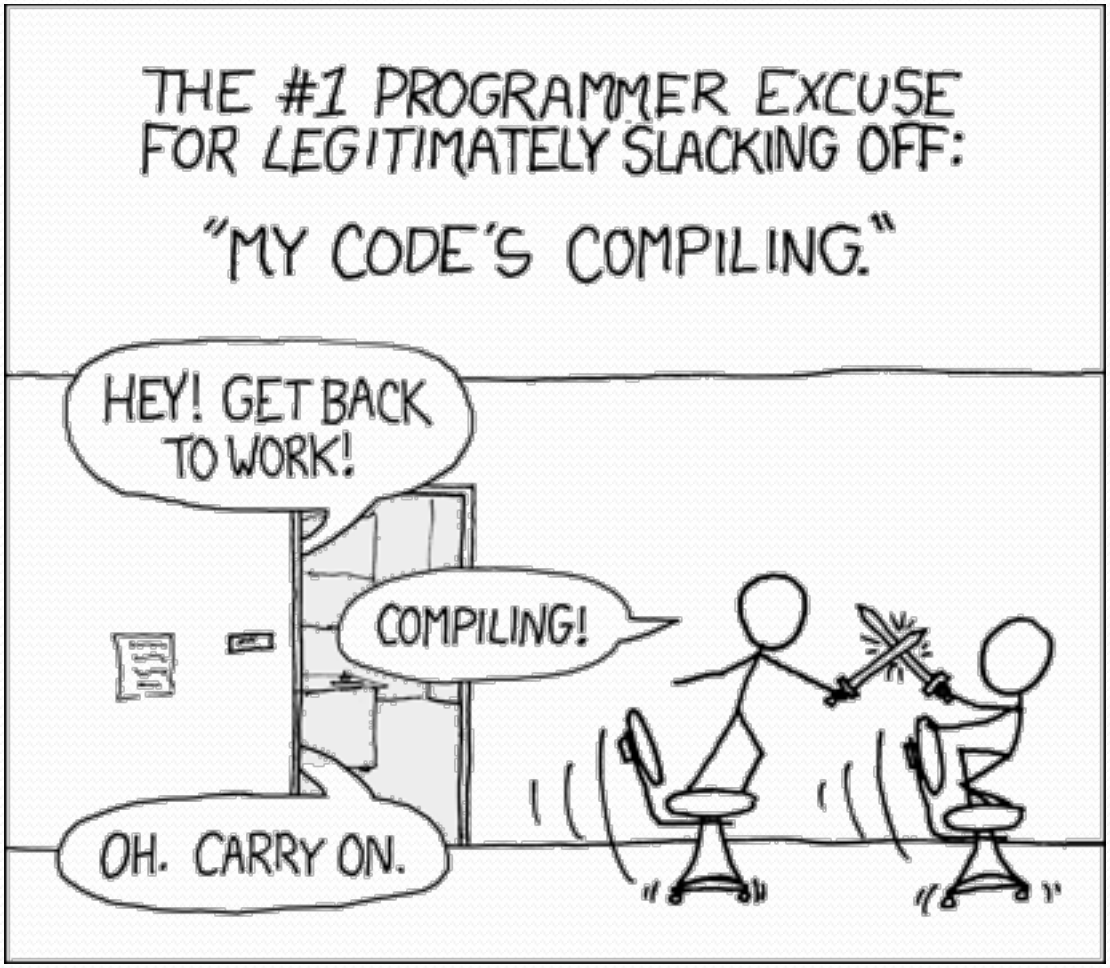
Building Java Programs

Chapter 7

Lecture 7-3: Arrays for Tallying; Text Processing

reading: 4.3, 7.6



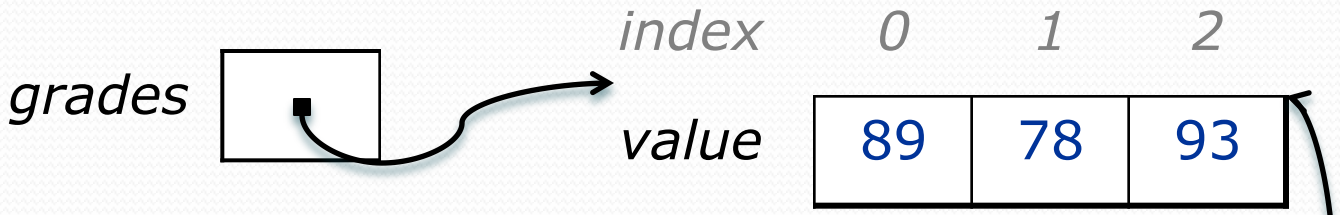
Value/Reference Semantics

- Variables of primitive types store values directly:

- Values are copied from one variable to another:

```
cats = age;
```

- Variables of object types store references to memory:



- References are copied from one variable to another:

```
scores = grades;
```

Text processing

reading: 7.2, 4.3

String traversals

- The `char`s 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.

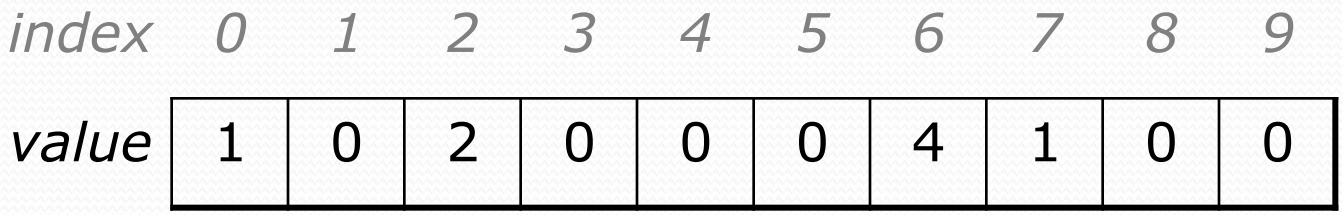
    `mostFrequentDigit(57135203)` returns 3.

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:



- 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;
}
```

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      12345123451234512345123451234512345123451234512345
week           1    2    3    4    5    6    7    8    9
section 1    yynyyynayayynyyyayanyyyaynayyayyanayyyanyayna
section 2    ayyanyyyyayanaayyyanayyyananayayaynyayayynynya
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)

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 → 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;
    }
}
```

File output

reading: 6.4 - 6.5

Output to files

- **`PrintStream`**: An object in the `java.io` package that lets you print output to a destination such as a file.
  - Any methods you have used on `System.out` (such as `print`, `println`) will work on a `PrintStream`.

- Syntax:

```
PrintStream name = new PrintStream(new File("file name"));
```

Example:

```
PrintStream output = new PrintStream(new File("out.txt"));
output.println("Hello, file!");
output.println("This is a second line of output.");
```

Details about `PrintStream`

```
PrintStream name = new PrintStream(new File("file name"));
```

- If the given file does not exist, it is created.
- If the given file already exists, it is overwritten.

- The output you print appears in a file, not on the console. You will have to open the file with an editor to see it.

- Do not open the same file for both reading (`Scanner`) and writing (`PrintStream`) at the same time.
  - You will overwrite your input file with an empty file (0 bytes).

System.out and PrintStream

- The console output object, `System.out`, is a `PrintStream`.

```
PrintStream out1 = System.out;
PrintStream out2 = new PrintStream(new File("data.txt"));
out1.println("Hello, console!");   // goes to console
out2.println("Hello, file!");      // goes to file
```

  - A reference to it can be stored in a `PrintStream` variable.
    - Printing to that variable causes console output to appear.
  - You can pass `System.out` as a parameter to a method expecting a `PrintStream`.
    - Allows methods that can send output to the console or a file.

PrintStream question

- Modify our previous Sections program to use a `PrintStream` to output to the file `sections_out.txt`.

```
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]
```

PrintStream answer

```
// Section attendance program
// This version uses a PrintStream for output.

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"));
        PrintStream out = new PrintStream(new File("sections_out.txt"));
        while (input.hasNextLine()) {   // process one section
            String line = input.nextLine();
            int[] attended = countAttended(line);
            int[] points = computePoints(attended);
            double[] grades = computeGrades(points);
            results(attended, points, grades, out);
        }
    }

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

Prompting for a file name

- We can ask the user to tell us the file to read.
  - The file name might have spaces; use `nextLine()`, not `next()`

```
// prompt for input file name
Scanner console = new Scanner(System.in);
System.out.print("Type a file name to use: ");
String filename = console.nextLine();
Scanner input = new Scanner(new File(filename));
```

- What if the user types a file name that does not exist?

Fixing file-not-found issues

- `File` objects have an `exists` method we can use:

```
Scanner console = new Scanner(System.in);
System.out.print("Type a file name to use: ");
String filename = console.nextLine();
File file = new File(filename);

if (!file.exists()) {
    // try a second time
    System.out.print("Try again: ");
    String filename = console.nextLine();
    file = new File(filename);
}
Scanner input = new Scanner(file);  // open the file
```

Output:

```
Type a file name to use: hourz.text
Try again: hours.txt
```