

Building Java Programs

Chapter 7

Lecture 7-2: Arrays as Parameters, Arrays for Tallying

reading: 4.3, 7.6

Why did the programmer quit his job?

Because he didn't get arrays.

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:

<i>index</i>	0	1	2	3	4	5
<i>value</i>	89	0	27	-5	42	11
	↑	↑	↑	↑	↑	↑

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;
```

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[] double(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[] doubled = double(iq);  
        System.out.println(Arrays.toString(doubled));  
    }  
    ...  
}
```

- Output:

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

Reference semantics

reading: 7.3

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

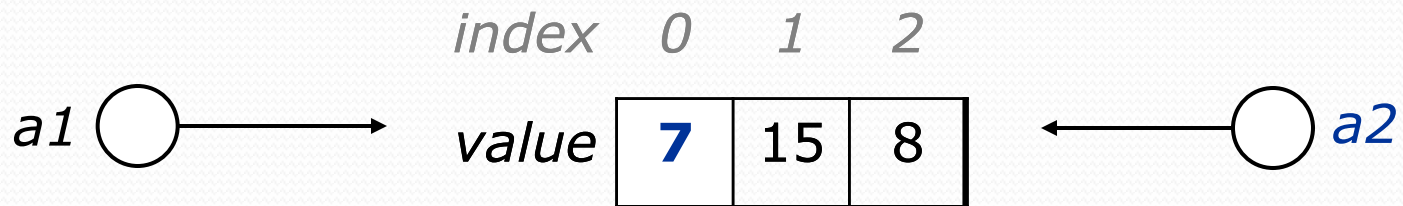
- **value semantics:** Behavior where values are copied when assigned, passed as parameters, or returned.
 - All primitive types in Java use value semantics.
 - When one variable is assigned to another, its value is copied.
 - Modifying the value of one variable does not affect others.

```
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.

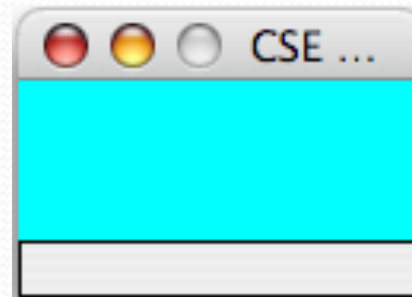
```
int[] a1 = {4, 15, 8};  
int[] a2 = a1;           // refer to same array as a1  
a2[0] = 7;  
System.out.println(Arrays.toString(a1)); // [7, 15, 8]
```



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.

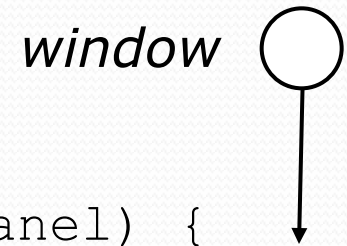
```
DrawingPanel panel1 = new DrawingPanel(80, 50);  
DrawingPanel panel2 = panel1; // same window  
panel2.setBackground(Color.CYAN);
```



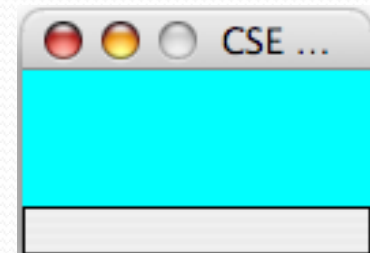
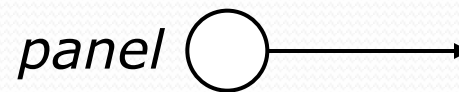
Objects as parameters

- 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);  
}
```



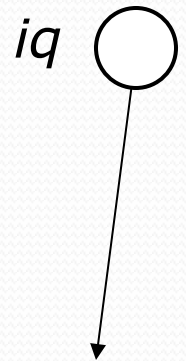
```
public static void example(DrawingPanel panel) {  
    panel.setBackground(Color.CYAN);  
    ...  
}
```



Arrays pass by reference

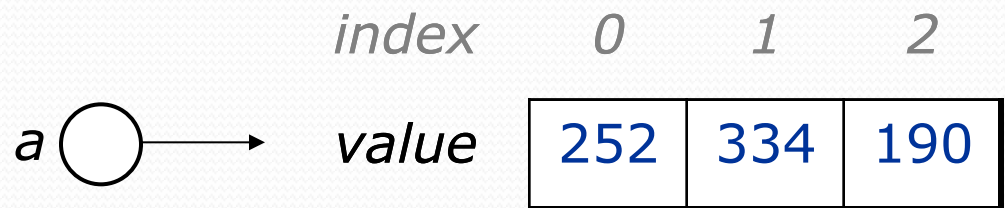
- Arrays are passed as parameters by *reference*.
 - Changes made in the method are also seen by the caller.

```
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);
}
```


Value/Reference Semantics

- Variables of primitive types store values directly:

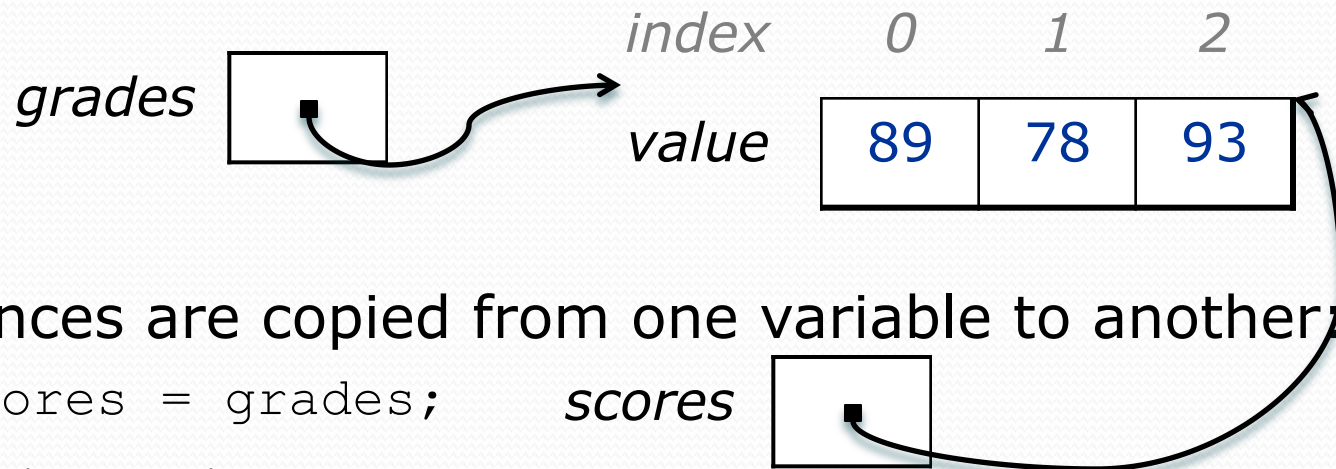
age 20

cats 3

- Values are copied from one variable to another:

`cats = age;` *age* 20 *cats* 20

- Variables of object types store references to memory:



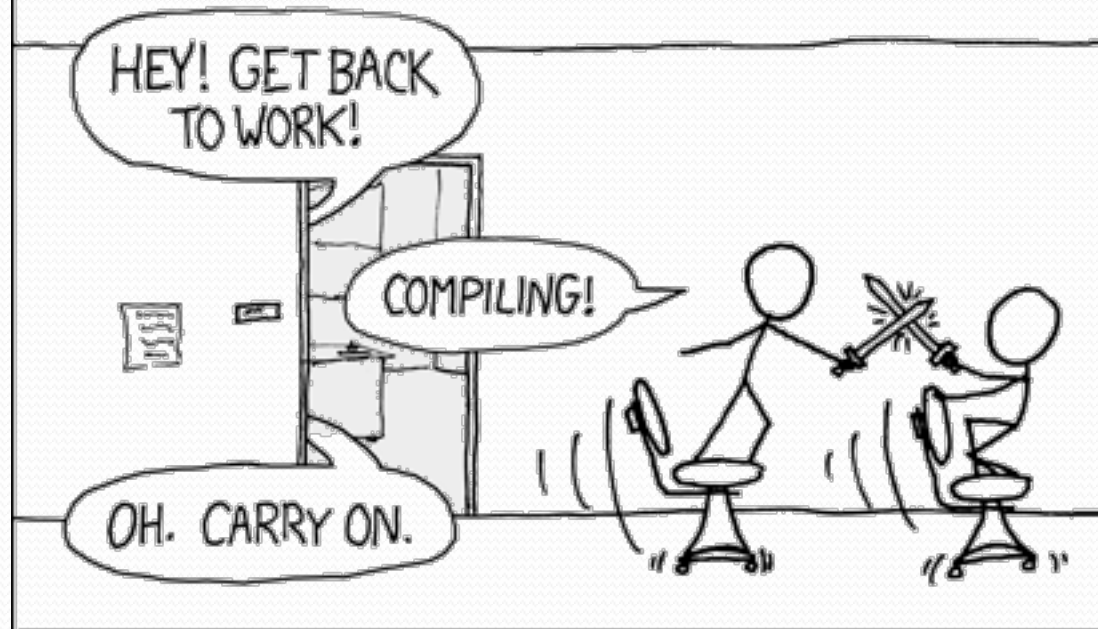
- References are copied from one variable to another:

`scores = grades;` *scores* □

Text processing

reading: 7.2, 4.3

THE #1 PROGRAMMER EXCUSE
FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."



String traversals

- The chars in a String can be accessed using the `charAt` method.
 - accepts an `int` index parameter and returns the `char` at that index

```
String food = "cookie";  
char firstLetter = food.charAt(0);    // 'c'  
System.out.println(firstLetter + " is for " + food);
```

- You can use a `for` loop to print or examine each character.

```
String major = "CSE";  
for (int i = 0; i < major.length(); i++) {    // output:  
    char c = major.charAt(i);                // C  
    System.out.println(c);                    // S  
}                                              // E
```

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:

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	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

<i>value</i>	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;  
}
```


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, 16, 17, 14, 11]  
Student grades: [100.0, 80.0, 85.0, 70.0, 55.0]
```

```
Section 2  
Student points: [16, 19, 14, 14, 8]  
Student grades: [80.0, 95.0, 70.0, 70.0, 40.0]
```

```
Section 3  
Student points: [16, 15, 16, 18, 14]  
Student grades: [80.0, 75.0, 80.0, 90.0, 70.0]
```

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

Section input file

student		12345	12345	12345	12345	12345	12345	12345	12345	12345	12345																																																																																																																																																																																														
week		1	2	3	4	5	6	7	8	9																																																																																																																																																																																															
section 1		<table border="1"> <tr> <td>y</td><td>y</td><td>n</td><td>y</td><td>y</td><td>n</td><td>a</td><td>y</td><td>a</td><td>y</td><td>n</td><td>y</td><td>y</td><td>y</td><td>a</td><td>y</td><td>a</td><td>n</td><td>y</td><td>y</td><td>y</td><td>a</td><td>y</td><td>n</td><td>a</td><td>y</td><td>y</td><td>a</td><td>y</td><td>y</td><td>a</td><td>n</td><td>a</td><td>y</td><td>y</td><td>y</td><td>a</td><td>n</td><td>y</td><td>a</td><td>y</td><td>n</td><td>a</td> </tr> </table>										y	y	n	y	y	n	a	y	a	y	n	y	y	y	a	y	a	n	y	y	y	a	y	n	a	y	y	a	y	y	a	n	a	y	y	y	a	n	y	a	y	n	a																																																																																																																																																			
y	y	n	y	y	n	a	y	a	y	n	y	y	y	a	y	a	n	y	y	y	a	y	n	a	y	y	a	y	y	a	n	a	y	y	y	a	n	y	a	y	n	a																																																																																																																																																															
section 2		a										y										y										y										a										a										y										a										y										y										n										y										n										y										a																																																											
section 3		y										y										a										y										n										y										y										a										y										n										n										n										y										y										a										y										a										y										n										y									

- 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 (+1 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 = 1;
                }
                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 \rightarrow count of each digit \rightarrow 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;
}
```