ITSC 1212 Media Programming Lab 12

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

OVERVIEW

1) Part A: Privacy blackout
2) Part B: Privacy blur
3) Part C: Colorful edge detection
4) Part D: Truth tables
5) Part E: Picture Roulette

Bonus #1: Animate edge detection and blurring

Bonus #2: Calculate truth table values

Part A: Privacy Blackout

1) Create a new method in Picture.java called `blackOut(…)`. This method takes `startX`, `startY`, and `endX`, `endY` integer parameters (in that order) and returns nothing. It must change every pixel in the rectangular region between (`startX, startY`) and `(endX, endY)` to black. The idea behind this method is that you could use it to redact (hide) part of an image that you don't want other people to see. In the image below, we've used the method to black out Bruce's mouth in the Bruce_Warholized.jpg picture below because he was in the Navy and he can swear like crazy.

2) Write this method by constructing a double `for` loop that iterates through <u>all</u> the columns and rows in the image. Use a <u>single</u> compound conditional to check that the current row and column are in the area defined by the parameters. If the current column and row is inside the specified area, change the color of the corresponding pixel to black.
3) To test your `blackOut(…)` method, download **Lab12PartA.java** from Canvas and compile and run it. It will prompt you to open a picture, then put the picture in a world, and then call the `blackOut(…)` method twice for two different areas of the image.
4) What do you think the pre-conditions and post-conditions for this method should be?

   Pre-conditions:

   Post-conditions:

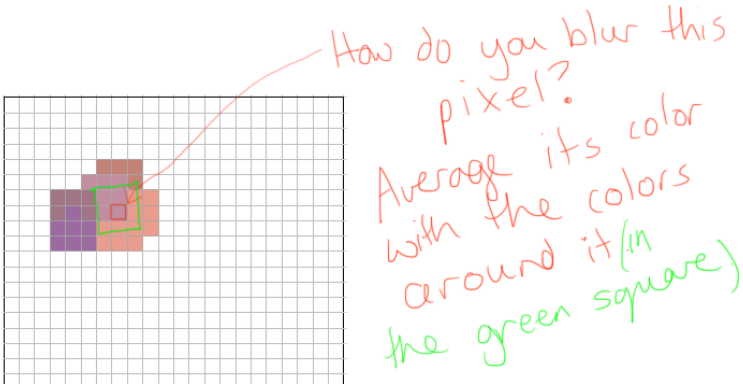
5) Add in appropriate checks for the preconditions at the top of your method. If the preconditions are not met, print out an error message that describes the error, and return out of the method.n
6) When you are satisfied that your `blackOut(…)` method is working, you are ready to move to Part B. For this part you just need to submit the updated Picture.java file.

Part B: Privacy Blur

1) Sometimes blacking out part of a picture just looks too stark. Perhaps you just want to blur out the faces in an image, the way that Google maps does with its street view. Now of course, Google maps uses facial detection algorithms to automatically find the areas to blur. That's a little sophisticated for now. But, assuming you know where the face is, you can write a method that would blur it out, especially if you are given code that will blur a single pixel, which is what we have done. Type the following private method into your **Picture.jav**a file:

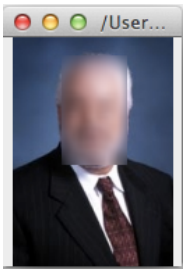
```
private void blurPixel(int x, int y, int size) {
  // blurs the passed in (x,y) pixel by averaging the colors of the surrounding
  // pixels size is how far out to go, so if size is 2, we will blur based on two
  // columns to the left, two to the right, two rows above and two below, in a
  // square

  Pixel tempPixel = null;
  Pixel pixel = null;

  int sumR = 0;
  int sumG = 0;
  int sumB = 0;
  int divisor = 0;

  // iterate over the size x size pixels in this area to add up the RGBs
  for (int subx = x-size; subx < x+size + 1; subx++) {
    for (int suby = y-size; suby < y+size + 1; suby++) {
      // check if this pixel is in the range of this image
      if ( ??????? ){
        // in range, so get this pixel, add it's colors to running average
        tempPixel = this.getPixel(subx, suby);
        sumR += tempPixel.getRed();
        sumG += tempPixel.getGreen();
        sumB += tempPixel.getBlue();
        // incease divisor
        divisor += 1;
      }
    }
  }
  // done adding up all the colors from surrounding pixels so
  // get this pixel and then set it's color to the average RGBs
  // making sure to divide by the divisor (num colors added in)
  pixel = this.getPixel(x,y);
  pixel.setRed((int)(sumR/divisor));
  pixel.setGreen((int)(sumG/divisor));
  pixel.setBlue((int)(sumB/divisor));
}
```

2) The `blurPixel(…)` method above takes a **SINGLE** specified pixel in the current image and averages it with the surrounding pixels. The sketch below depicts this, where we are blurring with a level of 1, meaning 1 row of pixels all the way around the current pixel:

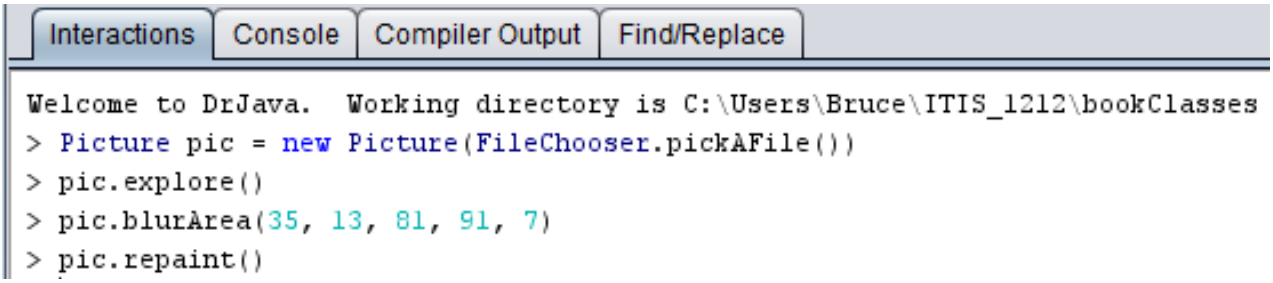


The only thing you have to do is figure out what the conditions in the if statement should be (see the question marks in the `blurPixel` code above – you need to replace them). The if statement should make sure that you aren't trying to average pixels that are outside of the picture. For instance, if you specify a size of 3, this method is going to try to average all the pixels in a 7 x 7 pixel square around the central pixel. In the middle of the picture that will be fine, but if you are trying to calculate the blur for a pixel at the edge or corner of the image, that might be problematic. So, figure out what the conditions need to be for this if statement. Hint: you should have four conditions compounded together.

3) Now, your job is to write a Picture.java method called: `blurArea(int startX, int startY, int endX, int endY, int level)` . This method must iterate through **all** the pixels in the image, and call the `blurPixel(…)` method for pixels that are inside the area specified by the first four parameters. The last parameter is for the level of the blur. This number should be passed into the `blurPixel(…)` method as it determines how many of the surrounding pixels to average in. A number between 1 and 10 is reasonable. Calling the `blurArea(…)` method on the Bruce_Long.jpg picture with a level of 7 blurs his face this much:



4) To test your `blurArea(…)` method, use the Interactions pane to create a new picture, find the x and y coordinates of the area you want to blur and call the `blurArea(…)` method with those coordinates. Then tell your picture to repaint itself like this:



5) What are the pre-conditions and post-conditions for this method?

   Pre-conditions:

   Post-conditions:

6) Add a test at the beginning of the method to make sure the preconditions are satisfied. If not, print out an error statement and return out of the method.
7) When you are satisfied that your `blurArea(…)` method is working, working, you are ready to move to Part C. For this part you just need to submit the updated Picture.java file.

Part C: Colorful Edge Detection

1) In this part of the lab, you need to add a method called `colorfulEdgeDetection(..)` to Picture.java. You should use **Program 37** in the textbook as a starting point. The objective is to make two changes to this method. The first change is that instead of turning all the edges (i.e., the top pixel in the comparison) black, it leaves them whatever color they already are. Make this change first and test it using the Interactions pane. Try a parameter value of about 5 or 7.
2) The second change is to only do edge detection when the average color of the top pixel is above a certain threshold that gets passed in as a second double parameter. This will ensure that really dark pixels stay filled in with their original color. The final effect should be a cool filter that looks something like this (shown applied to the pilot_bruce.jpg photo, with an edge detection threshold of 5 and a color level of 70):

```
Welcome to DrJava.  Working directory is /Users/clatulip/Document
> Picture pic = new Picture(FileChooser.pickAFile());
> pic.explore();
> pic.colorfulEdgeDetection(5, 70);
> pic.repaint();
>
```

   In order to do this, you will need to add an extra condition to the if statement in the method that checks whether the current pixel's average color is over the color threshold that is passed in as the second parameter. You need to figure out what this extra condition should be.
8) When you are satisfied that your `colorfulEdgeDetection` (…) method is working, working, you are ready to move to Part D. For this part you just need to submit the updated Picture.java file.

Part D: Truth tables

1) Download **Lab12PartD.java** and open it in DrJava. Look through the program to see if you can understand what it does. Then, compile it and run it. You should see something like this, which is an incomplete truth table:

```
Welcome to DrJava.  Working directory is /Use
> run Lab11PartD
  x      y      z       Condition      Result
false  false  false  !(x && (y || z))
false  false  true   !(x && (y || z))
false  true   false  !(x && (y || z))
false  true   true   !(x && (y || z))
true   false  false  !(x && (y || z))
true   false  true   !(x && (y || z))
true   true   false  !(x && (y || z))
true   true   true   !(x && (y || z))
>
```

2) Does the condition evaluate to true or to false for the values of x, y and z in each row? Your job is to use a pencil and paper to figure out what should go in the Results column. Then, complete the code (after the TODO comment on line 41) to fill in the last column of this 2D array with the correct answer for each row. **Just print the correct value, you don't have to write code to figure out the value.**
9) When you are finished, compile and test your program. If you think you have the answers right, you are ready to move on to Part E. For this part you just need to submit the updated Lab12PartD.java file.

Part E: Picture Roulette

10) Create a new java program called **PictureRoulette.java** that takes at least one runtime parameter and also a second possible runtime parameter. The first runtime parameter is the fully qualified filename of an image file. The second (optional) parameter is an integer between 0 and 9.

11) If there isn't a first parameter the program should print an error message and quit. You don't know how to check to see if this parameter represents a valid filename so just quit the program if there aren't any parameters entered.

12) Assuming an image filename is passed in correctly as the first parameter, the program should then check to see if a number between 0 and 9 has been passed in as the second parameter. If a second parameter is passed in, but is not an integer between 0 and 9, the program should again print out an error message and quit. The following line of code will be useful to you for converting a command line parameter to an integer:

```
num = Integer.parseInt(args[1]);
```

13) If there was no number passed in as a second parameter, your program should generate a random number between 0 and 9 then output that information to the console, something like this:

```
Number generated is 7.
```

14) Assuming that the filename is passed in and the number is passed in (or generated) correctly, the program should use the number as a switch value and run an image modification method on the photo, according to the number passed in. You should also output information to the console about what is being done to the image. For example, if you entered a 7, you should print out something like this (see the table below):

```
Running reverse method on image.
```

15) The table below shows what image methods from Picture.java should be called based on the number passed in or generated. (For number 1, assume "pic" is the name of your Picture object.) If you don't have one of those methods shown below you may substitute a different one but you need ten unique methods.

Number	Picture.java method and parameters
0	`blurArea(…)` // blur the whole picture with level 6
1	`crop(pic.getWidth()/3, pic.getHeight()/3, pic.getWidth()/3, pic.getHeight()/3)`
2	`horizontalShutters()`
3	`checkerboard(10, Color.BLACK)`
4	`verticalShutters()`
5	`mirrorHorizontalBottomToTop()`
6	`fade()` (5 times)
7	`reverse()`
8	`decreaseBlue(0.2)`
9	`colorfulEdgeDetection(7.0, 50)`

16) After running the appropriate method on the image, the image should be shown on screen.

17) When you are satisfied that your PictureRoulette.java program you are done with the required parts for this lab. For this part you will need to submit the PictureRoulette.java file.

Bonus #1 Animate edge detection and blurring: Write a program that uses the `blurArea(…)` and the `colorfulEdgeDetection(…)` methods, animating between the original image, changing slowly to more and more edge detection, then reversing back to the original, and then slowly blurring more and more, and finally changing back to the original. **Earn a stamp for your team!**

Bonus #2 Calculate truth table values: Change Lab12PartD.java so that it calculates what should go in the Results column, and can adapt if the expression in the Condition column changes. **Earn a stamp for your team!**

You're done!

If you are using the lab computer, remember to logout.

So what did you learn in this lab?

1) Practice using nested loops
2) Practice using compound conditionals to control an outcome
3) Writing private helper methods
4) Testing preconditions prior to executing a method
5) Evaluating Boolean expressions
6) Simulating animation
7) Calculating Boolean values

Lab12 Canvas submission

1. Navigate to the course Canvas page.
2. Click on Module 12
3. In Module 12 locate and click on the Lab 12 assignment and submit the following files
    a. Picture.java
    b. Lab12PartD.java
    c. PictureRoulette.java

4. If you completed the bonus submit your files and include a comment with your submission indicating which bonuses you completed.