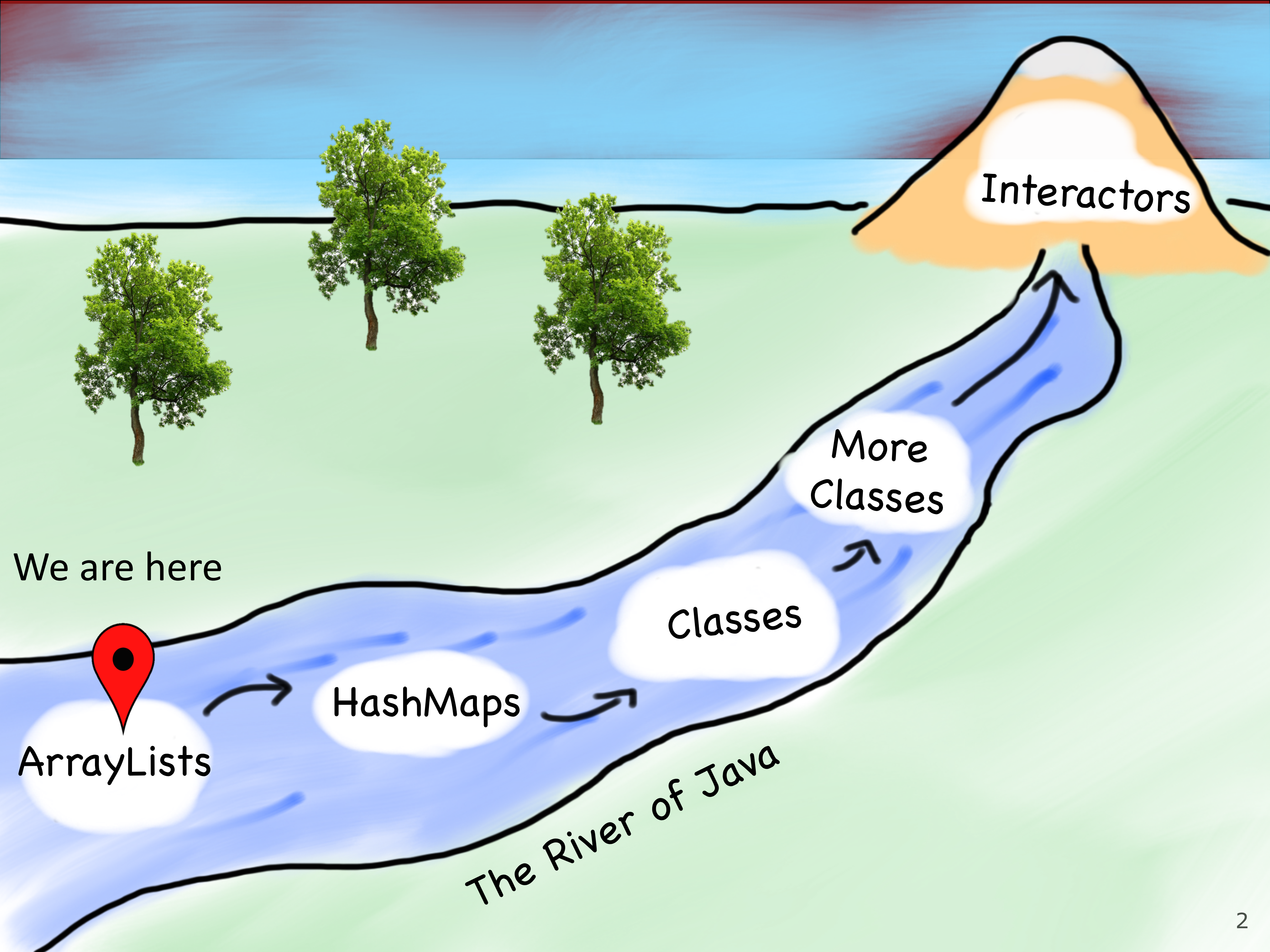
CS 106A, Lecture 19
ArrayLists

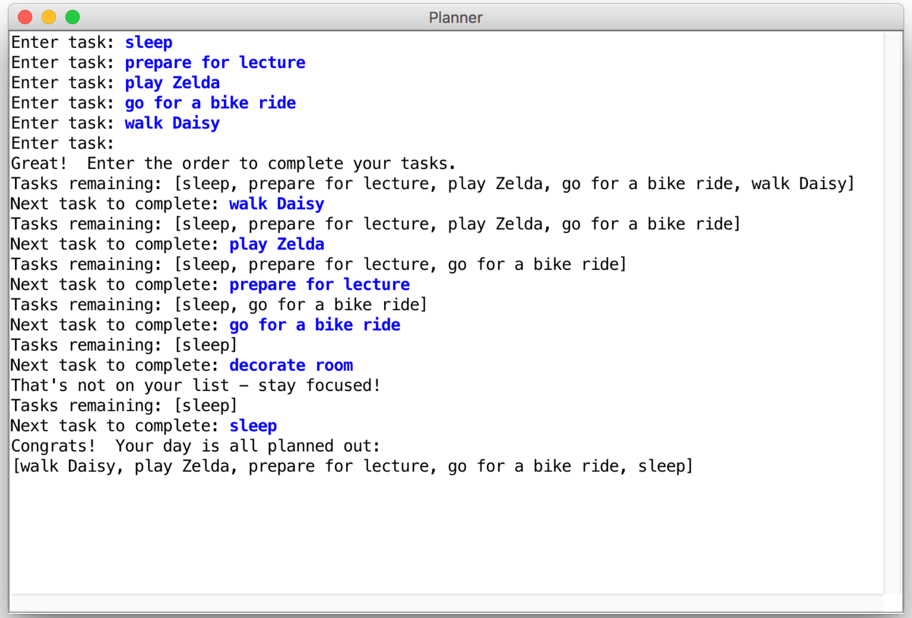
suggested reading:

Java Ch. 11.8


Interactors
More Classes
We are here
Classes
HashMaps
ArrayLists
The River of Java

Learning Goals

- Know how to store data in and retrieve data from an **`ArrayList`**.



```
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]
```

Plan for today

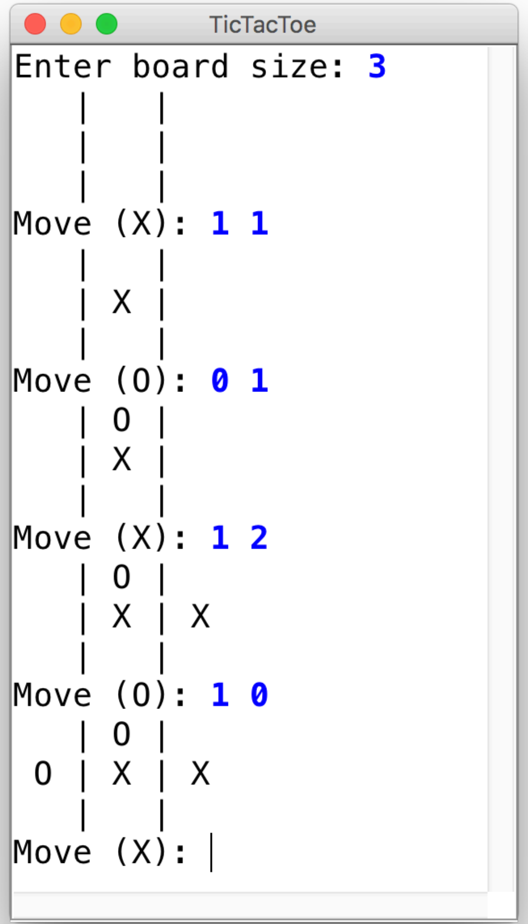
- Recap: Tic-Tac-Toe
- ArrayLists
- *Example*: reversible writing
- *Example:* planner
- ArrayLists vs. arrays
- Recap

Plan for today

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

Tic-Tac-Toe

Let’s use 2D arrays to create a ConsoleProgram version of Tic-Tac-Toe.



TicTacToe

```
Enter board size: 3
   |   |
   |   |
   |   |
Move (X): 1 1
   |   |
   | X |
   |   |
Move (O): 0 1
   | O |
   | X |
   |   |
Move (X): 1 2
   | O |
   | X | X
   |   |
Move (O): 1 0
   | O |
 O | X | X
   |   |
Move (X): 
```

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
- No built-in methods for searching, etc.
- Can’t print arrays without Arrays.toString (or Arrays.deepToString)

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
value	12	49	-2	26	5	17	-6	84	72	3

Introducing… ArrayLists!

- A variable type that represents a list of items.
- You access individual items by *index*.
- Store a single type of **object** (String, GRect, etc.)
- Resizable – can add and remove elements
- Has helpful methods for searching for items

Our First ArrayList

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

Our First ArrayList

```
import java.util.*;
```

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

Our First ArrayList

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

Our First ArrayList

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

Our First ArrayList

Type of items your ArrayList will store.

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

Our First ArrayList

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

Our First ArrayList

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

Our First ArrayList

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

Our First ArrayList

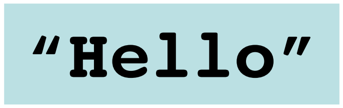
```
// Create an (initially empty) list
ArrayList<String> list = new ArrayList<>();
```

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
```

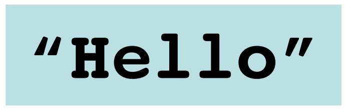


Our First ArrayList

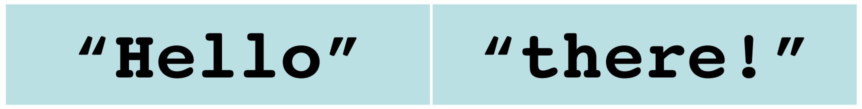
```
// Create an (initially empty) list
ArrayList<String> list = new ArrayList<>();



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

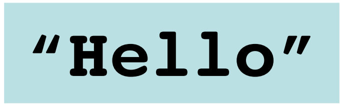


```
list.add("there!");    // now size 2
```

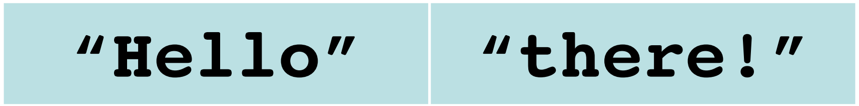


Our First ArrayList

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



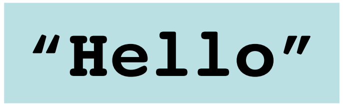
```
list.add("there!");    // now size 2
```



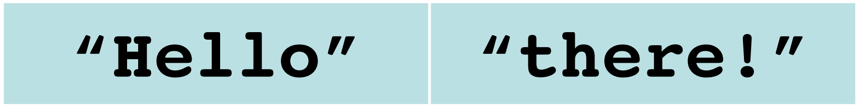
```
// Access elements by index (starting at 0!)
println(list.get(0));       // prints "Hello"
println(list.get(1));       // prints "there!”
println(list);         // prints ["Hello", "there!"]
```

Our First ArrayList

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



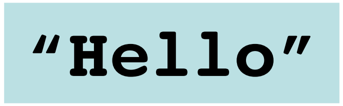
```
list.add("there!");    // now size 2
```



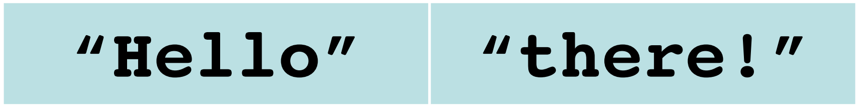
```
// Access elements by index (starting at 0!)
for (int i = 0; i < list.size(); i++) {
 println(list.get(i));
}
```

Our First ArrayList

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



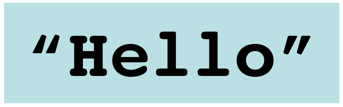
```
list.add("there!");    // now size 2
```



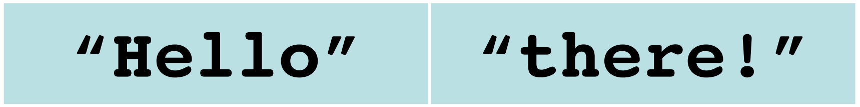
```
// Access elements by index (starting at 0!)
for (int i = 0; i < list.size(); i++) {
  println(list.get(i));
}
```

Our First ArrayList

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



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



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

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);
}
```

Bad Times with ArrayLists

```
// 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!  IndexOutOfBounds 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

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
http://poets.spice.org.hk/index.php?option=com_content&view=article&id=45:my-family&catid=6:reverse-poem&Itemid=7

Example: Reversible Writing

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

“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

```
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.");
}
```

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.");
}
```

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.");
}
```

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.");
}
```

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.");
}
```

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.");
}
```

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.");
}
```

Plan for today

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

ArrayList Methods

`list.add(value);`	appends value at end of list
`list.add(index, value);`	inserts given value just before the given index, shifting subsequent values to the right
`list.clear();`	removes all elements of the list
`list.get(index)`	returns the value at given index
`list.indexOf(value)`	returns first index where given value is found in list (-1 if not found)
`list.isEmpty()`	returns `true` if the list contains no elements
`list.remove(index);`	removes/returns value at given index, shifting subsequent values to the left
`list.remove(value);`	removes the first occurrence of the value, if any
`list.set(index, value);`	replaces value at given index with given value
`list.size()`	returns the number of elements in the list
`list.toString()`	returns a string representation of the list such as `"[3, 42, -7, 15]"`

Insert/remove

- If you insert/remove in the front or middle of a list, elements **shift** to fit.

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

  - shift elements right to make room for the new element

index	0	1	2	3	4
value	3	8	9	7	5

$\rightarrow$

index	0	1	2	3	4	5
value	3	8	**42**	9	7	5

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

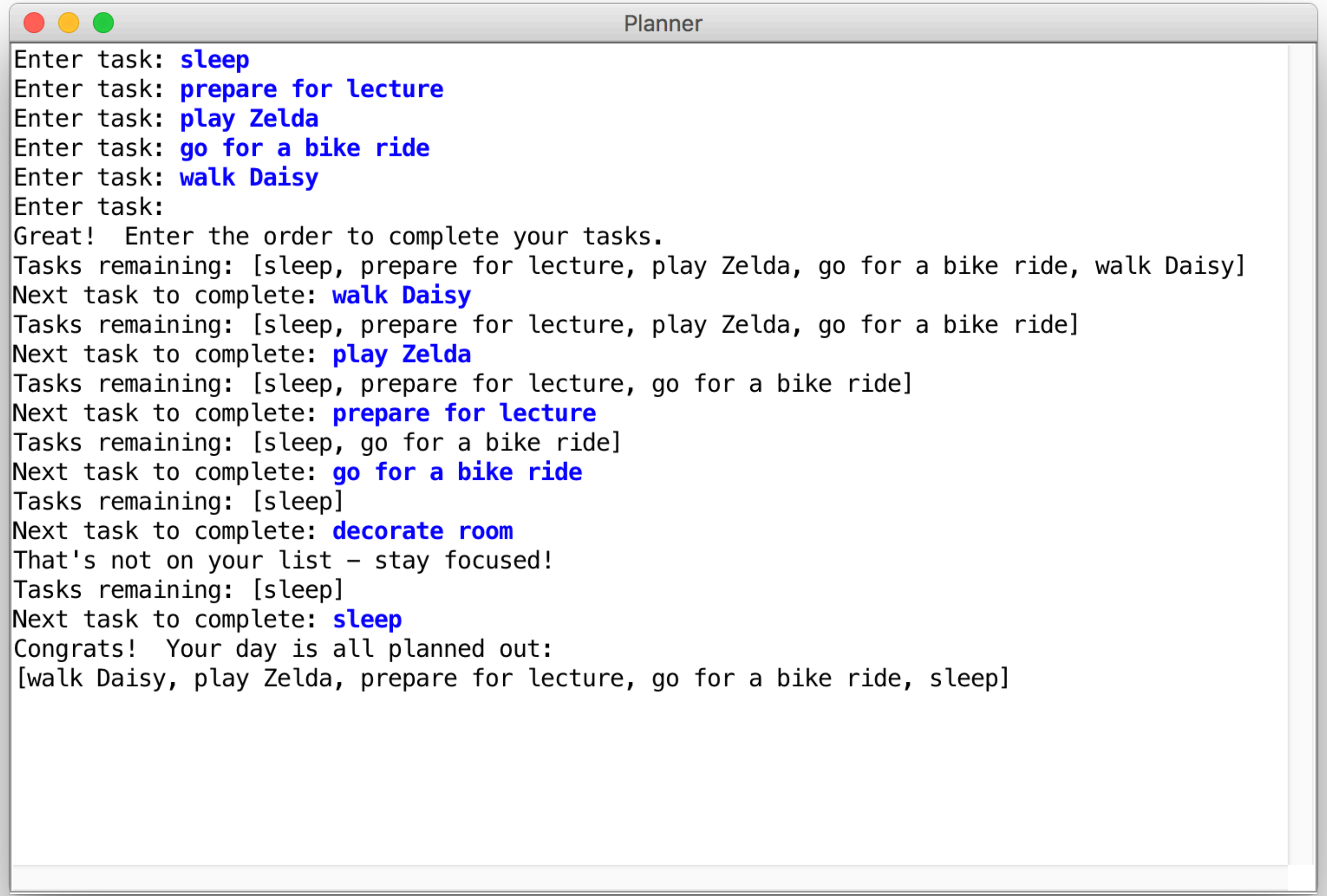
  - shift elements left to cover the space left by the removed element

index	0	1	2	3	4	5
value	3	~~**8**~~	42	9	7	5

$\rightarrow$

index	0	1	2	3	4
value	3	42	9	7	5

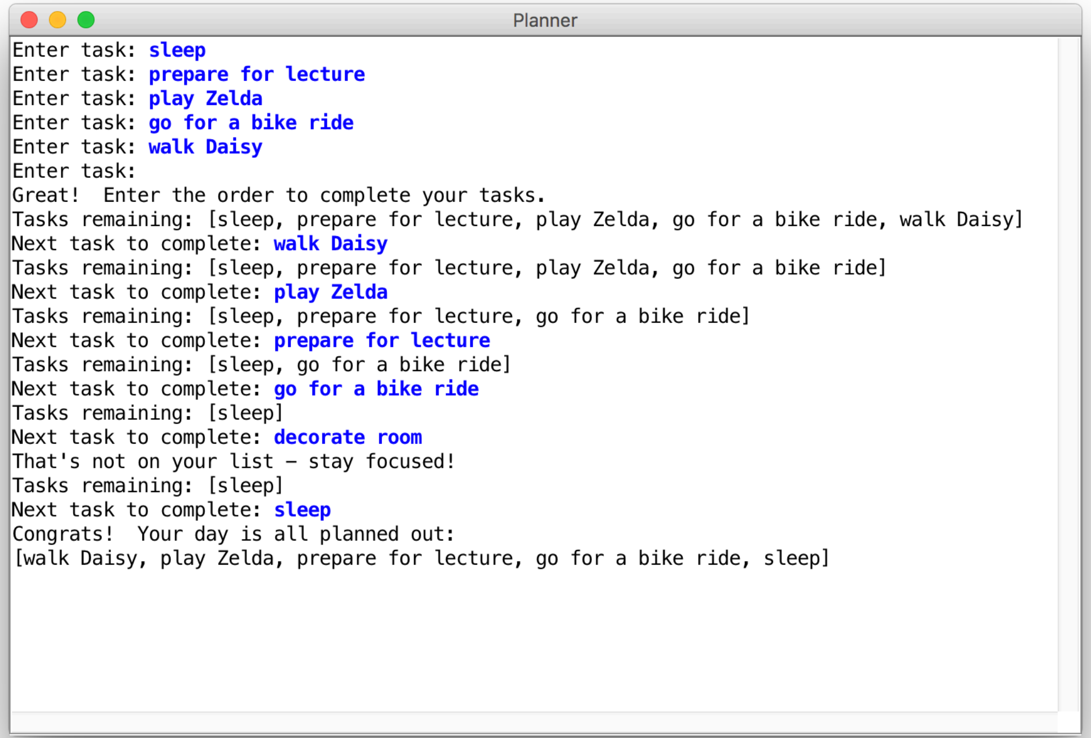
Example: Planner


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]
```

Planner: Approach

Todos:

```
“Walk
Daisy”
```

Planner: Approach

Todos:

`“Walk Daisy”`	`“Play Zelda”`

Planner: Approach

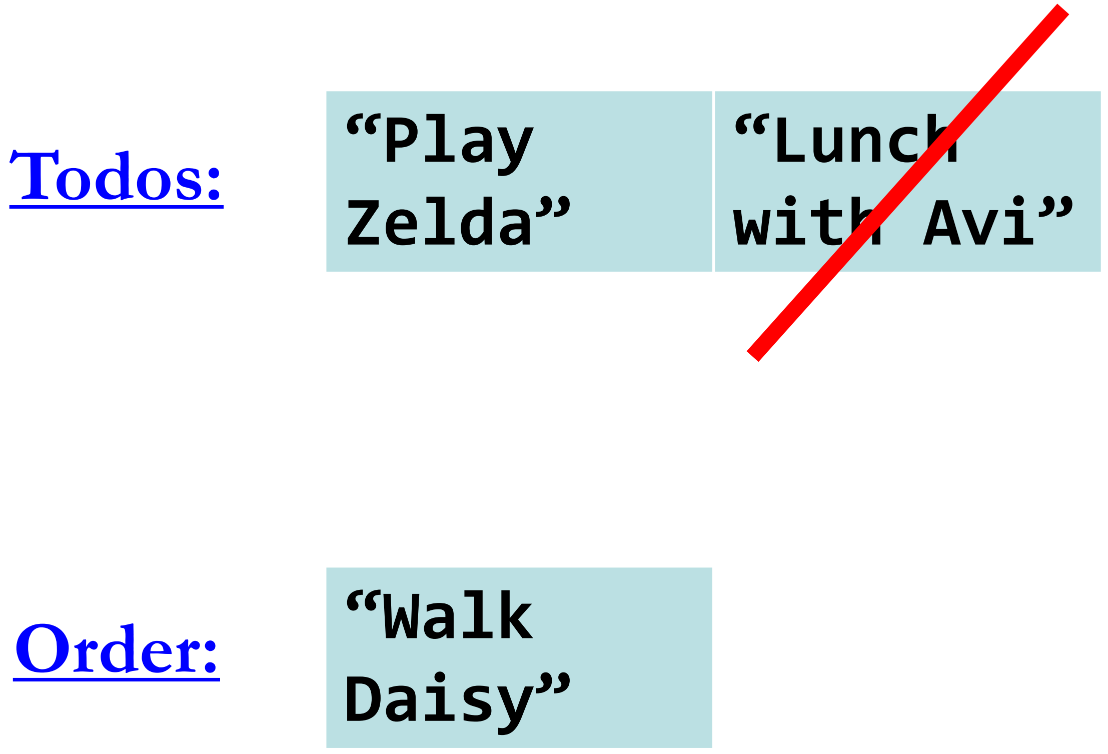
Todos:

“Walk Daisy”	“Play Zelda”	“Lunch with Avi”

Planner: Approach

Todos:	~~“Walk Daisy”~~	“Play Zelda”	“Lunch with Avi”
Order:	“Walk Daisy”		

Planner: Approach



Planner: Approach

Todos:

~~“Play Zelda”~~

Order:

“Walk Daisy”	“Lunch with Avi”

Planner: Approach

Todos: `DONE!`

Order:

“Walk Daisy”	“Lunch with Avi”	“Play Zelda”

Plan for today

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

ArrayLists + Primitives = 💔

```
// Doesn’t compile ☹
ArrayList<int> list = new ArrayList<>();
```

Unlike arrays, ArrayLists can only store **objects!**

ArrayLists + Primitives = 💔

Primitive	“Wrapper” Class
`int`	`Integer`
`double`	`Double`
`boolean`	`Boolean`
`char`	`Character`

ArrayLists + Wrappers = ❤️

```
// 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

ArrayList	**Array**
`ArrayList<Integer> list =`	`int[] arr =`
`   new ArrayList<>();`	`   new int[2];  // [0, 0]`
`list.add(1);      // [1]`	`arr[0] = 1;      // [1, 0]`
`list.add(2);      // [1, 2]`	`arr[1] = 2;      // [1, 2]`
`list.set(0, 3);   // [3, 2]`	`arr[0] = 3;      // [3, 2]`
`int x = list.get(0); // 3`	`int x = arr[0]; // 3`
`list.add(4);     // [3, 2, 4]`	`[no equivalent]`
`list.contains(2); // true`	

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