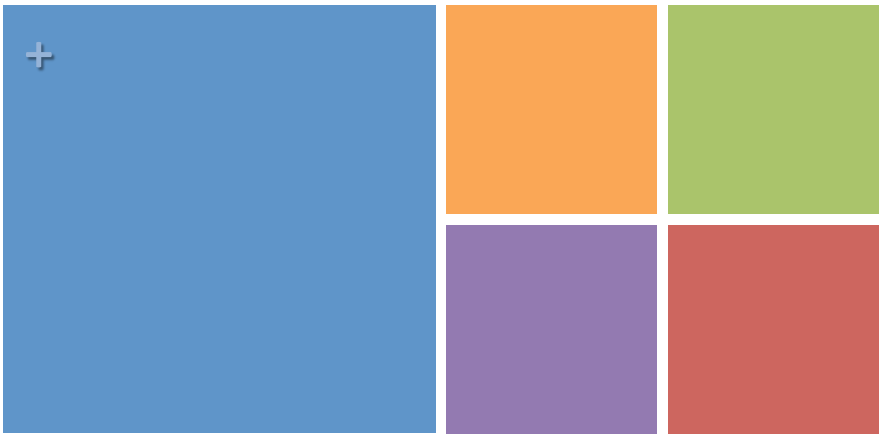


Lecture 14: Comparable + Comparator

and Ordered Structures

Today

- Reading
  - JS Ch. 11 (Ordered Structures)
- Objectives
  - Sorting with the Comparable Interface
  - Comparator Interface

Last Time

- Stacks and Queues (and Steques)
- These are *linear structures*
  - values are removed based upon the order in which they are added
- Inheritance simplifies implementation

Ordered Structures

- Structures that store data in *sorted* order
  - Make searching and sorting easier
  - Can't control the location of a data item that is added

3	5	11	13

```
add(7)
```

3	5	7	11	13

Sorting and comparison

```
void sort(int[] array) {
... if (array[i] < array[j]) ...
}

void sort(String[] array) {
... if (array[i].compareTo(array[j]) < 0) ...
}

void sort(Student[] array) {
...
if (array[i].getLastName().compareTo(array[j].getLastName()) < 0)
...
}
```

Ordering objects

- Could we write just one sort method…?
- Generalize to any class with ordering operator

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

$$x.\text{compareTo}(y) = \begin{cases} <0 & \text{if } x < y \\ 0 & \text{if } x = y \\ >0 & \text{if } x > y \end{cases}$$

Comparable Students

```
public class Student implements Comparable<Student> {
  protected String lastName;
  protected String firstName;
  ...
  // Order by "Last, First"
  public int compareTo(Student other) {
    int lastComp = lastName.compareTo(other.lastName);
    if (lastComp != 0) {
      return lastComp;
    } else {
      return firstName.compareTo(other.firstName);
    }
  }
}
```

String implements Comparable<String>

Sorting Comparable objects

```
public static <T extends Comparable<T>>
void sort(List<T> list) {
   ...
   if (list.get(i).compareTo(list.get(j)) < 0)
   ...
}
```

- Works for list of **any** class T of objects that implements Comparable<T>.

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 we want Associations where we 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 we can use in sort!

Can we make Objects Comparable by default?

```
publi   lass Object i   ents Comparable<Object> {
  public   t compar   Object other) {
    if (th   =   ) {
      return
    } else
      ???   ????
    }
  }
}
```

no "natural" order of Objects

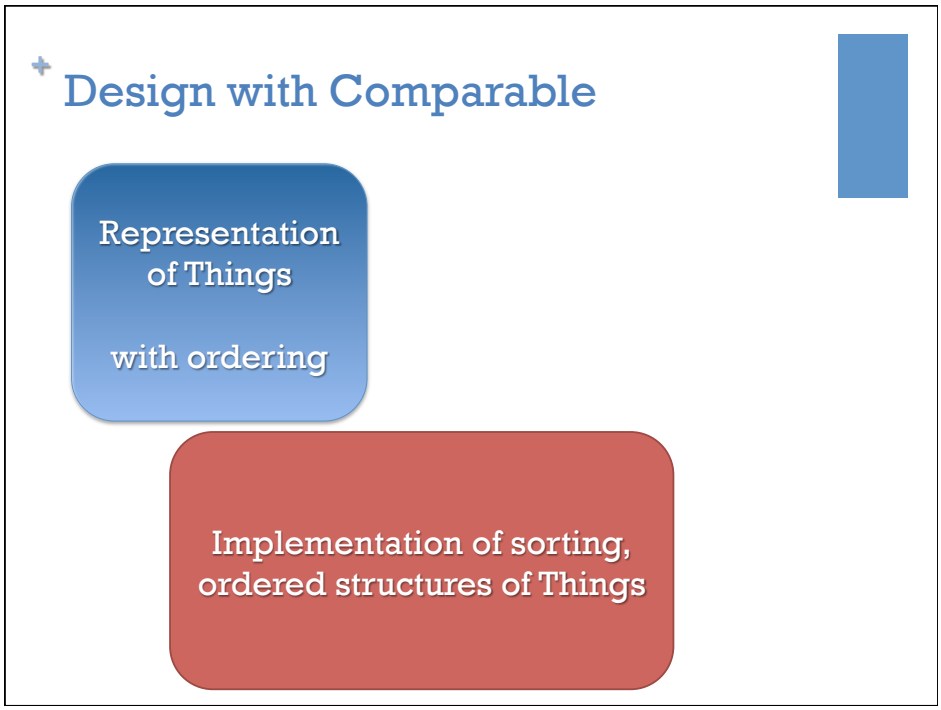
$$x.compareTo(y) = \begin{cases} <0 & \text{if } x < y \\ 0 & \text{if } x = y \\ >0 & \text{if } x > y \end{cases}$$

What if we want to sort the same objects in different orders?

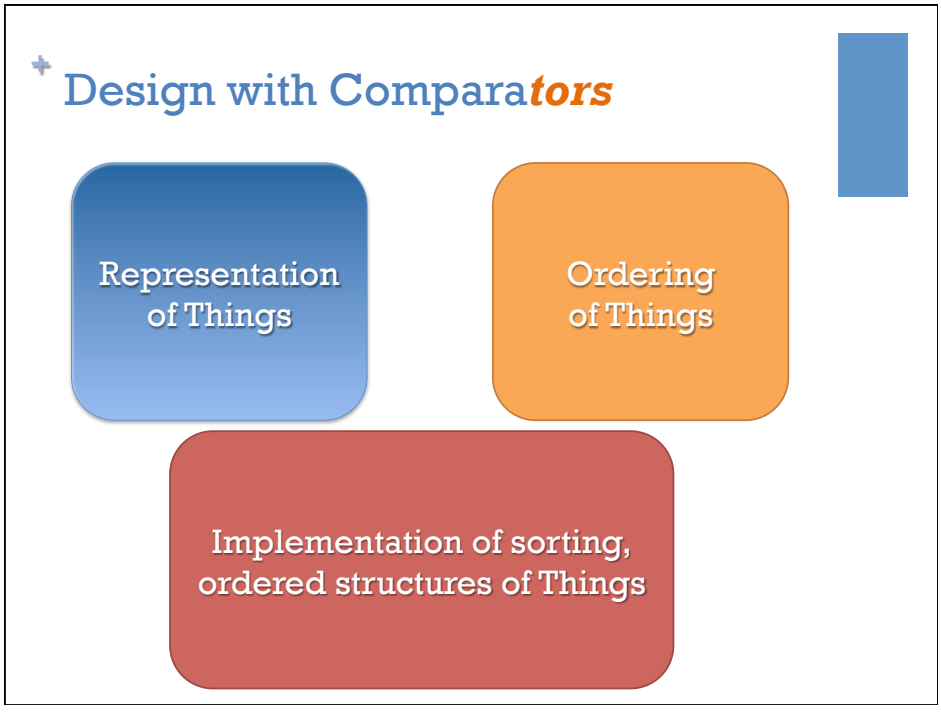
```
interfa   Comparable2<T
  int com   To2(T ot   );
}

...
public stat   T ext   s Comparable2<T>>
sort2(Lis   list) {   }
```

Design with Comparable



Design with Comparators



java.util.Comparator

```
public interface Comparator<T> {
    int compare(T x, T y);
}
```

compare(x, y) = { <0 if x < y ; 0 if x = y ; >0 if x > y }

Way of Comparing Strings

```
public class TrimComparator implements Comparator<String>
{
     // pre: x and y are Strings
     // post: returns negative, zero, or positive
     //       depending on relation
     //       between trimmed parameters.
     public int compare(String x, String y) {
         String xTrim = x.trim();
         String yTrim = y.trim();
         return xTrim.compareTo(yTrim);
     }
 }
```

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).*

Sort Students…

- by id number, in ascending order
- by "Last Name, First Name" in reverse alphabetical order
- by class year, and by "Last Name, First Name" within class year

```
public class Student {
  protected String lastName;
  protected String firstName;
  protected int id;
  protected int classYear;

  public String getLastName() { return lastName; }
  public String getFirstName() { return firstName; }
  public int getID() { return id; }
  public int getClassYear() { return classYear; }
}
```

Composing Comparators

```
public class TieBreakerComparator<T> implements Comparator<T> {
  protected Comparator<T> comp;
  protected Comparator<T> tieBreaker;

  public TieBreakerComparator(Comparator<T> c, Comparator<T> t) {
    this.comp = c;
    this.tieBreaker = t;
  }

  public int compare(T x, T y) {
    int result = comp.compare(x,y);
    if (result != 0) {
      return result;
    } else {
      return tieBreaker.compare(x,y);
    }
  }
}
```

Ordered Structures

- See OrderedArrayList
  - esp. locate method which does binary search
- See text for discussion of operations on ordered structures
  - e.g., contains, add, remove, etc.
- Why shouldn't OrderedArrayList extend ArrayList?