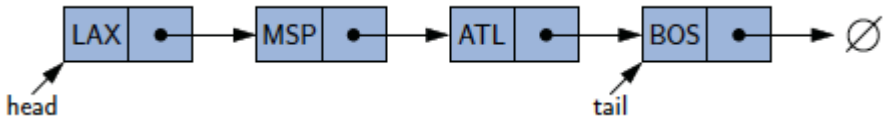
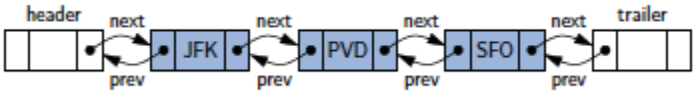
Linked List

Linked List:

- A collection of nodes
  - **Node**: An object that holds two important instance variables:
    - Data
    - Link or pointer to the next node
- The first and last node of a linked list usually called the head and tail respectively
- Transverse through the list by linkhopping from the beginning to end
- Each node has a pointer to the next node
- Two types of Linked List:
  - **Singly Linked List**



  - **Doubly Linked List**



LinkedList (JFC https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html):

- Doubly linked list implementation of the List Interface

Similarities with Arrays:

- Keeps its elements in a certain order
- Arrays used index to order its elements. Linked List uses the chain of links to order its elements

Advantages over Arrays:

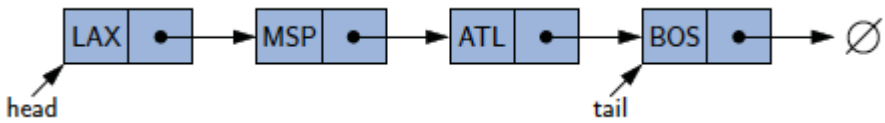
- No predetermined size
- Uses space proportional to the number of elements is has

Disadvantages:

- No indexes numbers for each element
- Cannot directly access 2nd, 5th, or 12th element as easily as Arrays

Singly Linked List:

- Node only has one pointer to the next element
- First node is usually called *Head*, the last node is usually called the *Tail*



What does each node store?

- Piece of data
- Pointer to the next node

What are the major components of a Linked List?

- Head, Tail
- Size

General Rules:

- Coding any method must end leaving all instance variable with a legal value

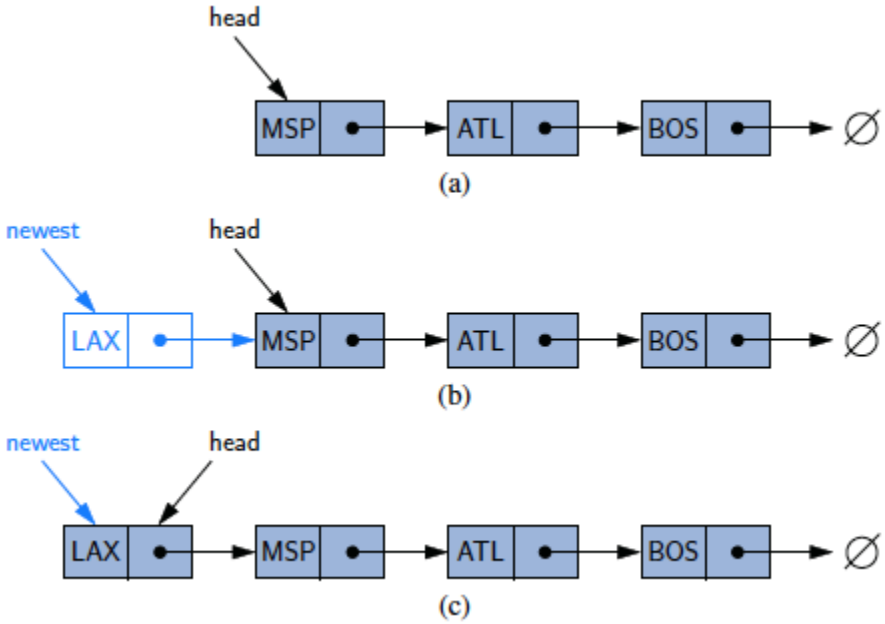
Advice:

- At the end of any method, ask for each variable whether it's correct and legal
- Plan what counts as legal values for instance variables in advance for data structures. Think about empty and near empty cases.

Since linked list is a data structure how do we store and retrieve data?

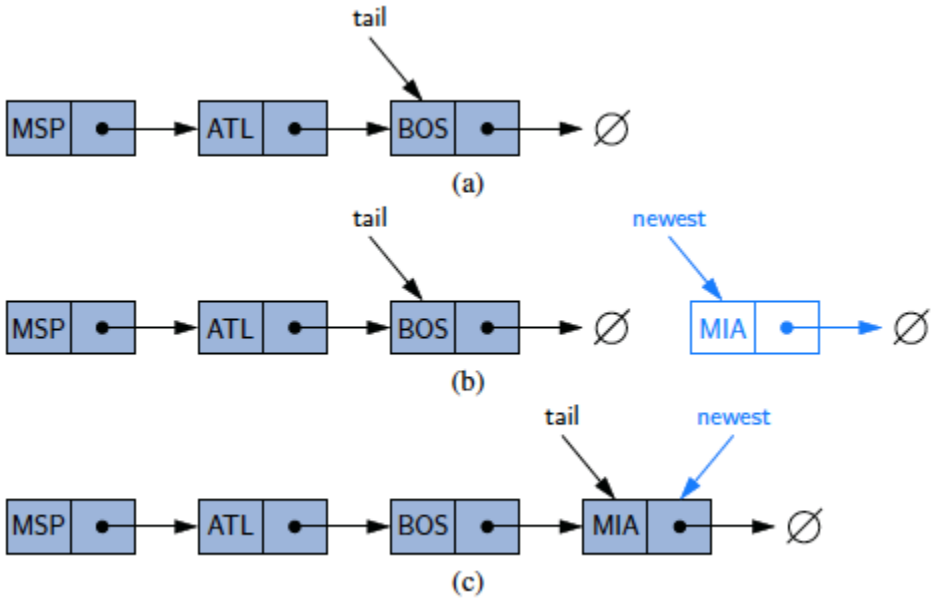
How can we store data?

1. We can add to the head of the list
    - addFirst(T x)
    - Add x as a new piece of data at head



        i. Create a new node and link it to the head
        ii. Increment the size
        iii. Move the head pointer
    - What are some special situations when added new elements to head? Think about when the list is empty.

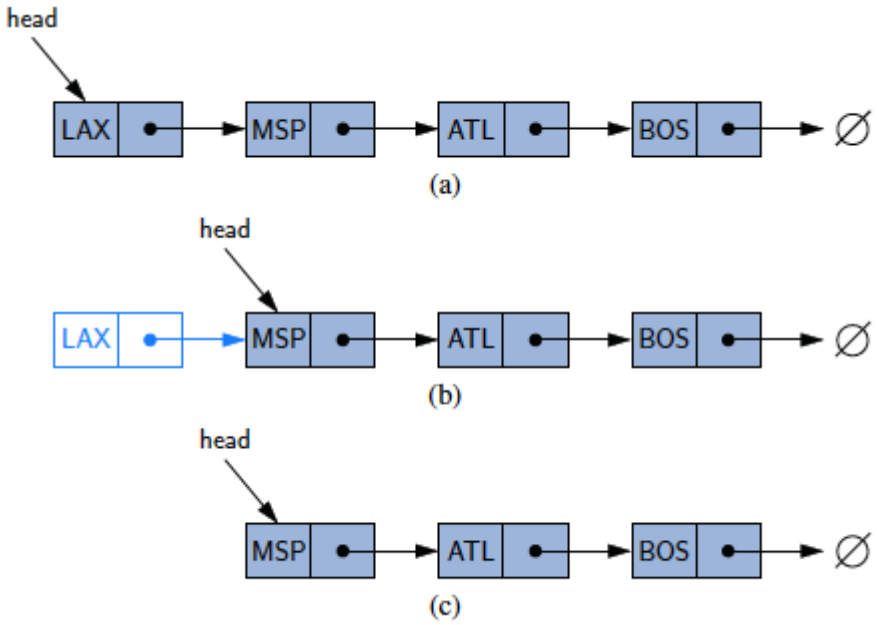
2. We can add to the rear of the list
    - addLast(T x)
    - Add x as a new piece of data at tail



        i. Create a new node
        ii. Set tail node to point to the new node
        iii. Increment the size
        iv. Move the tail pointer
    - What are some special situations when added new elements to tail? Think about when the list is empty.
3. We can add anywhere in between
    - **Problems**: If we want to add to a specific location:
        i. Add it as item #... that inefficient
        ii. Add it next to another node:
            1. Add before, that inefficient
            2. Add After, that easy

How do we retrieve data?

1. Remove from head
    - removeFirst()
    - Two special cases to consider: one node in list and an empty list

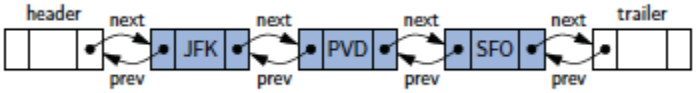


- Check is the head is null -> empty list, throw an exception
- Store the head current data, t
- Move the head pointer
- Check if the head is now null, empty list, reset the tail pointer as well
- Decrement the size
- Return the old heads data

2. Remove from tail
    - Not Efficient
3. Remove from anywhere in between
    - Not Efficient

Doubly Linked List:

- Node contains both a next pointer and a previous pointer

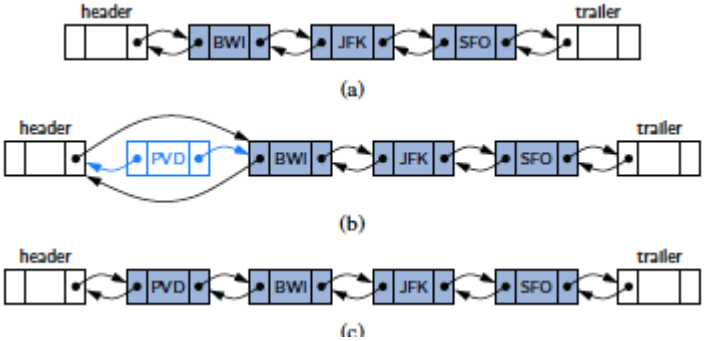


- A better Linked List, solves some issues of a singly linked list
- Having a dummy or sentinel node that do not store any element will solve issues of a empty list
- Header Node:
    - Node before the head
    - Valid next reference to the head
    - Null previous reference
- Trailer Node:
    - Node after the tail
    - Valid previous reference to the tail
    - Null next reference

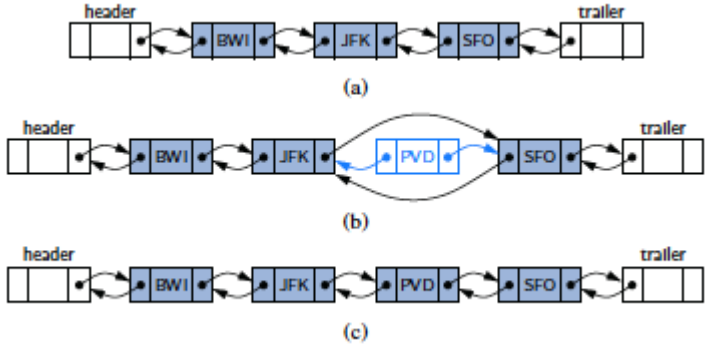
- The doubly linked list only needs:
    - Two sentinel nodes
    - Size

How do we store data?

1. Add to the head
    a. Inserting in the head, we are really just inserting after the header sentinel



2. Add to the tail
    a. Inserting in the tail, we are really just inserting before the trailer sentinel
3. Add to the middle
    a. If we want to insert anywhere on the list, really we are inserting before or after something



    b. If we insert PVD between JFK and SFO, we are either inserting after JFK or before SFO

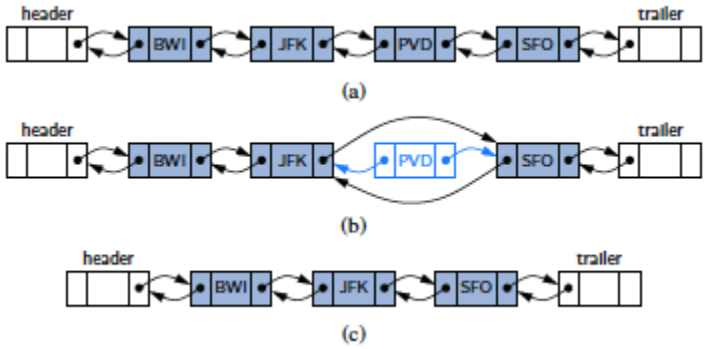
Overall inserting into a doubly linked list, we need two main methods:

- InsertBefore(T x)
- insertAfter(T x)

How do we retrieve data?

1. From the head
2. From the tail
3. From the middle

Really removing anywhere from the node, we just need the node we want to remove:



- Get the target nodes previous and next references, label as P and N respectively
- Link N with P and P with N
- Null out the target nodes two pointers
- Decrement the size

Be careful if the user ever tries to remove either of the sentinels. This will throw a null pointer exception.
If the nodes the user tries to remove are the sentinels, prevent them.