have this additional senda

parameter.

This is specifically, for instance, events, it is a reference variable parameter of the type of the

class in which the event was declared and when the race event triggers the event handler.

In our instance method, this parameter is automatically filled with the reference to the Trigorin object.

This parameter can be very useful because if your handler method does import it, then you get access

to all the public visibility section elements of the Trigorin object.

Now, with that being said about event handlers, for me, it's always better to see in code.

So we're going to continue expanding on the program that we created in the last lesson.

And specifically, we're going to add event handlers to the code.

So as you can see here, we're going to add a brand new class called The Way to Class.

And the way to class will have a constructor method importing a parameter called Who?

And this is going to be a string.

It will have a method call.

Go see the chef that is declared as the event handler method.

So I've included the additional statements here.

We've got four events and then we're specifying the event itself of the chef of the class.

And then we're going to declare another method.

This one is going to be go see the customer.

And again, this is an event handler method.

So we add the additional four event and this is going to be called for waita again.

But this one is of the customer class.

And just before we go on uncowed, just note, we are not registering the event handlers just yet,

so even when we complete the code, we're not going to see all working as we finally wanted.

That will come again in the next question.

So let's swap over to the ABA party, another code.

So what I'm going to do a scroll to the top of the program and the first thing we need to do is declare

our way to class.

So on the customer, let's do the definition section.

You know, sitting in class emplace.

Right now, defund the public section.

The first thing we want to do then is declare some methods.

So first of all, we have to declare a constructor method.

For the constructive, we're going to be importing a parameter who so call this eye for importing and

then who declared it as a time string.

Next, we got another method to declare, and this one is going to be go see the Sheth.

And because this is a event hunger method, this is where we need to Anthos Edition's.

Not specifying for event, and then we have to specify the events that were registering this event handler

for, well, not registering by creating the event handler for and this is going to be called for later.

And then finally, we specify which class the event handler is specifically for.

So we use of in chef.

And then once again, I'll copy and paste instead of go see the chef, we want to go see the customer

and we're using exactly the same vente name.

But don't forget, it is actually a different event and we need to specify customer.

And remember, with this one, we're going to be importing the table number from the customer.

So importing

your table number.

Super lusting for this classic declaration, I'll try to create a protected section.

And declare some data inside and of course, this is going to be our tribute called Who?

That said, that class is now defined.

We can now go down and do the implementation.

Scroll down.

Here we go.

Let's pop it in here.

So the first thing again is define the class

implementation section and we declare the method.

And this one, first of all, is going to be the constructor, pretty simple for the constructor, we

just say in our attribute, who is going to be set to the important contribution or importing parameter?

Who should I say, close out the method and then do the next one?

Each one is going to be go see the chef.

And like some of the others, I'm going to write out this right out to the screen to show that this

is being performed and.

The who is going to be in name, so I'm going to be writing out the name and then we'll put some text.

So who?

It was to see the chef.

Out that method no more.

Instead of go see the sheriff, go see the customer.

Again, writes out who this town will say, go to see the customer,

but because we're putting in the table, no, we will add at table.

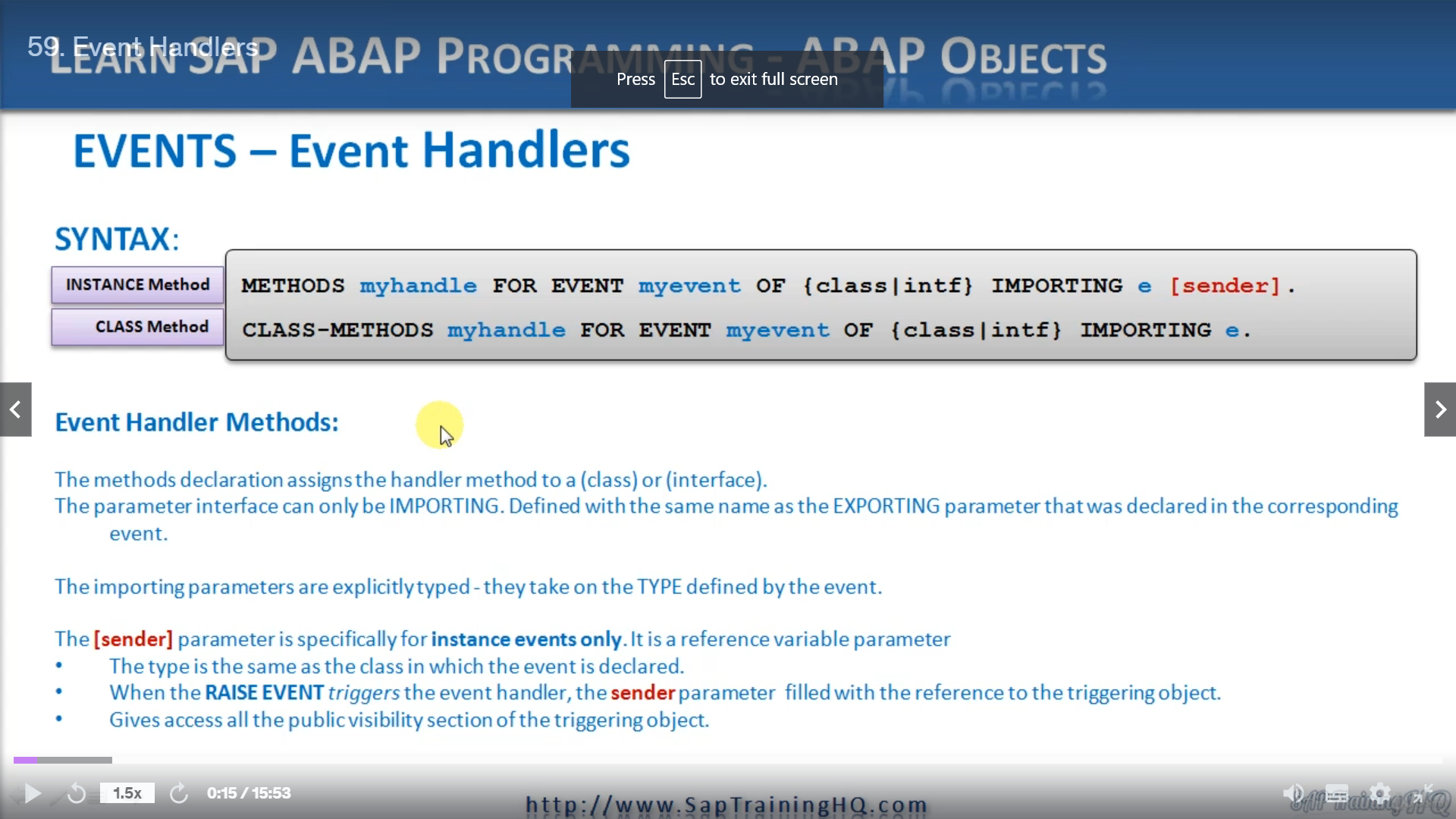
And specify the table number to make sure it's left justified just for presentation reasons.

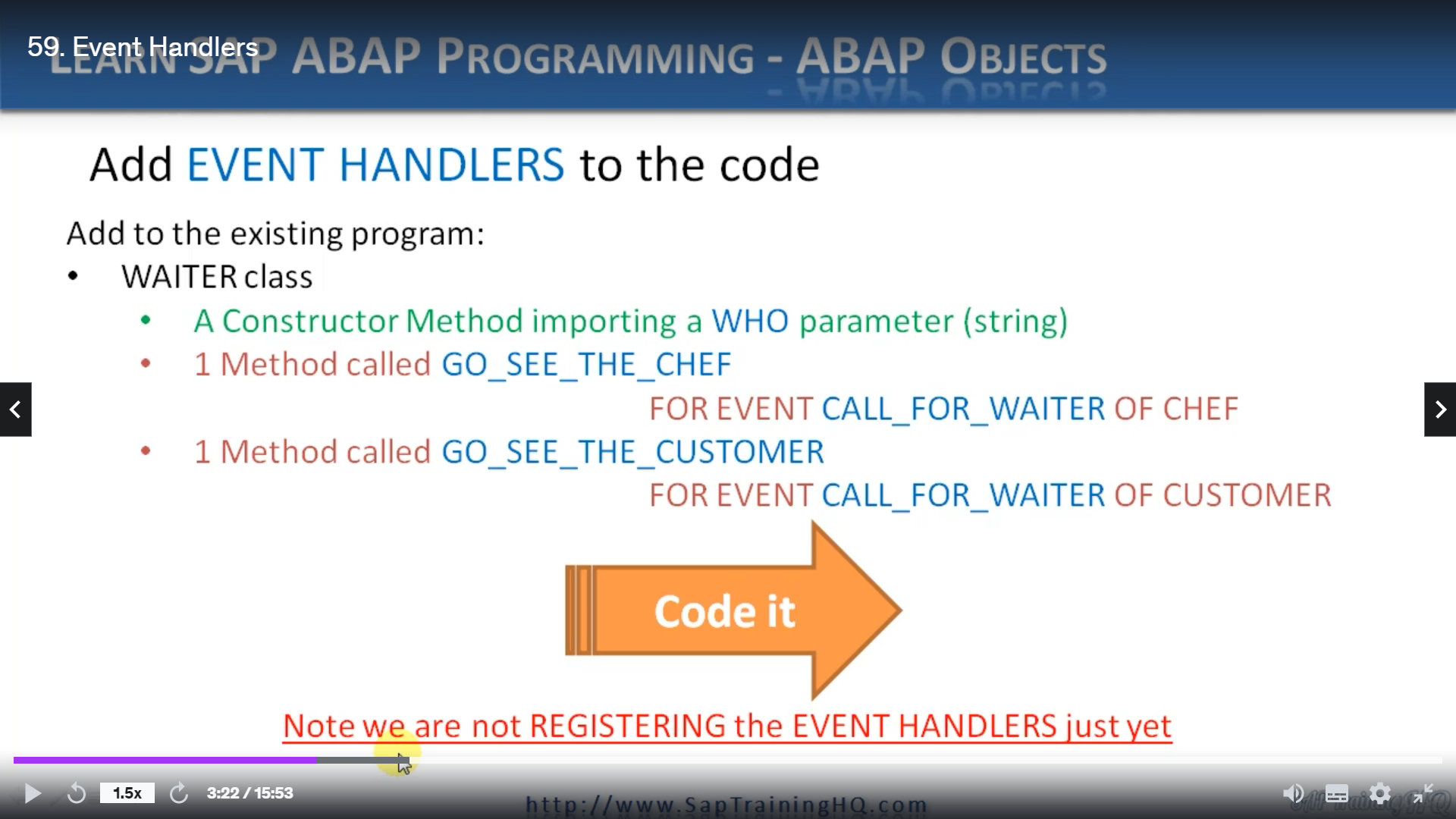
And then we can close out the method and finally, I'm just going to do and underline that this point

to help separate things on the screen and then we need to close out the class

and that's it for our class.

It is now fully defined and implemented.





REPORT ZYNY\_CLASS\_10.  
  
\*----------------------------------------------------------------------\*  
\*       CLASS chef DEFINITION  
\*----------------------------------------------------------------------\*  
\*  
\*----------------------------------------------------------------------\*  
CLASS chef DEFINITION.  
  PUBLIC SECTION.  
    METHODS: call\_service.  
    EVENTS: call\_for\_waiter.  
ENDCLASS.                    "chef DEFINITION  
  
  
\*----------------------------------------------------------------------\*  
\*       CLASS customer DEFINITION  
\*----------------------------------------------------------------------\*  
\*  
\*----------------------------------------------------------------------\*  
CLASS customer DEFINITION.  
  PUBLIC SECTION.  
    METHODS: constructor IMPORTING value(i\_tablenumber) TYPE i,  
             call\_for\_assistance.  
    EVENTS: call\_for\_waiter EXPORTING value(e\_tablenumber) TYPE i.  
  
  PROTECTED SECTION.  
    DATA tablenumber TYPE i.  
ENDCLASS.                    "customer DEFINITION  
  
CLASS waiter DEFINITION.  
  PUBLIC SECTION.  
    METHODS: constructor IMPORTING i\_who TYPE string,  
             go\_see\_the\_chef FOR EVENT call\_for\_waiter of chef,  
             go\_see\_the\_customer FOR EVENT call\_for\_waiter of customer IMPORTING e\_tablenumber.  
    PROTECTED SECTION.  
      DATA who TYPE string.  
ENDCLASS.  
  
\*----------------------------------------------------------------------\*  
\*       CLASS chef IMPLEMENTATION  
\*----------------------------------------------------------------------\*  
\*  
\*----------------------------------------------------------------------\*  
CLASS chef IMPLEMENTATION.  
  METHOD call\_service.  
    WRITE: / 'Chef calling WAITER EVENT'.  
    RAISE EVENT call\_for\_waiter.  
    WRITE: / 'Chef calling WAITER EVENT complete'.  
    ULINE.  
  ENDMETHOD.                    "call\_service  
ENDCLASS.                    "chef IMPLEMENTATION  
  
  
\*----------------------------------------------------------------------\*  
\*       CLASS customer IMPLEMENTATION  
\*----------------------------------------------------------------------\*  
\*  
\*----------------------------------------------------------------------\*  
CLASS customer IMPLEMENTATION.  
  METHOD: constructor.  
    tablenumber = i\_tablenumber.  
  ENDMETHOD.                    "constructor  
  METHOD call\_for\_assistance.  
    WRITE: / 'Customer calling WAITER EVENT'.  
    RAISE EVENT call\_for\_waiter EXPORTING e\_tablenumber = tablenumber.  
    WRITE: / 'Customer calling WAITER EVENT complete'.  
    ULINE.  
  ENDMETHOD.                    "call\_for\_assistance  
ENDCLASS.                    "customer IMPLEMENTATION  
  
CLASS waiter IMPLEMENTATION.  
  METHOD: constructor.  
    who = i\_who.  
  ENDMETHOD.  
  METHOD: go\_see\_the\_chef.  
    WRITE: / who, 'goes to see the Chef'.  
  ENDMETHOD.  
  METHOD: go\_see\_the\_customer.  
    WRITE: / who, 'goes to see the Customer at table:', e\_tablenumber LEFT-JUSTIFIED.  
    ULINE.  
  ENDMETHOD.  
ENDCLASS.  
  
\*Global Data  
  
DATA: o\_chef TYPE REF TO chef,  
      o\_customer\_1 TYPE REF TO customer,  
      o\_customer\_2 TYPE REF TO customer.  
  
DATA: o\_head\_waiter TYPE REF TO waiter,  
      o\_waiter      TYPE REF TO waiter.  
  
\* Program Starts Here  
  
START-OF-SELECTION.  
  CREATE OBJECT o\_chef.  
  CREATE OBJECT o\_customer\_1 EXPORTING i\_tablenumber = 2.  
  CREATE OBJECT o\_customer\_2 EXPORTING i\_tablenumber = 5.  
  
  CREATE OBJECT o\_head\_waiter EXPORTING i\_who = 'Sarah the head waiter'.  
  CREATE OBJECT o\_waiter      EXPORTING i\_who = 'Bob the waiter'.  
  
  CALL METHOD: o\_chef->call\_service,  
               o\_customer\_1->call\_for\_assistance,  
               o\_customer\_2->call\_for\_assistance.

So the next thing we need to do is, well, instead of extending that same data statement, OK, to

everyone, you were just to help separate things.

I'm going to create some objects now based on our way to class.

And first of all, I'm going to say headwaiter type reference to waiter.

And another one just saying waita.

And again, this is a reference to Waita on creating two objects at this point.

One represents a head waiter and the head waiter is going to be the guy who goes to see the chef, and

then the normal waiter will be the waiter who goes to see our customer.

So just before our method call, let's create some space.

Let's instantiate these objects.

And remember, we have to explore some parameters

of the actual names of who he calls, let's say Sarah.

The head waiter.

And then we'll create just a normal way to.

Let's line them up.

Now, I find you a pretty picture, and this is going to separate these lines again, isn't it?

I won't do that.

So our normal waiter is going to be called Bob.

Bob, the waiter.

And that is it for our code.

Let me bring these back together into a safe, I'll do a syntax check, then we'll walk through it and

just have an overview of exactly what's going to happen.

You may define methods between class implementation.

OK, put a button here.

Oh, yes, I've got the one in the wrong place it should.

That's it should be here before our end method stamens.

Let's try again, that's better.

Everything looks good and it hasn't reformatted my coat.

Excellent.

So now let's activate this and we'll step through it.

So back to the top.

So let's go through what we've got.

We have a chef class and in the chef class we have a medical call service and we have an event called

Call for Waiter.

When we come down to our chef class, we can see the whole service method implements the raise event

code for waiter.

So now when that gets executed, we will then have an event handler.

So if we come down to our waiter.

Oh, back up to our weights.

Should I say, on the declaration?

This is where we define this event handler.

So we're saying when the call for waiter of chef event, which is this one, is actually raised.

We're going to perform the go see the chef method to go down, to go see the chef method, and it's

going to right out to the screen.

Who goes to see the chef and who is going to contain Sara, the head waiter, so it will say Sara,

the head waiter, goes to see the chef.

And then if we do the same thing goes through for customer, we have a constructor, a call for assistance

method, this time not call for service.

And then again, we have a call for Waita event.

Which has the extra exporting parameter this time, so let's go down to the customer implementation.

Here we have the constructor, so it's going to be passed in a table.

No, that's fine.

Nothing new there.

Then the call for assistance method, when that gets cold, is going to raise the customer call for

later event.

So we know that our way to class has the event handler methods for both our chef and customer objects.

So if we go down, it's back up, isn't it?

If we go out of our way to definition here, we can see the colorful way to events.

Of the customer is then going to trigger this method.

Go see the customer.

So if we go down to the implementation for Waita, we see the go see the customer method.

And this one is going to say, I think it's Bob, the waiter, isn't it, Bob?

The waiter goes to see the customer at table and then the table name.

So now now it should start to make sense that the event handler is the method that handles these triggered

events and that will force the relevant method that responds to these events to actually get triggered.

So, again, we can test this out and have a look what happens, but you'll notice very little happens

again, we just get the same output.

And that's because if you remember, when I I go back to the slide, I have this little caveat down

at the bottom saying, no, we have not registered our event handlers yet.

Yes, we've set them up in our code and we've set this class.

And this method is now capable of handling events.

But we haven't said to our runtime system, I want to turn it on, I want to register this event now.

So it's listening.

Because what we can do with these events and with these event handler methods is turn them on and off

if they're on all the time.

Then there's times when you probably don't want them to be on.

You want to turn them off.

So we've got that flexibility.

So in the next video, I'm going to show you exactly how we do that.