Maps

Talk to your neighbors:
If you had to choose one meal to eat repetitively for the rest of your life, what would it be?
(Mine’s definitely oatmeal.)

Music: The Middle - Jimmy Eat World
Lecture Outline

• Announcements

• Map Review

• Debrief PCM: Count Words

• Practice: joinRosters

• Practice: mostFrequentStart
Announcements

• Quiz 1 is Tuesday, October 31\textsuperscript{st}
• Resubmission Cycle 2 (R2) form opens up tomorrow!
  - Due Tuesday, October 31\textsuperscript{st} by 11:59 PM
• Creative Project 1 (C1) due tomorrow by 11:59 PM!
• Programming Assignment 2 (P2) released Friday
  - Due Thursday, \textbf{November 9\textsuperscript{th}} by 11:59 PM
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 P0!

• 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

```java
// 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

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

```java
Map<String, Integer> map1 = new TreeMap<>();
Map<String, Integer> map2 = new HashMap<>();
...
```
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");
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"
- 98126 = "Seattle"
- 98101 = "Seattle"

A. \( m.\text{put}(98178, \text{"Tukwila"}) \);

B. \( m.\text{remove}(98178) \);

C. \( m.\text{put}(98126, \text{"Seattle"}) \);

D. \( m.\text{get}(98178, \text{"Seattle"}) \);

E. \( m.\text{put}(98101, \text{"Seattle"}) \);
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:

\{Marcus=BA, Arthur=AA, Atharva=AB, Rucha=AC\}

AC: Alan - Rucha
AB: Jerry - Atharva
AB: Sharon - Atharva
AB: Steven - Atharva
AA: Yueying - Arthur
BA: Zewditu - Marcus
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'.