CS 106A, Lecture 17
2D Arrays and Images

suggested reading:
Java Ch. 11.6-11.7
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
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*.

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>12</td>
<td>49</td>
<td>-2</td>
<td>26</td>
<td>5</td>
<td>17</td>
<td>-6</td>
<td>84</td>
<td>72</td>
<td>3</td>
</tr>
</tbody>
</table>

*length = 10*

**element 0**

**element 4**

**element 9**
## Data Structures

<table>
<thead>
<tr>
<th>Operation</th>
<th>Strings</th>
<th>Arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make a new one</td>
<td>String <code>str = “abc”;</code></td>
<td></td>
</tr>
<tr>
<td>Get length?</td>
<td><code>str.length()</code></td>
<td></td>
</tr>
<tr>
<td>Get element?</td>
<td><code>str.charAt(i)</code></td>
<td></td>
</tr>
<tr>
<td>Set element?</td>
<td><em>Not allowed</em></td>
<td></td>
</tr>
<tr>
<td>Loop?</td>
<td><code>for(int i = 0; i &lt; str.length(); i++)</code></td>
<td></td>
</tr>
</tbody>
</table>
## Data Structures

<table>
<thead>
<tr>
<th>Operation</th>
<th>Strings</th>
<th>Arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make a new one</td>
<td>String str = “abc”;</td>
<td>int arr = new int[5];</td>
</tr>
<tr>
<td>Get length?</td>
<td>str.length()</td>
<td>arr.length</td>
</tr>
<tr>
<td>Get element?</td>
<td>str.charAt(i)</td>
<td>arr[i]</td>
</tr>
<tr>
<td>Set element?</td>
<td><em>Not allowed</em></td>
<td>arr[i] = 5;</td>
</tr>
<tr>
<td>Loop?</td>
<td>for(int i = 0; i &lt; str.length(); i++)</td>
<td>for(int i = 0; i &lt; arr.length; i++)</td>
</tr>
</tbody>
</table>
Creating an Array

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

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

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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.
Sometimes, we want to hardcode the elements of an array. Luckily, Java has a special syntax for initializing arrays to hardcoded numbers.

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

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

- Like Strings, indices go from 0 to the array's length - 1.
  ```java
  for (int i = 0; i < 7; i++) {
    println(numbers[i]);
  }
  println(numbers[9]);  // exception
  println(numbers[-1]); // exception
  ```

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
Putting Data In An Array

```java
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
```

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
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!
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:

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

```java
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:

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

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

```java
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:

```java
println(a1); // [I@98f8c4]
```
### Arrays Methods To The Rescue!

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

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrays.binarySearch(<code>array</code>, <code>value</code>)</td>
<td>returns the index of the given value in a <em>sorted</em> array (or &lt; 0 if not found)</td>
</tr>
<tr>
<td>Arrays.copyOf(<code>array</code>, <code>length</code>)</td>
<td>returns a new copy of array of given length</td>
</tr>
<tr>
<td>Arrays.equals(<code>array1</code>, <code>array2</code>)</td>
<td>returns <code>true</code> if the two arrays contain same elements in the same order</td>
</tr>
<tr>
<td>Arrays.fill(<code>array</code>, <code>value</code>);</td>
<td>sets every element to the given value</td>
</tr>
<tr>
<td>Arrays.sort(<code>array</code>);</td>
<td>arranges the elements into sorted order</td>
</tr>
<tr>
<td>Arrays.toString(<code>array</code>)</td>
<td>returns a string representing the array, such as &quot;[10, 30, -25, 17]&quot;</td>
</tr>
</tbody>
</table>
Plan for Today

• Recap: Arrays
• 2D Arrays
• Images as 2D Arrays
• Modifying Images
• Practice: Brighten, Grayscale, Shrink
The Matrix
2D Arrays ("Matrices")

\[
\begin{bmatrix}
a_1 & b_1 & c_1 \\
& a_2 & b_2 & c_2 \\
& & a_3 & b_3 & c_3
\end{bmatrix}
\]

WELCOME ..... TO THE MATRIX!!!!!!
2D Arrays

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

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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a[0][0]</td>
<td>a[0][1]</td>
<td>a[0][2]</td>
<td>a[0][3]</td>
<td>a[0][4]</td>
</tr>
<tr>
<td>1</td>
<td>a[1][0]</td>
<td>a[1][1]</td>
<td>a[1][2]</td>
<td>a[1][3]</td>
<td>a[1][4]</td>
</tr>
</tbody>
</table>
Manipulating 2D Arrays

\[
\text{name}[\text{row}][\text{col}] \quad \text{// get element at row,col}
\]

\[
\text{name}[\text{row}][\text{col}] = \text{value}; \quad \text{// set element at row,col}
\]
2D arrays are arrays of arrays!
```java
int[][] a = new int[3][4];
```

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0][0]</td>
<td>a[0][1]</td>
<td>a[0][2]</td>
<td>a[0][3]</td>
</tr>
<tr>
<td>a[1][0]</td>
<td>a[1][1]</td>
<td>a[1][2]</td>
<td>a[1][3]</td>
</tr>
</tbody>
</table>
A 2D array is an array where every element is *itself* an array.

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

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

“array of” `int`

“array of” `int[]`
A 2D array is an array where every element is itself an array.

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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0][0]</td>
<td>a[0][1]</td>
<td>a[0][2]</td>
<td>a[0][3]</td>
</tr>
<tr>
<td>a[1][0]</td>
<td>a[1][1]</td>
<td>a[1][2]</td>
<td>a[1][3]</td>
</tr>
</tbody>
</table>
2D Array Dimensions

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

```javascript
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:

```javascript
arr[0].length  // # entries in row 0 ( = # cols)
```
We can use **double for-loops** to loop over each row, and then each column, in a 2D array.

```java
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];
    }
}
```

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>75</td>
<td>61</td>
<td>83</td>
<td>71</td>
</tr>
<tr>
<td>1</td>
<td>94</td>
<td>89</td>
<td>98</td>
<td>91</td>
</tr>
<tr>
<td>2</td>
<td>63</td>
<td>54</td>
<td>51</td>
<td>49</td>
</tr>
</tbody>
</table>
```

“row-major” order
• **Q:** What is the array state after the code below?

```java
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];
    }
}
```

// A.  B.  C.  D.

```
0 | 1  3  5 |
1 | 1  3  5 |
2 | 1  3  5 |
3 | 1  3  5 |

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

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

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`.

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

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

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

- Get and set values using bracket notation
  
  ```java
  name[row][col]       // get elem at row,col
  name[row][col] = value; // set elem at row,col
  ```

- Get the number of rows and columns
  
  ```java
  arr.length        // # rows
  arr[0].length     // # columns
  ```

- Iterate over a 2D array using a double for-loop
  
  ```java
  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 are just grids (2D arrays) of pixels! Pixels are just integer values that have red, green, and blue components (each between 0 and 255).
Images as 2D Arrays

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

```java
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

```java
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);
}
```
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 = 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 snowman = new GImage(“res/snowman.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 snowman = new GImage("res/snowman.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 snowman = new GImage("res/snowman.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 snowman = new GImage("res/snowman.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);
}
// 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**.

```java
int[][][] pixels = image.getPixelArray();
int px = pixels[0][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.

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

```java
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);
```
There are many cool image algorithms based around modifying individual pixels in an image: grayscale, brighten, normalize, remove red-eye...
GImage Pixel Methods

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

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>img.getPixelArray()</code></td>
<td>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</td>
</tr>
<tr>
<td><code>img.setPixelArray(array)</code>;</td>
<td>updates pixels using the given 2D array of ints</td>
</tr>
<tr>
<td><code>GImage.createRGBPixel(r, g, b)</code></td>
<td>returns an int that merges the given amounts of red, green and blue (each 0-255)</td>
</tr>
<tr>
<td><code>GImage.getRed(px)</code></td>
<td>returns the redness, greenness, or blueness of the given pixel as an integer from 0-255</td>
</tr>
</tbody>
</table>
Recap: Modifying Pixels

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

  ```java
  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:

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

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

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

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

  ```java
  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.

```java
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