



Linked Lists

Problem Set: Assignment #2 due Thursday, February 16

E-1



From Last Time: OOP Review

- These should be familiar terms by now:
 - Instance variables and methods (definition and invocation)
 - Class variables and methods (definition and invocation)
- Example with Songs and Array of Songs
- Quiz: How would you define

```
/** returns a new Song object
 * that is a copy of the input Song S */
public static Song copySong (Song S) {

}
```

E-2



Instance Method compareTo

```
/** Compares this song object to argument Song object.
 * Returns -1 if ...
 * Returns 0 if ...
 * Returns 1 if ...
 */
public int compareTo (Song S) {

}
```

E-3



What is an Abstract Data Type (ADT)?

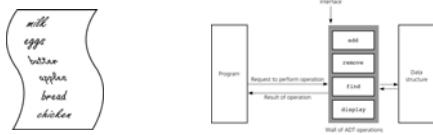
An **Abstract Data Type (ADT)** is a contract that specifies the behavior of methods that operate on a data structure without exposing the representation of the data structure.

In general, there are many different implementations of a given ADT.

E-4



LinkedList ADT

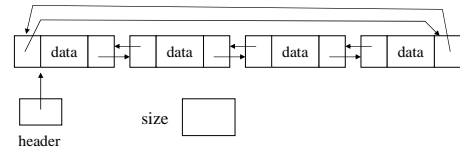


- A simple way of organizing data, like a shopping list
- **Stretchable:**
Unlike arrays, no need to know how long a list will be when you create it
- **Slower:**
Less efficient than arrays when you are searching for a particular item

E-5



LinkedList Intuition

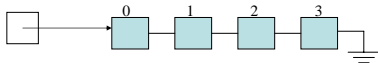


- Java's **LinkedList** class:
Generic list of objects and methods for efficient access
- Access through the header node and by index
- Indices start at 0, like arrays
- Data are objects (i.e., pointers)

E-6



Java LinkedList Class



Abstract picture to be used:
simple and clear about accessibility

To use it you need to start your code by

```
import java.util.*;
```

To get a new list:

```
LinkedList L = new LinkedList();
```



E-7



Helper Methods

```
public int size()
    what you would expect
    L.size()
public void clear()
    empties the list of objects (garbage collector will take care)
    L.clear()
public boolean isEmpty()
    checks if there are any elements in list
    if (L.isEmpty()) ...
public String toString()
    the usual
    System.out.println(L);
```

Axioms (simple statements that explain behavior):

- (new LinkedList()).size() => 0
- (new LinkedList()).isEmpty() => true
- (anyList.clear()).isEmpty() => true

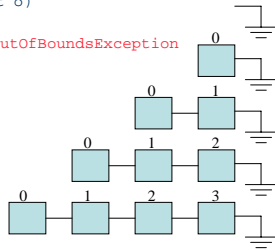
E-8



Adding Nodes

```
public void add (int index, Object o)
    indices change automatically!
    if index > size() throws IndexOutOfBoundsException
public void addFirst (Object o)
public void addLast (Object o)
```

```
LinkedList L1 = new LinkedList();
L1.add(0, "eggs");
L1.addFirst(" milk ");
L1.addLast("bread");
L1.add(2, "chicken");
System.out.println("contents of L1: " + L1);
```



To add an integer you need to wrap it into an Object first
addFirst(new Integer(5));

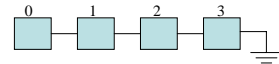
E-9



Getting List Items (Objects)

```
public Object getFirst ()
public Object getLast ()
public Object get (int index)
```

```
int i;
LinkedList L2 = new LinkedList();
for (i = L1.size()-1; i >= 0; i--)
    L2.add(L1.get(i));
System.out.println("contents of L2: " + L2);
```



new contents of L2: [bread, chicken, eggs, milk]

E-10



Set and Remove Items (Objects)

```
public Object set (int index, Object o)
public Object remove (int index)
public Object removeFirst ()
public Object removeLast ()
```

```
L1.remove(1);
L1.set(2, "beer");
L1.set(1, L1.removeFirst());
L1.addFirst(L1.getLast());
L1.add(1, "butter");
System.out.println("new contents of L1: " + L1);
=> new contents of L1: [ milk, butter, chicken, milk ]
      (what happened to the beer?)
```

E-11



The Need For Correct Casting

```
LinkedList L3 = new LinkedList();
L3.add(new Song("Happy Birthday", 48));
L3.add(new Song("Happy Birthday", 48));
L3.add(new Song("Mass in B Minor", 241));
System.out.println("contents of L3: " + L3);
```

```
Song birthday = new Song("Happy Birthday", 48);
for (int i = 0; i < L3.size(); i++)
    if (L3.get(i).equals(birthday))
        System.out.println("birthday found at index " + i);
    else
        System.out.println("birthday not found ");
```

The if-condition will fail. Why? b/c birthday is a Song, not an Object

- birthday not found
- birthday not found
- birthday not found

E-12



Casting Objects

```
System.out.println("second search for birthday:");
birthday = new Song("Happy Birthday", 48);
for (int i = 0; i < L3.size(); i++)
    if (((Song)L3.get(i)).equals(birthday))
        System.out.println("birthday found at index " + i);
```

Equals compares Songs, which is fine. Finds both birthdays

- second search for birthday:
- birthday found at index 0
- birthday found at index 1

```
System.out.println("third search for birthday:");
birthday = (Song)L3.get(1);
for (i = 0; i < L3.size(); i++)
    if (L3.get(i).equals(birthday))
        System.out.println("birthday found at index " + i);
```

Only birthday at location 1 will be found. Why?

- third search for birthday:
- birthday found at index 1

E-13



Generics

```
class LinkedList<E>
```

E is a type parameter

Example methods

```
public void add (int index, E element)
public void addFirst (E o)
public void addLast (E o)
public E get(int index)
public E remove(int index)
public E set(int index, E element)
```

E-14



Using the Generic LinkedList class

```
LinkedList L1 = new LinkedList();
L1.add("CS230 rocks!");
int n = ((String)L1.getFirst()).length();
```

Compare

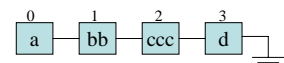
```
LinkedList<String> L1 = new LinkedList<String>();
L1.add("CS230 rocks!");
int n = L1.getFirst().length();
```

E-15



Beware...

```
/** Tries to remove all nodes from L that contain
 * a string with fewer than three characters
 */
public static void removeSmallStrings0 (LinkedList<String> L)
{
    for (int i = 0; i < L.size(); i++)
        if (L.get(i).length() < 3)
            L.remove(i);
}
```



Buggy! How do you fix it?

E-16



Another Attempt

```

/** removes all nodes from L that contain a String with fewer than
 * three characters, using a while loop */
public static void removeSmallStrings (LinkedList<String> L) {
    int i = 0;
    while (i < L.size()) {
        if (L.get(i).length() < 3)
            L.remove(i);
        else
            i++;
    }
}

/** returns a new list that contains all of the nodes from L that
 * have at least three characters -- keeps list L intact */
public static LinkedList<String> collectSmallStrings (LinkedList<String> L)
{
    LinkedList<String> L0 = new LinkedList<String>();
    for (int i = 0; i < L.size(); i++)
        if (L.get(i).length() > 2)
            L0.add(L.get(i));
    return L0;
}

```

E - 17

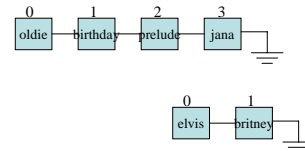


Append and Share (Song lists)

```

/** appends the list L2 onto the end of L1
 */
public static void append (LinkedList<Song> L1,
    LinkedList<Song> L2) {
    for (int i = 0; i < L2.size(); i++)
        L1.addLast(L2.get(i));
    // The nodes of L1 & L2 are now shared!
}

```



E - 18

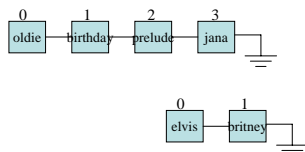


Append and Share (any lists)

```

/** appends the list L2 onto the end of L1
 */
public static <E> void append (LinkedList<E> L1,
    LinkedList<E> L2) {
    for (int i = 0; i < L2.size(); i++)
        L1.addLast(L2.get(i));
    // The nodes of L1 & L2 are now shared!
}

```



E - 19

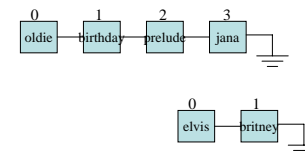


Append and Do Not Share

```

/** appends a copy of the nodes of L2 (Song objects)
 * to the end of L1 */
public static void copyAppend (LinkedList<Song> L1,
    LinkedList<Song> L2) {
    for (int i = 0; i < L2.size(); i++)
        L1.addLast(copySong(L2.get(i)));
}

```



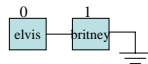
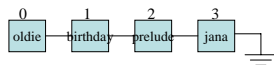
E - 20



Insert

```
/** inserts nodes of L2 into the list L1, starting at the input
    index
 */
public static <E> void insertList (LinkedList<E> L1,
                                   LinkedList<E> L2, int index) {

    for (int i = 0; i < L2.size(); i++)
        L1.add(index + i, L2.get(i));
}
```



E-21