Week 11: Topics

- Data Structures

Generic Data Structures

- Dynamic data structures grow and shrink at execution time

- Linked lists are collections of data items "linked up in a chain"

  Insertions and deletions can be made anywhere in a linked list

Generic Data Structures (cont.)

- Stacks are important in compilers and operating systems
  - Insertions and deletions can only be made at one end of the stack

    The top

- Queues represent waiting lines

  Insertions are made at the back i.e., tail

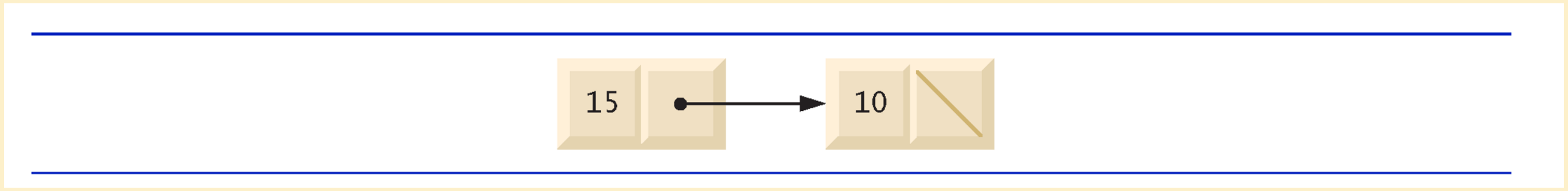
  Deletions are made from the front i.e., head

Self-Referential Classes

- A self-referential class contains an instance variable that refers to another object of the same class type

- The figure below illustrates two self-referential objects lined together to form a list

  15 and 10 are the data values



- A backslash representing a null reference is placed in the second self-referential object

  Indicates that the link does not refer to another object

Generic Node Class Declaration

```
Blackboard: Week11/Node.java
```

Linked Lists

- A linked list is a sequence of self-referential-class objects, called **nodes**, connected by reference links

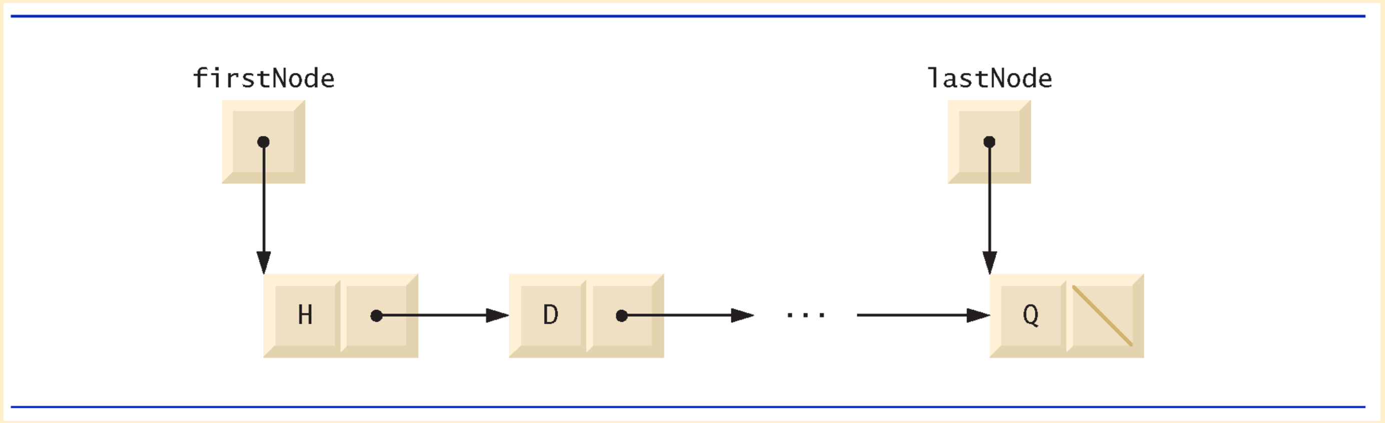
  Typically, a program accesses a linked list via a reference to its first node

- A linked list is appropriate when the number of elements to be represented in the data structure is unpredictable
  - Linked lists become full only when the system has insufficient memory to satisfy dynamic storage allocation requests

Singly Linked Lists

- Linked list nodes normally are not stored contiguously in memory

    Rather, they are logically contiguous



- Often, linked lists are implemented as doubly linked lists

    Each node contains a reference to the next node in the list and a reference to the preceding one.

Quick Tutorial:

Implement the generic list class given below

```
class ListNode<E>
{
    E data;
    ListNode<E> nextNode; // reference to next linked node

    public ListNode(E object) { /* constructor body */ }
    public ListNode(E object, ListNode<E> node) {/*constructor body*/}
    public E getData() { /* method body */ }
    public ListNode<E> getNext() { /* method body */ }
}
```

Implementing a Generic `List` Class

`Blackboard: Week11/List/List.java`

Class `List` Definition

`Blackboard: Week11/List/List.java`

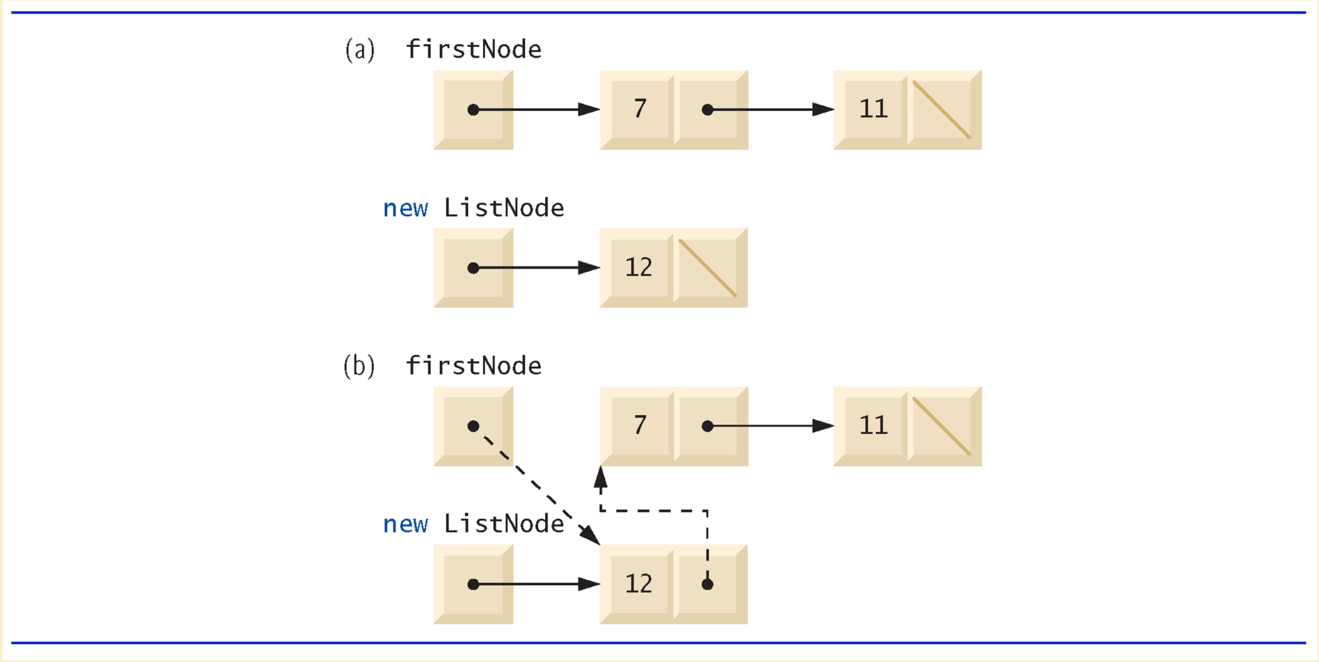
List Method `insertAtFront()`

```
Blackboard: Week11/List/List.java
```

List Method `insertAtFront()`

- If the list is not empty, the new node is “linked” into the list by setting `firstNode` to a new `ListNode` object

And initializing that object with insertItem and firstNode



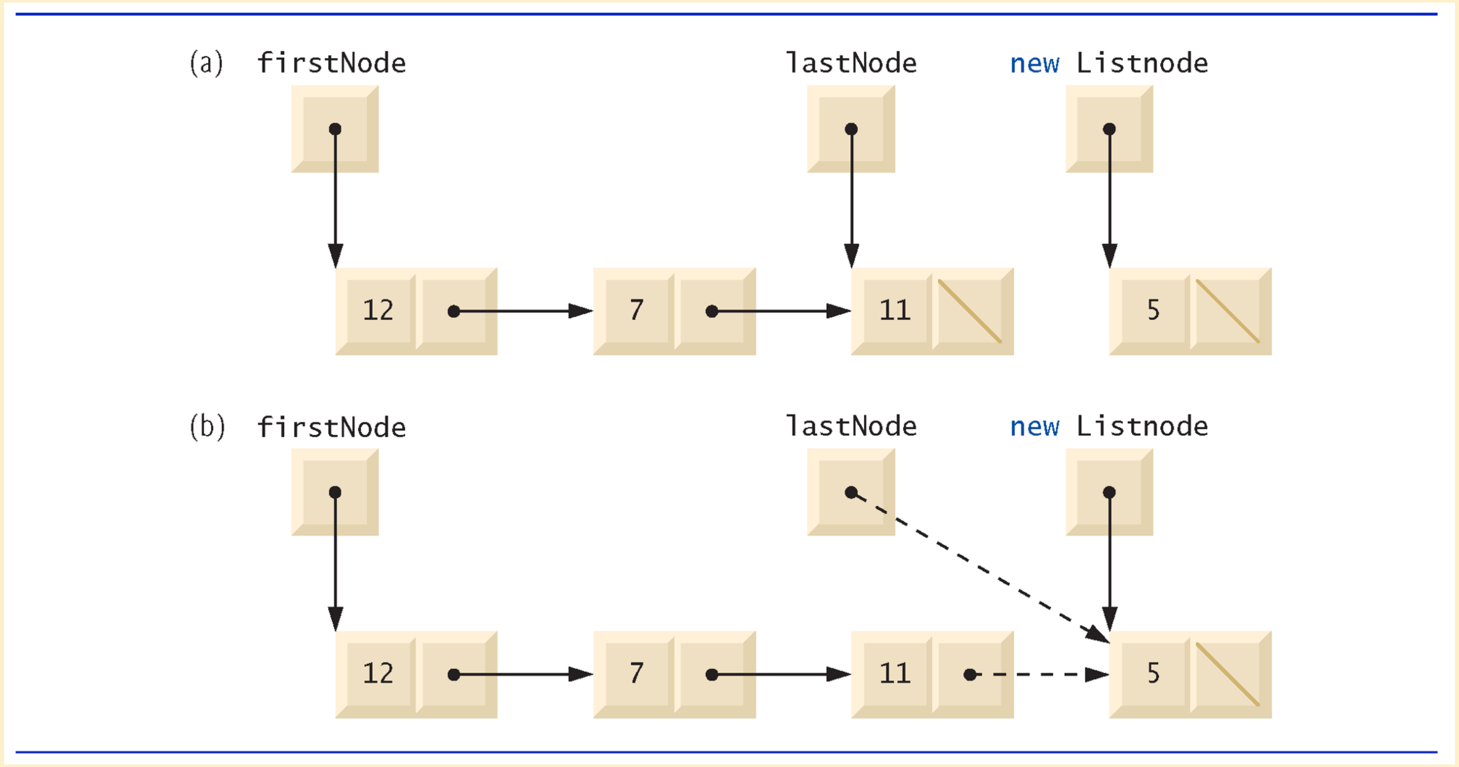
Quick Tutorial:

Implement the `List` method `insertAtBack()`

`Blackboard: Week11/List/List.java`

List Method `insertAtBack()`

- If list is not empty, link the new node into the list By assigning to `lastNode` and `lastNode.nextNode` the reference to the new `ListNode` that was initialized with `insertItem`

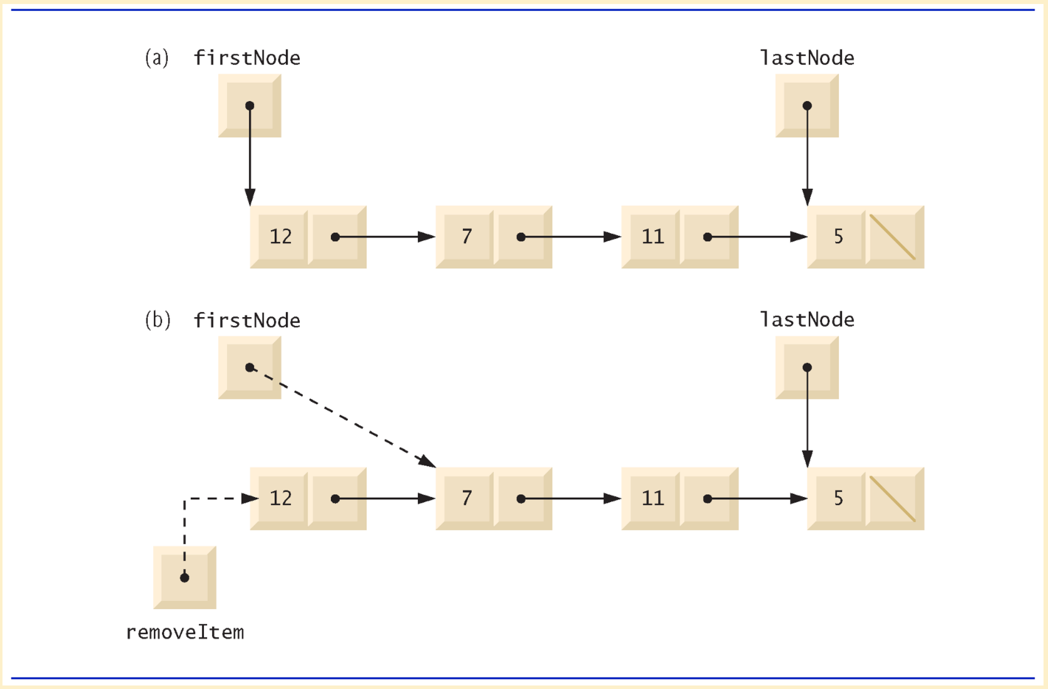


List Method removeFromFront()

Blackboard: Week11/List/List.java

List Method removeFromFront()

- If `firstNode` and `lastNode` refer to the same object
  - List has only one element
- If the list has more than one node, we assign the value of firstNode.nextNode to firstNode



Quick Tutorial:

Implement the `List` method `removeFromBack()`

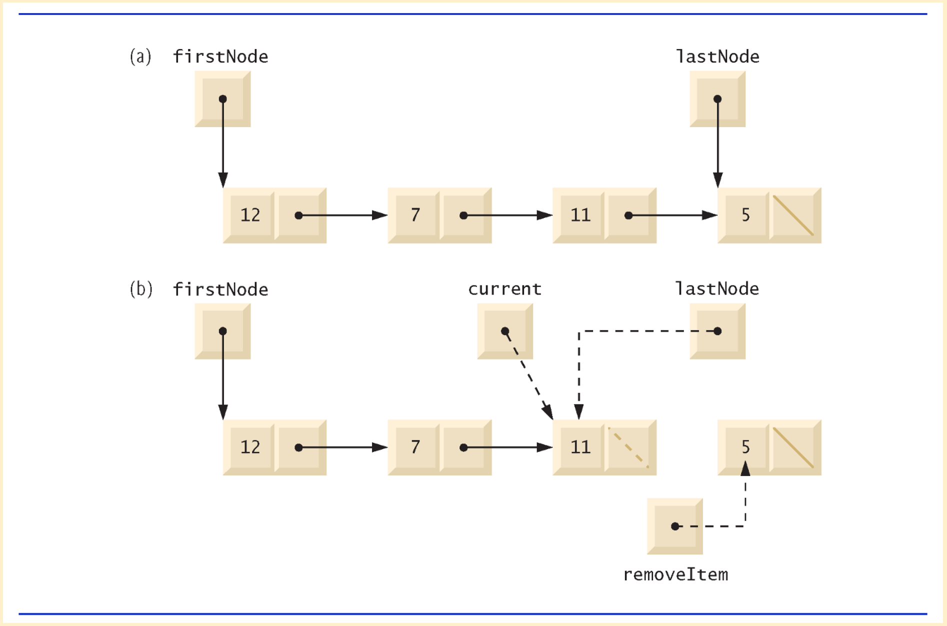
`Blackboard: Week11/List/List.java`

List Method `removeFromBack()`

- If list has more than one node, create the reference `current` and assign it `firstNode`

- Now "walk the list" with `current` until it references the node before the last node

Assign current to lastNode



Queues

- A queue is similar to a checkout line in a supermarket
  - The cashier services the person at the beginning of the line first

    Other customers enter the line only at the end and wait for service

- Queue nodes are removed only from the head (or front) of the queue and are inserted only at the tail (or end)

  For this reason, a queue is a first-in, first-out (FIFO) data structure

Quick Tutorial:

Create a `Queue` class that contains a `List()` object and provides methods `enqueue`, `dequeue`, `isEmpty` and `print`

Implementing a Queue using Class `List`

`Blackboard: Week11/Queue/Queue.java`

Quick Tutorial:

Test the functionality of the Queue class using the `QueueTest` class

Testing the Queue Class Functionality

`Blackboard: Week11/Queue/QueueTest.java`

Testing the Queue Class Functionality (cont.)

`Blackboard: Week11/Queue/QueueTest.java`