

Now you have a good understanding of inheritance, it's time to turn our attention to polymorphism.

And polymorphism comes from the Greek for many forms, it is considered one of the most powerful elements

of object oriented programming, but at the same time, it's one of the most complex.

One of the reasons for this is that you must understand other elements of object orientation, such

as inheritance before you can fully appreciate polymorphism, if we were to try and sum up what polymorphism

is, we could say it's automatically allows us to carry out the same action method on objects of different

times, resulting in a different result.

Now, that's pretty vague, isn't it?

So let's consider a simple example to try and explain it a little bit better.

In our pop, if we have two variables and we wish to add them together using the plus sign, well,

if these two variables were both integers, the system would add them together and give us the result.

But let's say the first variable was a date.

This time, when we try to add the two variables together, we end up with a completely different result,

the system automatically carries out the same action, the addition on different variables resulting

in a different result.

We can use the same idea with our own classes and our own objects.

Here's an example.

If I define a bank account class in a financial application and it contains methods such as withdraw

and deposit, along with attributes such as account number and balance, I can then create more specialized

subclasses that inherit from it, like the current account class and a 30 day notice savings account

plus.

They both inherit from the superclass and share the same basic definition.

They will both have account number and balance and they can both withdraw money and they can deposit

money.

But let's say the 30 day notice savings account as a fixed interest rate attribute, that the current

account does not and it can get a little more complex, and then there could be a rule that says if

you withdraw funds from the 30 day notice savings account and don't give 30 days notice, you will incur

a penalty fee.

So the behavior of the withdraw method is a little more complex.

Up to now, this is just simple inheritance, but now we can introduce the concept of polymorphism to

let us work freely with objects that are being created from any of these classes.

I could now have an internal table holding many thousands of account object references that are a mix

of account types, and I know that I can call the withdrawal method on any one of them without knowing

exactly what classI was instantiated from.

And it will perform the correct behavior for each one.

So polymorphism gives us flexibility.

It gives us the ability to make our programs flexible, allowing the right action to be carried out

at the right time without the need for complex nested control structures such as, if else, endive

and the case and case statement in order to determine the correct dynamic times.

But our reference variables.

So this is great in theory, isn't it?

But I think the best thing to do is switch over to the other editor and let's go through an example

of exactly how this will work, and we will do that in the next lesson.