CSSE 220 Day 21

LinkedList Implementation

CSSE 220 Day 21

- Turn in your written problems
- Reminder: Exam #2 is 1 week from today
- Markov Progress:
  - Milestone 1 official due time Tuesday 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.

- Questions: Data structures? Markov?

- Today: LinkedList implementation

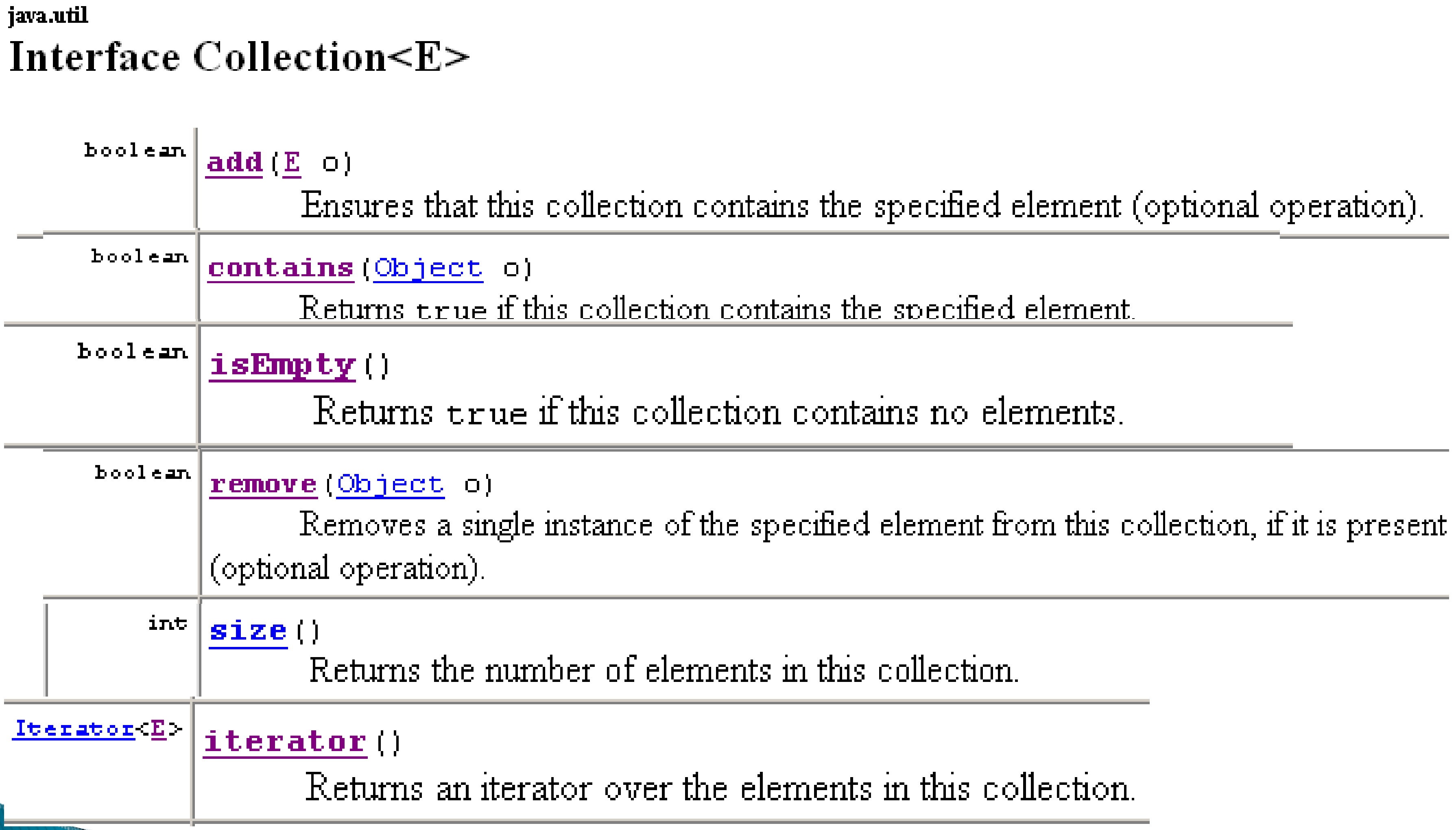
Citizenship grade

- From the Syllabus:
- **Citizenship Counts!**
- I may adjust your overall average up or down by up to 5 percent, based on your citizenship in the CSSE 220 learning community. This includes attendance, promptness, preparation for class, positive participation in class and the online discussion forums, constructive partnership in pair and group assignments, timely completion of various surveys, and peer evaluation of other students' code and of your team members for group projects.
- The in-class time in this course constitutes an important learning experience. You should be there, Two unexcused absences will affect your Citizenship grade. **Three or more unexcused absences may result in failure of the course.** I will use the ANGEL attendance manager as a record of your attendance. Be sure to record your attendance each day.

Citizenship grade – 5 points

- If you missed zero or one classes, the starting point for your grade is 2.
- **Plusses**: regular class participation, asking and answering questions, working well with others on in-class pair projects, posting to the discussion forums (especially if you answer other students questions), many bug reports, assisting other students, making suggestions for course improvement, etc..
- **Minuses**: Missing more than one class (unexcused), being late a lot, not "being there" when I have asked you questions in class, not working well with your partners on projects, not submitting partner evaluations.

The Collection interface



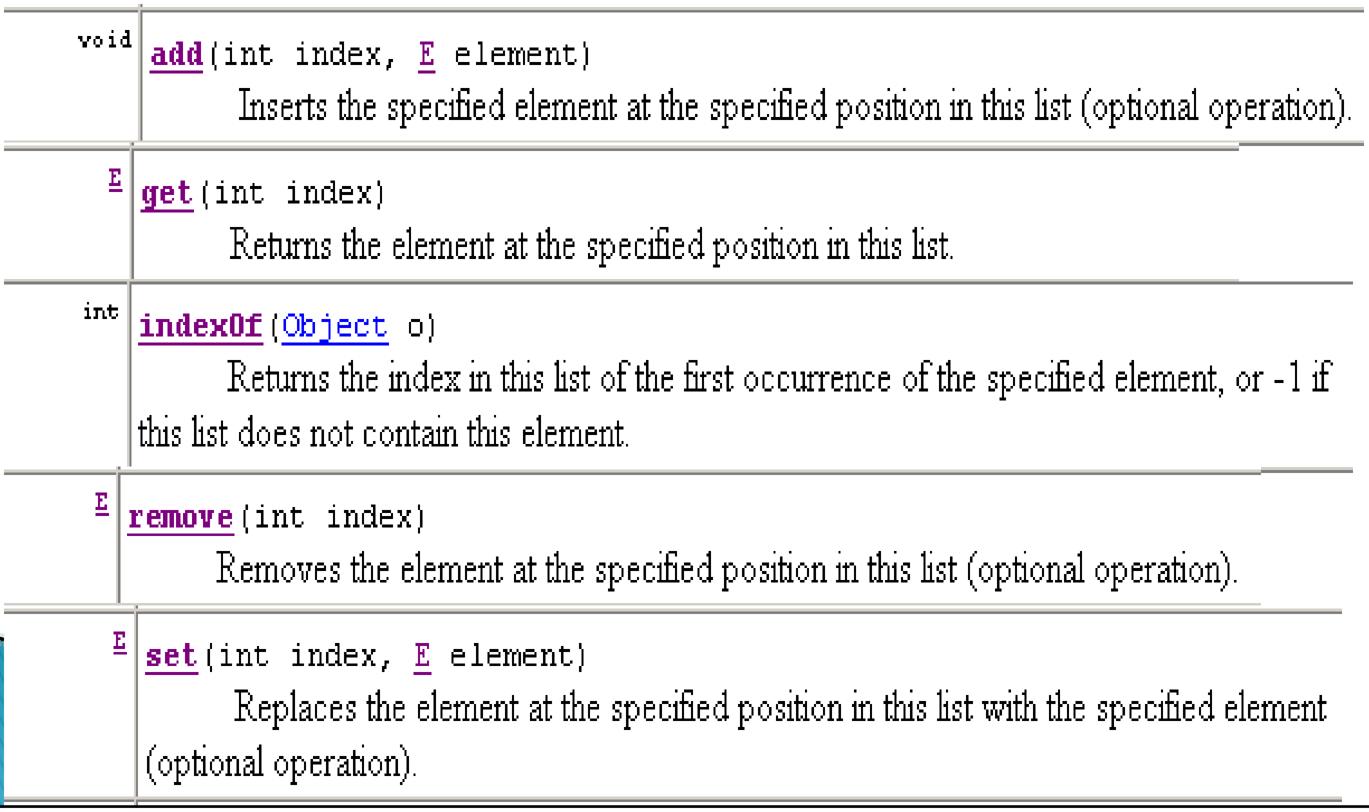
java.util

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.

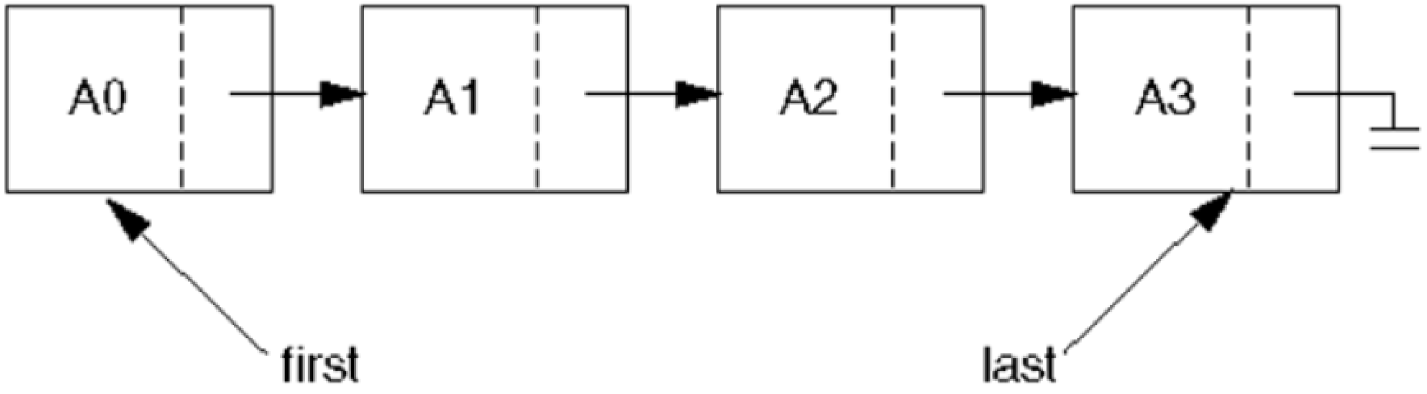
`List` Interface (extends `Collection`)

- 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)`

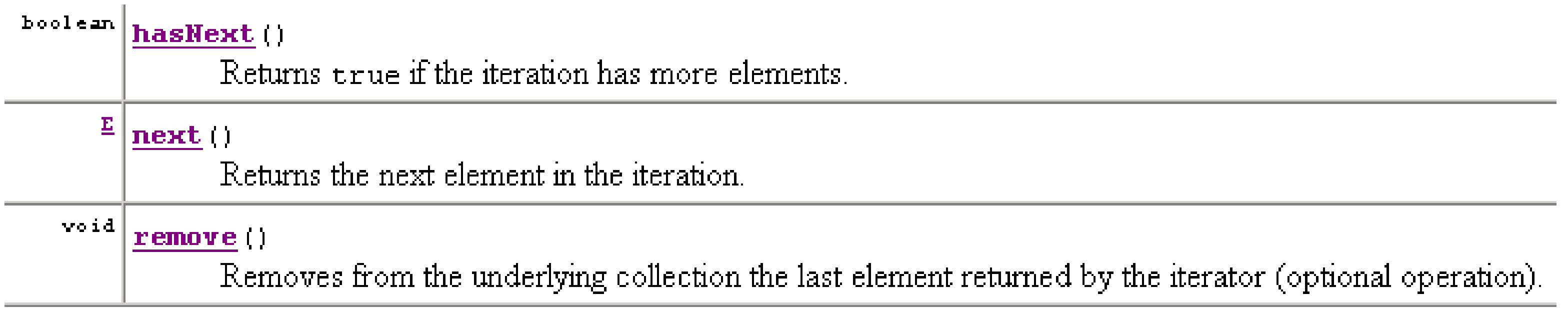
Removes the first occurrence (in this list) of the specified element.

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

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

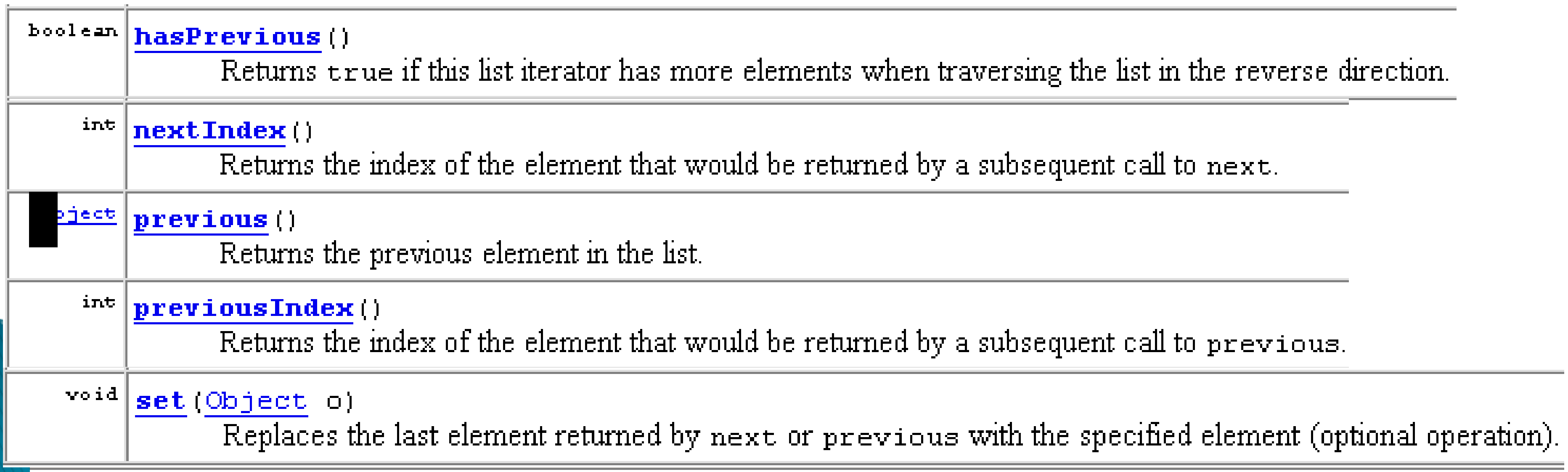
What's an iterator?

- 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).