Abstraction - What is it?

Now consider these sentences.

- We adopted a new pet this weekend.
- I ordered something I really wanted from the store.
- I bought a ticket and won a prize.

If I said any of these things to a friend or coworker, it might be frustrating for them.

I haven't given them enough information in each of these cases.

They can't visualise what I am talking about, because they lack details. Which I didn't provide.

New pet, something ordered, and a ticket, are too general when talking about one item.

Abstraction - What is it?

On the other hand, when we talk about **groups of things**, we don't usually need too many specifics.

Consider these sentences.

I need to get home to feed the animals.

I'm waiting for my box of stuff from an online store to be delivered.

So here, animals, and stuff are probably enough information, to fully describe the situation.

The abstract class

The abstract class is declared with the **abstract** modifier.

Here I declare an abstract class called Animal.

```
abstract class Animal {}     // An abstract class is declared with the abstract
                             // modifier.
```

An **abstract** class is a class that's **incomplete**.

You can't create an instance of an abstract class..

```
Animal a = new Animal();  // INVALID, an abstract class never gets instantiated
```

An abstract class can still have a constructor, which will be called by its subclasses during their construction.

The abstract class

An abstract class's purpose is to define the behavior its subclasses are required to have, so it always participates in **inheritance**.

For the examples on this slide, assume that Animal is an abstract class.

Classes extend abstract classes and can be concrete.

Here, Dog extends Animal. Animal is abstract, but Dog is concrete.

```
class Dog extends Animal {}  // Animal is abstract, Dog is not
```

The abstract class

A class that extends an abstract class can also be abstract itself, as I show with this next example.

Mammal is declared abstract, and it extends Animal, which is also abstract.

```
abstract class Mammal extends Animal {}  // Animal is abstract, Mammal is also
                                         // abstract
```

And finally an abstract class can extend a concrete class.

Here we have BestOfBreed, an abstract class, extending Dog, which is concrete.

```
abstract class BestOfBreed extends Dog {}  // Dog is not abstract, but
                                           // BestOfBreed is
```

What's an abstract method?

An abstract method is declared with the modifier **abstract**.

You can see on this slide, that we're declaring an abstract method called move with a void return type.

It simply ends with a semi-colon.

It doesn't have a body, not even curly braces.

```java
abstract class Animal {

    public abstract void move();
}
```

Abstract methods can only be declared on an abstract class or interface.

What good is an abstract method, if it doesn't have any code in it?

An abstract method tells the outside world that all Animals will move, in the example I show here.

```java
abstract class Animal {

    public abstract void move();
}
```

Any code that uses a subtype of Animal, knows it can call the move method, and the subtype will implement this method with this signature.

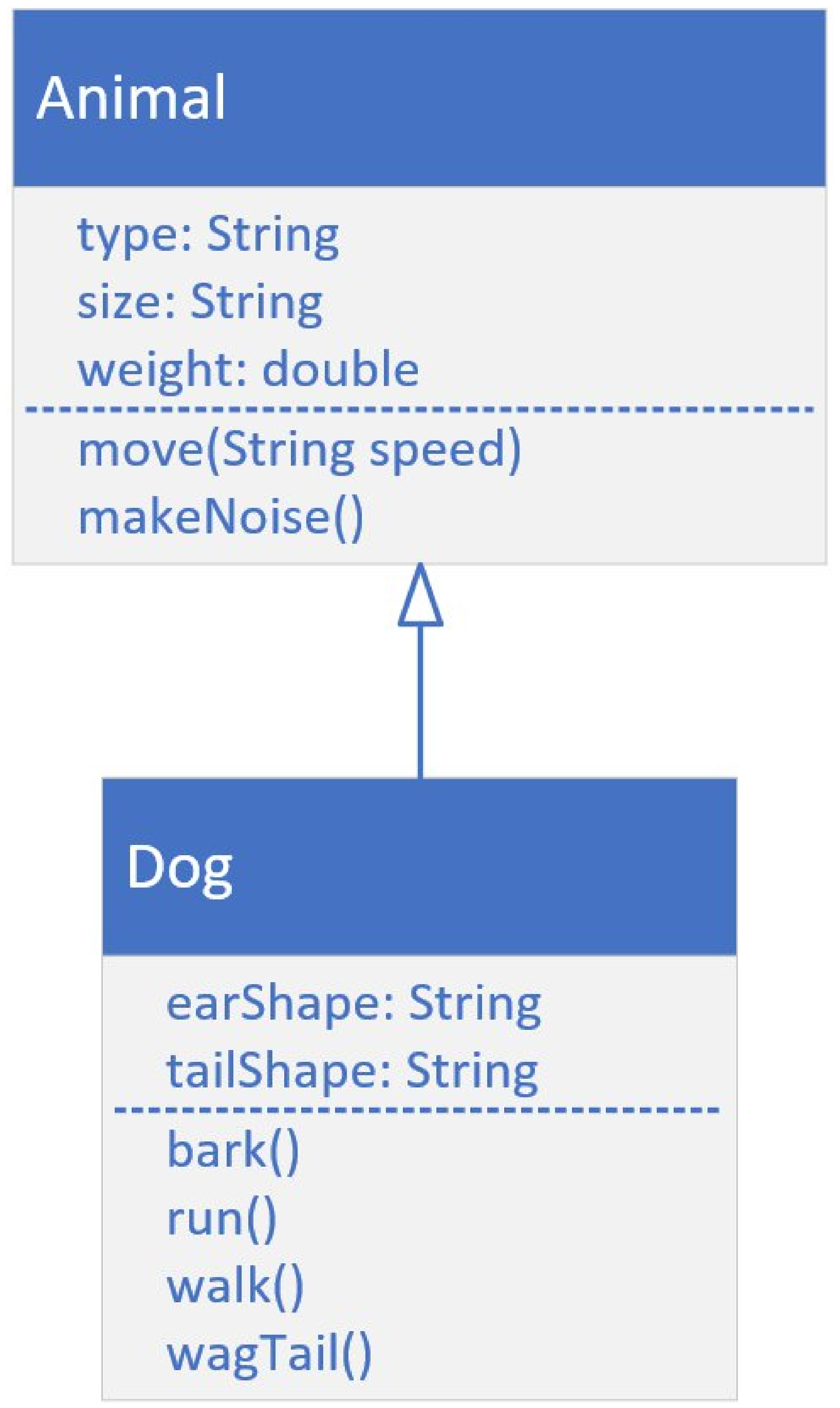
This is also true for a concrete class, and a concrete method that's overridden.

You might be asking, what's the difference, and when would you use an abstract class.

Animal and Dog Class Diagram from our Inheritance example

In the videos on **inheritance**, we created a very basic Animal class, and then we extended it to create a Dog.

This is the class diagram from that video again.



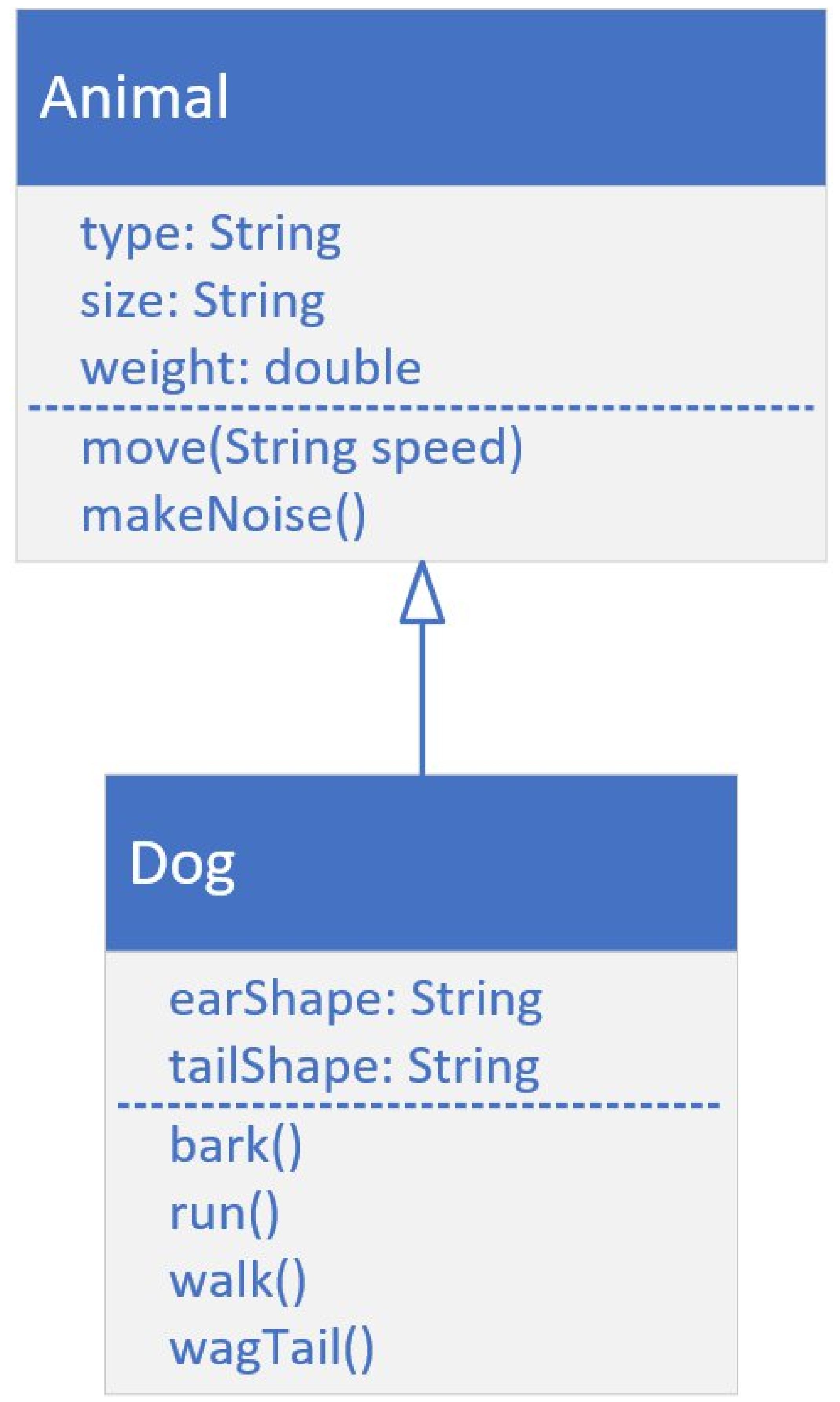
In that example, Animal was a concrete class, so the move and makeNoise methods had code in their method bodies.

Subclasses have choices when they extend a concrete class with concrete methods.

- They can inherit the same behavior from their parent. This means they don't have to even declare the methods in their class bodies.
- They can override the behavior from their parent. This means they have a method with the same signature, but with their own code in there, ignoring the parent's code altogether.
- They can also override the behavior but leverage the parent's method code, by calling the parent's method, using super in their overridden code.

Animal and Dog Class Diagram, What if Animal were abstract?

What happens if Animal is declared as abstract, and the move and makeNoise methods are also abstract?



If Animal is abstract and it's methods are abstract, subclasses no longer have the options I just talked about.

There is no concrete method for a subclass to inherit code from.

Instead, the subclass must provide a concrete method for any abstract method declared on its parent.

The subclass won't compile if it doesn't implement the abstract methods.