CHAPTER 14

Let C++ Simplify Your Life

> Our life is frittered away by detail . . . Simplify, simplify.
>
> —H. D. Thoreau, *Walden*

THE PROGRAMMING LANGUAGE C++, UNDER development since 1979 by Bjarne Stroustrup[1] at Bell Laboratories, is an extension of C that promises to dominate the field of software development. C++ supports the principles of object-oriented programming, which is based on the tenet that programs, or, better, processes, comprise a set of objects that interact exclusively through their interfaces. That is, they exchange information or accept certain external commands and process them as a task. In this the *methods* by which an object carries out a task are an internal affair "decided upon" autonomously by the object alone. The data structures and functions that represent the internal state of an object and effect transitions between states are the private affair of the object and should not be detectable from the outside. This principle, known as *information hiding*, assists software developers in concentrating on the tasks that an object has to fulfill within the framework of a program without having to worry about implementation details. (Another way of saying this is that the focus is on "what," not on "how.")

The structural designs for what goes on in the "internal affairs" of objects, containing complete information on the organization of data structures and functions, are the *classes*. With these the external interface of an object is established, and this is decisive for the suite of behaviors that an object can perform. Since all objects of a class reflect the same structural design, they also

[1] The following, from Bjarne Stroustrup's Internet home page (`http://www.research.att.com/~bs/`), may help to answer the question, How do you pronounce "Bjarne Stroustrup"?: "It can be difficult for non-Scandinavians. The best suggestion I have heard yet was 'start by saying it a few times in Norwegian, then stuff a potato down your throat and do it again' :-). Both of my names are pronounced with two syllables: Bjar-ne Strou-strup. Neither the B nor the J in my first name are stressed and the NE is rather weak so maybe Be-ar-neh or By-ar-ne would give an idea. The first U in my second name really should have been a V making the first syllable end far down the throat: Strov-strup. The second U is a bit like the OO in OOP, but still short; maybe Strov-stroop will give an idea."

possess the same interface. But once they have been created (computer scientists say that classes are *instantiated* by objects), they lead independent lives. Their internal states are changed independently of one another and they execute different tasks corresponding to their respective roles in the program.

Object-oriented programming propagates the use of classes as the building blocks of larger structures, which can again be classes or groups of classes, into complete programs, just as houses or automobiles are constructed of prefabricated modules. In the ideal case programs can be cobbled together from libraries of preexisting classes without the necessity for the creation of a significant amount of new code, at least not on the order of magnitude as is typical in conventional program development. As a result it is easier to orient program development to reflect the actual situation, to model directly the actual processes, and thereby to achieve successive refinement until the result is a collection of objects of particular classes and their interrelations, in which the underlying real-world model can still be recognized.

Such a way of proceeding is well known to us from many aspects of our lives, for we do not generally operate directly with raw materials if we wish to build something, but we use, rather, completed modules about whose construction or inner workings we have no detailed knowledge, nor the necessity of such knowledge. By standing on the shoulders of those who built before us, it becomes possible for us to create more and more complex structures with a manageable amount of effort. In the creation of software this natural state of affairs has not previously found its true expression, as software developers turn again and again to the raw materials themselves: Programs are constructed out of atomic elements of a programming language (this constructive process is commonly called *coding*). The use of run-time libraries such as the C standard library does not improve this situation to any great degree, since the functions contained in such libraries are too primitive to permit a direct connection to a more complex application.

Every programmer knows that data structures and functions that provide acceptable solutions for particular problems only seldom can be used for similar but different tasks without modification. The result is a reduction in the advantage of being able to rely on fully tested and trusted components, since any alteration contains the risk of new errors—as much in the design as in programming. (One is reminded of the notification in manuals that accompany various consumer products: “Any alteration by other than an authorized service provider voids the warranty.”)

In order that the reusability of software in the form of prefabricated building blocks not founder on the rocks of insufficient flexibility, the concept of *inheritance*, among a number of other concepts, has been developed. This makes it possible to modify classes to meet new requirements without actually altering them. Instead, the necessary changes are packaged in an extension layer.