Doubly Linked Lists

Like a singly linked list but with pointers in both directions.
They look like this

5  8  2  7
Previous and next pointers

class Node
{
    int info;
    Node prev, next;

    // constructor
    public Node(int info, Node prev, Node next) {
        this.info = info;
        this.prev = prev;
        this.next = next;
    }
}

prev is new
Doubly linked list declaration

```java
public class DLL
{
    public class Node
    {
        // <node stuff here>
    }

    Node head, tail;
    int size;

    public DLL() // constructor
    {
        head = tail = null;
        size = 0;
    }
}
```

Since there are two directions, track head and tail.

OK, why not?
Let’s add to front

Similar to a singly linked list we can try:

Node head = new Node(4, null, null);

Not done yet.
Node head = new Node(4, null, null);

Better also say:

tail = head;
size++;

Do it again

Again, like the singly linked list:

Node head = new Node(5, null, head);

Not done yet: 4 needs to point back to 5.
To finish

The node after the head node must point back to it.

```java
Node head = new Node(5, null, head);
head.next.prev = head;
```
Concluding addToFront()

```java
public void addToFront(int newVal) {
    head = new Node(newVal, null, head);

    if (tail == null)
        tail = head;
    else
        head.next.prev = head;

    size++;
}
```
Add to arbitrary location

Add a value, say 7, after a specified location…

…like here: position
Step 1

Node after = position.next;
Step 2

Node after = position.next;
position.next = new Node(7, position, after);
Step 3

Node after = position.next;
position.next = new Node(7, position, after);
after.prev = position.next;
public void addTo(int newVal, Node position) {
    if (position == null)
        return;
    Node after = position.next;
    position.next = new Node(newVal, position, after);
    if (after == null)
        tail = position.next;
    else
        after.prev = position.next;
    size++;
}
public void addToEnd(int newVal) {
    addTo(newVal, tail);
}

Delete a node

Given a node position, delete that node. This is different than being given a value to delete, which may not be there.
Delete a node

position.next.prev = position.prev;

Watch out:
if (position.next == null)
    tail = position.prev;
else ...
<do this>
Delete a node

```java
position.next.prev = position.prev;
position.prev.next = position.next;
```

Watch out:
if (position.prev == null)
head = position.next;
else ...
<do this>
Concluding deleteNode()

```java
public void deleteNode(Node position) {
    if (position == null)
        return;
    if (position.next == null)
        tail = position.prev;
    else
        position.next.prev = position.prev;
    if (position.prev == null)
        head = position.next;
    else
        position.prev.next = position.next;
    size--;
}
```