

But object-oriented programming is often considered superior to procedural programming due to the encapsulation of data with related functionality.

But if we try and explain it at a very high level, you know, why should you use object-oriented techniques?

Well, the answer really comes down to the encapsulation of data and functionality into a single object.

See, the idea of object-oriented programming is that we create objects that store data elements, think of normal variables and methods that perform action on the data or anything else in that matters, such as drawing to the screen, performing calculations on your data, or maybe calling other methods in other objects.

When we use object-oriented techniques and create objects encapsulating data and functionality, these objects can then be reused throughout your system many times over.

Now, we do have that backwards compatibility still to fall back on, so are procedural programs still work?

Fine, and a great part of the language is that Sap have done a massive amount of work for its customers to ensure that backwards compatibility is maintained throughout every single release of SAP systems that are based on ABAP.

So, any code from the 1980s will still work today on a modern installation.

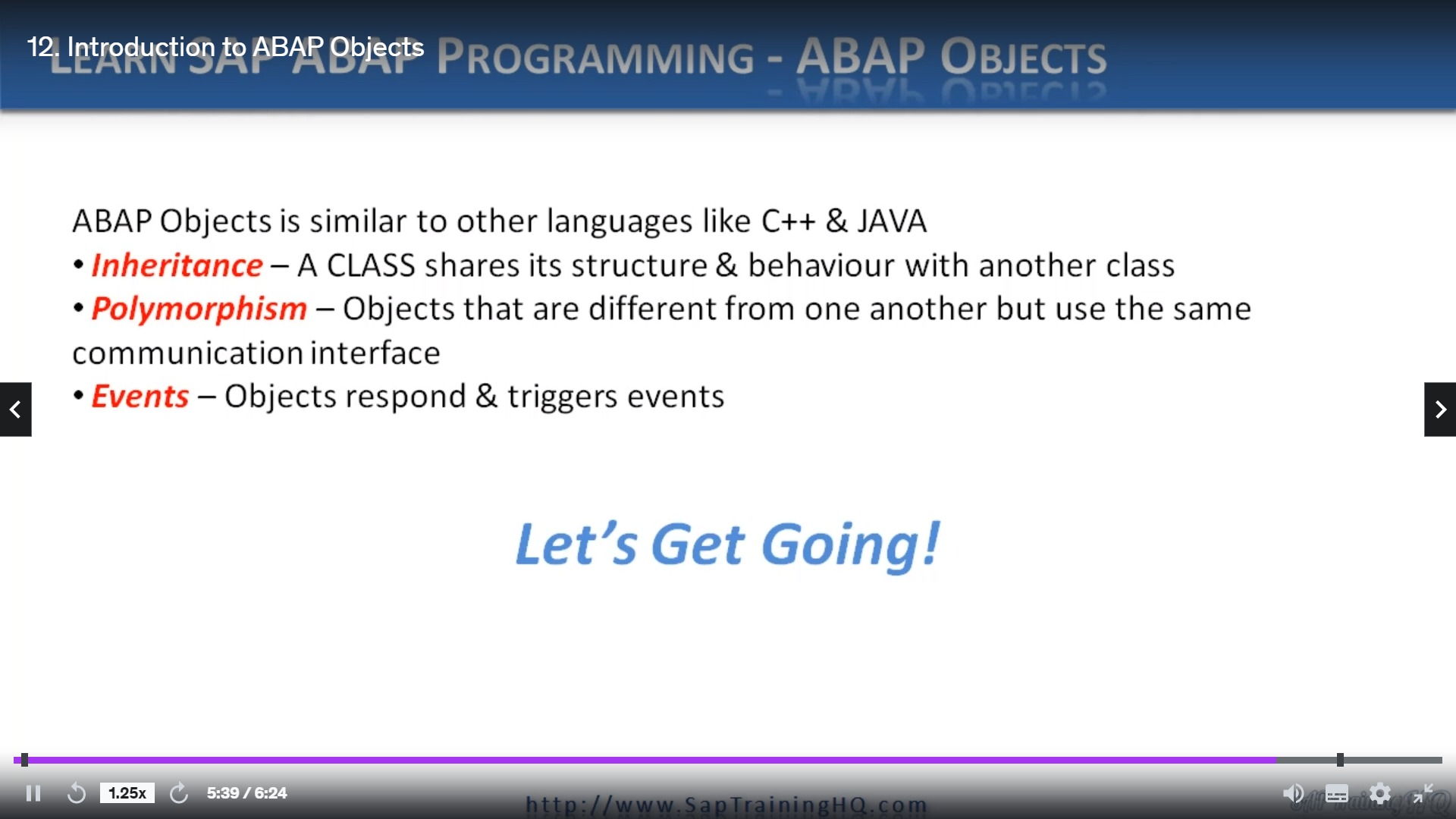
Now, when our object was first introduced, something in the language had to give a Sap. So, the decision to use a purified abap syntax when using objects and what this means is some language constructs are now considered deprecated and are not allowed when you're using them in the context of objects programming.

And you will come across these restrictions as you're right, your object-oriented code.

An example would be trying to use internal tables with header records.

This is not allowed, and the system will throw syntax errors and tell you can't use this type of coding when using old techniques.

But just note, you can still use the old ABAB statements that you may be used to for the procedural coding part.



So, APAP Objects is very similar to other object-oriented languages in that it supports inheritance,

which is just one class, she its structure and behaviour with another polymorphism, which means

different objects that are related to each other, have the same communication interface and defence

where objects respond when events are triggered in the system.

And you will see the object can trigger events themselves as well.

So, yep, Abbott is like any other object-oriented language, but we do have that benefit of backwards

compatibility.

Now let's start digging a little deeper and we'll focus on the first part that we need to understand.

And that's really classes.