Polymorphism

Mason Vail
Boise State University Computer Science

Pillars of Object-Oriented Programming

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction (sometimes)

Background: Object References and Method Calls

An object reference tells us where to find an object. It isn’t the object, itself.

At runtime, when a method is called via the dot operator, we follow the reference to find the object and *then* look up the object’s method with that signature. This “just-in-time,” dynamic look-up is called **late binding**.

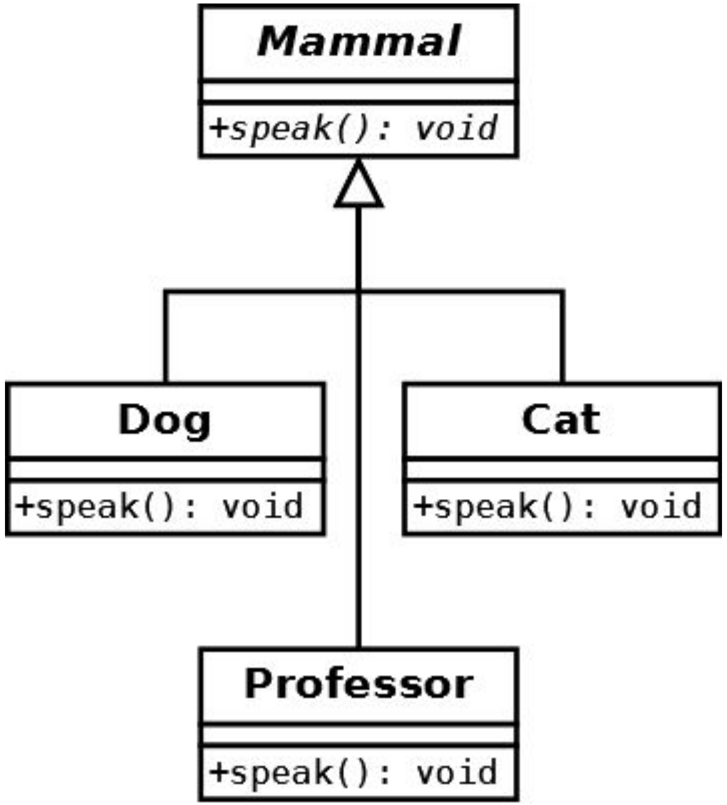
Background: Constructors and Assignment

When a class constructor is called, an object of that type is instantiated in memory with the properties and methods defined by the class, and it is initialized according to the code in the constructor.

A reference to that object is returned and may be assigned to a compatible object reference.

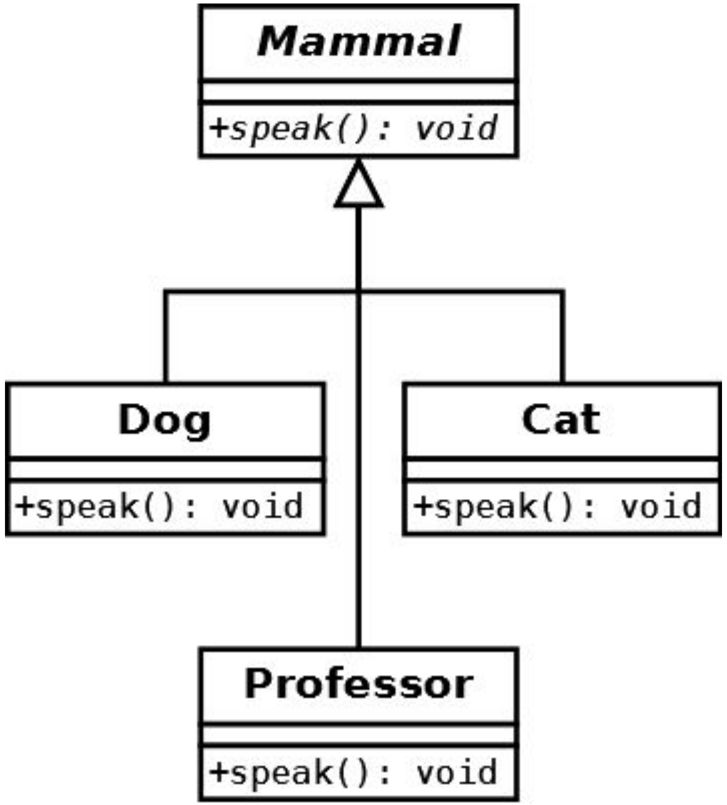
Here’s the big surprise: The object reference and the object *may not be exactly the same type*. All that is necessary is that the object “is-a” instance of the reference type. In other words, the reference type must be an ancestor of the object type.

Polymorphic Assignment



```
Mammal m = new Dog();
```

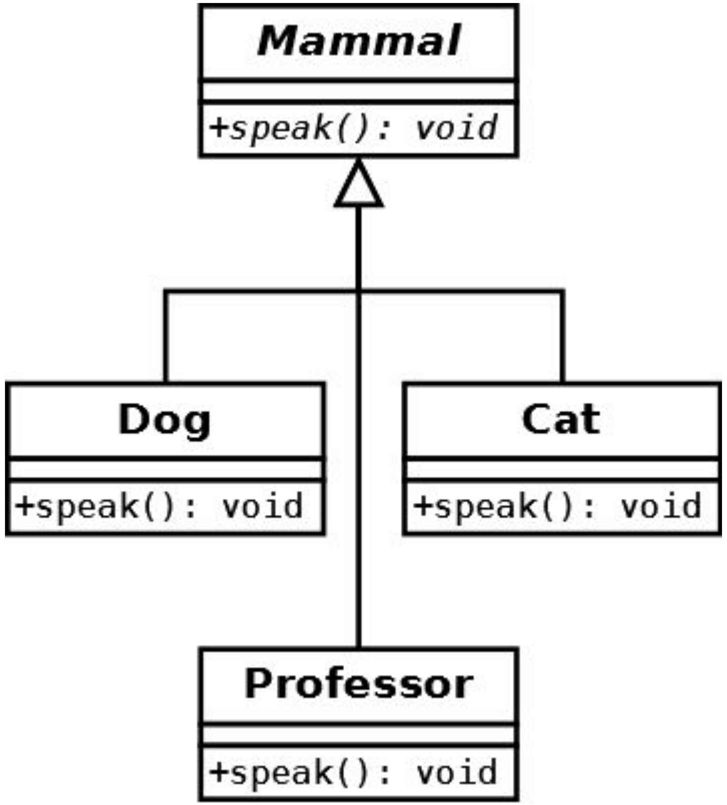
Polymorphic Assignment



```
Mammal m = new Dog();
m.speak();
```

Woof.

Polymorphic Assignment



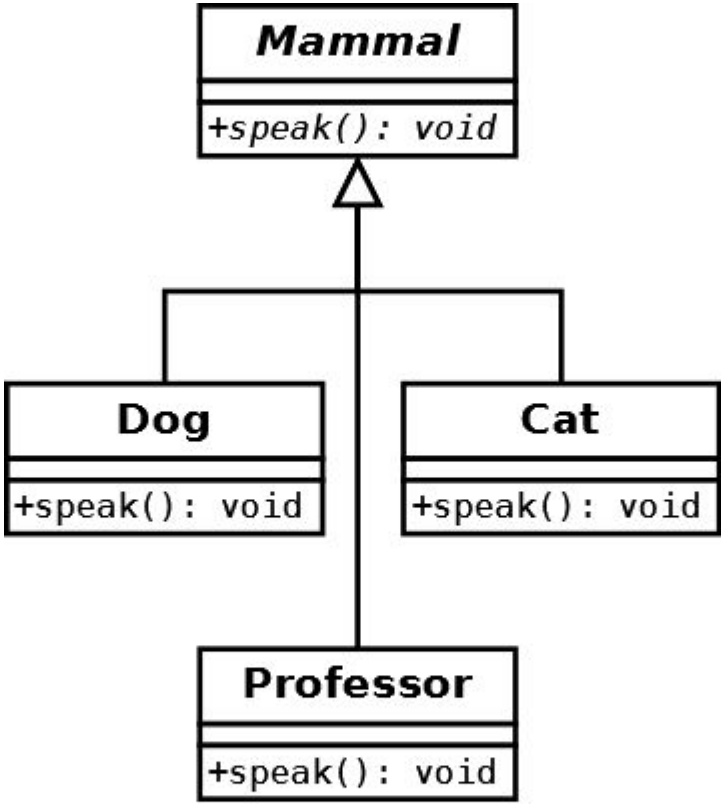
```
Mammal m = new Dog();
m.speak();
```

Woof.

```
m = new Cat();
m.speak();
```

Meow.

Polymorphic Assignment



```
Mammal m = new Dog();
m.speak();
```

Woof.

```
m = new Cat();
m.speak();
```

Meow.

```
m = new Professor();
m.speak();
```

Do your homework!

References Restrict Functionality

The reference type - not the object - limits what methods are available.

Dog may have added methods that it did not inherit from Mammal, such as fetch(), but those methods will not be available if the Dog is assigned to a Mammal reference.

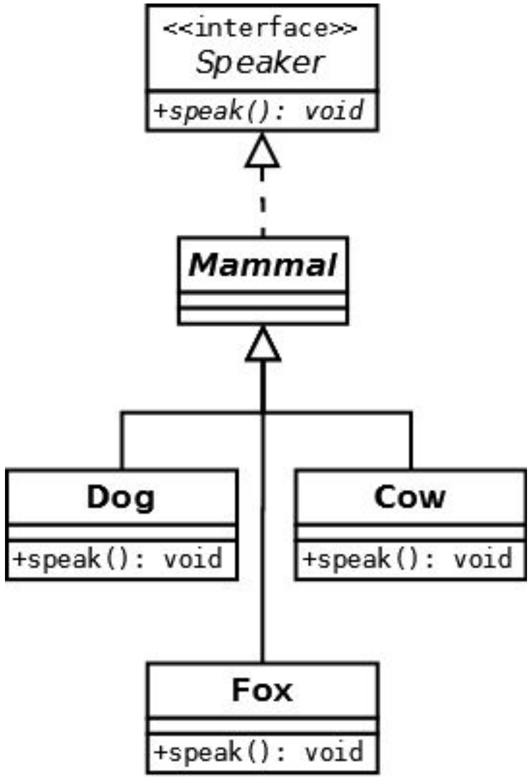
Polymorphic Collections

An array or other collection can store polymorphic references to many related objects. When looping through such a collection and calling a method of the reference type, each object will carry out its own version of that method.

```
Mammal[] mammalChoir = new Mammal[3];
mammalChoir[0] = new Dog();
mammalChoir[1] = new Cat();
mammalChoir[2] = new Professor();
for(Mammal m : mammalChoir) {
    m.speak();
}
```

```
    Woof.
    Meow.
    Do your homework!
```

References Can Be Any Type



Although an object can only be constructed from a concrete, instantiable class, a reference can be a class, abstract class, interface, or enumerated type.

```
Speaker[] speakers = new Speaker[3];
speakers[0] = new Dog();
speakers[1] = new Cow();
speakers[2] = new Fox();
for (Speaker s : speakers) {
    s.speak();
}
```

```
Woof.
Moo.
?
```

Examples of Polymorphism You’ve Probably Used

The for-each loops in previous examples depend on polymorphism. Any collection used in a for-each loop must have implemented the Iterable interface. The loop doesn’t care what kind of collection it is, as long as it is Iterable.

When you pass an object reference into System.out.println(), all that matters is the object is a descendent of Object. All println() needs to know is that the object will have a toString() method.

Graphical user interfaces are highly polymorphic. Containers, like JPanel, organize JComponents. They don’t care if those components are buttons, labels, or other panels - they just have to be JComponents. Listeners only need to be objects that implement the appropriate Listener interface.

Polymorphism Summarized

Polymorphism can be summarized in only two statements:

- An object reference can be assigned any compatible object.
- The reference type limits what methods you can call, but the object at the other end determines what the method will do.

Polymorphism

Mason Vail
Boise State University Computer Science