Now in the last lesson in order to implement our animations we had to add something called a Single TicketProviderStateMixin to our welcome screen state class and we did that using the 'with' keyword.

And so in this lesson we're going to do a deep dive on exactly what these mixins are and why we need them. Now the first thing to keep in mind is this sentence that you find on the Dart language tool and it tells you that mixins are a way of reusing a class's code in multiple class hierarchies.

So while we're going through this lesson, keep this sentence. In mind in order to explain how mixin work I'm going to dive straight into the code.

So let's build a new class and let's call it Animal.

Now animals have this one behavior that they share and it's the fact that they can move.

So we're going to build a method for our animal class to describe how they behave, namely how they move right?

So if you're an animal you might move by, let's say for simplicity's sake, you'll print changed position right?

So every time you move, you're probably going to have changed your position.

So now if we create a new animal, we say animal.move, then what we'll get out put it into the console is that animal changed position right.

That's what you're telling it to do when you tell it to move.

Now let's say that we create another class and this is a fish class and this class is going to inherit from our animal class.

And the key word in Dart is extends, so the class of fish will extend from the animal class and this means that it now gets access to all of the behavior that's in the animal class.

So that means we could create a new fish and I could say fish move straight away without having to rewrite the code that defined how move worked right?

So now you can see that when I say fish.move in my main function it will also change position.

So this is how we are reusing a class's is code because we are able to inherit from that class.

Now let's say that we had another class right?

Let's say this is the class of birds and birds are also able to move and they're also going to inherit from animal.

But if we think about it the way that a fish moves versus the way that a bird moves is quite different right?

So we might want to override the move method that we inherited from animal.

So we can do that by using the @override annotation and then we can specify our move method which is what we're going to override.

And now we can start off by saying, let's do what our superclass does.

So in the superclass definition of move, we print change position. But after we've done that, after we've called that then we're going to do our own thing too.

We're going to say maybe print change position by swimming.

And for our bird class we're going to do exactly the same thing.

We're going to override the parents version of move which is to simply just change position, but then we're also going to print by flying.

So now our fish can move and it can also swim, our birds can move and it can also fly.

And when I decide to print my fish.move, you can see it's now gonna say change position by swimming.

And when I print my bird.move then in this case it's going to say change position by flying.

Now we have seen how we can inherit.

Well let's create another class.

Well this class is going to be a class of ducks right?

And ducks at least wild ducks, can both fly as well as swim.

So what do we inherit exactly? Do inherit from fish or do we inherit from bird? Because we kind of need ducks to be able to swim and also fly. So in Dart and also a lot of modern object oriented programming languages, there is a single inheritance pattern.

So you can only extend one class.

You can have your pick.

You can choose an animal or you can choose fish or you can choose bird but you can't say extends fish and extend bird and extend animal and for every other class whose code you want to reuse, doesn't work like this.

We can only extend one class.

So we have to make a choice.

Now even if we were able to extend fish and bird, well if we were to implement the move method for our duck, well is it going to use the birds move method where it flies or is it gonna use the move that inherited from fish where it swims?

If we were to inherit that move method from two classes which method do we end up actually getting because they both have a move method right?

So this is what mixins actually solve.

Instead of having a class of fish which has the move method and a class of bird which has a move method, we can actually instead create a mixin and we can call it CanSwim. So CanSwim basically has a method called swim.

And when you use this, it will print changing position by swimming. And we can create another mixin called

CanFly and so now we can have a fly method in the CanFly and we can change position by flying.

So now our duck, it could inherit by extending from our animal class maybe we want to be able to use a basic move where the duck just waddles and changes position, but we can also give it the CanFly and CanSwim ability by adding in the mixin. And we incorporate a mix into a class by adding the keyword 'with' after any class extensions and then we specify the names of our mixins.

So we have our CanSwim mixin followed by a comma and then we can add our CanFly mixin.

And so now our duck class not only has a move method so it's reusing this code that we wrote here, but it's also able to swim and fly.

So we're reusing code from lots of different classes.

So now if we were to create a duck object not only can it move, but it can also fly and it can also swim.

Now the benefit of this is that we can now reuse a class's code in multiple class hierarchies so it doesn't really matter who you inherit from. You don't even have to inherit from anybody.

You can reuse bits of code that you've created for other classes in lots of different classes.

So for example I might create another class later on called Airplane right? And airplane can simply just add that mixin of CanFly.

And already my airplane is able to fly without me having to write extra code and repeating that code that I wrote inside here.

Now these are all very trivial examples because I'm just putting in one line print statements, but you can imagine if you were programming a robot to be able to swim, to be able to say how the propellers would move or how you would displace water or how you would even worse try to figure out how to make something flap its wings and fly, that involves a lot of code and you want to try and keep it modular right?

By keeping the implementation of methods and properties and things that belonged to a particular class together in one class and then being able to simply just add it in into any classes that need it as a mixin, and this is the key you keep the implementation when using a mixin. You keep all the functionality that you've coded up in the fly method and the swim methods.

This is what the documentation means with adding features to a class.

So mixins are really powerful and often they provide you with an alternative to using a subclass. For example instead of getting our fish to override the parent move method, we could change the fish class to a class that now can swim by using that with keyword together with the mixin.

Well why is this useful?

Well big benefits tend to be with large programs with lots of classes that need to share some capabilities.

Having that shared capability split out into a mixin makes it easier to maintain too.

For example if you need to make a change to the swim method, then both the duck and the fish will know about that change without you having to change the fish or the duck class.

So if we're creating a large module with lots of classes that share capabilities and properties, then coming back to this definition mixins, they're a way of getting you to reuse a class as code in multiple hierarchies.

So it's kind of agnostic to what you inherit or where you are in the class hierarchy, it doesn't matter who your parents are or who your siblings are or who your children are.

You can always add these capabilities.

It's a bit like learning a new skill right?

Your parents don't have to be doctors for you to be able to learn how to become a doctor right?

It's a skill that you can have instead of something that you have to inherit.

