CSCI 136
Data Structures & Advanced Programming

Ordered Structures

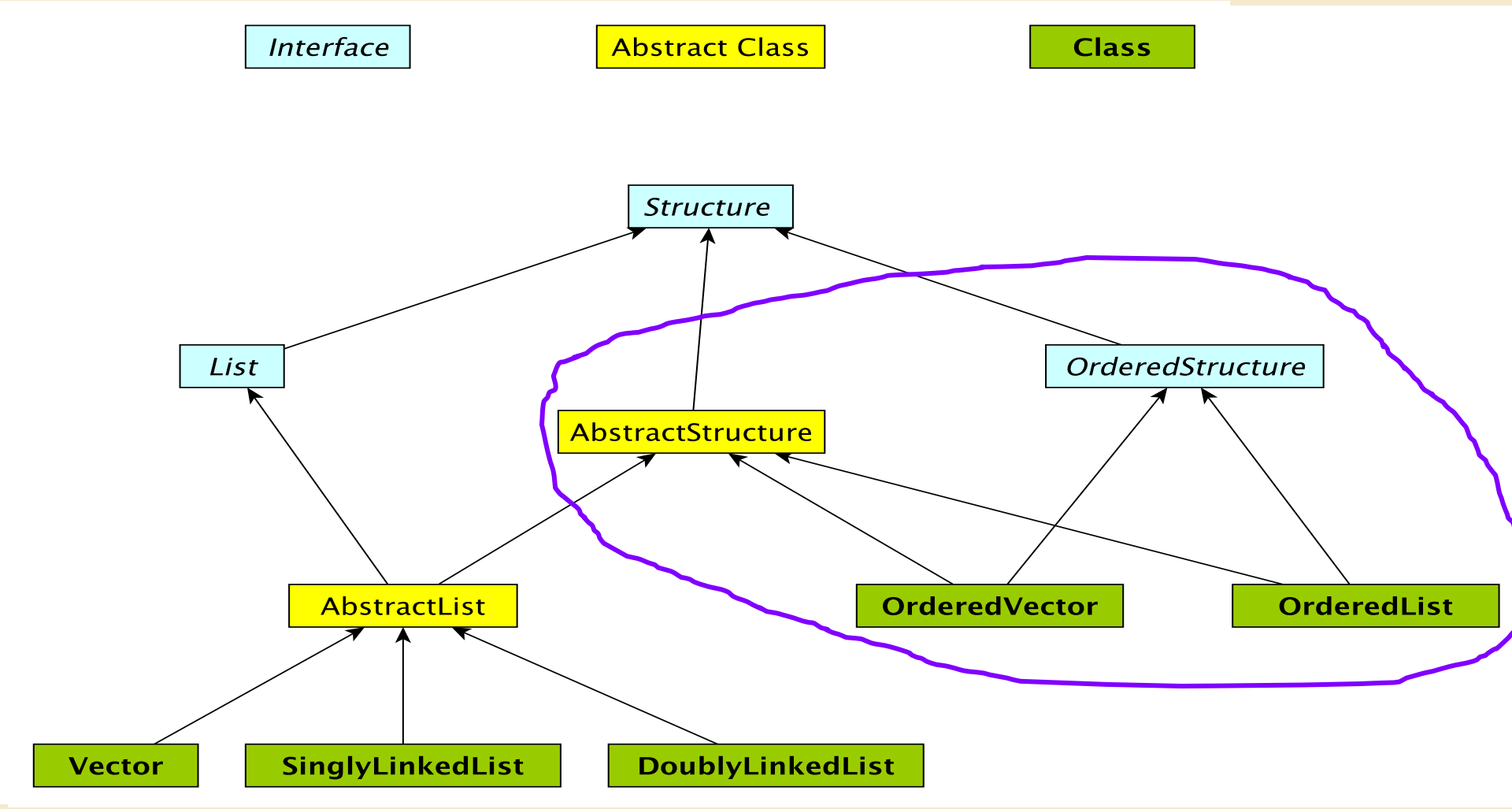
Ordered Structures

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, and median values, as well as rank (relative position)

The Structure Hierarchy

(partial view)



Ordering Structures

- The key to establishing order is being able to compare objects
- We already know how to compare two objects…how?
- Comparators and `compare(T a, T b)`
- Comparable interface and `compareTo(T that)`
- Two means to an end: which should we use?

BOTH!

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 want many of the standard set of Vector methods
    - add, remove, contains, size, iterator, …
  - But not all!
    - set(I, value) would be a problem!
- Two choices : Extend Vector or Contain a Vector
  - We choose: Contain a Vector
    - Allows for more focused interface
    - Avoid corrupting order by controlled access to Vector
- We will implement a new class (OrderedVector)
  - Start with Comparable
  - Generalize to use Comparator instead of Comparable

OrderedVector Methods

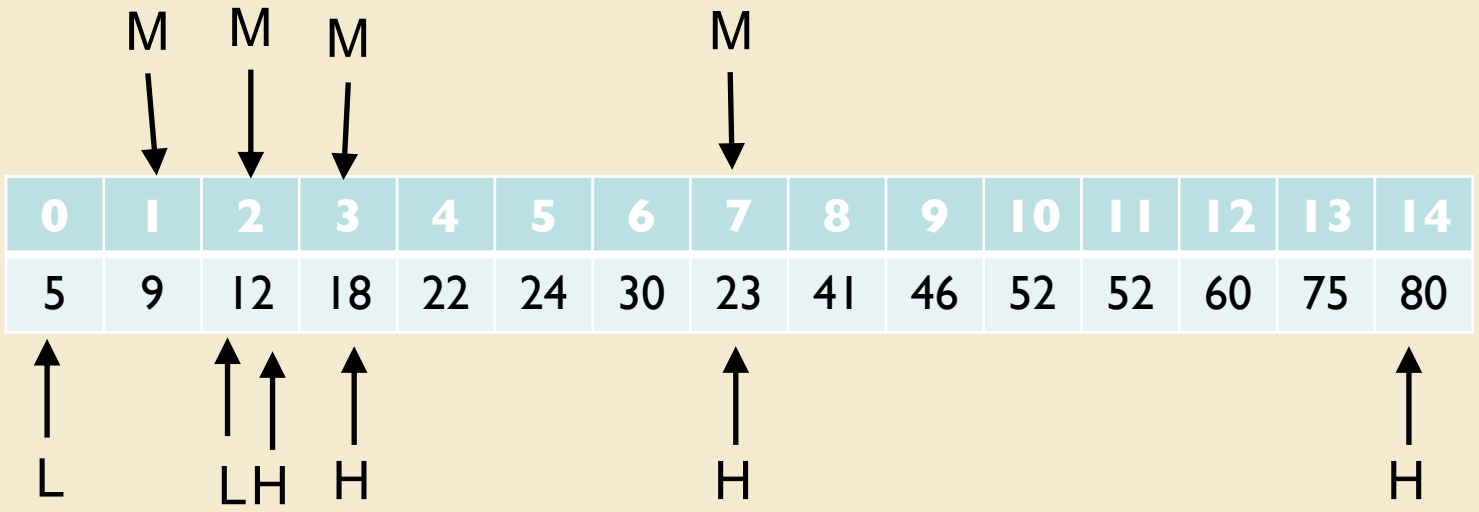
```
public class OrderedVector<E extends Comparable<E>>
  implements OrderedStructure<E> {
   protected Vector<E> data;

   public OrderedVector() {
       data = new Vector<E>();
   }

   public void add(E value) {
       int pos = locate(value);
       data.add(pos, value);
   }
```

What does locate do?

- Uses binary search to find either
  - Location of value if value is in Vector, or
  - Location where value should be added
- uses iterative version of modified binary search

Binary Search in Sorted Array

Let's picture the idea: Search for 12



Notes

- Need to keep track of current *search range*: `low..high`
- Need to know when search has failed
  - Search for 11 : Same sequence until failure

OrderedVector Methods

```
protected int locate(E target) {

   Comparable<E> midValue;

   int low = 0;                        // lowest location
   int high = data.size();             // highest location
   int mid = (low + high)/2;           // low <= mid <= high

   while (low < high) {
       midValue = data.get(mid);

       if (midValue.compareTo(target) < 0) low = mid+1;
       else high = mid;

       mid = (low+high)/2;             // NB: 0 ≤ mid ≤ data.size()
   }
   return low;
}
```

OrderedVector Methods

```
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:

- locate - O(log n)
- add - O(n)
- contains - O(log n)
- remove - O(n)

Adding Flexibility with Comparators

- We would like to be able to customize the ordering of our ordered structures
- Idea: Add constructor that has a Comparator parameter
- Q: How does structure know whether to use the Comparator or the Comparable ordering?
- A: The NaturalComparator class....

An Aside: Natural Comparators

- NaturalComparators bridge the gap between Comparators and Comparables

```
class NaturalComparator<E extends Comparable<E>>
implements Comparator<E> {
    public int compare(E a, E b) {
          return a.compareTo(b);
    }
}
```

Generalizing OrderedVector

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

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

   public OrderedVector(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…
```

A Confession

- The previous slide demonstrated how to add flexibility to the OrderedVector class using Comparators
- Structure5 did not implement this use of Comparators for the OrderedVector class!
- But did implement it for OrderedList!
- Let's take a look….

Ordered Lists

- Similar to OrderedVector
- Can't efficiently use SinglyLinkedList like OrderedVector used Vector
  - Most methods would traverse list multiple times
- So, we just build a SinglyLinkedList-like structure
- add, contains, remove runtime?
  - All O(n) : Must traverse list
- Let's look at a few details….

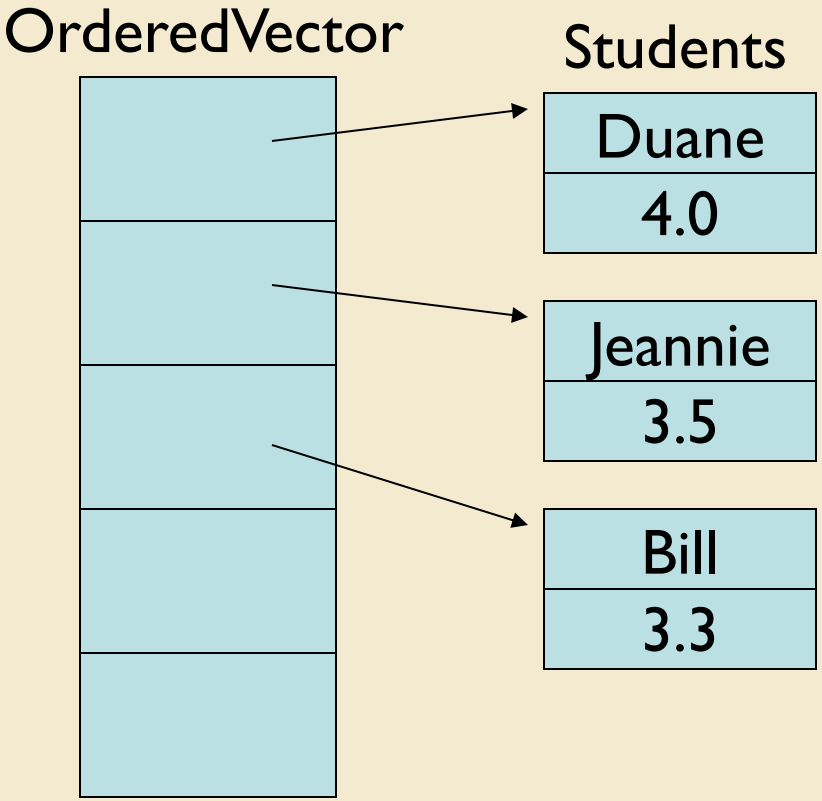
OrderedList Methods

```
public class OrderedList<E extends Comparable<E>>
    extends AbstractStructure<E> implements
                    OrderedStructure<E> {

    protected Node<E> data; // smallest value
    protected int count;    // size of list
    protected Comparator<? super E> ordering;

    public OrderedList() {
        this(new NaturalComparator<E>());
    }
    public OrderedList(Comparator<? super E> ordering){
        this.ordering = ordering;
        clear();
    }
```

OrderedList Methods

```
public void clear() {
      data = null;
      count = 0;
}
public boolean contains(E value) {
   Node<E> finger = data; // target

   while ((finger != null) &&
      ordering.compare(finger.value(),value)<0)

      finger = finger.next();

  return finger!=null && value.equals(finger.value());
}
```

What Could Go Wrong?



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

What's the problem?

- We have to recompute GPAs each semester
- What happens if the values are allowed to change?
- We may need to resort vector
  - But since this isn't part of the interface, it may be forgotten
- Options:
  - Avoid changing values in OrderedStructures
  - Incorporate an update method that repositions element
  - Incorporate a resort method
    - This invites adding a "setComparator" method....
  - Always update a value by removing and re-adding

Bonus : Type Safety & Generics

- Question: Since String extends Object, does List<String> extend List<Object>?
  - I.e., can I say List<Object> = new List<String>()?
- No. It would compromise the type system:

```
List<String> slist = new List<String>();
List<Object> olist = slist;    // If this were possible
olist.add(new Object());       // This would be bad!
```

- It generates a compiler error.
- On the other hand…

```
String[] sa = {“I”, “love”, “java”, “!”};
Object[] oa = sa;
oa[1] = new Object()); // This would be bad!
```

- …actually compiles
  - But causes a run-time error!

Summary & Observations

- Imposing order on the elements in a structure can improve performance of order-related queries.
  - A sorted Vector improves search from Θ(n) to Θ(log n)
  - Didn't improve search for linked list, but…
- Consider the *Rank Problem*: Given a collection of comparable objects, find the k^{th} smallest (or k^{th} largest) object in the collection.
  - How would you do this with an ordered linked list?
  - How would you do this with an *unordered* linked list?!
- Using Comparators allows ordered structures to order the same data in a variety of ways.
  - Especially if the Comparator can be replaced!