```
public class FallingParticlePlotApp extends AbstractCalculation {
   PlotFrame plotFrame = new PlotFrame("t", "y", "Falling Ball");

   public void calculate() {
      // data not cleared at beginning of each calculation
      plotFrame.setAutoclear(false);
      // gets initial conditions
      double y0 = control.getDouble("Initial y");
      double v0 = control.getDouble("Initial v");
      // sets initial conditions
      Particle ball = new FallingParticle(y0, v0);
      // gets parameters
      ball.dt = control.getDouble("dt");
      double t = ball.t; // gets value of time from ball object
      while(ball.y>0) {
         ball.step();
         plotFrame.append(0, ball.t, ball.y);
         plotFrame.append(1, ball.t, ball.analyticPosition());
      }
   }

   public void reset() {
      control.setValue("Initial y", 10);
      control.setValue("Initial v", 0);
      control.setValue("dt", 0.01);
   }

   // sets up calculation control structure using this class
   public static void main(String[] args) {
      CalculationControl.createApp(new FallingParticlePlotApp());
   }
}
```

The two data sets, indexed by 0 and 1, correspond to the numerical data and the analytical results, respectively. The default action in the Open Source Physics library is to clear the data and redraw data frames when the Calculate button is clicked. This automatic clearing of data can be disabled using the `setAutoclear` method. We have disabled it here to allow the user to compare the results of multiple calculations. Data is automatically cleared when the Reset button is clicked.

Exercise 2.17 Data output

(a) Run `FallingParticlePlotApp`. Under the Views menu choose Data Table to see a table of data corresponding to the plot. You can copy this data and use it in another program for further analysis.

(b) Your plotted results probably look like one set of data because the numerical and analytical results are similiar. Let `dt` = 0.1 and click the Calculate button. Does the discrepancy between the numerical and analytical results become larger with increasing time? Why?

(c) Run the program for two different values of `dt`. How do the plot and the table of data differ when two runs are done, first separated without clicking Reset, and then done by clicking Reset between calculations? Make sure you look at the entire table to see the difference. When is the data cleared? What happens if you eliminate the `plotFrame.setAutoclear(false)` statement? When is the data cleared now?

(d) Modify your program so that the velocity is shown in a separate window from the position. ■

2.6 ■ ANIMATION AND SIMULATION

The `AbstractCalculation` class provides a structure for doing a single computation for a fixed amount of time. However, frequently we do not know how long we want to run a program, and it would be desirable if the user could intervene at any time. In addition, we would like to be able to visualize the results of a simulation and do an animation. To do so involves a programming construct called a *thread*. Threads enable a program to execute statements independently of each other as if they were run on separate processors (which would be the case on a multiprocessor computer). We will use one thread to update the model and display the results. The other thread, the event thread, will monitor the keyboard and mouse so that we can stop the computation whenever we desire.

The `AbstractSimulation` class provides a structure for doing simulations by performing a series of computations (steps) that can be started and stopped by the user using a graphical user interface. You will need to know nothing about threads because their use is "hidden" in the `AbstractSimulation` class. However, it is good to know that the Open Source Physics library is written so that the graphical user interface does not let us change a program's input parameters while the simulation is running. Most of the programs in the text will be done by extending the `AbstractSimulation` class and implementing the `doStep` method as shown in Listing 2.12. Just as the `AbstractCalculation` class uses the graphical user interface of type `CalculationControl`, the `AbstractSimulation` class uses one of type `SimulationControl`. This graphical user interface has three buttons whose labels change depending on the user's actions. As was the case with `CalculationControl`, the buttons in `SimulationControl` invoke specific methods.

Listing 2.12 A simple example of the extension of the `AbstractSimulation` class.

```
package org.opensourcephysics.sip.ch02;
import org.opensourcephysics.controls.AbstractSimulation;
import org.opensourcephysics.controls.SimulationControl;

public class SimulationApp extends AbstractSimulation {
   int counter = 0;

   public void doStep() { // does a simulation step
      control.println("Counter = "+(counter--));
   }

   public void initialize() {
      counter = control.getInt("counter");
   }

   public void reset() { // invoked when reset button is pressed
      // allows dt to be changed after initializaton
      control.setAdjustableValue("counter", 100);
   }

   public static void main(String[] args) {
      // creates a simulation structure using this class
```