Lecture 22: Ordered Structures

CS 62
Spring 2018
Alexandra Papoutsaki & William Devanny
Comparing Objects

• To compare references
  \( \text{o1}==\text{o2} \) or \( \text{o1}!=\text{o2} \):
  • Compare to see if reference is \text{null}
  • Compare to see if pointing to same object

• To compare object equality
  \( \text{o1}.\text{equals}(...) \)
  • Automatically inherited from all classes
  • If not overridden same as \( == \)
  • Already implemented in standard Java classes
  • Has to be overridden to perform intelligent comparisons for your own classes
Sorting

• Examples earlier used ints, doubles or Strings
• Work with any class with ordering operator

```java
interface Comparable<T> {
    int compareTo(T other);
}
```

• compareTo returns:
  • Negative if self < other
  • 0 if equal
  • Positive if self > other

• Throws a NullPointerException if comparing to null
  • e.equals(null) returns false. If e is null then NullPointerException
Override `equals()`

```java
public int compareTo(Ratio that) {
    return this.getNumerator() * that.getDenominator() -
        that.getNumerator() * this.getDenominator();
}

public boolean equals(Object that) {
    return compareTo((Ratio)that) == 0;
}
```

Notice that need to cast to `Ratio`, as `equals` requires an `Object`. Need to also implement `hashCode()` (later)
Classes with ordering

• Classes with ordering written as:
  • `class C implements Comparable<C>`
  • 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

• 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
Comparable Association

```java
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 that) {
        return this.getKey().compareTo(that.getKey());
    }

    ...
}
```

Now we can use sort!
Comparators

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

```java
public interface Comparator {
    // 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);
}
```
Example: how to compare strings

```java
public class TrimComparator implements Comparator<String> {
    /*
    * pre: s1 and s2 are strings
    * 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 void sort(List<T> list, Comparator<T> c)
  • Actual types a bit more general (and complex). Collections.sort(data, new TrimComparator());
  • If you try to sort a collection whose elements do not implement Comparable or cannot be compared with the Comparator, it will throw a ClassCastException
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.java`, especially `locate` method which does binary search
- Also `OrderedList.java` with singly-linked list implementation
- See text for discussion of operations on ordered structures
  - E.g., find, add, etc.