

CHAPTER

2

Introduction to Classes and Objects

CHAPTER CONTENTS

- 2.1 Thinking Objectively
- 2.2 Creating Objects
- 2.3 How Methods Work
- 2.4 The Computer Screen
- 2.5 Creating a Rectangle Object
- 2.6 The DrawingKit Class
- 2.7 Creating and Displaying Graphics Objects
- 2.8 Writing to the Console
- 2.9 Summary

The world is made up of objects, and Java programs are no different. In this chapter, you learn how to create objects from existing classes. We use Java 2D classes to create shapes such as rectangles, ellipses, curves, and lines. In subsequent chapters, you learn how to write your own classes.

Early on in this chapter, we will use a class called `DrawingKit` to create a window and display graphics on this window. The `DrawingKit` class has been provided for you to make it easy to display graphics on the screen as you start learning Java and is not a part of Java's standard class libraries. By the time you have completed Chapter 9, on GUI (Graphical User Interface) programming, you will be able to write code to display graphics without needing to use this class. So let's dive in, starting with the basics.

2.1 Thinking Objectively

Java is an *object-oriented* language. An object-oriented language allows a programmer to create and use *objects*. This, naturally, brings up the following question: What are objects?

An object is something that is real or *exists*—it has characteristics called *states*, and things it can do called *behaviors*. Each behavior changes the object's state. If you look around your room, what are some of the things that you see? Your list might include computers, books, cups, lamps, pencils—all of these are *objects*. A ball can have *states* such as its shape, color, and current position. It can have *behavior*, such as bouncing and rolling. A lamp can have certain states such as height, wattage, type, and whether it is lighted. It can possess behaviors such as turning on or off. Figure 2–1 gives more examples of states and behaviors.

In Java, objects exist while the program is running. As the program runs, an object can change state depending on its behavior. In addition, objects can interact with each other to do useful work.

2.2 Creating Objects

A **class** contains information that is needed to create an object. Suppose that you are working on a project. First, you will need to write down the project specifications—this is similar to writing a class. Next, you will use these specifications to build the project. Similarly, the class can be used to

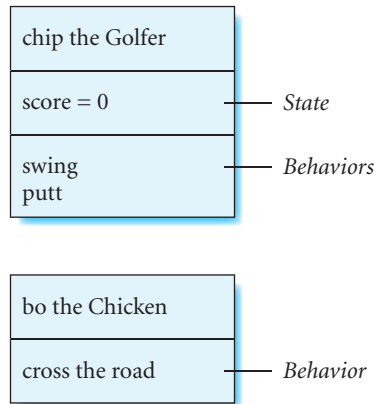


Figure 2-1
State and behaviors of two objects—chip and bo.

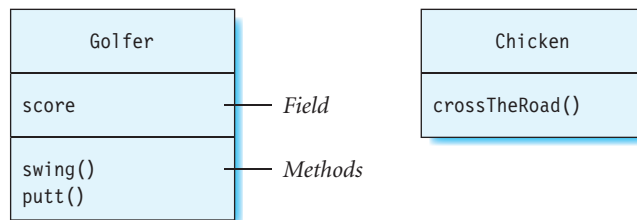


Figure 2-2
Two classes—Golfer and Chicken.

create an object. Figure 2-2 shows two classes: *Golfer* and *Chicken*. Java refers to states and behaviors as **fields** and **methods** respectively. A pair of parentheses () is added at the end of each method name in Figure 2-2 to indicate that this is a method.

There is an important difference between the classes in Figure 2-2 and the objects in Figure 2-1: *A class does not have specific values assigned to its fields, whereas an object does.* Therefore, the object *chip* has a value assigned to its *score* field. Different objects of the same class can have different field values. Figure 2-3 shows several objects created from the *Golfer* class.

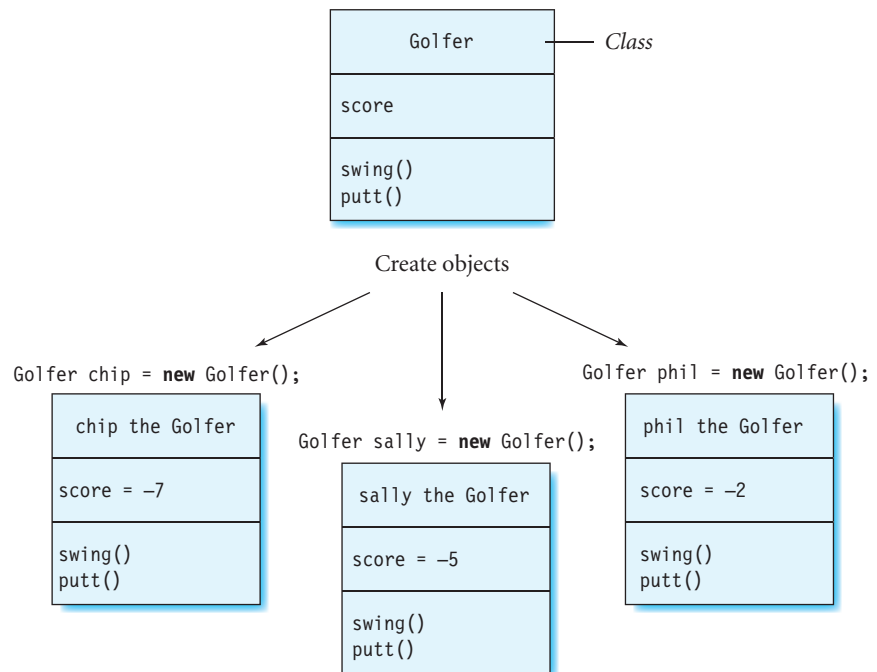
The following statement creates an object (called *chip*) of the *Golfer* class:

```

Class name      Object reference
   ↓            variable      ↓
Golfer chip = new Golfer();

```

Figure 2–3
Creating three objects
(chip, sally, and phil) from
class Golfer.



The keyword `new` is reserved in Java and is used along with a *constructor* to create a new object. The **constructor** is a special method that knows how to create an object. The name of the constructor is the same as the name of the class and is followed by a pair of parentheses. The new object is put in a part of the computer’s memory known as the **heap**.

The **heap** is an area in the computer’s memory that can be used by a program when it executes. A program creates an object and stores it on the heap using the `new` operator. A special program called “garbage collector” runs periodically to remove objects that are no longer needed by the program.

You cannot see the object named *chip* because it is stored inside the computer; however, you can interact with this object by printing some information about it or sending it some data through the console. Later in this chapter, you will learn how to use the `DrawingKit` class to draw graphics objects on the screen so that you can view them. Here, *chip* is called an **object reference variable** (or more simply, a **reference variable**) because it refers to a particular object. An object of a class is also called an **instance** of

that class. Figure 2–3 shows three instances of the *Golfer* class that are assigned to the reference variables *chip*, *sally*, and *phil*. Fields are also known as **instance variables** because each instance keeps a separate copy of its field values.

2.3 How Methods Work

An object calls its methods to do some work. For example, the object *chip* calls its *swing* method, and *bo* calls its *crossTheRoad* method, as shown here:

```
chip.swing();
bo.crossTheRoad();
```

Objects can call their methods as follows:

```
objectName.dotmethodName();
```

The names of the object and the method are separated by a dot. When a method is called, some data might be displayed on the screen, or changes might be made to the object’s fields. For example, after *chip* calls the method *swing*, its *score* field is updated.

Java 2D is a Java toolkit that is used for drawing two-dimensional graphical shapes. It has several classes, a few of which are `Rectangle2D.Float`, `RoundRectangle2D.Float`, `Line2D.Float`, `Ellipse2D.Float`, `BasicStroke`, and `Font`. In the following sections, you will learn how to use these classes to create graphical objects such as rectangles and lines. We start by discussing some basics about how graphics are drawn.

2.4 The Computer Screen

The screen on a computer monitor contains millions of **pixels**. A pixel is a tiny, colored dot that is used to display a point in an image. The pixels are arranged very closely together in the form of a grid. Although a single pixel is hardly visible, groups of these can be combined to form images and text on the computer screen. In Figure 2–4, suppose that each square represents

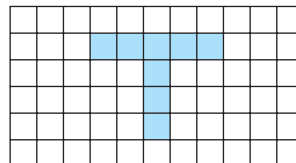


Figure 2–4

An enlarged view of pixels on a computer screen.

a pixel. The squares that are colored blue form the letter “T.” Colored pixels form graphics and text similarly on the screen.

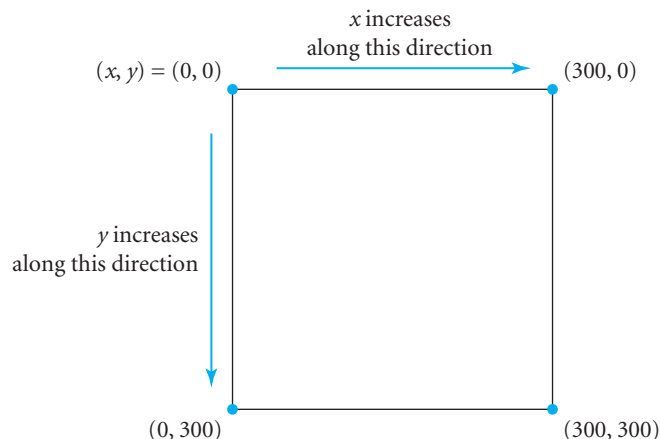
Graphics are drawn inside a *window* on the screen. Later in the chapter, you will learn how to draw a window using the class `DrawingKit`. To draw a graphical shape, you must specify the location and size of this shape inside the window. We discuss window coordinates next.

2.4.1 Window Coordinates

The window uses the Cartesian coordinate system, in which the x - and y -coordinates of points must be specified. A point in the window is represented by its x - and y -coordinates as (x, y) . The top-left corner of the window is the origin $(0, 0)$. The values of the x -coordinates increase from left to right as in a typical graph, whereas the values of the y -coordinates increase from top to bottom. Figure 2–5 represents a window 300 pixels wide and 300 pixels high. The bottom-right corner of this window is the point $(300, 300)$. Using this coordinate system, you can calculate the coordinates of the points in the window where you want to draw the graphics.

This leads to an important question: How large is this window on the screen in inches? This depends on the screen **resolution**, which is the number of pixels in a given area on the screen. For example, 300 pixels will be displayed as 12 inches on a 25 pixels/inch screen, but only 3 inches on a 100 pixels/inch screen. This means that a window that is 500 pixels high and

Figure 2–5
Window coordinates.



500 pixels wide will have a height and width of 5 inches on a 100 pixels/inch screen.

In the examples in this chapter, we draw objects in a square window with a width and height of 500 pixels. Later, you will see how to create a window of a specific size.

Example 1

Find the (x, y) coordinates of the points labeled A , B , and C in Figure 2–6. How many inches long is a line joining the points A and B on a screen with a resolution of 72 pixels/inch?

Solution: Apply basic graph theory. For each point, check its x - and y -coordinates. For example, the x - and y -coordinates of point A are 100 and 200, respectively. Similarly, read the values of the coordinates of points B and C from the grid. The coordinates are A (100, 200), B (100, 400), and C

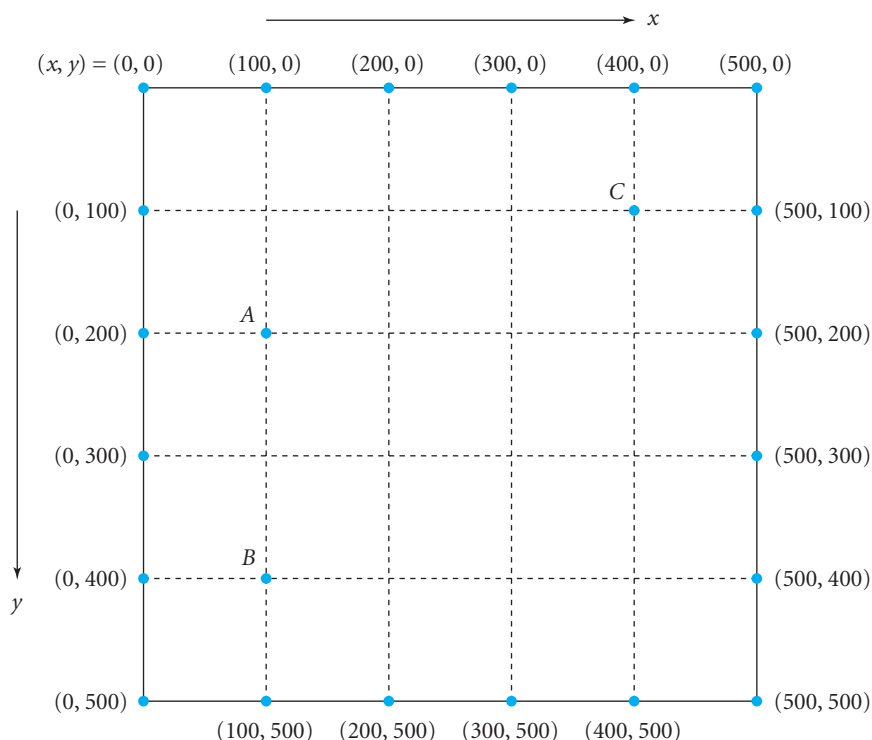


Figure 2–6
Points A , B , and C on a grid.

(400, 100). The line joining *A* and *B* is 200 pixels, and there are 72 pixels in one inch; thus its length is $200/72$, or about 2.78 inches. ■

2.5 Creating a Rectangle Object

Java 2D's `Rectangle2D.Float` class is used to create rectangles. Figure 2–7(a) shows the fields in this class. Figure 2–7(b) shows how an instance can be created using a constructor in this class. (This class also has several methods, which are not shown here.)

The fields *x* and *y* are the (*x*, *y*) coordinates of the rectangle's top-left corner inside the window. For example, to create a rectangle called *rect* with its top-left corner at the point (50, 100), and a width and height of 200 and 300, respectively, we can write:

```
Rectangle2D.Float rect = new Rectangle2D.Float(50, 100, 200, 300);
```

This is similar to how we created the objects named *chip* and *bo*, except that we are also assigning some values to *rect*'s fields when creating it. The `Rectangle2D.Float` constructor takes four numbers as input, called **arguments**. Using arguments, we can create objects with different values for their instance variables. By changing the arguments 50, 100, 200, and 300 of the constructor, you can create a rectangle of a different size at another position in the window.

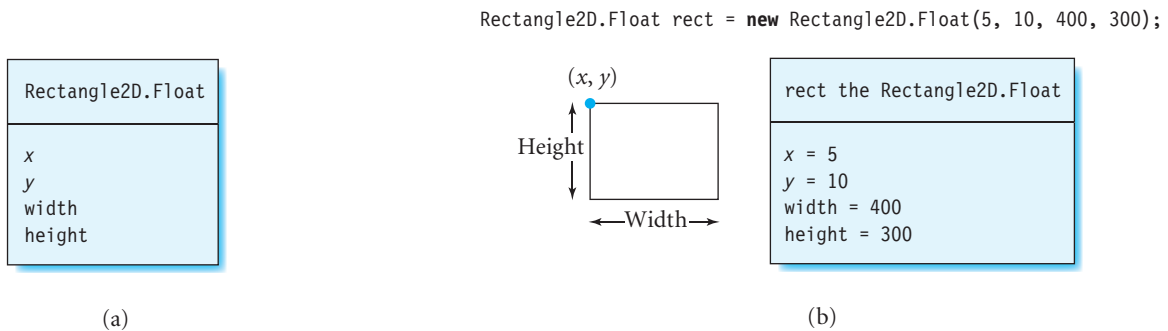


Figure 2–7

(a) Fields in the `Rectangle2D.Float` class, and (b) creating an instance of this class.

The order in which the arguments appear is very important! The first argument is the value of x , the second argument is the value of y , and the remaining two arguments are the values of width and height, in that order. The name of this object is `rect`, but you can choose another name. The names of each of the reference variables in a program should be unique. We discuss the rules and conventions for naming variables in the next chapter.

2.6 The DrawingKit Class

A class called `DrawingKit` is provided for you in the `JavaBook\com\programwith-java\basic` folder on the CD-ROM. Using this class, you can draw a window on the screen and add graphics to it. Internally, `DrawingKit` uses existing classes in Swing and Java 2D to create a window and add graphics to it. (Swing is a Java toolkit that is used to create graphical user interfaces, or GUIs.) You have to know only how to use the methods in the `DrawingKit` class, and not its internal details.

Figure 2–8(a) shows a constructor and some of the methods in `DrawingKit`. Figure 2–8(b) shows a window. You can use an instance of class `DrawingKit` to create a window, and draw and color graphical shapes on this window.

| DrawingKit | |
|---------------------------|-------------------------------|
| <code>DrawingKit()</code> | Constructor creates a window. |
| <code>draw()</code> | Draw an object. |
| <code>setPaint()</code> | Set the color. |
| <code>fill()</code> | Color the object. |
| <code>setStroke()</code> | Set the line thickness. |
| <code>setFont()</code> | Set the font. |
| <code>drawString()</code> | Write text. |

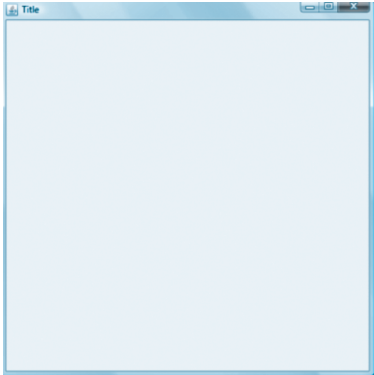
(a)

(b)

Figure 2–8

(a) A constructor and some methods in class `DrawingKit`. **(b)** You can create this window, and then draw and color shapes on it using an instance of class `DrawingKit`.

Note that when you click at any point in this window, the x - and y -coordinates of that point will be displayed on the console.

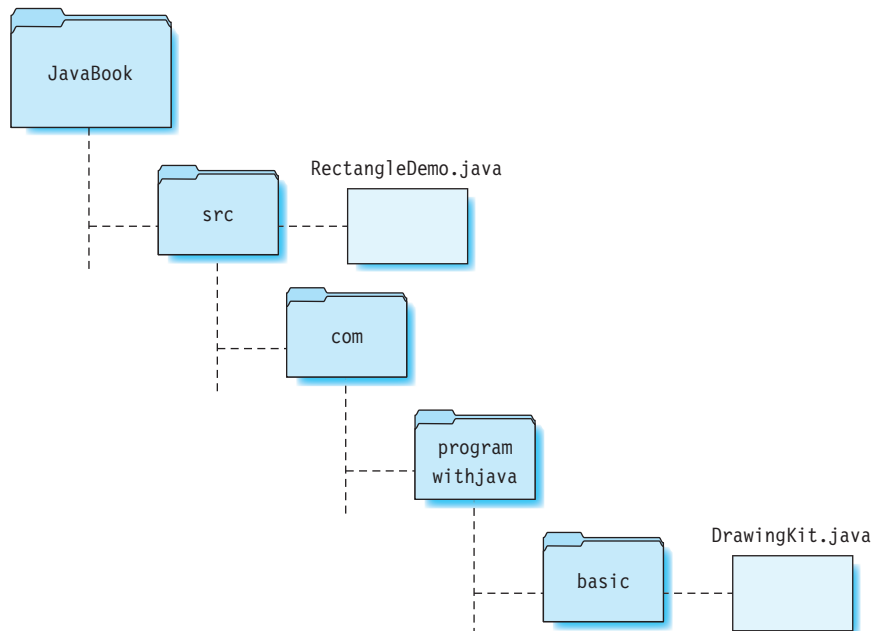
2.6.1 Compiling and Running Your Program with Class `DrawingKit`

To run the programs correctly, you must put the `DrawingKit.java` file in the correct directory. We set up the `JavaBook` directory in the previous chapter to run the `HelloWorld` program. Create a directory named `com` inside `src`, and another directory named `programwithjava` inside `com`, and finally a directory named `basic` inside `programwithjava`. Copy the file `DrawingKit.java` from the CD-ROM into the `JavaBook\src\com\programwithjava\basic` directory in your computer. The directory structure is shown in Figure 2–9.

Copy the program shown in Example 2 into a file called `RectangleDemo.java`. Place this file in the `src` folder. This program uses the `DrawingKit` class in the `DrawingKit.java` file. You should provide the names of both files when you compile the program. On a PC, use:

```
C:\JavaBook> javac -d bin src\com\programwithjava\basic\DrawingKit.java  
src\RectangleDemo.java
```

Figure 2–9
The directory structure to use for your programs.



You will compile it similarly on a Macintosh:

```
computer:~/JavaBook YourName$ javac -d bin
src/com/programwithjava/basic/DrawingKit.java src/RectangleDemo.java
```

If you get any compilation errors, check the directory structure. Also, check that you are running the preceding command in the JavaBook directory. If there are no errors, look inside the `bin` directory. You will see that a `com\programwithjava\basic` directory has been created and that the `DrawingKit.class` file has been placed in it. In addition, the `bin` directory should contain the `RectangleDemo.class` file. To run this program on a PC, type the following into the JavaBook directory:

```
C:\JavaBook> java -classpath bin RectangleDemo
```

Use the same command to run the program on a Macintosh.

It is important to note that the name of the file should match that of the class defined in it. Thus, if the class is called `Foo`, it should be placed in a file named `Foo.java`.

Example 2

This example shows how to draw and display a rectangle with its top-left corner at (50, 100) and a width and height of 200 and 100, respectively. The following code draws a window on the computer screen, and then displays the rectangle inside this window. When you compile and run this program, you will see a rectangle inside the window similar to the one in Figure 2–10.

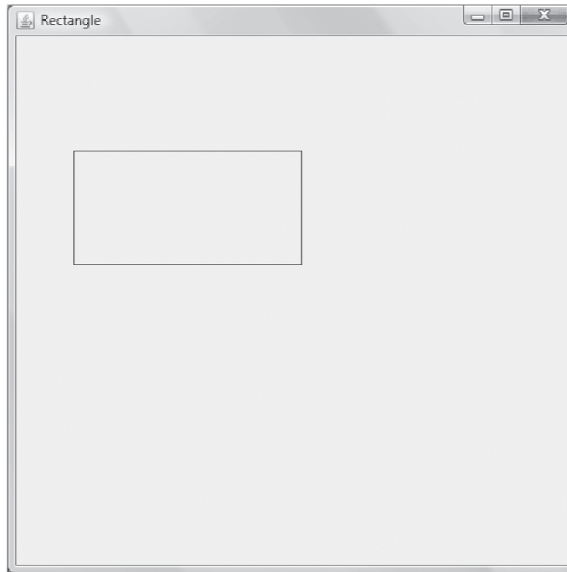
```
import java.awt.*;
import java.awt.geom.*;
import com.programwithjava.basic.DrawingKit;

public class RectangleDemo {
    public static void main(String[] args) {
        DrawingKit dk = new DrawingKit("Rectangle");
        Rectangle2D.Float rect = new Rectangle2D.Float(50, 100, 200, 100);
        dk.draw(rect);
    }
}
```

Congratulations! You have started on an exciting journey to create fantastic artwork. ■

Figure 2–10

Displaying a rectangle in a window.



2.6.2 Nuts and Bolts of the Program

We used several new Java keywords, such as **public**, **static**, and **class**, in Example 2. Rest assured that these are explained in detail in later chapters. For now, a brief explanation of the Java statements used in this program follows.

Java contains many classes that have the necessary information for implementing graphics. These classes are organized into *packages* for convenience. To use a class, you have to specify the package in which it is placed. For example, the `java.awt.geom` package contains the class `Rectangle2D.Float` and many others that are used to create geometrical shapes. The `import` keyword lets you use the classes in this package:

```
import java.awt.geom.*;
```

This imports the `DrawingKit` class from the `com.programwithjava.basic` package:

```
import com.programwithjava.basic.DrawingKit;
```

A commonly used package is `java.awt`, which contains many Java 2D classes (such as `BasicStroke`, `Color`, and `Font`). We will use it in our programs when required.

The next statement declares a class named `RectangleDemo`:

```
public class RectangleDemo
```

You can change the name of the class to another if you wish. Two keywords used here are `public` and `class`. Remember that keywords must be copied *exactly* as shown with all letters in lowercase because Java is case-sensitive.

Every program must contain a method called `main`. The computer starts executing the program from the start of the `main` method. It has to be declared as shown here:

```
public static void main(String[] args) { }
```

A bare-bones Java program that does not perform any actions contains an empty `main` method inside a class, as shown here:

```
public class RectangleDemo {  
    public static void main(String[] args) {  
        // your code goes here  
    }  
}
```

Now let us examine the code inside the example's `main` method. These lines create the various graphical objects such as the window and the graphics inside this window. The following statement creates a window on the computer screen using the `DrawingKit` constructor:

```
DrawingKit dk = new DrawingKit("Rectangle");
```

We use this instance `dk` of class `DrawingKit` to draw and color graphics inside this window. The title on the window is `Rectangle`. You can change this title, and the new title will appear on the window. For example, to give the window a title of `My First Program`, you write:

```
DrawingKit dk = new DrawingKit("My First Program");
```

The next statement creates a rectangle:

```
Rectangle2D.Float rect = new Rectangle2D.Float(50, 100, 200, 100);
```

The last statement draws the rectangle `rect` in the window:

```
dk.draw(rect);
```

Here, the instance `dk` of class `DrawingKit` calls its `draw` method to draw the object in the window. The variable that references the object to be drawn is

used as an argument to the draw method. It should be specified within parentheses, as shown.

2.6.3 Coloring a Rectangle

A shape can be colored by using `DrawingKit`'s `setPaint` and `fill` methods. The object `dk` calls the `setPaint` method with a color as the argument. It then calls the `fill` method with the variable that references the object to be colored as its argument.

Suppose you would like to color the rectangle created in the previous example with the color *red*. To do so, you must add these two lines to the code in the `RectangleDemo.java` file below the `dk.draw` statement:

```
dk.setPaint(Color.red);  
dk.fill(rect);
```

The argument `Color.red` is a constant value defined in Java 2D's `Color` class. After adding these two lines, compile and run the program again. You will see the rectangle is colored red on the screen.

There are many other colors that can be used, such as `white`, `gray`, `lightGray`, `black`, `pink`, `yellow`, `green`, `orange`, `cyan`, `blue`, `darkGray`, and `magenta`. If you want to color the rectangle gray instead, the code becomes:

```
dk.setPaint(Color.gray);  
dk.fill(rect);
```

Try using other colors and see how the rectangle's color changes!

2.7 Creating and Displaying Graphics Objects

The `Rectangle2D.Float` class has been used to draw rectangles with sharp corners. Another useful class is `RoundRectangle2D.Float`. This class is used to draw rectangles with *rounded* corners. Like the `Rectangle2D.Float` class, this class also takes the *x*- and *y*-coordinates, which specify the position of the rectangle's top-left corner, as well as its width and height as arguments. The difference is that `RoundRectangle2D.Float` takes two additional arguments called *corner width* and *corner height*, which specify the width and height of each corner, respectively. Thus, the following line of code creates a rectangle called `rectRounded` positioned at (10, 20) with width 30, height 40, corner width 15, and corner height 25:

```
RoundRectangle2D.Float rectRounded = new RoundRectangle2D.Float(10, 20, 30,  
40, 15, 25);
```

As an exercise, modify the `RectangleDemo` class to create rectangles with different corner widths and heights.

2.7.1 Drawing and Coloring an Ellipse

Next, you learn how to draw an ellipse (oval). The class that can be used to create this shape is called `Ellipse2D.Float`. In this statement, four arguments are passed to the `Ellipse2D.Float` constructor:

```
Ellipse2D.Float myellipse = new Ellipse2D.Float(50, 100, 300, 200);
```

This creates an ellipse called `myellipse` that fits perfectly in a rectangle that has its top-left corner at $(50, 100)$ and has a width of 300 and a height of 200. The first two arguments are the x - and y -coordinates of the rectangle's top-left corner, and the third and fourth arguments are the *width* and *height* of the ellipse, respectively. The relation between the arguments to the `Ellipse2D.Float` constructor and the resulting shape are shown in Figure 2-11.

If the width is set equal to height, the resulting shape is a circle. The ellipse can be drawn on the screen by using `DrawingKit`'s `draw` method:

```
dk.draw(myellipse);
```

Like the rectangle shape drawn earlier, it can be colored by using the `setPaint` and `fill` methods. For example, to color the ellipse magenta, use these two lines:

```
dk.setPaint(Color.magenta);  
dk.fill(myellipse);
```

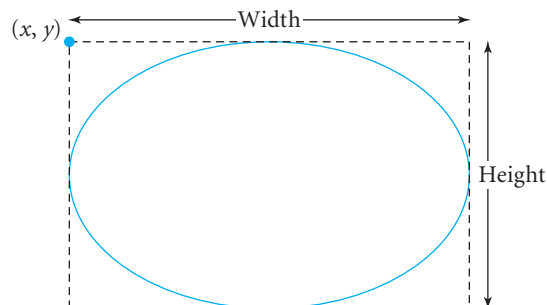


Figure 2-11

Drawing an ellipse using the `Ellipse2D.Float` constructor.

2.7.2 Drawing a Line

A line can be drawn using the `Line2D.Float` class. The x - and y -coordinates of the endpoints must be specified. The following code shows how to use the `Line2D.Float` constructor to create a `Line2D.Float` object called `myLine` that joins the points (10, 20) and (40, 50):

```
Line2D.Float myLine = new Line2D.Float(10, 20, 40, 50);
```

The first two arguments specify the coordinates of the starting point, and the next two arguments give the coordinates of the ending point. To display a line of a particular color (say, magenta), use the `setPaint` and `draw` methods:

```
dk.setPaint(Color.magenta);  
dk.draw(myLine);
```

Attempting to fill a line will display the same result as drawing it.

2.7.3 Changing the Line Thickness

The thickness of a line (and other details) can be set using a class called `BasicStroke`. The only argument to the `BasicStroke` constructor is the width of a line. The following code shows how to create a stroke called `myStroke` with a width of 10:

```
BasicStroke myStroke = new BasicStroke(10);  
dk.setStroke(myStroke);
```

After these statements are executed, graphics objects will be drawn with a line thickness of 10. If the number 10 is changed to a larger value, a thicker line will be drawn.

Example 3

This program shows how to change the line width using the constructor of class `BasicStroke`, and the `setStroke` method of `DrawingKit`. The first ellipse is drawn with a line width of 1.0, which is the default width. The second ellipse has the line width set to 8.0 using `setStroke`.

```
import java.awt.*;  
import java.awt.geom.*;  
import com.programwithjava.basic.DrawingKit;
```



```
public class TwoEllipsesDemo {
    public static void main(String[] args) {
        DrawingKit dk = new DrawingKit("Two Ellipses");

        // create and draw the first ellipse with the default line width
        Ellipse2D.Float one = new Ellipse2D.Float(50, 100, 50, 60);
        dk.draw(one);

        Ellipse2D.Float two = new Ellipse2D.Float(300, 100, 50, 60);

        // change the line thickness to 8
        BasicStroke s2 = new BasicStroke(8);
        dk.setStroke(s2);

        // draw the second ellipse with the new line width
        dk.draw(two);
    }
}
```

The ellipses are shown in Figure 2–12. ■

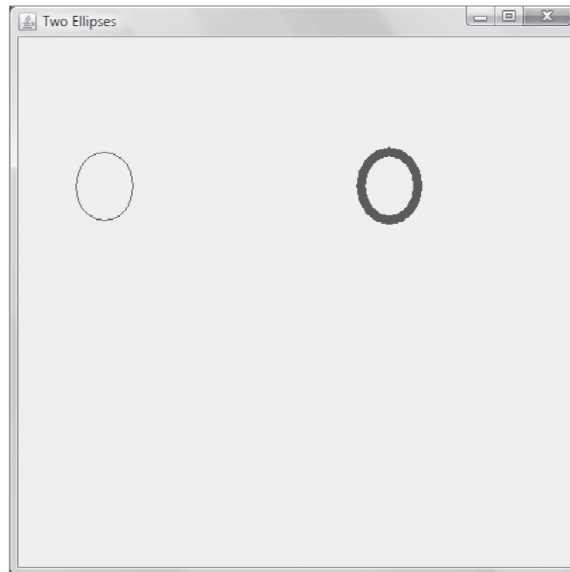


Figure 2–12

Two ellipses drawn using lines of different widths.

2.7.4 Writing Text

Text can be written on the screen using `DrawingKit`'s `drawString` method. The arguments to this method are the text that you want to write and the coordinates of the starting location. For example, to write the word “Hello” starting at the point (50, 100) on the window, use the following statement:

```
dk.drawString("Hello", 50, 100);
```

Note that the word to be displayed is enclosed in double quotes. A group of words that is enclosed in double quotes is called a **string literal**. The name, style, and size of the font used for the string literal should be specified. To do this, create a new font using a constructor in a class called `Font`:

```
Font myFont = new Font("Times New Roman", Font.ITALIC, 12);
```

This creates a new font object of the given name, style, and size. The first argument is the name of a font, such as “Times New Roman”, “Arial”, “Lucida Sans Regular”, “Lucida Sans Bold”, or some other font. The second argument is the style, and it can be `Font.PLAIN`, `Font.ITALIC`, `Font.BOLD`, or `Font.ITALIC | Font.BOLD`. The third argument is the font size. The `setFont` method is called by `dk` to set the font to this new font:

```
dk.setFont(myFont);
```

The following example shows how you can use it in your code.

Example 4

This example shows how to write to the screen. Insert the following code after the last statement in Example 3:

```
// Write the words "Two Ellipses"  
Font myfont = new Font("ARIAL", Font.BOLD, 32);  
dk.setFont(myfont);  
dk.drawString("Two Ellipses", 100, 300);
```

You can also color the ellipses in different colors by adding the following statements:

```
// color ellipse called "one" with magenta  
dk.setPaint(Color.magenta);
```

```
dk.fill(one);

// color ellipse called "two" with yellow
dk.setPaint(Color.yellow);
dk.fill(two);
```

When you run your program, the window shown in Figure 2–13 appears on the screen. ■

2.7.5 Drawing a Curve

A quadratic curve can be drawn using the `QuadCurve2D.Float` class. The `setCurve` method of this class takes the x - and y -coordinates of three points as arguments. These statements draw a curve joining the points (10, 20) and (100, 200) through the point (80, 90):

```
QuadCurve2D.Float curve1 = new QuadCurve2D.Float();
curve1.setCurve(10, 20, 80, 90, 100, 200);
dk.draw(curve1);
```

We use this method in the next example to draw a spaceship.



Figure 2–13

Writing the words “Two Ellipses” in a window.

2.7.6 Drawing an Image Stored in a File

It is easy to draw an image from a file using `DrawingKit`. You must use the `drawPicture` method of this class and specify the name of the file as an argument to this method. For example, the following statement draws the picture in the file `clouds.jpg` on the screen:

```
dk.drawPicture("clouds.jpg");
```

This file should be present in the working directory, where you are compiling your program. The working directory can be different on different computers (for example, `C:\JavaBook` on a PC or `/Users/YourName/JavaBook` on a Macintosh), but Java will pick up the file from the correct directory. This way the program will run correctly on different computers without your having to change the file pathname. We are using the **relative pathname** of the file because it is specified relative to the working directory. For example, if you moved the file into the `C:/JavaBook/image` directory, you would use the relative pathname `image/`.

The next example shows how you can use the `drawPicture` method.

2.7.7 Changing the Window Size

The default window size is 500 by 500. If you want to create a window of a different size (say, 800 by 200), you can do so by specifying it in the following manner using a `DrawingKit` constructor:

```
DrawingKit dk = new DrawingKit("title", 800, 200);
```

The first number (800) is the window's width and the second number (200) is its height.

Example 5

The following program draws a spaceship in a window. The picture stored in the file `space.jpg` will be used for the background. Create a folder called `image` inside the `JavaBook` directory on your computer. Copy the `space.jpg` file from the `JavaBook/image` folder on the CD-ROM into the `JavaBook/image` folder in your computer. We will then draw graphical shapes on this background using the various Java 2D methods we discussed earlier.

```
import java.awt.*;
import java.awt.geom.*;
import com.programwithjava.basic.DrawingKit;

public class SpaceShip {
```

```
public static void main(String[] args) {
    DrawingKit dk = new DrawingKit("Spaceship", 1200, 900);

    // draw the picture
    dk.drawPicture("image/space.jpg");

    // draw the body
    BasicStroke stroke1 = new BasicStroke(2);
    dk.setStroke(stroke1);
    dk.setPaint(Color.red);
    QuadCurve2D.Float curve1 = new QuadCurve2D.Float();
    curve1.setCurve(500, 500, 600, 400, 700, 500);
    dk.draw(curve1);
    QuadCurve2D.Float curve2 = new QuadCurve2D.Float();
    curve2.setCurve(500, 500, 600, 610, 700, 500);
    dk.draw(curve2);
    dk.setPaint(Color.yellow);
    dk.fill(curve1);
    dk.fill(curve2);

    // draw a window
    dk.setPaint(Color.lightGray);
    Ellipse2D.Float curve3 = new Ellipse2D.Float(580, 425, 50, 60);
    dk.fill(curve3);
    dk.setPaint(Color.red);
    dk.draw(curve3);

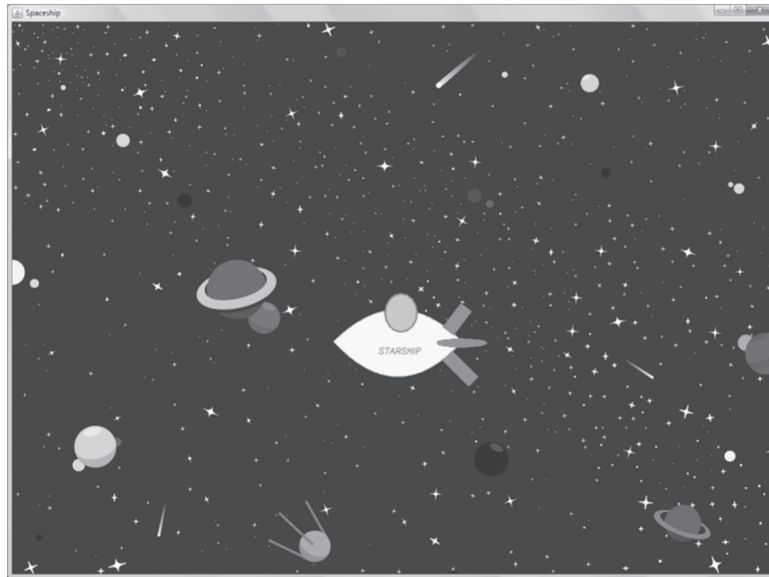
    // draw the wings
    Ellipse2D.Float ellipse1 = new Ellipse2D.Float(660, 495, 80, 15);
    dk.fill(ellipse1);
    BasicStroke stroke2 = new BasicStroke(22);
    dk.setStroke(stroke2);
    Line2D.Float line1 = new Line2D.Float(685, 475, 700, 455);
    dk.draw(line1);
    Line2D.Float line2 = new Line2D.Float(685, 530, 710, 555);
    dk.draw(line2);

    // name the spaceship
    Font font1 = new Font("ARIAL", Font.ITALIC, 13);
    dk.setFont(font1);
    dk.drawString("STARSHIP", 570, 520);
}
}
```

Compile and run the program. This draws the spaceship in a window, as shown in Figure 2–14. ■

Figure 2–14

Drawing a picture and graphical shapes in a window using `DrawingKit`. `space.jpg` © Adrian Neiderhäuser, 123RF.com.



2.8 Writing to the Console

We have only seen examples of graphical programs so far. The next program that we explore does not use graphics; it simply displays text messages on the console. This type of console output is especially useful for debugging a program. To display the string literal “This is text output,” you can write:

```
System.out.print("This is text output");
```

Here, `System.out` refers to the console. This calls the `print` method to display messages on the console. Another way to display messages is to use the `println` method. If there are multiple calls to this method, each message is displayed on a new line.

Example 6

Write a program to display the following two lines of text:

```
Text output is useful for debugging programs.  
Graphics are more fun, though.
```

Solution: Note that only two `println` statements are needed to print out these two lines, and they can be placed directly inside `main`. We do not have to create the window here because no graphics are being drawn. If you

replace the `println` method with `print`, the output will be displayed on a single line. Be sure to place the text within quotes on a single line, because a compiler error will occur otherwise.

```
public class ConsoleOutputDemo {
    public static void main(String[] args) {
        System.out.println("Text output is useful for debugging programs.");
        System.out.println("Graphics are more fun, though.");
    }
}
```

2.9 Summary

This chapter described a subset of Java 2D graphics to create simple graphical shapes and text in different fonts. These shapes are rectangles, lines, ellipses, and curves. You can create a window, and display and color graphical shapes using the `DrawingKit` class. Furthermore, `DrawingKit` also lets you load images from files into a window. We will use this background in Java 2D to explore Java in subsequent chapters, using colorful and fun examples.

Graphics Statements

A summary of the Java statements discussed in this chapter to draw a window and other graphics is provided here for your reference.

Create a window The following statement creates a window with a default width and height of 500:

```
DrawingKit dk = new DrawingKit();
```

The following statement creates a window with the specified string literal set as the title of the window:

```
DrawingKit dk = new DrawingKit("Two Ellipses");
```

The following statement creates a window with the specified title, width, and height:

```
DrawingKit dk = new DrawingKit("Two Ellipses", width, height);
```

Draw a rectangle To draw a rectangle called `rect1` of the given width and height with its top-left corner coordinates at the point (x, y) , use the following statement:

```
Rectangle2D.Float rect1 = new Rectangle2D.Float(x, y, width, height);
dk.draw(rect1);
```

Draw an ellipse The following statements draw an ellipse called `myEllipse` of the specified width and height and positioned at (x, y) :

```
Ellipse2D.Float myEllipse = new Ellipse2D.Float(x, y, width, height);  
dk.draw(myEllipse);
```

Draw a line The following statements draw a line called `myLine`:

```
Line2D.Float myLine = new Line2D.Float(x1, y1, x2, y2);  
dk.draw(myLine);
```

The first two arguments $(x1, y1)$ specify the coordinates of the starting point, and the next two arguments $(x2, y2)$ give the coordinates of the ending point.

Draw a rounded rectangle To draw a rectangle with rounded corners called `rectRounded` having the given width and height, with its top-left corner coordinates at the point (x, y) and the specified corner width and height, use the following statement:

```
RoundRectangle2D.Float rectRounded = new RoundRectangle2D.Float(x, y,  
width, height, cornerWidth, cornerHeight);
```

Color a shape To color a Java 2D graphics object called `shape1` a given color (say, red), use the following statements:

```
dk.setPaint(Color.red);  
dk.fill(shape1);
```

Write text Write the string literal “Hello there” starting at the position (x, y) with the following statements:

```
dk.drawString("Hello there", x, y);
```

Change font To change the font to Arial with style `Bold` and size 12, use the following statements:

```
Font myfont = new Font("ARIAL", Font.BOLD, 12);  
dk.setFont(myfont);
```

Change line thickness The following statements change the thickness of the lines used to draw shapes:

```
BasicStroke myStroke = new BasicStroke(thickness);  
dk.setStroke(myStroke);
```


Draw an image The following statement draws an image from the specified file on the window:

```
dk.drawPicture("space.jpg");
```

Exercises

1. Explain briefly:
 - a. What is a class?
 - b. How can you create an object of a class?
 - c. What is a reference variable?
 - d. How is a class different from an object?
 - e. What is an instance of a class?
 - f. What is a method?
 - g. What is a constructor?
 - h. What is a package?
 - i. What is the `main` method?
2. Write a program that uses the `System.out.println` method to print out your first and last names on different lines.
3. Write a program that uses the `System.out.print` method to print out the current date and time.
4. Fix the errors in the following program so that it compiles and runs correctly:

```
public class ProgramWithErrors {  
    public void main(String[] args) {  
        System.out.println("Program runs correctly")  
    }  
}
```

5. Fix the errors in the following program so that it compiles and runs correctly:

```
public class AnotherProgramWithErrors {  
    public static void main(String args) {  
        System.out.println("Program runs correctly");  
    }  
}
```

Graphics Problems

6. Give the statements needed to create an object of the following classes:
 - a. `Rectangle2D.Float`
 - b. `Ellipse2D.Float`
 - c. `Line2D.Float`
 - d. `QuadCurve2D.Float`
7. What do the following methods do?
 - a. `draw` method of `DrawingKit`
 - b. `fill` method of `DrawingKit`
 - c. `drawPicture` method of `DrawingKit`
 - d. `setCurve` method of `QuadCurve2D.Float`
8. Fix the order of statements in the following program so that it draws a red ellipse on the screen:

```
import java.awt.*;
import java.awt.geom.*;
import com.programwithjava.basic.DrawingKit;

public class EllipseDemo {
    public static void main(String[] args) {
        DrawingKit dk = new DrawingKit("Ellipse");
        dk.draw(rect);
        Ellipse2D.Float rect = new Ellipse2D.Float(50, 100, 200, 100);
        dk.fill(rect);
        dk.setPaint(Color.red);
    }
}
```

9. Fix the compilation errors in this program:

```
import java.awt.*;
import java.awt.geom.*;
import com.programwithjava.basic.DrawingKit;

public class RectangleDemo {
    DrawingKit dk = new DrawingKit("Ellipse");
    Rectangle2D.Float rect = new Rectangle2D.Float(50, 100, 200);
    dk.draw(rect1);
    BasicStroke stroke = new BasicStroke(22);
    dk.setStroke(stroke);
}
```

```
    dk.setPaint(Color = blue);  
    line1 = new Line2D.Float(285, 175, 300, 155);  
    dk.draw(line1);  
}
```

10. Write a program to draw a rectangle inside a window with its top-left corner at (30, 50) and a width and height of 100 and 300, respectively. Color this rectangle yellow.
11. Repeat the previous problem to create the rectangle with rounded corners.
12. Write a program to draw a line inside a window joining the points (0, 0) and (500, 500) with thickness 7. Color this line blue.
13. Write a program to draw an ellipse having width 50 and height 60 that just fits in a rectangle with its top-left corner at (100, 100).
14. Write a program to draw a curve inside a window joining the points (30, 100) and (100, 300), and passing through the point (50, 150).
15. Write a program to write your name inside a window in italics using Arial font of size 15.
16. Write a program to draw a robot using the Java statements discussed in this chapter, and your imagination.
17. Repeat the previous problem to draw a vehicle of your choice.

Further Reading

For more information on Java, the interested reader can consult the references [1–5]. [1] is a set of online Java tutorials. You can find detailed information on other Java 2D classes and advanced graphics techniques in [6].

References

1. “The Java™ Tutorials.” Web.
<<http://download.oracle.com/javase/tutorial/>>.
2. Guzdial, Mark, and Barbara Ericson. *Introduction to Computing and Programming in Java: A Multimedia Approach*. Upper Saddle River, NJ: Pearson Prentice Hall, 2007. Print.

3. Eckel, Bruce. *Thinking in Java*. Upper Saddle River, NJ: Prentice Hall, 2006. Print.
4. Anderson, Julie, and Herve Franceschi. *Java 6 Illuminated: An Active Learning Approach*. Sudbury, MA: Jones and Bartlett, 2008. Print.
5. Sierra, Kathy, and Bert Bates. *Head First Java*. Sebastopol, CA: O'Reilly, 2005. Print.
6. Knudsen, Jonathan. *Java 2D Graphics*. Beijing: O'Reilly, 1999. Print.