on the subject revolve around the design patterns presented in that book. Without dismissing the others, it is the 32 patterns presented there that certainly deserve to be called classics.

A fascinating part of that book, however, is not the design patterns themselves but the introductory chapter, which provides the rationale behind many of them and a common thread for linking them together. There we find principles of reusable object-oriented design. The second such principle is *to favor object composition ("has") over class inheritance ("is-a")*. (The first principle is "to program to an interface, not an implementation," which should be clear to anybody who has seen any advice on encapsulation in the last 40 years.)

To programmers who have not followed the arrival of object-oriented programming to center stage in the 1980s and 1990s, this rule might not seem overly important. But if one recalls that time, one of the defining concepts in object-oriented programming was inheritance. Take, for instance, Bjarne Stroustrup's description of C++ in *The C++ Programming Language* (1985). We find there (p. 21) that:

> C++ is a general purpose programming language with a bias toward systems programming that:
>
> - Is a better C
> - Supports data abstraction
> - Supports object-oriented programming
> - Supports generic programming

If we then turn to see exactly what "supports object-oriented programming" entails (p. 39), we find:

> The programming paradigm is:
>
> - Decide which classes you want
> - Provide a full set of operations for each class
> - Make commonality explicit by using inheritance

Now consider by contrast yet another classic, Joshua Bloch's *Effective Java* (2008). We find there no less than three injunctions against inheritance:

- Favor Composition Over Inheritance
- Design and Document Inheritance or Else Prohibit It
- Prefer Interfaces to Abstract Classes

Is then inheritance to be avoided? That is not an academic point. Programming Microsoft Windows can be very frustrating, even when approached by way of a very pleasant book (Charles Petzold's *Programming Windows* [1999]). When the first frameworks for Windows programming came out (from Borland and Microsoft), they felt like a breath of fresh air. Before then, creating a single window was nowhere near simple: programmers were interested to learn that to program in Microsoft Windows they had to work with something called a window

Download at WoweBook.Com