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

- Create a variable, specifying its name and type. That variable becomes available for storing data.
  Example: `int count;
- Copy literal data into an already created variable
  Example: `count = 98;
- Combined create a variable and copy 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 existing variable into another
  Example `count = countTwo;

In this lab, you will add to your `PaintablePicture` class two behaviors. A behavior is an operation that the computer will become capable of doing ON `this PaintablePicture`. You will program each new behavior by programming a method into your `PaintablePicture` class definition. Make your `Lab07` directory, copy your work from `Lab06 Upload to BB...for pts.

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

In brief: It should copy the `Colors` from the `source Picture` into `this PaintablePicture` so the upper-leftmost `Pixel` of `source` (with coordinates (0,0)) is copied to the `Pixel` with coordinates `(originX, originY)` of `this PaintablePicture`. 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.

Program it to run a doubly nested loop, that runs its body once for each `Pixel` in the source `Picture`. This is easy: Relate the contents of the first 4 pages of "Modifying Pixels in a Matrix" with discussion during the lab. Copies of those 4 pages are available in the lab.

Here is what you must program the body of the doubly nested loop to do: Copy the `Color` information from one `Pixel` extracted from the `source Picture` into one `Pixel` extracted from `this PaintablePicture`. 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 extract the `Pixel` from the `source Picture` with

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

You can extract the `Color` from the above `Pixel` with

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

So make the computer extract the destination Pixel with

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

and set its Color with

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

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:
   
   ```java
   java.awt.Color tempColor;
   ```
2. Copy the Color into it:
   
   ```java
   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 `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.)
public PaintablePicture copyOut(int originX, int originY, int width, int height)

First: Make sure your PaintablePicture class has the following constructors, which call super class constructors: (We also gave you a free start.)

public class PaintablePicture extends Picture {

    public PaintablePicture( int width, int height )
    { super( width, height ); }

    public PaintablePicture( Picture p )
    { super( p ); }

    public PaintablePicture( String fname )
    { super( fname ); }

    public void copyIn(int originX, int originY, Picture source)
    { /*your work from Part 1 of this lab*/ }

    public PaintablePicture copyOut
    (int originX, int originY, int width, int height)
    {
        PaintablePicture destP = new PaintablePicture( width, height );

        CODE THE SECOND PART WORK HERE!

        return destP;
    }
}

It should first create a new PaintablePicture whose dimensions are given by the parameter values of width and height. It will then copy the color information into that new PaintablePicture from the rectangular region of this PaintablePicture specified by the originX, originY, width and height parameter values. Finally, your method must return as a return value the (reference to) new PaintablePicture that it had created.

How: 1. Code

    PaintablePicture destPict = new PaintablePicture( 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 PaintablePicture 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: