AP® Computer Science A Picture Lab Student Guide

The AP Program wishes to acknowledge and thank Barbara Ericson of the Georgia Institute of Technology, who developed this lab and the accompanying documentation.

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.

A5: Modifying a picture

Even though digital pictures have millions of pixels, modern computers are so fast that they can process all of them quickly. You will write methods in the `Picture` class that modify digital pictures. The `Picture` class inherits from the `SimplePicture` class and the `SimplePicture` class implements the `DigitalPicture` interface as shown in the Unified Modeling Language (UML) class diagram in Figure 5.

A UML class diagram shows classes and the relationships between the classes. Each class is shown in a box with the class name at the top. The middle area shows attributes (instance or class variables) and the bottom area shows methods. The open triangle points to the class that the connected class inherits from. The straight line links show associations between classes. Association is also called a "has-a" relationship. The numbers at the end of the association links give the number of objects associated with an object at the other end. For example, in Figure 5 it shows that one `Pixel` object has one `Color` object associated with it and that a `Color` object can have zero to many `Pixel` objects associated with it. You may notice that the UML class diagram doesn't look exactly like Java code. UML isn't language specific.

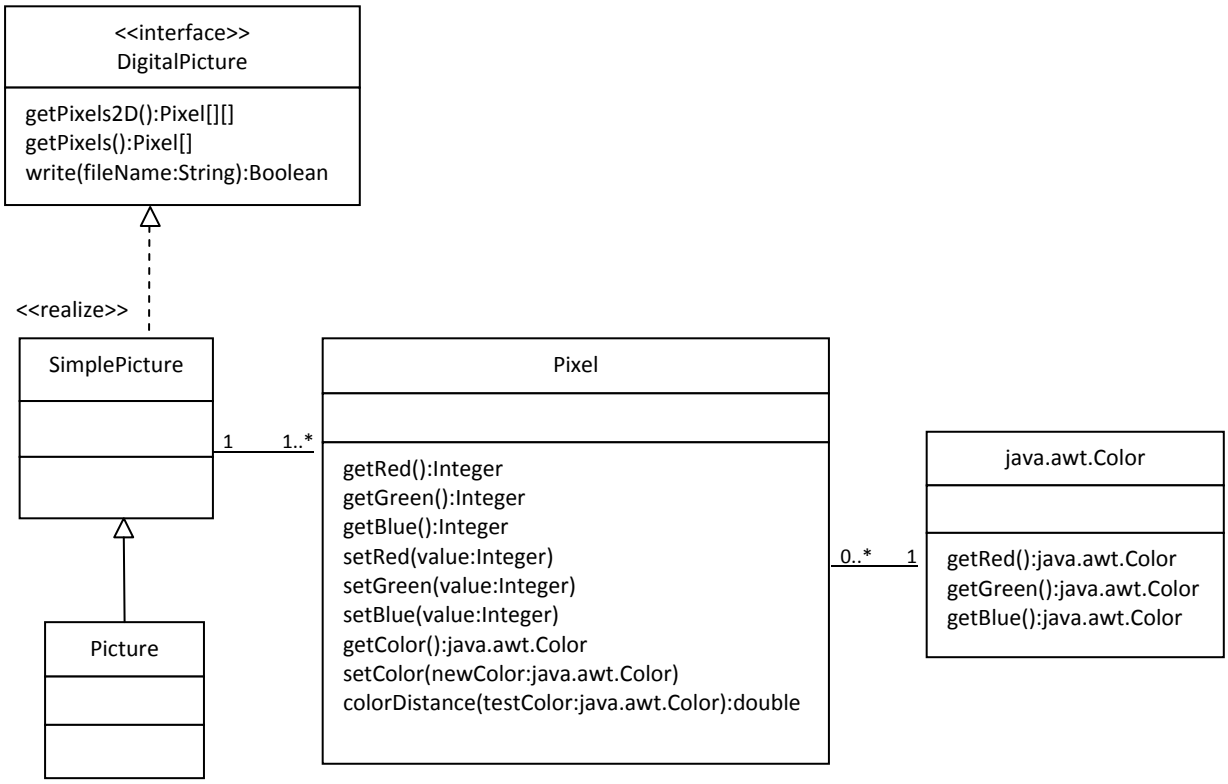

Figure 5: A UML Class Diagram

Questions

1. Open `Picture.java` and look for the method `getPixels2D`. Is it there?
2. Open `SimplePicture.java` and look for the method `getPixels2D`. Is it there?
3. Does the following code compile?
```
DigitalPicture p = new DigitalPicture();
```
4. Assuming that a no-argument constructor exists for `SimplePicture`, would the following code compile?
```
DigitalPicture p = new SimplePicture();
```
5. Assuming that a no-argument constructor exists for `Picture`, does the following code compile?
```
DigitalPicture p = new Picture();
```
6. Assuming that a no-argument constructor exists for `Picture`, does the following code compile?
```
SimplePicture p = new Picture();
```
7. Assuming that a no-argument constructor exists for `SimplePicture`, does the following code compile?
```
Picture p = new SimplePicture();
```

`DigitalPicture` is an *interface*. An *interface* most often only has public abstract methods. An *abstract method* is not allowed to have a body. Notice that none of the methods declared in `DigitalPicture` have a body. If a method can't have a body, what good is it?

Interfaces are useful for separating **what** from **how**. An interface specifies **what** an object of that type needs to be able to do but not **how** it does it. You cannot create an object using an interface type. A class can *implement* (*realize*) an interface as `SimplePicture` does. A non-abstract class provides bodies for all the methods declared in the interface, either directly or through inheritance. You can declare a variable to be of an interface type and then set that variable to refer to an object of any class that implements that interface. For example, Java has a `List` interface that declares the methods that a list should have such as `add`, `remove`, and `get`, etc. But, if you want to create a `List` object you will create an `ArrayList` object. It is recommended that you declare a variable to be of type `List`, not `ArrayList`, as shown below (for a list of names).

```
List<String> nameList = new ArrayList<String>();
```

Why wouldn't you just declare `nameList` to be of the type `ArrayList<String>`? There are other classes in Java that implement the `List` interface. By declaring `nameList` to be of the type `List<String>` instead of `ArrayList<String>`, it is easy to change your mind in the future and use another class that implements the same interface. Interfaces give you some flexibility and reduce the number of changes you might need to make in the future, as long as your code only uses the functionality defined by the interface.

Because `DigitalPicture` declares a `getPixels2D` method that returns a two-dimensional array of `Pixel` objects, `SimplePicture` implements that interface, and `Picture` inherits

from SimplePicture, you can use the getPixels2D method on a Picture object. You can loop through all the Pixel objects in the two-dimensional array to modify the picture. You can get and set the red, green, and/or blue value for a Pixel object. You can also get and/or set the Color value for a Pixel object. You can create a new Color object using a constructor that takes the red, green, and blue values as integers as shown below.

```
Color myColor = new Color(255,30,120);
```

What do you think you will see if you modify the beach picture in the images folder to set all the blue values to zero? Do you think you will still see a beach? Run the main method in the Picture class. The body of the main method will create a Picture object named beach from the "beach.jpg" file, open an explorer on a copy of the picture (in memory), call the method that sets the blue values at all pixels to zero, and then open an explorer on a copy of the resulting picture.

The following code is the main method from the Picture class.

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

Exercises

1. Open PictureTester.java and run its main method. You should get the same results as running the main method in the Picture class. The PictureTester class contains class (static) methods for testing the methods that are in the Picture class.
2. Uncomment the appropriate test method in the main method of PictureTester to test any of the other methods in Picture.java. You can comment out the tests you don't want to run. You can also add new test methods to PictureTester to test any methods you create in the Picture class.

The method zeroBlue in the Picture class gets a two-dimensional array of Pixel objects from the current picture (the picture the method was called on). It then declares a variable that will refer to a Pixel object named pixelObj. It uses a nested for-each loop to loop through all the pixels in the picture. Inside the body of the nested for-each loop it sets the blue value for the current pixel to zero. Note that you cannot change the elements of an array when you use a for-each loop. If, however, the array elements are references to objects that have methods that allow changes, you can change the internal state of objects referenced in the array (pixels).

The following code is the `zeroBlue` method in the `Picture` class.

```
public void zeroBlue()
{
  Pixel[][] pixels = this.getPixels2D();
  for (Pixel[] rowArray : pixels)
  {
    for (Pixel pixelObj : rowArray)
    {
      pixelObj.setBlue(0);
    }
  }
}
```

Exercises

3. Using the `zeroBlue` method as a starting point, write the method `keepOnlyBlue` that will keep only the blue values, that is, it will set the red and green values to zero. Create a class (static) method to test this new method in the class `PictureTester`. Be sure to call the new test method in the `main` method in `PictureTester`.
4. Write the `negate` method to negate all the pixels in a picture. To negate a picture, set the red value to 255 minus the current red value, the green value to 255 minus the current green value and the blue value to 255 minus the current blue value. Create a class (static) method to test this new method in the class `PictureTester`. Be sure to call the new test method in the `main` method in `PictureTester`.
5. Write the `grayscale` method to turn the picture into shades of gray. Set the red, green, and blue values to the average of the current red, green, and blue values (add all three values and divide by 3). Create a class (static) method to test this new method in the class `PictureTester`. Be sure to call the new test method in the `main` method in `PictureTester`.
6. Challenge — Explore the "`water.jpg`" picture in the `images` folder. Write a method `fixUnderwater()` to modify the pixel colors to make the fish easier to see. Create a class (static) method to test this new method in the class `PictureTester`. Be sure to call the new test method in the `main` method in `PictureTester`.