

Final Exam Review 1

CS106A, Summer 2019

Ryan Cao & Peter Hansel

Slides mostly from Sarai Gould & Laura Cruz-Albrecht



Plan for Today

- Announcements / Exam Logistics
- Learning Goals
- Graphics & Events
- Arrays
- 2D Arrays
- ArrayLists

Plan for Today

- Announcements / Exam Logistics
- Learning Goals
- Graphics & Events
- Arrays
- 2D Arrays
- ArrayLists

Final Logistics

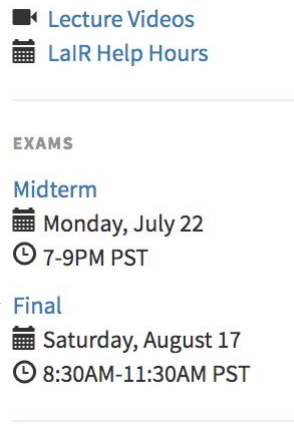
- When and where?
 - Saturday August 17th, 8:30-11:30AM
 - Bishop Auditorium
- Bluebook
 - Download from CS106A website
 - Try it out before the exam (practice finals available)
- What to bring
 - Laptop and charger
 - 2 double-sided 8.5" x 11" sheets of physical notes (**no notes on your laptop**)
 - Your two-step authentication device (probably your phone)
 - Make sure you download the encrypted final onto Bluebook before the exam!

Final Logistics

Important info (logistics, practice exams, and sample syntax reference sheet, etc) can be found at:

<http://web.stanford.edu/class/cs106a/exams/final.html>

Linked on left sidebar
of class website



Lecture Videos
LaIR Help Hours

EXAMS

Midterm
Monday, July 22
7-9PM PST

Final
Saturday, August 17
8:30AM-11:30AM PST

Final Logistics

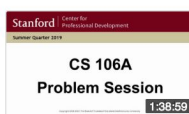
- Is the final exam cumulative?
- What will be tested on the final exam?

Final Logistics

- Is the final exam cumulative?
- What will be tested on the final exam?
- What about all this stuff you aren't covering today?
 - Expressions and Variables
 - Java Control Statements
 - Console Programs
 - Methods, parameters, returns
 - Randomness
 - Strings and chars
 - Scanners and file processing
 - Memory

Midterm Review: Friday of week 4.

▶ Lecture Videos



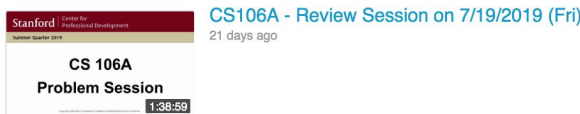
CS106A - Review Session on 7/19/2019 (Fri)
21 days ago

Final Logistics

- Is the final exam cumulative?
- What will be tested on the final exam?
- What about all this stuff you aren't covering today?
 - Expressions and Variables
 - Java Control Statements
 - Console Programs
 - Methods, parameters, returns
 - Randomness
 - Strings and chars
 - Scanners and file processing
 - Memory
- How can I practice for the final?

Midterm Review: Friday of week 4.

Lecture Videos



Practicing for the Final

- Review concepts you're unsure of
- Review programs we wrote in lecture
- Do section problems
- Do practice final under real conditions
 - Don't look at the answer key unless you've given it a good go first :)
- <https://www.codestepbystep.com/>

Plan for Today

- Announcements / Exam Logistics
- **Learning Goals**
- Graphics & Events
- Arrays
- 2D Arrays
- ArrayLists

Learning Goals

“After this lecture, I want you to be able to...”

- Lectures 1-3 (Karel): Apply programmatic thinking and decomposition to logical tasks
- Lecture 4 (Intro to Java): Create variables of primitive types, perform console I/O, and evaluate expressions using primitive types
- Lecture 5 (Booleans and Control Flow): Use loops to perform repeated tasks, use conditions to decide which tasks to perform

Learning Goals

“After this lecture, I want you to be able to...”

- Lecture 6 (Methods, Parameters, and Scope): Identify a variable’s scope, and write functions that pass parameters and leverage return values to overcome the limitation of scope in program decomposition.
- Lecture 7 (Nested For Loops & Intro to Graphics): Learn how to use nested loops and create simple graphics.

Learning Goals

“After this lecture, I want you to be able to...”

- Lecture 8 (More Methods & Graphics): Write programs using five types of graphical objects (rectangles, ovals, lines, labels, and images), call methods on Objects
- Lecture 9 (Animation & Randomness): Use loops and pausing to animate graphical programs, and understand how to use a RandomGenerator
- Lecture 10 (Mouse Events & Instance Variables): Write programs that respond to mouse events, identify when it is appropriate to use instance variables

Learning Goals

“After this lecture, I want you to be able to...”

- Lecture 11 (Methods and Scope with Tracing and Debugging): Understand how to trace through a Java program, use the Java debugger
- Lecture 12 (Memory): Recall that primitives are passed by value while Objects are passed by reference in Java, apply that knowledge to know which variables' values change when they are modified in other methods

Learning Goals

“After this lecture, I want you to be able to...”

- Lecture 13 (Characters and Strings): recall that Java understands chars as ASCII values (ints from 0 - 255), create String variables, recall that Strings are immutable
- Lecture 14 (String Processing): Identify situations where common String methods like length and substring are useful, solve problems that involve manipulating Strings (often through creating new Strings)
- Lecture 15 (File Reading): Write programs that use files as sources of input data

Learning Goals

“After this lecture, I want you to be able to...”

- Lecture 16 (Arrays): Describe the purpose of data structures in programming, know how to store data in and retrieve data from arrays
- Lecture 17 (2D Arrays): Recognize 2D arrays as grids or arrays of arrays, apply nested for loops to work with 2D arrays, process images as 2D arrays of pixels
- Lecture 18 (ArrayLists): Know how to store data in and retrieve data from ArrayLists

Learning Goals

“After this lecture, I want you to be able to...”

- Lecture 19 (HashMaps): Know how to store data in and retrieve data from HashMaps
- Lecture 20 (DataStructures: Bringing it all Together): Write programs that leverage different data structures, and identify the most appropriate data structure between arrays, ArrayLists, and HashMaps for different storage needs.
- Lecture 21-22 (Classes): Learn how to design and use Java classes. Understand that classes define a new variable type.

Learning Goals

“After this lecture, I want you to be able to...”

- Lecture 23 (Interactors): Know how to use Java's interactive components.
- Lecture 24 (Internet Applications): Understand the two types of Internet programs - servers and clients - and how they interact.
- Lecture 25 (How to Start your own Java Program): How to start your own Java program and leverage jar files.
- Lecture 26 (Life After CS106A): Identify next learning steps, and see real-world applications where computer science can help.

Plan for Today

- Announcements / Exam Logistics
- Learning Goals
- **Graphics & Events**
- Arrays
- 2D Arrays
- ArrayLists

Graphics

- Look at lecture slides for lists of different GObject types and their methods.
- Remember: the x and y of GRect, GImage, GOval, etc. is their upper-left corner; and the x and y of GLabel is the lower-left baseline corner.

Events

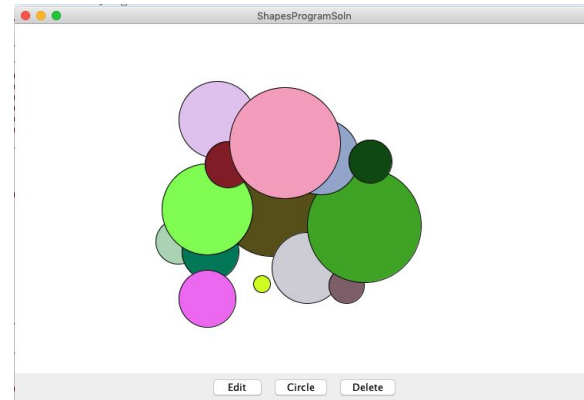
- Two ways for Java to run your code: from `run()` and from event handlers (`mouseClicked`, `mouseMoved`, `actionPerformed`, etc.)
- Event handlers must have exactly the specified signature; otherwise they won't work!

e.g., `public void mouseClicked(MouseEvent e)`

- If you need access to a variable in an event handler that you use elsewhere in your code, it should be an instance variable (e.g., `paddle` in `Breakout`)

Live Demo: Simple Circles Program

- A simple graphics program! Has three buttons -- adding a Circle, editing (i.e. moving around) a clicked Circle, and deleting a clicked circle.
- Circles must have a random fill color, a radius between `MIN_RADIUS` and `MAX_RADIUS`, and be centered at the point of the mouse click.
- When Circles are dragged in Edit mode, they must follow the mouse exactly as they were clicked.



Plan for Today

- Announcements / Exam Logistics
- Learning Goals
- Graphics & Events
- **Arrays**
- 2D Arrays
- ArrayLists

First - Data Structures

Data Structures allow us to store more data in more interesting ways. We've seen several:

- Arrays
- 2D Arrays
- ArrayLists
- Hashmaps

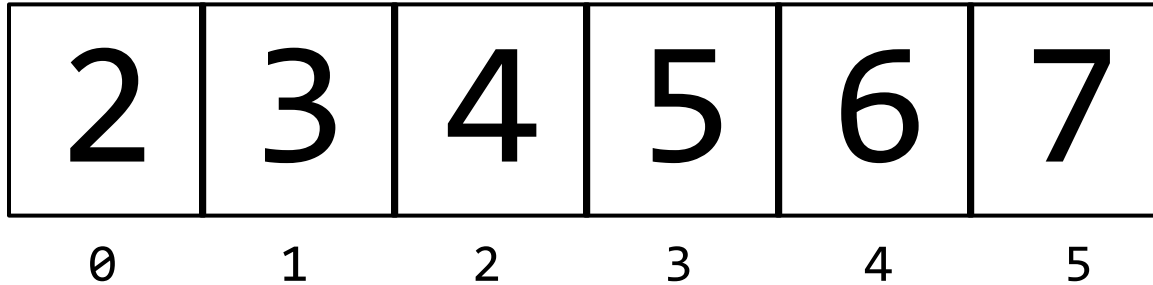
Let's first talk about arrays!

Arrays

- An array is a fixed-length list of a single type of thing.
- An array can store **primitives** and **Objects**.
- You cannot call methods on arrays, e.g., no `myArray.contains()`

Arrays

- Each location is assigned an index, going from 0 to length-1.
- The type of data at each index depends on the type of array!



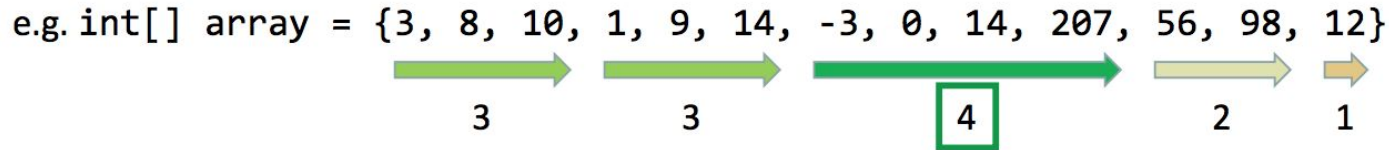
```
int arrayLen = myArray.length;    // 6
int last = myArray[arrayLen - 1];  // 7
```

Arrays

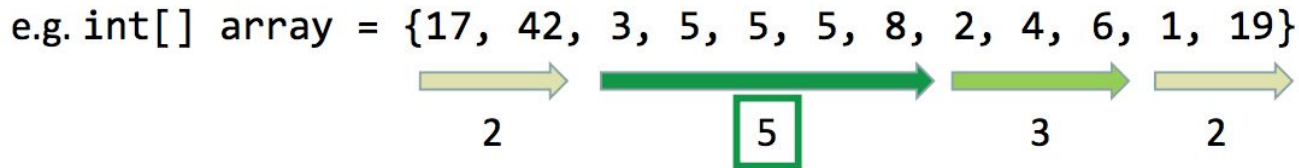
```
int[] myArray = new int[5];  
// OR  
int[] myArray = {2, 3, 4, 5, 6, 7};  
  
int arrayLen = myArray.length;  
  
// Access elements with bracket notation  
int first = myArray[0];  
int last = myArray[arrayLen - 1];  
  
// In arrays, we can change elements!  
myArray[0] = 22;
```

1D Array Practice

Write the method `int longestSortedSequence(int[] array)`



Sorted in this case means nondecreasing, so a sequence could contain duplicates:



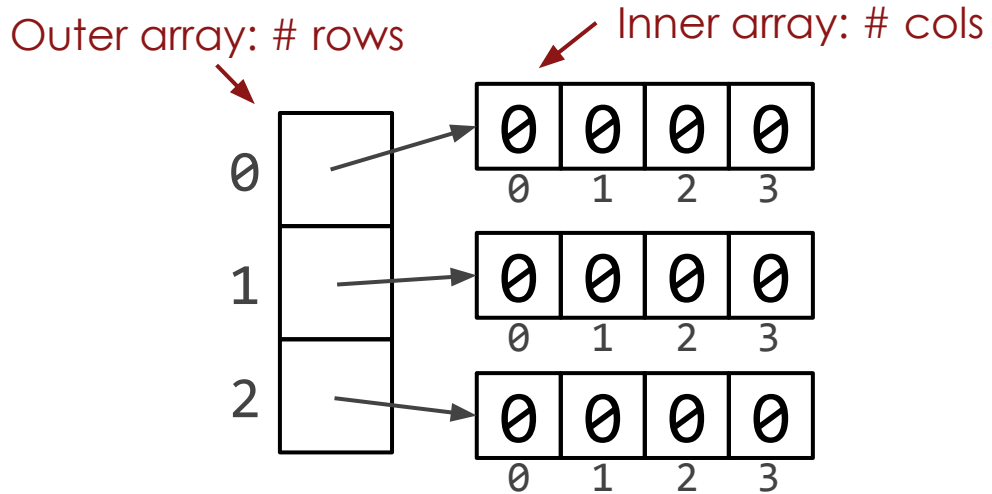
Plan for Today

- Announcements / Exam Logistics
- Learning Goals
- Graphics & Events
- Arrays
- **2D Arrays**
- ArrayLists

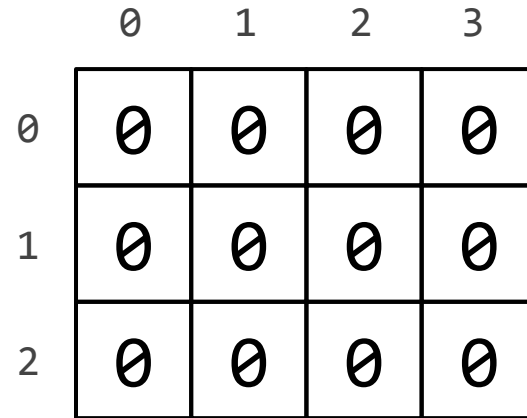
2D Arrays

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

rows # columns



An array of arrays



A unit (ie, a grid/matrix)

2D Arrays

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

```
matrix[row][col];           // get element  
matrix[row][col] = value;   // set element
```

2D Arrays For Loops

- The canonical way to loop over a 2D array is with a double for loop

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


2D Arrays For Loops

- The canonical way to loop over a 2D array is with a double for loop

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

arr.length (with arrow pointing to `numRows(arr)`)

arr[0].length (with arrow pointing to `numCols(arr)`)

2D Arrays on 1 Slide

1. Make a 2D array

```
double[][] grid = new double[nRows][nCols];
```

2. Set and get values from a 2D array using bracket notation

```
grid[2][1] = 2.2;  
println("Upper left val is: " + grid[0][0]);
```

3. Get the number of rows and columns of a 2D array (tip: define method)

```
int numRows = grid.length;  
int numCols = grid[0].length;
```

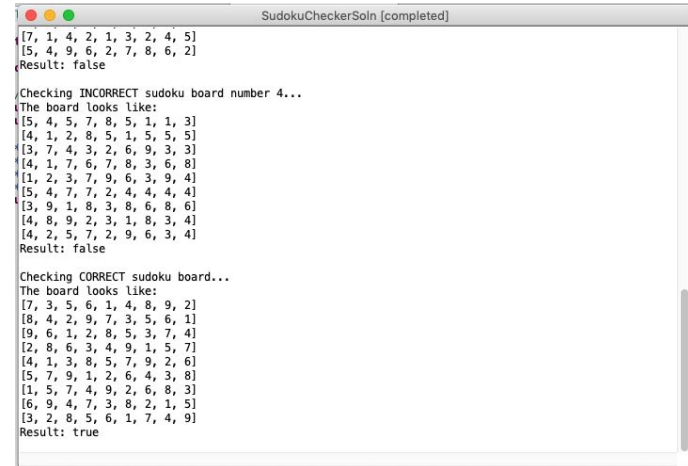
4. Use a double for loop to iterate over the entire 2D array

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

Live Demo: Sudoku Checker

- Sudoku is a game played on a 9 x 9 grid, in which the goal is to fill every row, column, and sector with the numbers 1 through 9 exactly once.
- Our goal is, given a sudoku board as input, to output whether the solution presented is valid or not.
- To do this, we will need to implement three separate methods -- `checkRows()`, `checkCols()`, and `checkSector()`.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

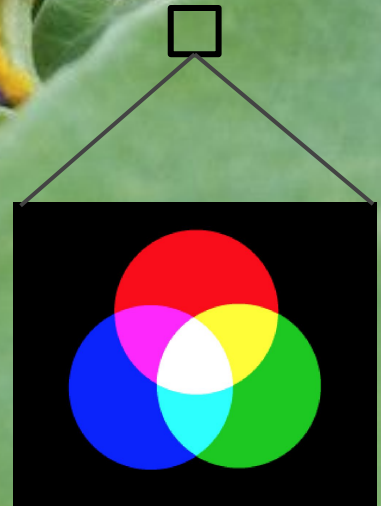


```
SudokuCheckerSoln [completed]
[7, 1, 4, 2, 1, 3, 2, 4, 5]
[5, 4, 9, 6, 2, 7, 8, 6, 2]
Result: false

Checking INCORRECT sudoku board number 4...
The board looks like:
[5, 4, 5, 7, 8, 5, 1, 1, 3]
[4, 1, 2, 8, 5, 1, 5, 5, 5]
[3, 7, 4, 3, 2, 6, 9, 3, 3]
[4, 1, 7, 6, 7, 8, 3, 6, 8]
[1, 2, 3, 7, 9, 6, 3, 9, 4]
[5, 4, 7, 7, 2, 4, 4, 4, 4]
[3, 9, 1, 8, 3, 8, 6, 8, 6]
[4, 8, 9, 2, 3, 1, 8, 3, 4]
[4, 2, 5, 7, 2, 9, 6, 3, 4]
Result: false

Checking CORRECT sudoku board...
The board looks like:
[7, 3, 5, 6, 1, 4, 8, 9, 2]
[8, 4, 2, 9, 7, 3, 5, 6, 1]
[9, 6, 1, 2, 8, 5, 3, 7, 4]
[2, 8, 6, 3, 4, 9, 1, 5, 7]
[4, 1, 3, 8, 5, 7, 9, 2, 6]
[5, 7, 9, 1, 2, 6, 4, 3, 8]
[1, 5, 7, 4, 9, 2, 6, 8, 3]
[6, 9, 4, 7, 3, 8, 2, 1, 5]
[3, 2, 8, 5, 6, 1, 7, 4, 9]
Result: true
```

Review: images are
2D arrays of *pixels*.



Modifying Pixels

- Get 2D pixel array from an image

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

- 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 values back together into a single int.

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

- Update image with your modified pixels when finished.

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

Make Grayscale

```
public class GrayscaleSoln extends GraphicsProgram {
    private static final String IMG_NAME = "res/stanford.jpg";

    public void run() {
        GImage image = new GImage(IMG_NAME);
        setCanvasSize(image.getWidth(), image.getHeight());
        add(image);

        // Calculate the luminosity of a pixel using NTSC formula
        private int computeLuminosity(int r, int g, int b) {
            return GMath.round(0.299 * r + 0.587 * g + 0.114 * b);
        }
    }
}
```

Make Grayscale

```
public class GrayscaleSoln extends GraphicsProgram {
    private static final String IMG_NAME = "res/stanford.jpg";

    public void run() {
        GImage image = new GImage(IMG_NAME);
        setCanvasSize(image.getWidth(), image.getHeight());
        add(image);

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

        // ...

    }

    // Calculate the luminosity of a pixel using NTSC formula
    private int computeLuminosity(int r, int g, int b) {
        return GMath.round(0.299 * r + 0.587 * g + 0.114 * b);
    }
}
```

Make Grayscale

```
public class GrayscaleSoln extends GraphicsProgram {
    private static final String IMG_NAME = "res/stanford.jpg";

    public void run() {
        GImage image = new GImage(IMG_NAME);
        setCanvasSize(image.getWidth(), image.getHeight());
        add(image);

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

        // Loop over all pixels and convert them to grayscale
        for (int row = 0; row < pixels.length; row++) {
            for (int col = 0; col < pixels[0].length; col++) {

                }

            }

        }

        // Calculate the luminosity of a pixel using NTSC formula
        private int computeLuminosity(int r, int g, int b) {
            return GMath.round(0.299 * r + 0.587 * g + 0.114 * b);
        }
    }
}
```


Make Grayscale

```
public class GrayscaleSoln extends GraphicsProgram {
    private static final String IMG_NAME = "res/stanford.jpg";

    public void run() {
        GImage image = new GImage(IMG_NAME);
        setCanvasSize(image.getWidth(), image.getHeight());
        add(image);

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

        // Loop over all pixels and convert them to grayscale
        for (int row = 0; row < pixels.length; row++) {
            for (int col = 0; col < pixels[0].length; col++) {
                int pixel = pixels[row][col];

                int red = GImage.getRed(pixel);
                int green = GImage.getGreen(pixel);
                int blue = GImage.getBlue(pixel);

                int luminosity = computeLuminosity(red, green, blue);
                int newPixel = GImage.createRGBPixel(luminosity, luminosity, luminosity);
                pixels[row][col] = newPixel;
            }
        }

        // Calculate the luminosity of a pixel using NTSC formula
        private int computeLuminosity(int r, int g, int b) {
            return GMath.round(0.299 * r + 0.587 * g + 0.114 * b);
        }
    }
}
```

Make Grayscale

```
public class GrayscaleSoln extends GraphicsProgram {
    private static final String IMG_NAME = "res/stanford.jpg";

    public void run() {
        GImage image = new GImage(IMG_NAME);
        setCanvasSize(image.getWidth(), image.getHeight());
        add(image);

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

        // Loop over all pixels and convert them to grayscale
        for (int row = 0; row < pixels.length; row++) {
            for (int col = 0; col < pixels[0].length; col++) {
                int pixel = pixels[row][col];

                int red = GImage.getRed(pixel);
                int green = GImage.getGreen(pixel);
                int blue = GImage.getBlue(pixel);

                int luminosity = computeLuminosity(red, green, blue);
                int newPixel = GImage.createRGBPixel(luminosity, luminosity, luminosity);
                pixels[row][col] = newPixel;
            }
        }

        // Update the image with the new pixels
        image.setPixelArray(pixels);
    }

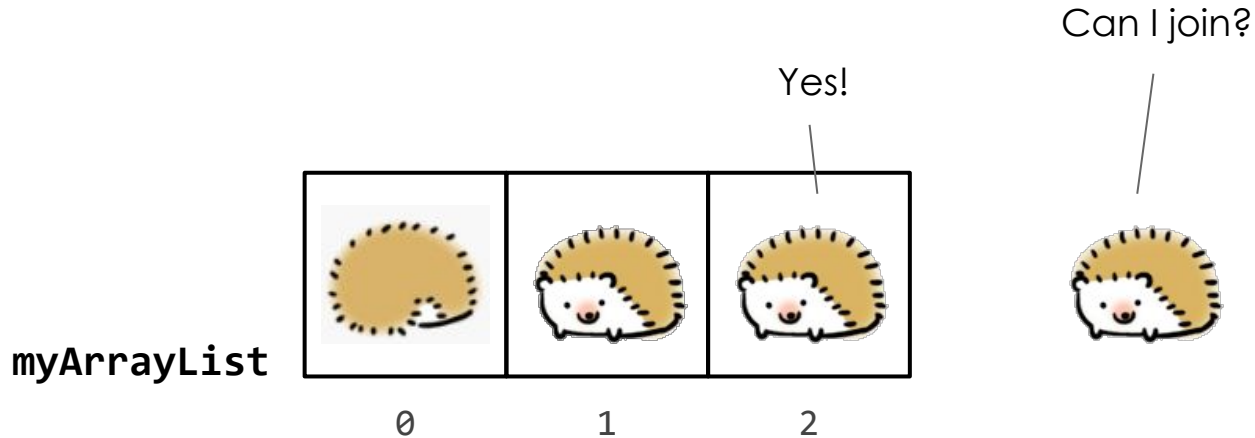
    // Calculate the luminosity of a pixel using NTSC formula
    private int computeLuminosity(int r, int g, int b) {
        return GMath.round(0.299 * r + 0.587 * g + 0.114 * b);
    }
}
```

Plan for Today

- Announcements / Exam Logistics
- Learning Goals
- Graphics & Events
- Arrays
- 2D Arrays
- **ArrayLists**

ArrayLists

- An ordered, *resizable* list of information
- Can add and remove elements (among other cool functionality)



ArrayLists

- An ArrayList is a flexible-length list of a single type of thing.
- An ArrayList can only store Objects.
 - For primitives, use wrapper classes; ie `ArrayList<Integer>` instead of `ArrayList<int>` (`Integer` is a wrapper class for `int`)
- An ArrayList has a variety of methods you can use like `.contains`, `.get`, `.add`, `.remove`, `.size`, etc

Arrays vs. ArrayLists

Operation

Arrays

ArrayLists

Make a new one

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

```
ArrayList<String> list = new  
ArrayList<String>();
```

Length?

```
arr.length
```

```
list.size()
```

Get element?

```
arr[i]
```

```
list.get(i)
```

Set element?

```
arr[i] = value
```

```
list.set(i, value)
```

Loop?

```
for(int i = 0; i < arr.length; i++)
```

```
for(String value : list)
```

Arrays vs. ArrayLists

- Array
 - Fixed size
 - Efficient (not a concern in this class)
 - No methods, can only use `myArray.length` (no parentheses!)
 - Can store any object or primitive
- ArrayList
 - Expandable
 - Less efficient than Array (not a concern in this class)
 - Convenient methods like `.add()`, `.remove()`, `.contains()`
 - Cannot store primitives, so use their wrapper classes instead

deleteDuplicates

```
private void deleteDuplicates(ArrayList<String> list)
```

- Guaranteed that list is in sorted order

Before: {"be", "be", "is", "not", "or", "question", "that", "the", "to", "to"}

After: {"be", "is", "not", "or", "question", "that", "the", "to"}

- Solution strategy:
 - Loop through ArrayList
 - Compare pairs of elements
 - If `element.equals(nextElement)`, remove element from the list

deleteDuplicates

- Loop through ArrayList
- Compare pairs of elements
- If `element.equals(nextElement)`, remove element from the list

```
private void deleteDuplicates(ArrayList<String> list) {  
    for (int i = 0; i < list.size() - 1; i++) {  
        String elem = list.get(i);  
        // If two adjacent elements are equal  
        if (list.get(i + 1).equals(elem)) {  
            list.remove(i);  
            i--;  
        }  
    }  
}
```

deleteDuplicatesReverse

- Loop through ArrayList **in reverse**
- Compare pairs of elements
- If `element.equals(previousElement)`, remove element from the list

```
private void deleteDuplicatesReverse(ArrayList<String> list) {  
    for (int i = list.size() - 1; i > 0; i--) {  
        String elem = list.get(i);  
        // If two adjacent elements are equal  
        if (list.get(i - 1).equals(elem)) {  
            list.remove(i);  
        }  
    }  
}
```

Plan for Today

- Announcements / Exam Logistics
- Learning Goals
- Graphics & Events
- Arrays
- 2D Arrays
- ArrayLists

Next Time: Final Exam Review 2

Questions

Any questions? :D