

LEC 07

**CSE 122****Maps**

Questions during Class?  
Raise hand or send here

sli.do #cse122

**BEFORE WE START*****Talk to your neighbors:***

*What was your last  
YouTube/Wikipedia/Google/ChatGPT rabbit  
hole?*

**Respond on sli.do!****Instructor:** Brett Wortzman

| <b>TAs:</b> | Archit   | Daniel   | Naomi    | Sreshta |
|-------------|----------|----------|----------|---------|
|             | Blake P. | David    | Neha     | Suyash  |
|             | Blake R. | Hanna    | Nicole   | Trisha  |
|             | Cady     | Isis     | Rio      | Wesley  |
|             | Carson   | Jun      | Rohan    | Yash    |
|             | Cole     | Landon   | Shreyank |         |
|             | Connor   | Michelle | Somya    |         |

Music: 🎸 [CSE 122 26au Lecture Tunes](#) 🎵

# Lecture Outline

- **Announcements** 
- Review: Maps
- Practice: joinRosters
- Practice: mostFrequentStart

# Announcements/Reminders

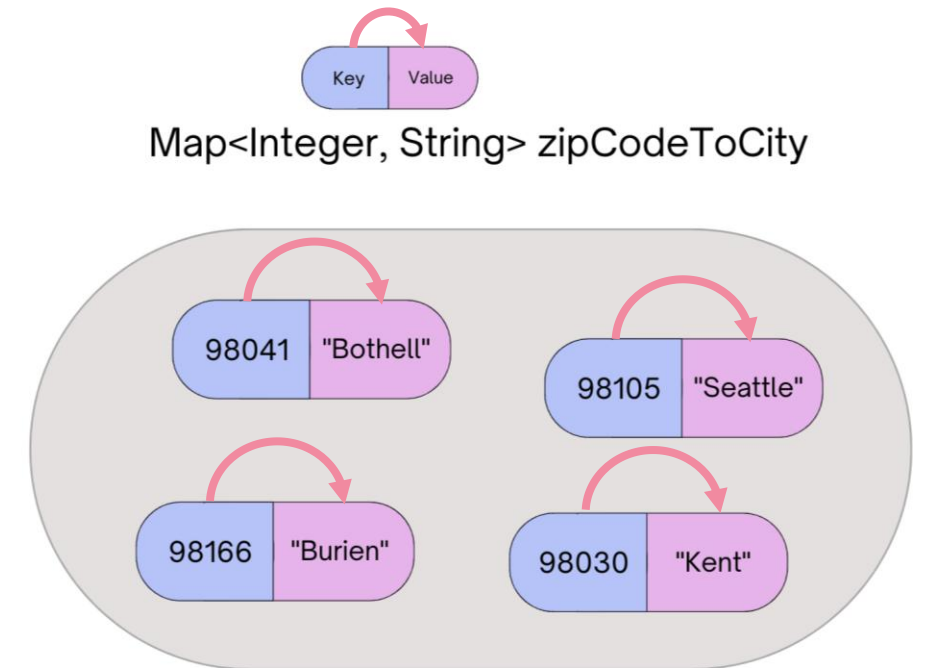
- P1 out tonight; due Thursday, Oct. 29
- PQ0 on Tuesday, Oct. 27 in section
  - [Policies posted on Ed](#)
  - Will cover material through Lesson 5 (up to stacks)
- CQ1 will begin Friday, Oct. 30
  - Scheduling will open early next week
- Reminder: ***you may not discuss the computer quiz with other students until the quiz window is closed***

# Lecture Outline

- Announcements
- **Review: Maps** ◀
- Practice: joinRosters
- Practice: mostFrequentStart

# (PCM) Maps (ADT)

- Data structure to map keys to values
  - Keys can be any type; Keys must be unique
  - Values can be any type
- Key operations:
  - `put(key, value)`: Associate key to value
    - 🚨 Overwrites existing keys 🚨
  - `get(key)`: Get value for key
  - `remove(key)`: Remove key/value pair

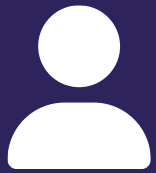


# (PCM) Maps in Java

- Map is an interface in Java
  - In `java.util` package, like `List`, `Queue`, `Stack`, `Set`
- Two implementations: `HashMap` and `TreeMap`
  - `HashMap`: Very fast! Implemented using a *hash table*
    - *Keys are stored in an unpredictable order*
  - `TreeMap`: Pretty fast! Implemented using a *binary search tree*
    - *Keys are stored in sorted order*

# (PCM) Map Methods

| Methods                                    | Description                                                                                                                      |
|--------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <code>put(<b>key</b>, <b>value</b>)</code> | adds a mapping from the given key to the given value;<br>if the key already exists, <u>replaces</u> its value with the given one |
| <code>get(<b>key</b>)</code>               | returns the value mapped to the given key ( <code>null</code> if not found)                                                      |
| <code>containsKey(<b>key</b>)</code>       | returns <code>true</code> if the map contains a mapping for the given key                                                        |
| <code>remove(<b>key</b>)</code>            | removes any existing mapping for the given key                                                                                   |
| <code>keySet()</code>                      | returns a set of all keys in the map                                                                                             |
| <code>values()</code>                      | returns a collection of all values in the map                                                                                    |
| <code>clear()</code>                       | removes <b>all</b> key/value pairs from the map                                                                                  |
| <code>size()</code>                        | returns the number of key/value pairs in the map                                                                                 |
| <code>isEmpty()</code>                     | returns <code>true</code> if the map's size is 0                                                                                 |
| <code>toString()</code>                    | returns a string such as <code>"{a=90, d=60, c=70}"</code>                                                                       |



# 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
- Review: Maps
- **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=AD, Jerry=BE, Yueying=BG, Sharon=BE, Steven=BE, Zewditu=AF}`

And `taSections` stores the following map

`{Blake=AF, Dani=BG, Naomi=BE, Wesley=AD}`

```
AD: Alan - Wesley
AF: Zewditu - Blake
BE: Jerry - Naomi
BE: Sharon - Naomi
BE: Steven - Naomi
BG: Yueying - Dani
```

# Lecture Outline

- Announcements
- Review: Maps
- 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 'l' and modify the Set to only contain Strings starting with 'l'.