A singly linked list is a concrete data structure consisting of a sequence of nodes, starting from a head pointer. Each node stores:

- an element
- a link to the next node

Head: A → B → C → D → ∅
A Nested Node Class

```java
import java.util.List;

public class SinglyLinkedList<E> {
    // ----------------------- nested Node class -----------------------
    private static class Node<E> {
        private E element; // reference to the element stored at this node
        private Node<E> next; // reference to the subsequent node in the list
        public Node(E e, Node<E> n) {
            element = e;
            next = n;
        }
        public E getElement() { return element; }
        public Node<E> getNext() { return next; }
    }
    // rest of SinglyLinkedList class will follow...
}
```

Accessor Methods

```java
public class SinglyLinkedList<E> {
    ... // instance variables of the SinglyLinkedList
    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
    // access methods
    public int size() { return size; }
    public boolean isEmpty() { return size == 0; }
    public E first() { // returns (but does not remove) the first element
        if (isEmpty()) return null;
        return head.getElement();
    }
    public E last() { // returns (but does not remove) the last element
        if (isEmpty()) return null;
        return tail.getElement();
    }
}
```
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

```java
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<> 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

```java
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--; // special case as list is now empty
51       if (size == 0)
52           tail = null;
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