

And as you are familiar with ABAP, you will already know that we use ATTRIBUTEs in our programs and all an object is a special type of ATTRIBUTE.

An object contains attributes and or method's attributes are just the basic data types, such as integer or a character string and methods, I just modular sections of code that perform a specific task.

Let's consider an example.

Vehicles have attributes such as make model and colour.

These are just ATTRIBUTEs that will help describe the object.

A vehicle carries out certain tasks, certain actions, and we call these methods.

Typical methods for vehicle are things like accelerate, turn, and stop.

So, it's easy to understand what an object is, but what about a class?

A class is just a definition of an object.

A class declares the design for an object.

It describes how to create the objects themselves.

It's just like a blueprint for a house.

The class blueprint defines the complete design for the house when we want to create the house, we use to design, to create it, to make the object.

If we have a building company and we want to create a whole street of houses, we can use our one class as the design to build multiple houses, multiple objects.

So back in us about programming world, we create a class to enable others to create object instances, and when we create an object instance, what we are doing is storing an object, a reference in our object ATTRIBUTE.

All the object reference does is point to our object that is stored in memory and is designed on a particular class.

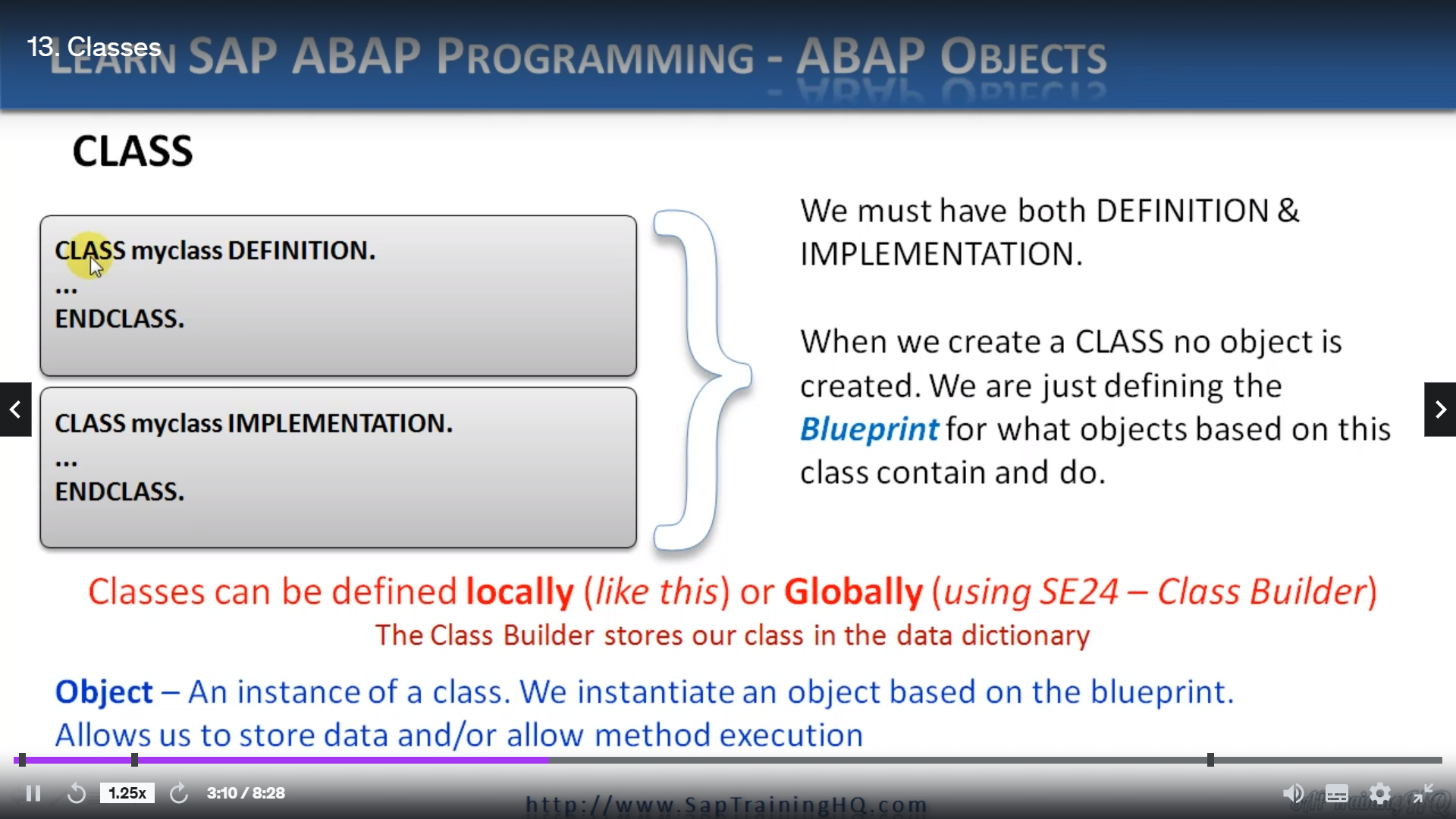
How we define the class determines the class interface that describes how different elements of your program can communicate with the object instance itself.

So, when we want to define a cause, what we must do in ABAP is, first, define its definition.

We declare the attributes and the methods interface.

Once we've declared the interface, the ATTRIBUTEs we're going to use and what we call the methods,

we then implement these we implement the definition of the methods.



So now let's have a look at the very basic syntax for creating a class with Bob.

As you can see here on the left top left-hand side, we use the key word class.

Then we define the class name, followed by the word definition, so when we define a class, we must define the definition first.

Whatever the class holds goes then between the class and the class Keyword.

Then we must define the implementation of the class, so we start with class again.

Give it the last name, followed by implementation, we implement the methods and then we have the endclass Keyword.

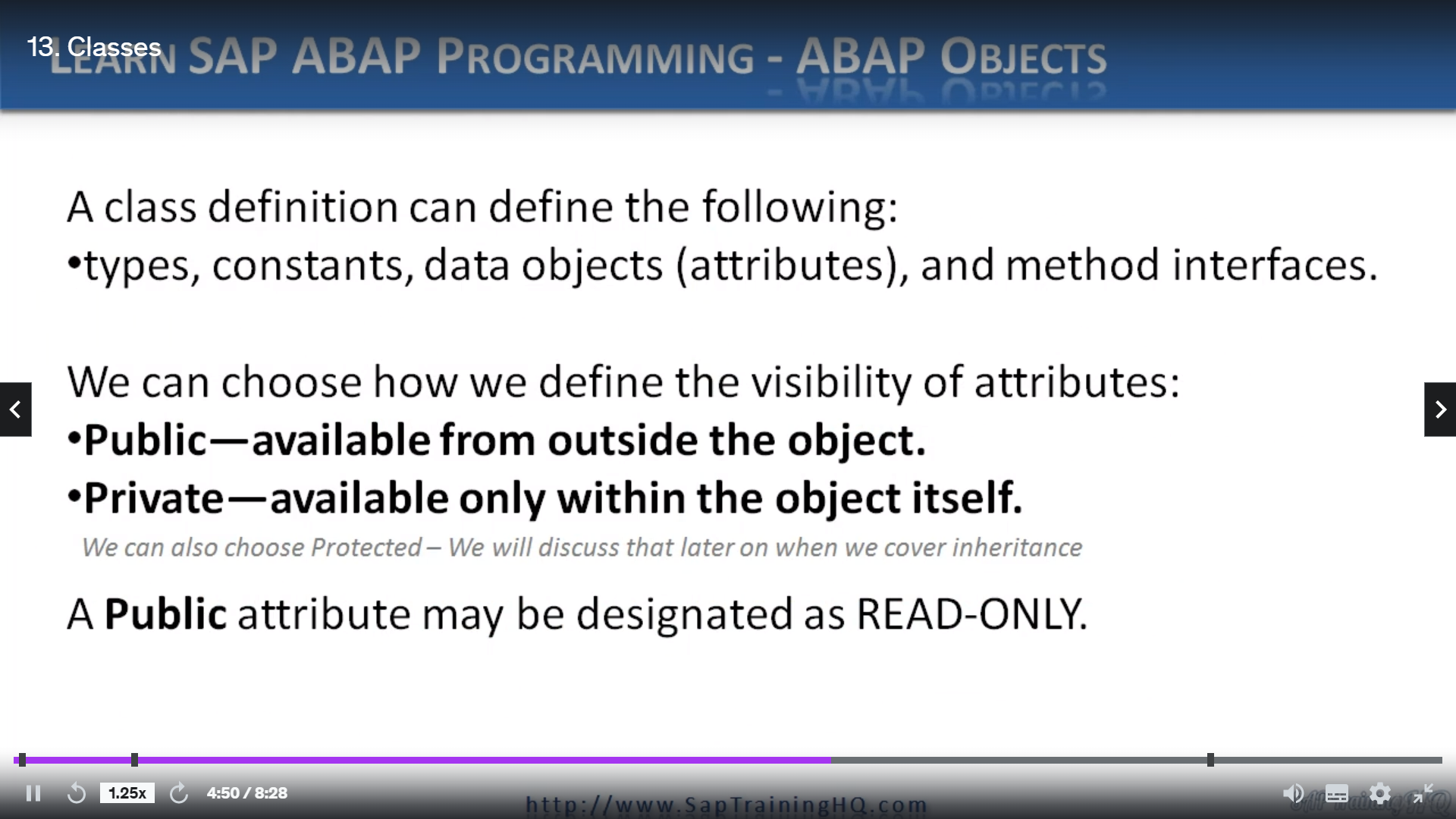
So, when defining a class, bear in mind, you've got to define the definition and then the implementation.

And don't forget, a class, when we create a class, no object is created, we are just defined in the blueprint for what objects based on this class contain and do.

Now, in ABAP, we can define classes in two ways, we can define local classes like we have here, where we just typed, according to our op ed, or we can define global classes.

And there's a transaction SE24, which is the class builder, which allows us to define global classes that can be used by any other program in the system or any other object in the system.

By creating global classes, our SAP system stores our classes in the data dictionary so we can call them, view them and edit them whenever we like.



Now, when we define a class, we have the option of defining types, constant data objects, which are attributes and method interfaces.

When we do this, we need to decide how we are going to make them available and what I mean is.

Do we make them available publicly or privately to the class or does a special one called Protected, which we will look at much later?

And this brings about the topic of visibility.

So, when we define a class, we must declare the visibility of each individual component that makes up the class, we have the option of choosing from those three different types of visibility.

When we define attributes, methods of our class, we place these insides to different visibility sections.

Now, public, whatever class elements we define in our public section, we are choosing to make these elements visible to any part of a calling program.

This means the calling program can't change attribute contents of an object declared using this class and call methods declared as public methods to.

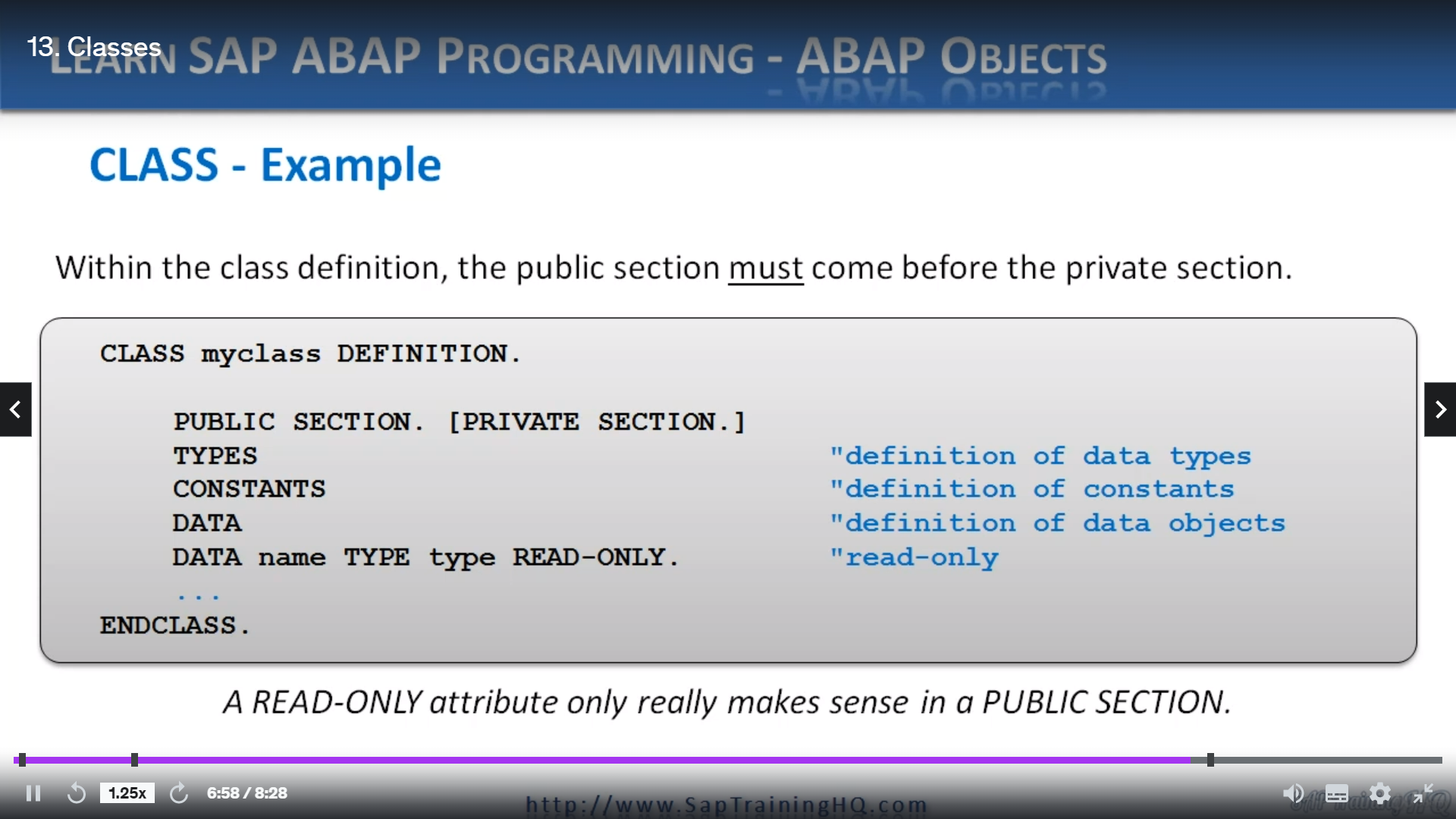
Any elements of our class placed in the privatization are only visible to the class itself, they're not visible outside.

This means a calling program cannot access the elements of an object directly.

This is where you would normally create a public method that would carry out actions on private elements

of an object.

And as I mentioned, we do have the protected section, which we will focus on later, but any elements defined in the protected section are used for object inheritance.



So, as you can see on the screen, within the class definition, the public section must come before the private section.

OK, so we start with the class Keyword, then we have the class name, then we have definition.

Then we declare the public section, that's the public section visibility then if we wish or let's say if we didn't have a public section, we can just go straight to the private section.

But within the public section, we would then be able to declare types constants, data object and so on.

Then once a definition is complete, we use the ENDCLASS keyword.

Now, you'll notice here in the sentence, I've got two data statements, and this is just to represent we can define normal data statements, but then we can say our data statement is read-only.

And if you use a read only attribute within your class, it really makes sense to only have it in the public section.

Because what we're doing here, we're saying this is a read-only ATTRIBUTE, the only our object, only our object methods can update.

We still want it made visible to external calling programs, but we don't want to allow them to change the ATTRIBUTE so we can use the read-only addition to declare data objects as read-only.