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

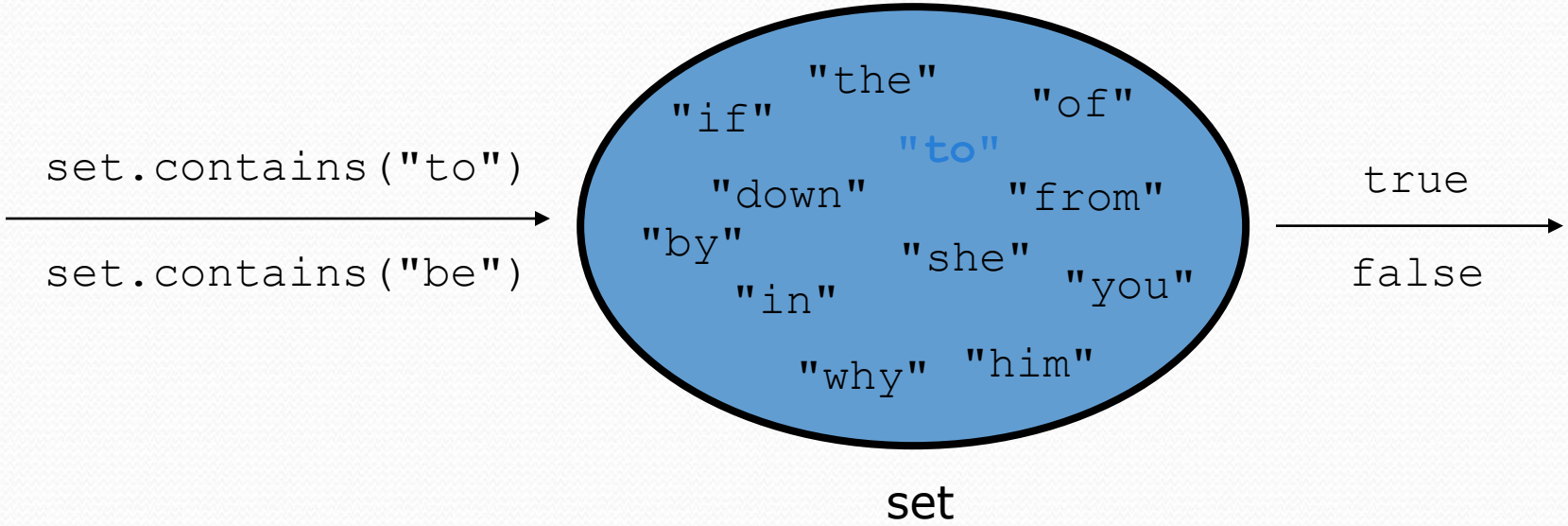
- Arrays
- ArrayList 🛠
- LinkedList 🛠
- Stack
- TreeSet / TreeMap
- HashSet / HashMap
- PriorityQueue

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



`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

Exercise

- Write a program to <u>count the number of occurrences</u> 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

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

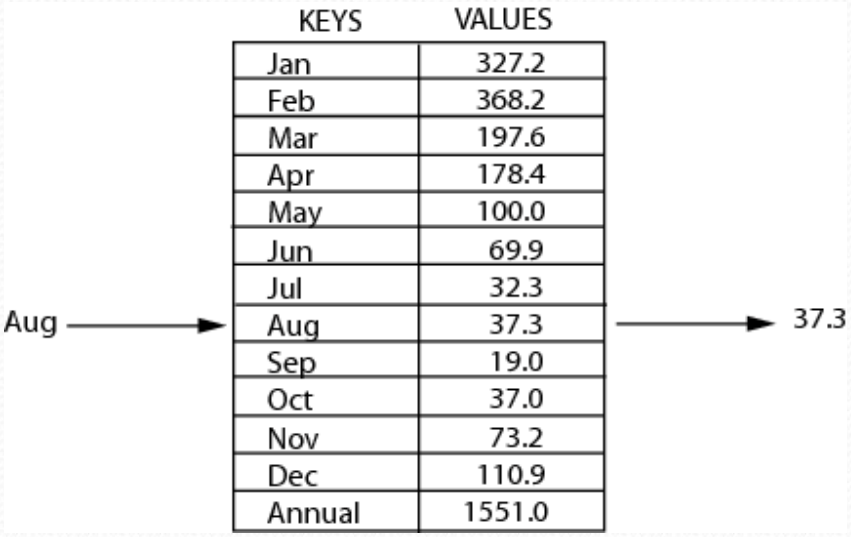
  - count votes: `"CVVVVVVCCCCCVVVVVVCVCCSCVCCSCVCCSV"`

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.



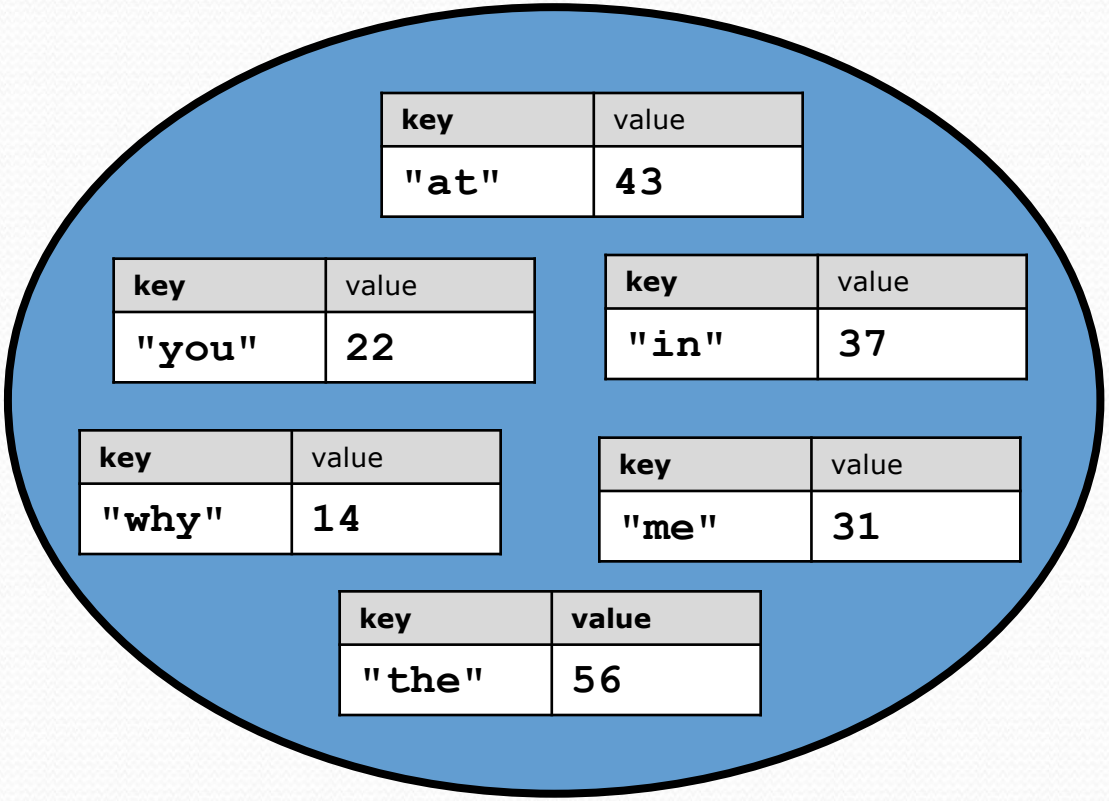
`myMap.get("Aug")` returns `37.3`

Maps (11.3)

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

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

```
56
```



`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

`put(`**key**`, `**value**`)`	adds a mapping from the given key to the given value; if the key already exists, replaces its value with the given one
`get(`**key**`)`	returns the value mapped to the given key (`null` if not found)
`containsKey(`**key**`)`	returns `true` if the map contains a mapping for the given key
`remove(`**key**`)`	removes any existing mapping for the given key
`clear()`	removes all key/value pairs from the map
`size()`	returns the number of key/value pairs in the map
`isEmpty()`	returns `true` if the map's size is 0
`toString()`	returns a string such as `"{a=90, d=60, c=70}"`

`keySet()`	returns a set of all keys in the map
`values()`	returns a collection of all values in the map
`putAll(`**map**`)`	adds all key/value pairs from the given map to this map
`equals(`**map**`)`	returns `true` if given map has the same mappings as this one