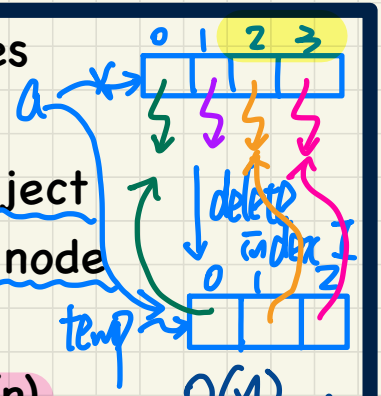
Lecture 2

Part C

Singly-Linked Lists - Intuitive Introduction

Singly-Linked Lists (SLL): Visual Introduction

- A chain of connected nodes
- Each node contains:
  - + reference to a data object
  - + reference to the next node
- Accessing a node in a list:
  - + Relative positioning: O(n)
  - + Absolute indexing: O(1)
- The chain may grow or shrink dynamically.
- Head vs. Tail



O(n) always.

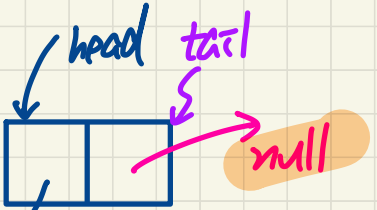
O(1) LL

int[] a = new int[_];

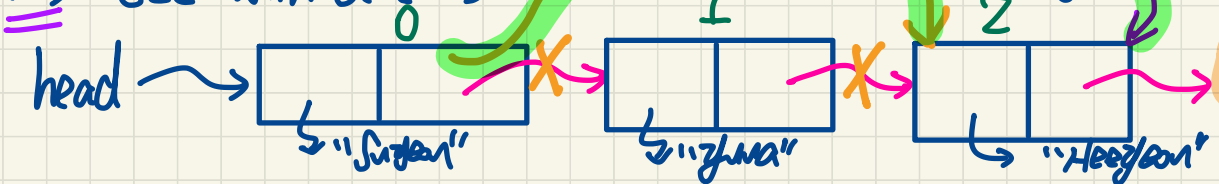
fixed length

(1) Empty SLL

(2) SLL with size 1



(3) SLL with size 3



"Sufyan"

tail Ref. Aliasing

tail == head.next

True.

Lecture 2

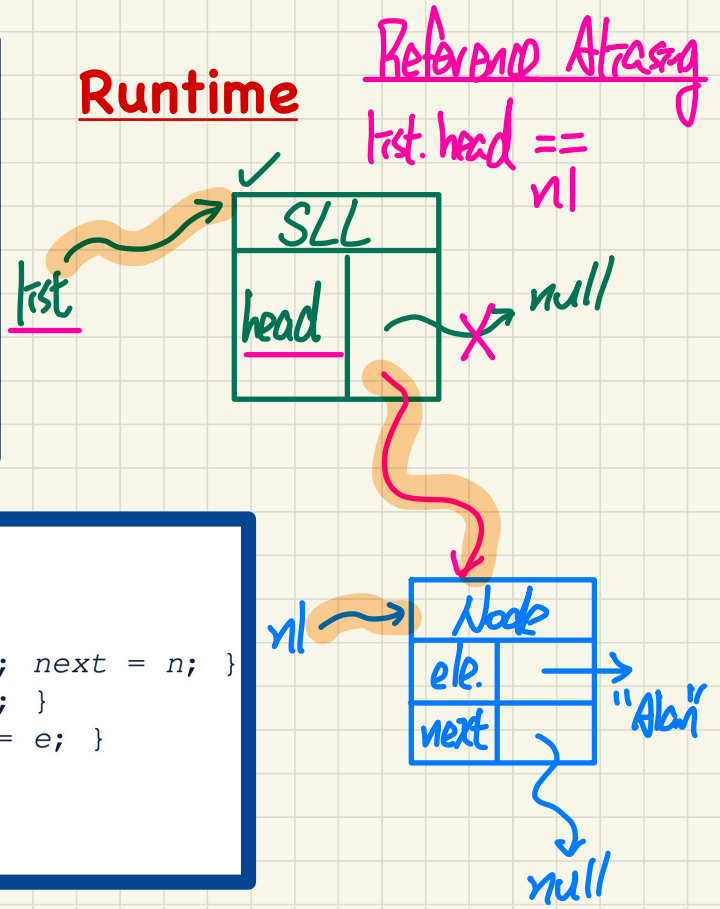
Part D

Singly-Linked Lists -
Java Implementation: String Lists
Initializing a List

Implementing SLL in Java: SinglyLinkedList vs. Node

```
public class SinglyLinkedList {
   private Node head = null;
   public void setHead(Node n) { head = n; }
   public int getSize() { ... }
   public Node getTail() { ... }
   public void addFirst(String e) { ... }
   public Node getNodeAt(int i) { ... }
   public void addAt(int i, String e) { ... }
   public void removeLast() { ... }
}
```

```
public class Node {
   private String element;
   private Node next;
   public Node(String e, Node n) { element = e; next = n; }
   public String getElement() { return element; }
   public void setElement(String e) { element = e; }
   public Node getNext() { return next; }
   public void setNext(Node n) { next = n; }
}
```



SLL: Constructing a Chain of Nodes

```
public class Node {
  private String element;
  private Node next;
  public Node(String e, Node n) { element = e; next = n; }
  public String getElement() { return element; }
  public void setElement(String e) { element = e; }
  public Node getNext() { return next; }
  public void setNext(Node n) { next = n; }
}
```

n = tom

n = mark

this.next

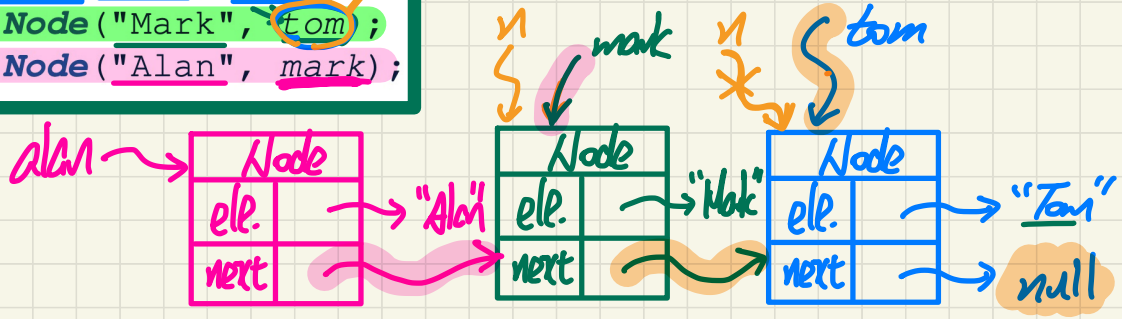
Ref. Aliasing

1. tom
2. mark.next
3. alan.next.next

call by value

Approach 1

```
Node tom = new Node("Tom", null);
Node mark = new Node("Mark", tom);
Node alan = new Node("Alan", mark);
```



SLL: Constructing a Chain of Nodes

```
public class Node {
   private String element;
   private Node next;
   public Node(String e, Node n) { element = e; next = n; }
   public String getElement() { return element; }
   public void setElement(String e) { element = e; }
   public Node getNext() { return next; }
   public void setNext(Node n) { next = n; }
}
```

Approach 2

```
Node alan = new Node("Alan", null);
Node mark = new Node("Mark", null);
Node tom = new Node("Tom", null);
alan.setNext(mark);
mark.setNext(tom);
```

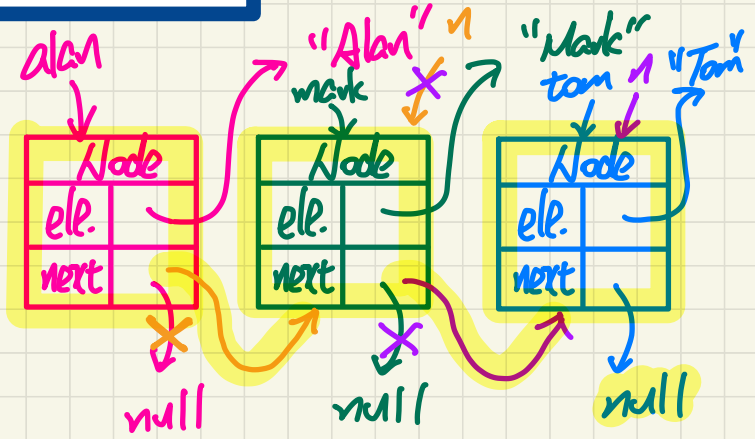
context object

call by value

alan.next = mark

n = mark

n = tom



SLL: Setting a List's Head to a Chain of Nodes

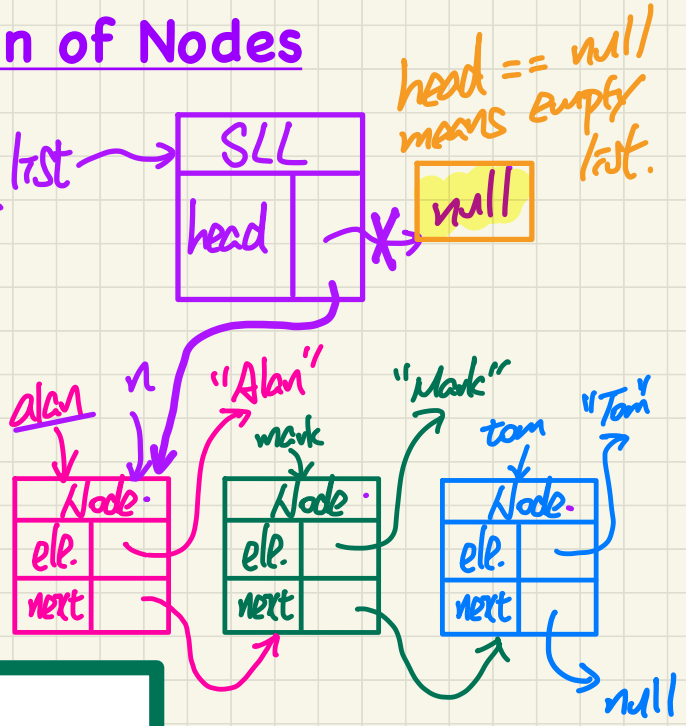
```
public class SinglyLinkedList {
   private Node head = null;
   public void setHead(Node n) { head = n; }
   public int getSize() { ... }
   public Node getTail() { ... }
   public void addFirst(String e) { ... }
   public Node getNodeAt(int i) { ... }
   public void addAt(int i, String e) { ... }
   public void removeLast() { ... }
}
```

n = alan last this.head = n.

Approach 1

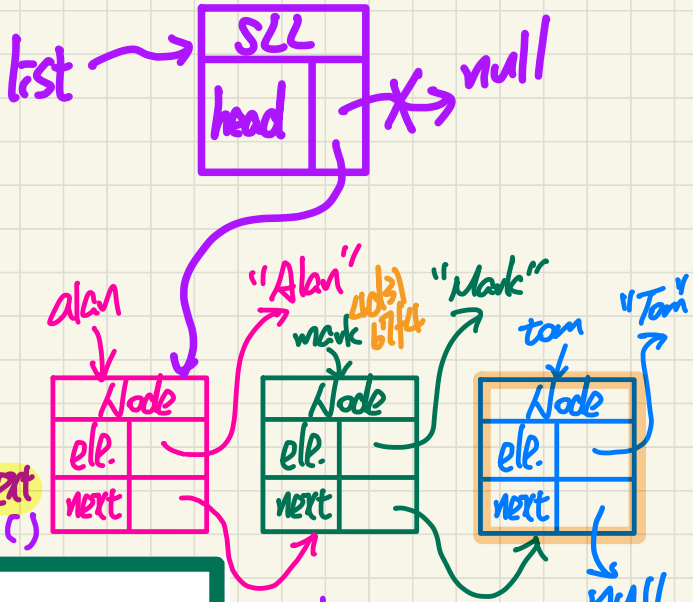
```
Node tom = new Node("Tom", null);
Node mark = new Node("Mark", tom);
Node alan = new Node("Alan", mark);
SinglyLinkedList list = new SinglyLinkedList();
list.setHead(alan);
```

call by value



SLL: Setting a List's Head to a Chain of Nodes

```
public class SinglyLinkedList {
   private Node head = null;
   public void setHead(Node n) { head = n; }
   public int getSize() { ... }
   public Node getTail() { ... }
   public void addFirst(String e) { ... }
   public Node getNodeAt(int i) { ... }
   public void addAt(int i, String e) { ... }
   public void removeLast() { ... }
}
```



Approach 2

④ list.getFirst().getNext().getNext()

```
Node alan = new Node("Alan", null);
Node mark = new Node("Mark", null);
Node tom = new Node("Tom", null);
alan.setNext(mark);
mark.setNext(tom);
SinglyLinkedList list = new SinglyLinkedList();
list.setHead(alan);
```

Q. How many paths to reach "Tom" object?

① tom
② mark.getNext()
③ alan.getNext().getNext()

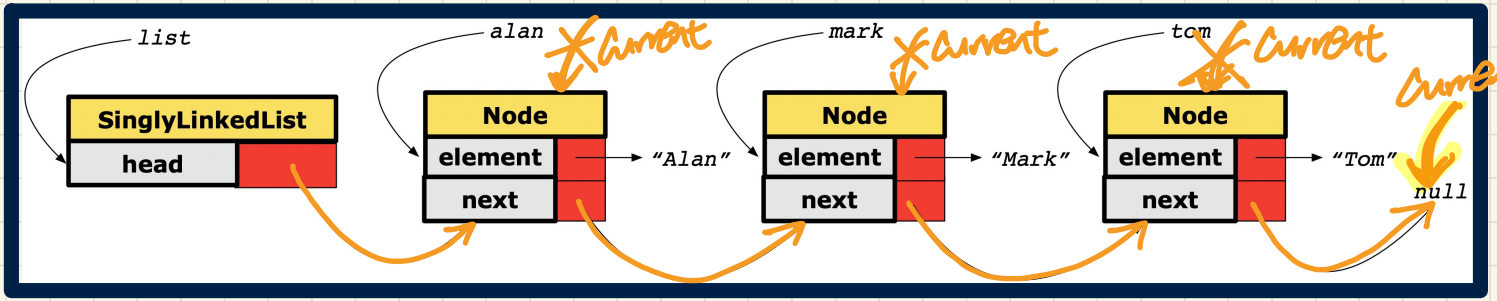
Lecture 2

Part E

Singly-Linked Lists -
Java Implementation: String Lists
Operations on a List

SLL Operation: Counting the Number of Nodes

i = i + 1




```
1  int getSize() {
2    int size = 0;
3    Node current = head;
4    while (current != null) {
5      current = current.getNext();
6      size ++;
7    }
8    return size;
9  }
```

Exit: current == null

O(1)

starting from 1st node in chain, we keep calling getNext() until current becomes null

Trace: list.getSize()

current	current != null	End of Iteration	size
alan	true	1	1
mark	true	2	2
tom	true	3	3
null	false	O(1)	

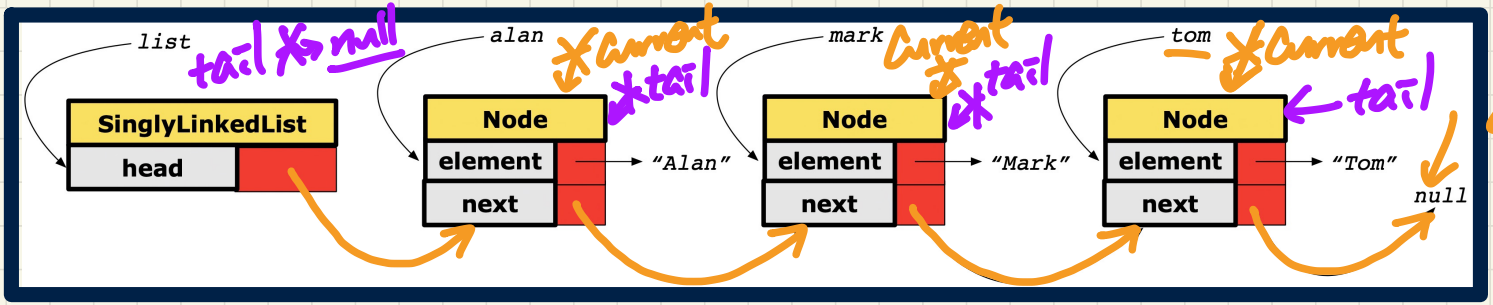
reaching null. (n iterations)

1. Empty list
2. Non-empty list

head → null

Empty SLL

SLL Operation: Finding the Tail of the List

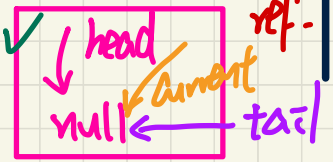



```
1  Node getTail() {
2      Node current = head;
3      Node tail = null;
4      while (current != null) {
5          tail = current;
6          current = current.getNext();
7      }
8      return tail;
9  }
```

O(n)

how many iterations before getting null

1. Empty list
2. Non-empty list



Trace: list.getTail()

current	current != null	End of Iteration	tail
alan	true	1	alan
mark	true	2	mark
tom	true	3	tom
null	fake		

SLL Operation: Inserting to the Front of the List

Exercise: Use debugger to trace.

```
@Test
public void testSLL_02() {
    SinglyLinkedList list = new SinglyLinkedList();
    assertTrue(list.getSize() == 0);
    assertTrue(list.getFirst() == null);

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

return values of att. size and head ⇒ O(1)

user/caller friendly
Alt: Node n

```
1  void addFirst (String e) {
2      head = new Node(e, head);
3      if (size == 0) {
4          tail = head;
5      }
6      size ++;
7  }
```

O(1)

overhead of declaring size as an attribute.