

CS 106A, Lecture 16

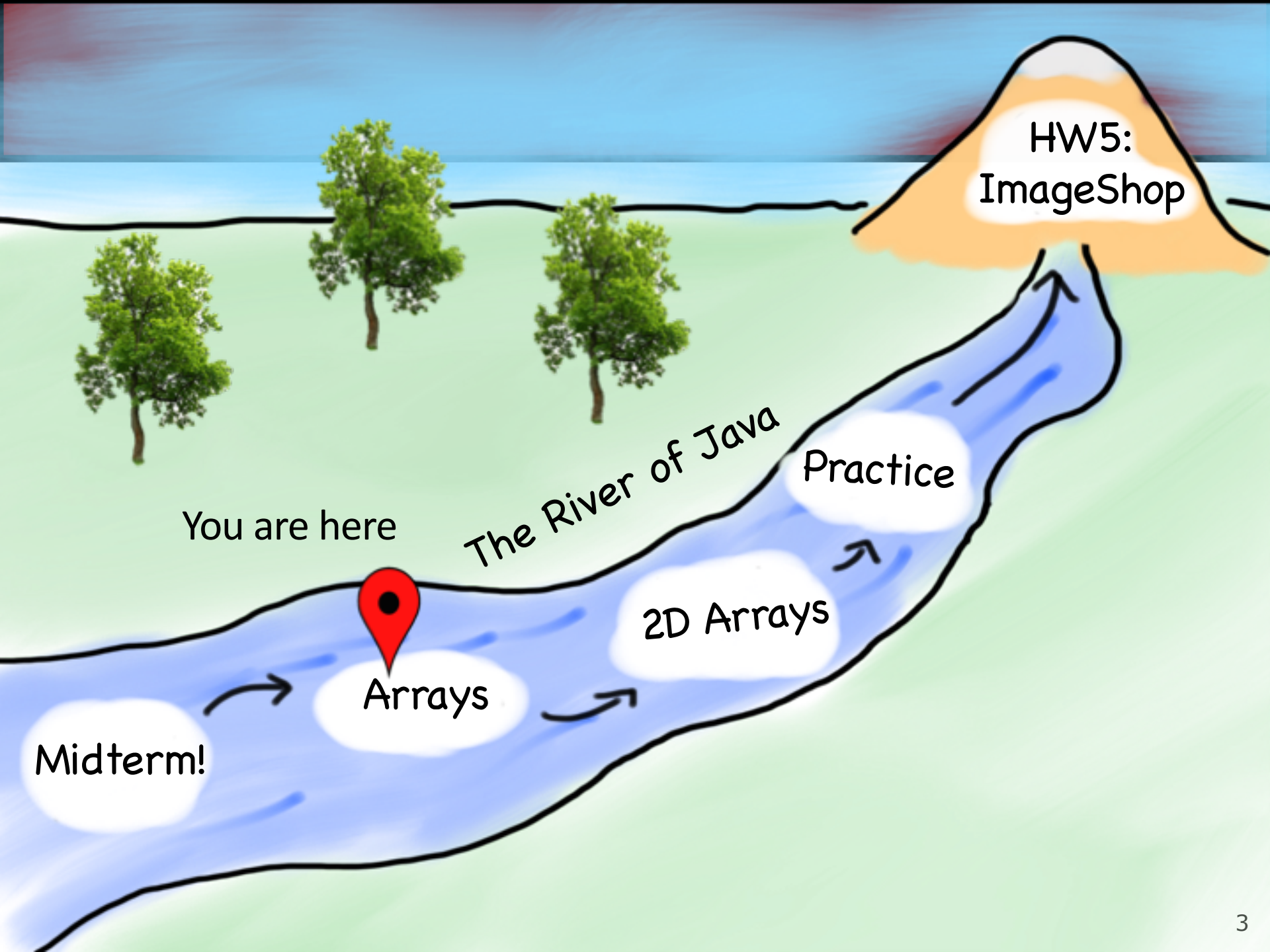
Arrays

suggested reading:

Java Ch. 11.1-11.5

Where Are We in CS 106A?

- Karel the Robot
- Java
- Console Programs
- Text Processing
- Graphics Programs
- Data Structures
- Defining our own Variable Types
- GUIs



Plan for Today

- Data Structures
- Arrays
- Arrays as Parameters and Return Values
- Announcements
- Practice: Swapping Elements
- Practice: WeatherStation
- Recap

Plan for Today

- Data Structures
- Arrays
- Arrays as Parameters and Return Values
- Announcements
- Practice: Swapping Elements
- Practice: WeatherStation
- Recap

What are Data Structures?

Data structures are variable types that can store data in useful ways.

Why Are Data Structures Useful?

Consider a program similar to Weather from HW2 that prompts for daily temperatures and prints averages, high/low, etc.

- Why is this hard to write with what we've learned so far?

How many days' temperatures? 7

Day 1's high temp: 45

Day 2's high temp: 44

Day 3's high temp: 39

Day 4's high temp: 48

Day 5's high temp: 37

Day 6's high temp: 46

Day 7's high temp: 53

All temperatures: [45, 44, 39, 48, 37, 46, 53]

Average temp = 44.6

4 days were above average.

Two coldest days: 37, 39

Two hottest days: 53, 48

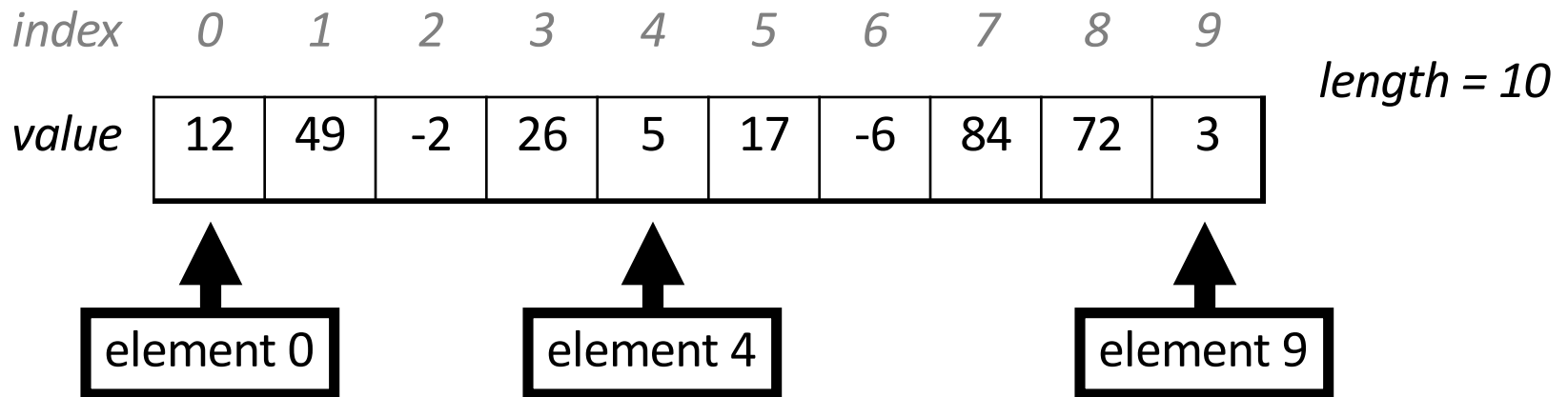
Plan for Today

- Data Structures
- **Arrays**
- Arrays as Parameters and Return Values
- Announcements
- Practice: Swapping Elements
- Practice: WeatherStation
- Recap

Arrays

A new variable type that is an object that represents an ordered, homogeneous list of data.

- Arrays have many *elements* that you can access using *indices*



Creating an Array

```
type[] name = new type[Length];
```

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

<i>index</i>	0	1	2	3	4
<i>value</i>	0	0	0	0	0



Java automatically initializes elements to **0**.

Accessing Data In An Array

name[*index*] // get element at *index*

- Like Strings, indices go from **0** to the **array's length - 1**.

```
for (int i = 0; i < 7; i++) {  
    println(numbers[i]);  
}
```

```
println(numbers[9]);    // exception
```

```
println(numbers[-1]);    // exception
```

<i>index</i>	0	1	2	3	4	5	6
<i>value</i>	0	1	2	3	4	5	6

Putting Data In An Array

name[*index*] = *value*; // set element at *index*

- Like Strings, indices go from **0** to the **array's length - 1**.

```
int[] numbers = new int[7];  
for (int i = 0; i < 7; i++) {  
    numbers[i] = i;  
}  
numbers[8] = 2;    // exception  
numbers[-1] = 5;    // exception
```

<i>index</i>	0	1	2	3	4	5	6
<i>value</i>	0	1	2	3	4	5	6

Practice



arrayElements1

Q: What are the contents of numbers after executing this code?

```
int[] numbers = new int[8];  
numbers[1] = 3;  
numbers[4] = 7;  
numbers[6] = 5;  
  
int x = numbers[1];  
numbers[x] = 2;  
numbers[numbers[4]] = 9;
```

- // 0 1 2 3 4 5 6 7
- A. {0, 3, 0, 2, 7, 0, 5, 9}
- B. {0, 3, 0, 0, 7, 0, 5, 0}
- C. {3, 3, 5, 2, 7, 4, 5, 0}
- D. {0, 3, 0, 2, 7, 6, 4, 4}

Arrays Of Other Types

You can create arrays of any variable type. For example:

```
double[] results = new double[5];
```

```
String[] names = new String[3];
```

```
boolean[] switches = new boolean[4];
```

```
GRect[] rects = new GRect[5];
```

- Java initializes each element of a new array to its *default value*, which is **0** for `int` and `double`, `'\0'` for `char`, **false** for `boolean`, and **null** for objects.

Array Length

Similar to a String, you can get the length of an array by saying

myArray.length

Note that there are *no parentheses* at the end!

Practice:

- What is the index of the *last element* of an array in terms of its length?
- What is the index of the *middle element* of an array in terms of its length?

Arrays + For Loops = ❤️

Just like with Strings, we can use an array's length, along with its indices, to perform cool operations.

Arrays + For Loops = ❤️

Just like with Strings, we can use an array's length, along with its indices, to perform cool operations.

For instance, we can efficiently initialize arrays.

```
int[] numbers = new int[8];  
for (int i = 0; i < numbers.length; i++) {  
    numbers[i] = 2 * i;  
}
```

<i>index</i>	0	1	2	3	4	5	6	7
<i>value</i>	0	2	4	6	8	10	12	14

Arrays + For Loops = ❤️

Just like with Strings, we can use an array's length, along with its indices, to perform cool operations.

For instance, we can read in numbers from the user:

```
int length = readInt("# of numbers? ");
int[] numbers = new int[length];
for (int i = 0; i < numbers.length; i++) {
    numbers[i] = readInt("Elem " + i + ": ");
}
```

Arrays + For Loops = ❤️

Just like with Strings, we can use an array's length, along with its indices, to perform cool operations.

For instance, we can *sum up* all of an array's elements.

```
int sum = 0;
for (int i = 0; i < numbers.length; i++) {
    sum += numbers[i];
}
println(sum);
```

Brief Aside: Creating Arrays

Sometimes, we want to hardcode the elements of an array.

```
int numbers = new int[7];  
numbers[0] = 5;  
numbers[1] = 32;  
numbers[3] = 12;  
...
```

// This is tedious!

Brief Aside: Creating Arrays

Sometimes, we want to hardcode the elements of an array. Luckily, Java has a special syntax for initializing arrays to hardcoded numbers.

```
type[] name = { elements };
```

```
// Java infers the array length
```

```
int[] numbers = {5, 32, 12, 2, 1, -1, 9};
```

Limitations of Arrays

- An array's length is **fixed**. You cannot resize an existing array:

```
int[] a = new int[4];  
a.length = 10;           // error
```

- You cannot compare arrays' contents with == or equals:

```
int[] a1 = {42, -7, 1, 15};  
int[] a2 = {42, -7, 1, 15};  
if (a1 == a2) { ... }           // false!  
if (a1.equals(a2)) { ... }      // false!
```

- An array does not know how to print itself:

```
println(a1);                 // [I@98f8c4]
```

Arrays Methods To The Rescue!

- The class `Arrays` in package `java.util` has useful methods for manipulating arrays:

Method name	Description
<code>Arrays.binarySearch(<i>array</i>, <i>value</i>)</code>	returns the index of the given value in a <i>sorted</i> array (or < 0 if not found)
<code>Arrays.copyOf(<i>array</i>, <i>length</i>)</code>	returns a new copy of array of given length
<code>Arrays.equals(<i>array1</i>, <i>array2</i>)</code>	returns true if the two arrays contain same elements in the same order
<code>Arrays.fill(<i>array</i>, <i>value</i>);</code>	sets every element to the given value
<code>Arrays.sort(<i>array</i>);</code>	arranges the elements into sorted order
<code>Arrays.toString(<i>array</i>)</code>	returns a string representing the array, such as "[10, 30, -25, 17]"

Example: Arrays.toString

`Arrays.toString` accepts an array as a parameter and returns a string representation of its elements.

```
int[] e = {0, 2, 4, 6, 8};  
e[1] = e[3] + e[4];  
println("e is " + Arrays.toString(e));
```

Output:

```
e is [0, 14, 4, 6, 8]
```


Plan for Today

- Data Structures
- Arrays
- **Arrays as Parameters and Return Values**
- Announcements
- Practice: Swapping Elements
- Practice: WeatherStation
- Recap

Passing Arrays Between Methods

- Arrays are just another variable type, so methods can take arrays as parameters and return an array.

```
private int sumArray(int[] numbers) {  
    ...  
}
```

```
private int[] makeSpecialArray(...) {  
    int[] myArray = ...  
    return myArray;  
}
```

Passing Arrays Between Methods

- Arrays are just another variable type, so methods can take arrays as parameters and return an array.
- However, arrays are **objects**, so per A Variable Origin Story, an array variable box actually stores its *location*.
- This means changes to an array passed as a parameter *affect the original array!*

Arrays: Pass By Reference

```
public void run() {  
    int[] numbers = new int[7];  
    fillArray(numbers); // modifies numbers  
    println(Arrays.toString(numbers));  
}
```

```
private void fillArray(int[] arr) {  
    for (int i = 0; i < arr.length; i++) {  
        arr[i] = 2 * i;  
    }  
}
```

Plan for Today

- Data Structures
- Arrays
- Arrays as Parameters and Return Values
- **Announcements**
- Practice: Swapping Elements
- Practice: WeatherStation
- Recap

Announcements

- Assignment 4 due next Monday
- Graded midterms returned next week
- Thursday OH shifted to Monday this week
- OH back to every day starting next week
- Honor Code reminder

Plan for Today

- Data Structures
- Arrays
- Arrays as Parameters and Return Values
- Announcements
- **Practice: Swapping Elements**
- Practice: WeatherStation
- Recap

Practice: Swapping Elements

Let's write a method called **swapElements** that swaps two elements of an array. How can we do this?

What parameters should it take (if any)? What should it return (if anything)?

```
private ??? swapElements(???) {  
    ...  
}
```


Swap: Take 1

```
public void run() {  
    int[] array = new int[5];  
    ...  
    swapElements(array[0], array[1]);  
    ...  
}  
  
private void swapElements(int x, int y) {  
    int temp = x;  
    x = y;  
    y = temp;  
}
```

Swap: Take 1

```
public void run() {
```

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

Ints are primitives, so they are passed by **value**!
Their variable boxes store their *actual values*. So
changes to the parameter *do not affect the
original*.

```
}
```

```
private void swapElements(int x, int y) {
```

```
    int temp = x;
```

```
    x = y;
```

```
    y = temp;
```

```
}
```

Swap: Take 2

```
public void run() {  
    int[] array = new int[5];  
    ...  
    swapElements(array, 0, 1);  
    ...  
}
```

```
private void swapElements(int[] arr, int pos1, int pos2) {  
    int temp = arr[pos1];  
    arr[pos1] = arr[pos2];  
    arr[pos2] = temp;  
}
```

Swap: Take 2

```
public void run() {  
    int[] array = new int[5];  
}
```

Arrays are **objects**, so they are passed by **reference**! Their variable boxes store their *location*. So changes to the parameter *do affect the original*.

```
private void swapElements(int[] arr, int pos1, int pos2) {  
    int temp = arr[pos1];  
    arr[pos1] = arr[pos2];  
    arr[pos2] = temp;  
}
```

Plan for Today

- Data Structures
- Arrays
- Arrays as Parameters and Return Values
- Announcements
- Practice: Swapping Elements
- **Practice: WeatherStation**
- Recap

WeatherStation



- Write a **WeatherStation** program that prompts the user to enter daily temperatures, and uses an array to produce this output:

How many days' temperatures? 7

Day 1's high temp: 45

Day 2's high temp: 44

Day 3's high temp: 39

Day 4's high temp: 48

Day 5's high temp: 37

Day 6's high temp: 46

Day 7's high temp: 53

All temperatures: [45, 44, 39, 48, 37, 46, 53]

Average temp = 44.6

4 days were above average.

Two coldest days: 37, 39

Two hottest days: 53, 48

Plan for Today

- Data Structures
- Arrays
- Arrays as Parameters and Return Values
- Announcements
- Practice: Swapping Elements
- Practice: WeatherStation
- Recap

Recap: Arrays

- An array is an ordered, homogeneous list of data.
- Arrays can store both primitives and objects
- An array's length *cannot be changed* once it is created.
- There are no methods you can call on an array; however, there is the helpful **Arrays** class, with methods such as **Arrays.toString**.

Recap

- Data Structures
- Arrays
- Arrays as Parameters and Return Values
- Announcements
- Practice: Swapping Elements
- Practice: WeatherStation

Next time: 2D Arrays

Extra Slides

Array reverse exercise

- Write a **reverse** method that reverses the elements of an array.
 - Example:


```
int[] numbers = {11, 42, -5, 27, 0, 89};  
reverse(numbers);
```
 - After the call, it should store:

```
[89, 0, 27, -5, 42, 11]
```
 - The code should work for an array of any size.

Algorithm idea

- Swap pairs of elements from the edges; work inwards:

<i>index</i>	0	1	2	3	4	5
<i>value</i>	89	0	27	-5	42	11



Possible algorithm

- **Q:** What is the effect of the code below? Does it reverse the array?

```
int[] numbers = {11, 42, -5, 27, 0, 89};
```

```
// reverse the array
```

```
for (int i = 0; i < numbers.length; i++) {  
    int temp = numbers[i];  
    numbers[i] = numbers[numbers.length - 1 - i];  
    numbers[numbers.length - 1 - i] = temp;  
}
```

- **A.** Code is correct and reverses the array properly.
- **B.** Elements are reversed, but some are lost/missing.
- **C.** Indexes are off-by-1.
- **D.** Array contents are the same at the end; the code does nothing.
- **E.** None of the above

Correct algorithm

- Corrected version:

```
int[] numbers = {11, 42, -5, 27, 0, 89};  
// reverse the array  
for (int i = 0; i < numbers.length / 2; i++) {  
    int temp = numbers[i];  
    numbers[i] = numbers[numbers.length - 1 - i];  
    numbers[numbers.length - 1 - i] = temp;  
}
```

<i>index</i>	0	1	2	3	4	5
<i>value</i>	89	0	27	-5	42	11
	