LEC 06

CSE 122

Maps

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Slido vote & chat with your neighbors:

What is your favorite song from the smallest artist you listen to?

Instructor: Ivory & Colin

TAs: Dani
Rohan
Connor
Wesley
Yang

Lecture Outline (1/4)

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

Announcements/Reminders

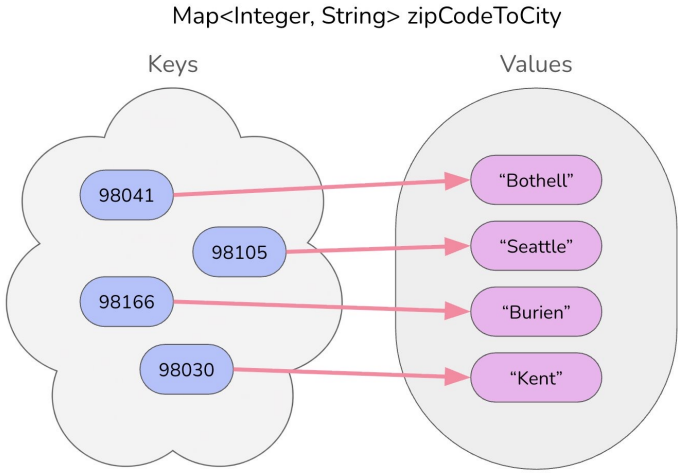
- Programming Assignment 1 (P1) releasing today due July 23rd 11:59pm!
- Resubmission Cycle (R1) due Tuesday July 21st 11:59pm

Lecture Outline (2/4)

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

Map ADT

- Data structure to map keys to values
  - Keys can be any* type; Keys must be unique
  - Values can be any type
- 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`

Programming with Maps in Java (1/2)

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

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

// adding elements to the above Map
favArtistToSong.put("PinkPantheress", "Stateside");
favArtistToSong.put("Rosalía", "La Perla");
favArtistToSong.put("Taylor Swift", "Dress");
favArtistToSong.put("Taylor Swift", "State of Grace"); // Overwrite!

// Getting a value for a key
String song = favArtistToSong.get("Taylor Swift");
System.out.println(song); // "State of Grace"
```

Programming with Maps in Java (2/2)

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

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 (3/4)

- Announcements
- Map Review
- **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, Yueying=AA, Sharon=AB, Steven=AB, Zewditu=BA}
```

And `taSections` stores the following map

```
{Blake=BA, Nicolae=AA, Sushma=AB, Dani=AC}
```

```
AC: Alan - Dani
AB: Jerry - Sushma
AB: Sharon - Sushma
AB: Steven - Sushma
AA: Yueying - Nicolae
BA: Zewditu - Blake
```

Lecture Outline (4/4)

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

mostFrequentStart (1/2)

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
- Finds the size of the largest "word families"
- Returns only the largest "word families" and the number of words in them

mostFrequentStart (2/2)

For example, if `words` stored the values

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

The word families produced would be

```
'h' -> 1 word ("hello")
'g' -> 2 words ("goodbye", "guile")
'l' -> 2 words ("library", "literary")
'r' -> 1 word ("repel")
```

Since the "word families" with the most words are `'g'` and `'l'`, each with 2 words, `mostFrequentStart(words)` would return the map

```
['g'=2,'l'=2]
```