





This video continues on with the subject of inheritance.

Now, if we were to try and summarize inheritance, really what it means to a Zenab up at a high level, we can say something like when we define super classes, we make sure we only define the most common elements, the attributes and methods that all subclasses would share so that when a subclass is created,

it inherits all the super classes, attributes and methods.

One of the key benefits of using inheritance in our ABAB code is that we don't produce redundant code.

We don't have to repeat the same lines of code that were defined in our super class.

In our child classes.

We define common elements once and only have to define the different elements for our classes, for asset subclasses.

Now, if you are familiar with other programming languages, you may have come across the fact that in those languages a child class can have multiple subclasses.

Well, this doesn't happen in ABA.

ABCP implements single inheritance only if I went back to the previous slide, you can see in this inheritance tree, our parent class can have multiple child classes, but our child class can only have one parent.

This structure is fixed.

And like I mentioned in a previous video, in actual fact, you've already used inheritance when you've been creating some of these example programs in this course, because every single class in abar objects, which is not inherited specifically from another class, are actually direct subclasses of the real class called object.

But it's just that with AMAP, we don't have to add any code to tell our program that our class is inheriting from object.

It's already implied by the system.

We don't need to have that code at all.

The system knows that whenever we create class is going to inherit all the elements from a system class called object.

Now, let's move on and have a look at the syntax we use for declaring subclasses.

Now, it's important to realize that you can't delete any elements of a superclass when creating your subclass, but you can redefine the implementation of methods.

And this means if I were superclass performs a method with specific code already defined, we can redefine it to make it do something else in our subclasses.

Now, also, the visibility, the private and public sections and the interface, the parameters always remain unchanged.

We can't change how these are declared in our subclasses because we inherited from our superclass, our parent.

And at this point, this is where I'm going to introduce this protected visibility.

Up to now, we have just looked at private and public, but I did mention a couple of times that there is a protected visibility section and what this means is attributes or methods placed in the protected section of a class definition are visible within the class and it's subclasses.

So it's like a halfway house between the public section and private section.

Whatever we place in public is visible to a call in program and everything else within the class.

When we place things in a private section, it's only visible to methods of the classes self.

When we place things in the protected section halfway, it's not visible to the public, but it is visible to child classes, to these classes.

And when we define the different sections, when we're defined in a class, we have to declare our sections in this specific order.

We always have the public section first, then the protected if we need it, and then the private again if we need it.

So when we come here and have a look at the syntax for defining a subclass, it's quite easy, but you can see there's some specific keywords you need to include.

So we use class, as usual, to define the class.

We give our class a name, but then we say definition inheriting from and then we specify the superclass.

Then we have whatever code we want, followed by inkless at the end, so it's quite similar to what we've seen already, but we need to define the inheriting from UniSuper class.

Now, this slide does focus just on the visibility that I was mentioning just before, but let's go

through it again so we get it clear.

Whatever we define in the public section of a class will exist in any subclasses that are inherited

from it, as well as any additional public components that we additionally add to our subclass.

For protected this halfway house, all elements of a class will exist in the subclass that are inherited

from it, as well as any additional protected component added to the subclass.

And this is the important bit.

This is why I say it's halfway.

These components cannot be addressed externally through our normal components, selectors, our calling

programs cannot access these components.

Then we are private, and if we declare any elements of your superclusters private, they will not become

available in any subclasses.

They are not inherited.

A subclass will only see its own private elements.