CSE 143

Lecture 11: Sets and Maps

reading: 11.2 - 11.3

SUBSTITUTIONS
THAT MAKE READING THE NEWS MORE FUN:

WITNESSES ➔ THESE DUDES I KNOW
ALLEGEDLY ➔ KINDA PROBABLY
NEW STUDY ➔ TUMBLR POST
REBUILD ➔ AVENGE
SPACE ➔ SPAACE
GOOGLE GLASS ➔ VIRTUAL BOY
SMARTPHONE ➔ POKÉDEX
ELECTRIC ➔ ATOMIC
SENATOR ➔ ELF-LORD
CAR ➔ CAT
ELECTION ➔ EATING CONTEST
CONGRESSIONAL LEADERS ➔ RIVER SPIRITS
HOMELAND SECURITY ➔ HOMESTAR RUNNER
COULD NOT BE REACHED FOR COMMENT ➔ IS GUILTY AND EVERYONE KNOWS IT
The "for each" loop (7.1)

for (type name : collection) {
    statements;
}

- Provides a clean syntax for looping over the elements of a Set, List, array, or other collection

Set<Double> grades = new HashSet<Double>();
...

for (double grade : grades) {
    System.out.println("Student's grade: " + grade);
}

- needed because sets have no indexes; can't get element i
Exercise

- Write a program to count the number of occurrences of each unique word in a large text file (e.g. *Moby Dick*).
  - Allow the user to type a word and report how many times that word appeared in the book.
  - Report all words that appeared in the book at least 500 times, in alphabetical order.

- What collection is appropriate for this problem?
• **map**: Holds a set of unique *keys* and a collection of *values*, where each key is associated with one value.
  • a.k.a. "dictionary", "associative array", "hash"

• basic map operations:
  • **put(key, value)**: Adds a mapping from a key to a value.
  • **get(key)**: Retrieves the value mapped to the key.
  • **remove(key)**: Removes the given key and its mapped value.

```java
myMap.get("Juliet") returns "Capulet"
```
Map implementation

- in Java, maps are represented by `Map` type in `java.util`
- `Map` is implemented by the `HashMap` and `TreeMap` classes
  - `HashMap`: implemented using an array called a "hash table"; extremely fast: $O(1)$; keys are stored in unpredictable order
  - `TreeMap`: implemented as a linked "binary tree" structure; very fast: $O(\log N)$; keys are stored in sorted order
  - `LinkedHashMap`: $O(1)$; keys are stored in order of insertion
- A map requires 2 type params: one for keys, one for values.

```java
// maps from String keys to Integer values
Map<String, Integer> votes = new HashMap<String, Integer>();
```
# Map methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>put(key, value)</code></td>
<td>adds a mapping from the given key to the given value; if the key already exists, replaces its value with the given one</td>
</tr>
<tr>
<td><code>get(key)</code></td>
<td>returns the value mapped to the given key (null if not found)</td>
</tr>
<tr>
<td><code>containsKey(key)</code></td>
<td>returns true if the map contains a mapping for the given key</td>
</tr>
<tr>
<td><code>remove(key)</code></td>
<td>removes any existing mapping for the given key</td>
</tr>
<tr>
<td><code>clear()</code></td>
<td>removes all key/value pairs from the map</td>
</tr>
<tr>
<td><code>size()</code></td>
<td>returns the number of key/value pairs in the map</td>
</tr>
<tr>
<td><code>isEmpty()</code></td>
<td>returns true if the map's size is 0</td>
</tr>
<tr>
<td><code>toString()</code></td>
<td>returns a string such as <code>{a=90, d=60, c=70}</code></td>
</tr>
<tr>
<td><code>keySet()</code></td>
<td>returns a set of all keys in the map</td>
</tr>
<tr>
<td><code>values()</code></td>
<td>returns a collection of all values in the map</td>
</tr>
<tr>
<td><code>putAll(map)</code></td>
<td>adds all key/value pairs from the given map to this map</td>
</tr>
<tr>
<td><code>equals(map)</code></td>
<td>returns true if given map has the same mappings as this one</td>
</tr>
</tbody>
</table>
Using maps

- A map allows you to get from one half of a pair to the other.
  - Remembers one piece of information about every index (key).

```java
//   key        value
put("Suzy", "206-685-2181")
```

- Later, we can supply only the key and get back the related value:
  - Allows us to ask: What is Suzy's phone number?

```java
get("Suzy")
```

```
"206-685-2181"
```
Maps and tallying

- A map can be thought of as a generalization of a tallying array.
- The "index" (key) doesn't have to be an int.
- Count digits: 22092310907.
- Count votes: "MOOOOOOMMMMMO0000001022"

<table>
<thead>
<tr>
<th>key</th>
<th>&quot;M&quot;</th>
<th>&quot;O&quot;</th>
<th>&quot;I&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>16</td>
<td>14</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

// (M)cCain, (O)bama, (I)ndependent
keySet and values

- **keySet** method returns a **Set** of all keys in the map
  - can loop over the keys in a foreach loop
  - can get each key's associated value by calling **get** on the map

```java
Map<String, Integer> ages = new TreeMap<String, Integer>();
ages.put("Marty", 19);
ages.put("Geneva", 2); // ages.keySet() returns Set<String>
ages.put("Vicki", 57);
for (String name : ages.keySet()) {
    int age = ages.get(name); // Geneva -> 2
    System.out.println(name + " -> " + age); // Marty -> 19
    // Vicki -> 57
}
```

- **values** method returns a collection of all values in the map
  - can loop over the values in a foreach loop
  - no easy way to get from a value to its associated key(s)