LEC 09

CSE 122

Maps

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Talk to your neighbors:

What is your favorite form of potato?

Music: 122 24au Lecture Tunes

Instructors: Miya Natsuhara and Elba Garza

TAs:			
Ayush	Heon	Harshitha	Aishah
Andrew	Izak	Marcus	Ben
Logan	Colin	Carson	Ivory
Kyle	Jessica	Jack	Cady
Maggie	Shivani	Connor	Diya
Nicole H	Ken	Cora	Katharine
Caleb	Mia	Hannah	
Nicole W	Ashley	Leo	
Jacob	Chaafen	Anya	

Lecture Outline

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

Announcements

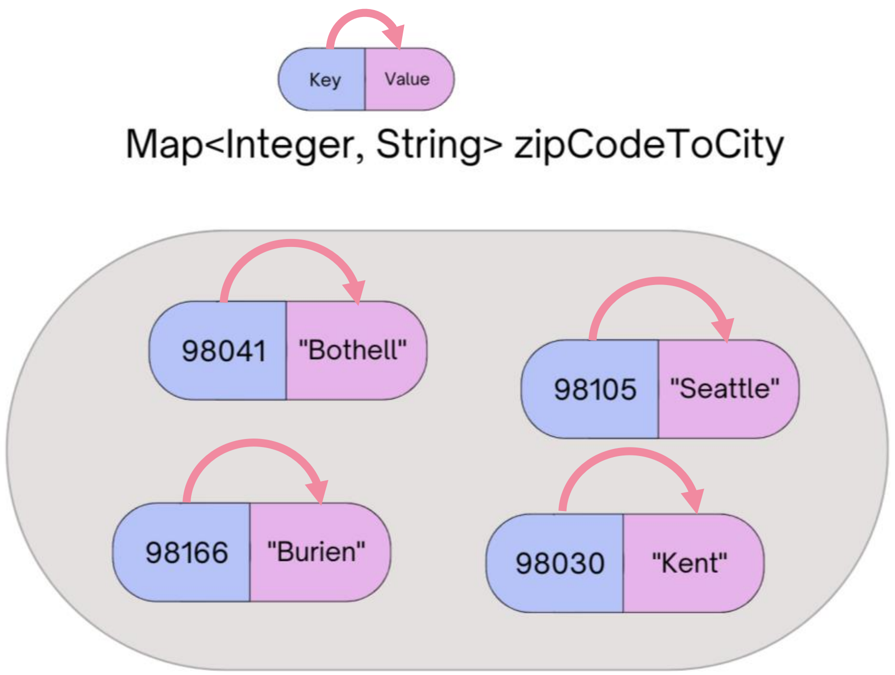
- Reminder: Quiz 1 is Tuesday, November 5
- Resubmission Cycle 2 (R2) form open now
  - Due Tuesday, Oct 29 by 11:59 PM
  - Eligible Assignments: **C0**, P0, C1
- Programming Assignment 2 (P2) released later today!
  - Due Thursday, **Nov 7** by 11:59pm

Lecture Outline

- Announcements
- **Map Review**
- Debrief PCM: Count Words
- 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
- Example: Mapping nucleotides to counts in C0!
- 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

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

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

// adding elements to the above Map
favArtistToSong.put("Iron Maiden", "Wasted Years");
favArtistToSong.put("Foxes", "Body Talk");
favArtistToSong.put("Vampire Weekend", "Campus");

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

Programming with Maps in Java

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

Map Implementations

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

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, Yueying=AA, Sharon=AB, Steven=AB, Zewditu=BA}
```

And `taSections` stores the following map

```
{Ayush=BA, Marcus=AA, Aishah=AB, Chaafen=AC}
```

```
AC: Alan – Chaafen
AB: Jerry – Aishah
AB: Sharon – Aishah
AB: Steven – Aishah
AA: Yueying – Marcus
BA: Zewditu – Ayush
```

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 `'l'` and modify the `Set` to only contain `Strings` starting with 'l'.