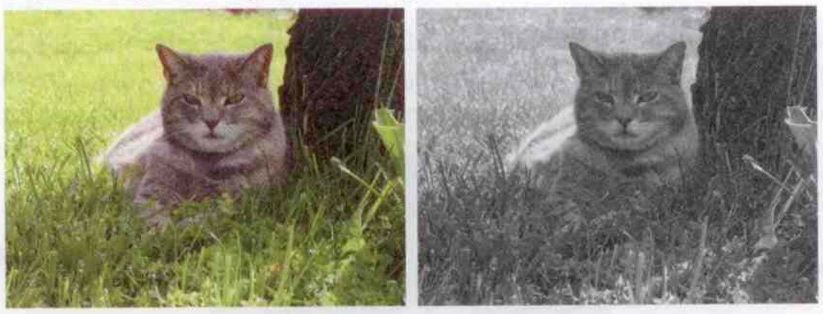
Some of the algorithms for these tasks are surprisingly simple. Consider the task of converting a color image to gray scale (see Figure 5-2).

FIGURE 5-2
A color image and a grayscale image



Grayscale values in the RGB system consist of the 256 distinct combinations of values in which the red, green, and blue components are the same. Thus, the set {(0,0,0), (1,1,1), ..., (255,255,255)} specifies these RGB values. For each pixel in the image, the conversion algorithm must compute a new RGB value that represents the grayscale color corresponding to its current color value. To roughly approximate a grayscale value, one such algorithm computes the average of the pixel's red, green, and blue values and resets each of these values to that average. Here is the pseudocode for this algorithm:

```
For each pixel p in the image
    Set red to p's red value
    Set green to p's green value
    Set blue to p's blue value
    Set average to (red + green + blue) / 3
    Set p's red value to average
    Set p's green value to average
    Set p's blue value to average
```

EXERCISE 5.1

1. Describe the difference between Cartesian coordinates and screen coordinates.

5.2 The `images` Package

To facilitate our discussion of image-processing algorithms, we now present a small package of high-level Java resources. This package, which is named `images`, defines classes that allow the programmer to load an image from a file, view the image in a window, examine and manipulate an image's pixels, update the window with changes, and save the image back to a file. The `images` package is a nonstandard, open source Java toolkit. Installation instructions can be found in Appendix A, but placing the `images` directory with its bytecode files and some sample image files in your current working directory will get you started.

The APImage and Pixel Classes

The two most important classes in the images package are APImage and Pixel. The APImage class represents an image as a two-dimensional grid of Pixel objects. The methods for the APImage class are listed in Table 5-1.

TABLE 5-1
The methods of the APImage class

APImage METHOD	WHAT IT DOES
`APImage()`	Creates an image from a file dialog selection, or creates a blank, 200 by 200, black image if the user cancels the dialog
`APImage(String filename)`	Creates an image from the given file; throws an exception if the file does not exist or the file is not in JPEG format
`APImage(int width, int height)`	Creates a blank image of the given width and height, with a color of black
`int getWidth()`	Returns the image's width in pixels
`int getHeight()`	Returns the image's height in pixels
`Pixel getPixel(int x, int y)`	Returns the pixel at the given position, where x is the column and y is the row
`void setPixel(int x, int y, Pixel p)`	Resets the pixel at the given position, where x is the column and y is the row, to p
`void draw()`	Makes the image's window visible and draws the image in it
`APImage clone()`	Returns a new instance of APImage that is a copy of this image
`String toString()`	Returns the string representation of the image containing the information (filename, width, and height)
`Iterator<Pixel> iterator()`	Returns an iterator on the image, allowing the programmer to visit its pixels with a for loop
`boolean save()`	Saves the image to its current file, or runs saveAs if the file does not yet exist; returns true if the file is saved or false otherwise
`boolean saveAs()`	Saves the image using a file dialog to obtain the file; returns true if the file is saved or false otherwise (if the user cancels)

The `Pixel` class represents a pixel. An object of this class contains three integer values that represent the pixel's RGB color components. The methods for the `Pixel` class are listed in Table 5-2.

TABLE 5-2
The methods of the `Pixel` class

`Pixel` METHOD	WHAT IT DOES
`Pixel(int red, int green, int blue)`	Creates a pixel with the given RGB values
`int getRed()`	Returns the pixel's red value
`int getGreen()`	Returns the pixel's green value
`int getBlue()`	Returns the pixel's blue value
`void setRed(int red)`	Resets the pixel's red value to `red`
`void setGreen(int green)`	Resets the pixel's green value to `green`
`void setBlue(int blue)`	Resets the pixel's blue value to `blue`
`Pixel clone()`	Returns a copy of this pixel
`String toString()`	Returns the string representation of the pixel (red, green, and blue values)

The Structure of a Simple Image-Processing Program

Before we discuss some standard image-processing algorithms, let's try out the resources of the `images` package. This version of the `images` package accepts only image files in JPEG or GIF format. For the purposes of this exercise, we also assume that a JPEG image of my cat, Smokey, has been saved in a file named `smokey.jpg` in the current working directory. The following short Java program loads this image from its file and draws it in a window. The resulting image display window is shown in Figure 5-3. Note that the placement of the image in the window might vary with your operating system.

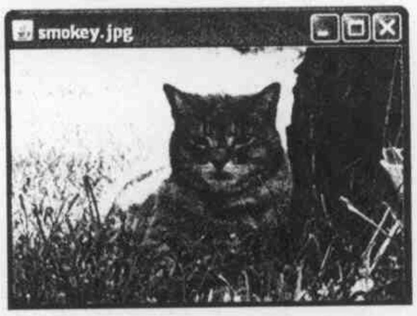
```
// Example 5.1

import images.APImage;

public class TestDraw{

  public static void main(String[]args){
     APImage image = new APImage("smokey.jpg");
     image.draw();
  }
}
```

FIGURE 5-3
An image display window



There are three important statements in our first program. The program first imports the relevant class, `APImage`, from the `images` package, using the following syntax:

```
import <package name>.<class name>;
```

The program then creates an object of the `APImage` class and assigns this object to a variable. This process, called ***object instantiation***, makes a new object available to a program. A program instantiates a class by using one of its constructors with the `new` operator, according to the following syntax:

```
new <class name>(<any parameters>)
```

Note that the constructor name is also the name of the class. The `APImage` class has three different constructors.

After the program saves a reference to the new `APImage` object, it runs the `draw` method on this object to display the image in a window. The syntax for running a method on an object is as follows:

```
<object>.<method name>(<parameters>)
```

Java raises an error if it cannot locate the image filename in the current directory, or if the named file is not in JPEG format. Note that the user quits the program by closing the image window. Every program that we develop in this section will have roughly this same basic structure.

Working with Large Images

Java might raise an error if there is not enough RAM to hold an image. For example, the image of Smokey the cat, with a file size of 120 kilobytes and an image size of 300 by 225 pixels, is reasonably small. But suppose the image of a landscape in the file `spring.jpg` has an image size of 2048 by 1536 pixels and a file size of 668 kilobytes. Attempting to load this image might cause a memory error on many Java platforms. The area of RAM reserved for Java objects is called the *heap space*. To prevent a crash, you can adjust the heap space used for data memory with Java's `Xmx` command-line option, as follows:

```
java -Xmx<integer value>m <main class name>
```