

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, " we have to say this method is for that specfic event which is declared in class  
             go\_see\_the\_customer FOR EVENT call\_for\_waiter of customer IMPORTING e\_tablenumber.  
\*    METHODS: constructor IMPORTING i\_who TYPE string,  
\*             go\_see\_the\_chef,  
\*             go\_see\_the\_customer IMPORTING e\_tablenumber type i.  
    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.  
  ENDMETHOD.  
ENDCLASS.  
  
\*Global Data  
  
DATA: o\_chef TYPE REF TO chef,  
      o\_chef1 TYPE REF TO chef,  
      o\_chef2 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\_chef1.  
  CREATE OBJECT o\_chef2.  
  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'.  
  
  
\*  although we have declared event and the method for the event, it wont call that method even if the event is triggered, we have to  
\* sort of declare the class->method for the instance, we want to handle the event  
  
\* one waiter can serve several chef or customer each at a time  
  set HANDLER: o\_head\_waiter->go\_see\_the\_chef FOR o\_chef, "this is for the specific object,  
\*                                  as an example for o\_chef1 go\_see\_the\_chef method will not be called, although event will be triggered  
               o\_waiter->go\_see\_the\_customer  FOR ALL INSTANCES, " this is for all the instances of customer class  
               o\_head\_waiter->go\_see\_the\_chef FOR o\_chef2.  
  
  CALL METHOD: o\_chef->call\_service,  
               o\_chef1->call\_service, " here go\_see\_the\_chef method will not be called  
               o\_chef2->call\_service,  
               o\_customer\_1->call\_for\_assistance,  
               o\_customer\_2->call\_for\_assistance.

In this lesson, we're going to continue with the subject of events and specifically we're going to

look at registering event handlers and I'll just say this is the last lesson with events.

This is the one that will bring everything together and allow our program to start doing what we want

it to do, which is responding to raised events.

So now we have defined the event handlers in our program, we need to implement some code to register

the event handlers in order for each event handler to respond to a triggered event.

This is where we create this loose coupling, the loose relationship between the methods in Hungar objects

to the triggering objects.

Now, you can create this loose coupling by using the set handler statement and that you can see there's

two variations.

Again, once when we're creating an instance, an object and ones when it's a static event just based

on the class.

And I'll just mention for these static events, we must not use the full addition that's missing from

this example that you can see on the screen, because the registration applies automatically to the

whole class.

So we've no need to specify which object instance where registering for.

So let's go to the instance and we'll go through it in a bit more detail, you can see the list of instance

methods is listed here.

We have handled the object one, method one, and we object to method two.

And the list can go on and on and on so we can list different instance methods in different objects

and they can all get registered for one event.

And this is where we have the four and then we specify which object it is.

This is an object reference, the sending object reference that's raising the event.

And if we go back over here, we can see where listing individual methods, which means that each of

the methods that we do list responds only to the event for which it is declared.

Now, as well as registering event handlers, this statement can also unregister event handlers.

And we do that with this activation act addition at the end, if you want this onto the set handler

statement and you have activation, and then where we have this act parameter, you just fill in a space

that will unregister the methods for the event.

And finally, we also have the ability to register the listed handlers that we have here for all instances

of objects which trigger the event.

So instead of just listing the objects that trigger the event here, we can use all instances in its

space and that acts like a catchall for that type of object, which raises the event.

So now you know that we have to use this set handler's statement to register these events.

Let's go and do that.

Let's add them to our program and see all working together.

So swap over to the ABA, Ed.

And we'll get going now.

From the slide, you can see that we had to we don't have to do that much, all we have to do is say

system where you want to register some handlers.

So the coding is very minimal.

All we have to do is if I come down, let's say we're creating these objects, creating five objects

we're going to put in here before we make the method calls.

So that way, because we're going to register our event handlers in the system when we do meet the method

calls, our event will be triggered.

So.

As we saw with the syntax in the slide, we use set handler and we specify which handler methods we

want to register.

So it's going to be.

Our head waiter object, and it's going to be to go see the chef method, that's the event handler method,

and we have to specify which object is full and that's for object.

Oh, chef.

And the last one.

Oh, look, I've done a spelling mistake here.

They would go.

And the last one is customer one.

So not customer one.

It's waiter, object waiter.

And the method is go see the customer and we go and because we actually have two instance objects for

our customer, instead of listing the amount let's use for all instances.

And that's it, that's all the code we need.

So let me do a say, even a check, that looks good.

So now now we've registered our event handlers.

When we run through the code, these events are going to be triggered.

So let's start off with actually, you've seen all the code where we're creating the objects.

Nothing much is going to happen until we call our methods.

So I will put the breakpoint at this point.

Well, once I activate the program, they would go, let's done debugging.

So he would go the first line, we're going to set the event handler.

And you'll notice here when I do this, there's very little to see.

All it does is executed.

But you might want to know that actually behind the scenes, the system is managing a hidden table,

that storing information about which handlers are being registered against which events, which means

when the events are erased, it can then loop through that table and cold, irrelevant methods.

And also, if any code is included to actually unregister an event handler, then it can remove it from

that table.

Let's continue on and we'll finally get to the call methods.

Now we're going to call the chef call service method first.

So when we're going there, will write to the screen and then we'll raise the event code for waiter.

Now, when that's executed, it then jumps in to our way to class and performs that method to go see

the chef, so execute that, then he steps back out to where the event was raised and then carries on.

And we're calling the first customer.

And this one's a little different because we're passing parameters now.

So we're right to the screen and then we'll raise the event export exporting the table, no.

So now the GOCE customer method is executed in the way to class, will right to the screen do an online?

Then right back out to the screen again and come back out and then finally call the same thing again

for customer number two.

And that's it, it's complete and I think Cardinal Dolan in the wrong place in time, so so it's separating

the output a little more than I wanted.

But we can see in the output our chef Colin Waita event was triggered then.

This is the output from the event itself, said to the head waiter goes to see the chef and then the

event is complete.

And then for the customers, it's the customer calling waiter event.

And because Bob, the waiter, is the person that handles customer queries, Bob, the waiter goes to

see the customer at table number two and then we close the customer event and then it happens again.

So what I'm going to do is step back out of this.

And I'm just going to amend the underlying.

So we'll run the program again and then the output's fine, so that's it, that's events, that's as

far as I want to go with events.

And that really gives you a good introduction of how you can use them, how you can set up events,

define event handler methods, and how we registered the methods so that when an event is raised, our

event method, our event handler methods get triggered to carry out the necessary code.

Let's move on.