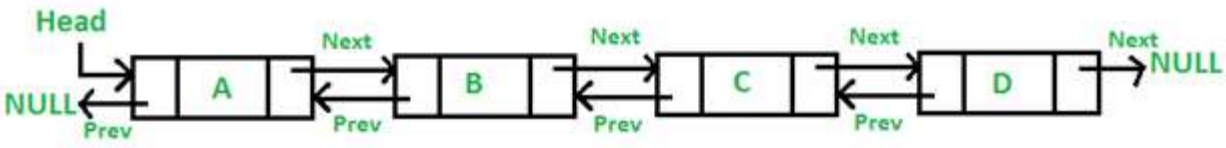
Doubly Linked List

Node:

Date
Next → null
null← Prev

Doubly Linked List: Each node references both its predecessor and its successor:



Doubly Node Code:

```
public class DNode <T extends Comparable<T>>{
   T data;
   DNode next;
   DNode prev;

   public DNode(T data) { this.data = data; }
   public T getData() { return data;  }
   public DNode getNext () { return next; }
   public DNode getPrev () { return prev; }

   public void setNext(DNode next) {   this.next = next; }
   public void setPrev(DNode prev) {   this.prev = prev; }
   public String toString() { return this.data.toString();  }
}
```

Doubly Linked List code:

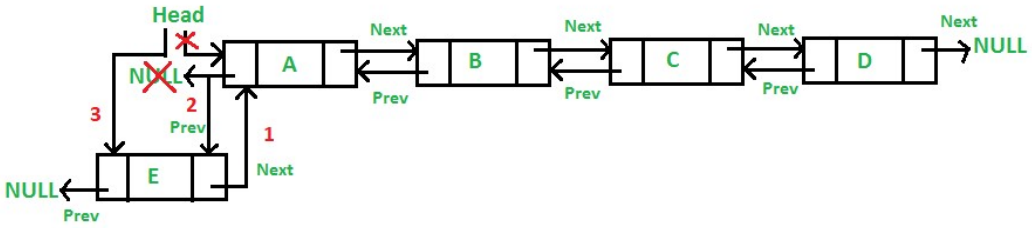
```
public class DLinkedList <T extends Comparable<T>>{
   DNode head;
}
```

Override toString method code:

```
public String toString() {
   String res = "Head-->";
   DNode<T> curr = this.head;
   while (curr != null) {
      res += "["+curr + "]";
      curr = curr.getNext();
      if(curr!=null)
         res +="<=>";
   }
   return res + "-->NULL";
}
```

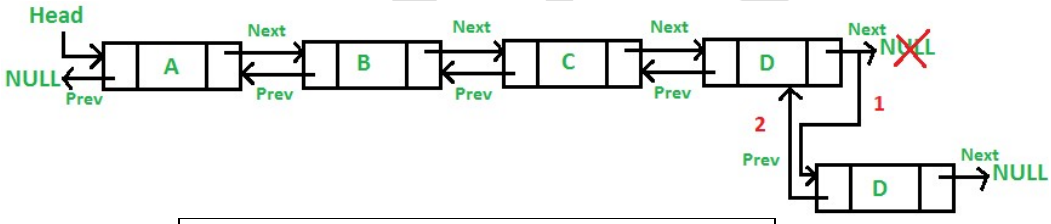
Insert a new node (not sorted)

Case 1: Insert at head:




```
public void insertAtHead(T data) {
   DNode<T> newNode = new DNode(data);
   if(head==null) // empty linkedlist
      head = newNode;
   else {
      newNode.setNext(head);
      head.setPrev(newNode);
      head = newNode;
   }
}
```

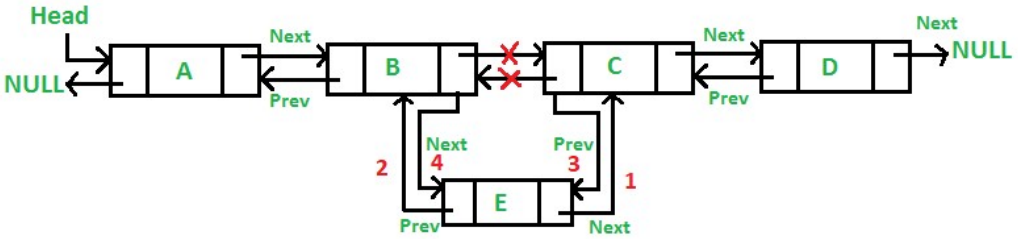
Case 2: Insert at end:




```
public void insertAtEnd(T data) {
   DNode<T> newNode = new DNode(data);
   if (head == null) // empty linkedlist
      head = newNode;
   else  {   // find last node
      DNode<T> last = head;
      while(last.getNext() != null)
         last = last.getNext();
      last.setNext(newNode);
      newNode.setPrev(last);
   }
}
```

Insert a new Node (Sorted):

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



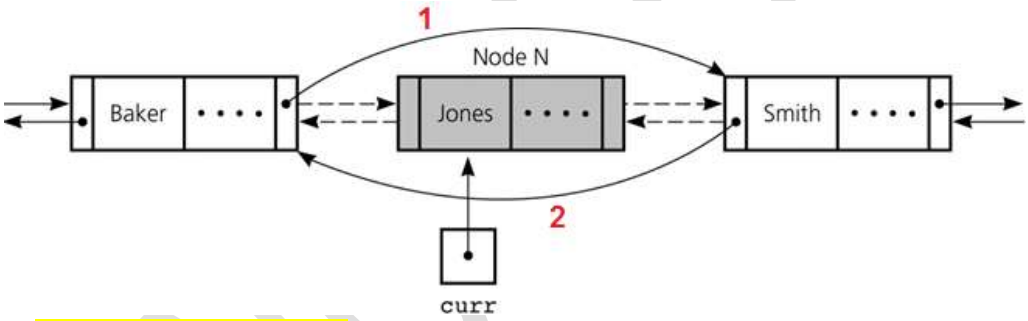
```
newNode.next = curr;                // 1
newNode.prev = curr.prev;           // 2
curr.prev = newNode;                // 3
newNode.prev.next = newNode;        // 4
```

Length of a doubly linked list code:

```
public int length() {
   int length = 0;
   DNode<T> curr = head;
   while (curr != null) {
      length++;
      curr = curr.getNext();
   }
   return length;
}
```

Delete a node:

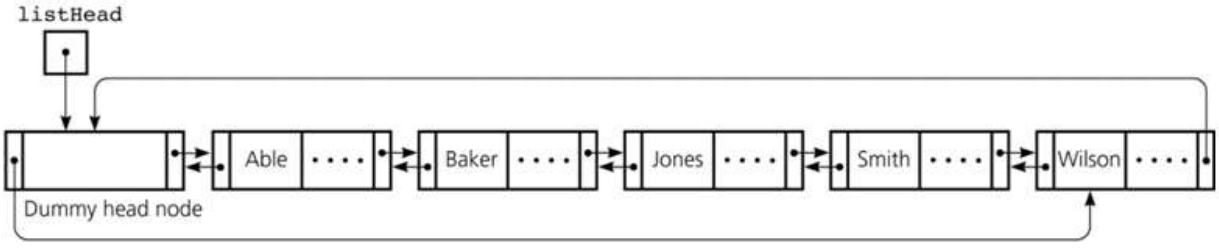
- To delete the node that **curr** pointer references




```
curr.prev.next = curr.next;
curr.next.prev = curr.prev;
```

Student Activity: Consider all the delete cases

Circular doubly linked list with dummy head:



- Preceding 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.**