CS 367 - Introduction to Data Structures
Thursday, February 11, 2016

Homework 3 due 10 pm tomorrow, February 12th
Homework 4 assigned by Monday, February 15th
Program 1 due 10 pm this Sunday, February 14th
Program 2 assigned Monday?

Assignment questions? Post on Piazza or consult with a TA during scheduled hours.

Report exam conflicts or McBurney exam accommodations by tomorrow, 2/12

Last Time

Chains of Linked Nodes
- more practice with chains of nodes

Java Visibility Modifiers

Today

LinkedList Class
Linked List Variations
- header node
- tail reference

LinkedListIterator Class
Iterable and For-Each Loops

Next Time

Read: finish *Linked Lists*, start *Complexity*
More Linked List Variations
- double linking
- circular linking

Complexity
- concept
- big-O notation
- analyzing algorithms practice
- analyzing Java code

Recall the List ADT

Concept

A List is a general, position-oriented container that stores a contiguous collection of items where duplicates are allowed. It maintains relative ordering and uses zero-based indexing.

Operations

```
void add(E item);
void add(int pos, E item);
E get(int pos);
E remove(int pos);
boolean contains(E item);
int size();
boolean isEmpty();
```

Issues

Null item – detect then signal with `IllegalArgumentException`
Bad position – detect then signal with `IndexOutOfBoundsException`
Empty list – handle as a bad position

`LinkedList` - Implementing `ListADT` using a Chain of Nodes

```
public class LinkedList<E> implements ListADT<E> {

   private Listnode<E> head;
   private int numItems;


   public LinkedList() {




   }

   public void add(E item) {
```

`LinkedList` (cont.)

```
public class LinkedList<E> implements ListADT<E> {

   private Listnode<E> head;
   private int numItems;

   public LinkedList() { ... }
   public void add(E item) { ... }


   public E get(int pos) {
```

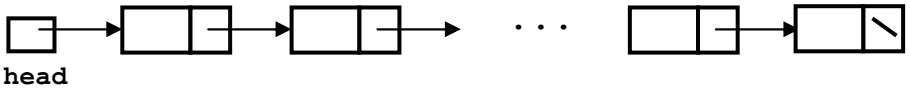
Header Node

Concept

-

empty

non-empty



-
-

Code Example

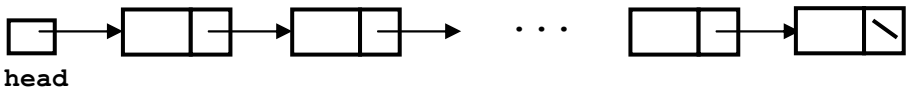
```
public class LinkedList<E> implements ListADT<E> {

   private Listnode<E> head;
   private int numItems;

   public LinkedList() {
      head = null;
      numItems = 0;
   }

   public void add(E item) {
      if (item == null) throw new IllegalArgumentExceptions();

      Listnode<E> newnode = new Listnode<E>(item);

      //Special Case: empty list
      if (head == null) {
         head = newnode;
      }
      //General Case: non-empty list
      else {
         Listnode<E> curr = head;
         while (curr.getNext() != null)
            curr = curr.getNext();
         curr.setNext(newnode);
      }

      numItems++;
   }
```

Tail Reference

Concept

-

empty non-empty

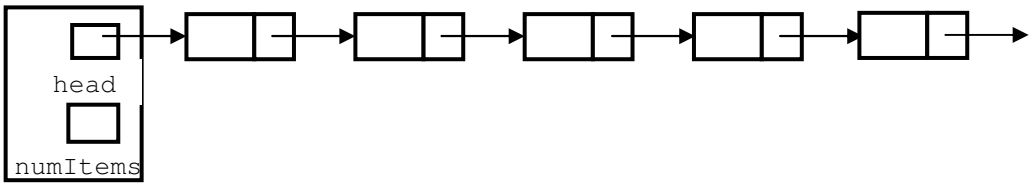


Code Example

```
public class LinkedList<E> implements ListADT<E> {

   private Listnode<E> head;

   private int numItems;

   public LinkedList() {
      head = null;

      numItems = 0;
   }

   public void add(E item) {
      if (item == null) throw new IllegalArgumentExceptions();

      Listnode<E> newnode = new Listnode<E>(item);

      //Special Case: empty list
      if (head == null) {
         head = newnode;

      }
      //General Case: non-empty list
      else {
         Listnode<E> curr = head;
         while (curr.getNext() != null)
            curr = curr.getNext();
         curr.setNext(newnode);
      }

      numItems++;
   }
```

Implementing `LinkedListIterator`

→ Should an indirect or a direct iterator implementation be used with a `LinkedList`?




```
import java.util.*;

public class LinkedListIterator<E> implements Iterator<E> {


   LinkedListIterator(                              ) {


   }
   public boolean hasNext() {


   }
   public E next() {
       if (                      ) throw new NoSuchElementException();




   }
   public void remove() {
       throw new UnsupportedOperationException();
   }
}
```

`Iterable` Interface – The Rest of the Story

Java's For-Each Loops

```
ListADT<String> list ... //assume list of words

Iterator<String> itr = list.iterator();
while (itr.hasNext())
   System.out.println(itr.next());
```

Problem - Implementing Multiple Interfaces

```
public class LinkedList<E> implements ListADT<E>, Iterable<E> { ... }

ListADT<String> list = new LinkedList<String>();
Iterator<String> itr = list.iterator();



Iterable<String> list = new LinkedList<String>();
Iterator<String> itr = list.iterator();



LinkedList<String> list = new LinkedList<String>();
Iterator<String> itr = list.iterator();
```

Solution - Sub-interfaces

```
public interface ListADT<E>
```

Making `LinkedList` Iterable

```
public class LinkedList<E> implements ListADT<E> {

   private Listnode<E> head;
   private int numItems;


   public LinkedList() { ... }
   public void add(E item) { ... }
   public E get(int pos) { ... }
      .
      .
      .

   public Iterator<E> iterator() {



   }

}
```