## Exercises

*Sections 3.1-3.4*

**1.** Mark true or false and explain:

(a) The name of a class in Java must be the same as the name of its source file (excluding the extension `.java`). _____

(b) The names of classes are case-sensitive. _____

(c) The `import` statement tells the compiler which other classes use this class. _____ ✓

**2.** Mark true or false and explain:

(a) The *FootTest* program consists of three classes. ______ ✓

(b) A Java program can have as many classes as necessary. ______

(c) A Java program is allowed to create only one object of each class. ______

(d) Every class has a method called `main`. ______ ✓

3. Navigate your browser to Sun's Java API (Application Programming Interface) documentation web site (for example, `http://java.sun.com/j2se/1.5.0/docs/api/index.html`) or, if you have the JDK documentation installed on your computer, open the file `<JDK base folder>\docs\api\index.html` (for example, `C:\Program Files\Java\jdk1.5.0_06\docs\api\index.html`).

   (a) Approximately how many different packages are listed in the API spec?

   (b) Find `JFrame` in the list of classes in the left column and click on it. Scroll down the main window to the "Method Summary" section. Approximately how many methods does the `JFrame` class have, including methods inherited from other classes? 3? 12? 25? 300? ✓

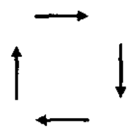
4. Explain the difference between public and private methods.

5. Mark true or false and explain:

   (a) Fields of a class are usually declared `private`. ______
   (b) An object has to be created before it can be used. ______ ✓
   (c) A class may have more than one constructor. ______
   (d) The programmer names objects in his program. ______
   (e) When an object is created, the program always calls its `init` method. ______ ✓

6. Modify the `FootTest` program (`JM\Ch03\FirstSteps\FootTest.java`) to show

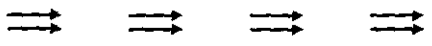
   (a) four feet facing north, spaced horizontally 100 pixels from each other
   (b) four feet facing north, spaced vertically 100 pixels from each other
   (c) four feet aligned along the sides of a square, as follows:



   Each side should be 100 pixels.

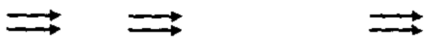
*Sections 3.5-3.7*

**7.■** (a) Using the `FootTest` class as a prototype, create a class `WalkerTest`. Your program should display the same `Walker` in four positions, spaced horizontally by one full "step," facing east:



≤ Hint: the distance of one full step is covered by calls to `firstStep`, `nextStep`, and `stop`. ≥

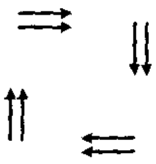
(b) Change `WalkerTest` from Part (a) to show



**8.■** (a) Change `WalkerTest` from Question 7 into `PacerTest`. This program should display four pairs of feet, as in Part (a), but facing west rather than east.

(b) Add the `turnLeft` and `turnRight` methods to the `Pacer` class (`JM\Ch03\FirstSteps\Pacer.java`) ≤ Hint: for a right turn, turn each foot 90 degrees to the right, then move the left foot by `PIXELS_PER_INCH * 8` appropriately sideways and forward. ≥

(c) Change the `PacerTest` class from Part (a) and use the modified `Pacer` class from Part (b) to show four pairs of feet, as follows:



**9.■** Add a third `Walker`, named `cat`, to the `WalkingGroup` class in `JM\Ch03\FirstSteps`. Position `cat` in the middle between `amy` and `ben`. `cat` should "walk" in sync with the other two. Change `cat`'s foot pictures to the ones from the `leftpaw.gif` and `rightpaw.gif` image files (in `JM\Ch03\Exercises`). Run *First Steps* to test `cat`.

**10.■** (a) Using the class `Walker` as a prototype, create a new class `Hopper`. A `Hopper` should move both feet forward together by `stepLength` in `firstStep` and `nextStep` and not move at all in `stop`.

(b) Test your `Hopper` class by making `cat` in Question 9 a `Hopper` rather than a `Walker`.

**11.■** Change the `PacingGroup` class (in `JM\Ch03\FirstSteps`) to make one `Pacer` walk counterclockwise along the perimeter of a square, turning 90 degrees after every few steps. Leave only `amy` in the `PacingGroup` — exclude the other pacers . Use a `Pacer` object with the `turnLeft` and `turnRight` methods, added to `Pacer` in Question 8 (b). ⁅ Hints: initially position `amy` at `x = width/8, y = height*7/8`; allow `amy` to travel in one direction for `danceFloor.getWidth()/2` pixels. ⁆ Repeat the exercise with a `Pacer` walking clockwise.

**12.■** (a) Write a subclass of `Walker` called `Bystander`. `Bystander` should redefine (override) `Walker`'s `firstStep`, `nextStep`, and `stop` methods in such a way that a `Bystander` alternates turning its left foot by 45 degrees left and right on subsequent steps but never moves the right foot. `Bystander` should also redefine the `distanceTraveled` method, to always return 0. ⁅ Hints: (1) To redefine (override) a superclass's method in a subclass, keep its header but change the code inside the braces. (2) Define a new field (for example, `tapsCount`), which will help determine the direction of the left foot's turn in each "step." (3) Do not duplicate the methods inherited from the superclass that remain the same. ⁆

(b) Change a couple of words in the `WalkingGroup` class (in `JM\Ch03\FirstSteps`) to test your `Bystander` class. ⁅ Hint: turn one of the `Walkers` into a `Bystander`. ⁆

**13.**◆ Using the *Banner* applet from Chapter 2 as a prototype (`Banner.java` and `TestBanner.html` in `JM\Ch02\HelloGui`), create and test an applet that shows a spinning foot.

⋮ Hints:

1. Create a new class `SpinningFoot` adapted from `Banner`.
2. Use two fields: `Image pic` and `Foot foot`.
3. In the `init` method, load `pic` from an image file, for example, `leftshoe.gif`. Set up a timer that fires every 30 ms.
4. In the `paint` method, check whether `foot` has been created. If not yet —

```
        if (foot == null)
        {
          ...
        }
```

   — then set `foot` to a new `Foot` object in the middle of the content pane.

5. In the `actionPerformed` method, turn `foot` by 6 degrees.
6. Adapt `SpinningFoot.html` from `TestBanner.html`, changing `Banner.class` to `SpinningFoot.class` in its `<applet>` tag.
7. Add `Foot.java` and `CoordinateSystem.java` to the project. ⋮

**14.**◆ The class `Circle` (`Circle.java` in `JM\Ch03\Exercises`) describes a circle with a given radius. The radius has the type `double`, which is a primitive data type used for representing real numbers. The `CircleTest.java` class in `JM\Ch03\Exercises` is a tiny console application that prompts the user to enter a number for the radius, creates a `Circle` object of that radius, and displays its area by calling the `Circle`'s `getArea` method.

Create a class `Cylinder` with two fields: `Circle base` and `double height`. Is it fair to say that a `Cylinder` HAS-A `Circle`? Provide a constructor that takes two `double` parameters, `r` and `h`, initializes `base` to a new `Circle` with radius `r`, and initializes `height` to `h`. Provide a method `getVolume` that returns the volume of the cylinder (which is equal to the base area times height). Create a simple test program `CylinderTest`, that would prompt the user to enter the radius and height of a cylinder, create a new cylinder with these dimensions, and display its volume.

**15.**◆ Create an application that shows a picture of a coin in the middle of a window and "flips" the coin every two seconds. Your application should consist of two classes: `Coin` and `CoinTest`.

The `Coin` class should have one constructor that takes two parameters of the type `Image`: the heads and tails pictures of the coin. The constructor saves these images in the coin's fields. The `Coin` class should have two methods:

```
// Flips this coin
public void flip()
{
  ...
}
```

and

```
// Draws the appropriate side of the coin
// centered at (x, y)
public void draw(Graphics g, int x, y)
{
  ...
}
```

The `CoinTest` class's constructor should create a `Timer` and a `Coin`. It also should have a `paint` method that paints the coin and an `actionPerformed` method that flips the coin and repaints the window.

⁝ Hints:

1. Use bits and pieces of code from the `Walker` class and from `Banner.java` and `HelloGraphics.java` in `JM\Ch02\HelloGui`, and ideas from Question 16 in Chapter 2.
2. The class `Graphics` has a method that draws an image at a given location. Call it like this:

   ```
   g.drawImage(pic, x, y, null);
   ```

   This method places the upper-left corner of `pic` at $(x, y)$. Explore the documentation for the library class `Image` or look at the `CoordinateSystem` class to find methods that return the width and height of an image.
3. Find copyright-free image files for the two sides of a coin on the Internet.

⁝