8.1 — Welcome to object-oriented programming

BY ALEX ON AUGUST 23RD, 2007 | LAST MODIFIED BY ALEX ON MAY 19TH, 2018

Back in lesson **1.3 -- A first look at variables, initialization, and assignment**, we defined an object in C++ as, “a piece of memory that can be used to store values”. An object with a name is called a variable.

In traditional programming (what we’ve been doing prior to this point), programs are basically lists of instructions to the computer that define data (via objects) and then work with that data (via statements and functions). Data and the functions that work on that data are separate entities that are combined together to produce the desired result. Because of this separation, traditional programming often does not provide a very intuitive representation of reality. It’s up to the programmer to manage and connect the properties (variables) to the behaviors (functions) in an appropriate manner. This leads to code that looks like this:

```
1 driveTo(you, work);
```

So what is object-oriented programming? As with many things, it is perhaps understood most easily through use of an analogy. Take a look around you -- everywhere you look are objects: books and buildings and food and even you. Objects have two major components to them: 1) A list of relevant properties (e.g. weight, color, size, solidity, shape, etc…), and 2) Some number of behaviors that they can exhibit (e.g. being opened, making something else hot, etc…). These properties and behaviors are inseparable.

Object-oriented programming (OOP) provides us with the ability to create objects that tie together both properties and behaviors into a self-contained, reusable package. This leads to code that looks more like this:

```
1 you.driveTo(work);
```

This not only reads more clearly, it also makes it clearer who the subject is (you) and what behavior is being invoked (driving somewhere). Rather than being focused on writing functions, we’re focused on defining objects that have a well-defined set of behaviors. This is why the paradigm is called “object-oriented”.

This allows programs to be written in a more modular fashion, which makes them easier to write and understand, and also provides a higher degree of code-reusability. These objects also provide a more intuitive way to work with our data by allowing us to define how we interact with the objects, and how they interact with other objects.

Note that OOP doesn’t replace traditional programming methods. Rather, it gives you additional tools in your programming tool belt to manage complexity when needed.

Object-oriented programming also brings several other useful concepts to the table: inheritance, encapsulation, abstraction, and polymorphism (language designers have a philosophy: never use a small word where a big one will do). We will be covering all of these concepts in the upcoming tutorials over the next few chapters. It’s a lot of new material, but once you’ve been properly familiarized with OOP and it clicks, you may never want to go back to pure traditional programming again.

Note that the term “object” is overloaded a bit, and this causes some amount of confusion. In traditional programming, an object is a piece of memory to store values. And that’s it. In object-oriented programming, an “object” implies that it is both an object in the traditional programming sense, and that it combines both properties and behaviors. From this point forward, when we use the term “object”, we’ll be referring to “objects” in the object-oriented sense.


8.2 -- Classes and class members


Index


7.x -- Chapter 7 comprehensive quiz