Fundamentals of Data Structure

Mahesh Shirole

VJTI, Mumbai-19

Slides are prepared from

1. Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

2.Data Structures and Algorithms in Java, by Robert Lafore, Second Edition, Sams Publishing

Linked List

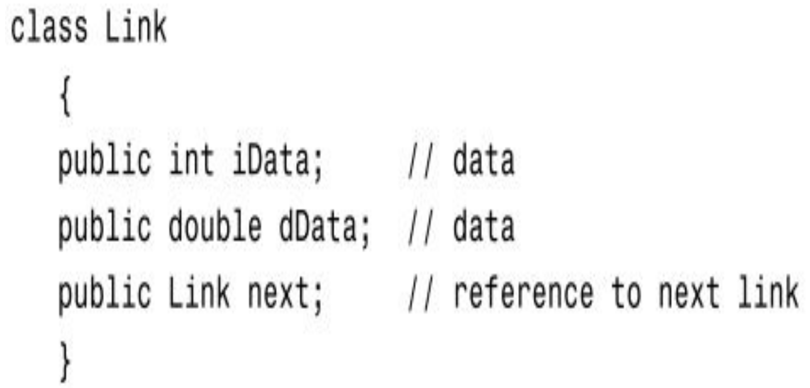
- Linked lists are probably the second most commonly used *general purpose storage structures* after arrays
- The linked list is a versatile mechanism **suitable for** use in many kinds of *general-purpose databases*
- In a linked list, each data item is embedded in a link.
- A link is an object of a class called something like **Link**.
- Each Link object contains a reference (usually called next) to the next link in the list.
- A field in the list itself contains a reference to the first link

Linked List

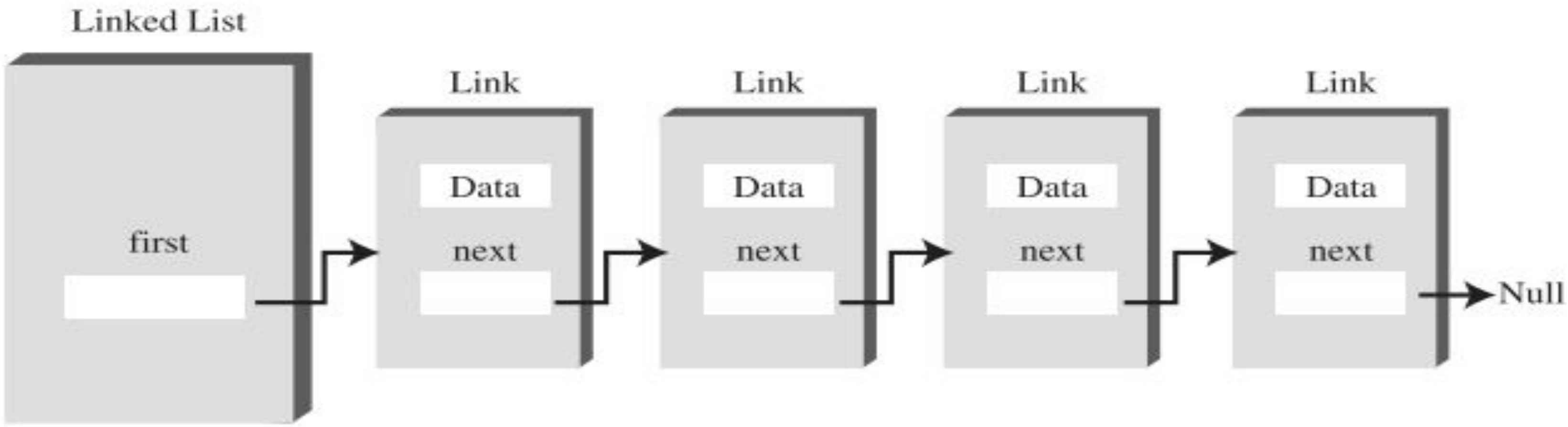
- A linked list provides an *alternative* to an array-based structure
- Arrays are great for storing things in a certain order, but they have drawbacks
- The *capacity of the array must be fixed* when it is created
- In array, insertions and deletions at interior positions of an array can be time consuming if many elements must be shifted
- In an unordered array, *searching is slow*, whereas in an ordered array, insertion is slow. In both kinds of arrays, *deletion is slow*
- Linked lists solves some of these problems

Singly Linked Lists

- A linked list, in its simplest form, is a collection of nodes that collectively form a *linear sequence*
- In a singly linked list, each node stores a **reference to an object** that is *an element of the sequence*, as well as a **reference to the next node** of the list
- In a linked list, each *data item is embedded in a link*
- A field (data) in the list itself contains a reference to the previous link

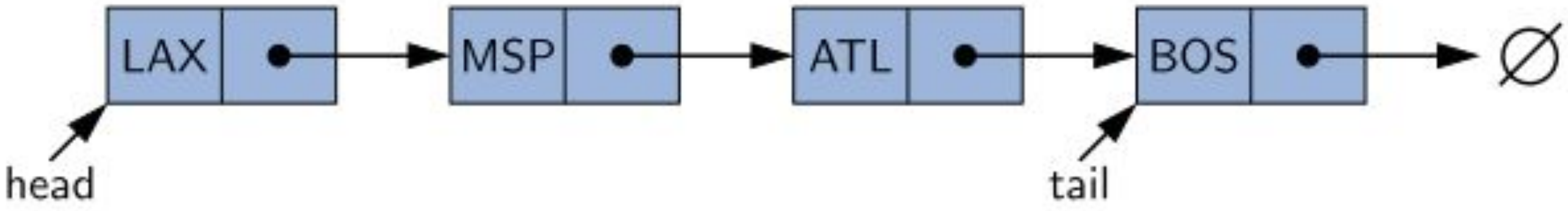


```
class Link
   {
   public int iData;       // data
   public double dData;    // data
   public Link next;       // reference to next link
   }
```



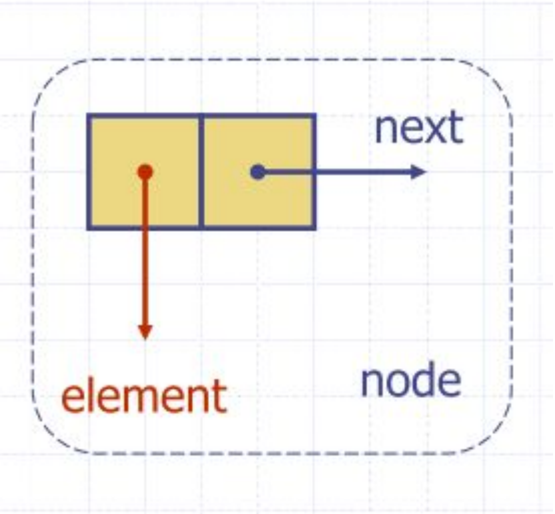
Singly Linked Lists

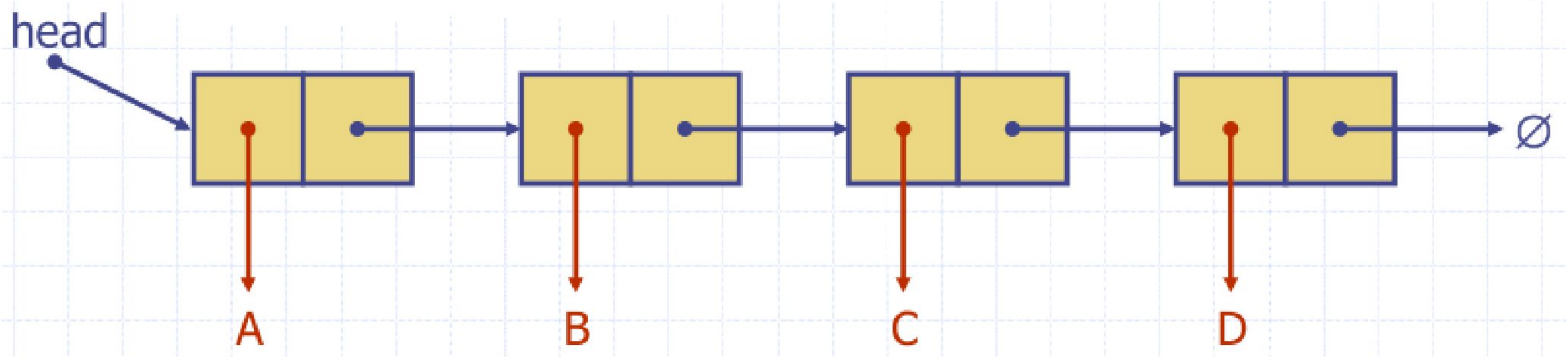
- A linked list’s representation relies on the collaboration of many objects
- Minimally, the linked list instance must keep a reference to the first node of the list, known as the **head**
- Without an explicit reference to the head, there would be no way to locate that node
- The last node of the list is known as the **tail**
- The tail of a list can be found by *traversing the linked list*— **starting at the head** and *moving from one node to another* by following each **node’s next reference**
- We can identify the tail as the node having ***null*** as its next reference



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





Implementing a Singly Linked List Class

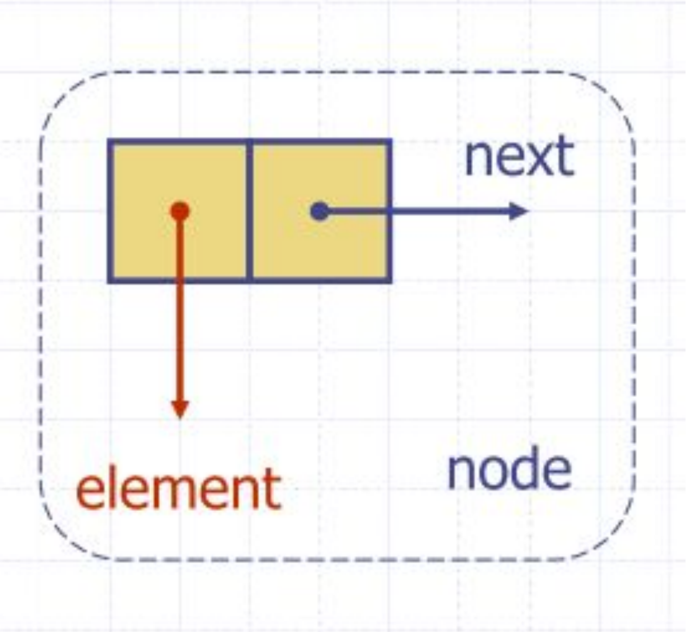
- A complete implementation of a Singly Linked List class, supporting the following methods:
  - **size()**: Returns the number of elements in the list.
  - **isEmpty()**: Returns true if the list is empty, and false otherwise.
  - **first()**: Returns (but does not remove) the first element in the list.
  - **last()**: Returns (but does not remove) the last element in the list.
  - **addFirst(e)**: Adds a new element to the front of the list.
  - **addLast(e)**: Adds a new element to the end of the list.
  - **removeFirst()**: Removes and returns the first element of the list.

Implementing a Singly Linked List Class

- A linked list is a ***collection of nodes***
- Each node stores a ***reference to an object*** that is an element of the sequence, as well as a ***reference to the next node of the list***
- Java's support for ***nested classes*** - a class inside the class
- we define a private Node class within the scope of the public SinglyLinkedList class
- Having Node as a nested class provides *strong encapsulation*, shielding users of our class from the underlying details about nodes and links

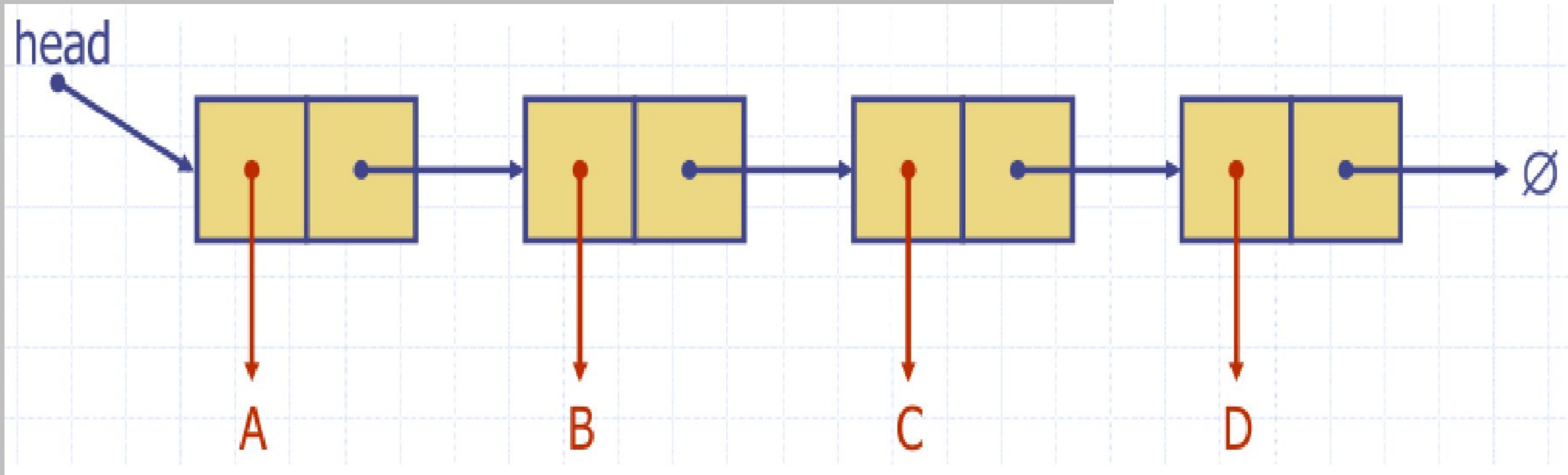
Singly Linked List - node class

```
private static class Node<E> {
    private E element;
    private Node<E> next;
    public Node(E e, Node<E> n) {
        element = e;     // reference to the element stored at this node
        next = n;        // reference to the subsequent node in the list
    }

    public E getElement() {  return element; }


    public Node<E> getNext() {  return next; }


    public void setNext(Node<E> n) { next = n; }
}
```



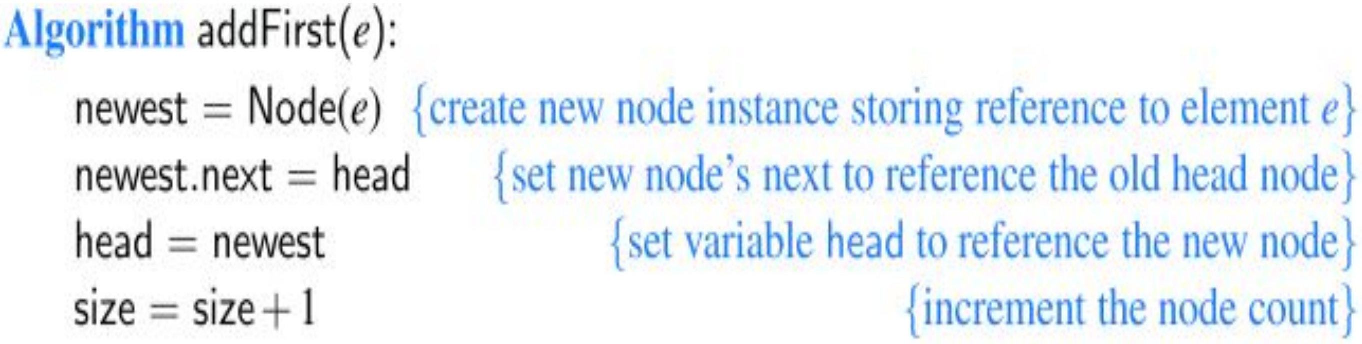
Singly Linked List Class

```
public class SinglyLinkedList<E> { /* node class definition*/
    private Node<E> head = null;  //head node of the list (or null if empty)
    private Node<E> tail = null;   //last node of the list (or null if empty)
    private int size = 0; //number of nodes in the list

    public SinglyLinkedList() { }    // constructs an initially empty list

    public int size() { return size; }
    public boolean isEmpty() { return size == 0; }

    public E first() {
        if (isEmpty()) return null;
        return head.getElement();
    }
    public E last() {
        if (isEmpty()) return null;
        return tail.getElement();
    }
}
```

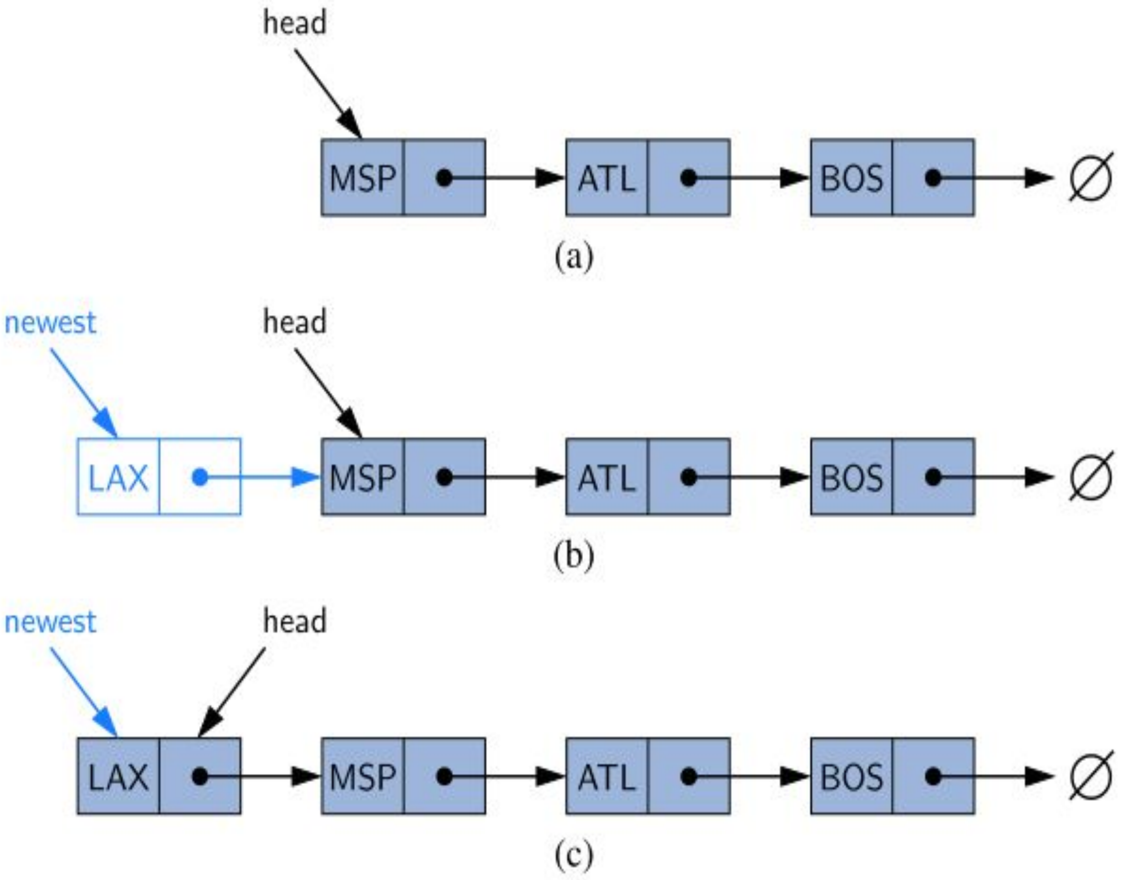


Inserting an Element at the Head of a Singly Linked List

1. Create a new node,
2. Set its element to the new element,
3. Set its next link to refer to the current head, and
4. Set the list's head to point to the new node.

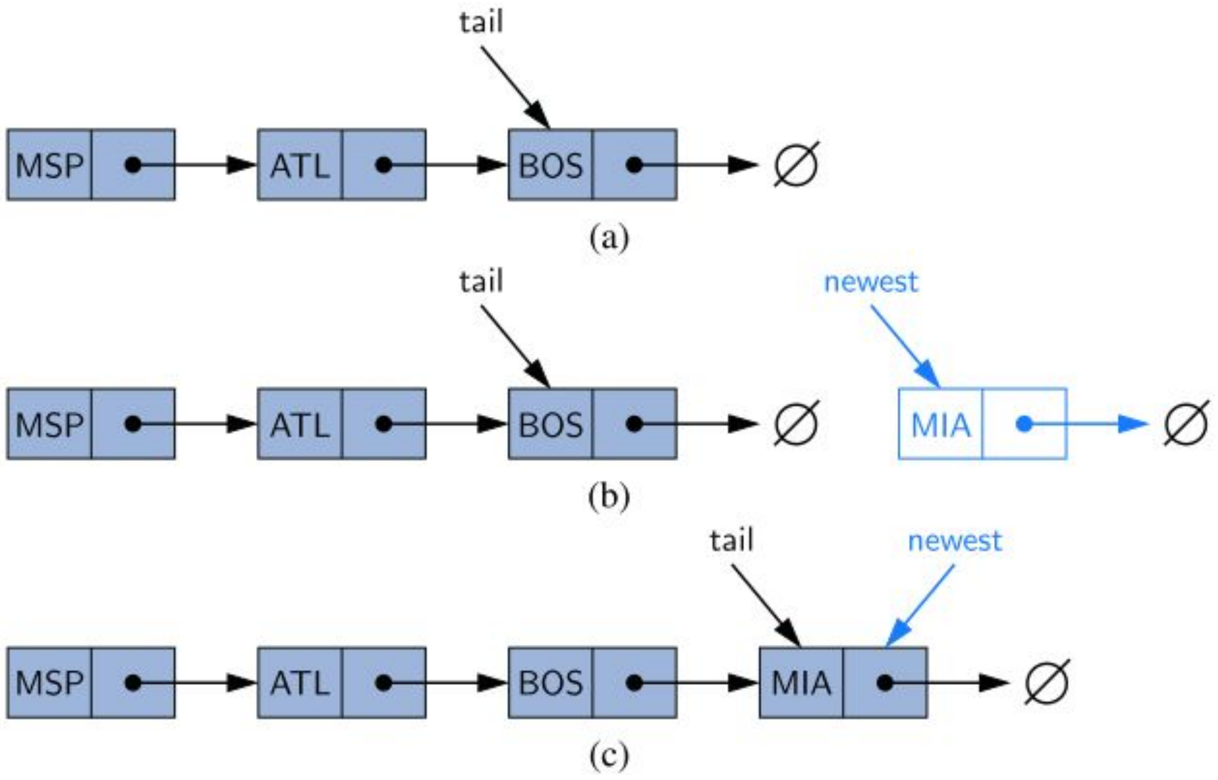


```
Algorithm addFirst($e$):
    newest = Node($e$)    {create new node instance storing reference to element $e$}
    newest.next = head    {set new node's next to reference the old head node}
    head = newest         {set variable head to reference the new node}
    size = size + 1       {increment the node count}
```



Inserting an Element at the Tail of a Singly Linked List

1. Create a new node
2. Assign its next reference to null
3. Set the next reference of the tail to point to this new node, and
4. Update the tail reference itself to this new node



```
Algorithm addLast(e):
    newest = Node(e)   {create new node instance storing reference to element e}
    newest.next = null           {set new node's next to reference the null object}
    tail.next = newest                       {make old tail node point to new node}
    tail = newest                       {set variable tail to reference the new node}
    size = size + 1                                      {increment the node count}
```

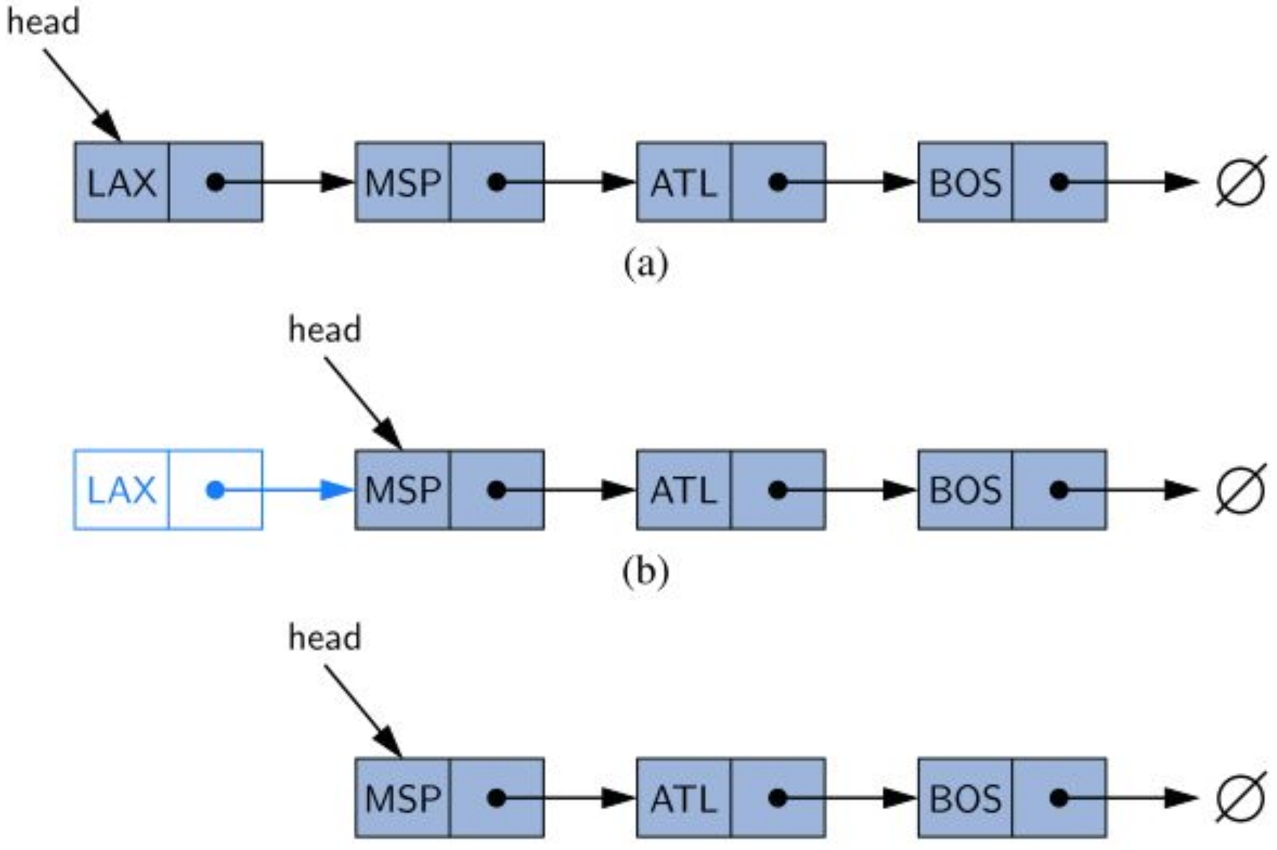
Java code to insert an element in LinkedList

```
public void addFirst(E e) {
 // adds element e to the front of the list
head = new Node<>(e, head);
// create and link a new node
if (size == 0)
        tail = head;
        // special case: new node becomes tail also
size++;
}
```

```
public void addLast(E e) {
// adds element e to the end of the list
Node<E> newest = new Node<>(e, null);
// node will eventually be the tail
if (isEmpty())
        head = newest;
        // special case: previously empty list
else
        tail.setNext(newest);
        // new node after existing tail
tail = newest;  // new node becomes the tail
size++;
}
```

Removing an Element from head of a Singly Linked List

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



(a)

(b)

(c)

```
Algorithm removeFirst( ):
    if head == null then
        the list is empty.
    head = head.next                    {make head point to next node (or null)}
    size = size − 1                     {decrement the node count}
```

Java code to delete an element in LinkedList

```
public E removeFirst() {   // removes and returns the first element
    if (isEmpty())
        return null;          // nothing to remove
    E answer = head.getElement();
    head = head.getNext();   // will become null if list had only one node
    size−−;
    if (size == 0)
        tail = null;       // special case as list is now empty
    return answer;
}
```

Class Assignment

- Study of Java inbuilt library classes of Stack and Queue.
- Make comparative analysis of Java library classes and Stack and Queue we learned in this course.
-

Lab Assignment

- Array based implementation of Java Stack and Queue API using generic classes.
- Implementation of applications discussed in the class using Stack and Queue classes implemented above [optional, not compulsory]