Building Java Programs

Chapter 7
Lecture 7-3: Arrays for Tallying; Text Processing
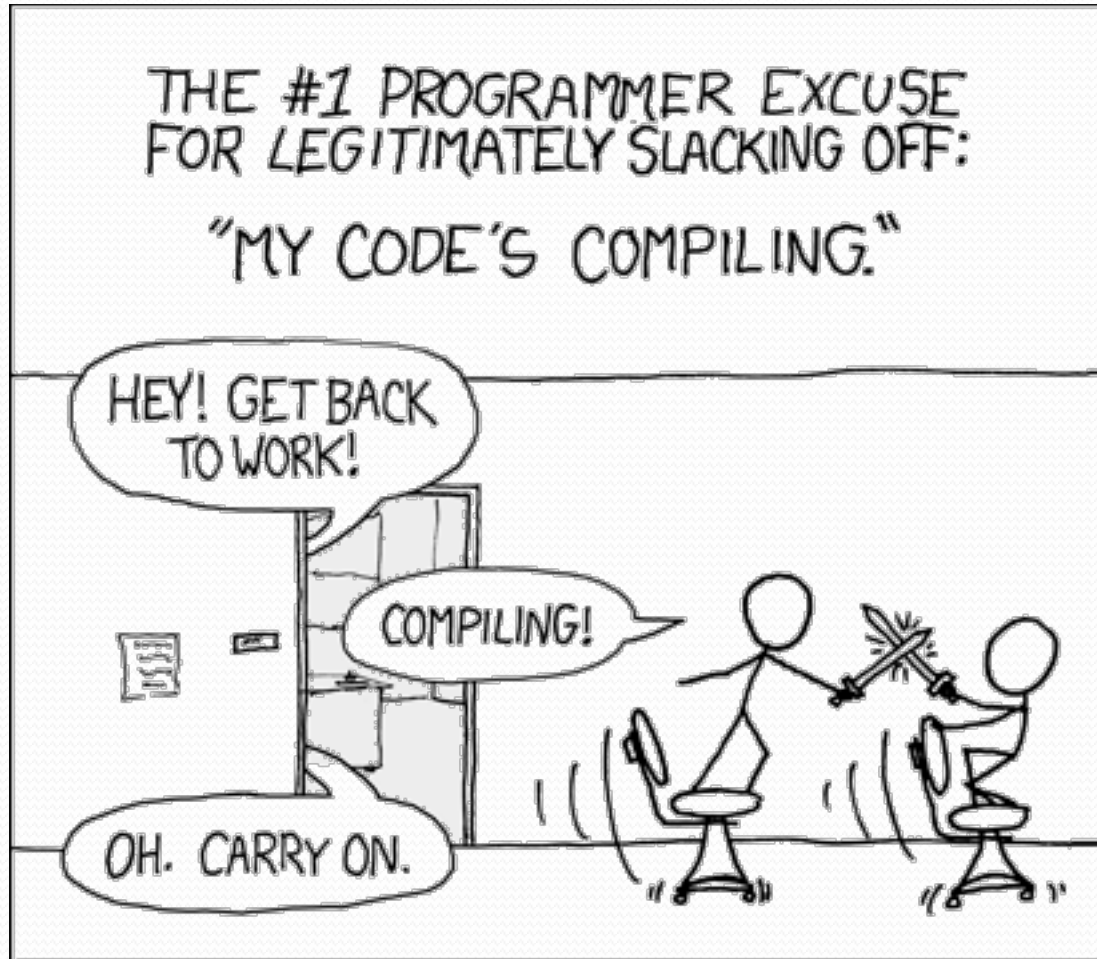
reading: 4.3, 7.6

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

```java
// 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
ayyanyyyayanaayyanayyyananayayaynyayayynynya
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		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 (+1 points)
  - `y` means they attended and did the problems   (+3 points)

8

Section attendance answer

```java
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++;
        }
    }
}
```

9

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 ith 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

```java
// 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;
}
}
```