CSSE 220 Day 29

Linked List Implementation

Check out *MyLinkedListReal* project from SVN

Announcements

- Blood Drive today!
- CSSE Senior Project presentations 11:00-1:30 today. Union Lobby
- Hulbert/Cook lecture tomorrow 10:50 Hatfield Hall
- Minesweeper due at 8:05 AM Thursday
  - If you plan to use a late day, please fill out the survey by noon Thursday
    - So I can begin grading the ones that are done.
- Markov due at 11:59 Friday.
- Course evaluations in class tomorrow.
- I will hold a review (Q &A) session Tuesday at 4:00 PM. In O-201.

Final Exam
Wed 6 PM
O269

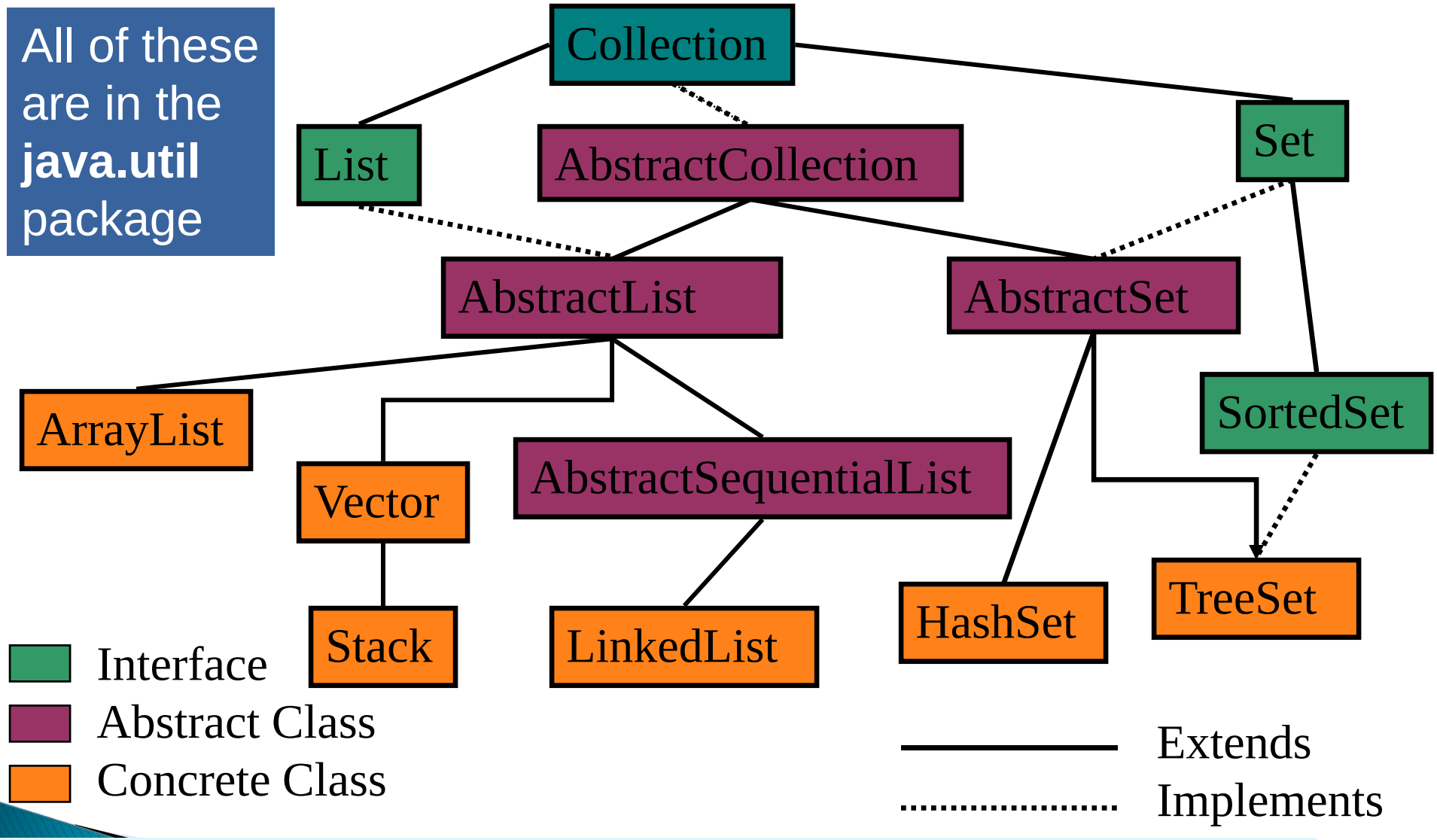
Questions

Remainder of the slides

- The rest of the slides are from last session; they are here for reference purposes.

Some `Collection` interfaces and classes

All of these are in the **java.util** package



This is the Java 1.2 picture. Java 1.5 added **Queue**, **PriorityQueue**, and a few other interfaces and classes.

Some Methods From the Collection Interface

java.util

Interface Collection<E>

boolean	**add**(E o) Ensures that this collection contains the specified element (optional operation).
boolean	**contains**(Object o) Returns true if this collection contains the specified element.
boolean	**isEmpty**() Returns true if this collection contains no elements.
boolean	**remove**(Object o) Removes a single instance of the specified element from this collection, if it is present (optional operation).
int	**size**() Returns the number of elements in this collection.
Iterator<E>	**iterator**() Returns an iterator over the elements in this collection.

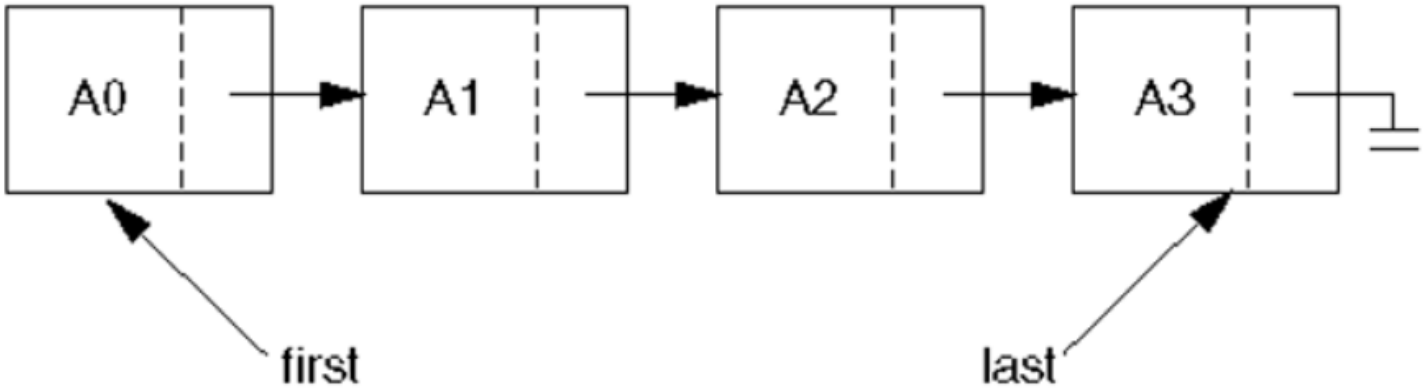
Additional List Interface methods (List extends Collection)

- A List is an ordered collection, items accessible by position. Here, *ordered* does not mean *sorted*.
- interface java.util.List<E>
- User may insert a new item at a specific position.
- Some important List methods:

void	**add**(int index, E element) Inserts the specified element at the specified position in this list (optional operation).
E	**get**(int index) Returns the element at the specified position in this list.
int	**indexOf**(Object o) Returns the index in this list of the first occurrence of the specified element, or -1 if this list does not contain this element.
E	**remove**(int index) Removes the element at the specified position in this list (optional operation).
E	**set**(int index, E element) Replaces the element at the specified position in this list with the specified element (optional operation).

Break

`LinkedList` implementation of the `List` Interface



- Stores items (non-contiguously) in nodes; each contains a reference to the next node.
- Lookup by index is linear time (worst, average).
- Insertion or removal is constant time once we have found the location.
  - show how to insert A4 after A1.
- If **`Comparable`** list items are kept in sorted order, finding an item still takes linear time.

Consider Part of a `LinkedList` implementation:

```
class ListNode{
 Object element; // contents of this node
 ListNode next;  // link to next node

 ListNode (Object element,
           ListNode next) {
   this.element = element;
   this.next = next;
 }

 ListNode (Object element) {
   this(element, null);
 }
 ListNode () {
   this(null);
 }
}
```

How to implement LinkedList?

- fields
- Constructors
- Methods

Note that the fields of this class have "package" access, so that other classes in the same package can access them directly. **ListNode** objects are used like C structs.

Let's do parts of a `LinkedList` implementation

```
class LinkedList implements List {
  ListNode first;
  ListNode last;
```

Constructors: (a) default (b) single element.

methods:

Attempt these in the order shown here.

`public boolean add(Object x)`

Appends the specified element to the end of this list (returns `true`)

`public int size()` Returns the number of elements in this list.

`public void add(int i, Object x)` adds x at index i.

`throws IndexOutOfBoundsException`

`public boolean contains(Object x)`

Returns true if this list contains the specified element. (2 versions).

`public boolean remove(Object x)`

Removes the first occurrence (in this list) of the specified element.

`public Iterator iterator()` **Can we also write listIterator( ) ?**

Returns an iterator over the elements in this list in proper sequence.

What's an iterator?

- More specifically, what is a `java.util.Iterator`?
  - It's an interface:
  - **`interface java.util.Iterator<E>`**
  - with the following methods:

boolean	**hasNext**() Returns `true` if the iteration has more elements.
E	**next**() Returns the next element in the iteration.
void	**remove**() Removes from the underlying collection the last element returned by the iterator (optional operation).

An extension, `ListIterator`, adds:

boolean	**hasPrevious**() Returns `true` if this list iterator has more elements when traversing the list in the reverse direction.
int	**nextIndex**() Returns the index of the element that would be returned by a subsequent call to `next`.
[redacted]ject	**previous**() Returns the previous element in the list.
int	**previousIndex**() Returns the index of the element that would be returned by a subsequent call to `previous`.
void	**set**(Object o) Replaces the last element returned by `next` or `previous` with the specified element (optional operation).

Work on Linked Lists

- Live coding together.

Markov work time