CS 106A, Lecture 22
HashMap

reading:

Art & Science of Java, 13.2

Learning Goals

- A **`HashMap`** is a new type of variable that stores **pairs**
- Learn how to create and use HashMaps

What's Trending?

- Write a program to find the subjects most discussed in the Tweets
  - Allow the user to type a word and report *how many times* that word appeared.
  - Report all words that appeared at least 200 times.

Plan for Today

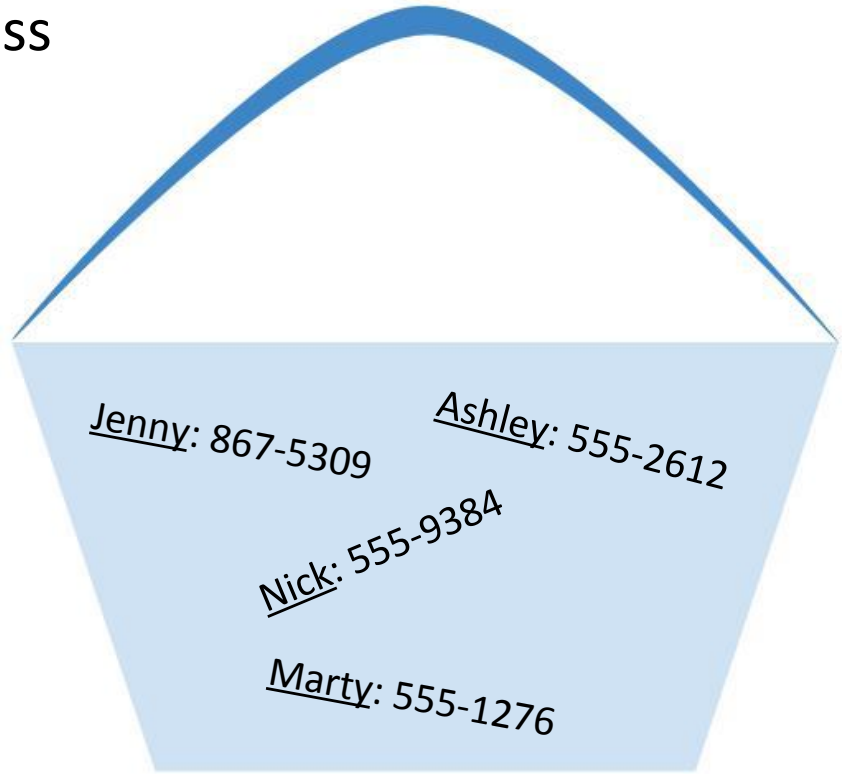
- What is a `HashMap`?
- Creating `HashMaps` and using `HashMaps`
- *Example*: Dictionary
- Counting with `HashMaps`
- *Example:* What's Trending?

Plan for Today

- What is a `HashMap`?
- Creating `HashMaps` and using `HashMaps`
- *Example*: Dictionary
- Counting with `HashMaps`
- *Example:* What's Trending?

Maps

- **map**: A collection that stores pairs, where each pair consists of a first half called a *key* and a second half called a *value*.
  - sometimes called a "dictionary", "associative array", or "hash"
  - usage: add (*key*, *value*) pairs; lookup a value by supplying a key.
- In Java, we'll use the **`HashMap`** class
- real-world examples:
  - dictionary of words and definitions
  - phone book
  - URLs to webpages



Plan for Today

- What is a `HashMap`?
- Creating `HashMaps` and using `HashMaps`
- *Example*: Dictionary
- Counting with `HashMaps`
- *Example:* What's Trending?

Creating a HashMap

- A map requires 2 type parameters: one for keys, one for values.

```
// a map from string keys to string values
HashMap<String, String> phoneBook =
                        new HashMap<String, String>();

// a map from string keys to integer values
HashMap<String, Integer> votes = new HashMap<String,
  Integer>();

// a map from int keys to BankAccount values
HashMap<Integer, BankAccount> bankVault =
          new HashMap<Integer, BankAccount>();
```

  - Java also has another type of map called a `TreeMap`
    - slightly slower than a `HashMap`
    - different internal implementation and ordering (sorted by keys)
    - both kinds of maps implement exactly the same operations

HashMap operations

- ***m***`.put(`***key***`, `***value***`);` Adds a key/value pair to the map.

  ```
  m.put("Abby", "555-1276");
  ```

  - Replaces any previous value for that key.

- ***m***`.get(`***key***`);` Returns the value paired with the given key.

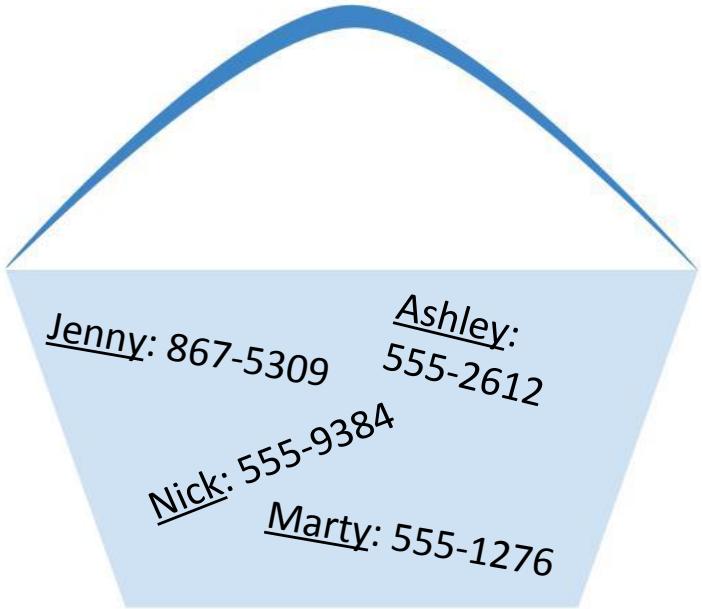
  ```
  String phoneNum = m.get("Ashley"); // "555-2612"
  ```

  - Returns null if the key is not found.

- **m**`.remove(`***key***`);` Removes the given key and its paired value.

  ```
  m.remove("Marty");
  ```

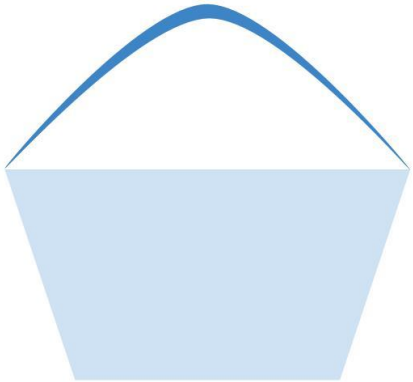
  - Has no effect if the key is not in the map.



Using HashMaps

- A map 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");
```

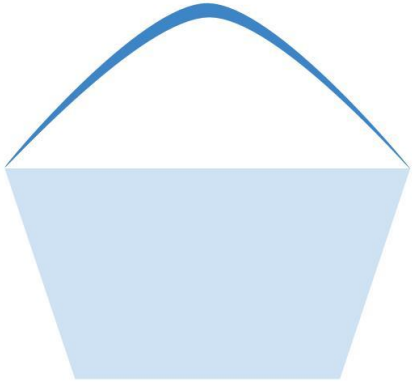


  - Later, we can supply only the key and get back the related value:

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

```
m.get("Jenny")
```

```
"867-5309"
```



HashMap members

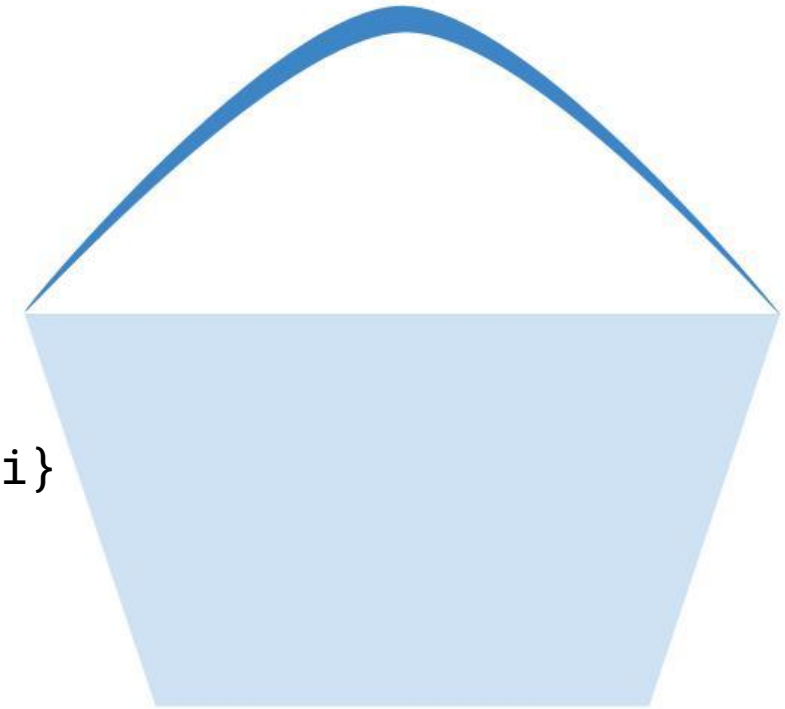
`m.clear();`	removes all key/value pairs from the map
`m.containsKey(key)`	returns `true` if the map contains a mapping for the given key
`m.get(key)`	returns the value mapped to the given key *(if not found, returns `null`)*
`m.isEmpty()`	returns `true` if the map contains no key/value pairs (size 0)
`m.keySet()`	returns a set of all keys in the map
`m.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)*
`m.remove(key);`	removes any existing mapping for the given key *(if not found, does nothing)*
`m.size()`	returns the number of key/value pairs in the map
`m.toString()`	returns a string such as `"{a=90, d=60, c=70}"`
`m.values()`	returns a collection of all values in the map

HashMap mystery

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

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

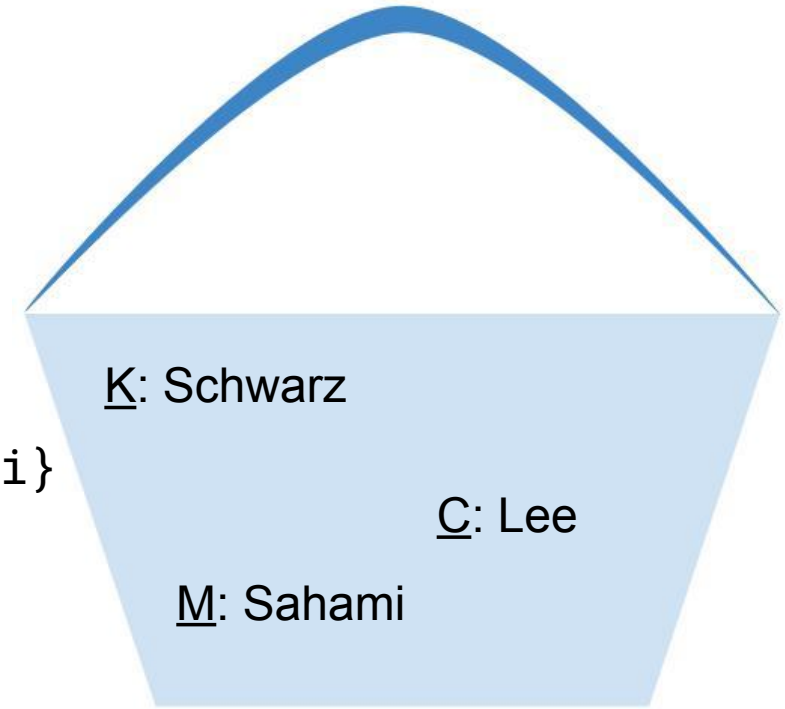


HashMap mystery

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

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

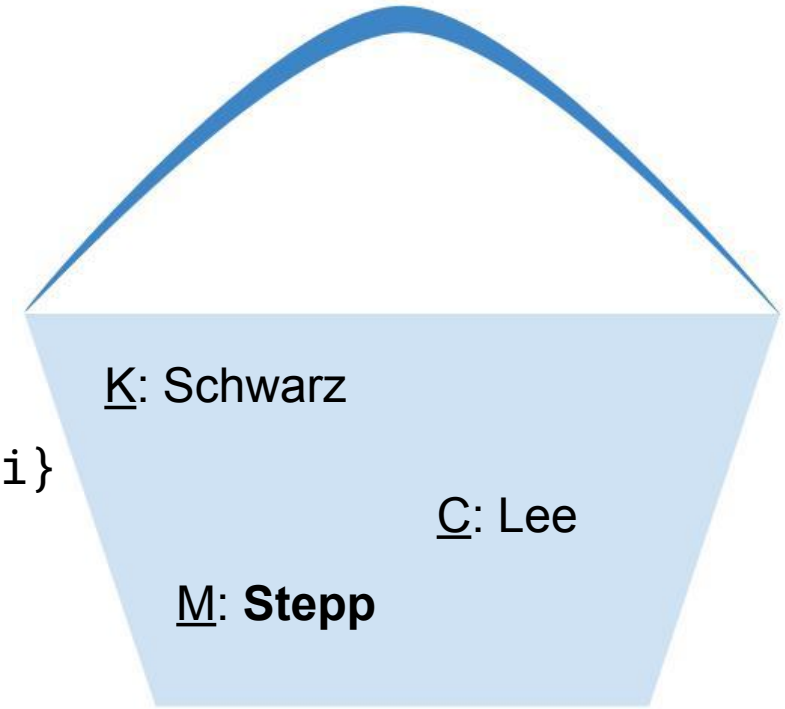


HashMap mystery

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

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

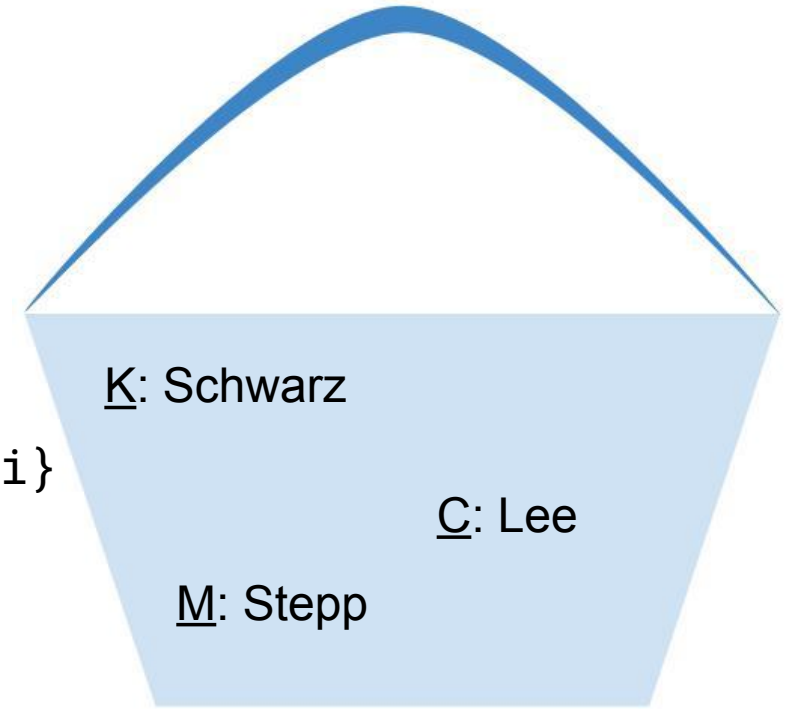


HashMap mystery

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

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

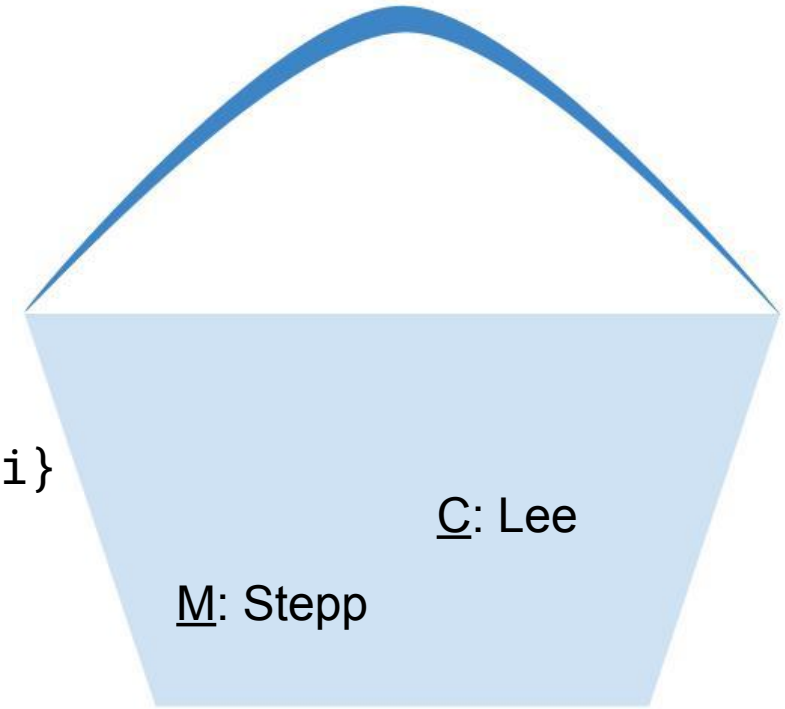


HashMap mystery

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

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

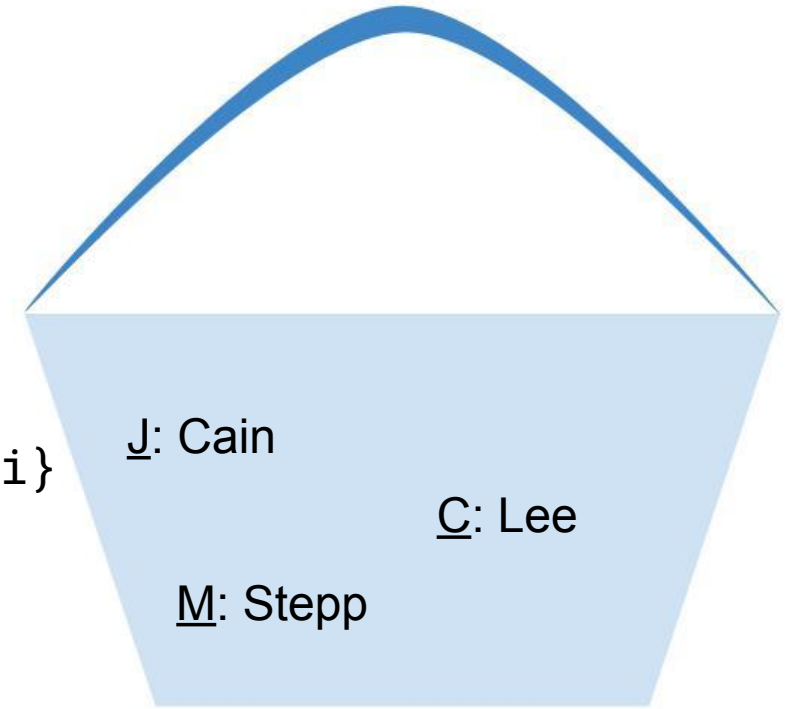


HashMap mystery

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

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

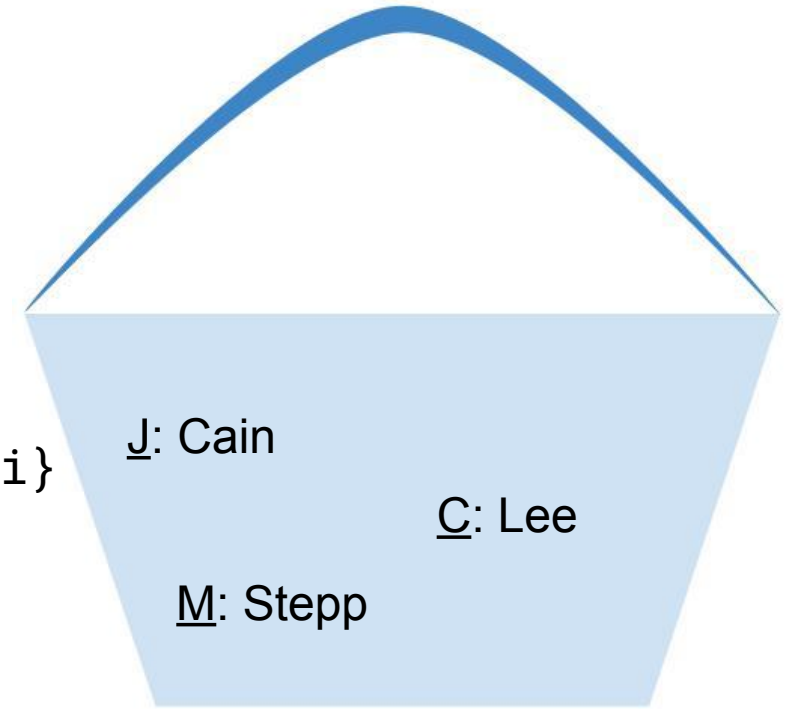


HashMap mystery

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

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



Plan for Today

- What is a `HashMap`?
- Creating `HashMaps` and using `HashMaps`
- *Example*: Dictionary
- Counting with `HashMaps`
- *Example:* What's Trending?

Dictionary exercise

- 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

- What is a `HashMap`?
- Creating `HashMaps` and using `HashMaps`
- *Example*: Dictionary
- Counting with `HashMaps`
- *Example:* What's Trending?

Looping over a HashMap

- Typical way to loop over a map is a **for-each** loop over its keys.

```
HashMap<String, Double> GPAs =
        new HashMap<String, Double>();
...
for (String name : GPAs.keySet()) {
    double gpa = GPAs.get(name);
    println(name + "'s GPA is " + gpa);
}
```

  – The keys occur in an unpredictable order in a HashMap.

HashMaps and tallying

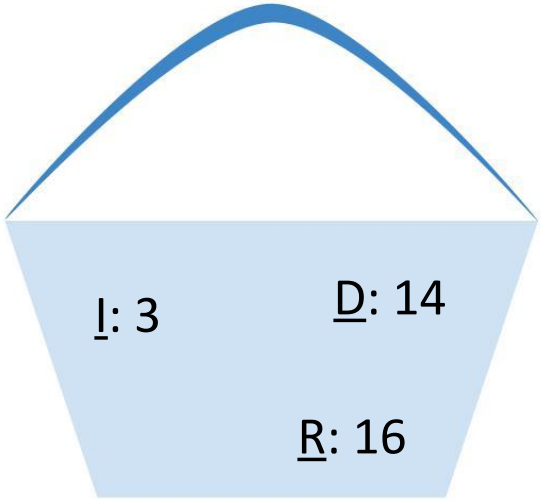
- a map can be thought of as generalization of a tallying array
  - the "index" (key) doesn't have to be an `int`
  - count digits: `22092310907`

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: `"RDDDDDDRRRRRDDDDDDRDRRIRDRRIRDRRID"`

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



Counting exercise

- Write a program to count the number of occurrences of each unique word in a large text file (e.g. Washington Post Tweets).
  - Allow the user to type a word and report how many times that word appeared in the tweets.
  - Report all words that appeared in the tweets at least 200 times.

- How can a **`HashMap`** 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 ...
^
```

Plan for Today

- What is a `HashMap`?
- Creating `HashMaps` and using `HashMaps`
- *Example*: Dictionary
- Counting with `HashMaps`
- *Example:* What's Trending?

What's trending?

- Social media can be used to monitor popular conversation topics.
- Write a program to count the frequency of words in tweets:
  - Read saved tweets from a large text file.
  - Report keywords that occur at least 200 times.
  - Punctuation has already been removed
- How can a **`HashMap`** help us solve this problem?

Given this tweet

```
CS 106A is
the best CS
class
```

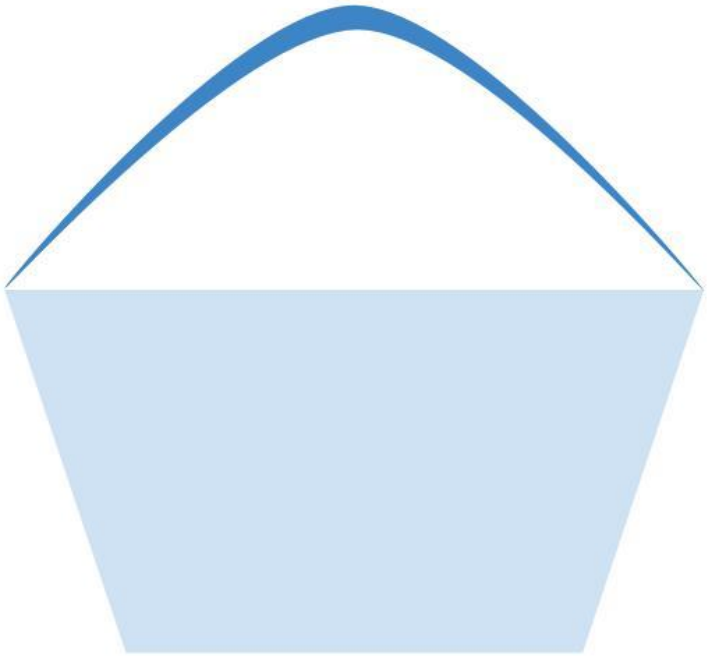
We want to store...

```
"CS"    → 2
"is"    → 1
"the"   → 1
"best"  → 1
"class" → 1
```

Compound collections

- *Example:* how would you store multiple phone numbers?
  - i.e., Home and cell numbers

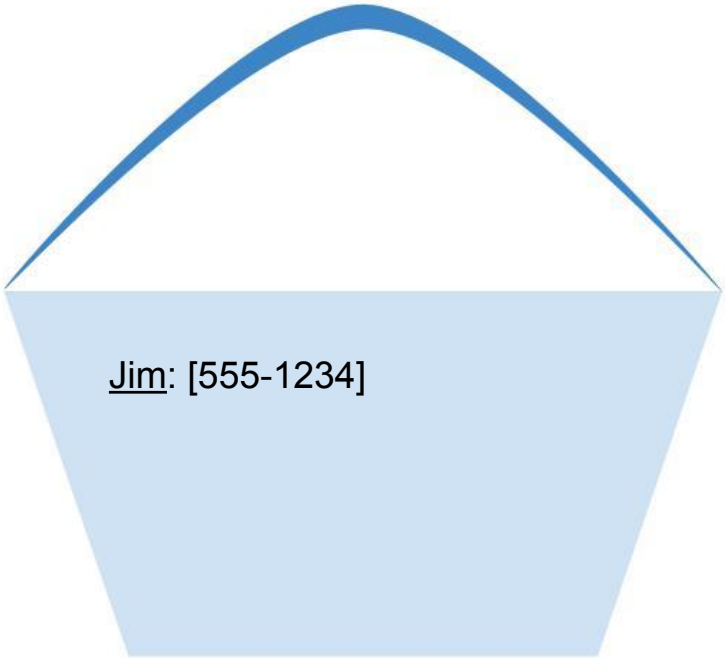
```
HashMap<String, ArrayList<String>> pals = new
    HashMap<String, ArrayList<String>>();
```



Compound collections

- *Example:* how would you store multiple phone numbers?
  - i.e., Home and cell numbers

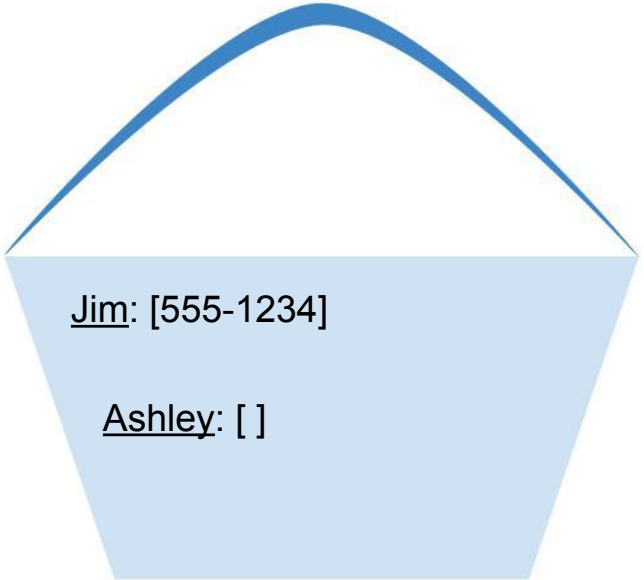
```
HashMap<String, ArrayList<String>> phonebook = new
    HashMap<String, ArrayList<String>>();
phonebook.put("Jim", new ArrayList<String>());
phonebook.get("Jim").add("555-1234");
```



Compound collections

- *Example:* how would you store multiple phone numbers?
  - i.e., Home and cell numbers

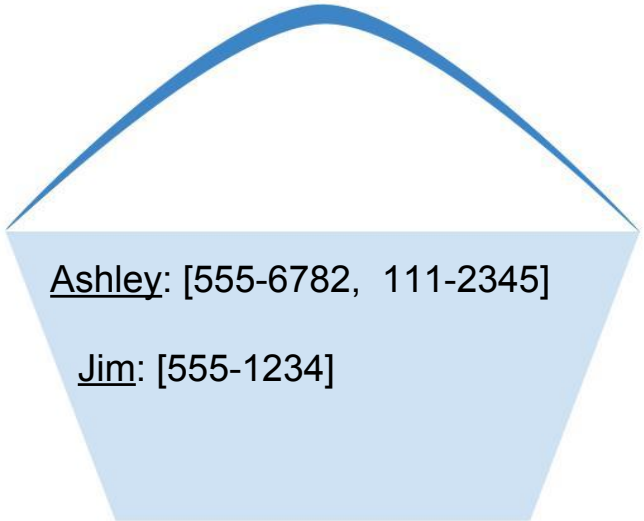
```
HashMap<String, ArrayList<String>> phonebook = new
    HashMap<String, ArrayList<String>>();
phonebook.put("Jim", new ArrayList<String>());
phonebook.get("Jim").add("555-1234");
phonebook.put("Ashley", new ArrayList<String>());
```



Compound collections

- *Example:* how would you store multiple phone numbers?
  - i.e., Home and cell numbers

```
HashMap<String, ArrayList<String>> phonebook = new
    HashMap<String, ArrayList<String>>();
phonebook.put("Jim", new ArrayList<String>());
phonebook.get("Jim").add("555-1234");
phonebook.put("Ashley", new ArrayList<String>());
phonebook.get("Ashley").add("555-6782");
phonebook.get("Ashley").add("111-2345");
println(phonebook);
// {Ashley: [555-6782, 111-2345],
//  Jim: [555-1234]}
```



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 **`HashMap`** help us solve this problem?

Anagram observation

- Every word has a *sorted form* where its letters are arranged into alphabetical order.

```
"fare"   → "aefr"
"fear"   → "aefr"
"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]: ");
}
```

Compound Anagram code

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

// build map of {sorted form => all words with that sorted form}
HashMap<String, ArrayList<String>> anagrams = new
    HashMap<String, ArrayList<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)) {
            anagrams.put(sorted, new ArrayList<String>());
        }
        if (!anagrams.get(sorted).contains(word)) {
            anagrams.get(sorted).add(word);
        }
        // {acders=[cadres, caders, sacred, scared], ...}
    }
} catch (FileNotFoundException fnfe) {
    println("Error reading file: " + fnfe);
}
```

Compound Anagram 2

```
// 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 + ":");
        for (String ana : anagrams.get(sorted)) {
            println(ana);
        }
    } else {
        println("No anagrams for " + word + ".");
    }
    word = readLine("Type a word [Enter to quit]: ");
}
```