EECS1012
MOBILE COMPUTING

COLLECTIONS

(SLIDES ADAPTED FROM PROF.H. ROUMANI)

PROF. Y. LESPÉRANCE
Dept. of Electrical Engineering & Computer Science

1

ABOUT COLLECTIONS

- Problem: naming a bunch of things
  Cannot use variables … will run out of names!
- Solutions
  Traditional approach: name + index = array
  Modern approach: object with API = list, set, map
- Comparison
  Arrays have no API and suffer from fixed allocation
  The modern collection framework has a rich API
- But we occasionally use arrays
  For compatibility with low-level API (e.g. split and args)

2

ARRAYS (SEE SEC. L.2.1.E)

- Represent a collection of entities of the same type
- Declaration: *type*[] *name*; e.g. int[] bag;
- Instantiation: *new type*[*size*], e.g.

  bag = new int[100];
- Refer to elements by name[index] , e.g.
  bag[0] = 123; bag[1] = bag[0] + 5;

3

ARRAYS (SEE SEC. L.2.1.E)

- *name.length* represents the array's length
- Indices go from 0 to length – 1
- Multidimensional arrays can also be used

4

Example 1

If we pick an integer in [1,1M] randomly, how likely is it to get one whose digit sum is divisible by 7?

Compute the probability by sampling 10% of those integers and store the sample in a collection.

1. Use Arrays
   See SumDiv7_array.java
2. Use Collections
   See SumDiv7_coll.java

5

Java Collection Framework

- List vs Set vs Map
  List: may contain duplicates and elements are ordered. Set: no duplicates and no order. Map: key-value pairs, key unique.
- The Interfaces (aka Abstract Data Types)
  List<E>, Set<E>, and Map<K,V> (use generics)
- The Classes (aka Implementations)
  List: ArrayList and LinkedyList; Set: HashSet and TreeSet
  Map: HashMap and TreeMap
- Common APIs
  size(), clear(), iterator(), toString()
  Methods to insert, delete, and search → CRUD

6

The Collections API

Basic
size()
clear()
iterator()

List/Set
add(E)
remove(E)
contains(E)

List Only
add(int, E)
remove(int)
get(int)

Map
put(K,V), get(K), keySet()
containsKey(K)
containsValue(V)

Other API
The enhanced for loop
Collections.sort(List)
Arrays… *see API*

7

Notes on Collections

- add(E e) on a set returns false if e is already in it (for a list always returns true)
- remove(E e) returns true iff e is found in the set or list; for a list removes only first occurrence
- Collections.sort(List <E> l) rearranges l to make it sorted (according to natural order)
- Arrays.asList(E[] a) returns a List representation of array a

8

Notes on Collections

- Traversing a List<E> bag i.e. going through all of its elements one by one, is a common operation:

```
for (E e: bag) {
    System.out.println(e);
}
```

- Similarly for sets
- For lists, can also do an indexed traversal:

```
for (int i = 0; i < bag.size(); i++) {
    E e = bag.get(i); System.out.println(e);
}
```

9

Example 2

Given a list, determine whether it contains duplicate elements.

Can be done in 3 ways:

1. Sort the list and then traverse it to check for adjacent duplicates
2. Create a set and then try to add each list element to it checking if add succeeds
3. Traverse the list, and for each element traverse the list again to see if it occurs elsewhere

10

Example 2 – Sorting-Based Solution

```
Collections.sort(bag);
boolean distinct = true;
for (int i = 0; i < bag.size() - 1; i++) {
    distinct = distinct && !bag.get(i).equals(bag.get(i+1));
}
```

- Can also exit as soon as a duplicate is found:

```
for (int i = 0; i < bag.size() – 1 && distinct; i++) {
    distinct = !bag.get(i).equals(bag.get(i+1));
}
```

11

Example 2 – Set-Making Solution

```
Set<Integer> tmp = new HashSet<Integer>();
boolean distinct = true;
for (int i = 0; i < bag.size() ; i++) {
    distinct = distinct && tmp.add(bag.get(i));
}
```

- Can also exit as soon as a duplicate is found:

```
for (int i = 0; i < bag.size() && distinct; i++) {
    distinct = tmp.add(bag.get(i)) ;
}
```

12

Example 2 – Indexed Traversal-Based Solution

```
boolean distinct = true;
for (int i = 0; i < bag.size(); i++) {
    int x = bag.get(i);
    for (int j = 0; j < bag.size(); j++) {
            int y = bag..get(j);
            distinct = distinct && (i = j || x != y);
    }
}
```

Can also exit as soon as a duplicate is found by adding && distinct to both loop conditions

13

Example 2 – Iterator Traversal-Based Solution

```
Iterator<Integer> outer = bag.iterator();
boolean distinct = true;
while (outer.hasNext() && distinct) {
    Integer x = outer.next();
    Iterator<Integer> inner= bag.iterator();
    while (inner.hasNext() && distinct) {
            Integer y = inner.next();
            distinct = !x.equals(y) || x = y;
    }
}
```

14

Example 3

Given a long sentence, find all its words; the distinct ones (regardless of case); display them; sort them; and then locate the longest and most frequent ones.

A "word" is defined as a sequence of characters terminated by space, punctuation, or end-of-string.

1. Use split with a regex
2. Turn array to a collection
3. Use collection API

See WordSmith.java

15

Iterators

- An iterator is an object that allows you to traverse a collection
- Given a reference bag of type List<E> or Set<E> one can get an iterator for it as follows:

  Iterator<E> itr = bag.iterator();
- To check if there is a next element one calls the boolean method itr.hasNext()
- To obtain the next element (provided there is one) we write

  E e = itr.next();

16

Extended for loop notation

The extended for loop for a reference bag of type List<E> or Set<E>:

```
for (E e: bag) {
    …
}
```

just abbreviates/stands for

```
for (Iterator<E> itr = bag.iterator(); itr.hasNext(); ) {
    E e = itr.next();
    …
}
```

17

Example 4 – Set Intersection

```
public static Set<E> intersect(Set<E> set1, Set<E> set2){
    Set<E> result = new HashSet<E>();
    for(E e : set1) {
        if (set2.contains(e){
            result.add(e);
        }
    }
    return result;
}
```

18

Example 4 – Set Union

```
public static Set<E> union(Set<E> set1, Set<E> set2){
    Set<E> result = new HashSet<E>();
    for(E e :  set1) {
        result.add(e);
    }
    for (E e : set2){
        result.add(e);
    }
    return result;
}
```

19

The Collections API

Basic
size()
clear()
iterator()

List/Set
add(E)
remove(E)
contains(E)

List Only
add(int, E)
remove(int)
get(int)

Map
put(K,V), get(K), keySet()
containsKey(K)
containsValue(V)

Other API
The enhanced for loop
Collections.sort(List)
Arrays… *see API*

20

Maps

- A Map<K,V> is represents a mapping between a set of keys (of type K) and a set of values (of type V)
- Each element in a map is a key-value pair
- The keys must form a set and all be distinct
- There are 2 classes that implement the interface Map<K,V>: TreeMap<K,V> and HashMap<K,V>
- Use map.put(k,v) to add or update a key-value pair
- Use map.get(k) to retrieve the value associated with key k
- Other methods: remove(k,v), containsKey(k), containsValue(v), keySet(), size(), toString(), etc.

21

Map Simple Example

```
Map<String,Integer> m = new HashMap<String,Integer>();
m.put("John",23);
m.put("Mary",22);
m.put("Paul",19);
System.out.println("Mary is" + m.get("Mary"));
m.put("Mary",21);
System.out.println("Mary is" + m.get("Mary"));
```

22

Map Traversal Example

```
Map<String,Integer> m = new HashMap<String,Integer>();
m.put("John",23); m.put("Mary",22); m.put("Paul",19);
for (String k : m.keySet()){
    System.out.println(k + "  is " + m.get(k));
}
```

23

Example 3

Given a long sentence, find all its words; the distinct ones (regardless of case); display them; sort them; and then locate the longest and most frequent ones.

A "word" is defined as a sequence of characters terminated by space, punctuation, or end-of-string.

1. Use split with a regex
2. Turn array to a collection
3. Use collection API

See WordSmith.java

24

Inverting a Map

- Suppose we want the inverse of a Map<K,V> m
- Just taking the set of all pairs value-keys from m does not work because the values may not be unique
- One solution is to create a new map Map<V, List<K>> inv which associates each value v in m to the list of all the keys that m maps to v
- E.g. inverse of the map

  {"John" =23, "Paul" =21, "Mary" =23} would then be {21=[”Paul"], 23= ["John", "Mary"]}

25

Inverting a Map

```
public static Map<Integer, List<String>> invert_all(Map<String, Integer> map){
    Map<Integer, List<String>> result = new  TreeMap<Integer, List<String>>();
    for (String k : map.keySet(){
      int v = map.get(k);
      if (!result.containsKey(v){
            List<String> list = new ArrayList<String>();
            list.add(k); result.put(v,list);
      } else {
            List<String> existing = result.get(v);
            existing.add(k);
      }}
     return result;
}
```

26

Computational Complexity

- Addresses questions such as How much time (or space) does a given algorithm take depending on the size of the input?
- That is if N is the size of the input, and T(N) is the running time of the algorithm for an input of size N, we want to know how T(N) grows as N grows
- E.g. How long does it take to sort a list of size N?
- Running time may depend on the actual input value (e.g. list); usually we simplify by looking at the worst case

27

Big-O Notation

- Big-O notation is used to characterize how fast the running time T(N) of the algorithm grows as the size of the input N grows
- T(N) is O(f(N)) if $T(N) \leq C\, f(N)$ for all $N \geq K$ for some constants C and K
- That is, if T(N) is always less than f(N) multiplied by some constant beyond a given value of N

28

Example: Searching a List

- To find out if an element x is contained in an unsorted List<Integer> list we can do a linear search:

```
int result = -1; boolean found =  false;
for (int i = 0; i < list.size() && !found; i++) {
    int e = list.get(i);
    if (e == x){
        result = i; found = true;
    }
}
```

29

Example: Searching a List

- This is essentially what the contains() method does
- The running time is O(N) where N is the size for the list
- This is because in the worst case we have to check and compare all of the N elements of the list to x

30

Example Searching a Sorted List

- To search a sorted list, there is a much faster algorithm, binary search, which is used by Collections.binarySearch(List<E> l, E x)
- Compare x with the middle element of the list: if it is equal we are done, and if it is less we can eliminate all elements after it, and similarly if it is greater; then repeat with the remaining part of the list
- So after each comparison we cut down the remaining part of the list by half
- So the running time is O(lgN) where N is the size for the list

31

Example: Sorting a List

- To sort a list, there are pretty fast algorithms such as quicksort, which is used by Collections.sort(List<E> l)
- The running time is O(N lgN) where N is the size for the list

32

Running Time of List Methods

ArrayList

- get(int) and add(E) are O(1)
- add(int, E), get(E), contains(E), remove(E), and remove(int) are O(N)

LinkedList

- get(int) is O(N)
- add(0,E) is O(1)

33

Running Time of Set Methods

TreeSet

- add(E) is O(lgN)
- contains(E) is O(lgN)
- remove(E) is O(lgN)

HashSet

- add(E) is O(1)
- contains(E) is O(1)
- remove(E) is O(1)

Similar for TreeMap and HashMap

34