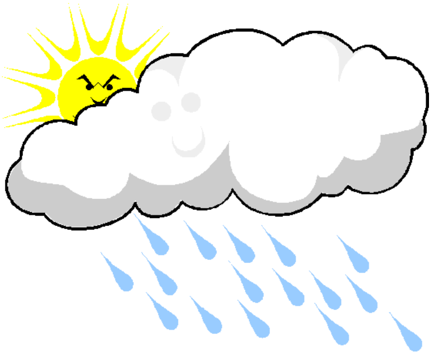
Array basics

Readings: 7.1

How would you solve this?



- Consider the following program:

```
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.
```

What makes the problem hard?

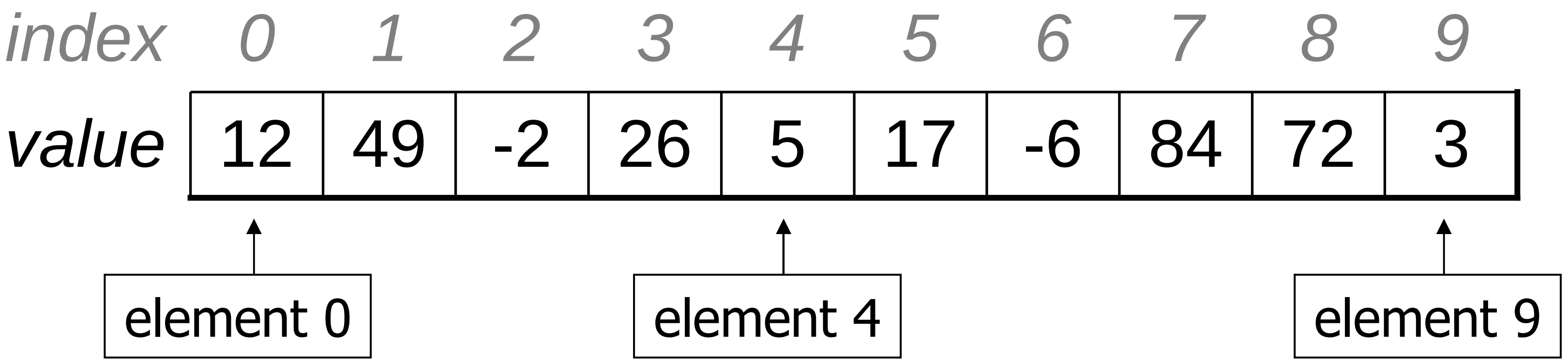
- We need each input value twice
  - … to compute the average via a cumulative sum
  - … to count how many were above the average

- What about putting the values into variables?
  - How many variables would we declare?

- Need a way to declare many variables at once.

Arrays

- **array**: An object that stores many values of the same type.
  - **element**: a value in an array
  - **index**: an integer indicating the position of a value in an array



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

element 0

element 4

element 9

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:

```
  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`
  - object type: `null` (`null` means "no object")

Array auto-initialization: Example

- An array of doubles

index	*0*	*1*	*2*	*3*	*4*
value	0.0	0.0	0.0	0.0	0.0

- An array of booleans

index	*0*	*1*	*2*	*3*
value	false	false	false	false

Assigning 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

- 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.");
}
```

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

Don't go out of bounds!

- Reading or writing any index outside the valid 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!
```

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

Example

```
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!
```

x: 4

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
numbers:	0	4	11	0	44	0	0	2

Arrays and `for` loops

- Arrays are very commonly used with `for` loops to access each element

- Example:

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

Output:

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

Arrays and `for` loops

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

- What's in the array?

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

Arrays and `for` loops

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

- What's in the array?

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
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`

- NB: Because it's a field (i.e. not a method), it does not use parentheses like a String's `.length()`!

Example

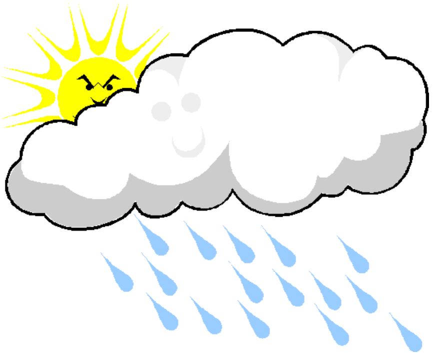
```
for (int i = 0; i < numbers.length; i++) {
    System.out.print(numbers[i] + " ");
}
```

Output:

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

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

How it all started…



- 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.
```

Solution

```java
// 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 / tallying

Readings: 7.1

A multi-counter problem

- Problem: Examine a number and count the number of occurrences of every digit.
  - Example: The number 229231007 contains: two 0s, one 1, three 2s, one 7, and one 9

- Solution?
  - Declare 10 counter variables—one per digit. Eeewww!!!!

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

A multi-counter problem

- Problem: Examine a number and count the number of occurrences of every digit.
  - Example: The number 229231007 contains: two 0s, one 1, three 2s, one 7, and one 9

- Solution:
  - Declare an array of 10 elements—the element at index *i* will store the counter for digit value *i*.

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

An array of 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

Histogram: Exercise

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

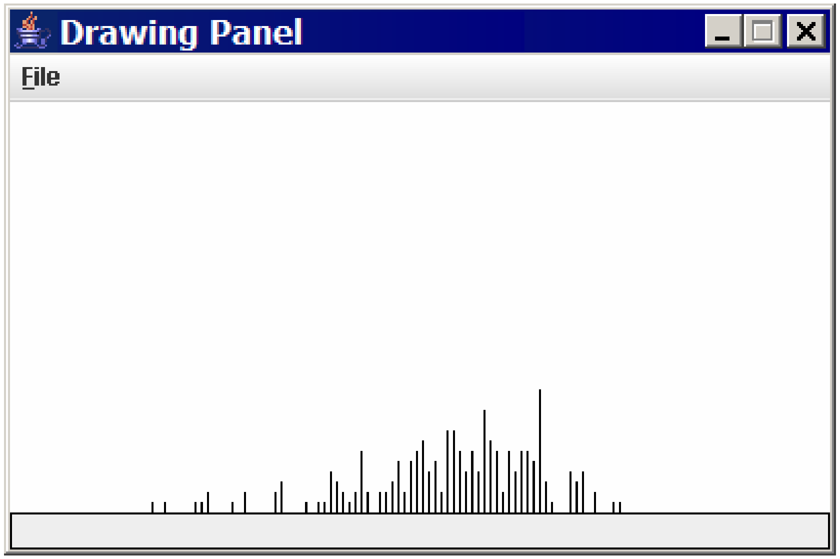
```
82
66
79
63
83
```

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: Exercise

- 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`.



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 = 7;   // 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();
            }
        }

        ...
```

Histogram: Solution

```
        ...

        // 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

- Arrays store a large amount of data accessible from one variable.
- Arrays help us group related data into elements.
- Arrays let us access data in random order.
  - Cassette tape vs. DVD

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 in advance what the array's element values will be.

Example

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

- What's in the array?

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	2	5 7	1 8	6 14	14 28	7 35	9 44

Printing arrays: `Arrays.toString`

- `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));
```

<u>Output</u>:

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

Traversal algorithms

Readings: 7.2

Array traversal

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

- Traversal algorithms often takes the following form:

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

- Examples:
  - printing out the elements
  - searching for a specific value
  - rearranging the elements
  - computing a value based on the elements

Example: Printing array elements

```
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
```

Example: Searching 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
```

- What assumptions does this code make?

String traversal

- `String`s are like arrays of `char`s.

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	`'l'`	`'e'`	`'t'`	`'t'`	`'e'`	`'r'`	`'s'`

- We can write algorithms to traverse strings to compute information.

- What useful information might the following string have?

```
"BDRBRRBDRRBDMBDBRRRBRBRBBDBDDRDDRRDBDBBD"
```

String traversal: Example

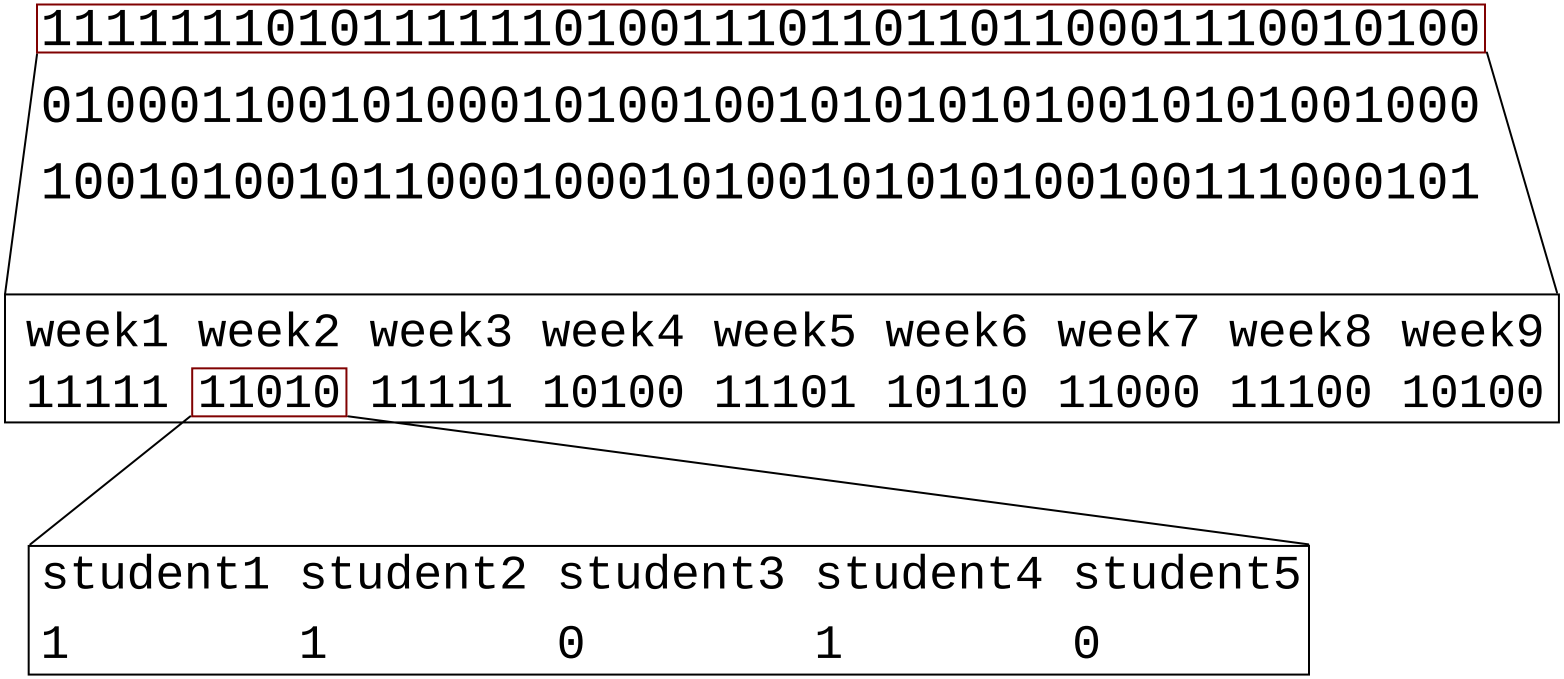
```
// string stores voters' votes
// (R)EPUBLICAN, (D)EMOCRAT, (B)ENSON, (M)ARTY
String votes = "BDRBRRBDRRBDMBDBRRRBRBRBBDBDDRDDRRDBDBBD";
int[] counts = new int[4]; // R -> 0, D -> 1, B -> 2, M -> 3
for (int i = 0; i < votes.length(); i++) {
    char c = votes.charAt(i);
    if (c == 'R') {
        counts[0]++;
    } else if (c == 'D') {
        counts[1]++;
    } else if (c == 'B') {
        counts[2]++;
    } else {  // c == 'M'
        counts[3]++;
    }
}
System.out.println(Arrays.toString(counts));
```

Output:

```
[13, 12, 14, 1]
```

Example data: Section attendance

- Consider the following dataset which represents attendance for three sections of five students:



```
111111101011111101001110110110110001110010100
010001100101000101001001010101010010101001000
100101001011000100010100101010100100111000101
```

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

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

Data transformations

- Sometimes we will use data in one form to compute new data in another form.
  - Often each *transformation* is stored into its own array.

- Transformations require a *mapping* between the original data and array indices.

Typical mappings

- Tally

  “If the input value is the integer *i*, do something with array index *i.*”

- Based on the position in the data

  “Store the *i* th value we read into index *i*.”

- Explicit mappings

  “Count occurrences of `'R'` into index 0 and occurrences of `'D'` into index 1.”

Exercise: Section attendance

- 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, 20, 20, 16, 12]
Student grades: [100.0, 100.0, 100.0, 80.0, 60.0]

Section #2:
Sections attended: [4, 6, 2, 2, 3]
Student scores: [16, 20, 8, 8, 12]
Student grades: [80.0, 100.0, 40.0, 40.0, 60.0]

Section #3:
Sections attended: [5, 4, 2, 5, 3]
Student scores: [20, 16, 8, 20, 12]
Student grades: [100.0, 80.0, 40.0, 100.0, 60.0]
```

Solution: Section attendance

```
// 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 = 1;    // used to count sections

        while (input.hasNextLine()) {
            String line = input.nextLine();    // one section's data
            processSection(section, line);
            section++;
        }
    }
```

Solution: Section attendance

```
public static void processSection(int sectionNum, String line) {
    System.out.println("Section #" + sectionNum + ":");

    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));

    ...
```

Solution: Section attendance

```
        // compute section score out of 20 points
        int[] scores = new int[5];
        for (int i = 0; i < scores.length; i++) {
            scores[i] = Math.min(4 * 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();
    }
}
```

Arrays and methods

Readings: 7.1

Arrays as parameters

- Declaration, syntax:

```
public static <type> <name>(<type>[] <name>) {
```

<u>Example</u>:

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

- Method call, syntax:

```
<method name>(<array name>);
```

<u>Example</u>:

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

Example: Arrays as parameters

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

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 are objects

- When arrays are passed as parameters, they are passed by *reference.*

  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:

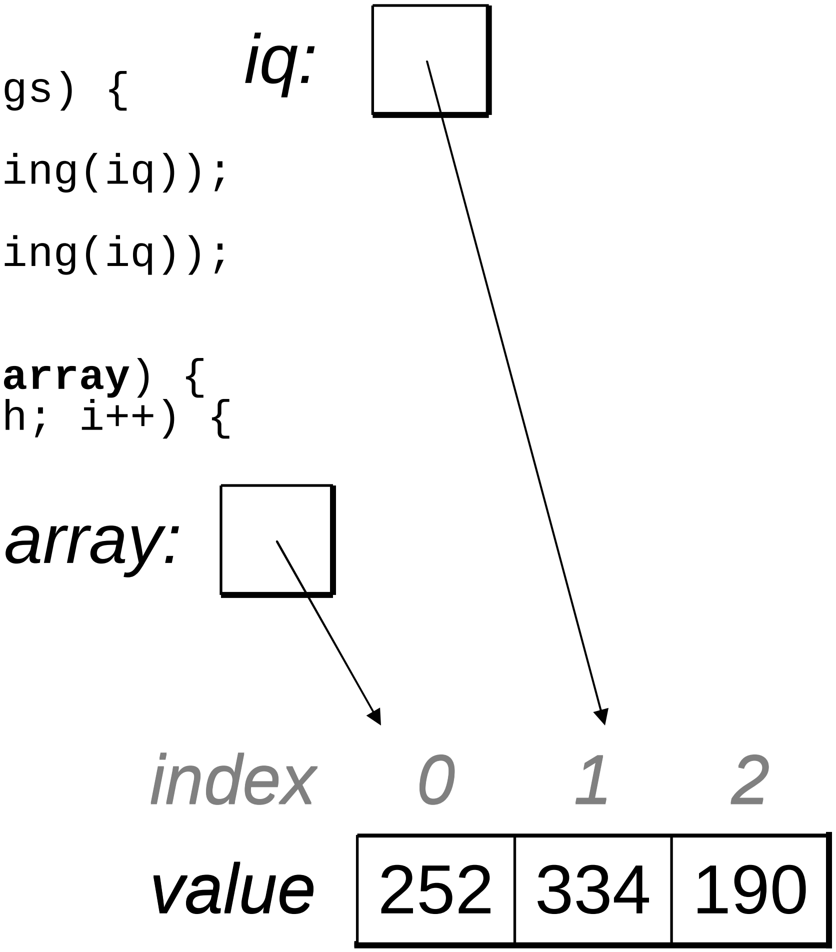
  ```
  [126, 167, 95]
  [252, 334, 190]
  ```

Arrays are objects

```
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];
    }
}
```

<u>Output</u>:

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



index	*0*	*1*	*2*
value	252	334	190

Useful result: Output parameter

- **output parameter**: An object passed as a parameter that has its contents altered by the method.

- We can pass an array to a method and the method can change its contents in useful ways.

  <u>Example</u>:

  After calling `Arrays.sort(`***<array>***`)`, the array passed in will be in sorted order.

Arrays as return values

- Declaration, syntax:

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

  Example:

```
public static int[] readAllNumbers(Scanner input) {
```

- Method call, syntax:

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

  Example:

```
Scanner fileScan = new Scanner(new File("nums.txt"));
int[] numbers = readAllNumbers(fileScan);
```

Example: Arrays as return values

```
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]
```

Exercises

- 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 grade histogram and section attendance programs by making them use parameterized methods.

Solutions

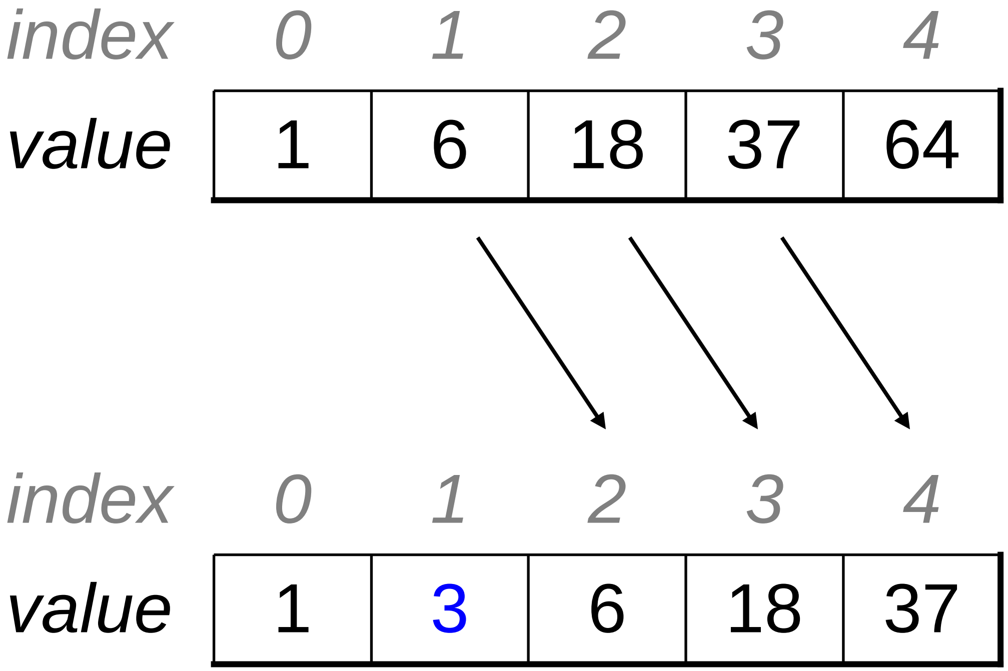
```
public static double average(int[] numbers) {
    int sum = 0;
    for (int i = 0; i < numbers.length; i++) {
        sum += numbers[i];
    }
    return (double) sum / numbers.length;
}

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]);
    }
}
```

Shifting elements in an array

Readings: 7.3 (pg. 408 – 413)

Array insertion

- How would you insert a number into an array of sorted integers? Assume that the largest number gets bumped off the array.



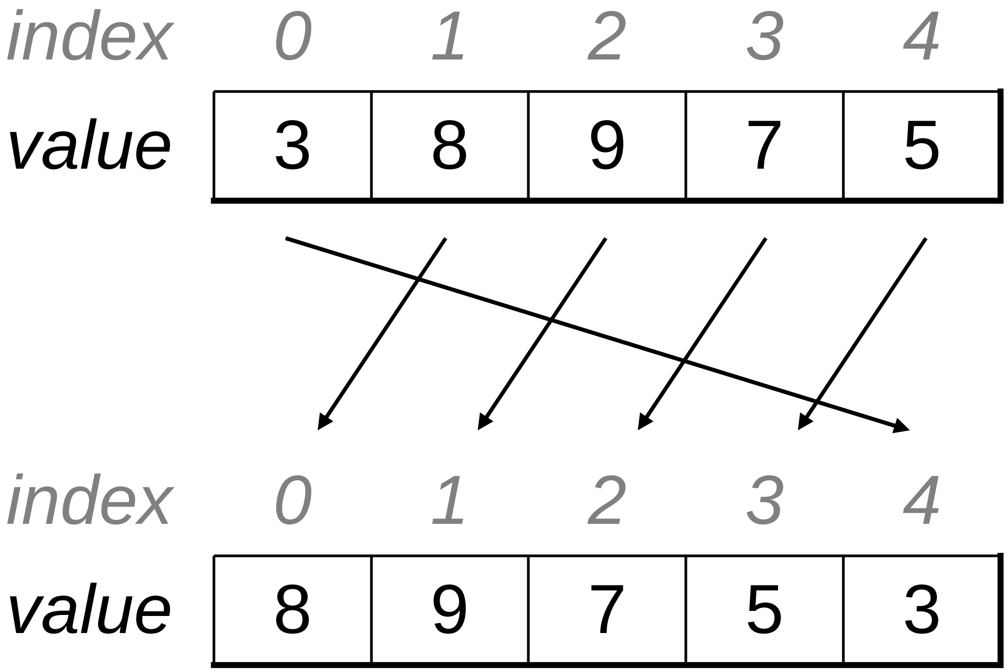
index	*0*	*1*	*2*	*3*	*4*
value	1	6	18	37	64

index	*0*	*1*	*2*	*3*	*4*
value	1	3	6	18	37

Array insertion: Solution

```
public static void insertInOrder(int[] array, int num) {
    int insertionIndex = findInsertionIndex(array, num);
    if (insertionIndex < array.length) {
        for (int i = array.length - 1; i >= insertionIndex + 1; i--) {
             array[i] = array[i-1];
        }
        array[insertionIndex] = num;
    }
}

public static int findInsertionIndex(int[] array, int num) {
    for (int i = 0; i < array.length; i++) {
        if (num < array[i]) {
            return i;
        }
    }

    return array.length;
}
```

Rotating elements left



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

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

Rotating elements left: Solution

```
public static void rotateLeft(int[] array) {
    int first = array[0];
    for (int i = 0; i < array.length - 1; i++) {
        array[i] = array[i + 1];
    }
    array[array.length - 1] = first;
}
```

- What assumptions does this code make?

Exercise: Random elements

- Write a method named `printRandomNumbers` that accepts an array of integers as its parameter and a number of numbers to print. The method will print out *n* random elements (without repetition) from the array, where *n* is the second parameter.

Solution: Random elements

```
public static void printRandomNumbers(int[] numbers, int n) {
    Random rand = new Random();

    int numNumbers = numbers.length;
    for (int i = 1; i <= n; i++) {
        int index = rand.nextInt(numNumbers);
        System.out.println(numbers[index]);

        // shift elements to the left
        for (int j = index; j < numNumbers - 1; j++) {
            numbers[j] = numbers[j + 1];
        }

        numNumbers--;
    }
}
```

- What happens to the array after this method finishes?
  - How could you preserve the contents of the array?