Array Lists

Lecture 18

CS106A, Summer 2019
Sarai Gould && Laura Cruz-Albrecht

With inspiration from slides created by Keith Schwarz, Mehran Sahami, Eric Roberts, Stuart Reges, Chris Piech and others.
Announcements

- Assignment 4 due Monday July 29th at 10AM
- Blank lecture code on website schedule
Know how to store data in and retrieve data from an ArrayList.
Plan for Today

- Review: 2D Arrays
- ArrayLists
- Example: Reversible Writing
- Example: Trip Planner
- ArrayLists vs. Arrays
Plan for Today

● Review: 2D Arrays
● ArrayLists
● Example: Reversible Writing
● Example: Trip Planner
● ArrayLists vs. Arrays
Review 2D: Arrays

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

# rows  # columns
int[][] matrix = new int[3][4];
Review 2D: Arrays

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

matrix[row][col]; // get element
matrix[row][col] = value; // set element
```
private int numRows(int[][][] matrix) {
    return matrix.length;
}

private int numCols(int[][][] matrix) {
    return matrix[0].length;
}
The canonical way to loop over a 2D array is with a double for loop.

```java
type[][][] arr = ...;
for (int row = 0; row < numRows(arr); row++) {
    for (int col = 0; col < numCols(arr); col++) {
        // do something with arr[row][col] ...
    }
}
```
Review: images are 2D arrays of pixels.
## GImage Pixel Methods

<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>
<tr>
<td><code>GImage.getGreen(px)</code></td>
<td></td>
</tr>
<tr>
<td><code>GImage.getBlue(px)</code></td>
<td></td>
</tr>
</tbody>
</table>
Plan for Today

- Review: 2D Arrays
- **ArrayLists**
- Example: Reversible Writing
- Example: Trip Planner
- ArrayLists vs. Arrays
Limitations of Arrays

- Size must be specified upon creation
- Can’t add/remove/insert elements later (because size is fixed)
Limitations of Arrays

- Size must be specified upon creation
- Can’t add/remove/insert elements later (because size is fixed)
Limitations of Arrays

- Size must be specified upon creation
- Can’t add/remove/insert elements later (because size is fixed)
How can we help?
Introducing ArrayLists

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

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

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

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

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

```
myArrayList
  0  1  2  3
  [ ] [ ] [ ] [ ]
```

Ooh can I come too??
Introducing ArrayLists

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

```
myArrayList
```

```
0  1  2  3  4
0  1  2  3  4
```
ArrayLists

- An ordered, resizable list of information
- Can add and remove elements (among other cool functionality)
- Homogenous
- Can store any `Object` type
- Access individual items by `index`

```java
myArrayList
```

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Our First ArrayList

```java
ArrayList<String> myArrayList = new ArrayList<String>();
```
Our First ArrayList

import java.util.*;

ArrayList<String> myArrayList = new ArrayList<String>();
Our First ArrayList

```java
ArrayList<String> myArrayList = new ArrayList<String>();
```
Our First ArrayList

Type of thing your ArrayList will store

ArrayList<String> myArrayList = new ArrayList<String>();
Our First ArrayList

ArrayList<String> myArrayList = new ArrayList<String>();
Our First ArrayList

ArrayList<String> myArrayList = new ArrayList<String>();
Our First ArrayList

```java
ArrayList<String> myArrayList = new ArrayList<String>();
```

Same type here.
Our First ArrayList

ArrayList<String> myArrayList = new ArrayList<>();

Can optionally leave empty because of type inference
Our First ArrayList

```java
ArrayList<String> myArrayList = new ArrayList<String>();
```
ArrayList<String> myArrayList = new ArrayList<String>(); // initially empty
Our First ArrayList

ArrayList<String> myArrayList = new ArrayList<String>(); // initially empty

// Adds elements to the back
myArrayList.add("hi");
Our First ArrayList

ArrayList<String> myArrayList = new ArrayList<String>(); // initially empty

// Adds elements to the back
myArrayList.add("hi");
myArrayList.add("there");
Our First ArrayList

ArrayList<String> myArrayList = new ArrayList<String>(); // initially empty

// Adds elements to the back
myArrayList.add("hi");
myArrayList.add("there");

// Access elements by index (starting at 0!)
println(myArrayList.get(0)); // prints “hi”
println(myArrayList.get(1)); // prints “there”
Our First ArrayList

ArrayList<String> myArrayList = new ArrayList<String>(); // initially empty

// Adds elements to the back
myArrayList.add("hi");
myArrayList.add("there");

// Access elements by index (starting at 0!)
println(myArrayList.get(0));  // prints "hi"
println(myArrayList.get(1));  // prints "there"

// Wrong type - bad times! Won’t compile
GLabel label = new GLabel("hi there");
myArrayList.add(label);
Our First ArrayList

ArrayList<String> myArrayList = new ArrayList<String>();  // initially empty

// Adds elements to the back
myArrayList.add("hi");
myArrayList.add("there");

// Access elements by index (starting at 0!)
println(myArrayList.get(0));  // prints “hi”
println(myArrayList.get(1));  // prints “there”

// Wrong type - bad times! Won’t compile
GLabel label = new GLabel("hi there");
myArrayList.add(label);

// Invalid index – crashes! IndexOutOfBoundsException Exception
println(myArrayList.get(2));
ArrayList<String> myArrayList = new ArrayList<String>();

// Adds elements to the back
myArrayList.add("hi");
myArrayList.add("there");

// Access elements by index (starting at 0!)
for (int i = 0; i < myArrayList.size(); i++) {
    String str = myArrayList.get(i);
    println(str);
}

// hi
// there
ArrayList<String> myArrayList = new ArrayList<String>();

// Adds elements to the back
myArrayList.add("hi");
myArrayList.add("there");

// Access elements by index (starting at 0!)
for (int i = 0; i < myArrayList.size(); i++) {
    String str = myArrayList.get(i);
    println(str);
}

// A beautiful way to access each element
for (String str : myArrayList) {
    println(str);
}
ArrayList<String> myArrayList = new ArrayList<String>();

// Adds elements to the back
myArrayList.add("hi");
myArrayList.add("there");

// Access elements by index (starting at 0!)
for (int i = 0; i < myArrayList.size(); i++) {
    String str = myArrayList.get(i);
    println(str);
}

// A beautiful way to access each element
for (String str : myArrayList) {
    println(str);
}
### ArrayList Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean add(&lt;T&gt; element)</td>
<td>Adds a new element to the end of the ArrayList; the return value is always true.</td>
</tr>
<tr>
<td>void add(int index, &lt;T&gt; element)</td>
<td>Inserts a new element into the ArrayList before the position specified by index.</td>
</tr>
<tr>
<td>&lt;T&gt; remove(int index)</td>
<td>Removes the element at the specified position and returns that value.</td>
</tr>
<tr>
<td>boolean remove(&lt;T&gt; element)</td>
<td>Removes the first instance of element, if it appears; returns true if a match is found.</td>
</tr>
<tr>
<td>void clear()</td>
<td>Removes all elements from the ArrayList.</td>
</tr>
<tr>
<td>int size()</td>
<td>Returns the number of elements in the ArrayList.</td>
</tr>
<tr>
<td>&lt;T&gt; get(int index)</td>
<td>Returns the object at the specified index.</td>
</tr>
<tr>
<td>&lt;T&gt; set(int index, &lt;T&gt; value)</td>
<td>Sets the element at the specified index to the new value and returns the old value.</td>
</tr>
<tr>
<td>int indexOf(&lt;T&gt; value)</td>
<td>Returns the index of the first occurrence of the specified value, or -1 if it does not appear.</td>
</tr>
<tr>
<td>boolean contains(&lt;T&gt; value)</td>
<td>Returns true if the ArrayList contains the specified value.</td>
</tr>
<tr>
<td>boolean isEmpty()</td>
<td>Returns true if the ArrayList contains no elements.</td>
</tr>
</tbody>
</table>
Plan for Today

- Review: 2D Arrays
- ArrayLists
- Example: Reversible Writing
- Example: Trip Planner
- ArrayLists vs. Arrays
Example: Reversible Writing

Let’s write a program that reverses a text file.

I am not a person who contributes
And I refuse to believe that
I will be useful
Example: Reversible Writing

Let’s write a program that reverses a text file.

I am not a person who contributes
And I refuse to believe that
I will be useful

I will be useful
And I refuse to believe that
I am not a person who contributes

"I Have a Dream" by Antonia Lee, Sara Fung, Christy Fung, Rachel Lam
Example: Reversible Writing

Let’s write a program that reverses a text file.

“"I am not a person who contributes""
Let’s write a program that reverses a text file.

<table>
<thead>
<tr>
<th>“I am not a person who contributes”</th>
</tr>
</thead>
<tbody>
<tr>
<td>“And I refuse to believe that”</td>
</tr>
</tbody>
</table>
Example: Reversible Writing

Let's write a program that reverses a text file.

<table>
<thead>
<tr>
<th>“I am not a person who contributes”</th>
</tr>
</thead>
<tbody>
<tr>
<td>“And I refuse to believe that”</td>
</tr>
<tr>
<td>“I will be useful”</td>
</tr>
</tbody>
</table>
Example: Reversible Writing

Let’s write a program that reverses a text file.

| “I am not a person who contributes” |
| “And I refuse to believe that”       |
| “I will be useful”                   |

Key Idea # 1: fill an ArrayList with each line in the file.
Example: Reversible Writing

Let’s write a program that reverses a text file.

| “I am not a person who contributes” |
| “And I refuse to believe that”       |
| “I will be useful”                   |
Example: Reversible Writing

Let’s write a program that reverses a text file.

| “I am not a person who contributes” |
| “And I refuse to believe that”      |
| “I will be useful”                  |
Example: Reversible Writing

Let’s write a program that reverses a text file.

| “I am not a person who contributes” |
| “And I refuse to believe that”       |
| “I will be useful”                  |
Example: Reversible Writing

Let’s write a program that reverses a text file.

<table>
<thead>
<tr>
<th>“I am not a person who contributes”</th>
</tr>
</thead>
<tbody>
<tr>
<td>“And I refuse to believe that”</td>
</tr>
<tr>
<td>“I will be useful”</td>
</tr>
</tbody>
</table>

Key Idea # 2: print the ArrayList items in reverse order.
try {
    Scanner scanner = new Scanner(new File("FILENAME"));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
    scanner.close();
} catch (IOException ex) {
    println("Could not read file.");
}
```java
try {
    Scanner scanner = new Scanner(new File("FILENAME"));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
    scanner.close();
} catch (IOException ex) {
    println("Could not read file.");
}
```
Example: Reversible Writing

```java
try {
    Scanner scanner = new Scanner(new File("FILENAME"));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from! back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }

    scanner.close();
} catch (IOException ex) {
    println("Could not read file.");
}
```
try {
    Scanner scanner = new Scanner(new File("FILENAME"));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }

    scanner.close();
} catch (IOException ex) {
    println("Could not read file.");
}
Example: Reversible Writing

```java
try {
    Scanner scanner = new Scanner(new File("FILENAME"));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
    scanner.close();
} catch (IOException ex) {
    println("Could not read file.");
}
```
Example: Reversible Writing

try {
    Scanner scanner = new Scanner(new File("FILENAME"));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
    scanner.close();
} catch (IOException ex) {
    println("Could not read file.");
}
```java
try {
    Scanner scanner = new Scanner(new File("FILENAME"));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
    scanner.close();
}
```
try {
    Scanner scanner = new Scanner(new File(FILENAME));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }

    scanner.close();
} catch (IOException ex) {
    println("Could not read file.");
}
Plan for Today

- Review: 2D Arrays
- ArrayLists
- Example: Reversible Writing
- Example: Trip Planner
- ArrayLists vs. Arrays
A Note on Insert/Remove

• If you insert or remove an element from a list, any elements to the right of it shift to fit
A Note on Insert/Remove

- If you insert or remove an element from a list, any elements to the right of it shift to fit

```java
list.add(2, 42); // add the value 42 before index 2
```

<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>3</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>3</td>
<td>8</td>
<td>42</td>
<td>9</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>
A Note on Insert/Remove

- If you insert or remove an element from a list, any elements to the right of it shift to fit.

```java
list.add(2, 42); // add the value 42 before index 2
```

```java
list.remove(1); // remove the element at index 1
```

![Diagram](image-url)
It’s summer, and you want to travel! Let’s write a program to plan our itinerary.

- Program first prompts the user for all the cities they want to visit
- Then, it asks user to re-enter them in the order they’d like to visit them
- Finally, outputs the itinerary: the order in which to visit the cities

```
Enter a city: Florence
Enter a city: Singapore
Enter a city: Seattle
Enter a city:
That's an exciting list! Enter the order you'd like to visit those cities.
Cities remaining: [Florence, Singapore, Seattle]
Next city to visit: Singapore
Cities remaining: [Florence, Seattle]
Next city to visit: Seattle
Cities remaining: [Florence]
Next city to visit: Florence
Great! Your itinerary is ready:
[Singapore, Seattle, Florence]```
Trip Planner: Approach

Cities:

Order:

Florence
Trip Planner: Approach

Cities:

- Florence
- Singapore

Order:
Trip Planner: Approach

Cities:

Florence | Singapore | Seattle

Order:
Trip Planner: Approach

Cities:
- Florence
- Singapore
- Seattle

Order:
Trip Planner: Approach

Cities: Florence Seattle

Order: Singapore
Cities: Florence, Seattle
Order: Singapore
Trip Planner: Approach

Cities: Florence

Order: Singapore Seattle
Trip Planner: Approach

Cities:
- Florence

Order:
- Singapore
- Seattle
Trip Planner: Approach

Cities: Done!

Order: Singapore Seattle Florence
Let’s Code It!
Plan for Today

- Review: 2D Arrays
- ArrayLists
- Example: Reversible Writing
- Example: Trip Planner
- ArrayLists vs. Arrays
ArrayLists + Primitives

// Doesn’t compile ~ sad times :(
ArrayList<int> myArrayList = new ArrayList<int>();

Unlike Arrays, ArrayLists can only store Objects.
# Wrapper Classes

<table>
<thead>
<tr>
<th>Primitive</th>
<th>“Wrapper” Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
</tbody>
</table>
// Use wrapper classes when making an ArrayList
ArrayList<Integer> numList = new ArrayList<Integer>();

// Java converts Integer <-> int automatically!
numList.add(22);
numList.add(44);

int firstNum = numList.get(0); // 22
int secondNum = numList.get(1); // 44

Conversion happens automatically!
An Example

Let’s check out an example
# Arrays vs. ArrayLists

<table>
<thead>
<tr>
<th>Operation</th>
<th>Arrays</th>
<th>ArrayLists</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make a new one</td>
<td><code>int arr = new int[5];</code></td>
<td><code>ArrayList&lt;String&gt; list = new ArrayList&lt;String&gt;();</code></td>
</tr>
<tr>
<td>Length?</td>
<td><code>arr.length</code></td>
<td><code>list.size()</code></td>
</tr>
<tr>
<td>Get element?</td>
<td><code>arr[i]</code></td>
<td><code>list.get(i)</code></td>
</tr>
<tr>
<td>Set element?</td>
<td><code>arr[i] = value</code></td>
<td><code>list.set(i, value)</code></td>
</tr>
<tr>
<td>Loop?</td>
<td><code>for(int i = 0; i &lt; arr.length; i++)</code></td>
<td><code>for(String value : list)</code></td>
</tr>
</tbody>
</table>
Array vs. ArrayLists

Why do both of these exist in the language?
● Arrays are Java's fundamental data storage
● ArrayList is a library built on top of an array

When would you choose an array over an ArrayList?
● When you need a fixed size that you know ahead of time
  ○ Simpler syntax for getting/setting, more efficient
● Multi-dimensional arrays (e.g. images)
● Histograms/tallying
[Extra Practice] Picking Berries
[Extra Practice] Picking Berries
When you don’t know how many are coming to the party

I love ArrayLists!!
I brought all my friends

hedgehogPartyList
Plan for Today

- Review: 2D Arrays
- ArrayLists
- Example: Reversible Writing
- Example: Trip Planner
- ArrayLists vs. Arrays

Next Time: HashMaps!