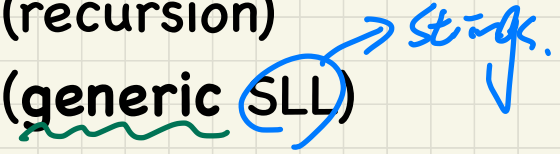
Lecture 11 - Monday, February 15

Announcements

- Assignment 2 released
  + Required & Recommended Studies
  + Looking Ahead: Programming Test 1
    - Monday, Feb. 27; during class time; WSC; 1 hour
    - Covers:
      * Assignment 1 (recursion)
      * Assignment 2 (generic SLL)

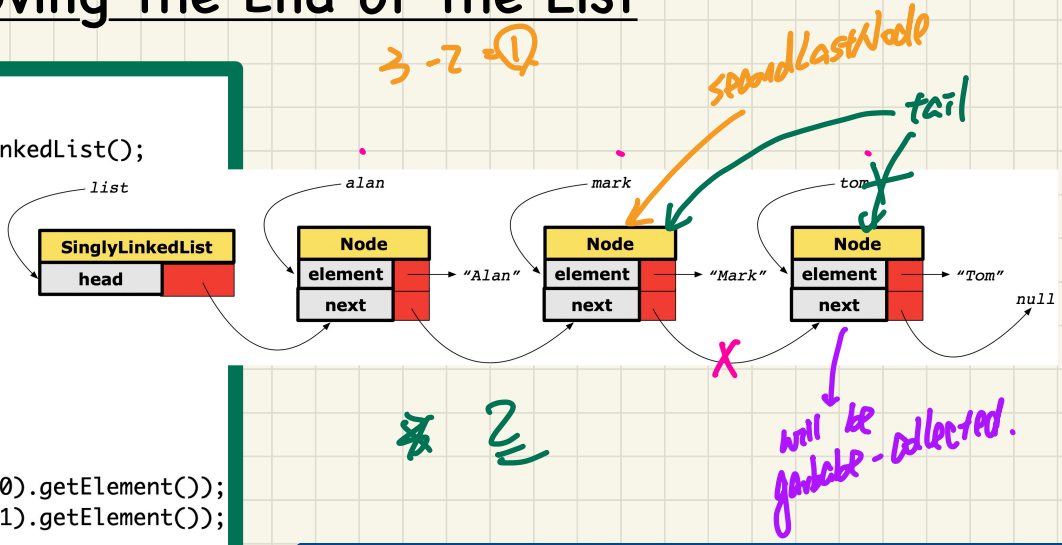


- Assignment 1 solution released

SLL Operation: Removing the End of the List

```java
@Test
public void testSLL_removeLast() {
    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);

    list.removeLast();
    assertTrue(list.getSize() == 2);
    assertEquals("Alan", list.getNodeAt(0).getElement());
    assertEquals("Mark", list.getNodeAt(1).getElement());

    list.removeLast();
    assertTrue(list.getSize() == 1);
    assertEquals("Alan", list.getNodeAt(0).getElement());

    list.removeLast();
    assertTrue(list.getSize() == 0);
    assertNull(list.getFirst());
}
```

O(n)



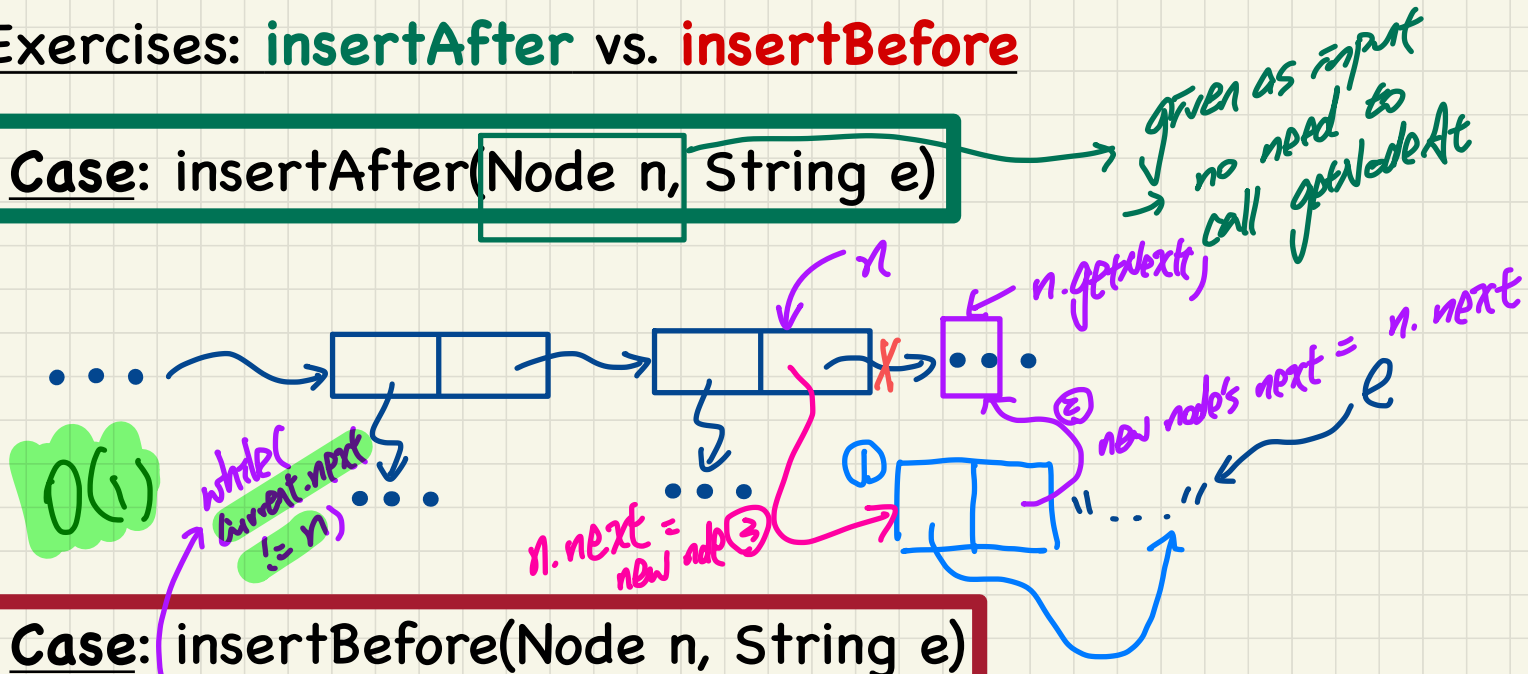

```java
1  void removeLast () {
2    if (size == 0) {
3      throw new IllegalArgumentException("Empty List.");
4    }
5    else if (size == 1) {
6      removeFirst();
7    }
8    else {
9      Node secondLastNode = getNodeAt(size - 2);
10     secondLastNode.setNext(null);
11     tail = secondLastNode;
12     size --;
13   }
14 }
```

O(n)

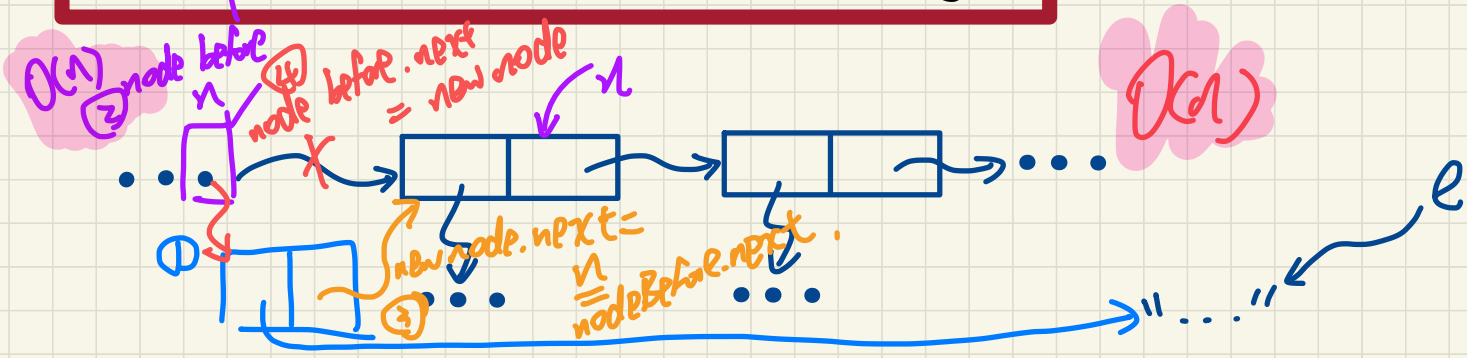
size-1: index of last node

Exercises: insertAfter vs. insertBefore

Case: insertAfter(Node n, String e)



Case: insertBefore(Node n, String e)



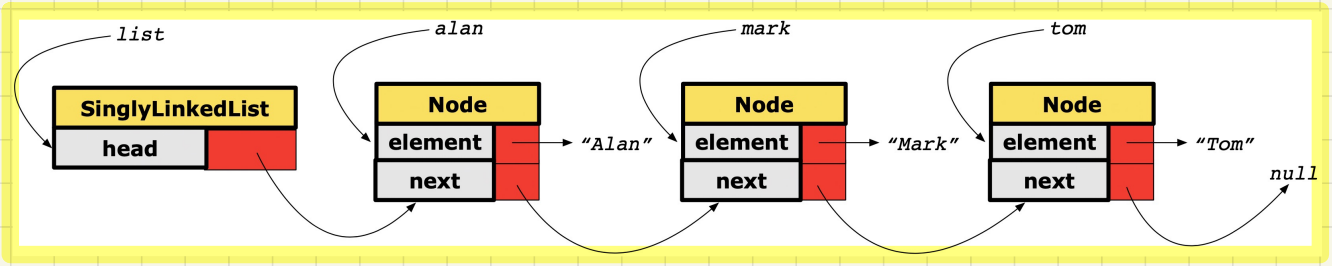
Lecture

Arrays vs. Linked Lists

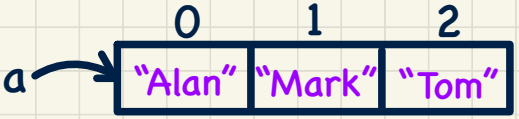
Singly-Linked Lists -
Comparing Arrays and Singly-Linked Lists

Running Time: Arrays vs. Singly-Linked Lists

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



SLL: remove ith node
↳ given the ref to (i-1)th node



Lecture

Arrays vs. Linked Lists

Doubly-Linked Lists - Intuitive Introduction

Why DLL?

1. performance (e.g. removeLast)

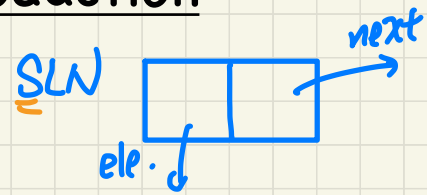
2. code structure
   - don't need to worry about edge cases

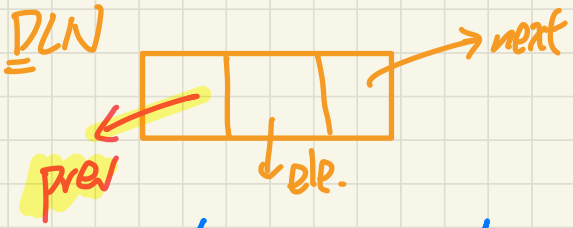
Doubly-Linked Lists (DLL): Visual Introduction

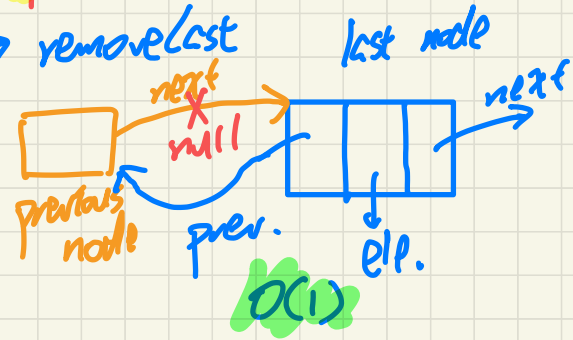
- A chain of bi-directionally connected nodes
- Each node contains:
  - + reference to a data object
  - + reference to the next node
  - + reference to the previous node
- A DLL is also a SLL:
  - + many methods implemented the same way
  - + some method implemented more efficiently
- Accessing a node in a list:
  - + Relative positioning: O(n)
  - + Absolute indexing: O(1)
- The chain may grow or shrink dynamically.
- Dedicated Header vs. Trailer Nodes
  (no head refeference and no tail reference)

having the prev. ref. does not help.

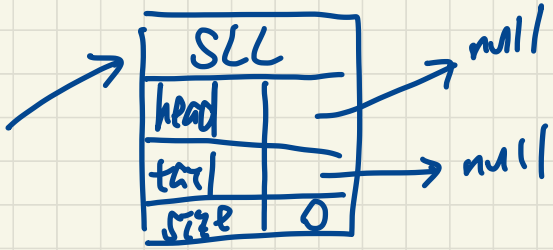
DLN



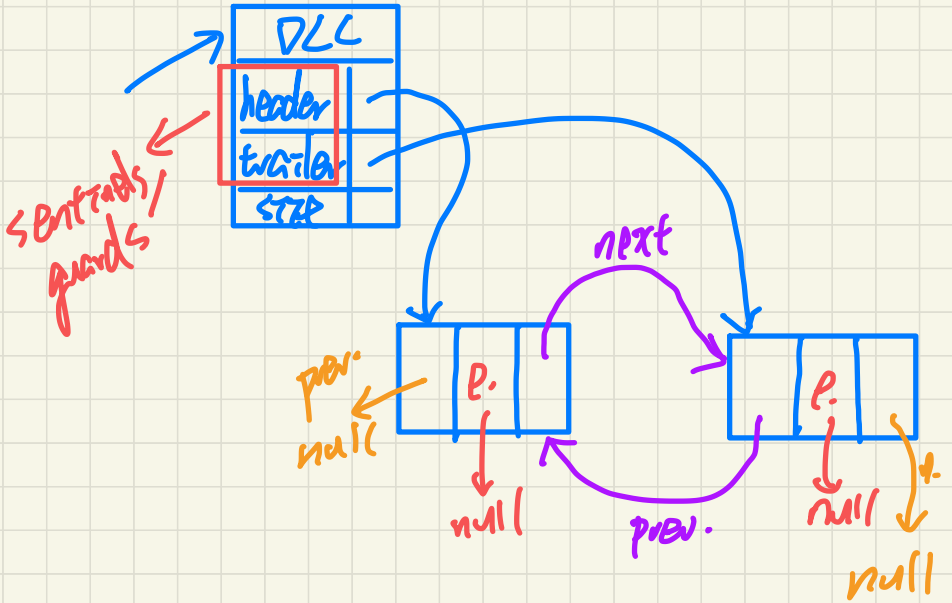




SLL


SLL
head
tail
size
0
null
null

DLL


DLL
header
trailer
size
sentinels/
guards
next
prev.
null
e.
null
prev.
e.
null
null

Lecture

Arrays vs. Linked Lists

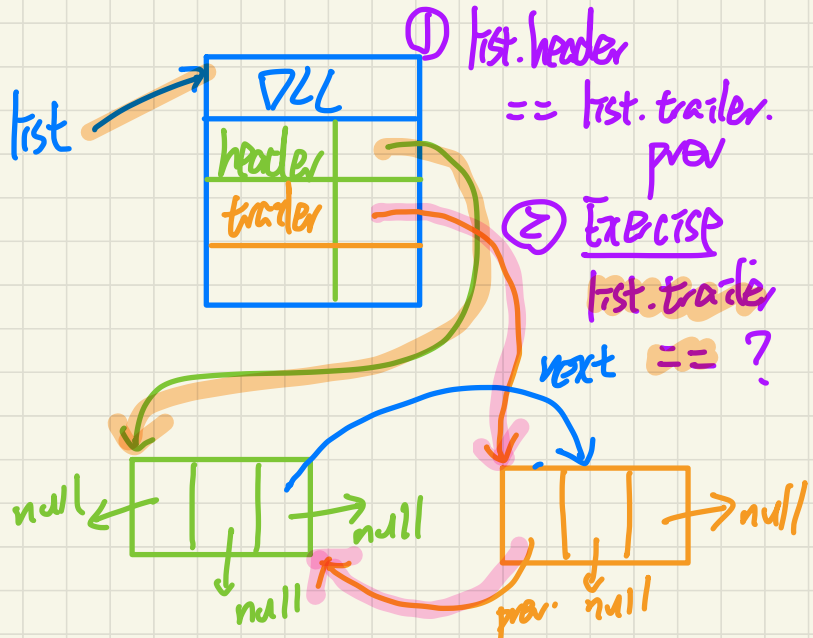
Doubly-Linked Lists -
Java Implementation: Generic Lists
Initializing a List

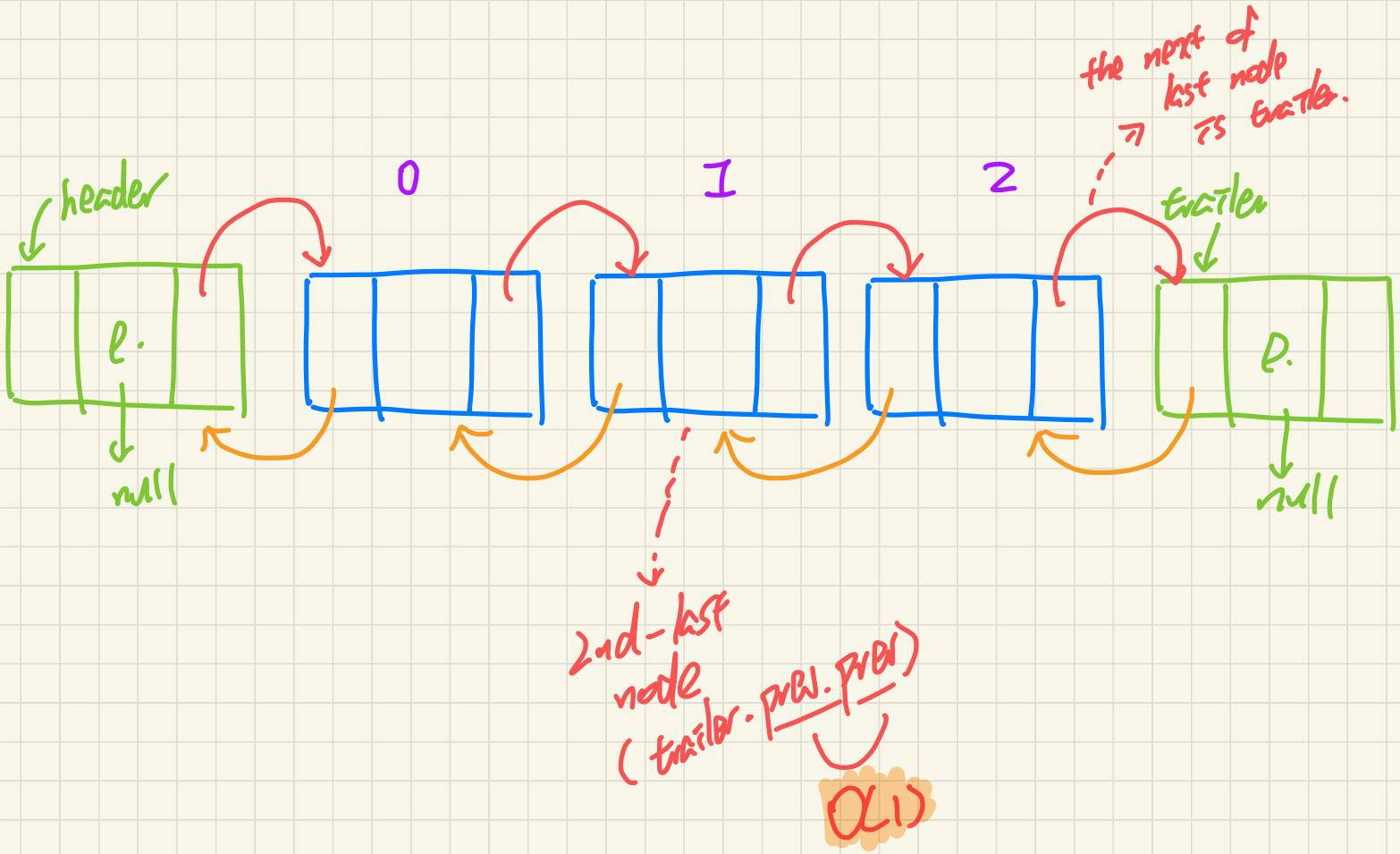
Generic DLL in Java: DoublyLinkedList vs. Node

```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
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;
   }
}
```

```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);
}
```




header
0
1
2
the next of last node is trailer.
trailer
e.
e.
null
null
2nd-last node
(trailer.prev.prev)
O(1)

Lecture

Arrays vs. Linked Lists

Doubly-Linked Lists -
Java Implementation: Generic Lists
Operations on a List

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.next == succ, succ.prev == pred.

without these two lines, the new node remains unreachable from the list.

Node<E>	
element	
next	
	prev

Assumption: pred and succ are directly connected.