


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)

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`;

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

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 LinkedList; 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	List/Set	List Only
size()	add(E)	add(int, E)
clear()	remove(E)	remove(int)
iterator()	contains(E)	get(int)

Map	Other API
put(K,V), get(K), keySet()	The enhanced for loop
containsKey(K)	Collections.sort(List)
containsValue(V)	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