Building Java Programs

Chapter 7

Lecture 7-1: Arrays

reading: 7.1

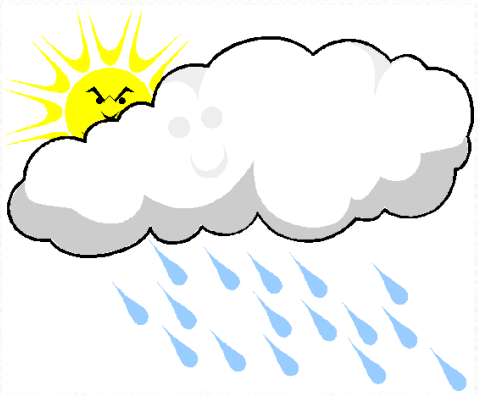
self-checks: #1-9

videos: Ch. 7 #4

Can we solve this problem?

- Consider the following program (input underlined):

```
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
Average temp = 44.6
4 days were above average.
```

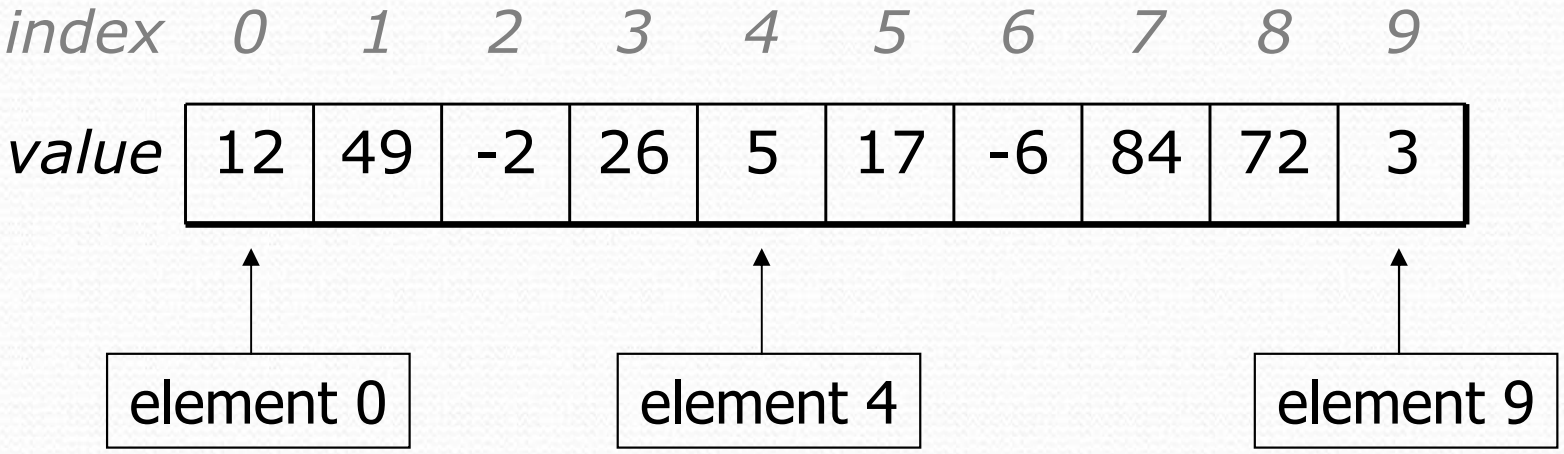


Why the problem is hard

- We need each input value twice:
  - to compute the average (a cumulative sum)
  - to count how many were above average
- We could read each value into a variable... but we:
  - don't know how many days are needed until the program runs
  - don't know how many variables to declare
- We need a way to declare many variables in one step.

Arrays

- **array**: object that stores many values of the same type.
  - **element**: One value in an array.
  - **index**: A 0-based integer to access an element from an array.

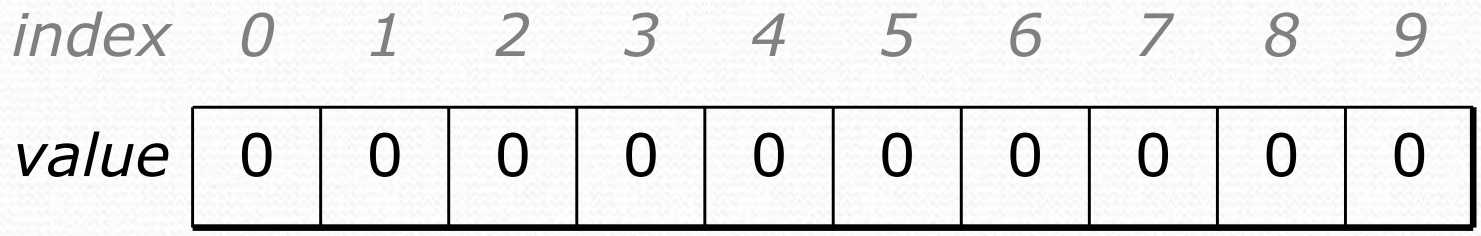


Array declaration

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

- Example:

```
int[] numbers = new int[10];
```



Array declaration, cont.

- The length can be any integer expression.

```
int x = 2 * 3 + 1;
int[] data = new int[x % 5 + 2];
```

- Each element initially gets a "zero-equivalent" value.

Type	Default value
`int`	`0`
`double`	`0.0`
`boolean`	`false`
`String` or other object	`null` (means, "no object")

Accessing elements

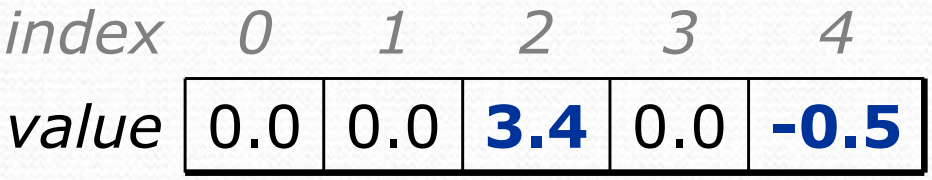
```
name[index]                    // access
name[index] = value;           // modify
```

- Example:

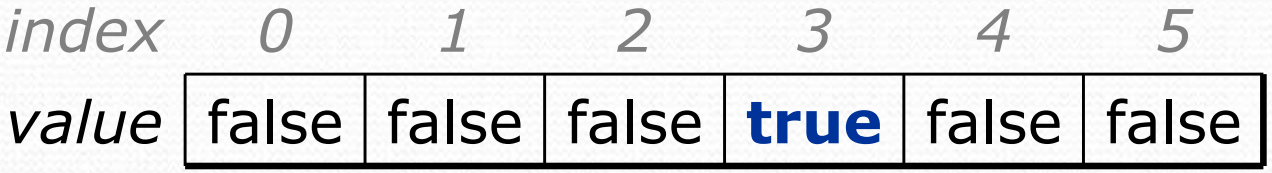
```
numbers[0] = 27;
numbers[3] = -6;

System.out.println(numbers[0]);
if (numbers[3] < 0) {
    System.out.println("Element 3 is negative.");
}
```

Arrays of other types

```
double[] results = new double[5];
results[2] = 3.4;
results[4] = -0.5;
```




```
boolean[] tests = new boolean[6];
tests[3] = true;
```

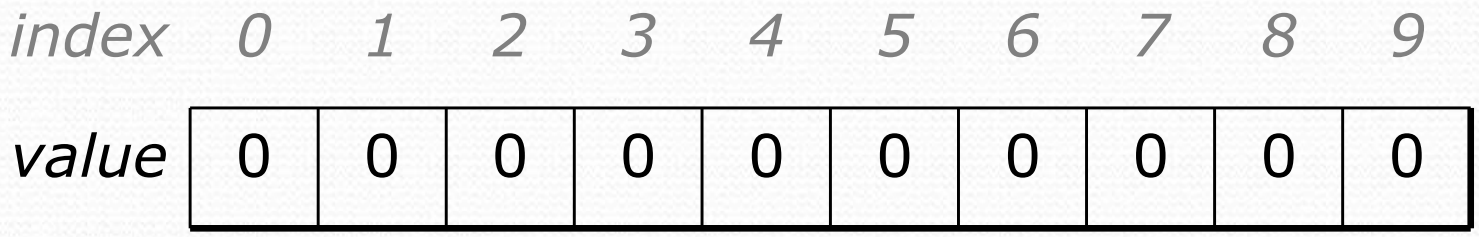


Out-of-bounds

- Legal indexes: between **0** and the **array's length - 1**.
  - Reading or writing any index outside this range will throw an `ArrayIndexOutOfBoundsException`.

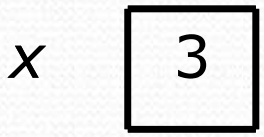
- Example:

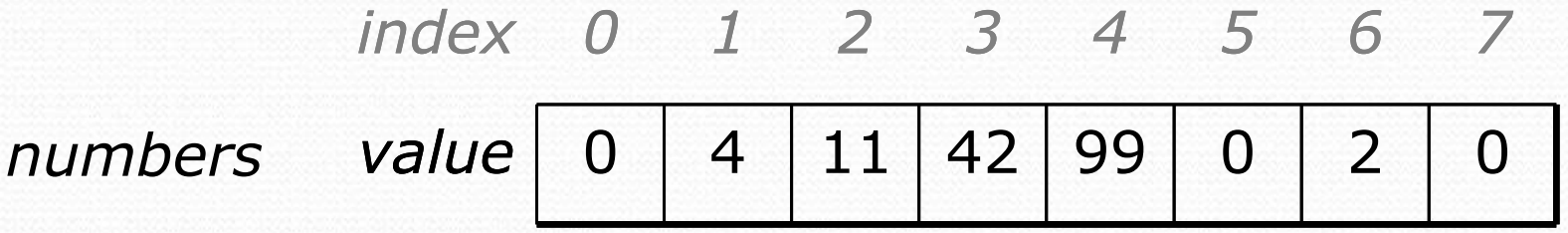
```
int[] data = new int[10];
System.out.println(data[0]);        // okay
System.out.println(data[9]);        // okay
System.out.println(data[-1]);       // exception
System.out.println(data[10]);       // exception
```



Accessing array elements

```
int[] numbers = new int[8];
numbers[1] = 3;
numbers[4] = 99;
numbers[6] = 2;

int x = numbers[1];
numbers[x] = 42;
numbers[numbers[6]] = 11; // use numbers[6] as index
```





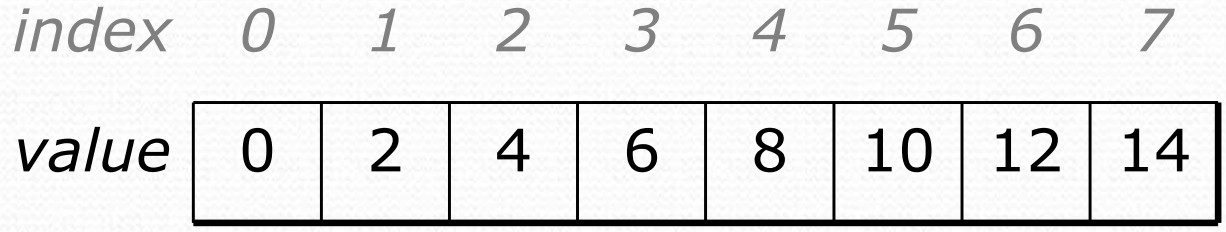
Arrays and for loops

- It is common to use for loops to access array elements.

```
for (int i = 0; i < 8; i++) {
    System.out.print(numbers[i] + " ");
}
System.out.println();  // output: 0 4 11 0 44 0 0 2
```

- Sometimes we assign each element a value in a loop.

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



The `length` field

- An array's `length` field stores its number of elements.

```
name.length
```

```
for (int i = 0; i < numbers.length; i++) {
    System.out.print(numbers[i] + " ");
}
// output: 0 2 4 6 8 10 12 14
```

  - It does not use parentheses like a String's `.length()`.

- What expressions refer to:
  - The last element of any array?
  - The middle element?

Weather question

- Use an array to solve the weather problem:

```
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
Average temp = 44.6
4 days were above average.
```

Weather answer

```
// Reads temperatures from the user, computes average and # days above average.
import java.util.*;

public class Weather {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("How many days' temperatures? ");
        int days = console.nextInt();

        int[] temperatures = new int[days];  // array to store days' temperatures
        int sum = 0;

        for (int i = 0; i < days; i++) {     // read/store each day's temperature
            System.out.print("Day " + (i + 1) + "'s high temp: ");
            temperatures[i] = console.nextInt();
            sum += temperatures[i];
        }
        double average = (double) sum / days;

        int count = 0;                       // see if each day is above average
        for (int i = 0; i < days; i++) {
            if (temperatures[i] > average) {
                count++;
            }
        }

        // report results
        System.out.printf("Average temp = %.1f\n", average);
        System.out.println(count + " days above average");
    }
}
```

Arrays for counting and tallying

reading: 7.1

self-checks: #8

A multi-counter problem

- Problem: Examine a large integer and count the number of occurrences of every digit from 0 through 9.
  - Example: The number 229231007 contains:

    two 0s, one 1, three 2s, one 7, and one 9.

- We could declare 10 counter variables for this...

```
int counter0, counter1, counter2, counter3, counter4,
    counter5, counter6, counter7, counter8, counter9;
```

  - Yuck!

A multi-counter problem

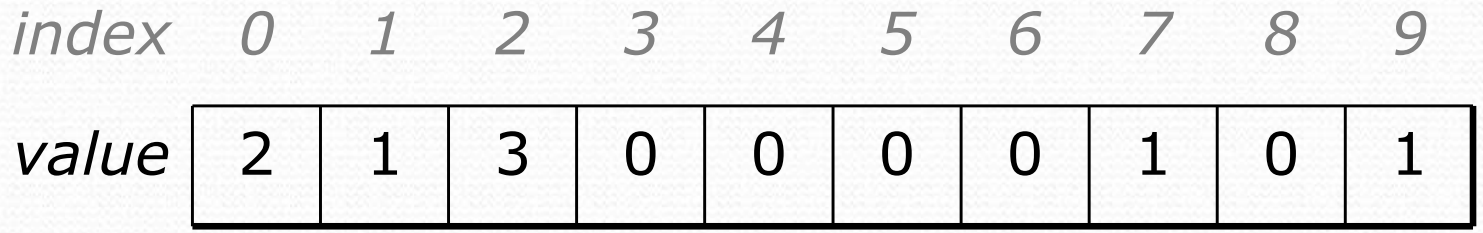
- A better solution is to use an array of size 10.
  - The element at index *i* will store the counter for digit value *i*.
  - for integer value 229231007, our array should store:

index	0	1	2	3	4	5	6	7	8	9
value	2	1	3	0	0	0	0	1	0	1

- The index at which a value is stored has meaning.
  - Sometimes it doesn't matter.
  - What about the weather case?

Creating an array of tallies

```
int num = 229231007;
int[] counts = new int[10];
while (num > 0) {
    // pluck off a digit and add to proper counter
    int digit = num % 10;
    counts[digit]++;
    num = num / 10;
}
```



Array histogram question

- Given a file of integer exam scores, such as:

```
82
66
79
63
83
```

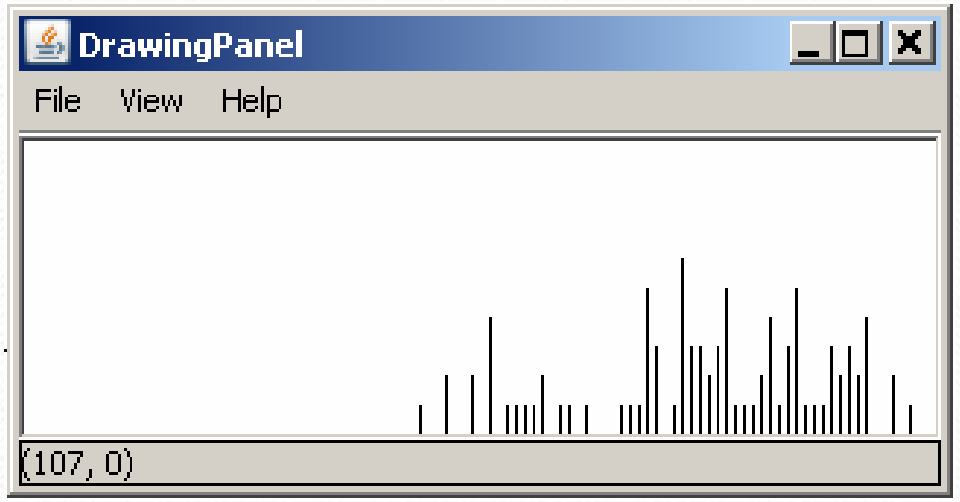
Write a program that will print a histogram of stars indicating the number of students who earned each unique exam score.

```
85: *****
86: ***********
87: ***
88: *
91: ****
```

Histogram variations

- Curve the scores; add a fixed number to each score.
  (But don't allow a curved score to exceed the max of 101.)

- Chart the data with a `DrawingPanel`.
  - window is 100px tall
  - 2px between each bar
  - 10px tall bar for each student who earned that score



Array histogram answer

```
// Reads an input file of test scores (integers) and displays a
// graphical histogram of the score distribution.
import java.awt.*;
import java.io.*;
import java.util.*;

public class Histogram {
    public static final int CURVE = 5;   // adjustment to each exam score

    public static void main(String[] args) throws FileNotFoundException {
        Scanner input = new Scanner(new File("midterm.txt"));
        int[] counts = new int[101];     // counters of test scores 0 - 100

        while (input.hasNextInt()) {     // read file into counts array
            int score = input.nextInt();
            score = Math.min(score + CURVE, 100);    // curve the exam score
            counts[score]++;             // if score is 87, then counts[87]++
        }

        for (int i = 0; i < counts.length; i++) {    // print star histogram
            if (counts[i] > 0) {
                System.out.print(i + ": ");
                for (int j = 0; j < counts[i]; j++) {
                    System.out.print("*");
                }
                System.out.println();
            }
        }
```

Array histogram solution 2

```
        ...

        // use a DrawingPanel to draw the histogram
        DrawingPanel p = new DrawingPanel(counts.length * 3 + 6, 200);
        Graphics g = p.getGraphics();
        g.setColor(Color.BLACK);
        for (int i = 0; i < counts.length; i++) {
            g.drawLine(i * 3 + 3, 175, i * 3 + 3, 175 - 5 * counts[i]);
        }
    }
}
```

Array traversals, text processing

reading: 7.1, 4.4

self-check: Ch. 7 #8, Ch. 4 #19-23

Array traversals

- **traversal**: An examination of each element of an array.

```
for (int i = 0; i < array.length; i++) {
    do something with array[i];
}
```

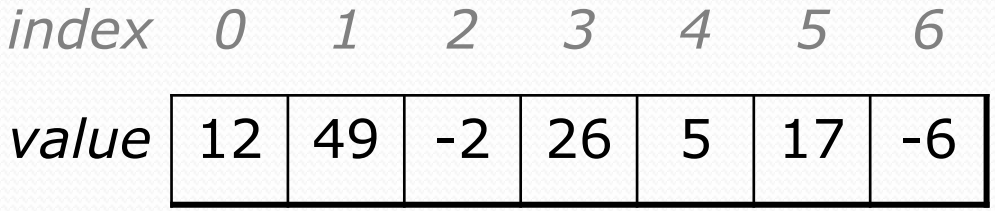
- Examples:
  - printing the elements
  - searching for a specific value
  - rearranging the elements
  - computing the sum, product, etc.

Quick array initialization

type`[]` **name** `=` `{`**value**`,` **value**`,` … **value**`};`

- Example:

```
int[] numbers = {12, 49, -2, 26, 5, 17, -6};
```



- Useful when you know what the array's elements will be
- The compiler figures out the size by counting the values

"Array mystery" problem

- What element values are stored in the following array?

```
int[] a = {1, 7, 5, 6, 4, 14, 11};
for (int i = 0; i < a.length - 1; i++) {
    if (a[i] > a[i + 1]) {
        a[i + 1] = a[i + 1] * 2;
    }
}
```

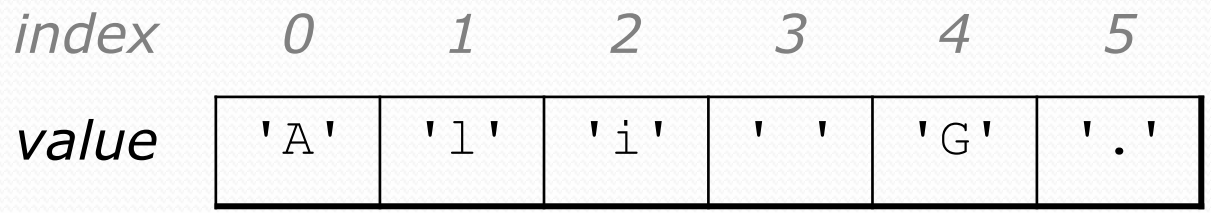
index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	1	7	10	12	8	14	22

Text processing

- **text processing**: Examining, editing, formatting text.
  - Often involves `for` loops to examine each letter of a `String`.
    - Count the number of times the letter 's' occurs in a file.
    - Find which letter is most common in a file.
    - Count A, C, T and Gs in `String`s representing DNA strands.

- `String`s are represented internally as arrays of `char`.

```
String str = "Ali G.";
```



Recall: type `char`

- **`char`**: A primitive type representing a single character.
  - Values are surrounded with apostrophes: `'a'` or `'4'` or `'\n'`

- Access a string's characters with its `charAt` method.

```
String word = console.next();
char firstLetter = word.charAt(0);
if (firstLetter == 'c') {
    System.out.println("That's good enough for me!");
}
```

- Use `for` loops to examine each character.

```
String coolMajor = "CSE";
for (int i = 0; i < coolMajor.length(); i++) {
    System.out.println(coolMajor.charAt(i));
}
```

Text processing question

- Write a method `tallyVotes` that accepts a `String` parameter and prints the number of McCain, Obama and independent voters.

```
// (M)cCain, (O)bama, (I)ndependent
String voteText = "MOOOOOOMMMMMOOOOOOMOMMIMOMMIMOMMIO";
tallyVotes(voteText);
```

  - Output:

```
Votes: [16, 14, 3]
```

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];
System.out.println("e is " + Arrays.toString(e));
```

Output:

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

- Must `import java.util.*;`

The `Arrays` class

- Class `Arrays` in package `java.util` has useful static methods for manipulating arrays:

Method name	Description
`binarySearch(`**array**`, `**value**`)`	returns the index of the given value in a sorted array (< 0 if not found)
`equals(`**array1**`, `**array2**`)`	returns `true` if the two arrays contain the same elements in the same order
`fill(`**array**`, `**value**`)`	sets every element in the array to have the given value
`sort(`**array**`)`	arranges the elements in the array into ascending order
`toString(`**array**`)`	returns a string representing the array, such as `"[10, 30, 17]"`

Text processing answer

```
public static int[] tallyVotes(String votes) {
    int[] tallies = new int[3];   // M -> 0, O -> 1, I -> 2

    for(int i = 0; i < votes.length(); i++) {
        if(votes.charAt(i) == 'M') {
            tallies[0]++;
        } else if(votes.charAt(i) == 'O') {
            tallies[1]++;
        } else {                        // votes.charAt(i) == 'I'
            tallies[2]++;
        }
    }

    System.out.println("Votes: " + Arrays.toString(tally));;
}
```

Arrays as parameters and returns; values vs. references

reading: 7.1, 3.3, 4.3

self-checks: Ch. 7 #5, 8, 9

exercises: Ch. 7 #1-10

Swapping values

```
public static void main(String[] args) {
    int a = 7;
    int b = 35;

    // swap a with b (incorrectly)
    a = b;
    b = a;

    System.out.println(a + " " + b);
}
```

- What is wrong with this code? What is its output?

- The red code should be replaced with:

```
int temp = a;
a = b;
b = temp;
```

A `swap` method?

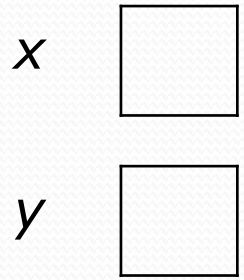
- Does the following `swap` method work? Why or why not?

```
public static void main(String[] args) {
    int a = 7;
    int b = 35;

    // swap a with b
    swap(a, b);

    System.out.println(a + " " + b);
}

public static void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
```

Value semantics (primitives)

- **value semantics**: Behavior where values are copied when assigned to each other or passed as parameters.
  - When one primitive variable is assigned to another, its value is copied.
  - Modifying the value of one variable does not affect others.

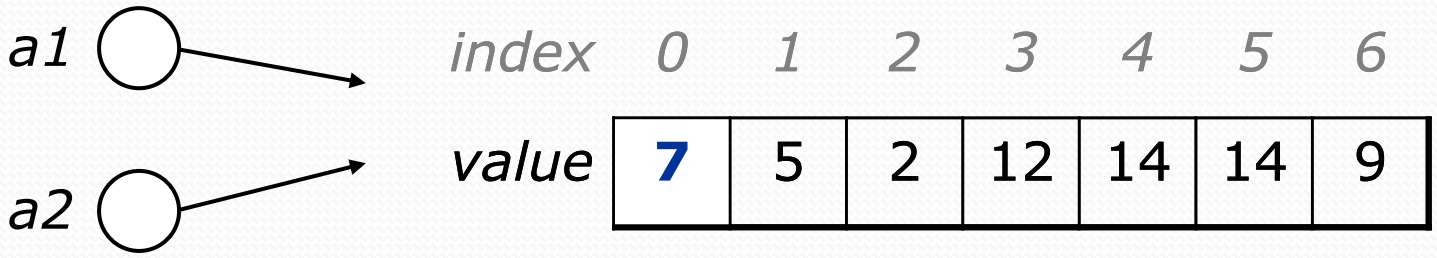
```
int x = 5;
int y = x;      // x = 5, y = 5
y = 17;         // x = 5, y = 17
x = 8;          // x = 8, y = 17
```



Reference semantics (objects)

- **reference semantics**: Behavior where variables actually store the address of an object in memory.
  - When one reference variable is assigned to another, the object is *not* copied; both variables refer to the *same object*.
  - Modifying the value of one variable *will* affect others.

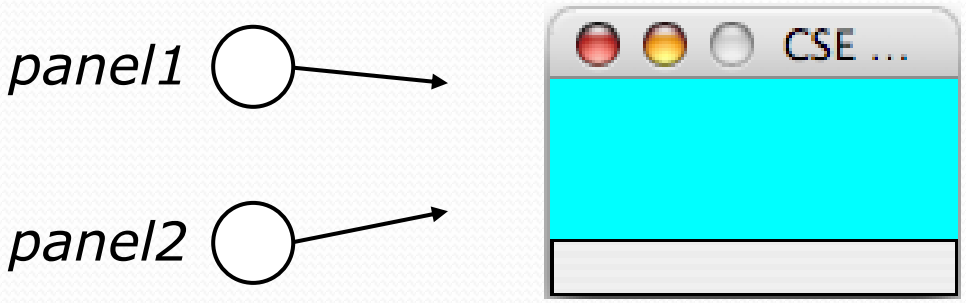
```
int[] a1 = {4, 5, 2, 12, 14, 14, 9};
int[] a2 = a1;      // refer to same array as a1
a2[0] = 7;
System.out.println(a1[0]);   // 7
```



References and objects

- Arrays and objects use reference semantics. Why?
  - *efficiency.* Copying large objects slows down a program.
  - *sharing.* It's useful to share an object's data among methods.

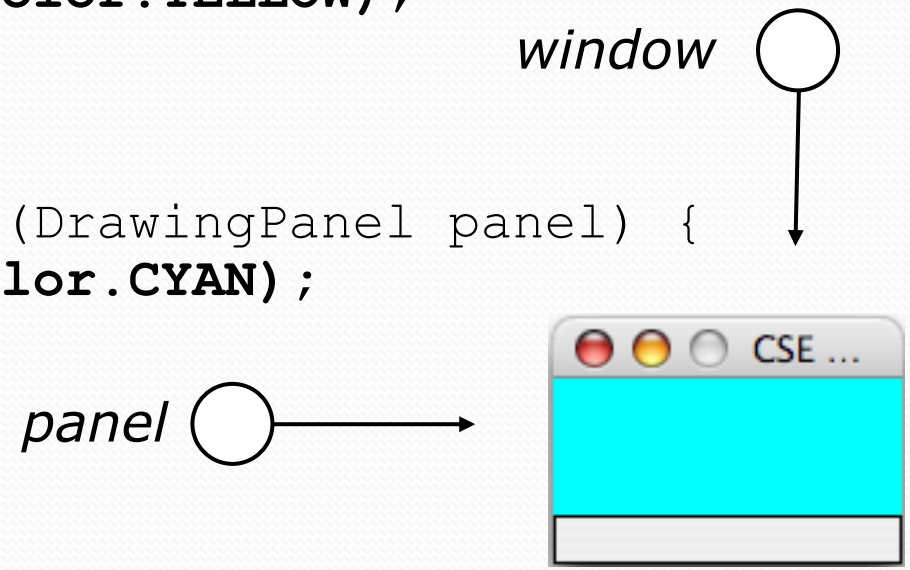
```
DrawingPanel panel1 = new DrawingPanel(80, 50);
DrawingPanel panel2 = panel1;   // same window
panel2.setBackground(Color.CYAN);
```



Objects as parameters

- When an object is passed as a parameter, the object is *not* copied. The parameter refers to the same object.
  - If the parameter is modified, it *will* affect the original object.

```
public static void main(String[] args) {
    DrawingPanel window = new DrawingPanel(80, 50);
    window.setBackground(Color.YELLOW);
    example(window);
}

public static void example(DrawingPanel panel) {
    panel.setBackground(Color.CYAN);
}
```



Arrays as parameters

- Declaration:

```
public static type methodName(type[] name) {
```

  - Example:

```
public static double average(int[] numbers) {
```

- Call:

```
methodName(arrayName);
```

  - Example:

```
int[] scores = {13, 17, 12, 15, 11};
double avg = average(scores);
```

Building Java Programs

Chapter 7

Lecture 7-3: Arrays as Parameters; File Output

reading: 7.1, 4.3, 3.3

self-checks: Ch. 7 #19-23

exercises: Ch. 7 #5

Section attendance question

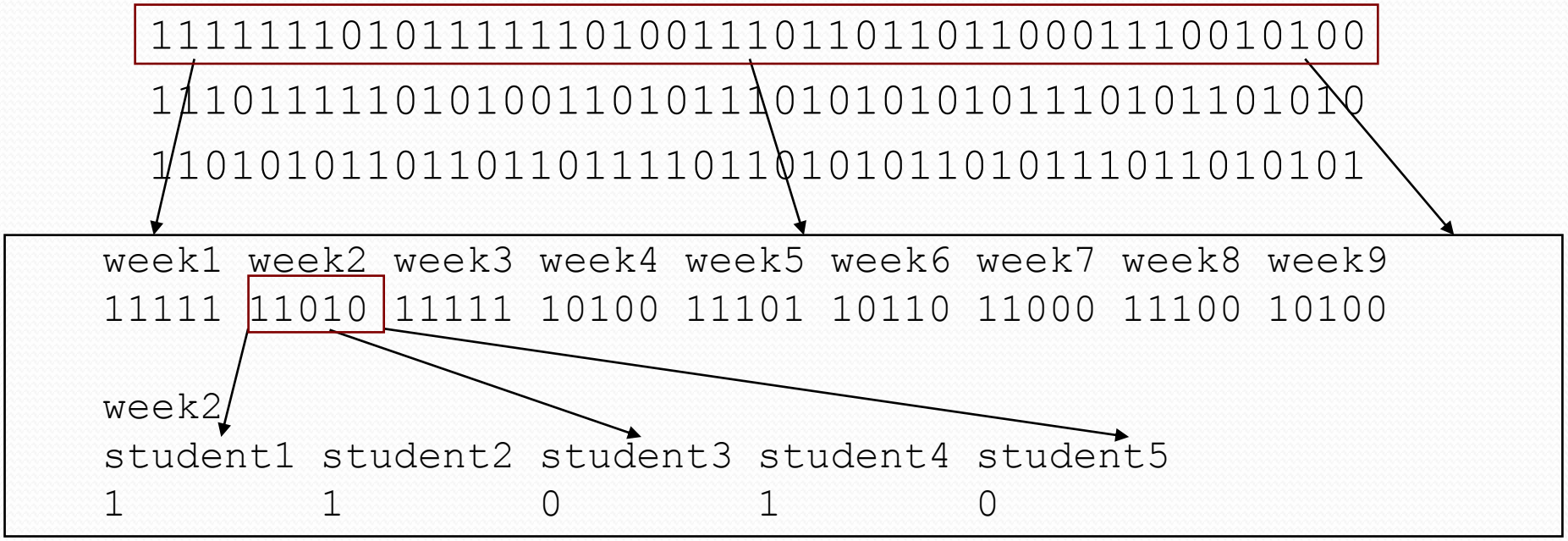
- Write a program that reads a data file of section attendance and produces the following output:

```
Sections attended: [9, 6, 7, 4, 3]
Student scores: [20, 18, 20, 12, 9]
Student grades: [100.0, 90.0, 100.0, 60.0, 45.0]

Sections attended: [6, 7, 5, 6, 4]
Student scores: [18, 20, 15, 18, 12]
Student grades: [90.0, 100.0, 75.0, 90.0, 60.0]

Sections attended: [5, 6, 5, 7, 6]
Student scores: [15, 18, 15, 20, 18]
Student grades: [75.0, 90.0, 75.0, 100.0, 90.0]
```

  - Students earn 3 points for each section attended up to 20.

Section input file

- The input file contains section attendance data:



- Each line represents a section (5 students, 9 weeks).
  - 1 means the student attended; 0 not.

Data transformations

- In this problem we go from 0s and 1s to student grades
  - This is called *transforming* the data.
  - Often each transformation is stored in its own array.

- We must map between the data and array indexes.

  Examples:
  - by position (store the *i* th value we read at index *i*)
  - tally (if input value is *i*, store it at array index *i*)
  - explicit mapping (count `'M'` at index 0, count `'O'` at index 1)

Section attendance answer

```
// This program reads a file representing which students attended which
// discussion sections and produces output of their attendance and scores.

import java.io.*;
import java.util.*;

public class Sections {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner input = new Scanner(new File("sections.txt"));
        while (input.hasNextLine()) {
            String line = input.nextLine();      // process one section
            int[] attended = new int[5];
            for (int i = 0; i < line.length(); i++) {
                if (line.charAt(i) == '1') {     // c == '1'  or  c == '0'
                    attended[i % 5]++;           // student attended section
                }
            }
            int[] points = new int[5];
            for (int i = 0; i < attended.length; i++) {
                points[i] = Math.min(20, 3 * attended[i]);
            }
            double[] grades = new double[5];
            for (int i = 0; i < points.length; i++) {
                grades[i] = 100.0 * points[i] / 20.0;
            }
            System.out.println("Sections attended: " + Arrays.toString(attended));
            System.out.println("Sections scores: " + Arrays.toString(points));
            System.out.println("Sections grades: " + Arrays.toString(grades));
            System.out.println();
        }
    }
}
```

Array parameter example

```
public static void main(String[] args) {
    int[] iq = {126, 84, 149, 167, 95};
    double avg = average(iq);
    System.out.println("Average = " + avg);
}

public static double average(int[] array) {
    int sum = 0;
    for (int i = 0; i < array.length; i++) {
        sum += array[i];
    }
    return (double) sum / array.length;
}
```

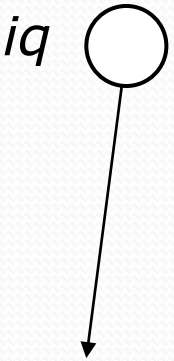
Output:

```
Average = 124.2
```

Arrays passed by reference

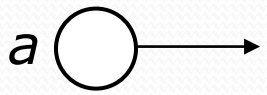
- Arrays are objects.
  - When passed as parameters, they are passed by *reference.* (Changes made in the method are also seen by the caller.)
- Example:

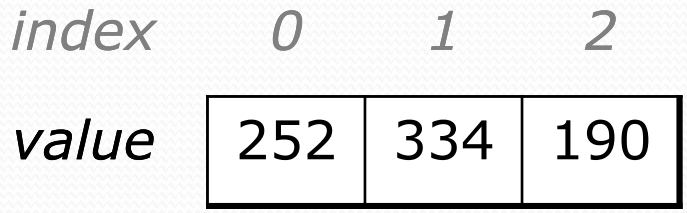
```
public static void main(String[] args) {
    int[] iq = {126, 167, 95};
    doubleAll(iq);
    System.out.println(Arrays.toString(iq));
}

public static void doubleAll(int[] a) {
    for (int i = 0; i < a.length; i++) {
        a[i] = a[i] * 2;
    }
}
```



- Output:

```
[252, 334, 190]
```





Arrays as return (declaring)

```
public static type[] methodName(parameters) {
```

- Example:

```
public static int[] countDigits(int n) {
    int[] counts = new int[10];
    while (n > 0) {
        int digit = n % 10;
        n = n / 10;
        counts[digit]++;
    }
    return counts;
}
```

Arrays as return (calling)

```
type[] name = methodName(parameters);
```

- Example:

```
public static void main(String[] args) {
    int[] tally = countDigits(229231007);
    System.out.println(Arrays.toString(tally));
}
```

Output:

```
[2, 1, 3, 1, 0, 0, 0, 1, 0, 1]
```

Array param/return question

- Modify our previous Sections program to use static methods that use arrays as parameters and returns.

```
Sections attended: [9, 6, 7, 4, 3]
Student scores: [20, 18, 20, 12, 9]
Student grades: [100.0, 90.0, 100.0, 60.0, 45.0]

Sections attended: [6, 7, 5, 6, 4]
Student scores: [18, 20, 15, 18, 12]
Student grades: [90.0, 100.0, 75.0, 90.0, 60.0]

Sections attended: [5, 6, 5, 7, 6]
Student scores: [15, 18, 15, 20, 18]
Student grades: [75.0, 90.0, 75.0, 100.0, 90.0]
```

Array param/return answer

```
// This program reads a file representing which students attended
// which discussion sections and produces output of the students'
// section attendance and scores.

import java.io.*;
import java.util.*;

public class Sections {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner input = new Scanner(new File("sections.txt"));
        while (input.hasNextLine()) {
            // process one section
            String line = input.nextLine();
            int[] attended = countAttended(line);
            int[] points = computePoints(attended);
            double[] grades = computeGrades(points);
            results(attended, points, grades);
        }
    }

    // Produces all output about a particular section.
    public static void results(int[] attended, int[] points, double[] grades) {
        System.out.println("Sections attended: " + Arrays.toString(attended));
        System.out.println("Sections scores: " + Arrays.toString(points));
        System.out.println("Sections grades: " + Arrays.toString(grades));
        System.out.println();
    }

    ...
```

Array param/return answer

```
...
// Counts the sections attended by each student for a particular section.
public static int[] countAttended(String line) {
    int[] attended = new int[5];
    for (int i = 0; i < line.length(); i++) {
        char c = line.charAt(i);
        // c == '1'  or  c == '0'
        if (c == '1') {
            // student attended their section
            attended[i % 5]++;
        }
    }
    return attended;
}

// Computes the points earned for each student for a particular section.
public static int[] computePoints(int[] attended) {
    int[] points = new int[5];
    for (int i = 0; i < attended.length; i++) {
        points[i] = Math.min(20, 3 * attended[i]);
    }
    return points;
}

// Computes the percentage for each student for a particular section.
public static double[] computeGrades(int[] points) {
    double[] grades = new double[5];
    for (int i = 0; i < points.length; i++) {
        grades[i] = 100.0 * points[i] / 20.0;
    }
    return grades;
}
}
```

File output

reading: 6.4 - 6.5

Output to files

- **`PrintStream`**: An object in the `java.io` package that lets you print output to a destination such as a file.
  - Any methods you have used on `System.out` (such as `print`, `println`) will work on a `PrintStream`.

- Syntax:

```
PrintStream name = new PrintStream(new File("file name"));
```

Example:

```
PrintStream output = new PrintStream(new File("out.txt"));
output.println("Hello, file!");
output.println("This is a second line of output.");
```

Details about `PrintStream`

```
PrintStream name = new PrintStream(new File("file name"));
```

- If the given file does not exist, it is created.
- If the given file already exists, it is overwritten.

- The output you print appears in a file, not on the console. You will have to open the file with an editor to see it.

- Do not open the same file for both reading (`Scanner`) and writing (`PrintStream`) at the same time.
  - You will overwrite your input file with an empty file (0 bytes).

System.out and PrintStream

- The console output object, `System.out`, is a `PrintStream`.

```
PrintStream out1 = System.out;
PrintStream out2 = new PrintStream(new File("data.txt"));
out1.println("Hello, console!");   // goes to console
out2.println("Hello, file!");      // goes to file
```

  - A reference to it can be stored in a `PrintStream` variable.
    - Printing to that variable causes console output to appear.
  - You can pass `System.out` as a parameter to a method expecting a `PrintStream`.
    - Allows methods that can send output to the console or a file.

`PrintStream` question

- Modify our previous Sections program to use a `PrintStream` to output to the file `sections_out.txt`.

```
Section #1:
Sections attended: [9, 6, 7, 4, 3]
Student scores: [20, 18, 20, 12, 9]
Student grades: [100.0, 90.0, 100.0, 60.0, 45.0]

Section #2:
Sections attended: [6, 7, 5, 6, 4]
Student scores: [18, 20, 15, 18, 12]
Student grades: [90.0, 100.0, 75.0, 90.0, 60.0]

Section #3:
Sections attended: [5, 6, 5, 7, 6]
Student scores: [15, 18, 15, 20, 18]
Student grades: [75.0, 90.0, 75.0, 100.0, 90.0]
```

PrintStream answer

```
// Section attendance program
// This version uses a PrintStream for output.

import java.io.*;
import java.util.*;

public class Sections {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner input = new Scanner(new File("sections.txt"));
        PrintStream out = new PrintStream(new File("sections_out.txt"));
        while (input.hasNextLine()) {   // process one section
            String line = input.nextLine();
            int[] attended = countAttended(line);
            int[] points = computePoints(attended);
            double[] grades = computeGrades(points);
            results(attended, points, grades, out);
        }
    }

    // Produces all output about a particular section.
    public static void results(int[] attended, int[] points,
            double[] grades, PrintStream out) {
        out.println("Sections attended: " + Arrays.toString(attended));
        out.println("Sections scores: " + Arrays.toString(points));
        out.println("Sections grades: " + Arrays.toString(grades));
        out.println();
    }
    ...
```

Prompting for a file name

- We can ask the user to tell us the file to read.
  - The file name might have spaces; use `nextLine()`, not `next()`

```
// prompt for input file name
Scanner console = new Scanner(System.in);
System.out.print("Type a file name to use: ");
String filename = console.nextLine();
Scanner input = new Scanner(new File(filename));
```

- What if the user types a file name that does not exist?

Fixing file-not-found issues

- `File` objects have an `exists` method we can use:

```
Scanner console = new Scanner(System.in);
System.out.print("Type a file name to use: ");
String filename = console.nextLine();
File file = new File(filename);

if (!file.exists()) {
    // try a second time
    System.out.print("Try again: ");
    String filename = console.nextLine();
    file = new File(filename);
}
Scanner input = new Scanner(file);  // open the file
```

Output:

```
Type a file name to use: hourz.text
Try again: hours.txt
```