6.2 Program Units in Java

9

You've already been working with various program units in Java. You write programs by combining new methods and classes with predefined ones available in the **Java Application Programming Interface** (also referred to as the **Java API** or **Java class library**) and in various other class libraries. Related classes are typically grouped into *packages* so that they can be *imported* into programs and *reused*. You'll learn how to group your own classes into *packages* in Section 21.4.10. Java 9 introduces another program unit called *modules*, which we discuss in Chapter 36, Java Module System and Other Java 9 Features.

The Java API provides a rich collection of predefined classes that contain methods for performing common mathematical calculations, string manipulations, character manipulations, input/output operations, database operations, networking operations, file processing, error checking and more.

 Software Engineering Observation 6.1

Familiarize yourself with the rich collection of classes and methods provided by the Java API (`http://docs.oracle.com/javase/8/docs/api`). Section 6.8 overviews several common packages. Online Appendix F explains how to navigate the API documentation. Don't reinvent the wheel. When possible, reuse Java API classes and methods. This reduces program development time and avoids introducing programming errors.

Divide and Conquer with Classes and Methods

Classes and methods help you modularize a program by separating its tasks into self-contained units. The statements in the method bodies are written only once, are hidden from other methods and can be reused from several locations in a program.

One motivation for modularizing a program into classes and methods is the *divide-and-conquer* approach, which makes program development more manageable by constructing programs from small, simple pieces. Another is **software reusability**—using existing classes and methods as building blocks to create new programs. Often, you can create programs mostly from existing classes and methods rather than by building customized code. For example, in earlier programs, we did not define how to read data from the keyboard—Java provides these capabilities in the methods of class `Scanner`. A third motivation is to *avoid repeating code*.

Dividing a program into meaningful classes and methods makes the program easier to debug and maintain.

Software Engineering Observation 6.2

To promote software reusability, every method should be limited to performing a single, well-defined task, and the name of the method should express that task effectively.

Error-Prevention Tip 6.1

A method that performs one task is easier to test and debug than one that performs many tasks.

Software Engineering Observation 6.3

If you cannot choose a concise name that expresses a method's task, your method might be attempting to perform too many tasks. Break such a method into several smaller ones.

Hierarchical Relationship Between Method Calls

As you know, a method is invoked by a method call, and when the called method completes its task, it returns control and possibly a result to the caller. An analogy to this program structure is the hierarchical form of management (Fig. 6.1). A boss (the caller) asks a worker (the called method) to perform a task and report back (return) the results after completing the task. The boss method does not know how the worker method performs its designated tasks. The worker may also call other worker methods, unbeknown to the boss. This "hiding" of implementation details promotes good software engineering. Figure 6.1 shows the `boss` method communicating with several worker methods in a hierarchical manner. The `boss` method divides its responsibilities among the various `worker` methods. Here, `worker1` acts as a "boss method" to `worker4` and `worker5`.

Error-Prevention Tip 6.2

Some methods return a value indicating whether the method performed its task successfully. When you call such a method, be sure to check the return value of that method and, if that method was unsuccessful, deal with the issue appropriately.

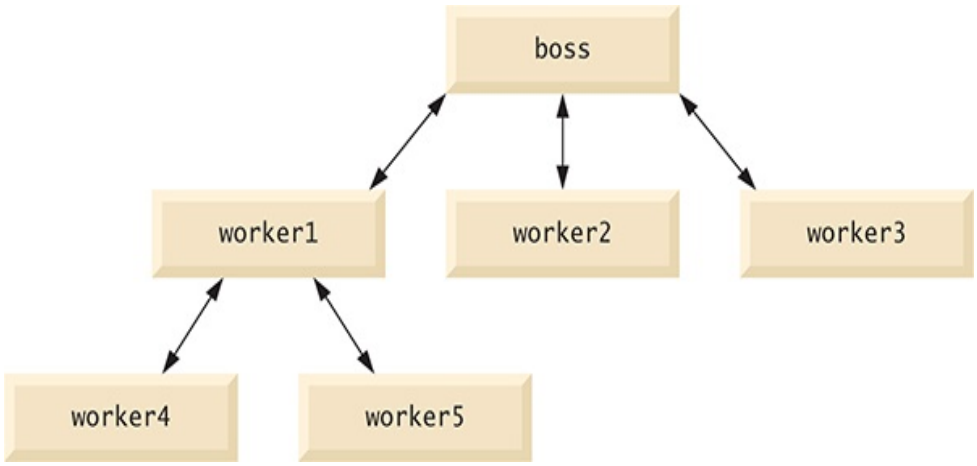

Fig. 6.1

Hierarchical boss-method/worker-method relationship.