What Happens Behind the Code

The code that you write is designed to be more or less readable by humans. Behind the scenes, there are two other types of code. At the most basic level, there is *machine code*: these are the instructions that are executed by the computer itself (its *central processing unit* or *CPU*). Each type of computer may have its own language for machine code. In practice, today the machine code is specific to the type of chip that is at the heart of the computer. There may be several chips, but they typically use the same machine code so that it can be executed on whatever chip is available at any time. (See "Threads" in Chapter 6, "Building Components.")

Machine language can be turned into *assembly* by the use of a program called an *assembler*. Assembly (or assembly language) is a bit easier for humans to read. The key component here is the assembler program itself that does the conversion from assembly language to machine code. The code is called assembly language or assembler; while the tool that generates it is often called an assembler. The context clarifies whether you are talking about the tool or the code.

Even more human-readable than assembly is a computer programming language. The first languages such as COBOL and FORTRAN in the 1950s are referred to as higher-level programming languages. They are turned into assembly by programs called *compilers*.

Tip For more information, do research on Grace Hopper and her colleagues.

The hierarchy of programming languages from simple to more complex is this:

- machine code (the most basic);
- assembly;
- higher-level languages such as FORTRAN and COBOL;
- later on languages such as C, Pascal, and now Swift and Python among many others make reading and writing code by and for humans much easier.

Higher-level languages are ideally machine independent so that someone who knows how to write COBOL, FORTRAN, or Swift should be able to have that code compiled into assembler and then assembled into machine language on any computer. Assemblers are usually machine- (or chip-) specific. Thus if you write a program in a higher-level language, it can be compiled into assembler and thence to machine code. Two points are critically important:

- Although higher-level programming languages are *portable* (the common term for running on multiple computer architectures), the compilers that turn them into assembler and machine code are specific to a specific architecture of computer or chip.
- There are often *cross-compilers* that take a higher-level programming language and compile it into assembler for a different computer than the one on which it is running.

Because assembler and machine code are specific to computer and chip architectures as well as the even more important point that most programming and coding today is done in higher-level languages, this book (like most computer science references today) does not go into assembler and machine language except for broad conceptual views such as this section.

In today's coding environment, compiled code is often combined with graphics and many other assets. This process is referred to as *building* (and the result is called a *build*.) The build process is often machine specific. Thus, the code is machine independent if it uses a language such as C, but the build process means that it is not portable. To write code that can run on a specific computer, you need a compiler or cross-compiler that can turn the code into assembler and thence to machine code; you also need a build program to build it for the computer you are targeting (the verb "target" is used to identify the ultimate computer on which your code will run). A build program can even combine several programs in several programming languages if you want.

Compiling and Interpreting Code

Today's computers are much more powerful than those from the early days of programming languages (the 1940s to 1970s). As a result, the very clear pattern of machine code, assembly, and higher-level languages with assemblers and compilers is more complex. Instead of compiling programming language code, in some cases it can be *interpreted*: the higher-level language is processed and turned into executable code so quickly that it happens as quickly as it is typed.

Tip *Executable code* is the term for code that can actually run on a computer. Sometimes, people refer to executable code as an *executable*.
