Lecture 9 - Wednesday, February 8

Announcements

- Released soon:
  + WrittenTest1 result (Friday or Monday the latest)
  + Assignment1 solution
- Assignment 2 to be released by the end of today or early tomorrow (Thursday)

by Thursday.

- To make up the test time on Monday, videos will be released
  ↳ Assumed by next Week's class

Lecture

Arrays vs. Linked Lists

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

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

linear: each node has a unique successor

int[] a = new int[_]; fixed length.

head != null : 1st node

head.next != null : 2nd node

head.next.next != null : 3rd node

head.data : "Alan"

head.next.data : "Mark"

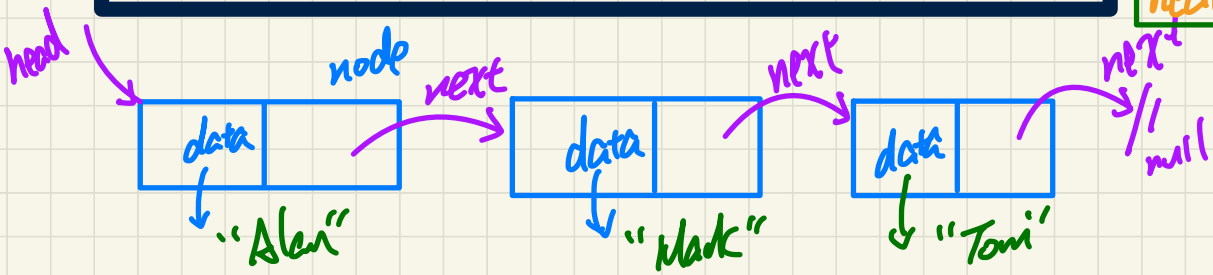
head.next.next.data : "Tom"

(1) head.next.next.next (null!)

(2) head.next.next.next.data

null

NullPointerExcep.



ArrayList library class

↳ resizable array

↳ doubling

Linked-Lists

↳ good for implementing specialized ops.

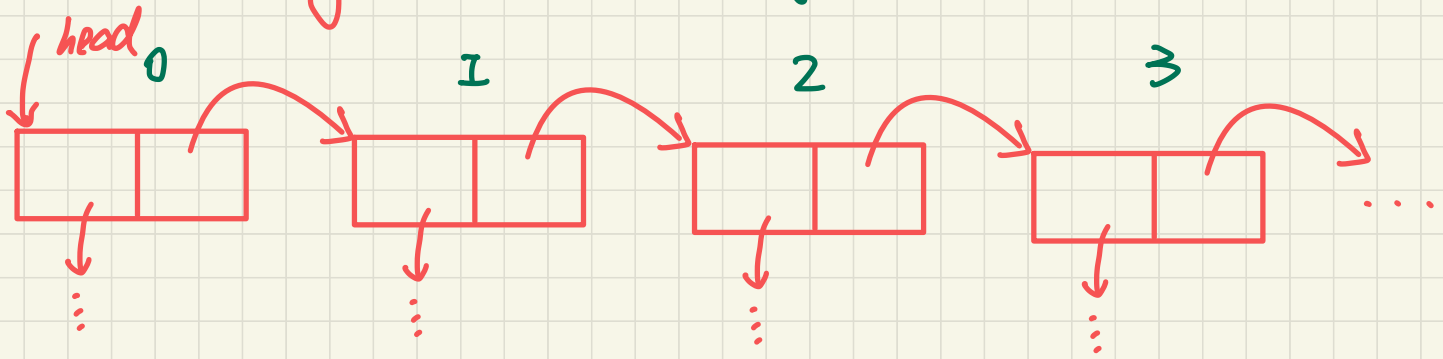
Absolute Indexing of Arrays

$a[i] \rightarrow O(1)$

int.

Relative Positioning of LL

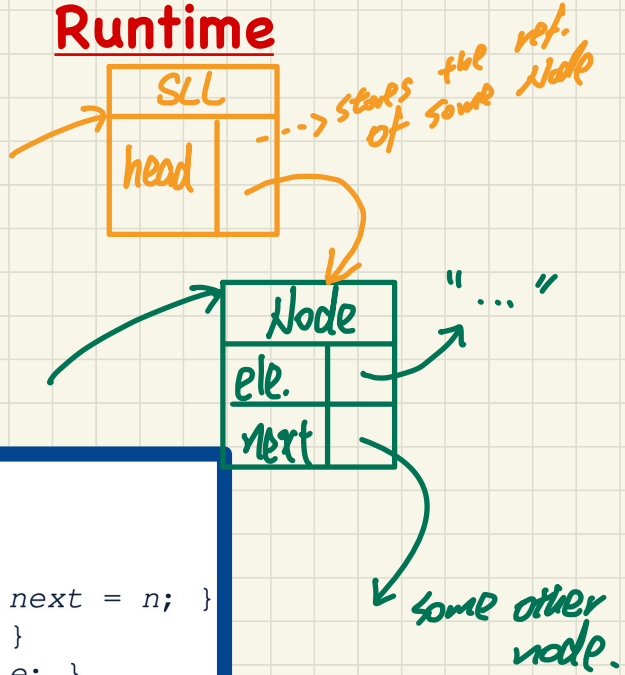
$O(n)$ getNodeAt(i) — position in chain of nodes



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

tom → mark → alan.

```
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; }
}
```

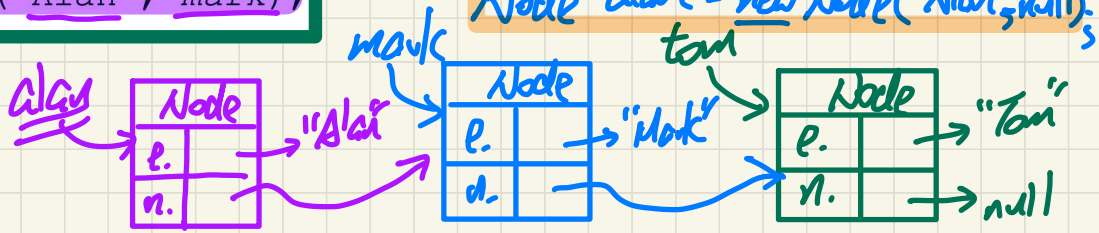
Approach 1

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

Exercise

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

not compiling!



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 x) { next = x; }
}
```

mark this alan mark.
tom mark tom

Approach 2

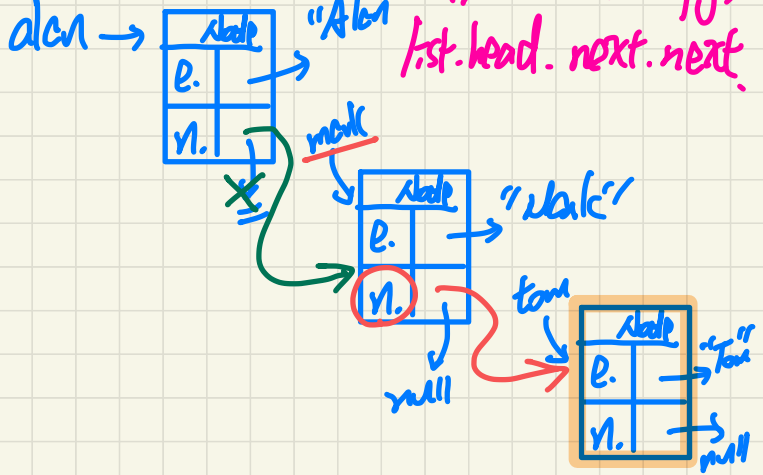
```
Node alan = new Node("Alan", null);
Node mark = new Node("Mark", null);
Node tom = new Node("Tom", null);
alan.setNext(mark);
mark.setNext(tom);
```

alan.next = mark ;
mark.next = tom ;

Aliasing

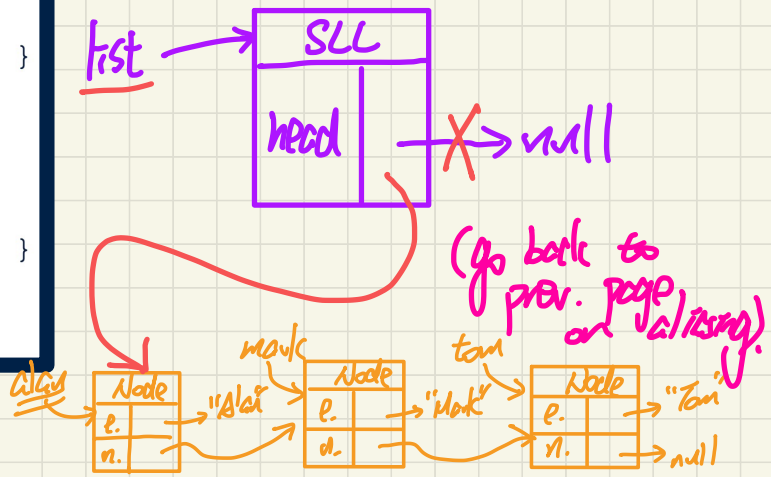
↳ an object's ref being stored in multiple variables.

1. tom
2. mark.next
3. alan.next.next
4. (see next pg.) list.head.next.next



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() { ... }
}
```



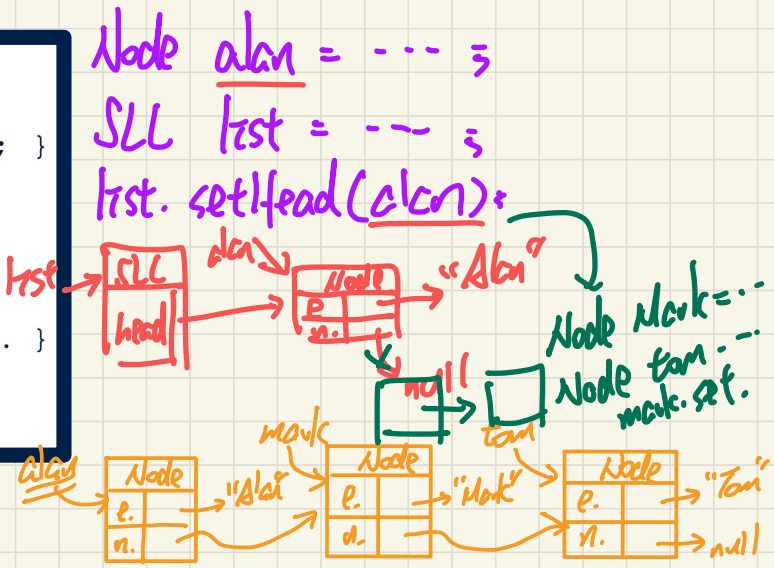
Approach 1

```
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);
```

initialize head to default null

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

```
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);
```

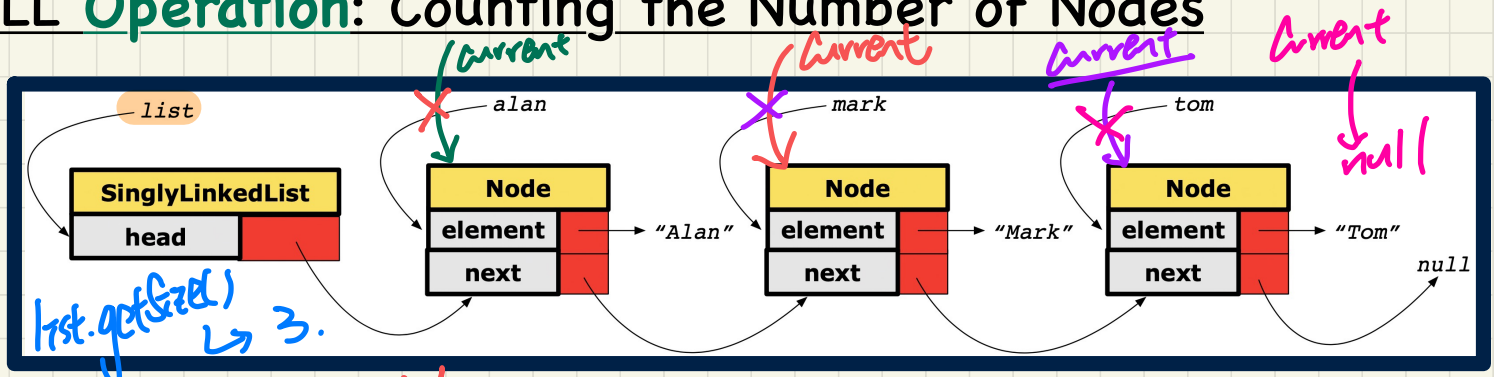
→ identical to Approach 1.

Lecture

Arrays vs. Linked Lists

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

SLL Operation: Counting the Number of Nodes



SLL method

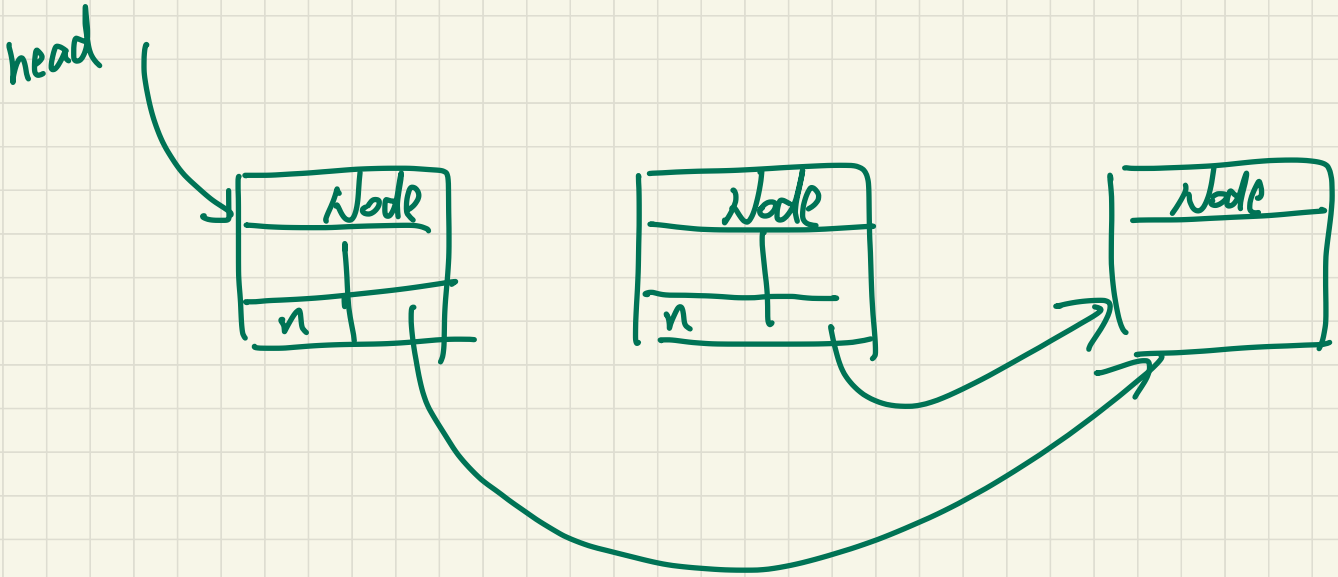
```
1  int getSize() {
2     int size = 0;
3     Node current = head;
4     while (current != null) {
5        current = current.getNext();
6        size ++;
7     }
8     return size;
9  }
```

exit when current == null

Trace: list.getSize()

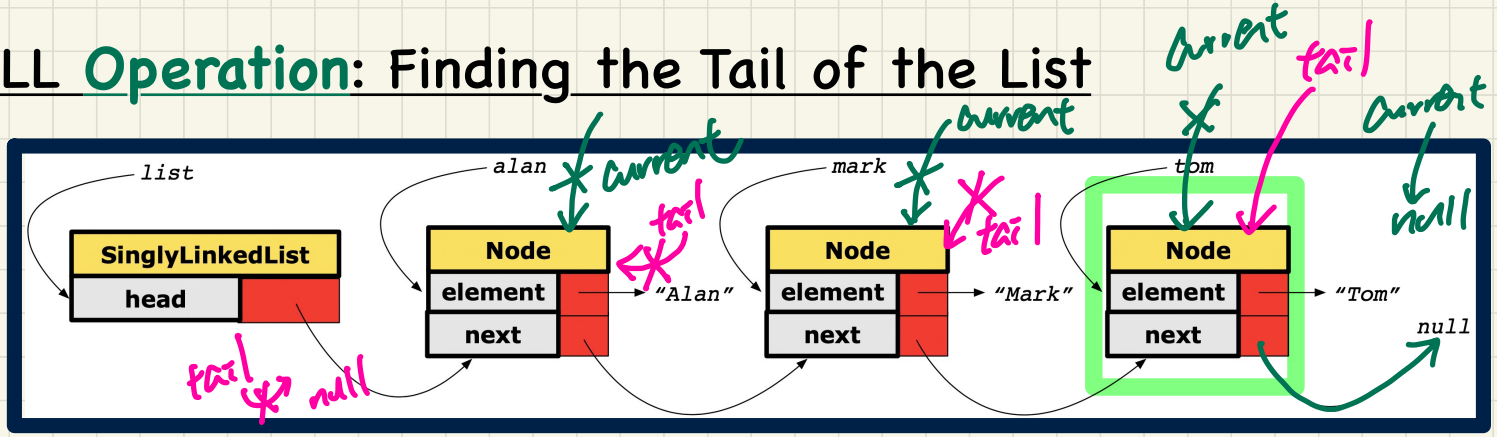
current	current != null	End of Iteration	size
alan	alan != null (T)	current == mark	1
mark	mark != null (T)	current == tom	2
tom	tom != null (T)	current == null	3
null	null != null (F)		

0(n)



(2).

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)

Trace: list.getTail()

current	current != null	End of Iteration	tail

SLL class

- head
- tail
- size

list.tail
list.size

O(1)

tracking size for time.