Homework 3 due 10 pm Friday, February 10th

Program 1 due 10 pm Sunday, February 12th

Assignment questions? Teams, search first/then post on Piazza or consult with a TA during scheduled hours.

Last Week
Exceptions Review: throws and checked vs. unchecked and defining, Java Primitives vs. References Review, Chains of Linked Nodes

This Week
Read: continue Linked Lists

Chains of Linked Nodes
• Listnode class
• practice with chains of nodes

Java Visibility Modifiers

LinkedList Class

LinkedListIterator Class

Linked List Variations
• tail reference
• header node
• double linking
• circular linking

Next Week
Read: finish Linked Lists and start Complexity
Concept, big-O notation, analyzing algorithms practice, analyzing Java code
Recall Chain of Linked Nodes Data Structure

Listnode class

class Listnode<E> {
    private E data;
    private Listnode<E> next;

    public Listnode(E d)               { . . . }
    public Listnode(E d, Listnode<E> n){ . . . }

    public E getData()                 { return data; }
    public Listnode<E> getNext()       { return next; }
    public void setData(E d)           { data = d; }
    public void setNext(Listnode<E> n) { next = n; }
}

→ Show how the memory diagrams change as a result of executing the code beneath each:

```
head
head.setNext(head.getNext().getNext().getNext().getNext());
```

```
head
head.getNext().getNext().setNext(head);
```

```
head
curr
n1
n1.setNext(curr.getNext());
curr.setNext(n1);
```
Practice: Making a Chain of Nodes

Create a chain of Listnodes containing the Strings "yippie", "ki", and "yay" (as shown below) in as few statements as you can.
Practice: Traversing a Chain of Nodes

Assume `head` points to the first node in a chain of `Listnode`es containing `Strings`.

→ **Write a code fragment** that counts the number of strings in the chain of nodes.

```java
int count = 0;
```
Practice: Adding a Node at the Chain’s End

Assume head points to the first node in a chain of Listnodes containing Strings.

→ Write a code fragment that adds a node containing “rear” to the end of the chain of nodes. You may assume the chain has at least one item.
Practice: Removing a Node from a Chain

Assume \texttt{head} points to the first node in a chain of \texttt{Listnode}s containing \texttt{Strings}.

→ Write a code fragment that removes the third item from the chain of nodes. You may assume the chain has at least three items.

→ How would you generalize your code so it removes the Nth item from the chain of nodes?
Practice: Challenge Question

Assume \texttt{head} points to the first node in a chain of \texttt{Listnode}s containing \texttt{Strings}.

\textbf{Write a code fragment} that reverses the order of the nodes in the chain.
Java Visibility Modifiers

public    public class ArrayList<E>

private   private Object[] items

protected protected String name

package   class ListNode<E>
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
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

```java
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 vs. non-empty

Code Example

```java
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?

```java
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

```java
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() {

    }
}
```