Exercise 2.8 Multiple constructors

(a) Add a second constructor with the argument `double dt` to `FallingBall`, but make no other changes. Run your program. Nothing changed because you didn't use this new constructor.

(b) Now modify `FallingBallApp` to use the new constructor:

```
// declaration and instantiation
FallingBall ball = new FallingBall(0.01);
```

What statement in `FallingBallApp` can now be removed? Run your program and make sure it works. How can you tell that the new constructor was used?

(c) Show that the number of parameters and their type in the argument list determines which constructor is used in `FallingBall`. For example, show that the statements

```
double tau = 0.01;
// declaration and instantiation
FallingBall ball = new FallingBall(tau);
```

are equivalent to the syntax used in part (b). ■

It is easy to create additional models for other kinds of motion. Cut and paste the code in the `FallingBall` into a new file named SHO.java, and change the code to solve the following two first-order differential equations for a ball attached to a spring:

$$\frac{dx}{dt} = v \tag{2.11a}$$

$$\frac{dv}{dt} = -\frac{k}{m}x, \tag{2.11b}$$

where x is the displacement from equilibrium and k is the spring constant. Note that the new class shown in Listing 2.4 has a structure similar to that of the class shown in Listing 2.2.

Listing 2.4 `SHO` class.

```
package org.opensourcephysics.sip.ch02;
public class SHO {
   double x, v, t;
   double dt;
   double k = 1.0;                  // spring constant
   double omega0 = Math.sqrt(k); // assume unit mass

   public SHO() {                   // constructor
      System.out.println("A new harmonic oscillator object is
           created.");
   }

   public void step() {
      // modified Euler algorithm
      v = v-k*x*dt;
      x = x+v*dt; // note that updated v is used
      t = t+dt;
   }

   public double analyticPosition(double y0, double v0) {
      return y0*Math.cos(omega0*t)+v0/omega0*Math.sin(omega0*t);
   }

   public double analyticVelocity(double y0, double v0) {
      return -y0*omega0*Math.sin(omega0*t)+v0*Math.cos(omega0*t);
   }
}
```


Exercise 2.9 Simple harmonic oscillator

(a) Explain how the implementation of the Euler algorithm in the `step` method of class `SHO` differs from what we did previously.

(b) The general form of the analytical solution of (2.11) can be expressed as

$$y(t) = A\cos\omega_0 t + B\sin\omega_0 t, \tag{2.12}$$

where $\omega_0^2 = k/m$. What is the form of $v(t)$? Show that (2.12) satisfies (2.11) with $A = y(t=0)$ and $B = v(t=0)/\omega_0$. These analytical solutions are used in class `SHO`.

(c) Write a target class called `SHOApp` that creates an `SHO` object and solves (2.11). Start the ball with displacements of $x = 1$, $x = 2$, and $x = 4$. Is the time it takes for the ball to reach $x = 0$ always the same? ■

The methods that we have written so far have been nonstatic methods (except for `main`). As we have seen, these methods cannot be used without first creating or instantiating an object. In contrast, *static* methods can be used directly without first creating an object. A class that is included in the core Java distribution and that we will use often is the `Math` class, which provides many common mathematical methods, including trigonometric, logarithmic, exponential, and rounding operations, and predefined constants. Some examples of the use of the `Math` class include:

```
double theta = Math.PI/4;       // constant pi defined in Math class
double u = Math.sin(theta);     // sine of theta
double v = Math.log(0.1);       // natural logarithm of 0.1
double w = Math.pow(10,0.4);    // 10 to the 0.4 power
double x = Math.atan(3.0);      // inverse tangent
```

Note the use of the dot notation in these statements and the Java convention that constants such as the value of π are written in uppercase letters, that is, `Math.PI`. Exercise 2.10 asks you to read the `Math` class documentation to learn about the methods in the `Math` class. To use these methods we need only to know what mathematical functions they compute; we do not need to know about the details of how the methods are implemented.

Exercise 2.10 The Math class

The documentation for Java is a part of most development environments. It can also be downloaded from `<java.sun.com/docs/>`. Look for API docs and a link to the latest standard edition.

(a) Read the documentation of the `Math` class and describe the difference between the two versions of the arctangent method.

(b) Write a program to verify the output of several of the methods in the `Math` class. ■