Module III:

NESTED LOOPS

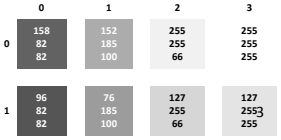
Objectives

- Understand and use a 2D array
- Understand nested loops and how they allow you to manipulate two-dimensional objects.
- Create a piece of code that implements an algorithm, and test it.
- Copy pixels from one image to another.
- Declare, initialize, and use multiple variables within a for loop.

2

Pictures and Two-Dimensional Arrays

- As we explored earlier, a picture can be thought as a two-dimensional array of pixels.
- If we think of a picture this way, each pixel has an x value (its horizontal location) and a y value (its vertical location).
- We learned that the syntax pictureObj.getPixel(x,y) can be used to retrieve the pixel at a specific location.
- This 2D array represents a 4 pixels wide and 2 pixels high picture. getPixel(0,0) would return the rust colored pixel, whose RGB values are (198, 82, 82). getPixel(3,1) would return the grey pixel whose RGB values are (217, 217, 217).



Nested Loops

- You already know how to use loops, and you should have some experience with accessing and manipulating arrays using while and for loops. But how do we use loops with two-dimensional arrays?
- nested loop
  - (definition) A loop that is placed inside another loop. This will cause the inner loop to iterate across a set of values once for each iteration of the outer loop.

4

Nested Loops

- Let's start by thinking about an algorithm for retrieving pixels in an orderly fashion. We could go left to right, and top to bottom, as follows:
  - Get all of the pixels in the first row. Assuming a 640x480 image, this would result in getting the pixel at (0, 0), then (0, 1), then (0, 2) and so on up to (0, 639).
  - Then get all of the pixels in the second row. This would result in getting the pixel at (1, 0), then (1, 1), then (1, 2) and so on up to (1, 639)
  - ...
  - Then get all of the pixels in the last row. This would result in getting the pixel at (479, 0), then (479, 1), then (479, 2) and so on up to (479, 639,)

5

Nested Loops

- But how is this algorithm implemented?
- The most common solution to this problem is to use nested loops, which is when we have a single loop on the outside to loop through the rows (top to bottom) and another one inside that loop to loop through the columns (left to right).
- We can refine the pseudo code given above as follows:
- for each row r (0 to height-1):
  - for each column c (0 to width-1):
    - do something with the pixel at (r, c)

6

Nested Loops

- When one loop is nested *within* another, the inner loop is executed, *in full* each time the outer loop goes through one iteration.
- In the example above, the outer loop's first iteration is concerned with the first (r=0) row. The inner loop will iterate through *every column* (0-width-1) of the 0th row. Then the outer loop's first iteration will complete, and it will begin the second iteration, which is concerned with the second row (r=1). Again, the inner loop will iterate through every column in that row. And so on, until the final (r=height-1) row is completed.

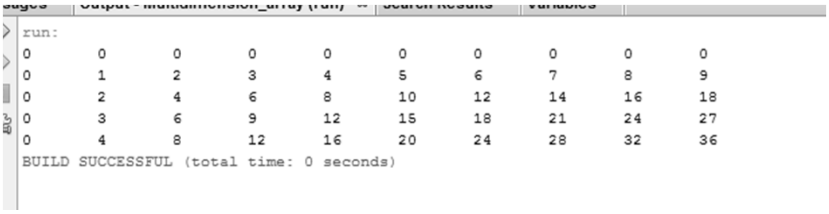
7

- Observation...
  - It doesn't actually matter whether you iterate through a picture row first or column first. Either way, you will eventually get to every pixel in the image exactly once.

8

Example:

- Lets start with something a bit easier: create a 2D array representing the multiplication table.



9

Solution

```
int[][] table = new int[5][10];
for (int row = 0; row < table.length; row++) {
  for (int col = 0; col < table[row].length; col++) {
      table[row][col] = row * col;
  }
}
for (int row = 0; row < table.length; row++) {
  for (int col = 0; col < table[row].length; col++) {
      System.out.print(table[row][col] + "\t");
  }
  System.out.println();
}
```

10

Nested Loops

- Here is a simple template with nested loops that you might use to go through the pixels in an image.

```
Pixel pixelObj;
for (int y = 0; y < this.getHeight(); y++) {   // loop through the rows
   for (int x = 0; x < this.getWidth(); x++) {    // loop through the
   columns
    //get the current pixel
    pixelObj = this.getPixel(x, y);
   // do something with the pixel such as pixelObj.setColor(...)
   }
}
```

11

Lightening an Image

- To lighten the color of an image, we can call pixel.brighter() for each pixel in the image.
- Our algorithm should look something like this:

```
for each row r (0 to height-1):
   for each column c (0 to width-1):
      Get the pixel at (r, c)
      Lighten the color of that pixel
      Set the color of that pixel to the lighter color
```

12

Can you implement the algorithm?

```
public void lighten() {
    Pixel pixel;  Color c;

    for (int y = 0; y < this.getHeight(); y++) {  //rows
      for (int x = 0; x < this.getWidth(); x++) {  //columns
        pixel = this.getPixel(x, y);
        //brighten the current color of the pixel
        c = pixel.getColor().brighter();
        //set the color of the pixel to the lighter color
        pixel.setColor(c);
      }
    }
}
```

13

Learn by Doing

- Create the method clearBlue( ) to use a nested loop to loop through all the pixels setting blue to 0.
- Run the method to check that it works.

14

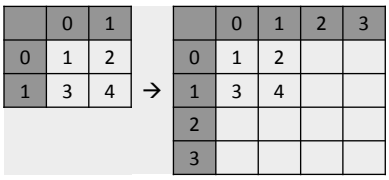
solution

```
public void clearBlue() {
    Pixel pixel; Color c;
    for (int y = 0; y < this.getHeight(); y++) {  // rows
        for (int x = 0; x < this.getWidth(); x++) {  // col
            pixel = this.getPixel(x,y);
            //set the blue on the pixel to 0
            pixel.setBlue(0);
        }
    }
}
```

15

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:



16

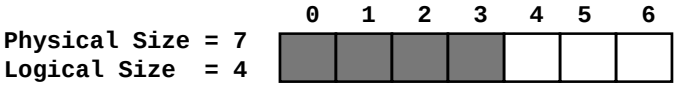
Arrays

- Just a reminder, arrays in java are not dynamic. If you want to change the size of the array then you need to do that manually.
- Before we move on with pictures, let us create a resize method that adjusts the size of the array when we have reached the max number of elements.

17

Physical vs Logical Size

- Physical Size
  - The physical size of an array is the total number of array cells. That is, the number used to specify the capacity when the array was created or resized.
  - Car[] lot= new Car[100];
  - The physical size of this array is lot.length = 100.
- Logical Size
  - The logical size of an array is the number of items that have been added to the array. If we want to keep track of the logical size of an array we need to do it ourselves with a counter.



18

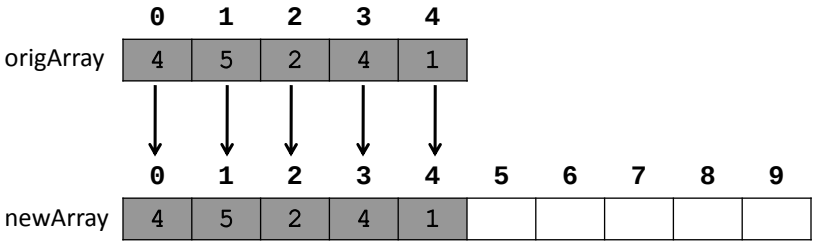
Algorithm

- The algorithm looks like this:
  - Create a new larger array
  - Loop through source array
    - Get the current value
    - Set the value of the target array element to the value of the source array element
  - Refer the old array variable to the new array

    - 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*

19

Example




```
int[] newArray = new int[ origArray.length * 2];
for (int i = 0; i < origArray.length; i++ ) {
      newArray[i] = origArray[i];
}
origArray = newArray;
```

20

What about a 2D array?

- 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 a 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.

21

- Next we need to decide which image the algorithm should operate on.
- 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.

22

- Why is this?
  - Both *targetImage* and *sourceImage* are instances of the Picture class, we will add 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).

23

- how do we loop through all of the pixels that need to be copied?
  - We have to realize we need to loop through two 2-d arrays.
- First, how do we loop through the source image?

```
for (int srcX=0; srcX < srcPicture.getWidth(); srcX++) {
   for (int srcY=0; srcY < srcPicture.getHeight(); srcY++) {
```

24

- Second, how do we loop through the target image?
  - We 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.

```
for (int tarX=0; tarX < srcPicture.getWidth(); tarX++) {
   for (int tarY=0; tarY < srcPicture.getHeight(); tarY++) {
```

25

- Do you notice anything similar about these two loops?

```
// loop through the source image
for (int srcX=0; srcX < srcPicture.getWidth(); srcX++) {
   for (int srcY=0; srcY < srcPicture.getHeight(); srcY++) {

// loop through the target image
for (int tarX=0; tarX < srcPicture.getWidth(); tarX++) {
   for (int tarY=0; tarY < srcPicture.getHeight(); tarY++) {
```

26

- Recall the syntax for a for loop:

  for (initialization; condition; incrementer)

- We can create multiple values in the initialization e.g., int sourceX=0, targetX=0.
- For the condition test, we'll want srcX < srcPicture.getWidth() and srcY < srcPicture.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 incrementer, we can use the syntax sourceX++, targetX++ to update both of our counter variables.

27

- With that, how do we combine the two loops together ?

```
for (int srcX=0, tarX=0;
                        srcX < srcPicture.getWidth();
                                                srcX++, tarX++) {
   for (int srcY=0, tarY=0;
                        srcY < srcPicture.getHeight();
                                                srcY++, tarY++) {
```

28

public void copyPicture(Picture srcPicture) {

```
Pixel srcPixel, tarPixel;
//loop through both pictures simultaneously
for (int srcX=0, tarX=0;   srcX < srcPicture.getWidth();   srcX++, tarX++) {
    for (int srcY=0, tarY=0;  srcY < srcPicture.getHeight();  srcY++, tarY++) {
        //get the source pixel from the source image
        srcPixel = srcPicture.getPixel(srcX, srcY);
        //get the target pixel from the target image (this)
        tarPixel = this.getPixel(tarX, tarY);
        //set the color of the tar pixel to the color of the src pixel
        tarPixel.setColor(srcPixel.getColor());
    }
}
```

29

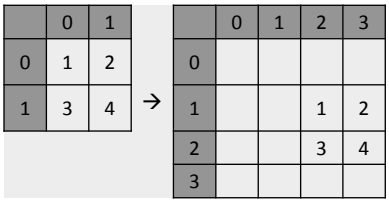
Copying pixels to a different location

- Technically, the algorithm we just implemented doesn't need both srcX and tarX 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).

30

Copying pixels to a different location

- What is the algorithm to the following?



31

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?

32

- 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 (startX, 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

33

Solution

Create variables to store the current srcX, srcY, tarX, and tarY

Initialize srcX and srcY to 0

Initialize tarX and tarY 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 (tarX, tarY) to the value of the current source element corresponding to (srcX, srcY)

        // note: when (srcX, srcY) = (0,0), the source

        // pixel will be copied to (startX, startY) and so on

        now increment srcX and srcY and tarX and tarY by one

34

public void copyPictureOffset(Picture srcPicture) {

```
Pixel srcPixel, tarPixel;  int startX = 100, startY = 100;
//loop through both pictures simultaneously
for (int srcX = 0, tarX = startX; srcX < srcPicture.getWidth(); srcX++, tarX++) {
    for (int srcY = 0, tarY = startY; srcY < srcPicture.getHeight(); srcY++,
      tarY++) {
        //get the src pixel from the src image
        srcPixel = srcPicture.getPixel(srcX, srcY);
        //get the tar pixel from the tar image (this)
        tarPixel = this.getPixel(tarX, tarY);
        //set the color of the tar pixel to the color of the src pixel
        tarPixel.setColor(srcPixel.getColor());
    }
}
```

35

COPY

OFFSET

36

Writing Good Methods

- By now you should be able to use methods to accomplish a lot of specific tasks, but these methods really aren't very general.
- Any time we want to do something different with them, like copy our image to a different location, we have to modify the code and recompile it.
- If we want to keep the old version we also have to create a new method, with a new name, and copy our new code into it.

37

Writing Good Methods

- To avoid issues like this, we'll make a few general guidelines for what makes a "good" method:
  - A method should do one thing, and do it well. The name of the method should tell you what task it does.
  - We should avoid copying code between methods.
    - Instead we should take pieces of common code and put it into a new method called by all methods using that code.
    - We call these helper methods.
    - Any time you are rewriting the same code over and over again, it makes sense to put it in a helper method. That way you can write (and debug) that code once, and use it many times.

38

- A method should have one clear and do-able goal.
  - To achieve that goal, it can (and should) call other methods as needed. If you find that your method is doing many different things, break it up into more than one method.
- A method should be in the class that contains the data it is working with, unless there is a good reason to put it somewhere else.

39

Writing Good Methods

- The last few methods we wrote were *NOT* general.\
- For example, we have to modify copyPictureOffset () for each place that we might want to put the object, and we need to make a new copy of the method for each new picture we want to copy (even the name is specific to the task the method executes in the current implementation).
- Instead, we can give the method parameters, which tell it the picture to copy from, as well as the start and end positions for the source and target images.
- To make our methods apply to all images, in this case we can basically replace our hard-coded values with parameters.

40

- An algorithm for copying pixels from a passed source picture that is more general might look like this:
- Giving a start x and y and end x and y for the source picture
  - If the start x and y and end x and y cover the entire picture then the whole picture will be copied
  - If the start x and y and end x and y are part of the picture then cropping will occur
- Copy the defined region of the source picture to the current picture object with a target start x and target start y
  - If the start x and y are 0 then it copies to the upper left corner
- Loop through the x values between xStart and xEnd
  - Loop through the y values between yStart and yStart
    - Get the pixel from the source picture for the current x and y values
      - Get the pixel from the target picture for the targetStartX + x and targetStartY + y values
      - Set the color in the target pixel to the color in the source pixel

41

```
public void copy(Picture sourcePicture, int startX, int startY,
                 int endX, int endY, int targetStartX, int targetStartY) {
    // loop through the x values
    for (int x = startX, tx = targetStartX; x < endX; x++, tx++) {
      // loop through the y values
      for (int y = startY, ty = targetStartY; y < endY; y++, ty++) {
        // copy the source color to the target color
        Pixel sourcePixel = sourcePicture.getPixel(x, y);
        Pixel targetPixel = this.getPixel(tx, ty);
        targetPixel.setColor(sourcePixel.getColor());
      }
```

42

- Now if we want to copy a whole image or any part of an image to another image, all we have to do is call copy.
- For example, if we have a picture target and a picture source, and we want to copy the rectangle between (0,0) and (70,70) of source to (375,350) of target, we would do target.copy(source, 0,0,70,70,375,350);

43

Summary

- We revisited the topic of two-dimensional arrays, which have columns and rows that can be indexed separately to specify any x,y location. In images, this is done with getPixel(x,y).
- we introduced nested loops, and showed how they can be used to work with two-dimensional arrays.
  - One loop is used for the x direction and the other loop for the y direction. When they are nested, the inner loop will do a complete set of iterations once for each iteration of the outer loop. In a sense the inner loop is running very quickly over and over again, while the outer loop is waiting patiently for the inner loop at each step.

44

Summary

- We 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, ...; condition; var1++, var2++, ...*)
- To copy one part of a picture to another, we have to control the start and end X and Y values in our source picture as well as the target X and Y values in the target image.
- A method should aim to be as generic as possible, and reduce the amount of redundant code and code copied from other methods. At the same time, it has one clear and do-able goal.

45

Summary

- We can use parameters to make methods reusable.
- If you're copying lots of code from one method to another, *you're doing it wrong*. Instead, create a *helper* method to implement common behavior (like the copy method we made here), and call it from other methods that do more specific things.
- A method should be in the class that contains the data it is working with, unless there is a good reason to put it somewhere else.

46