Lecture 14: Ordered Structures

CS 62
Spring 2015
Kim Bruce & America Chambers

Stacks, Queues, & ....

- Steque:
  - Add and remove from one end. Only add from other.
- java.util.Deque: Double-Ended Queue
  - Can add or remove from either end.
  - Resizable array implementation
  - Faster than Stack when used as stack, faster than LinkedList (doubly-linked) when used as queue.

Sorting

- Examples earlier used doubles or Strings
- Work with any class with ordering operator

```
interface Comparable[T] {
  int compareTo(T other);
}
```

- compare returns negative if self < other,
  0 if equal,
  positive if self > other

Classes with ordering

- Classes with ordering written as:
  - class C implements Comparable<C>
  - Means must have method
    - public int compareTo(C other) {...}
- Collections class contains
  - public static <T extends Comparable<T>> void sort(List<T> list)
  - Implemented as optimized mergesort
  - What if no natural order or want different order?

Ordered Association

- Earlier talked about:
  - public class Association<K,V> {
  -     protected K theKey;   // key of the key-value pair
  -     protected V theValue; // value of key-value pair
- Now want associations where can order by key

ComparableAssociation

```
public class ComparableAssociation<K extends Comparable<K>,V>
    extends Association<K,V>
    implements Comparable<ComparableAssociation<K,V>>{

   public ComparableAssociation(K key, V value) {
        super(key,value);
   }

   public int compareTo(ComparableAssociation<K,V> that) {
        return this.getKey().compareTo(that.getKey());
   }
...
}
```

Now can use in sort!

Comparators

- Can include own ordering function:
  - java.util.Comparator interface in Java:

```
public interface Comparator<T> {
      // returns negative if o1 < o2,
      // 0 if o1 ==o2,
      // positive if o1 > o2
      // in the ordering being supported by object.
      int compare(T o1, T o2);
}
```

Way of Comparing Strings

```
public class TrimComparator
                implements Comparator<String> {
   // pre: o1 and o2 are string
   // post: returns negative, zero, or positive
   //      depending on relation
   //      between trimmed parameters.
   public int compare(String s1, String s2) {
       String s1trim = s1.trim();
       String s2trim = s2.trim();
       return s1trim.compareTo(s2trim);
   }
}
```

Using Comparators

- Classes supporting sort or other operations using comparisons generally have two versions:
- From Collections class:
  - `static <T extends Comparable<T>> void sort(List<T> list)`
  - `static <T> void sort(List<T> list, Comparator<T> c)`
  - *Actual types a bit more general (and complex).*

```
Collections.sort(data,new TrimComparator());
```

Using Lambda Expressions

- In Java 8, can use lambda expression rather than Comparator method:

```
Collections.sort(data,
   (s1,s2) -> {
      String s1trim = s1.trim();
      String s2trim = s2.trim();
      return s1trim.compareTo(s2trim);
   });
```

See TestComparator.java

Ordered Structures

- See OrderedArrayList
  - esp. locate method which does binary search
  - Also OrderedList with singly-linked list implementation
- See text for discussion of operations on ordered structures
  - E.g., find, add, etc.