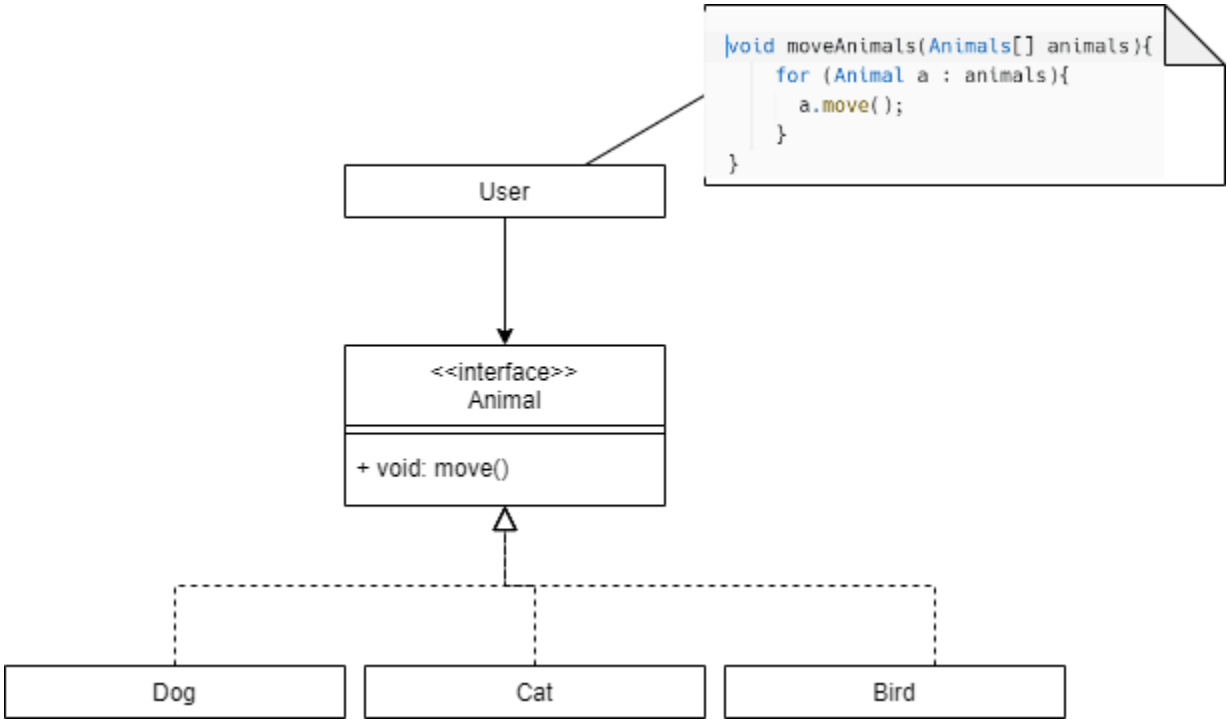
Abstractions in OOP

In OOP abstraction of the process of hiding implementation details from the user. Abstractions help us define what an object does without revealing how.

Interfaces

Often you may hear language such as **“the user would invoke the object via a known interface”** as this is exactly what interfaces do. We can treat objects as an instance of the interface it implements (and thus decoupling ourselves from the object.



Interfacing with objects

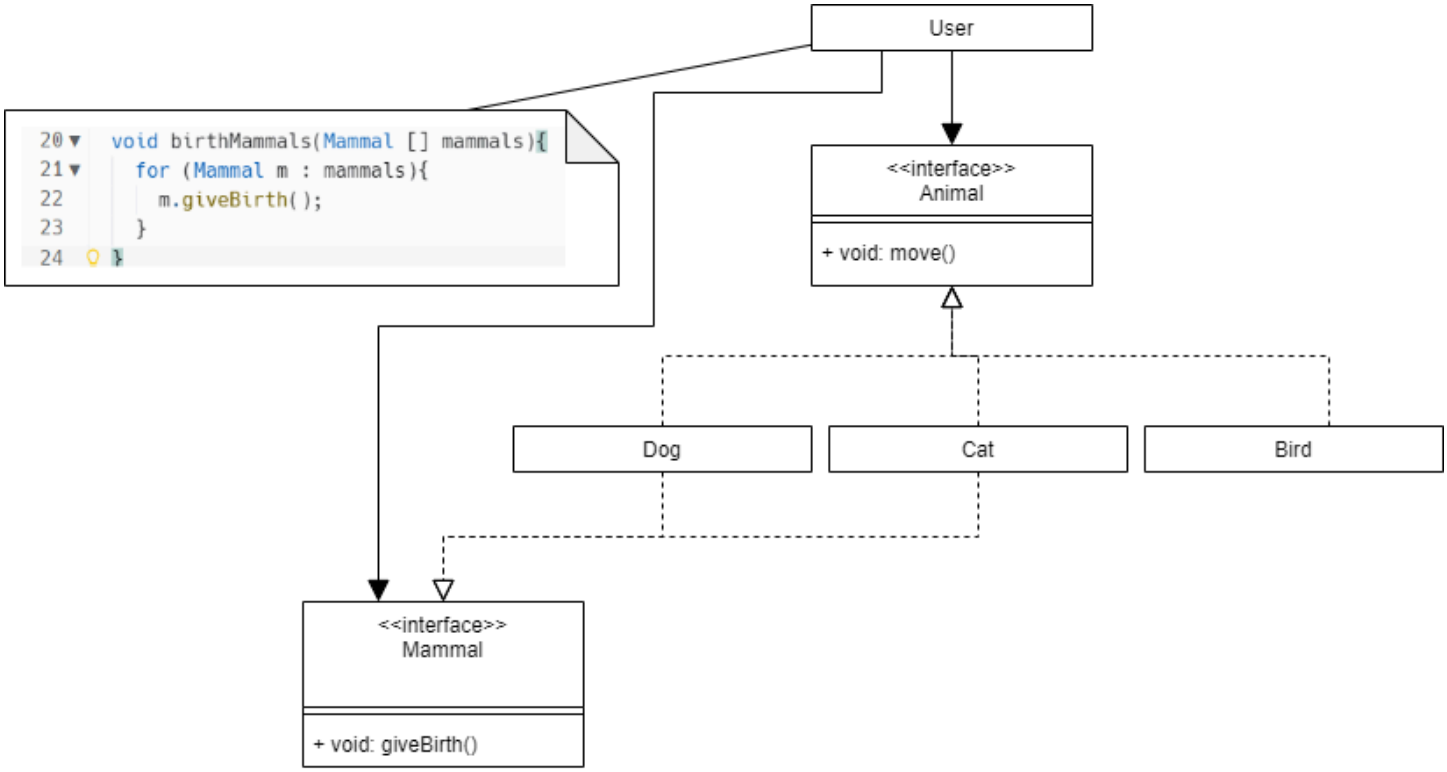
As you can’t create instances of interfaces, the elements of the **animals** array in the user code are likely dogs/cats/birds objects. Thus, it is said that the user code **interfaces** with these objects via the **Animal** type.

In other words; the user code only recognizes these objects as Animals and only uses the methods specified by the **Animal** interface that these objects implement.

Using abstractions like this lets us create really extendable code as existing modules would work with any newly created **Animal subclass** via the **Animal** interface.

Implementing multiple interfaces

Objects can implement multiple interfaces, this lets us treat the same object as instances of different types based on the context. We can group different objects together according to the interfaces they share.



Hence the user can use the Mammal interface to exclude bird objects. This approach keeps the hierarchy flatter than chaining classes, eg Animal -> Mammals -> Dog.

Abstract Classes

Java doesn't support dual inheritance so a class can only inherit from one abstract class. Abstract classes are best used when they provide inherited state or at least 1 concrete common functionally that its children can use. Otherwise using an interface might be more beneficial.

Abstract classes can also provide abstract methods to enforce that their concrete children implement them similarly to how interfaces specify methods to be implemented.

Although Abstract classes cannot be instantiated they can still have a constructor. When a constructor is defined in an abstract class you are forcing subclasses to override it using **super().** This is usually done to initialize the state of the properties defined in abstract class.