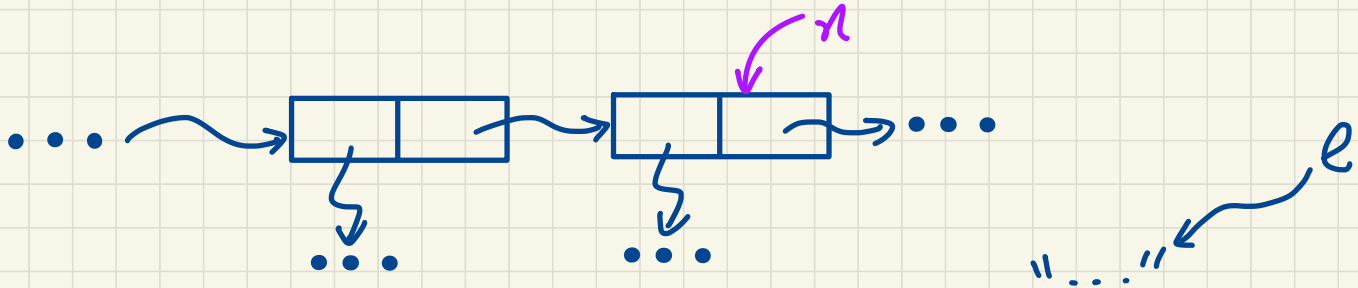
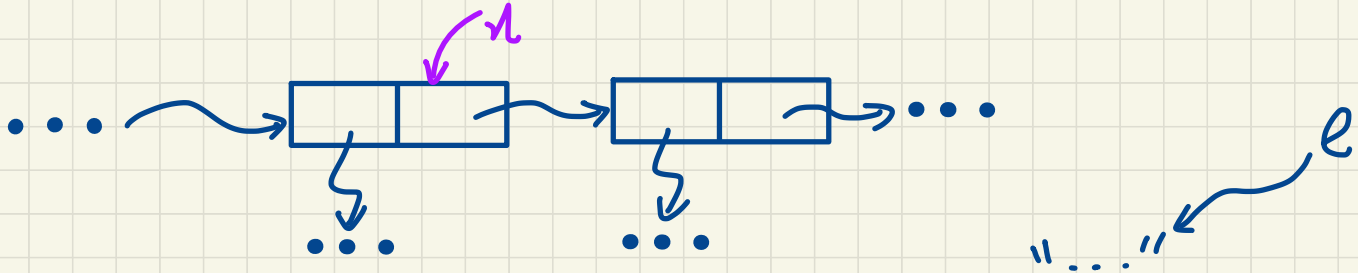
Exercises: insertAfter vs. insertBefore

Case: insertAfter(Node n, String e)

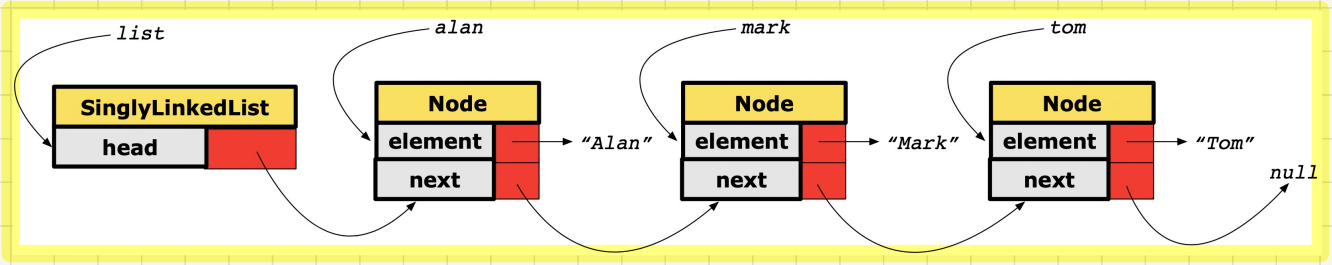


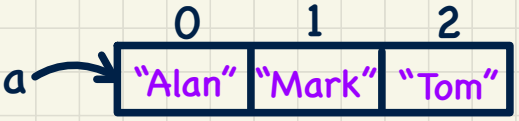
Case: insertBefore(Node n, String e)



Running Time: Arrays vs. Singly-Linked Lists

Operation \ Data Structure		Array	Singly-Linked List
get size			
get first/last element			
get element at index i			
remove last element			
add/remove first element, add last element			
add/remove i^{th} element	given reference to $(i-1)^{th}$ element		
	not given		





Doubly-Linked Lists (DLL): Visual Introduction

- A chain of bi-directionally connected nodes
- Each node contains:
  - + reference to a data object
  - + reference to the next node
  - + reference to the previous node
- A DLL is also a SLL:
  - + many methods implemented the same way
  - + some method implemented more efficiently
- Each DLL stores dedicated Header & Trailer Nodes (no head refeference and no tail reference)
- The chain may grow or shrink dynamically.
- Accessing a node in a DLL (via next or prev):
  - + Relative positioning: O(n)

Empty Lists: SLLs vs. DLLs

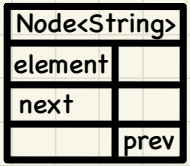
DLLs: Relative Positioning

Generic DLL in Java: DoublyLinkedList vs. Node

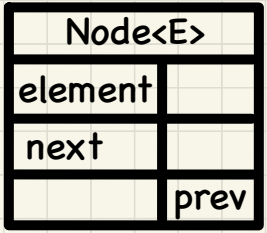
```java
public class DoublyLinkedList<E> {
   private int size = 0;
   public void addFirst(E e) { ... }
   public void removeLast() { ... }
   public void addAt(int i, E e) { ... }
   private Node<E> header;
   private Node<E> trailer;
   public DoublyLinkedList() {
      header = new Node<>(null, null, null);
      trailer = new Node<>(null, header, null);
      header.setNext(trailer);
   }
}
```

```java
@Test
public void test_String_DLL_Empty_List() {
    DoublyLinkedList<String> list = new DoublyLinkedList<>();
    assertTrue(list.getSize() == 0);
    assertTrue(list.getFirst() == null);
    assertTrue(list.getLast() == null);
}
```

```java
public class Node<E> {
   private E element;
   private Node<E> next;
   public E getElement() { return element; }
   public void setElement(E e) { element = e; }
   public Node<E> getNext() { return next; }
   public void setNext(Node<E> n) { next = n; }
   private Node<E> prev;
   public Node<E> getPrev() { return prev; }
   public void setPrev(Node<E> p) { prev = p; }
   public Node(E e, Node<E> p, Node<E> n) {
      element = e;
      prev = p;
      next = n;
   }
}
```

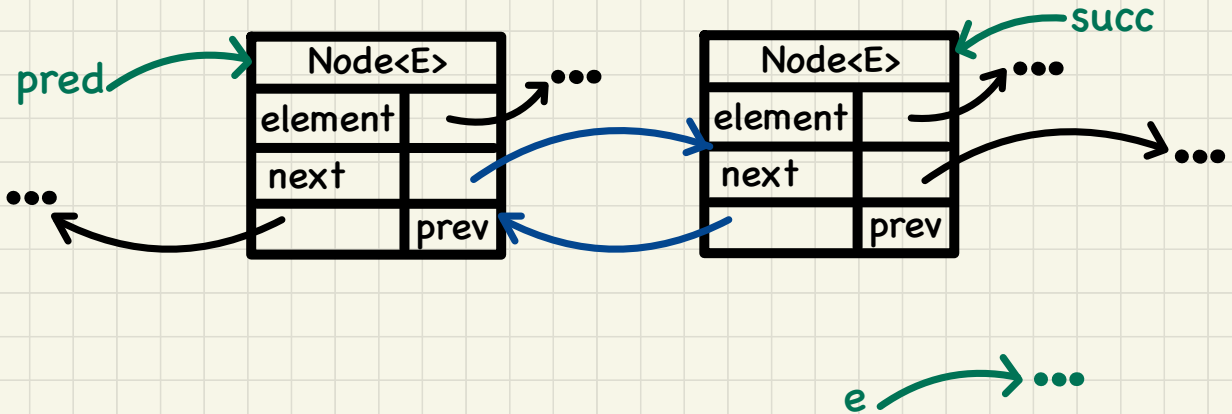


Generic DLL in Java: Inserting between Nodes

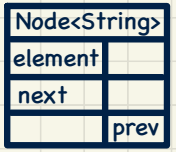



```
1  void addBetween(E e, Node<E> pred, Node<E> succ) {
2     Node<E> newNode = new Node<>(e, pred, succ);
3     pred.setNext(newNode);
4     succ.setPrev(newNode);
5     size ++;
6  }
```

Assumption: pred and succ are directly connected.



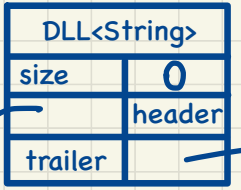
Generic DLL in Java: Inserting to the Front/End

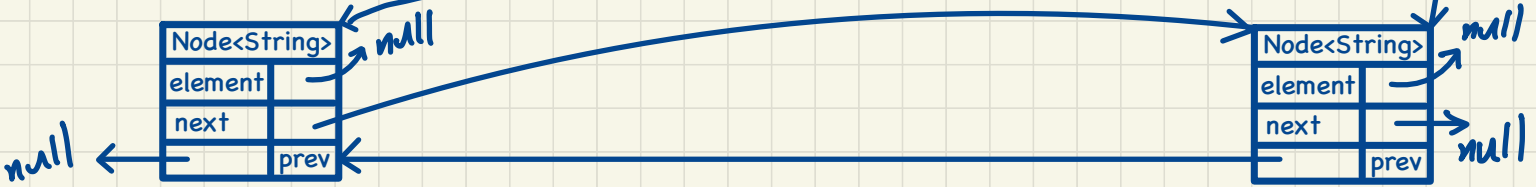



```java
@Test
public void test_String_DLL_Insert_Front_End() {
    DoublyLinkedList<String> list = new DoublyLinkedList<>();
    list.addFirst("Mark");
    list.addFirst("Alan");

    assertTrue(list.getSize() == 2);
    assertEquals("Alan", list.getFirst().getElement());
    assertEquals("Mark", list.getFirst().getNext().getElement());

    list = new DoublyLinkedList<>();
    list.addLast("Mark");
    list.addLast("Alan");

    assertTrue(list.getSize() == 2);
    assertEquals("Alan", list.getLast().getElement());
    assertEquals("Mark", list.getLast().getPrev().getElement());
}
```

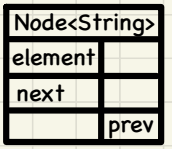
```java
void addFirst(E e) {
   addBetween(e, header, header.getNext())
}
```

```java
void addLast(E e) {
   addBetween(e, trailer.getPrev(), trailer)
}
```





Generic DLL in Java: Inserting to the Middle

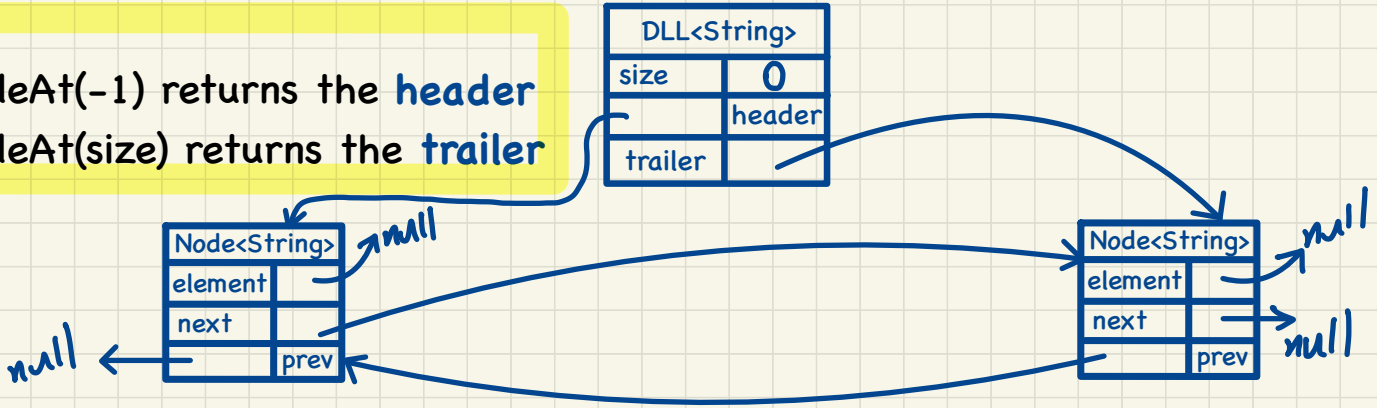



```java
@Test
public void test_String_DLL_addAt() {
    DoublyLinkedList<String> list = new DoublyLinkedList<>();
    list.addAt(0, "Alan");
    list.addAt(1, "Tom");
    list.addAt(1, "Mark");

    assertTrue(list.getSize() == 3);
    assertEquals("Alan", list.getFirst().getElement());
    assertEquals("Mark", list.getFirst().getNext().getElement());
    assertEquals("Tom", list.getFirst().getNext().getNext().getElement());
}
```

```java
addAt(int i, E e) {
  if (i < 0 || i > size) {
    throw new IllegalArgumentException
  else {
    Node<E> pred = getNodeAt(i - 1);
    Node<E> succ = pred.getNext();
    addBetween(e, pred, succ);
  }
}
```

Notes.

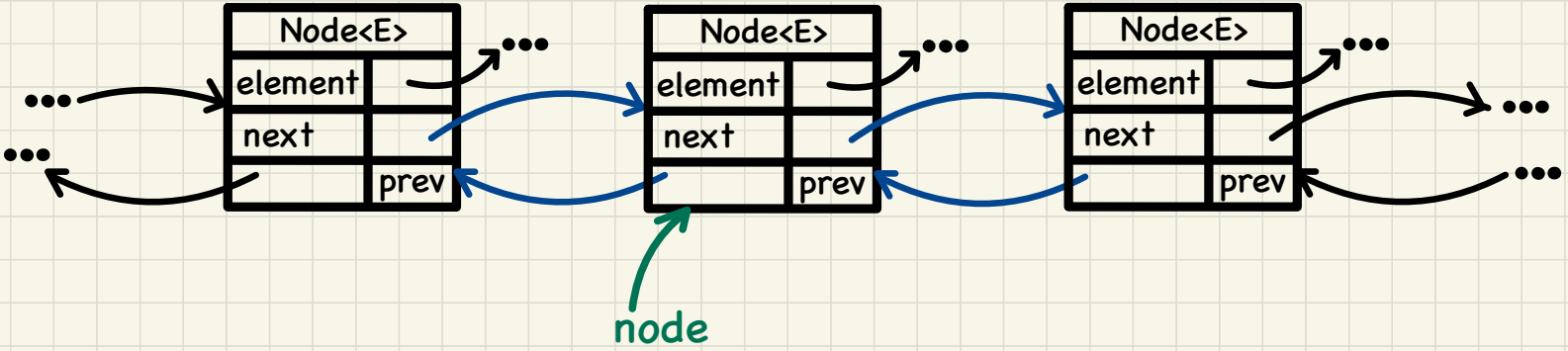
+ getNodeAt(-1) returns the **header**
+ getNodeAt(size) returns the **trailer**



Generic DLL in Java: Removing a Node

```
1  void remove (Node<E> node) {
2    Node<E> pred = node.getPrev();
3    Node<E> succ = node.getNext();
4    pred.setNext(succ);
5    succ.setPrev(pred);
6    node.setNext(null);
7    node.setPrev(null);
8    size --;
9  }
```

Assumption: node exists in some DLL.

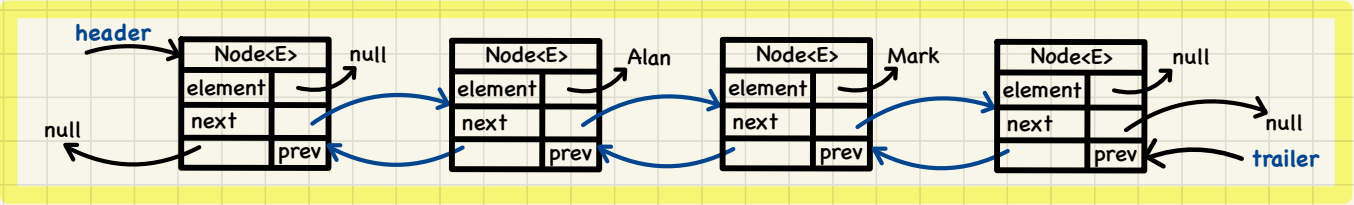


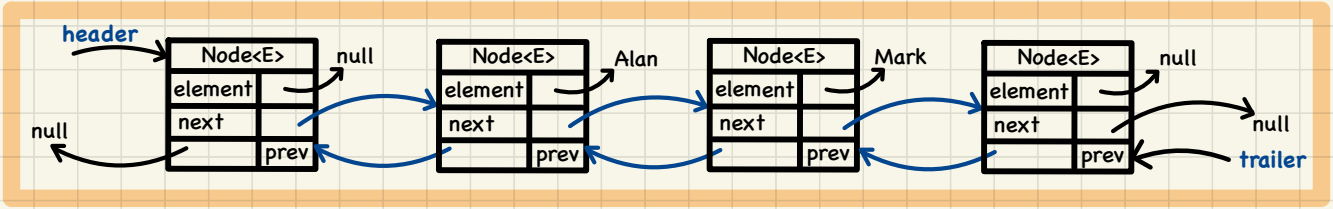
Generic DLL in Java: Removing from the Front/End

```java
@Test
public void test_String_DLL_Remove_Front_End() {
    DoublyLinkedList<String> list = new DoublyLinkedList<>();
    list.addFirst("Mark");
    list.addFirst("Alan");
    list.removeFirst();
    list.removeFirst();
    assertTrue(list.getSize() == 0);

    list = new DoublyLinkedList<>();
    list.addFirst("Mark");
    list.addFirst("Alan");
    list.removeLast();
    list.removeLast();
    assertTrue(list.getSize() == 0);
}
```

```java
void removeFirst() {
  if (size == 0) { throw new IllegalArgumentException("Empty"); }
  else { remove(header.getNext()); }
}
```

```java
void removeLast() {
  if (size == 0) { throw new IllegalArgumentException("Empty"); }
  else { remove(trailer.getPrev()); }
}
```





Generic DLL in Java: Removing from the Middle

```java
@Test
public void test_String_DLL_removeAt() {
    DoublyLinkedList<String> list = new DoublyLinkedList<>();
    list.addFirst("Mark");
    list.addFirst("Alan");
    list.addFirst("Tom");
    assertTrue(list.getSize() == 3);
    list.removeAt(1);
    assertTrue(list.getSize() == 2);
    list.removeAt(0);
    assertTrue(list.getSize() == 1);
    list.removeAt(0);
    assertTrue(list.getSize() == 0);
}
```

```java
removeAt (int i) {
  if (i < 0 || i >= size) {
    throw new IllegalArgumentExce
  else {
    Node<E> node = getNodeAt(i);
    remove (node);
  }
}
```