

CS 106A, Lecture 20

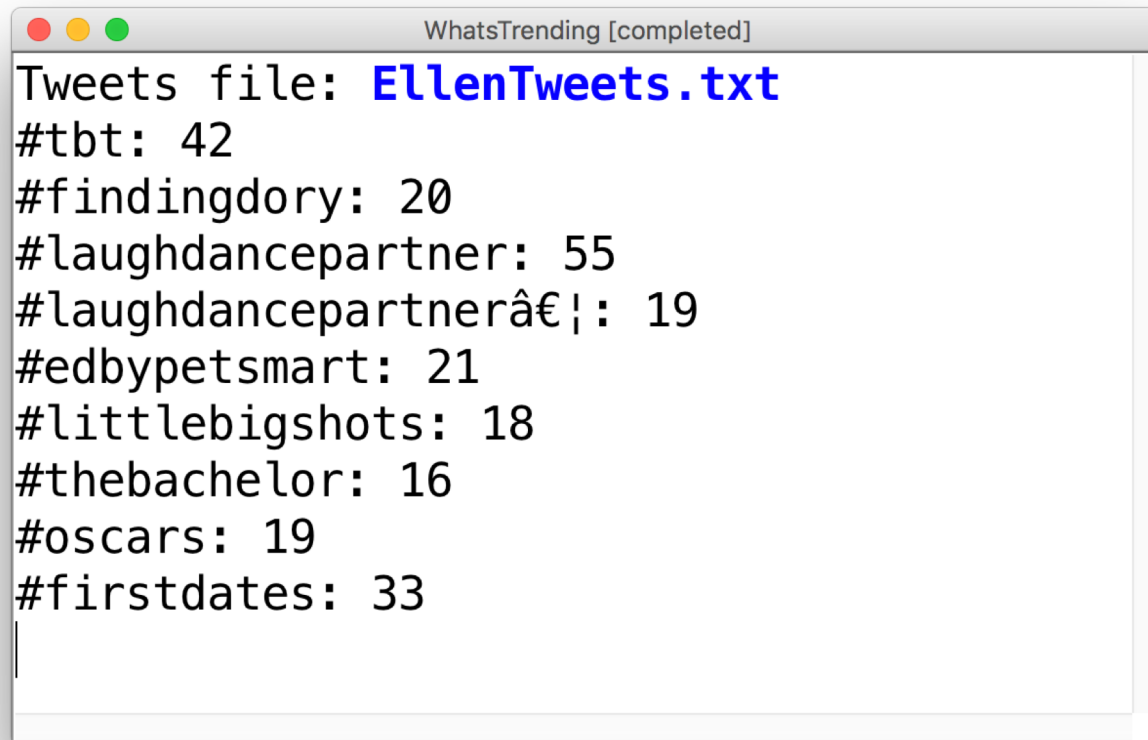
HashMaps

suggested reading:

Java Ch. 13.2

Learning Goals

- Know how to store data in and retrieve data from a **HashMap**.



```
WhatsTrending [completed]
Tweets file: EllenTweets.txt
#tbt: 42
#findingdory: 20
#laughdancepartner: 55
#laughdancepartnerâ€¦: 19
#edbypetsmart: 21
#littlebigshots: 18
#thebachelor: 16
#oscars: 19
#firstdates: 33
|
```

Plan for today

- Finish up ArrayLists
- HashMaps
- Practice: Dictionary
- HashMaps as Counters
- Practice: What's Trending
- Recap

Plan for today

- Finish up ArrayLists
- HashMaps
- Practice: Dictionary
- HashMaps as Counters
- Practice: What's Trending
- Recap

Our First ArrayList

```
// Create an (initially empty) list  
ArrayList<String> list = new ArrayList<>();
```

```
// Add an element to the back  
list.add("Hello");    // now size 1
```

"Hello"

```
list.add("there!");    // now size 2
```

"Hello"

"there!"

```
// Access elements by index (starting at 0!)  
println(list.get(0));    // prints "Hello"  
println(list.get(1));    // prints "there!"
```

Example: Reversible Writing

Let's write a program that reverses a text file.

I am not a person who contributes
And I refuse to believe that
I will be useful

I will be useful
And I refuse to believe that
I am not a person who contributes

Example: Reversible Writing

```
String filename = promptUserForFile("Filename: ", "res");
try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
} catch (IOException ex) {
    println("Could not read file.");
}
```

Example: Reversible Writing

```
String filename = promptUserForFile("Filename: ", "res");
try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
} catch (IOException ex) {
    println("Could not read file.");
}
```


Example: Reversible Writing

```
String filename = promptUserForFile("Filename: ", "res");
try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
} catch (IOException ex) {
    println("Could not read file.");
}
```

Example: Reversible Writing

```
String filename = promptUserForFile("Filename: ", "res");
try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
} catch (IOException ex) {
    println("Could not read file.");
}
```

Example: Reversible Writing

```
String filename = promptUserForFile("Filename: ", "res");
try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
} catch (IOException ex) {
    println("Could not read file.");
}
```

Example: Reversible Writing

```
String filename = promptUserForFile("Filename: ", "res");
try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
} catch (IOException ex) {
    println("Could not read file.");
}
```

Example: Reversible Writing

```
String filename = promptUserForFile("Filename: ", "res");
try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
} catch (IOException ex) {
    println("Could not read file.");
}
```

Example: Reversible Writing

```
String filename = promptUserForFile("Filename: ", "res");
try {
    Scanner s = new Scanner(new File(filename));
    ArrayList<String> lines = new ArrayList<>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
} catch (IOException ex) {
    println("Could not read file.");
}
```

ArrayList Methods

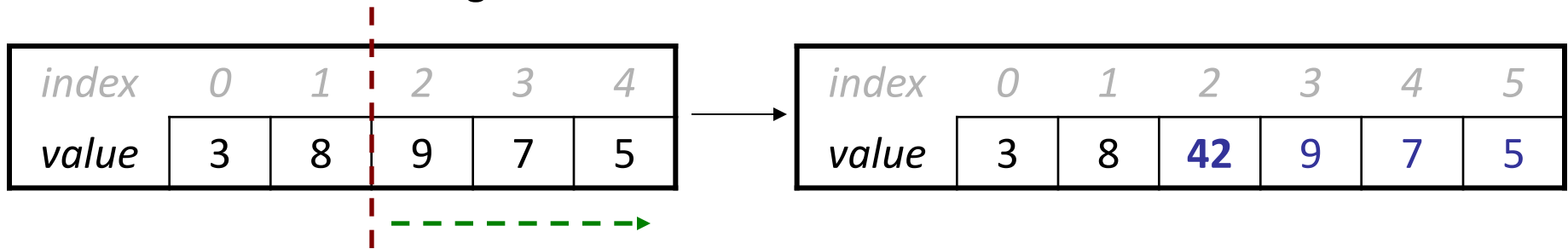
<i>list.add(value);</i>	appends value at end of list
<i>list.add(index, value);</i>	inserts given value just before the given index, shifting subsequent values to the right
<i>list.clear();</i>	removes all elements of the list
<i>list.get(index)</i>	returns the value at given index
<i>list.indexOf(value)</i>	returns first index where given value is found in list (-1 if not found)
<i>list.isEmpty()</i>	returns true if the list contains no elements
<i>list.remove(index);</i>	removes/returns value at given index, shifting subsequent values to the left
<i>list.remove(value);</i>	removes the first occurrence of the value, if any
<i>list.set(index, value);</i>	replaces value at given index with given value
<i>list.size()</i>	returns the number of elements in the list
<i>list.toString()</i>	returns a string representation of the list such as "[3, 42, -7, 15]"

Insert/remove

- If you insert/remove in the front or middle of a list, elements **shift** to fit.

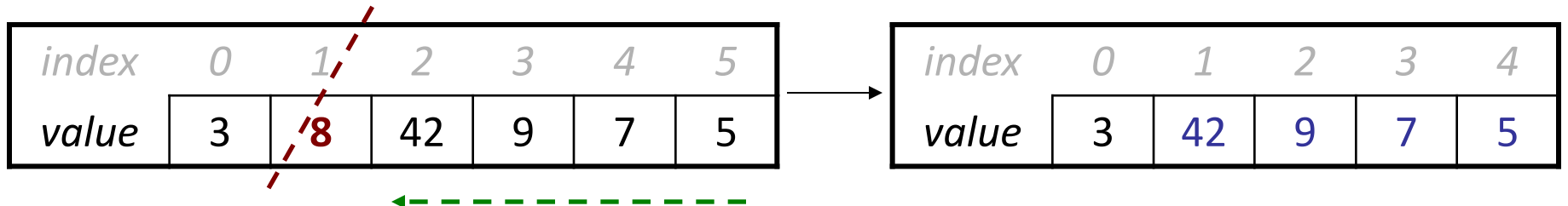
```
list.add(2, 42);
```

- shift elements right to make room for the new element



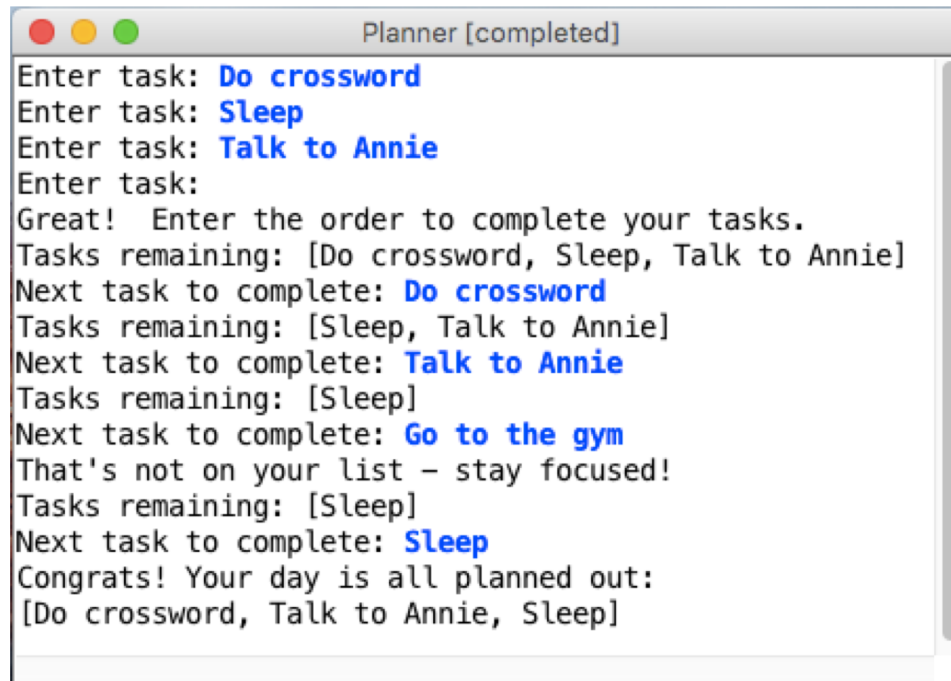
```
list.remove(1);
```

- shift elements left to cover the space left by the removed element



Example: Planner

- Let's write a program to help plan out our day
 - The program first prompts for things you want to do today
 - Then, it asks the user to re-input them in order of completion
 - Finally, it outputs the order the user has chosen for their tasks



```
Planner [completed]
Enter task: Do crossword
Enter task: Sleep
Enter task: Talk to Annie
Enter task:
Great! Enter the order to complete your tasks.
Tasks remaining: [Do crossword, Sleep, Talk to Annie]
Next task to complete: Do crossword
Tasks remaining: [Sleep, Talk to Annie]
Next task to complete: Talk to Annie
Tasks remaining: [Sleep]
Next task to complete: Go to the gym
That's not on your list - stay focused!
Tasks remaining: [Sleep]
Next task to complete: Sleep
Congrats! Your day is all planned out:
[Do crossword, Talk to Annie, Sleep]
```

Planner: Approach

Todos:

**“Do
crossword”**

Planner: Approach

Todos:

**“Do
crossword”**

“Sleep”

Planner: Approach

Todos:


**“Do
crossword”**

“Sleep”

**“Talk to
Annie”**

Planner: Approach

Todos:



“Do crossword”	“Sleep”	“Talk to Annie”
---------------------------	----------------	----------------------------


Order:

“Do crossword”

Planner: Approach

Todos:

"Sleep"	"Talk to Annie"
---------	-----------------



Order:

"Do crossword"

Planner: Approach

Todos:

~~“Sleep”~~

Order:

“Do
crossword”

“Talk to
Annie”

Planner: Approach

Todos:

DONE !

Order:

“Do
crossword”

“Talk to
Annie”

“Sleep”

Planner: Execution

To the code!

ArrayLists + Primitives =

// Doesn't compile 😞

```
ArrayList<int> list = new ArrayList<>();
```

Unlike arrays, ArrayLists can only
store **Objects!**

ArrayLists + Primitives =

Primitive	“Wrapper” Class
int	Integer
double	Double
boolean	Boolean
char	Character

ArrayLists + Wrappers = ❤️

// Use wrapper classes when making an ArrayList

```
ArrayList<Integer> list = new ArrayList<>();
```

// Java converts Integer <-> int automatically!

```
int num = 123;
```

```
list.add(num);
```

```
int first = list.get(0);    // 123
```

Conversion happens automatically!

Array vs. ArrayList

ArrayList

```
ArrayList<Integer> list =  
    new ArrayList<>();
```

```
list.add(1);        // [1]  
list.add(2);        // [1, 2]
```

```
list.set(0, 3);     // [3, 2]  
int x = list.get(0); // 3
```

```
list.add(4);        // [3, 2, 4]  
list.contains(2);    // true
```

Array

```
int[] arr =  
    new int[2];    // [0, 0]
```

```
arr[0] = 1;        // [1, 0]  
arr[1] = 2;        // [1, 2]
```

```
arr[0] = 3;        // [3, 2]  
int x = arr[0];    // 3
```

[no equivalent]

Array vs. ArrayList

Why do both of these exist in the language?

- Arrays are Java's fundamental data storage
- ArrayList is a library built on top of an array

When would you choose an array over an ArrayList?

- When you need a fixed size that you know ahead of time
 - Simpler syntax for getting/setting
 - More efficient
- *Multi-dimensional* arrays (e.g., images)
- *Histograms/tallying*

Plan for today

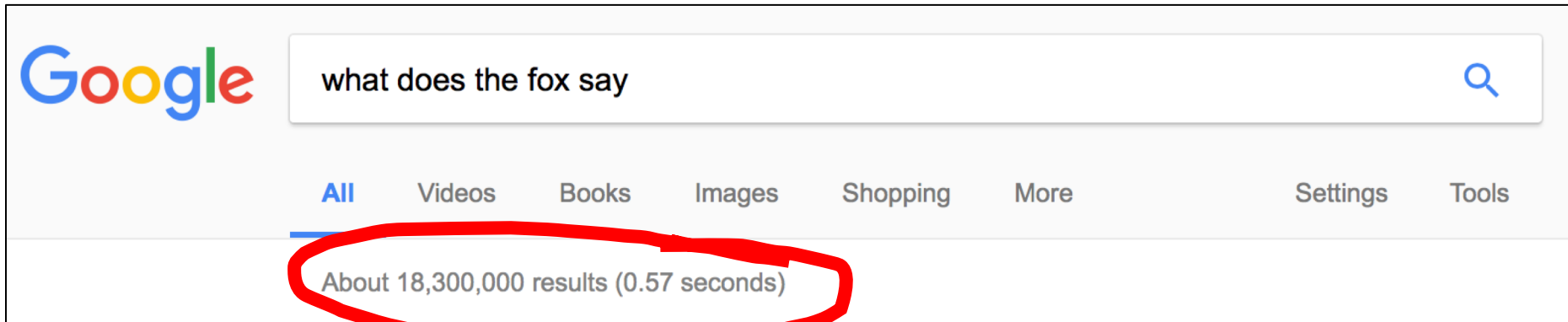
- Finish up ArrayLists
- **HashMaps**
- Practice: Dictionary
- HashMaps as Counters
- Practice: What's Trending
- Recap

Limitations of Lists

- Can only look up by *index* (int), not by String, etc.
- Cumbersome for preventing duplicate information
- Slow for lookup

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	12	49	-2	26	5	17	-6	84	72	3

How Is Webpage Lookup So Fast?



Introducing... HashMaps!

- A variable type that represents a collection of **key-value pairs**
- You access values by *key*
- Keys and values can be any type of **Object**
- Resizable – can add and remove pairs
- Has helpful methods for searching for keys

HashMap Examples

- **Phone book:** name -> phone number
- **Search engine:** URL -> webpage
- **Dictionary:** word -> definition
- **Bank:** account # -> balance
- **Social Network:** name -> profile
- **Counter:** text -> # occurrences
- And many more...

Our First HashMap

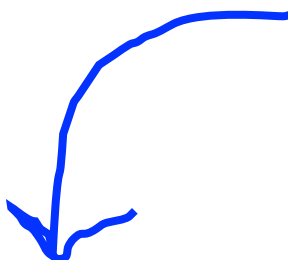
```
import java.util.*;
```

```
HashMap<String, String> myHashMap = new HashMap<>();
```

Our First HashMap

```
HashMap<String, String> myHashMap = new HashMap<>();
```

Our First HashMap

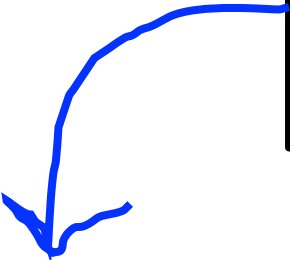


Type of keys your
HashMap will store.

```
HashMap<String, String> myHashMap = new HashMap<>();
```

Our First HashMap

Type of values your
HashMap will store.



```
HashMap<String, String> myHashMap = new HashMap<>();
```

Our First HashMap

```
HashMap<String, String> myHashMap = new HashMap<>();
```


Our First HashMap

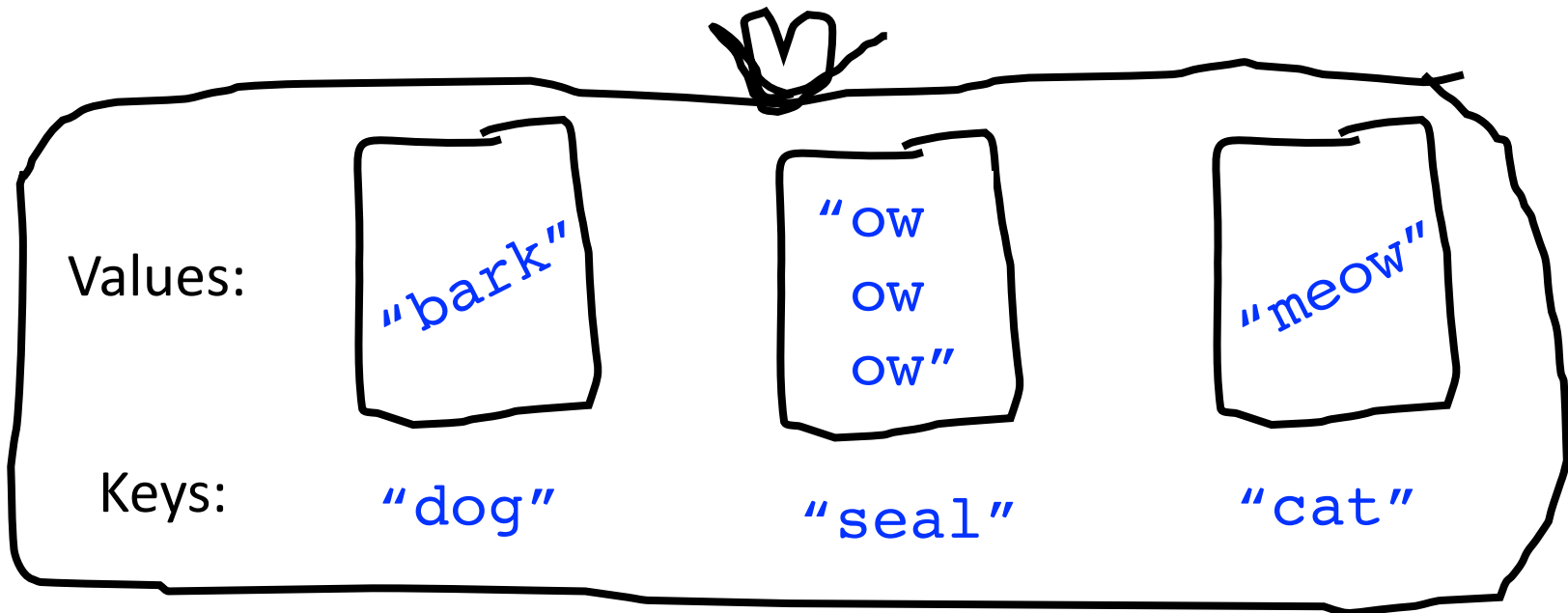
```
HashMap<String, String> myHashMap = new HashMap<>();
```

Our First HashMap

```
HashMap<String, String> myHashMap = new HashMap<>();
```

Our First HashMap - Put

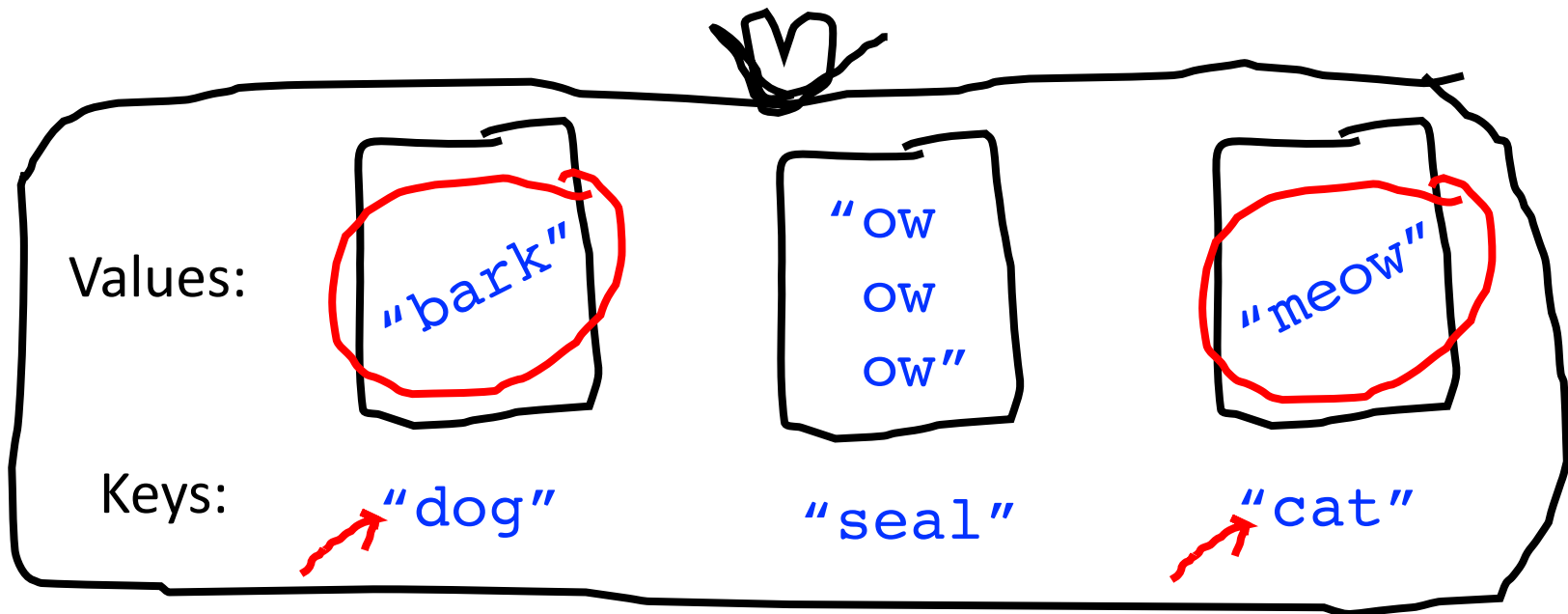
```
// Create an (initially empty) HashMap  
HashMap<String, String> map = new HashMap<>();  
map.put("dog", "bark"); // Add a key-value pair  
map.put("cat", "meow"); // Add another pair  
map.put("seal", "ow ow"); // Add another pair  
map.put("seal", "ow ow ow"); // Overwrites!
```



Our First HashMap - Get

...

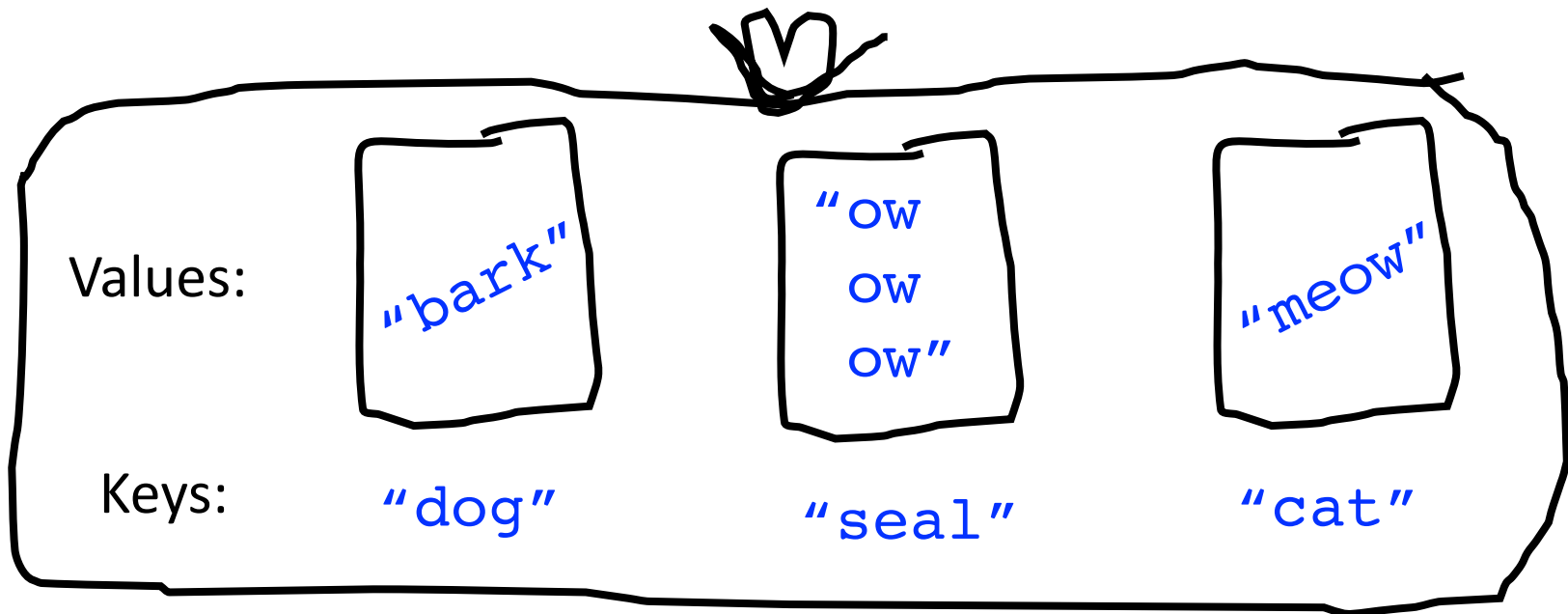
```
String s = map.get("dog"); // Get a value for a key  
String s = map.get("cat"); // Get a value for a key  
String s = map.get("fox"); // null
```



Our First HashMap - Remove

...

```
map.remove("dog"); // Remove pair from map  
map.remove("seal"); // Remove pair from map  
map.remove("fox"); // Does nothing if not in map
```



Review: HashMap Operations

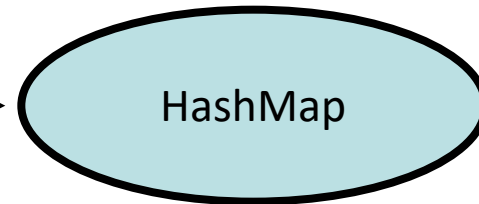
- ***m.put(key, value)***; Adds a key/value pair to the map.
`m.put("Eric", "650-123-4567");`
 - Replaces any previous value for that key.
- ***m.get(key)*** Returns the value paired with the given key.
`String phoneNum = m.get("Jenny");` // "867-5309"
 - Returns null if the key is not found.
- ***m.remove(key)***; Removes the given key and its paired value.
`m.remove("Annie");`
 - Has no effect if the key is not in the map.

<u>key</u>	<u>value</u>
"Jenny"	→ "867-5309"
"Mehran"	→ "123-4567"
"Marty"	→ "685-2181"
"Chris"	→ "947-2176"

Using HashMaps

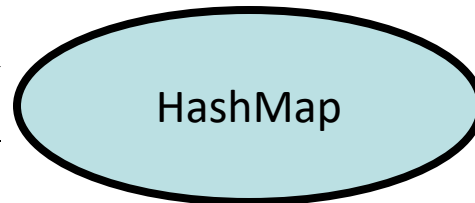
- A HashMap allows you to get from one half of a pair to the other.
 - Remembers one piece of information about every key.

```
//      key      value  
m.put("Jenny", "867-5309");
```



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

```
m.get("Jenny")  
←  
"867-5309"
```



Practice: Map Mystery

Q: What are the correct map contents after the following code?

```
HashMap<String, String> map = new HashMap<>();  
map.put("K", "Schwarz");  
map.put("C", "Lee");  
map.put("M", "Sahami");  
map.put("M", "Stepp");  
map.remove("Stepp");  
map.remove("K");  
map.put("J", "Cain");  
map.remove("C, Lee");
```

- A. {C=Lee, J=Cain, M=Stepp, M=Sahami}
- B. {C=Lee, J=Cain, M=Stepp}
- C. {J=Cain M=Sahami, M=Stepp}
- D. {J=Cain, K=Schwarz, M=Sahami}
- E. other

Practice: Map Mystery

Q: What are the correct map contents after the following code?

```
HashMap<String, String> map = new HashMap<>();  
map.put("K", "Schwarz");  
map.put("C", "Lee");  
map.put("M", "Sahami");  
map.put("M", "Stepp");  
map.remove("Stepp");  
map.remove("K");  
map.put("J", "Cain");  
map.remove("C, Lee");
```

Values:

"Schwarz"

"Sahami"

"Lee"

Keys:

"K"

"M"

"C"

Practice: Map Mystery

Q: What are the correct map contents after the following code?

```
HashMap<String, String> map = new HashMap<>();  
map.put("K", "Schwarz");  
map.put("C", "Lee");  
map.put("M", "Sahami");  
map.put("M", "Stepp");  
map.remove("Stepp");  
map.remove("K");  
map.put("J", "Cain");  
map.remove("C, Lee");
```

Values:

"Schwarz"

"Stepp"

"Lee"

Keys:

"K"

"M"

"C"

Practice: Map Mystery

Q: What are the correct map contents after the following code?

```
HashMap<String, String> map = new HashMap<>();  
map.put("K", "Schwarz");  
map.put("C", "Lee");  
map.put("M", "Sahami");  
map.put("M", "Stepp");  
map.remove("Stepp");  
map.remove("K");  
map.put("J", "Cain");  
map.remove("C, Lee");
```

Values:

"Schwarz"

"Stepp"

"Lee"

Keys:

"K"

"M"

"C"

Practice: Map Mystery

Q: What are the correct map contents after the following code?

```
HashMap<String, String> map = new HashMap<>();  
map.put("K", "Schwarz");  
map.put("C", "Lee");  
map.put("M", "Sahami");  
map.put("M", "Stepp");  
map.remove("Stepp");  
map.remove("K");  
map.put("J", "Cain");  
map.remove("C, Lee");
```

Values:

"Stepp"

"Lee"

Keys:

"M"

"C"

Practice: Map Mystery

Q: What are the correct map contents after the following code?

```
HashMap<String, String> map = new HashMap<>();  
map.put("K", "Schwarz");  
map.put("C", "Lee");  
map.put("M", "Sahami");  
map.put("M", "Stepp");  
map.remove("Stepp");  
map.remove("K");  
map.put("J", "Cain");  
map.remove("C, Lee");
```

Values:

"Cain"

"Stepp"

"Lee"

Keys:

"J"

"M"

"C"

Practice: Map Mystery

Q: What are the correct map contents after the following code?

```
HashMap<String, String> map = new HashMap<>();  
map.put("K", "Schwarz");  
map.put("C", "Lee");  
map.put("M", "Sahami");  
map.put("M", "Stepp");  
map.remove("Stepp");  
map.remove("K");  
map.put("J", "Cain");  
map.remove("C", "Lee");
```

Values:

"Cain"

"Stepp"

"Lee"

Keys:

"J"

"M"

"C"

Plan for today

- Finish up ArrayLists
- HashMaps
- **Practice: Dictionary**
- HashMaps as Counters
- Practice: What's Trending
- Recap

Exercise: Dictionary

- Write a program to read a dictionary of words and definitions from a file, then prompt the user for words to look up.
 - Example data from the dictionary input file:

```
abate  
to lessen; to subside  
pernicious  
harmful, injurious
```

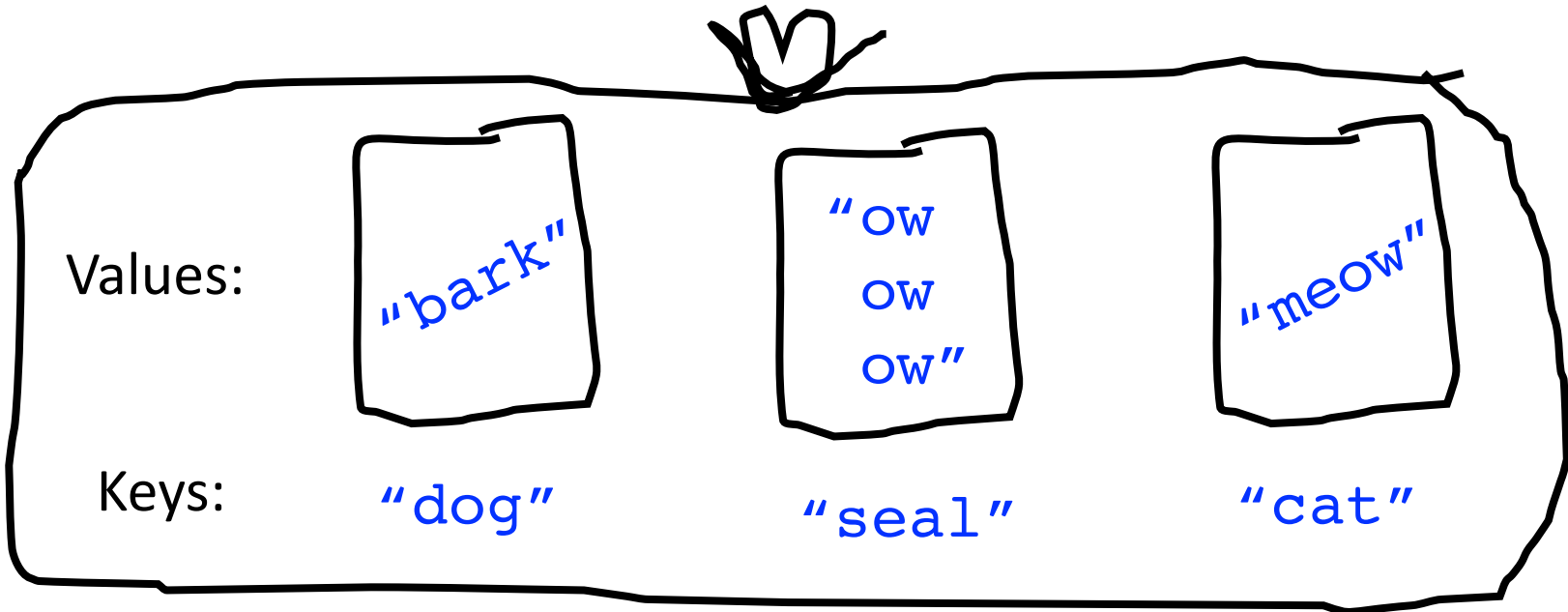
- How can a **HashMap** help us solve this problem?

Plan for today

- Finish up ArrayLists
- HashMaps
- Practice: Dictionary
- **HashMaps as Counters**
- Practice: What's Trending
- Recap

Iterating Over HashMaps

```
...  
for (String key : map.keySet()) {  
    String value = map.get(key);  
    // do something with key/value pair...  
}  
// Keys occur in an unpredictable order!
```



Counting 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.
- How can a **map** help us solve this problem?
 - Think about scanning over a file containing this input data:

To be or not to be or to be a bee not two bees ...
^

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

→

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

// (R)epublican, (D)emocrat, (I)ndependent

- count votes: "RDDDDDDRRRRRRDDDDDDRRDRRIRDRRIRDRRID"

key	"R"	"D"	"I"
value	16	14	3

<u>key</u>	<u>value</u>
"R"	→ 16
"D"	→ 14
"I"	→ 3

Plan for today

- Finish up ArrayLists
- HashMaps
- Practice: Dictionary
- HashMaps as Counters
- **Practice: What's Trending**
- Recap

Practice: What's Trending?

- Social media can be used to monitor popular conversation topics.
- Write a program to count the frequency of **#hashtags** in tweets:
 - Read saved tweets from a large text file.
 - Report hashtags that occur at least 15 times.
- How can a **map** help us solve this problem?

Given these hashtags...

```
#stanford  
#summer  
#california  
#stanford
```

We want to store...

```
"#stanford"    → 2  
"#summer"     → 1  
"#california" → 1
```

Recap

- ArrayLists are a variable type representing a list of items
- Unlike arrays, ArrayLists have:
 - The ability to resize dynamically
 - Useful methods you can call on them
- Unlike ArrayLists, arrays have:
 - The ability to store any type of item, not just Objects
- HashMaps are a variable type representing a key-value pairs
- Unlike arrays and ArrayLists, HashMaps:
 - Are not ordered
 - Store information associated with a key of any Object type

Next Time: defining our own variable types!

Overflow (extra) slides

Anagram exercise

- Write a program to find all **anagrams** of a word the user types.

Type a word [Enter to quit]: **scared**

Anagrams of scared:

cadres cedars sacred scared

- How can a **map** help us solve this problem?

Anagram observation

- Every word has a *sorted form* where its letters are arranged into alphabetical order.
 - "fare" → "aeifr"
 - "fear" → "aeifr"
 - "swell" → "ellsw"
 - "wells" → "ellsw"
- Notice that anagrams have the same sorted form as each other.
 - How is this helpful for solving the problem?
 - Suppose we were given a **sortLetters** method. How to use it?

Anagram solution

```
public String sortLetters(String s) { ... }    // assume this exists
...

// build map of {sorted form => all words with that sorted form}
HashMap<String, String> anagrams = new
    HashMap<String, String>();
try {
    Scanner input = new Scanner(new File("dictionary.txt"));
    while (true) {
        String word = input.next();
        String sorted = sortLetters(word);      // "acders"
        if (anagrams.containsKey(sorted)) {
            String rest = anagrams.get(sorted);
            anagrams.put(sorted, rest + " " + word);    // append
        } else {
            anagrams.put(sorted, word);                // new k/v pair
        }
        // {"acders" => "cadres caders sacred scared", ...}
    }
} catch (FileNotFoundException fnfe) {
    println("Error reading file: " + fnfe);
}
```

Anagram solution cont'd.

```
// prompt user for words and look up anagrams in map
String word = readLine("Type a word [Enter to quit]: ");
while (word.length() > 0) {
    String sorted = sortLetters(word.toLowerCase());
    if (anagrams.containsKey(sorted)) {
        println("Anagrams of " + word + ":");
        println(anagrams.get(sorted));
    } else {
        println("No anagrams for " + word + ".");
    }
    word = readLine("Type a word [Enter to quit]: ");
}
```