**Day 6: Understanding Object- Oriented Programming**

**Is C++ Object-Oriented?**

C was developed as a middle ground between high-level business application languages such as COBOL and the pedal-to-the-metal, high-performance, but difficult-to-use Assembler language. C was to enforce “structured” programming, in which problems were “decomposed” into smaller units of repeatable activities called *procedures* and data was assembled into packages called *structures.*

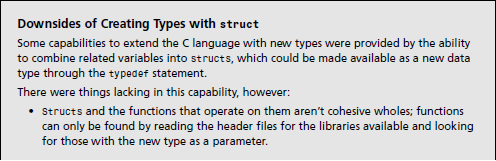
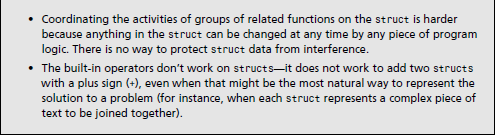
But research languages such as Smalltalk and CLU had begun to pave a new direction— object-orientation—which combined the data locked away in assemblies like structures with the capabilities of procedures into a single unit: the object.

The world is filled with objects: cars, dogs, trees, clouds, flowers. Objects. Each *object* has characteristics (fast, friendly, brown, puffy, and pretty). Most objects have behavior (move, bark, grow, rain, wilt). You don’t generally think about a car’s specifications and how those specifications might be manipulated. Rather, a car is thought about as an object that looks and acts a certain way. And the same should be true with any real-world object that is brought into the domain of the computer.

The programs being written early in the twenty-first century are much more complex than those written at the end of the twentieth century. Programs created in procedural languages tend to be difficult to manage, hard to maintain, and expensive to extend. Graphical user interfaces, the Internet, digital and wireless telephony, and a host of new technologies have dramatically increased the complexity of our projects at the same time that consumer expectations for the quality of the user interface are rising.

Object-oriented software development offers a tool to help with the challenges of software development. Though there are no silver bullets for complex software development, object-oriented programming languages build a strong link between the data structures and the methods that manipulate that data and have a closer fit to the way humans (programmers and clients) think, improving communication and improving the quality of delivered software. In object-oriented programming, you no longer think about data structures and manipulating functions; you think instead about objects as if they were their real-world counterparts: as things that look and act a certain way.

C++ was created as a bridge between object-oriented programming and C. The goal was to provide object-oriented design to a fast, commercial software development platform, with a special focus on high performance.



**Introducing Classes and Members**

**­­­Class** (***New*** ***type***) - A class is just a collection of variables— often of different types—combined with a set of related functions. A class can consist of any combination of the variable types and also other class types.

One way to think about is a car (class) is as a collection of (variables such as,) wheels, doors, seats, windows, and so forth. Another way is to think about what a car can do: It can move, speed up, slow down, stop, and park, and so on.

**Object** (***Variable* *of* *the* *New* *type***) - A class enables you to encapsulate, or **bundle**, these various parts and various functions into one collection.

**Encapsulation** – bundle in one collection (**class**).

Encapsulating everything you know about a car into one class has a number of advantages for a programmer. Everything is in one place, which makes it easy to refer to, copy, and call on functions that manipulate the data.

**Client** (***of*** ***a*** ***class***) - The parts of the program that use your class.

**Data** **hiding** - can use your object without worrying about what is in it or how it works.

**Data** **Member** (***Member* *Variables***) - The variables in the class. A Car class might have member variables representing the seats, radio type, tires, and so forth. It’s also like a Characteristic of an Object.

**Method** (***Member Function) -*** A class can also contain functions. They determine what your class can do. The member functions in the class typically manipulate the member variables. It’s also like a Verb! That is functioning or can do action.

**Declaring a Class**

**Class** **Declaration** - To declare a class, use the class keyword followed by the class name, an opening brace, and then a list of the data members and methods of that class. End the declaration with a closing brace and a semicolon.

Here’s the declaration of a class called Cat:

**class** ***Cat***

{

***unsigned*** ***int*** itsAge;

***unsigned*** ***int*** itsWeight;

***void*** Meow();

};

*“Declaring this class doesn’t allocate memory for a Cat. It just tells the compiler what a Cat is, what data members it contains (itsAge and itsWeight), and what it can do (Meow()). Although memory isn’t allocated, it does let the compiler know how big a Cat is—that is, how much room the compiler must set aside for each Cat that you will create.*

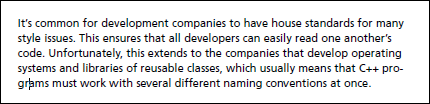
*In this example, if an integer is four bytes, a Cat is eight bytes big: itsAge is four bytes, and itsWeight is another four bytes. Meow() takes up only the room required for storing information on the location of Meow().”*

**A Word on Naming Conventions**

As a programmer, you must name all your member variables, member functions, and classes, these should be easily understood and meaningful names. Cat, Rectangle, and Employee are good class names. Meow(), ChaseMice(), and StopEngine() are good function names because they tell you what the functions do. Many programmers name the member variables with the prefix “its,” as in itsAge, itsWeight, and itsSpeed. This helps to distinguish member variables from nonmember variables.

Similarly, many programmers begin all functions with capital letters and all variables with lowercase. By capitalizing each word—for example, ChaseMice or DrawCircle.

The important idea is that you should pick one style and stay with it through each program. Over time, your style will evolve to include not only naming conventions, but also indentation, alignment of braces, and commenting style.



**Defining an Object**

After you declare a class, you can then use it as a new type to declare variables of that type. You declare an object of your new type the same as you declare an integer variable:

**unsigned** ***int*** GrossWeight; // define an unsigned integer (local variable)

**Cat** ***Frisky***; // define a Cat

This code defines a variable called GrossWeight, whose type is an unsigned integer. It also defines Frisky, which is an object whose class (or type) is Cat.