```
   public double getStepSize();
}
```

A system of first-order differential equations is now solved by creating an object that implements a particular algorithm and repeatedly invoking the `step` method for that solver class. The argument for the solver class constructor must be a class that implements the `ODE` interface. As an example of the use of `ODESolver`, we again consider the dynamics of a falling particle.

Listing 3.7 A falling particle program that uses an `ODESolver`.

```
package org.opensourcephysics.sip.ch03;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.numerics.*;

public class FallingParticleODEApp extends AbstractCalculation {
   public void calculate() {
      // gets initial conditions
      double y0 = control.getDouble("Initial y");
      double v0 = control.getDouble("Initial v");
      // creates ball with initial conditions
      FallingParticleODE ball = new FallingParticleODE(y0, v0);
      // note how particular algorithm is chosen
      ODESolver solver = new Euler(ball);
      // sets time step dt in the solver
      solver.setStepSize(control.getDouble("dt"));
      while(ball.state[0]>0) {
         solver.step();
      }
      control.println("final time = "+ball.state[2]);
      control.println("y = "+ball.state[0]+" v = "+ball.state[1]);
   }

   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 for this class
   public static void main(String[] args) {
      CalculationControl.createApp(new FallingParticleODEApp());
   }
}
```

The ODE classes are located in the numerics package, and thus we need to import this package as done in the third statement of `FallingParticleODEApp`. We declare and instantiate the variables `ball` and `solver` in the `calculate` method. Note that `ball`, an instance of `FallingParticleODE`, is the argument of the `Euler` constructor. The object `ball` can be an argument because `FallingParticleODE` implements the `ODE` interface.

It would be a good idea to look at the source code of the ODE `Euler` class in the numerics package. The `Euler` class gets the state of the system using `getState` and then sends this state to `getRate` which stores the rates in the `rate` array. The `state` array is then modified

using the `rate` array in the Euler algorithm. You don't need to know the details, but you can read the `step` method of the various classes that implement `ODESolver` if you are interested in how the different algorithms are programmed.

Because `FallingParticleODE` appears to be more complicated than `FallingParticle`, you might ask what we have gained. One answer is that it is now much easier to use a different numerical algorithm. The only modification we need to make is to change the statement

```
ODESolver solver = new Euler(ball);
```

to, for example,

```
ODESolver solver = new EulerRichardson(ball);
```

We have separated the physics (in this case a freely falling particle) from the implementation of the numerical method.

Exercise 3.6 ODE solvers
Run `FallingParticleODEApp` and compare your results with our previous implementation of the Euler algorithm in `FallingParticleApp`. How easy is it to use a different algorithm? ■

3.7 ■ EFFECTS OF DRAG RESISTANCE

We have introduced most of the programming concepts that we will use in the remainder of this text. If you are new to programming, you will likely feel a bit confused at this point by all the new concepts and syntax. However, it is not necessary to understand all the details to continue and begin to write your own programs. A prototypical simulation program is given in Listings 3.8 and 3.9. These classes simulate a projectile on the surface of the Earth with no air friction, including a plot of position versus time and an animation of a projectile moving through the air. In the following, we discuss more realistic models that can be simulated by modifying the projectile classes.

Listing 3.8 A simple projectile simulation that is useful as a template for other simulations.

```
package org.opensourcephysics.sip.ch03;
import java.awt.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.numerics.*;

public class Projectile implements Drawable, ODE {
   static final double g = 9.8;
   double[] state = new double[5]; // {x,vx,y,vy,t}
   // pixel radius for drawing projectile
   int pixRadius = 6;
   EulerRichardson odeSolver = new EulerRichardson(this);

   public void setStepSize(double dt) {
      odeSolver.setStepSize(dt);
   }

   public void step() {
```