Picture Lab: Student Guide

Introduction
In this lab you will be writing methods that modify digital pictures. In writing these methods you will learn how to traverse a two-dimensional array of integers or objects. You will also be introduced to nested loops, binary numbers, interfaces, and inheritance.

Activities
You will be working through a set of activities. These activities will help you learn about how:

- digital pictures are represented on a computer;
- the binary number system is used to represent values;
- to create colors using light;
- Java handles two-dimensional arrays;
- data from a picture is stored; and
- to modify a digital picture.

Set-up
You will need the `pixLab` folder and a Java Development Kit, also known as a JDK (see http://www.oracle.com/technetwork/java/javase/downloads/index.html). A development environment is also useful. DrJava is a free development environment for Java that allows students to try out code in an interactions pane. It also has a debugger, and can be downloaded from http://drjava.org. However, you can use any development environment with this lab. Just open the files in the `classes` folder and compile them. Please note that there are two small pictures in the `classes` folder that need to remain there: `leftArrow.gif` and `rightArrow.gif`. If you copy the Java source files to another folder you must copy these gif files as well.

Keep the `images` folder and the `classes` folder together in the `pixLab` folder. The `FileChooser` expects the images to be in a folder called `images`, at the same level as the `classes` folder. If it does not find the images there it also looks in the same folder as the class files that are executing. If you wish to modify this, change the `FileChooser.java` class to specify the folder where the pictures are stored. For example, if you want to store the images in "`r://student/images/`," change the following line in the method `getMediaDirectory()` in `FileChooser.java`:

```
URL fileURL = new URL(classURL,"../images/");
```

And modify it to

```
URL fileURL = new URL("r://student/images/");
```

Then recompile.

A1: Introduction to digital pictures and color

If you look at an advertisement for a digital camera, it will tell you how many *megapixels* the camera can record. What is a megapixel? A digital camera has sensors that record color at millions of points arranged in rows and columns (Figure 1). Each point is a *pixel* or *picture (abbreviated **pix**) **el**ement*. A *megapixel* is one million pixels. A 16.2 megapixel camera can store the color at over 16 million pixels. That's a lot of pixels! Do you really need all of them? If you are sending a small version of your picture to a friend's phone, then just a few megapixels will be plenty. But, if you are printing a huge poster from a picture or you want to zoom in on part of the picture, then more pixels will give you more detail.

How is the color of a pixel recorded? It can be represented using the RGB (Red, Green, Blue) color model, which stores values for red, green, and blue, each ranging from 0 to 255. You can make yellow by combining red and green. That probably sounds strange, but combining pixels isn't the same as mixing paint to make a color. The computer uses light to display color, not paint. Tilt the bottom of a CD in white light and you will see lots of colors. The CD acts as a prism and lets you see all the colors in white light. The RGB color model sometimes also stores an alpha value as well as the red, green, and blue values. The alpha value indicates how transparent or opaque the color is. A color that is transparent will let you see some of the color beneath it.

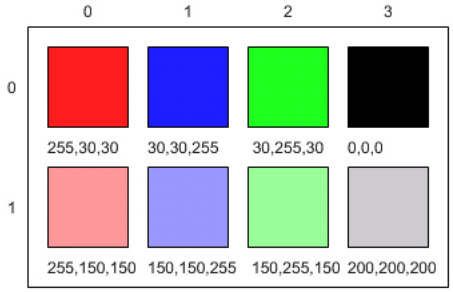

Figure 1: RGB values and the resulting colors displayed in rows and columns

How does the computer represent the values from 0 to 255? A decimal number uses the digits 0 to 9 and powers of 10 to represent values. The decimal number 325 means 5 ones (10^0) plus 2 tens (10^1) plus 3 hundreds (10^2) for a total of three hundred and twenty-five. Computers use *binary numbers*, which use the digits 0 and 1 and powers of 2 to represent values using groups of bits. A *bit* is a **b**inary dig**it,** which can be either 0 or 1. A group of 8 bits is called a *byte*. The binary number 110 means 0 ones (2^0) plus 1 two (2^1) plus 1 four (2^2), for a total of 6.

Questions

1. How many bits does it take to represent the values from 0 to 255?
2. How many bytes does it take to represent a color in the RBG color model?
3. How many pixels are in a picture that is 640 pixels wide and 480 pixels high?

A2: Picking a color

Run the `main` method in `ColorChooser.java`. This will pop up a window (Figure 2) asking you to pick a color. Click on the RGB tab and move the sliders to make different colors.

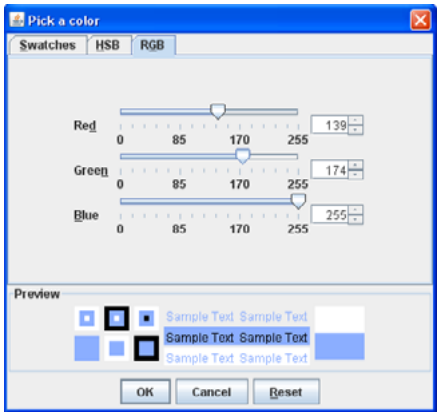

Figure 2: The Color Chooser (This is the version from Java 6.)

When you click the OK button, the red, green, and blue values for the color you picked will be displayed as shown below. The `Color` class has a `toString` method that displays the class name followed by the red, green, and blue values. The `toString` method is automatically called when you print an object.

```
java.awt.Color[r=139,g=174,b=255]
```

Java represents color using the `java.awt.Color` class. This is the *full name* for the `Color` class, which includes the *package* name of `java.awt` followed by a period and then the class name `Color`. Java groups related classes into *packages*. The *awt* stands for Abstract Windowing Toolkit, which is the package that contains the original Graphical User Interface (GUI) classes developed for Java. You can use just the short name for a class, like `Color`, as long as you include an import statement at the beginning of a class source file, as shown below. The `Picture` class contains the following import statement.

```
import java.awt.Color;
```

Use the `ColorChooser` class (run the `main` method) to answer the following questions.

Questions

1. How can you make pink?
2. How can you make yellow?
3. How can you make purple?
4. How can you make white?

5. How can you make dark gray?

A3: Exploring a picture

Run the `main` method in `PictureExplorer.java`. This will load a picture of a beach from a file, make a copy of that picture in memory, and show it in the explorer tool (Figure 3). It makes a copy of the picture to make it easier to explore a picture both before and after any changes. You can use the explorer tool to explore the pixels in a picture. Click any location (pixel) in the picture and it will display the row index, column index, and red, green, and blue values for that location. The location will be highlighted with yellow crosshairs. You can click on the arrow keys or even type in values and hit the enter button to update the display. You can also use the menu to change the zoom level.

Figure 3: The Picture Explorer

Questions

1. What is the row index for the top left corner of the picture?
2. What is the column index for the top left corner of the picture?
3. The width of this picture is 640. What is the right most column index?
4. The height of this picture is 480. What is the bottom most row index?
5. Does the row index increase from left to right or top to bottom?
6. Does the column index increase from left to right or top to bottom?
7. Set the zoom to 500%. Can you see squares of color? This is called *pixelation*. Pixelation means displaying a picture so magnified that the individual pixels look like small squares.

Creating and exploring other pictures

Here is the `main` method in the class `PictureExplorer`. Every class in Java can have a `main` method, and it is where execution starts when you execute the command `java ClassName`.

```
public static void main( String args[])
{
  Picture pix = new Picture("beach.jpg");
  pix.explore();
}
```

The body of the `main` method declares a reference to a `Picture` object named `pix` and sets that variable to refer to a `Picture` object created from the data stored in a JPEG file named "`beach.jpg`" in the `images` folder. A JPEG file is one that follows an international standard for storing picture data using *lossy compression*. *Lossy compression* means that the amount of data that is stored is much smaller than the available data, but the part that is not stored is data we won't miss.

Exercises

1. Modify the `main` method in the `PictureExplorer` class to create and explore a different picture from the `images` folder.
2. Add a picture to the `images` folder and then create and explore that picture in the `main` method. If the picture is very large (for instance, one from a digital camera), you can scale it using the `scale` method in the `Picture` class.
   For example, you can make a new picture ("`smallMyPicture.jpg`" in the `images` folder) one-fourth the size of the original ("`myPicture.jpg`") using:

   ```
   Picture p = new Picture("myPicture.jpg");
   Picture smallP = p.scale(0.25,0.25);
   smallP.write("smallMyPicture.jpg");
   ```