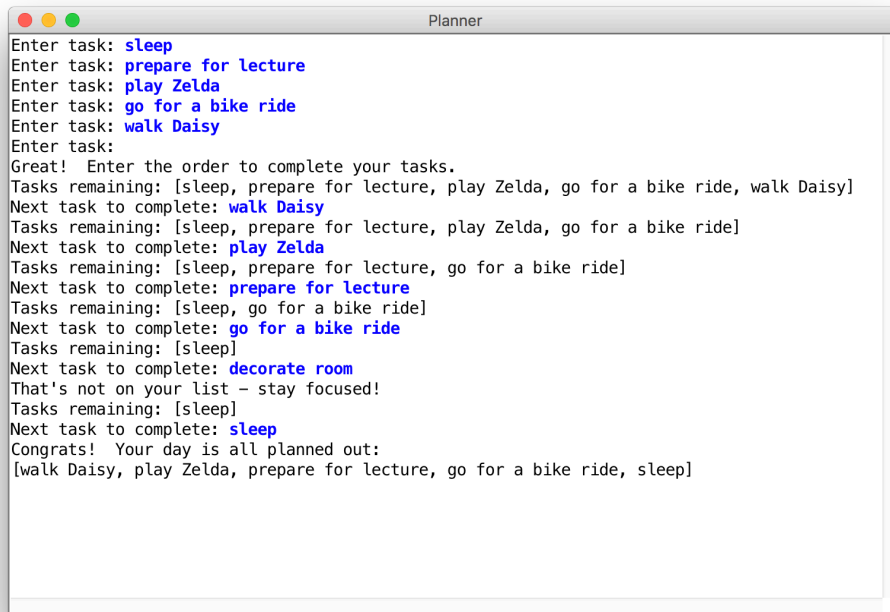# ArrayLists

Nick Troccoli

Reading:
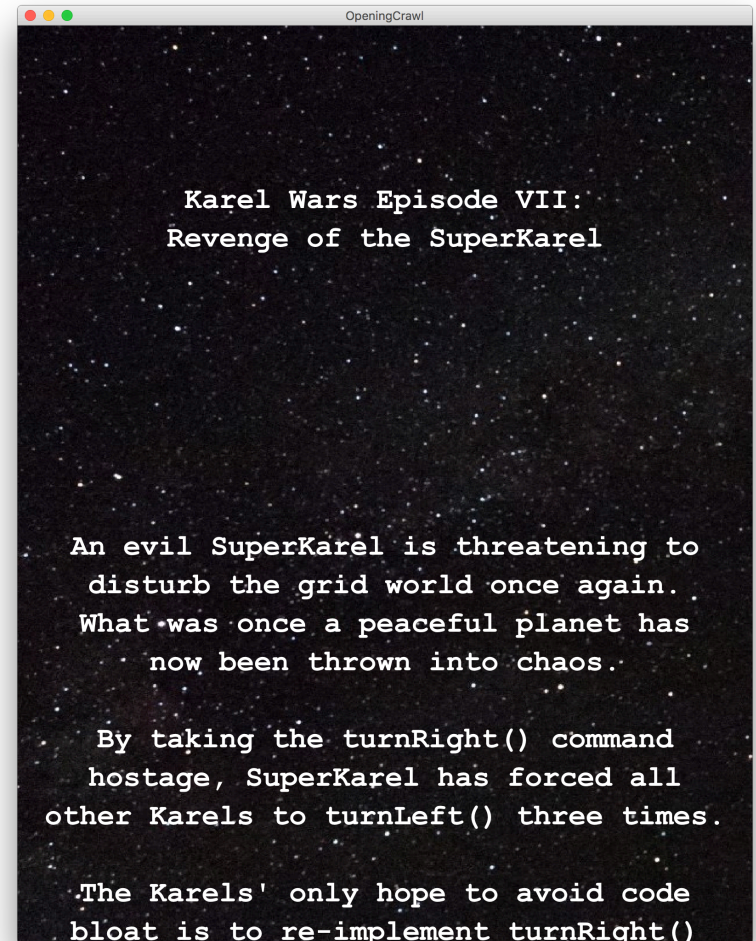
Art & Science of Java, Ch. 11.8

Learning Goals

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

```
                              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]
```

```
                              OpeningCrawl

            Karel Wars Episode VII:
            Revenge of the SuperKarel




        An evil SuperKarel is threatening to
         disturb the grid world once again.
         What was once a peaceful planet has
              now been thrown into chaos.


          By taking the turnRight() command
          hostage, SuperKarel has forced all
        other Karels to turnLeft() three times.

           The Karels' only hope to avoid code
          bloat is to re-implement turnRight()
```

Plan for today

- Arrays review

- ArrayLists

- *Example*: reversible writing

- *Example:* planner

- ArrayLists vs. arrays

- *Example:* opening crawl

- Recap

Plan for today

- Arrays review
- ArrayLists
- *Example*: reversible writing
- *Example:* planner
- ArrayLists vs. arrays
- *Example:* opening crawl
- Recap

Previously: Arrays

- An **array** is a variable type that represents a list of items.
- You access individual items in an array by *index*.
- Store a single type of item (int, double, GRect, etc.)

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

Limitations of Arrays

- Size must be specified upon creation
- Can't add/remove/insert elements later
- No handy methods for searching, etc.

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

Plan for today

• Arrays review

• **ArrayLists**

• *Example*: reversible writing

• *Example:* planner

• ArrayLists vs. arrays

• *Example:* opening crawl

• Recap

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<String>();
```

Our First ArrayList

```
import java.util.*;



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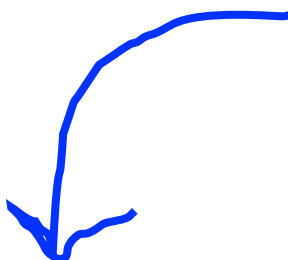
Our First ArrayList

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

Our First ArrayList

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

Our First ArrayList

Type of items 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

Same type here, but followed by ( ).

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

Our First ArrayList

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

Our First ArrayList

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

Our First ArrayList

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

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

Our First ArrayList

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

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

Our First ArrayList

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

// 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!"
```

Our First ArrayList

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

// 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

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

// 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

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

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

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

Bad Times with ArrayLists

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

Bad Times with ArrayLists

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

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

Bad Times with ArrayLists

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

// 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

- Arrays review

- ArrayLists

- *Example*: reversible writing

- *Example:* planner

- ArrayLists vs. arrays

- *Example:* opening crawl

- 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

Example: Reversible Writing

```java
String filename = readLine("Enter filename: ");
try {
  Scanner s = 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));
  }
} catch (FileNotFoundException ex) {
  println("Could not find file.");
}
```

Example: Reversible Writing

```java
String filename = readLine("Enter filename: ");
try {
  Scanner s = 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));
  }
} catch (FileNotFoundException ex) {
  println("Could not find file.");
}
```

Example: Reversible Writing

```java
String filename = readLine("Enter filename: ");
try {
  Scanner s = 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));
  }
} catch (FileNotFoundException ex) {
  println("Could not find file.");
}
```

Example: Reversible Writing

```java
String filename = readLine("Enter filename: ");
try {
  Scanner s = 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));
  }
} catch (FileNotFoundException ex) {
  println("Could not find file.");
}
```

Example: Reversible Writing

```java
String filename = readLine("Enter filename: ");
try {
  Scanner s = 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));
  }
} catch (FileNotFoundException ex) {
  println("Could not find file.");
}
```

Example: Reversible Writing

```
String filename = readLine("Enter filename: ");
try {
  Scanner s = 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));
  }
} catch (FileNotFoundException ex) {
  println("Could not find file.");
}
```

Example: Reversible Writing

```
String filename = readLine("Enter filename: ");
try {
  Scanner s = 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));
  }
} catch (FileNotFoundException ex) {
  println("Could not find file.");
}
```

Example: Reversible Writing

```java
String filename = readLine("Enter filename: ");
try {
  Scanner s = 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));
  }
} catch (FileNotFoundException ex) {
  println("Could not find file.");
}
```

Plan for today

- Arrays review

- ArrayLists

- *Example*: reversible writing

- *Example:* planner

- ArrayLists vs. arrays

- *Example:* opening crawl

- Recap

ArrayList Methods

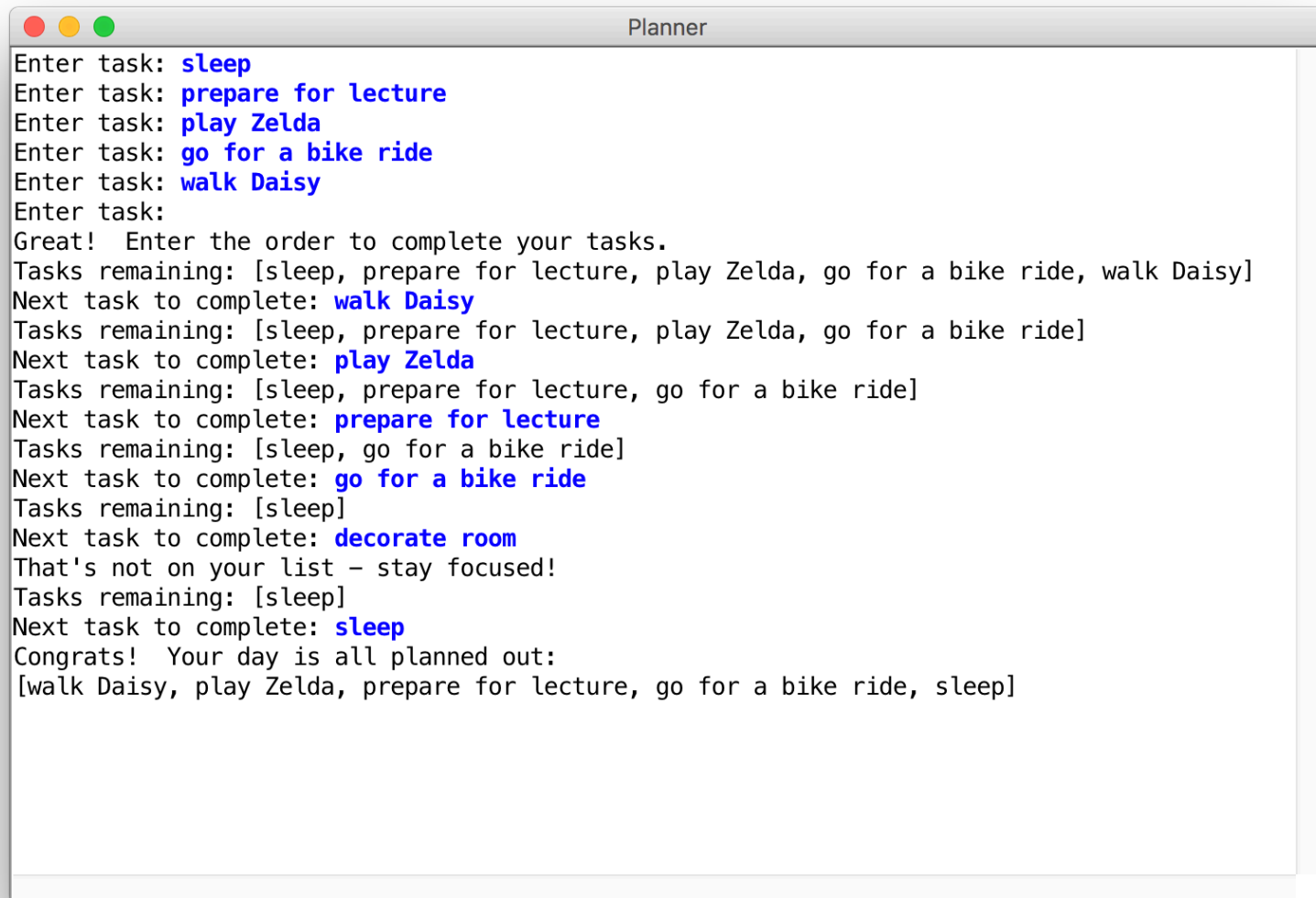|	
List.add(*value*);	appends value at end of list
List.add(*index, value*);	inserts given value just before the given index, shifting subsequent values to the right
List.clear();	removes all elements of the list
List.get(*index*)	returns the value at given index
List.indexOf(*value*)	returns first index 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

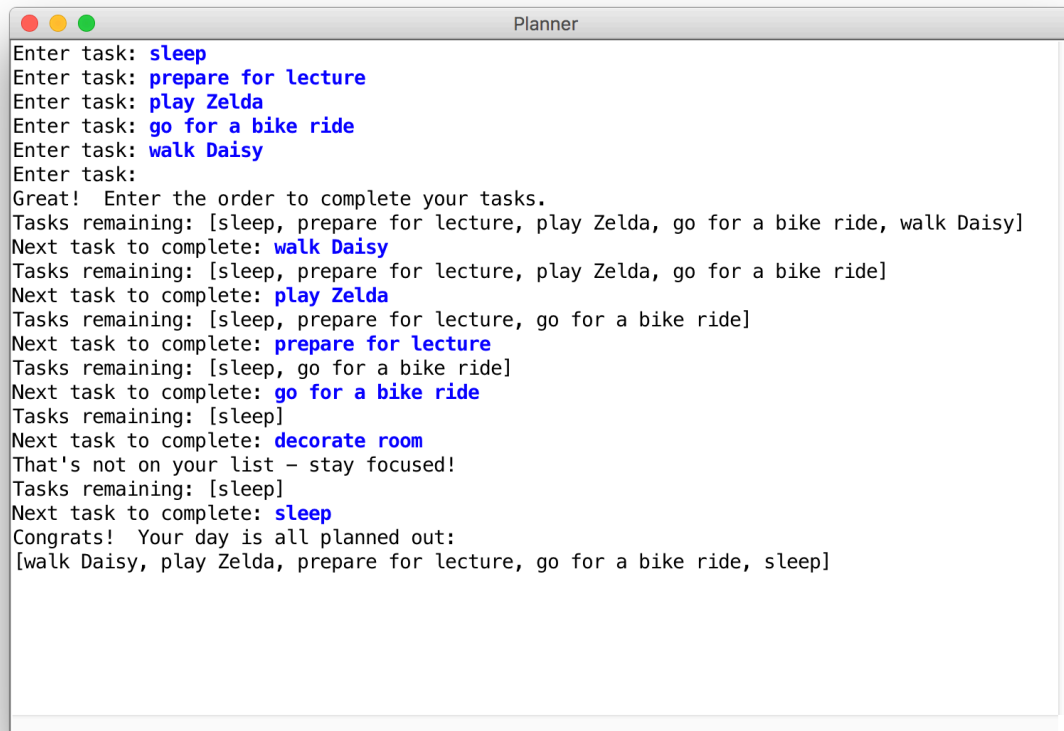| index	0	1	2	3	4	5
value	3	8	42	9	7	5
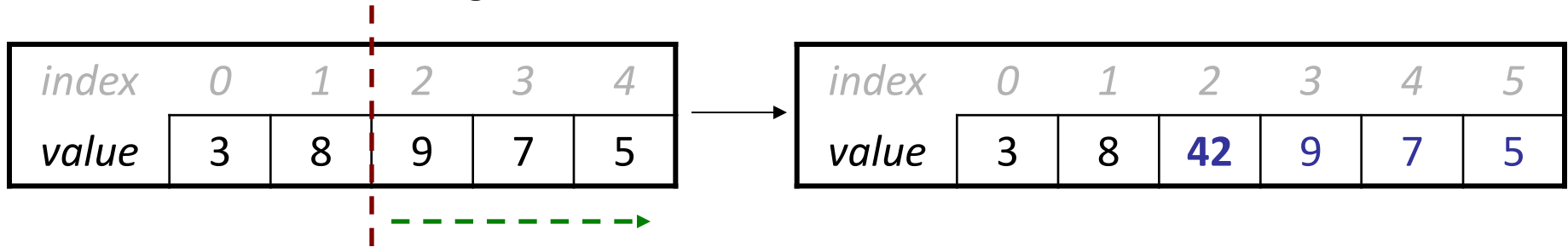
Example: Planner


```
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]
```

Plan for today

•Arrays review

•ArrayLists

•*Example*: reversible writing

•*Example:* planner

•**ArrayLists vs. arrays**

•*Example:* opening crawl

•Recap

ArrayLists + Primitives = 💔

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

Unlike arrays, ArrayLists can only

store **objects!**

ArrayLists + Primitives = 💔

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

ArrayLists + Wrappers = ❤️

```java
// Use wrapper classes when making an ArrayList
ArrayList<Integer> list = new ArrayList<Integer>();
```

ArrayLists + Wrappers = ❤️

```java
// Use wrapper classes when making an ArrayList
ArrayList<Integer> list = new ArrayList<Integer>();

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

ArrayLists + Wrappers = ❤️

```java
// Use wrapper classes when making an ArrayList
ArrayList<Integer> list = new ArrayList<Integer>();

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

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

ArrayLists + Wrappers = ❤️

```java
// Use wrapper classes when making an ArrayList
ArrayList<Integer> list = new ArrayList<Integer>();

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

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

Conversion happens automatically!

Array vs. ArrayList

ArrayList

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

```
list.add(1);        // [1]
list.add(2);        // [1, 2]
```

```
list.set(0, 3);     // [3, 2]
int x = list.get(0); // 3
```

```
list.add(4);      // [3, 2, 4]
list.contains(2); // true
```

Array

```
int[] arr =
    new int[2];  // [0, 0]
```

```
arr[0] = 1;     // [1, 0]
arr[1] = 2;     // [1, 2]
```

```
arr[0] = 3;      // [3, 2]
int x = arr[0]; // 3
```

```
[no equivalent]
```

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)
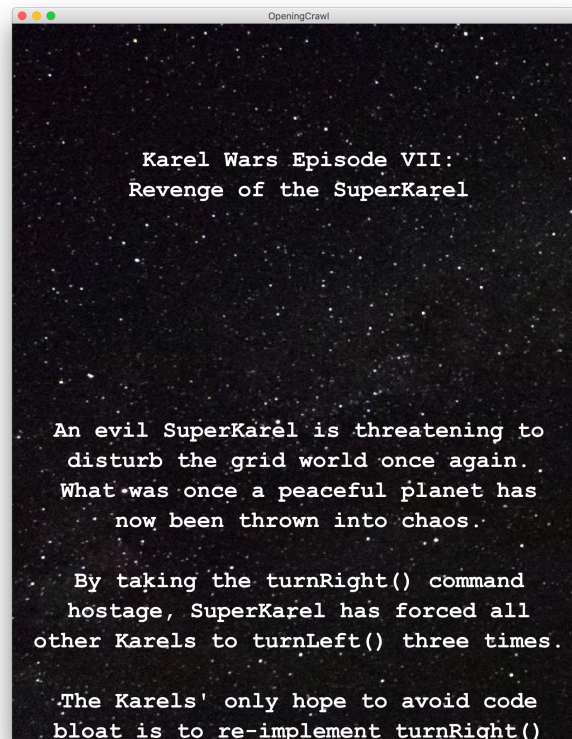- *Histograms*/tallying

Plan for today

- Arrays review

- ArrayLists

- *Example*: reversible writing

- *Example:* planner

- ArrayLists vs. arrays

- *Example:* opening crawl

- Recap

Example: Opening Crawl
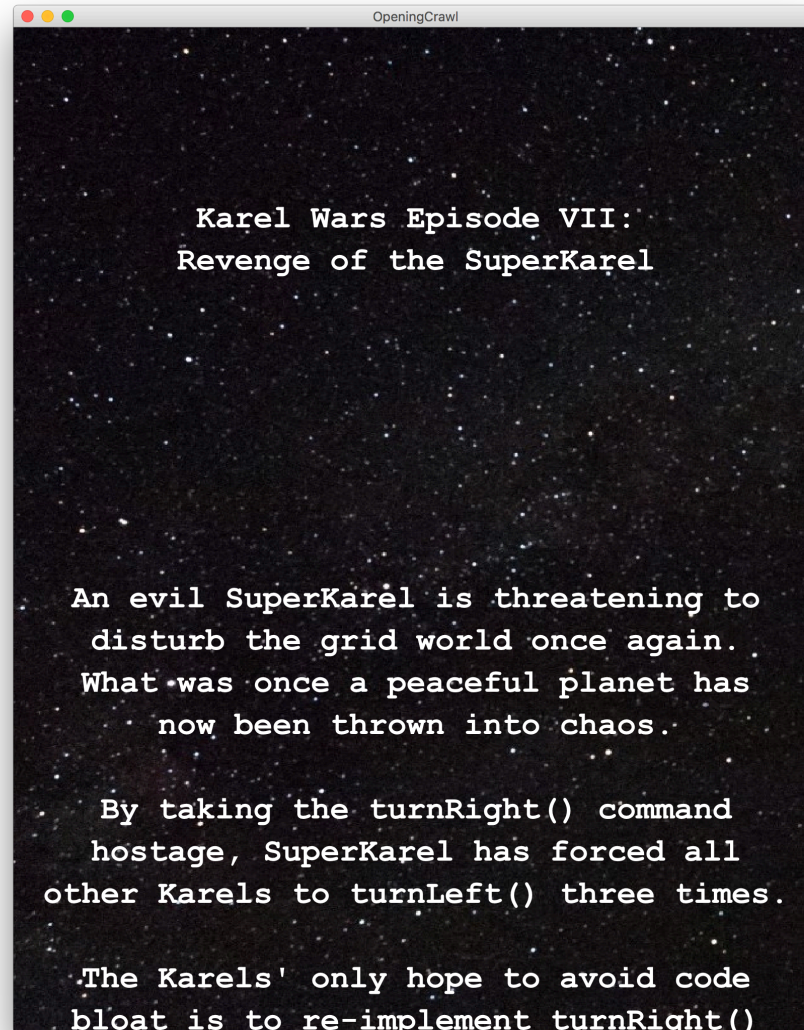
Example: Planner

- Let's write a program that emulates the Star Wars "opening crawl"
  - The program first reads in a text file
  - It then animates this text flowing upwards
  - The text should start and end offscreen

Plan for today

- Arrays review

- ArrayLists

- *Example*: reversible writing

- *Example:* planner

- ArrayLists vs. arrays

- *Example:* opening crawl

- 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