

Building Java Programs

Chapter 11
Sets and Maps

reading: 11.2 - 11.3



Road Map

CS Concepts

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



Data Structures

- Lists
- Stacks
- Queues
- Sets
- **Maps**
- Priority Queues

Java Language

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

Java Collections

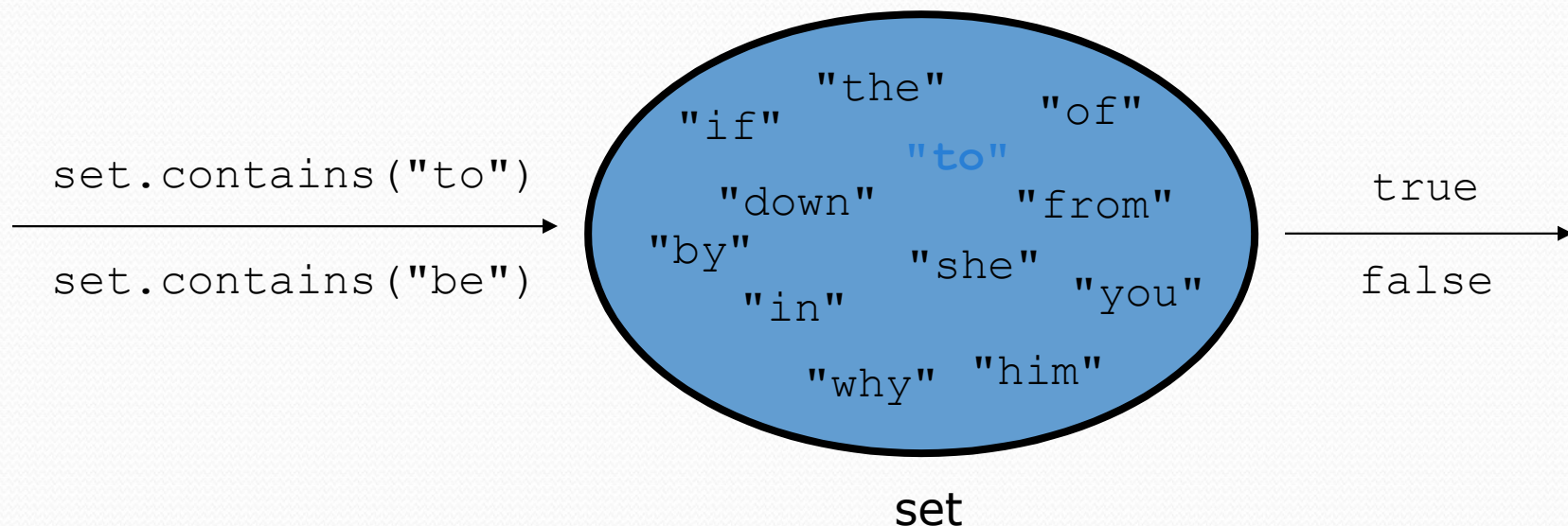
- Arrays
- ArrayList 
- LinkedList 
- Stack
- TreeSet / **TreeMap**
- HashSet / **HashMap**
- PriorityQueue

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



Set implementation

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

Set methods

```
List<String> list = new ArrayList<String>();  
...  
Set<Integer> set = new TreeSet<Integer>(); // empty  
Set<String> set2 = new HashSet<String>(list);
```

- can construct an empty set, or one based on a given collection

<code>add (value)</code>	adds the given value to the set
<code>contains (value)</code>	returns <code>true</code> if the given value is found in this set
<code>remove (value)</code>	removes the given value from the set
<code>clear ()</code>	removes all elements of the set
<code>size ()</code>	returns the number of elements in list
<code>isEmpty ()</code>	returns <code>true</code> if the set's size is 0
<code>toString ()</code>	returns a string such as "[3, 42, -7, 15]"

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?

Counting

- What if we wanted to use something other than an int as an index?
 - count digits: 22092310907

→

index	0	1	2	3	4	5	6	7	8	9
value	3	1	3	0	0	0	0	1	0	2

- count votes: // (C)hocolate, (V)anilla, (S)trawberry
"CVVVVVVVCCCCCVVVVVVVCVCCSCVCCSCVCCSV"

key	"C"	"V"	"S"
value	16	14	3

Maps (11.3)

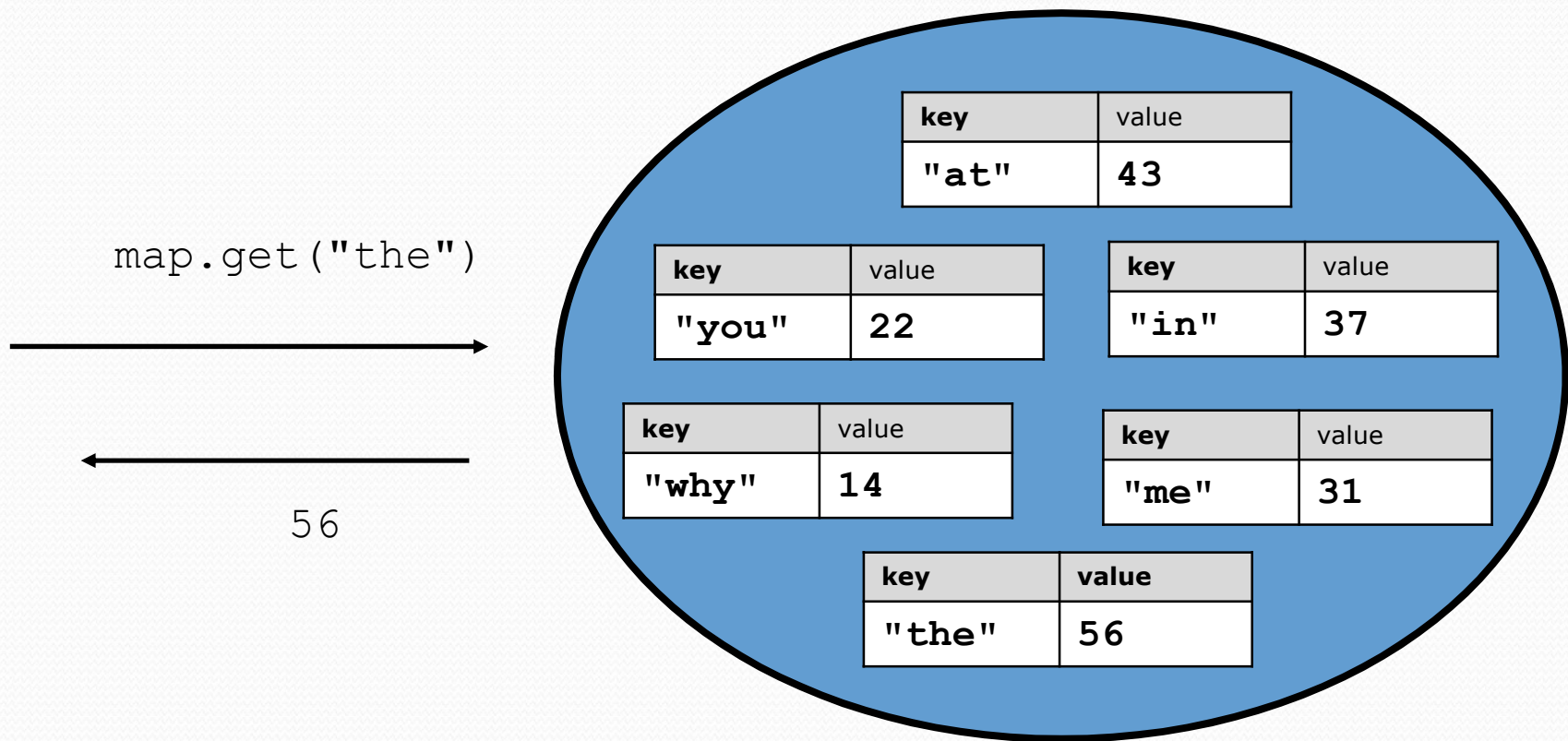
- **map**: Holds a set of unique *keys* and a collection of *values*, where each key is associated with one value.
 - a.k.a. "dictionary", "associative array", "hash"
- basic map operations:
 - **put**(*key*, *value*): Adds a mapping from a key to a value.
 - **get**(*key*): Retrieves the value mapped to the key.
 - **remove**(*key*): Removes the given key and its mapped value.

	KEYS	VALUES	
	Jan	327.2	
	Feb	368.2	
	Mar	197.6	
	Apr	178.4	
	May	100.0	
	Jun	69.9	
	Jul	32.3	
Aug →	Aug	37.3	→ 37.3
	Sep	19.0	
	Oct	37.0	
	Nov	73.2	
	Dec	110.9	
	Annual	1551.0	

`myMap.get("Aug")` returns 37.3

Maps (11.3)

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



Map implementation

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

```
// maps from String keys to Integer values
```

```
Map<String, Integer> votes = new HashMap<String, Integer>();
```

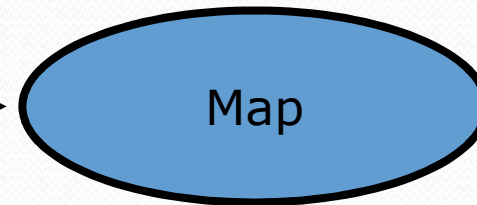
Map methods

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

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).

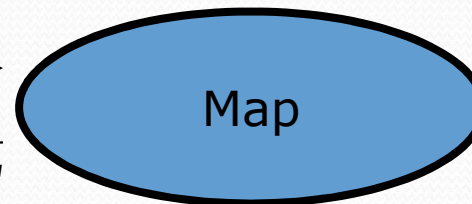
```
// key value  
put("Suzy", "206-685-2181")
```



- Later, we can supply only the key and get back the related value:

Allows us to ask: *What is Suzy's phone number?*

```
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

```
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()) { // Geneva -> 2
    int age = ages.get(name); // 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)