```
         frame.append(0, x, Math.sin(x)/x);
      }
      frame.setVisible(true);
      frame.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
   }
}
```

The `import` statement tells the Java compiler where to find the Open Source Physics classes that are needed. A *frame* is often referred to as a window and can include a title and a menu bar as well as objects such as buttons, graphics, and text information. The Open Source Physics frames package defines several frames that contain data visualization and analysis tools. We will use the `PlotFrame` class to plot x-y data. The constructor for `PlotFrame` has three arguments corresponding to the name of the horizontal axis, the name of the vertical axis, and the title of the plot. To add data to the plot, we use the `append` method. The first argument of `append` is an integer that labels a particular set of data points, the second argument is the horizontal (x) value of the data point, and the third argument is the vertical (y) value. The `setVisible(true)` method makes a frame appear on the screen or brings it to the front. The last statement makes the program exit when the frame is closed. What happens when this statement is not included?

The example from the Open Source Physics library in Listing 2.9 illustrates how to control a *calculation* with two buttons, determine the value of an input parameter, and display the result in the text message area.

Listing 2.9 An example of a `Calculation`.

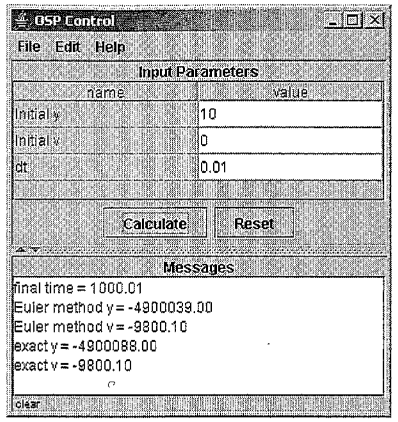
```
package org.opensourcephysics.sip.ch02;
// * means get all classes in controls
// subdirectory
import org.opensourcephysics.controls.*;

public class CalculationApp extends AbstractCalculation {
   public void calculate() { // Does a calculation
      control.println("Calculation button pressed.");
      // String must match argument of setValue
      double x = control.getDouble("x value");
      control.println("x*x = "+(x*x));
      control.println("random = "+Math.random());
   }

   public void reset() {
      // describes parameter and sets its value
      control.setValue("x value", 10.0);
   }

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

`AbstractCalculation` is an *abstract* class, which as we have seen means that it cannot be instantiated directly and must be extended in order to implement the `calculate` method, that is, you must write (implement) the `calculate` method. You can also write an optional `reset` method, which is called whenever the Reset button is clicked. Finally, we need to



name	value
Initial y	10
Initial v	0
dt	0.01

```
final time = 1000.01
Euler method y = -4900039.00
Euler method v = -9800.10
exact y = -4900088.00
exact v = -9800.10
```

Figure 2.1 An Open Source Physics control that is used to input parameter values and display results.

create a graphical user interface that will invoke methods when the Calculate and Reset buttons are clicked. This user interface is an object of type `CalculationControl`:

```
CalculationControl.createApp(new CalculationApp());
```

The method `createApp` is a static method that instantiates a `CalculationControl` object and returns this object. We could have written

```
CalculationControl control =
     CalculationControl.createApp(new CalculationApp());
```

which shows explicitly the returned object which we gave the name `control`. However, because we do not use the object `control` explicitly in the `main` method, we do not need to actually declare an object name for it.

Exercise 2.13 CalculationApp
Compile and run `CalculationApp`. Describe what the graphical user interface looks like and how it works by clicking the buttons (see Figure 2.1). ■

The `reset` method is called automatically when a program is first created and whenever the Reset button is clicked. The purpose of this method is to clear old data and recreate the initial state with the default values of the parameters and instance variables. The default values of the parameters are displayed in the control window so that they can be changed by the user. An example of how to show values in a control follows:

```
public void reset () {
   // describes parameter and sets the value
   control.setValue("x value",10.0);
}
```

The string appearing in the `setValue` method must be identical to the one appearing in the `getDouble` method. If you write your own `reset` method, it will override the `reset` method that is already defined in the `AbstractCalculation` superclass.

After the `reset` method stores the parameters in the control, the user can edit the parameters, and we can later read these parameters using the `calculate` method: