After the program finishes the loop, the end results of the simulation are displayed using the `System.out.println` method. We will explain the meaning of this syntax later. The parameter passed to this method, which appears between the parentheses, is a `String`. A `String` is a sequence of characters and can be created by enclosing text in quotation marks as shown in the first `println` statement in Listing 2.1. We displayed our numerical results by using the + operator. When applied to a `String` and a number, the number is converted to the appropriate `String` and the two strings are concatenated (joined). This use is shown in the next three `println` statements in lines 27, 29, and 33 of Listing 2.1. Note the different outputs produced by the following two statements:

```
System.out.println(("x = " + 2) + 3);     // displays x = 23
System.out.println("x = " + (2 + 3));     // displays x = 5
```

The parentheses in the second line force the compiler to treat the enclosed + operator as the addition operator, but both + operators in the first line are treated as concatenation operators.

Exercise 2.4 Exploring FirstFallingBallApp

(a) Run `FirstFallingBallApp` for various values of the time step Δt. Do the numerical results become closer to the analytic results as Δt is made smaller?

(b) Use an acceptable value for Δt and run the program for various values for the number of iterations. What criteria do you have for acceptable? At approximately what time does the ball hit the ground at $y = 0$?

(c) What happens if you replace `System.out.println` by `System.out.print`?

(d) What happens if you try to access the value of the counter variable `n` outside the for loop? The *scope* of `n` extends from its declaration to the end of the loop block; `n` is said to have *block scope*. If a loop variable is not needed outside the loop, it should be declared in the initialization expression so that its scope is limited. ■

You might have found that doing Exercise 2.4 was a bit tedious and frustrating. To do Exercise 2.4(a) it would be desirable to change the number of iterations at the same time that the value of Δt is changed so that we could compare the results for y and v at the same time. And it is difficult to do Exercise 2.4(b) because we don't know in advance how many iterations are needed to reach the ground. For starters we can improve `FirstFallingBallApp` using a `while` statement instead of the `for` loop.

```
while (y > 0) {
   // statements go here
}
```

In this example the boolean test for the `while` statement is done at the beginning of a loop. It is also possible to do the test at the end:

```
do {
   // statements go here
}
while (y > 0);
```

Exercise 2.5 Using while statements

Modify `FirstFallingBallApp` so that the `while` statement is used and the program ends when the ball hits the ground at $y = 0$. Then repeat Exercise 2.4(b). ■

Exercise 2.6 Summing a series

(a) Write a program to sum the following series for a given value of N:

$$S = \sum_{m=1}^{N} \frac{1}{m^2}. \tag{2.10}$$

The following statements may be useful:

```
double sum = 0;           // sum is equivalent to S in (2.9)
for (int m = 1; m <= N; m++) {
   sum = sum + 1.0/(m*m); // put this statement in for loop
}
```

Note that in this case it is more convenient to start the loop from $m = 1$ instead of $m = 0$. Also note that we have not followed the Java convention, because we have used the variable name `N` instead of `n` so that the Java statements look more like the mathematical equations.

(b) First run your program with $N = 10$. Then run for larger values of N. Does the series converge as $N \to \infty$? What value of N is needed to obtain S to within two decimal places?

(c) Modify your program so that it uses a while loop so that the summation continues until the added term to the sum is less than some value ϵ. Run your program for $\epsilon = 10^{-2}$, 10^{-3}, and 10^{-6}.

(d) Instead of using the = operator in the statement

```
sum = sum + 1.0/(m*m);
```

use the equivalent operator:

```
sum += 1.0/(m*m);
```

Check that you obtain the same results. ■

Java provides several shortcut assignment operators that allow you to combine an arithmetic and an assignment operation. Table 2.1 shows the operators that we will use most often.

2.3 ■ GETTING STARTED WITH OBJECT-ORIENTED PROGRAMMING

The first step in making our program more object-oriented is to separate the implementation of the model from the implementation of other programming tasks such as producing output. In general, we will do so by creating two classes. The class that defines the model is shown in Listing 2.2. The `FallingBall` class first declares several (instance) variables and one constant that can be used by any method in the class. To aid reusability, we need to be very careful about the accessibility of these class variables to other classes. For example, if we write `private double dt`, then the value of `dt` would only be available to the methods in `FallingBall`. If we wrote `public double dt`, then `dt` would be available to any class in any package that tried to access it. For our purposes we will use the default package protection, which means that the instance variables can be accessed by classes in the same package.