public class Util {

public static void bubbleSort(int [] arr){

int passes = arr.length - 1;

int comps = arr.length - 1;

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

int count = 0;

for(int j = 0; j < (comps - i); j++) {

if(arr[j] > arr[j+1]) {

int temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

count++;

}

}

if(count == 0) {

System.out.println("Passes: " + (i+1));

return;

}

}

}

public static void selectionSort(int [] arr) {

for(int i = 0; i < arr.length - 1; i++) {

int min = i;

for(int j = i + 1; j < arr.length; j++) {

if(arr[j] < arr[min]) {

min = j;

}

}

if(min != i) {

int temp = arr[i];

arr[i] = arr[min];

arr[min] = temp;

}

}

}

public static void insertionSort(int [] arr) {

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

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

if(arr[i] < arr[j]) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

public static void quickSort(int [] arr, int left, int right) {

if(left >= right) {

return;

}

int pivotLoc = partition(arr, left, right);

quickSort(arr, left, pivotLoc - 1);

quickSort(arr, pivotLoc + 1, right);

}

private static int partition(int [] arr, int left, int right) {

int pivot = arr[left];

while(left < right) {

while (arr[right] > pivot && left != right) {

right--;

}

/\*if(left == right) {

arr[left] = pivot;

return left;

}\*/

if(left != right) {

arr[left] = arr[right];

left++;

}

while (arr[left] < pivot && left != right) {

left++;

}

if(left != right) {

arr[right] = arr[left];

right--;

}

}

arr[left] = pivot;

return right;

}

public static void mergeSort(int [] arr, int left, int right) {

if(left >= right) {

return;

}

int mid = (left + right) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid+1, right);

merge(arr, left, mid, mid+1, right);

}

private static void merge(

int [] arr,

int leftStart,

int leftEnd,

int rightStart,

int rightEnd

) {

int noOfElements = rightEnd - leftStart + 1;

int [] tempArr = new int[noOfElements];

int tempPos = -1;

while((leftStart <= leftEnd) && (rightStart <= rightEnd)) {

if (arr[leftStart] < arr[rightStart]) {

//shift ls element to temp

tempArr[++tempPos] = arr[leftStart];

leftStart++;

} else {

//shift rs element to temp

tempArr[++tempPos] = arr[rightStart];

rightStart++;

}

}

while(leftStart <= leftEnd) {

tempArr[++tempPos] = arr[leftStart];

leftStart++;

}

while(rightStart <= rightEnd) {

tempArr[++tempPos] = arr[rightStart];

rightStart++;

}

for(int i = 1; i <= noOfElements; i++) {

arr[rightEnd] = tempArr[tempPos];

tempPos--;

rightEnd--;

}

}

public static void heapSort(int [] arr, int len) {

if(len <= 1) {

return;

}

for(int i = len - 1; i > 0; i--) {

if(arr[i] > arr[(i-1)/2]) {

int temp = arr[i];

arr[i] = arr[(i-1)/2];

arr[(i-1)/2] = temp;

}

}

int temp = arr[0];

arr[0] = arr[len-1];

arr[len-1] = temp;

heapSort(arr, len-1);

}

}

public class Main {

public static void main(String[] args) {

int [] arr = {20, 9, 60, 5, 23, 30, 12, 16, 18, 55, 43 };

//Util.bubbleSort(arr);

//Util.selectionSort(arr);

//Util.insertionSort(arr);

//Util.quickSort(arr, 0, arr.length-1);

//Util.heapSort(arr, arr.length);

Util.mergeSort(arr, 0, arr.length-1);

for(int val : arr) {

System.out.print(val + " ");

}

System.out.println();

}

}

public class Node {

private int data;

private Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

public int getData() {

return data;

}

public void setData(int data) {

this.data = data;

}

public Node getNext() {

return next;

}

public void setNext(Node next) {

this.next = next;

}

}

import java.util.Stack;

public class LinkedList {

private Node head;

/\*private int size = 0;

private Node last;\*/

public LinkedList() {

head = null;

}

public boolean insert(int data) {

Node newNode = new Node(data);

if(newNode == null) {

return false;

}

if(head == null) {

this.head = newNode;

return true;

}

Node last = head;

while(last.getNext() != null) {

last = last.getNext();

}

last.setNext(newNode);

return true;

}

public void display() {

Node temp = head;

while(temp != null) {

System.out.print(temp.getData() + " ");

temp = temp.getNext();

}

System.out.println();

}

public void display(Node head) {

if(head == null) {

return;

}

System.out.print(head.getData() + " ");

display(head.getNext());

}

public Node getHead() {

return head;

}

public void displayRev() {

Node temp = head;

Stack<Node> s = new Stack<>();

while(temp != null) {

s.push(temp);

temp = temp.getNext();

}

while(!s.isEmpty()) {

System.out.print(s.pop().getData() + " ");

}

System.out.println();

}

public void displayRev(Node head) {

if(head == null) {

return;

}

displayRev(head.getNext());

System.out.print(head.getData() +" ");

}

public boolean insert(int data, int position) {

if(position <= 0 || (head == null && position > 1)) {

return false;

}

Node newNode = new Node(data);

if(newNode == null) {

return false;

}

if( position == 1) {

newNode.setNext(head);

head = newNode;

return true;

}

//locate prev node

Node prev = head;

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

prev = prev.getNext();

if(prev == null) {

return false;

}

}

newNode.setNext(prev.getNext());

prev.setNext(newNode);

return true;

}

public boolean deleteByVal(int data) {

if(head == null) {

return false;

}

if(head.getData() == data) {

head = head.getNext();

return true;

}

Node prev = head, del = head;

while(del.getData() != data) {

prev = del;

del = del.getNext();

if(del == null) {

return false;

}

}

prev.setNext(del.getNext());

return true;

}

public boolean deleteByPosition(int position) {

if(head == null || position <= 0) {

return false;

}

if(position == 1) {

head = head.getNext();

return true;

}

Node prev = head;

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

prev = prev.getNext();

if(prev.getNext() == null) {

return false;

}

}

Node del = prev.getNext();

/\*if(del == null) {

return false;

}\*/

prev.setNext(del.getNext());

return true;

}

public void reverse() {

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

return;

}

Node n1 = head, n2 = head.getNext();

Node n3 = null;

while(n2 != null) {

n3 = n2.getNext();

n2.setNext(n1);

n1 = n2;

n2 = n3;

}

head.setNext(null);

head = n1;

}

public void reverse(Node n1, Node n2) {

if(n2 == null) {

head.setNext(null);

head = n1;

return;

}

reverse(n2, n2.getNext());

n2.setNext(n1);

}

}

public class Main {

public static void main(String[] args) {

LinkedList l1 = new LinkedList();

l1.display();

l1.reverse();

l1.displayRev();

l1.displayRev(l1.getHead());

System.out.println();

l1.insert(10);

l1.insert(20);

l1.insert(30);

l1.insert(40);

l1.insert(50);

l1.display();

l1.insert(60, 1);

l1.display();

l1.insert(70, 4);

l1.display();

l1.insert(80, 8);

l1.display();

System.out.println( l1.insert(100, 10) );

l1.display();

l1.insert(60);

l1.insert(60, 4);

l1.display();

//l1.reverse();

if(l1.getHead() != null && l1.getHead().getNext() != null) {

l1.reverse(l1.getHead(), l1.getHead().getNext());

}

l1.display();

/\*l1.deleteByPosition(1);

l1.display();

l1.deleteByPosition(4);

l1.display();

l1.deleteByPosition(8);

l1.display();

l1.deleteByPosition(8);

l1.display();\*/

/\* while( l1.deleteByVal(60))

;

l1.display();\*/

/\*l1.deleteByVal(60);

l1.display();

l1.deleteByVal(70);

l1.display();

l1.deleteByVal(80);

l1.display();

l1.deleteByVal(60);

l1.display();\*/

}

}

package stack;

public class StackUsingArray<T> {

    private T [] arr;

    private int size;

    private int top;

    public StackUsingArray(int size) {

        this.size = size;

        top = -1;

        arr = (T []) new Object[size];

    }

    public boolean isEmpty() {

        return top == -1;

    }

    public boolean isFull() {

        return top == (size-1);

    }

    public T push(T data) {

        if(isFull()) {

            return null;

        }

        arr[++top] = data;

        return data;

    }

    public T pop() {

        if(isEmpty()) {

            return null;

        }

        return arr[top--];

    }

    public T peek() {

        if(isEmpty()) {

            return null;

        }

        return arr[top];

    }

    public static void main(String[] args) {

     StackUsingLinkedList<Integer> stack = new StackUsingLinkedList<>();

        System.out.println("Push:"+stack.push(10));

        System.out.println("Push:"+stack.push(20));

        System.out.println("Push:"+stack.push(30));

        System.out.println("Pop:"+stack.pop());

        System.out.println("Peek:"+stack.peek());

    }

}

public class StackUsingLinkedList<T> {

    private static class Node<T> {

        T data;

        Node<T> next;

        Node(T data) {

            this.data = data;

            this.next = null;

        }

    }

    private Node<T> top;

    private int size;

    public StackUsingLinkedList() {

        top = null;

        size = 0;

    }

    public int size() {

        return size;

    }

    public boolean isEmpty() {

        return top == null;

    }

    public T push(T data) {

     Node<T> newNode = new Node<>(data);

        newNode.next = top;

        top = newNode;

        size++;

return data;

    }

    public T pop() {

        if (isEmpty()) {

            return null;

        }

        T data = top.data;

        top = top.next;

        size--;

        return data;

    }

    public T peek() {

        if (isEmpty()) {

            return null;

        }

        return top.data;

    }

    public static void main(String[] args) {

        StackUsingLinkedList<Integer> stack = new StackUsingLinkedList<>();

        System.out.println("Push:"+stack.push(10));

        System.out.println("Push:"+stack.push(20));

        System.out.println("Push:"+stack.push(30));

        System.out.println("Pop:"+stack.pop());

        System.out.println("Peek:"+stack.peek());

    }

}

public class DoubleStack {

int [] arr;

int size;

int top1, top2;

public DoubleStack(int size) {

this.size = size;

arr = new int[size];

top1 = -1;

top2 = size;

}

public boolean isEmpty1() {

return top1 == -1;

}

public boolean isEmpty2() {

return top2 == size;

}

public boolean isFull() {

return (top1 + 1) == top2;

}

public boolean push1(int data) {

if(isFull()) {

return false;

}

arr[++top1] = data;

return true;

}

public boolean push2(int data) {

if(isFull()) {

return false;

}

arr[--top2] = data;

return true;

}

public int pop1() {

if(isEmpty1()) {

return -999;

}

return arr[top1--];

}

public int pop2() {

if(isEmpty2()) {

return -999;

}

return arr[top2++];

}

}

public class Node {

private int data;

private Node left, right;

public Node(int data) {

this.data = data;

left = right = null;

}

public int getData() {

return data;

}

public void setData(int data) {

this.data = data;

}

public Node getLeft() {

return left;

}

public void setLeft(Node left) {

this.left = left;

}

public Node getRight() {

return right;

}

public void setRight(Node right) {

this.right = right;

}

}

import java.util.Stack;

public class BinarySearchTree {

private Node root;

public BinarySearchTree() {

root = null;

}

public boolean insert(int data) {

Node newNode = new Node(data);

// if(newNode == null) {

// return false;

// }

if(root == null) {

root = newNode;

return true;

}

Node temp = root;

while(true) {

if(data == temp.getData()) {

return false;

}

if(data < temp.getData()) {

//insert to left

//check if temp is already having a left child

if(temp.getLeft() == null) {

temp.setLeft(newNode);

return true;

}

temp = temp.getLeft();

}

else {

//insert to right

//check if temp has a right child?

if(temp.getRight() == null) {

temp.setRight(newNode);

return true;

}

temp = temp.getRight();

}

}

}

public void preOrder() { //P-L-R

Node temp = root;

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

System.out.print("PreOrder: ");

while(temp != null || !stack.empty()) {

while(temp != null) {

System.out.print(temp.getData() + " ");

stack.push(temp);

temp = temp.getLeft();

}

temp = stack.pop().getRight();

}

System.out.println();

}

public void inOrder() {

Node temp = root;

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

System.out.print("InOrder: ");

while(temp != null || !stack.empty()) {

while(temp != null) {

stack.push(temp);

temp = temp.getLeft();

}

temp = stack.pop();

System.out.print(temp.getData() + " ");

temp = temp.getRight();

}

System.out.println();

}

public int getMax() {

if(root == null) {

return -1;

}

Node temp = root;

while(temp.getRight() != null) {

temp = temp.getRight();

}

return temp.getData();

}

public boolean find(int data) {

Node temp = root;

while(temp != null) {

if(temp.getData() == data) {

return true;

}

if(data < temp.getData()) {

temp = temp.getLeft();

}

else {

temp = temp.getRight();

}

}

return false;

}

}

public class Main {

public static void main(String[] args) {

BinarySearchTree bst = new BinarySearchTree();

bst.preOrder();

bst.insert(50);

bst.insert(20);

bst.insert(10);

bst.insert(30);

bst.insert(40);

bst.insert(25);

bst.insert(90);

bst.insert(60);

bst.insert(100);

bst.insert(55);

bst.insert(80);

System.out.println(bst.insert(100));

bst.preOrder();

bst.inOrder();

}

}

class Customer {

private String name;

private int age;

private String email;

private Customer prev; // Reference to the previous node in the linked list

private Customer next; // Reference to the next node in the linked list

public Customer(String name, int age, String email) {

this.name = name;

this.age = age;

this.email = email;

this.prev = null;

this.next = null;

}

// Getters and setters (You can add more as needed)

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public Customer getPrev() {

return prev;

}

public void setPrev(Customer prev) {

this.prev = prev;

}

public Customer getNext() {

return next;

}

public void setNext(Customer next) {

this.next = next;

}

}

class DoublyLinkedList {

private Customer head; // Points to the first node in the list

private Customer tail; // Points to the last node in the list

public DoublyLinkedList() {

this.head = null;

this.tail = null;

}

public void addCustomer(Customer customer) {

if (head == null) { // If the list is empty

head = customer;

tail = customer;

} else {

customer.setPrev(tail);

tail.setNext(customer);

tail = customer;

}

}

public void displayCustomers() {

Customer current = head;

while (current != null) {

System.out.println("Name: " + current.getName() + ", Age: " + current.getAge() + ", Email: " + current.getEmail());

current = current.getNext();

}

}

public Customer searchCustomerByName(String name) {

Customer current = head;

while (current != null) {

if (current.getName().equals(name)) {

return current;

}

current = current.getNext();

}

return null;

}

public void removeCustomer(Customer customer) {

if (customer == null) {

return;

}

if (customer.getPrev() != null) {

customer.getPrev().setNext(customer.getNext());

} else {

head = customer.getNext();

}

if (customer.getNext() != null) {

customer.getNext().setPrev(customer.getPrev());

} else {

tail = customer.getPrev();

}

}

}

public class Main {

public static void main(String[] args) {

// Creating the doubly linked list object

DoublyLinkedList customerList = new DoublyLinkedList();

// Adding customers to the list using Customer objects

customerList.addCustomer(new Customer("Alice", 25, "alice@example.com"));

customerList.addCustomer(new Customer("Bob", 30, "bob@example.com"));

customerList.addCustomer(new Customer("Charlie", 22, "charlie@example.com"));

// Displaying all customers in the list

customerList.displayCustomers();

}

}

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

package Q1;

public class Book {

public int BookId;

public String BookTitle;

public String BookAuthor;

public double Price;

public Book(int bookId, String bookTitle, String bookAuthor, double price) {

super();

BookId = bookId;

BookTitle = bookTitle;

BookAuthor = bookAuthor;

Price = price;

}

public int getBookId() {

return BookId;

}

public void setBookId(int bookId) {

BookId = bookId;

}

public String getBookTitle() {

return BookTitle;

}

public void setBookTitle(String bookTitle) {

BookTitle = bookTitle;

}

public String getBookAuthor() {

return BookAuthor;

}

public void setBookAuthor(String bookAuthor) {

BookAuthor = bookAuthor;

}

public double getPrice() {

return Price;

}

public void setPrice(double price) {

Price = price;

}

@Override

public String toString() {

return " | BookId : " + BookId + " | BookTitle : " + BookTitle + " | BookAuthor : " + BookAuthor + " | Price : " + Price

+ " | ";

}

}

package Q1;

public class Node {

public Book B;

public Node next;

public Node(Book B) {

this.B = B;

this.next = null;

}

public Book getData() {

return B;

}

public void setData(Book B) {

this.B = B;

}

public Node getNext() {

return next;

}

public void setNext(Node head) {

this.next = head;

}

}

package Q1;

import java.util.Stack;

public class SinglyLinkedList {

private Node head;

public int count = 0;

public SinglyLinkedList() {

this.head = null;

}

public int getCount() {

return count;

}

public Node getHead() {

return head;

}

public void setHead(Node head) {

this.head = head;

}

public boolean insertByValue(Book B) {

Node newNode = new Node(B);

if (head == null) {

head = newNode;

count++;

return true;

}

Node last = head;

while (last.getNext() != null) {

last = last.getNext();

}

last.setNext(newNode);

count++;

return true;

}// end of insertByValue()

public void display() {

Node temp = head;

if (temp == null) {

return;

}

while (temp != null) {

System.out.println(temp.getData() + " | ");

temp = temp.getNext();

}

System.out.println(" ");

System.out.println(" ");

}// end of display()

public boolean insertByPosition(Book B, int position) {

Node newNode = new Node(B);

if (position <= 0 || (head == null && position > 1)) {

return false;

}

if (position == 1) {

newNode.setNext(head);

head = newNode;

count++;

return true;

}

Node prev = head;

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

prev = prev.getNext();

if (prev == null) {

return false;

}

}

newNode.setNext(prev.getNext());

prev.setNext(newNode);

return true;

}// end of insertByPosition()

public boolean deleteByValue(Book B) {

if (head == null) {

return false;

}

if (head.getData() == B) {

head = head.getNext();

return true;

}

Node prev = head;

Node del = head;

while (del.getData() != B) {

prev = del;

del = del.getNext();

if (del == null) {

return false;

}

} // end of while

prev.setNext(del.getNext());

System.out.println(" ");

return true;

}// end of deleteByValue()

public boolean deleteByPosition(int position) {

if (head == null || position < 1) {

return false;

}

if (position == 1) {

head = head.getNext();

return true;

}

Node prev = head;

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

prev = prev.getNext();

if (prev.getNext() == null) {

return false;

}

} // end of for loop

Node del = prev.getNext();

prev.setNext(del.getNext());

System.out.println(" ");

return true;

}// end of deleteByPosition()

public Book search(int id) {

Node temp = head;

if (temp.getData().getBookId() == id) {

return temp.getData();

}

while (temp != null) {

if (temp.getData().getBookId() == id) {

return temp.getData();

}

else {

temp = temp.getNext();

}

}

return null;

}

public void descendingOrder() {

Node temp = head;

Stack<Book> S = new Stack();

if (temp == null) {

return;

}

while (temp != null) {

S.push(temp.getData());

temp = temp.getNext();

}

while (!S.isEmpty()) {

System.out.print(S.pop() + " | ");

}

}

public boolean insertAtFront(SinglyLinkedList SLL, Book B) {

SLL.insertByPosition(B, 1);

return true;

}

public boolean deleteAtFront(SinglyLinkedList SLL, int position) {

SLL.deleteByPosition(1);

return true;

}

// public void displayReverse() {

//

// Node temp = head;

// Stack<Node> S = new Stack();

//

// if (temp == null) {

// return;

// }

//

// while (temp != null) {

// S.push(temp);

// temp = temp.getNext();

// }

//

// while (!S.isEmpty()) {

// System.out.print(S.pop().getData() + " | ");

// }

// }// end of diplayReverse()

//

}

package Q1;

import java.util.Scanner;

public class Tester {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Book B1 = new Book(1, "XYZ", "XYZ", 400);

Book B2 = new Book(2, "ABC", "ABC", 500);

Book B3 = new Book(3, "LMN", "LMN", 600);

Book B4 = new Book(4, "PQR", "PQR", 700);

Book B5 = new Book(5, "DEF", "DEF", 800);

SinglyLinkedList SLL = new SinglyLinkedList();

SLL.insertByValue(B1);

SLL.insertByValue(B2);

SLL.insertByValue(B3);

SLL.insertByValue(B4);

SLL.insertByValue(B5);

boolean exit = false;

while (!exit) {

System.out.println(" -------------------< Welcome To Book Shop >--------------------- ");

System.out.println(" | Choose : 1. Insert At End | 2. Display SLL | 3. Insert Book By Position "

+ "| 4. Delete Book By Position | 5. Delete At The End | 6. Search A Book | 7. Print Books In Descending Order ");

int choice = sc.nextInt();

switch (choice) {

case 1:

System.out.println("Enter Book Details : ID / Title / Author / Price ");

Book NewBook = new Book(sc.nextInt(), sc.next(), sc.next(), sc.nextDouble());

SLL.insertByValue(NewBook);

System.out.println(" | Congratulations ! New Book Inserted Succesfully !");

break;

case 2:

System.out.println(" | Displaying The SLL : - ");

SLL.display();

break;

case 3:

System.out.println("Enter Book Details : ID / Title / Author / Price ");

Book NewBook1 = new Book(sc.nextInt(), sc.next(), sc.next(), sc.nextDouble());

SLL.insertByValue(NewBook1);

System.out.println("Enter Position To Insert At : - ");

int position = sc.nextInt();

SLL.insertByPosition(NewBook1, position);

System.out.println(" | Congratulations ! New Book Inserted Succesfully !");

break;

case 4:

System.out.println("Please Enter Position To Delete The Book : - ");

int position1 = sc.nextInt();

SLL.deleteByPosition(position1);

SLL.display();

break;

case 5:

SLL.deleteByValue(B5);

System.out.println(" | Congratulations ! Book Deleted From The End ! ");

break;

case 6:

System.out.println(" Enter Book Id : - ");

System.out.println(SLL.search(sc.nextInt()));

System.out.println(" | Found This Book ! ");

break;

case 7:

System.out.println(" | Printing Books In Descending Order :- ");

SLL.descendingOrder();

break;

case 8:

System.out.println(" | Total No. Of Books In The SLL Are :- " + SLL.getCount());

break;

case 9:

System.out.println("Enter Book Details : ID / Title / Author / Price ");

Book NewBook2 = new Book(sc.nextInt(), sc.next(), sc.next(), sc.nextDouble());

SLL.insertAtFront(SLL, NewBook2);

System.out.println(" | Congratulations ! New Book Inserted Succesfully !");

break;

case 10:

SLL.deleteAtFront(SLL, 1);

System.out.println(" | Congratulations ! Book Deleted Succesfully !");

break;

default:

exit = true;

break;

}// end of switch

} // end of while

}

}

**package** stack;

**public** **class** StackUsingLinkedList<T> {

**private** **static** **class** Node<T> {

T data;

Node<T> next;

Node(T data) {

**this**.data = data;

**this**.next = **null**;

}

}

**private** Node<T> top;

**private** **int** size;

**public** StackUsingLinkedList() {

top = **null**;

size = 0;

}

**public** **int** size() {

**return** size;

}

**public** **boolean** isEmpty() {

**return** top == **null**;

}

**public** T push(T data) {

Node<T> newNode = **new** Node<>(data);

newNode.next = top;

top = newNode;

size++;

**return** data;

}

**public** T pop() {

**if** (isEmpty()) {

**return** **null**;

}

T data = top.data;

top = top.next;

size--;

**return** data;

}

**public** T peek() {

**if** (isEmpty()) {

**return** **null**;

}

**return** top.data;

}

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

StackUsingLinkedList<Integer> stack = **new** StackUsingLinkedList<>();

System.***out***.println("Push:"+stack.push(10));

System.***out***.println("Push:"+stack.push(20));

System.***out***.println("Push:"+stack.push(30));

System.***out***.println("Pop:"+stack.pop());

System.***out***.println("Peek:"+stack.peek());

}

}