CS 106A, Lecture 16
Multi-dimensional Arrays

reading:

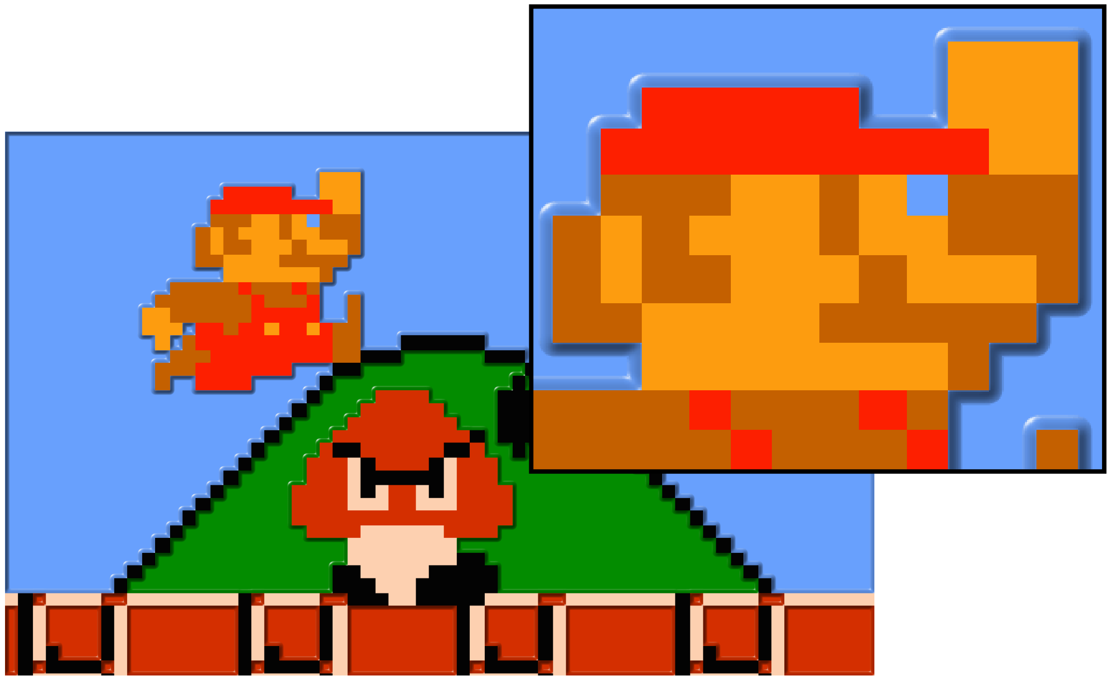
Art & Science of Java, 12.4

Lecture Outline

- Today we will learn about **multi-dimensional arrays**.
  - Motivating example: 2D arrays of **pixels** to manipulate **images**.

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	1	7	10	12	8	14	22

	0	*1*	*2*	*3*
0	75	61	83	71
1	94	89	98	91
2	63	54	51	49



2-D arrays

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

- You can create multidimensional arrays to represent multidimensional data.

```
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]

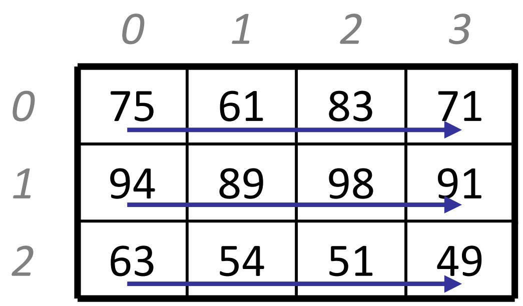
Looping over 2D array

array2dMystery1

- Loop over a 2D array's elements using nested `for` loops.

- Row-major order:

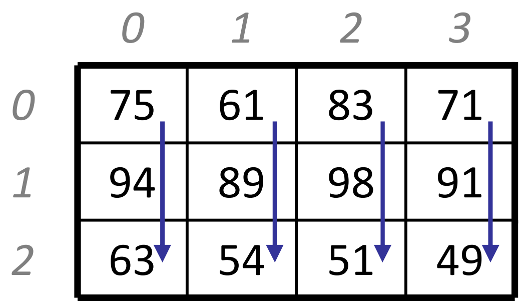
```
for (int r = 0; r < a.length; r++) {
    for (int c = 0; c < a[r].length; c++) {
        do something with a[r][c];
    }
}
```



	0	1	2	3
0	75	61	83	71
1	94	89	98	91
2	63	54	51	49

- Column-major order:

```
for (int c = 0; c < a[0].length; c++) {
    for (int r = 0; r < a.length; r++) {
        do something with a[r][c];
    }
}
```



	0	1	2	3
0	75	61	83	71
1	94	89	98	91
2	63	54	51	49

What's the array state?

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

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

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

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

Printing a 2D array

- The typical ways of printing don't work on a 2D array:

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

- Instead, use the special **deepToString** method to print it:

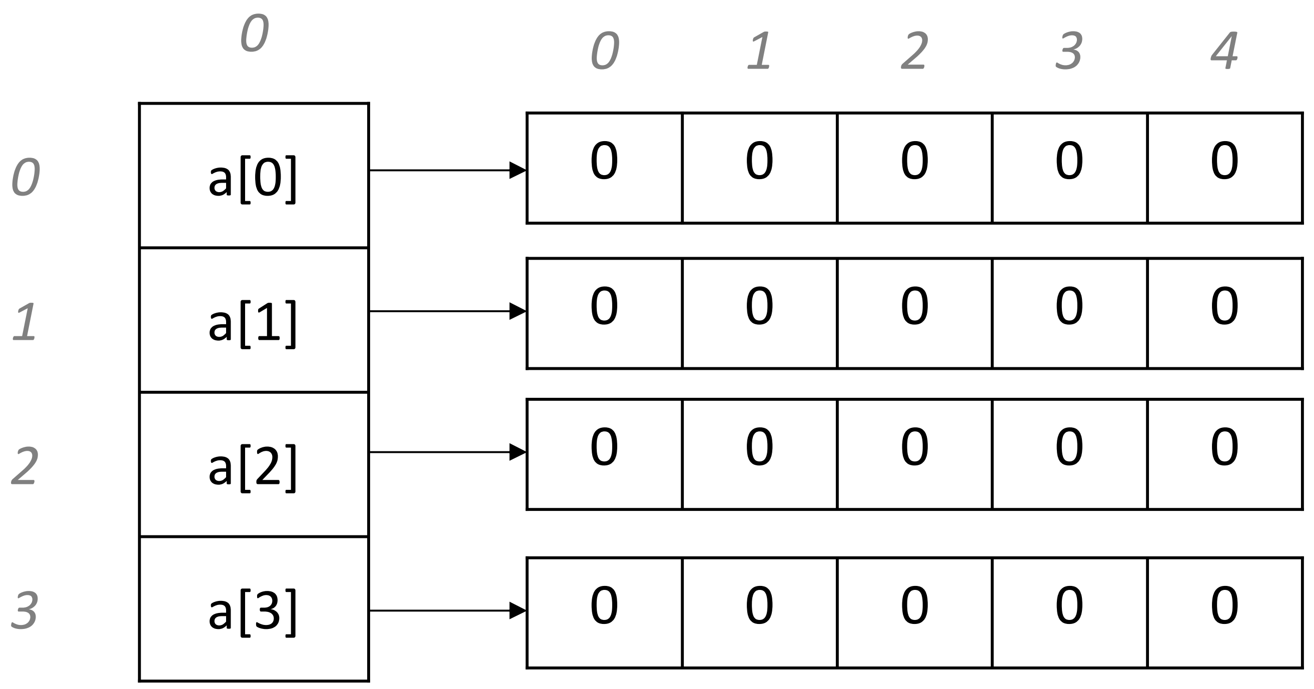
```
println(Arrays.deepToString(a));
// [[0, 1, 2, 3, 4], [1, 2, 3, ...
```

	0	*1*	*2*	*3*	*4*
0	0	1	2	3	4
1	1	2	3	4	5
2	2	3	4	5	6
3	3	4	5	6	7

Reasoning in 2D

- There are two main ways of intuiting a multidimensional array.
  - *2D grid or matrix:* `a[r][c]` is the grid element at position (r, c).
  - *Array of arrays:* Each `a[r]` is a one-dimensional array.

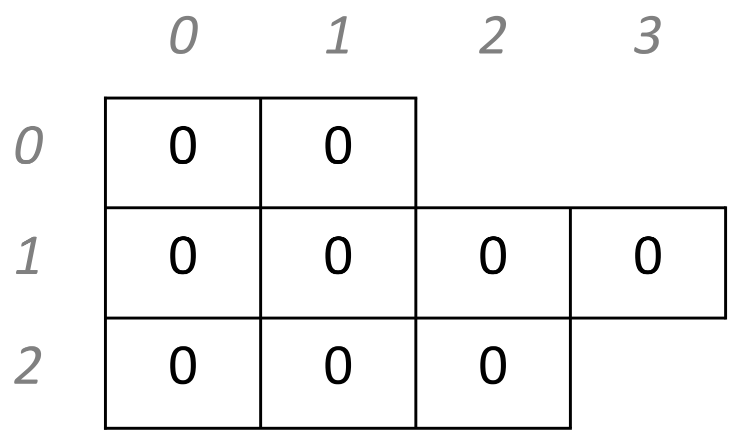
```
int[] a = new int[4][5];
```



Jagged 2D arrays

- The rows of a jagged array don't need to be the same length:

```
int[][] jagged = new int[3][];
jagged[0] = new int[2];
jagged[1] = new int[4];
jagged[2] = new int[3];
```



Jagged array example

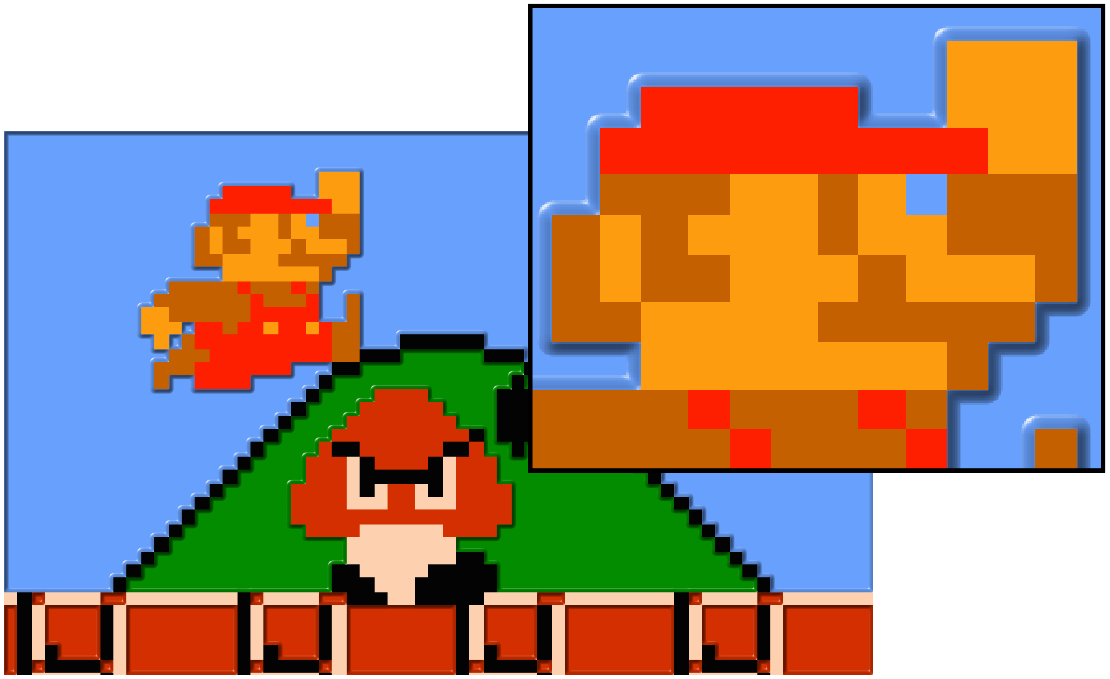
- This array represents **Pascal's Triangle**, the binomial coefficients:

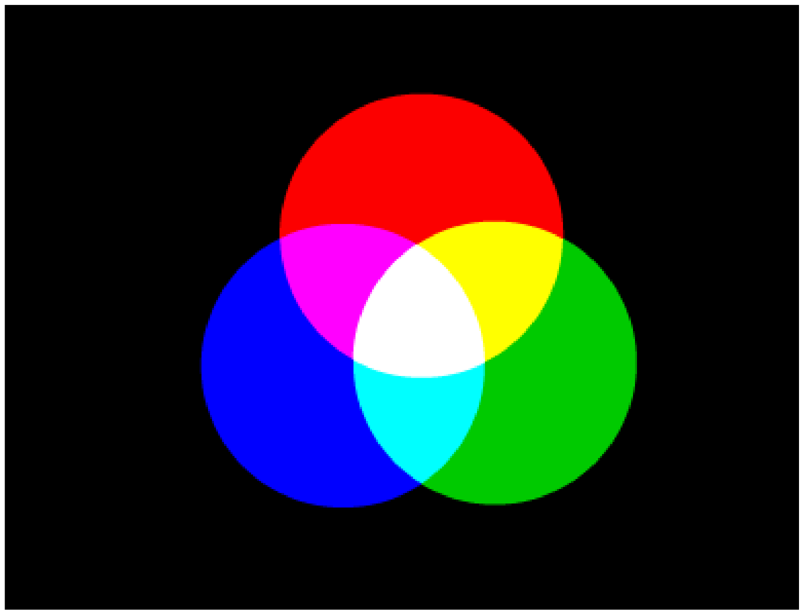
```
int[][] triangle = new int[6][];
for (int i = 0; i < triangle.length; i++) {
    triangle[i] = new int[i + 1];
    triangle[i][0] = 1;
    triangle[i][i] = 1;
    for (int j = 1; j < i; j++) {
        triangle[i][j] = triangle[i - 1][j - 1]
                       + triangle[i - 1][j];
    }
}
```

	0	*1*	*2*	*3*	*4*	*5*
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

Image as 2D array

- Images are typically made up of small dots called *pixels* (picture elements).
- Computers usually represent color as RGB triplets:
  - Values range from 0 (min) to 255 (max), inclusive.





http://www.videogameobsession.com/videogame/wallpapers/VGO-NES-SuperMario_03-vgo.png

GImage pixel methods

- A `GImage` object displays an image file on the screen.

```
GImage img = new GImage("res/mario.png");
```

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
	
	(extracts the given byte of data from the given 4-byte integer value)

GImage and 2D arrays

- You can extract an **array of pixels** from a GImage by calling

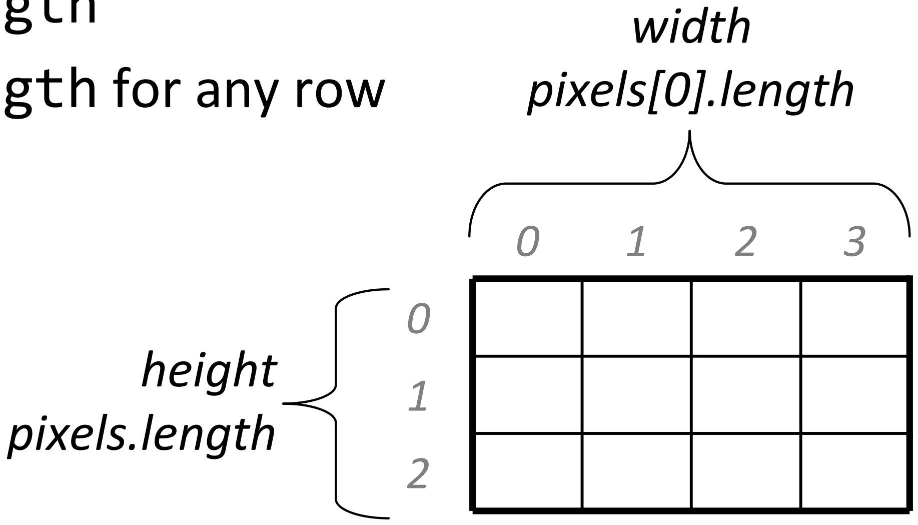
```
int[][] pixels = image.getPixelArray();
```

  - Each pixel is a single **`int`** containing red, green, and blue color info.

```
int px = pixels[0][0];    // top/left pixel
```

  - first dimension = *row* (*y*), **second dimension** = column (*x*).
    - height of image: `pixels.length`
    - width of image: `pixels[0].length`

      or: `pixels[r].length` for any row



GImage pixels

```
int[][] pixels = image.getPixelArray();
```

- Each pixel is a single `int` containing red, green, and blue color info.

```
int px = pixels[0][0];    // top/left pixel
```

R,G,B in one int?

- Each `int` in the `int[][]` of pixels stores an entire color, consisting of a red, green, and blue component from 0-255.
  - Technically it also stores an *alpha* (opacity) value, usually set to 255.
  - How does a single `int` store 4 integer values inside it??

- A Java `int` can store any unique integer from roughly 0 .. +/- 2^{31}.
  - An `int` consists of 32 *bits* or 4 *bytes* of data.
  - Each byte is 8 bits and can store an integer from 0 - 255.
  - In our library, byte 0 = alpha; 1 = red; 2 = green; 3 = blue.

- Contents of RGB pixel representing (R=227, G=14, B=139):

  int px = (**255** * 256*256*256) + (**227** * 256*256) + (**14** * 256) + (**139**);

byte	*0 (A)*	*1 (R)*	*2 (G)*	*3 (B)*
value	**255**	**227**	**14**	**139**

Extracting R,G,B

```
int[][] pixels = image.getPixelArray();
```

- Each pixel is a single `int` containing red, green, and blue color info.

```
int px = pixels[0][0];    // top/left pixel
```

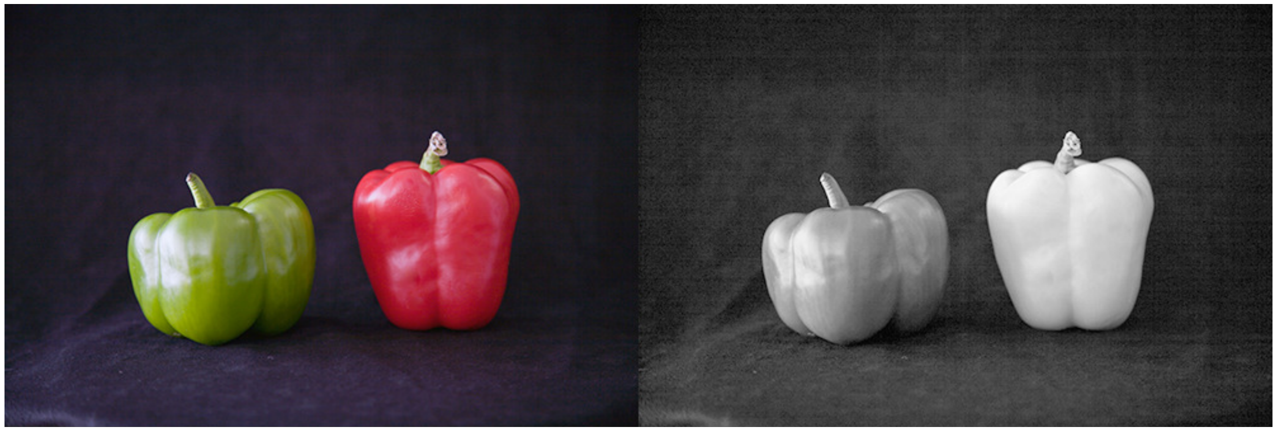
- You can extract the individual RGB color components of a pixel with `GImage.getRed`, `getBlue`, and `getGreen`.

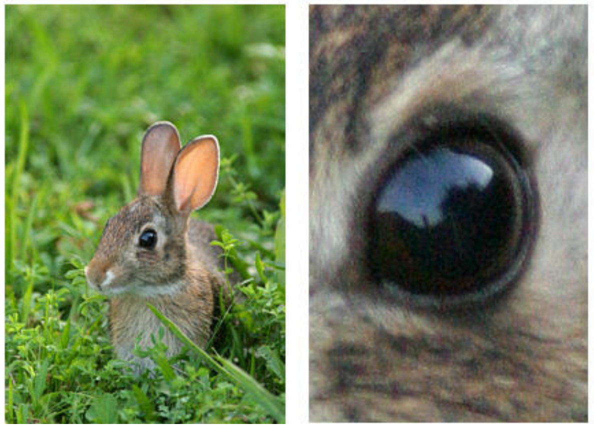
```
int red   = GImage.getRed(pixels[0][0]);     // 0-255
int green = GImage.getGreen(pixels[0][0]);   // 0-255
int blue  = GImage.getBlue(pixels[0][0]);    // 0-255
```

Modifying image pixels

- It is possible to directly create a `GImage` by specifying the RGB values of every pixel in the image.
  - Create/extract an `int[][]` **array** to hold the pixel values.
  - Separate the **RGB** components of each pixel as needed.
  - Use **`GImage.createRGBPixel`** to convert new RGB triplets to `int`.
  - Construct a new `GImage` from the array.

- Examples: convert to grayscale; zoom an image





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);
```

GImage pixel example

- **Remove all redness** from the given image.

```
GImage image = new GImage("res/example.jpg");
int[][] pixels = image.getPixelArray();
for (int r = 0; r < pixels.length; r++) {
    for (int c = 0; c < pixels[r].length; c++) {
        int red   = GImage.getRed(pixels[r][c]);
        int green = GImage.getGreen(pixels[r][c]);
        int blue  = GImage.getBlue(pixels[r][c]);
        pixels[r][c] = GImage.createRGBPixel(
                                  0, green, blue);
    }
}
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: **Double the size** of an image.

```
int[][] pixels = img.getPixelArray();
int[][] bigger = new int[pixels.length * 2]
                        [pixels[0].length * 2];
...
// fill the pixels of 'bigger'
img.setPixelArray(bigger);
```

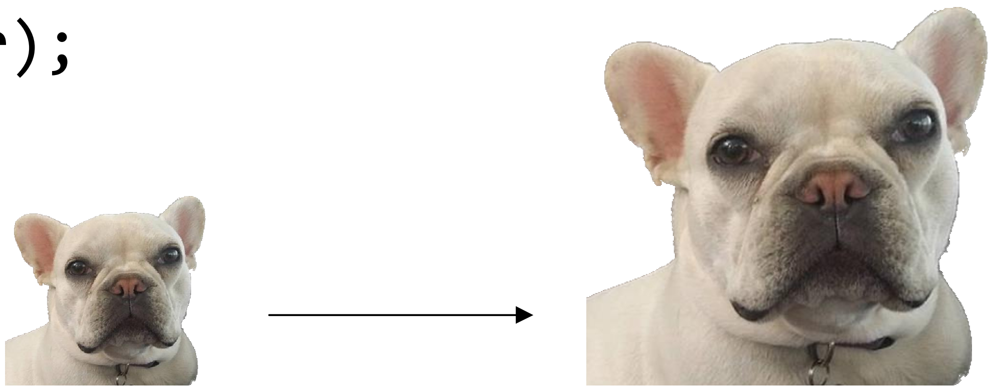


Image exercises

brighten
shrink
grow

- Write a method **`redToGreen`** that accepts a `GImage` and swaps the red/green components of the pixel colors, returning the result.

- Write a method **`brighten`** that accepts a `GImage` and modifies the image so that its pixel colors are 20 out of 255 "brighter".

- Write a method **`shrink`** that accepts a `GImage` and modifies it to be half as tall/wide as the original one.
- Write a method **`grow`** that makes an image become twice as large.

- Create an image that shows a color spectrum from darkest (top) to brightest (bottom).
- Create an image of randomly colored "static".

Shrink solution

```
public void shrink(GImage image) {
    GImage image = new GImage("res/example.jpg");
    int[][] pixels = image.getPixelArray();
    int[][] result = new int[pixels.length / 2]
                            [pixels[0].length / 2];
    for (int r = 0; r < result.length; r++) {
        for (int c = 0; c < result[0].length; c++) {
            result[r][c] = pixels[r / 2][c / 2];
        }
    }
    image.setPixelArray(result);
}
```