Logistics

HW 3
- Will be up in a few days
- Mashup of hash table, array list, linked list

Goals Today
- Maps and Sets
- Tree beginnings

Upcoming
- Midterms back on Wed
- Midterm Feedback Wed
Map and Set

Set: HashSet and TreeSet
- Collection of distinct objects
- Supports add(x), remove(x), contains(x), sometimes get(x)
- x is either in the set or not in the set

Map: HashMap and TreeMap
- (key,value) pairs
- Each key has exactly one value
- Insert value into a map according to its key
- Same key maps to same "place" in the data structure
- Supports put(k,v), get(k), remove(k), contains(k)
Examples of Sets and Maps

A data type

```java
class Student{
    String name;
    int gNumber;
}
```

A set of students

Contents

- {Kyle, 1234}
- {Stan, 4321}
- {Eric, 2486}
- {Kenny, 1313}
- {Stan, 1357}

A map of IDs to students

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>{Kyle, 1234}</td>
</tr>
<tr>
<td>4321</td>
<td>{Stan, 4321}</td>
</tr>
<tr>
<td>2486</td>
<td>{Eric, 2486}</td>
</tr>
<tr>
<td>1313</td>
<td>{Kenny, 1313}</td>
</tr>
<tr>
<td>1357</td>
<td>{Stan, 1357}</td>
</tr>
</tbody>
</table>

A map of Students to Majors

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Kyle, 1234}</td>
<td>World Religions</td>
</tr>
<tr>
<td>{Stan, 4321}</td>
<td>Geology</td>
</tr>
<tr>
<td>{Eric, 2486}</td>
<td>Nutrition</td>
</tr>
<tr>
<td>{Kenny, 1313}</td>
<td>Mortuary Sciences</td>
</tr>
<tr>
<td>{Stan, 1357}</td>
<td>Genetics/Cloning</td>
</tr>
</tbody>
</table>
Questionable Sets and Maps

A set of majors?

Contents:
World Religions
Geology
Nutrition
Mortuary Sciences
Genetics/Cloning

A set of names?

Contents:
Kyle
Stan
Eric
Kenny
Stan

A map of IDs to names?

Key   Value
1234  -> Kyle
4321  -> Stan
2486  -> Eric
1313  -> Kenny
1357  -> Stan

A map of names to IDs?

Key   Value
Kyle  -> 1234
Stan  -> 4321
Eric  -> 2486
Kenny -> 1313
Stan  -> 1357
Array Analogy

Arrays and ArrayList are like a Map where
- Keys are integers: store at array index
- Values are the objects at those indices

A Set of Integers is naturally represented as an array of booleans
- Represent sets of Integers 0 to 10
- Use arrays of size 11
- The set \{1, 8, 9\} is the array

```java
boolean set1[] = new boolean[]{
    false, true, false, false, false, false,
    // 1
    false, false, true, true, false,
    // 8 9
};
```

More efficient with BitSet if you’re willing...
General Observations

Set

- A set must guarantee uniqueness of elements
- Typical approach is during add(x), check contains(x) and don’t add duplicates but there are other approaches
- Efficient implementation of contains(x) and get(x) becomes important for sets

Map

- The collection so keys is a set - each key must be unique
- contains(k)/get(k) important - make them efficient
- Collection of values is not unique
- Usually not efficient to look up whether a given value is present
- Collection of (key,value) pairs is unique due to keys being unique
General Implementations

How would you implement a Set<T>?
▶ Using an ArrayList?
▶ Using an LinkedList?
▶ Using a hash table?

How would you implement a Map<K,V>?
▶ Using an ArrayList?
▶ Using an LinkedList?
▶ Using a hash table?
Have Set, Build Map

Q: If I have Set, how would I build Map?

Given: SimpleSet

- Collection of distinct objects
- Uniqueness determined by equals() method
- Operations add(x), get(x), remove(x), contains(x)
- SimpleSet implementation may be based on arrays, hash tables, trees, linked lists...
  you don’t know

Build: SimpleMap

- Set of (key,value) pairs
- Compare pairs only on whether their key is equal
  (a,x) == (a,x)
  (a,x) == (a,y)
  (a,x) != (b,y)
  (a,x) != (b,x)
- Use the set to ensure no redundant keys enter
- Implement put(k,v), get(k), remove(k), contains(k)
Trick 1: Use an internal class

```java
public class MapFromSet<K, V>{
    // Trick: Use a nested class
    // Class to carry around (key,val) pairs
    public static class KeyVal<K, V>{
        public K key; public V value;
        public KeyVal(K key, V value){
            this.key = key; this.value = value;
        }
        // Required for any set to work
        // Compare only based on key
        public boolean equals(Object o);
        // Required for HashSet to work right
        public int hashCode();
        // Required for TreeSet to work
        public int compareTo(KeyVal<K,V> kv);
    }
}
```
Trick 2: Use a set of the key/val pairs

Prototypes

```java
public interface SimpleSet<T> {
    boolean contains(T x);
    boolean add(T x);
    boolean remove(T x);
    T get(T x);
}

public class MapFromSet<K, V> {
    private SimpleSet<KeyVal<K, V>> theSet;

    // Implement these using theSet
    public MapFromSet();
    public void put(K key, V value);
    public void remove(K key);
    public boolean contains(K key);
    public V get(K key);
}
```

Exercise

Implement the put(), remove(), contains(), get() methods