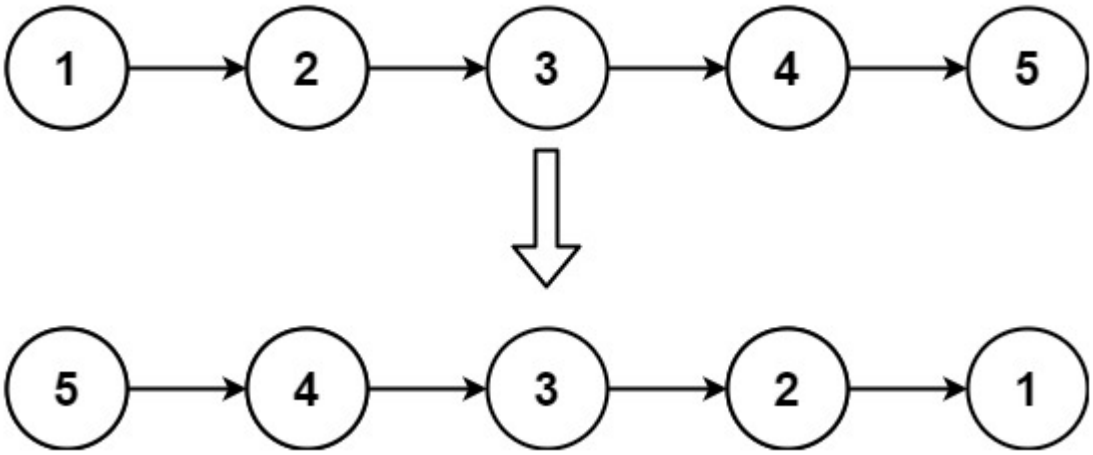
LinkedList Immersion

HARD Q:-1

Reverse Linked List

Given the head of a singly linked list, reverse the list, and return the reversed list.

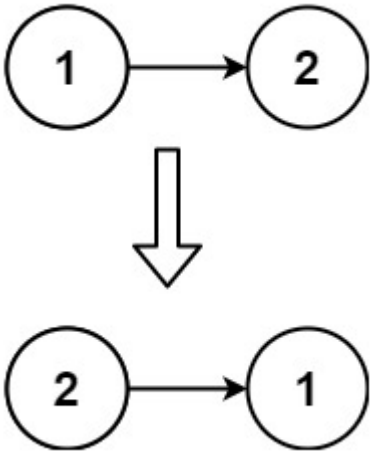
Example 1:



Input: head = [1,2,3,4,5]

Output: [5,4,3,2,1]

Example 2:



Input: head = [1,2]

Output: [2,1]

Example 3:

Input: head = []

Output: []

Constraints:

The number of nodes in the list is the range [0, 5000].

-5000 <= Node.val <= 5000

Solution:-

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next
= next; }
 * }
 */
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode currnode = head;
        ListNode temp =null;
        ListNode pre =null;
        while(currnode!=null)
        {
            temp=currnode.next;
            currnode.next=pre;
            pre=currnode;
            currnode=temp;
        }
        return pre;
    }
}
```

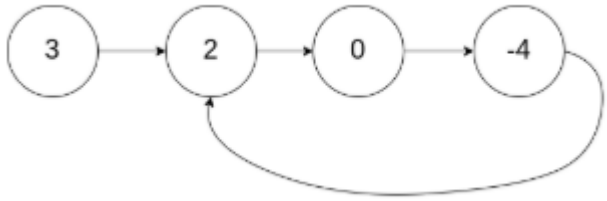
Q:-2 MEDIUM

Given head, the head of a linked list, determine if the linked list has a cycle in it.

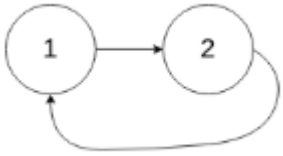
There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. Note that pos is not passed as a parameter.

Return true if there is a cycle in the linked list. Otherwise, return false.

Example 1:



Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).
Example 2:



Input: head = [1,2], pos = 0
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.
Example 3:

Input: head = [1], pos = -1
Output: false
Explanation: There is no cycle in the linked list.
Constraints:
The number of the nodes in the list is in the range [0, 104].
-105 <= Node.val <= 105
pos is -1 or a valid index in the linked-list.

Solution:-

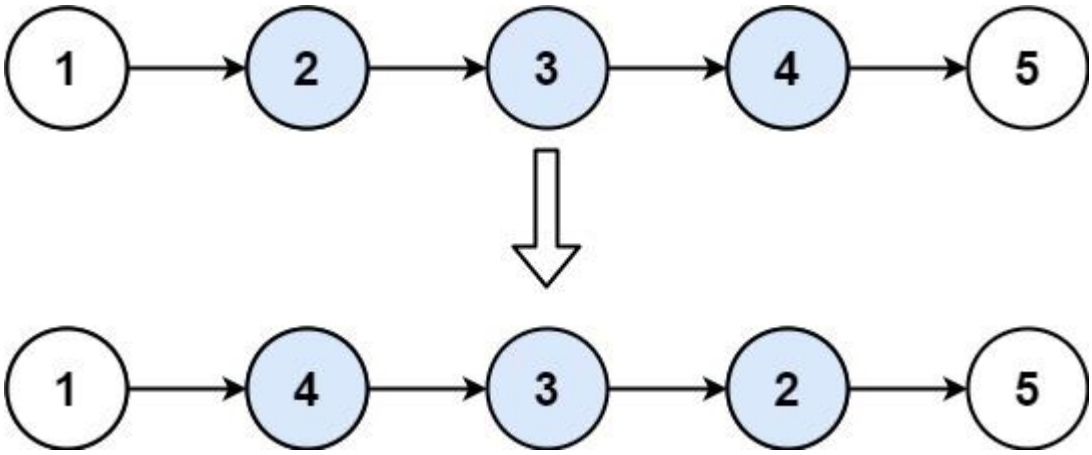
```
/**
 * Definition for singly-linked list.
 * class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
```

```
 */
public class Solution {
    public boolean hasCycle(ListNode head) {
        ListNode slow=head;
        ListNode fast=head;
        while(fast!=null && fast.next!=null)
        {
            slow=slow.next;
            fast=fast.next.next;
            if(slow==fast)
            {
                return true;
            }
        }
        return false;
    }
}
```

HARD Q:-3 Reverse Linked List II

Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right, and return *the reversed list*.

Example 1:



Input: head = [1,2,3,4,5], left = 2, right = 4
Output: [1,4,3,2,5]
Example 2:
Input: head = [5], left = 1, right = 1
Output: [5]
Constraints:

- The number of nodes in the list is n.
- 1 <= n <= 500
- -500 <= Node.val <= 500

- 1 <= left <= right <= n

Solution:-

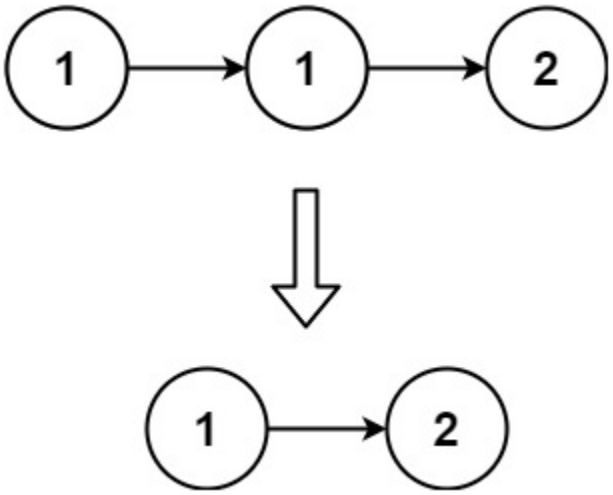
```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next
= next; }
 * }
 */
class Solution {
    public ListNode reverseBetween(ListNode head, int left, int
right) {
        ListNode dummy=new ListNode(0);
        dummy.next=head;
        ListNode preleft=dummy;
        ListNode currnode=head;
        for(int i=0;i<left-1;i++)
        {
            preleft=preleft.next;
            currnode=currnode.next;
        }
        ListNode subnode=currnode;
        ListNode prenode=null;
        for(int i=0;i<=right-left;i++)
        {
            ListNode temp=currnode.next;
            currnode.next=prenode;
            prenode=currnode;
            currnode=temp;
        }
        preleft.next=prenode;
        subnode.next=currnode;
        return dummy.next;
    }
}
```

Q:-4 MEDIUM

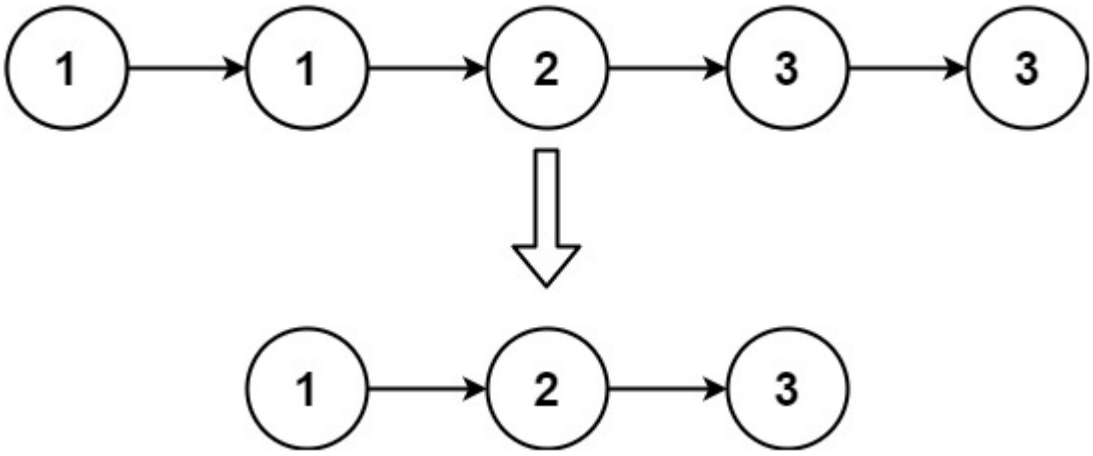
Remove Duplicates from Sorted List.

Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.

Example 1:



Input: head = [1,1,2]
Output: [1,2]
Example 2:



Input: head = [1,1,2,3,3]
Output: [1,2,3]

Constraints:
The number of nodes in the list is in the range [0, 300].
-100 <= Node.val <= 100
The list is guaranteed to be sorted in ascending order.
Solution:-

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
```

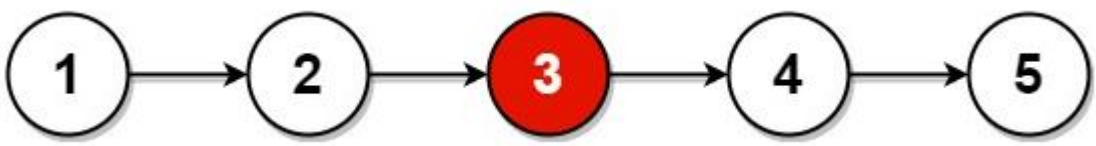
```
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next
= next; }
 * }
 */
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode temp=head;
        if(temp==null)
        {
            return null;
        }
        while(temp.next!=null)
        {
            if(temp.val==temp.next.val)
            {
                temp.next=temp.next.next;
            }
            else
            {
                temp=temp.next;
            }
        }
        return head;
    }
}
```

MEDIUM Q:-5 Middle of the Linked List.

Given the head of a singly linked list, return the middle node of the linked list.

If there are two middle nodes, return the second middle node.

Example 1:-

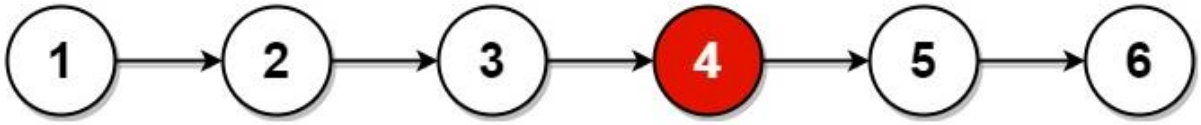


Input: head = [1,2,3,4,5]

Output: [3,4,5]

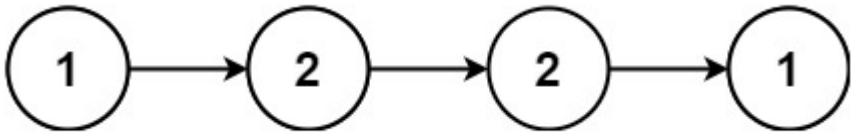
Explanation: The middle node of the list is node 3.

Example 2:

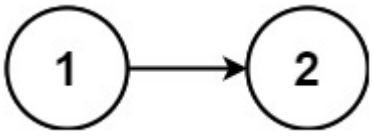


Input: head = [1,2,3,4,5,6]
Output: [4,5,6]
Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.
Constraints:
The number of nodes in the list is in the range [1, 100].
1 <= Node.val <= 100

Q:-MEDIUM HARD Palindrome Linked List
Given the head of a singly linked list, return true if it is a palindrome or false otherwise.
Example 1:



Input: head = [1,2,2,1]
Output: true
Example 2:



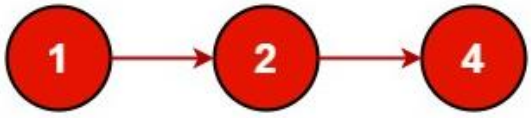
Input: head = [1,2]
Output: false
Constraints:
The number of nodes in the list is in the range [1, 105].
0 <= Node.val <= 9
Solution:-

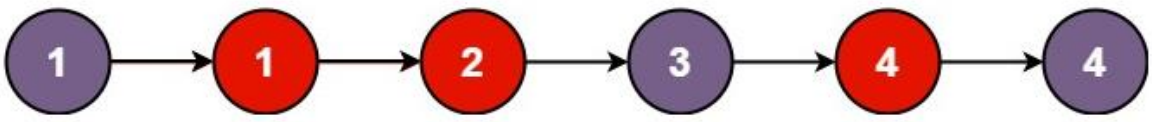
```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next
= next; }
```

```
 * }
 */
class Solution {
    public ListNode middleNode(ListNode head) {
        ListNode slow=head;
        ListNode fast=head;
        while(fast!=null && fast.next!=null)
        {
            slow=slow.next;
            fast=fast.next.next;
        }
        return slow;
    }
}
```

HARD Q:-6 Merge Two Sorted Lists.

You are given the heads of two sorted linked lists list1 and list2.
Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists.
Return the head of the merged linked list.
Example 1:





Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]
Example 2:
Input: list1 = [], list2 = []
Output: []
Example 3:
Input: list1 = [], list2 = [0]
Output: [0]
Constraints:

The number of nodes in both lists is in the range [0, 50].
-100 <= Node.val <= 100
Both list1 and list2 are sorted in non-decreasing order.
Solution:-

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next
= next; }
 * }
 */
class Solution {
  public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode move=null;
        ListNode head=null;
        if (list1==null)
        {
            return list2;
        }
        if (list2==null)
        {
            return list1;
        }
        while (list1!=null && list2!=null)
        {
            if (head==null)
            {
                if (list1.val<list2.val)
                {
                    move=head=list1;
                    list1=list1.next;
                }
                else
                {
                    move=head=list2;
                    list2=list2.next;
                }
             continue;
            }
```

```
            if (list1.val<list2.val)
               {
                   move.next=list1;
                   move=list1;
                   list1=list1.next;
               }
               else
               {
                   move.next=list2;
                   move=list2;
                   list2=list2.next;
               }
         }
      if (list1!=null)
      {
          move.next=list1;
      }
      else
      {
          move.next=list2;
      }
      return head;
    }
}
```

MEDIUM Example Single Linked List:-7 In this example we insert the records in ascending order.

```
package com.vikas.ds;
public class SingleLinkedListDemo1
{
Node head;
int size;
class Node{
int data;
Node next;
Node(int data){
this.data = data;
this.next = null;
```

```
size++;
}
Node(int data,Node temp){
this.data = data;
this.next = temp;
size++;
}
}
int getSize(){
return this.size;
}
void printList(){
if(head==null){
System.out.println("list is empty");
}
else
{
Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("null");
}
}
void sortedInsertAsc(int data){
Node newNode = new Node(data);
Node currNode = head;
if(currNode==null||currNode.data>data){
newNode.next = head;
head = newNode;
return;
```

```
}
while(currNode.next!=null && currNode.next.data<data){
currNode = currNode.next;
}
newNode.next = currNode.next;
currNode.next = newNode;
}
public static void main(String[] args)
{
SingleLinkedListDemo1 slld=new SingleLinkedListDemo1();
slld.sortedInsertAsc(4444);
slld.sortedInsertAsc(5555);
slld.sortedInsertAsc(2222);
slld.sortedInsertAsc(6666);
slld.sortedInsertAsc(3333);
slld.sortedInsertAsc(1111);
slld.printList();
System.out.println("Size= "+slld.getSize());
}
}
```

Result:-

```
1111 => 2222 => 3333 => 4444 => 5555 => 6666 => null
Size= 6
```

MEDIUM Example Single Linked List:-8 In this example we insert the records in descending order.

```
package com.vikas.ds;
public class SingleLinkedListDemo1
{
Node head;
int size;
class Node{
int data;
```

```
Node next;
Node(int data){
this.data = data;
this.next = null;
size++;
}
Node(int data,Node temp){
this.data = data;
this.next = temp;
size++;
}
}
int getSize(){
return this.size;
}
void printList(){
if(head==null){
System.out.println("list is empty");
}
else
{
Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("null");
}
}
void sortedInsertDesc(int data){
Node newNode = new Node(data);
Node currNode = head;
```

```
if(currNode==null||currNode.data<data){
newNode.next = head;
head = newNode;
return;
}
while(currNode.next!=null && currNode.next.data>data){
currNode = currNode.next;
}
newNode.next = currNode.next;
currNode.next = newNode;
}

public static void main(String[] args)
{
SingleLinkedListDemo1 slld=new SingleLinkedListDemo1();
slld.sortedInsertDesc(4444);
slld.sortedInsertDesc(5555);
slld.sortedInsertDesc(2222);
slld.sortedInsertDesc(6666);
slld.sortedInsertDesc(3333);
slld.sortedInsertDesc(1111);
slld.printList();
System.out.println("Size= "+slld.getSize());
}
}
```

Result:-

```
6666 => 5555 => 4444 => 3333 => 2222 => 1111 => null
Size= 6
```

EASY Example Single Linked List:-9 In this example we performed add element first, traversed list(print list), and size of the list.

```
package com.vikas.ds;
public class SingleLinkedListDemo1
{
```

```
Node head;
int size;
class Node{
int data;
Node next;
Node(int data){
this.data = data;
this.next = null;
size++;
}
Node(int data,Node temp){
this.data = data;
this.next = temp;
size++;
}
}
int getSize(){
return this.size;
}
void printList(){
if(head==null){
System.out.println("list is empty");
}
else
{

Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("null");
```

```
}
}
void addFirst(int data){
Node newNode = new Node(data);
if(head==null)
{
head = newNode;
return;
}
else
{
newNode.next = head;
head = newNode;
}

}
public static void main(String[] args)
{
SingleLinkedListDemo1 slld=new SingleLinkedListDemo1();
slld.addFirst(444);
//slld.addFirst(333);
//slld.addFirst(222);
//slld.addFirst(111);
slld.printList();
System.out.println("Size= "+slld.getSize());
}
}
```

Result:- 111 => 222 => 333 => 444 => null
Size= 4

EASY Example Single Linked List:-10 In this example we performed add element at last, traversed list(print list), and size of the list.

```
package com.vikas.ds;
public class SingleLinkedListDemo1
{
Node head;
int size;
class Node{
int data;
Node next;
Node(int data){
this.data = data;
this.next = null;
size++;
}
Node(int data,Node temp){
this.data = data;
this.next = temp;
size++;
}
}
int getSize(){
return this.size;
}
void printList(){
if(head==null){
System.out.println("list is empty");
}
else
{
Node currNode = head;
while(currNode!=null)
{
```

```
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("null");
}
}
void addLast(int data)
{
Node newNode = new Node(data);
if(head==null)
{
head = newNode;
return;
}
else
{
Node currNode = head;
while(currNode.next!=null)
{
currNode = currNode.next;
}
currNode.next = newNode;
}
}
public static void main(String[] args)
{
SingleLinkedListDemo1 slld=new SingleLinkedListDemo1();
slld.addLast(121);
slld.addLast(122);
slld.addLast(125);
slld.addLast(126);
slld.printList();
```

```
System.out.println("Size= "+slld.getSize());
}
}
```

Result:-

```
123 => 124 => 125 => 126 => null
Size= 4
```

EASY Example Single Linked List:-11 In this example we performed Inserting the data at a particular position.

```
package com.vikas.ds;
public class SingleLinkedListDemo1
{
Node head;
int size;
class Node{
int data;
Node next;
Node(int data){
this.data = data;
this.next = null;
size++;
}
Node(int data,Node temp){
this.data = data;
this.next = temp;
size++;
}
}
int getSize(){
return this.size;
}
void printList(){
if(head==null){
```

```
System.out.println("list is empty");
}
else
{
Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("null");
}
}
void addLast(int data)
{
Node newNode = new Node(data);
if(head==null)
{
head = newNode;
return;
}
else
{
Node currNode = head;
while(currNode.next!=null)
{
currNode = currNode.next;
}
currNode.next = newNode;
}
}
void addPos(int data,int pos){
int i=0;
```

```
Node newNode = new Node(data);
if(head==null)
{
head = newNode;
return;
}
else
{
if(pos!=0){
Node currNode = head;
Node prevNode = null;
while(currNode.next!=null && i<pos)
{
prevNode = currNode;
currNode = currNode.next;
i++;
}
prevNode.next = newNode;
newNode.next = currNode;
}
else{
newNode.next = head;
head = newNode;
}
}
}
public static void main(String[] args)
{
SingleLinkedListDemo1 slld=new SingleLinkedListDemo1();
slld.addLast(121);
slld.addLast(122);
slld.addLast(125);
slld.addLast(126);
```

```
slld.printList();
System.out.println("Size= "+slld.getSize());
slld.addPos(123,2);
slld.printList();
System.out.println("Size= "+slld.getSize());
}
}
```

Result:-

```
121 => 122 => 125 => 126 => null
Size= 4
121 => 122 => 123 => 125 => 126 => null
Size= 5
```

EASY Example Single Linked List:-12 In this example we performed delete operations like:-

Deleting from first element.

Deleting from the last element.

Delete from particular position.

Deleting Element first match.

Deleting Elements all match.

```
package com.vikas.ds;
public class SingleLinkedListDemo1
{
Node head;
int size;
class Node{
int data;
Node next;
Node(int data){
this.data = data;
this.next = null;
size++;
}
```

```
Node(int data,Node temp){
this.data = data;
this.next = temp;
size++;
}
}
int getSize(){
return this.size;
}
void printList(){
if(head==null){
System.out.println("list is empty");
}
else
{
Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("null");
}
}
void addLast(int data)
{
Node newNode = new Node(data);
if(head==null)
{
head = newNode;
return;
}
else
```

```
{
Node currNode = head;
while(currNode.next!=null)
{
currNode = currNode.next;
}
currNode.next = newNode;
}
}
void deleteFirst(){
if(head==null)
{
System.out.println("List is empty");
return;
}
else
{
size--;
head=head.next;
}
}
void deleteLast()
{
if(head==null){
System.out.println("list is empty");
return;
}
else if(head.next==null)
{
head=null;
return;
}
else
```

```
{
size--;
Node temp1=head,temp2=head.next;
while(temp2.next!=null)
{
temp2 = temp2.next;
temp1 = temp1.next;
}
temp1.next = null;
}
}
//by using this method we delete first match element.
void deleteElement(int data)
{
Node temp = head;
if(temp==null){
System.out.println("empty");
return;
}
else if(temp.data == data){
head = head.next;
size--;
return;
}
while(temp.next!=null)
{
if(temp.next.data == data){
temp.next = temp.next.next;
size--;
return;
}
else
{
```

```
temp = temp.next;
}
}
}
//by using this method we all mathch element.
void deleteElements(int data)
{
Node temp = head;
if(temp==null)
{
System.out.println("empty");
return;
}
else if(temp.data == data)
{
head = head.next;
size--;
}
else
{
while(temp.next!=null){
if(temp.next.data == data)
{
temp.next = temp.next.next;
size--;
}
else if(temp.next!=null)
{
temp = temp.next;
}}}}
//delete element at particular position
void deleteElementAtPos(int pos){
Node temp = head;
```

```
int i=0;
if(temp==null){
System.out.println("empty");
return;
}
else if(pos==0){
head = head.next;
size--;
return;
}
else
{
while(temp.next!=null && i<pos){
if(i==pos-1)
{
temp.next = temp.next.next;
size--;
return;
}
i++;
temp = temp.next;
}
}
}
public static void main(String[] args)
{
SingleLinkedListDemo1 slld=new SingleLinkedListDemo1();
slld.addLast(121);
slld.addLast(122);
slld.addLast(123);
slld.addLast(125);
slld.addLast(125);
slld.addLast(125);
```

```
slld.addLast(126);
slld.printList();
System.out.println("Size= "+slld.getSize());
slld.deleteFirst();
slld.printList();
System.out.println("Size= "+slld.getSize());
slld.deleteLast();
slld.printList();
System.out.println("Size= "+slld.getSize());
slld.deleteElement(125);
slld.printList();
System.out.println("Size= "+slld.getSize());
slld.deleteElements(125);
slld.printList();
System.out.println("Size= "+slld.getSize());
slld.deleteElementAtPos(1);
slld.printList();
System.out.println("Size= "+slld.getSize());
}
}
```

Result:-

```
121 => 122 => 123 => 125 => 125 => 125 => 126 => null
Size= 7
122 => 123 => 125 => 125 => 125 => 126 => null
Size= 6
122 => 123 => 125 => 125 => 125 => null
Size= 5
122 => 123 => 125 => 125 => null
Size= 4
122 => 123 => null
Size= 2
122 => null
Size= 1
```