The `FallingBall` class in Listing 2.2 cannot be used in isolation because it does not contain a `main` method. Thus, we create a target class which we place in a separate file in the same package. This class will communicate with `FallingBall` and include the output statements. This class is shown in Listing 2.3.

Listing 2.3 `FallingBallApp` class.

```
// package statement appears before beginning of class definition
package org.opensourcephysics.sip.ch02;
// beginning of class definition
public class FallingBallApp {
   // beginning of method definition
   public static void main(String[] args) {
      // declaration and instantiation
      FallingBall ball = new FallingBall();
      // example of declaration and assignment statement
      double y0 = 10;
      double v0 = 0;
      // note use of dot operator to access instance variable
      ball.t = 0;
      ball.dt = 0.01;
      ball.y = y0;
      ball.v = v0;
      while(ball.y>0) {
         ball.step();
      }
      System.out.println("Results");
      System.out.println("final time = "+ball.t);
      // displays numerical results
      System.out.println("y = "+ball.y+" v = "+ball.v);
      // displays analytic results
      System.out.println("analytic y = "+ball.analyticPosition(y0,v0));
      System.out.println("analytic v = "+ball.analyticVelocity(v0));
      System.out.println("acceleration = "+FallingBall.g);
   } // end of method definition
} // end of class definition
```

Note how `FallingBall` is declared and *instantiated* by creating an object called `ball` and how the instance variables and the methods are accessed. The statement

```
FallingBall ball = new FallingBall();  // declaration and instantiation
```

is equivalent to two statements:

```
FallingBall ball;          // declaration
ball = new FallingBall();  // instantiation
```

The declaration statement tells the compiler that the variable `ball` is of type `FallingBall`. It is analogous to the statement `int x` for an integer variable. The `new` operator allocates memory for this object, initializes all the instance variables, and invokes the constructor. We can create two identical balls using the following statements:

```
FallingBall ball1 = new FallingBall();
FallingBall ball2 = new FallingBall();
```

The variables and methods of an object are accessed by using the *dot operator*. For example, the variable `t` of object `ball` is accessed by the expression `ball.t`, and the

method `step` is called as `ball.step()`. Because the methods `analyticPosition` and `analyticVelocity` return values of type `double`, they can appear in any expression in which a double-valued constant or variable can appear. In the present context, the values returned by these two methods will be displayed by the `println` statement. Note that the static variable `g` in class `FallingBallApp` is accessed through the class name.

Exercise 2.7 Use of two classes

(a) Enter the `FallingBall` listing into a file FallingBall.java and `FallingBallApp` into a file FallingBallApp.java and put the files in the same directory. Run your program and make sure your results are the same as those found in Exercise 2.5.

(b) Modify `FallingBallApp` by adding a second instance variable `ball2` of the same type as `ball`. Add the necessary code to initialize `ball2`, iterate `ball2`, and display the results for both objects. Write your program so that the only difference between the two balls is the value of Δt. How much smaller does Δt have to be to reduce the error in the numerical results by a factor of two for the same final time? What about a factor of four? How does the error depend on Δt?

(c) Add the statement `FallingBall.g = 2.0` to your program from part (b) and use the same value of `dt` for `ball` and `ball2`. What happens when you try to compile the program?

(d) Delete the `final` qualifier for `g` in `FallingBall` and recompile and run your program. Is there any difference between the results for the two balls? Is there a difference between the results compared to what you found for $g = 9.8$?

(e) Remove the qualifier `static`. Now `g` must be accessed using the object name `ball` or `ball2` instead of `FallingBall`. Recompile your program again, and run your program. How do the results for the two balls compare now?

(f) Explain the meaning of the qualifiers `static` and `final`. ■

It is possible for a class to have more than one constructor. For example, we could have a second constructor defined by

```
public FallingBall(double dt) {
   // "this.dt" refers to an instance variable that has the
   // same name as the argument
   this.dt = dt;
}
```

Note the possible confusion of the variable name `dt` in the argument of the `FallingBall` constructor and the variable defined near the beginning of the `FallingBall` class. A variable that is passed to a method as an argument (parameter) or that is defined (created) within a method is known as a *local variable*. A variable that is defined outside of a method is known as an *instance variable*. Instance variables are more powerful than local variables because they can be referenced (used) anywhere within an object, and because their values are not lost when the execution of the method is finished. When a variable name conflict occurs, it is necessary to use the keyword `this` to access the instance variable. Otherwise, the program would access the variable in the argument (the local variable) with the same name.