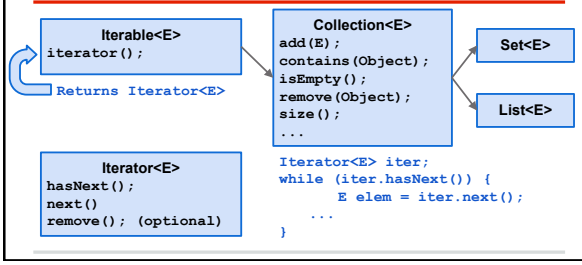
Recitation 8

Iterable, Iterator, and Examples

Iterable and Iterator




```
Iterator<E> iter;
while (iter.hasNext()) {
        E elem = iter.next();
    ...
}
```


Iterator.remove()

We will not implement `remove`, but it's still useful to know about.

Example of filtering all negative integers from a LinkedList collection:

```
Iterator<Integer> iter = linkedList.iterator();
while (iter.hasNext()) {
        Integer i = iter.next();
    if (i < 0)
        iter.remove();
}
```

Removes last thing returned from iterator

Note: ***Don't use for data structures that have O(n) remove like ArrayLists!***

Syntactic sugar!

```
LinkedList<String> list;
Iterator<String> iter =
list.iterator();
while (iter.hasNext()) {
      String s = s.next();
      // use s
}
```

Any object that implements `Iterable<E>` will be able to use the for-each loop.

```
LinkedList<String> list;
for (String s : list) {
      // use s
}
```

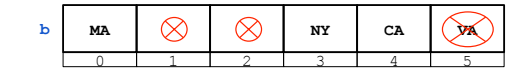

Last week: Linear probing

Three possible entries:

1. **null**
2. A HashEntry object with isInSet **false**
3. A HashEntry object with isInSet **true**

Iterate over →

```
HashEntry<String>[] b;
```



HashSet<T>

```
public class HashSet<T> {
    private HashEntry<T>[] b;
    private int size = 0; //number of active entries in set
    public boolean add(T x) { ... }
    public boolean contains(Object x) { ... }
    public boolean remove(Object x) { ... }

    private class HashSetIterator implements Iterator<T> {
        public boolean hasNext() { ... }
        public T next() { ... }
        public void remove() { ... }
    }
}
```

Has access to field `size`

HashSetIterator

```
/** An instance is an Iterator of this HashSet */
private class HashSetIterator implements Iterator<T> {
   // all elements in b[0..pos] have been enumerated
   private int pos = -1;
   // number of elements that have been enumerated
   private int enumerated = 0;

   /** = "there is another element to enumerate". */
   public boolean hasNext() {
       return enumerated != size;
   }
   // continued on next slide
}
```

field size of class HashSet

HashSetIterator

```
/** An instance is an Iterator of this HashSet */
private class HashSetIterator implements Iterator<T> {
    //continued from previous slide
    public T next() {
            if (!hasNext()) throw new
NoSuchElementException();
       pos = pos+1;
            while (b[pos] == null || !b[pos].isInSet) {
          pos = pos+1;
            }
            enumerated = enumerated + 1;
            return b[pos].element;
       }
}
```


HashSet<T>

```
public class HashSet<T> implements Iterable<T> {
    private HashEntry<T>[] b;
    private int size = 0;
    public boolean add(T x) { ... }
    public boolean contains(Object x) { ... }
    public boolean remove(Object x) { ... }
    public Iterator<T> iterator() {
           return new HashSetIterator();
    }
    private class HashSetIterator implements Iterator<T> {
          ...
    }
}
```


Don't change the set while iterating!

```
HashSet<Integer> hs = new HashSet<Integer>();
// Add a bunch of strings to hs;
for (Integer k : hs) {
      hs.add(  );
}
```

This may change array `b` and int field `size`. May cause rehash. `hs`'s class invariant (meanings of `hs.pos` and `iter.enumerated`) no longer holds. Will get `ConcurrentModificationException`.

Nested Classes: class within class

```
public class HashSet<T> implements Iterable<T>;
   public boolean add(T x)
   …
   public Iterator<T> iterator() {}
   private class HashSetIterator implements Iterator<T> {}
   private static class HashEntry<T> {}
}
```

Inner Class:

- When instances are created, they live within outer class object
- `HashSetIterator` needs to access `b` and `size`

Nested Static Class:

- Do **not** live within outer class object
- `HashEntry` objects do not reference `HashSet` fields or instance methods.

iter.next() vs. list.get(i)

```
for (Integer i : linkedList) {
   System.out.println(i);
}
```

O(n)

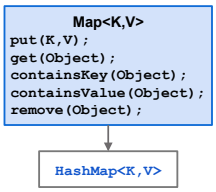
Are these equivalent time complexity? No.

```
for (int i = 0; i < linkedList.size(); i++) {
      System.out.println(linkedList.get(i));
}
```

O(n²)

Data Structure Problems

Review: HashMap



HashMap
- Key-Value pair relationships
- Uses hashing like HashSet
- **expected**: O(1) insert and lookup
- **worst case**: O(n) insert and lookup

Review: HashMap

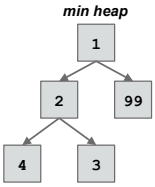
HashMap<String,Integer>

to	2
be	2
or	1
not	1
that	1
is	1
the	1
question	1

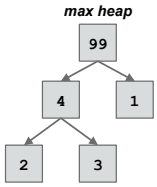
Example of **HashMap**: frequencies of words in a document

"To be or not to be that is the question"

Review: Binary heap



PriorityQueue
- Maintains max or min of collection
- Follows *heap order invariant* at every level
- Always balanced!
- **worst case**:
  O(log n) insert
  O(log n) update
  O(1) peek
  O(log n) removal



Least Recently Used (LRU) Cache

```
Supported Operations all in O(1) time:
      add(E element)
      update(E element)
```

A cache (container) should contain at most N elements. When the N+1 element tries to get added, the element that was least recently used gets evicted.
The elements are immutable and distinct.

Finding top k elements in array

Given an array `b`, print the top k elements.
(Order not important)

100	7	3	1	5	10
0	1	2	3	4	5

Where k = 3: `100,7,10`