LEC 08

CSE 122

Maps

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Talk to your neighbors:

What's your favorite movie genre?

Instructor Melissa Lin

TAs Poojitha Arangam, Darel Gunawan, Colton Harris, Atharva Kashyap, Eesha Kunisetty, Audrey Lin, Di Mao, Steven Nguyen, Ben Wang, Jaylyn Zhang

Lecture Outline

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

Announcements

- Quiz 0 grades were released
  - Regrade Request form
- C1 due tomorrow
- P2 released Friday
- Quiz 1 is **Monday, July 24**
  - Topics: Reference Semantics, 2D Arrays, Sets, Maps, Nested Collections

Lecture Outline

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

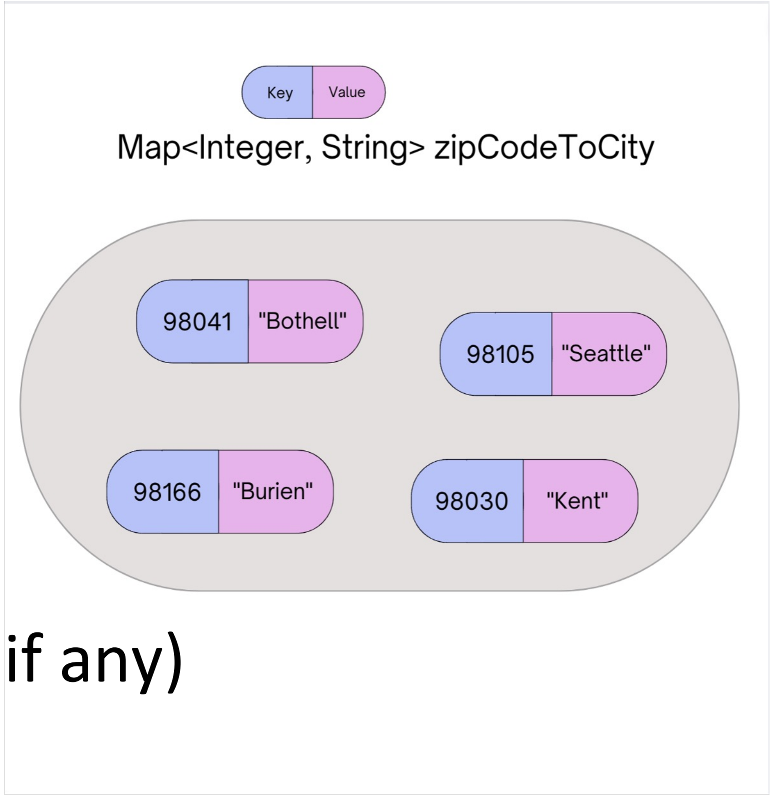
(PCM) Map - What is it good for?

What is it?

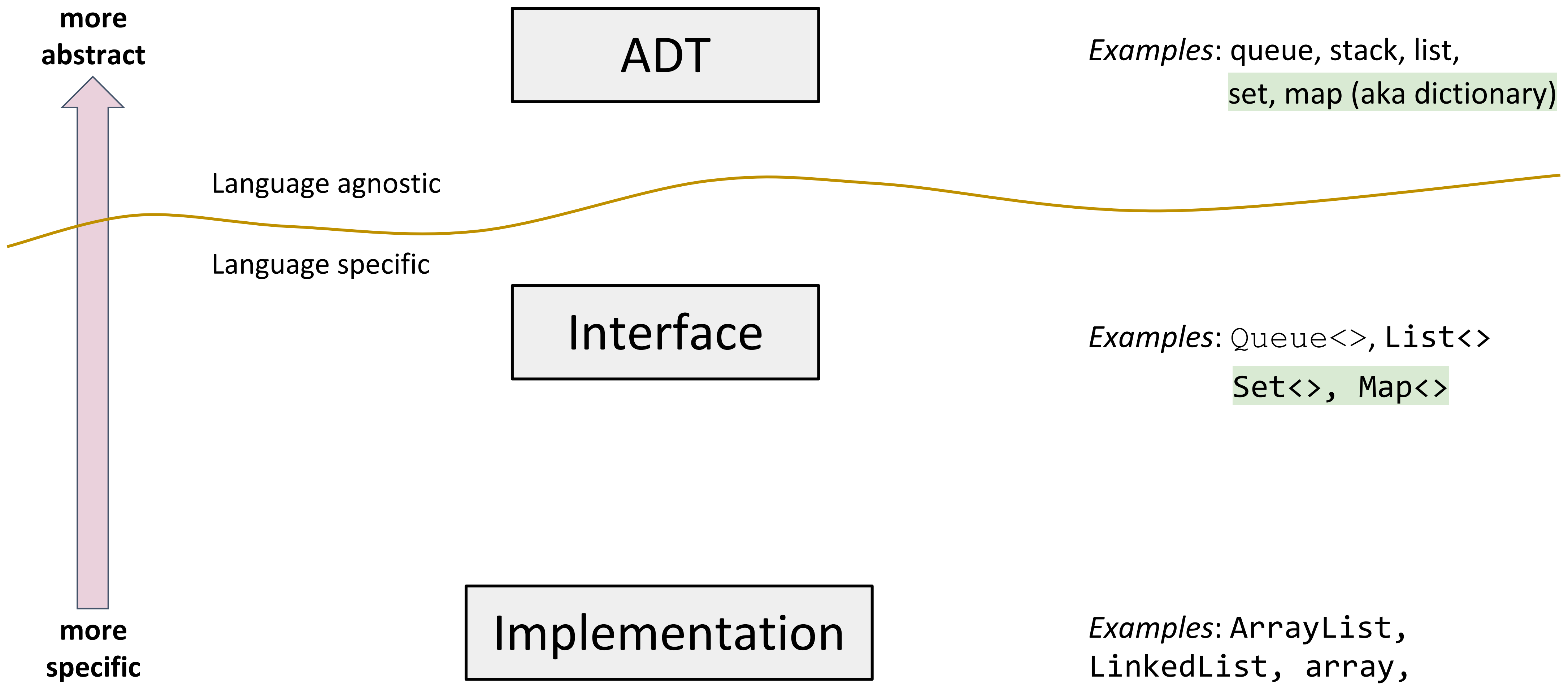
- Keeps associations between *unique* keys and (non-unique) values
- All *keys* are one type. All *values* are one type
  - But a *keys* might be a different type from *values*
- Dynamically sized

What is Map particularly good at?

- `put(key, value)` - associates key with a value
- `get(key)` - returns the value associated with a key (if any)
- `remove(key)` – remove key/value pair



(PCM) Abstract Data Types



Examples: queue, stack, list, set, map (aka dictionary)

Examples: `Queue<>`, `List<>` `Set<>`, `Map<>`

Examples: `ArrayList`, `LinkedList`, `array`, `Stack`, `2D array`, `HashSet`, `TreeSet`, `HashMap`, `TreeMap`

(PCM) Maps in Java

- Interface: `Map`
- Implementations: `TreeMap`, `HashMap`
  - `TreeMap` – Pretty fast, sorted keys
  - `HashMap` – Extremely fast, unsorted keys

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

(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
`keySet()`	returns a set of all keys in the map
`values()`	returns a collection of all values in the map
`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}"`

(PCM) Programming with Maps

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

// adding elements to the above Map
musicalToFavSong.put(”Hamilton", ”Wait for It");
musicalToFavSong.put(”Les Miserables", ”Stars");
musicalToFavSong.put(”Waitress", ”She Used to Be Mine");

// Getting a value for a key
String song = musicalToFavSong.get(”Hamilton");
System.out.println(song); // "Wait for It”
```

Practice : Think

sli.do #cse122

What does the map store after the following code?

```
Map<String, String> musicalToFavSong = new TreeMap<>();

musicalToFavSong.put("Hamilton", "Non-Stop");
musicalToFavSong.put("Hamilton", "Wait for It");
musicalToFavSong.put("Les Miserables", "Stars");
musicalToFavSong.put("Waitress", "She Used to Be Mine");
musicalToFavSong.remove(”Les Miserables");
musicalToFavSong.put(”Hairspray", ”Without Love");
```

```
Hamilton -> Non-Stop
Hamilton -> Wait for It
Waitress -> She Used to Be Mine
Hairspray -> Without Love
```

A

```
Waitress -> She Used to Be Mine
Hamilton -> Wait for It
Hairspray -> Without Love
```

B

```
Hairspray -> Without Love
Hamilton -> Wait for It
Waitress -> She Used to Be Mine
```

C

```
Error
```

D

Practice : Pair

sli.do #cse122

What does the map store after the following code?

```
Map<String, String> musicalToFavSong = new TreeMap<>();

musicalToFavSong.put("Hamilton", "Non-Stop");
musicalToFavSong.put("Hamilton", "Wait for It");
musicalToFavSong.put("Les Miserables", "Stars");
musicalToFavSong.put("Waitress", "She Used to Be Mine");
musicalToFavSong.remove(”Les Miserables");
musicalToFavSong.put(”Hairspray", ”Without Love");
```

```
Error
```

```
Hamilton -> Non-Stop
Hamilton -> Wait for It
Waitress -> She Used to Be Mine
Hairspray -> Without Love
```

A

```
Waitress -> She Used to Be Mine
Hamilton -> Wait for It
Hairspray -> Without Love
```

B

```
Hairspray -> Without Love
Hamilton -> Wait for It
Waitress -> She Used to Be Mine
```

C

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=AD, Jerry=AB, Nina=AA, Sharon=AB, Tanya=AD}
```

And `taSections` stores the following map

```
{Jaylyn=AB, Darel=AD, Atharva=AA}
```

```
AD: Alan - Darel
AB: Jerry - Jaylyn
AA: Nina - Atharva
AB: Sharon - Jaylyn
AD: Tanya - Darel
```

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'.