10 : REGULAR EXPRESSION:

**import** java.util.regex.\*;

**public** **class** regularExpnAssisted {

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

String pattern = "[a-z]+";

String check = "Regular Expressions";

Pattern p = Pattern.*compile*(pattern);

Matcher c = p.matcher(check);

**while** (c.find())

System.***out***.println( check.substring( c.start(), c.end() ) );

}

}

2:ACCESS MODIFERS:

PUBLIC:

//4. using public access specifiers

**package** pack1;

**public** **class** pubaccessspecifiers {

**public** **void** display()

{

System.***out***.println("This is Public Access Specifiers");

}

}

//create another package

**package** pack2;

**import** pack1.\*;

**public** **class** accessSpecifiers4 {

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

pubaccessspecifiers obj = **new** pubaccessspecifiers();

obj.display();

}

}

:PRIVATE:

//2. using private access specifiers

**class** priaccessspecifier

{

**private** **void** display()

{

System.***out***.println("You are using private access specifier");

}

}

**public** **class** accessSpecifiers2 {

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

//private

System.***out***.println("Private Access Specifier");

priaccessspecifier obj = **new** priaccessspecifier();

//trying to access private method of another class

//obj.display();

}

}

:DEFAULT:

//1. Class is having Default access modifier

**class** defAccessSpecifier

{

**void** display()

{

System.***out***.println("You are using defalut access specifier");

}

}

**public** **class** accessSpecifiers1 {

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

//default

System.***out***.println("Dafault Access Specifier");

defAccessSpecifier obj = **new** defAccessSpecifier();

obj.display();

}

}

:PROCTECTABLE:

//3. using protected access specifiers

**package** pack1;

**public** **class** proaccessspecifiers {

**protected** **void** display()

{

System.***out***.println("This is protected access specifier");

}

}

//create another package

**package** pack2;

**import** pack1.\*;

**public** **class** accessSpecifiers3 **extends** proaccessspecifiers {

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

accessSpecifiers3 obj = **new** accessSpecifiers3 ();

obj.display();

}

}

7: INNER CLASS:

**public** **class** innerClassAssisted1 {

**private** String msg="Welcome to Java";

**class** Inner{

**void** hello(){System.***out***.println(msg+", Let us start learning Inner Classes");}

}

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

innerClassAssisted1 obj=**new** innerClassAssisted1();

innerClassAssisted1.Inner in=obj.**new** Inner();

in.hello();

}

}

**public** **class** innerClassAssisted2 {

**private** String msg="Inner Classes";

**void** display(){

**class** Inner{

**void** msg(){

System.***out***.println(msg);

}

}

Inner l=**new** Inner();

l.msg();

}

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

innerClassAssisted2 ob=**new** innerClassAssisted2 ();

ob.display();

}

}

//anonymous inner class

**abstract** **class** AnonymousInnerClass {

**public** **abstract** **void** display();

}

**public** **class** innerClassAssisted3 {

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

AnonymousInnerClass i = **new** AnonymousInnerClass() {

**public** **void** display() {

System.***out***.println("Anonymous Inner Class");

}

};

i.display();

}

}

**7: MAP**

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

**public** **class** mapDemo {

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

// map

//Hashmap

HashMap<Integer,String> hm=**new** HashMap<Integer,String>();

hm.put(1,"Tim");

hm.put(2,"Mary");

hm.put(3,"Catie");

System.***out***.println("\nThe elements of Hashmap are ");

**for**(Map.Entry m:hm.entrySet()){

System.***out***.println(m.getKey()+" "+m.getValue());

}

//HashTable

Hashtable<Integer,String> ht=**new** Hashtable<Integer,String>();

ht.put(4,"Ales");

ht.put(5,"Rosy");

ht.put(6,"Jack");

ht.put(7,"John");

System.***out***.println("\nThe elements of HashTable are ");

**for**(Map.Entry n:ht.entrySet()){

System.***out***.println(n.getKey()+" "+n.getValue());

}

//TreeMap

TreeMap<Integer,String> map=**new** TreeMap<Integer,String>();

map.put(8,"Annie");

map.put(9,"Carlotte");

map.put(10,"Catie");

System.***out***.println("\nThe elements of TreeMap are ");

**for**(Map.Entry l:map.entrySet()){

System.***out***.println(l.getKey()+" "+l.getValue());

}

}

}

8:STRING TO STRING BUFFER ND BULIDERS:

**public** **class** stringDemo {

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

//methods of strings

System.***out***.println("Methods of Strings");

String sl=**new** String("Hello World");

System.***out***.println(sl.length());

//substring

String sub=**new** String("Welcome");

System.***out***.println(sub.substring(2));

//String Comparison

String s1="Hello";

String s2="Heldo";

System.***out***.println(s1.compareTo(s2));

//IsEmpty

String s4="";

System.***out***.println(s4.isEmpty());

//toLowerCase

String s5="Hello";

System.***out***.println(s1.toLowerCase());

//replace

String s6="Heldo";

String replace=s2.replace('d', 'l');

System.***out***.println(replace);

//equals

String x="Welcome to Java";

String y="WeLcOmE tO JaVa";

System.***out***.println(x.equals(y));

System.***out***.println("\n");

System.***out***.println("Creating StringBuffer");

//Creating StringBuffer and append method

StringBuffer s=**new** StringBuffer("Welcome to Java!");

s.append("Enjoy your learning");

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

//insert method

s.insert(0, 'w');

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

//replace method

StringBuffer sb=**new** StringBuffer("Hello");

sb.replace(0, 2, "hEl");

System.***out***.println(sb);

//delete method

sb.delete(0, 1);

System.***out***.println(sb);

//StringBuilder

System.***out***.println("\n");

System.***out***.println("Creating StringBuilder");

StringBuilder sb1=**new** StringBuilder("Happy");

sb1.append("Learning");

System.***out***.println(sb1);

System.***out***.println(sb1.delete(0, 1));

System.***out***.println(sb1.insert(1, "Welcome"));

System.***out***.println(sb1.reverse());

//conversion

System.***out***.println("\n");

System.***out***.println("Conversion of Strings to StringBuffer and StringBuilder");

String str = "Hello";

// conversion from String object to StringBuffer

StringBuffer sbr = **new** StringBuffer(str);

sbr.reverse();

System.***out***.println("String to StringBuffer");

System.***out***.println(sbr);

// conversion from String object to StringBuilder

StringBuilder sbl = **new** StringBuilder(str);

sbl.append("world");

System.***out***.println("String to StringBuilder");

System.***out***.println(sbl);

}

}

5: CONSTRUCTORS:

**class** EmpInfo{

**int** id;

String name;

**void** display() {

System.***out***.println(id+" "+name);

}

}

**public** **class** constructorDemo {

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

EmpInfo emp1=**new** EmpInfo();

EmpInfo emp2=**new** EmpInfo();

emp1.display();

emp2.display();

}

}

//parameterized constructor

**class** Std{

**int** id;

String name;

Std(**int** i,String n)

{

id=i;

name=n;

}

**void** display() {

System.***out***.println(id+" "+name);

}

}

**public** **class** paramConstrDemo {

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

Std std1=**new** Std(2,"Alex");

Std std2=**new** Std(10,"Annie");

std1.display();

std2.display();

}

}

6 **: COLLECTION**

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

**public** **class** collectionAssisted {

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

//creating arraylist

System.***out***.println("ArrayList");

ArrayList<String> city=**new** ArrayList<String>();

city.add("Bangalore");//

city.add("Delhi");

System.***out***.println(city);

//creating vector

System.***out***.println("\n");

System.***out***.println("Vector");

Vector<Integer> vec = **new** Vector();

vec.addElement(15);

vec.addElement(30);

System.***out***.println(vec);

//creating linkedlist

System.***out***.println("\n");

System.***out***.println("LinkedList");

LinkedList<String> names=**new** LinkedList<String>();

names.add("Alex");

names.add("John");

Iterator<String> itr=names.iterator();

**while**(itr.hasNext()){

System.***out***.println(itr.next());

//creating hashset

System.***out***.println("\n");

System.***out***.println("HashSet");

HashSet<Integer> set=**new** HashSet<Integer>();

set.add(101);

set.add(103);

set.add(102);

set.add(104);

System.***out***.println(set);

//creating linkedhashset

System.***out***.println("\n");

System.***out***.println("LinkedHashSet");

LinkedHashSet<Integer> set2=**new** LinkedHashSet<Integer>();

set2.add(11);

set2.add(13);

set2.add(12);

set2.add(14);

System.***out***.println(set2);

}

}

}

4 : METHODS AND DIFF RETURN TYPES

**public** **class** methodExecution {

**public** **int** multipynumbers(**int** a,**int** b) {

**int** z=a\*b;

**return** z;

}

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

methodExecution b=**new** methodExecution();

**int** ans= b.multipynumbers(10,3);

System.***out***.println("Multipilcation is :"+ans);

}

//call by value

**public** **class** callMethod {

**int** val=150;

**int** operation(**int** val) {

val =val\*10/100;

**return**(val);

}

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

callMethod d = **new** callMethod();

System.***out***.println("Before operation value of data is "+d.val);

d.operation(100);

System.***out***.println("After operation value of data is "+d.val);

}

}

//method overloading

**public** **class** overloadMethod {

**public** **void** area(**int** b,**int** h)

{

System.***out***.println("Area of Triangle : "+(0.5\*b\*h));

}

**public** **void** area(**int** r)

{

System.***out***.println("Area of Circle : "+(3.14\*r\*r));

}

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

{

overloadMethod ob=**new** overloadMethod();

ob.area(10,12);

ob.area(5);

}

}

11: ARRAYS OF SINGLE AND MULTI DIMENSIONAL:

**public** **class** arrayAssisted {

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

//single-dimensional array

**int** a[]= {10,20,30,40,50};

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

System.***out***.println("Elements of array a: "+a[i]);

}

//multidimensional array

**int**[][] b = {

{2, 4, 6, 8},

{3, 6, 9} };

System.***out***.println("\nLength of row 1: " + b[0].length);

}

}

12:THREADS

public class MyThread extends Thread

{

public void run()

{

System.out.println("concurrent thread started running..");

}

public static void main( String args[] )

{

MyThread mt = new MyThread();

mt.start();

}

}

* Enter **MyRunnableThread** in class name, check the checkbox “public static void main(String[] args)”, and click on “Finish.”

public class MyRunnableThread implements Runnable{

public static int myCount = 0;

public MyRunnableThread(){

}

public void run() {

while(MyRunnableThread.myCount <= 10){

try{

System.out.println("Expl Thread: "+(++MyRunnableThread.myCount));

Thread.sleep(100);

} catch (InterruptedException iex) {

System.out.println("Exception in thread: "+iex.getMessage());

}

}

}

public static void main(String a[]){

System.out.println("Starting Main Thread...");

MyRunnableThread mrt = new MyRunnableThread();

Thread t = new Thread(mrt);

t.start();

while(MyRunnableThread.myCount <= 10){

try{

System.out.println("Main Thread: "+(++MyRunnableThread.myCount));

Thread.sleep(100);

} catch (InterruptedException iex){

System.out.println("Exception in main thread: "+iex.getMessage());

}

}

System.out.println("End of Main Thread...");

}

}

7 :SLEEP WAIT:

public class MyClass

{

private static Object LOCK = new Object();

public static void main(String args[]) throws InterruptedException

{

Thread.sleep(1000);

System.out.println("Thread '" + Thread.currentThread().getName() + "' is woken after sleeping for 1 second");

synchronized (LOCK)

{

LOCK.wait(1000);

System.out.println("Object '" + LOCK + "' is woken after" + " waiting for 1 second");

}

}

}

14: WITH SYCHRONIZATION import java.io.\*;

import java.util.\*;

class Sender

{

public void send(String msg)

{

System.out.println("Sending\t" + msg );

try

{

Thread.sleep(1000);

}

catch (Exception e)

{

System.out.println("Thread interrupted.");

}

System.out.println("\n" + msg + "Sent");

}

}

class ThreadedSend extends Thread

{

private String msg;

private Thread t;

Sender sender;

ThreadedSend(String m, Sender obj)

{

msg = m;

sender = obj;

}

public void run()

{

synchronized(sender)

{

sender.send(msg);

}

}

}

class SyncDemo

{

public static void main(String args[])

{

Sender snd = new Sender();

ThreadedSend S1 =

new ThreadedSend( " Hi " , snd );

ThreadedSend S2 =

new ThreadedSend( " Bye " , snd );

S1.start();

S2.start();

try

{

S1.join();

S2.join();

}

catch(Exception e)

{

System.out.println("Interrupted");

}

}

} 15:TRY CATCH:

public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

16: THROWS public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

: THROWS public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

:FINALLY

public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

:CUSTOM:

public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

17:EXCEPTION

public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

18:CREATE

public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

:READ:

public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

:UPDATE:

public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

:DELETE:

public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

19:CLASSES AND OBJECTS:

public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

:POLYMORPHISM:

public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

:INHERITANCE:

public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

:ENCAPSULATION:

public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

:ABSTRACTION:

public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

20: DIAMOND

public class MyClass

{

public static void main(String args[])

{

int[] array = new int[3];

try

{

array[7] = 3;

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index is out of bounds!");

}

finally

{

System.out.println("The array is of size " + array.length);

}

}

}

22:ARRAY ROTATION:

class RotateArray {

public void rotate(int[] nums, int k) {

if(k > nums.length)

k=k%nums.length;

int[] result = new int[nums.length];

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

result[i] = nums[nums.length-k+i];

}

int j=0;

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

result[i] = nums[j];

j++;

}

System.arraycopy( result, 0, nums, 0, nums.length );

}

}

public class Main

{

public static void main(String[] args) {

RotateArray r = new RotateArray();

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

r.rotate(arr, 5);

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

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

}

}

}

23:ORDER STATISTICS:

class KthSmallst

{

int kthSmallest(int arr[], int l, int r, int k)

{

if (k > 0 && k <= r - l + 1)

{

int pos = randomPartition(arr, l, r);

if (pos-l == k-1)

return arr[pos];

if (pos-l > k-1)

return kthSmallest(arr, l, pos-1, k);

return kthSmallest(arr, pos+1, r, k-pos+l-1);

}

return Integer.MAX\_VALUE;

}

void swap(int arr[], int i, int j)

{

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

int partition(int arr[], int l, int r)

{

int x = arr[r], i = l;

for (int j = l; j <= r - 1; j++)

{

if (arr[j] <= x)

{

swap(arr, i, j);

i++;

}

}

swap(arr, i, r);

return i;

}

int randomPartition(int arr[], int l, int r)

{

int n = r-l+1;

int pivot = (int)(Math.random()) \* (n-1);

swap(arr, l + pivot, r);

return partition(arr, l, r);

}

}

public class Main

{

public static void main(String[] args) {

KthSmallst ob = new KthSmallst();

int arr[] = {12, 3, 5, 7, 4, 19, 26};

int n = arr.length,k = 4;

System.out.println("K'th smallest element is "+ ob.kthSmallest(arr, 0, n-1, k));

}

}

24:RANGE QUERIES:

public class RangeQueries

{

static int k = 16;

static int N = 100000;

static long table[][] = new long[N][k + 1];

static void buildSparseTable(int arr[], int n)

{

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

table[i][0] = arr[i];

for (int j = 1; j <= k; j++)

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

table[i][j] = table[i][j - 1] + table[i + (1 << (j - 1))][j - 1];

}

static long query(int L, int R)

{

long answer = 0;

for (int j = k; j >= 0; j--)

{

if (L + (1 << j) - 1 <= R)

{

answer = answer + table[L][j];

L += 1 << j;

}

}

return answer;

}

public static void main(String args[])

{

int arr[] = { 3, 7, 2, 5, 8, 9 };

int n = arr.length;

buildSparseTable(arr, n);

System.out.println(query(0, 5));

System.out.println(query(3, 5));

System.out.println(query(2, 4));

}

}

25:WORKING MATRICES:

public class MultiplyMatrices

{

public static void main(String[] args)

{

int r1 = 2, c1 = 3;

int r2 = 3, c2 = 2;

int[][] firstMatrix = { {3, -2, 5}, {3, 0, 4} };

int[][] secondMatrix = { {2, 3}, {-9, 0}, {0, 4} };

int[][] product = multiplyMatrices(firstMatrix, secondMatrix, r1, c1, c2);

displayProduct(product);

}

public static int[][] multiplyMatrices(int[][] firstMatrix, int[][] secondMatrix, int r1, int c1, int c2)

{

int[][] product = new int[r1][c2];

for(int i = 0; i < r1; i++)

{

for (int j = 0; j < c2; j++)

{

for (int k = 0; k < c1; k++)

{

product[i][j] += firstMatrix[i][k] \* secondMatrix[k][j];

}

}

}

return product;

}

public static void displayProduct(int[][] product)

{

System.out.println("Product of two matrices is: ");

for(int[] row : product)

{

for (int column : row)

{

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

}

System.out.println();

}

}

}

26:SINGLE LINKED LIST:

import java.io.\*;

public class LinkedList

{

Node head; // head of list

static class Node

{

int data;

Node next;

Node(int d)

{

data = d;

next = null;

}

}

// Method to insert a new node

public static LinkedList insert(LinkedList list, int data)

{

// Create a new node with given data

Node new\_node = new Node(data);

new\_node.next = null;

// If the Linked List is empty, then make the new node as head

if (list.head == null)

{

list.head = new\_node;

}

else

{

// Else traverse till the last node and insert the new\_node there

Node last = list.head;

while (last.next != null)

{

last = last.next;

}

// Insert the new\_node at last node

last.next = new\_node;

}

return list;

}

public static void printList(LinkedList list)

{

Node currNode = list.head;

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

// Traverse through the LinkedList

while (currNode != null)

{

// Print the data at current node

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

// Go to next node

currNode = currNode.next;

}

System.out.println();

}

// Method to delete a node in the LinkedList by KEY

public static LinkedList deleteByKey(LinkedList list, int key)

{

// Store head node

Node currNode = list.head, prev = null;

If (currNode != null && currNode.data == key)

{

list.head = currNode.next; // Changed head

System.out.println(key + " found and deleted");

return list;

}

while (currNode != null && currNode.data != key)

{

prev = currNode;

currNode = currNode.next;

}

if (currNode != null)

{

prev.next = currNode.next;

System.out.println(key + " found and deleted");

}

if (currNode == null)

{

System.out.println(key + " not found");

}

return list;

}

// method to create a Singly linked list with n nodes

public static void main(String[] args)

{

/\* Start with the empty list. \*/

LinkedList list = new LinkedList();

// Insert the values

list = insert(list, 1);

list = insert(list, 2);

list = insert(list, 3);

list = insert(list, 4);

list = insert(list, 5);

list = insert(list, 6);

list = insert(list, 7);

list = insert(list, 8);

// Print the LinkedList

printList(list);

// Delete node with value 1

deleteByKey(list, 1);

// Print the LinkedList

printList(list);

// Delete node with value 4

deleteByKey(list, 4);

// Print the LinkedList

printList(list);

// Delete node with value 10

deleteByKey(list, 10);

// Print the LinkedList

printList(list);

}

}

27:CIRCULAR LINKED LIST:

public class LinkedList

{

static class Node

{

int data;

Node next;

Node(int d)

{

data = d;

next = null;

}

}

Node head;

LinkedList()

{

head = null;

}

void sortedInsert(Node new\_node)

{

Node current = head;

if (current == null)

{

new\_node.next = new\_node;

head = new\_node;

}

else if (current.data >= new\_node.data)

{

while (current.next != head)

current = current.next;

current.next = new\_node;

new\_node.next = head;

head = new\_node;

}

else

{

while (current.next != head && current.next.data < new\_node.data)

current = current.next;

new\_node.next = current.next;

current.next = new\_node;

}

}

void printList()

{

if (head != null)

{

Node temp = head;

do

{

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

temp = temp.next;

} while (temp != head);

}

}

public static void main(String[] args)

{

LinkedList list = new LinkedList();

int arr[] = new int[] {12, 56, 2, 11, 1, 90};

Node temp = null;

for (int i = 0; i < 6; i++)

{

temp = new Node(arr[i]);

list.sortedInsert(temp);

}

list.printList();

}

}

28:DOUBLY LINKED LIST:

public class DLL

{

     Node head;

class Node

{

         int data;

         Node prev;

         Node next;

Node(int d)

{

data = d;

}

     }

public void push(int new\_data)

     {

Node new\_Node = new Node(new\_data);

new\_Node.next = head;

         new\_Node.prev = null;

if (head != null)

             head.prev = new\_Node;

head = new\_Node;

     }

public void InsertAfter(Node prev\_Node, int new\_data)

     {

if (prev\_Node == null)

{

             System.out.println("The given previous node cannot be NULL ");

             return;

         }

Node new\_node = new Node(new\_data);

new\_node.next = prev\_Node.next;

prev\_Node.next = new\_node;

new\_node.prev = prev\_Node;

if (new\_node.next != null)

             new\_node.next.prev = new\_node;

     }

     void append(int new\_data)

     {

Node new\_node = new Node(new\_data);

   Node last = head;

new\_node.next = null;

if (head == null)

{

             new\_node.prev = null;

             head = new\_node;

             return;

         }

while (last.next != null)

             last = last.next;

last.next = new\_node;

new\_node.prev = last;

     }

public void printlist(Node node)

     {

         Node last = null;

         System.out.println("Traversal in forward Direction");

         while (node != null)

{

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

             last = node;

             node = node.next;

         }

         System.out.println();

         System.out.println("Traversal in reverse direction");

         while (last != null)

{

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

             last = last.prev;

         }

     }

public static void main(String[] args)

     {

DLL dll = new DLL

dll.append(6);

dll.push(7);

dll.push(1);

dll.append(4);

dll.InsertAfter(dll.head.next, 8);

   System.out.println("Created DLL is: ");

         dll.printlist(dll.head);

     }

}

29:OPERATIONS ON STACK:

public class Stack

{

static final int MAX = 1000;

int top;

int a[] = new int[MAX];

boolean isEmpty()

{

return (top < 0);

}

Stack()

{

top = -1;

}

boolean push(int x)

{

if (top >= (MAX-1))

{

System.out.println("Stack Overflow");

return false;

}

else

{

a[++top] = x;

System.out.println(x + " pushed into stack");

return true;

}

}

int pop()

{

if (top < 0)

{

System.out.println("Stack Underflow");

return 0;

}

else

{

int x = a[top--];

return x;

}

}

public static void main(String args[])

{

Stack s = new Stack();

s.push(10);

s.push(20);

s.push(30);

System.out.println(s.pop() + " Popped from stack");

}

}

30:WORKING OF QUEUE:

public class QueueExample

{

public static void main(String[] args)

{

Queue<String> locationsQueue = new LinkedList<>();

locationsQueue.add("Kolkata");

locationsQueue.add("Patna");

locationsQueue.add("Delhi");

locationsQueue.add("Gurgaon");

locationsQueue.add("Noida");

System.out.println("Queue is : " + locationsQueue);

System.out.println("Head of Queue : " + locationsQueue.peek());

locationsQueue.remove();

System.out.println("After removing Head of Queue : " + locationsQueue);

System.out.println("Size of Queue : " + locationsQueue.size());

}

}

31:INCREASING SUBSEQUENCE:

NOT THERE

32:LINEAR SEARCH:

import java.util.Scanner;  
  
public class linearSearch {  
  
 public static void main(String[] args){  
  
 int[] arr = {10,20,30,40,50};  
  
 Scanner sc = new Scanner(System.*in*);  
 System.*out*.println("Enter the element to be searched");  
 int searchValue = sc.nextInt();  
 int result = (int) *linearing*(arr,searchValue);  
  
 if(result==-1){  
  
 System.*out*.println("Element not in the array");  
 } else {  
  
 System.*out*.println("Element found at "+result+" and the search key is "+arr[result]);  
 }  
  
  
 }  
  
  
  
  
public static int linearing(int arr[], int x) {  
  
 int arrlength = arr.length;  
 for (int i = 0; i < arrlength - 1; i++) {  
  
 if (arr[i] == x) {  
  
 return i;  
  
 }  
 }  
  
 return -1;  
  
 }  
  
}

33:BINARY SEARCH:

public class binarySearch {  
  
 public static void main(String[] args){  
  
  
 int[] arr = {3,6,9,12,15};  
 int key = 12;  
 int arrlength = arr.length;  
 *binarySearch*(arr,0,key,arrlength);  
 }  
  
public static void binarySearch(int[] arr, int start, int key, int length){  
  
 int midValue = (start+length)/2;  
 while(start<=length){  
  
 if(arr[midValue]<key){  
  
 start = midValue + 1;  
 } else if(arr[midValue]==key){  
 System.*out*.println("Element is found at index :"+midValue);  
 break;  
 }else {  
  
 length=midValue-1;  
 }  
 midValue = (start+length)/2;  
 }  
 if(start>length){  
  
 System.*out*.println("Element is not found");  
 }  
  
}  
  
}

34:EXPONENTIAL SEARCH:

import java.util.Arrays;  
  
public class expSearch {  
  
public static void main(String[] args){  
  
 int[] arr = {6,12,18,24,32};  
 int length= arr.length;  
 int value = 18;  
 int outcome = *exponentialSearch*(arr,length,value);  
  
 if(outcome<0){  
  
 System.*out*.println( "Element is not present in the array");  
  
 }else {  
  
 System.*out*.println( "Element is present in the array at index :"+outcome);  
 }  
  
 }  
  
 public static int exponentialSearch(int[] arr ,int length, int value ){  
  
 if(arr[0]==value){  
 return 0;  
 }  
 int i=1;  
 while(i<length && arr[i]<=value){  
  
 i=i\*2;  
 }  
 return Arrays.*binarySearch*(arr,i/2,Math.*min*(i,length),value);  
 }  
  
  
}

35:SELECTION SORT:

public class selectionSort {  
  
 public static void main(String[] args) {  
  
 int[] arr = {9,6,3,1,2,4,5};  
 int length = arr.length;  
 *selectionSort*(arr);  
 System.*out*.println("The sorted elements are:");  
 for(int i:arr){  
  
 System.*out*.println(i);  
 }  
 }  
  
 public static void selectionSort(int[] arr){  
  
 for(int i=0;i<arr.length-1;i++){  
  
 int index =i;  
 for(int j=i+1;j<arr.length;j++){  
 if(arr[j]<arr[index]){  
  
 index =j;  
 }  
  
 }  
 int smallNumber = arr[index];  
 arr[index]= arr[i];  
 arr[i]= smallNumber;  
 }  
  
 }  
}

36:BUBBLE SORT:

public class bubbleSort {  
  
  
 public static void main(String[] args){  
  
 int[] arr= {25,20,15,5,10};  
 *bubbleSort*(arr);  
 for(int i=0;i<arr.length;i++){  
  
 System.*out*.println(arr[i]);  
 }  
 }  
  
 public static void bubbleSort(int[] arr){  
 int len = arr.length;  
 int temp = 0;  
 for(int i=0;i<len;i++){  
 for (int j=1;j<(len);j++){  
 if(arr[j-1]>arr[j]){  
 temp = arr[j-1];  
 arr[j-1]= arr[j];  
 arr[j]= temp;  
  
 }  
  
  
 }  
  
 }  
  
 }  
37:INSERTION SORT:

public class insertionSort {  
  
 public static void main(String[] args){  
  
 int[] arr = {9,12,3,21,44};  
 *insertionSort*(arr);  
 for(int i=0;i<arr.length;i++){  
  
 System.*out*.println(arr[i]);  
  
 }  
 }  
 public static void insertionSort(int[] arr){  
  
 int len = arr.length;  
 for(int j=1;j<len;j++){  
 int key = arr[j];  
 int i=j-1;  
 while ((i>-1) && (arr[i]>key)){  
  
 arr[i+1]=arr[i];  
 i--;  
 }  
 arr[i+1]=key;  
 }  
  
 }  
}

38:MERGE SORT:

class MergeSort  
{  
   
 void merge(int arr[], int l, int m, int r)  
 {  
   
 int n1 = m - l + 1;  
 int n2 = r - m;  
  
 /\* Create temp arrays \*/  
 int L[] = new int [n1];  
 int R[] = new int [n2];  
  
 /\*Copy data to temp arrays\*/  
 for (int i=0; i<n1; ++i)  
 L[i] = arr[l + i];  
 for (int j=0; j<n2; ++j)  
 R[j] = arr[m + 1+ j];  
  
  
  
 int i = 0, j = 0;  
  
 int k = l;  
 while (i < n1 && j < n2)  
 {  
 if (L[i] <= R[j])  
 {  
 arr[k] = L[i];  
 i++;  
 }  
 else  
 {  
 arr[k] = R[j];  
 j++;  
 }  
 k++;  
 }  
 while (i < n1)  
 {  
 arr[k] = L[i];  
 i++;  
 k++;  
 }  
  
   
 while (j < n2)  
 {  
 arr[k] = R[j];  
 j++;  
 k++;  
 }  
 }  
 void sort(int arr[], int l, int r)  
 {  
 if (l < r)  
 {  
   
 int m = (l+r)/2;  
  
   
 sort(arr, l, m);  
 sort(arr , m+1, r);  
  
   
 merge(arr, l, m, r);  
 }  
 }  
  
 static void printArray(int arr[])  
 {  
 int n = arr.length;  
 for (int i=0; i<n; ++i)  
 System.*out*.print(arr[i] + " ");  
 System.*out*.println();  
 }  
  
 // Driver method  
 public static void main(String args[])  
 {  
 int arr[] = {12, 11, 13, 5, 6, 7};  
  
 System.*out*.println("Given Array");  
 *printArray*(arr);  
  
 MergeSort ob = new MergeSort();  
 ob.sort(arr, 0, arr.length-1);  
  
 System.*out*.println("\nSorted array");  
 *printArray*(arr);  
 }  
}

39: QUICK SORT:

class QuickSort  
{  
   
 int partition(int arr[], int low, int high)  
 {  
 int pivot = arr[high];  
 int i = (low-1); // index of smaller element  
 for (int j=low; j<high; j++)  
 {   
 if (arr[j] <= pivot)  
 {  
 i++;  
  
 // swap arr[i] and arr[j]  
 int temp = arr[i];  
 arr[i] = arr[j];  
 arr[j] = temp;  
 }  
 }  
  
 // swap arr[i+1] and arr[high] (or pivot)  
 int temp = arr[i+1];  
 arr[i+1] = arr[high];  
 arr[high] = temp;  
  
 return i+1;  
 }  
  
  
  
 void sort(int arr[], int low, int high)  
 {  
 if (low < high)  
 {  
  
 int pi = partition(arr, low, high);  
  
   
 sort(arr, low, pi-1);  
 sort(arr, pi+1, high);  
 }  
 }  
 static void printArray(int arr[])  
 {  
 int n = arr.length;  
 for (int i=0; i<n; ++i)  
 System.*out*.print(arr[i]+" ");  
 System.*out*.println();  
 }  
  
 // Driver program  
 public static void main(String args[])  
 {  
 int arr[] = {10, 7, 8, 9, 1, 5};  
 int n = arr.length;  
  
 QuickSort ob = new QuickSort();  
 ob.sort(arr, 0, n-1);  
  
 System.*out*.println("sorted array");  
 *printArray*(arr);  
 }  
}