```
public void calculate () {
   // String must match argument of setValue
   double x = control.getDouble("x value");
   control.println("x*x = " + (x*x));
}
```

Exercise 2.14 Changing parameters

(a) Run `CalculateApp` to see how the control window can be used to change a program's parameters. What happens if the string in the `getDouble` method does not match the string in the `setValue` method?
(b) Incorporate the plot statements in Listing 2.8 into a new class that extends the class `AbstractCalculation` and plots the function sin kx for various values of the input parameter k. ■

When you run the modified `CalculationApp` in Exercise 2.14, you should see a window with two buttons and an input parameter and its default value. Also, there should be a text area below the buttons where messages can appear. When the Calculate button is clicked, the `calculate` method is executed. The `control.getDouble` method reads in values from the control window. These values can be changed by the user. Then the calculation is performed and the result is displayed in the message area using the `control.println` method, similar to the way we used `System.out.println` earlier. If the Reset button is clicked, the message area is cleared and the `reset` method is called.

We will now use a `CalculationControl` to change the input parameters for a falling particle. The modified `FallingParticleApp` is shown in Listing 2.10.

Listing 2.10 `FallingParticleCalcApp` class.

```
package org.opensourcephysics.sip.ch02;
import org.opensourcephysics.controls.*;

// beginning of class definition
public class FallingParticleCalcApp extends AbstractCalculation {
   public void calculate() {
      // gets initial conditions
      double y0 = control.getDouble("Initial y");
      double v0 = control.getDouble("Initial v");
      // sets initial conditions
      Particle ball = new FallingParticle(y0, v0);
      // reads parameters and sets dt
      ball.dt = control.getDouble("dt");
      while(ball.y>0) {
         ball.step();
      }
      control.println("final time = "+ball.t);
      // displays numerical results
      control.println("y = "+ball.y+" v = "+ball.v);
      // displays analytic position
      control.println("analytic y = "+ball.analyticPosition());
      // displays analytic velocity
      control.println("analytic v = "+ball.analyticVelocity());
   }

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

   // creates a calculation control structure using this class
   public static void main(String[] args) {
      CalculationControl.createApp(new FallingParticleCalcApp());
   }
} // end of class definition
```

Exercise 2.15 Input of parameters and initial conditions

(a) Run `FallingParticleCalcApp` and make sure you understand how the control works. Try inputting different values of the parameters and the initial conditions.
(b) Vary Δt and find the value of t when $y = 0$ to two decimal places. ■

Exercise 2.16 Displaying floating point numbers
Double precision numbers store 16 significant digits, and every digit is included when the number is converted to a string. We can reduce the number of digits that are displayed using the `DecimalFormat` class in the `java.text` package. A formatter is created using a pattern, such as `#0.00` or `#0.00E0`, and this format is applied to a number to produce a string.

```
DecimalFormat decimal2 = new DecimalFormat("#0.00");
double x = 1.0/3.0;
System.out.println("x = "+decimal2.format(x)); // displays 3.33
```

(a) Use the `DecimalFormat` class to modify the output from `FallingParticleCalcApp` so that it matches the output shown in Figure 2.1.
(b) Modify the output so that results are shown using scientific notation with three decimal places.
(c) The Open Source Physics `ControlUtils` class in the controls package contains a static method `f3` that formats a floating point number using three decimal places. Use this method to format the output from `FallingParticleCalcApp`. ■

You probably have found that it is difficult to write a program so that it ends exactly when the falling ball is at $y = 0$. We could write the program so that Δt keeps changing near $y = 0$ so that the last value computed is at $y = 0$. Another limitation of our programs that we have written so far is that we have shown the results only at the end of the calculation. We could put `println` statements inside the while loop, but it would be better to plot the results and have a table of the data. An example is shown in Listing 2.11.

Listing 2.11 `FallingParticlePlotApp` class.

```
package org.opensourcephysics.sip.ch02;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.*;
```