languages; some of those that we do not see give us reason to pause and reflect on the reasons why.

We do not assume any prior knowledge of Smalltalk here, although we will cover a significant part of Smallktalk by the time we reach the end of this chapter. We will highlight design principles by using small code snippets to illustrate them. One of the strengths of strong design principles is that there are few things to learn, and once you grasp them, the whole infrastructure flows from them. The Smalltalk system we will be referring to is Squeak (*http://www.squeak.org*), an open source implementation. Some code examples may be difficult to understand at first reading, since we introduce concepts somewhat informally, but they are illustrated in subsequent examples, so it is prudent to make an effort to go through to the end and then return to any offending parts. At the same time, we do not insult the reader's intelligence.

Exploring Smalltalk will show language features that are not necessarily available in your preferred language. That should not be a problem. It is a time-tested maxim in software development that it is not necessary for the language you are using to natively support a feature in order to program with it; with some diligence, you can find an elegant way to find an alternative for it in your language of choice. According to Steve McConnell's *Code Complete* (2004), this is called programming *into* a language (p. 69):

> Understanding the distinction between programming in a language and programming into one is critical.... Most of the important programming principles depend not on specific languages but on the way you use them. If your language lacks constructs that you want to use or is prone to other kinds of problems, try to compensate for them. Invent your own coding conventions, standards, class libraries, and other augmentations.

Indeed, in a reversal of the SWH that is an homage to programmer creativity (or stubbornness), the author remembers, when object orientation had become *de rigueur* in 1990, coming upon a book at the local technical bookstore treating the subject of object-oriented assembly language. More recently, Randall Hide's High Level Assembler (HLA) has combined assembly with classes, inheritance, and other trappings.

We will approach a programming language as we would approach a classic book. Before we go on in earnest, let us begin with a few suggested definitions from Italo Calvino's essay "Why Read the Classics" (1986):

> The classics are the books of which we usually hear people say, "I am rereading..." and never "I am reading...."
>
> We use the word "classics" for books that are treasured by those who have read and loved them; but they are treasured no less by those who have the luck to read them for the first time in the best conditions to enjoy them.