Building Java Programs

Chapter 7: Arrays

Lecture outline

Lecture 18

- **Array basics**
  - **declaring and initializing an array**
  - **getting and setting values of elements of an array**
  - **limitations of arrays**
  - **Arrays for counting and tallying**

Lecture 19

- Array traversal algorithms
  - printing an array's elements
  - searching an array
  - reversing an array

Lecture 20

- Advanced array usage
  - Shifting elements in an array
  - arrays as parameters to methods
  - String and Graphics methods that use arrays
  - the Arrays class
  - command-line arguments

Array basics

- suggested reading: 7.1

A problem we can't solve (yet)

- 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 = 44.57142857142857
4 days were above average.
```

- We need the temperatures to compute the average, and again to tell how many were above average.

Why the problem is tough

- We appear to need each input value twice:
  - once to compute the average
  - a second time to count how many were above average

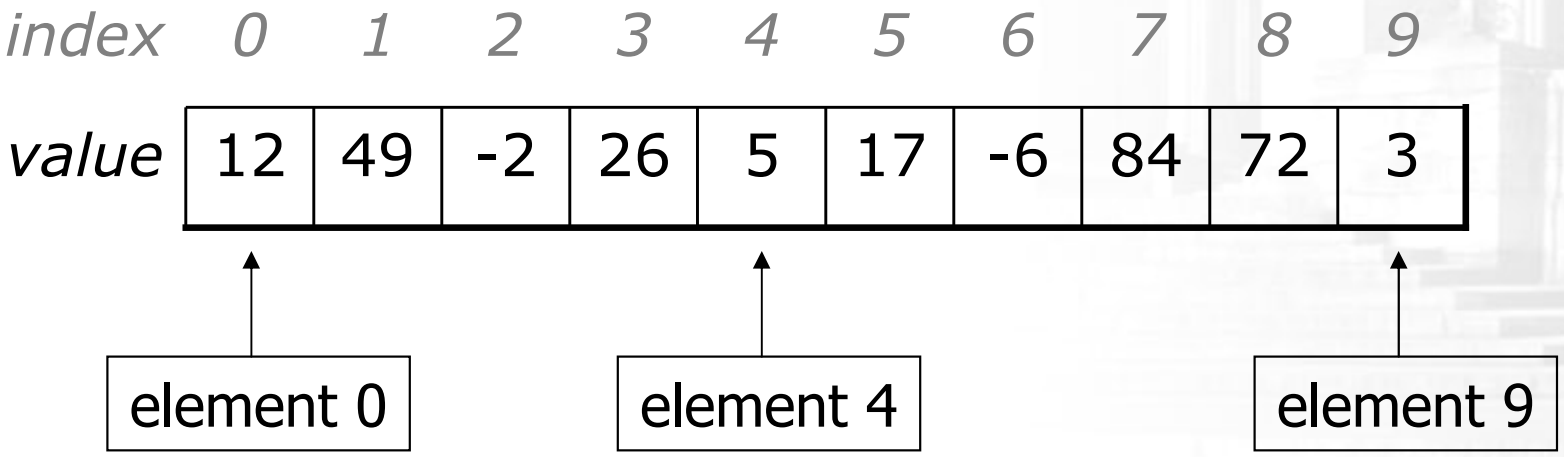
- We could examine each value twice if we could read them into variables.
  - However, we don't know how many variables to declare. We don't know how many days' weather will be typed until the program is running.

- We need a way to declare many variables in one step.

Arrays

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

- We usually draw an array as a row or column of boxes.
  - Example: an array of ten integers



Array declaration

- Declaring/initializing an array:

  <type> `[]` ***<name>*** `= new` ***<type>*** `[` ***<length>*** `];`

  - The length of the array is specified between [ ] brackets.
  - Example:

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

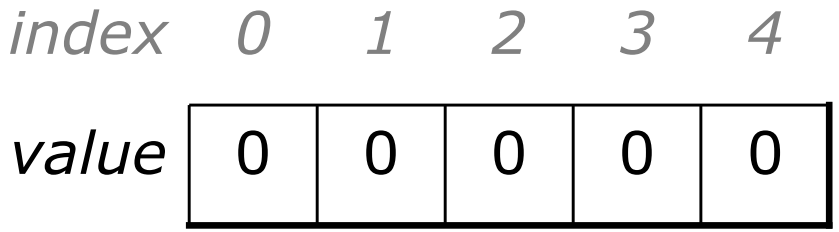
index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
value	0	0	0	0	0	0	0	0	0	0

- The array's length can be any expression.
  - Example:

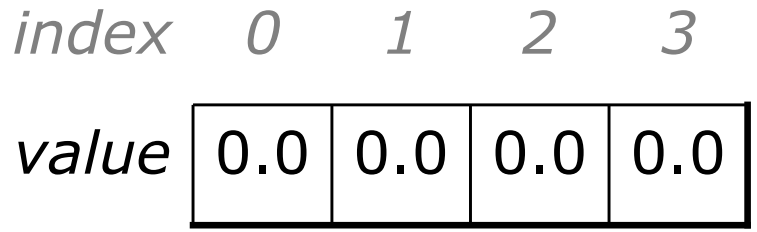
```
int x = 2 * 3 + 1;
int[] data = new int[x % 5 + 2];
```

Array auto-initialization

- When arrays are initially constructed, every element is automatically initialized to a "zero-equivalent" value.
  - `int`: `0`
  - `double`: `0.0`
  - `boolean`: `false`
  - `char`: `'\0'` (the "null character")
  - `String` or object: `null` (`null` means "no object")



An array of integers

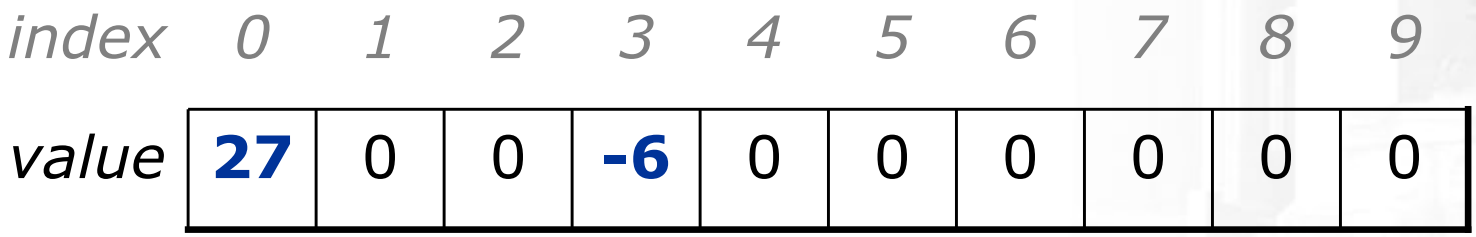


An array of real numbers

Accessing array elements

- Assigning a value to an array element:
  <array name> `[` ***<index>*** `] = ` ***<value>*** `;`
  - Example:

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

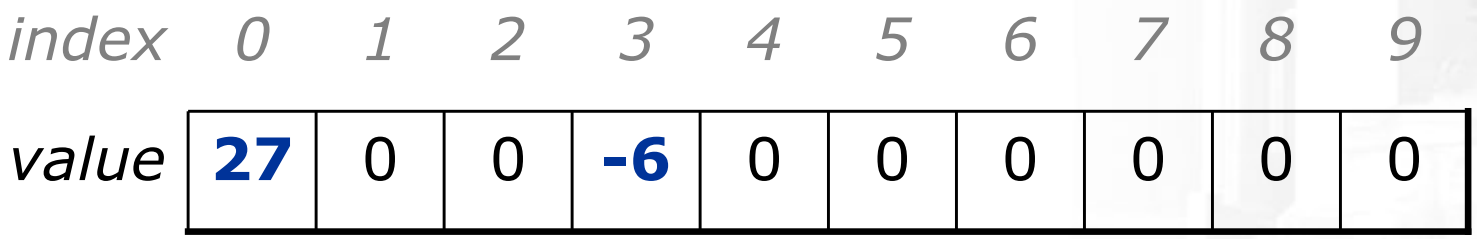


Accessing array elements

- Using an array element's value in an expression:
  <array name> [***<index>***]

  - Example:

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

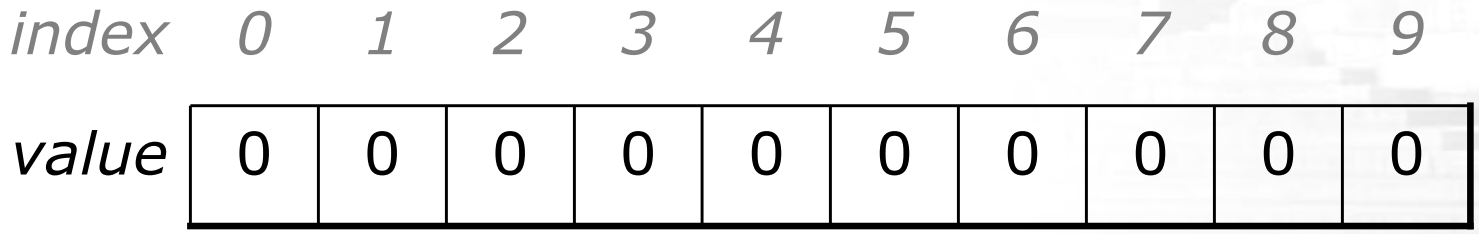


Out-of-bounds indexes

- The indexes that are legal to access in an array are those in the range of **0** to 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[-1]);       // exception
System.out.println(data[9]);        // okay
System.out.println(data[10]);       // exception
```



Arrays of other types

- Arrays can contain other types, such as `double`.
  - Example:

```
double[] results = new double[6];
results[2] = 3.4;
results[5] = -0.5;
```

index	*0*	*1*	*2*	*3*	*4*	*5*
value	0.0	0.0	**3.4**	0.0	0.0	**-0.5**

  - Example:

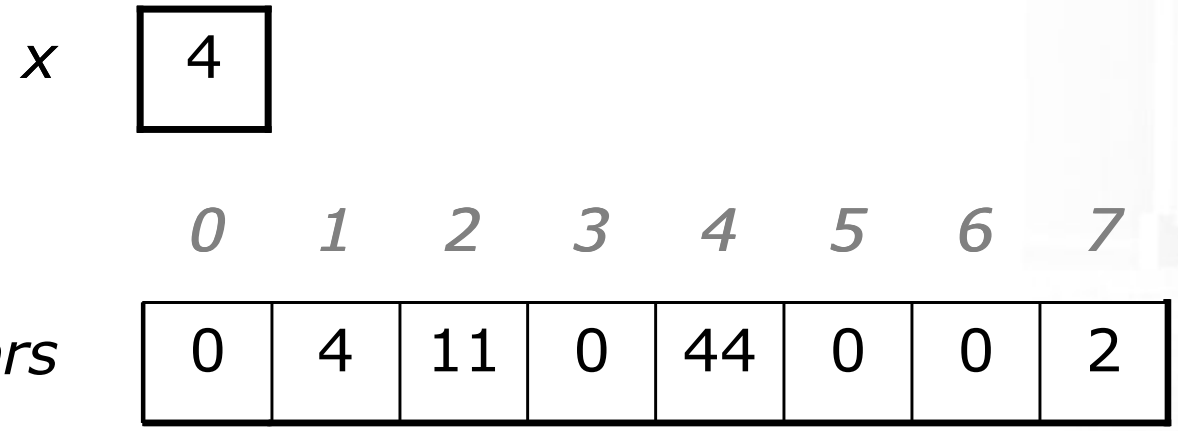
```
boolean[] tests = new boolean[6];
tests[3] = true;
```

index	*0*	*1*	*2*	*3*	*4*	*5*
value	false	false	false	**true**	false	false

Accessing array elements

- A longer example of accessing and changing elements:

```
int[] numbers = new int[8];
numbers[1] = 4;
numbers[4] = 99;
numbers[7] = 2;

int x = numbers[1];
numbers[x] = 44;
numbers[numbers[7]] = 11;  // use numbers[7] as index
```



Arrays and for loops

- Arrays are very commonly used with `for` loops that pass over each element and process it in some way:
  - Example (print each element of an array):

```
for (int i = 0; i < 8; i++) {
    System.out.print(numbers[i] + " ");
}
System.out.println();   // end the line of output
```

  - Output (when used on array from previous slide):

```
0 4 11 0 44 0 0 2
```

More arrays and for loops

- Sometimes we assign each array element a value in a for loop.
  - Example:

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

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

- What values would be stored into the array after this code?

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

value	0	1	4	9	16	25	36	49

The .length field

- An array's `length` field stores its number of elements.
  - General syntax:
    <array name> `.length`
  - It does not use parentheses like a String's `.length()`.

- Example (using array from previous slide):

```
for (int i = 0; i < numbers.length; i++) {
    System.out.print(numbers[i] + " ");
}
```

  - Output:

```
0 1 4 9 16 25 36 49
```

- What expression refers to the last element of the array? The middle element?

Weather problem

- Use an array to solve the following 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 = 44.57142857142857
4 days were above average.
```

Weather solution

```
// This program reads several days' temperatures from the user
// and computes the average and how many days were 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.println("Average temp = " + average);
        System.out.println(count + " days above average");
    }
}
```

Arrays for counting and tallying

- suggested reading: 7.1

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 need to examine each digit of the large integer and count how many times we've seen that digit.
  - This will require counters for each of the values 0--9.

- We could declare 10 counter variables for this...
  - A better solution is to use an array of size 10.
  - The element at index *i* will store the counter for digit value *i*.

Creating an array of tallies

- The following code builds an array of digit counters:

```
int num = 229231007;
int[] counts = new int[10];
while (num > 0) {
    int digit = num % 10;
    counts[digit]++;
    num = num / 10;
}
```

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

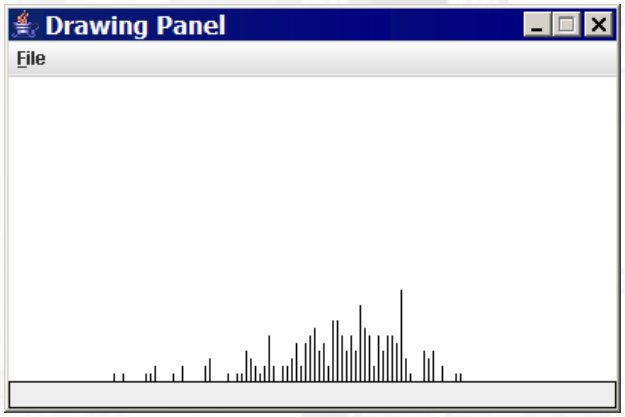
Array histogram question

- 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:  ****
```



- Variations:
  - Make a curve that adds a fixed number of points to each score. (But don't allow a curved score to exceed the max of 100.)
  - Chart the data with a `DrawingPanel`.

Array histogram solution

```
// 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]);
        }
    }
}
```

Lecture outline

Lecture 18

- Array basics
  - declaring and initializing an array
  - getting and setting values of elements of an array
  - limitations of arrays
  - Arrays for counting and tallying

Lecture 19

- **Array traversal algorithms**
  - **printing an array's elements**
  - **searching an array**
  - **reversing an array**

Lecture 20

- Advanced array usage
  - Shifting elements in an array
  - arrays as parameters to methods
  - String and Graphics methods that use arrays
  - the Arrays class
  - command-line arguments

Why are arrays useful?

- Arrays store a large amount of data in one variable.
  - Example: Read in a file of 1000 numbers, then print out the numbers in reverse order.
- Arrays help us group related data into elements.
  - Example: For a school exam, open a file of exam scores and count how many students got each score from 0 through 100.
- Arrays let us access data in random order.
  - Example: Read a file of weather data, store each month's weather stats as an element in a large array, and then examine the stats to find overall weather statistics for the year.

Array initialization statement

- Quick array initialization, general syntax:

  <type> `[]` ***<name>*** `= {`***<value>***`, ` ***<value>***`, ..., ` ***<value>***`};`

  - Example:

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

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	12	49	-2	26	5	17	-6

  - This syntax is useful when you know in advance what the array's element values will be.
  - You don't explicitly specify the array's size in this syntax.
    - The Java compiler figures out the size by looking at the number of values written between { and }.

Array practice problem

- What element values are stored in the following array?

```
int[] a = {2, 5, 1, 6, 14, 7, 9};
for (int i = 1; i < a.length; i++) {
    a[i] += a[i - 1];
}
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	2	7	8	14	28	35	44

The Arrays class

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

Method name	Description
`binarySearch(`*array*`,` *value*`)`	returns the index of the given value in this array (< 0 if not found)
`equals(`*array1*`,` *array2*`)`	returns `true` if the two given arrays contain exactly 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]"`

Arrays.toString

- The `Arrays.toString` method is useful when you want to print an array's elements.
  - `Arrays.toString` accepts an array as a parameter and returns the `String` representation, which you can then print.

  - Example:

```
int[] a = {2, 5, 1, 6, 14, 7, 9};
for (int i = 1; i < a.length; i++) {
    a[i] += a[i - 1];
}
System.out.println("a is " + Arrays.toString(a));
```

Output:

```
a is [2, 7, 8, 14, 28, 35, 44]
```

Traversal algorithms

- suggested reading: 7.1, 7.2, 4.4

Array traversal

- **traversal**: An examination of each element of an array.
  - Traversal algorithms often take the following form:

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

- Traversals are useful in many situations:
  - printing the elements of an array
  - searching an array for a specific value
  - rearranging the elements of an array
  - computing the sum, product, etc. of an array's elements
  - ...

Printing array elements

- Example (print each element of an array on a line):

```
int[] list = {4, 1, 9, 7};
for (int i = 0; i < list.length; i++) {
    System.out.println(i + ": " + list[i]);
}
```

Output:

```
0: 4
1: 1
2: 9
3: 7
```

- How could we change the code to print the following?

```
4, 1, 9, 7
```

(Do not use `Arrays.toString`.)

Examining array elements

- Example (find the largest even integer in an array):

```
int[] list = {4, 1, 2, 7, 6, 3, 2, 4, 0, 9};
int largestEven = 0;
for (int i = 0; i < list.length; i++) {
    if (list[i] % 2 == 0 && list[i] > largestEven) {
        largestEven = list[i];
    }
}
System.out.println("Largest even: " + largestEven);
```

Output:

```
Largest even: 6
```

String traversal

- `Strings` are like arrays of `chars`; they also use 0-based indexes.
  - We can write algorithms to traverse strings to compute information:

```
// string stores voters' votes
// (R)epublican, (D)emocrat, (I)ndependent
String votes = "RDRDRRIDRRRDDDDIRRRDRRRDIDIDDRDDRRDRDIDD";
int[] counts = new int[3];   // R -> 0, D -> 1, I -> 2
for (int i = 0; i < votes.length(); i++) {
    char c = votes.charAt(i);
    if (c == 'R') {
        counts[0]++;
    } else if (c == 'D') {
        counts[1]++;
    } else {  // c == 'I')
        counts[2]++;
    }
}
System.out.println(Arrays.toString(counts));
```

Output:

```
[17, 18, 5]
```

Section attendance problem

- Consider an input file containing course attendance data.
  - Each line represents a section, and each section has 5 students.
  - Every 5 numbers represent 1 week's attendance record.
    - The first of the 5 marks whether the first student attended, the second whether the second student attended, ...
    - A 1 means the student attended, and a 0 means the student did not attend.
    - A student should earn **3** points for each week attended, up to a max of **20**.
  - Example (contains 3 sections and a 9-week attendance record):

```
11111110101111110100111011011011000111001010 0
11101111101010011010111010101011101011010 1
11010101101101111011010101101011011010101 2
```

```
week1 week2 week3 week4 week5 week6 week7 week8 week9
11111 11010 11111 10100 11101 10110 11000 11100 10100

week2
student1 student2 student3 student4 student5
1        1        0        1        0
```

Array transformations

- Problems like this are examples where we use data in one form to compute new data in another form.
  - We sometimes call this *transforming* the data.
  - Often each transformation is stored into its own array.
- Many transformation problems require a mapping between the original data and array indexes.
  - Sometimes the mapping is a tally effect
    (if the input value is the integer *i*, store it into array index *i*)
  - Sometimes the mapping relates to the position of the data
    (store the *i* th value we read into index *i*)
  - Sometimes the mapping is done explicitly
    (count occurrences of "X" into index 0,
    count occurrences of "O" into index 1)

Section attendance problem

- Write a program that reads the preceding section data file and produces output such as the following:

```
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]
```

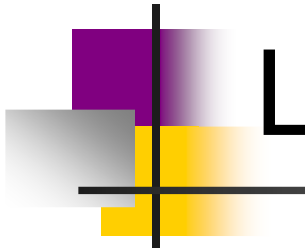
  - Assume the input file exists and is valid.

Section attendance solution

```
// 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"));
        int section = 0;    // used to count sections

        while (input.hasNextLine()) {
            String line = input.nextLine();    // one section's data
            section++;
            System.out.println("Section #" + section + ":");

            int[] attended = new int[5];       // count sections attended
            for (int i = 0; i < line.length(); i++) {
                char c = line.charAt(i);
                if (c == '1') {                // student attended section
                    attended[i % 5]++;
                }
            }
            System.out.println("Sections attended: " + Arrays.toString(attended));

            ...
```

Section attendance solution 2

```
            ...

            // compute section score out of 20 points
            int[] scores = new int[5];
            for (int i = 0; i < scores.length; i++) {
                scores[i] = Math.min(3 * attended[i], 20);
            }
            System.out.println("Student scores: " + Arrays.toString(scores));

            // compute section grade out of 100%
            double[] grades = new double[5];
            for (int i = 0; i < scores.length; i++) {
                grades[i] = 100.0 * scores[i] / 20;
            }
            System.out.println("Student grades: " + Arrays.toString(grades));
            System.out.println();
        }
    }
}
```



Lecture outline

Lecture 18

- Array basics
  - declaring and initializing an array
  - getting and setting values of elements of an array
  - limitations of arrays
  - Arrays for counting and tallying

Lecture 19

- Array traversal algorithms
  - printing an array's elements
  - searching an array
  - reversing an array

Lecture 20

- **Advanced array usage**
  - **arrays as parameters and return values**
  - **String and Graphics methods that use arrays**
  - **the Arrays class**
  - **command-line arguments**
  - **shifting elements in an array**

Arrays as parameters

- suggested reading: 7.1, 3.3

Arrays as parameters

- An array can be passed as a parameter.
  - Syntax (declaration):
    ```
    public static <type> <name>(<type>[] <name>) {
    ```
  - Example:
    ```
    public static double average(int[] numbers) {
    ```
  - Syntax (call):
    ```
    <method name>(<array name>);
    ```
  - Example:
    ```
    int[] scores = {13, 17, 12, 15, 11};
    double avg = average(scores);
    ```

Array parameter example

- Array as parameter example:

```
public static void main(String[] args) {
    int[] iq = {126, 167, 95};
    int result = max(iq);
    System.out.println("Max = " + result);
}

public static int max(int[] array) {
    int largest = array[0];
    for (int i = 1; i < array.length; i++) {
        if (array[i] > largest) {
            largest = array[i];
        }
    }
    return largest;
}
```

- Output:

```
Max = 167
```

Arrays as parameters, contd.

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

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

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

  - Output:

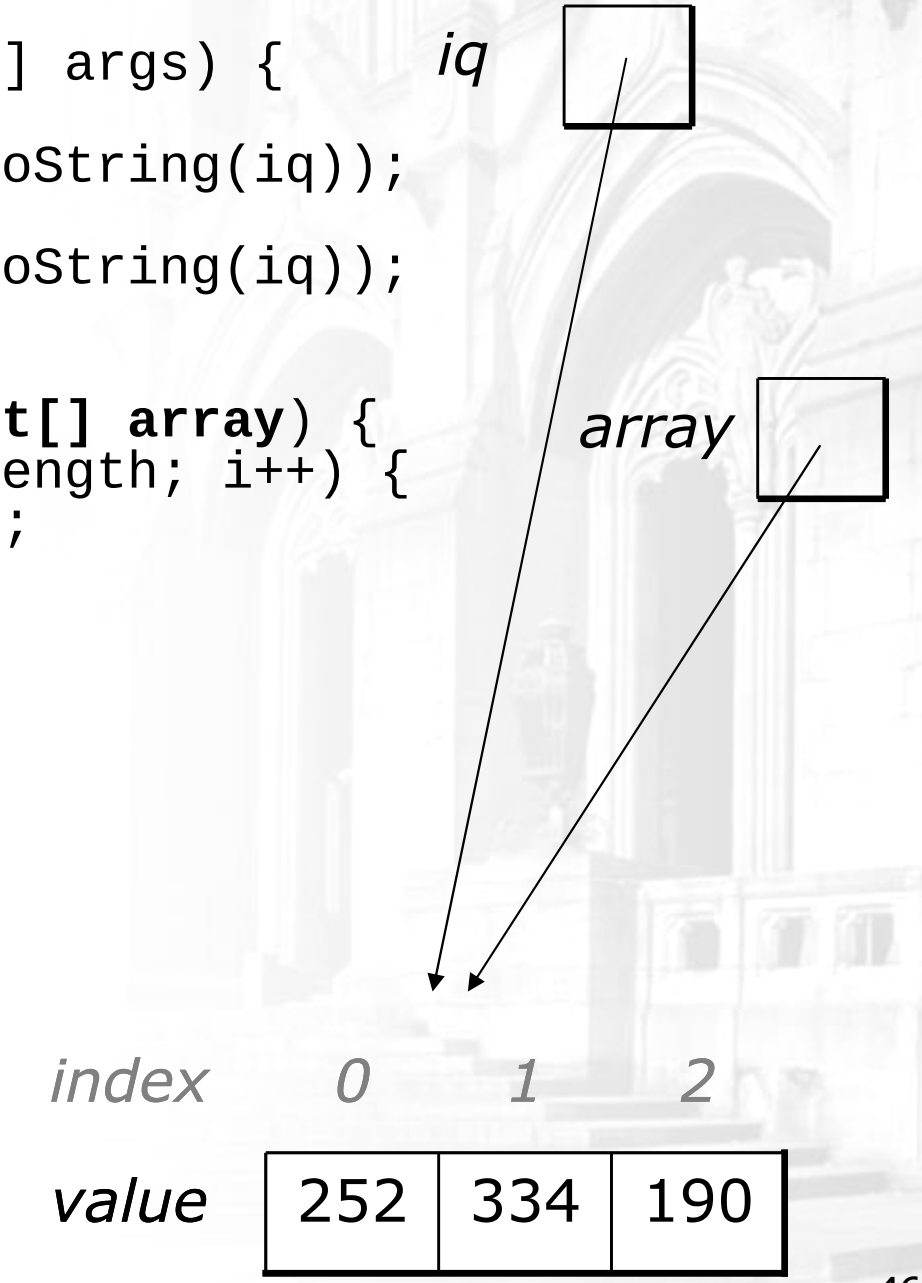
```
[126, 167, 95]
[252, 334, 190]
```

Array parameter diagram

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

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

- Output:

```
[126, 167, 95]
[252, 334, 190]
```



Output parameters

- **output parameter**: An array or object passed as a parameter to a method that has its contents set or altered by that method.
  - We can pass in an array to a method and the method can change its contents in useful ways for us.
  - Examples: The methods `Arrays.fill` and `Arrays.sort`.

Arrays as return values

- An array can also be returned from a method.
  - Syntax (declaration):
    ```
    public static <type>[] <name>(<parameters>) {
    ```
  - Example:
    ```
    public static int[] readAllNumbers(Scanner input) {
    ```
  - Syntax (call):
    ```
    <type>[] <name> = <method name>(<parameters>);
    ```
  - Example:
    ```
    Scanner fileScan = new Scanner(new File("weather.dat"));
    int[] numbers = readAllNumbers(fileScan);
    ```

Array return example

- Example (digit-counting problem from an earlier slide):

```
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;
}

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

Output:

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

Array parameter problems

- Write a method named `average` that accepts an array of integers as its parameter and returns the average of the values in the array.

- Write a method named `contains` that accepts an array of integers and a target integer value as its parameters and returns whether the array contains the target value as one of its elements.

- Write a method named `roundAll` that accepts an array of `doubles` as its parameter and modifies each element of the array so that it is rounded to the nearest whole number.

- Improve the previous Histogram and Sections programs by making them use parameterized methods.

Array parameter solutions

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

public static boolean contains(int[] values, int target) {
    for (int i = 0; i < values.length; i++) {
        if (values[i] == target) {
            return true;
        }
    }
    return false;
}

public static void roundAll(double[] array) {
    for (int i = 0; i < array.length; i++) {
        array[i] = Math.round(array[i]);
    }
}
```

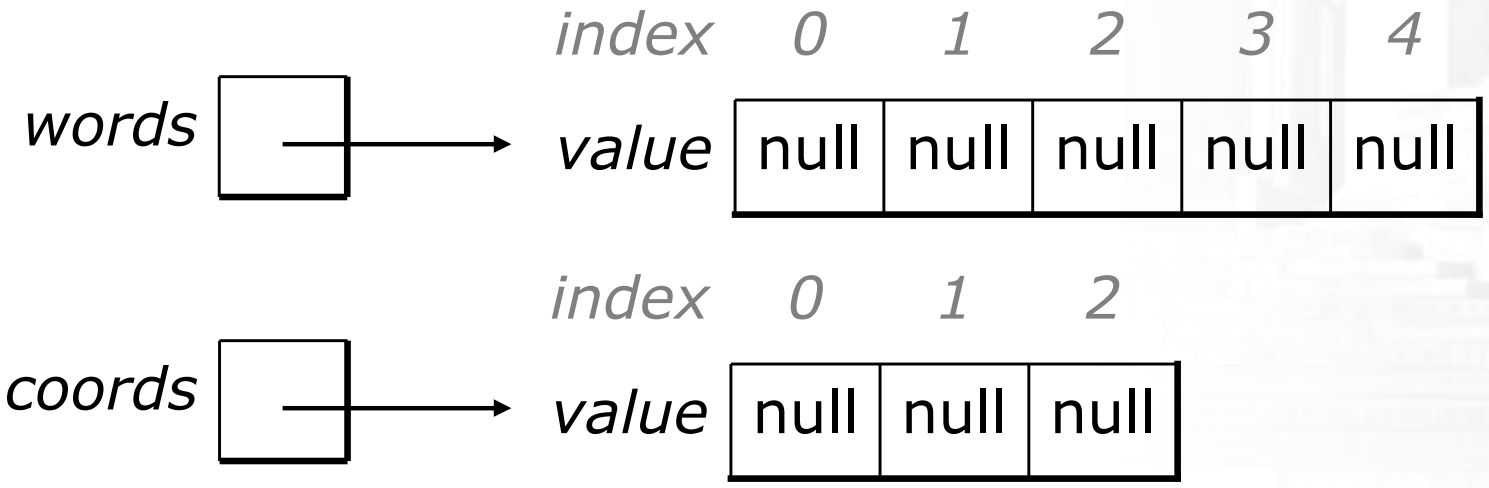
Arrays of objects and `null`

- suggested reading: 7.2

Arrays of objects

- Recall: when you construct an array of primitive values like `int`s, the elements' values are all initialized to 0.
- The elements of an array of objects are initialized to store a special reference value called `null`.
  - **`null`:** A reference that does not refer to any object.

```
String[] words = new String[5];
Point[] coords = new Point[3];
```



Properties of null

- It is legal (will not throw an exception) to:
  - store `null` in a variable or array element
    - (often as a dummy value, an initial value to be overwritten later)

```
String s = null;
words[2] = null;
```

  - print a `null` reference

```
System.out.println(s);   // output: null
```

  - ask whether a variable or array element is `null`

```
if (words[i] == null) { ...
```

  - pass `null` as a parameter to a method
  - return `null` from a method
    - (often as an indication of failure, such as a method that searches for an object in a file/array but does not find it)

NullPointerException

- It is *not* legal to dereference a null value (trying to access any of its methods or data using . notation)
  - Since `null` is not an actual object, it has no methods or data. Trying to access them crashes your program.

- Example:

```
String[] words = new String[5];
System.out.println("word 0 is: " + words[0]);
words[0] = words[0].toUpperCase();   // bad
System.out.println("word 0 is now: " + words[0]);
```

- Output:

```
word 0 is: null
Exception in thread "main"
java.lang.NullPointerException
        at Example.main(Example.java:8)
```

Looking before you leap

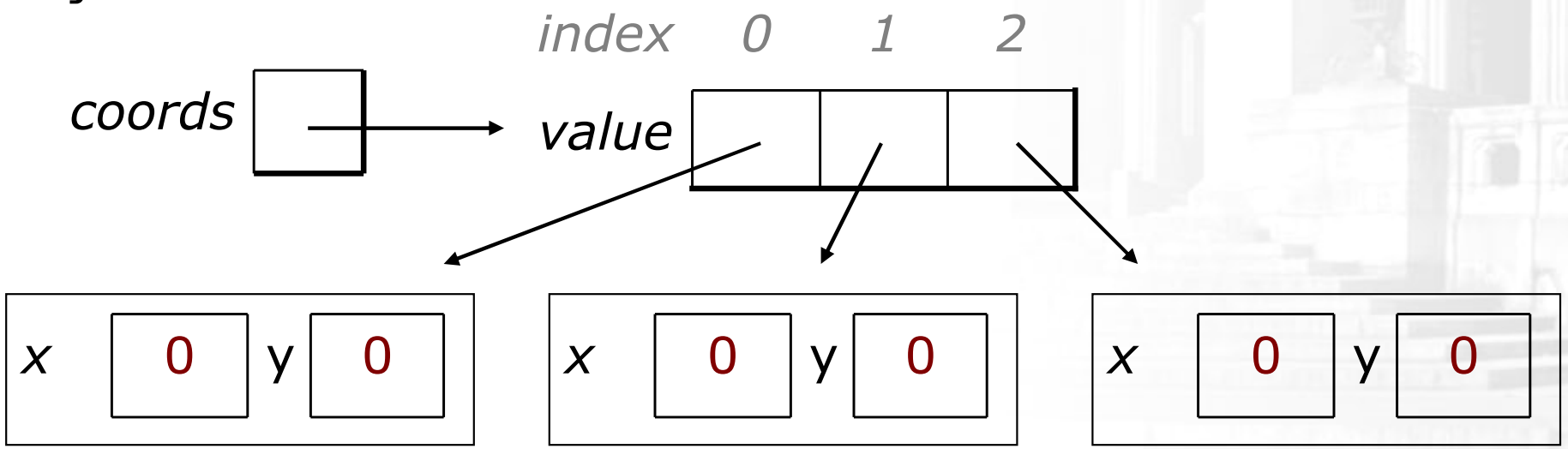
- When dealing with elements that may be `null`, you must take care to check for `null` before any calls.

```
String[] words = new String[5];
words[0] = "hello";
words[2] = "goodbye";   // words[1], [3], [4] are null

int totalLetters = 0;
for (int i = 0; i < words.length; i++) {
    if (words[i] != null) {
        totalLetters += words[i].length();
    }
}
System.out.println("letters: " + totalLetters);  // 12
```

Two-phase initialization

- Arrays of objects should use a *two-phase initialization*:
  1) initializing the array itself
  2) initializing the object stored into each element of the array
  - Example two-phase initialization:

```
Point[] coords = new Point[3];              // phase 1
for (int i = 0; i < coords.length; i++) {
    coords[i] = new Point(0, 0);            // phase 2
}
```

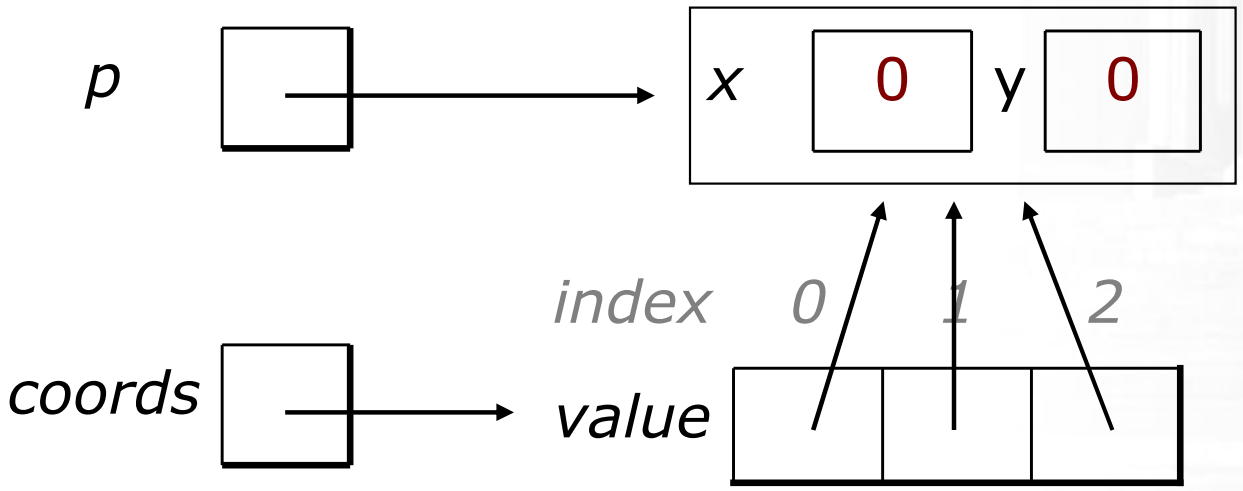


A subtlety

- Consider the following code:

```
Point[] coords = new Point[3];                    // phase 1
Point p = new Point(0, 0);
for (int i = 0; i < coords.length; i++) {
    coords[i] = p;                                // phase 2
}
```

  - Does it do the same thing as the code on the previous slide?
  - What happens if we change `coords[0]` ?



Arrays of objects question

- Given a file containing a number of cities' (x, y) coordinates, which begins with the number of cities:

```
6
50 20
90 60
10 72
74 98
5 136
150 91
```

- Write a program to read this data, then prompt the user for their home coordinates and output distances:

```
Type your x and y coordinates: 10 20
You are 40.0 miles from (50, 20)
You are 89.44 miles from (90, 60)
You are 52.0 miles from (10, 72)
You are 100.89 miles from (74, 98)
You are 116.11 miles from (5, 136)
You are 156.97 miles from (150, 91)
```

- Note: Use arrays and `Point` objects as part of your solution.

Arrays of objects solution

```
// Reads city x/y data from a file and computes distances.
import java.awt.*;
import java.io.*;
import java.util.*;

public class Cities {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner input = new Scanner(new File("cities.txt"));
        int numCities = input.nextInt();    // file begins with # cities

        Point[] cities = new Point[numCities];
        for (int i = 0; i < numCities; i++) {  // read cities into array
            cities[i] = new Point(input.nextInt(), input.nextInt());
        }

        Scanner console = new Scanner(System.in);
        System.out.print("Type your x and y coordinates: ");
        Point you = new Point(console.nextInt(), console.nextInt());

        for (int i = 0; i < numCities; i++) {
            double dist = you.distance(cities[i]);
            System.out.println("You are " + dist + " miles from (" +
                               cities[i].x + ", " + cities[i].y + ")");
        }
    }
}
```

String methods with arrays

- These `String` methods return arrays:

```
String s = "long book";
```

Method name	Description	Example
`toCharArray()`	separates this string into an array of its characters	`s.toCharArray()` returns `{'l', 'o', 'n', 'g', ' ', 'b', 'o', 'o', 'k'}`

`split(`*delimiter*`)`	separates this string into substrings by the given delimiting string	`s.split(" ")` returns `{"long", "book"}`
		
		`s.split("o")` returns `{"l", "ng b", "", "k"}`

String/array problems

- Write a method named `areAnagrams` that accepts two `Strings` as its parameters and returns whether those two strings contain the same letters (possibly in different orders).
  - `areAnagrams("bear", "bare")` returns `true`
  - `areAnagrams("sale", "sail")` returns `false`

  Use the methods from the previous slide in your answer, as well as any relevant methods from the `Arrays` class.

- Write a method named `wordCount` that accepts a `String` as its parameter and returns the number of words in that string. Words are separated by single spaces.
  - `wordCount("the quick brown fox")` returns `4`

  Use the methods from the previous slide in your answer.

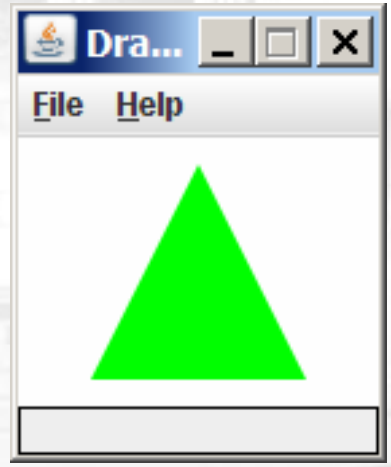
Graphics methods w/ arrays

- These methods of `Graphics` objects use arrays:

Method name
`drawPolygon(int[] xPoints, int[] yPoints, int length)`
`drawPolyline(int[] xPoints, int[] yPoints, int length)`
`fillPolygon(int[] xPoints, int[] yPoints, int length)`

- Example:

```
DrawingPanel p = new DrawingPanel(100, 100);
Graphics g = p.getGraphics();
int[] xPoints = {10, 50, 90};
int[] yPoints = {90, 10, 90};
g.setColor(Color.GREEN);
g.fillPolygon(xPoints, yPoints, 3);
```



Command-line arguments

- **command-line arguments**: Parameters passed to your program as it is run.
  - The parameters are passed into `main` as an array of `Strings`.
  - The parameters can be typed into a command prompt or terminal window, or directly in some editors such as DrJava.

```
public static void main(String[] args) {
   for (int i = 0; i < args.length; i++) {
      System.out.println("arg " + i + ": " + args[i]);
   }
}
```

- Example:

```
> java ExampleProgram how are you?
arg 0: how
arg 1: are
arg 2: you?
```

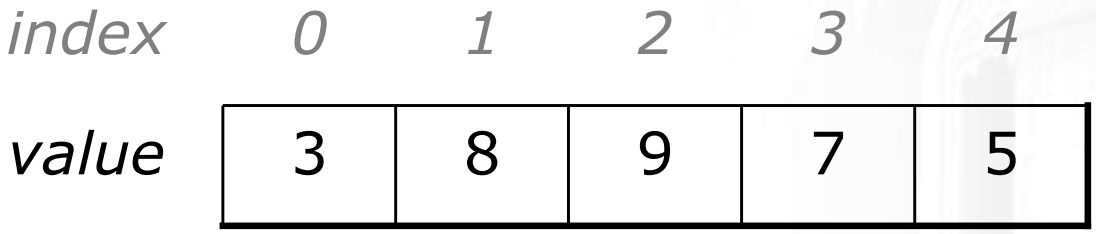
Shifting elements in an array

- suggested reading: 7.2

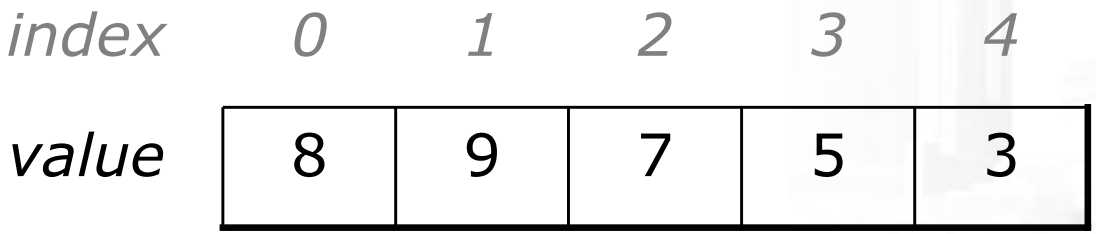
Concept of an array rotation

- Imagine we want to 'rotate' the elements of an array; that is, to shift them left by one index.
  - The element that used to be at index 0 will move to the last slot in the array.
  - For example, `{3, 8, 9, 7, 5}` becomes `{8, 9, 7, 5, 3}`.

Before:



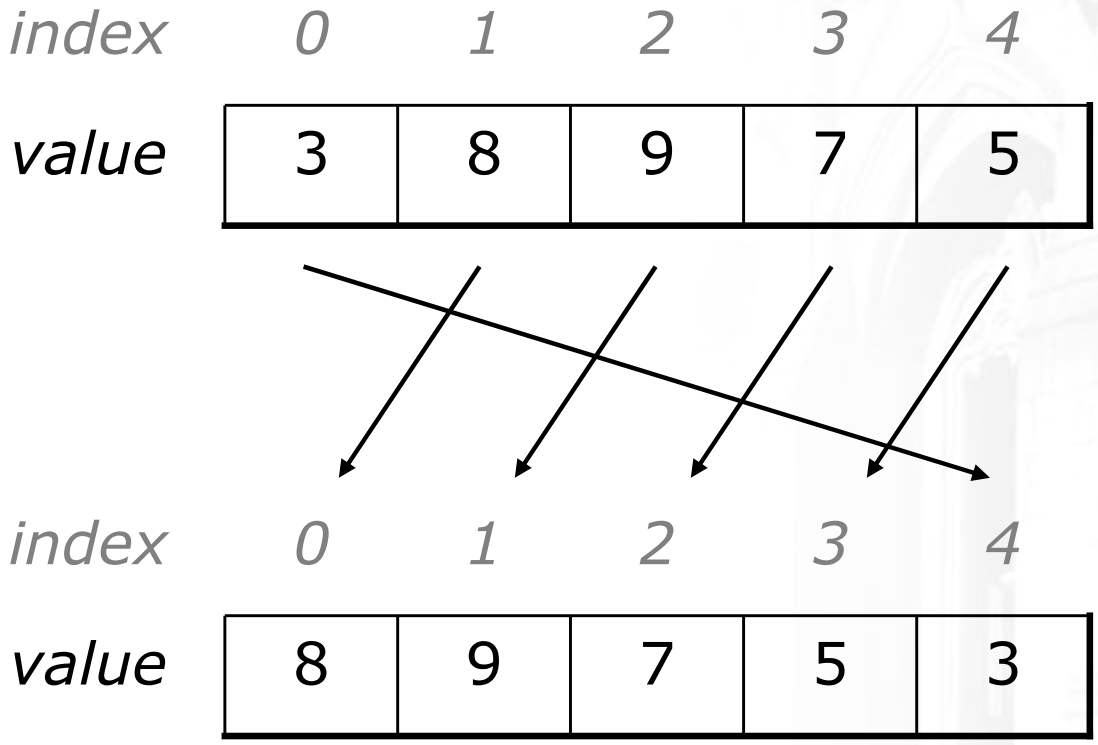
After:



  - Shifting elements is useful when inserting and removing values from arrays after they have already been filled with data.

Shifting elements left

- A left shift of the elements of an array:



- Let's write the code to do the left shift.
  - Can we generalize it so that it will work on an array of any size?
  - Can we write a right-shift as well?

Shifting practice problem

- Write a method `insertInOrder` that accepts a sorted array *a* of integers and an integer value *n* as parameters, and inserts *n* into *a* while maintaining sorted order.

  In other words, assume that the element values in *a* occur in sorted ascending order, and insert the new value n into the array at the appropriate index, shifting to make room if necessary. The last element in the array will be lost after the insertion.

  - Example: calling `insertInOrder` on array `{1, 3, 7, 10, 12, 15, 22, 47, 74}` and value 11 produces `{1, 3, 7, 10, 11, 12, 15, 22, 47}`.