LEC 09

CSE 122

Nested Collections

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Talk to your neighbors:

What was the example you found for Creative Project 1 about how data set bias has caused harm to people?

Music: Miya's 23wi CSE 122 Playlist

Instructor Miya Natsuhara

TAs

Ayush	Atharva	Ernie	Ambika
Connor	Julia	Di	Elizabeth
Poojitha	Megana	Logan	Joe
Andrew A	Joey	Shivani	Jin
Andrew C	Eesha	Michelle	Ben
Jasmine	Lilian	Steven	Evelyn
Darel	Thomas	Kevin	Kent
Gabe	Leon	Ken	
Karen	Melissa	Vivek	
Colton	Audrey	Autumn	

Agenda

- Announcements
- Review/Finish: mostFrequentStart
- Recap: Nested Collections
- Practice: Search Engine
- Images Debrief

Announcements

- Programming Assignment 2 will be out today and due next Thursday
  - Start early! Fairly complex assignment
  - Due Thursday 2/9 @ 11:59 pm
- Assignment Resubmission Form posted.
  - Due Tuesday 2/7 @ 11:59 pm
- Quiz Retake Form posted for the 2/7 Retakes
  - Due Sunday 2/5 @ 11:59 pm
- Quiz 1 on Tuesday (2/7)
  - Reference Semantics, Stacks & Queues, 2D Arrays, Sets

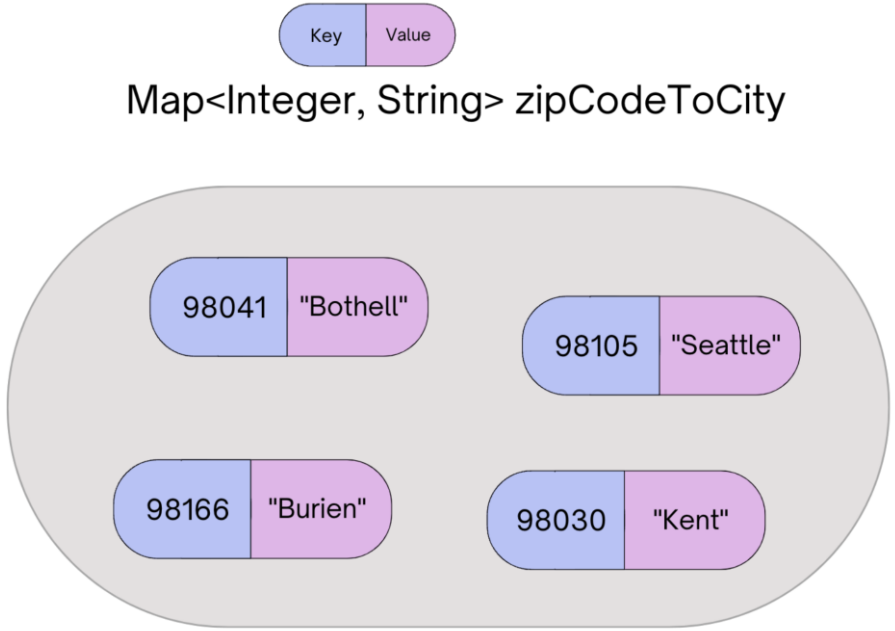
Agenda

- Announcements
- **Review/Finish: mostFrequentStart**
- Recap: Nested Collections
- Practice: Search Engine
- Images Debrief

(Review) Map ADT

- Data structure to map keys to values
  - Keys can be any* type; Keys are unique
  - Values can be any type
- Example: Mapping nucleotides to counts!
- Operations
  - `put(key, value)`: Associate key to value
    - Overwrites duplicate keys
  - `get(key)`: Get value for key
  - `remove(key)`: Remove key/value pair

Same as Python's dict

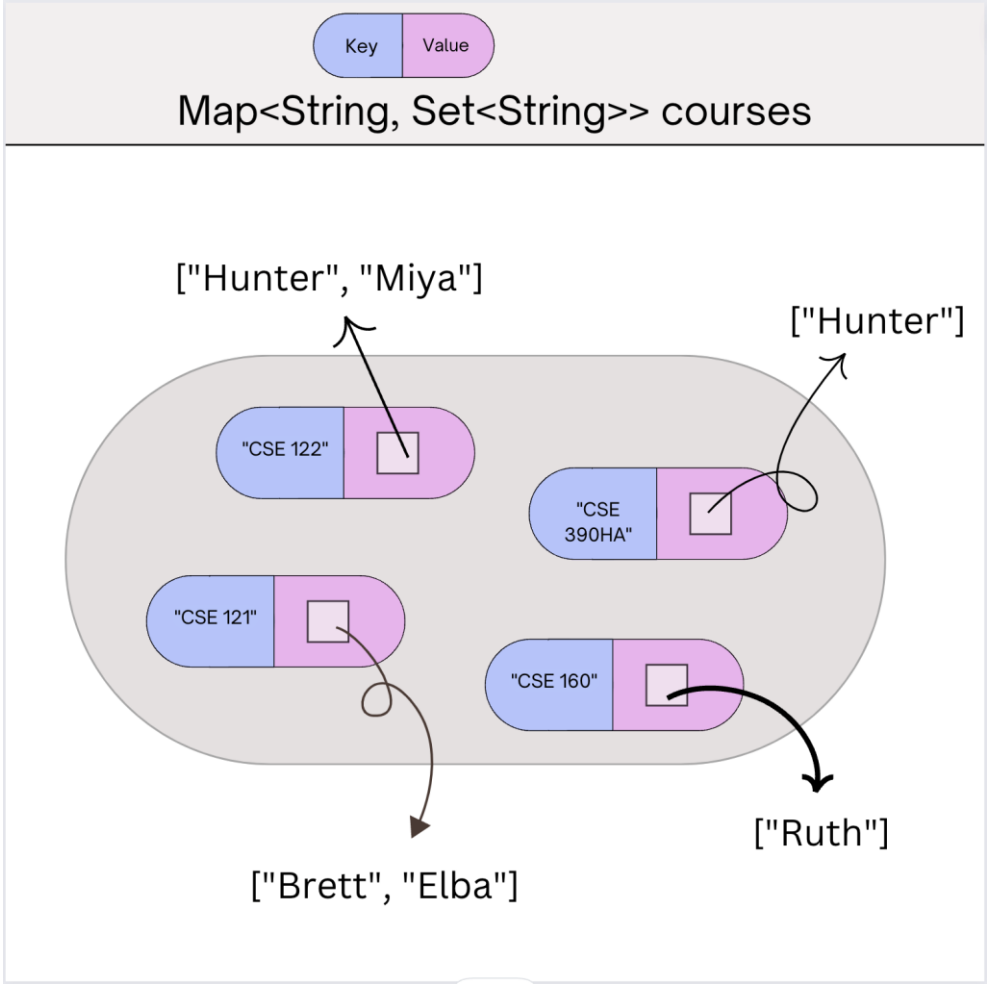


Agenda

- Announcements
- Review/Finish: mostFrequentStart
- **Recap: Nested Collections**
- Practice: Search Engine
- Images Debrief

(PCM) Nested Collections

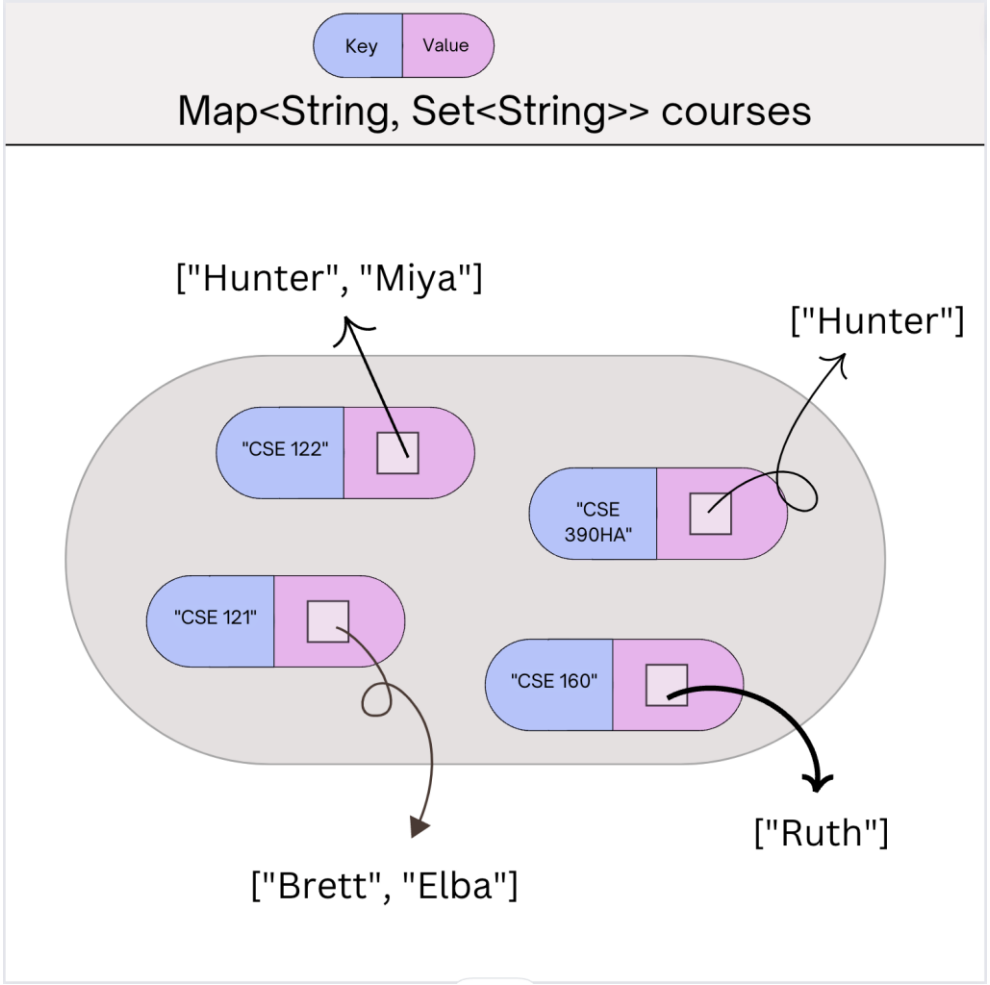
- The values inside a Map can be any type, including *data structures*
- Common examples:
  - Mapping Section -> Set of students in that section
  - Mapping Recipe -> Set of ingredients in that recipe
    - Or even Map<String, Map<String, Double>> for units!



(PCM) Updating Nested Collections

The value inside the Map is a reference to the data structure!

- Think carefully about number of references to a particular object




```
courses.put("CSE 123", new HashSet<String>());
courses.get("CSE 123").add("Kasey");

Set<String> cse123 = courses.get("CSE 123");
cse123.add("Brett");
```

Practice : Think

sli.do #cse122

Suppose map had the following state. What would its state be after running this code?

```
map: {"KeyA"=[1, 2], "KeyB"=[3], "KeyC"=[4, 5, 6]}
```

```
Set<Integer> nums = map.get("KeyA");
nums.add(7);
map.put("KeyB", nums);
map.get("KeyA").add(8);
map.get("KeyB").add(9);
```

A. `{"KeyA"=[1, 2], "KeyB"=[1, 2, 7], "KeyC"=[4, 5, 6]}`

B. `{"KeyA"=[1, 2, 8], "KeyB"=[1, 2, 7, 9], "KeyC"=[4, 5, 6]}`

C. `{"KeyA"=[1, 2, 7, 8], "KeyB"=[1, 2, 7, 9], "KeyC"=[4, 5, 6]}`

D. `{"KeyA"=[1, 2, 7, 8, 9], "KeyB"=[1, 2, 7, 8, 9], "KeyC"=[4, 5, 6]}`

Practice : Pair

sli.do #cse122

Suppose map had the following state. What would its state be after running this code?

```
map: {"KeyA"=[1, 2], "KeyB"=[3], "KeyC"=[4, 5, 6]}
```

```
Set<Integer> nums = map.get("KeyA");
nums.add(7);
map.put("KeyB", nums);
map.get("KeyA").add(8);
map.get("KeyB").add(9);
```

A. `{"KeyA"=[1, 2], "KeyB"=[1, 2, 7], "KeyC"=[4, 5, 6]}`

B. `{"KeyA"=[1, 2, 8], "KeyB"=[1, 2, 7, 9], "KeyC"=[4, 5, 6]}`

C. `{"KeyA"=[1, 2, 7, 8], "KeyB"=[1, 2, 7, 9], "KeyC"=[4, 5, 6]}`

D. `{"KeyA"=[1, 2, 7, 8, 9], "KeyB"=[1, 2, 7, 8, 9], "KeyC"=[4, 5, 6]}`

Agenda

- Announcements
- Review/Finish: mostFrequentStart
- Recap: Nested Collections
- **Practice: Search Engine**
- Images Debrief

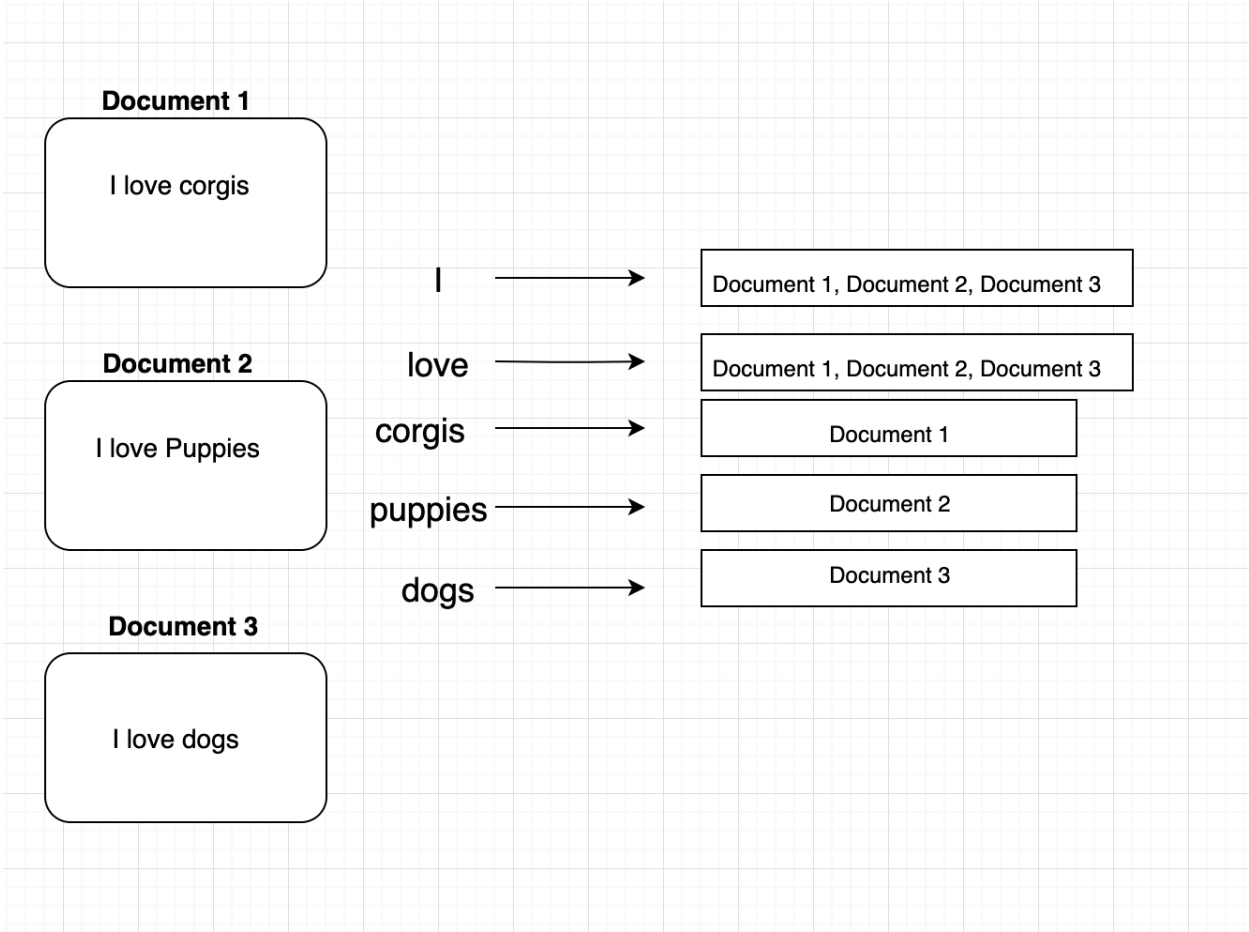
Background: Search Engines

- A **search engine** receives a **query** and returns a set of relevant **documents.** Examples: Google.com, Mac Finder, more.
  - Queries often can have more than one word, but we'll focus on single-word queries today.
- A search engine involves two main components
  - An **index** to efficiently find the set of documents for a query
    - Will focus on “single word queries” for today’s example
  - A **ranking algorithm** to order the documents from most to least relevant
    - Not the focus of this example

- Goal: *Precompute* a data structure that helps find the relevant documents for a given query

Inverted Index

- An **inverted index** is a Mapping from possible query words to the set of documents that contain that word
  - Answers the question: "What documents contain the word 'corgis'?"



Our task

I've written an incomplete program that will look through a set of files, build up an inverted index, then allow the user to enter a single-word query and prints out the results.

- It is *not necessary* for you to understand much of the existing starter code (particularly the details around files and directories).

We will focus on writing the code to build up the inverted index. There are a few TODOs:

✓ Determine the appropriate type for the inverted index data structure

✓ Implement the `addFileToIndex` method that adds the data from a given file to the inverted index

(Optional) Ranking Results

- There is no one right way to define which documents are "most relevant" There are approximations, but make decisions about what relevance means
- Idea 1: Documents that have more hits of the query should come first
  - Pro: Simple
  - Con: Favors longer documents (query: "the dogs" will favor long documents with lots of "the"s)
- Idea 2: Weight query terms based on their "uniqueness". Often use some sort of score for "Term Frequency – Inverse Document Frequency (TF-IDF)
  - Pro: Doesn't put much weight on common words like "the"
  - Cons: Complex, many choices in how to compute that yield pretty different rankings
- Idea 3: Much more! Most companies keep their ranking algorithms very very secret ☺