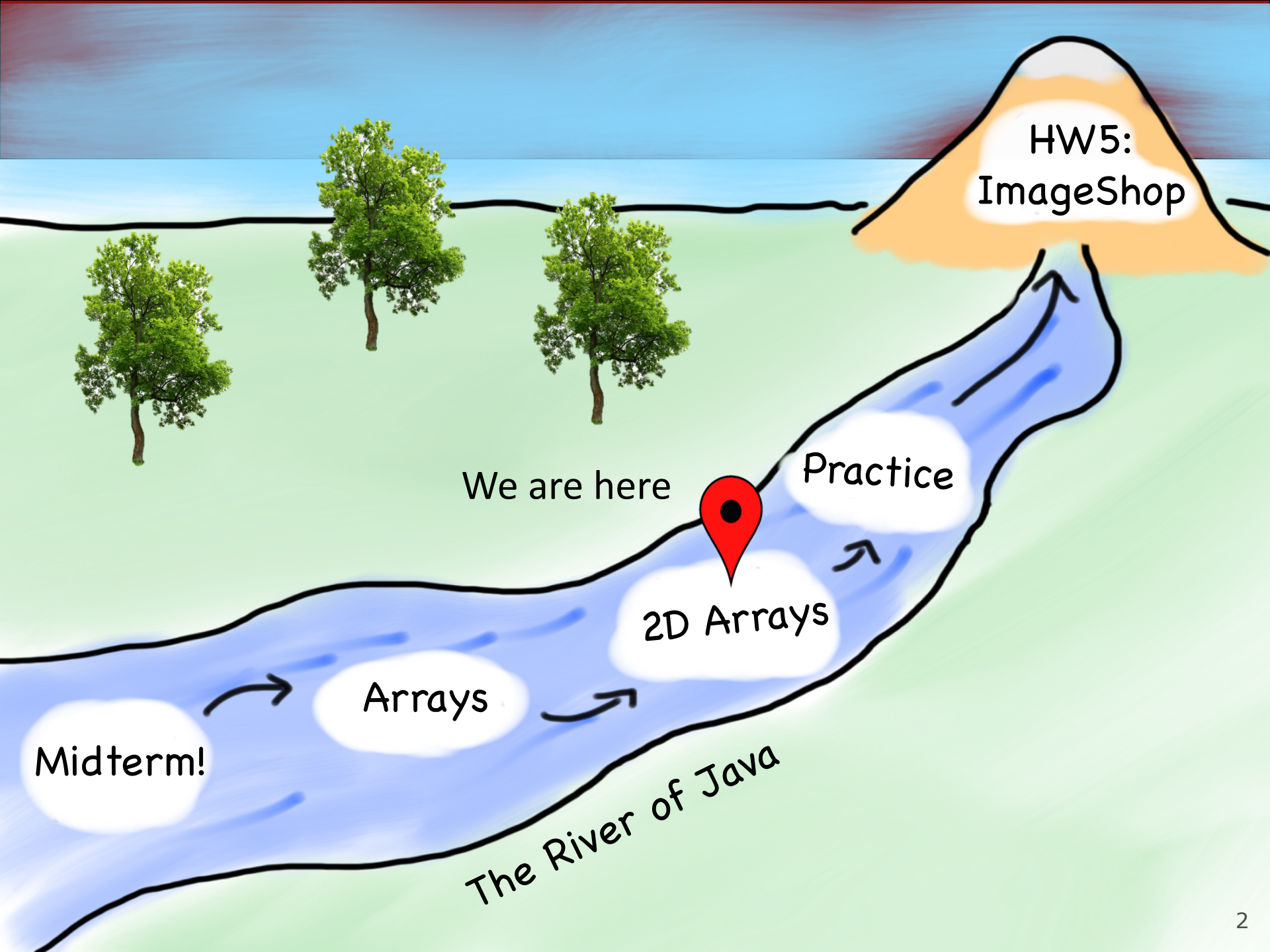
CS 106A, Lecture 17
2D Arrays and Images

suggested reading:

Java Ch. 11.6-11.7


HW5:
ImageShop
We are here
Practice
2D Arrays
Arrays
Midterm!
The River of Java

Plan for Today

- Recap: Arrays
- 2D Arrays
- Images as 2D Arrays
- Modifying Images
- Practice: Brighten, Grayscale

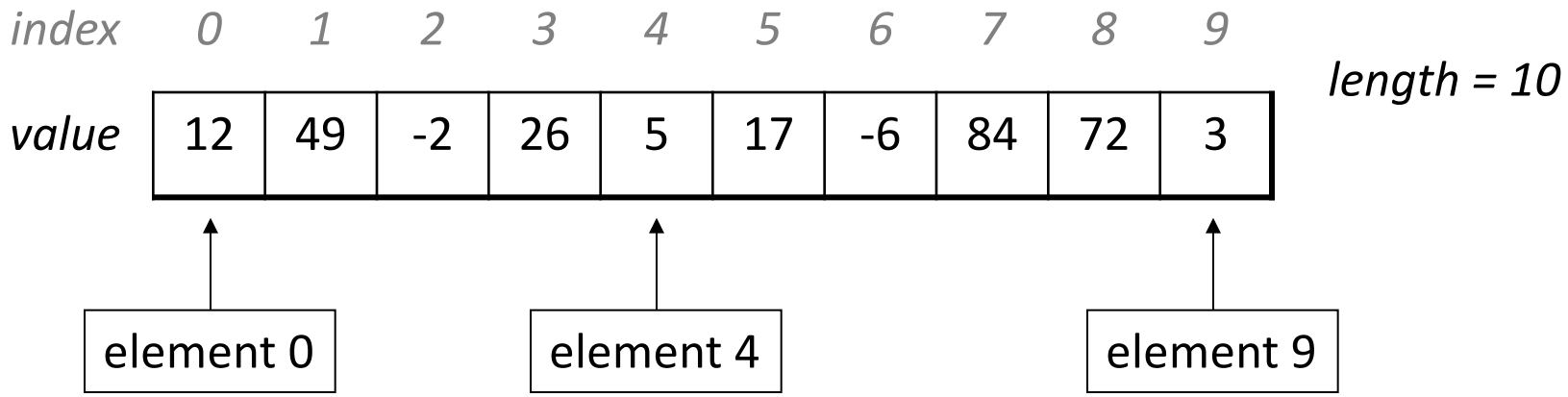
Plan for Today

- Recap: Arrays
- 2D Arrays
- Images as 2D Arrays
- Modifying Images
- Practice: Brighten, Grayscale

From Last Time: Arrays

A new variable type that is an **object** that represents an ordered, homogeneous list of data.

– Arrays have many *elements* that you can access using *indices*



Data Structures

Operation	Strings	Arrays
Make a new one	`String str = "abc";`	
Get length?	`str.length()`	
Get element?	`str.charAt(i)`	
Set element?	*Not allowed*	
Loop?	`for(int i = 0; i < str.length(); i++)`	

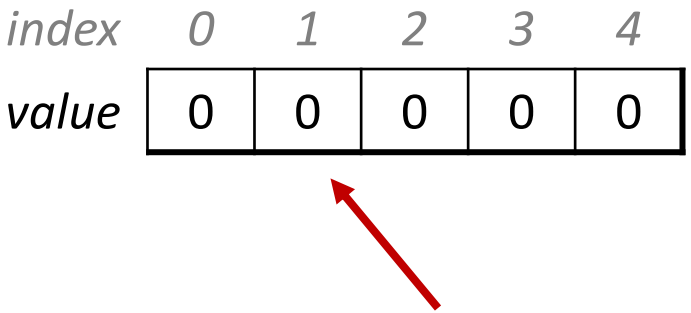
Data Structures

Operation	Strings	Arrays
Make a new one	`String str = "abc";`	`int arr = new int[5];`
Get length?	`str.length()`	`arr.length`
Get element?	`str.charAt(i)`	`arr[i]`
Set element?	*Not allowed*	`arr[i] = 5;`
Loop?	`for(int i = 0; i < str.length(); i++)`	`for(int i = 0; i < arr.length; i++)`

Creating an Array

```
type[] name = new type[length];
```

```
int[] numbers = new int[5];
```



Java initializes each element of a new array to its *default value*, which is **0** for `int` and `double`, `'\0'` for `char`, **`false`** for `boolean`, and **`null`** for objects.

Creating an Array

Sometimes, we want to hardcode the elements of an array.

Luckily, Java has a special syntax for initializing arrays to hardcoded numbers.

```
type[] name = { elements };
```

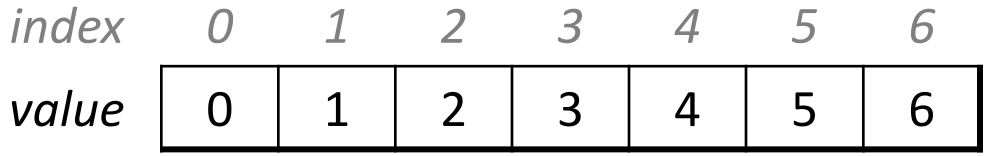
```
// Java infers the array length
int[] numbers = {5, 32, 12, 2, 1, -1, 9};
```

Accessing Data In An Array

```
name[index]    // get element at index
```

- Like Strings, indices go from **0** to the **array's length - 1**.

```
for (int i = 0; i < 7; i++) {
    println(numbers[i]);
}
println(numbers[9];    // exception
println(numbers[-1]);  // exception
```

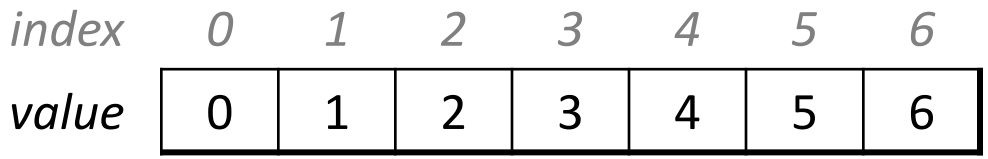


Putting Data In An Array

```
name[index] = value;    // set element at index
```

- Like Strings, indices go from **0** to the **array's length - 1**.

```
int[] numbers = new int[7];
for (int i = 0; i < 7; i++) {
    numbers[i] = i;
}
numbers[8] = 2;    // exception
numbers[-1] = 5;   // exception
```



Array Length

Similar to a String, you can get the length of an array by saying

```
myArray.length
```

Note that there are *no parentheses* at the end!

Arrays + For Loops = ❤️

Just like with Strings, we can use an array's length, along with its indices, to perform cool operations.

For instance, we can read in numbers from the user:

```
int length = readInt("# of numbers? ");
int[] numbers = new int[length];
for (int i = 0; i < numbers.length; i++) {
    numbers[i] = readInt("Elem " + i + ": ");
}
```

Arrays + For Loops = ❤️

Just like with Strings, we can use an array's length, along with its indices, to perform cool operations.

For instance, we can *sum up* all of an array's elements.

```
int sum = 0;
for (int i = 0; i < numbers.length; i++) {
     sum += numbers[i];
}
println(sum);
```

Limitations of Arrays

- An array's length is **fixed**. You cannot resize an existing array:

```
int[] a = new int[4];
a.length = 10;                          // error
```

- You cannot compare arrays with == or `equals` :

```
int[] a1 = {42, -7, 1, 15};
int[] a2 = {42, -7, 1, 15};
if (a1 == a2) {  ... }                  // false!
if (a1.equals(a2)) {  ... }             // false!
```

- An array does not know how to print itself:

```
println(a1);                          // [I@98f8c4]
```

Arrays Methods To The Rescue!

- The class `Arrays` in package `java.util` has useful methods for manipulating arrays:

Method name	Description
`Arrays.binarySearch(`***`array, value`***`)`	returns the index of the given value in a *sorted* array (or < 0 if not found)
`Arrays.copyOf(`***`array, length`***`)`	returns a new copy of array of given length
`Arrays.equals(`***`array1, array2`***`)`	returns `true` if the two arrays contain same elements in the same order
`Arrays.fill(`***`array, value`***`);`	sets every element to the given value
`Arrays.sort(`***`array`***`);`	arranges the elements into sorted order
`Arrays.toString(`***`array`***`)`	returns a string representing the array, such as `"[10, 30, -25, 17]"`

Plan for Today

- Recap: Arrays
- 2D Arrays
- Images as 2D Arrays
- Modifying Images
- Practice: Brighten, Grayscale, Shrink

The Matrix

Image used under “fair use” for educational purposes.
Source: https://www.themarysue.com/decoding-the-transgender-matrix-the-matrix-as-a-transgender-coming-out-story/

2D Arrays ("Matrices")

2D Arrays

type`[][] ***name*** = new ***type***[***rows***][***columns***];`

```
int[][] a = new int[3][5];
```

	0	*1*	*2*	*3*	*4*
0	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
1	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
2	a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]

Manipulating 2D Arrays

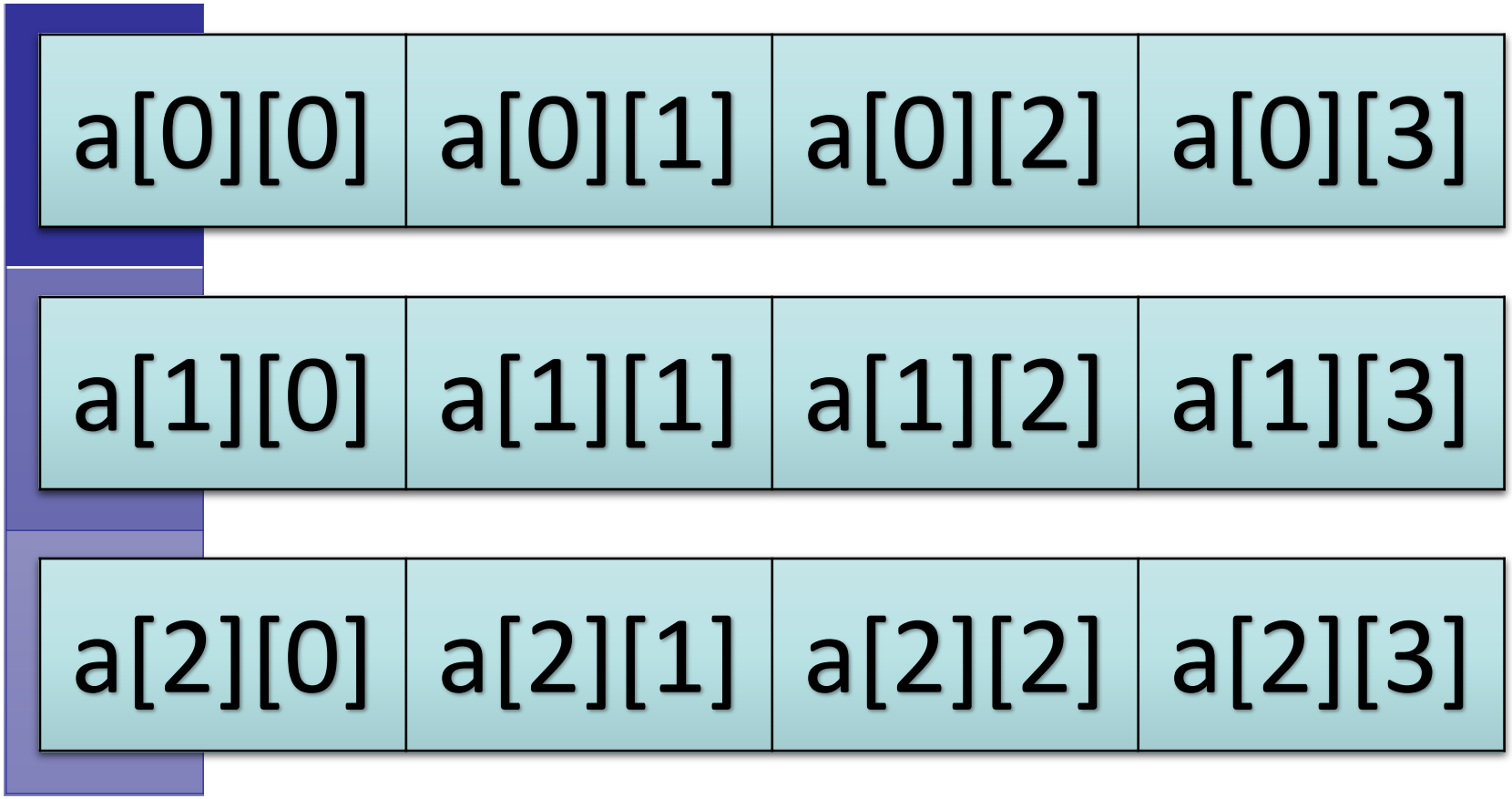
```
name[row][col]   // get element at row,col

name[row][col] = value;   // set element at row,col
```

2D arrays are arrays *of arrays*!

2D Arrays = Arrays of Arrays!

```
int[][] a = new int[3][4];
```

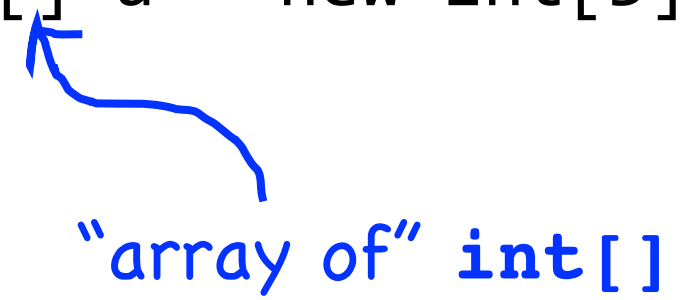


2D Arrays = Arrays of Arrays!

A 2D array is an array where every element is *itself* an array.

```
int[] a = new int[3];
```


```
int[][] a = new int[3][4];
```



2D Arrays = Arrays of Arrays!

A 2D array is an array where every element is *itself* an array.

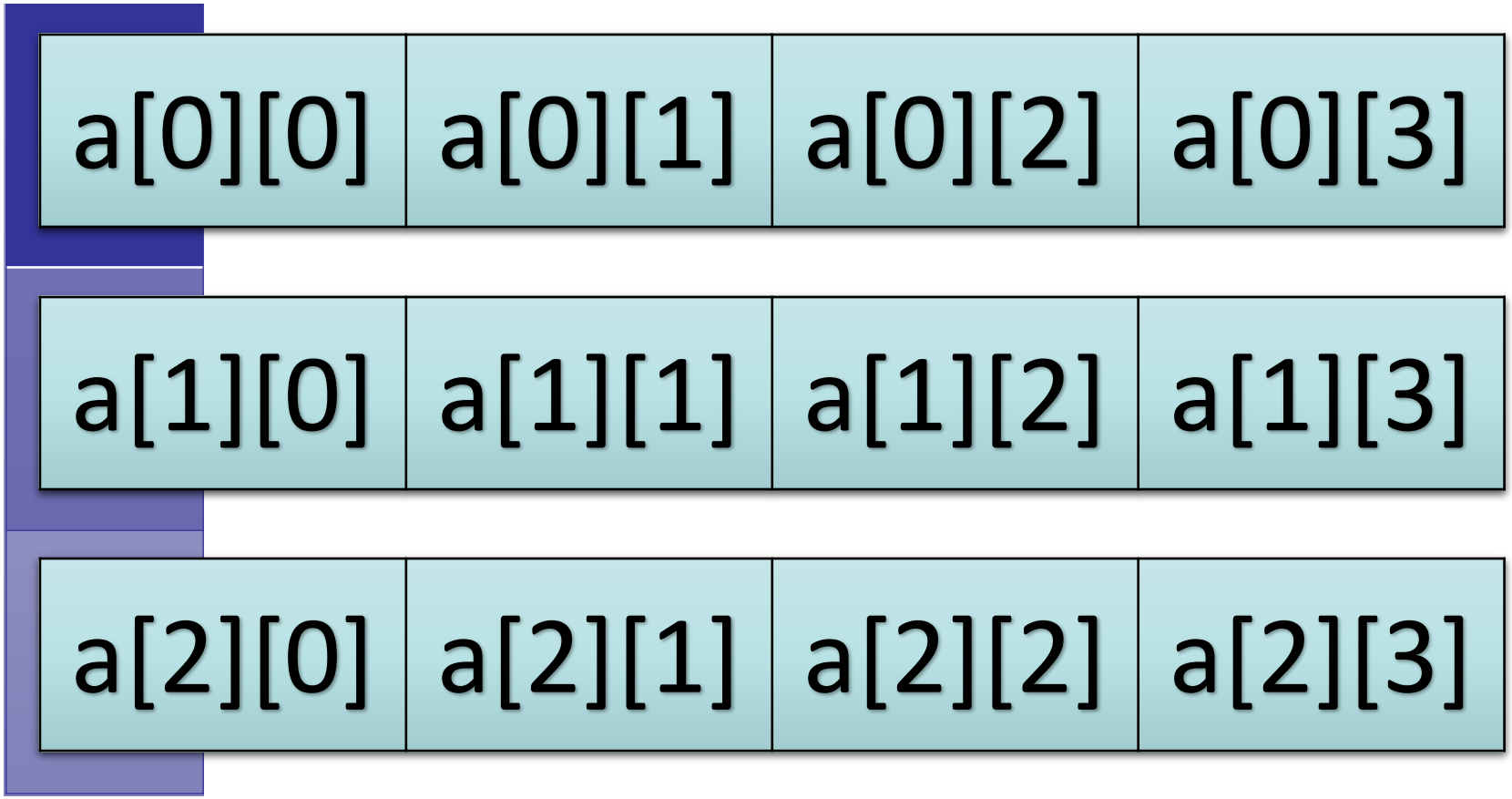
```
int[][] a = new int[3][4];
...
int x = a[1][1]; // int at position (1, 1)

int[] firstRow = a[0]; // 1D array!

// NOTE: no way to get a single column ☹
```

2D Array Dimensions

How do we get the number of rows of a 2D array using the **length** property? How about the number of columns?



2D Array Dimensions

You can get the number of rows in a 2D array by saying:

```
arr.length    // # rows in our 2D array
```

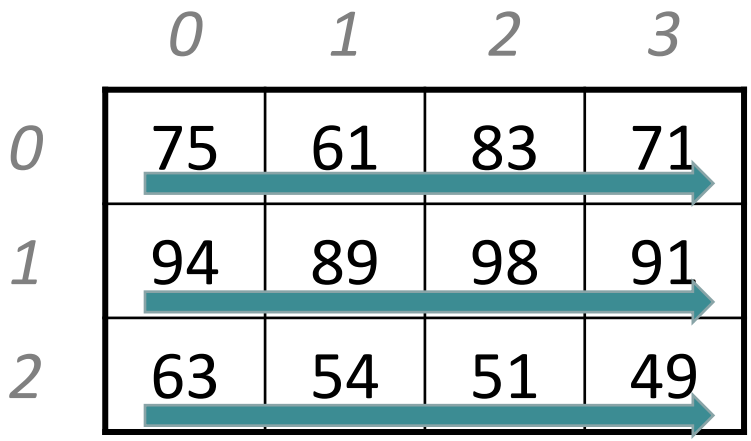
Since the length of each row is the same, you can get the number of columns in a 2D array by saying:

```
arr[0].length  // # entries in row 0 ( = # cols)
```

2D Arrays + For Loops = ❤️

We can use **double for-loops** to loop over each row, and then each column, in a 2D array.

```
int[][] arr = ...
for (int row = 0; row < arr.length; row++) {
    for (int col = 0; col < arr[0].length; col++) {
        // do something with arr[row][col];
    }
}
```



"row-major" order

Practice: 2D Arrays

- **Q:** What is the array state after the code below?

```
int[] a = new int[4][3];
...   // fill with data at right

for (int r = 1; r < 4; r++) {
    for (int c = 0; c < 3; c++) {
        a[r][c] += a[r - 1][c];
    }
}
```

r\c	0	1	2
0	1	2	3
1	1	2	3
2	1	2	3
3	1	2	3

```
//      A.                B.                C.                D.
```

A.

	0	1	2
0	1	3	5
1	1	3	5
2	1	3	5
3	1	3	5

B.

	0	1	2
0	1	3	6
1	1	3	6
2	1	3	6
3	1	3	6

C.

	0	1	2
0	1	2	3
1	2	4	6
2	2	4	6
3	2	4	6

D.

	0	1	2
0	1	2	3
1	2	4	6
2	3	6	9
3	4	8	12

Limitations of 2D Arrays

- Unlike 1D arrays, you *cannot compare 2D arrays with* **`Arrays.equals`**. You must use **`Arrays.deepEquals`**.

```
int[][] a1 = ...
int[][] a2 = ...
if (Arrays.deepEquals(a1, a2)) {  ... }
```

- A 2D array does not know how to print itself:

```
int[][] a = new int[rows][cols];
println(a);                          // [[I@8cf420
println(Arrays.toString(a));         // [[I@6b3f44,[I@32c2a8]...

// [[0, 1, 2, 3, 4], [1, 2, ...
println(Arrays.deepToString(a));
```

Summary: 2D Arrays

- Make a new 2D array

```
type[][] name = new type[rows][columns];
```

- Get and set values using bracket notation

```
name[row][col]          // get elem at row,col
name[row][col] = value; // set elem at row,col
```

- Get the number of rows and columns

```
arr.length    // # rows
arr[0].length    // # columns
```

- Iterate over a 2D array using a double for-loop

```
for (int row = 0; row < arr.length; row++) {
    for (int col = 0; col < arr[0].length; col++) {
        // do something with arr[row][col];
    }
}
```

Plan for Today

- Recap: Arrays
- 2D Arrays
- Images as 2D Arrays
- Modifying Images
- Practice: Brighten, Grayscale

Images

Images are just grids (2D arrays!) of pixels! Pixels are just integer values from 0-255.

Images as 2D Arrays

We can get a GImage as a 2D array of pixels.

```
GImage img = new GImage("res/snowman.jpg");
int[][] pixels = img.getPixelArray();
int pixel = pixels[0][0];  // top-left pixel
```

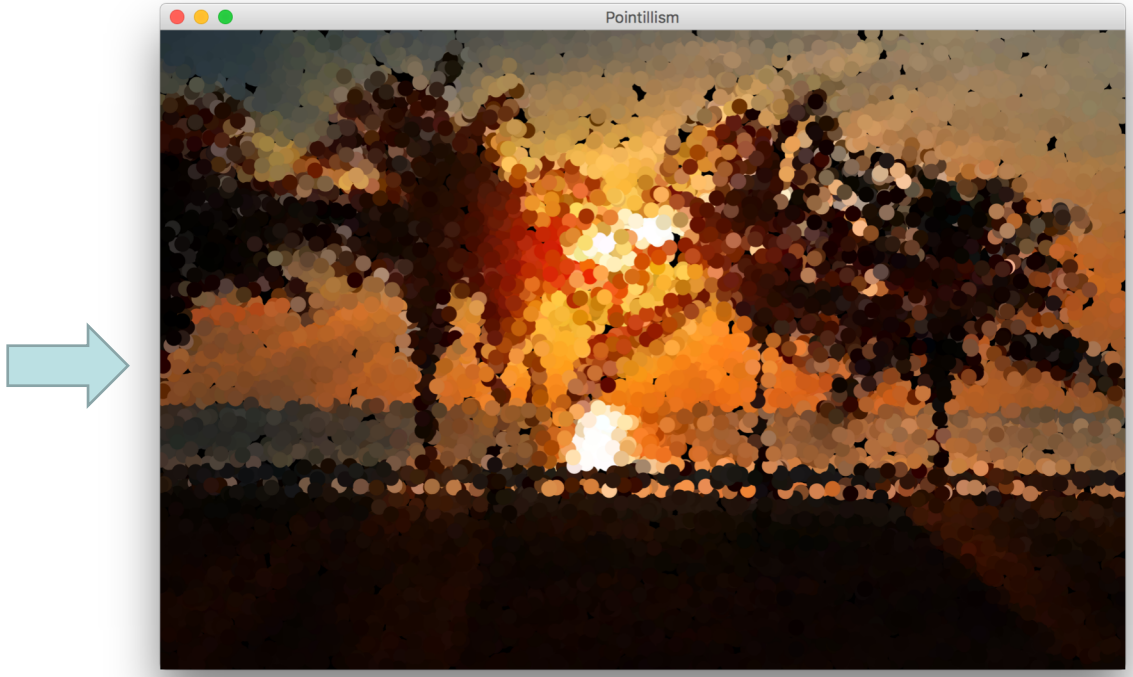
Example: Pointillism

Pointillism is an art style where many small dots of color are combined to make a larger image.

A Sunday on La Grande Jatte, Georges Seurat

Example: Pointillism

Pointillism is an art style where many small dots of color are combined to make a larger image.



Example: Pointillism

Pointillism is an art style where many small dots of color are combined to make a larger image.

Example: Pointillism

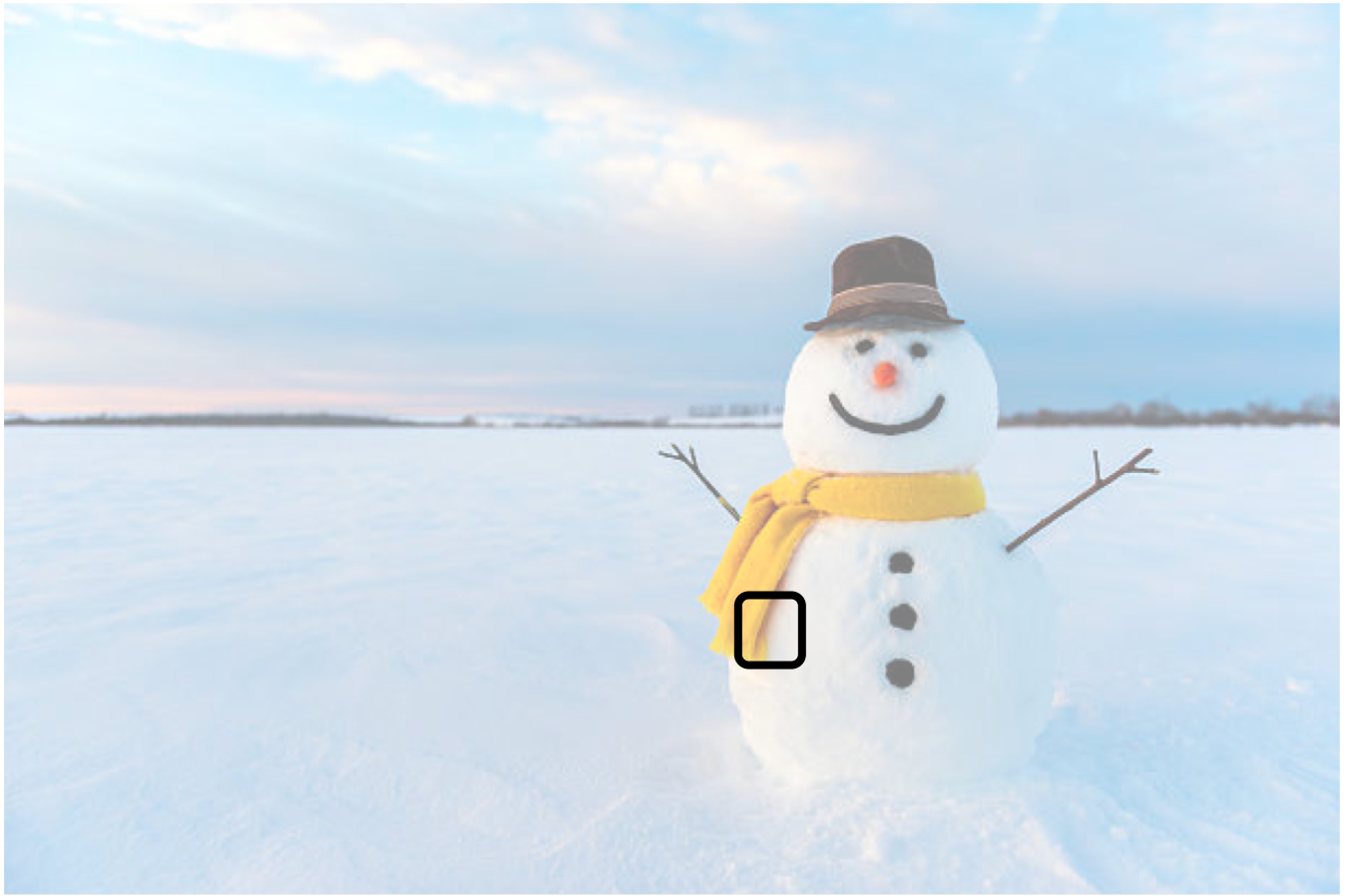
Pointillism is an art style where many small dots of color are combined to make a larger image.

```
Repeat many times:
  1. Pick a random pixel from an image
  2. Find the pixel's color
  3. "Paint" a large brush stroke of that color in
     the corresponding location
```

Example: Pointillism

Example: Pointillism

`c = 46`



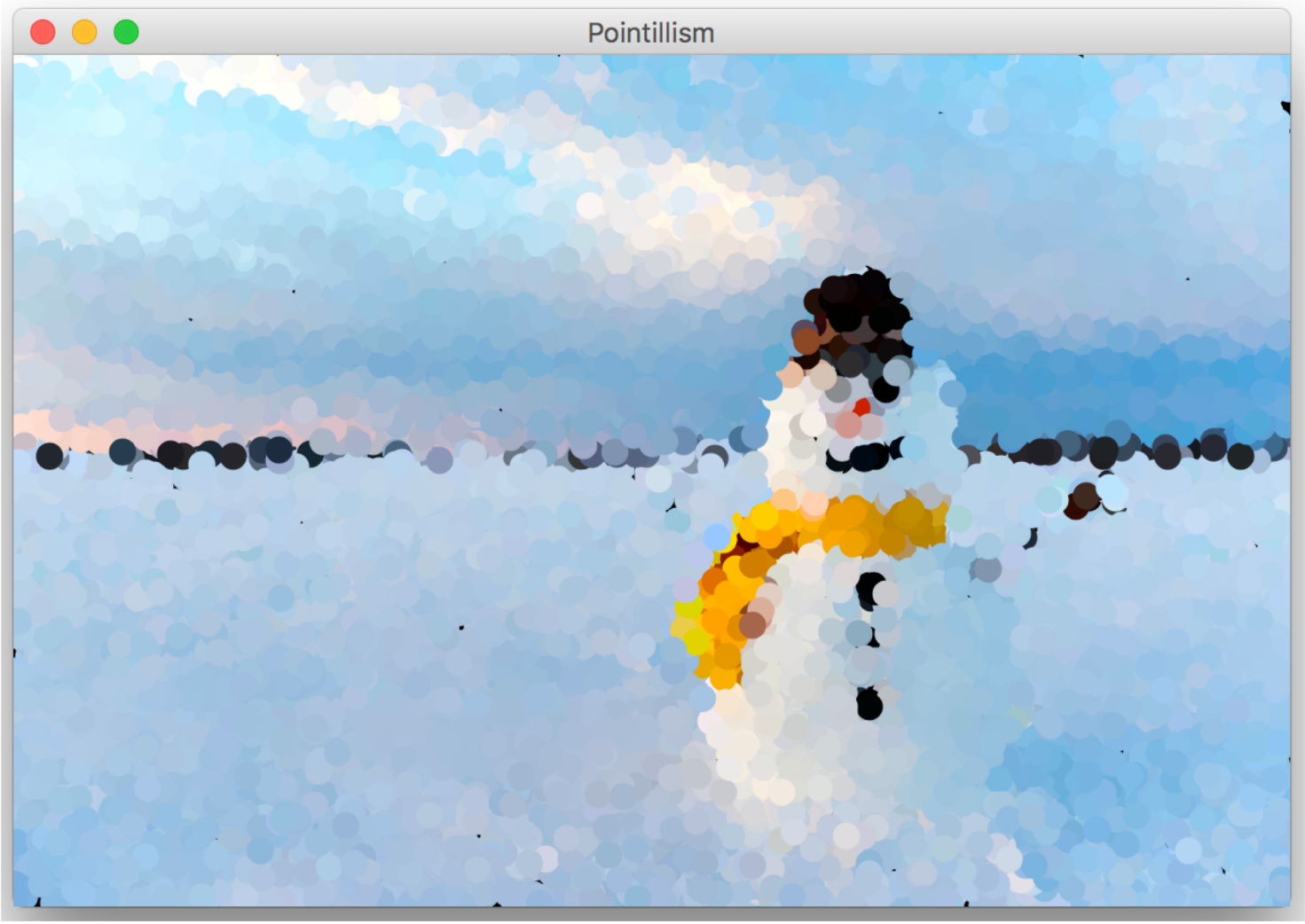
`r = 36`

Example: Pointillism

```
c = 46
```

```
r = 36
```

Example: Pointillism



Example: Pointillism

```
GImage snowman = new GImage("res/snowman.jpg");
int[][] pixels = snowman.getPixelArray();
int rows = pixels.length;
int cols = pixels[0].length;

for (int i = 0; i < NUM_SAMPLES; i++) {
  int c = RandomGenerator.getInstance().nextInt(cols);
  int r = RandomGenerator.getInstance().nextInt(rows);
  int pixel = pixels[r][c];
  Color color = new Color(pixel);
  addColoredCircle(r, c, color);
}
```

Example: Pointillism

```
GImage snowman = new GImage("res/snowman.jpg");
int[][] pixels = snowman.getPixelArray();
int rows = pixels.length;
int cols = pixels[0].length;

for (int i = 0; i < NUM_SAMPLES; i++) {
  int c = RandomGenerator.getInstance().nextInt(cols);
  int r = RandomGenerator.getInstance().nextInt(rows);
  int pixel = pixels[r][c];
  Color color = new Color(pixel);
  addColoredCircle(r, c, color);
}
```

Example: Pointillism

```
GImage daisy = new GImage("res/daisy.jpg");
int[][] pixels = daisy.getPixelArray();
int rows = pixels.length;
int cols = pixels[0].length;

for (int i = 0; i < NUM_SAMPLES; i++) {
  int c = RandomGenerator.getInstance().nextInt(cols);
  int r = RandomGenerator.getInstance().nextInt(rows);
  int pixel = pixels[r][c];
  Color color = new Color(pixel);
  addColoredCircle(r, c, color);
}
```

Example: Pointillism

```
GImage daisy = new GImage("res/daisy.jpg");
int[][] pixels = daisy.getPixelArray();
int rows = pixels.length;
int cols = pixels[0].length;

for (int i = 0; i < NUM_SAMPLES; i++) {
  int c = RandomGenerator.getInstance().nextInt(cols);
  int r = RandomGenerator.getInstance().nextInt(rows);
  int pixel = pixels[r][c];
  Color color = new Color(pixel);
  addColoredCircle(r, c, color);
}
```

Example: Pointillism

```
GImage daisy = new GImage("res/daisy.jpg");
int[][] pixels = daisy.getPixelArray();
int rows = pixels.length;
int cols = pixels[0].length;

for (int i = 0; i < NUM_SAMPLES; i++) {
  int c = RandomGenerator.getInstance().nextInt(cols);
  int r = RandomGenerator.getInstance().nextInt(rows);
  int pixel = pixels[r][c];
  Color color = new Color(pixel);
  addColoredCircle(r, c, color);
}
```

Example: Pointillism

```
GImage daisy = new GImage("res/daisy.jpg");
int[][] pixels = daisy.getPixelArray();
int rows = pixels.length;
int cols = pixels[0].length;

for (int i = 0; i < NUM_SAMPLES; i++) {
  int c = RandomGenerator.getInstance().nextInt(cols);
  int r = RandomGenerator.getInstance().nextInt(rows);
  int pixel = pixels[r][c];
  Color color = new Color(pixel);
  addColoredCircle(r, c, color);
}
```

Example: Pointillism

```
GImage daisy = new GImage("res/daisy.jpg");
int[][] pixels = daisy.getPixelArray();
int rows = pixels.length;
int cols = pixels[0].length;

for (int i = 0; i < NUM_SAMPLES; i++) {
  int c = RandomGenerator.getInstance().nextInt(cols);
  int r = RandomGenerator.getInstance().nextInt(rows);
  int pixel = pixels[r][c];
  Color color = new Color(pixel);
  addColoredCircle(r, c, color);
}
```

Example: Pointillism

```
// Assume canvas is exactly image size
private void addColoredCircle(int r, int col, Color c) {
  double size = CIRCLE_RADIUS * 2;
  GOval circle = new GOval(size, size);
  circle.setFilled(true);
  circle.setColor(c);
  add(oval, col - CIRCLE_RADIUS, r – CIRCLE_RADIUS);
}
```

Plan for Today

- Recap: Arrays
- 2D Arrays
- Images as 2D Arrays
- Modifying Images
- Practice: Brighten, Grayscale

Red, Green and Blue in one int?

Images *encode* the R, G, and B values of a pixel into a single integer between 0 and 255. You can convert between this **pixel value** and the individual **RGB values**.

```
int[][] pixels = image.getPixelArray();
int px = pixels[0][0];
int red = GImage.getRed(px);
int green = GImage.getGreen(px);
int blue = GImage.getBlue(px);
```

Creating New Pixels

Images *encode* the R, G, and B values of a pixel into a single integer between 0 and 255. You can convert between this **pixel value** and the individual **RGB values**.

You can also create pixels with your own RGB values.

```
int r = ...
int g = ...
int b = ...
int pixel = GImage.createRGBPixel(r, g, b);
```

Images as 2D Arrays

We can get a GImage as a 2D array of pixels, and modify it any way we want. Then, we can create a new GImage with the modified pixels.

```
GImage img = new GImage("res/snowman.jpg");
int[][] pixels = img.getPixelArray();
... // (modify pixels)
img.setPixelArray(pixels);  // update image

// or make a new GImage
GImage newImg = new GImage(pixels);
```

Modifying Image Pixels

- There are many cool image algorithms based around modifying individual pixels in an image: grayscale, brighten, normalize, remove red-eye…

grayscale

zoom

GImage Pixel Methods

```
GImage img = new GImage("res/snowman.jpg");
```

Method name	Description
`img.getPixelArray()`	returns pixels as 2D array of ints, where each int in the array contains all 3 of Red, Green, and Blue merged into a single integer
`img.setPixelArray(array);`	updates pixels using the given 2D array of ints
`GImage.createRGBPixel(r, g, b)`	returns an int that merges the given amounts of red, green and blue (each 0-255)

`GImage.getRed(px)`
`GImage.getGreen(px)`
`GImage.getBlue(px)`
	returns the redness, greenness, or blueness of the given pixel as an integer from 0-255

Recap: Modifying Pixels

- **Extract** pixel RGB colors with `GImage.getRed/Blue/Green`.

```
int red   = GImage.getRed(pixels[0][0]);    // 0-255
int green = GImage.getGreen(pixels[0][0]);  // 0-255
int blue  = GImage.getBlue(pixels[0][0]);   // 0-255
```

- **Modify** the color components for a given pixel.

```
red = 0;   // remove redness
```

- **Combine** the RGB back together into a single `int`.

```
pixels[0][0] = GImage.createRGBPixel(red, green, blue);
```

- **Update** the image with your modified pixels when finished.

```
image.setPixelArray(pixels);
```

Changing Image Size

- Destination image is same size → often modify array in place.
- Destination image is different size → need a new array.

- Example: **Half the size** of an image.

```
int[][] pixels = img.getPixelArray();
int[][] smaller = new int[pixels.length / 2]
                          [pixels[0].length / 2];
...
// set to be the pixels of 'smaller'
img.setPixelArray(smaller);
```

Plan for Today

- Recap: Arrays
- 2D Arrays
- Images as 2D Arrays
- Modifying Images
- Practice: Brighten, Grayscale

Recap

- Recap: Arrays
- 2D Arrays
- Images as 2D Arrays
- Modifying Images
- Practice: Brighten, Grayscale

Next time: more practice with arrays