SINGLE LINKED LIST

```
package possll;
import java.util.Scanner;

class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}
public class PosSll {
                private Node head;

            public void insertAtFront(int data) {
                Node newNode = new Node(data);
                newNode.next = head;
                head = newNode;
                System.out.println(data + " inserted at the front.");
            }

            public void insertAtEnd(int data) {
                Node newNode = new Node(data);
                if (head == null) {
```

```
            head = newNode;
            System.out.println(data + " inserted at the end.");
            return;
        }
        Node current = head;
        while (current.next != null) {
            current = current.next;
        }
        current.next = newNode;
        System.out.println(data + " inserted at the end.");
    }

    public void insertAtPos(int data) {
        Node newNode = new Node(data);
        Scanner sc= new Scanner(System.in);
        Node prev=null;
        Node current= head;
        int i=1;
        System.out.println("Enter the position to be inserted");
        int pos=sc.nextInt();
        for(i=1;i<pos;i++)
        {
        prev=current;
        current=current.next;
        }
        newNode.next=current;
        prev.next=newNode;
```

```
        return;
        }

    public void deleteAtFront() {
        if(head==null){
        System.out.println("List is empty");
        return;
        }
        Node cur = head;
        head = head.next;
        //cur.next=null;
        System.out.println("Data deleted is: "+cur.data);
        }
        // Deletion at the end
        public void deleteAtEnd() {
        if(head==null){
        System.out.println("List is empty");
        return;
        }
        Node cur = head;
        if(cur.next == null){
        head = null;
        return;
        }
        Node prev=null;
        while(cur.next != null){
        prev = cur;
```

```
cur = cur.next;
}
prev.next = null;
System.out.println("Data deleted is: "+cur.data);
}

public void deleteAfterData(int data) {
if(head==null){
System.out.println("List is empty");
return;
}
Node cur = head;
while(cur.next != null && cur.data != data)
cur = cur.next;
if(cur.next != null){
        System.out.println("Data deleted is: "+cur.next.data);
        cur.next = cur.next.next;
}
else if(cur.data != data)
System.out.println("Node is not present in the list");
else
System.out.println("Deletion not possible this is the last node");
}
public boolean search(int key) {
 Node temp = head;
 while (temp != null) {
    if (temp.data == key) {
```

```
                return true; // Key found in the list
            }
            temp = temp.next;
        }
        return false; // Key not found in the list
    }
    public void display() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " -> ");
            current = current.next;
        }
        System.out.println("null");
    }

    public static void main(String []args) {
        PosSll list = new PosSll();
        Scanner sc = new Scanner(System.in);
        int op, item;
        System.out.println("Implementation of Singly Linked List");
        while(true) {
        System.out.print("1.Insertion at the begining\n2.Insertion at the end\n3.Insertion at specified position(data)\n4.Deletion at the begining\n5.Deletion at the end\n6.Deletion after a given node(data)\n7.Search for key 8.Display list\n9.Exit\n");
        op = sc.nextInt();
        switch(op) {
        case 1:
```

```
System.out.print("Enter value: ");
item = sc.nextInt();
list.insertAtFront(item);
list.display();
break;
case 2:
System.out.print("Enter value: ");
item = sc.nextInt();
list.insertAtEnd(item);
list.display();
break;
case 3:         System.out.print("Enter the value to insert: ");
item = sc.nextInt();
list.insertAtPos(item);
list.display();
break;
case 4:
list.deleteAtFront();
list.display();
break;
case 5:
list.deleteAtEnd();
list.display();
break;
case 6:
System.out.print("Enter data after which you want to do deletion: ");
item = sc.nextInt();
```

```
                list.deleteAfterData(item);
                list.display();
                break;
                case 7:System.out.println("Enter the key to be searched");
                int keyToSearch=sc.nextInt();
               if (list.search(keyToSearch)) {
                  System.out.println("Key " + keyToSearch + " found in the linked list.");
               } else {
                  System.out.println("Key " + keyToSearch + " not found in the linked list.");
               }
                case 8:
                list.display();
                break;
                case 9:
                System.exit(1);
                }
                }
                }
                }
```

STACK IMPLEMENTATION USING SINGLE LINKED LIST:

```
package possll;
import java.util.Scanner;
class Node {
    int data;
    Node next;
    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}
public class Stack {
                private Node head;

                public void insertAtEnd(int data) {
                Node newNode = new Node(data);
                if (head == null) {
                    head = newNode;
                    System.out.println(data + " inserted at the end.");
                    return;
                }
                Node current = head;
                while (current.next != null) {
                    current = current.next;
                }
                current.next = newNode;
```

```
        System.out.println(data + " inserted at the end.");
    }
                // Deletion at the end
        public void deleteAtEnd() {
        if(head==null){
        System.out.println("List is empty");
        return;
        }
        Node cur = head;
        if(cur.next == null){
        head = null;
        return;
        }
        Node prev=null;
        while(cur.next != null){
        prev = cur;
        cur = cur.next;
        }
        prev.next = null;
        System.out.println("Data deleted is: "+cur.data);
        }
public void display() {
        Node current = head;
        while (current != null) {
           System.out.print(current.data + " -> ");
           current = current.next;
        }
```

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

public static void main(String []args) {
    PosSll list = new PosSll();
    Scanner sc = new Scanner(System.in);
    int op, item;
    System.out.println("Implementation of Singly Linked List");
    while(true) {
     System.out.print("1. Insertion at the end 2.Deletion at the end\n
    3.Display list\n4.Exit\n");
    op = sc.nextInt();
    switch(op) {
    case 1:  System.out.print("Enter value: ");
    item = sc.nextInt();
    list.insertAtEnd(item);
    list.display();
    break;
    case 2:
    list.deleteAtEnd();
    list.display();
    break;
    case 3:
    list.display();
    break;
    case 4:
    System.exit(1);
    }
```

```
        }
        }
        }
```

QUEUE IMPLEMENTATION USING SINGLE LINKED LIST:

```
package possll;
import java.util.Scanner;

class Node {
   int data;
   Node next;
   public Node(int data) {
      this.data = data;
      this.next = null;
   }
}
public class Queue {
        private Node head;

    public void insertAtEnd(int data) {
       Node newNode = new Node(data);
       if (head == null) {
          head = newNode;
          System.out.println(data + " inserted at the end.");
          return;
       }
       Node current = head;
```

```
    while (current.next != null) {
        current = current.next;
    }
    current.next = newNode;
    System.out.println(data + " inserted at the end.");
}
public void deleteAtFront() {
      if(head==null){
      System.out.println("List is empty");
      return;
      }
      Node cur = head;
      head = head.next;
      //cur.next=null;
      System.out.println("Data deleted is: "+cur.data);
      }

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

public static void main(String []args) {
```

```
PosSll list = new PosSll();
Scanner sc = new Scanner(System.in);
int op, item;
System.out.println("Implementation of Singly Linked List");
while(true) {
System.out.print("1. Insertion at the end\2. Deletion at the begining\n3.Display list\n4.Exit\n");
op = sc.nextInt();
switch(op) {
case 1:System.out.print("Enter value: ");
item = sc.nextInt();
list.insertAtEnd(item);
list.display();
break;
case 2:
list.deleteAtEnd();
list.display();
break;
case 3:
list.display();
break;
case 4:
System.exit(1);
}
}
}
}
```

CIRCULAR LINKED LIST:

```
package circularlinkedlist;
import java.util.*;
import java.util.Scanner;
class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}
public class CLLPOS {

                                private Node head;

                                public void insertAtFront(int data) {
                                    Node newNode = new Node(data);
                                    if (head == null) {
                                        head = newNode;
                                        head.next = head;
                                    } else {
                                        Node cur = head;
                                        while (cur.next != head) {
                                                cur = cur.next;
                                        }
```

```
            cur.next = newNode;
            newNode.next = head;
            head = newNode;
        }
    }

    public void insertAtEnd(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            head.next = head;
        } else {
            Node cur = head;
            while (cur.next != head) {
                cur = cur.next;
            }
            cur.next = newNode;
            newNode.next = head;
        }
    }

public void insertAtPos(int data) {
    Node newNode = new Node(data);
    Scanner sc=new Scanner(System.in);
    if (head == null) {
        head = newNode;
        head.next = head;
```

```
        } else {
                System.out.print("Enter the position to insert: ");
                        int pos= sc.nextInt();
                        Node current = head;
                for (int i = 1; i < pos-1; i++) {
                current = current.next;
            }
            newNode.next = current.next;
            current.next = newNode;
        }
        }
```

public void deleteAtFront() {

```
        if (head == null) {
            System.out.println("List is empty.");
            return;
        }

        Node cur = head;
        if (cur.next == head) {
            head = null;
        } else {
            while (cur.next != head) {
                cur = cur.next;
            }
            cur.next = head.next;
            head = head.next;
```

```
    }
}

     // Deletion at the end
public void deleteAtEnd() {
    if (head == null) {
        System.out.println("List is empty.");
        return;
    }

    Node cur = head;
    Node prev = null;
    if (cur.next == head) {
        head = null;
        return;
    }
    else {
      while (cur.next != head) {
        prev = cur;
        cur = cur.next;
    }
      prev.next = head;
    }
}

public void deleteData(int key) {
    //list is empty
```

```
  if (head == null) {
    System.out.println("List is empty.");
    return;
  }

  Node cur = head;
  Node prev = null;

  //Single node
  if (cur.next == head && cur.data==key) {
    head = null;
    return;
  }

  while (cur.data != key && cur.next!=head) {
    prev = cur;
    cur = cur.next;
  }
  //data not found
  if(cur.data!=key) {
    System.out.println("Given node is not found");
    return;
}
  //more than one node, check if it first node
  else if (cur == head) {
    prev = head;
    while (prev.next != head)
```

```
            prev = prev.next;
        head = cur.next;
        prev.next = head;
    }

    //check if node is last node
    else if (cur.next == head) {
        prev.next = head;
    }
    //node at mid
    else {
        prev.next = cur.next;
    }
    return;
}

public void display() {
      if (head == null) {
        System.out.println("List is empty.");
        return;
    }
    Node current = head;
     do{
        System.out.print(current.data + " -> ");
        current = current.next;
    }while (current != head);
```

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

    public static void main(String []args) {
        CLLPOS list = new CLLPOS();
        Scanner sc = new Scanner(System.in);
        int op, item;
        System.out.println("Implementation of Circular Linked List");
        while(true) {
        System.out.print("1.Insertion at the begining\n2.Insertion at the end\n3.Insertion at given Position (data)\n4.Deletion at the begining\n5.Deletion at the end\n6.Deletion of a given node(data)\n7.Display list\n8.Exit\n");
        op = sc.nextInt();
        switch(op) {
        case 1:
        System.out.print("Enter value: ");
        item = sc.nextInt();
        list.insertAtFront(item);
        list.display();
        break;
        case 2:
        System.out.print("Enter value: ");
        item = sc.nextInt();
        list.insertAtEnd(item);
        list.display();
        break;
        case 3:         System.out.print("Enter the value to insert: ");
```

```
item = sc.nextInt();
list.insertAtPos(item);
list.display();
break;
case 4:
list.deleteAtFront();
list.display();
break;
case 5:
list.deleteAtEnd();
list.display();
break;
case 6:
System.out.print("Enter data you want to do deletion: ");
item = sc.nextInt();
list.deleteData(item);
list.display();
break;
case 7:
list.display();
break;
case 8:
System.exit(1);
}
}
}
}
```

FACTORIAL OF A NUMBER USING RECURSION:

❖ GENERAL METHOD

```
package recursion;
public class Factorial {

            // Recursive factorial function
        public static int recursiveFactorial(int n) {
            if (n == 0 || n == 1) {
                return 1;
            } else {
                return n * recursiveFactorial(n - 1);
            }
        }

        // Main method to test the recursive and stack-based factorial functions
        public static void main(String[] args) {
            int num = 5; // Number for which factorial is calculated

            // Calculate and print factorial using recursion
            System.out.println("Factorial of " + num + " using recursion: " +
recursiveFactorial(num));

        }
    }
```

❖ USING STACK

```
package recursion;
import java.util.Stack;
public class Factorial {

        // Factorial with stack function
        public static int factorialWithStack(int n) {
            // Create a stack to simulate recursion
            Stack<Integer> stack = new Stack<>();
            stack.push(n);  // Push the initial value onto the stack
            int result = 1; // Initialize the result variable to 1

            // Iterate until the stack is empty
            while (!stack.isEmpty()) {
```

```
            int num = stack.pop(); // Pop a number from the stack

            // Multiply the result by the popped number
            result *= num;

            // If the popped number is greater than 1, push (num - 1) onto the
stack
            if (num > 1) {
                stack.push(num - 1);
            }
        }
        return result; // Return the factorial result
    }

    // Main method to test the recursive and stack-based factorial functions
    public static void main(String[] args) {
        int num = 5; // Number for which factorial is calculated

        // Calculate and print factorial using stack
        System.out.println("Factorial of " + num + " using stack: " +
factorialWithStack(num));
    }
}
```

TOWER OF HANOI USING RECURSION:

- **TOWER OF HANOI (General Method)**

```
package recursion;

public class TOH {

        public static void main(String[] args) {

    int numberOfDisks = 3;

    char source = 'A';

    char auxiliary = 'B';

    char destination = 'C';
```

```
        System.out.println("Steps to solve Tower of Hanoi with " +
numberOfDisks + " disks:");

        solveTowerOfHanoi(numberOfDisks, source, auxiliary, destination);

    }


    public static void solveTowerOfHanoi(int n, char source, char
auxiliary, char destination) {

        if (n == 1) {

            System.out.println("Move disk from " + source + " to " +
destination);

            return;

        } else {

            solveTowerOfHanoi(n - 1, source, destination, auxiliary);

            solveTowerOfHanoi(1, source, auxiliary, destination );

            solveTowerOfHanoi(n - 1, auxiliary, source, destination);

        }

    }
```

❖ TOWER OF HANOI (Using Stack)

```
package recursion;
import java.util.Stack;
public class TowerOfHanoi {
        public static void towerOfHanoi(int numDisks, Stack<Integer>
source, Stack<Integer> auxiliary, Stack<Integer> destination) {
            if (numDisks == 1) {
                destination.push(source.pop());
                System.out.println("Move disk 1 from source to destination");
                return;
            }

            towerOfHanoi(numDisks - 1, source, destination, auxiliary);
            destination.push(source.pop());
            System.out.println("Move disk " + numDisks + " from source to
destination");
            towerOfHanoi(numDisks - 1, auxiliary, source, destination);
        }

        public static void main(String[] args) {
            int numDisks = 3;
            Stack<Integer> source = new Stack<>();
            Stack<Integer> auxiliary = new Stack<>();
            Stack<Integer> destination = new Stack<>();

            // Initialize source stack with disks
            for (int i = numDisks; i >= 1; i--) {
                source.push(i);
            }

            System.out.println("Initial configuration:");
            System.out.println("Source: " + source);
            System.out.println("Auxiliary: " + auxiliary);
            System.out.println("Destination: " + destination);

            // Solve Tower of Hanoi problem
            towerOfHanoi(numDisks, source, auxiliary, destination);

            System.out.println("Final configuration:");
            System.out.println("Source: " + source);
```

```
        System.out.println("Auxiliary: " + auxiliary);
        System.out.println("Destination: " + destination);
    }
}
```