

CSC 221: Computer Programming I

Fall 2009

ArrayLists and arrays

- example: letter frequencies
- autoboxing/unboxing
- ArrayLists vs. arrays
- example: word frequencies
- parallel lists

1

Letter frequency

one common tool for identifying the author of an unknown work is letter frequency

- i.e., count how frequently each of the letters is used in the work
- analysis has shown that an author will tend to have a consistent pattern of letter usage

will need 26 counters, one for each letter

- traverse each word and add to the corresponding counter for each character
- having a separate variable for each counter is not feasible
- instead have an ArrayList of 26 counters
 - `this.counts.get(0)` is the counter for 'a'
 - `this.counts.get(1)` is the counter for 'b'
 - ...
 - `this.counts.get(25)` is the counter for 'z'

letter frequencies from the
Gettysburg address

a:	93 (9.2%)
b:	12 (1.2%)
c:	28 (2.8%)
d:	49 (4.9%)
e:	150 (14.9%)
f:	21 (2.1%)
g:	23 (2.3%)
h:	65 (6.4%)
i:	59 (5.8%)
j:	0 (0.0%)
k:	2 (0.2%)
l:	39 (3.9%)
m:	14 (1.4%)
n:	71 (7.0%)
o:	81 (8.0%)
p:	15 (1.5%)
q:	1 (0.1%)
r:	70 (6.9%)
s:	36 (3.6%)
t:	109 (10.8%)
u:	15 (1.5%)
v:	20 (2.0%)
w:	26 (2.6%)
x:	0 (0.0%)
y:	10 (1.0%)
z:	0 (0.0%)

2

Letter frequency example

initially, have ArrayList of 26 zeros:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

after processing "Fourscore" :

0	0	1	0	1	1	0	0	0	0	0	0	0	0	2	0	0	2	1	0	1	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

after processing the entire Gettysburg address

93	12	28	49	150	21	23	65	59	0	2	39	14	71	81	15	1	70	36	109	15	20	0	0	10	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

3

Autoboxing & unboxing

natural assumption: will store the frequency counts in an ArrayList of ints

```
private ArrayList<int> counts; // SORRY, WON'T WORK!
```

- unfortunately, ArrayLists can only store object types (i.e., no primitives)
- fortunately, there exists a class named `Integer` that encapsulates an `int` value

```
private ArrayList<Integer> counts;
```

- the Java compiler will automatically
 - convert an `int` value into an `Integer` object when you want to store it in an `ArrayList` (called *autoboxing*)
 - convert an `Integer` value back into an `int` when need to apply an arithmetic operation on it (called *unboxing*)

BE CAREFUL: Java will not unbox an `Integer` for comparison

```
if (this.counts.get(0) == this.counts.get(1)) {  
    ...  
}
```

== will test to see if they are the same Integer objects

4

LetterFreq1 design

```
public class LetterFreq1 {
    private ArrayList<Integer> counts;
    private int numLetters;

    public LetterFreq1(String fileName) throws java.io.FileNotFoundException {
        INITIALIZE this.counts AND this.numLetters

        FOR EACH WORD IN THE FILE
        FOR EACH CHARACTER IN THE WORD
        IF THE CHARACTER IS A LETTER
            DETERMINE ITS POSITION IN THE ALPHABET
            INCREMENT THE CORRESPONDING COUNT IN this.counts
            INCREMENT this.numLetters
        }

    public int getCount(char ch) {
        IF ch IS A LETTER
            DETERMINE ITS POSITION IN THE ALPHABET
            ACCESS & RETURN THE CORRESPONDING COUNT IN this.counts
        OTHERWISE
            RETURN 0
    }

    public double getPercentage(char ch) {
        IF ch IS A LETTER
            DETERMINE ITS POSITION IN THE ALPHABET
            ACCESS THE CORRESPONDING COUNT IN this.counts
            CALCULATE & RETURN THE PERCENTAGE
        OTHERWISE
            RETURN 0.0
    }

    public void showCounts() {
        FOR EACH LETTER IN THE ALPHABET
            DISPLAY THE LETTER, ITS COUNT & PERCENTAGE
    }
}
```

5

LetterFreq1 implementation

```
import java.util.ArrayList;
import java.util.Scanner;
import java.io.File;

public class LetterFreq1 {
    private static final String LETTERS = "abcdefghijklmnopqrstuvwxyz";
    private ArrayList<Integer> counts;
    private int numLetters;

    public LetterFreq1(String fileName) throws java.io.FileNotFoundException {
        this.counts = new ArrayList<Integer>();
        for (int i = 0; i < LetterFreq1.LETTERS.length(); i++) {
            this.counts.add(0);
        }
        this.numLetters = 0;

        Scanner infile = new Scanner(new File(fileName));
        while (infile.hasNext()) {
            String nextWord = infile.next();

            for (int c = 0; c < nextWord.length(); c++) {
                char ch = nextWord.charAt(c);
                if (Character.isLetter(ch)) {
                    int index = LetterFreq1.LETTERS.indexOf(Character.toLowerCase(ch));
                    this.counts.set(index, this.counts.get(index)+1);
                    this.numLetters++;
                }
            }
        }
    }
}
```

will use a constant to store the alphabet

initialize the letter counts

for each word, process each letter ...

... get letter's index, increment its count

6

LetterFreq1 implementation (cont.)

```

    . . .

    public int getCount(char ch) {
        if (Character.isLetter(ch)) {
            int index = LetterFreq1.LETTERS.indexOf(Character.toLowerCase(ch));
            return this.counts.get(index);
        }
        else {
            return 0;
        }
    }

    public double getPercentage(char ch) {
        if (Character.isLetter(ch) && this.numLetters > 0) {
            int index = LetterFreq1.LETTERS.indexOf(Character.toLowerCase(ch));
            return Math.round(1000.0*this.counts.get(index)/this.numLetters)/10.0;
        }
        else {
            return 0.0;
        }
    }

    public void showCounts() {
        for (int i = 0; i < LetterFreq1.LETTERS.length(); i++) {
            char ch = LetterFreq1.LETTERS.charAt(i);
            System.out.println(ch + ": " + this.getCount(ch) + "\t(" +
                this.getPercentage(ch) + "%)");
        }
    }
}

```

if it is a letter,
access & return
its count

if it is a letter,
calculate & return
its percentage

display all letters,
counts and
percentages

7

Interesting comparisons

letter frequencies from the Gettysburg address	letter frequencies from Alice in Wonderland	letter frequencies from Theory of Relativity
a: 93 (9.2%)	a: 8791 (8.2%)	a: 10936 (7.6%)
b: 12 (1.2%)	b: 1475 (1.4%)	b: 1956 (1.4%)
c: 28 (2.8%)	c: 2398 (2.2%)	c: 5272 (3.7%)
d: 49 (4.9%)	d: 4930 (4.6%)	d: 4392 (3.1%)
e: 150 (14.9%)	e: 13572 (12.6%)	e: 18579 (12.9%)
f: 21 (2.1%)	f: 2000 (1.9%)	f: 4228 (2.9%)
g: 23 (2.3%)	g: 2531 (2.4%)	g: 2114 (1.5%)
h: 65 (6.4%)	h: 7373 (6.8%)	h: 7607 (5.3%)
i: 59 (5.8%)	i: 7510 (7.0%)	i: 11937 (8.3%)
j: 0 (0.0%)	j: 146 (0.1%)	j: 106 (0.1%)
k: 2 (0.2%)	k: 1158 (1.1%)	k: 568 (0.4%)
l: 39 (3.9%)	l: 4713 (4.4%)	l: 5697 (4.0%)
m: 14 (1.4%)	m: 2104 (2.0%)	m: 3253 (2.3%)
n: 71 (7.0%)	n: 7013 (6.5%)	n: 9983 (6.9%)
o: 81 (8.0%)	o: 8145 (7.6%)	o: 11181 (7.8%)
p: 15 (1.5%)	p: 1524 (1.4%)	p: 2678 (1.9%)
q: 1 (0.1%)	q: 209 (0.2%)	q: 344 (0.2%)
r: 70 (6.9%)	r: 5437 (5.0%)	r: 8337 (5.8%)
s: 36 (3.6%)	s: 6500 (6.0%)	s: 8982 (6.2%)
t: 109 (10.8%)	t: 10686 (9.9%)	t: 15042 (10.5%)
u: 15 (1.5%)	u: 3465 (3.2%)	u: 3394 (2.4%)
v: 20 (2.0%)	v: 846 (0.8%)	v: 1737 (1.2%)
w: 26 (2.6%)	w: 2675 (2.5%)	w: 2506 (1.7%)
x: 0 (0.0%)	x: 148 (0.1%)	x: 537 (0.4%)
y: 10 (1.0%)	y: 2262 (2.1%)	y: 2446 (1.7%)
z: 0 (0.0%)	z: 78 (0.1%)	z: 115 (0.1%)

8

ArrayLists and arrays

ArrayList enables storing a collection of objects under one name

- can easily access and update items using `get` and `set`
- can easily `add` and `remove` items, and shifting occurs automatically
- can pass the collection to a method as a single object

ArrayList is built on top of a more fundamental Java data structure: the *array*

- an array is a *contiguous, homogeneous* collection of items, accessible via an index
- arrays are much less flexible than ArrayLists
 - e.g., the size of an array is fixed at creation, so you can't add items indefinitely*
 - when you add/remove from the middle, it is up to you to shift items*

so, why use arrays?

1st answer: DON'T! when possible, take advantage of ArrayList's flexibility

2nd answer: arrays can store primitives directly, without autoboxing/unboxing

3rd answer: initializing/accessing an array can be easier in some circumstances

9

Arrays

to declare an array, designate the type of value stored followed by `[]`

```
String[] words;                int[] counters;
```

to create an array, must use `new` (an array is an object)

- specify the type and size inside brackets following `new`

```
words = new String[100];      counters = new int[26];
```

- or, if you know what the initial contents of the array should be, use shorthand:

```
int[] years = {2001, 2002, 2003, 2004, 2005};
```

to access or assign an item in an array, use brackets with the desired index

- similar to the `get` and `set` methods of ArrayList

```
String str = word[0];          // note: index starts at 0
                                // (similar to ArrayLists)
for (int i = 0; i < 26, i++) {
    counters[i] = 0;
}
```

10

LetterFreq2 implementation

```
import java.util.Scanner;
import java.io.File;

public class LetterFreq2 {
    private static final String LETTERS = "abcdefghijklmnopqrstuvwxyz";
    private int[] counts;
    private int numLetters;

    public LetterFreq2(String fileName) throws java.io.FileNotFoundException {
        this.counts = new int[LetterFreq1.LETTERS.length()];
        for (int i = 0; i < LetterFreq1.LETTERS.length(); i++) {
            this.counts[i] = 0;
        }
        this.numLetters = 0;

        Scanner infile = new Scanner(new File(fileName));
        while (infile.hasNext()) {
            String nextWord = infile.next();

            for (int c = 0; c < nextWord.length(); c++) {
                char ch = nextWord.charAt(c);
                if (Character.isLetter(ch)) {
                    int index = LetterFreq1.LETTERS.indexOf(Character.toLowerCase(ch));
                    this.counts[index]++;
                    this.numLetters++;
                }
            }
        }
    }
}
```

could instead
make the field an
array

initialize array to
desired size

access/assign an
entry using []

increment is
simpler (no need
to get then set)

11

LetterFreq2 implementation (cont.)

```
. . .

public int getCount(char ch) {
    if (Character.isLetter(ch)) {
        int index = LetterFreq1.LETTERS.indexOf(Character.toLowerCase(ch));
        return this.counts[index];
    }
    else {
        return 0;
    }
}

public double getPercentage(char ch) {
    if (Character.isLetter(ch) && this.numLetters > 0) {
        int index = LetterFreq1.LETTERS.indexOf(Character.toLowerCase(ch));
        return Math.round(1000.0*this.counts[index]/this.numLetters)/10.0;
    }
    else {
        return 0.0;
    }
}

public void showCounts() {
    for (int i = 0; i < LetterFreq1.LETTERS.length(); i++) {
        char ch = LetterFreq1.LETTERS.charAt(i);
        System.out.println(ch + ": " + this.getCount(ch) + "\t(" +
            this.getPercentage(ch) + "%)");
    }
}
```

other method
essentially the
same (array
access uses []
instead of the get
method for
ArrayLists)

12

Why arrays?

general rule: ArrayLists are better, more abstract, arrays – USE THEM!

- they provide the basic array structure with many useful methods provided for free

```
get, set, add, size, contains, indexOf, remove, ...
```

- plus, the size of an ArrayList automatically adjusts as you add/remove items

when *might* you want to use an array?

- if the size of the list will never change and you merely want to access/assign items, then the advantages of arrays may be sufficient to warrant their use
 - ✓ if the initial contents are known, they can be assigned when the array is created

```
String[] answers = { "yes", "no", "maybe" };
```

- ✓ the [] notation allows for both access and assignment (instead of get & set)

```
int[] counts = new int[11];  
...  
counts[die1.roll() + die2.roll()]++;
```

- ✓ you can store primitive types directly, so no autoboxing/unboxing

13

Related example: word frequencies

recall:

- Dictionary class reads in words from a file, stores unique words
- LetterFreq class reads in words from a file, stores letter frequencies

could combine ideas from these two classes to perform a different task: keep frequency counts for individual words

```
fourscore: 1  
and: 5  
seven: 1  
years: 1  
ago: 1  
our: 2  
fathers: 1  
brought: 1  
forth: 1  
on: 2  
this: 3  
.  
.  
.
```

as in Dictionary, will need an ArrayList to store the words as they are read in

as in LetterFreq, will need to keep a counter for each word that is stored:

- first time read, create new entry with count of 1
- each subsequent time, just increment the count

14

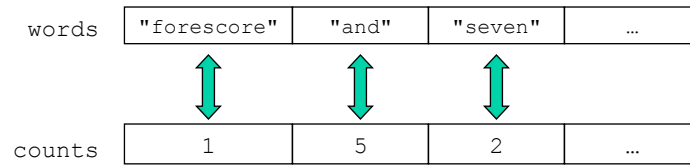
Parallel lists

one approach to storing the words and their associated counts is with *parallel lists*

- related values are stored in separate lists, with corresponding values at corresponding indices

e.g., words stores the words

counts stores the frequency counts for those words



note: `counts.get(i)` is the number of times that `words.get(i)` appears in the file

15

WordFreq implementation

```
public class WordFreq {
    private ArrayList<String> words;
    private ArrayList<Integer> counts;

    public WordFreq() {
        this.words = new ArrayList<String>();
        this.counts = new ArrayList<Integer>();
    }

    public WordFreq(String fileName) throws java.io.FileNotFoundException {
        this.words = new ArrayList<String>();
        this.counts = new ArrayList<Integer>();

        Scanner infile = new Scanner(new File(fileName));
        while (infile.hasNext()) {
            String nextWord = infile.next();
            this.addWord(nextWord);
        }
    }

    public void addWord(String newWord) {
        int index = words.indexOf(newWord);
        if (index == -1) {
            this.words.add(newWord);
            this.counts.add(1);
        }
        else {
            this.counts.set(index, this.counts.get(index)+1);
        }
    }
    . . .
}
```

must use an ArrayList for the counts since the size will grow

initialize both lists to be empty

call addWord to add each word as it is read in

if not already stored, add the word with count of 1, else increment its count

16

WordFreq implementation (cont.)

```
...  
  
public int wordCount(String desiredWord) {  
    int index = this.words.indexOf(desiredWord);  
    if (index == -1) {  
        return 0;  
    }  
    else {  
        return this.counts.get(index);  
    }  
}  
  
public int numWords() {  
    return this.words.size();  
}  
  
public void showAll() {  
    for (int i = 0; i < this.words.size(); i++) {  
        System.out.println(this.words.get(i) + ": " + this.counts.get(i));  
    }  
}
```

wordCount looks up the word, and if found returns its count

numWords simply returns the size of the lists

showAll traverses the lists and shows each word & count

17