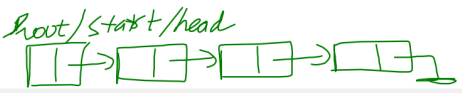
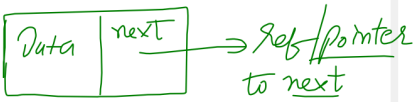


Linked List:linear,dynamic,can operate from any side
Collection of nodes arranged in a sequential manner.

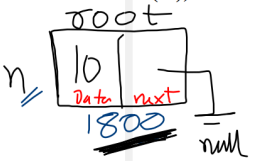
Node:???????




```
class Node
{
 int data;
 Node next;//self-ref pointer/reference
 Node(int data)
    {
        this.data=data;
        this.next=null;// not ref anyone
    }
}
```

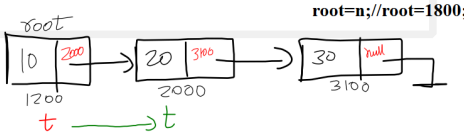
```
class Node
{
 int data;
 Node next;//self-ref pointer/reference
 Node(int data)
    {
        this.data=data;
        this.next=null;// not ref anyone
    }
}
```

Node n=new Node(10);



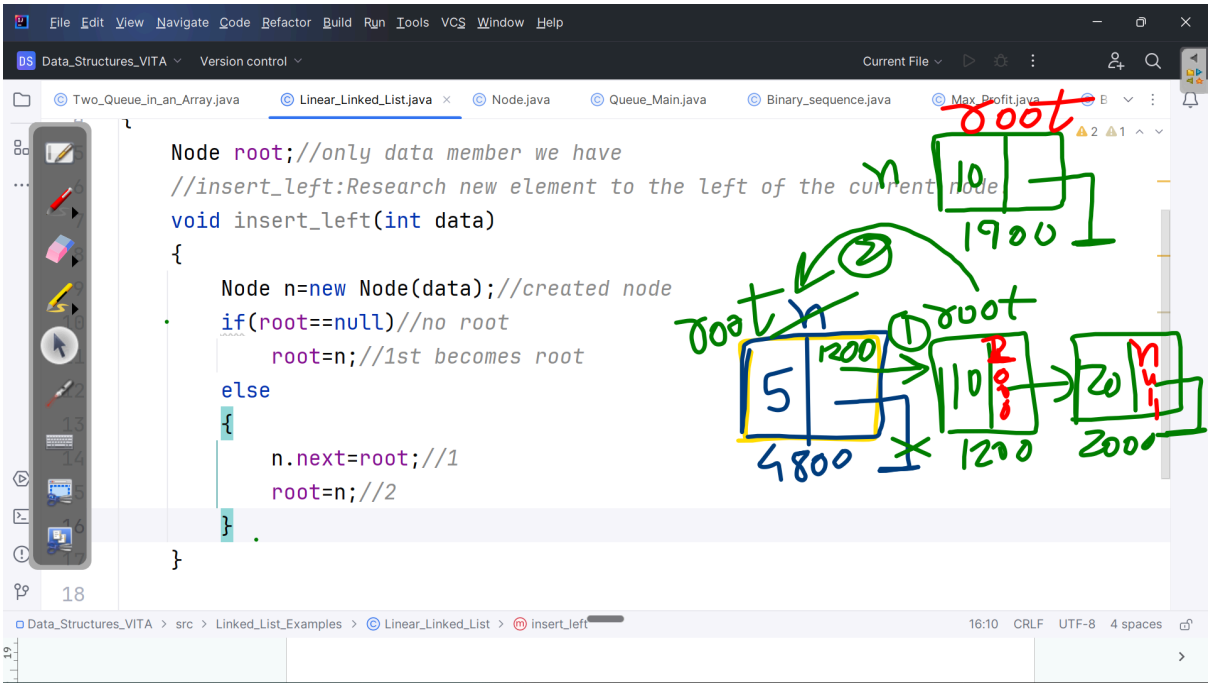
Node root;//empty root

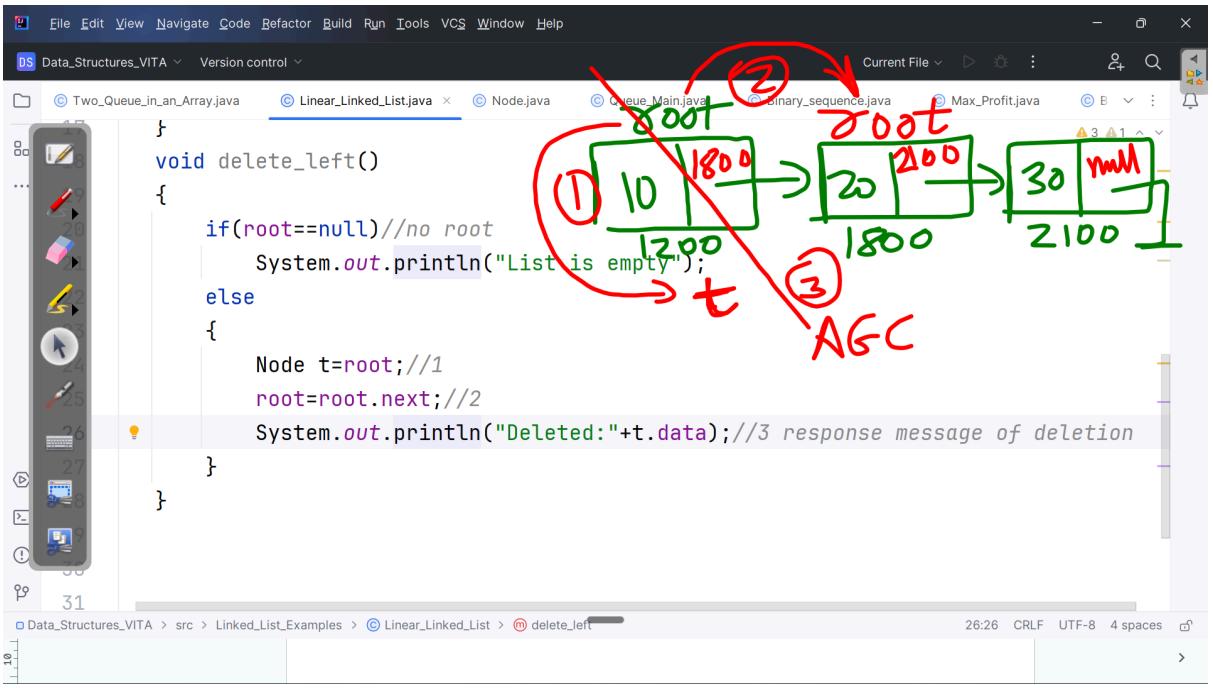
root=n;//root=1800;

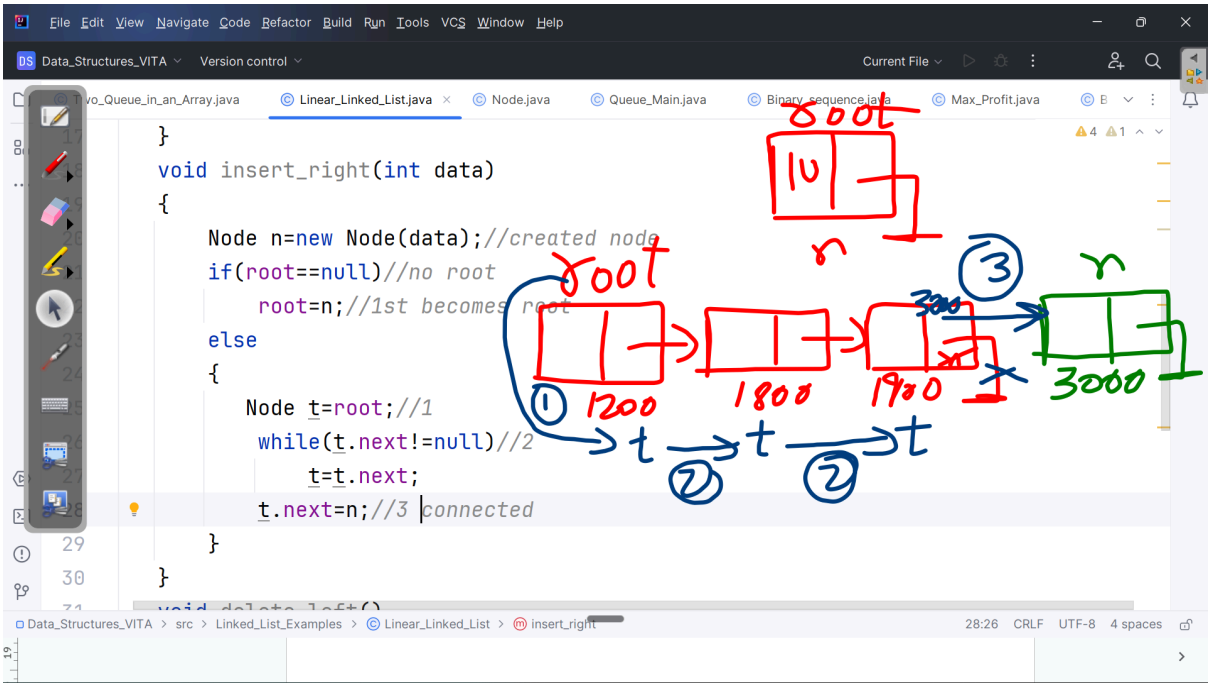


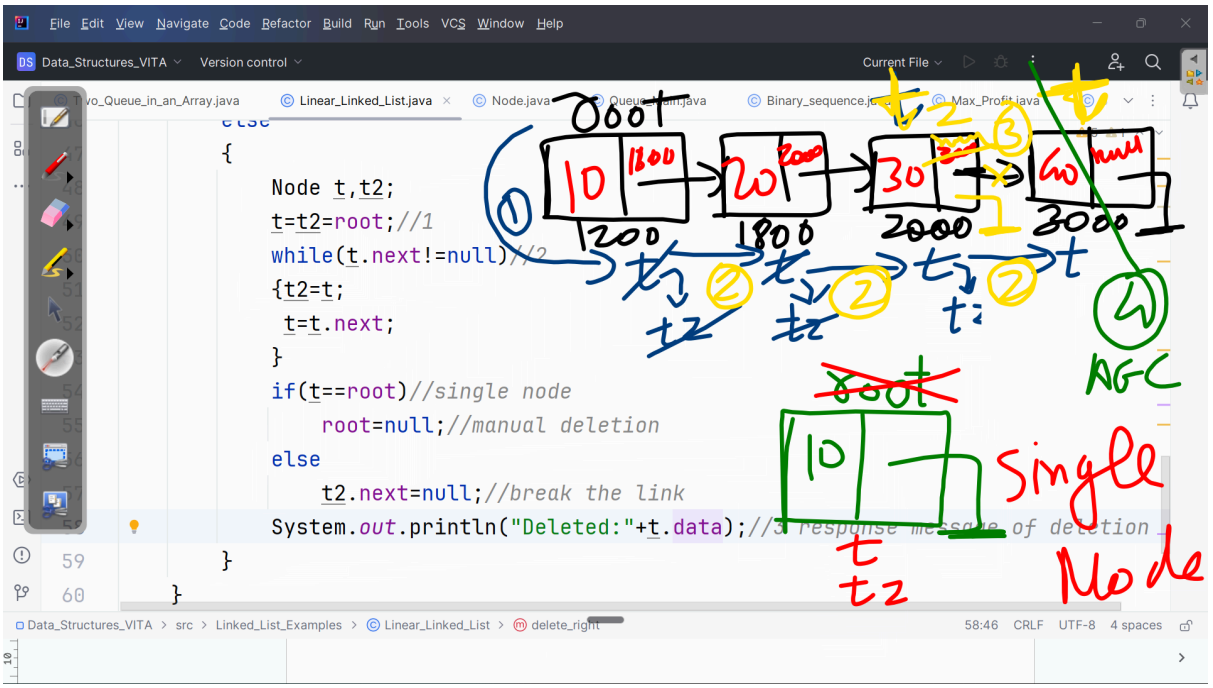
Node t=root; t=1200
t=t.next; t=2000(move to next)

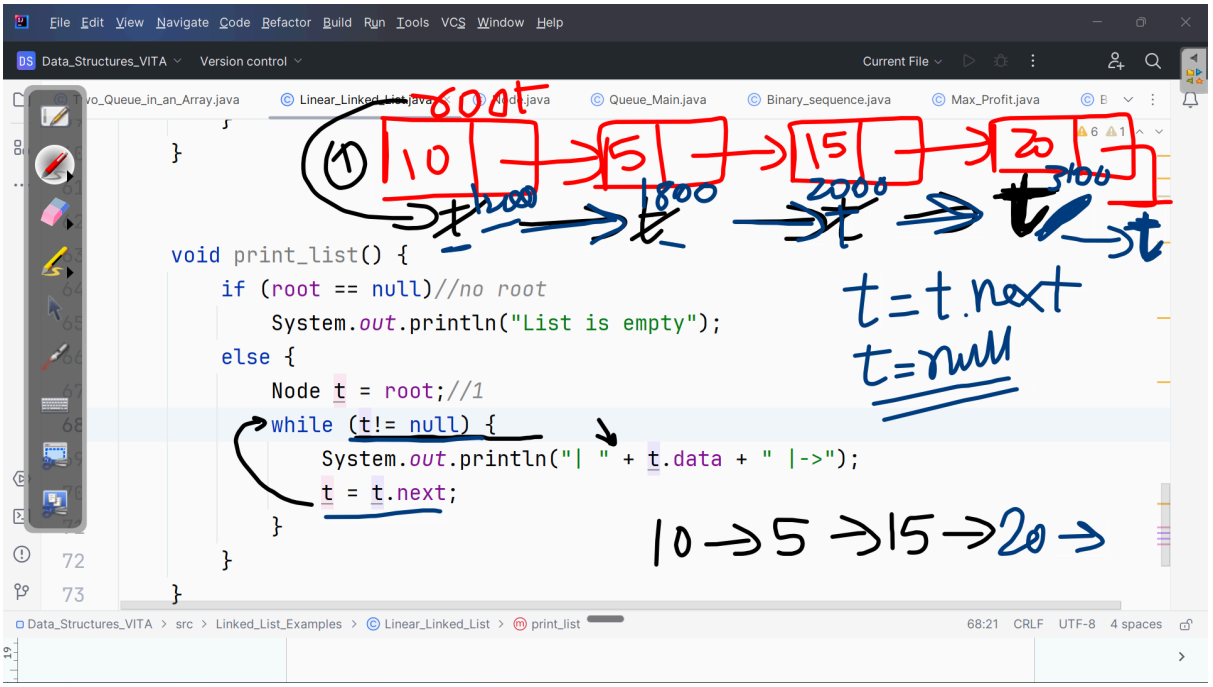
1st/left most should be root only.
right most/last has .next as null.

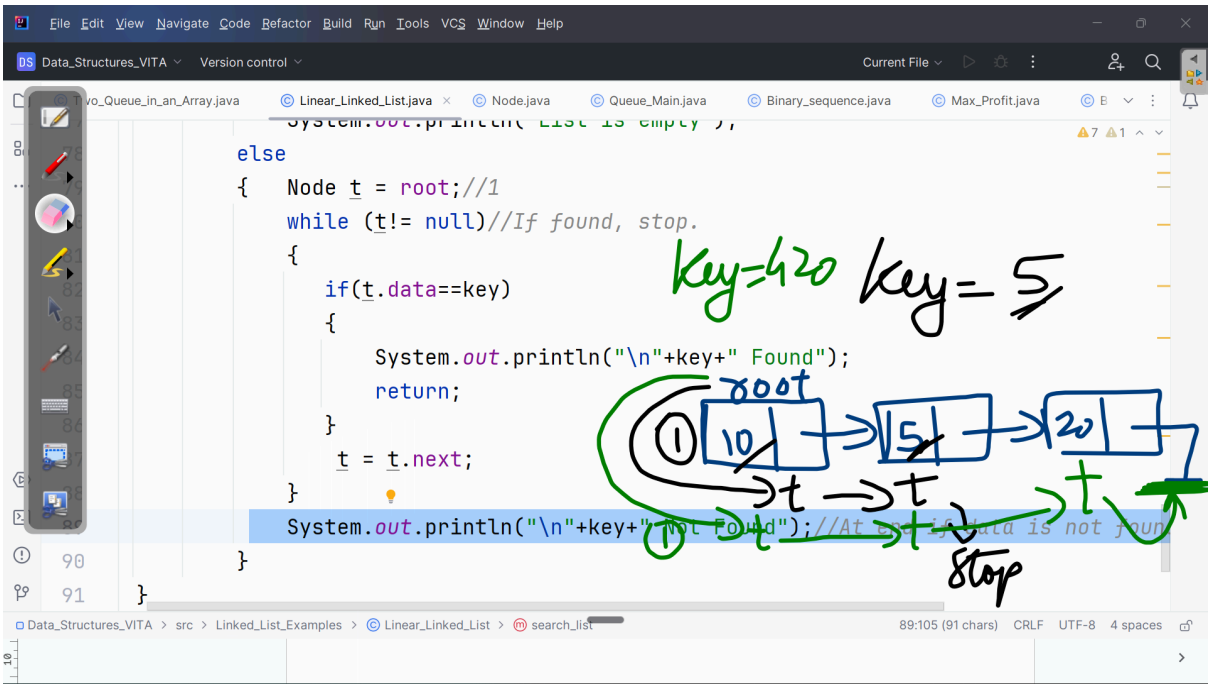

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
Data_Structures_VITA
Version control
Current File
Two_Queue_in_an_Array.java
Linear_Linked_List.java
Node.java
Queue_Main.java
Binary_sequence.java
Max_Profit.java
Node root;//only data member we have
//insert_left:Research new element to the left of the current node
void insert_left(int data)
{
Node n=new Node(data);//created node
if(root==null)//no root
root=n;//1st becomes root
else
{
n.next=root;//1
root=n;//2
}
}
root
10
1900
n
2
root
n
1200
5
4800
1
root
10
1200
20
2000
Data_Structures_VITA > src > Linked_List_Examples > Linear_Linked_List > insert_left
16:10 CRLF UTF-8 4 spaces


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
Data_Structures_VITA
Version control
Current File
Two_Queue_in_an_Array.java
Linear_Linked_List.java
Node.java
Max_Profit.java
}
void delete_left()
{
if(root==null)//no root
System.out.println("List is empty");
else
{
Node t=root;//1
root=root.next;//2
System.out.println("Deleted:"+t.data);//3 response message of deletion
}
}
root
10
1800
1200
t
2
root
20
200
1800
3
AGC
30
null
2100
Data_Structures_VITA > src > Linked_List_Examples > Linear_Linked_List > delete_left
26:26 CRLF UTF-8 4 spaces


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
Data_Structures_VITA
Version control
Current File
Linear_Linked_List.java
Node.java
Queue_Main.java
Max_Profit.java
}
void insert_right(int data)
{
Node n=new Node(data);//created node
if(root==null)//no root
root=n;//1st becomes root
else
{
Node t=root;//1
while(t.next!=null)//2
t=t.next;
t.next=n;//3 connected
}
}
root
10
n
root
1200
1800
1900
3000
3
1
2
t
Data_Structures_VITA > src > Linked_List_Examples > Linear_Linked_List > insert_right
28:26 CRLF UTF-8 4 spaces


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
Data_Structures_VITA
Version control
Current File
Linear_Linked_List.java
Node.java
Max_Profit.java
{
Node t,t2;
t=t2=root;//1
while(t.next!=null)//2
{t2=t;
t=t.next;
}
if(t==root)//single node
root=null;//manual deletion
else
t2.next=null;//break the link
System.out.println("Deleted:"+t.data);//3 response message of deletion
}
}
root
10
1200
20
1800
30
2000
40
3000
null
t
t2
GC
Single
Node
Data_Structures_VITA > src > Linked_List_Examples > Linear_Linked_List > delete_right
58:46 CRLF UTF-8 4 spaces






```
package Linked_List_Examples;

public class Linear_Linked_List
{
   Node root;//only data member we have
   //insert_left:Research new element to the left of the current node.
   void insert_left(int data)
   {
      Node n=new Node(data);//created node
      if(root==null)//no root
```

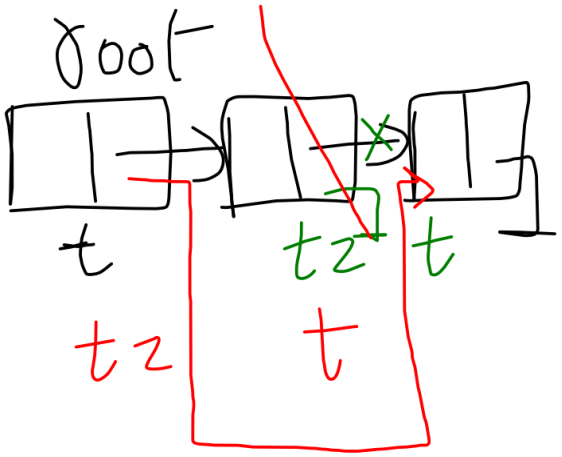
```
        root=n;//1st becomes root
    else
    {
        n.next=root;//1
        root=n;//2
    }
}
void insert_right(int data)
{
    Node n=new Node(data);//created node
    if(root==null)//no root
        root=n;//1st becomes root
    else
    {
       Node t=root;//1
        while(t.next!=null)//2
            t=t.next;
        t.next=n;//3 connected
    }
}
void delete_left()
{
    if(root==null)//no root
        System.out.println("List is empty");
    else
    {
        Node t=root;//1
        root=root.next;//2
        System.out.println("Deleted:"+t.data);//3 response message of deletion
    }
}
void delete_right()
{
    if(root==null)//no root
        System.out.println("List is empty");
    else
    {
        Node t,t2;
        t=t2=root;//1
        while(t.next!=null)//2
        {t2=t;
         t=t.next;
        }
        if(t==root)//single node
            root=null;//manual deletion
        else
            t2.next=null;//break the link
        System.out.println("Deleted:"+t.data);//3 response message of deletion
```

```
        }
    }


    void print_list() {
        if (root == null)//no root
            System.out.println("List is empty");
        else {
            Node t = root;//1
            while (t != null) {
                System.out.println("| " + t.data + " |->");
                t = t.next;
            }
        }
    }
        void search_list(int key)
        {
            if (root == null)//no root
                System.out.println("List is empty");
            else
            {   Node t = root;//1
                while (t!= null)//If found, stop.
                {
                  if(t.data==key)
                  {
                      System.out.println("\n"+key+" Found");
                      return;
                  }
                   t = t.next;
                }
                System.out.println("\n"+key+" Not Found");//At end if data is not found, t will be
NULL.
            }
    }

}
```

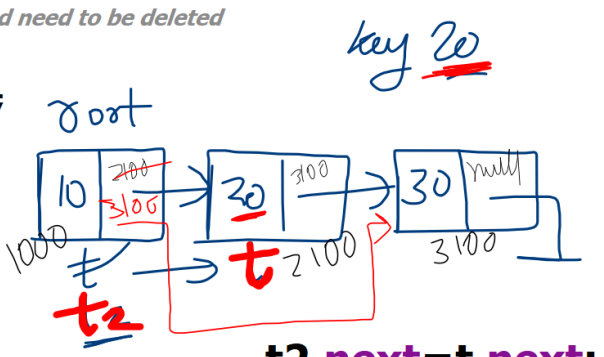
case 1: left most: if(t==root)

case 2: righ most: else if(t.next==null)

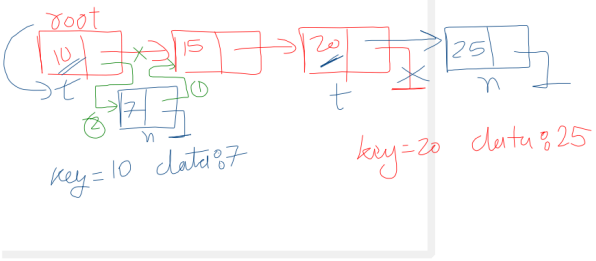
case 3:in-between: else




```
void delete_element(int key)//key if found need to be deleted
  {
    if (root == null)//no root
       System.out.println("List is empty");
    else {
       Node t, t2;
       t = t2 = root;//1
       while (t != null)//If found, stop.
       {
          if (t.data == key)
             break;
          t2=t;
          t = t.next;
       }
       if (t == null)//not found
          System.out.println("\n" + key + " Not Found");
       else
       {
          System.out.println("\n" + key + " Not Found");
          //deletion part
          if(t==root)//case 1:delete left
             root=root.next;
          else if(t.next==null)//case 2 : right most: delete right
             t2.next=null;//deleted
          else //case 3 in-between deletion
             t2.next=t.next;//redirect to next of t
          System.out.println("\n"+t.data+" deleted.");
       }
    }
```




```
//We'll search for the key element if found.
//We'll create a new node with given data and insert it after that key node.
void insert_after(int key,int data)
{
   if (root == null)//no root
      System.out.println("List is empty");
   else {
      Node t;
      t = root;//1
      while (t != null)//If found, stop.
      {
         if (t.data == key)
            break;
         t = t.next;
      }
      if (t == null)//not found
         System.out.println("\n" + key + " Not Found");
      else
      {
         System.out.println("\n" + key + " Found");
         Node n=new Node(data);//created new node
         //insertion part
        n.next=t.next;//1 ref to who ever is t.next
         t.next=n;//2 t ref to n

         System.out.println("\n"+t.data+" deleted.");
      }
   }
}
```



```
package Linked_List_Examples;

public class Linear_Linked_List {
    Node root;//only data member we have

    //insert_left:Research new element to the left of the current node.
    void insert_left(int data) {
        Node n = new Node(data);//created node
        if (root == null)//no root
            root = n;//1st becomes root
        else {
            n.next = root;//1
            root = n;//2
        }
    }

    void insert_right(int data) {
        Node n = new Node(data);//created node
        if (root == null)//no root
            root = n;//1st becomes root
        else {
            Node t = root;//1
            while (t.next != null)//2
                t = t.next;
            t.next = n;//3 connected
        }
    }

    void delete_left() {
        if (root == null)//no root
            System.out.println("List is empty");
        else {
            Node t = root;//1
            root = root.next;//2
            System.out.println("Deleted:" + t.data);//3 response message of deletion
        }
    }

    void delete_right() {
        if (root == null)//no root
            System.out.println("List is empty");
        else {
            Node t, t2;
            t = t2 = root;//1
            while (t.next != null)//2
            {
                t2 = t;
                t = t.next;
```

```
            }
            if (t == root)//single node
                root = null;//manual deletion
            else
                t2.next = null;//break the link
            System.out.println("Deleted:" + t.data);//3 response message of deletion
        }
    }


    void print_list() {
        if (root == null)//no root
            System.out.println("List is empty");
        else {
            Node t = root;//1
            while (t != null) {
                System.out.println("| " + t.data + " |->");
                t = t.next;
            }
        }
    }

    void search_list(int key) {
        if (root == null)//no root
            System.out.println("List is empty");
        else {
            Node t = root;//1
            while (t != null)//If found, stop.
            {
                if (t.data == key) {
                    System.out.println("\n" + key + " Found");
                    return;
                }
                t = t.next;
            }
            System.out.println("\n" + key + " Not Found");//At end if data is not found, t will be
NULL.
        }
    }

    void delete_element(int key)//key if found need to be deleted
    {
        if (root == null)//no root
            System.out.println("List is empty");
        else {
            Node t, t2;
            t = t2 = root;//1
            while (t != null)//If found, stop.
```

```
            {
                if (t.data == key)
                    break;
                t2=t;
                t = t.next;
            }
            if (t == null)//not found
                System.out.println("\n" + key + " Not Found");
            else
            {
                System.out.println("\n" + key + " Found");
                //deletion part
                if(t==root)//case 1:delete left
                    root=root.next;
                else if(t.next==null)//case 2 : right most: delete right
                    t2.next=null;//deleted
                else //case 3 in-between deletion
                    t2.next=t.next;//redirect to next of t
                System.out.println("\n"+t.data+" deleted.");
        }
    }
}
    //We'll search for the key element if found.
    //We'll create a new node with given data and insert it after that key node.
    void insert_after(int key,int data)
    {
        if (root == null)//no root
            System.out.println("List is empty");
        else {
            Node t;
            t = root;//1
            while (t != null)//If found, stop.
            {
                if (t.data == key)
                    break;
                t = t.next;
            }
            if (t == null)//not found
                System.out.println("\n" + key + " Not Found");
            else
            {
                System.out.println("\n" + key + " Found");
                Node n=new Node(data);//created new node
                //insertion part
               n.next=t.next;//1 ref to who ever is t.next
                t.next=n;//2 t ref to n

                System.out.println("\n"+t.data+" deleted.");
```

```
        }
      }
    }

  void sort_list()
  {
     if(root==null)
        System.out.println("Empty list");
     else
     {
        for(Node i=root;i.next!=null;i=i.next)//for passes
        {
           for(Node t=root,t2=t.next;t2!=null;t=t.next,t2=t2.next)//sorts
           {
              if(t.data>t2.data)
              {
                 int temp=t.data;t.data=t2.data;t2.data=temp;
              }
           }
        }
     }
     System.out.println("\nSorting done..");
  }
}


package Linked_List_Examples;
import java.util.Scanner;
public class Linked_List_Main {

   public static void main(String[] args)
   {

      Scanner sc = new Scanner(System.in);
      int ch=0;
      Linear_Linked_List list = new Linear_Linked_List();

      do {
         System.out.println("\n======== LINKED LIST MENU ========");
         System.out.println("1. Insert Left");
         System.out.println("2. Insert Right");
         System.out.println("3. Delete Left");
         System.out.println("4. Delete Right");
         System.out.println("5. Print List");
         System.out.println("6. Search Element");
         System.out.println("7. Delete Element");
         System.out.println("8. Insert After");
         System.out.println("9. Sort List");
```

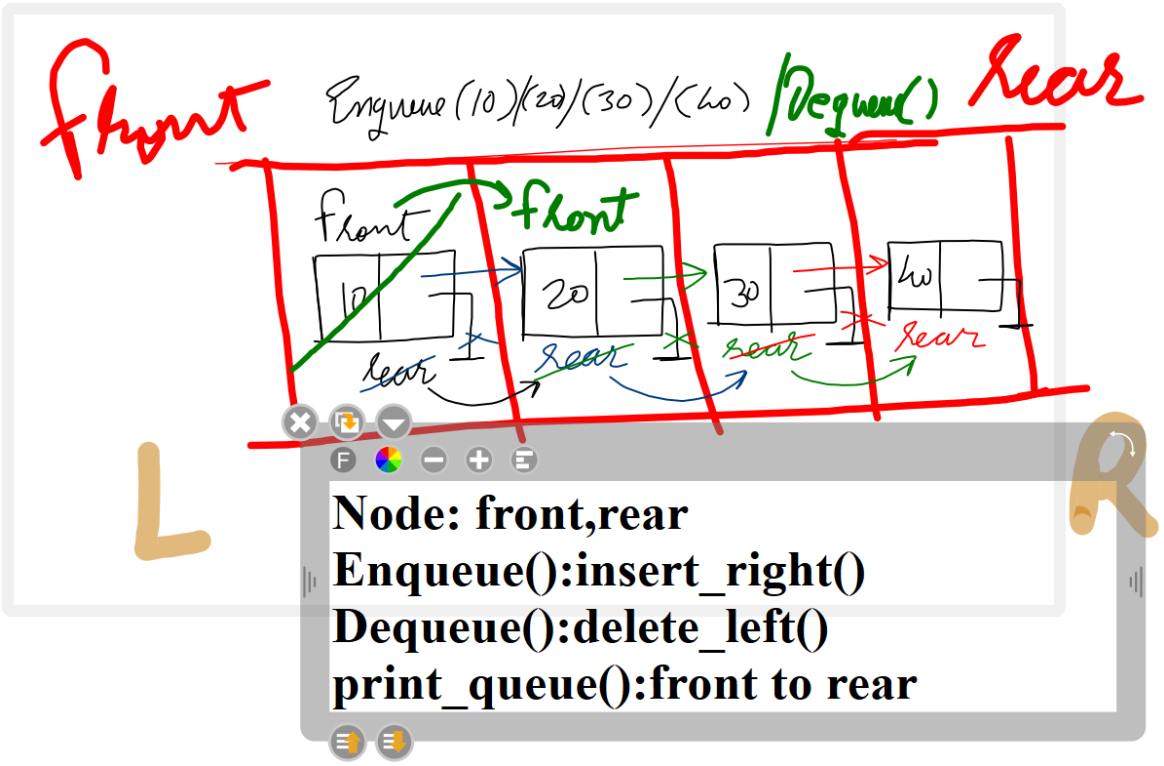
```
System.out.println("0. Exit");
System.out.println("==================================");
System.out.print("Enter your choice: ");
ch = sc.nextInt();
switch (ch) {

    case 1:
        System.out.print("Enter data to insert left: ");
        list.insert_left(sc.nextInt());
        break;

    case 2:
        System.out.print("Enter data to insert right: ");
        list.insert_right(sc.nextInt());
        break;

    case 3:
        list.delete_left();
        break;

    case 4:
        list.delete_right();
        break;

    case 5:
        list.print_list();
        break;

    case 6:
        System.out.print("Enter element to search: ");
        list.search_list(sc.nextInt());
        break;

    case 7:
        System.out.print("Enter element to delete: ");
        list.delete_element(sc.nextInt());
        break;

    case 8:
        System.out.print("Enter key after which to insert: ");
        int key = sc.nextInt();
        System.out.print("Enter data to insert: ");
        int data = sc.nextInt();
        list.insert_after(key, data);
        break;

    case 9:
        list.sort_list();
```

```
                break;

            case 0:
                System.out.println("Exiting .. coded by amar.career👋");
                return;

            default:
                System.out.println("invalid choice. Please select from the menu.");
        }
    }while(ch!=0);
  }
}
```

```
package Linked_List_Examples;

import Stack_Examples.Stack_Class;

import java.util.Scanner;

public class Dynamic_Stack
{
   Node tos;//only data member we have


   void push(int data)
   {
      Node n = new Node(data);//created node
      if (tos == null)//no root
         tos = n;//1st becomes root
      else {
         n.next = tos;//1
         tos= n;//2
      }
   }
   void pop() {
      if (tos == null)//no root
         System.out.println("Stack is empty");
      else {
         Node t = tos;//1
         tos = tos.next;//2
         System.out.println("poped:" + t.data);//3 response message of deletion
      }
   }

   void print_stack() {
      if (tos == null)//no root
         System.out.println("Stack is empty");
      else {
         Node t = tos;//1
         while (t != null) {
            System.out.println(t.data );
            System.out.println("======");
            t = t.next;
         }
      }
   }
      int  peek() {
         if (tos == null)//no root
         {
            System.out.println("Stack is empty");
```

```
            return (-1);
        }
        else {
          return (tos.data);
            }
        }
public static void main(String[] args)
{
    Dynamic_Stack obj=new Dynamic_Stack();
    int choice;
    Scanner in=new Scanner(System.in);
    do
    {
        System.out.println("\nStack Menu");
        System.out.println("==========");
        System.out.println("1.Push");
        System.out.println("2.Pop");
        System.out.println("3.Peek");
        System.out.println("4.Print");
        System.out.println("0.Exit");
        System.out.println("--------");
        System.out.print(":");
        choice=in.nextInt();
        switch(choice)
        {
            case 1:
                  System.out.println("Enter element to push:");
                  int e=in.nextInt();
                  obj.push(e);
                  System.out.println("Element pushed");
                  break;
            case 2:
                  obj.pop();
                  break;
            case 3:
                  int res=obj.peek();
                  if (res!=-1)//if not empty
                      System.out.println("At peek:"+res);
               break;
            case 4:
              obj.print_stack();
               break;
            case 0:
               System.out.println("Thanks for using the code: amar.career");
               break;
            default:
               System.out.println("check the option selected.");
               break;
```

```
        }
    }while(choice!=0);
}

}
```




```
package Linked_List_Examples;

import Queue_Examples.Priority_Queue;

import java.util.Scanner;

public class Dynamic_Queue
{
    Node front,rear;

    void enqueue(int data)
    {
        Node n = new Node(data);//created node
        if (rear == null)//no root
```

```
    {
        front=rear=n;//both on one
    }
    else {
        rear.next=n;//1 connect to last
        rear=n;//2 move rear
    }
}
void dequeue() {
    if (front == null)//no root
        System.out.println("Queue is empty");
    else
    {
        Node t=front;
        if(front==rear)//single node
            front=rear=null;//manual reset
        else
            front=front.next;//move ahead
        System.out.println("Deleted:" + t.data);//3 response message of deletion
    }
}
void print_queue() {
    if (front == null)//no root
        System.out.println("Queue is empty");
    else {
        Node t = front;//1
        while (t != null) {
            System.out.print("-|" + t.data + "|_");
            t = t.next;
        }
    }
}
public static void main(String[] args)
{
    Dynamic_Queue obj=new Dynamic_Queue();
    int choice;
    Scanner in=new Scanner(System.in);

    do
    {
        System.out.println("\nQueue Menu");
        System.out.println("==========");
        System.out.println("1.Enqueue");
        System.out.println("2.Dequeue");
        System.out.println("3.Print");
        System.out.println("0.Exit");
        System.out.println("--------");
        System.out.print(":");
```

```
            choice=in.nextInt();
            switch(choice)
            {
               case 1:
                     System.out.println("Enter element to enter:");
                     int e=in.nextInt();
                     obj.enqueue(e);
                     System.out.println("Element Enqueued");
                     break;
               case 2:
                  obj.dequeue();
                  break;
               case 3:
                  obj.print_queue();
                  break;
               case 0:
                  System.out.println("Thanks for using the code: amar.career");
                  break;
               default:
                  System.out.println("check the option selected.");
                  break;

            }
         }while(choice!=0);
      }
}
```

LinkedList Using Java Collections — Theory + Methods + Code

1. High-Level Theory

Java's **LinkedList** (java.util.LinkedList) is a ready-made, optimized, doubly-linked list implementation.
Think of it as a plug-and-play data structure that gives you:

- **O(1) insertion/deletion** at head/tail
- **O(n) access** when fetching by index
- **Queue + Stack + List** capabilities in one hybrid container
- Flexibility without the headache of manual node creation

2. Key Operations (A.K.A. The API Cheat-Sheet)

Core List Ops

Method	Purpose
add(E e)	Add element at end
addFirst(E e)	Insert at head
addLast(E e)	Insert at tail
remove()	Remove head
removeFirst()	Remove head
removeLast()	Remove tail
get(int index)	Fetch by index
getFirst()	Access head
getLast()	Access tail
size()	Count elements
isEmpty()	Check emptiness

Queue-Style Ops

Method	Use Case
offer(E e)	Enqueue element
poll()	Dequeue (null if empty)
peek()	View front without removal

Stack-Style Ops

Method	Use Case
push(E e)	Add on top
pop()	Remove from top

Utility Ops

Method	Use Case
contains(Object o)	Search element
clear()	Remove all items

---

3. Full-Flow Example Code

```
import java.util.LinkedList;
import java.util.Scanner;

public class LinkedListCollectionDemo {
    public static void main(String[] args) {
        LinkedList<Integer> list = new LinkedList<>();
        Scanner sc = new Scanner(System.in);

        while (true) {
            System.out.println("\n===== LinkedList Operations Menu
=====");
            System.out.println("1. Add at End");
            System.out.println("2. Add at Beginning");
            System.out.println("3. Remove First");
            System.out.println("4. Remove Last");
            System.out.println("5. Display List");
```

```
            System.out.println("6. Get Element by Index");
            System.out.println("7. Check if List Contains Element");
            System.out.println("8. List Size");
            System.out.println("9. Clear List");
            System.out.println("0. Exit");
            System.out.print("Enter your choice: ");

            int ch;
            try {
                ch = sc.nextInt();
            } catch (Exception e) {
                System.out.println("Invalid input, fam. Try
again.");
                sc.nextLine();
                continue;
            }

            switch (ch) {
                case 1:
                    System.out.print("Enter value: ");
                    list.add(sc.nextInt());
                    System.out.println("Added at end.");
                    break;

                case 2:
                    System.out.print("Enter value: ");
                    list.addFirst(sc.nextInt());
                    System.out.println("Added at beginning.");
                    break;

                case 3:
                    if (!list.isEmpty()) {
                        System.out.println("Removed: " +
list.removeFirst());
                    } else {
                        System.out.println("List is empty.");
                    }
                    break;

                case 4:
                    if (!list.isEmpty()) {
```

```
                        System.out.println("Removed: " +
list.removeLast());
                    } else {
                        System.out.println("List is empty.");
                    }
                    break;

                case 5:
                    System.out.println("LinkedList: " + list);
                    break;

                case 6:
                    System.out.print("Enter index: ");
                    int idx = sc.nextInt();
                    if (idx >= 0 && idx < list.size()) {
                        System.out.println("Element: " +
list.get(idx));
                    } else {
                        System.out.println("Index out of range.");
                    }
                    break;

                case 7:
                    System.out.print("Enter element to check: ");
                    int val = sc.nextInt();
                    System.out.println("Contains? " +
list.contains(val));
                    break;

                case 8:
                    System.out.println("Size: " + list.size());
                    break;

                case 9:
                    list.clear();
                    System.out.println("List cleared.");
                    break;

                case 0:
                    System.out.println("Exiting program. Bye!");
                    return;
```

```
                default:
                    System.out.println("Invalid choice. Try
again.");
            }
        }
    }
}
```