Pictures and Arrays

Understand and use an array

Understand that picture files are made up of pixels and how pixels are represented

As described in the last module, images are basically one big list of color and location information, and can be easily manipulated in code. The learning goals for this material focus on the programmatic structures used to represent pictures.

Digital Pictures

Digital pictures are represented by pixels (or picture elements). These pixels tell the computer how to display an image by providing color information at specific locations. Pixels store color information in red, green and blue components, and any color can be obtained by varying the mixture of these three colors.

pixel
: (definition) A pixel (or picture element) is a single point in an image and represents the smallest element of an image that can be manipulated. Think of them as the atoms making up the image.

An image may be made up of thousands or even millions of pixels. Computers can handle this because they are very fast and can store many pixels. New technical advances frequently lead to increases in the amount of storage and speed of computers. However, the total amount of media online is growing at a fast pace and making pictures smaller can make them faster to download and easier to store. As a result, digital pictures are commonly compressed when they are stored. For example, the JPEG format is an international standard with lossy compression that allows file sizes of the images to be very small. This makes JPEG a popular file format for distributing images over the Internet. Other formats for representing digital images include GIF (Graphics Interchange Format), BMP (Bitmaps), and PNG (Portable Network Graphics).

JPEG
: (definition) JPEG (Joint Photographic Expert Group) is a commonly used format for encoding digital images. JPEG files can be adjusted for either quality or space saving.

PNG
: (definition) PNG (Portable Network Graphics) is another popular image format for storing images using lossless compression, meaning the image information is not lost. Usually PNG is used in situations in which we care less about image size, or our image contains lots of the same color information.

Lossy Compression
: (definition) Lossy compression is a method to make image files smaller by losing some data. The data that is "lost" is not that important, and an image can still be displayed with reasonable quality even when its file size has been greatly reduced. In comparison with a Bitmap file, where the information of every pixel is stored, JPEG files of similar dimensions regularly have less than half the file size of a Bitmap.

Lossless Compression
: (definition) Lossless compression is a way of storing information in less space without losing information. The PNG and GIF formats use forms of lossless compression. You may also be familiar with formats like ZIP which use lossless compression to store files in less space. On images that tend to contain solid or repeating colors, lossless compression can reduce the size of an image significantly without losing image quality. Images containing text or solid color logos, for example, could have their file size reduced significantly using lossless compression like that used in PNG and GIF formats, often to less than 10% of their original size.

Pictures are made up of lots of pixels, and in order to manipulate these pictures, we need some way of referring to each one. We could name each pixel with a number, and refer to them like so: pixel1, pixel2, pixel3, ...

That kind of naming convention works if there are a small number of pixels, but most images today are at least 640 x 480 pixels , which equates to 307,200 pixels. That's a lot of pixels! How then, do we deal with such a large number of pixels (or data) that are of the same type? We can use an array, which is explained in the next section.

Arrays

What is an array? It's basically storage for a sequence of items of the same type. An array can hold items of a primitive type (such as int) or an object type (such as a Pixel) as long as they are all the same. If you want to store a mixture of pixels, numbers, strings, etc, then you cannot store them in a single array, and have to use other means.

Just as a variable's value is stored in something analogous to a mailbox in memory, an array can be thought of as a contiguous line of mailboxes in memory. You can put one item in each mailbox.

So when you create an array, you can store it in a variable, and that variable can point at the whole sequence of mailboxes. But how do you get at the value in a specific mailbox? In java, the syntax for retrieving a value from an array is *array[index]*.

The *index* tells java which array element to retrieve. The first entry in an array is indexed by the number 0, the second by number 1, and so on. Counting from 0 takes some getting used to, but you'll get the hang of it after a while.

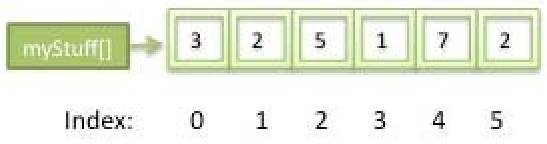
did I get this?

What are some other things you can think of that are similar to arrays of pixels?

Answer: A movie might be represented as an array of frames (pictures that can be played one after another very quickly to create the impression of motion). A piece of music might be represented by an array of notes. Arrays have lots of uses in storing media.

Array Example

Suppose you have an array of numbers called myStuff[]. As depicted below, each number is stored in a separate box. Each box has an index value, with the first box indexed by 0, the second by 1, and so on. The variable int myStuff[] refers to the whole array. myStuff[0] will return 3 when it is evaluated, myStuff[1] will return 2, and so on.



Arrays know how long they are, through the use of their length field. For example, myStuff.length will return 6 (since there are 6 boxes in the image above). Knowing how much data you have in an array is a very important step in manipulating images and working with arrays in general.

Observation...

As we just described, arrays are indexed from 0. For example, the index for the last value in myStuff is 5. However, the length of an array is counted from 1. For example, myStuff has a length of 6.

This series of exercises will help you to test your understanding of how arrays work.

learn by doing

Suppose we have an array (called animals) with the following items, in the given order:
[dog] [cat] [bird] [elephant] [giraffe]
What would animals.length return?
What is the index of 'bird'?
What value can be found at animals[4]?

Our solution
What would animals.length return?
The correct answer is 5. Think about the meaning of the length field of an array.
What is the index of 'bird'?
The correct answer is 2. Remember, array indexes start from 0.
What value can be found at animals[4]?
The correct answer is giraffe. Remember that array indexes start at 0

did I get this?
Suppose you have an array called myNumbers that contain [2][3][1][5].
What is the index of the last number in the array?
What is the length of the array (what would the result of a call to myNumbers.length be)?

Our solution:
What is the index of the last number in the array?
Remember that the index of the first element of the array is 0, not 1. Count up from there. Therefore, it is 3.

What is the length of the array (what would the result of a call to myNumbers.length be)?
since myNumbers has 4 elements in it, the length of myNumbers is 4. To figure out the length of an array, simply count the number of elements in it. For length, you don't count from 0, you count from 1.

Observation...
Since arrays are indexed from 0, but their length is calculated counting from 1, you need to take this difference into account when using length to index into an array. If you have an array myStuff but do not know its length, you can use myStuff.length to index the last value of the array. However, you need to subtract one, as follows: myStuff[myStuff.length - 1]. If you do not subtract one, myStuff[myStuff.length] will throw an error.

To learn more about arrays, visit the Java tutorial on arrays (docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html)

Common Array Pitfalls

Here are a few common errors to watch for when working with arrays.

1. *Zero-based indexing*. Remember, arrays use zero-based indexing, which means that *the first element in an array is at index 0.*
2. *Index out of bounds error*. If you try to get an index of an array that is either negative or larger than the array size, you will get an IndexOutOfBounds error that looks like this: java.lang.ArrayIndexOutOfBoundsException: [line number] . The number after the colon will give you the line number that your error is at. To fix this, check your code to make sure you aren't trying to access indeces of the array that are larger than the array size.
3. *Null pointer exception*. If you forget to initialize your array but try to access indices in it, you will get a null pointer exception that looks like this: java.lang.NullPointerException at [method name]([file_name]:[line_number]). The exception will tell you the method name, file name, and line number that caused this null pointer exception. This error means that a variable hasn't yet been initialized before it is being accessed, it has nothing to point to (null means 'nothing' in programming terms). To fix this, make sure you've initialized your array (i.e. int[] array = new int[4]; before using it.

Manipulating Images

To manipulate a picture, we need to manipulate the pixels that make up the image. For example, we might want to change the color of the image. To do that, we need to change the red, green and/or blue values that each pixel contains.

Start by loading in a new Picture, using one of the methods we talked about in the last set of notes.

A Picture object has an array of pixel objects, that it reads in from the picture file. Using the Picture object, we can get the image width using the pictureObj.getWidth() method, and the height using the pictureObj.getHeight() method.

learn by doing

Assume we have a picture object named myPicture. Use System.out.println to display the width and height of the picture.

Our solution

```
System.out.println("Width = " + myPicture.getWidth());
System.out.println("Height = "+ myPicture.getHeight());
```

The picture object can also return an array of pixels representing the image:

```
Pixel[] pixelArray = pictureObj.getPixels();
```

Each pixel is an object (of type Pixel) that has a red, green and blue value, and these values can be obtained using the getRed(), getGreen() and getBlue() methods. Similarly, the values for the colors can be changed using the setRed(v), setGreen(v) and setBlue(v) methods. You can also access a pixels' location using the getX() and getY() methods.

Observation...

It would be impossible to represent a pixel with a primitive type, because it has to contain multiple pieces of information (at a minimum, the red, green, and blue values used to display it in a certain color. This is a good illustration of an important thing that objects are used for -- to contain a number of related variables (properties) and provide functionality (methods).

Example

Let's try getting a pixel from pictureObj to see what color it is. Type Pixel pixel = pixelArray[0]. Now you can find out the red value for this pixel using pixel.getRed().

learn by doing

Retrieve the first pixel in your pixelArray
Pixel pixel = pixelArray[0]
What is the blue value of this pixel?

What is the x location (the location of pixel in terms of column: in this case, the index of the pixel at the array) of this pixel?

What is the x location of the last pixel in your picture's pixelArray? Note: We are not asking about the last pixel in your image (i.e. the bottom right pixel), we are asking about the last pixel in pixelArray. These may or may not be the same thing.

Our solution

What is the blue value of this pixel?

The answer is different for every picture, but if you used pixel.getBlue() you are correct. Remember that getBlue()returns the blue value of a pixel.

What is the x location (the location of pixel in terms of column: in this case, the index of the pixel at the array) of this pixel?

getX()

What is the x location of the last pixel in your picture's pixelArray? Note: We are not asking about the last pixel in your image (i.e. the bottom right pixel), we are asking about the last pixel in pixelArray. These may or may not be the same thing.

You should have used something like pixelArray[pixelArray.length-1].getX(). Remember that the index for the last pixel is at pixelArray.length-1. You can retrieve the last pixel using Pixel lastPixel = pixelArray[pixelArray.length-1]. Once you have the pixel, you can call getX().

learn by doing

Recall that we just created a picture object and retrieved the first pixel from it using the following sequence of commands:

```
Picture pictureObj = new Picture(FileChooser.pickAFile());
Pixel[] pixelArray = pictureObj.getPixels();
```

How can you ask Java to tell you how many pixels are in your image? Which of the following is correct?

- pixelArray.size
- pictureObj.size
- pixelArray.length
- pixelArray.numPixels()
- pictureObj.length

Which index should be used to retrieve the last pixel in pixelArray?

- pixelArray.length-1
- pixelArray.length
- 234,325

Our solution

How can you ask Java to tell you how many pixels are in your image? Which of the following is correct?

pixelArray.length: pixelArray functions just like any other array.

Which index should be used to retrieve the last pixel in pixelArray?

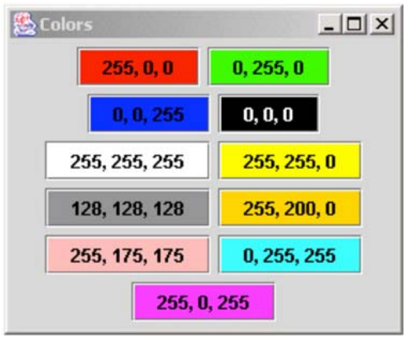
pixelArray.length-1 : Just as with the array of numbers above, pixelArray's length is counted from 1, but the indexes are counted from 0. This means that you will need to subtract one from pixelArray's length to get the correct index for the last pixel in pixelArray.

NOTE: You can find some simple implementations on the above in the Netbeans project MedaTwoEx, in the file MediaTwoEx1.java

Colors

Instead of setting the red, green and blue values individually, the Pixel class allows you to use a Color object to represent the current color values of pixel objects. This Color object is found in the java.awt package. In order to use it, you should: import java.awt.Color; so that java will know about the Color object (more on this a little later).

You can create a Color object by providing the red, green and blue values. Color myColor = new Color(255, 168, 34); This Color object can then be used to set a pixel object's color by doing: pixel.setColor(myColor);. Note that manipulating color and manipulating a pixel are different matters. Recall that a Pixel is a single point in an image. Color is just a color that has a red, blue, and green values. Think of a Pixel as an empty spot that has to be colored and a Color object as a color pencil. We can apply color object to change the color of the pixel, just as coloring an empty spot with a color pencil. If color object is altered (the color of the color pencil has changed) but is not applied to pixel as described above (using the setColor() method), then the pixel itself will not change.



The Color class has many predefined color constants, and this can allow you to save effort when you are trying to use a common color. For example, if you want a Color object representing the color red, you could use Color.RED. You can also do this for some common colors like blue, cyan, magenta, *etc*.

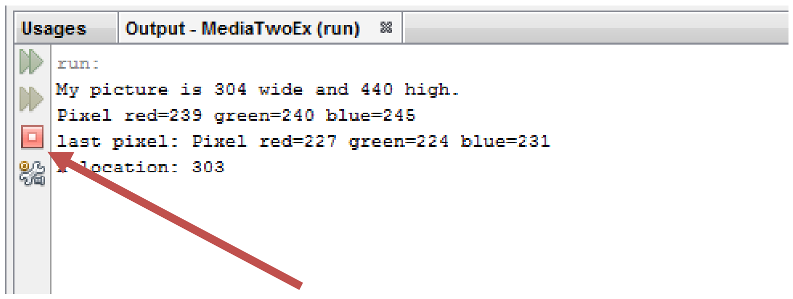
Now, even after you change the color of a pixel, the change won't be visible. In order to see the changes, you have to perform a repaint operation on your Picture object. You do this like so: pictureObj.repaint(). The repaint() method tells Java that your picture has changed, and needs to be re-displayed on the screen.

Example

Let's set the color of the first pixel in your images pixel array to black. Try pixel.setColor(Color.BLACK). Now use pictureObj.repaint() to update your picture. Can you tell which corner turned black? Probably not -- each pixel in most images is so tiny that even knowing that you changed a corner pixel it's hard to see the difference.

There are several ways to select a new color for a pixel. In addition to creating a new color as we did above, you can specify a color using setRed(), setBlue(), and so on.

NOTE: In Netbeans, if the program is still executing in the background because it did not shut down properly you will see a little red square that is light up. Hit it to stop the code execution.



```
run:
My picture is 304 wide and 440 high.
Pixel red=239 green=240 blue=245
last pixel: Pixel red=227 green=224 blue=231
x location: 303
```

did I get this?

Name two different ways you can set the color of a pixel: give an example of each.

Hint: For instance, you can think of using either setRed method or setColor method.

Our Answer

There are 2 ways to set the color of a pixel object.

1. Set each color component individually pixelObj.setRed(255)
2. Use a Color object to set the pixel's color pixelObj.setColor(myColorObj);

The Color class also has methods for making a color object brighter or darker. To make a color brighter, we can use the aptly named brighter() method, like this: colorObj.brighter(). Similarly, to make a color darker, we can use the darker() method, like this: colorObj.darker().

Example: Using brighter() and darker()

Even though you can't easily see how a change to a single pixel affects your image, you can look at the RGB values of a color. For example, now that you've set your pixel to Color.BLACK, you can System.out.println(pixel.getColor()) to see the RGB values for black. Let's pick a different color for the pixel.

```
pixel.setBlue(30);
System.out.println(pixel);
```

Now let's try making that pixel brighter:

```
Color c = pixel.getColor();
c = c.brighter();
System.out.println(c);
```

Now we can make it darker again

```
c = c.darker();
System.out.println(c);
```

You might have expected that if you try to darken a color and then brighten it again, you should get back the same color as the original? Not exactly... The resulting color might be slightly off from the original color. When darkening or brightening a color, the change is calculated in floating point, but *the result is stored in integer form*. Recall how a calculation such as 5/4 (1.25) is stored as 1 when it is converted to an int. Everything after the decimal point is lost. A similar thing happens to the color when it is converted from double to int. This is a loss in precision, since int cannot represent fractional values like double and float can. When we convert back again, it is possible to compound that error, with the result that we end up with a slightly different color than we start with. Another name for this type of error is a rounding error.

rounding error

(definition) Numerical errors that occur when converting from a variable with higher precision (*e.g.,* float, double) to one with lower precision (*e.g.*, int).

Pictures in Memory versus Picture Files

It's important to understand the difference between pictures in memory and a picture file. When you load a picture in Java, we are loading the image into an internal data structure in memory. In other words, we are reading data from a file into memory. Then, we modify our image in memory, and if we want to save our changes, we write them out to the hard disk again either to the original file or to a different file.

As we discussed before, picture files are stored in a particular image format, like PNG, JPEG or GIF. The format is often optimized to take up less space than it would to just write out the entire picture array, using different compression techniques. When we read or write the image in Java, it automatically converts to and from these formats, so we don't have to worry about knowing how they work.

Modifying things directly in memory is faster than modifying things on disk, so usually when we want to change any type of object stored on disk, we will read it in to Java to some easy-to-understand format, like a 2-D array, and then write that data back out to disk when we are done.

In general, everything you write in your program is manipulated in memory. Because of this, you have to read the picture into memory to manipulate it. The picture on the disk changes after the manipulation only if we save (write them out to the hard disk again) the manipulation onto the disk somehow. We will cover this later.

However, it is very important to note that your computer has a *finite* amount of memory. Pictures take up an especially large amount of space, and sometimes you will try to load a picture which is too large for Java to handle. In this case, you will get an out of memory error. Therefore, if you are getting out of memory errors when loading pictures, try using smaller images.

Summary

An array lets you store and access a series of similar things. An array may hold primitive types (such as an ints) or objects (such as Strings).

- Create an array using type [] name = new type[size]. For example you could do int [] nums = new int[4] to create an array of 4 integers, or String [] names = new String[4] to create an array of 4 Strings
- The size of an array can be retrieved using name.length
- Access an array using name[index] where index should be between 0 and name.length-1 (since arrays are indexed from 0)

Pictures are stored on disk in may formats. Some of them, like Bitmaps, are very large, storing information about every pixel in the picture Others use compression, either lossy (such as JPG) or lossless (such as PNG).

A picture is really an *array of pixels* that we can look at and manipulate programmatically. An array lets you store and retrieve values that are all of the same type using an index. In addition to an array of Pixel objects, each Picture has a width and a height. Each Pixel has a color and location.