**STACK**

class Stack {

    private ListNode top = null;

    private int length = 0;

    private class ListNode {

        private int data;

        private ListNode next;

        public ListNode(int data) {

            this.data = data;

        }

    }

    public boolean isEmpty() {

        return length == 0;

    }

    public void len() {

        System.out.println(length);

    }

    public void push(int data) {

        ListNode temp = new ListNode(data);

        temp.next = top;

        top = temp;

        length++;

    }

    public void pop() {

        if (!isEmpty()) {

             int res = top.data;

             top = top.next;

            length--;

            System.out.println(res);

        }

    }

    public void peek() {

        if (!isEmpty()) {

            System.out.println(top.data);

        }

    }

}

public class Main {

    public static void main(String[] args) {

        Stack s = new Stack();

        s.len();

        s.push(10);

        s.push(20);

        s.push(30);

        s.len();

        s.peek();

        s.pop();

        s.len();

        s.peek();

    }

}

**QUEUE**

class Queue {

    private ListNode front = null;

    private ListNode rear = null;

    private int length = 0;

    private class ListNode {

        private int data;

        private ListNode next;

        public ListNode(int data) {

            this.data = data;

            this.next = null;

        }

    }

    public boolean isEmpty() {

        return length == 0;

    }

    public void len() {

        System.out.println(length);

    }

    public void enqueue(int data) {

        ListNode temp = new ListNode(data);

        if(isEmpty()) {

            front = temp;

        } else {

            rear.next = temp;

        }

        rear = temp;

        length++;

    }

    public void dequeue() {

        if(!isEmpty()) {

            int res = front.data;

            front = front.next;

            if (front == null) {

                rear = null;

            }

            length--;

            System.out.println(res);

        }

    }

    public void peek() {

        System.out.println(front.data);

    }

}

public class Main {

    public static void main(String[] args) {

        Queue q = new Queue();

        q.len();

        q.enqueue(10);

        q.enqueue(20);

        q.enqueue(30);

        q.len();

        q.peek();

        q.dequeue();

        q.len();

        q.peek();

    }

}

**SINGLY LINKED LIST**

class LinkedList {

    private ListNode head = null;

    private int length = 0;

    private class ListNode {

        private int data;

        private ListNode next;

        public ListNode(int data) {

            this.data = data;

            this.next = null;

        }

    }

    public boolean isEmpty() {

        return length == 0;

    }

    public void len() {

        System.out.println(length);

    }

    public void insert(int data) {

        ListNode temp = new ListNode(data);

        if(head != null) {

            ListNode current = head;

            while (current.next != null) {

                current = current.next;

            }

            current.next = temp;

            length++;

        } else {

            head = temp;

        }

    }

    public void delete() {

        if (head == null || head.next == null) {

            System.out.println("Linked list is empty!!");

        }

        ListNode current = head;

        ListNode previous = null;

        while(current.next != null) {

            previous = current;

            current = current.next;

        }

        previous.next = null;

        System.out.println("Deleted value: " + current.data);

    }

    public void display() {

        if (!isEmpty()) {

            ListNode current = head;

        while (current != null) {

            System.out.print(current.data + "-->");

            current = current.next;

        }

        System.out.print("null");

        System.out.println();

        } else {

            System.out.println("head --> null");

        }

    }

}

public class Main {

    public static void main(String[] args) {

        LinkedList ll = new LinkedList();

        ll.len();

        ll.display();

        ll.insert(10);

        ll.insert(20);

        ll.insert(30);

        ll.len();

        ll.display();

        ll.delete();

        ll.len();

        ll.display();

    }

}

**TRAINER IMPLEMENTATION**

class Queue {

int[] arr;

int max\_size, front, rear;

public Queue(int size) {

max\_size = size;

front = 0;

rear = 0;

}

void enque(int n) {

arr[rear] = n;

rear++;

}

int deque() {

int k;

k = arr[front];

for(int i=0;i<rear-1;i++){

arr[i] = arr[i+1];

}

rear--;

System.out.println(k);

return k;

}

boolean isEmpty() {

if(front==rear) {

return true;

}

else {

return false;

}

}

boolean isFull(){

if(rear==max\_size) {

return true;

}

else {

return false;

}

}

}

public class Main {

public static void main(String args[]) {

}

}

---------------------------------------------------------------------------------------------------------------------------------------

class Stack {

int[] arr;

int max\_size,top;

public Stack(int n) {

arr = new int[n];

max\_size = n;

top = -1;

}

boolean isFull(){

if(top==(max\_size-1)){

return true;

}

else {

return false;

}

}

boolean isEmpty() {

if(top==-1)

{

return true;

}

else {

return false;

}

}

void push(int k){

if(!isFull()){

top++;

arr[top] = k;

System.out.println("Element PUshed");

}

else

{

System.out.println("Cannot push further, stack is full");

}

}

int pop() {

int k;

if(!isEmpty())

{

k = arr[top];

top--;

System.out.println(k);

return k;

}

else {

System.out.println("Stack is empty, nothing to pop");

return -1;

}

}

int peek() {

if(!isEmpty()){

return arr[top];

}

else {

System.out.println("Stack is empty, nothing to peek");

return -1;

}

}

}

public class Main {

public static void main(String args[]) {

Stack st = new Stack(3);

st.push(5);

st.push(8);

st.push(3);

st.push(1);

st.pop();

st.pop();

st.pop();

st.pop();

}

}

---------------------------------------------------------------------------------------------------------------------------------------.

class Car {

String color;

int max\_speed;

int curr\_speed;

public Car(String car\_color, int car\_max\_speed) {

color = car\_color;

max\_speed = car\_max\_speed;

curr\_speed = 0;

}

int accelerate(int accelerate\_by) {

curr\_speed = curr\_speed + accelerate\_by;

System.out.println("Vroom....");

return curr\_speed;

}

void applyBrakes() {

System.out.println("Krrrr......");

}

}

public class Main {

public static void main(String args[]) {

Car c1 = new Car("RED",180);

int new\_speed;

System.out.println("Top Speed : "+(c1.max\_speed));

System.out.println("Color : "+c1.color);

new\_speed = c1.accelerate(10);

System.out.println("Current Speed : "+new\_speed);

new\_speed = c1.accelerate(10);

System.out.println("Current Speed : "+new\_speed);

new\_speed = c1.accelerate(10);

System.out.println("Current Speed : "+new\_speed);

new\_speed = c1.accelerate(10);

System.out.println("Current Speed : "+new\_speed);

new\_speed = c1.accelerate(10);

System.out.println("Current Speed : "+new\_speed);

c1.applyBrakes();

}

}

import java.util.Scanner;

class Node {

int data;

Node next;

public Node(int n){

data = n;

}

}

class LinkedList {

Node head;

void insertAtFirst(int data) {

if(head==null){

head = new Node(data);

}

else {

Node new\_node = new Node(data);

new\_node.next = head;

head = new\_node;

}

}

void deleteAtFirst() {

Node new\_head = head.next;

head = new\_head;

}

}

class Stack {

LinkedList ll;

public Stack()

{

ll = new LinkedList();

}

void push(int n){

ll.insertAtFirst(n);

}

int pop(){

int k = ll.head.data;

ll.deleteAtFirst();

return k;

}

void print(){

Node iterator = ll.head;

while(iterator.next != null){

System.out.println(iterator.data);

iterator = iterator.next;

}

}

}

public class Main {

public static void main(String[] args) {

Stack s = new Stack();

Scanner sc = new Scanner(System.in);

s.push(sc.nextInt());

s.push(sc.nextInt());

s.push(sc.nextInt());

s.push(sc.nextInt());

s.push(sc.nextInt());

s.pop();

s.pop();

s.print();

}

}

import java.util.Scanner;

public class StackUsingLinkedList {

private LinkedList<Integer> stack;

public StackUsingLinkedList() {

stack = new LinkedList<>();

}

public void push(int value) {

stack.push(value);

}

public int pop() {

return stack.pop();

}

public int peek() {

return stack.peek();

}

public boolean isEmpty() {

return stack.isEmpty();

}

public int size() {

return stack.size();

}

public void display() {

if (isEmpty()) {

System.out.println("Stack is empty.");

} else {

System.out.println("Stack elements:");

for (int i = stack.size() - 1; i >= 0; i--) {

System.out.println(stack.get(i));

}

}

}

public static void main(String[] args) {

Scanner sc=new Scanner (System.in);

System.out.println("Enter 5 numbers: ");

StackUsingLinkedList stack = new StackUsingLinkedList();

for (int i = 0; i < 5; i++) {

int input = sc.nextInt() ;

stack.push(input);

}

stack.display();

}

}

**FIBONACCI SERIES**

public class Main {

    public static int fibo (int n) {

        if (n <= 1) {

            return n;

        }

        return fibo (n - 1) + fibo (n - 2);

    }

    public static void main(String[] args) {

       int n = 0;

       int m = 1;

       System.out.print(n + " " + m);

       for (int i = 3; i <= 10; i++) {

           int o = n + m;

           System.out.print(" " + o);

            n = m;

            m = o;

       }

       System.out.println();

       for (int i = 0; i < 10; i++) {

           System.out.print(fibo(i) + " ");

       }

    }

}

**REVERSE A STRING USING RECURSION**

public class ReversingString {

public static void main(String[] args) {

String s = "hello";

System.out.println(s);

System.out.println(reverse(s));

}

public static String reverse(String s) {

if (s.isEmpty()) {

return "";

}

char firstChar = s.charAt(0);

String remains = s.substring(1);

String reversed = reverse(remains);

return reversed+firstChar;

}

}

**1. Stack**

arr = [\_,\_,\_,\_,\_,\_]

maxSize = 6

top = -1

operations

push(n)

if not isFull() {

    top = top + 1

    arr[top] = n

}

pop

if not isEmpty() {

n = arr[top]

top = top - 1

return n

}

peek

if not  IsEmpty

return arr[top]

isFull()

if(top==(maxSize-1)){

    return True

}

else {

    return False

}

isEmpty()

if(top==-1){

    return True

}

else {

    return False

}

2. Queue

3. Linkedlists

4. Recursion

**VALID PARENTHESES**

import java.util.\*;

public class Main {

    public static void main(String[] args) {

        String s = "((1+3)\*(9\*7+16)";

        Stack<Character> stack = new Stack<>();

        for (int i = 0; i < s.length(); i++) {

            if (s.charAt(i) == '(' || s.charAt(i) == ')') {

                if (s.charAt(0) != '(') {

                    stack.push(s.charAt(0));

                    break;

                } else {

                    if (s.charAt(i) == '(') {

                        stack.push(s.charAt(i));

                    } else if (s.charAt(i) == ')') {

                        stack.pop();

                    }

                }

            }

        }

        if (stack.isEmpty()) {

            System.out.println("Valid");

        } else {

            System.out.println("Not valid");

        }

    }

}

LINEAR SEARCH USING STACK

import java.util.\*;

public class Main {

    public static void main(String[] args) {

        String s = "Hello";

        char t = 'o';

        int index = - 1;

        Stack<Character> stack = new Stack<>();

        for (char c : s.toCharArray()) {

            stack.push(c);

        }

        for (int i = stack.size() - 1; i >= 0; i--) {

            if (stack.elementAt(i) == t) {

                index = i;

                break;

            }

        }

        if (index < 0) {

            System.out.println("Element not found");

        } else {

            System.out.println("Element found at index: " + index);

        }

    }

}