Announcements

• Homework 1 is due now.
• Late penalty in effect
• Homework 2 released on website
  Due Oct. 6th at 5:40 PM (14 days)
Review

• Extra Big-Oh analysis example
• List Abstract Data Type
• Array Lists
• Linked Lists
Today’s Plan

• List code study
• MyArrayList
• MyLinkedList
• The Iterable interface
• Definition of Stack ADT
public class MyArrayList<AnyType> implements Iterable<AnyType> {

    private static final int DEFAULT_CAPACITY = 10;

    private AnyType [] theItems;
    private int theSize;
}
/**
 * Construct an empty ArrayList.
 */
public MyArrayList()
{
    clear();
}
/**
 * Change the size of this collection to zero.
 */
public void clear()
{
    theSize = 0;
    ensureCapacity(DEFAULT_CAPACITY);
}
/**
 * Checks if there is room for newCapacity items. If not
 * increases size of list
 */

@ SuppressWarnings("unchecked")
public void ensureCapacity( int newCapacity )
{
    if( newCapacity < theSize )
        return;

    AnyType [] old = theItems;
    theItems = (AnyType []) new Object[newCapacity];
    for( int i = 0; i < size( ); i++ )
        theItems[ i ] = old[ i ];
}
/**
 * Returns the number of items in this collection.
 */
public int size()
{
    return theSize;
}

/**
 * Returns true if this collection is empty.
 */
public boolean isEmpty()
{
    return size() == 0;
}
/**
 * Returns the item at position idx.
 */
public AnyType get( int idx )
{
    if( idx < 0 || idx >= size( ) )
        throw new ArrayIndexOutOfBoundsException( "Index " + idx + "; size " + size( ) );
    return theItems[ idx ];
}

/**
 * Changes the item at position idx.
 */
public AnyType set( int idx, AnyType newVal )
{
    if( idx < 0 || idx >= size( ) )
        throw new ArrayIndexOutOfBoundsException( "Index " + idx + "; size " + size( ) );
    AnyType old = theItems[ idx ];
    theItems[ idx ] = newVal;
    return old;
}
/* Adds an item to this collection at the end. */
public boolean add( AnyType x )
{
    add( size( ), x );
    return true;
}

/**
 * Adds an item to this collection, at the specified index.
 */
public void add( int idx, AnyType x )
{
    if( theItems.length == size( ) )
        ensureCapacity( size( ) * 2 + 1 );

    for( int i = theSize; i > idx; i-- )
        theItems[ i ] = theItems[ i - 1 ];

    theItems[ idx ] = x;
    theSize++;
}
/**
 * Removes an item from this collection.
 */

public AnyType remove( int idx )
{
    AnyType removedItem = theItems[ idx ];

    for( int i = idx; i < size( ) - 1; i++ )
        theItems[ i ] = theItems[ i + 1 ];
    theSize--;  

    return removedItem;
}
**
* LinkedList class implements a doubly-linked list.
*/
public class MyLinkedList<AnyType> implements Iterable<AnyType> {

    private int theSize;
    private Node<AnyType> beginMarker;
    private Node<AnyType> endMarker;
}
/**
 * LinkedList class implements a doubly-linked list.
 */
public class MyLinkedList<AnyType> implements Iterable<AnyType> {

/**
 * This is the doubly-linked list node.
 */
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;
}

private int theSize;
private Node<AnyType> beginMarker;
private Node<AnyType> endMarker;
}
/**
 * Construct an empty LinkedList.
 */
public MyLinkedList()
{
    clear();
}

/**
 * Change 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;
}
/**
 * Returns the number of items in this collection.
 */

public int size()
{
    return theSize;
}

public boolean isEmpty()
{
    return size() == 0;
}
/**
 * Adds an item to this collection, at the end.
 */
public boolean add( AnyType x ) {
    add( size( ), x );
    return true;
}

/**
 * Adds an item to this collection, at specified position.
 * Items at or after that position are slid one position higher.
 */
public void add( int idx, AnyType x ) {
    addBefore( getNode( idx, 0, size( ) ), x );
}

/**
 * Adds an item to this collection, at specified position p.
 * Items at or after that position are slid one position higher.
 */
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++;
}
/**
 * Returns the item at position idx.
 */
public AnyType get( int idx )
{
    return getNode( idx ).data;
}

/**
 * Changes the item at position idx.
 */
public AnyType set( int idx, AnyType newVal )
{
    Node<AnyType> p = getNode( idx );
    AnyType oldVal = p.data;
    p.data = newVal;
    return oldVal;
}
/**
 * Removes an item from this collection.
 */
public AnyType remove( int idx )
{
    return remove( getNode( idx ) );
}

/**
 * Removes the object contained in Node p.
 */
private AnyType remove( Node<AnyType> p )
{
    p.next.prev = p.prev;
    p.prev.next = p.next;
    theSize--;

    return p.data;
}
/**
 * Gets the Node at position idx,
 * which must range from 0 to size( ) - 1.
 */

private Node<AnyType> getNode( int idx )
{
    return getNode( idx, 0, size( ) - 1 );
}
Iterables

- The Iterable interface standardizes efficient navigation of Collection classes
- Creates Iterator object, which has methods hasNext(), next()
- Enhanced for loop:
  ```java
  for (Object x : list)
      // Do something with x
  ```
We explicitly code our ADTs as Object classes; we don’t have to!

- e.g., you should use ADTs even when programming machine language

- We can do list operations on arrays, but thinking in terms of lists is cleaner
Stacks

• A Stack is an ADT very similar to a list
• Can be implemented with a list, but limited to some O(1) operations
• Yet many important and powerful algorithms use stacks
Stack Definition

- Essentially a very restricted List
- Two (main) operations:
  - Push(AnyType x)
  - Pop(AnyType x)
- Analogy – Cafeteria Trays, PEZ
Stack Applications

• Recursion
• Parsing text: infix vs. postfix
• Syntax checking ( ), { }, “”
Reading

- Weiss Ch. 3 up to 3.7