1.Find Middle of the Linked list
Problem statement : Given a singly linked list of 'N' nodes. The objective is to determine the middle node of a singly linked list. However, if the list has an even number of nodes, we return the second middle node.

Code:

```
package com.handson_prblm_sol_10;
import java.util.Scanner;
class Node1 {
  int data;
  Node1 next;
  Node1(int data) {
     this.data = data;
     this.next = null;
  }
}
class Link1 {
  Node1 head = null;
  public void addNode(int data) {
     Node1 newnode = new Node1(data);
     if (head == null) {
        head = newnode;
     } else {
        Node1 current = head;
        while (current.next != null) {
           current = current.next;
        }
        current.next = newnode;
     }
  }
  public void getEle(int pos) {
     if (head == null) {
        System.out.print("List is empty");
     } else {
        Node1 current = head;
        int cur = 1;
        while (cur < pos && current != null) {
           current = current.next;
           cur++;
        }
        while (current != null) {
           System.out.print(current.data + " ");
           current = current.next;
        }
     }
  }
  public int size() {
     Node1 current = head;
     int s = 0;
```

```
        while (current != null) {
            current = current.next;
            s++;
        }
        return s;
    }
}
public class Middle_1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Link1 obj = new Link1();
        System.out.print("Enter elements (end with -1):");
        int i;
        while ((i = sc.nextInt()) != -1) {
            obj.addNode(i);
        }
        int size = obj.size();
        int mid = (size % 2 == 0) ? (size / 2) + 1 : (size / 2) + 1;
        obj.getEle(mid);
    }
}
```

Output:
Enter elements (end with -1):1 2 3 4 5 6 -1
4 5 6

2.Remove nth node Problem Statement: Given the head of a linked list, remove the n th node from the end of the list and return its head.

Code:

```
package com.handson_prblm_sol_10;
import java.util.*;
class Node2 {
    int data;
    Node2 next;
    Node2(int data) {
        this.data = data;
        this.next = null;
    }
}
class Link2 {
    Node2 head = null;
    public void addNode(int data) {
        Node2 newnode = new Node2(data);
        if (head == null) {
            head = newnode;
        } else {
            Node2 current = head;
            while (current.next != null) {
```

```
            current = current.next;
        }
        current.next = newnode;
    }
}

public void DelAtPos(int pos)
{
        if(head==null)
        {
                System.out.print("List is Empty");
        }
        else
        {
                Node2 current = head;
                int cur=1;
                while(cur<pos-1 && current.next!=null)
                {
                        current = current.next;
                        cur++;
                }
                current.next=current.next.next;
        }
}
        public void display() {
            Node2 current = head;
            if(isEmpty()) {
                System.out.print("List is empty");
            } else {
                while(current != null) {
                    System.out.print(current.data + " ");
                    current = current.next;
                }
            }
        }
                public boolean isEmpty() {
                return head == null;
            }
                  public int size() {
                      Node2 current = head;
                      int s = 0;
                      while (current != null) {
                          current = current.next;
                          s++;
                      }
                      return s;
                  }
              }
public class Removenth_2 {
```

```
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
    Link2 obj = new Link2();
    System.out.print("Enter elements (end with -1):");
    int i;
    while ((i = sc.nextInt()) != -1) {
       obj.addNode(i);
    }
    System.out.print("Enter n:");
    int n =sc.nextInt();
    int size = obj.size();
    obj.DelAtPos((size-n)+1);
    obj.display();
        }
}
```

Output:
Enter elements (end with -1):1 2 -1
Enter n:1
1

3.Delete Alternate Nodes Problem statement : Given a Singly Linked List of integers, delete all the alternate nodes in the list.

Code:

```
package com.handson_prblm_sol_10;
import java.util.*;
class Node3{
  int data;
  Node3 next;
  Node3(int data) {
     this.data = data;
     this.next = null;
  }
}
class Link3 {
  Node3 head = null;
  public void addNode(int data) {
     Node3 newnode = new Node3(data);
     if (head == null) {
        head = newnode;
     } else {
        Node3 current = head;
        while (current.next != null) {
           current = current.next;
        }
        current.next = newnode;
     }
  }
```

```
  public void altDel()
  {
     Node3 current = head;
     while (current != null && current.next != null) {
        current.next = current.next.next;
        current = current.next;
     }
          Node3 current1 = head;
          while(current1!=null)
          {
                    System.out.print(current1.data+" ");
                    current1 = current1.next;
          }
  }
}

public class DelAlt_3 {
          public static void main(String[] args) {
                    Scanner sc = new Scanner(System.in);
     Link3 obj = new Link3();
     System.out.print("Enter elements (end with -1):");
     int i;
     while ((i = sc.nextInt()) != -1) {
        obj.addNode(i);
     }
    obj.altDel();
          }
}
```

Output:
Enter elements (end with -1):1 2 3 4 5 -1
1 3 5

4.Change start Node Problem statement You are given a singly linked list and an integer ‘K’. You aresupposed to make ‘K’ th node from the end of a linked list as the starting node of the linked list.

Code:

```
package com.handson_prblm_sol_10;
import java.util.Scanner;
class Node4 {
  int data;
  Node4 next;
  Node4(int data) {
     this.data = data;
     this.next = null;
  }
}
class Link4 {
```

```
Node4 head = null;
public void addNode(int data) {
   Node4 newnode = new Node4(data);
   if (head == null) {
      head = newnode;
   } else {
      Node4 current = head;
      while (current.next != null) {
         current = current.next;
      }
      current.next = newnode;
   }
}
public int DelAtPos(int pos)
{
        int val=0;
        if(head==null)
        {
                System.out.print("List is Empty");
        }
        else
        {
                Node4 current = head;
                int cur=1;
                while(cur<pos-1 && current.next!=null)
                {
                        current = current.next;
                        cur++;
                }
                 val = current.next.data;
                current.next=current.next.next;
        }
        return val;
}
public void insertAtBeg(int val)
{
        Node4 newnode = new Node4(val);
        newnode.next=head;
        head=newnode;
}
public void display() {
   Node4 current = head;
   if(isEmpty()) {
      System.out.print("List is empty");
   } else {
      while(current != null) {
         System.out.print(current.data + " ");
         current = current.next;
      }
```

```
    }
}
        public boolean isEmpty() {
       return head == null;
    }
        public int size() {
            Node4 current = head;
            int s = 0;
            while (current != null) {
               current = current.next;
               s++;
            }
            return s;
         }
  }
public class StartNode_4 {
        public static void main(String[] args) {
                 Scanner sc = new Scanner(System.in);
    Link4 obj = new Link4();
    System.out.print("Enter elements (end with -1):");
    int i;
    while ((i = sc.nextInt()) != -1) {
       obj.addNode(i);
    }
    System.out.print("Enter n:");
    int n =sc.nextInt();
    int size = obj.size();
    int da = obj.DelAtPos((size-n)+1);
    obj.insertAtBeg(da);
    obj.display();
        }
}
```

Output:
Enter elements (end with -1):7 11 13 17 19 23 29 -1
Enter n:3
19 7 11 13 17 23 29

5.Problem statement You are given a linked list of 'N' nodes. Your task is to reverse the linked list from position 'X' to 'Y'.

Code:

```
package com.handson_prblm_sol_10;
import java.util.Scanner;
class Node5 {
  int data;
  Node5 next;
  Node5(int data) {
     this.data = data;
```

```
        this.next = null;
    }
}
public class SubList_5 {
    private Node5 head;
    public Node5 createNode(int data) {
        return new Node5(data);
    }
    public void addNode(int data) {
        Node5 newNode = createNode(data);
        if (head == null) {
            head = newNode;
        } else {
            Node5 current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newNode;
        }
    }
    public void display() {
        Node5 current = head;
        if (head == null) {
            System.out.println("List is empty");
            return;
        }
        while (current != null) {
            System.out.print(current.data + " -> ");
            current = current.next;
        }
        System.out.println("-1");
    }
    public void reverse(int x, int y) {
        if (head == null || x >= y) {
            return;
        }
        Node5 prev = null;
        Node5 current = head;
        for (int i = 1; current != null && i < x; i++) {
            prev = current;
            current = current.next;
        }
        Node5 start = current;
        Node5 end = start.next;
        for (int i = 0; i < y - x; i++) {
            start.next = end.next;
            end.next = current;
            current = end;
            end = start.next;
```

```
        }
        if (prev != null) {
            prev.next = current;
        } else {
            head = current;
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        SubList_5 list = new SubList_5();
        System.out.println("Enter elements: (-1 to stop)");
        while (true) {
            int data = sc.nextInt();
            if (data == -1) {
                break;
            }
            list.addNode(data);
        }
        System.out.println("Enter x and y value:");
        int x = sc.nextInt();
        int y = sc.nextInt();
        sc.close();
        list.reverse(x, y);
        list.display();
    }
}
```

Output:
Enter elements: (-1 to stop)
10 20 30 40 50 60 -1
Enter x and y value:
2 5
10 -> 50 -> 40 -> 30 -> 20 -> 60 -> -1

6.Sort Linked List Problem statement: You are given a Singly Linked List of integers which is sorted based on absolute value. You have to sort the Linked List based on actual values. The absolute value of a real number x, denoted |x|, is the non-negative value of x without regard to its sign.

Code:

```
package com.handson_prblm_sol_10;
import java.util.*;
class Node6 {
    int data;
    Node6 next;
    Node6(int data) {
        this.data = data;
        this.next = null;
```

```
  }
}
class Link6 {
  Node6 head = null;
  public void addNode(int data) {
     Node6 newnode = new Node6(data);
     if (head == null) {
        head = newnode;
     } else {
        Node6 current = head;
        while (current.next != null) {
           current = current.next;
        }
        current.next = newnode;
     }
  }
  public void sort()
  {
         Node6 current = head;
         for(Node6 i=current;current.next!=null;current=current.next)
         {
                for(Node6 index = current.next;index!=null;index=index.next)
                {
                       if(current.data>index.data)
                       {
                              int temp = current.data;
                              current.data=index.data;
                              index.data=temp;
                       }
                }
         }
  }
  public void display() {
     Node6 current = head;
     if(isEmpty()) {
        System.out.print("List is empty");
     } else {
        while(current != null) {
           System.out.print(current.data + " ");
           current = current.next;
        }
     }
   }
         public boolean isEmpty() {
        return head == null;
     }
  }
public class Sort_6 {
         public static void main(String[] args) {
```

```
                    Scanner sc = new Scanner(System.in);
        Link6 obj = new Link6();
        System.out.print("Enter elements (end with -1):");
        int i;
        while ((i = sc.nextInt()) != -1) {
            obj.addNode(i);
        }
        obj.sort();
        obj.display();
             }
}
```

Output:
Enter elements (end with -1):9 8 7 6 5 -1
5 6 7 8 9

7.Palindrome check Problem Statement: You are given the head of a singly linked list. Determine if the linked list is a palindrome, return true if it is a palindrome or false otherwise. A Linked List is a palindrome if it reads the same from left to right and from right to left.

Code:

```
package com.handson_prblm_sol_10;
import java.util.Scanner;
class Node7 {
  int data;
  Node7 next;
  Node7(int data) {
      this.data = data;
      this.next = null;
  }
}
class Link7 {
  Node7 head = null;
  StringBuilder s = new StringBuilder();
  public void addNode(int data) {
      Node7 newnode = new Node7(data);
      s.append(data);
      if (head == null) {
          head = newnode;
      } else {
          Node7 current = head;
          while (current.next != null) {
              current = current.next;
          }
          current.next = newnode;
      }
  }
  public void isPalin() {
      String original = s.toString();
```

```
        String reversed = s.reverse().toString();
        s.reverse();
        if (original.equals(reversed)) {
            System.out.print("True");
        } else {
            System.out.print("False");
        }
    }
}
public class Palindrome_7 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Link7 obj = new Link7();
        System.out.print("Enter elements (end with -1): ");
        int i;
        while ((i = sc.nextInt()) != -1) {
            obj.addNode(i);
        }
        obj.isPalin();
    }
}
```

Output:
Enter elements (end with -1): 1 2 1 -1
True

8.Divide Linked List in Two Problem statement You have been given a singly Linked List of integers. Your task is to divide this list into two smaller singly-linked lists in which the nodes appear in alternating fashion from the original list.

Code:

```
package com.handson_prblm_sol_10;
import java.util.*;
class Node8{
    int data;
    Node8 next;
    Node8(int data) {
        this.data = data;
        this.next = null;
    }
}
class Link8 {
    Node3 head = null;
     StringBuilder s = new StringBuilder();
    public void addNode(int data) {
        Node3 newnode = new Node3(data);
        if (head == null) {
            head = newnode;
        } else {
```

```
            Node3 current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newnode;
        }
    }
    public void altDel()
    {
        Node3 current = head;
        while (current != null && current.next != null) {
             s.append(current.next.data);
            current.next = current.next.next;
            current = current.next;
        }
    }
    public void display()
    {
             Node3 current1 = head;
             while(current1!=null)
             {
                      System.out.print(current1.data+" ");
                      current1 = current1.next;
             }
             System.out.println();
             String res = s.toString();
             for(int i=0;i<res.length();i++)
             System.out.print(res.charAt(i)+" ");
    }
}

public class Divide_8 {
             public static void main(String[] args) {
                      Scanner sc = new Scanner(System.in);
        Link8 obj = new Link8();
        System.out.print("Enter elements (end with -1):");
        int i;
        while ((i = sc.nextInt()) != -1) {
            obj.addNode(i);
        }
       obj.altDel();
       obj.display();
             }
}
```

Output:

```
Enter elements (end with -1):1 2 3 4 5 -1
1 3 5
2 4
```

9.Convert Binary Number in a Linked List to Integer Problem statement Given head which is a reference node to a singly-linked list. The value of each node in the linked list is either 0 or 1. The linked list holds the binary representation of a number. Return the decimal value of the number in the linked list. The most significant bit is at the head of the linked list.

Code:

```
package com.handson_prblm_sol_10;
import java.util.Scanner;
class Node9 {
  int data;
  Node9 next;
  Node9(int data) {
     this.data = data;
     this.next = null;
  }
}
class Link9 {
  Node9 head = null;
  StringBuilder s = new StringBuilder();
  public void addNode(int data) {
     Node9 newnode = new Node9(data);
     s.append(data);
     if (head == null) {
        head = newnode;
     } else {
        Node9 current = head;
        while (current.next != null) {
           current = current.next;
        }
        current.next = newnode;
     }
  }
  public void printBinaryValue() {
     System.out.println("Integer value: " + Integer.parseInt(s.toString(), 2));
  }
}
public class Binary_9 {
  public static void main(String[] args) {
     Scanner sc = new Scanner(System.in);
     Link9 obj = new Link9();
     System.out.print("Enter elements (end with -1): ");
     int i;
     while ((i = sc.nextInt()) != -1) {
        obj.addNode(i);
     }
     obj.printBinaryValue();
  }
}
```

Output:
Enter elements (end with -1): 1 0 1 -1
Integer value: 5

10.Count Triplets Problem statement You have been given an integer ‘X’ and a non-decreasing sorted doubly linked list with distinct nodes.

Code:

```
package com.handson_prblm_sol_10;
import java.util.*;
class Node10 {
  int data;
  Node10 next, prev;
  Node10(int data) {
     this.data = data;
     next = prev = null;
  }
}
class Link10 {
  Node10 head;
         public void create(int data) {
     Node10 newnode = new Node10(data);
     if (head == null) {
        head = newnode;
     } else {
        Node10 current = head;
        while (current.next != null) {
           current = current.next;
        }
        current.next = newnode;
        newnode.prev = current;
     }
  }
  public int countTriplets(int X) {
     if (head == null) {
        return 0;
     }
     Node10 current, first, last;
     int count = 0;
     last = head;
     while (last.next != null) {
        last = last.next;
     }
     for (current = head; current != null; current = current.next) {
        int target = X - current.data;
        first = current.next;
        while (first != null && last != null && first != last && last.next != first) {
```

```
            int sum = first.data + last.data;
            if (sum == target) {
                count++;
                first = first.next;
                last = last.prev;
            } else if (sum < target) {
                first = first.next;
            } else {
                last = last.prev;
            }
        }
        last = head;
        while (last.next != null) {
            last = last.next;
        }
    }
    return count;
  }
}
public class Triplet_10 {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
    Link10 obj = new Link10 ();
    int i;
    System.out.println("Enter integers to add to the list (-1 to end):");
    while ((i = sc.nextInt()) != -1) {
        obj.create(i);
    }
    System.out.print("Enter target:");
    int tar = sc.nextInt();
   System.out.print(obj.countTriplets(tar));
        }
}
```

Output:
Enter integers to add to the list (-1 to end):
-4 2 3 8 9 -1
Enter target:13
2

11.Add two numbers represented as Linked Lists Problem statement :You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list. You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Code:

```
package com.handson_prblm_sol_10;
import java.util.*;
```

```
class Node11 {
  int data;
  Node11 next;
  Node11(int data) {
     this.data = data;
     this.next = null;
  }
}
class Link11 {
          StringBuilder s = new StringBuilder();
  Node11 head = null;
  public void addNode(int data) {
     Node11 newNode = new Node11(data);
     if(head == null)
          head=newNode;
     else
     {
          newNode.next=head;
          head=newNode;
     }
  }
  public String display() {
          s.replace(0, s.length(),"");
     Node11 current = head;
     if(head==null) {
        System.out.print("List is empty");
     } else {
        while(current != null) {
          s.append(current.data);
            current = current.next;
        }
     }
     return s.toString();

}

}
public class AddTwo_11 {
          public static void main(String[] args) {
                    Scanner sc = new Scanner(System.in);
     Link11 obj = new Link11();
     Link11 obj1 = new Link11();
     System.out.print("Enter elements for list 1 (end with -1):");
     int i;
     while ((i = sc.nextInt()) != -1) {
        obj.addNode(i);
     }
     System.out.print("Enter elements for list 2 (end with -1):");
     int j;
```

```
        while ((j= sc.nextInt()) != -1) {
            obj1.addNode(j);
        }
        int sum = Integer.parseInt(obj.display())+Integer.parseInt(obj1.display());
        StringBuilder res = new StringBuilder(Integer.toString(sum));
        res.reverse();
        for(int k=0;k<res.length();k++)
        {
             System.out.print(res.charAt(k)+" ");
        }
             }
}
```

Output:

```
Enter elements for list 1 (end with -1):2 4 3
-1
Enter elements for list 2 (end with -1):5 6 4 -1
7 0 8
```

12.Odd Even Linked List Problem statement :Given the head of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return the reordered list. The first node is considered odd, and the second node is even, and so on. Note that the relative order inside both the even and odd groups should remain as it was in the input.

Code:

```
package com.handson_prblm_sol_10;
import java.util.*;
class Node12{
  int data;
  Node12 next;
  Node12(int data) {
     this.data = data;
     this.next = null;
  }
}
class Link12 {
  Node12 head = null;
   StringBuilder s = new StringBuilder();
  public void addNode(int data) {
     Node12 newnode = new Node12(data);
     if (head == null) {
        head = newnode;
     } else {
        Node12 current = head;
        while (current.next != null) {
           current = current.next;
        }
        current.next = newnode;
     }
```

```
  }
  public void altDel()
  {
     Node12 current = head;
     while (current != null && current.next != null) {
          s.append(current.next.data);
        current.next = current.next.next;
        current = current.next;
     }
  }
  public void display()
  {
          Node12 current1 = head;
          while(current1!=null)
          {
                  System.out.print(current1.data+" ");
                  current1 = current1.next;
          }
          String res = s.toString();
          for(int i=0;i<res.length();i++)
          System.out.print(res.charAt(i)+" ");
  }
}

public class OddEven_12 {
          public static void main(String[] args) {
                  Scanner sc = new Scanner(System.in);
     Link12 obj = new Link12();
     System.out.print("Enter elements (end with -1):");
     int i;
     while ((i = sc.nextInt()) != -1) {
        obj.addNode(i);
     }
    obj.altDel();
    obj.display();
          }
}
```

Output:
Enter elements (end with -1):2 1 3 5 6 4 7 -1
2 3 6 7 1 5 4

13.Reorder List: You are given the head of a singly linked-list. The list can be represented as:L0 → L1 → … → Ln - 1 → Ln Reorder the list to be on the following form: L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → … You may not modify the values in the list's nodes. Only nodes themselves may be changed.

Code:

```
package com.handson_prblm_sol_10;
import java.util.Scanner;
```

```
class Node13 {
  int data;
  Node13 next;
  Node13(int data) {
    this.data = data;
    this.next = null;
  }
}
class Link13 {
  Node13 head = null;
  public void addNode(int data) {
    Node13 newnode = new Node13(data);
    if (head == null) {
      head = newnode;
    } else {
      Node13 current = head;
      while (current.next != null) {
        current = current.next;
      }
      current.next = newnode;
    }
  }
  public void delAtEnd() {
    if (head == null) {
      System.out.print("List is empty");
      return;
    }
    Node13 current = head;
    Node13 prev = null;
    while (current.next != null) {
      prev = current;
      current = current.next;
    }
    if (prev == null) {
      head = null;
    } else {
      prev.next = null;
    }
  }
  public void insertAt(int index, int data) {
    Node13 newnode = new Node13(data);
    if (index == 0) {
      newnode.next = head;
      head = newnode;
      return;
    }
    Node13 current = head;
    for (int i = 0; i < index - 1 && current != null; i++) {
      current = current.next;
```

```
        }
        if (current != null) {
            newnode.next = current.next;
            current.next = newnode;
        }
    }
    public int size() {
        int count = 0;
        Node13 current = head;
        while (current != null) {
            count++;
            current = current.next;
        }
        return count;
    }
    public void reorderList() {
        int count = 2;
        Node13 currentnode = head;
        while (count <= size() - 1) {
            Node13 current = currentnode;
            while (current.next != null) {
                current = current.next;
            }
            int temp = current.data;
            delAtEnd();
            insertAt(count - 1, temp);
            count += 2;
            currentnode = currentnode.next;
        }
    }
    public void display() {
        if (head == null) {
            System.out.println("List is empty");
            return;
        }
        else {
        Node13 current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
      }}
}
public class Reorder_13 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Link13 obj = new Link13();
        System.out.print("Enter elements (end with -1): ");
```

```
        int i;
        while ((i = sc.nextInt()) != -1) {
            obj.addNode(i);
        }
        obj.reorderList();
        obj.display();
        sc.close();
    }
}
```

Output:
Enter elements (end with -1): 1 2 3 4 5 -1
1 5 2 4 3

14. 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.

Code:

```
package com.handson_prblm_sol_10;
import java.util.Scanner;
class Node14 {
    int data;
    Node14 next;
    Node14(int data) {
        this.data = data;
        this.next = null;
    }
}
public class Merge_14 {
        Node14 head;
        public void Mergesort(Merge_14 list2) {
            Merge_14 newList = new Merge_14();
            Node14 current = head;
            Node14 current2 = list2.head;
            while (current != null && current2 != null) {
                if (current.data < current2.data) {
                    newList.insert(current.data);
                    current = current.next;
                } else {
                    newList.insert(current2.data);
                    current2 = current2.next;
                }
            }
            while (current != null) {
                newList.insert(current.data);
                current = current.next;
            }
            while (current2 != null) {
```

```
            newList.insert(current2.data);
            current2 = current2.next;
        }
        head = newList.head;
    }
    public void insert(int data) {
        Node14 newNode = new Node14(data);
        if (head == null) {
            head = newNode;
            return;
        }
        Node14 last = head;
        while (last.next != null) {
            last = last.next;
        }
        last.next = newNode;
    }
    public void printList() {
        Node14 current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        Merge_14 list1 = new Merge_14();
        Merge_14 list2 = new Merge_14();
        int i = 0, j = 0;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter integers to add to list 1 (-1 to end):");
        while ((i = sc.nextInt()) != -1) {
            list1.insert(i);
        }
        System.out.println("Enter integers to add to list 2 (-1 to end):");
        while ((j = sc.nextInt()) != -1) {
            list2.insert(j);
        }
        sc.close();
        System.out.println("List1 before merge sort:");
        list1.printList();
        System.out.println("List2 before merge sort:");
        list2.printList();
        list1.Mergesort(list2);
        System.out.println("Merged and sorted list:");
        list1.printList();
    }
}
```

Output:
Enter integers to add to list 1 (-1 to end):
1 2 3 -1
Enter integers to add to list 2 (-1 to end):
1 3 4 -1
List1 before merge sort:
1 2 3
List2 before merge sort:
1 3 4
Merged and sorted list:
1 1 2 3 3 4

15.Merge in Between linked list: You are given two linked lists: list1 and list2 of sizes n and m respectively. Remove list1's nodes from the a th node to the b th node, and put list2 in their place. The blue edges and nodes in the following figure indicate the result:

Code:

```
package com.handson_prblm_sol_10;
import java.util.*;
class Node15 {
  int data;
  Node15 next;
  Node15(int data) {
     this.data = data;
     this.next = null;
  }
}
class Link15{
        Node15 head;
        public void mergeBetween(int a, int b, Link15 list2) {
           if (a > size() || a < 0 || b > size() || b < 0 || a > b) {
              System.out.println("Invalid range");
              return;
           }
           if (list2.head == null) {
              System.out.println("The second list is empty, nothing to merge");
              return;
           }
           Node15 current = head;
           int count = 1;
           while (count < a - 1) {
              current = current.next;
              count++;
           }
           Node15 start = (a > 0) ? current : null;
           while (count < b) {
              current = current.next;
```

```
                count++;
            }
            Node15 end = current.next;
            Node15 list2End = list2.head;
            while (list2End.next != null) {
                list2End = list2End.next;
            }
            list2End.next = end;
            if (a > 0) {
                start.next = list2.head;
            } else {
                head = list2.head;
            }
        }
        public void addNode(int data) {
            Node15 newNode = new Node15(data);
            if (head == null) {
                head = newNode;
                return;
            }
            Node15 last = head;
            while (last.next != null) {
                last = last.next;
            }
            last.next = newNode;
        }
        public int size() {
            int size = 0;
            Node15 current = head;
            while (current != null) {
                size++;
                current = current.next;
            }
            return size;
        }
        public void printList() {
            Node15 current = head;
            while (current != null) {
                System.out.print(current.data + " ");
                current = current.next;
            }
            System.out.println();
        }
    }
public class MergeBet_15 {
    public static void main(String[] args) {
            Scanner sc = new Scanner(System.in);
        Link15 list1 = new Link15();
        Link15 list2 = new Link15();
```

```
            System.out.print("Enter elements 1 (end with -1): ");
            int i;
            while ((i = sc.nextInt()) != -1) {
                list1.addNode(i);
            }
            System.out.print("Enter elements 2 (end with -1): ");
            int j;
            while ((j = sc.nextInt()) != -1) {
                list2.addNode(j);
            }
                    System.out.print("Enter a and b:");
                    int a =  sc.nextInt();
                    int b =sc.nextInt();
                    list1.mergeBetween(a, b, list2);
                    list1.printList();
        }
}
```

Output:
Enter elements 1 (end with -1): 10 1 13 6 9 5 -1
Enter elements 2 (end with -1): 10000 10001
100002 -1
Enter a and b:3 4
10 1 10000 10001 100002 9 5

16.Remove Duplicate Nodes from Sorted Linked 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.

Code:

```
package com.handson_prblm_sol_10;
import java.util.Scanner;
class Node16{
        int data;
        Node16 next;
        Node16(int data)
        {
                this.data=data;
                this.next = null;
        }
}
class Link16{
        Node16 head = null;
        public void addNode(int data)
        {
                Node16 newnode = new Node16(data);
                if(head==null)
                {
                        head = newnode;
```

```
                }
                else
                {
                        Node16 current = head;
                        while(current.next!=null)
                        {
                                current = current.next;
                        }
                        current.next=newnode;
                }
        }
        public void remove() {
    if (head == null) {
        return;
    }
    Node16 current = head;
    Node16 prev = null;
    while (current != null) {
        boolean isDup = false;
        while (current.next != null && current.data == current.next.data) {
            current = current.next;
            isDup = true;
        }
        if (isDup) {
            current = current.next;
            if (prev != null) {
                prev.next = current;
            }
            else {
                head = current;
            }
        } else {
            if (prev == null) {
                prev = head;
            }
            else {
                prev = prev.next;
            }
            current = current.next;
        }
    }
}
        public void display()
        {

                if(head==null)
                        System.out.print("List is empty");
                else
                {
```

```
                    Node16 current = head;
                    while(current!=null)
                    {
                System.out.print(current.data+" ");
                current = current.next;
                    }
                }
        }
}
public class Duplicate_16 {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
            Link16 obj = new Link16();
            System.out.print("Enter elements (end with -1):");
            int i;
            while ((i = sc.nextInt()) != -1) {
               obj.addNode(i);
            }
            obj.remove();
            obj.display();
        }
}
```

Output:
Enter elements (end with -1):1 2 3 3 4 4 5 -1
1 2 5

17.Detect And Remove Cycle Problem statement: You have been given a Singly Linked List of integers, determine if it forms a cycle or not. If there is a cycle, remove the cycle and return the list. A cycle occurs when a node's ‘next’ points back to a previous node in the list.

Code:

```
package com.handson_prblm_sol_10;
import java.util.Scanner;
class Node {
  int data;
  Node next;
  Node(int data) {
     this.data = data;
     this.next = null;
  }
}
public class Detect_17{
  private Node head;
  public Node createNode(int data) {
     return new Node(data);
  }
  public void addNode(int data) {
     Node newNode = createNode(data);
```

```
    if (head == null) {
        head = newNode;
    } else {
        Node current = head;
        while (current.next != null) {
            current = current.next;
        }
        current.next = newNode;
    }
}
public void cycledisplay(int pos) {
        Node dummy = new Node(0);
    Node current = head;
    int c = 0;
    if (head == null) {
        System.out.println("List is empty");
        return;
    }
    while (current != null) {
         if(pos == c) {
                    System.out.print(current.data + " -> ");
                    dummy.data= current.data;
                    pos =-1;
         }
         else {
                    System.out.print(current.data + " -> ");
                    c++;
         }
        current = current.next;
    }
    System.out.println(dummy.data);
}
public int getSize() {
                int size=0;
                Node current = head;
                while(current != null) {
                        size++;
                        current = current.next;
                }
                return size;
        }

public void display() {
                Node current = head;
                if(head == null) {
                        System.out.println("List is empty");
                        return;
                }
                current = head;
```

```
                while(current != null) {
                        System.out.print(current.data+"->");
                        current = current.next;
                }
                System.out.print("-1");

        }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Detect_17 list = new Detect_17();
        System.out.println("Enter elements for the list (-1 to stop):");
        while (true) {
            int data = sc.nextInt();
            if (data == -1) {
                break;
            } else {
                list.addNode(data);
            }
        }
        System.out.println("Enter position");
        int pos = sc.nextInt();
        sc.close();
        System.out.println("Cycle formed");
        list.cycledisplay(pos);
        if(pos<0 || pos>list.getSize()) {
            System.out.println("False");
        }
        else {
            System.out.println("True");
        }
        System.out.println("Cycle remove");
        list.display();

    }
}
```

Output:

```
Enter elements for the list (-1 to stop):
3 2 0 -4 -1
Enter position
1
Cycle formed
3 -> 2 -> 0 -> -4 -> 2
True
Cycle remove
3->2->0->-4->-1
```

18.Swap Node in pairs : Given a linked list, swap every two adjacent nodes and return its head. You must solve the problem without modifying the values in the list's nodes (i.e., only nodes themselves may be changed.)

Code:

```
package com.handson_prblm_sol_10;
import java.util.*;
public class SwapPair_18 {
  class Node {
     int data;
     Node next;
     Node(int x) {
        this.data = x;
        this.next = null;
     }
  }
  Node head = null;
  public void addNode(int data) {
     Node newnode = new Node(data);
     if (head == null) {
        head = newnode;
     } else {
        Node current = head;
        while (current.next != null) {
           current = current.next;
        }
        current.next = newnode;
     }
  }
  public void swap(Node current) {
     while (current != null && current.next != null) {
        int temp = current.data;
        current.data = current.next.data;
        current.next.data = temp;
        current = current.next.next;
     }
  }
  public void display() {
     if (head == null) {
        System.out.println("The list is empty");
     } else {
        Node current = head;
        while (current != null) {
           System.out.print(current.data + " -> ");
           current = current.next;
        }
        System.out.println("null");
     }
  }
```

```
  public static void main(String args[]) {
     Scanner sc = new Scanner(System.in);
     SwapPair_18 s = new SwapPair_18();
     System.out.println("Enter the size of the list: ");
     int n = sc.nextInt();
     System.out.println("Enter the values of the list: ");
     for (int i = 0; i < n; i++) {
        s.addNode(sc.nextInt());
     }
     s.display();
     System.out.println("After swapping: ");
     s.swap(s.head);
     s.display();
     sc.close();
  }
}
```

Output:
Enter the size of the list:
6
Enter the values of the list:
1 2 3 4 5 6
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null
After swapping:
2 -> 1 -> 4 -> 3 -> 6 -> 5 -> null

19.Partition Linked List by Value x : Given the head of a linked list and a value x, partition it such that all nodes less than x come before

Code:

```
package com.handson_prblm_sol_10;
import java.util.*;
class Node19{
        int data;
        Node19 next;
        Node19(int data)
        {
                this.data=data;
                this.next=null;
        }
}
class Link19{
        Node19 head = null;
        Node19 current = head;
         public void addNode(int data) {
              Node19 newnode = new Node19(data);
              if (head == null) {
                 head = newnode;
              } else {
```

```
            Node19 current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newnode;
        }
    }
    public void display(int x) {
             Link19 ans1 = new Link19();
             Link19 ans2 = new Link19();
        Node19 current = head;
        if(head==null) {
            System.out.print("List is empty");
        } else {
            while(current != null) {
             if(current.data<x) {
                ans1.addNode(current.data);
             }
             else
             {
                        ans2.addNode(current.data);
             }

                       current = current.next;
            }
            ans1.dis();
            ans2.dis();
        }
    }
    public void dis()
    {
                if (head == null) {
                    System.out.println("List is empty");
                    return;
                }
                else {
                Node19 current = head;
                while (current != null) {
                    System.out.print(current.data + " ");
                    current = current.next;
                }
               }}
    }

public class Partition_19 {
    public static void main(String[] args) {
             Scanner sc = new Scanner(System.in);
        Link19 obj = new Link19();
        Link19 ans1 = new Link19();
```

```
                Link19 ans2 = new Link19();
                System.out.print("Enter elements (end with -1): ");
                int i;
                while ((i = sc.nextInt()) != -1) {
                    obj.addNode(i);
                }
                System.out.print("Enter x:");
                int x = sc.nextInt();
                obj.display(x);

        }
}
```

Output:
Enter elements (end with -1): 1 4 3 2 5 2 -1
Enter x:3
1 2 2 4 3 5

20.Insert into a sorted circular linked list Given a Circular Linked List node, which is sorted in non-descending order, write a function to insert a value insertVal into the list such that it remains a sorted circular list. The given node can be a reference to any single node in the list and may not necessarily be the smallest value in the circular list. If there are multiple suitable places for insertion, you may choose any place to insert the new value. After the insertion, the circular list should remain sorted. If the list is empty (i.e., the given node is null), you should create a new single circular list and return the reference to that single node. Otherwise, you should return the originally given node.

Code:

```
package com.handson_prblm_sol_10;
import java.util.Scanner;
class Node20{
        int data;
        Node20 next;
        Node20(int data)
        {
                this.data=data;
                this.next=null;
        }
}
class Link20
{
        Node20 head = null;
          public void create(int data) {
            Node20 newnode = new Node20(data);
            if (head == null) {
                head = newnode;
                newnode.next = head;
            } else {
                Node20 current = head;
                while (current.next != head) {
```

```
                current = current.next;
            }
            current.next = newnode;
            newnode.next = head;
        }
    }
    public void insert(int data) {
        Node20 newnode = new Node20(data);
        if (head == null) {
            head = newnode;
            newnode.next = head;
            return;
        }
        Node20 current = head;

        while (true) {
            if (current.data <= data && data <= current.next.data) {
                break;
            }
            if (current.data > current.next.data) {
                if (data <= current.next.data || data >= current.data) {
                    break;
                }
            }
            current = current.next;
            if (current == head) {
                break;
            }
        }
        newnode.next = current.next;
        current.next = newnode;
    }


    public void display() {
        if (head == null) {
            return;
        }
        Node20 current = head;
        System.out.print(current.data + "->");
        current = current.next;
        while (current != head) {
            System.out.print(current.data + "->");
            current = current.next;
        }
        System.out.println();
        }
}
public class Insert_20 {
```

```
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
    Link20 obj = new Link20();
    int i;
    System.out.println("Enter integers to add to the list (-1 to end):");
    while ((i = sc.nextInt()) != -1) {
       obj.create(i);
    }
    System.out.print("Enter ele:");
    obj.insert(sc.nextInt());
    obj.display();
        }
}
```

Output:
Enter integers to add to the list (-1 to end):
3 4 2 1 -1
Enter ele:5
3->4->5->2->1->

21.Remove duplicates from a sorted Doubly Linked List Problem statement : A doubly-linked list is a data structure that consists of sequentially linked nodes, and the nodes have reference to both the previous and the next nodes in the sequence of nodes. You are given a sorted doubly linked list of size 'n'. Remove all the duplicate nodes present in the linked list.

Code:

```
package com.handson_prblm_sol_10;
import java.util.*;
class Node21{
        int data;
        Node21 next;
        Node21 prev;
        Node21(int data)
        {
                this.data=data;
                this.next=null;
                this.prev=null;
        }
}
class Link21{

        Node21 head = null;
        public void create(int data) {
    Node21 newnode = new Node21(data);
    if (head == null) {
       head = newnode;
    } else {
       Node21 current = head;
       while (current.next != null) {
```

```
            current = current.next;
        }
        current.next = newnode;
        newnode.prev = current;
    }
  }
        public void remove()
        {
                Node21 current = head;
                while(current!=null && current.next!=null)
                {
                        if(current.data==current.next.data)
                        {
                                current.prev.next=current.next;
                                current.next.prev=current.prev;
                        }
                        current=current.next;
                }
        }
          public void display() {
              Node21 current = head;
              while (current != null) {
                 System.out.print(current.data + "->");
                 current = current.next;
              }
              System.out.println("null");
           }
}
public class DupliDoubly_21 {
        public static void main(String[] args) {
                 Scanner sc = new Scanner(System.in);
      Link21 obj = new Link21 ();
      int i;
     System.out.println("Enter integers to add to the list (-1 to end):");
     while ((i = sc.nextInt()) != -1) {
        obj.create(i);
     }
    obj.remove();
    obj.display();
        }
}
```

Output:
Enter integers to add to the list (-1 to end):
1 2 2 2 3 -1
1->2->3->null