CMSC 341

Linked Lists, Stacks and Queues
Implementing Your Own Linked List

- To create a doubly linked list as seen below
  - MyLinkedList class
  - Node class
  - LinkedListIterator class
  - Sentinel nodes at head and tail
Empty Linked List

- An empty double linked list with sentinel nodes.

![Diagram of an empty linked list with sentinel nodes.](image)
Inner classes

- Inner class objects require the construction of an outer class object before they are instantiated.
- Compiler adds an implicit reference to outer class in an inner class (MyArrayList.this).
- Good for when you need several inner objects to refer to exactly one outer object (as in an Iterator object).
Nested classes

- Considered part of the outer class, thus no issues of visibility.
- No reference to the outer class. If a nested (static) class has public accessibility, then it can be instantiated without the outer class.
- Making an inner class private means only the outer class may access the data fields within the nested class.
- Is Node a prime candidate for nested or inner class? public or private?
Implementation for MyLinkedList

1. Class declaration and nested Node class

```java
public class MyLinkedList<AnyType> implements Iterable<AnyType>
{
    // static key word makes Node a nested class
    private static class Node<AnyType>
    {
        public Node( AnyType d, Node<AnyType> p, 
                     Node<AnyType> n )
        { data = d; prev = p; next = n; }

        public AnyType data;
        public Node<AnyType> prev;
        public Node<AnyType> next;
    }
}
```
2. Data Fields and Accessors

```java
private int theSize;
// used to help iterator detect changes in List
private int modCount = 0;
private Node<AnyType> beginMarker; // head node
private Node<AnyType> endMarker; // tail node

public int size() {
    return theSize;
}
public boolean isEmpty() {
    return size() == 0;
}
```
3. Constructor(s)

public **MyLinkedList**()  
    { clear(); }  

    // Changes the size of this collection to zero.  
    public void **clear**()  
    {  
        beginMarker = new Node<AnyType>(null, null, null);  
        endMarker =  
            new Node<AnyType>(null, beginMarker, null);  
        beginMarker.next = endMarker;  
    }  
    
        theSize = 0;  
    modCount++;  
    }
4. More Accessors and Mutators

public boolean add( AnyType x )
    { add( size( ), x ); return true; }

public void add( int idx, AnyType x )
    { addBefore( getNode( idx ), x ); }

public AnyType get( int idx )
    { return getNode( idx ).data; }

public AnyType set( int idx, AnyType newVal )
    {
        Node<AnyType> p = getNode( idx );
        AnyType oldVal = p.data;
        p.data = newVal;
        return oldVal;
    }

public AnyType remove( int idx )
    { return remove( getNode( idx ) ); }
5. getNodeType Method

private Node<AnyType> getNodeType( int idx ) {
    Node<AnyType> p;
    if( idx < 0 || idx > size() )
        throw new IndexOutOfBoundsException();
    if( idx < size() / 2 ) {
        p = beginMarker.next;
        for( int i = 0; i < idx; i++ )
            p = p.next;
    } else {
        p = endMarker;
        for( int i = size(); i > idx; i-- )
            p = p.prev;
    }
    return p;
}
6. addBefore Method

```java
private void addBefore(Node<AnyType> p, AnyType x)
{
    Node<AnyType> newNode
        = new Node<AnyType>(x, p.prev, p);
    newNode.prev.next = newNode;
    p.prev = newNode;
    theSize++;
    modCount++;
}
```
private AnyType remove( Node<AnyType> p )
{
    p.next.prev = p.prev;
    p.prev.next = p.next;
    theSize--;
    modCount++;

    return p.data;
}

//required by the Iterable interface
public java.util.Iterator<AnyType> iterator( )
{
    return new LinkedListListIterator( );
}
import java.util.*;
private class LinkedListIterator<AnyType>
    implements Iterator<AnyType>
{
    private Node<AnyType> current = beginMarker.next;

    // used to check for modifications to List
    private int expectedModCount = modCount;
    private boolean okToRemove = false;

    public boolean hasNext() {
        return current != endMarker;
    }

    // continues on next slide...
public AnyType next( ) {  
    if( modCount != expectedModCount )  
        throw new ConcurrentModificationException( );

    if( !hasNext( ) )  
        throw new NoSuchElementException( );

    AnyType nextItem = current.data;
    current = current.next;
    okToRemove = true;
    return nextItem;

}  //continues on next slide...
public void remove()
{
    if ( modCount != expectedModCount )
        throw new ConcurrentModificationException();
    if ( !okToRemove )
        throw new IllegalStateException();
    MyLinkedList.this.remove(current.prev);
    okToRemove = false;
    ++expectedModCount;

} // end of remove Method

} // end of LinkedListIterator class

};//end of MyLinkedList class
Stacks

- A restricted list where insertions and deletions can only be performed at one location, the end of the list (top).

- LIFO – Last In First Out
  - Laundry Basket – last thing you put in is the first thing you remove
  - Plates – remove from the top of the stack and add to the top of the stack
Stack ADT

- Basic operations are push, pop, and top
- Stack Model
Adapting Lists to Implement Stacks

- Adapter Design Pattern
- Allow a client to use a class whose interface is different from the one expected by the client
- Do not modify client or class, write adapter class that sits between them
- In this case, the List is an adapter for the Stack. The client (user) calls methods of the Stack which in turn calls appropriate List method(s).
Adapter Model for Stack

Client (Stack user)

theStack.push(10)

Stack (adapter)

theList.add(0,10);

List (adaptee)
Queues

- Restricted List
  - only add to head
  - only remove from tail

- Examples
  - line waiting for service
  - jobs waiting to print

- Implement as an adapter of List
Queue ADT

- Basic Operations are enqueue and dequeue

Queue diagram with dequeue and enqueue arrows pointing in and out of the queue.
Adapter Model for Queue

Client (Queue user)
\[ \text{theQ.enqueue(10)} \]
Queue (adapter)
\[ \text{theList.add(theList.size()-1, 10)} \]
List (adapter)