



CSE 143 Java

Linked Lists

Reading: Ch. 23

7/23/2003 (c) 2001-3, University of Washington 15-1

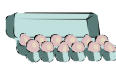
Review: List Implementations

- The external interface is already defined
- Implementation goal: implement methods “efficiently”
- **ArrayList** approach: use an array with extra space internally
- **ArrayList** efficiency
 - Iterating, indexing (get & set) is fast
Typically a one-liner
 - Adding at end is fast, except when we have to grow
 - Adding or removing in the middle is slow: requires sliding all later elements

7/23/2003 (c) 2001-3, University of Washington 15-2

A Different Strategy: Lists via Links

Instead of packing all elements together in an array,



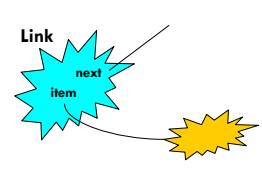
create a *linked chain* of all the elements



7/23/2003 (c) 2001-3, University of Washington 15-3

Links

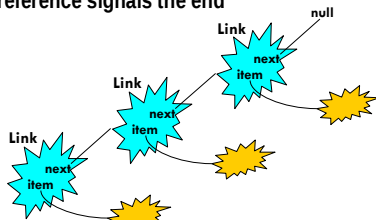
- For each element in the list, create a **Link** object
- Each **Link** points to the *data item* (element) at that position, and also points to the *next Link* in the chain



7/23/2003 (c) 2001-3, University of Washington 15-4

Linked Links

- Each Link points to the next
- No limit on how many can be linked!
- A null reference signals the end



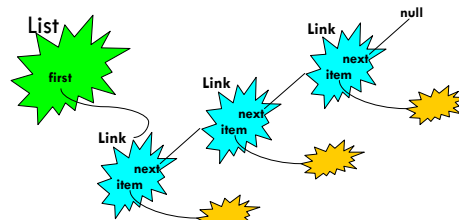
7/23/2003

(c) 2001-3, University of Washington

15-5

Linked List

- The List has a reference to the first Link
- Altogether, the list involves 3 different object types



7/23/2003

(c) 2001-3, University of Washington

15-6

Link Class: Data

```
/** Link for a simple list */
public class Link {
    public Object item;           // data associated with this link
    public Link next;            // next Link, or null if no next link
    //no more instance variables
    //but maybe some methods
} //end Link
```



Note 1: This class does NOT represent the chain, only one link of a chain
 Note 2: "public" violates usual rules – but appropriate in this context
 Note 3: The links are NOT part of the data. The data is totally unaware that it is part of a chain.

7/23/2003

(c) 2001-3, University of Washington

15-7

Link Constructor

```
/** Link for a simple list */
public class Link {
    public Object item;           // data associated with this link
    public Link next;            // next Link, or null if none

    /** Construct new link with given data item and next link (or null if none) */
    public Link(Object item, Link next) {
        this.item = item;
        this.next = next;
    }
    ...
}
```



7/23/2003

(c) 2001-3, University of Washington

15-8

Exercise: Add a Node (1)

- Suppose we've got a linked list containing "lion", "tiger", and "bear" in that order, with a variable pointing to the head of the list

Link head; // first link in the list, or null if list is empty

- Draw a picture of the list

7/23/2003

(c) 2001-3, University of Washington

15-9

Exercise: Add a Node (2)

- Now, write the code needed to insert "wolf" between "tiger" and "bear"

7/23/2003

(c) 2001-3, University of Washington

15-10

Exercise: Delete a Node (1)

- Suppose we've got a list containing "IBM", "Dell", "Compaq", and "Apple" in that order

Link head; // first link in the list, or null if list is empty

- Draw a picture

7/23/2003

(c) 2001-3, University of Washington

15-11

Exercise: Delete a Node (2)

- Now, write the code needed to delete "Compaq" from the list

7/23/2003

(c) 2001-3, University of Washington

15-12

LinkedList Data

```
/** Simple version of LinkedList for CSE143 lecture example */
public class SimpleLinkedList implements List {
    // instance variables
    private Link first; // first link in the list, or null if list is empty
    ...
}
```



7/23/2003

(c) 2001-3, University of Washington

15-13

LinkedList Data & Constructor

```
/** Simple version of LinkedList for CSE143 lecture example */
public class SimpleLinkedList implements List {
    // instance variables
    private Link first; // first link in the list, or null if list is empty
    ...

    // construct new empty list
    public SimpleLinkedList() {
        first = null; // no links yet!
    }
    ...
}
```



7/23/2003

(c) 2001-3, University of Washington

15-14

List Interface (review)

• Operations to implement:

```
int size()
boolean isEmpty()
boolean add(Object o)
boolean addAll(Collection other)
void clear()
Object get(int pos)
boolean set(int pos, Object o)
int indexOf(Object o)
boolean contains(Object o)
Object remove(int pos)
boolean remove(Object o)
boolean add(int pos, Object o)
Iterator iterator()
```

• What isn't included here??

7/23/2003

(c) 2001-3, University of Washington

15-15

Method add (First Try)

```
public boolean add(Object o) {
    // create new link and place at end of list:
    Link newLink = new Link(o, null);
    // find last link in existing chain: it's the one whose next link is null:
    Link p = first;
    while (p.next != null) {
        p = p.next;
    }
    // found last link; now add the new link after it:
    p.next = newLink;
    return true; // we changed the list => return true
}
```



7/23/2003

(c) 2001-3, University of Washington

15-16

Draw the Picture

- Client code:

```
LinkedList vertexes = new SimpleLinkedList();
Point2D p1 = new Point2D.Double(100.0, 50.0);
Point2D p2 = new Point2D.Double( 250, 310);
Point2D p3 = new Point2D.Double(90, 350.0);
vertexes.add(p1);
vertexes.add(p2);
vertexes.add(p3);
vertexes.add(p1);
```

7/23/2003

(c) 2001-3, University of Washington

15-17

Problems with naïve *add* method

- Inefficient: requires traversal of entire list to get to the end
 - One loop iteration per link
 - Gets slower as list gets longer
 - Solution??
- Buggy: fails when adding first link to an empty list
 - Check the code: where does it fail?
 - Solution??

7/23/2003

(c) 2001-3, University of Washington

15-18

Improvements to naïve *add* method

- Inefficient: requires traversal of entire list to get to the end
 - A solution: Change *LinkedList* to keep a pointer to *last* link as well as the *first*
- Buggy: fails when adding first link to an empty list
 - A solution: check for this case and execute special code
- Q: “Couldn’t we?” Answer: “probably”. There are many ways link lists could be implemented

7/23/2003

(c) 2001-3, University of Washington

15-19

List Data & Constructor (revised)

```
public class SimpleLinkedList implements List {
    // instance variables
    private Link first;           // first link in the list, or null if list is empty
    private Link last;           // last link in the list, or null if list is empty
    ...

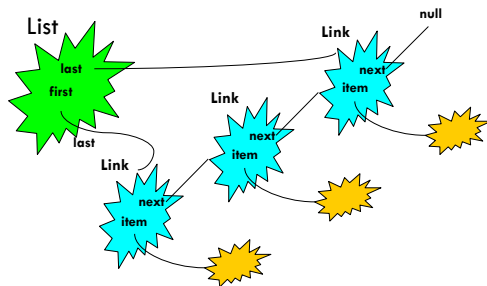
    // construct new empty list
    public SimpleLinkedList() {
        first = null;           // no links yet!
        last = null;           // no links yet!
    }
    ...
}
```

7/23/2003

(c) 2001-3, University of Washington

15-20

Link List with *last*



7/23/2003

(c) 2001-3, University of Washington

15-21

Method *add* (Final Version)

```
public boolean add(Object o) {
    // create new link to place at end of list:
    Link newLink = new Link(o, null);
    // check if adding the first link
    if (first == null) {
        // we're adding the first link
        first = newLink;
    } else {
        // we have some existing links; add the new link after the old last link
        last.next = newLink;
    }
    // update the last link
    last = newLink;
    return true; // we changed the list => return true
}
```

7/23/2003

(c) 2001-3, University of Washington

15-22

Method *size()*

- First try it with this restriction: you can't add or redefine instance variables

- Hint: count the number of links in the chain

```
/** Return size of this list */
public int size() {
    int count = 0;
```

```
        return count;
    }
```

- Critique?

7/23/2003

(c) 2001-3, University of Washington

15-23

Method *size* (faster)

- Add an instance variable to the list class

```
int numLinks; // number of links in this list
```

- Add to constructor:

```
numLinks = 0;
```

- Add to method *add*:

```
numLinks ++;
```

- Method *size*

```
/** Return size of this list */
public int size() {
    return numLinks;
}
```

- Critique?



7/23/2003

(c) 2001-3, University of Washington

15-24

clear

- **Simpler than with arrays or not?**

```
/** Clear this list */
public void clear() {
    first = null;
    last = null;
    numLinks = 0;
}
```

- **No need to "null out" the elements themselves**
 - **Garbage Collector will reclaim the Link objects automatically**
(Some GCs might reclaim the objects quicker if we did null out the links, but good ones shouldn't need this)
- **Note that this is subtly from the situation with arrays!**

7/23/2003

(c) 2001-3, University of Washington

15-25

get

```
/** Return object at position pos of this list. 0 <= pos < size, else IndexOutOfBoundsException */
public Object get(int pos) {
    if (pos < 0 || pos >= numLinks) {
        throw new IndexOutOfBoundsException();
    }
    // search for pos'th link
    Link p = first;
    for (int k = 0; k < pos; k++) {
        p = p.next;
    }
    // found it; now return the element in this link
    return p.item;
}
```

- **Critique?**
- **DO try this at home. Try "set" too**

7/23/2003

(c) 2001-3, University of Washington

15-26

add and remove at given position

- **Observation: to *add* a link at position k, we need to change the next pointer of the link at position k-1**



- **Observation: to *remove* a link at position k, we need to change the next pointer of the link at position k-1**



7/23/2003

(c) 2001-3, University of Washington

15-27

Helper for add and remove

- **Possible helper method: get link given its position**

```
// Return the link at position pos
// precondition (unchecked): 0 <= pos < size
private Link getLinkAtPos(int pos) {
    Link p = first;
    for (int k = 0; k < pos; k++) {
        p = p.next;
    }
    return p;
}
```

- **Use this in get, too**
- **How is this different from the get(pos) method of the List?**

7/23/2003

(c) 2001-3, University of Washington

15-28

remove(pos): Study at Home!

```
/** Remove the object at position pos from this list. 0 <= pos < size, else IndexOutOfBoundsException */
public Object remove(int pos) {
    if (pos < 0 || pos >= numLinks) { throw new IndexOutOfBoundsException(); }
    Object removedElem;
    if (pos == 0) {
        removedElem = first.item;           // remember removed item, to return it
        first = first.next;                 // remove first link
        if (first == null) { last = null; }  // update last, if needed
    } else {
        Link prev = getLinkAtPos(pos-1);    // find link before one to remove
        removedElem = prev.next.item;       // remember removed item, to return it
        prev.next = prev.next.next;         // splice out link to remove
        if (prev.next == null) { last = prev; } // update last, if needed
    }
    numLinks--;                             // remember to decrement the size!
    return removedElem;
}
```

7/23/2003

(c) 2001-3, University of Washington

15-29

add(pos): Study at Home!

```
/** Add object o at position pos in this list. 0 <= pos <= size, else IndexOutOfBoundsException */
public boolean add(int pos, Object o) {
    if (pos < 0 || pos >= numLinks) { throw new IndexOutOfBoundsException(); }
    if (pos == 0) {
        first = new Link(o, first);        // insert new link at the front of the chain
        if (last == null) { last = first; } // update last, if needed
    } else {
        Link prev = getLinkAtPos(pos-1);    // find link before one to insert
        prev.next = new Link(o, prev.next); // splice in new link between prev & prev.next
        if (last == prev) { last = prev.next; } // update last, if needed
    }
    numLinks++;                             // remember to increment the size!
    return true;
}
```

7/23/2003

(c) 2001-3, University of Washington

15-30

Implementing iterator()

- To implement an iterator, could do the same thing as with SimpleArrayLists: return an instance of SimpleListIterator
- Recall: SimpleListIterator tracks the List and the position (index) of the next item to return
 - How efficient is this for LinkedLists?
 - Can we do better?

7/23/2003

(c) 2001-3, University of Washington

15-31

Summary

- SimpleLinkedList presents same illusion to its clients as SimpleArrayList
- Key implementation ideas:
 - a chain of links
- Different efficiency trade-offs than SimpleArrayList
 - must search to find positions, but can easily insert & remove without growing or sliding
 - get, set a lot slower
 - add, remove faster (particularly at the front): no sliding required



7/23/2003

(c) 2001-3, University of Washington

15-32