Every programmer every day, perhaps unconsciously, thinks about and uses the computer operations below:

- Create a variable, specifying its name and type. That variable is computer memory for storing data.
  Example: `int count;`
- Copy literal data into an already created variable
  Example: `count = 98;`
- Combined create a variable and copy literal data into it
  Example `int countTwo = 2;`
- Combined create a variable and copy data from an existing variable into it
  Example: `int countThree = count;`
- Copy data from one already existing variable into another already existing variable
  Example: `count = countTwo;`

In this lab, you will add to your `EditablePicture` class two behaviors. A behavior is an operation that the computer will become capable of doing on this `EditablePicture`. You will program each new behavior by programming a method into your `EditablePicture` class definition. Make your `Lab7` directory, copy your work from `Lab7 Upload to BB...for pts.`

```java
public void copyIn(int originX, int originY, Picture source)
```

In brief: It should copy the Colors from the `source Picture` into this `EditablePicture` so the upper-leftmost pixel of `source` (with coordinates (0,0)) is copied to the Pixel with coordinates `(originX, originY)` of this `EditablePicture`. Of course, the rest of the `source Picture` should be copied to corresponding Pixels to the right of and/or and below `(originX, originY)`.

You'll have to call the methods to find out the width and height of the source `Picture`. Those numbers are returned by the method calls `source.getWidth()` and `source.getHeight()` respectively. More accurately, you will write code in `EditablePicture` to eventually call these methods in the future, when the `copyIn` method is eventually called.

Program `copyIn` to run a doubly nested loop, that runs its body once for each `Pixel` in the source `Picture`. This is easy: Relate this lab to the code for making an entire `Picture` turn white from our Mar6 lecture. A copy and perhaps helpful questions appear on page 4.

Here is what you must program the body of the doubly nested loop to do: Copy the Color information from one `Pixel` gotten from the `source Picture` into one `Pixel` gotten from this `EditablePicture`. How? We explain it step-by-step below.

Suppose the loop variables `x` and `y` run from 0 to `source.getWidth() - 1` and `source.getHeight() - 1` respectively.

You can get the `Pixel` from the `source Picture` with

```
source.getPixel( x, y )
```

You can extract the `Color` from the `Pixel` just gotten above with
source.getPixel( x, y ).getColor()

IMPORTANT: Suppose the source Pixel with has coordinates \((x, y)\). The coordinates of the Pixel in this EditablePicture corresponding to that source Pixel are \((\text{originX}+x, \text{originY}+y)\)

So make the computer get the destination Pixel with

\[
\text{this.getPixel(\text{originX}+x,\text{originY}+y)}
\]

and set its Color with

\[
\text{this.getPixel(\text{originX}+x,\text{originY}+y).setColor( ????) ;}
\]

But what do you put in ???? for the Color? Take your own pick: It's a matter of style!

**Verbose, step-by-step style:**
1. Set up a temporary Color variable:
   \[
   \text{java.awt.Color tempColor;}
   \]
2. Copy the Color into it:
   \[
   \text{tempColor = source.getPixel( x, y ).getColor();}
   \]
3. Use tempColor for ???? to denote the Color you want to copy.

**Succinct, functional style:**
Just code the expression that expresses the Color that was gotten by
\[
\text{getColor( ) for ???? That is where that Color should be used in the computation.}
\]

Testing: Copy One.jpg from /usr/local/depts/cs/geintro/One.jpg
Make your app make and then show or explore an image like below:
(Warning: If you test app causes accesses to Pixels that are not in the Picture, your program will crash. Don't worry: Determine whether there is a bug in your new method's code or simply your app is called with the wrong parameter values or the Picture you are copying in is too big to fit. It's possible however that BOTH of these problems occur.)
```java
public EditablePicture copyOut(int originX, int originY, int width, int height)

First: Make sure your EditablePicture class has the following constructors, which call super
class constructors: (We also gave you a free start.)
public class EditablePicture extends Picture
{
    public EditablePicture( int width, int height )
    {   super( width, height ); }
    public EditablePicture( Picture p )
    {   super( p ); }
    public EditablePicture( String fname )
    {   super( fname ); }
    public void copyIn(int originX, int originY, Picture source)
    { /*your work from Part 1 of this lab*/ }
    public EditablePicture copyOut
    {   (int originX, int originY, int width, int height)
        EditablePicture destP = new EditablePicture( width, height );
        CODE THE SECOND PART WORK HERE!
        return destP;
    }
}

It should first create a new EditablePicture whose dimensions are given by the parameter values
of width and height. It will then copy the color information into that new EditablePicture
from the rectangular region of this EditablePicture specified by the originX, originY,
width and height parameter values. Finally, your method must return as a return value the
(reference to) new EditablePicture that it had created. We teach that in step 3 below.
How: 1. Code
    EditablePicture destPict = new EditablePicture( width, height );
    2. Code the loop to copy the Color information into destP (details below)
    3. Code the return operation with what value to return.
    return destP;
Details: Program a doubly nested loop that one-by-one copies the Color of a Pixel located at
(originX + x, originY + y) in this EditablePicture TO the Pixel of destP located
at (x, y). Use the same operations from the first part of the lab in different ways!

Testing: Make your app make and then show or
explore an image like:
public void makeAllWhite()
{
    int x; //set up a variable to track how many columns we finished whitening.
    x = 0; //No columns were whitened when we begin.
    while( x < this.getWidth() ) //loop until we whitened all the columns
    {
        //whiten a whole column.
        int y; //set up a var. to track how many pixels in column x we finished whitening
        y = 0; //None were whitened when we begin a column.
        while( y < this.getHeight() ) //loop until we whiten all the pixels in one column
        {
            //Set up a var. to refer to the Pixel we will whiten.
            Pixel pRef; //Set up a var. to refer to the Pixel we will whiten.
            pRef = this.getPixel(x,y); //Get the Pixel to whiten.
            y = y + 1; //Count one more whitened Pixel in the column.
        }
        x = x + 1; //Count one more finished column.
    }
    return; //Continue computing according to the caller's code.
}

Questions you must ask yourself((and then answer!)) in order to adapt the ideas in the above code to the current lab:

1. Which Picture will the colors COME FROM? (this EditablePicture or the Picture referred to by source?)

2. Which Picture will the colors BE COPIED TO? (this EditablePicture or the Picture referred to by source?)

3. Is the part of the Picture to be copied THE WHOLE of the Picture the colors come from, or is it a square or rectangular PIECE of the Picture the colors come from? If it is a whole, the values that the loops make for variables x and y will range from 0 to the width-1 and the height-1 of the whole Picture. That is what is done in makeAllWhite. But, if it is a PIECE, then the values you must program are determined (in this lab) from your method's parameters.

4. What are the (simple) mathematical and geometric relationships between the coordinates of the Pixel to be copied FROM within its Picture to the coordinates of the Pixel to be copied TO within the other Picture? It's simple; the only mathematical operation involved is addition.

5. Pretend a TA or the prof. pointed to line of code, or there was a blank space next to some code in a method printed in the 201 midterm or final exam, and you were asked: “What is the purpose of that line of code? Why did the programmer write it? What would happen if it were omitted?”.