Software Design & Consulting

Home Apps Blog Contact

What is Object Oriented Programming? – A Basic Explanation

Posted on July 18, 2012 by Thomas De Leon

In this post, I'll explain the basics of Object Oriented Programming (OOP). There are plenty of books and online resources to learn the details of OOP, but I'll focus on more of the high-level object oriented programming concepts.

It's all about Objects

As the name would imply, object oriented programming is all about, big surprise… objects. But what does that really mean? Let's look at some real-world examples to see.

Describing an Object

Imagine an object that most people can relate to, a car. How can we describe a car? Well, it has attributes such as the *color*, *make*, *model*, *year*, *mileage*, and *vin*. Each of these attributes make each car what it is. In OOP, attributes, the features that describe an object, are called **properties**.

Actions of an Object

The next question is, what are the actions a car can do, or can be done to the car? Let's take some of the obvious ones:

- Turn the car on
- Turn the car off
- Accelerate
- Brake

In OOP, these actions, are called **methods**. They allow the objects to do things, and as you'll see next, also provide a way to manipulate the properties of the object.

Changing and Reading the Description

When you buy a car, it has all the properties we described above (color, make, model, etc.) already defined. The car is painted, was made by a certain manufacturer, and is of a certain mode type, and so on. Some of those are fixed, for example if the car was made by Toyota, it's going to be a Toyota for the rest of it's life. But what about something like the color? A few years after you buy the car, you may decide you're tired of black and want it repainted to green. In object oriented programming, we call this **setter methods**, which are just special methods for setting properties on an object. Typically, they are just the property name prepended with "set", so in our case, it would be *setColor*. The opposite can also be done, let's say we get stopped for a speeding ticket, and the officer wants to get the VIN number. You would read the VIN through a **getter method**, which typically have 'get' prepending the property, so the officer would be using would be *getVin*.

Creating Objects

Now that we have properties, methods, and ways to read or set/change the properties through setter and getter methods, you may be asking, how do we create the objects in the first place?

Object Oriented Programming Constructors

The answer introduces another term in OOP, called a **constructor**. A constructor is a special method which creates an object. After you've created an object, you have an instance of it. This process is called **instantiation**.

Example

To illustrate this, lets go back to the car example. We are at the factory, and we instantiate our car by using a constructor method to create it. This special method takes inputs for the model, color, and VIN number, and outputs a brand new car. If you recall though, we had more properties than those, what about the make, year, and mileage?

In our example, if we are at a Toyota factory, the cars produced will always be Toyota, and the mileage will always be zero. The year will change, but will always match the current year. While we could simply pass those into our constructor, since those values are (mostly) fixed, we can simply set them in the constructor itself. This way we can be sure that *every* new car created with this constructor will have the make set to Toyota, the mileage to zero, and the year will always be current. The idea of setting properties when an object is created is called **initializing** them.

Constructor Abilities

Note that a constructor can take all, some, or no properties as inputs, but will always create one as output. For one taking no inputs, there you would initialize the properties to some value (which is always good practice), expected to be changed at some later point. The other piece that all constructors do is allocate memory. This differs from language to language, but will always do the actual creating of the object. Also, there are methods which do the opposite, called **destructors**, which destroy the object. They are important because they deallocate any memory allocated in the constructor. Often, this is all they will do, but other code can be added to them which will be executed when the object is destroyed.

How can this save you time?

So far, you may see how this can help organize how you store data (in fact it can be comparable to how you'd store data in fields and tables in a relational database), but you may wonder how this can help you save time?

Up to now, you may have just been thinking of our car as a typical sedan. Ok, lets change the manufacturer now to Volvo. Volvo makes cars, trucks *and* buses, so what do we do now? No problem, you say! We can just create bus and truck objects just like we did for cars, and we're all set.

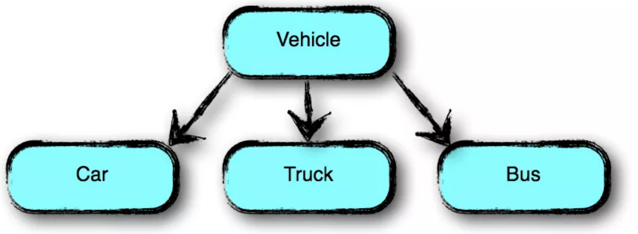
Yes, that works, but it's not a good solution. All three types share common properties, and even methods. But it means you'd have to write those three times, to do the exact same thing.

An Efficient Solution

The better solution, and what OOP is all about, is to create another object, which holds all the properties and methods that are common between cars, trucks, and buses. Here is where we'll change terms, and instead of calling these objects, we'll now call them **classes**. Classes are objects when they have been created, but this is a better term because it also refers to them when they haven't been created yet.

Back to our example. So now we have a class called vehicle, with all the properties and methods common to our three types of vehicles. This is called the **parent class**, and the three vehicles are it's **subclasses**. What's important to note is that those common properties and methods only need to be defined in the parent class level. The children automatically have access to these, and so defining them there is not necessary. This concept is called **inheritance**.

Subclasses



In the subclasses, we only need to define the things that are unique to them. For example, for a truck the maximum amount of cargo is important, so this could be a property, and a bus may have a method to *loadPassengers* or *unloadPassengers*.

Also imagine that we started to produce other types of vehicles, the inheritance concept really makes things easier to add them, as we simply need to define the properties which are not already covered by the common vehicle class.

In OOP, you will sometimes have classes that are not intended to be used directly (such as our vehicle class in this example), but instead simply exist as a common parent. In these cases, it is meant for the subclasses to be used. In other cases, the parent and subclasses can be used. Or, subclassing can be a way for you to extend an existing class that you would like to add functionality to.

Summary

Hopefully this has helped with a basic understanding of Object Oriented Programming concepts, for those unfamiliar. It is a great concept to learn, and can make your programming much more robust and faster to develop. Also, these concepts are pretty general and apply to many different object oriented languages.

To summarize, here are some of the main object oriented programming concepts:

- Objects (classes) consist of properties and methods
- Constructors are used to create objects, destructors are used to destroy them.
- Subclassing and inheritance can help you save time by re-using and better organizing your code

Definitions

class – another term for an object; contains properties and methods

constructor – a special method which creates an object and initializes its properties

destructor – a special method which destroys an object

getter method – a special method which gets the value of a property

initializing – setting properties to their initial values when a class is instantiated

instantiation – the process of creating an instance of a class

methods – functions which are actions that belong to an object

parent class – the "main" class where other classes inherit properties and methods from

properties – attributes (variables) which describe an object

setter method – a special method which sets the value of a property

subclass – classes which inherit properties and methods from a parent class; it can also define it's own properties and methods

Online Resources

- Introduction to Object Oriented Programming and More
- Wikipedia – Object Oriented Programming
- Object Oriented Programming with Objective-C
- Introduction to Object Oriented Programming Using C++
- Essentials of the Java Programming Language: A Hands-On Guide, Part 2
- Object Oriented Programming – Swinburne University of Technology (Free iTunes U course)

Books

- Object-Oriented Programming
- Object Oriented Programming (Free Online Book)
- Object Oriented Programming in C++

- Objective C: Object Oriented Programming Techniques
- Introduction to Object-Oriented Programming With Java – 5th edition

Tweet G+1 7 Share 9

12 Comments

Ankit
July 19, 2012 at 5:08 am Reply

Is Operator Overloading is Possible in Objective C?

Thomas De Leon
July 19, 2012 at 11:01 am Reply

Hi Ankit,

Good question. No, Objective C does not include operator overloading as a feature. However, it is possible to mix Objective C and C++ (which will of course allow operator overloading) code by using Objective C++. Here are some links that might help you:
http://stackoverflow.com/questions/3613980/can-i-overload-an-operator-in-objective-c
http://en.wikipedia.org/wiki/Objective-C#Objective-C.2B.2B

Also, we do plan on doing some later posts on more advanced OOP features, including things like operator overloading, as well as some tutorials in actual code, so stay tuned.

Hope this helps!

Pingback: Tech Trends | Pearltrees

Zarkon
September 23, 2012 at 9:39 pm Reply

Thanks for this – just what I needed!

Chidi
December 22, 2012 at 3:40 pm Reply

This helps a lot to a newbie like me. Good job and God bless you.

Tinu
January 30, 2013 at 2:43 pm Reply

Excellently written. Thanks

James
January 30, 2013 at 11:34 pm Reply

Good stub…Thanks a lot…

Pingback: [Learn advanced programming using OOP and Design Patterns | Learn and teach good programming]()

Crismon
August 8, 2013 at 1:35 pm Reply

Thomas, you have the gift of a true professor. I was able to perceive almost every word in this post and I truly thank you.

Robert
September 26, 2013 at 7:25 pm Reply

This is one of the best descriptions of oop that I have ever seen. Thank you very much for sharing this with all of us.

Ridza
July 25, 2015 at 3:39 am Reply

Best description OOP ever. Thanks a million Thomas!

pascal
December 12, 2015 at 8:31 am Reply

The most comprehensive and understandable OOP I have read.

Just a quick question; Assuming I intend to still categorize the subclasses into different manufacturers (toyota,honda,volvo…e.tc). how do I go about this? I thought about creating an instance of the subclasses with the manufacturers name. Pls clarify.

Thanks

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

☐ Notify me of follow-up comments by email.

☐ Notify me of new posts by email.

Share