

ITSC 1212 Introduction to Computer Science Lab 9

You can use any lab machine or personal machine that has been setup with Java and DrJava.

OVERVIEW

- 1) Part A: Picture fade out
- 2) Part B: Copy Picture method (assume same dimensions)
- 3) Part C: Method to average two pictures (assume same dimensions)
- 4) Bonus 1: Method to do weighted average of two pictures (same dimensions)
- 5) Bonus 2: Program to morph two pictures (same dimensions)
- 6) Bonus 3: Write a method that continuously morphs between two pictures.

Part A: Picture fade out

- 1) Create a new method in **Picture.java** called `fade()` using the code below, which you don't need to change at all. This method takes no parameters and returns nothing. It must change every pixel in the picture object by decreasing all three components (red, green and blue) by 40%. Here is the code that needs to go in the fade method, note that this code uses a basic **for** loop, instead of a **while** loop, to iterate through the array. The **for** loop has the incrementer built in.

```
public void fade() {
    Pixel[] pixelArray = this.getPixels();
    Pixel pixel = null;
    for (int i = 0; i < pixelArray.length; i++) {
        // get the current pixel
        pixel = pixelArray[i];
        // get, modify, then set the values
        pixel.setRed((int) (pixel.getRed()*0.6));
        pixel.setGreen((int) (pixel.getGreen()*0.6));
        pixel.setBlue((int) (pixel.getBlue()*0.6));
    }
}
```

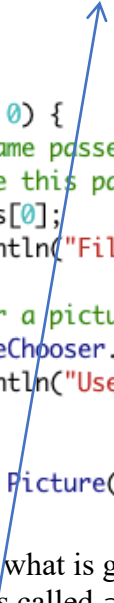
- 2) To test your `fade()` method, download **Lab9PartA.java** from Canvas and compile and run it. It will prompt you to open a picture (you pick the picture), then put the picture in a world, and then call the `fade()` method twice, with a pause in between (the `Thread.sleep(1000)` command tells Java to pause for 1000 milliseconds, which is about a second).
- 3) Lab9PartA.java (and many of the Java lab files from here on out) makes use of *command line parameters* also called *command line arguments*. This is so that these programs can be controlled at run time by providing different values. Without this capability you'd only be able to modify a program by changing the values assigned to variables inside the code then recompiling. The following screenshot shows what you should see at the top of Lab9PartA.java:

ITSC 1212 Introduction to Computer Science Lab 9

```
public static void main(String [] args) throws InterruptedException
{
    String filename;

    if (args.length > 0) {
        // got a filename passed into program as a parameter
        // don't change this part of the code: needed by the Autograder.
        filename = args[0];
        System.out.println("Filename passed in: " + filename);
    } else {
        // ask user for a picture
        filename = FileChooser.pickAFile();
        System.out.println("User picked file: " + filename);
    }

    Picture pic = new Picture(filename);
}
```



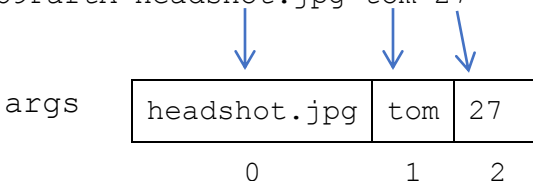
It's important to take note of what is going on here. The main method has one optional parameter, an array of Strings called `args`. These are the arguments that get passed in when the program is run from the command line. For the most part, the programs we've worked with so far haven't taken any parameters, so we've ignored this array of Strings. But `Lab9PartA.java` has the ability to read a string as a command line parameter and use that as the name of a file to open to create a `Picture`.

So far we've usually run programs like this (in the Interactions Pane):

```
> run Lab9PartA
```

But Java recognizes anything that follows the `Lab9PartA` part of the command and stores each separate item in the `args` array. An "item" is any string of contiguous characters separated from the next item by at least one space character. The first item after `Lab9PartA` gets stored in `args[0]`. The next item gets stored in `args[1]`, etc., like this:

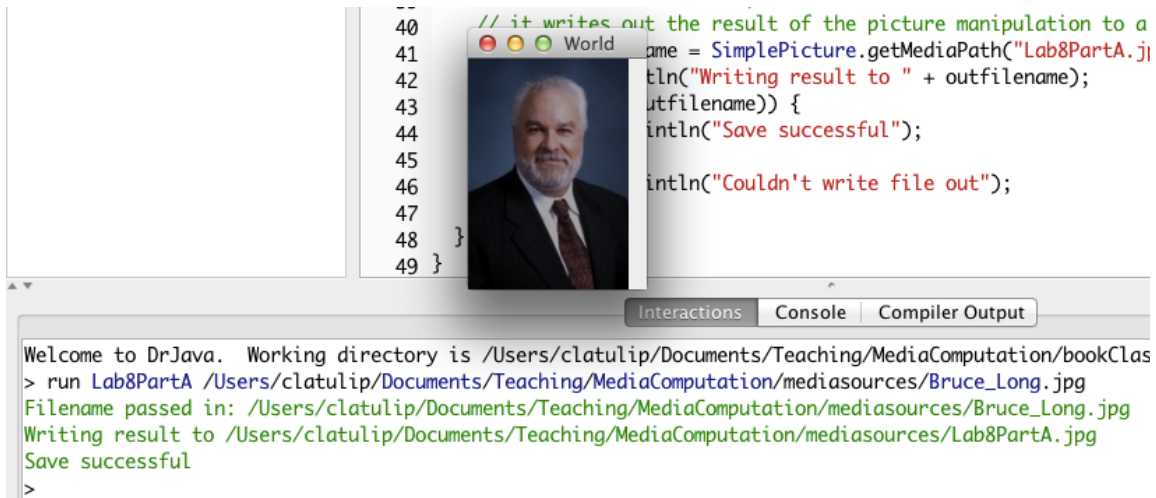
```
> run Lab9PartA headshot.jpg tom 27
```



args	headshot.jpg	tom	27
	0	1	2

Look at the `if` statement at line 8 of `Lab9PartA`: if there was something following `Lab9PartA`, the length of the array (which Java automatically computes for us) will be greater than zero so we store whatever is in `args[0]` in the String variable `filename` then try to open that file. However, if nothing followed `Lab9PartA`, then the `args` array is empty (that is, it has length 0), and so we use the `FileChooser()` method to prompt the user to pick a file. Either way, we get a filename and use it to create a new `Picture` object. The screen shot below shows `Lab9PartA` being run from the Interactions pane, with a filename passed in as a command line parameter. Note that the fully qualified filename is being used.

ITSC 1212 Introduction to Computer Science Lab 9



- 4) If you run Lab9PartA by just clicking the Run button and selecting a file by using the FileChooser, you'll see the fully qualified filename for the file you picked. Copy that filename and try running Lab9PartA.java with that file used as a command line parameter in the Interactions pane, to see how this works. (If you're using a Mac you may have to change back-slashes to slashes to get this to work.)
- 5) Now, alter the **Lab9PartA.java** code. Replace the current update/pause/fade process with a while loop that repeats the fade, update, and pause operations 10 times. Change the Thread.sleep command to wait 500 milliseconds to speed this up a little. This should cause most images to fade almost completely to black. Compile and test your code.
- 6) When you are satisfied that your fade () method is working and that your while loop in Lab9partA.java is working, you are ready to move to Part B. When you test your code make sure to specify the file by using a command line parameter. For this part you just need to submit the updated Lab9PartA.java and Picture.java files.

Part B: Copy Picture method (assume same dimensions)

- 1) Add a method to **Picture.java** that copies the pixels from a source image into the current (target) image. This method should be called copyPixelsFromSource (Picture source) . It should take a Picture object as a parameter and it returns nothing (void). To get started, you will need the following initialization in your method:

```
public void copyPixelsFromSource(Picture source) {
    // create the pixel array for the target image
    Pixel[] targetArray = this.getPixels();
    // create the pixel array for the source image
    Pixel[] sourceArray = source.getPixels();
    // create variables to hold the current pixel's RGB values
    int redValue = 0;
    int greenValue = 0;
    int blueValue = 0;
```

ITSC 1212 Introduction to Computer Science Lab 9

- 2) Now, add the following code immediately after the “public void ...” line. This code tests to make sure that the two pictures are the same size, and exits the method if that is not true. It’s important to test your preconditions before you start any other work in a method. What’s the point of creating arrays of pixels and initializing variables if your initial conditions are in error and the method will fail?

```
// are the two images exactly the same size?
if (this.getWidth() != source.getWidth() || this.getHeight() != source.getHeight()){
    System.out.println("Error! Source image size is not the same as this image.");
    return;
}
```

- 3) Now, add a for loop that one-by-one gets the current pixel from of the `sourceArray` (that points to the picture passed in), and stores the red, green and blue components for that pixel into the red, green, and blue color variables you created.
- 4) Still in the for loop, use those color variables and store their values into the current pixel of the `targetArray`, effectively copying that pixel from the source picture into the target picture.
- 5) Test your `copyPixelsFromSource()` method by downloading **Lab9PartB.java** from Canvas. Compile and run this code. It will prompt you to open one picture, show it, and then prompt you to open a second picture. Note that you **MUST** open two different pictures that have the exact same dimensions. You can test with the **bread1.jpg** and **bread2.jpg** images that are available for download from Canvas. What you should see after you open the second picture is that the world is changed and shows the second picture you opened, because it copied all those pixels into the first picture pixel array. [bread1.jpg and bread2.jpg are pictures of bread that are the exact same picture, but bread2.jpg is flipped upside-down].
- 7) Once you are satisfied that your `copyPixelsFromSource()` method is working correctly, you are ready to move to Part C. When you test your code this time, you have to specify both files by using command line parameters. For this part you just need to submit the updated `Picture.java` file.

Part C: Method to average two pictures (assume same dimensions)

- 1) In this part of the lab, you need to make a method called `averageWithSource(Picture source)` in the `Picture.java` file. This method will be very similar to the method you created in Part B, except that the target picture will be a new picture that is the average of the source picture and the original target picture. To start, copy and paste the `copyPixelsFromSource()` method and rename it to `averageWithSource()`.
- 2) You need to modify the code you have so, for example, the red value you store in a given target pixel is the sum of the original red value from the target picture for that pixel and the red value from the source picture for that pixel, divided by two of course. Also do this for the green and blue values of every pixel.

ITSC 1212 Introduction to Computer Science Lab 9

- 3) When you are finished, test your `averageWithSource()` method by downloading and compiling the **Lab9PartC.java** file. Compile and run this file. Just like in Part B, this will prompt you for two different images (and the two images again must be the same dimensions). If you want to, you can again use `bread1.jpg` and `bread2.jpg` for testing. But this time, what you should see in the world is an image that is a blend of the two images.
- 8) Once you are satisfied your `averageWithSource()` method is working, you are done with the required parts for this lab. As before, when you test your code you have to specify both files by using command line parameters. For this part you just need to submit the updated `Picture.java` file.

Bonus 1. Method to do a weighted average of two pictures

- 1) In this part of the lab, you need to make a method called `weightedAverageWithSource(Picture source, double sourcePercent)`. This method will be very similar to the last one so again, copy, paste and rename to get started.
- 2) What is different about this target is that we are passing in a number (that must be between 0.0 and 1.0 inclusive). We are going to use this to compute a weighted average of the two images, instead of a regular average. A regular average gives you a pixel that is half the value from one image and half the value from the other image. Weighted averages give you a value that is more towards one of the pixels. If you don't know about weighted averages, read the box below.

Weighted Average of Two Values

To calculate the weighted average of two values, you first need the two weights, which must add up to 1, for example: 0.25 and 0.75 or 0.1 and 0.9.

The weighted average is calculated like this for two numbers:

$$\text{weightedAvg} = \text{weight1} * \text{num1} + \text{weight2} * \text{num2}$$

Note that when `weight1` and `weight2` are both 0.5, this turns into:

$$\begin{aligned}\text{weightedAvg} &= 0.5 * \text{num1} + 0.5 * \text{num2} \\ &= 0.5 * (\text{num1} + \text{num2}) \\ &= (\text{num1} + \text{num2}) / 2\end{aligned}$$

which is the formula for normal averaging.

- 3) Make sure that when copying, pasting and renaming to create this method, you add in the second parameter, `sourcePercent`.
- 4) Now, you need to change the three lines of code in the method that calculate the average of the two pixels and store them in the color variables to be weighted averages, using the `sourcePercent` value passed in. The percent passed in should be used for how much of the `sourcePixel` is seen in the resulting picture. (Hint: you will probably want to create a variable that is the other weight, since only one of the weights is passed in as a parameter. So if `sourcePercent` is 0.3, the other weight will be 0.7.) Note that because you are now

ITSC 1212 Introduction to Computer Science Lab 9

multiplying each pixel color value by a percentage, which is a double, you will need to cast the results of the calculation back into an int.

- 5) When you want to test your method, download **Lab9PartD.java** and compile and run it. Just like in Parts B & C, this program will prompt you to open two images and they must be exactly the same dimensions. Again, you can use `bread1.jpg` and `bread2.jpg` for testing.
- 6) Note that the `sourcePercent` parameter passed in to the method is 0.8, which means that the final image is going to look mostly (80%) like the second image you import, and you'll only see a bit (20%) of the first image. Play around with different values of this number to see what happens (but keep it between 0.0 and 1.0).
- 7) Now, add two `if` statements to the `weightedAverageWithSource()` method that check to ensure the `sourcePercent` parameter passed in is in the right range. If it is below 0.0, you should print the following message: "Error! sourcePercent is below 0.0". If the parameter is above 1.0, print the following message: "Error! sourcePercent is above 1.0". In both cases, exit the method without making any changes to the image.
- 8) Test your method by changing the value in `Lab9PartD.java` to something that is outside of the range, and make sure that both tests you just added are working.

Bonus 2. Program to morph two pictures

- 1) Now that you have the `weightedAverageWithSource()` method working, you can use that to program image morphing, where one image slowly turns into a different image. Download **Lab9PartE.java**, compile it and run it. Right now, this program prompts you to open two different images, and then it calls the `weightedAverageWithSource()` method once, with the value 0.25, which results in an image that is $\frac{1}{4}$ the second (source) image and $\frac{3}{4}$ the first (target) image.
- 2) Your job is to change the code so that the image displayed in the world slowly changes over time to the other image. You will need to create a `for` loop and call the `weightedAverageWithSource(...)` method (as well as `modelChanged()` and `Thread.sleep()`) inside the `for` loop. The loop should execute 20 times, once each for each percentage from 0% up to 95%, incrementing by 5% each time. You should use the `Thread.sleep()` call to pause slightly after each change. Make it wait for 100 milliseconds (about $\frac{1}{10}$ th of a second). You will want to use the loop index as a way to change the percentage parameter.
- 3) When you are done, compile and test your method.

Bonus 3. Continuous Morphing

- 1) Copy and rename `Lab9PartE` to **Lab9PartF** then modify it so it continuously morphs back and forth between the two pictures. Remember that in your first pass you're going to replace your target picture with your source picture so when your first pass is done you'll have two copies of the source picture and no target picture to use in the second pass. So what will you

ITSC 1212 Introduction to Computer Science Lab 9

use for the source picture in the second pass? Here's a hint: just create a copy of your target picture and use that.

- 2) You need to create an outer `while` loop that encloses the existing `for` loop with a condition that is always **true**, so that it runs infinitely. And finally, inside this you need to alternate between morphing source and target to morphing your copy and target. It would help to draw a sketch of this process. Have fun! Earn a stamp for your team!

Lab 9 Canvas submission

1. Navigate to the course Canvas page.
2. Click on Module 9
3. In Module 9 locate and click on the Lab 9 assignment and submit the following files
 - a. Picture.java
 - b. Lab9PartA.java
4. If you completed the bonus submit your files and include a comment with your submission indicating which bonuses you completed.

You're done! If you are using the lab computer, remember to logout.

So what did you learn in this lab?

- 1) Using command line parameters to control a program's execution
- 2) How to use the args array
- 3) Practice using a while loop
- 4) Practice pausing to give your program the illusion of animation
- 5) Practice testing preconditions before letting your code continue
- 6) How to systematically change all the pixels in a picture
- 7) How to replace one picture with another of the same size
- 8) How to create a new picture that is the average of two pictures
- 9) How to gradually morph from one picture to another
- 10) How to continuously morph back and forth between two pictures

Terminology and concepts:

- Command line parameters