Building Java Programs

Chapter 11
Sets and Maps

reading: 11.2 - 11.3
Wednesday Notecards

SEE ANYTHING

WATCH TOWER
N+1

SEE ANYTHING

I ♥ CATS & DOGS

W is for Wisdom!

Recursion
Wednesday Notecards

- Believe in the power of Recursion!!

- I use recursion...
- I use trees...
- I use... calculating...
- And oh look... a recursive emulator...
- I especially love programmers who use recursion...
- ...and programmers who use recursion...
- ...and programmers who love recursion...
- ...and programmers who love recursion...
Wednesday Notecards

- Does style matter for exam?
  - Not unless specified on the problem!
- Cheat sheet for exam?
  - Yes
- Is recursion useful outside of these toy examples and writing directory crawlers?
  - Yes!!!! Compilers, Parallelism, Proving Algorithm Correctness
- Would you rather fight 100 duck-sized horses or one horse-sized duck?
- Anonymous feedback for TAs
  - You can send it to me using my anonymous feedback form and I’ll forward it along
Road Map

CS Concepts
- Client/Implementer
- Efficiency
- Recursion
- Regular Expressions
- Grammars
- Sorting
- Backtracking
- Hashing
- Huffman Compression

Java Language
- Exceptions
- Interfaces
- References
- Comparable
- Generics
- Inheritance/Polymorphism
- Abstract Classes

Data Structures
- Lists
- Stacks
- Queues
- Sets
- Maps
- Priority Queues

Java Collections
- Arrays
- ArrayList
- LinkedList
- Stack
- TreeSet / TreeMap
- HashSet / HashMap
- PriorityQueue
Exercise

- Write a program that counts the number of unique words in a large text file (say, *Moby Dick* or the King James Bible).
  - Store the words in a collection and report the # of unique words.
  - Once you've created this collection, allow the user to search it to see whether various words appear in the text file.

- What collection is appropriate for this problem?
Sets (11.2)

- **set**: A collection of unique values (no duplicates allowed) that can perform the following operations efficiently:
  - add, remove, search (contains)

- We don't think of a set as having indexes; we just add things to the set in general and don't worry about order.

```java
set.contains("to")  // true
set.contains("be")  // false
```
Set methods

In Java, Set is an interface that allows you to call the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(\textit{value})</td>
<td>adds the given value to the set</td>
</tr>
<tr>
<td>contains(\textit{value})</td>
<td>returns \textit{true} if the given value is found in this set</td>
</tr>
<tr>
<td>remove(\textit{value})</td>
<td>removes the given value from the set</td>
</tr>
<tr>
<td>clear()</td>
<td>removes all elements of the set</td>
</tr>
<tr>
<td>size()</td>
<td>returns the number of elements in list</td>
</tr>
<tr>
<td>isEmpty()</td>
<td>returns \textit{true} if the set's size is 0</td>
</tr>
<tr>
<td>toString()</td>
<td>returns a string such as &quot;[3, 42, -7, 15]&quot;</td>
</tr>
</tbody>
</table>
Set implementation

- in Java, sets are represented by `Set` type in `java.util`
- `Set` is implemented by `HashSet` and `TreeSet` classes
  - `HashSet`: implemented using a "hash table" array; very fast: $O(1)$ for all operations; elements are stored in unpredictable order
  - `TreeSet`: implemented using a "binary search tree"; pretty fast: $O(\log N)$ for all operations; elements are stored in sorted order

```java
Set<Integer> numbers = new TreeSet<Integer>();
Set<String> words = new HashSet<String>();
```
The "for each" loop (7.1)

```java
for (type name : collection) {
    statements;
}
```

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

```java
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?
Maps (11.3)

- **map**: Holds a set of key-value pairs, where each key is unique
  a.k.a. "dictionary", "associative array", "hash"

```
map.get("the")
```

```
<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;at&quot;</td>
<td>43</td>
</tr>
<tr>
<td>&quot;in&quot;</td>
<td>37</td>
</tr>
<tr>
<td>&quot;me&quot;</td>
<td>22</td>
</tr>
<tr>
<td>&quot;the&quot;</td>
<td>56</td>
</tr>
<tr>
<td>&quot;you&quot;</td>
<td>22</td>
</tr>
<tr>
<td>&quot;why&quot;</td>
<td>14</td>
</tr>
</tbody>
</table>
```
Maps (11.3)

- **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.

```
myMap.get("Aug") returns 37.3
```
Maps and tallying

- a map can be thought of as generalization of a tallying array
  - the "index" (key) doesn't have to be an int
  - count digits: 22092310907

<table>
<thead>
<tr>
<th>key</th>
<th>&quot;C&quot;</th>
<th>&quot;V&quot;</th>
<th>&quot;S&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>16</td>
<td>14</td>
<td>3</td>
</tr>
</tbody>
</table>

count votes: "CVVVVVVCCCCCVCVVVVVCVCCSCVCCSCVCCSV"

// (C) hocolate, (V) anilla, (S) trawberry

index 0 1 2 3 4 5 6 7 8 9
value 3 1 3 0 0 0 0 1 0 2
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
- Maps require 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>();

// maps from Integer keys to String values
Map<Integer, String> words = new TreeMap<Integer, String>();
```
## 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 (<code>null</code> if not found)</td>
</tr>
<tr>
<td><code>containsKey(key)</code></td>
<td>returns <code>true</code> 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 <code>true</code> if the map's size is 0</td>
</tr>
<tr>
<td><code>toString()</code></td>
<td>returns a string such as &quot;{a=90, d=60, c=70}&quot;</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 <code>true</code> 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"
```
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
    System.out.println(name + " -> " + age);  // 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)
S
  /\  \
 NP  VP
 /
 John

  \
 V
 /
 lost

  \
 NP
   /
 Det  N
     /
    his  pants
Languages and grammars

- **(formal) **language: A set of words or symbols.

- **grammar**: A description of a language that describes which sequences of symbols are allowed in that language.
  - describes language syntax (rules) but not semantics (meaning)
  - can be used to generate strings from a language, or to determine whether a given string belongs to a given language
Backus-Naur (BNF)

- **Backus-Naur Form (BNF):** A syntax for describing language grammars in terms of transformation rules, of the form:

  `<symbol> ::= <expression> | <expression> ... | <expression>`

- **terminal:** A fundamental symbol of the language.
- **non-terminal:** A high-level symbol describing language syntax, which can be transformed into other non-terminal or terminal symbol(s) based on the rules of the grammar.
- developed by two Turing-award-winning computer scientists in 1960 to describe their new ALGOL programming language
An example BNF grammar

<s>::=<n> <v>
<n>::=Marty | Victoria | Stuart | Jessica
<v>::=cried | slept | belched

- Some sentences that could be generated from this grammar:
  Marty slept
  Jessica belched
  Stuart cried
BNF grammar version 2

<s>::=<np> <v>

<np>::=<pn> | <dp> <n>

<pn>::=Marty | Victoria | Stuart | Jessica
<dp>::=a | the
<n>::=ball | hamster | carrot | computer
<v>::=cried | slept | belched

- Some sentences that could be generated from this grammar:

  the carrot cried
  Jessica belched
  a computer slept
BNF grammar version 3

<s>::=<np>  <v>

<np>::=<pn>  |  <dp>  <adj>  <n>

<pn>::=Marty | Victoria | Stuart | Jessica

<dp>::=a  |  the

<adj>::=silly | invisible | loud | romantic

<n>::=ball | hamster | carrot | computer

<v>::=cried | slept | belched

- Some sentences that could be generated from this grammar:

  the invisible carrot cried
  Jessica belched
  a computer slept
  a romantic ball belched
Grammars and recursion

<s>::=<np> <v>
<np>::=<pn> | <dp> <adjp> <n>
<pn>::=Marty | Victoria | Stuart | Jessica
<dp>::=a | the
<adjp>::=<adj> <adjp> | <adj>
<adj>::=silly | invisible | loud | romantic
<n>::=ball | hamster | carrot | computer
<v>::=cried | slept | belched

- Grammar rules can be defined *recursively*, so that the expansion of a symbol can contain that same symbol.
- There must also be expressions that expand the symbol into something non-recursive, so that the recursion eventually ends.
Grammar, final version

\[ <s> ::= <np> <vp> \]
\[ <np> ::= <dp> <adjp> <n> | <pn> \]
\[ <dp> ::= the | a \]
\[ <adjp> ::= <adj> | <adj> <adjp> \]
\[ <adj> ::= big | fat | green | wonderful | faulty | subliminal \]
\[ <n> ::= dog | cat | man | university | father | mother | child \]
\[ <pn> ::= John | Jane | Sally | Spot | Fred | Elmo \]
\[ <vp> ::= <tv> <np> | <iv> \]
\[ <tv> ::= hit | honored | kissed | helped \]
\[ <iv> ::= died | collapsed | laughed | wept \]

- Could this grammar generate the following sentences?
  Fred honored the green wonderful child
  big Jane wept the fat man fat

- Generate a random sentence using this grammar.
Fred honored the green wonderful child.