CSSE 220 Day 21

LinkedList Implementation

CSSE 220 Day 21

- Turn in your written problems
- Reminder: Exam #2 is Thursday, Jan 31.
  - In order to reduce time pressure, you optionally may take the non-programming part 7:10-7:50 AM.
- Markov repositories:
  - http://svn.cs.rose-hulman.edu/repos/220-200820-markovXX
    - where XX is your 2-digit team number.
- Markov Progress:
  - Milestone 1 official due time Monday 8:05 AM
  - But you should think of the real due time as Saturday at noon, so you can make progress on Milestone 2 this weekend.

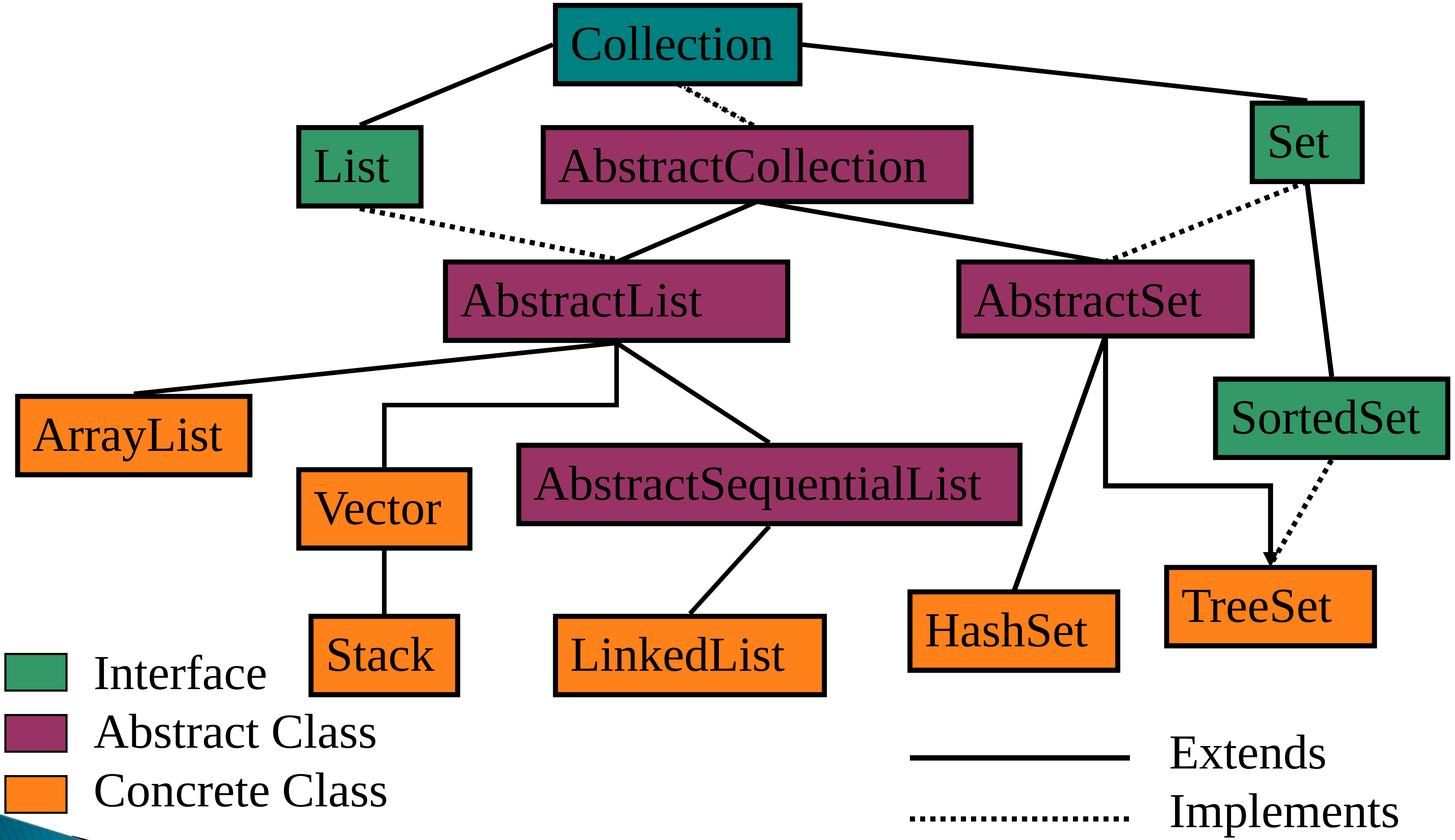
Answers to your questions

- Abstract Data Types and Data Structures
- Collections and Lists
- Markov
- Material you have read
- Anything else

Today's agenda

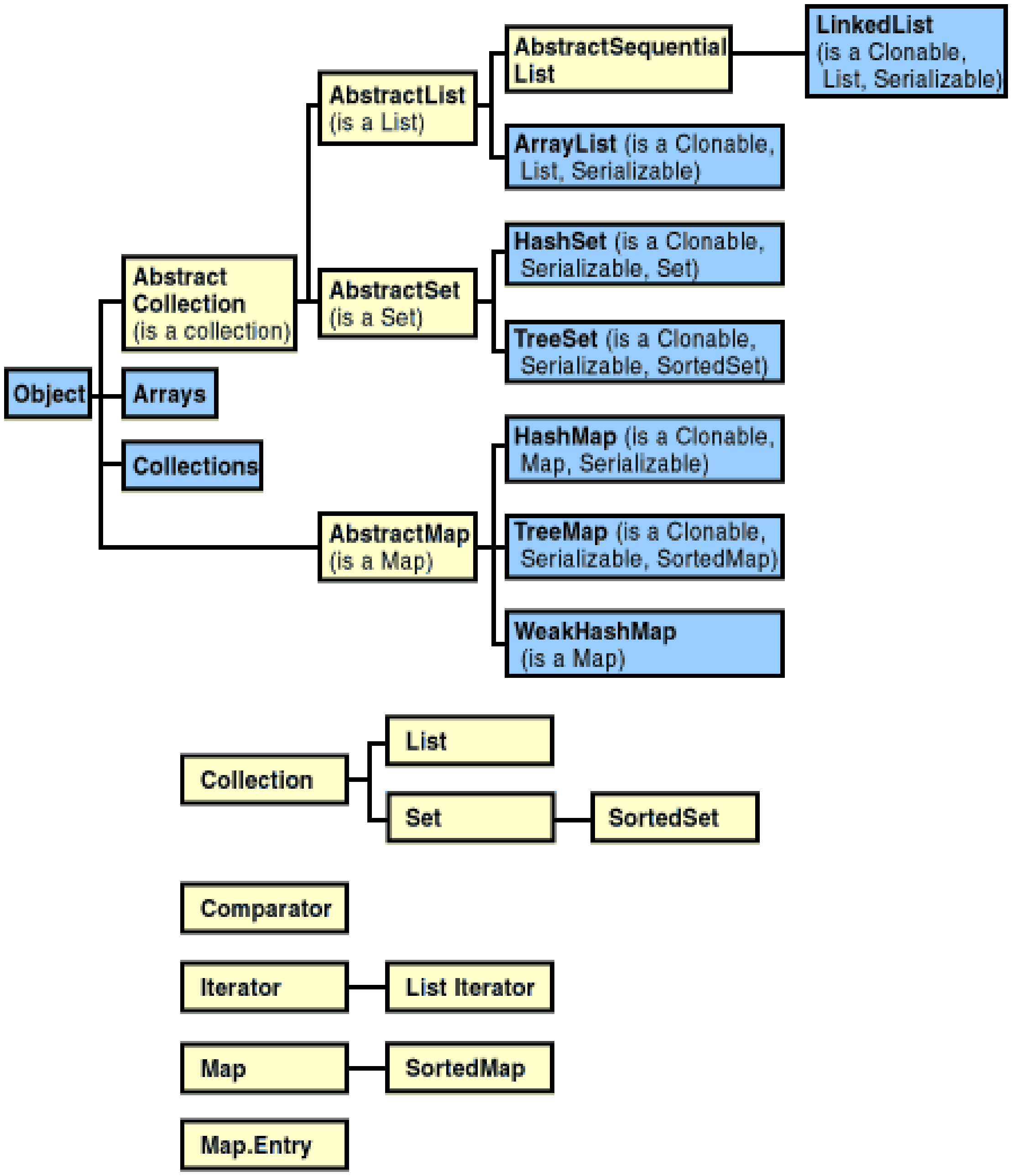
- LinkedList Implementation
- Recursion

Recap: Some `Collection` interfaces and classes

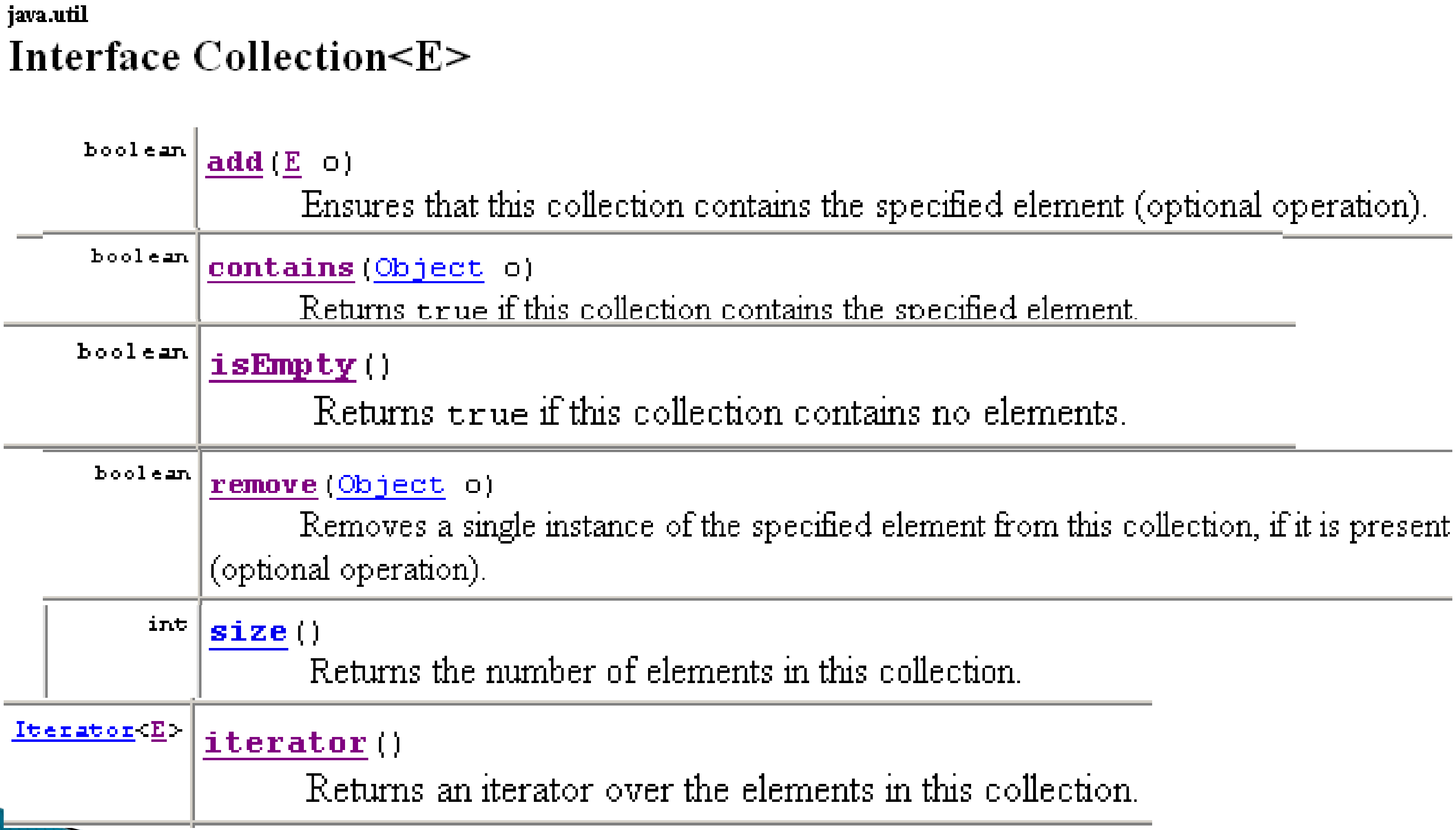


This is the Java 1.2 picture. Java 1.5 added Queue, PriorityQueue, and a few other interfaces and classes.

Collections classes and interfaces (classes at top, interfaces at bottom)


Object
Abstract Collection (is a collection)
Arrays
Collections
AbstractList (is a List)
AbstractSet (is a Set)
AbstractMap (is a Map)
AbstractSequential List
LinkedList (is a Clonable, List, Serializable)
ArrayList (is a Clonable, List, Serializable)
HashSet (is a Clonable, Serializable, Set)
TreeSet (is a Clonable, Serializable, SortedSet)
HashMap (is a Clonable, Map, Serializable)
TreeMap (is a Clonable, Serializable, SortedMap)
WeakHashMap (is a Map)
Collection
List
Set
SortedSet
Comparator
Iterator
List Iterator
Map
SortedMap
Map.Entry

The Collection interface



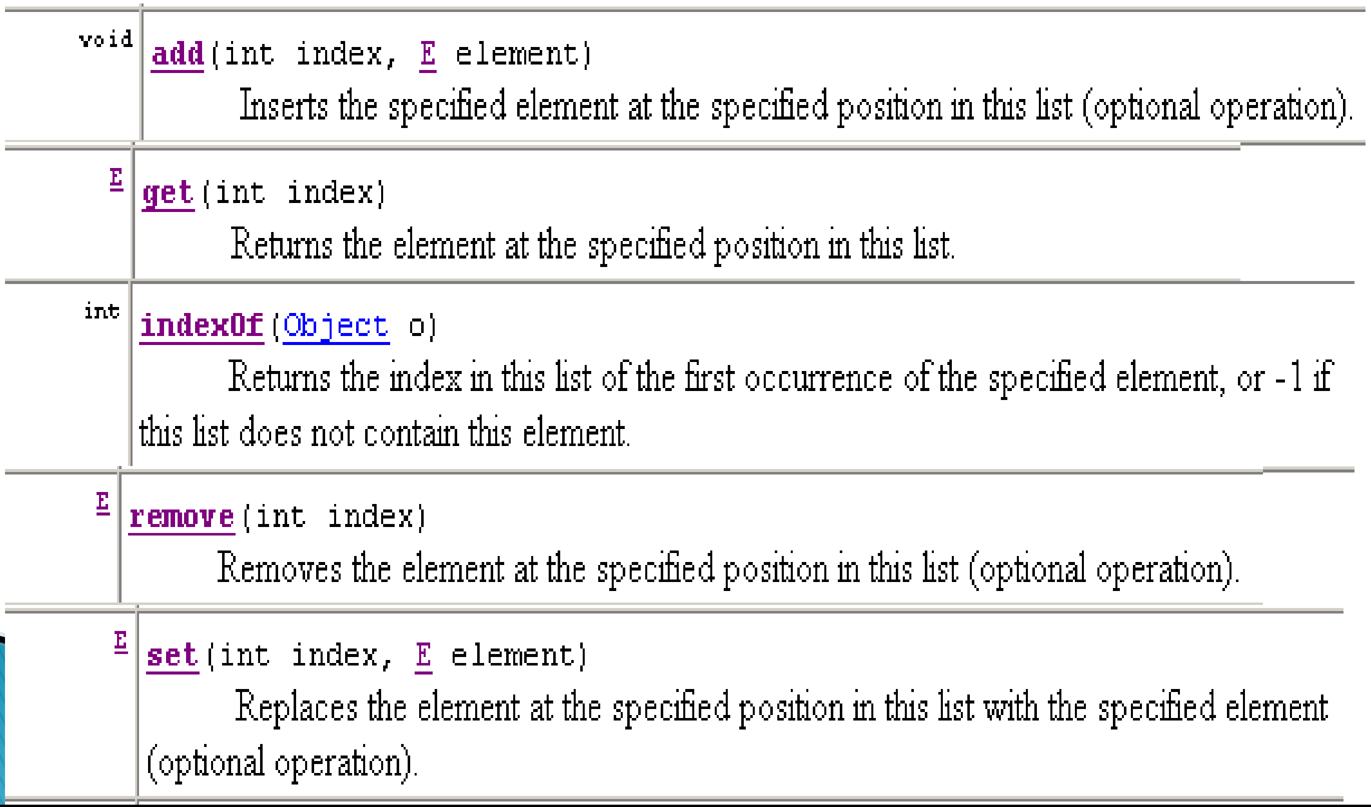
java.util

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.

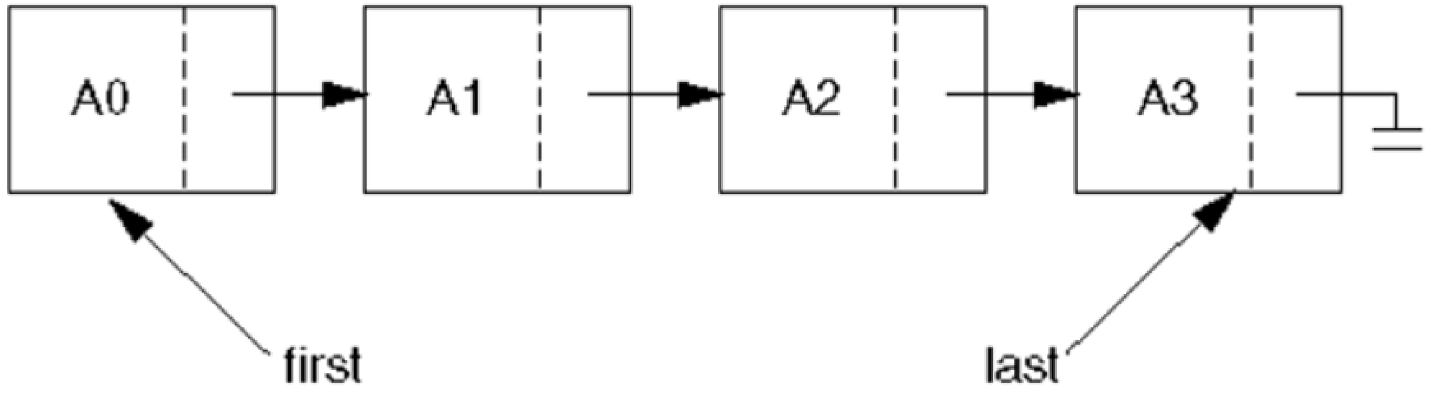
List Interface (extends Collection)

- 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).

`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 parts 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?

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 o)`**

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 o)` adds o at index i.
`  throws IndexOutOfBoundsException`

`public boolean contains(Object o)`

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

`public boolean remove(Object o)`

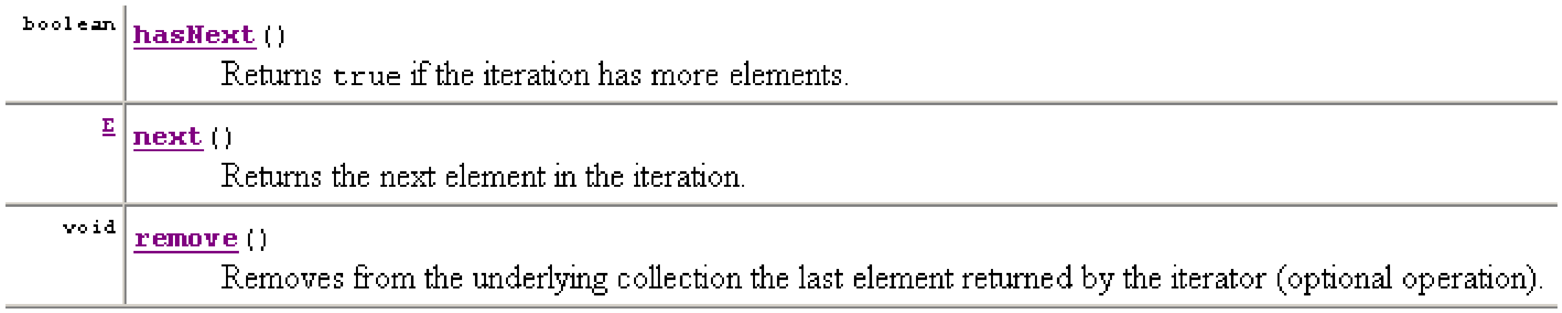
Removes the first occurrence (in this list) of the specified element.

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

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

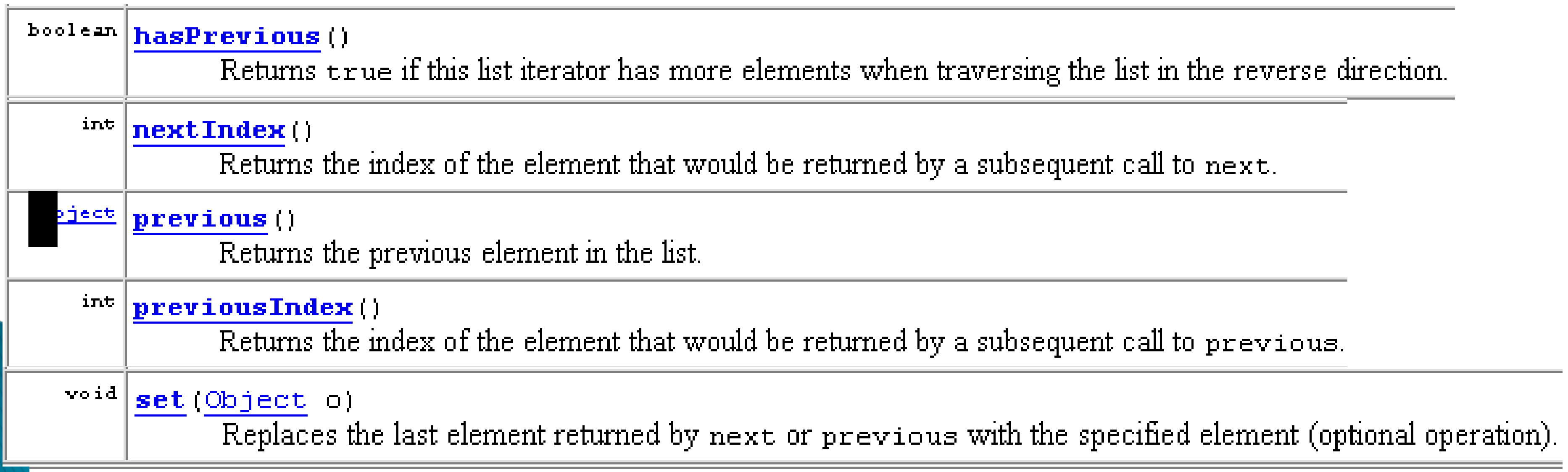
What's an iterator?

- 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).