Table 1.1 Analogies between a computer simulation and a laboratory experiment.

Laboratory Experiment	Computer Simulation
sample	model
physical apparatus	computer program
calibration	testing of program
measurement	computation
data analysis	data analysis

a more direct comparison with laboratory experiments. Computation has become a third way of doing physics and complements both theory and experiment.

Computer simulations, like laboratory experiments, are not substitutes for thinking, but are tools that we can use to understand natural phenomena. The goal of all our investigations of fundamental phenomena is to seek explanations of natural phenomena that can be stated concisely.

1.3 ■ PROGRAMMING LANGUAGES

There is no single best programming language any more than there is a best natural language. Fortran is the oldest of the more popular scientific programming languages and was developed by John Backus and his colleagues at IBM between 1954 and 1957. Fortran is commonly used in scientific applications and continues to evolve. Fortran 90/95/2000 has many modern features that are similar to C/C++.

The Basic programming language was developed in 1965 by John Kemeny and Thomas Kurtz at Dartmouth College as a language for introductory courses in computer science. In 1983 Kemeny and Kurtz extended the language to include platform independent graphics and advanced control structures necessary for structured programming. The programs in the first two editions of our textbook were written in this version of Basic, known as True Basic.

C was developed by Dennis Ritchie at Bell Laboratories around 1972 in parallel with the Unix operating system. C++ is an extension of C designed by Bjarne Stroustrup at Bell laboratories in the mid-eighties. C++ is considerably more complex than C and has object-oriented features as well as other extensions. In general, programs written in C/C++ have high performance but can be difficult to debug. C and C++ are popular choices for developing operating systems and software applications because they provide direct access to memory and other system resources.

Python, like Basic, was designed to be easy to learn and use. Python enthusiasts like to say that C and C++ were written to make life easier for the computer, but Python was designed to be easier for the programmer. Guido van Rossum created Python in the late 80's and early 90's. It is an interpreted, object-oriented, general-purpose programming language that is also good for prototyping. Because Python is interpreted, its performance is significantly less than optimized languages like C or Fortran.

Java is an object-oriented language that was created by James Gosling and others at Sun Microsystems. Since Java was introduced in late 1995, it has rapidly evolved and is the

language of choice in most introductory computer science courses. Java borrows much of its syntax from C++ but has a simpler structure. Although the language contains only fifty keywords, the Java *platform* adds a rich library that enables a Java program to connect to the internet, render images, and perform other high-level tasks.

Most modern languages incorporate *object-oriented* features. The idea of object-oriented programming is that functions and data are grouped together in an *object*, rather than treated separately. A program is a structured collection of objects that communicate with each other causing the internal state within a given object to change. A fundamental goal of object-oriented design is to increase the understandability and reusability of program code by focusing on what an object does and how it is used, rather than how an object is implemented.

Our choice of Java for this text is motivated in part by its platform independence, flexible standard graphics libraries, good performance, and its no cost availability. The popularity of Java ensures that the language will continue to evolve, and that programming experience in Java is a valuable and marketable skill. The Java programmer can leverage a vast collection of third-party libraries, including those for numerical calculations and visualization. Java is also relatively simple to learn, especially the subset of Java that we will need to simulate physical systems.

Java can be thought of as a platform in itself, similar to the Macintosh and Windows, because it has an application programming interface (API) that enables cross-platform graphics and user interfaces. Java programs are compiled to a platform neutral byte code so that they can run on any computer that has a Java Virtual Machine. Despite the high level of abstraction and platform independence, the performance of Java is becoming comparable with native languages. If a project requires more speed, the computationally demanding parts of the program can be converted to C/C++ or Fortran.

Readers who wish to use another programming language should find the algorithmic components of the Java program listings in the text to be easily convertible into a language of their choice.

1.4 ■ OBJECT-ORIENTED TECHNIQUES

If you already know how to program, try reading a program that you wrote several years or even several weeks ago. Many of us would not be able to follow the logic of our own program and would have to rewrite it. And your program would probably be of little use to a friend who needs to solve a similar problem. If you are learning programming for the first time, it is important to learn good programming habits to minimize this problem. One way is to employ object-oriented techniques such as encapsulation, inheritance, and polymorphism.

Encapsulation refers to the way that an object's essential information is exposed through a well-documented interface, but unnecessary details of the code are hidden. For example, we can model a particle as an object. Whenever a particle moves, it calculates its acceleration from the total force on it. Someone who wishes to use the trajectory of the particle, for example to animate the particle's trajectory, needs to refer only to the interface and does not need to know how the trajectory is calculated.

Inheritance allows a programmer to add capabilities to existing code without having to rewrite it or even know the details of how the code works. For example, you will write pro-