LEC 08

CSE 122

Maps

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Talk to your neighbors:
What is your least favorite root vegetable?

Music: Miya's 23wi CSE 122 Playlist

Instructor Miya Natsuhara

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

Lecture Outline

- **Announcements**
- Map Review
- Debrief PCM: Count Words
- Practice: joinRosters
- Practice: mostFrequentStart

Announcements

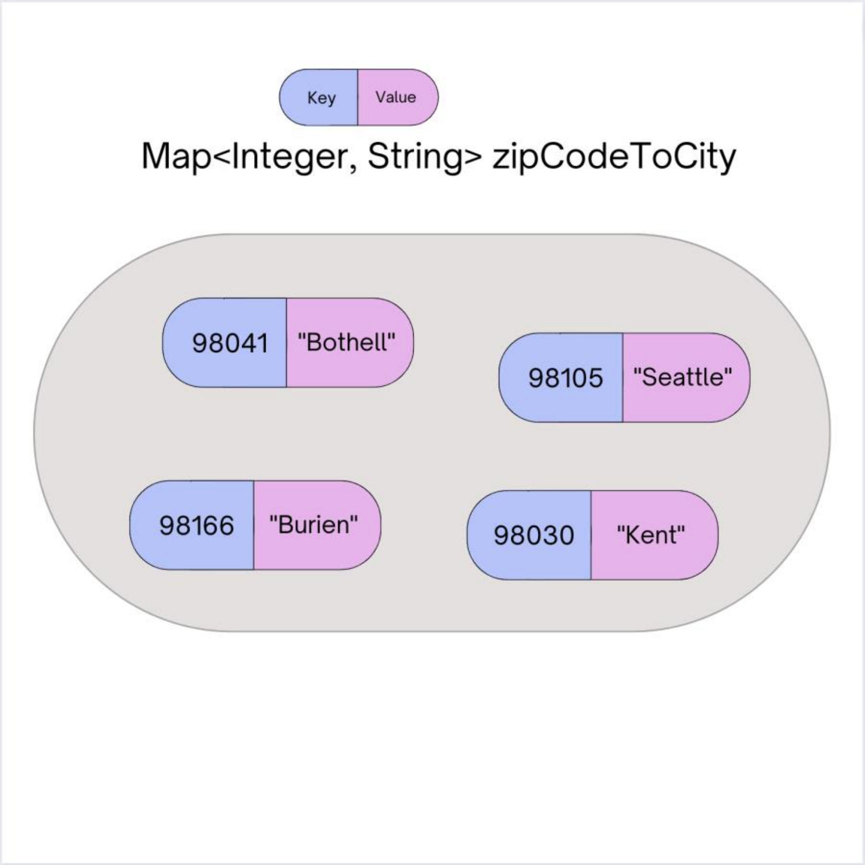
- Quiz 1 is Tuesday, Feb 7
- Retake and Resubmission forms for next week will be posted later today
- C1 due tomorrow (Thurs, Feb 2)
- P2 released Fri, Feb 3

Lecture Outline

- Announcements
- **Map Review** ◀
- Debrief PCM: Count Words
- Practice: joinRosters
- Practice: mostFrequentStart

(PCM) 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

(PCM) Programming with Maps

- Interface: `Map`
- Implementations: `TreeMap`, `HashMap`

```
// Making a Map
Map<String, String> favArtistToSong = new TreeMap<>();

// adding elements to the above Map
favArtistToSong.put("Steve Lacy", "Dark Red");
favArtistToSong.put("The Cranberries", "Linger");
favArtistToSong.put("Umi", "Bet");

// Getting a value for a key
String song = favArtistToSong.get("Umi");
System.out.println(song);
```

(PCM) Programming with Maps

Methods	Description
`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

(PCM) Map Implementations

- Our first data structures with marked differences in how their implementations behave
- One `Map` ADT / Interface
- Two `Map` implementations
  - `TreeMap` – Pretty fast, sorted keys
  - `HashMap` – Extremely fast, unsorted keys

```
Map<String, Integer> map1 = new TreeMap<>();
Map<String, Integer> map2 = new HashMap<>();
...
```

Practice : Think

sli.do #cse122

Select the method calls required to modify the given map m as follows:

Assume m's contents are

```
98030="Kent"
98178="Seattle"
98166="Burien"
98041="Bothell"
```

We want to modify m so that its contents are

```
98030="Kent"
98178="Tukwila"
98166="Burien"
98041="Bothell"
98101="Seattle"
98126="Seattle"
```

A. `m.put(98178, "Tukwila");`

B. `m.remove(98178);`

C. `m.put(98126, "Seattle");`

D. `m.get(98178, "Seattle");`

E. `m.put(98101, "Seattle");`

Practice : Pair

sli.do #cse122

Select the method calls required to modify the given map m as follows:

Assume m's contents are

```
        98030="Kent"
        98178="Seattle"
        98166="Burien"
        98041="Bothell"
```

We want to modify m so that its contents are

```
        98030="Kent"
        98178="Tukwila"
        98166="Burien"
        98041="Bothell"
        98101="Seattle"
        98126="Seattle"
```

```
A. m.put(98178, "Tukwila");

B. m.remove(98178);

C. m.put(98126, "Seattle");

D. m.get(98178, "Seattle");

E. m.put(98101, "Seattle");
```

Lecture Outline

- Announcements
- Map Review
- **Debrief PCM: Count Words**
- Practice: joinRosters
- Practice: mostFrequentStart

Lecture Outline

- Announcements
- Map Review
- Debrief PCM: Count Words
- **Practice: joinRosters** ◀
- Practice: mostFrequentStart

joinRosters

Write a method `joinRosters` that combines a Map from student name to quiz section, and a Map from TA name to quiz section and prints all pairs of students/TAs.

For example, if `studentSections` stores the following map:

```
{Alan=AC, Jerry=AB, Nina=AA, Sharon=AB, Steven=AB, Tanya=BA}
```

And `taSections` stores the following map

```
{Ben=BA, Melissa=AA, Andrew=AB, Atharva=AC}
```

```
AC: Alan – Atharva
AB: Jerry – Andrew
AA: Nina - Melissa
AB: Sharon – Andrew
AB: Steven – Andrew
BA: Tanya - Ben
```

Lecture Outline

- Announcements
- Map Review
- Debrief PCM: Count Words
- Practice: joinRosters
- **Practice: mostFrequentStart**

mostFrequentStart

Write a method called mostFrequentStart that takes a Set of words and does the following steps:

- Organizes words into “word families” based on which letter they start with
- Selects the largest “word family” as defined as the family with the most words in it
- Returns the starting letter of the largest word family (and if time, should update the Set of words to only have words from the selected family).

mostFrequentStart

For example, if the `Set` words stored the values

```
["hello", "goodbye", "library", "literary", "little", "repel"]
```

The word families produced would be

```
'h' -> 1 word ("hello")
'g' -> 1 word ("goodbye")
'l' -> 3 words ("library", "literary", "little")
'r' -> 1 word ("repel")
```

Since 'l' has the largest word family, we return 3 and modify the `Set` to only contain `Strings` starting with 'l'.