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;

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

---

**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 \(\rightarrow\) CRUD

---

**The Collections API**

<table>
<thead>
<tr>
<th>Basic</th>
<th>List/Set</th>
<th>List Only</th>
<th>Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>size()</td>
<td>add(E)</td>
<td>add(int, E)</td>
<td>put(K,V), get(K), keySet()</td>
</tr>
<tr>
<td>clear()</td>
<td>remove(E)</td>
<td>remove(int)</td>
<td>containsKey(K)</td>
</tr>
<tr>
<td>iterator()</td>
<td>contains(E)</td>
<td>get(int)</td>
<td>containsValue(V)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other API</th>
</tr>
</thead>
<tbody>
<tr>
<td>The enhanced for loop</td>
</tr>
<tr>
<td>Collections.sort(List)</td>
</tr>
<tr>
<td>Arrays... see API</td>
</tr>
</tbody>
</table>

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

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

Similarly for sets:

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

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.

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));
}
```

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)) ;
}
```
Example 2 – Traversal-Based Solution

```java
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;
    }
}
```

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

Exercises

- Given a list<E>, pick a random element from it.
- Given a set<E>, pick a random element from it.
- Given a map<K,V>, pick a random value from it.
- Given a list<E>, print it sorted or reverse-sorted.
- Given two lists <E>, merge them.
- Given two lists <E>, intersect them.
- Given a map<K,V>, invert it assuming distinct values.
- Given a map<K,V>, invert it using largest value to break ties.
- Given a map<K,V>, invert it into <V, List<K>>

Inheritance in a Nutshell

- Abstract Declaration
  Reserve a spot for a "GTA university student", a "YU student", an "EECS student", ... Expect a parent but accept a child.

- Removes Redundancy
  If classes A and B share 80% of the methods, why not put those methods in parent class C and have A and B extend it.

- Exercises Control
  If classes A and B share 80% of the API, why not put the shared API headers in interface C and have A and B implement it.

- Provides Defaults
  All Java classes extend Object. Android gadgets extend View.