

+ *Worked Example 6.2*

Manipulating the Pixels in an Image

A digital image is made up of *pixels*. Each pixel is a tiny square of a given color. In this Worked Example, we will use a class `Picture` that has methods for loading an image and accessing its pixels (`ch06/image/Picture.java`). The implementation of this class uses the Java image library and is beyond the scope of this book, but here are the relevant parts of the public interface:

```
public class Picture
{
    . . .
    /**
     * Gets the width of this picture.
     * @return the width
     */
    public int getWidth() { . . . }

    /**
     * Gets the height of this picture.
     * @return the height
     */
    public int getHeight() { . . . }

    /**
     * Loads a picture from a given source.
     * @param source the image source. If the source starts
     * with http://, it is a URL, otherwise, a filename.
     */
    public void load(String source) { . . . }

    /**
     * Gets the color of a pixel.
     * @param x the column index (between 0 and getWidth() - 1)
     * @param y the row index (between 0 and getHeight() - 1)
     * @return the color of the pixel at position (x, y)
     */
    public Color getColorAt(int x, int y) { . . . }

    /**
     * Sets the color of a pixel.
     * @param x the column index (between 0 and getWidth() - 1)
     * @param y the row index (between 0 and getHeight() - 1)
     * @param c the color for the pixel at position (x, y)
     */
    public void setColorAt(int x, int y, Color c) { . . . }

    . . .
}
```

Now consider the task of converting an image into its negative: turning white to black, cyan to red, and so on. The result is a negative image of the kind that old-fashioned film cameras used to produce:



The negative of a `Color` object is computed like this:

```
Color original = ...;
Color negative = new Color(255 - original.getRed(),
    255 - original.getGreen(),
    255 - original.getBlue());
```

We want to apply this operation to each pixel in the image.

To process all pixels, we can use one of the following two strategies:

```
For each row
  For each pixel in the row
    Process the pixel.
```

or

```
For each column
  For each pixel in the column
    Process the pixel.
```

Because our pixel class uses x/y coordinates to access a pixel, it turns out to be more natural to use the second strategy. (In Chapter 7, you will encounter two-dimensional arrays that are accessed with row/column coordinates. In that situation, use the first form.)

To traverse each column, the x -coordinate starts at 0. Since there are `pic.getWidth()` columns, we use the loop

```
for (int x = 0; x < pic.getWidth(); x++)
```

Once a column has been fixed, we need to traverse all y -coordinates in that column, starting from 0. There are `pic.getHeight()` rows, so our nested loops are

```
for (int x = 0; x < pic.getWidth(); x++)
    for (int y = 0; y < pic.getHeight(); y++)
    {
        Color original = pic.getColorAt(x, y);
        . . .
    }
```

The following program, `ch06/image/Negative.java`, solves our image manipulation problem:

```
import java.awt.Color;

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

```
Picture pic = new Picture();
pic.load("queen-mary.png");
for (int x = 0; x < pic.getWidth(); x++)
    for (int y = 0; y < pic.getHeight(); y++)
    {
        Color original = pic.getColorAt(x, y);
        Color negative = new Color(255 - original.getRed(),
                                   255 - original.getGreen(),
                                   255 - original.getBlue());
        pic.setColorAt(x, y, negative);
    }
}
```
