Building Java Programs
Chapter 7

Arrays

Can we solve this problem?

- Consider the following program (input underlined):

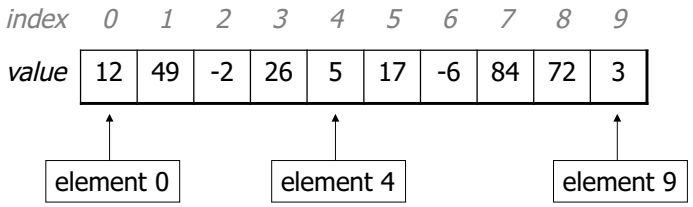
```
How many days' temperatures? 7
Day 1's high temp: 45
Day 2's high temp: 44
Day 3's high temp: 39
Day 4's high temp: 48
Day 5's high temp: 37
Day 6's high temp: 46
Day 7's high temp: 53
Average temp = 44.6
4 days were above average.
```

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];
```

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

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.");
}
```

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

Arrays of other types

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

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

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

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

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
```

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

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
```

x	3

numbers

index	0	1	2	3	4	5	6	7
value	0	4	11	42	99	0	2	0

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;
}
```

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

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[] temps = 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: ");
            temps[i] = console.nextInt();
            sum += temps[i];
        }
        double average = (double) sum / days;

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

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


Quick array initialization

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 elements will be

– The compiler figures out the size by counting the values

"Array mystery" problem

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

Limitations of arrays

- You cannot resize an existing array:

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

- You cannot compare arrays with == or `equals`:

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

- An array does not know how to print itself:

```
int[] a1 = {42, -7, 1, 15};
System.out.println(a1);         // [I@98f8c4]
```


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 (or < 0 if not found)
`copyOf(`**array**`,` **length**`)`	returns a new copy of an array
`equals(`**array1**`,` **array2**`)`	returns `true` if the two arrays contain same elements in the same order
`fill(`**array**`,` **value**`)`	sets every element to the given value
`sort(`**array**`)`	arranges the elements into sorted order
`toString(`**array**`)`	returns a string representing the array, such as `"[10, 30, -25, 17]"`

- Syntax: `Arrays.`**methodName**`(`**parameters**`)`

`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.*;`

Weather question 2

- Modify the weather program to print the following output:

```
How many days' temperatures? 7
Day 1's high temp: 45
Day 2's high temp: 44
Day 3's high temp: 39
Day 4's high temp: 48
Day 5's high temp: 37
Day 6's high temp: 46
Day 7's high temp: 53
Average temp = 44.6
4 days were above average.

Temperatures: [45, 44, 39, 48, 37, 46, 53]
Two coldest days: 37, 39
Two hottest days: 53, 48
```

Weather answer 2

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

public class Weather2 {
    public static void main(String[] args) {
        ...
        int[] temps = new int[days];        // array to store days' temperatures
        ...   (same as Weather program)

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


        System.out.println("Temperatures: " + Arrays.toString(temps));
        Arrays.sort(temps);
        System.out.println("Two coldest days: " + temps[0] + ", " + temps[1]);
        System.out.println("Two hottest days: " + temps[temps.length - 1] +
                           ", " + temps[temps.length - 2]);
    }
}
```


Arrays as parameters

Swapping values

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

    // swap a with b?
    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:

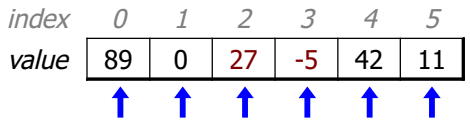
```
int temp = a;
a = b;
b = temp;
```


Array reversal question

- Write code that reverses the elements of an array.
  - For example, if the array initially stores:
    `[11, 42, -5, 27, 0, 89]`
  - Then after your reversal code, it should store:
    `[89, 0, 27, -5, 42, 11]`
    - The code should work for an array of any size.
    - Hint: think about swapping various elements...

Algorithm idea

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



Flawed algorithm

- What's wrong with this code?

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

- The loop goes too far and un-reverses the array! Fixed version:

```
for (int i = 0; i < numbers.length / 2; i++) {
    int temp = numbers[i];
    numbers[i] = numbers[numbers.length - 1 - i];
    numbers[numbers.length - 1 - i] = temp;
}
```


Array reverse question 2

- Turn your array reversal code into a `reverse` method.
  - Accept the array of integers to reverse as a parameter.

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

  - How do we write methods that accept arrays as parameters?
  - Will we need to return the new array contents after reversal?

  ...

Array parameter (declare)

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

- Example:

```
// Returns the average of the given array of numbers.
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;
}
```

  - You don't specify the array's length (but you can examine it).

Array parameter (call)

methodName(**arrayName**);

- Example:

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

  - Notice that you don't write the `[]` when passing the array.

Array return (declare)

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

- Example:

```
// Returns a new array with two copies of each value.
// Example: [1, 4, 0, 7] -> [1, 1, 4, 4, 0, 0, 7, 7]
public static int[] doubled(int[] numbers) {
    int[] result = new int[2 * numbers.length];
    for (int i = 0; i < numbers.length; i++) {
        result[2 * i]     = numbers[i];
        result[2 * i + 1] = numbers[i];
    }
    return result;
}
```


Array return (call)

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

- Example:

```
public class MyProgram {
    public static void main(String[] args) {
        int[] iq = {126, 84, 149, 167, 95};
        int[] twotimes = doubled(iq);
        System.out.println(Arrays.toString(twotimes));
    }
    ...
```

- Output:

```
[126, 126, 84, 84, 149, 149, 167, 167, 95, 95]
```


Reference semantics

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

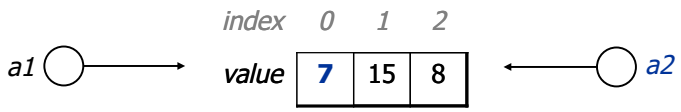
- **value semantics**: Behavior where values are copied when assigned, passed as parameters, or returned.
  - All primitive types in Java use value semantics.
  - When one variable is assigned to another, its value is copied.
  - Modifying the value of one variable does not affect others.

```
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 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.

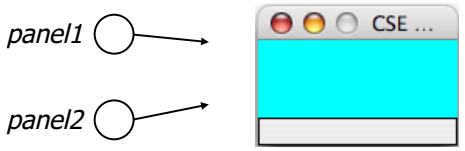
```
int[] a1 = {4, 15, 8};
int[] a2 = a1;          // refer to same array as a1
a2[0] = 7;
System.out.println(Arrays.toString(a1)); // [7, 15, 8]
```



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);
```

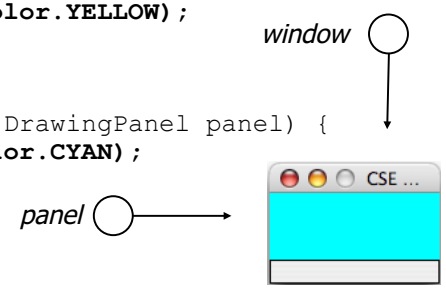


Objects as parameters

- 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);
    ...
}
```



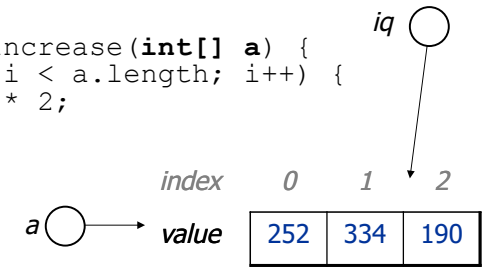
Arrays pass by reference

- Arrays are passed as parameters by *reference.*
  - Changes made in the method are also seen by the caller.

```
public static void main(String[] args) {
    int[] iq = {126, 167, 95};
    increase(iq);
    System.out.println(Arrays.toString(iq));
}
public static void increase(int[] a) {
    for (int i = 0; i < a.length; i++) {
        a[i] = a[i] * 2;
    }
}
```

  - Output:

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



Array reverse question 2

- Turn your array reversal code into a `reverse` method.
  - Accept the array of integers to reverse as a parameter.

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

- Solution:

```
public static void reverse(int[] numbers) {
    for (int i = 0; i < numbers.length / 2; i++) {
        int temp = numbers[i];
        numbers[i] = numbers[numbers.length - 1 - i];
        numbers[numbers.length - 1 - i] = temp;
    }
}
```


Array parameter questions

- Write a method `swap` that accepts an arrays of integers and two indexes and swaps the elements at those indexes.

```
int[] a1 = {12, 34, 56};
swap(a1, 1, 2);
System.out.println(Arrays.toString(a1));  // [12, 56, 34]
```

- Write a method `swapAll` that accepts two arrays of integers as parameters and swaps their entire contents.
  - Assume that the two arrays are the same length.

```
int[] a1 = {12, 34, 56};
int[] a2 = {20, 50, 80};
swapAll(a1, a2);
System.out.println(Arrays.toString(a1));  // [20, 50, 80]
System.out.println(Arrays.toString(a2));  // [12, 34, 56]
```

Array parameter answers

```
// Swaps the values at the given two indexes.
public static void swap(int[] a, int i, int j) {
    int temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}

// Swaps the entire contents of a1 with those of a2.
public static void swapAll(int[] a1, int[] a2) {
    for (int i = 0; i < a1.length; i++) {
        int temp = a1[i];
        a1[i] = a2[i];
        a2[i] = temp;
    }
}
```


Array return question

- Write a method `merge` that accepts two arrays of integers and returns a new array containing all elements of the first array followed by all elements of the second.

```
int[] a1 = {12, 34, 56};
int[] a2 = {7, 8, 9, 10};

int[] a3 = merge(a1, a2);
System.out.println(Arrays.toString(a3));
// [12, 34, 56, 7, 8, 9, 10]
```

- Write a method `merge3` that merges 3 arrays similarly.

```
int[] a1 = {12, 34, 56};
int[] a2 = {7, 8, 9, 10};
int[] a3 = {444, 222, -1};

int[] a4 = merge3(a1, a2, a3);
System.out.println(Arrays.toString(a4));
// [12, 34, 56, 7, 8, 9, 10, 444, 222, -1]
```


Array return answer 1

```
// Returns a new array containing all elements of a1
// followed by all elements of a2.
public static int[] merge(int[] a1, int[] a2) {
    int[] result = new int[a1.length + a2.length];

    for (int i = 0; i < a1.length; i++) {
        result[i] = a1[i];
    }
    for (int i = 0; i < a2.length; i++) {
        result[a1.length + i] = a2[i];
    }

    return result;
}
```


Array return answer 2

```
// Returns a new array containing all elements of a1,a2,a3.
public static int[] merge3(int[] a1, int[] a2, int[] a3) {
    int[] a4 = new int[a1.length + a2.length + a3.length];

    for (int i = 0; i < a1.length; i++) {
        a4[i] = a1[i];
    }
    for (int i = 0; i < a2.length; i++) {
        a4[a1.length + i] = a2[i];
    }
    for (int i = 0; i < a3.length; i++) {
        a4[a1.length + a2.length + i] = a3[i];
    }

    return a4;
}

// Shorter version that calls merge.
public static int[] merge3(int[] a1, int[] a2, int[] a3) {
    return merge(merge(a1, a2), a3);
}
```

Arrays for tallying

A multi-counter problem

- Problem: Write a method `mostFrequentDigit` that returns the digit value that occurs most frequently in a number.
  - Example: The number 669260267 contains:
    one 0, two 2s, four 6es, one 7, and one 9.
    `mostFrequentDigit(669260267)` returns 6.
  - If there is a tie, return the digit with the lower value.
    `mostFrequentDigit(57135203)` returns 3.

A multi-counter problem

- We could declare 10 counter variables ...

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

- But a better solution is to use an array of size 10.
  - The element at index *i* will store the counter for digit value *i*.
  - Example for 669260267:

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

  - How do we build such an array? And how does it help?

Creating an array of tallies

```
// assume n = 669260267
int[] counts = new int[10];
while (n > 0) {
    // pluck off a digit and add to proper counter
    int digit = n % 10;
    counts[digit]++;
    n = n / 10;
}
```

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

Tally solution

```
// Returns the digit value that occurs most frequently in n.
// Breaks ties by choosing the smaller value.
public static int mostFrequentDigit(int n) {
    int[] counts = new int[10];
    while (n > 0) {
        int digit = n % 10;  // pluck off a digit and tally it
        counts[digit]++;
        n = n / 10;
    }

    // find the most frequently occurring digit
    int bestIndex = 0;
    for (int i = 1; i < counts.length; i++) {
        if (counts[i] > counts[bestIndex]) {
            bestIndex = i;
        }
    }

    return bestIndex;
}
```


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: ****
```


Array histogram answer

```
// Reads a file of test scores and shows a histogram of score distribution.
import java.io.*;
import java.util.*;

public class Histogram {
    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();
            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();
            }
        }
    }
}
```


Section attendance question

- Read a file of section attendance (*see next slide*):

```
yynyyynayayynyyyayanyyyaynayyayyanayyyanyayna
ayyanyyyyayanaayyanayyyananayayaynyayayynynya
yyayaynyyayyanynnyyyayyanayaynannnyyayyayayny
```

- And produce the following output:

```
Section 1
Student points: [20, 17, 19, 16, 13]
Student grades: [100.0, 85.0, 95.0, 80.0, 65.0]

Section 2
Student points: [17, 20, 16, 16, 10]
Student grades: [85.0, 100.0, 80.0, 80.0, 50.0]

Section 3
Student points: [17, 18, 17, 20, 16]
Student grades: [85.0, 90.0, 85.0, 100.0, 80.0]
```

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

Section input file

```
student    123451234512345123451234512345123451234512345
week         1    2    3    4    5    6    7    8    9
section 1  yynyyynayayynyyyayanyyyaynayyayyanayyyanyayna
section 2  ayyanyyyyayanaayyanayyyananayayaynyayayynynya
section 3  yyayaynyyayyanynnyyyayyanayaynannnyyayyayayny
```

- Each line represents a section.
- A line consists of 9 weeks' worth of data.
  - Each week has 5 characters because there are 5 students.
- Within each week, each character represents one student.
  - `a` means the student was absent (+0 points)
  - `n` means they attended but didn't do the problems (+2 points)
  - `y` means they attended and did the problems (+3 points)

Section attendance answer

```
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;
        while (input.hasNextLine()) {
            String line = input.nextLine();      // process one section
            int[] points = new int[5];
            for (int i = 0; i < line.length(); i++) {
                int student = i % 5;
                int earned = 0;
                if (line.charAt(i) == 'y') {     // c == 'y' or 'n' or 'a'
                    earned = 3;
                } else if (line.charAt(i) == 'n') {
                    earned = 2;
                }
                points[student] = Math.min(20, points[student] + earned);
            }

            double[] grades = new double[5];
            for (int i = 0; i < points.length; i++) {
                grades[i] = 100.0 * points[i] / 20.0;
            }

            System.out.println("Section " + section);
            System.out.println("Student points: " + Arrays.toString(points));
            System.out.println("Student grades: " + Arrays.toString(grades));
            System.out.println();
            section++;
        }
    }
```


Data transformations

- In many problems we transform data between forms.
  - Example: digits → count of each digit → most frequent digit
  - Often each transformation is computed/stored as an array.
  - For structure, a transformation is often put in its own method.

- Sometimes we map between data and array indexes.
  - 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 `'J'` at index 0, count `'X'` at index 1)

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

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 Sections2 {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner input = new Scanner(new File("sections.txt"));
        int section = 1;
        while (input.hasNextLine()) {
            // process one section
            String line = input.nextLine();
            int[] points = countPoints(line);
            double[] grades = computeGrades(points);
            results(section, points, grades);
            section++;
        }
    }

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

    ...
```

Array param/return answer

```
    ...
    // Computes the points earned for each student for a particular section.
    public static int[] countPoints(String line) {
        int[] points = new int[5];
        for (int i = 0; i < line.length(); i++) {
            int student = i % 5;
            int earned = 0;
            if (line.charAt(i) == 'y') {     // c == 'y'  or  c == 'n'
                earned = 3;
            } else if (line.charAt(i) == 'n') {
                earned = 2;
            }
            points[student] = Math.min(20, points[student] + earned);
        }
        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;
    }
}
```