Linked List ⇒ need

Memory Management ✓

Null ptr Exception

Linked List → Free space

(Continuous / non-continuous)

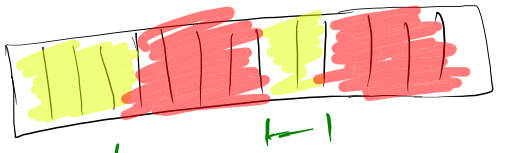
Arrays → direct Access

→ Real

→ Heap

→ Continuous mem allocation

Sec I →



15 bytes

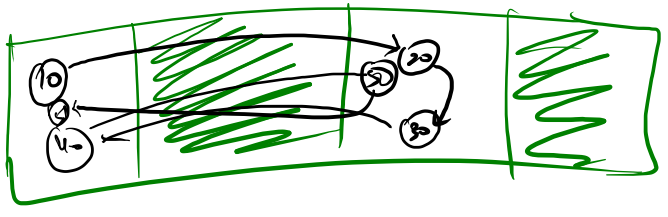
14 bytes

Total F. sp 29 bytes

1 int ⇒ 4 bytes

5 int ⇒ 20 bytes

int arr[] = new int[5] ✗



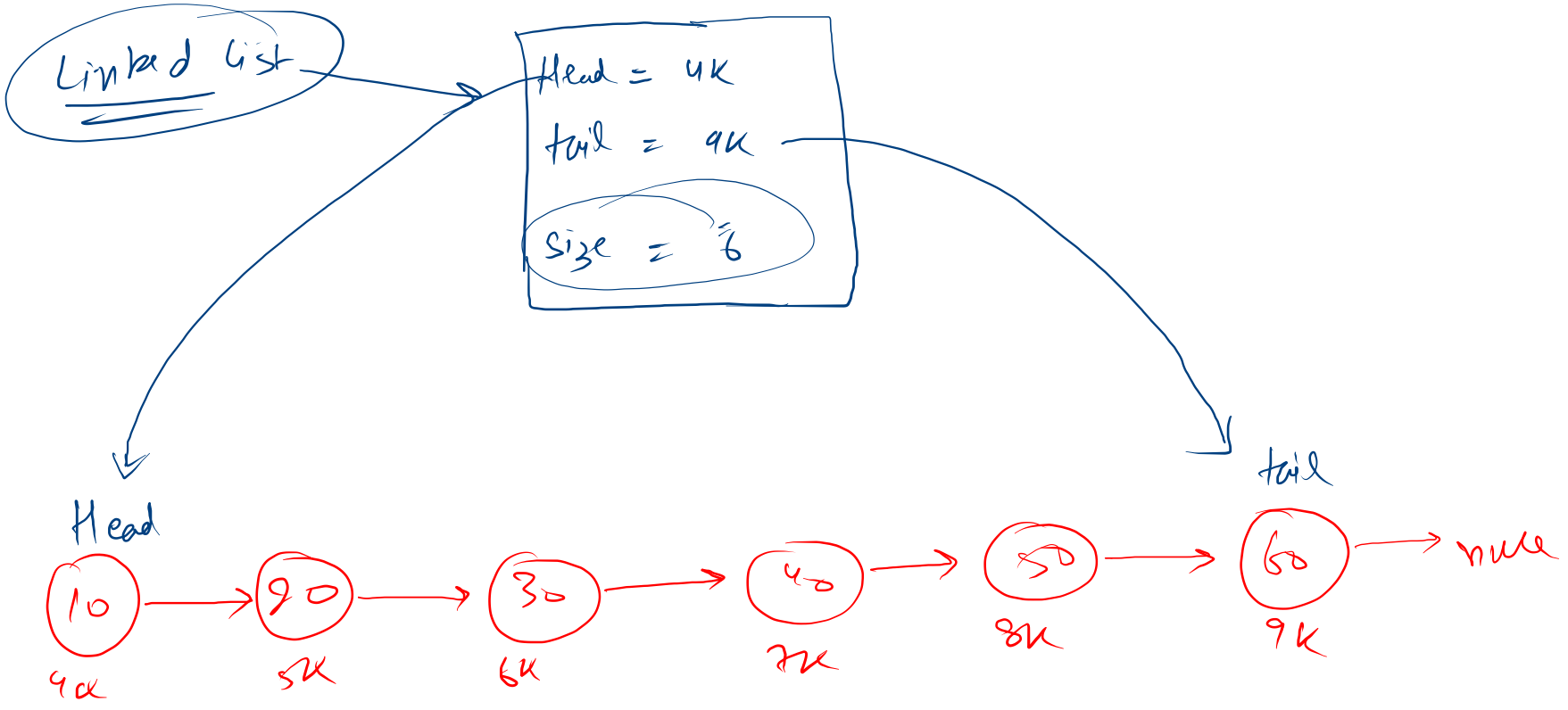
NODE → cell

↳ Data (int / String . . . . .)

↳ next :-

LinkedList → Collection of interconnected nodes + discipl

Overview


Linked List
Head = 4K
tail = 9K
size = 6
Head
tail
10
20
30
40
50
60
null
4K
5K
6K
7K
8K
9K

```java
public class Main {
    public static class Node{
        int data;
        Node next;
        Node(int data,Node next){
            this.data = data;
            this.next = next;
        }
    }

    public static class LinkedList{
      ✓ Node head , tail;
      ✓ int size;
    }

    Run | Debug
    public static void main(String[] args){
        LinkedList ll = new LinkedList();
    }
}
```

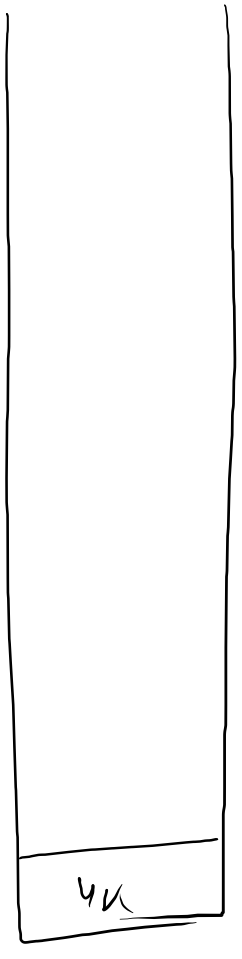
(1) Null Ptr Exception

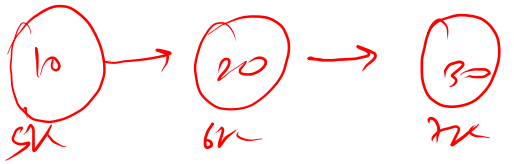
(2) Operations
↳ Summary ✓✓

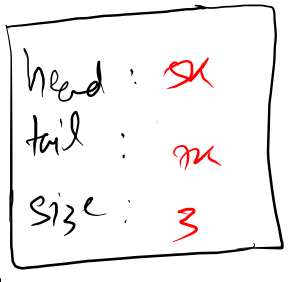
(3) oper/fun/procedure

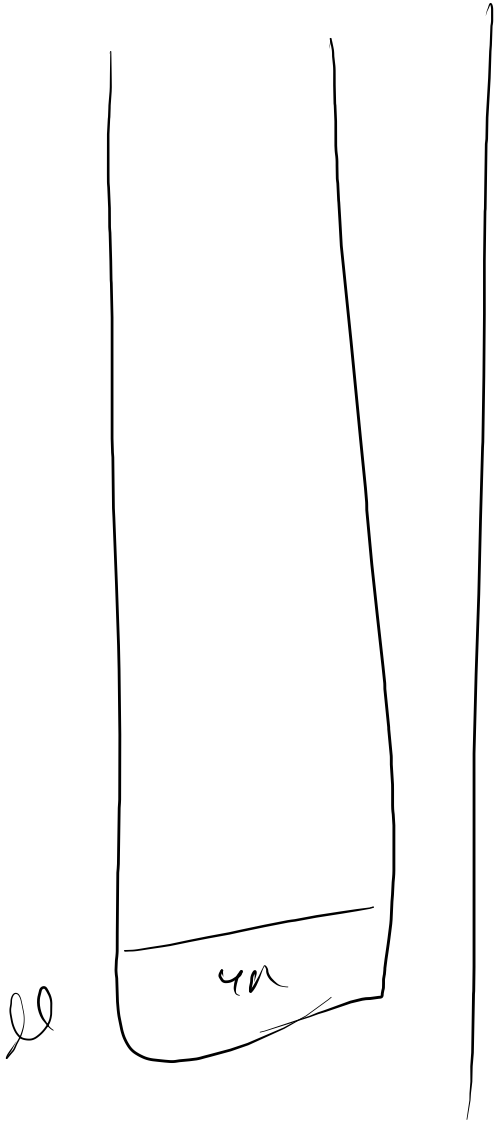
Odd
Even
+ve
-ve
1
2

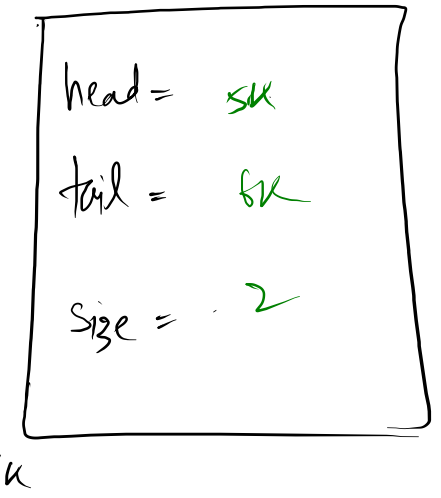
ll



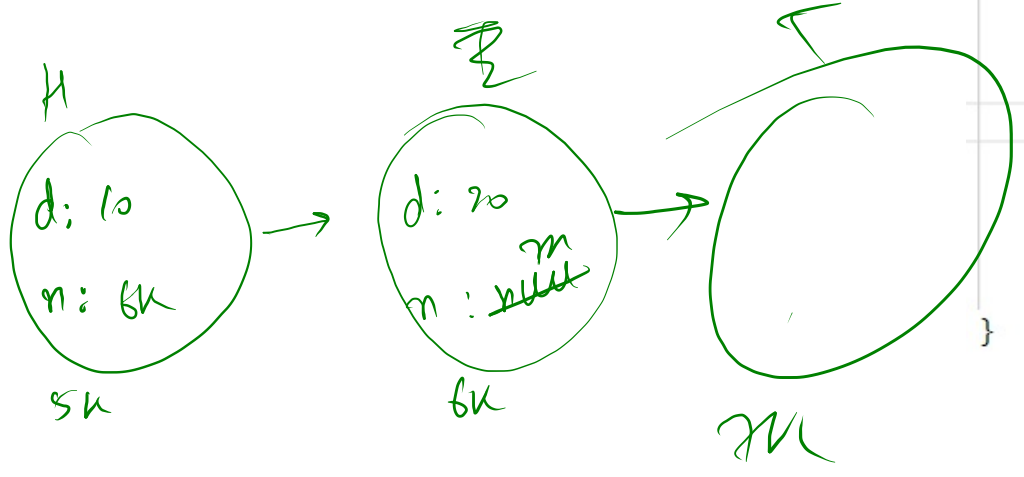








addLast 10 ✓
addLast 20 ✓
addLast 30
addLast 40




```
void addLast(int val) {
    Node node = new Node();
    node.data = val;

    if(this.size == 0){
        this.head = this.tail = node;
    }else{
        this.tail.next = node;
        this.tail = node;
    }
    this.size++;
}
```

Size & Display

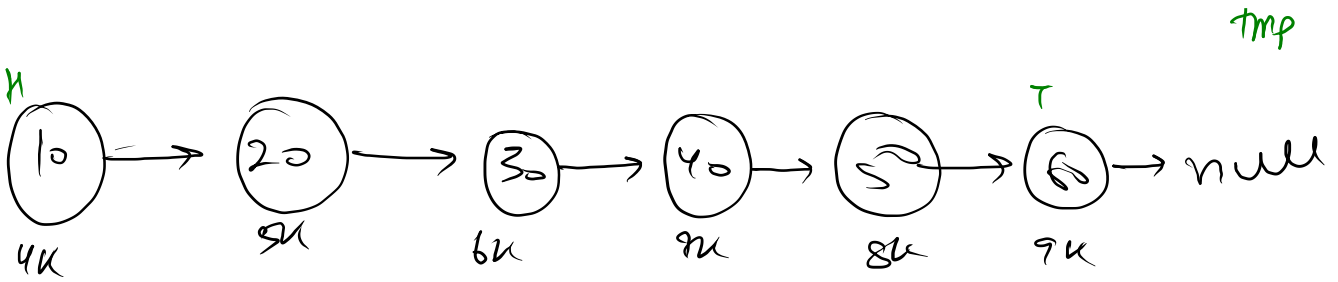
Head = 4k

Tail = 9k

Size = 6

```
public int size(){
    return this.size;
}
```

```
public void display(){
    Node tmp = this.head;

    while(tmp != null){
        System.out.print(tmp.data+" ");
        tmp = tmp.next;
    }

    System.out.println();
}
```



10 20 30 40 50 60

head = null
tail = null
size = 0

rf → 30

rf → 40

rf → List is empty

```java
public void removeFirst() {
  if (this.size == 0) {
    System.out.println("List is empty");
    return;
  } else if (this.size == 1) {
    this.head = this.tail = null;
    this.size = 0;
  } else {
    Node fwd = this.head.next;
    this.head.next = null;
    this.head = fwd;
    this.size--;
  }
}
```

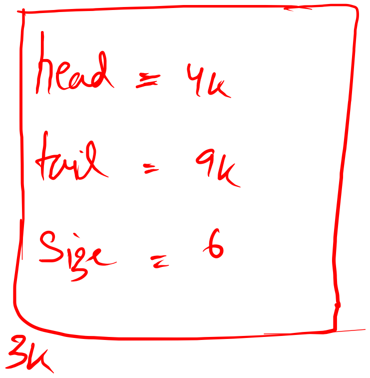
```
public int getFirst(){
    // write your code here
}

public int getLast(){
    // write your code here
}

public int getAt(int idx){
    // write your code here
}
```

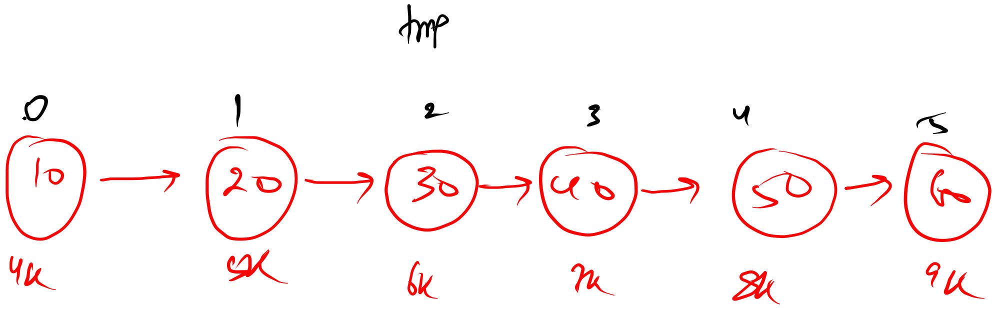
① Size == 0

② idx < 0 || idx >= Size

③

tmp





30

idx =

addFirst

head = 4k

tail = 4k

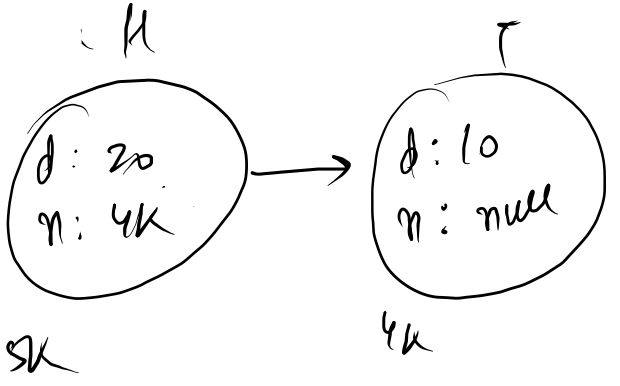
size = 1

3V

addFirst(10);

addFirst(20);

```java
public void addFirst(int val) {
  Node node = new Node();
  node.data = val;

  if (this.size == 0) {
    this.head = this.tail = node;
  } else {
    node.next = this.head;
    this.head = node;
  }
  this.size++;
}
```



```
public void removeLast(){
  // write your code here
}
```

① Size == 0

↳ List is empty

② Size == 1

↳ head = tail = null;

Size = 0;

head = null

tail = null

Size = 0

remove last() ✓

remove last() ✓

remove last() ✓

remove last() ✓

remove last() → List is Empty


```
public void removeLast(){
  if(this.size == 0){
      System.out.println("List is empty");
      return;
  }
  if(this.size == 1){
      this.head = this.tail = null;
      this.size = 0;
      return;
  }

  Node tmp = this.head;
  while(tmp.next != tail){
      tmp = tmp.next;
  }

  tmp.next = null;
  this.tail = tmp;
  this.size--;
}
```

```java
public void removeAt(int idx) {
    // write your code here
}
```

head : 1K
tail : 12K
size : 11

① idx<0 || idx>=size

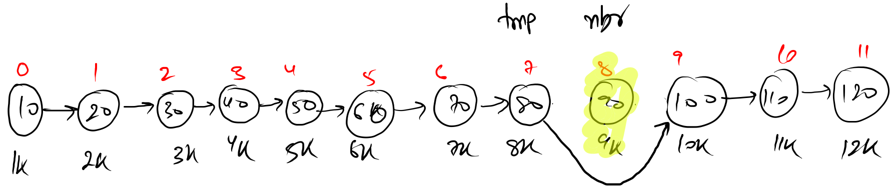
② idx == 0
↳ removeFirst();

③ idx == size-1
↳ removeLast();

idx = 8 7 6 5 4 3 2 1

idx != 0 | (idx-1) != 0 ✓

removeAt(13) → Invalid argument
removeAt(0) → rF
removeAt(11) → rL
removeAt(8)




```java
public void removeAt(int idx) {
    if(this.size() == 0){
        System.out.println("List is empty");
        return;
    }
    if(idx < 0 || idx >= size){
        System.out.println("Invalid arguments");
        return;
    }
    if(idx == 0){
        removeFirst();
    }else if(idx == size-1){
        removeLast();
    }else{
        Node tmp = this.head;
        while(idx-1 != 0){
            idx--;
            tmp = tmp.next;
        }

        Node nbr = tmp.next;
        tmp.next = nbr.next;
        nbr.next = null;

        this.size--;
    }
}
```

addAt(10, -1) ⇒ Invalid arguments (idx < 0 || idx > size)

addAt(0, -1) ⇒ add first (idx == 0)

addAt(8, -1) ⇒ add last (idx == size)

addAt(4, -1)

idx == 0 || idx == size

(idx - 1 != 0)

```
node.next = tmp.next;
tmp.next = node
sizeAt;
```

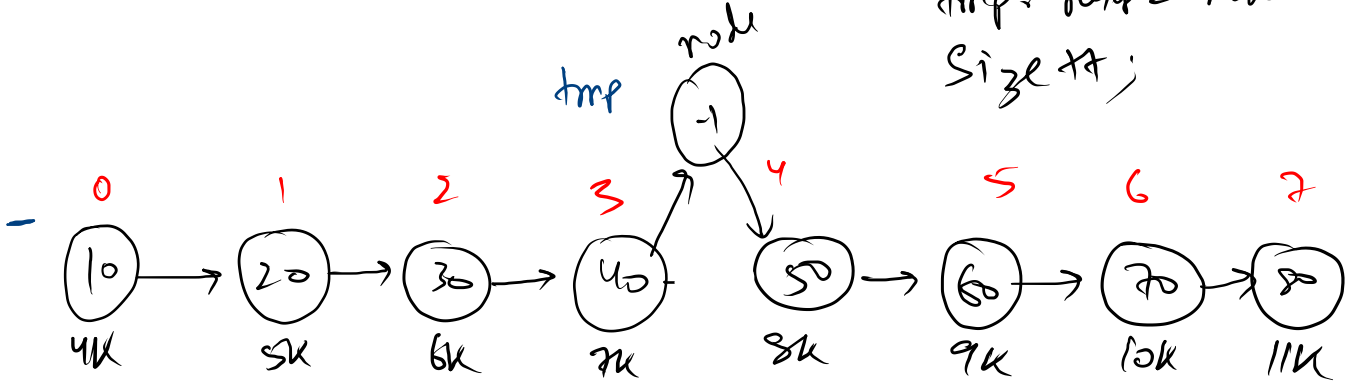
addAt

head = 4K

tail = 11K

size = 8

H.W. ✓



Sunday → Class x

→ Test ≡ 10:00 am - 12:00 pm

Stacks & Queues ≡≡