Building Java Programs

Chapter 7: Arrays

Chapter outline

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

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

- advanced array usage
  - arrays as parameters to methods
  - String and Graphics methods that use arrays
  - the Arrays class
  - shifting elements in an array

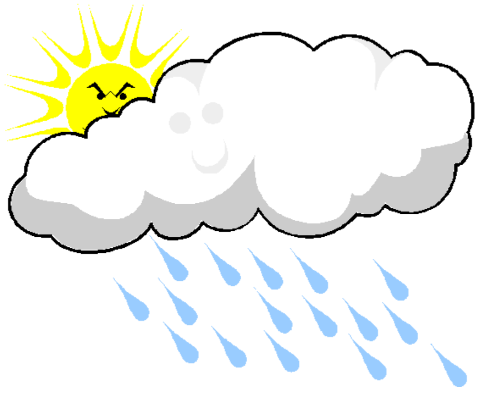
Array basics

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.
```



Why the problem is tough

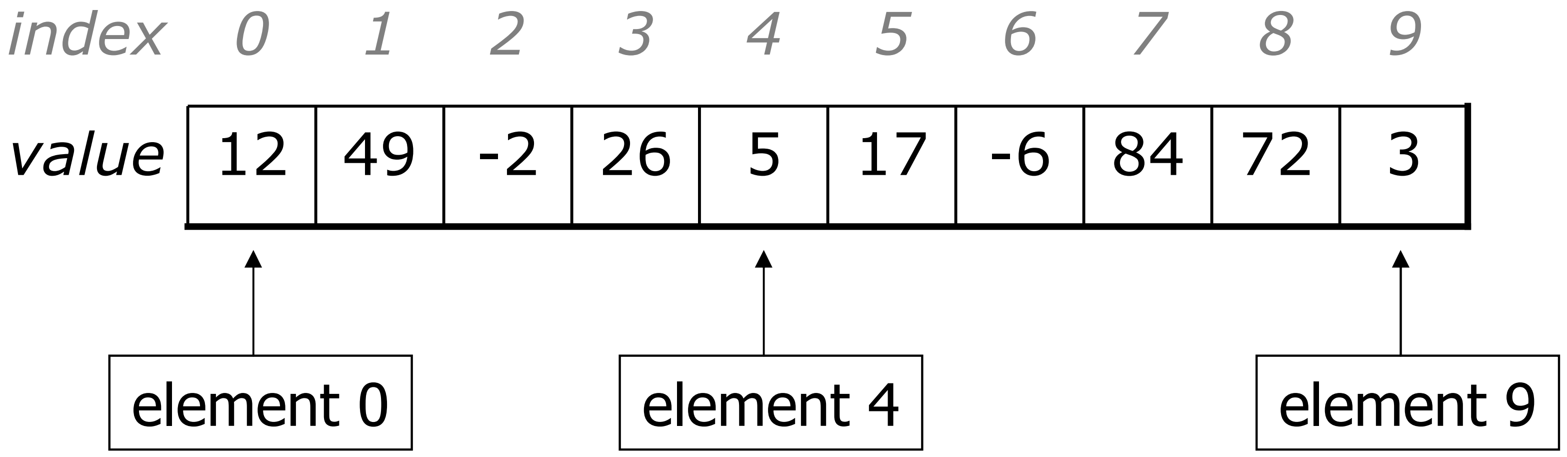
- We need each input value twice:
  - to compute the average (a cumulative sum)
  - to count how many were above average

- We could read each value into a variable...
  - However, we don't know how many variables to declare.
  - We don't know how many days are needed until the program runs.

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

Arrays

- **array**: An 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

- Declaring/initializing an array:

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

  - 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

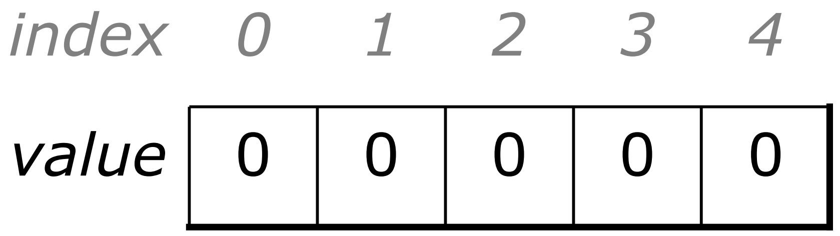
- The length can be any integer expression.
  - Example:

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

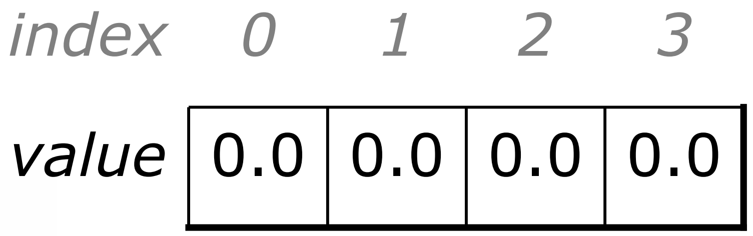
Array auto-initialization

- When arrays are constructed, each element is given a "zero-equivalent" value.

`int:`	`0`	
`double:`	`0.0`	
`boolean:`	`false`	
`char:`	`'\0'`	(the "null character")
object (e.g. `String`):	`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;
    ```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
value	**27**	0	0	**-6**	0	0	0	0	0	0

Accessing array elements

- Accessing an array element's value:
  <array name> [***<index>***]

  - Example:

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

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
value	**27**	0	0	**-6**	0	0	0	0	0	0

Out-of-bounds

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

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

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;
```

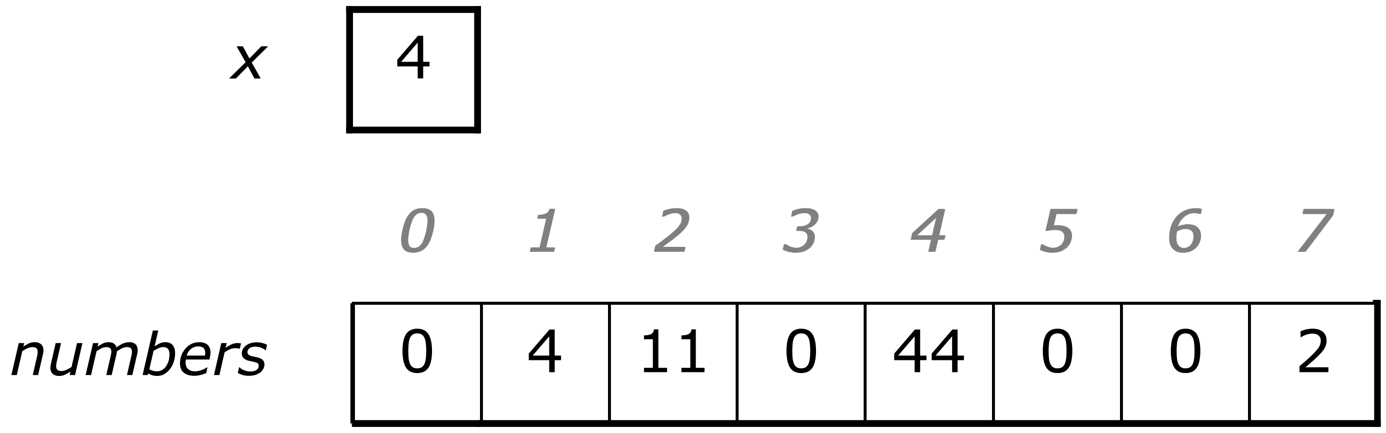
index	*0*	*1*	*2*	*3*	*4*	*5*
value	false	false	false	**true**	false	false

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

- It's common to use for loops to access array elements.
  - 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:
  ```
  for (int i = 0; i < numbers.length; i++) {
      System.out.print(numbers[i] + " ");
  }
  ```
  - Output:
    ```
    0 1 4 9 16 25 36 49
    ```
- What expressions refer to:
  - the last element of an 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 = 44.57142857142857
4 days were above average.
```

Weather answer

```
// 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

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 could declare 10 counter variables for this...

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

  - Yuck!

- 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:

```java
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;
}
```

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

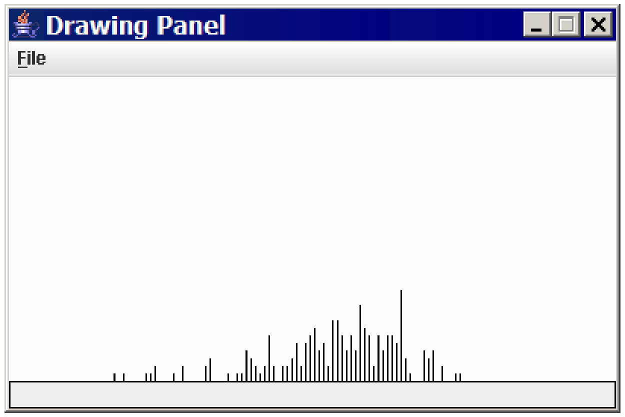
Array histogram question

- 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 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]);
        }
    }
}
```

Why are arrays useful?

- Storing a large amount of data
  - Example: Read a file of numbers and print them in reverse order.

- Grouping related data
  - Example: Tallying exam scores from 0 through 100.

- Accessing data multiple times, or in random order
  - Example: Weather program.

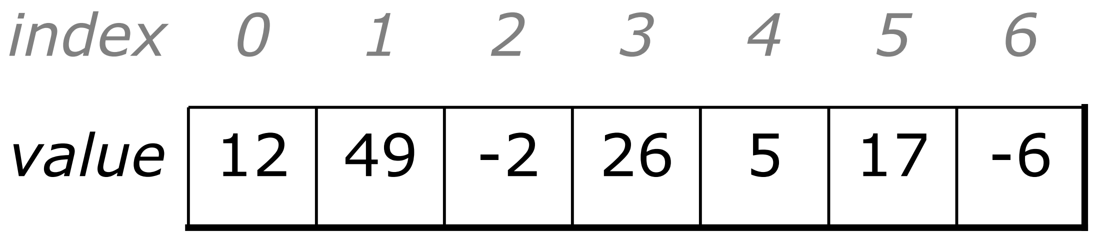
Array initialization statement

- 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

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

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

- `Arrays.toString` accepts an array as a parameter and returns its data as a `String`, which you can 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

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];
}
```

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

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
```

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
```

Strings and arrays

- `String`s are represented internally as arrays of `char`.
  - They also use 0-based indexes.
  - We can write algorithms to traverse strings.

  - Example:

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

index	*0*	*1*	*2*	*3*	*4*	*5*
value	`'A'`	`'l'`	`'i'`	`' '`	`'G'`	`'.'`

String traversal example

```
// string stores voters' votes: (R)epub., (D)emo., (I)ndep.
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') {             // put vote in proper box
        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 of course attendance data:

```
111111101011111101001110110110110001110010100
111011111010100110101110101010101110101101010
110101011011011011110110101011010111011010101
```

```
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
```

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

Array transformations

- In this problem we convert data from one form to another.
  - This is called *transforming* the data.
  - Often each transformation is stored into its own array.

- Transformation problems require a mapping between the original data and array indexes.

  Examples:

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

Section attendance problem

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

```
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]
```

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();
        }
    }
}
```

- The program can be improved:
  - It doesn't have any static methods.
  - To add methods, we'll need to pass arrays as parameters.

Arrays as parameters

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

```
public static void main(String[] args) {
    int[] iq = {126, 84, 149, 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 passed by reference

- Arrays are objects.
  - When 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};
    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:

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

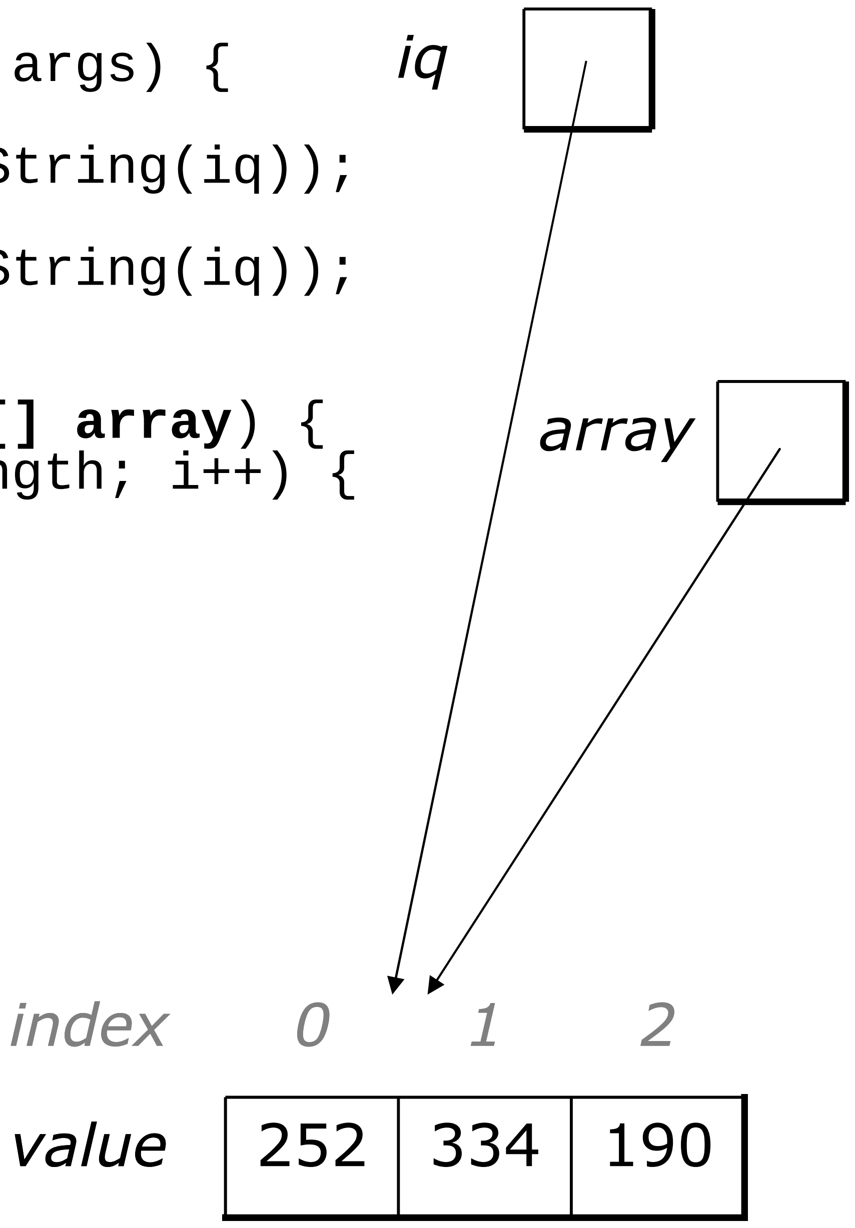
Array parameter diagram

```
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 that has its contents altered by the method.
  - We can pass an array and the method can change its contents.
  - Example:

```
int[] nums = {5, -1, 3, 14, 8, 7};
Arrays.sort(nums);
System.out.println(Arrays.toString(nums));
Arrays.fill(nums, 42);
System.out.println(Arrays.toString(nums));
```

Output:

```
[-1, 3, 5, 7, 8, 14]
[42, 42, 42, 42, 42, 42]
```

Arrays as return values

- An array can be returned from a method.
  - Syntax (declaration):

    public static ***<type>***[] ***<name>***(***<parameters>***) {

  - Example:

    ```
    public static int[] countDigits(int n) {
        ...
    }
    ```

  - Syntax (call):

    <type>[] ***<name>*** = ***<method name>***(***<parameters>***);

  - Example:

    ```
    int[] digits = countDigits(229231007);
    ```

Array return 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;
}

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 parameter questions

- Write a method named `average` that accepts an array of integers and returns the average of the element values.
- Write a method named `contains` that accepts an array of integers and a target value and returns whether the array contains the target value.
- Write a method named `roundAll` that accepts an array of `double`s and rounds each to the nearest whole number.
- Improve the previous Histogram and Sections programs by making them use parameterized methods.

Array parameter answers

```
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`

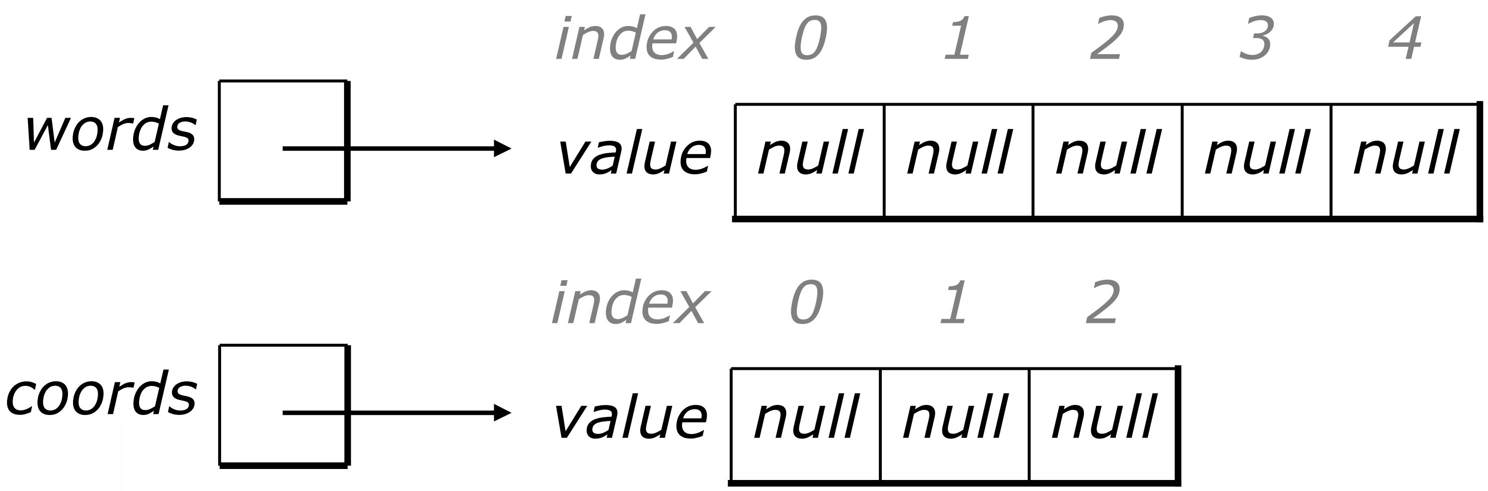
reading: 7.2

Arrays of objects

- Recall: When you construct an array of `int`s, the elements' values are initialized to 0.

- The elements of an array of objects are initialized to a 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 to:
  - store `null` in a variable or array element

    ```
    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)

Null pointer exception

- It is not legal to dereference a `null` value.
  - **dereference**: Accessing the fields or methods of an object using the . notation (such as `p.x` or `s.length()`).
  - Since `null` is not an actual object, it has no methods or data. Trying to access them causes an exception.

- 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:

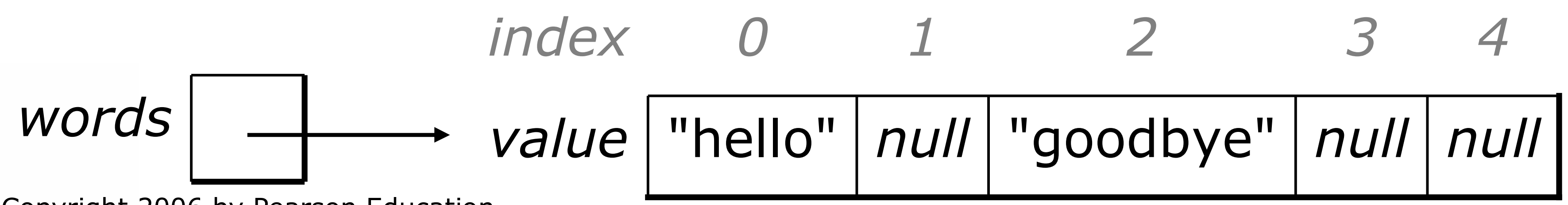
```
word 0 is: null
Exception in thread "main"
java.lang.NullPointerException
        at Example.main(Example.java:8)
```

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
```

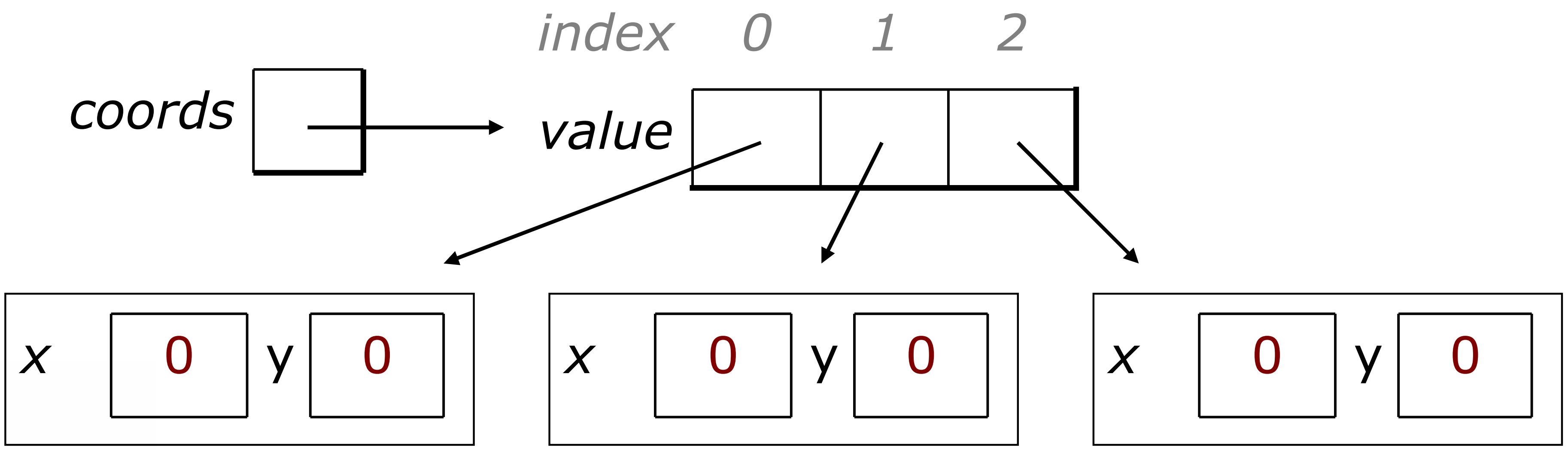


Two-phase initialization

- Arrays of objects use a *two-phase initialization*:
  1) initializing the array itself (each element is initially `null`)
  2) initializing each element of the array to be a new object

  - Example:

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

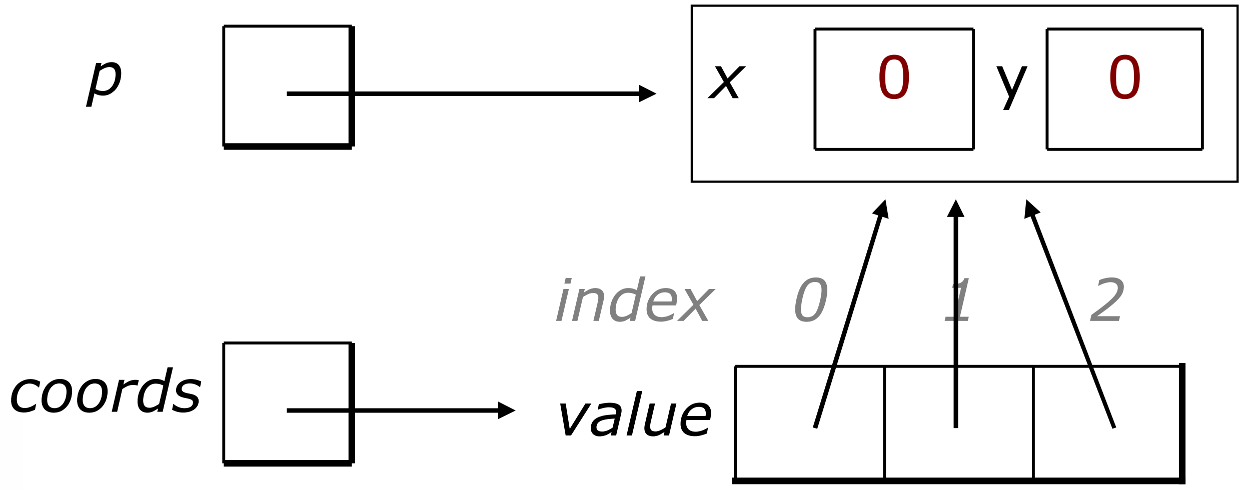


A subtlety

- 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 = "robot";
```

Method name	Description	Example
`toCharArray()`	separates this string into an array of its characters	`s.toCharArray()` returns `{'r', 'o', 'b', 'o', 't'}`

`split(`*delimiter*`)`	separates this string into substrings by the given delimiting string	`s.split("b")` returns `{"ro", "ot"}`
		
		`s.split("o")` returns `{"r", "b", "t"}`

String/array problems

- Write a method named `areAnagrams` that accepts two `Strings` and returns whether they contain the same letters.
  - `areAnagrams("bear", "bare")` returns `true`
  - `areAnagrams("sale", "sail")` returns `false`
  - Use methods from the previous slide and from the `Arrays` class.

- Write a method named `wordCount` that accepts a `String` and returns the number of words in that string (separated by " ").
  - `wordCount("the quick brown fox")` returns `4`
  - Use the methods from the previous slide.

```java
public static boolean areAnagrams(String s1, String s2) {
    char[] chars1 = s1.toCharArray();
    char[] chars2 = s2.toCharArray();
    Arrays.sort(chars1);
    Arrays.sort(chars2);
    return Arrays.equals(chars1, chars2);
}

public static int wordCount(String s) {
    String[] words = s.split(" ");
    return words.length;
}
```

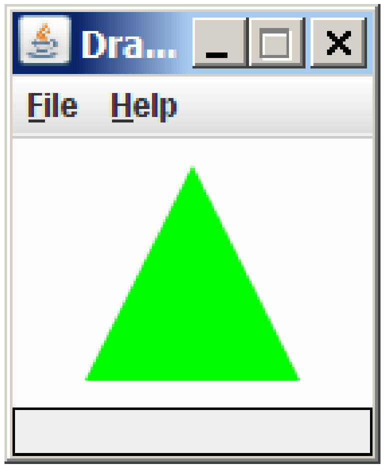
Graphics methods w/ arrays

- 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:

```
// triangle: (10, 90), (50, 10), (90, 90)
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 `String`s.
  - 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

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 from index 0 will move to the last slot.
  - Example: `{3, 8, 9, 7, 5}` becomes `{8, 9, 7, 5, 3}`

Before:

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

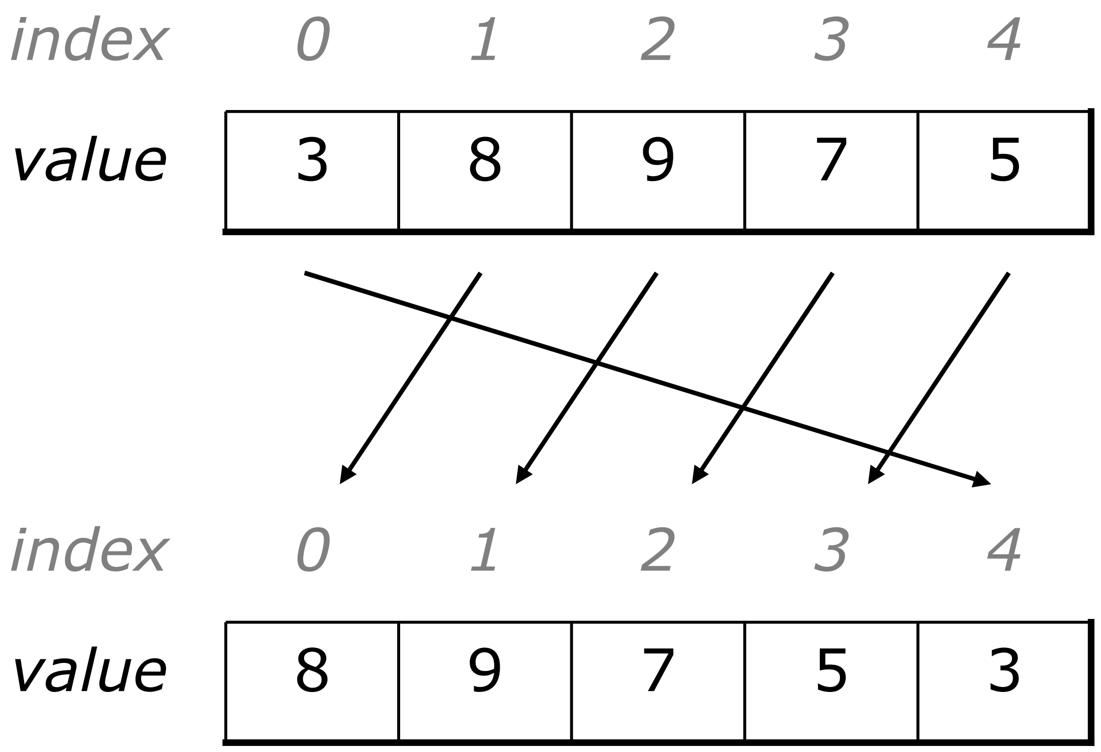
After:

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

  - Shifting elements is useful when inserting and removing values from arrays that already contain other 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?

Left rotate solution

```
public static void rotateLeft(int[] array) {
    // set aside first element
    int first = array[0];

    // shift all other elements left by 1
    for (int i = 0; i < array.length - 1; i++) {
        array[i] = array[i + 1];
    }

    // move first element to the end
    array[array.length - 1] = first;
}
```

Shifting question

- 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`*`}`.

Shifting answer

```
public static void insertInOrder(int[] a, int n) {
    for (int i = 0; i < a.length; i++) {
        if (n < a[i]) {
            // we've found the proper place to insert;
            // shift remaining elements to the right
            for (int j = a.length - 1; j >= i + 1; j--) {
                a[j] = a[j - 1];
            }

            // add the new element
            a[i] = n;
            break;
        }
    }
}
```