**Hero-vired Content**

BINARY\_SEARCH\_TREE

// import java.util.LinkedList;

// import java.util.Queue;

class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

class BinarySearchTree {

Node root;

public BinarySearchTree() {

root = null;

}

void insert(int data) {

root = insertda(root, data);

}

private Node insertda(Node current, int data) {

if (current == null) {

return new Node(data);

}

if (data < current.data) {

current.left = insertda(current.left, data);

} else if (data > current.data) {

current.right = insertda(current.right, data);

}

return current;

}

public static void preorder(Node root)

{

if(root==null)

return;

System.out.print(root.data+" ");

preorder(root.left);

preorder(root.right);

}

public static void main(String[] args) {

BinarySearchTree tree = new BinarySearchTree();

int[] nodes = new int[]{15,5, 2, 7, 4, 16, 6, 19, 8};

for (int node : nodes) {

tree.insert(node);

}

System.out.println(tree.root.data);

preorder(tree.root);

}

}

FILE\_OPERATIONS

import java.io.\*;

//import java.util.\*;

public class Fileoperation {

public static void main(String[] args) {

/\* //writing content into file

===========================================================

try{

FileWriter wr=new FileWriter("newFile.txt");

wr.write("we are adding content into the file \n");

wr.close();

}

catch(Exception e){

System.out.println("Erroroccured while accessing file:");

}

===========================================================

\*/

/\*//to read content from the file

try{

File myobj=new File("newFile.txt");

myobj.createNewFile();

Scanner sc=new Scanner(myobj);

while(sc.hasNextLine())

{

String line=sc.nextLine();

System.out.println(line);

}

sc.close();

}

catch(Exception e)

{

e.printStackTrace();

}

} \*/

//to see behaviour of file creating classes

try{

FileWriter nol=new FileWriter("Hell.txt");

nol.append("222222222222222222222\n");

File no=new File("Hell.txt");

no.createNewFile();

nol.append("11111111111111111111111111\n");

nol.close();

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

//OPERATIONS\_BASIC

// import java.io.File;

import java.io.FileWriter;

// import java.io.IOException;

// import java.io.Writer;

public class Files {

public static void main(String[] args) {

try {

//to create a new file

/\*

====================================================================

File file = new File("newFile.txt");

if (file.createNewFile()) {

System.out.println("File created: " + file.getName());

} else {

System.out.println("File already exists.");

}

===================================================================

\*/

//to write content into the file

String a[]={"AAAAAAAAAAAAAA","BBBBBBBBBBBBBBB","CCCCCCCCCCCCCCCCCC","DDDDDDDDDDDDDDDDDDD"};

FileWriter writer=new FileWriter("newFile.txt");

writer.write("sadfghhhhhhhhhhh");

writer.write("\nasdfghhhhhhhhgfdsasdfglkjhgfdsa");

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

writer.write("\n"+a[i]);

writer.close();

} catch (Exception e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

TREES

/\*Trees

[L][VAL][R]

/ \

/ \

/ \

[L][VAL][R] [L][VAL][R]

| | | |

| | | |

NULL NULL NULL NULL

PREORDER TRAVERSAL::ROOT lc rc

POSTORDER TRAVERSAL:: lc rc ROOT

INORDER TRAVERSAL:: lc ROOT rc

[L][1][R]

/ \

/ \

/ \

[L][2][R] [L][3][R]

| | | |

| | | |

NULL NULL NULL NULL

PREORDER::1 2 3

POSTORDER:2 3 1

INORDER:2 1 3

IF WE ARE VISITNG NODE FOR FIRST TIME IT WILL BE IN :PREORDER

FOR SECOND TIME IT WILL BE IN ::INORDER

FOR THIRD TIME IT WILL BE IN::POST ORDER

PREORDER TRAVERSAL

PREORDER(Node root)

{

if(root==null)

return;

System.out.println(root.val);

PREORDER(root.lc);

PREORDER(root.rc);

}

POSTORDER(Node root)

{

if(root==null)

return;

POSTORDER(root.lc);

POSTORDER(root.rc);

system.out.print(root.val);

}

INORDER(NOde root){

if(root==null)

return;

INORDER(root.lc);

S.O.P(root.data);

INORDER(root.rc);

}

\*/

class Node{

int val;

Node lc,rc;

Node(int val)

{

this.val=val;

this.lc=null;

this.rc=null;

}

}

class Trees{

public static void preorder(Node root)

{

if(root==null)

return;

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

preorder(root.lc);

preorder(root.rc);

}

public static void postorder(Node root)

{

if(root==null)

return;

postorder(root.lc);

postorder(root.rc);

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

}

public static void inorder(Node root)

{

if(root==null)

return;

inorder(root.lc);

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

inorder(root.rc);

}

public static Node createTree(int arr[],int i){

Node root = null;

// Base case for recursion

if (i < arr.length) {

root = new Node(arr[i]);

// insert lcchild

root.lc = createTree(arr, 2 \* i + 1);

// insert rc child

root.rc = createTree(arr, 2 \* i + 2);

}

return root;

}

//another method for direct insertion

public static void addNode(int value,Node root,char dir1 , char dir2){

Node temp = root;

if(dir1 == 'L'){

while(temp.lc!=null){

temp=temp.lc;

}

}else{

while(temp.rc != null){

temp = temp.rc;

}

}

if(dir2=='L'){

temp.lc= new Node(value);

}else{

temp.rc = new Node(value);

}

}

public static void main(String[] args) {

int a[]={ 1, 2, 3, 4, 5, 6,7,8,9 };

Node root=createTree(a, 0);addNode(2, root, 'L','L');

// addNode(3, root, 'R','R');

// addNode(6, root, 'L','L');

// addNode(7, root, 'L','R');

preorder(root);

// System.out.println();

// postorder(root);

// System.out.println();

// inorder(root);

}

}

INHERITENCE  
/\*

oops--------

|

---------

| |

class object

class-blueprint of real world entity

object-instance of class

topics

\*\*

-inheritance

-polymorphism

-abstraction

-encapsulation --these are important topics

ohters topics are --coupling, -cohesion, association, Aggregation, composition

INHERITANCE

used for code reusability

extends-==- used for achieving inheritance

there are 2 types of relations

1)--> is-a relationship

2)--> has-a relationship

example

--------

class Car{

int Nt;

Engine engine;

}

class Toyoto extends Car{

Fridge fridge;

}

TYPES OF INHERITANCES 5 TYPES

1-->SIMPLE INHERITANCE(SINGLE LEVEL)

A

|

|

B

2-->MULTILEVEL INHEITANCE

A

|

|

B

|

|

C

3-->HEIRARCHICAL INHERITANCE

A

/ \

/ \

/ \

B C

4-->MULTIPLE INHERITANCE--THIS TYPE OF INHERITANCE CAN ONLY IMPLEMENTED BY INTERFACES ONLY

THEY CAN'T BE IMPLEMENTED USING CLASS

A B

\ /

\ /

\ /

\ /

C

5-->HYBRID INHERITANCE-- IT IS A COMBINATION OF MORE THAN ONE TYPE OF INHERITANCE

A

/ \

/ \

B C

\ /

\ /

D

POLYMORPHISM::GIVING DIFFERENT IMPLEMETATIONS FOR SAME METHOD/CLASS/VARIABLE

multiple implementation of single entity is known as polymorphism.

INHERITANCE CAN ALSO BE USED TO ACHIEVE POLYMORPHISM.

DEFINING DIFFERENT IMPLEMENTATIONS FOR SAME METHOD IS KONWN AS METHOD OVERRIDDING

EX:

CLASS A{

FUN()

S.O.P("CLASS A");

}

CLASS B EXTENS A{

FUN()

S.O.P("CLASS B");

}

TYPES OF POLYMORPHISMS

-->COMPILETIME POLYMORPHISM:METHOD OVERLOADING:::we can expect output after compilation.

-->RUNTIME POLYMORPHISM::METHOD OVERRIDDING::: we can expect the output only after runnig the program.

ABSTRACTION-->WHAT ? WHY ? HOW?

what::ABSTRACTION::hiding the internal implementation and showing the functionality.

why:: To achieve Security.

example

calss k{

public abstract void show(){};

}

class p extends k

{

public void show()

{

S.o.p("i am broken");

}

}

ENCAPSULATION::wrapping the data and methods into single class.

example

class A{

private int num;

public int getnum(){return num;}----->getter method

public void setnum(int val){this.num=val;}----->setter method

publicstatic void main(String args[]){

A a=new A();

s.o.p(a.num);//produces error

}

}

\*/

class Car{

int noTyres=4;

int noWindows=6;

int seats=5;

}

class Toyota extends Car{

boolean Fridge=false;

}

class Ferari extends Car{

boolean Fridge=true;

}

class Iheritanceintro

{

public static void main(String[] args) {

Toyota t=new Toyota();

System.out.println(t.noTyres+" "+t.Fridge);

Ferari f=new Ferari();

System.out.println(f.noTyres+" "+f.Fridge);

while(true)

System.out.println("I");

}

}

DATA\_STRUCTURES  
/\*

data structures--->containers to keep and organise the data.

int,float,byte,short,long,boolean,double,char ,string-->are the primitive data structures

non - primitive data structures are derived based on primitive data structures.

ARRAY

/ \

/ \

FIXED-SIZE VARIABLE-SIZED

ARRAY ARRAY

for fixed size array::::datatype[] varaiabke name=new datatype[size];

variable name always points to first location of the array.

array will occupy continous memory locations only.

\*/

SPIRAL\_PROBLEM

import java.util.\*;

class Spiral

{

public static void main(String args[])

{

Scanner se=new Scanner(System.in);

int r=se.nextInt();

int a[][]={{2,3,-1},{3,4,6},{9,7,8}};

int i;

int top=0,down=r-1,left=0,right=r-1;

int dir=0;

while(top<=down && left<=right)

{

if(dir==0)

{

for(i=left;i<=right;i++)

System.out.print(a[top][i]+" ");

top+=1;

}

else if(dir==1)

{

for(i=top;i<=down;i++)

System.out.print(a[i][right]+" ");

right-=1;

}

else if(dir==2)

{

for(i=right;i>=left;i--)

System.out.print(a[down][i]+" ");

down-=1;

}

else if(dir==3)

{

for(i=down;i>=top;i--)

System.out.print(a[i][left]+" ");

left+=1;

}

dir=(dir+1)%4;

}

}

}

LINKED\_LIST  
import java.util.Scanner;

class Node{

int val;

Node next;

Node(int x){

this.val=x;

this.next=null;

}

}

class Linkedlist{

static Node createList(int a[])

{Node head=null;

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

{

Node j=new Node(a[i]);

if(head==null)

head=j;

else{

Node ptr=head;

while(ptr.next!=null)

{

ptr=ptr.next;

}

ptr.next=j;

}

}

return head;

}

static Node insertstart(Node head,int k)

{

Node j=new Node(k);

if(head==null)

{

head=j;

}

else

{

j.next=head;

head=j;

}

return head;

}

static Node insertend(Node head,int k)

{

Node j=new Node(k);

if(head==null)

head=j;

else{

Node ptr=head;

while(ptr.next!=null)

{

ptr=ptr.next;

}

ptr.next=j;

}

return head;

}

static void printList(Node head)

{

Node ptr=head;

while(ptr.next!=null)

{

System.out.print(ptr.val+"->");

ptr=ptr.next;

}

System.out.print(ptr.val);

System.out.println();

}

static Node insertAfter(Node head,int pos,int val)

{

Node ptr=head;

while(ptr.val!=pos)

{

ptr=ptr.next;

}

Node j=new Node(val);

j.next=ptr.next;

ptr.next=j;

return head;

}

static Node insertbefore(Node head,int pos,int val){

Node ptr=head;

Node pre=ptr;

while(ptr.val!=pos)

{

pre=ptr;

ptr=ptr.next;

}

Node j=new Node(val);

j.next=ptr;

pre.next=j;

ptr=head;

return head;

}

static Node deletestart(Node head)

{

if(head!=null)

head=head.next;

return head;

}

static Node deletelast(Node head)

{

if(head==null)

return head;

Node ptr=head;

Node pre=ptr;

while(ptr.next!=null)

{

pre=ptr;

ptr=ptr.next;

}

pre.next=null;

return head;

}

static Node deletevalue(Node head,int x)

{

if(head==null)

return head;

if(head.val==x)

return head.next;

Node ptr=head;

Node pre=ptr;

while(ptr.val!=x)

{

pre=ptr;

ptr=ptr.next;

}

pre.next=ptr.next;

return head;

}

static Node deleteBefore(Node head,int pos)

{

Node prev=head;

Node cur=head;

Node nex=head;

if(head==null)

return null;

else if(head.next==null||head.val==pos)

{

System.out.println("impossible to delete");

return head;

}

else if(head.next.val==pos)

{

return head.next;

}

else{

while(nex.val!=pos)

{

prev=cur;

cur=nex;

nex=nex.next;

}

prev.next=nex;

}

return head;

}

static Node reverselist(Node head)

{/\*

Another method for reversing the linked list..

Node p=null;

Node n=null;

Node c=head;

while(c!=null)

{

n=c.next;

c.next=p;

p=c;

c=n;

}

return p;

\*/

Node ptr=head;

Node k=new Node(ptr.val);

ptr=ptr.next;

while(ptr!=null)

{

Node j=new Node(ptr.val);

j.next=k;

k=j;

ptr=ptr.next;

}

return k;

}

static Node deleteAfter(Node head,int k)

{

Node ptr=head;

Node pre=head;

while(pre.val!=k)

{

pre=ptr;

ptr=ptr.next;

}

pre.next=ptr.next;

return head;

}

// static Node deleteBefore(Node head,int k){

// Node pre=null;

// Node cur=head;

// Node nex=null;

// while(nex!=null)

// {

// pre=cur;

// nex=cur.next;

// }

// return head;

// }

static void oddeven(Node head)

{

int o=0,e=0;

Node ptr=head;

while(ptr!=null)

{

if((ptr.val)%2==0)

e+=ptr.val;

else

o+=ptr.val;

ptr=ptr.next;

}

System.out.println(Math.abs(o-e));

}

static String pallin(Node head,Node hea)

{

Node ptr=head;

while(ptr!=null && hea!=null)

{

if(ptr.val!=hea.val)

return "not pallin";

ptr=ptr.next;

hea=hea.next;

}

return "pallin";

}

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

// Node head=null;

Node head=null;

int a[]={1,2,3,4,5,6,7};

head=createList(a);

// int n=sc.nextInt();

// for(int i=0;i<n;i++)

// {

// int k=sc.nextInt();

// head=insertend(head, k);

// }

printList(head);

// head=insertend(head, 9);

// printList(head);

// head=insertstart(head, 0);

// printList(head);

// head=insertAfter(head, 2, 5);

// printList(head);

// head=insertbefore(head, 3, 7);

// printList(head);

// head=deletestart(head);

// printList(head);

// head=deletelast(head);

// printList(head);

// head=deletevalue(head, 9);

// printList(head);

// head=reverselist(head);

// printList(head);

// oddeven(head);

// Node ptr=head;

// Node hea=reverselist(ptr);

// printList(hea);

// printList(head);

// System.out.println(pallin(head,hea));

System.out.println("Enter node to delete");

int k=sc.nextInt();

// head=deleteAfter(head, k);

head=deleteBefore(head, k);

printList(head);

sc.close();

}

}

DOUBLE\_LINKED\_LIST  
class Node{

int data;

Node prev;

Node next;

Node(int x)

{

this.data=x;

this.next=null;

this.prev=null;

}

}

public class Doublinklist {

static Node generate(int a[]){

Node head=null;

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

{

Node j=new Node(a[i]);

if(head==null)

head=j;

else{

j.next=head;

head.prev=j;

head=j;

}

}

return head;

}

static void printdll(Node head)

{

Node ptr=head;

while(ptr!=null)

{

System.out.print(ptr.data+" ");

ptr=ptr.next;

}

System.out.println();

}

static Node insertstart(Node head,int x){

Node j=new Node(x);

if(head==null)

head=j;

else

{

j.next=head;

head.prev=j;

head=j;

}

return head;

}

static Node insertend(Node head,int x)

{

Node j=new Node(x);

if(head==null)

head=j;

else{

Node ptr=head;

while(ptr.next!=null)

{

ptr=ptr.next;

}

ptr.next=j;

j.prev=ptr;

}

return head;

}

static Node insertafter(Node head,int x,int y)

{

Node ptr=head;

Node j=new Node(y);

while(ptr.data!=x)

{

ptr=ptr.next;

}

j.next=ptr.next;

ptr.next.prev=j;

ptr.next=j;

j.prev=ptr;

return head;

}

static Node insertbefore(Node head,int x,int y)

{

Node j=new Node(y);

if(head.data==x)

{

j.next=head;

head.prev=j;

head=j;

}

else

{

Node ptr=head;

Node pre=ptr;

while(ptr.data!=x)

{

pre=ptr;

ptr=ptr.next;

}

pre.next=j;

j.prev=pre;

j.next=ptr;

ptr.prev=j;

}

return head;

}

static Node deletestart(Node head)

{

if(head==null)

return head;

else

{

head=head.next;

head.prev=null;

}

return head;

}

static Node deleteend(Node head)

{

if(head==null)

{

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

return head;

}

else if(head.next==null)

{

head=null;

}

else

{

Node ptr=head;

while(ptr.next!=null)

ptr=ptr.next;

ptr.prev.next=null;

ptr.prev=null;

}

return head;

}

static Node deleteval(Node head,int x)

{

if(head==null)

return head;

else if(head.data==x)

{

head=head.next;

head.prev=null;

}

else{

Node ptr=head;

Node pre=ptr;

while(ptr.data!=x)

{

pre=ptr;

ptr=ptr.next;

}

if(ptr.next!=null)

{

pre.next=ptr.next;

ptr.next.prev=pre;

}

else{

pre.next=null;

ptr.prev=null;

}

}

return head;

}

static Node deleteafter(Node head,int x)

{

Node ptr=head;

Node pre=ptr;

while(pre.data!=x)

{

pre=ptr;

ptr=ptr.next;

}

pre.next=ptr.next;

ptr.next.prev=pre;

ptr.prev=null;

return head;

}

static Node deletebefore(Node head,int x)

{

Node ptr=head;

Node pre=ptr;

while(ptr.data!=x)

{

pre=ptr;

ptr=ptr.next;

}

if(pre.prev!=null)

{

ptr.prev=pre.prev;

pre.prev.next=ptr;

pre.next=null;

pre.prev=null;

}

else

{

ptr.prev=null;

head=pre.next;

}

return head;

}

static Node reverselist(Node head)

{

Node p=null;

Node n=null;

Node c=head;

while(c!=null)

{

n=c.next;

c.next=p;

p=c;

c=n;

}

return p;

}

public static void main(String[] args) {

Node head=null;

int a[]={1,2,3,4,5};

head=generate(a);

printdll(head);

// head=insertstart(head, 0);

// printdll(head);

// head=insertend(head, 0);

// printdll(head);

// head=insertafter(head, 3, 9);

// printdll(head);

// head=insertbefore(head, 1, 0);

// head=deletestart(head);

head=deleteend(head);

head=deleteend(head);

head=deleteend(head);

head=deleteend(head);

head=deleteend(head);

head=deleteend(head);

// head=deleteval(head, 3);

// head=deleteafter(head,3);

// head=deletebefore(head, 4);

// head=reverselist(head);

printdll(head);

}

}

DEADLOCK\_THREADS

public class Deadlock {

static final String name="Rajesh";

static final String sname="we";

static class T1 extends Thread{

public void run(){

try{

synchronized(name){

System.out.println(name);

synchronized(sname)

{

System.out.println(sname);

}

}

}

catch(Exception e)

{

System.out.println("already occupied");

}

}}

static class T2 extends Thread{

public void run(){

try{

synchronized(sname)

{

System.out.println(sname);

synchronized(name){

System.out.println(name);

}

}

}

catch(Exception e)

{

System.out.println("already occupied");

}

}}

public static void main(String[] args) {

T1 t=new T1();

T2 u=new T2();

t.run();

u.run();

}

}

QUEUES

import java.util.ArrayList;

class Queueim{

public static void push(ArrayList<Integer> queue,int val){

queue.add(val);

}

public static int pop(ArrayList<Integer> queue)

{

int k=-1;

if(!queue.isEmpty()){

k=queue.get(0);

queue.remove(0);

}

return k;

}

static int size(ArrayList<Integer> queue)

{

return queue.size();

}

static boolean empty(ArrayList<Integer> queue)

{

return queue.isEmpty();

}

public static void printQueue(ArrayList<Integer> queue)

{

for(int i=0;i<queue.size();i++)

System.out.print(queue.get(i)+" ");

System.out.println();

}

public static int top(ArrayList<Integer> queue)

{

int k=-1;

if(!queue.isEmpty())

k=queue.get(queue.size()-1);

return k;

}

public static void main(String[] args) {

ArrayList<Integer> queue=new ArrayList<>();

push(queue,8);

push(queue,23);

push(queue,67);

printQueue(queue);

pop(queue);

System.out.println(size(queue));

printQueue(queue);

System.out.println(empty(queue));

System.out.println(size(queue));

System.out.println(top(queue));

}

}

SLIDING\_WINDOW\_QUEUE

import java.util.\*;

public class Slidingwin {

public static int sumw(ArrayList<Integer> a)

{

int s=0;

for(int i=0;i<a.size();i++)

s+=a.get(i);

return s;

}

public static void main(String[] args) {

int a[]={1,2,3,4,5,6,7,8,9,10};

ArrayList<Integer> A=new ArrayList<>();

ArrayList<Integer> B=new ArrayList<>();

int k=3;

for(int i=0;i<k;i++)

A.add(a[i]);

B.add(sumw(A));

for(int i=k;i<a.length;i++)

{

A.remove(0);

A.add(a[i]);

B.add(sumw(A));

}

System.out.println((B));

}

}

PRIORITY\_QUEUE

import java.util.Comparator;

import java.util.PriorityQueue;

//import java.util.concurrent.PriorityBlockingQueue;

public class PriorityQueue2 {

static class myComparator implements Comparator<Integer>{

@Override

public int compare(Integer o1, Integer o2) {

int val1 = o1;

int val2 = o2;

if(val2 >= val1) return 1;

return -1;

}

}

public static void main(String[] args) {

PriorityQueue<Integer> myQueue = new PriorityQueue<>(new myComparator());

// PriorityQueue<Integer> myQueue = new PriorityQueue<>();

// myQueue.add(10);

myQueue.add(10);

myQueue.add(80);

myQueue.add(910);

System.out.println(myQueue.poll());

// myQueue.add(9);

// myQueue.add(-1);

// myQueue.add(-15);

// myQueue.add(90);

// printList(myQueue);

// System.out.println(myQueue.poll());

// System.out.println(myQueue.poll());

// PriorityQueue<String> myQueue = new PriorityQueue<>();

// PriorityBlockingQueue<String> myQueue = new PriorityBlockingQueue<>();

// myQueue.add("a");

// myQueue.add("a");

// myQueue.add("a");

// myQueue.add("d");

// myQueue.add("e");

// myQueue.add("ab");

// myQueue.add("ac");

// System.out.println(myQueue.poll());

// System.out.println(myQueue.poll());

// System.out.println(myQueue.poll());

}

public static void printList(PriorityQueue<Integer> myQueue){

for(int e:myQueue){

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

}

System.out.println();

}

}

// //import java.util.\*;

// import java.util.concurrent.PriorityBlockingQueue;

// class PriorityQueue2 {

// static void print(PriorityBlockingQueue<String> my)

// {

// for(String i:my){

// System.out.println(i+" ");

// }

// System.out.println();

// }

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

// // PriorityQueue<String> pr=new PriorityQueue();

// PriorityBlockingQueue<String> my=new PriorityBlockingQueue<>();

// my.add("XYZ");

// my.add("ABC");

// my.add("ABa");

// my.add("DEF");

// my.add("GHI");

// //in priorityBlockingQueue the pop() operation is done based on Lexographical order(dictionary order).

// //print(my);

// System.out.println(my.poll());

// System.out.println(my.poll());

// }

// }

Pyramid\_Pattern

for(int i=0;i<n;i++)

{

for(int j=n-1-i;j>=0;j--)

System.out.print(" ");

for(int k=0;k<=i;k++)

System.out.print("\* ");

System.out.println();

}