Now that we've already talked about abstraction and encapsulation and seen it in action, let's move on to the next pillar of OOP which is inheritance. And inheritance,we're talking about it in programming refers to when a class is able to inherit properties and methods from their parents.

So you for example might have inherited your eye color from your parents or you might have inherited certain behaviors like you are a neat freak and you really really need to keep everything super tidy.

You might have gotten that from your mom or your dad.

Similarly all classes can inherit from a superclass or a parent class and it can inherit properties as well as methods.

And this is important because when we're abstracting our program and we're creating lots of different modules lots of different components that all take care of a specific task.

It's likely that we will have some duplication in their functionality.

So for example our pastry chef and our chef will probably have some similar functionality.

They probably both know how to boil some water.

They both probably have a method for baking something but they might do it slightly differently.

So in order to prevent having to rewrite lots of code when we're abstracting into lots of different classes and separate modules, this is why inheritance is really key when you're creating objects. To demo this,I'm gonna go into a empty DartPad and I'm going to create a new class.

Let's say that I create a Car class and my Car class is going to have some properties.

Every car probably starts out with a number of seats property right?

And we're gonna set it to 5 as the default.

Every car I build is going to start out with 5 seats.

Seems pretty standard.

And then every car is also going to have a drive method because otherwise, why have a car If it doesn't? And when you call the drive method, it simply just turns the wheels.

So I'm going to create a print statement that says the 'wheels turn'. And now when I create my car object from my Car class,so let's create a new car. Let's call it myNormalCar because a little bit later on, we're going to obviously make this a lot more souped up. So myNormalCar is just going to be created from the bog standard car class.

So when I print myNormalCar .numberOfSeats for example, then you can see that I get 5. And when I say myNormalCar. drive, then it will try and perform the drive functionality and the wheels start turning.

That's all pretty standard and we're familiar with how classes objects work in this way now.

But what if I wanted to create a different type of car?

What if I wanted to create an electric car?

Well an electric car is kind of similar to a car right?

It probably also has five seats and it probably can drive as well, but it might have some other functionality.

Let's go ahead and create a new class and this class is going to be our ElectricCar.

But I don't really want to have to type out all of this again.

It's just the same as a normal car.

But I want to focus my time on explaining how it's different from a normal car.

So how can I inherit all of the things I get in car?

Well in Dart, the keyword is extends.

So my electric car is going to extend my Car class.

And what this means is that my ElectricCarr by default, is going to have everything that a car has.

So I could go ahead and create a new electric car.

Let's call it myTesla and let's create it as a new electric car. And now I can straight up just say

myTesla.drive and you'll see that it'll make the wheels turn.

Let's go ahead and comment out the previous ones that we had related to our car so we're not getting confused in the console, and you can see this myTesla drive triggers that electric car to turn its wheels because it inherited all of that capability from its parent class, the Car class, and it also has a starting default number of seats.

But by inheriting from the Car class, it means I don't have to repeat myself and type out everything that is common to cause an electric cars and I can just focus my time on creating code that is different.

For example my electric car might have a battery level and it's going to start out at 100%. But when it goes down, I can recharge my electric car so I can have a method called recharge.

And this will reset the battery level to 100% so now not only does my Tesla drive, it also can recharge and I didn't have to repeat the things that a car already has.

I've managed to incorporate the properties and methods simply by using that extend keyword.

This is how inheritance works and we can see this in our code as well.

Every single time that we've created a new stateful widget or stateless widget, we haven't have to create all the properties or the functionality of a stateless or stateful widget.

We've simply been able to tap into it by creating a class that extends the parent class a stateful or stateless widget.

And if you hold down COMMAND or CONTROL on Windows and you click on that class, you can see that it has various properties and methods that we've been able to simply use because we're extending it. We're inheriting from it and all we have to do is to describe all the things are different from the parent class and focus on building out our custom functionality rather than on the basic boring things such as how do we draw a widget on screen?

How do we give it a different color?

How do we change its shape and size?

How do we actually move the pixels around?

That's all sorted out for you by the Flutter SDK and all of that capability are packaged into these parent widgets that we're able to inherit from. Now in the next lesson,I want to talk a little bit about the last pillar which is polymorphism.

So for all of that and more, I'll see you on the next lesson.