Webinar 10B – Interfaces

# Learning Objectives

* Implement interfaces as a way of specifying class-specific behaviour

# Preparation

We’ll use **Processing** for this webinar. Download **Animals.zip** and we’ll walk through the code and identify the problems associated with the code.

# Part 1 – Identifying Inheritance-related Issues

We’ll start by looking at the inheritance hierarchy defined in Figure 1 and highlight some of the problems with it. This inheritance hierarchy could be expanded further, e.g. **Pig** could have a superclass of **Mammal**, which could then extend **Animal.**

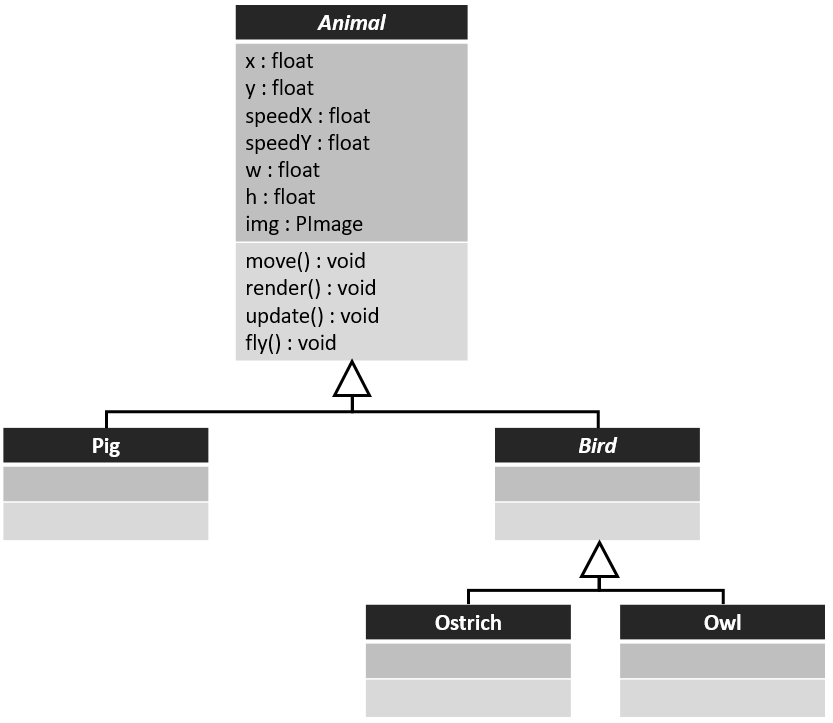


Figure Animal Inheritance Hierarchy

# Part 2 – Identifying a good place for fly()

We could move the fly method out of the **Animal** class and put it into the **Bird** class – but does every bird fly?

We could also just simply put the **fly** method into the **Bird** class (as depicted in Figure 2) and just remember not to call it if the object is an Ostrich, this is not a very good approach – having a method inside a class (even if the method is inherited) implies objects of that class are capable of performing that behaviour (and Ostriches cannot fly).

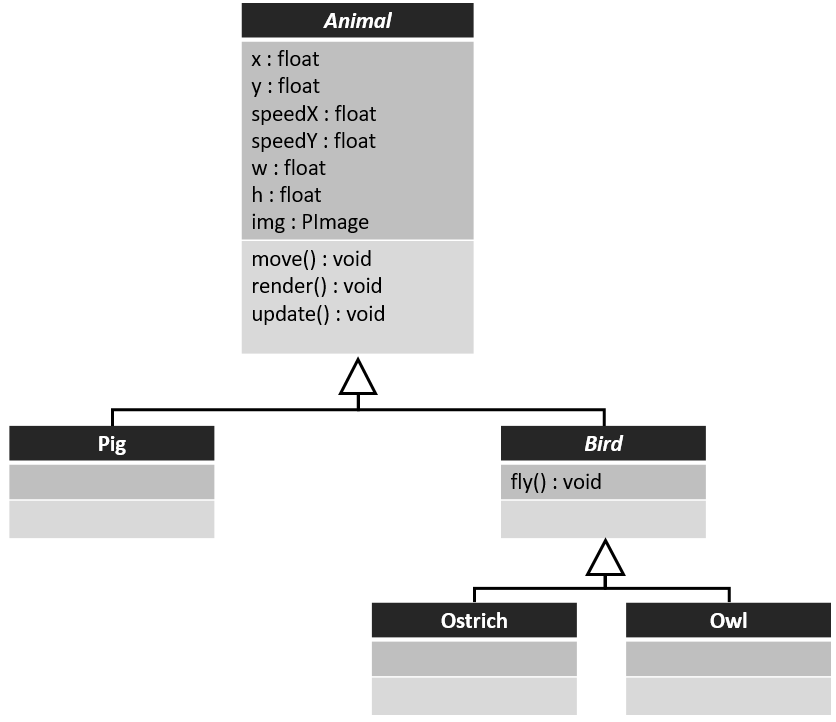


Figure fly method moved to Bird class

# Part 3 – Introducing Interfaces

To resolve this nuance, we can move the fly method to an interface, which looks similar to a class, except it provides abstract methods (i.e. the method headers, but no implementations for those methods).

A class *implements* an interface using the *implements* keyword. An interface is like a contract, any class which implements that interface promises to provide implementations for all of the methods provided in the interface. An interface also represents an “is-a” relationship (in the same way inheritance does).

We will create an interface called **Flyable**, which will contain a single abstract method (**fly**) and the Owl class will implement that interface and provide an implementation for that method. Interfaces are represented in UML as depicted in Figure 3.

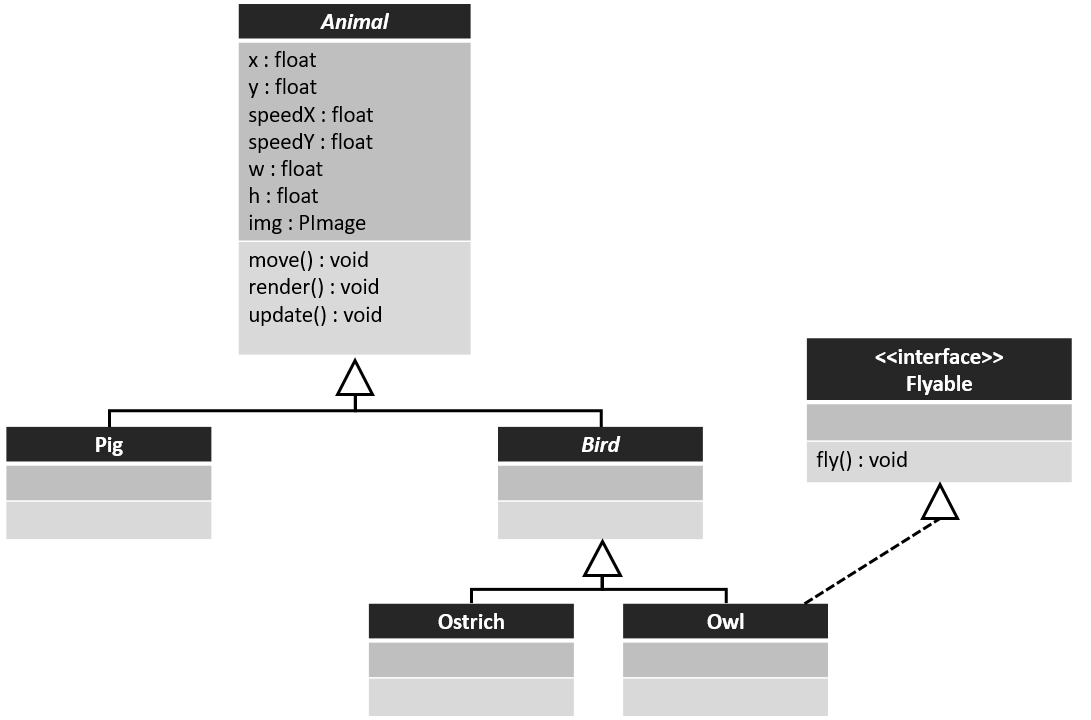


Figure UML Inheritance hierarchy (with an interface)

# Part 4 – Making use of the Interface

When we’re iterating through our ArrayList of animals, we can check to see if the current animal is a *type* of Flyable object (since interfaces are types themselves, like classes).

We’ll use the **instanceof** operator to check if the current animal if an instance of the flyable interface, and if it is, call the fly method.