Lecture 10: Sets and Maps

07/15/22

Midterm Information

- Friday, July 22: Midterm Exam, 10:50 - 11:50, GUG 220

- Midterm resources posted on the course website
- Left–handed seat request due EOD today! (see pinned post on Ed)

Midterm Content

1. Recursive Tracing
2. Recursive Programming
3. Collections Programming (lists, sets, maps)
4. Linked List Nodes (before / after pictures)
5. ArrayIntList
6. Stacks and Queues

Exam Logistics

- Carefully read through the exam rules and info on the course website
- Closed book, closed note
  - Cheat sheet of useful methods provided
- Assigned seating
  - Will be posted next week
- Bring your Husky ID and a pencil/eraser

If you are sick, please stay home! Email Taylor before the exam begins.

Exam Tips

- We give **lots** of partial credit! Write down everything you know.
  - Method header, throwing an exception, return value
- Style *generally* does not matter
  - Use interfaces, generics correctly
  - Forbidden features are still forbidden on the exam
- Stacks and Queues: peek() method is not allowed
- Use proper Java syntax

Practice!!

This exam is not about memorizing – practice to improve!

Many, many resources to practice on your own
- Practice exams
- Exam question database
- Section problems
- IPL, Ed message board

Structured practice opportunities
- Optional review session - Monday 7/18 @ 1:10pm in GUG 220 (here)
- Midterm review in section - Thursday 7/21

Taking a short break from recursion…

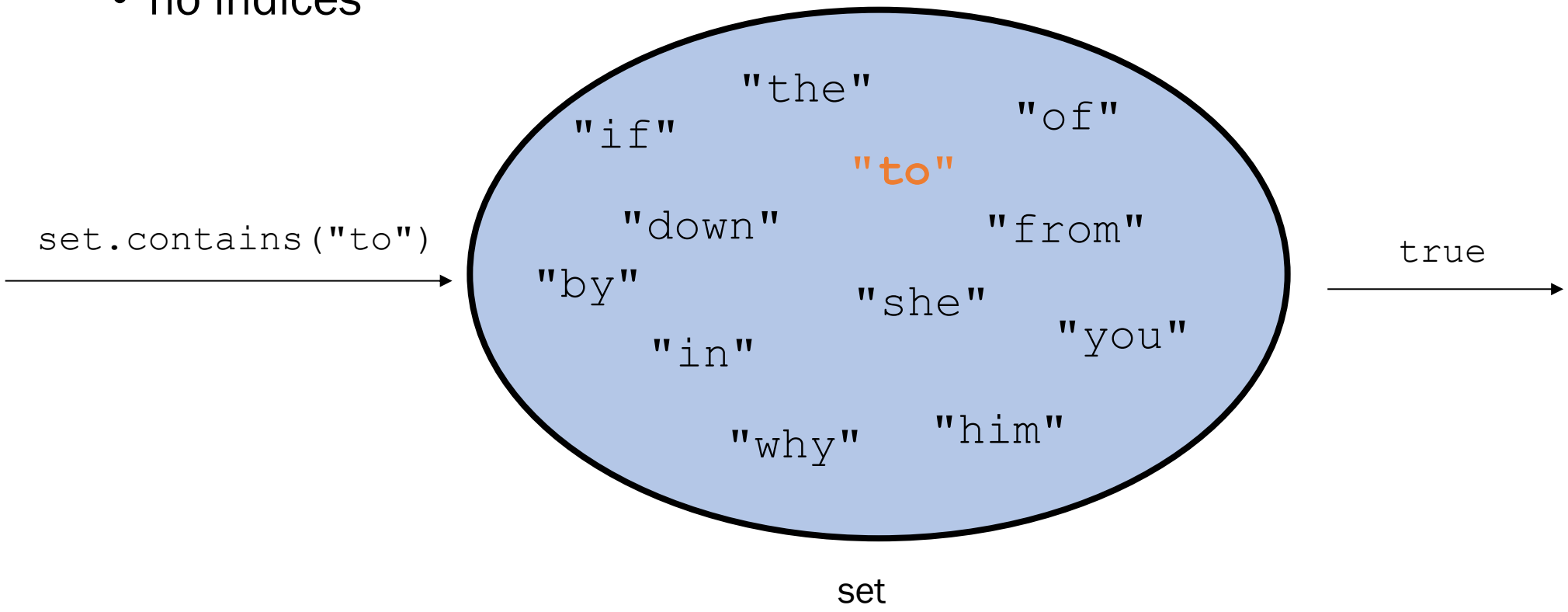
- Back to collections!
  - So far, we know about: Lists, Stacks, Queues
- Today:
  - Sets
  - Maps

countUnique

- Write a program that counts the number of unique words in a large text file (ex, *Moby Dick*).

Set ADT

- **set**: A collection of unique values (no duplicates allowed)
  - add, remove, contains
  - no indices



`Set` Interface in Java

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

```
Set<String> s1 = new TreeSet<>();
Set<Integer> s2 = new HashSet<>();
```

`Set` Implementations

- `Set` is implemented by `TreeSet` and `HashSet` classes
  - **`TreeSet`**: elements are stored in sorted order
    - *pretty fast: **O(log N)** for all operations*
  - **`HashSet`**: elements are stored in unpredictable order
    - *very fast: **O(1)** for all operations*

Note: This O(something) notation won't be covered until next week. It's okay not to know what it means yet.

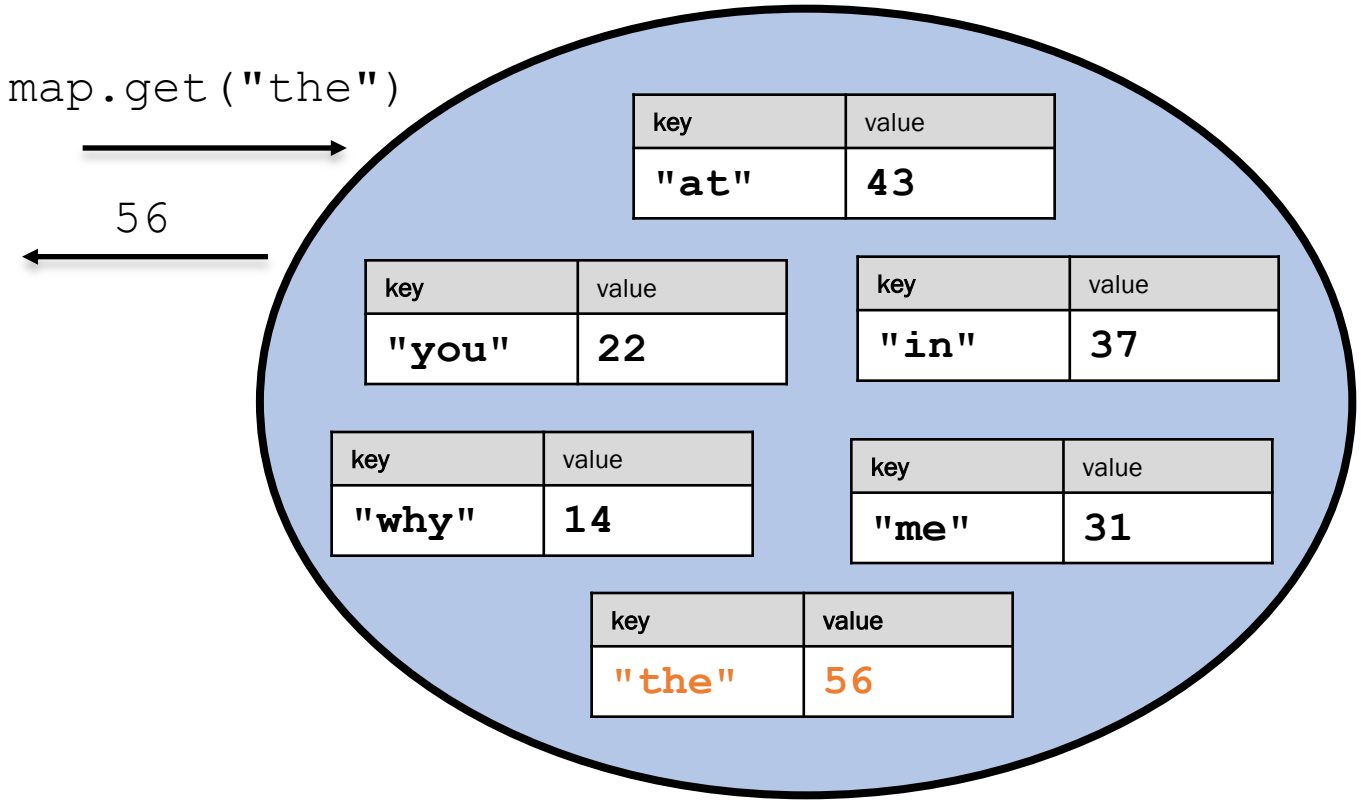
countWords

- Write a program to count the number of occurrences of each unique word in a large text file.
- Print out each unique word in alphabetical order along with its number of occurrences.

What collection is appropriate for this problem?

Map ADT

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



`Map` Interface in Java

`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

`Map` Implementations

- `Map` is implemented by `TreeMap` and `HashMap` classes
  - **`TreeMap`**: keys are stored in sorted order
    - *pretty fast: **O(log N)** for all operations*
  - **`HashMap`**: keys are stored in unpredictable order
    - *very fast: **O(1)** for all operations*
- A map requires 2 type params: one for keys, one for values.

```
// maps from String keys to Integer values
Map<String, Integer> map = new HashMap<String, Integer>();
```