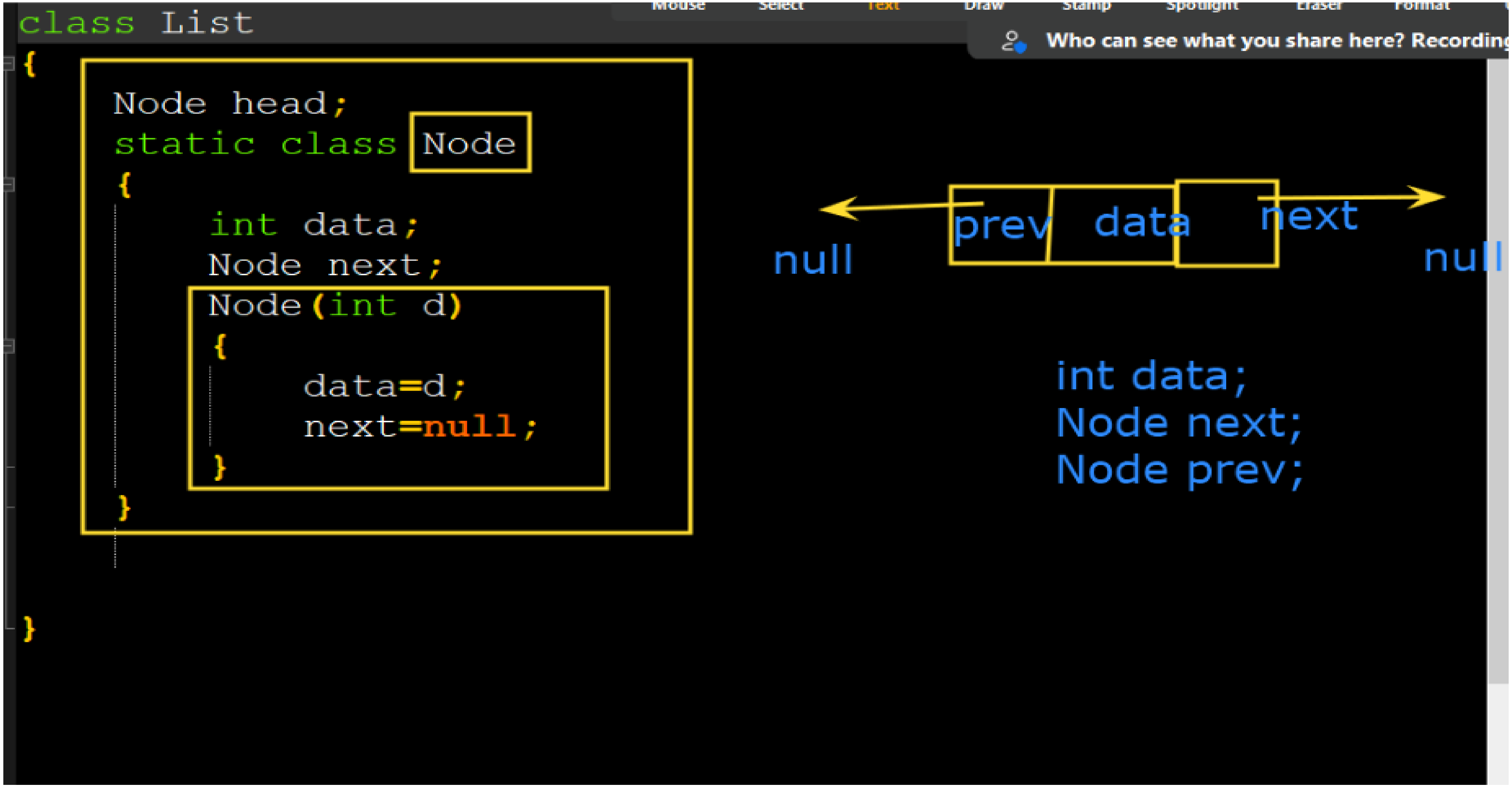
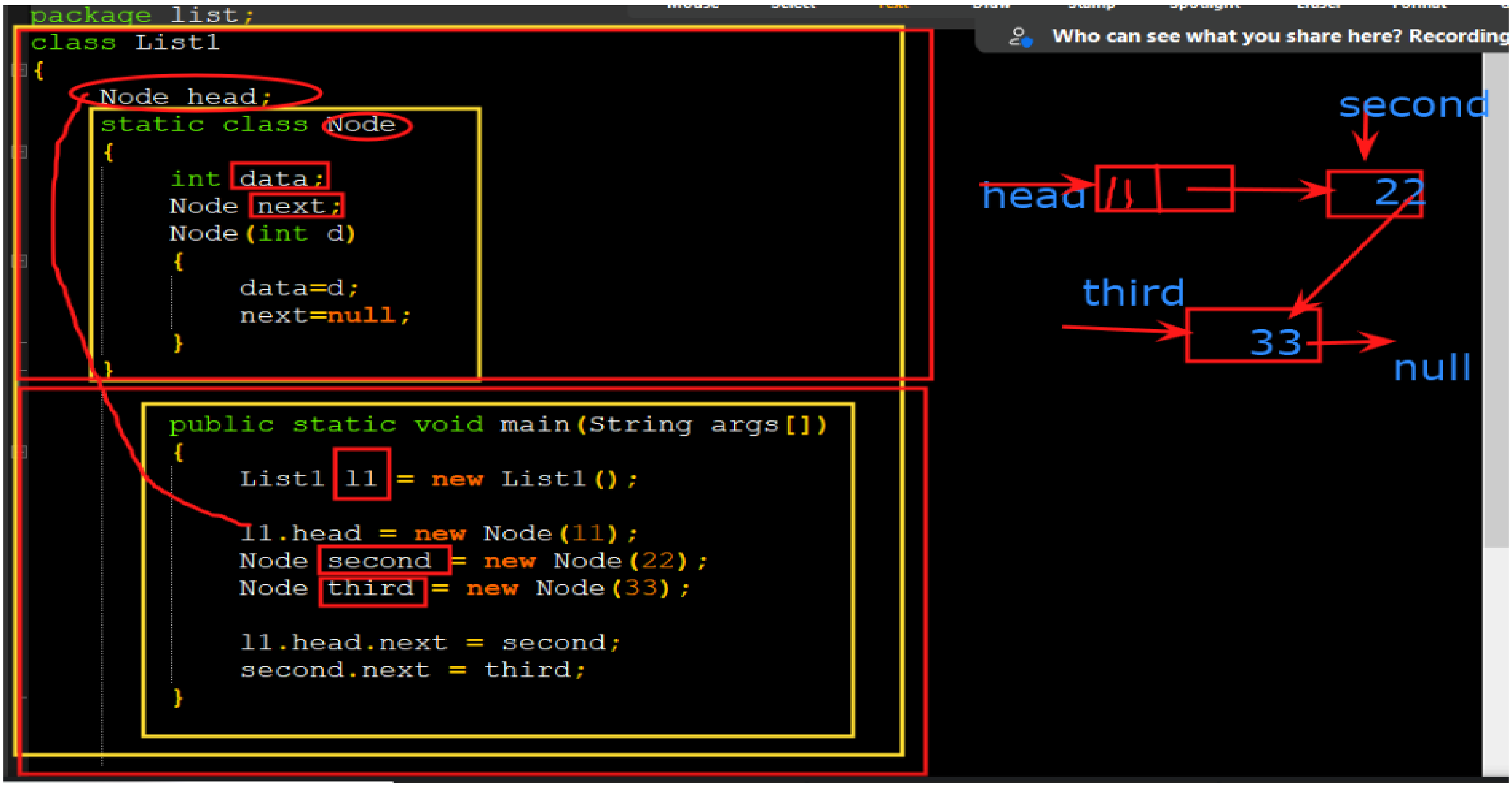
Algorithms & Data Structure

Kiran Waghmare

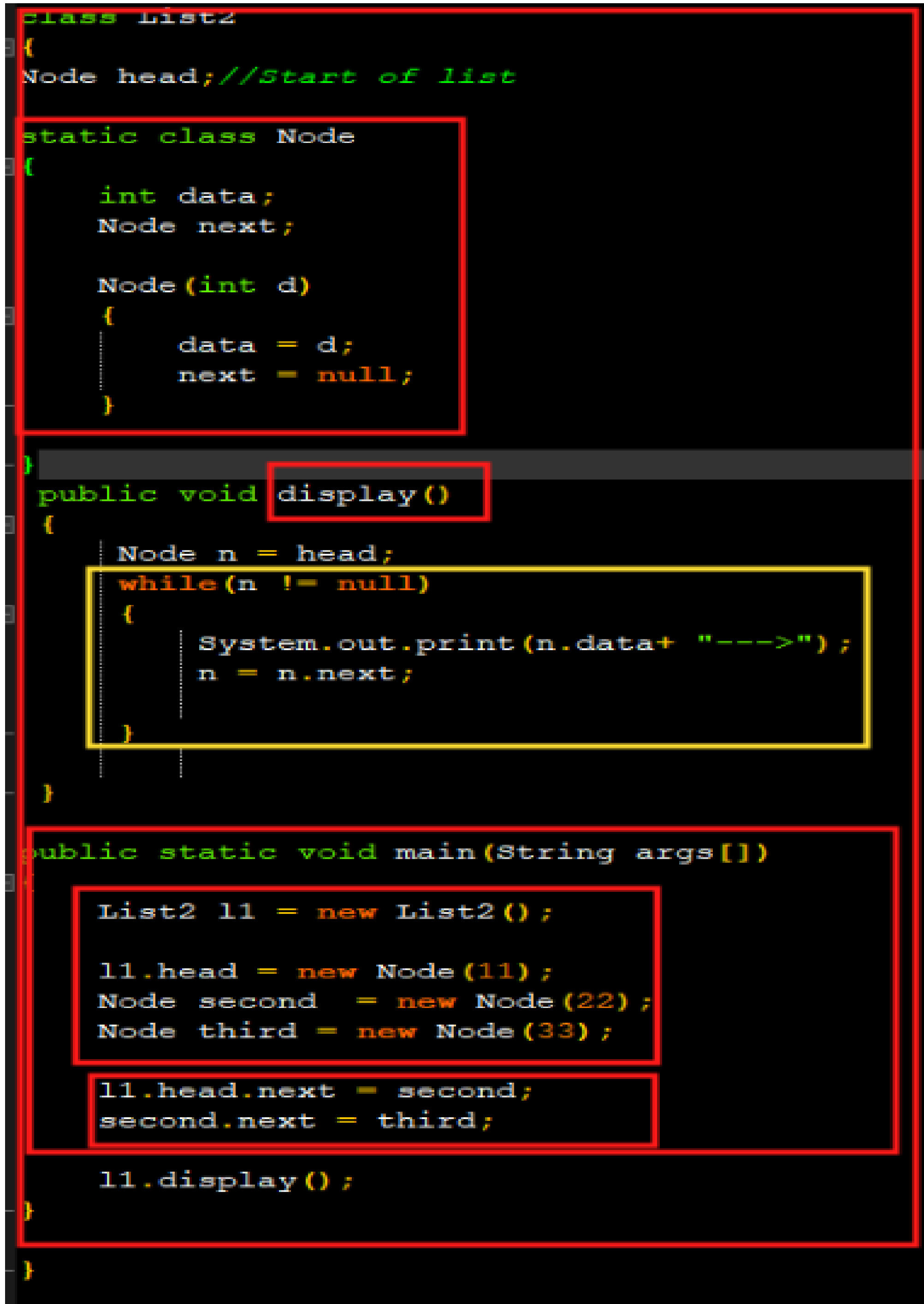


```
class List
{
    Node head;
    static class Node
    {
        int data;
        Node next;
        Node(int d)
        {
            data=d;
            next=null;
        }
    }


}
```

int data;
Node next;
Node prev;

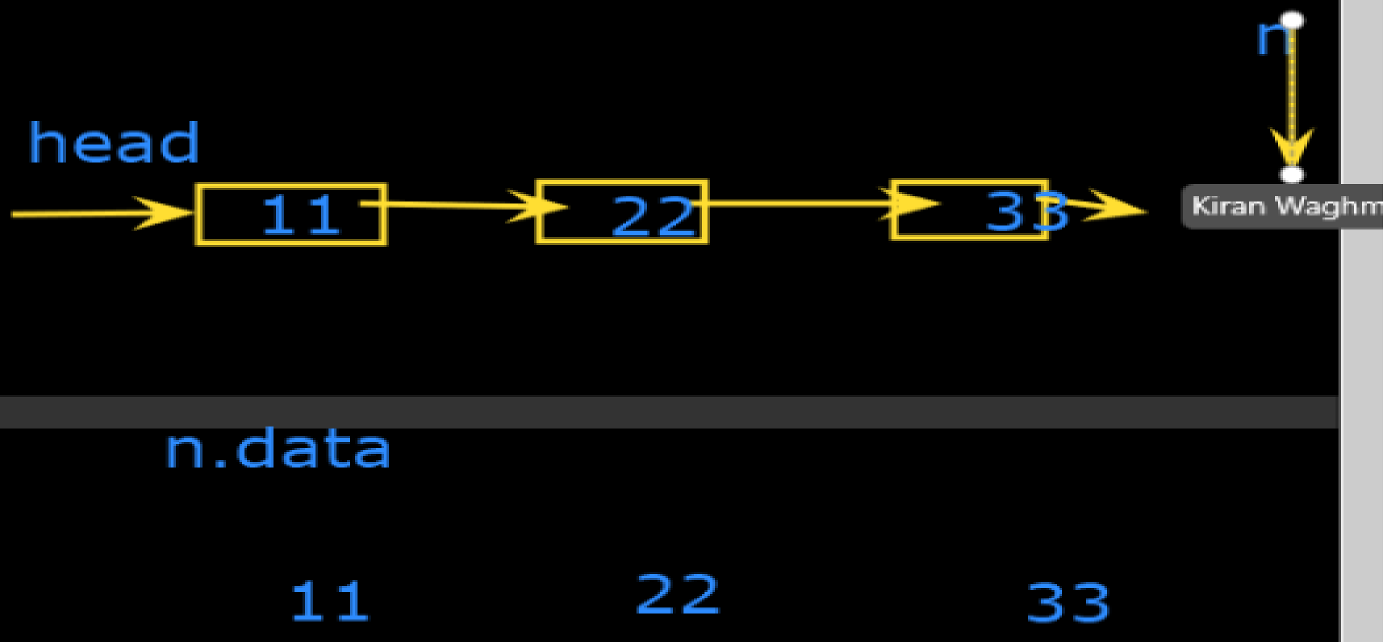


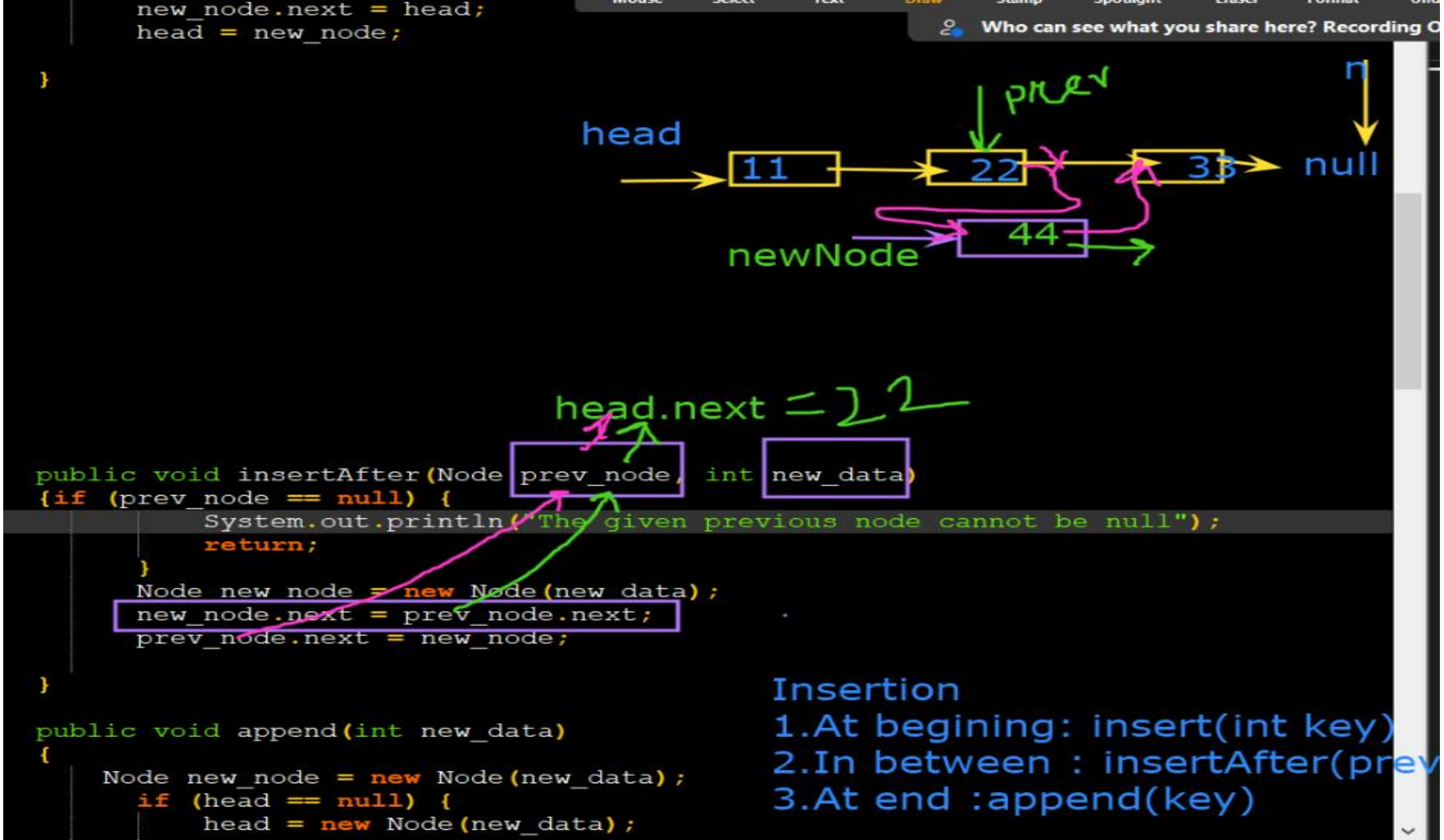
```
package list;
class List1
{
    Node head;
    static class Node
    {
        int data;
        Node next;
        Node(int d)
        {
            data=d;
            next=null;
        }
    }

        public static void main(String args[])
        {
            List1 l1 = new List1();

            l1.head = new Node(11);
            Node second = new Node(22);
            Node third = new Node(33);

            l1.head.next = second;
            second.next = third;
        }
```



```
class List2
{
Node head;//Start of list

static class Node
{
    int data;
    Node next;

    Node(int d)
    {
        data = d;
        next = null;
    }

}
 public void display()
 {
     Node n = head;
     while(n != null)
     {
         System.out.print(n.data+ "--->");
         n = n.next;

     }

 }

public static void main(String args[])
{
    List2 l1 = new List2();

    l1.head = new Node(11);
    Node second  = new Node(22);
    Node third = new Node(33);

    l1.head.next = second;
    second.next = third;

    l1.display();
}

}
```

Who can see what you share here? Recording

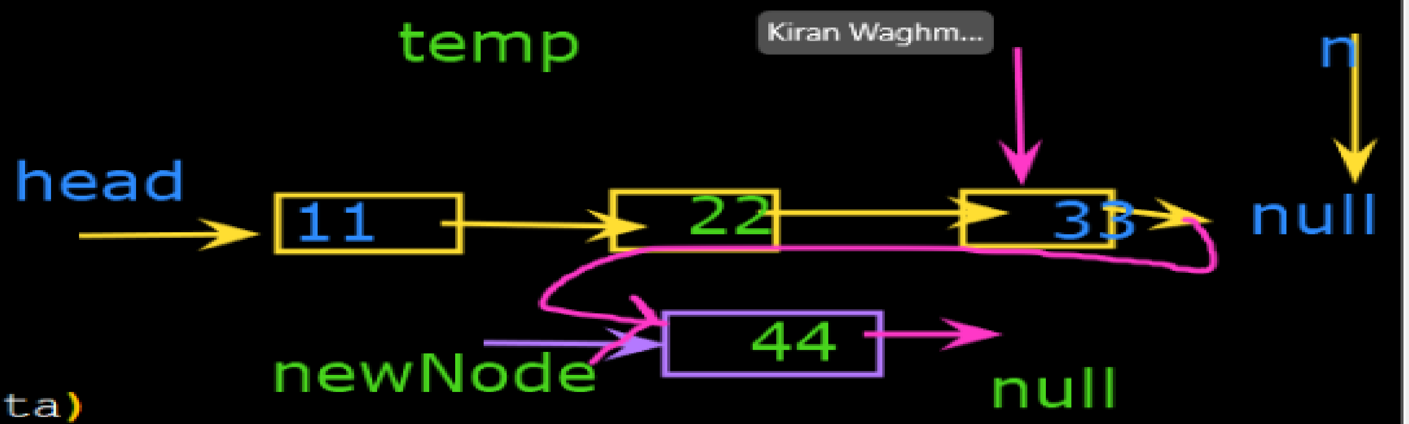




```
        new_node.next = head;
        head = new_node;

}
```

```
public void insertAfter(Node prev_node, int new_data)
{if (prev_node == null) {
            System.out.println("The given previous node cannot be null");
            return;
        }
        Node new_node = new Node(new_data);
        new_node.next = prev_node.next;
        prev_node.next = new_node;

}

public void append(int new_data)
{
    Node new_node = new Node(new_data);
        if (head == null) {
            head = new Node(new_data);
```

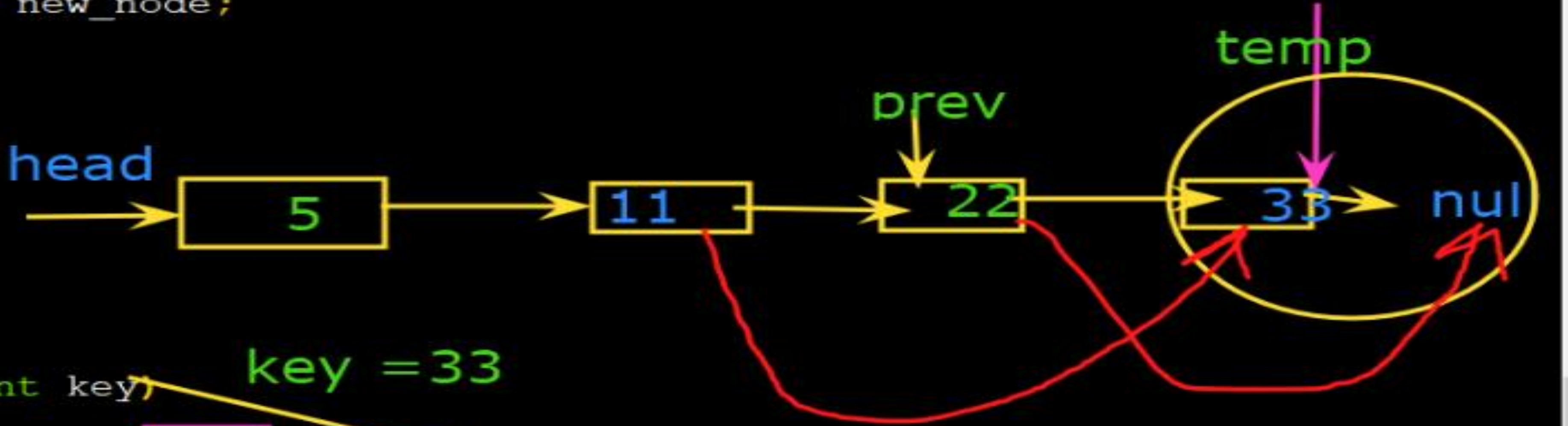
Insertion
1.At begining: insert(int key)
2.In between : insertAfter(prev
3.At end :append(key)



```
public void append(int new_data)
{
    Node new_node = new Node(new_data);
      if (head == null) {
          head = new Node(new_data);
          return;
      }
      new_node.next = null;
      Node last = head;
      while (last.next != null)
          last = last.next;
      last.next = new_node;
      return;


}
```

Insertion
1.At begining: insert(int key)
2.In between : insertAfter(prev
3.At end :append(key)

```
            last = last.next;
        last.next = new_node;
        return;

    }
void deleteNode(int key)
    {
        Node temp = head, prev = null;
        if(temp != null && temp.data == key)
        {
            head = temp.next;
        return;
        }
        while ( temp !=null && temp.data != key)
        {
            prev = temp;
            temp = temp.next;
        }
        if(temp == null)
            return;
        prev.next=temp.next;
    }
    public void deletelist()
    {
        head =null;
    }
```



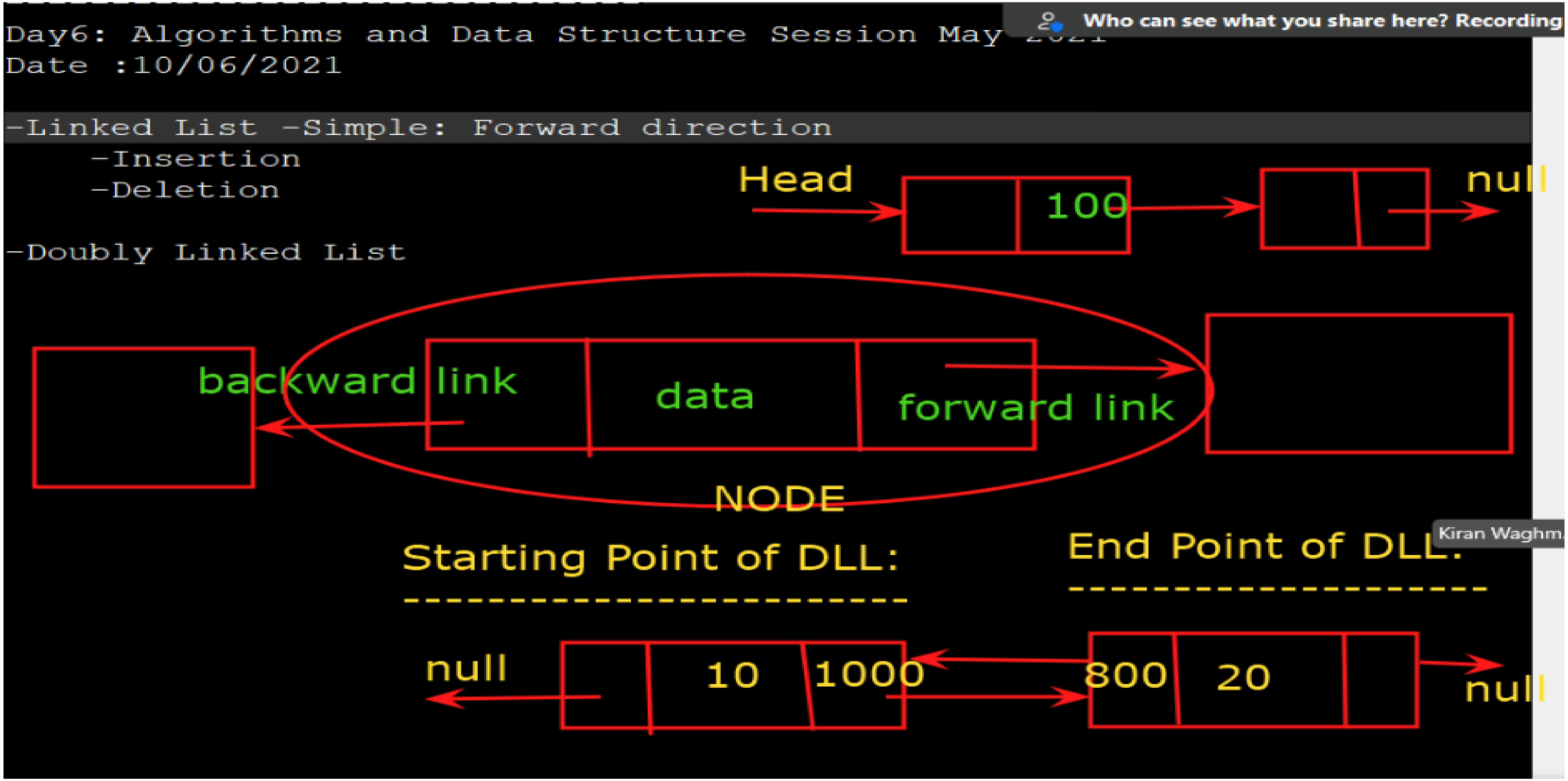
key =33

Node last;
last = head;

Insertion
1.At begining: insert(int key)
2.In between : insertAfter(prev
3.At end :append(key)

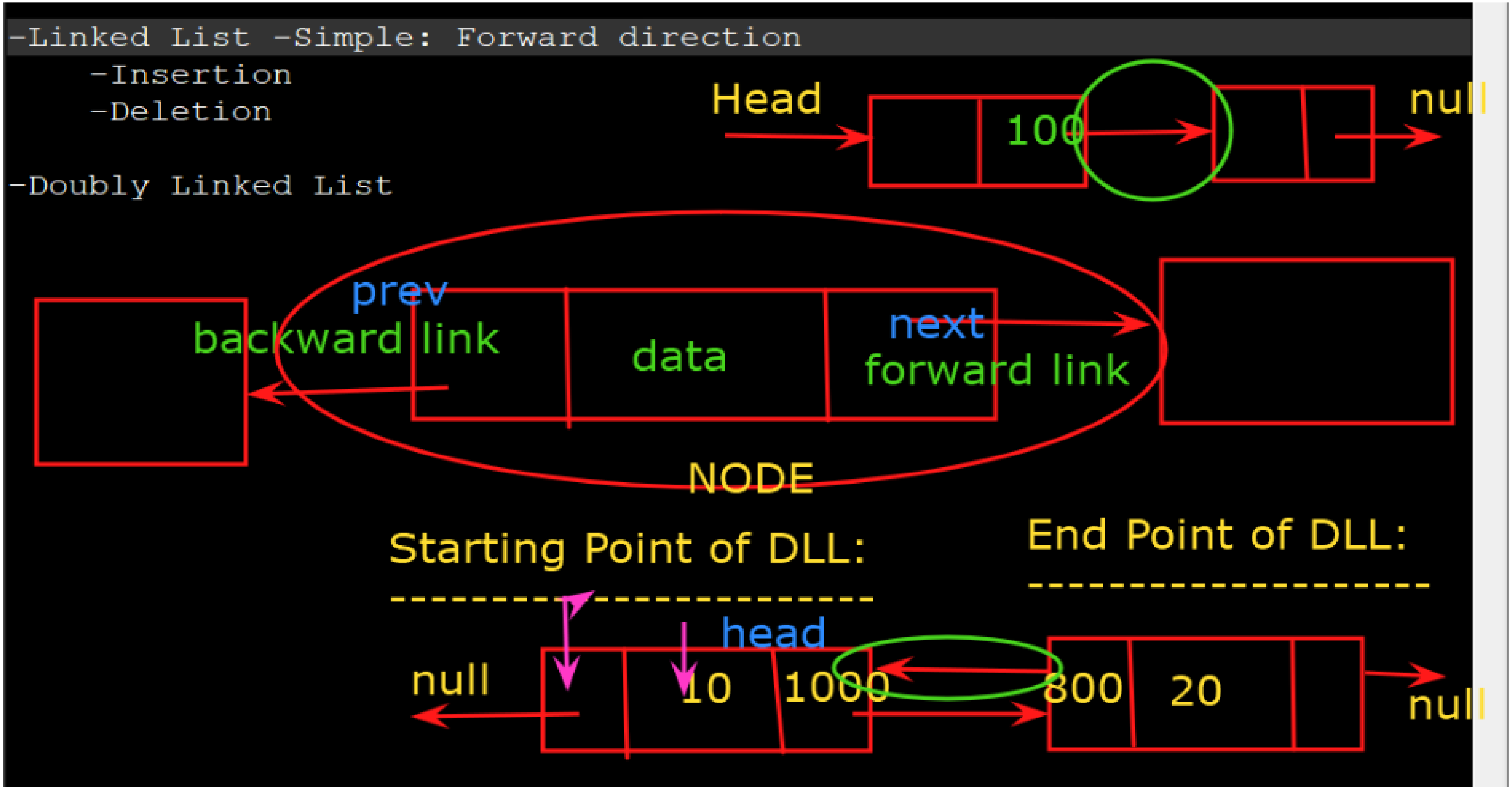
Doubly Linked List

Kiran Waghmare



Who can see what you share here? Recording

```
Day6: Algorithms and Data Structure Session May 2021
Date :10/06/2021

-Linked List -Simple: Forward direction
    -Insertion
    -Deletion

-Doubly Linked List
```



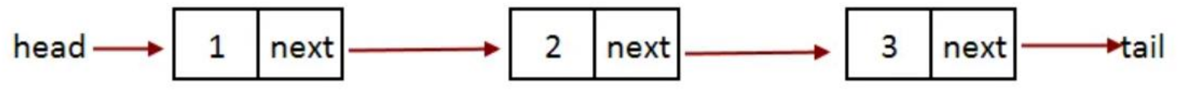
```
-Linked List -Simple: Forward direction
    -Insertion
    -Deletion

-Doubly Linked List
```

Starting Point of DLL:

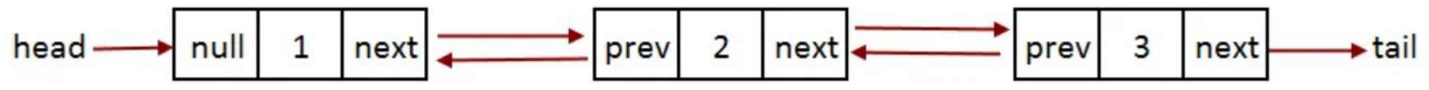
End Point of DLL:

Singly Linked List vs Doubly Linked List

Singly Linked List	Doubly Linked List
Easy Implement	Not easy
Less memory	More Memory
Can traverse only in forward direction	Traverse in both direction, back and froth



Singly Linked List



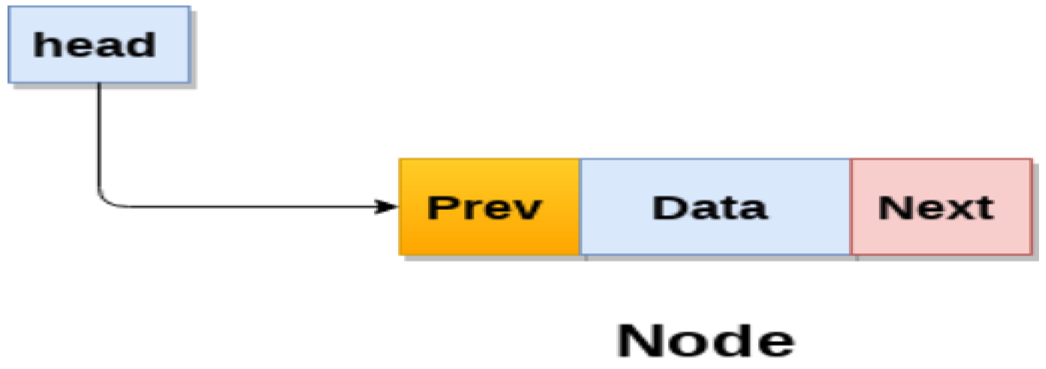
Doubly Linked List

Doubly linked list

- Doubly linked list is a complex type of linked list
  - in which a node contains a pointer to the previous as well as the next node in the sequence.

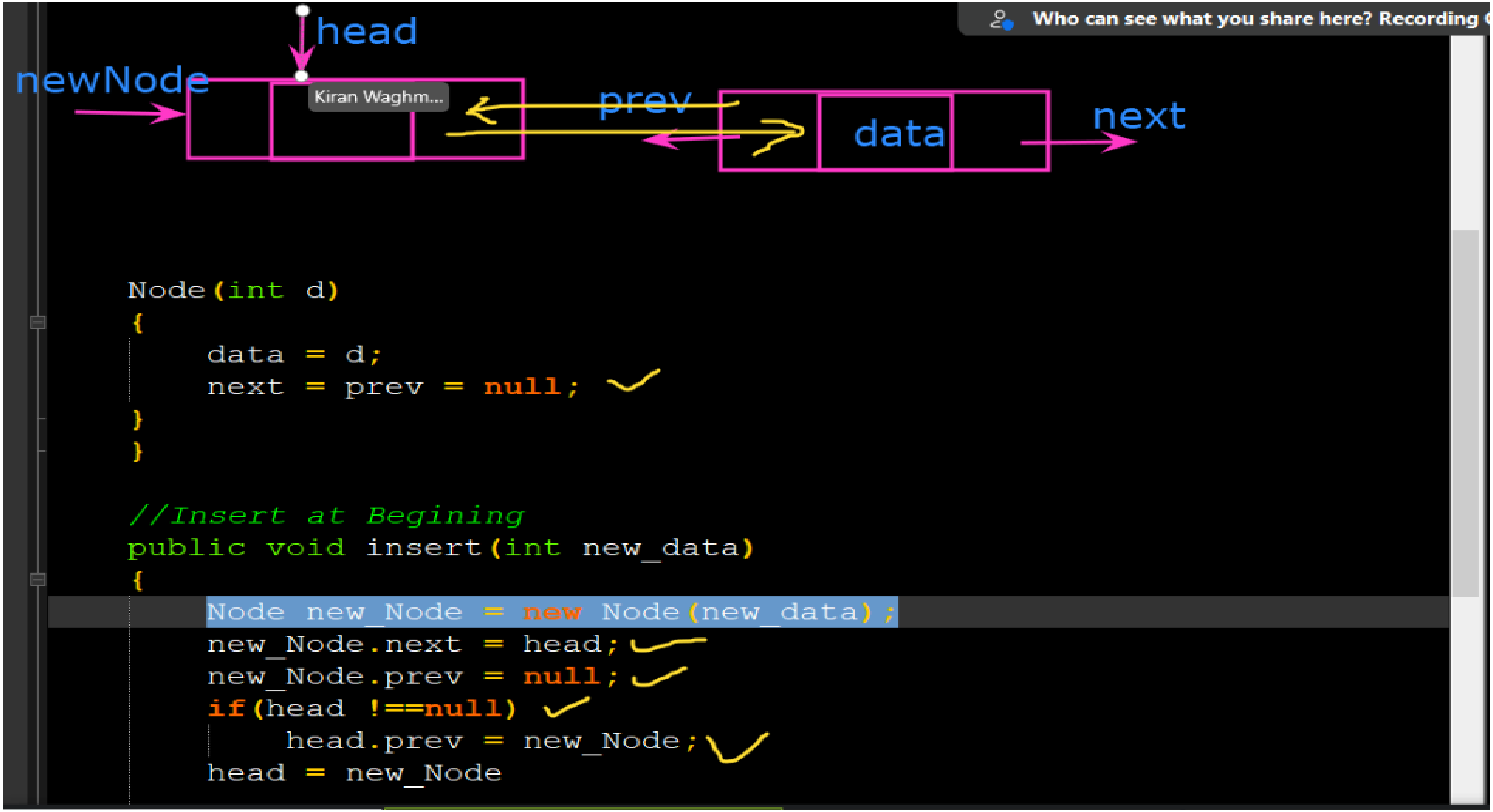
- In a doubly linked list, a node consists of three parts:

1. Data
2. Pointer to the previous node
3. pointer to the next node

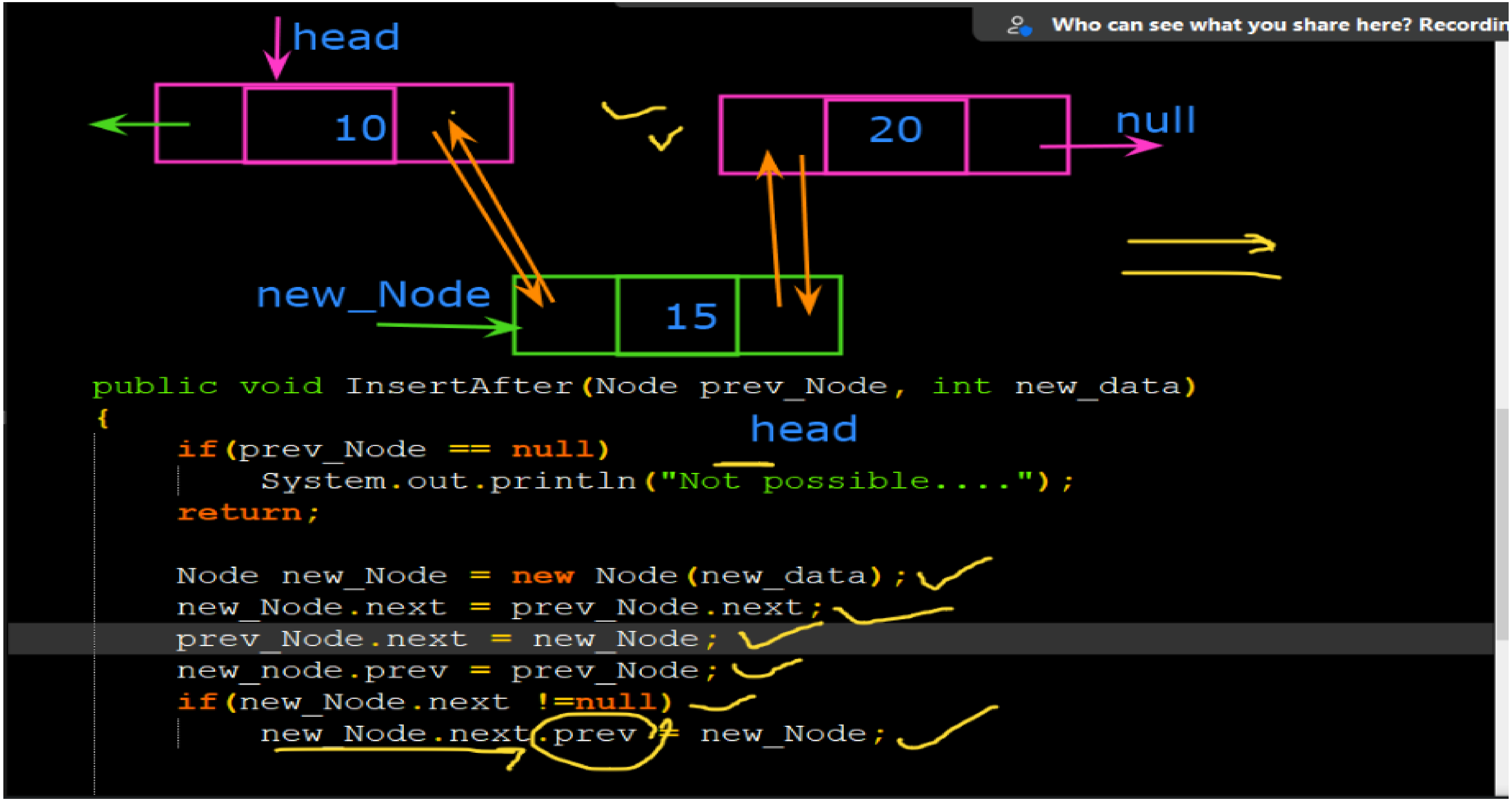


Inserting Nodes in a Doubly-Linked List

- A node can be inserted at any of the following positions in a doubly-linked list:
  - Beginning of the list
  - Between two nodes in the list
  - End of the list



```
Node(int d)
{
    data = d;
    next = prev = null; ✓
}
}

//Insert at Begining
public void insert(int new_data)
{
    Node new_Node = new Node(new_data);
    new_Node.next = head; ✓
    new_Node.prev = null; ✓
    if(head !==null) ✓
        head.prev = new_Node; ✓
    head = new_Node
```



```
public void InsertAfter(Node prev_Node, int new_data)
{
        if(prev_Node == null)
            System.out.println("Not possible....");
        return;

        Node new_Node = new Node(new_data);
        new_Node.next = prev_Node.next;
        prev_Node.next = new_Node;
        new_node.prev = prev_Node;
        if(new_Node.next !=null)
            new_Node.next.prev = new_Node;
```