





Now, let's start off with inheritance, and inheritance is a key concept in object oriented programming, and even if you've never written your own classes that used inheritance, you will be using it.

In fact, you may not realize it, but you have already used it in some of the example programs that we have created in this course.

Now, the best and simplest, easiest way for you to identify when you need to use inheritance within your programs is by using two simple words is a because inheritance describes on is a relationship.

And what I mean is think of how you describe things in the real world.

A car is a vehicle.

A bus is a vehicle.

As you can see on these lines, a boat is a vehicle.

A train is a vehicle.

It makes perfect sense.

But if we tried to say a car is a person, a car is a tree, it doesn't make sense at all, does it?

So we wouldn't use inheritance in those cases.

We naturally understand these ideas because this is how we talk in our day to day lives.

We can have multiple levels of this inheritance as well.

So again, back to this slide.

A car is a vehicle. And then when we take the next level down, a BMW five series is a car.

And that's the same as a sailboat is a boat and a steam train is a train with inheritance.

We are taking this idea and using it to help us with our code in ABAB. It helps us identify that there might be some shared attributes and methods between our objects.

And this can help us save some time making sure we do not keep reinventing the wheel by repeating the same code over and over again.

Now, when we use inheritance, we create a class and we create a subclass or a child class, they both

mean the same thing.

And we automatically have all the attributes and behaviors from the class that we're inheriting from.

And we call this the super class or the parent class.

So in our case, a vehicle is a superclass or parent class and the car and boat and train, it's a subclass

or a child class.

We bring in all the attributes and methods from the superclass, from the parent class, and then we

can start adding some extra information as we need.

Now, here, what you see on the screen is a simplified UML diagram representing pattern classes and

child courses and so on, because I want to get across exactly what inheritance means without Bob.

And if we start at the top, you can see we have a vehicle class and this is called a parent class,

a super class, and our vehicle class can have attributes and methods.

When we use inheritance to create child classes, well, the next level down here for our child classes

would be car, boat and train.

And in addition to these attributes and methods that the child class inherits.

We can also add additional attributes and methods.

So here you can see our car class inherits vehicle and it contains attribute one unmetered one.

But it's also added attribute to attribute three.

And by the way, the plus represents a public attribute and the minus sign represents a private attribute.

Same thing with methods, but it also contains additional methods as well.

And the same applies for boat and train.

You can see we've assigned additional attributes and methods to the child class.

Now, when we take one more step down, we can see our class has a child class of its own called BMW

and one called Audi.

So in this case, the vehicle has a child cos of car, but car is then the super class of BMW.

And BMW is the class of car, you see how the relationship goes.

When we look at boat, boat, house, child cases of sail and speed, and then train has the electric

train and the speed train, and you'll notice down here the different entries for additional attributes

and methods for the child classes just show that you can add any number of attributes and methods you

lied to your child classes or even none at all.

So as you can see with the electric train, one attribute has been added, but no additional methods.

It's just going to make use of the methods of train and vehicle.

So when you step back and look at this picture, you can now tell that actually the vehicle level is

a more general class.

It doesn't contain all the detailed attributes of car and BMW because it's a very general parent class.

It doesn't make sense to hold specific attributes or specific BMW attributes.

It's too high a level for those to exist.

Further down the inheritance tree we go, we start to get more specialized.

So when we get to the final child classes, they contain the individual attributes and individual methods

that just apply to that level of our inheritance tree.

Now, if we think about S&P and the S&P world, well, in example, there could be the various S&P documents

that were using the system at a high level.

We can say we have documents that represent different types of data and the document could be classed

as the parent class.

But when we break things down into a finance document that would then become a child class, but then

as we know, we can break finance documents down even more so the finance document can be a superclass

to a lower level document like an accounts receivable document.

When you start to think of that, you can then think about all the different relationships that exist

in an SAP system.