the parameters before the doStep method is executed. If you wish to allow a parameter to be changed between computations, you can use the optional startRunning method. This method is invoked once when the Step button is clicked and once when the Run button is clicked. In other words this method is called before the thread starts and insures that the simulation has the opportunity to read the most recent values.

In BouncingBallApp the time step dt is set using the setAdjustableValue method rather than the setValue method. Parameters that are set using setAdjustableValue are editable in the SimulationControl after the program has been initialized, whereas those that are set using setValue are only editable before the program has been initialized.

Exercise 2.19 Follow the bouncing balls

(a) Run BouncingBallApp and try the different buttons and note how they affect the input parameters.

(b) Add the statement enableStepsPerDisplay(true) to the reset method, and run your program again. You should see a new input in the control window that lets you change the number of simulation steps that are computed between redrawing the frame. Vary this input and note what happens.

(c) What is wrong with the physics of the simulation?

(d) Add a method to the BouncingBall class to calculate and return the total energy. Sum the energy of the balls in the program's doStep method and display this value in the message box. Does the simple model make sense?

(e) Look at the source code for the setXY method. If you are using an Integrated Development Environment (IDE), finding the method and looking at the source code is easy. What would you need to do to change the radius of the circle that is drawn? ■

Many of the visualization components in the Open Source Physics library are written using classes provided by others, including programmers at Sun Microsystems. The goal of this library is to make it easier for you to begin writing your own programs. You are encouraged to look under the hood as you gain experience. The Open Source Physics controls and visualizations will almost always inherit from the JFrame class. Drawing is almost always done on a DrawingPanel which inherits from the JPanel class. Both these superclasses are defined in the javax.swing package.

Exercise 2.20 Peeking into Open Source Physics

(a) Look at the source code for PlotFrame (in the frames package) and follow its inheritance until you reach the JFrame class. How may subclasses are there between JFrame and PlotFrame? Follow the inheritance from SimulationControl (in the controls package) to JFrame. Describe in general terms what features are added in each subclass.

(b) Read through the different methods in PlotFrame. Don't worry about how the methods are implemented, but try to understand what they do. Which methods have not yet appeared in a program listing? When might you use them?

(c) Look at the source code for PlottingPanel (in the display package), which is used in many of the frames. Follow its inheritance until you reach the JPanel class. Do you see why we have not described the PlottingPanel class in detail? Look through the various methods, and describe in your own words what several of them do and how they might be used.

(d) Find the closest common ancestor (superclass) for JFrame and JPanel in the core Java library. Note that all objects have Object as a common ancestor. ■

2.7 ■ MODEL-VIEW-CONTROLLER

Developing large software programs is best viewed as a design process. One criterion for good design is the reuse of data structures and behaviors that can facilitate reuse. Separating the physics (the model) from the user interface (the controller) and the data visualization (the view) facilitate good design. In Open Source Physics, the control object is responsible for handling user initiated events, such as button clicks, and passing them to other objects. The plots that we have constructed present visual representations of the data and are examples of a *view*. By using this design strategy, it is possible to have multiple views of the same data. For example, we can show a plot and a table view of the same data. The physics is expressed in terms of a *model* which contains the data and provides the methods by which the data can change.

At this point we have described a large fraction of the Java syntax and Open Source Physics tools that we will need in the rest of this book. One important topic that we still need to discuss is the use of *interfaces*. There is also much more in the Open Source Physics library that we can use. For example, there are classes to draw and manipulate lattices as well as classes to iterate differential equations more accurately than the Euler method used in this chapter.

At this stage we hope that you have gained a feel for how Java works, and can focus on the physics in the rest of the text. Additional aspects of Java will be taught by example as they are needed.

APPENDIX 2A: COMPLEX NUMBERS

Complex numbers are used in physics to represent quantities such as alternating currents and quantum mechanical wave functions which have an amplitude and phase (see Figure 2.2). Java does not provide a complex number as a primitive data type, so we will write a class that implements some common complex arithmetic operations. This class is an explicit example of the fact that classes are effectively new programmer-defined types.

If our new class is called Complex, we could test it by using code such as the following:

```
package org.opensourcephysics.sip.ch02;
public class ComplexApp {
   public static void main(String[] args) {
      Complex a = new Complex(3.0, 2.0);  // complex number 3 + i2
      Complex b = new Complex(1.0, -4.0); // complex number 1 - i4
      System.out.println(a); // display a using a.toString()
      System.out.println(b); // display b using b.toString()
      Complex sum = b.add(a); // add a to b
      System.out.println(sum); // display sum
      Complex product = b.multiply(a); // multiply b by a
      System.out.println(product);
```