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
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

```java
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; }
        public void setNext(Node<E> n) { next = n; }
    } //----------------------- end of nested Node class -----------------------
    ... rest of SinglyLinkedList class will follow ...
}
```
public class SinglyLinkedList<E> {
...  (nested Node class goes here)

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

![Diagram of linked list operations]

1. **Allocate new node**
2. **Insert new element**
3. **Have new node point to old head**
4. **Update head to point to new node**
Inserting at the Head

- Allocate new node
- Insert new element
Inserting at the Head

- Allocate new node
- Insert new element
- Have new node point to old head
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
Inserting at the Tail

- Allocate a new node
- Insert new element
Inserting at the Tail

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

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

(a)

(b)

(c)
```java
public E removeFirst() {
    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;
    return answer; // special case as list is now empty
}
```
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.