Administrative Details

• Lab 7 today
  • No partners this week
  • Review before lab; come to lab with design doc
  • Check out the javadoc pages for the 3 provided classes
    • Token – A wrapper for semantic PS elements,
    • Reader – An iterator to produce a stream of Tokens from standard input or a List of Tokens,
    • SymbolTable – A dictionary with String keys and Token values: For user-defined names
Last Time

• Iterators Recap
• Iterating over Iterators
• Ordered Structures
  • OrderedVector
Today: Ordered Structures & Introduction to Trees

• Ordered Structures
  • OrderedVector wrap-up
  • OrderedList

• Tree-like Structures
Ordered Vectors

• We implement a new class (OrderedVector)
  • Start with Comparable elements
  • Goal: Only provide operations that keep the Vector sorted at all times
    • So, for example, no add(int index, E item);

• OrderedVector will implement OrderedStructure
  • An Interface extending Structure
  • Merely forces items to be Comparable
    
    ```java
    public interface OrderedStructure<K extends Comparable<K>> extends Structure<K> {}
    ```

• Generalize to use Comparators instead of Comparables
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
        //uses iterative version of binary search (see text)
    }
}
protected int locate(E target) {
    Comparable<E> midValue;
    int low = 0; int high = data.size();
    int mid = (low + high) / 2;

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

        mid = (low + high) / 2;
    }
    return low;  // low = high so return either!
}
OrderedVector Methods

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

public E remove(E value) {
    if (contains(value)) {
        int pos = locate(value);
        return data.remove(pos);
    } else return null;
}

Performance:
    add - O(n)
    contains - O(log n)
    remove - O(n)
```
Adding Flexibility with Comparators

- We would like to be able to allow ordered structures to use different orders
- 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

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

- **Full disclosure**
  - The following is what OrderedVector could do
  - But it doesn’t....
Generalizing OrderedVector

```java
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...
```
public class OrderedVector<E extends Comparable<E>>
    implements OrderedStructure<E> {
    protected Vector<E> data;
    protected Comparator<? Super E> comp; // Even better!

    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...
Ordered Lists

• Similar to OrderedVector
• Can’t easily use SinglyLinkedList like OrderedVector used Vector (Why?)
• So, we just build a SinglyLinkedList-like structure
• Let’s look at some code…
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

```java
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());
}
```
Ordered Lists

• Similar to OrderedVector
• Can’t easily use SinglyLinkedList like OrderedVector used Vector (Why?)
• So, we just build a SinglyLinkedList-like structure
• Let’s look at some code…
• add, contains, remove runtime?
  • All O(n)…why?
Type Safety & Generic Types

- Question: Since String extends Object, does List<String> extend List<Object>?
  - I.e., can I say List<Object> l = new List<String>()?
- No. It would compromise the type system:
  ```java
  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...
  ```java
  String[] sa = {"I", "love", "java", "!"};
  Object[] oa = sa;
  oa[1] = new Object();   // This would be bad!
  ```
- …actually compiles
  - But causes a run-time error!
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

<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>Bill</td>
<td>3.3</td>
</tr>
</tbody>
</table>
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....
  - No perfect solution
Introducing Trees

- Our structures have had a linear organization
  - Stacks, queues
  - Even ordered vectors, ordered lists, arrays, vectors, lists are visualized linearly
- By linear we essentially mean that each element has at most one successor and at most one predecessor…
Branching Out: Trees

• A tree is a data structure where elements can have multiple successors (called children)
• But still only one predecessor (called parent)
“Computer Tree”
House of Normandy, Battle of Hastings, 1066

- William I
  - Robert
  - William II
  - Adela
    - Stephen
  - Henry I
    - William
    - Matilda
      - Henry II
Tree Features

- Hierarchical relationship
- Root at the top
- Leaf at the bottom
- Interior nodes in middle
- Parents, children, ancestors, descendants, siblings
- Degree (of node): number of children of node
- Degree (of tree): maximum of node degrees
- Depth of node: number of edges from root to node
- Height: maximum depth (across all nodes)
Other Trees

- Phylogenetic tree
- Directories of files
- Game trees
  - Build a tree
  - Search it for moves with high likelihood of winning
- Expression trees
Phylogenetic Tree of the Animal Kingdom

- Ancestral Protists
  - True Tissues
  - Bilateral symmetry
  - Coelom
  - Deuterostomes
    - Coelom from digestive tube
  - Protostomes
    - Coelom from cell masses
  - Radial symmetry
  - No body cavity
  - Pseudocoelom

- Chordata
- Arthropoda
- Echinodermata
- Annelida
- Mollusca
- Nematoda
- Platyhelminthes
- Parazoa
Expression Trees

$4 \times 2 + 3$

$\frac{(4 \times 2 + 3) + \left( \frac{10 - 2}{4} \right)}{\frac{10}{4}}$
Introducing Binary Trees

• Degree of all nodes $\leq 2$
• Recursive nature of tree
  • Empty
  • Root with left and right subtrees
• SLL: Recursive nature was captured by nodes (Node<E>) on inside
• Binary Tree: No “inner” node class; single BinaryTree class does it all
Expression Trees

4 * 2 + 3

BinaryTree<String> fourTimesTwo =
    new BinaryTree<String>("*",
    new BinaryTree<String>("4",
    new BinaryTree<String>("2");

BinaryTree<String> fourTimesTwoPlusThree =
    new BinaryTree<String>("+",
    fourTimesTwo,
    new BinaryTree<String>("3");

Or use Token class!
Expression Trees

• General strategy
  • Make a binary tree (BT) for each leaf node
  • Move from bottom to top, creating BTs
  • Eventually reach the root
  • Call “evaluate” on final BT

• Example
  • How do we make a binary expression tree for
    (((4+3)*(10-5))/2)
    • Postfix notation: 4 3 + 10 5 - * 2 /
int evaluate(BinaryTree<String> expr) {
    if (expr.height() == 0) {
        return Integer.parseInt(expr.value());
    } else {
        int left = evaluate(expr.left());
        int right = evaluate(expr.right());
        String op = expr.value();
        switch (op) {
            case "+": return left + right;
            case "-": return left - right;
            case "*": return left * right;
            case "/": return left / right;
        }
        Assert.fail("Bad op");
        return -1;
    }
}