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.

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.

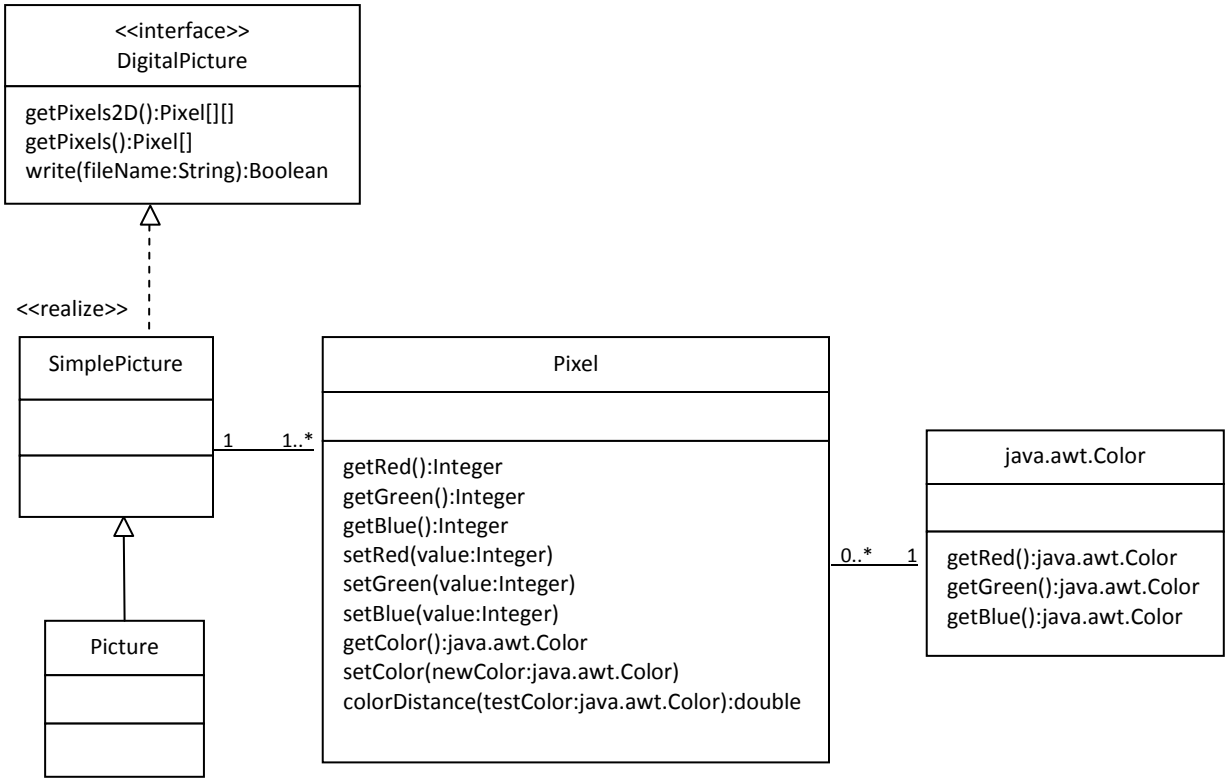

Figure 5: A UML Class Diagram

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`.

   Hint: Water filters out red.

How image processing is related to new scientific breakthroughs

Many of today's important scientific breakthroughs are being made by large, interdisciplinary collaborations of scientists working in geographically widely distributed locations, producing, collecting, and analyzing vast and complex datasets.

One of the computer scientists who works on a large interdisciplinary scientific team is Dr. Cecilia Aragon. She is an associate professor in the Department of Human Centered Design & Engineering and the eScience Institute at the University of Washington, where she directs the Scientific Collaboration and Creativity Lab. Previously, she was a computer scientist in the Computational Research Division at Lawrence Berkeley National Laboratory for six years, after earning her Ph.D. in Computer Science from UC Berkeley in 2004. She earned her B.S. in mathematics from the California Institute of Technology.

Her current research focuses on human-computer interaction (HCI) and computer-supported cooperative work (CSCW) in scientific collaborations, distributed creativity, information visualization, and the visual understanding of very large data sets. She is interested in how social media and new methods of computer-mediated communication are changing scientific practice. She has developed novel visual interfaces for collaborative exploration of very large scientific data sets, and has authored or co-authored many papers in the areas of computer-supported cooperative work, human-computer interaction, visualization, visual analytics, image processing, machine learning, cyberinfrastructure, and astrophysics.

In 2008, she received the Presidential Early Career Award for Scientists and Engineers (PECASE) for her work in collaborative data-intensive science. Her research has been recognized with four Best Paper awards since 2004, and she was named one of the Top 25 Women of 2009 by Hispanic Business Magazine. She was the architect of the Sunfall data visualization and workflow management system for the Nearby Supernova Factory, which helped advance the study of supernovae in order to reduce the statistical uncertainties on key cosmological parameters that categorize dark energy, one of the grand challenges in physics today.

Cecilia Aragon is also one of the most skilled aerobatic pilots flying today. A two-time member of the U.S. Aerobatic Team, she was a medalist at the 1993 U.S. National Championships and the 1994 World Aerobatic Championships, and was the California State Aerobatic Champion.

Glossary

1. Abstract class — You cannot create an object of an abstract class type. But, you can create an object of a subclass of an abstract class (as long as the subclass is not also an abstract class).
2. Abstract method — An abstract method cannot have a method body in the class where the method is declared to be abstract.
3. Algorithm — A step-by-step description of how to solve a problem.
4. AWT — The Abstract Windowing Toolkit. It is the package that contains the original Graphical User Interface (GUI) classes developed for Java.
5. Binary number — A binary number contains only the digits 0 and 1. Each place is a power of 2 starting with 2^0 on the right. The decimal number 6 would be 110 in binary. That would be $0 * 2^0 + 1 * 2^1 + 1 * 2^2 = 6$.
6. Bit — A **b**inary dig**it**, which means that it has a value of either 0 or 1.
7. Byte — A consecutive group of 8 bits.
8. Column-major order — An order for storing two-dimensional array data in a one-dimensional array, so that all the data for the first column is stored before all the data for the second column and so on. In a two-dimensional array represented using an array of arrays (like in Java) this means that the outer array represents the columns and the inner arrays represent the rows.
9. Digital camera — A camera that can take digital pictures.
10. Digital picture — A picture that can be stored on a computer.
11. Inheritance — In Java, a class can specify the parent class from which it inherits instance variables (object fields) and object methods. Even though instance variables may be inherited, if they are declared to be private they cannot be directly accessed using dot notation in the inheriting class. Private methods that are inherited can also not be directly called in an inheriting class.
12. Inner loop — In a nested loop (a loop inside of another loop) the loop that is inside of another loop is considered the inner loop.
13. Interface — A special type of class that can only have public abstract methods in it and/or static constants.
14. 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 that humans would not miss.
15. Media computation — A method of teaching programming by having students write programs that manipulate media: pictures, sounds, text, movies. This approach was developed by Dr. Mark Guzdial at Georgia Tech.
16. Megapixel — One million pixels.
17. Nested loop — One loop inside of another loop.
18. Outer loop — In a nested loop (a loop inside of another loop) the loop that is outside of another loop is considered the outer loop.
19. Package — A package in Java is a group of related classes.
20. Pixel — A picture (abbreviated **pix**) **el**ement.
21. RGB model — Represents color as amounts of red, green, and blue light. It sometimes also includes alpha, which is the amount of transparency.

22. Row-major order — An order for storing two-dimensional array data in a one-dimensional array, so that all the data for the first row is stored before all the data for the second row, and so on. In a two-dimensional array represented using an array of arrays (like in Java) this means that the outer array represents the rows and the inner arrays represent the columns.
23. Subclass — A class that has inherited from another class.
24. Superclass — A class that another class has inherited from.
25. UML —Unified Modeling Language. It is a general purpose modeling language used in object-oriented software development.

References

Dann, W., Cooper, S., & Ericson, B. (2009) *Exploring Wonderland: Java Programming Using Alice and Media Computation*. Englewood, NJ: Prentice-Hall.

Guzdial, M., & Ericson B. (2006) *Introduction to Computing and Programming in Java: A Multimedia Approach*. Englewood, NJ: Prentice-Hall.

Guzdial, M., & Ericson, B. (2009) *Introduction to Computing and Programming in Python: A Multimedia Approach*. (2nd ed.). Englewood, NJ: Prentice-Hall.

Guzdial, M., & Ericson, B. (2010) *Problem Solving with Data Structures using Java: A Multimedia Approach*. Englewood, NJ: Prentice-Hall.

Quick Reference

DigitalPicture Interface

```
Pixel[][] getPixels2D()         // implemented in SimplePicture
void explore()                  // implemented in SimplePicture
boolean write(String fileName)  // implemented in SimplePicture
```

SimplePicture Class (implements Digital Picture)

```
public SimplePicture()
public SimplePicture(int width, int height)
public SimplePicture(SimplePicture copyPicture)
public SimplePicture(String fileName)
public Pixel[][] getPixels2D()
public void explore()
public boolean write(String fileName)
```

Picture Class (extends SimplePicture)

```
public Picture()
public Picture(int height, int width)
public Picture(Picture copyPicture)
public Picture(String fileName)
public Pixel[][] getPixels2D()         // from SimplePicture
public void explore()                  // from SimplePicture
public boolean write(String fileName)  // from SimplePicture
```

Pixel Class

```
public double colorDistance(Color testColor)
public double getAverage()
public int getRed()
public int getGreen()
public int getBlue()
public Color getColor()
public int getRow()
public int getCol()
public void setRed(int value)
public void setGreen(int value)
public void setBlue(int value)
public void setColor(Color newColor)
```

java.awt.Color Class

```
public Color(int r, int g, int b)
public int getRed()
public int getGreen()
public int getBlue()
```