Object Oriented Programming

Objects

An object is a way to group related properties and functionality (behavior) into one package. It is meant to be a very intuitive way to organize your code. Some of the typical examples for illustrating objects and how they work show these concepts well - shapes and animals and/or people.

Properties

An object's properties are its data, the things that describe the object. These translate to variables in code. While each object defines the same set of properties, the values of those properties are unique for each actual object. For example, we're all people with names, birthdays, eye color, height, etc. However each actual person has different values for those properties.

Behavior

The things that the object can do are behaviors. These translate to functions (or methods) in code. These are defined once for all of the objects, but the results can be different based on the values of the object's properties. For example, let's say our person has a walk behavior. While everyone pretty much walks the same way, the speed may vary based on a person's age or height. If a person is wounded, they may limp, or if they think highly of themselves maybe they swagger.

Design

You'll often see objects put in boxes. This is a common way to illustrate objects when designing them. The name of the object (or class) is at the top, then the properties are shown, and at the bottom any behaviors.

Person
name birthday eye color height
walk() run() jump()

3 Principles of Object Oriented Programming

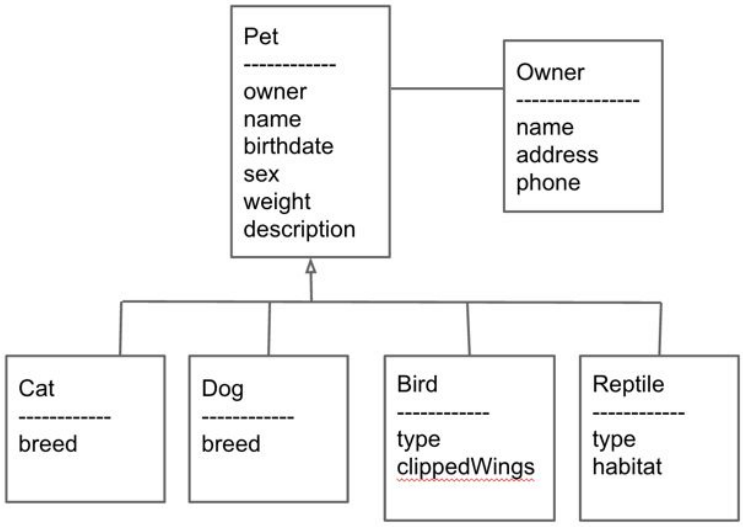
Encapsulation

Encapsulation is basically information hiding. Objects wrap their properties and only expose what they want other objects or code to see. Different programming languages do this a little bit differently - some are strict and some are by convention. Python works by convention, trusting people not to mess with things they shouldn't; Java / C# / C++ are have access control built into the language with keywords like public and private.

Inheritance

Inheritance allows us to build an object (or class) hierarchy. This allows us to group objects in the hierarchy and treat them the same way, even though they may be a little different. Animals are a great example here. Imagine you're writing software for a vet. They may see dogs, cats, birds, and various reptiles, but while they may medically treat these animals differently, a lot of the information they store about their animal client and it's owner is going to be the same.

They might make an object hierarchy like follows.



The classes underneath Pet, form an inheritance hierarchy. They all inherit the properties from the Pet object (or class), and then add-on the additional properties for their own object (or class). Cat, Dog, Bird & Reptile are children (or sub-classes) of Pet.

The Owner class that sits off to the side is different. It's not in the hierarchy. It is a separate class that is used by the Pet class. Notice that the line on it doesn't have the little arrow. It's just a plain line. This is an example of composition.

These show 2 different relationship, which get very technical terms - IS-A and HAS-A. So, as we look at the diagram, a Cat IS-A Pet, but a Pet HAS-A Owner. An Owner is not a Pet.

The nice thing about object oriented design and programming is that is should make logical sense. If something is not something else, then there should not be an inheritance relationship.

If we take this Vet example and extend it out to a Zoo, you might imagine a richer object (or class) hierarchy. You might have Animal at the top, and then sub-classes for different groups of animals maybe by species - Feline, Canine, Ungulate (hoofed-animals), etc. And then specific animals, like Lions and Tigers would be sub-classes of Feline, which is a sub-class of Animal. These hierarchies can be as deep or shallow as the scenario that they are modeling.

Try It

Let's think for a minute about some other common objects... How about clothing. What might you put together for a clothing class hierarchy? What if we were making a program to design a house? What kind of objects might we have? How might they fit into an inheritance tree? Do all the objects fit into the class hierarchy? (Do they need to?)

Polymorphism

What is polymorphism? Let's see what Dictionary.com has to say. There's a computing entry at the bottom:

> In object-oriented programming, the term is used to describe a variable that may refer to objects whose class is not known at compile time and which respond at run time according to the actual class of the object to which they refer.

Try spouting that definition out in a job interview. ;-)

OK, so if you got nothing else out of that, keep in mind that we're dealing with many (poly) changing forms (morph), and since this is OO programming, we're talking about objects. More specifically the object references (our remote controls) and the objects they refer to.

If we again go back to that Pet hierarchy, image we add a makeNoise behavior to the Pet object. Each one of those sub-classes makes noise a different way. A cat meows, a dog barks, a bird squaks, a reptile... hisses?

If we define the behavior on the parent object, each child can change or override that behavior. That allows us to make a list of Animals, use a loop to call the makeNoise method on each, and get a different response for each of the different animals.