CS 106A, Lecture 19
ArrayLists

suggested reading:
Java Ch. 11.8
Learning Goals

• Know how to store data in and retrieve data from an `ArrayList`.
Plan for today

• Recap: Tic-Tac-Toe
• ArrayLists
• *Example*: reversible writing
• *Example*: planner
• ArrayLists vs. arrays
• Recap
Plan for today

• Recap: Tic-Tac-Toe
• ArrayLists
• *Example*: reversible writing
• *Example*: planner
• ArrayLists vs. arrays
• Recap
Let’s use 2D arrays to create a ConsoleProgram version of Tic-Tac-Toe.
Plan for today

• Recap: Tic-Tac-Toe
• ArrayLists
  • Example: reversible writing
  • Example: planner
• ArrayLists vs. arrays
• Recap
Limitations of Arrays

• Size must be specified upon creation
• Can’t add/remove/insert elements later (because size is fixed)
• No built-in methods for printing, searching etc.
  – Mostly solved with Arrays methods, but they’re not built in

<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>
Introducing... ArrayLists!

• A variable type that represents a list of items
• You access individual items by index
  – Ordered
• Store a single type of Object (String, GRect, etc.)
  – Homogenous, but extra caveat: Objects only!
• Resizable – can add and remove elements
• Has helpful methods for printing, searching, etc.
Our First ArrayList

ArrayList<String> myArrayList = new ArrayList<>();
import java.util.*;

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

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

Type of items your ArrayList will store.

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

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

Our First ArrayList

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

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

Could contain the type of items your ArrayList will store, but you can leave it empty because of type inference.
Our First ArrayList

ArrayList<String> myArrayList = new ArrayList<>();
// Create an (initially empty) list
ArrayList<String> list = new ArrayList<>();
Our First ArrayList

// Create an (initially empty) list
ArrayList<String> list = new ArrayList<>();

// Add an element to the back
list.add("Hello");  // now size 1

“Hello”
Our First ArrayList

// Create an (initially empty) list
ArrayList<String> list = new ArrayList<>();

// Add an element to the back
list.add("Hello");  // now size 1

list.add("there!");  // now size 2
// Add an element to the back
list.add("Hello"); // now size 1

"Hello"

list.add("there!"); // now size 2

"Hello" "there!"

// Access elements by index (starting at 0!)
println(list.get(0)); // prints "Hello"
println(list.get(1)); // prints "there!"
println(list); // prints ["Hello", "there!"]
// Add an element to the back
list.add("Hello"); // now size 1

"Hello"

list.add("there!"); // now size 2

"Hello"  "there!"

// Access elements by index (starting at 0!)
for (int i = 0; i < list.size(); i++) {
    println(list.get(i));
}
// Add an element to the back
list.add("Hello"); // now size 1

"Hello"

list.add("there!"); // now size 2

"Hello" "there!"

// Access elements by index (starting at 0!)
for (int i = 0; i < list.size(); i++) {
    println(list.get(i));
}
Our First ArrayList

// Add an element to the back
list.add("Hello");  // now size 1

"Hello"

list.add("there!");  // now size 2

"Hello"  "there!"

// Access elements in order (also for arrays!)
for (String str : list) {
    println(str);
}

// Access elements in order (also for arrays!)
for (String str : list) {
    println(str);
}

// equivalent to

for (int i = 0; i < list.size(); i++) {
    String str = list.get(i);
    println(str);
}
Iterating Over ArrayLists

// Access elements in order (also for arrays!)
for (String str : list) {
    println(str);
}

// equivalent to

for (int i = 0; i < list.size(); i++) {
    String str = list.get(i);
    println(str);
}
// Create an (initially empty) list
ArrayList<String> list = new ArrayList<>();

// Wrong type - bad times! Won't compile
GLabel label = new GLabel("Hello there!");
list.add(label);

// Invalid index! IndexOutOfBoundsException Exception
println(list.get(2));
Plan for today

• Recap: Tic-Tac-Toe
• ArrayLists

• Example: reversible writing
• Example: planner
• ArrayLists vs. arrays
• Recap
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 Have a Dream" by Antonia Lee, Sara Fung, Christy Fung, Rachel Lam
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
Example: Reversible Writing

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

“\textquote{I am not a person who contributes}”
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""
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: 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” |

Key idea: print the ArrayList items in reverse order
Example: Reversible Writing

```java
String filename = promptUserForFile("Filename: ", "res");
try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

    // 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));
    }
} catch (IOException ex) {
    println("Could not read file.");
}
```
String filename = promptUserForFile("Filename: ", "res");
try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

    // 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));
    }
} catch (IOException ex) {
    println("Could not read file.");
}
String filename = promptUserForFile("Filename: ", "res");

try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

    // 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));
    }
} catch (IOException ex) {
    println("Could not read file.");
}
String filename = promptUserForFile("Filename: ", "res");

try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

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

String filename = promptUserForFile("Filename: ", "res");
try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

    // 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));
    }
} catch (IOException ex) {
    println("Could not read file.");
}
String filename = promptUserForFile("Filename: ", "res");
try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

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

```java
String filename = promptUserForFile("Filename: ", "res");
try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

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

```java
String filename = promptUserForFile("Filename: ", "res");
try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

    // 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));
    }
} catch (IOException ex) {
    println("Could not read file.");
}
```
Plan for today

• Recap: Tic-Tac-Toe
• ArrayLists
• *Example*: reversible writing
• *Example*: planner
• ArrayLists vs. arrays
• Recap
### ArrayList Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>list.add(value);</code></td>
<td>appends value at end of list</td>
</tr>
<tr>
<td><code>list.add(index, value);</code></td>
<td>inserts given value just before the given index, shifting subsequent values to the right</td>
</tr>
<tr>
<td><code>list.clear();</code></td>
<td>removes all elements of the list</td>
</tr>
<tr>
<td><code>list.get(index)</code></td>
<td>returns the value at given index</td>
</tr>
<tr>
<td><code>list.indexOf(value)</code></td>
<td>returns first index where given value is found in list (-1 if not found)</td>
</tr>
<tr>
<td><code>list.isEmpty()</code></td>
<td>returns <code>true</code> if the list contains no elements</td>
</tr>
<tr>
<td><code>list.remove(index);</code></td>
<td>removes/returns value at given index, shifting subsequent values to the left</td>
</tr>
<tr>
<td><code>list.remove(value);</code></td>
<td>removes the first occurrence of the value, if any</td>
</tr>
<tr>
<td><code>list.set(index, value);</code></td>
<td>replaces value at given index with given value</td>
</tr>
<tr>
<td><code>list.size()</code></td>
<td>returns the number of elements in the list</td>
</tr>
<tr>
<td><code>list.toString()</code></td>
<td>returns a string representation of the list such as &quot;[3, 42, -7, 15]&quot;</td>
</tr>
</tbody>
</table>
• If you insert/remove in the front or middle of a list, elements **shift** to fit.

```java
list.add(2, 42);
```

• shift elements right to make room for the new element

<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>

```java
list.remove(1);
```

• shift elements left to cover the space left by the removed element

<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>42</td>
<td>9</td>
<td>7</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>42</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Example: Planner

Enter task: sleep
Enter task: prepare for lecture
Enter task: play Zelda
Enter task: go for a bike ride
Enter task: walk Daisy
Enter task:
Great! Enter the order to complete your tasks.
Tasks remaining: [sleep, prepare for lecture, play Zelda, go for a bike ride, walk Daisy]
Next task to complete: walk Daisy
Tasks remaining: [sleep, prepare for lecture, play Zelda, go for a bike ride]
Next task to complete: play Zelda
Tasks remaining: [sleep, prepare for lecture, go for a bike ride]
Next task to complete: prepare for lecture
Tasks remaining: [sleep, go for a bike ride]
Next task to complete: go for a bike ride
Tasks remaining: [sleep]
Next task to complete: decorate room
That's not on your list – stay focused!
Tasks remaining: [sleep]
Next task to complete: sleep
Congrats! Your day is all planned out:
[walk Daisy, play Zelda, prepare for lecture, go for a bike ride, sleep]
Example: Planner

• Let’s write a program to help plan out our day
  – The program first prompts for things you want to do today
  – Then, it asks the user to re-input them in order of completion
  – Finally, it outputs the order the user has chosen for their tasks

Enter task: sleep
Enter task: prepare for lecture
Enter task: play Zelda
Enter task: go for a bike ride
Enter task: walk Daisy
Enter task:
Great! Enter the order to complete your tasks.
Tasks remaining: [sleep, prepare for lecture, play Zelda, go for a bike ride, walk Daisy]
Next task to complete: walk Daisy
Tasks remaining: [sleep, prepare for lecture, play Zelda, go for a bike ride]
Next task to complete: play Zelda
Tasks remaining: [sleep, prepare for lecture, go for a bike ride]
Next task to complete: prepare for lecture
Tasks remaining: [sleep, go for a bike ride]
Next task to complete: go for a bike ride
Tasks remaining: [sleep]
Next task to complete: decorate room
That's not on your list - stay focused!
Tasks remaining: [sleep]
Next task to complete: sleep
Congrats! Your day is all planned out:
[walk Daisy, play Zelda, prepare for lecture, go for a bike ride, sleep]
Todos:

“Do crossword”
Planner: Approach

<table>
<thead>
<tr>
<th>Todos:</th>
<th>“Do crossword”</th>
<th>“Sleep”</th>
</tr>
</thead>
</table>

### Planner: Approach

**Todos:**

<table>
<thead>
<tr>
<th>“Do crossword”</th>
<th>“Sleep”</th>
<th>“Talk to Annie”</th>
</tr>
</thead>
</table>


Todos:

- “Do crossword”
- “Sleep”
- “Talk to Annie”

Order:

- “Do crossword”
Planner: Approach

Todos:

Order:

“Sleep”  “Talk to Annie”

“Do crossword”
Planner: Approach

Todos:

Order:

“Sleep”

“Do crossword”  “Talk to Annie”
Planner: Approach

Todos:
DONE!

Order:
“Do crossword” “Talk to Annie” “Sleep”
Plan for today

• Recap: Tic-Tac-Toe
• ArrayLists
• *Example*: reversible writing
• *Example*: planner
• ArrayLists vs. arrays
• Recap
Unlike arrays, ArrayLists can only store objects!
ArrayLists + Primitives = 🥰

<table>
<thead>
<tr>
<th>Primitive</th>
<th>&quot;Wrapper&quot; 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> list = new ArrayList<>();

// Java converts Integer <-> int automatically!
int num = 123;
list.add(num);

int first = list.get(0);  // 123

Conversion happens automatically!
## Array vs. ArrayList

<table>
<thead>
<tr>
<th>ArrayList</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ArrayList&lt;Integer&gt;</code> list = new ArrayList&lt;&gt;();</td>
<td>int[] arr = new int[2]; // [0, 0]</td>
</tr>
<tr>
<td>list.add(1);         // [1]</td>
<td>arr[0] = 1; // [1, 0]</td>
</tr>
<tr>
<td>list.add(2);         // [1, 2]</td>
<td>arr[1] = 2; // [1, 2]</td>
</tr>
<tr>
<td>list.set(0, 3);      // [3, 2]</td>
<td>arr[0] = 3; // [3, 2]</td>
</tr>
<tr>
<td>int x = list.get(0); // 3</td>
<td>int x = arr[0]; // 3</td>
</tr>
<tr>
<td>list.add(4);         // [3, 2, 4]</td>
<td>[no equivalent]</td>
</tr>
<tr>
<td>list.contains(2);    // true</td>
<td></td>
</tr>
</tbody>
</table>
Array vs. ArrayList

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
Plan for today

• Recap: Tic-Tac-Toe
• ArrayLists
• *Example*: reversible writing
• *Example*: planner
• ArrayLists vs. arrays
• Recap
Recap

• ArrayLists are a variable type representing a list of items
• Unlike arrays, ArrayLists have:
  – The ability to resize dynamically
  – Useful methods you can call on them
• Unlike ArrayLists, arrays have:
  – The ability to store any type of item, not just objects

Next Time: HashMaps