**Image Lab**

Ever wondered how photo editing software modifies images? The Picture Lab will help you answer those questions. At the conclusion of the lab, you will be able to manipulate the colors within an image, flip portions of the image, and create your own effects.

In this lab, you will hone your skills by traversing two-dimensional array objects, as well as practice program analysis, binary numbers, and inheritance.

As you begin working with the Picture Lab, you will see the project is based on a collection of classes. Observe the relationships between the different classes. As you start opening classes, you may notice some provided code is beyond the AP Java subset. Consider this to be "black box" code. You do not need to know how it works; you will simply utilize it. The code you will be asked to create for the lab is to be related to the AP Java subset.

You are not expected to memorize any of the Picture Lab code, but you will need to understand how the code works. Have fun exploring images with Java code!

The Picture Lab is divided over multiple lessons. You will complete the first activities in this lesson. The Java files used are the same throughout, and you'll continue to expand on your work each time. Take time and pay attention to detail as you set up the files. Details matter! It is important to set up your files properly. The Java files will expect the images to be in a particular location.

### Set-up

Follow these steps to set up the Picture Lab files:

* Create a new project called 08.06 Picture Lab in the Mod08 Lessons folder.
* Download the [pixLab.zip](https://lti.flvsgl.com/flvs-cat-content/rl22dk0rtkul6je8icl8kkab7u/flvs-cat-session/apcomputersciencea_v20/module08/lesson06/docs/14_10b/pixLab.zip) file to the newly-created folder.
* Open the **pixLab.zip** file. Move the pixLab folder into the 08.06 Picture Lab folder.
* Within the pixLab folder are two other folders: classes and images. Open the classes folder and launch BlueJ by double clicking the package.bluej file. If needed, move your package.bluej file into the classes folder.
* Compile and run the PictureExplorer class. If an image of a beach appears, everything is set up and ready to go. If not, double check the placement of the images folder.

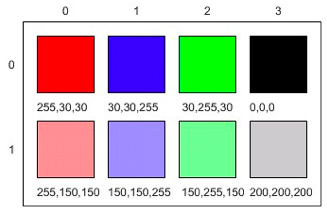
Keep the images folder and the classes folder together in the pixLab folder. The FileChooser class expects the images to be in a folder called images, at the same level as the classes folder. As you work through the project, add your own pictures to the images folder. Please note that there are two small 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.

As you complete the Activities, you'll be asked to record your observation answers to questions.

### Activity 1: 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 DVD in white light and you will see lots of colors. The DVD 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.

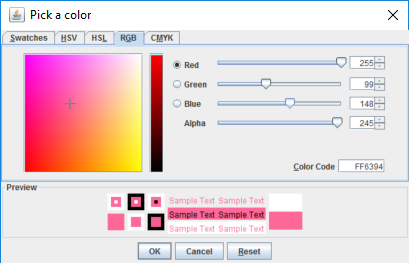
Figure 1: RGB values and the resulting colors displayed in rows and columns

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 (100) plus 2 tens (101) plus 3 hundreds (102) 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 (20) plus 1 two (21) plus 1 four (22), for a total of 6 in decimal. The decimal number 255 is represented as 1111 1111 in binary.

### Activity 2: Picking a color

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

Figure 2: The Color Chooser

When you select 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=255,g=99,b=148]

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.

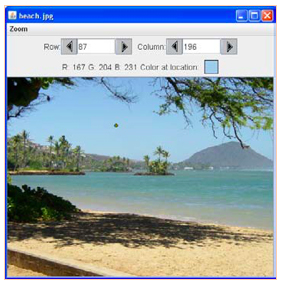
import java.awt.Color;

Open your saved copy of the **08.06 Worksheet** and answer the questions for Activity 2. Use the ColorChooser class (run the main method) to answer the questions.

### Activity 3: 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

Open your saved copy of the **08.06 Worksheet** and answer the questions for Activity 3. Use the PictureExplorer class to answer the questions.

#### Creating and exploring other pictures

Here is the main method in the class PictureExplorer:

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

Complete the exercises using the Java file and then make note of the results in the 08.06 Worksheet.

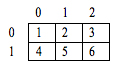
1. Modify the main method in the PictureExplorer class to create and explore a different picture from the images folder. If you use your own picture, place it in the images folder.
2. Scale the image you picked using the scale method in the Picture class.  
   For example, you can make a new picture 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");   
smallP.explore();

Update the **08.06 Worksheet** with the results of the Activity 3 Exercises.

### Activity 4: Two-dimensional arrays in Java

In this activity, you will work with integer data stored in a two-dimensional array. Some programming languages use a one-dimensional (1D) array to represent a two-dimensional (2D) array with the data in either row-major or column-major order. Row-major order in a 1D array means that all the data for the first row is stored before the data for the next row in the 1D array. Column-major order in a 1D array means that all the data for the first column is stored before the data for the next column in the 1D array. The order matters because you need to calculate the position in the 1D array based on the order, the number of rows and columns, and the current column and row numbers (indices). The rows and columns are numbered (indexed) and often that numbering starts at 0 as it does in Java. The top left row has an index of 0 and the top left column has an index of 0. The row number (index) increases from top to bottom, and the column number (index) increases from left to right, as shown below:



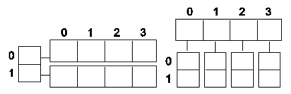
If the above 2D array is stored in a 1D array in row-major order, it would be:

2D array stored in a 1D Array in row-major order. Top row 0, 1, 2, 3, 4, 5. Bottom row: 1, 2, 3, 4, 5, 6

If the above 2D array is stored in a 1D array in column-major order, it would be:

2D array stored in a 2D array in column-major order. Top row 0, 1, 2, 3, 4, 5. Bottom row: 1, 4, 2, 5, 3, 6

Java actually uses arrays of arrays to represent 2D arrays. This means that each element in the outer array is a reference to another array. The data can be in either row-major or column-major order (Figure 4). The AP Computer Science A course specification tells you to assume that all 2D arrays are row-major, which means that the outer array in Java represents the rows and the inner arrays represent the columns.

Figure 4: A row-major 2D array (left) and a column-major 2D array (right)

The following table shows the Java syntax and examples for tasks with 2D arrays. Java supports 2D arrays of primitive and object types.

| **Task** | **Java Styntax** | **Examples** |
| --- | --- | --- |
| Declare a 2D array | type[][] name | int[][] matrix Pixel[][] pixels |
| Create a 2D array | new type[nRows][nCols] | new int[5][8] |
| Access an element | name[row][col] | int value = matrix[3][2]; Pixel pixel = pixels[r][c]; |
| Set the value of an element | name[row][col] = value | matrix[3][2] = 8; pixels[r][c] = aPixel; |
| Get the number of rows | name.length | matrix.length pixels.length |
| Get the number of columns | name[0].length | matrix[0].length pixels[0].length |

To loop through the values in a 2D array, you must have two indexes. One index is used to change the row index, and one is used to change the column index. You can use nested loops, which is one for loop inside of another, to loop through all the values in a 2D array.

Here is a method in the IntArrayWorker class that totals all the values in a 2D array of integers in a private instance variable (field in the class) named matrix. Notice the nested for loop and how it uses matrix.length to get the number of rows and matrix[0].length to get the number of columns. Since matrix[0] returns the inner array in a 2D array, you can use matrix[0].length to get the number of columns.

public int getTotal()  
{  
int total = 0;  
for (int row = 0; row < matrix.length; row++)  
{  
for (int col = 0; col < matrix[0].length; col++)  
{  
total = total + matrix[row][col];  
}  
}  
return total;  
}

Because Java two-dimensional arrays are actually arrays of arrays, you can also get the total using nested for-each loops as shown in getTotalNested below. The outer loop will loop through the outer array (each of the rows), and the inner loop will loop through the inner array (columns in that row). You can use a nested for-each loop whenever you want to loop through all items in a 2D array, and you don't need to know the row index or column index.

public int getTotalNested()  
{  
int total = 0;  
for (int[] rowArray : matrix)  
{  
for (int item : rowArray)  
{  
total = total + item;  
}  
}  
return total;  
}

#### Exercises

Complete the exercises using the Java file and then make note of the results in the 08.06 Worksheet.

1. Write a getCount method in the IntArrayWorker class that returns the count of the number of times a passed integer value is found in the matrix. To test, uncomment the method testGetCount and the call to it in the main method of IntArrayWorkerTester.
2. Write a getLargest method in the IntArrayWorker class that returns the largest value in the matrix. To test, uncomment the method testGetLargest and the call to it in the main method of IntArrayWorkerTester.
3. Write a getColTotal method in the IntArrayWorker class that returns the total of all integers in a specified column. To test, uncomment the method testGetColTotal and the call to it in the main method of IntArrayWorkerTester.