More Linked Structures

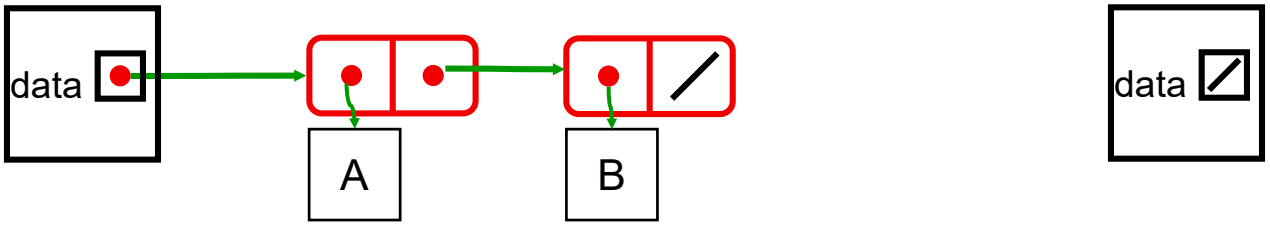
Lectures 13-14

Menu

- Linked structures for implementing Collections
- A collection class – Linked List
- Linked List methods

How do you make a good list class

- Must have an object that represent the empty list as an object
  - separate “header” object to represent a list



List using linked nodes with header

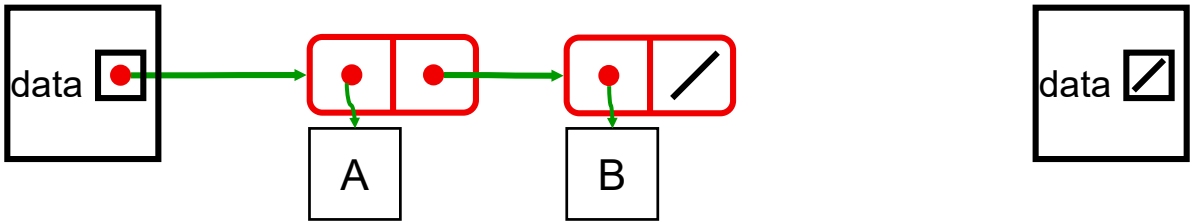
- LinkedList extends AbstractList
- Has fields for linked list of Nodes and count
- Has an inner class: Node, with public fields
- get(index), set(index, item),
  - loop to index'th node, then get or set value
- add(index, item), remove(index):
  - deal with special case of index == 0
  - loop along list to node one before index'th node (Why?), then add or remove
  - check if go past end of list
- remove(item),
  - deal with special case of item in first node (i.e. conversion into empty set after removal)
  - loop along list to node one before node containing item (Why?), then remove
  - check if go past end of list

A Linked List class:

```
public class LinkedList <E> extends AbstractList <E> {
    private Node<E> data;
    private int count;
    public LinkedList(){
        data = null;
        count = 0;
    }
    /** Inner class:  Node */
    private class Node <E> {
        public E value;
        public Node<E> next;
        public Node(E val, Node<E> node){
            value = val;
            next = node;
        }
    }
```

Linked List: get

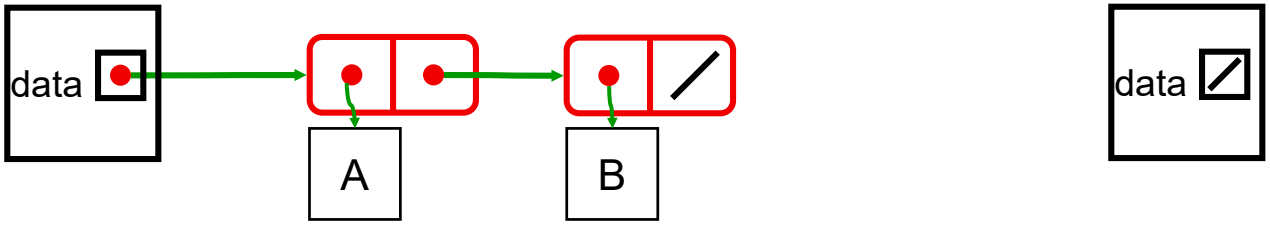
```
public E get(int index){
    if (index < 0) throw new IndexOutOfBoundsException();
    Node<E> node=data;
    int i = 0;  // position of node
    while (node!=null && i++ < index)  node=node.next;
    if (node==null)   throw new IndexOutOfBoundsException();
    return node.value;
}
```



Linked List: set

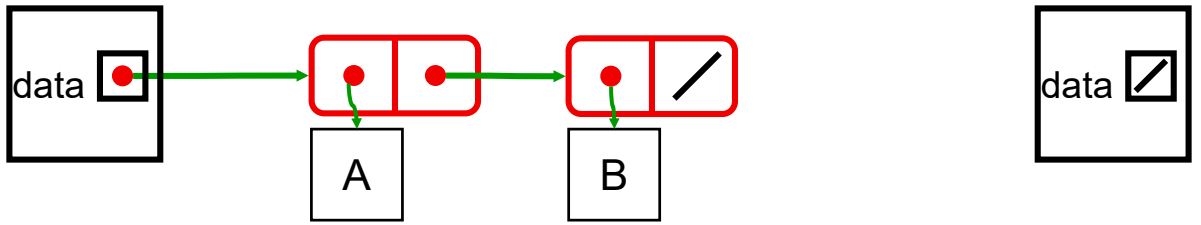
```
public E set(int index, E value){
    if (index < 0) throw new IndexOutOfBoundsException();
    Node<E> node=data;
    int i = 0;  // position of node
    while (node!=null && i++ < index) node=node.next;
    if (node==null)   throw new IndexOutOfBoundsException();
    E ans = node.value;
    node.value = value;
    return ans;
}
```

Same as get



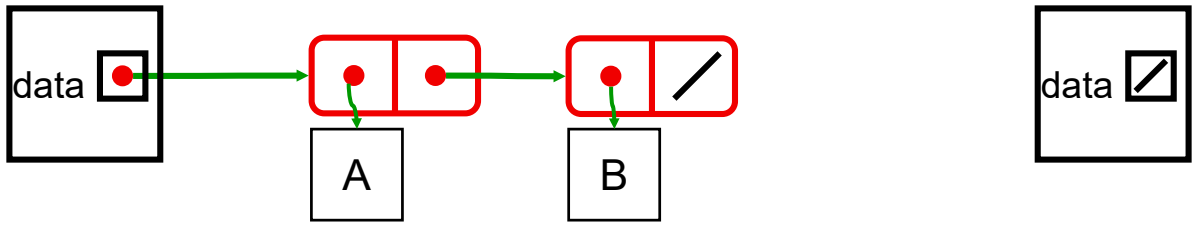
Linked List: add

```
public void add(int index, E item){
    if (item == null) throw new IllegalArgumentException();
    if (index==0){              // add at the front.
        data = new Node(item, data);
        count++;
        return;
    }
    Node<E> node=data;
    int i = 1;                  // position of next node
    while (node!=null && i++ < index) node=node.next;
    if (node == null) throw new IndexOutOfBoundsException();
    node.next = new Node(item, node.next);
    count++;
    return;
}
```



Linked List: remove

```
public boolean remove (Object item){
    if (item==null || data==null) return false;
    if (item.equals(data.value))   // remove the front item.
        data = data.next;
    else {          // find the node just before a node containing the item
        Node<E> node = data;
        while (node.next!=null && !node.next.value.equals(item))
            node=node.next;
        if (node.next==null)   return false;   // off the end
        node.next = node.next.next;   // splice the node out of the list
    }
    count--;
    return true;
}
```



Linked Collections: Cost

- Linked structures allow fast insertion and deletion
  Does it help?

Linked List:

- Cost of get / set:
- Cost of insert:
- Cost of remove:

Linked Set (items in sorted order):

- Cost of contains:
- Cost of insert:
- Cost of remove

No advantage to Linked List?

Menu

- Linked structures for implementing Collections
- A collection class – Linked List
- Linked List methods