Module III: Copying Pictures

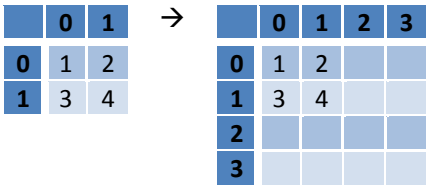
Learning Objectives

Copy pixels from one image to another.

Declare, initialize, and use multiple variables within a for loop.

Copying pixels to a new picture

Let's think about what it means to copy a simple matrix of numbers into a bigger two-dimensional array:



The algorithm looks like this:

Loop through source array (nested x and y loops)
Get the current source and target values
Set the value of the target array element to the value of the source array element
Note: in the case of pictures, we can be more specific and say instead *Set the color of the target pixel to the value of the source pixel*

To implement this, we first need a *source* image to copy from (we will use a ladybug) and a *target* image to copy into. To simplify things, we'll copy the ladybug into a blank picture. You may also use any other pictures, as long as the target picture is at least as big as the source picture you're copying.

Next we need to decide which image the algorithm should operate on. We'll add it to the Picture class, and we'll assume that it's always run on the target image. Let's call it copyPicture(). All this work is done in MediaSeven.

Note... This means that if we have a target image called targetImage and a source image called sourceImage, we'd call targetImage.copyPicture(sourceImage) to copy the source to the target. Why is this? Both targetImage and sourceImage are instances of the Picture class, so both will have a method called copyPicture(). Our implementation of copyPicture() will modify the object it belongs to using the argument passed in. If we call targetImage.copyPicture(sourceImage), the code inside copyPicture() will copy information from the parameter (sourceImage) to targetImage (accessible via the variable this).

Now we'll need a plan for how to copy the images. Let's start by thinking about how we should copy a single pixel into the equivalent pixel in the other image. This should work the same way as in our mirror algorithm: We simply copy the color of the source pixel into the target pixel using **targetPixel.setColor(sourcePixel.getColor()).** First, though we'll need to get the pixels. We can use the syntax ***Type variableName[= value], variableName[= value],*** *...* to create more than one variable of the same type (setting the value is optional, which is why we put it in ***[]***). Let's create two pixels **Pixel sourcePixel, targetPixel**. Later we'll assign values to them for the current pixel using **sourcePixel = sourcePicture.getPixel(sourceX, sourceY)** and **targetPixel = this.getPixel(targetX, targetY**).

Now let's figure out how to loop through all of the pixels that need to be copied. As you might guess, we'll use some sort of nested loop just like we did for mirrorVertical() and mirrorHorizontal(). But we have an unusual situation here, because we need

to loop through two two-dimensional arrays simultaneously (the source pixels and the target pixels). Let's start by thinking about how we would loop through each of them individually. We want something like:

```
for (int sourceX=0;   sourceX < sourcePicture.getWidth();  sourceX++) {
   for (int sourceY=0;  sourceY < sourcePicture.getHeight();   sourceY++) {
```

This nested loop should look very familiar. The second nested loop we need should focus on the target picture. It will need to calculate the location to which we should copy each pixel in the source picture. For that loop, we don't want to iterate through every pixel in the target picture, just a small subsection of it that is the same size as the source picture. It should look something like this:

```
for (int targetX=0;   targetX < sourcePicture.getWidth();     targetX++) {
   for (int targetY=0;        targetY < sourcePicture.getHeight();        targetY++) {
```

> Note... Look at the second nested loop. We are iterating through the target picture, but using the size of the *source picture* in the continuation test of the second loop.

Now that we've written these two loops separately, how do we combine them? Luckily, there is an easy way to do this. Recall the syntax for a for loop: **for (*initialization_area; continuation_test; change_area*).** We can create multiple values in the initialization area using the syntax we just introduced above (*e.g.*, **int sourceX=0, targetX=0**) For the continuation test, we'll want **sourceX < sourcePicture.getWidth()** and **sourceY < sourcePicture.getHeight()**. Since we're copying from the source picture, and we are requiring that it be smaller than the target, there's no need to continue past the edge of the source picture in either of our loops. Finally, for the change area, we can use the special syntax **sourceX++, targetX++** to update both of our counter variables. The result looks like this:

```
for (int sourceX=0, targetX=0;      sourceX < sourcePicture.getWidth();      sourceX++, targetX++) {
  for (int sourceY=0, targetY=0;        sourceY < sourcePicture.getHeight();        sourceY++, targetY++) { ...
```

Now that we've figured out all of the pieces of our algorithm, we can put them together into a complete implementation:

```
/**
 * copyPicture copies the source picture into this picture (the target)
 * @param Picture source the picture to copy from
 */
public void copyPicture(Picture sourcePicture) {
  Pixel sourcePixel, targetPixel;

  //loop through both pictures simultaenously
  for (int sourceX=0, targetX=0;   sourceX < sourcePicture.getWidth();   sourceX++, targetX++) {
    for (int sourceY=0, targetY=0;  sourceY < sourcePicture.getHeight();  sourceY++, targetY++) {
        //get the source pixel from the source image
        sourcePixel = sourcePicture.getPixel(sourceX, sourceY);
        //get the target pixel from the target image (this)
        targetPixel = this.getPixel(targetX, targetY);
        //set the color of the target pixel to the
        // color of the source pixel
        targetPixel.setColor(sourcePixel.getColor());
    }
  }
}
```

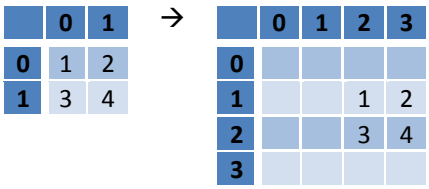
Let's try it out. Write out the copyPicture() method in your Picture.java class file and compile. Then load in a source and target picture and run copyPicture()

```
String fileSource = "<file directory>\\MediaSeven\\src\\mediaseven\\ladybug_sm.jpg";
String fileTarget = "<file directory>\\MediaSeven\\src\\mediaseven\\640x480.jpg";
```

You should see:

Copying pixels to a different location

Technically, the algorithm we just implemented doesn't need both sourceX and targetX since they both have the same value at the start of every iteration of the loop. But we could easily choose to copy a picture to a different location on the screen. For example, suppose we implemented copyPictureOffset(Picture source, int startX, int startY). This method should place the upper left hand corner of source in the location (startX, startY). Let's illustrate this in terms of simple two-dimensional number arrays:



	0	1
0	1	2
1	3	4

→

	0	1	2	3
0				
1			1	2
2			3	4
3				

Take a look at the matrixes above and see if you can come up with a simple algorithm for copying from one array to another.

learn by doing

Try to write down an algorithm for copying a two-dimensional array to a position that is offset by (startX, startY). from the top left corner of the target array. Your algorithm should contain step-by-step instructions. If you follow them without deviating, can you successfully copy an example array? See if you can come up with some sample problems to test it on. But don't use any test cases where the source array would end up partly off the target array.

Hint…

- First you will need to figure out how to represent the location you are copying to in terms of the offset.
- If you would normally copy a source pixel located at (x, y) to the position (x, y) in the target picture, what might you change the target position to?
- Since the offset is defined by (starX, startY), you'll want to copy each pixel to (startX+sourceX, startY+sourceY)
- Another way to do this is to initialize targetX and targetY to startX and startY respectively, instead of 0. Then, as you increment targetX and targetY, you will be implicitely adding targetX to source and targetY to sourceY.

Solution:

Your algorithm should look something like this:

```
Create variables to store the current sourceX, sourceY, targetX, and target
Initialize sourceX and sourceY to 0
Initialize targetX and targetY to startX and startY
//loop through the source elements
For each row in the source array
....For each column in the source array
```

.....set the value of the target element corresponding to (targetX, targetY) to the value of the current source element corresponding to (sourceX, sourceY)
.....// note: when (sourceX, sourcey) = (0,0), the source
.....// pixel will be copied to (startX, startY) and so on
.....now increment sourceX and sourceY and targetX and targetY by one

Now it's time to start thinking about this in terms of images and to implement your algorithm. Let's create a method called **copyPictureOffset(Picture source)** that will copy a source picture into a target picture at an offset of (100, 100).

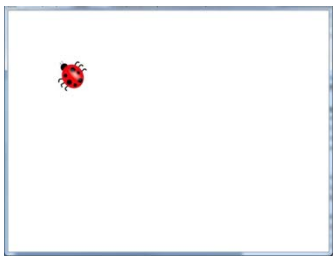
learn by doing:

Implement a working version of copyPictureOffset(Picture source) that will copy the source picture to a location starting at (100, 100) in the target. Create two local variables startX=100 and startY = 100 to specify the offset location to which we want to copy the image. Use them to initialize to specify the offset location to which we want to copy the image. Use them to initialize copy to any location in the image. Test your algorithm out and paste the code in below.

Solution:

```
public void copyPictureOffset(Picture sourcePicture) {
   Pixel sourcePixel, targetPixel;
   int startX = 100, startY = 100;

   //loop through both pictures simultaenously
   for (int sourceX = 0, targetX = startX; sourceX < sourcePicture.getWidth(); sourceX++, targetX++) {
      for (int sourceY = 0, targetY = startY; sourceY < sourcePicture.getHeight(); sourceY++, targetY++) {
         //get the source pixel from the source image
         sourcePixel = sourcePicture.getPixel(sourceX, sourceY);
         //get the target pixel from the target image (this)
         targetPixel = this.getPixel(targetX, targetY);
         //set the color of the target pixel to the
         // color of the source pixel
         targetPixel.setColor(sourcePixel.getColor());
      }
   }
}
```



There are many reasons you might want to copy an image to a location other than (100, 100). For example, a picture and a map copy situation.

Summary

We have introduced algorithms for copying from one picture to another. You can copy from a source image to any location in a target image by keeping track of the offset (startX and startY position).

We have also seen how to declare, initialize, and change more than one value in a for loop: for (*type var1, var2, ...; continuation_test; var1++, var2++, ...*)

These examples are implemented in the Netbeans project MediaSeven.