Administrative Details

- Lab 7 is today
- Any questions?
- Lab 6 was due yesterday
  - If you are using late days, please do not work on Lab 6 during lab today! You need to get started on Lab 7…

Last Time

- Discussed iterators (Ch 8)
  - Used for efficient data traversal
  - Reviewed the Iterator interface
    - next() and hasNext() (and remove())
  - Reviewed the AbstractIterator class
    - Leaves get(), next(), hasNext(), and reset() undefined (as indicated by “abstract” label in javadocs)

More Iterator Examples

- In addition to our “typical” iterators, we can also make specialized iterators
  - Another SLL Example (SpecialIterator.java)
  - TestIterator.java

Today’s Outline

- Learn about ordered structures (Ch 11)
  - An interesting twist on Lists and Vectors

Ordered Structures

- Until now, we have not required a specific ordering to the data stored in our structures
  - If we wanted the data ordered/sorted, we had to do it ourselves
- We often want to keep data ordered
  - Allows for faster searching
  - Easier data mining - easy to find best/worst/average/median values
Ordering Structures

- The key to establishing order is being able to compare objects and rank them
- We already know how to compare two objects...how?
- Comparators and `compare(Object a, Object b)`
- Comparable interface and `compareTo(Object that)`

An Aside: Natural Comparators

- NaturalComparators bridge the gap between Comparators and Comparables

```java
class NaturalComparator implements Comparator {
    public int compare(Object a, Object b) {
        return ((Comparable)a).compareTo(b);
    }
}
```

Another Aside: Comparable Associations

- What if we extend Associations to be Comparable?
- You might have used this in lab a few weeks ago...

```java
public class ComparableAssociation extends Association implements Comparable {
    public ComparableAssociation(Comparable key, Object val){
        super(key, val);
    }
    public int compareTo(Object other) {
        Comparable otherAssoc = (ComparableAssociation)other;
        Comparable thisKey = (Comparable) this.get(key);
        Comparable otherKey = (Comparable) other.get(key);
        return thisKey.compareTo(otherKey);
    }
}
```

Back to Ordered Vectors

- We want to create a Vector that is always sorted
- When new elements are added, they are inserted into correct position
- We still need the standard set of Vector methods
- Two choices
  - Extend Vector (like sorting lab)
  - New class (like StackVector)
    - Gives a more narrow interface
    - Not all vector methods are defined (e.g., random access add/set)
- Let's implement a new class (OrderedVector)
  - Start with Comparables
  - Generalize to use Comparators instead of Comparables

Summary

```java
public class OrderedVector<E extends Comparable<E>> implements OrderedStructure<E> {
    protected Vector<E> data;
    public OrderedVector() {
        data = new Vector<E>();
    }
    public void add(E value) {
        int pos = locate(value);
        data.add(pos, value);
    }
    protected int locate(E value) {
        //use modified binary search to find position of value
        return position;
    }
}
```

Summary

```java
public boolean contains(E value) {
    int pos = locate(value);
    return pos < size() && data.get(pos).equals(value);
}
public Object remove(E value) {
    if (contains(value)) {
        int pos = locate(value);
        return data.remove(pos);
    } else return null;
}
```

Performance:
- add - \(O(n)\)
- remove - \(O(n)\)

How would we generalize to Comparators?
Generalizing OV...

```java
public class OrderedVector<E extends Comparable<E>>
    implements OrderedStructure<E> {
    protected Vector<E> data;
    protected Comparator<E> comp;

    public OV() {
        data = new Vector<E>();
        this.comp = new NaturalComparator<E>();
    }

    public OV(Comparator<E> comp) {
        data = new Vector<E>();
        this.comp = comp;
    }

    protected int locate(E value) {
        // use modified binary search to find position of value
        // return position
        // use comp.compare instead of compareTo
    }
}

// rest stays same...
```

Ordered Lists

- Similar to OrderedVector
- Uses SinglyLinkedList instead of Vector as underlying data structure
- add, contains, remove runtime?
  - All O(n)...why?
- OrderedLists use Comparators rather than Comparables (as in OrderedVector) in structure5

Example

<table>
<thead>
<tr>
<th>Students</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duane</td>
<td>4.0</td>
</tr>
<tr>
<td>Jeannie</td>
<td>3.5</td>
</tr>
<tr>
<td>Brent</td>
<td>3.3</td>
</tr>
</tbody>
</table>

- Students compared to each other by GPA
- Suppose next semester I get a 3.3 and Brent gets a 3.7

What’s the problem?

- We have to recompute GPAs each semester
- What happens if the ordering changes?
- We may need to resort vector
- So...we need a resort method
  - But since this isn’t part of the interface, it may be forgotten
- Rule: Avoid using mutable keys in OrderedStructures
- So for our example, we should use names instead of GPAs to rank Students