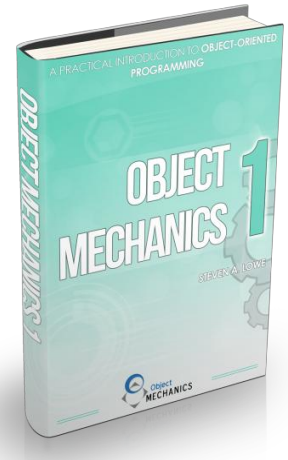
Object-Oriented Programming Explained in 90 Seconds



"LIFE IS REALLY SIMPLE, BUT WE INSIST ON MAKING IT COMPLICATED." --CONFUCIUS

Object-oriented programming evolved from the combination of a few simple yet powerful concepts. This overview explains them succinctly.

Reading this document out loud takes me about 90 seconds, hence the title.

I hope you find this information useful! If so, please share this document as you see fit, and visit http://object-mechanics.com for more information.

Best regards,
Steven A. Lowe
Author, *Object Mechanics 1*

Procedures

Lines of code are collected into *procedures* and used to perform actions such as:

1. feed some data in
2. apply some processing or transformation
3. get some data out.

Data Structures

Sometimes, some of the data elements are related to some of the other data elements, and it is convenient to group them together into a *data structure*, which can then be manipulated and addressed as a single unit.

Procedures + Data Structures

Now our procedure can take a data structure as its input. The procedure may alter the data in the structure and/or produce another data structure as it output.

Object, Methods, and Properties

Occasionally, we notice that some procedures are only concerned with a certain kind of data structure. It is convenient to group these procedures together with their data structure, and call it an *object*. This allows us to refer to a data structure and all of its associated procedures as a single unit.

The procedures associated with an object are called *methods*.

The elements of the data structure for an object are called *properties*.

Object-Oriented Programming Explained in 90 Seconds

Class and Instance

We may find that we need a lot of objects that behave the same but have different values for their data elements. A template for creating objects is called a *class*. An object is said to be an *instance* of its class.

Message Passing and Polymorphism

If we politely request an object to do something, instead of rudely executing its procedures directly, this is called *message passing* - even if no actual "message" is transmitted.

The joy here is that many different kinds of objects may respond to the same message in different ways; this is called *polymorphism*.

For example, we can ask many different kinds of machines to Start, and each responds appropriately. We see this every day: many different machines around us have some kind of a Start button, but exactly how they go about starting themselves is entirely up to the particular machine to accomplish.

Inheritance, Superclass, and Subclass

We may notice that one class is very much like another, sharing a lot of the same data elements and procedures. So, instead of copying and pasting code we let one class *inherit* from another: the *subclass* inherits from the *superclass* or "base class".

In this way the subclass has access to all of the data structures and procedures of the superclass, and can augment or override them as necessary to define the differences between the base class and the subclass.

This process is often called *differential programming*; it is a very powerful and convenient way to reuse code.

That's it! These are the basic concepts of object-oriented programming (OOP) - simple, subtle, and very powerful!

Of course, they can be misused and misapplied. This is where experience and practical advice becomes extraordinarily valuable. It's easy to learn the definitions. It's more difficult to know how to apply them. It's far more difficult to recognize and fix it when you have misapplied them!

Thanks for reading, and remember to share this document and visit http://object-mechanics.com!

