grams that show the evolution of planetary systems, quantum mechanical wave functions, and molecular models. Many of these programs will use (extend) code in the Open Source Physics library known as an `AbstractSimulation`. This code has a timer that periodically executes code in your program and then refreshes the on-screen animation. Using the Open Source Physics library will let you focus your efforts on programming the physics, because it is not necessary to write the code to produce the timer or to refresh the screen. Similarly, we have designed a general purpose graphical user interface (GUI) by extending code written by Sun Microsystems known as a `JFrame`. Our GUI has the features of a standard user interface such as a menu bar, minimize button, and title, even though we did not write the code to implement these features.

*Polymorphism* helps us to write reusable code. For example, it is easy to imagine many types of objects that are able to evolve over time. In Chapter 15 we will simulate a system of particles using random numbers rather than forces to move the particles. By using polymorphism, we can write general purpose code to do animations with both types of systems.

Science students have a rich context in which to learn programming. The past several decades of doing physics with computers has given us numerous examples that we can use to learn physics, programming, and data analysis. Unlike many programming manuals, the emphasis of this book is on learning by example. We will not discuss all aspects of Java, and this text is not a substitute for a text on Java. Think of how you learned your native language. First you learned by example, and then you learned more systematically.

Although using an object-oriented language makes it easier to write well-structured programs, it does not guarantee that your programs will be well written or even correct. The single most important criterion of program quality is readability. If your program is easy to read and follow, it is probably a good program. There are many analogies between a good program and a well-written paper. Few papers and programs come out perfectly on their first draft, regardless of the techniques and rules we use to write them. Rewriting is an important part of programming.

## 1.5 ■ HOW TO USE THIS BOOK

Most chapters in this text begin with a brief background summary of the nature of a system and the important questions. We then introduce the computer algorithms, explain new syntax as needed, and discuss a sample program. The programs are meant to be read as text on an equal basis with the discussions and are interspersed throughout the text. It is strongly recommended that all the problems be read, because many concepts are introduced after you have had a chance to think about the result of a simulation.

It is a good idea to maintain a computer-based notebook to record your programs, results, graphical output, and analysis of the data. This practice will help you develop good habits for future research projects, prevent duplication, organize your thoughts, and save you time. After a while you will find that most of your new programs will use parts of your earlier programs. Ideally, you will use your files to write a laboratory report or a paper on your work. Guidelines for writing a laboratory report are given in Appendix 1A.

Many of the problems in the text are open ended and do not lend themselves to simple "back of the book" answers. So how will you know if your results are correct? How will you

know when you have done enough? There are no simple answers to either question, but we can give some guidelines. First, you should compare the results of your program to known results whenever possible. The known results might come from an analytical solution that exists in certain limits or from published results. You should also look at your numbers and graphs and determine if they make sense. Do the numbers have the right sign? Are they the right order of magnitude? Do the trends make sense as you change the parameters? What is the statistical error in the data? What is the systematic error? Some of the problems explicitly ask you to do these checks, but you should make it a habit to do as many as you can whenever possible.

How do you know when you are finished? The main guideline is whether you can tell a coherent story about your system of interest. If you have only a few numbers and do not know their significance, then you need to do more. Let your curiosity lead you to more explorations. Do not let the questions asked in the problems limit what you do. The questions are only starting points, and frequently you will be able to think of your own questions.

The following problem is an example of the kind of problems that will be posed in the following chapters. Note its similarity to the questions posed on page 3. Although most of the simulations that we will do will be on the kind of physical systems that you will encounter in other physics courses, we will consider simulations in related areas such as traffic flow, small world networks, and economics. Of course, unless you already know how to do simulations, you will have to study the following chapters so that you will able to do problems like the following.

**Problem 1.1 Distribution of money**

The distribution of income in a society $f(m)$ behaves as $f(m) \propto m^{-1-\alpha}$, where $m$ is the income (money) and the exponent $\alpha$ is between 1 and 2. The quantity $f(m)$ can be taken to be the number of people who have an amount of money between $m$ and $m + \Delta m$. This power law behavior of the income distribution is often referred to as Pareto's law or the 80/20 rule (20% of the people have 80% of the income) and was proposed in the late 1800's by Vilfredo Pareto, an economist and sociologist. In the following, we consider some simple models of a closed economy to determine the relation between the microdynamics and the resulting macroscopic distribution of money.

(a) Suppose that $N$ agents (people) can exchange money in pairs. For simplicity, we assume that all the agents are initially assigned the same amount of money $m_0$, and the agents are then allowed to interact. At each time step, a pair of agents $i$ and $j$ with money $m_i$ and $m_j$ is randomly chosen and a transaction takes place. Again for simplicity, let us assume that $m_i \to m_i'$ and $m_j \to m_j'$ by a random reassignment of their total amount of money, $m_i + m_j$, such that

$$m_i' = \epsilon(m_i + m_j) \quad (1.1a)$$

$$m_j' = (1 - \epsilon)(m_i + m_j), \quad (1.1b)$$

where $\epsilon$ is a random number between 0 and 1. Note that this reassignment ensures that the agents have no debt after the transaction, that is, they are always left with an amount $m \geq 0$. Simulate this model and determine the distribution of money among the agents after the system has relaxed to an equilibrium state. Choose $N = 100$ and $m_0 = 1000$.