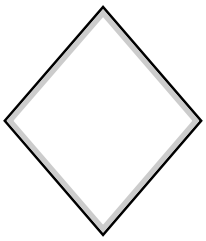

# CHAPTER 1

## *Objects, UML, and Java*

This book is about object-oriented (OO) software development. Writing real object-oriented programs that are used by real people is more than slapping down a few lines of code in Java (or C++, Eiffel, or any other object-oriented programming language). Ultimately, object-oriented software development includes the complete process—analysis of the problem, design of a solution, coding, and long-term maintenance. Object-oriented development can make any program better, from a small Web-based application to a full-blown business-critical software system.

Object orientation has the potential for building great software, but only if it is used as part of a complete process. Today, there are small, agile development methodologies suitable for teams of two to ten or so programmers, as well as large-scale methodologies for huge projects. Most of these development methodologies use or can benefit from the UML (Unified Modeling Language), a modeling tool that aids the design of any OO system. But before you can understand and use any of these methodologies, you need to move beyond merely getting a program to work to changing your thinking to be object-oriented.

It has been said that any programming language can be used to write object-oriented programs (and it has been done with C), but a true OO programming language makes it a lot easier. Just because you use an OO programming language, your programs are not necessarily object-oriented.

Object-oriented programming works much better when it is used together with an object-oriented analysis and design (OOAD) process. Trying to write an

OO program without first going through the analysis and design steps is like trying to build a house without first analyzing the requirements of the house, designing it, and producing a set of blueprints. You might end up with a roof over your head, but the rooms would likely be scattered all over the place, some rooms might be missing, and the whole thing would probably come tumbling down on your head during the first storm (see Figure 1.1). An OO program in any programming language written without at least some OOAD might seem to work, but it is much more likely to be full of bugs and break when you make the first modification.

## Object Orientation

Objects are the heart of object orientation. An object is a representation of almost anything you need to model in a program. An object can be a model of an employee, a representation of a sensor, a window in a user interface, a data structure, such as a list—virtually anything. One way to think of an object is as a black box with some buttons and lights (see Figure 1.2). This could be a TV, a car, whatever. To use the object, you need to know what the buttons do, which

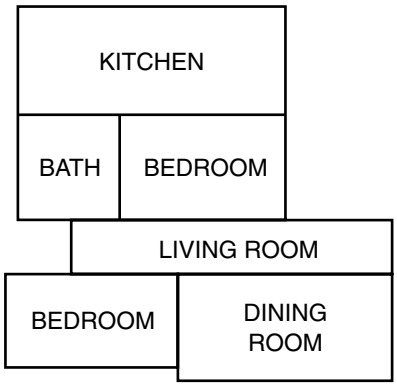


**Figure 1.1** A randomly planned house

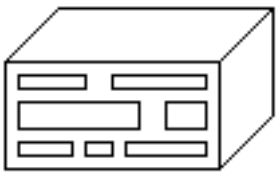

**Figure 1.2** A black box

ones you need to press to get the object to do what you need, and what the lights mean about the status of the object. The details of how the box is put together inside are irrelevant while you are using the box. What is important is that the object carries out its functions and responsibilities correctly. A software object is not much different. It has well-defined methods for interacting with the outside world, and it can provide information about its current state. The internal representation, algorithms, and data structures are hidden from the outside world.

In the simplest terms, designing an OO system consists of identifying which objects the system contains, the behaviors and responsibilities of those objects, and how the objects interact with each other. OO can produce elegant, easy-to-understand designs, which in turn lead to elegant and easy-to-understand programs. Individual objects can often be implemented and debugged independently. Libraries of existing objects can be easily reused and adapted to new designs. Most important, a good OO program is easy to modify and resistant to the introduction of bugs during program modification and maintenance.

Object-oriented development is a major advance for software development. Although it may not be a magic bullet that solves all the problems associated with producing software, it is better than other methodologies. While development methodologies, such as structured design and programming, have many valid points, many which carry over and are used for OO development, object-oriented designs are inherently easier to design and maintain over time.

## Object-Oriented Languages

There are several object-oriented programming languages, including Smalltalk, Eiffel, C++, Objective C, Objective Pascal, Java, Ada, and even a version of Lisp. There are two clear marketplace winners, C++ and Java.

Today, Java is the emerging object-oriented language of choice for many programmers and software projects. One of the reasons for Java's popularity is the World Wide Web and Java's ability to run Web applets directly on any computer or operating system with a Web browser. Another reason is that Java is an excellent programming language. It is a small, well-designed language that can be used not just for Web applets, but for full-blown programs on almost any computer today. Java was somewhat hampered in its early days because of its performance, but this is really no longer an issue. Because it is such a good language, Java has been widely adopted as the main language used to teach computer science at colleges and universities all over the world. In the whole history of computer science and programming, this is the first time the same programming language has been popular as both a teaching language and a language used for real-world programs.

C++ is also a widely used programming language. It is still the principal language used for the core applications (such as spreadsheets and word processors) used on most computers today. C++ was derived from C, and thus has a heritage of being able to do real things on real systems, and there is compatibility with existing C code. One problem with C++, however, is that it has grown into a large and complicated language, and it is difficult to achieve competence in the full language.

This book is mostly about object-oriented programming. That means it will focus on general principles of object-oriented programming that apply to any programming language. But the book will also show how to translate object-oriented designs to real programs using Java. The focus will be on how to use the capabilities of the Java language to implement OO designs. It is not a tutorial on Java. We assume that you've already learned the Java basics. Now you are ready to learn about objects and how to use Java to write better programs.

## Object-Oriented Design and the UML

There are several different object-oriented development methodologies in use today, each with its strengths and weaknesses. The older, more traditional methodologies are often called "heavyweight" methodologies, and are most useful for large software projects involving tens or even hundreds of programmers over years of development effort. The newer methodologies, called "lightweight" or "agile" methodologies, are more appropriate for smaller projects. Many of these are quite new and are still being standardized.

Design and development methodologies have always needed a graphical notation to express the designs. In the past, a major problem has been that each major methodology has had its own graphical notation. This all changed with the emergence of the UML as the standard notation. Any current design methodology, heavyweight or agile, uses or can benefit from the UML.

The UML originated in the mid-1990s from the efforts of James Rumbaugh, Ivar Jacobson, and Grady Booch (The Three Amigos). There is a standard specification of the UML coordinated by the Object Management Group (`www.omg.org`). OMG is an industry-sponsored organization devoted to supporting vendor-neutral standards for the object-oriented development community. The UML has become the de facto standard object-oriented notation.

The UML is designed for discussing object-oriented design. Its ability to show objects and object relationships is especially useful, and it will be used in examples throughout this book. The various features of the UML will be introduced as needed.

## The Payoff of Objects

Object orientation can lead to big payoffs in the software development game. An object-oriented design is likely to be simple and easy to understand. Once designed, you can often implement and test the individual objects separately. Once finished, each object tends to be robust and bug-free. As you make changes to the system, existing objects continue to work. And as you improve existing objects, their interface to the world stays the same, so the whole system continues to work. It is this ease of change and robustness that really make OO development different and well worth the effort.

## Chapter Summary

- Object orientation is a way to develop software that leads to well-designed systems that are robust and easy to maintain.
- The UML is a graphical representation useful for designing and understanding object-oriented systems.
- Java is an excellent object-oriented programming language useful for both Web applets and non-Web applications.