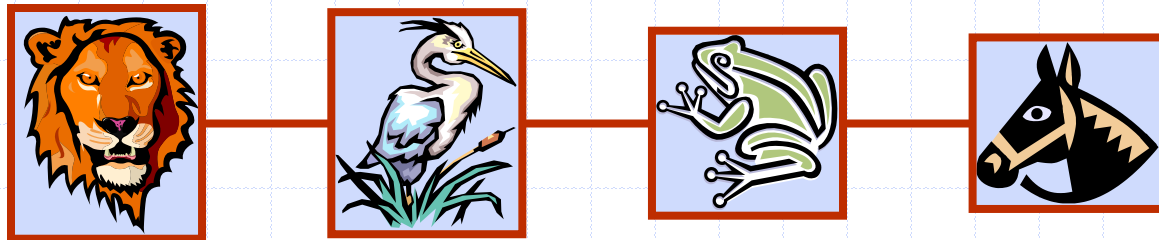


Presentation for use with the textbook **Data Structures and Algorithms in Java, 6<sup>th</sup> edition**, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

# Singly Linked Lists

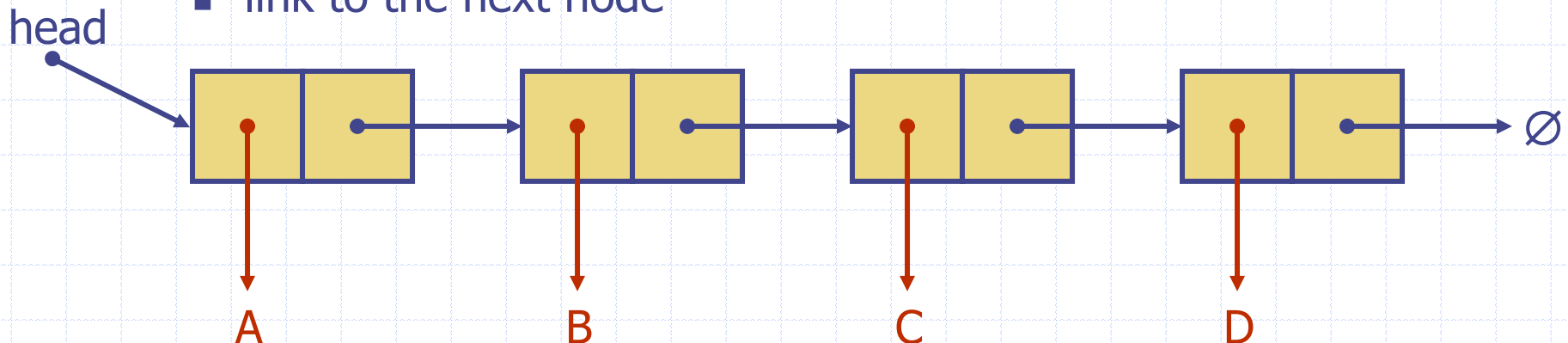
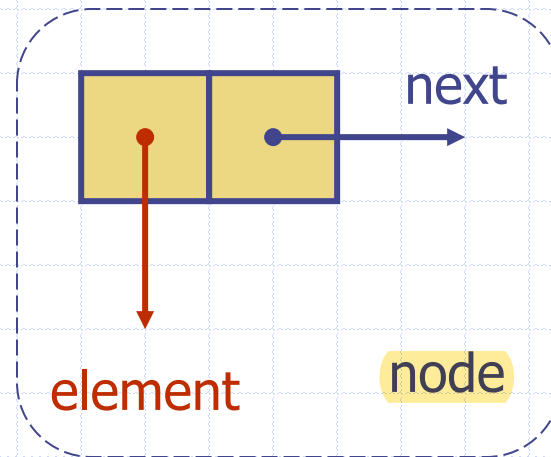


# Singly Linked List

◆ A singly linked list is a concrete data structure consisting of a sequence of nodes, starting from a head pointer

◆ Each node stores

- element
- link to the next node



# A Nested Node Class

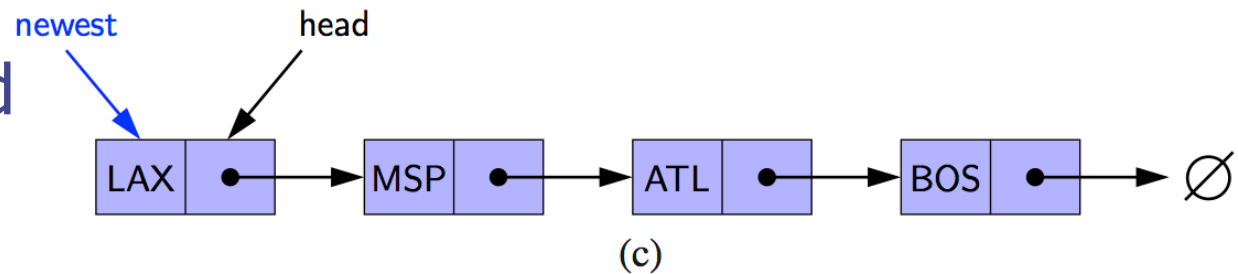
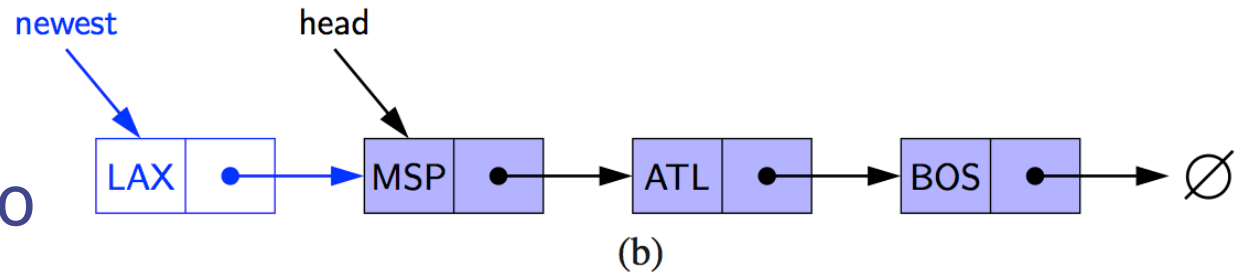
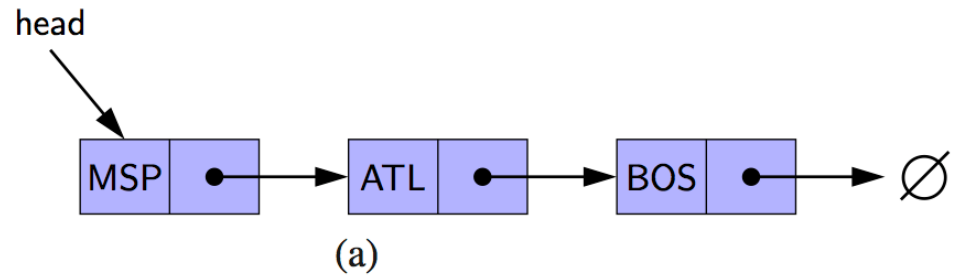
```
1  public class SinglyLinkedList<E> {
2      //----- nested Node class -----
3      private static class Node<E> {
4          private E element;           // reference to the element stored at this node
5          private Node<E> next;        // reference to the subsequent node in the list
6          public Node(E e, Node<E> n) {
7              element = e;
8              next = n;
9          }
10         public E getElement() { return element; }
11         public Node<E> getNext() { return next; }
12         public void setNext(Node<E> n) { next = n; }
13     } //----- end of nested Node class -----
    ... rest of SinglyLinkedList class will follow ...
```

# Accessor Methods

```
1  public class SinglyLinkedList<E> {  
...  (nested Node class goes here)  
  
14  // instance variables of the SinglyLinkedList  
15  private Node<E> head = null;           // head node of the list (or null if empty)  
16  private Node<E> tail = null;          // last node of the list (or null if empty)  
17  private int size = 0;                  // number of nodes in the list  
18  public SinglyLinkedList() { }           // constructs an initially empty list  
19  // access methods  
20  public int size() { return size; }  
21  public boolean isEmpty() { return size == 0; }  
22  public E first() {                     // returns (but does not remove) the first element  
23      if (isEmpty()) return null;  
24      return head.getElement();  
25  }  
26  public E last() {                       // returns (but does not remove) the last element  
27      if (isEmpty()) return null;  
28      return tail.getElement();  
29  }
```

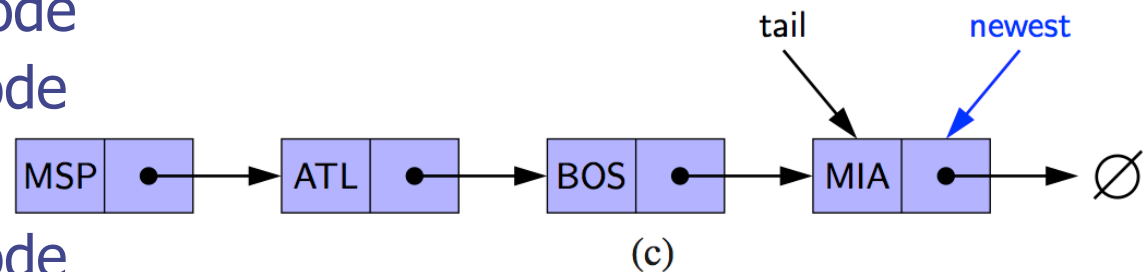
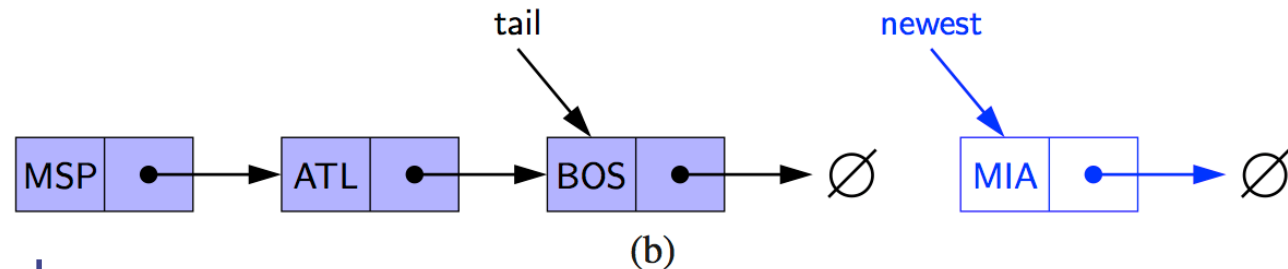
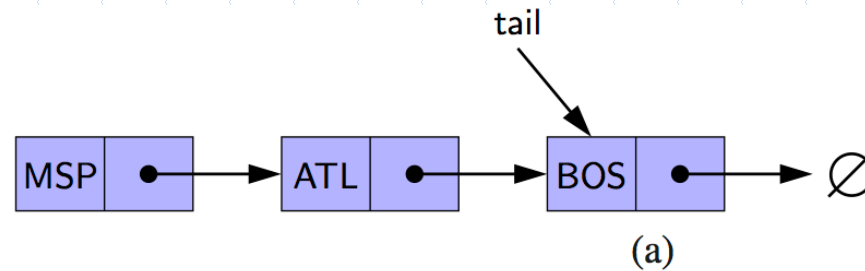
# Inserting at the Head

- Allocate new node
- Insert new element
- Have new node point to old head
- Update head to point to new node



# Inserting at the Tail

- Allocate a new node
- Insert new element
- Have new node point to null
- Have old last node point to new node
- Update tail to point to new node

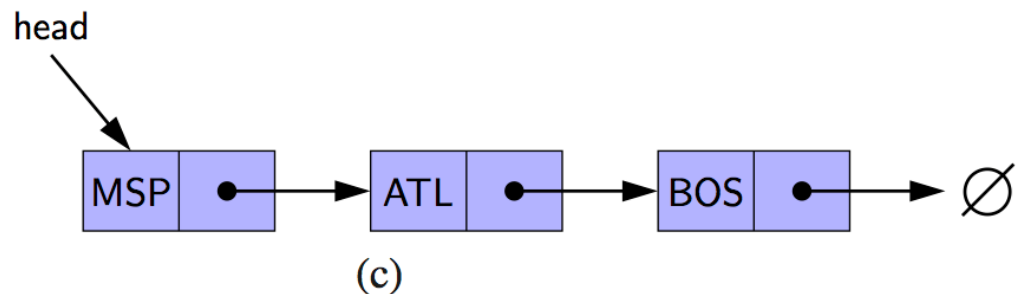
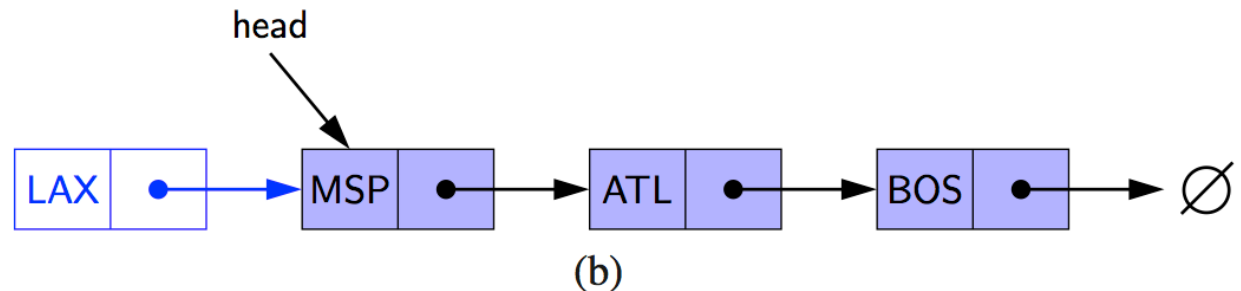
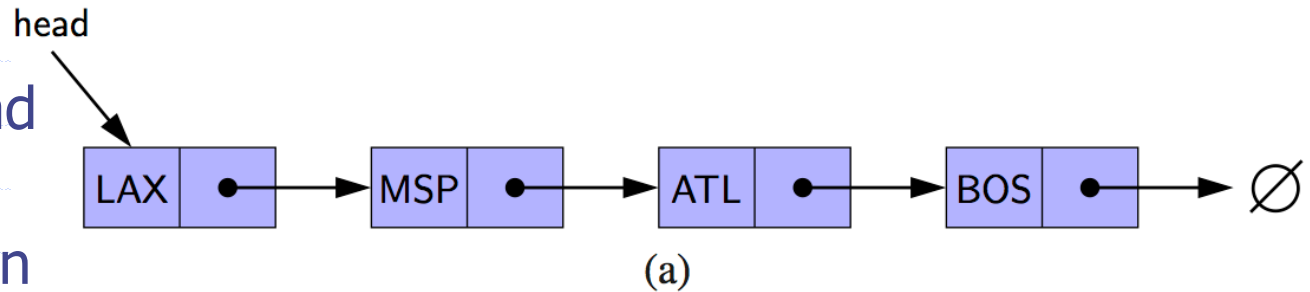


# Java Methods

```
31  public void addFirst(E e) {           // adds element e to the front of the list
32      head = new Node<>(e, head);       // create and link a new node
33      if (size == 0)
34          tail = head;                 // special case: new node becomes tail also
35      size++;
36  }
37  public void addLast(E e) {            // adds element e to the end of the list
38      Node<E> newest = new Node<>(e, null); // node will eventually be the tail
39      if (isEmpty())
40          head = newest;                 // special case: previously empty list
41      else
42          tail.setNext(newest);          // new node after existing tail
43      tail = newest;                     // new node becomes the tail
44      size++;
45  }
```

# Removing at the Head

- Update head to point to next node in the list
- Allow garbage collector to reclaim the former first node



# Java Method

```
46  public E removeFirst() {           // removes and returns the first element
47      if (isEmpty()) return null;    // nothing to remove
48      E answer = head.getElement();
49      head = head.getNext();         // will become null if list had only one node
50      size--;
51      if (size == 0)
52          tail = null;               // special case as list is now empty
53      return answer;
54  }
55 }
```

# Removing at the Tail

- Removing at the tail of a singly linked list is not efficient!
- There is no constant-time way to update the tail to point to the previous node

