ITSC 1212 Introduction to Computer Science Lab 11

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

OVERVIEW

1) Part A: Private isSameSize() method
2) Part B: Private copyPixel() method
3) Part C: addPicture method
4) Part D: Crop Method #1
5) Part E: Crop Method #2

Bonus: Copy and Paste SubPicture.

Part A: Private isSameSize() method

1) Create a new, private method in **Picture.java** called `isSameSize(Picture source)`. This method should return a boolean value: true if the source image is the same size as the image object this method is called on (i.e., "this") and false if they are not the same size. Here is the code for this private method, which is quite simple:

```
private boolean isSameSize(Picture source) {
  if (source.getWidth() == this.getWidth()) {
    if (source.getHeight() == this.getHeight()) {
      return true;
    }
  }
  return false;
}//end of isSameSize()
```

   We are making this a private method, because we will only use it inside of Picture.java, from within the public methods. Methods like this are often called "helper" methods because their usual purpose is to "help" other methods do their job.
2) Compile Picture.java to make sure that it has no errors. We'll test the `isSameSize()` method a little bit later in this lab. You don't have to submit anything for this part. Just go on to Part B.

Part B: Private copyPixel() Method

1) Create a new, private method in **Picture.java** called `copyPixel(Picture srcPic, int srcX, int srcY, Picture dstPic, int dstX, int dstY)`. This method will copy the color of **ONE** pixel in the source image to the **SINGLE** specified pixel in the destination image. We are making this a private method, since it is only going to be used by picture objects within the Picture class. Notice that the parameters define the source picture and its coordinates as well as the destination picture and its coordinates.
2) The pseudocode for the method is as follows:
   - Create local variables for srcPixel and dstPixel
   - Using the x and y parameters that are passed in, get **ONE** pixel from the source image
   - Do the same for the **SINGLE** destination pixel
   - Get the color from the source pixel and use it to set the color of the destination pixel.

   Note that if you aren't sure how to convert this pseudocode into code, look at the programs you typed in for homework. They all contained similar steps.

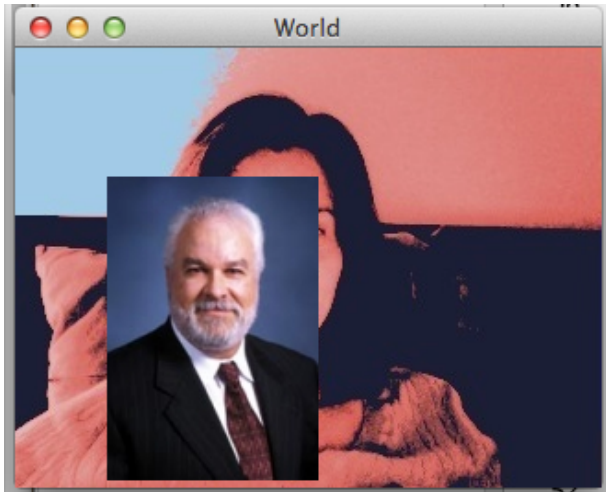
3) Compile your code to make sure that it doesn't have any errors.
4) Now, add the following method to **Picture.java**. This will do a simple copy from a target image onto "this" image. Back in Lab 9 you created a method to copy pixels from one image to another. The method below is a simplified version that accomplishes the same thing, but uses nested `for` loops and calls the private `copyPixel()` method you just created. It also calls the private `isSameSize()` method to check the pre-condition that the two pictures are the same size.

```
/* The target image is the Picture object this method is called on.  The method copies all
 * the pixels from the source Picture passed in to the target image, completely replacing it.
 */
public void simpleCopyFromSource(Picture source) {
  // Check that both source and target images are the same size.
  if (!this.isSameSize(source)) {
    System.out.println("Error! The source and target images are not the same size.");
    return;
  }
  // Iterate through the rows and columns of the source, copying each pixel to the target (this).
  for (int x=0; x<this.getWidth(); x++) {
    for (int y = 0; y<this.getHeight(); y++) {
      copyPixel(source, x, y, this, x, y);
    }
  }
}
```

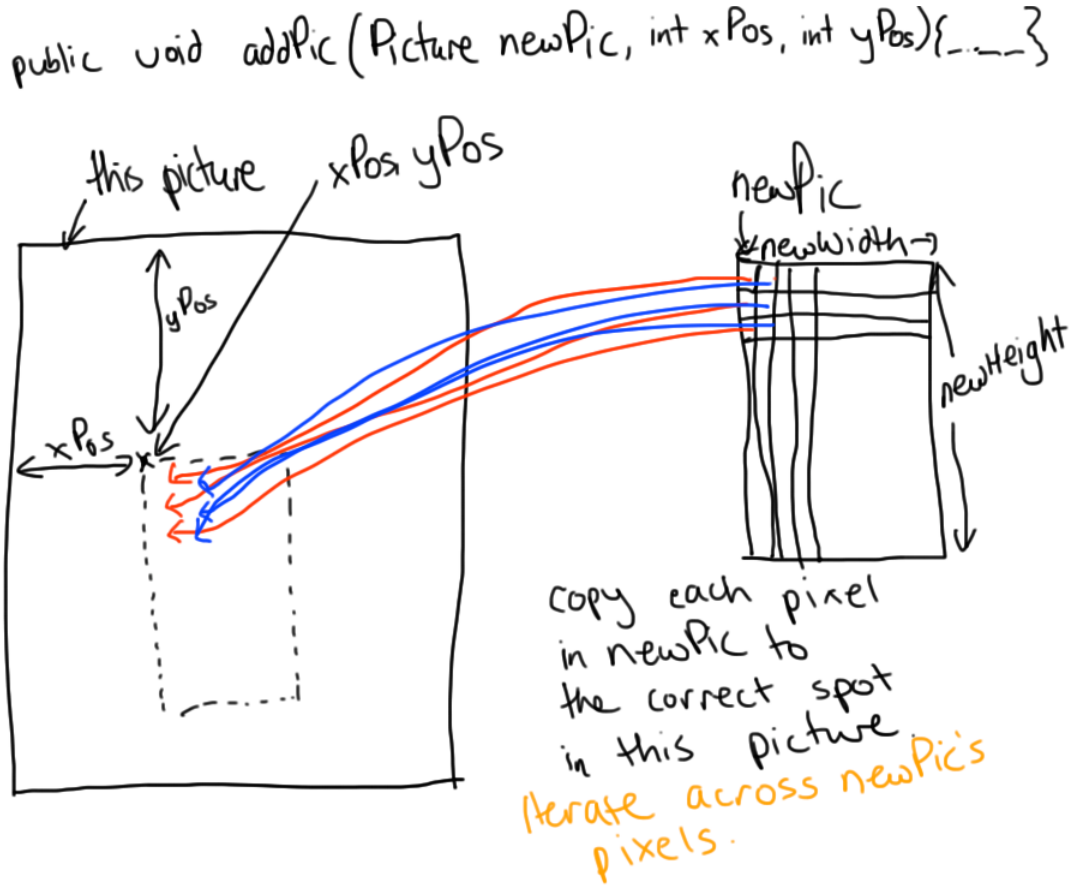
5) Compile Picture.java to make sure that the new methods have no errors.
6) Download **Lab11PartB.java** from Canvas and open it in Dr.Java. Compile and run it to test the `simpleCopyFromSource()` method (and by extension, this will test your two new private methods). Test your program by trying to copy two pictures that are the same dimension. You can use `right_side_up_bread.jpg` as the target and `upside_down_bread.jpg` (both available from Canvas) as the source. If your method is working correctly the last image displayed should be the upside down bread. Also test with two images that are not the same size, to be sure that you are getting the appropriate error message.
7) When your `simpleCopyFromSource()` method is working properly, you are ready to move to Part C. For this part you just need to submit the updated Picture.java file.

Part C: Add Picture method

1) Add a public method to **Picture.java** that copies the pixels from a smaller image and places them somewhere onto "this" image. This method should be called `addPic(Picture newPic, int xPos, int yPos)`. It should take a Picture object as the first parameter, and the x and y coordinates of where to place the image in "this" picture as the second and third parameters. The method returns nothing (void). The example below shows what we are trying to accomplish where we open the picture of Celine, then open the smaller picture of Bruce, and add Bruce's picture at the coordinates (50,70) to the picture of Celine:



2) Here is a diagram of how we are going to do this:



3) So, we are going to iterate through the pixels of the second picture (i.e., newPic, that's the picture of Bruce), and copy those colors into the first picture (i.e., "this") which is the picture of Celine, but at position `xPos+x` and `yPos+y`, (where x and y are the coordinates in newPic) so that the new picture appears at the correct offset in "this" picture.

Use your sketchbooks to draw a smaller example but label the actual row and column numbers. Identify for the relationship between the coordinates of the source pixel and the corresponding destination pixel. Given any source pixel's coordinates, what is the formula that determines the correct coordinates of the destination pixel? (We gave it to you above.)

Here is some of the code for the addPic method, to get you started:

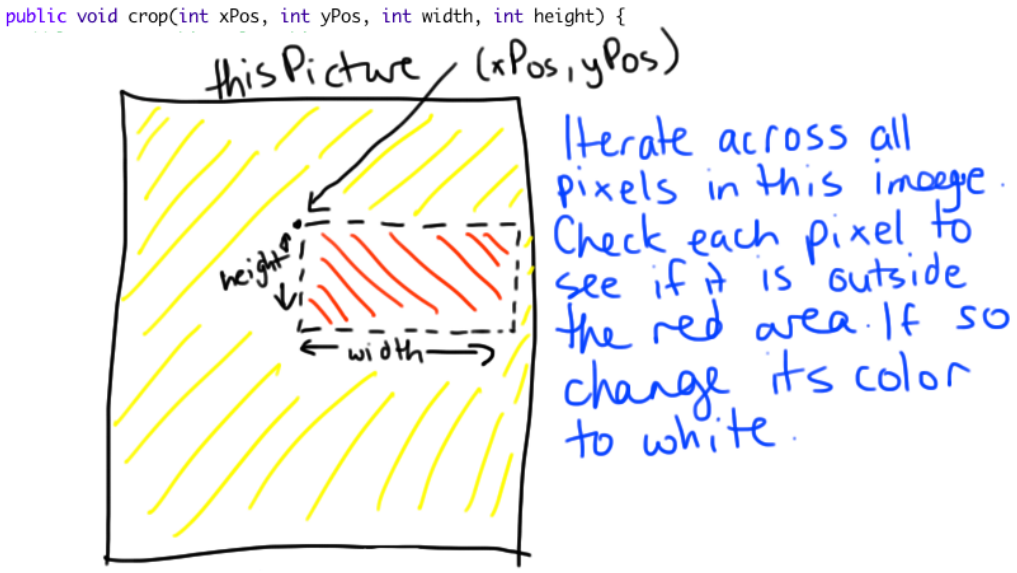
```
//adds newPic to "this" pic, at xPos, yPos
//preconditions: assumes newPic is smaller than current pic
//postconditions: newPic is added to "this" picture starting at (xPos, yPos).
public void addPic(Picture newPic, int xPos, int yPos)
{
  //first check preconditions, newPic has to be smaller than "this" in both dimensions
  if (newPic.getWidth() > this.getWidth() || newPic.getHeight() > this.getHeight()) {
    System.out.println("Error! The passed in picture is larger than this picture");
    return;
  }

  int newWidth = newPic.getWidth();
  int newHeight = newPic.getHeight();

  //iterate through each column and row
  for (int x = 0; x < newWidth; x++) {
    for (int y = 0; y < newHeight; y++) {
      //make sure we aren't trying to copy beyond either
      //the left side of "this" image or the bottom of "this" image
      if( ((x + xPos) < getWidth()) && ((y + yPos) < getHeight()))
      {
        // TODO: call our private copyPixel method
      }
    }
  }
}//end of add pic()
```

4) All you have to do is put in the call to the `copyPixel()` method, with the correct parameters. Once you have done this, download **Lab11PartC.java**, compile it and run it, to test that your method is working correctly.
5) When you have this working, you are ready to move to Part D. For this part you just need to submit the updated Picture.java file.

Part D: Crop method #1

1) In this part of the lab, you need to add a public method called `crop(int xPos, int yPos, int width, int height)` to the **Picture.java** file. This method returns nothing. It will crop the image so that a rectangular area of `width` by `height` remains unchanged starting at position (xPos, yPos), while everything outside of that area is set to white. Like this:

2) Here is a diagram of how this could be accomplished, for an arbitrary (x,y) position, width, and height:



This is another good opportunity to use your sketchbooks to see how this works using a smaller example with real row and column numbers. For any (x,y) coordinate, how can you tell if it's inside (or outside) the area you want to leave untouched? What questions should you be asking?

3) You will do this by iterating through every pixel in the image, and seeing if it is outside of the area that should be untouched. If it is outside of the untouched area, you simply set the color of that pixel to white. Hint: you can test this by checking 4 different cases. One of those cases is if your current pixel has an x value less than xPos. If that's true, the pixel should be changed to white. What are the other three conditions? You can write this method by putting four different if statements inside your nested `for` loop, one per condition. Or, you can create a compound condition that looks like this:

```
if (condition1 || condition2 || condition3 || condition4) { } // executes if any of them are true
```

The double pipes (||) are an OR symbol. This if statement says "if condition1 is true OR if condition2 is true OR if condition3 is true OR if condition4 is true then execute all the statements inside the curly braces." When you use || like this if any one of the individual conditions are true, the whole compound condition (i.e., everything inside the parentheses) is true.

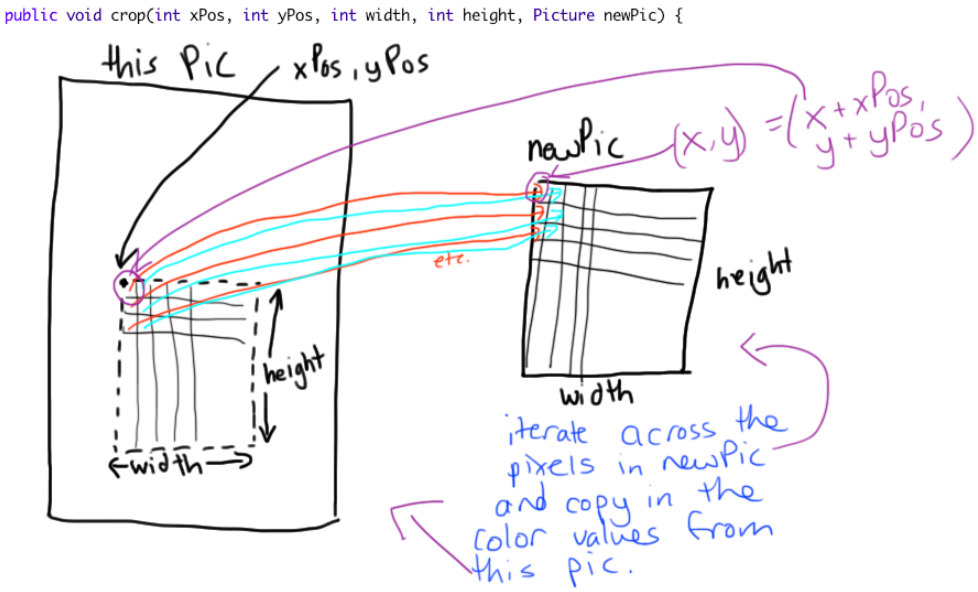
4) Include precondition tests to ensure that the untouched area defined by the method parameters is completely contained within the picture.
5) When you are finished, test your crop method by downloading the **Lab11PartD.java** file. Compile and run. This program will prompt you to open an image, and then it will crop the image so that everything outside of a small area in the upper-left is white.
6) Once you are satisfied your crop method is working, you are ready to move to Part E. For this part you just need to submit the updated Picture.java file.

Part E: Crop method #2

1) In this part of the lab, you are going to make another method called crop, but it will have an additional parameter: you will pass in a new, separate Picture object called `newPic`. The point of this method is to copy pixels from the source picture and place them into `newPic`. When you're done you'll still have the original picture but you'll now have a new one that's only a subset of the original.
2) Note that when you're done you're going to have two methods named `crop` in Picture.java. Having two methods with the same name is called *method overloading* and it is possible as long as the parameter list (or method *signature*) is different (i.e. there can be a different number of parameters, or the same number of parameters but with different data types, or the same number and data types of parameters but in a different order). The signature of your new crop method should look like this:

```
public void crop(int xPos, int yPos, int width, int height, Picture newPic) {…}
```

3) Here is a diagram of how this method should work:



Sketchbooks!

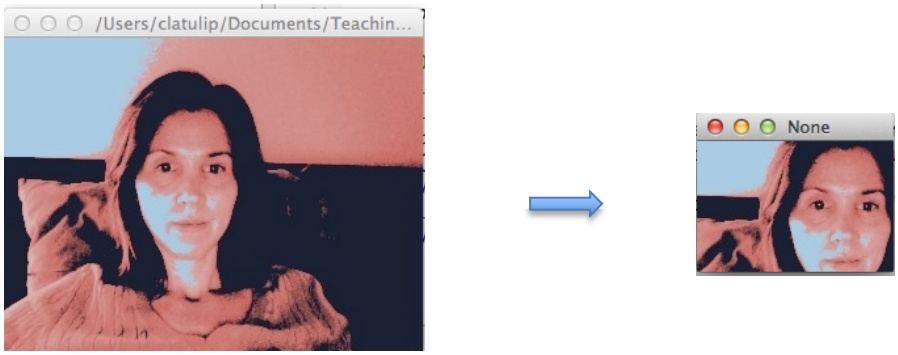
4) The pseudocode for this method is:

- Check precondition that the newPic is the correct size (equal to the width and height parameters passed in). If not, return without continuing the method
- Iterate through the pixels in the new, blank image
- For each pixel in `newPic`, copy over the appropriate pixel from "this" image object (you should be able to use the `copyPixel` method from Part B)
- Note that before copying you want to make sure that you aren't going out of bounds of "this" picture. This compound condition, which evaluates to true if both parts are true, should help:

```
if (x + xPos < getWidth() && y + yPos < getHeight()) { // copy pixel }
```

5) Test your method by downloading, compiling and running **Lab11PartE.java**. This file will prompt you to open a picture. Then it creates a new, blank picture that is 150 by 100 pixels,

and crops the part of the picture you selected into this new blank picture. The original picture isn't changed at all, but you end up with a new image that is the cropped part of the original:



6) When you are satisfied that your second crop method is working, you are done with the required parts for this lab. For this part you just need to submit the updated Picture.java file.

Bonus: Copy & Paste SubPicture. Write a method that copies a portion of a picture, and pastes it somewhere else in the <u>same</u> picture.

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

So what did you learn in this lab?

1) Created private "helper" methods
2) Used private methods
3) Practiced validating pre-conditions and generating error messages
4) Used compound conditionals in if statements
5) Used nested for-loops
6) Practiced referring to individual pixels using (x,y) coordinates
7) Moved pixels from one picture to another
8) Use parameters to control how methods function
9) Replaced all of one picture with a different picture
10) Replaced part of one picture with a different picture
11) Cropped part of a picture by making all but a subset white
12) Created a new picture by extracting a subset of a picture

Terminology and concepts:

- Nested for-loops
- Compound conditionals
- Two-dimensional references
- Overloading
- Method signature

<u>Lab 11 Canvas submission</u>

1. Navigate to the course Canvas page.
2. Click on Module 11
3. In Module 11 locate and click on the Lab 11 assignment and submit the following files
    a. Picture.java

4. If you completed the bonus submit your files (all work should be in Picture.java) and include a comment with your submission indicating which bonuses you completed.