A Reference-Based Implementation of the ADT List

- Default constructor
  - Initializes the data fields `numItems` and `head`
- List operations
  - Public methods
    - isEmpty
    - size
    - add
    - remove
    - get
    - removeAll
  - Private method
    - find

Constructor and simple methods

```
private Node head;
private int numItems; // number of items in list

public ListReferenceBased() {
  numItems = 0;
  head = null;
}

public boolean isEmpty() {
  return numItems == 0; // how else can this be done?
}

public int size() {
  return numItems;
}
```


A find method will be useful

Locates a specified node in a linked list.
Precondition: index is the number of the desired node. Assumes that 1 <= index <= numItems+1
Postcondition: Returns a reference to the desired node.

```
private Node find(int index) {
  Node curr = head;
  for (int skip = 1; skip < index; skip++) {
    curr = curr.getNext();
  }
  return curr;
}
```


Adding an item

```
public void add(int index, Object item) throws
   ListIndexOutOfBoundsException {
  if (index >= 1 && index <= numItems + 1) {
    if (index == 1) {
      Node newNode = new Node(item, head);
      head = newNode;
    } else {
      Node prev = find(index - 1);
      Node newNode = new Node(item, prev.getNext());
      prev.setNext(newNode);
    }
    numItems++;
  } else {
    throw new ListIndexOutOfBoundsException("List index
   out of bounds on add");
  }
}
```

Removing an item

```
public void remove(int index) throws
   ListIndexOutOfBoundsException {
  if (index >= 1 && index <= numItems) {
    if (index == 1) {
      head = head.getNext();
    } else {
      Node prev = find(index - 1);
      Node curr = prev.getNext();
      prev.setNext(curr.getNext());
    }
    numItems--;
  } else {
    throw new ListIndexOutOfBoundsException("List index
   out of bounds on remove");
  }
}
```


Processing Linked Lists Recursively

- Traversal
  - Recursive strategy to display a list
    - Write the first node of the list
    - Write the list minus its first node
  - Recursive strategies to display a list backward
    - Write the list minus its first node backward
    - Write the first node of the list
- Insertion (see page 254)

Variations of the Linked List: Tail References

- `tail` references
  - Remembers where the end of the linked list is
  - To add a node to the end of a linked list

```
tail.setNext(new Node(request, null));
```

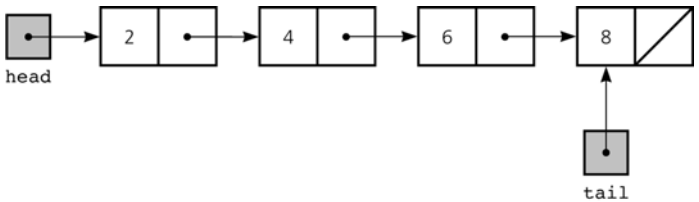

Figure 5-22

A linked list with ***head*** and ***tail*** references

Complete the addToEnd method (with a head and tail pointer)

```
public void addToEnd(Object item) {
  if (head == null) {
```

Circular Linked List

- Last node references the first node
- Every node has a successor

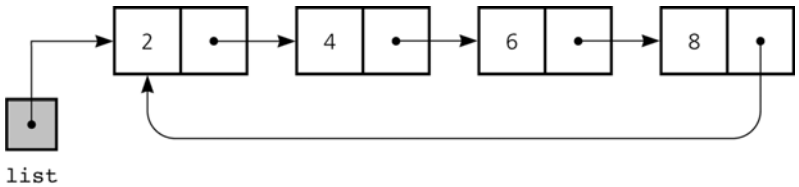

Figure 5-23

A circular linked list

Circular Linked List

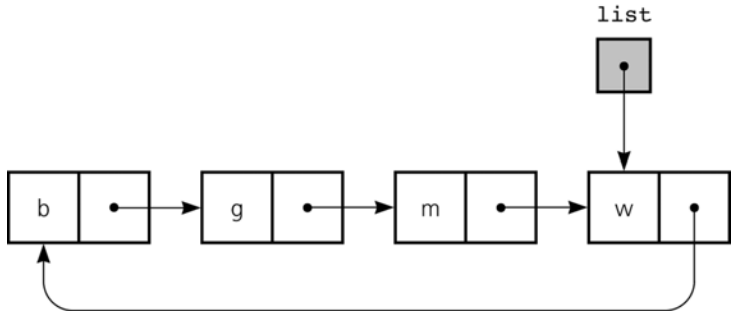

Figure 5-24

A circular linked list with an external reference to the last node

Dummy Head Nodes

- Dummy head node
  - Always present, even when the linked list is empty
  - Insertion and deletion algorithms initialize `prev` to reference the dummy head node, rather than `null`

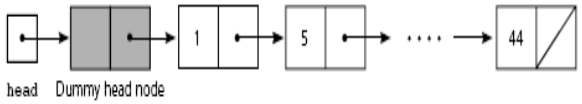

Figure 5-25

A dummy head node

Doubly Linked List

- Each node references both its predecessor and its successor
- Dummy head nodes are useful in doubly linked lists

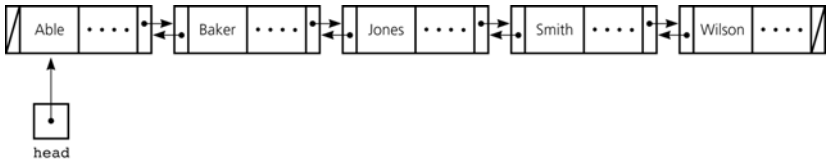

Figure 5-26

A doubly linked list

Doubly Linked List

- Circular doubly linked list
  - `precede` reference of the dummy head node references the last node
  - `next` reference of the last node references the dummy head node
  - Eliminates special cases for insertions and deletions

Doubly Linked List

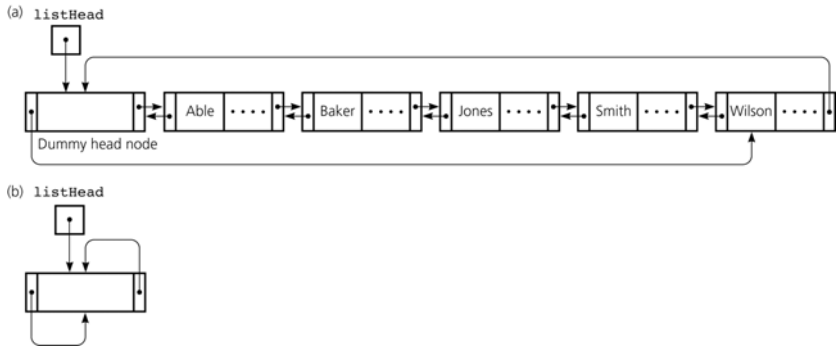

Figure 5-27

a) A circular doubly linked list with a dummy head node; b) an empty list with a dummy head node

Doubly Linked List

- To delete the node that `curr` references

```
curr.getPrecede().setNext(curr.getNext());
curr.getNext().setPrecede(curr.getPrecede());
```

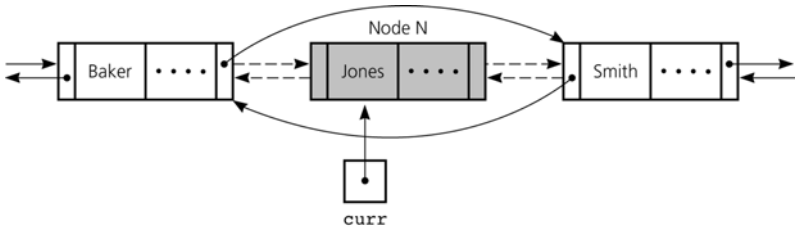

Figure 5-28

Reference changes for deletion

Doubly Linked List

- To insert a new node that `newNode` references before the node referenced by `curr`

```
newNode.setNext(curr);
newNode.setPrecede(curr.getPrecede());
curr.setPrecede(newNode);
newNode.getPrecede().setNext(newNode);
```

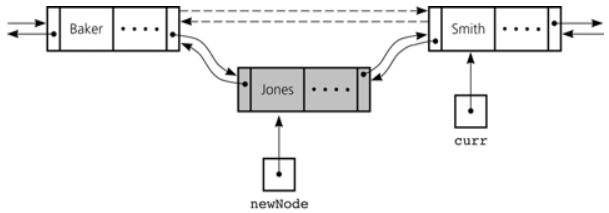

Figure 5-29

Reference changes for insertion