Prime no →

```
import java.util.Scanner;
class primeno{
  public static void main(String args[]){
    int n=2;
    if (n<=1){
      System.out.println("Special no");
    }
    else{
     boolean a = true;
     for (int i=2; i<=n/2; i++){
        if (n%i==0){
          a=false;
          System.out.println("not a prime no");
          break;
        }
     }
     if (a){
       System.out.println("prime no");
     }
     else{System.out.println("not prime no");
     }
  }
}
```

Factors of a given no :-

```
class factor{
    public static void main(String args[]){
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter a number");
    int n = sc.nextInt();
    for (int i=1; i<=n; i++){
    if (n%i==0){
        System.out.println(i);
    }
  }
}
```

```
import java.util.Scanner;
class primeno{
    public static void main (String args[]){
        Scanner sc = new Scanner (System.in);
        System.out.println ("Enter a number");
        int n = sc.nextInt();
        if (n==-1){
            System.out.println(" no is special ");
    else{
        boolean a= true;
        for(int i=2 ; i<=n/2 ; i++){
            if (n%i==0){
                a=false;
                break;
    if (a)
        System.out.println ("a prime no");
    else{
        System.out.println ("Not a prime no");
```

—o—

```
class prime no {
    psvm(String args[]) {
        int n=2;
        boolean a=true;
        for(int i=2; i<=n/2 ; i++) {
            if (n%i==0){
                a=false;
                break;
    if(a)
        sop ("prime");
    else
        sop ("not prime");
```

prime no in a given range.

```
import java.util.Scanner;
class primeinrange {
    public static void main (String args[]){
     Scanner sc = new Scanner (System.in);
     System.out.println ("Enter two number");
     int a = sc.nextInt();
     int b = sc.nextInt();
     for (int i=a; i<=b; i++){
       if (i<=1) {
           continue;
       }
      boolean flag = true;
      for (int j=2; j<=i/2; j++){
        if (i%j==0)
        { flag = false;
        } break;
      }
    if (flag){
        System.out.println(i);
    }
}
}
```

—o—

```
import java.util.Scanner;
class primenogivenrange{
    public static void main (String [] args){
     Scanner sc = new Scanner (System.in);
     System.out.println ("Enter a range");
     int a = sc.nextInt();
     int b = sc.nextInt();
     for (int i=a; i<=b; i++){
       if (i<=1){
           continue;
       }
```

with that create object of innerclass

```
boolean flag=true;
for(int j=2; j<=i/2; j++){
  if(i%j==0){
    flag=false;
  }break;
}
if(flag){
  System.out.println(i);
}}
```

—o—

prime no in a given range highest prime no —

```
import java.util.Scanner;
class highestprimeno{
  public static void main(String args[]){
  Scanner sc=new Scanner(System.in);
  System.out.println("Enter a range");
  int a=nextInt();
  int b=nextInt();
  for(int i=a; i<=b; i++){
    if(i<=0){
      continue;
  boolean flag=true;
  for(int j=2; j<=i/2; j++){
    flag=false;
  }break;
}
if(flag){
  System.out.println(i);
  break;
}}}}
```

```
import java.util.Scanner;
class palindrome{
    public static void main(String args[]){
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter a number");
    int n = sc.nextInt();
    int res = 0; int temp = n;
    while(n>0){
        int rem = n%10;
        res = res * 10 + rem;
        n/=10;
    }
    if(res = temp){
        System.out.println("palindrome");
    }
    else{
        System.out.println("not palindrome");
    }
}
```

Reverse of a number →

```
import java.util.Scanner;
class reverse{
    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a number");
        int n = sc.nextInt();
        int res = 0;
    while(n>0){
        int rem = n%10;
        res = res * 10 + rem;
        n/=10;
        System.out.println(res);
    }
    System.out.println(res);
}
```

Automorphic number →

```
import java.util.Scanner;
public class automorphic {
    public static void main (String args [ ]) {
        Scanner sc = new Scanner (System.in);
        System.out.println ("Enter a number");
        int n = sc.nextInt ( );
        int sq = n*n;
        boolean flag = true;
        while (n>0) {
            if (n%10 != sq%10) {
                flag = false;
                break;
            }
            n/=10;
            sq/=10;
        }
        if (flag) {
            System.out.println ("Automorphic");
        }
        else {
            System.out.println ("Not an automorphic");
        }
    }
}
```

Automorphic in a given range →

```
import java.util.Scanner;
class autoinrange {
    public static void main (String args [ ]) {
        Scanner sc = new Scanner (System.in);
        System.out.println ("Enter a range");
        int a = sc.nextInt ( );
        int b = sc.nextInt ( );
```

```
public static boolean
    for (int i=a; i<=b; i++) {
        if (auto (i)) {
            System.out.println (i);
}   }   }

public static boolean auto (int n) {
    int sq = n*n;
while(n>0) boolean flag = true;
  { if (n%10 != sq %10) {
        flag = false;
        break;
    n/=10;
    sq/=10;
  }
  if (flag) {
      return true;
}   }
```

even or odd number →

```
import java.util.Scanner;
class evenodd {
    public static void main (String args [] ) {
        Scanner sc = new Scanner (System.in);
        System.out.println ("Enter a number");
        int n = sc.nextInt ();
        if (n%2 == 0) {
            System.out.println("even no");
        else {
            System.out.println ("odd no");
}   }   }
```

even or odd in given range →

```
import java.util.Scanner;
class evenodd {
    public static void main (String args[]){
    Scanner sc = new Scanner (System.in);
    System.out.println ("Enter a range");
    int a = sc.nextInt();
    int b = sc.nextInt();
    for (int i=a; i<=b; i++) {
        if (isevenodd (i)) {
            System.out.println (i);
    }   }   }
public static boolean isevenodd (int n){
        boolean a= true;
        if (n%2 != 0) {
            return false;
        } break;
    } return a;
}
```

Fibonacci series →

```
import java.util.Scanner;
public class fibonacci {
    Scanner sc = new Scanner (System.in);
    System.out.println ("Enter a numbe of series");
    int n = sc.nextInt();
    int a=0;
    int b=1;
    for (int i=1; i<=n; i++) {
        System.out.println(a);
        int c = a+b;
        a= b;
        b= c;
}   }
```

Neon number →

```
import java.util.Scanner;
public class neon{
    public static void main (String args [])
    Scanner sc = new Scanner (System.in);
    System.out.println ("Enter a number");
    int n= sc.nextInt();
    int sq= n*n;
    int sum=0;
    while (sq>0){
        int res = sq%10;
        sum += res;
        sq/ = 10;
    if (sum == n) {
        System.out.println ("Neon no");
    else{
        System.out.println ("Not a neon no");
}}
```

Neon no. in a given range →

```
import java.util.Scanner;
public class neoninrange {
    public static void main (String args [])
        Scanner sc = new Scanner (System.in);
        System.out.println ("Enter a range");
        int a = sc.nextInt();
        int b = sc.nextInt();
    for (int i=a; i<=b; i++){
        if (isneon (i)){
            System.out.println (i);
}}
public static boolean isneon (int n){
    int sq = n*n;
    int sum = 0;
```

```
while (sq > 0){
    int res = sq % 10;
    sum += res;
    sq /= 10;
}
return true; if (sum == n)
}
}
return false;
}
return false;
```

—o—

Alternative prime number →

```
Import Java.util.Scanner;
public class alternativeprime{
    public static void main (String args[]){
        Scanner sc = new Scanner (System.in);
        System.out.println ("Enter a range");
        int a = sc.nextInt();
        int b = sc.nextInt();
        int skip = 1;
    for (int i = a; i <= b; i++){
        if (isprime(i)){
            if (skip % 2 == 1){
                System.out.println(i);
            }
        skip++;
    }
public static boolean isprime (int n){
    if (n <= 1) { return false; }
    for (int i = 2; i <= n/2; i++){
        if (n % i == 0){
            return false;
        }
    }
    return true;
}
```

—o—

perfect number →

```
import java.util.Scanner;
pubic class perfect {
    public static void main (String args [ ]){
        Scanner se = new Scanner (System.in) ;
        System.out.println ("Enter a number");
        int n = sc.nextInt ( ) ;
        int sum = 0;
for (int i =1 ; i<=n; i++) {
        if (n% i = = 0 ) {
            sum += i ;
if (sum= = n ) {
    System.out.println (" perfect number ");
else {
    System.out.println ("Not a perfect number ");
```

perfect no. in a given range —

```
import java.util.Scanner;
class perfectinrange {
        pubic Static void main (String args [ ]){
        Scanner sc = new Scanner (System.in);
        System.out.println ("Enter a range");
        int a = sc.nextInt ( );
        int b = sc.nextInt ( );
for (int i = a; i<=b; i++) {
        if (isperfect (i)){
            System.out.println (i);
pubic Static boolean isperfect (int n) { int sum=0;
    for (int j=1 ; j<=n; j++ ) {
        if (n%j = = 0) {
            sum += j ;
if (sum = = n) {
        return true;
return false;
```

~~Perfect~~ Alternative perfect nos →

```
import java.util.Scanner;
public class alternateperfect {
    public static boolean isperfect(int n) {
    Scanner sc = new Scanner(System.in);
     System.out.println("Enter a range");
     int a = sc.nextInt();
     int b = sc.nextInt();
int int skip = 1; sum = 0; int sum = 0;
for(int i = a; i <= b; i++){
  if(n%i == 0){
      if skip sum += i;
  }
if(sum == n){
     System.out.println return true;
}
else {
} return false;
}

public static void main(String args[]){
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter a range");
    int a = sc.nextInt();
    int b = sc.nextInt();
    int skip = 1;
for(int j=a; j<=b; j++){
    if(isperfect(j)){
        if(skip % 2 == 1){
            System.out.println(i);
        }
    }
    skip++
}
}
```

Spy number →

```
import java.util.Scanner;
public class Spy {
    public static void main (String args[]){
    Scanner sc = new Scanner (System.in);
    System.out.println ("Enter the number");
    int n = sc.nextInt();
    int sum = 0;
        int mul = 1;
    while (n>0) {
     int res = n%10;
         sum + = res;
         mul * = res;
        } n/=10;
    if (sum = = mul) {
       System.out.println ("Spy no");
    } else {
    } } System.out.println("Not a spy no");
}
```

Alternative spy no →

```
import java.util.Scanner;
class alternatespy {
   public static boolean isspy (int n) {
    int sum=0 ; mul = 1;
    while (n>0) {
        int res = n%10;
            sum + = res;
            mul * = res;
        } n/=10;
    if (sum = = mul){
       return true;
    }
    return false;
}
```

```
pubic static void main (String args [])
  Scanner sc = new Scanner (System.in);
  System.out.println ("Enter a range");
  int a = sc.nextInt();
  int b = sc.nextInt(); int skip = 1;
for (int i = a; i<=b; i++){
    if (isPy (i)){
      if (skip %2 == 1){
        System.out.println (i);
      }
    skip ++;
}
```

Strong number →

```
import java.util.Scanner;
pubic class Strong {
    pubic static void main (String [] args){
      Scanner sc = new Scanner (System.in);
      System.out.println ("Enter a number");
      int n = sc.nextInt();
      int sum=0; int temp = n;
  while (n>0) {
      res = n%10;
      int prod = 1;
    for (int i = 1; i<= res; i++) {
        prod *= i;
      sum += prod;
      n/=10;
  if (sum == temp) {
      System.out.println ("Strong no");
  }
  else{
      System.out.println ("Not a strong no");
  }
}
```

Alternate Strong no in a given range →

```
import java.util.Scanner;
class alternatestrong {
    public static boolean isStrong(int n){
      int sum = 0; int temp=n;
      while (n>0){
          res = n%10;
          int prod = 1;
        for(int i=1; i<=res; i++){
          prod *= i;
           sum += prod;
        } n/=10;
if (temp == sum){
        return true;
}
    return false;
}
public static void main (String args []){
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter a range");
    int a=sc.nextInt();
    int b = sc.nextInt(); int skip=1;
  for (int i=a; i<=b; i++){
      if (isStrong(i)){
          if (skip%2==1){
        System.out.println(i);
          skip++;
}
}
```

—o—

Arthmstrong number →

```
import java.util.Scanner;
class anhmstrong {
    public static void main (String args [])
    Scanner sc = new Scanner (System.in);
    System.out.println ("Enter a no");
    int n = sc.nextInt();
    int temp = n;  int count = 0;
while (n > 0) {
        count ++;
    }   n /= 10;
temp n = temp;
int sum = 0;
while (n > 0) {
 int res = n % 10;
int prod = 1;
for (int i = 1; i <= count; i++) {
        prod *= i;
    }  sum += prod;  n /= 10;
if (temp == sum) {
  System.out.println ("Arthmstrong number");
  }
else {
   System.out.println ("Not an arthmstrong no");
  }
 }
}
```

Alternative armstrong no →

```
import java.util.Scanner;
public class alternatearmstrong {
public static boolean isarmstrong(int n){
    int temp = n;
    int count = 0;
while (n>0) {
    count ++;
  n /=10;
n = temp;
int sum = 0;
while (n>0){
    int res = n%10;
    prod = 1;
for (int i = 1; i<= count; i++){
    prod *= i;
  sum += prod;
if (temp == sum) {
    return true;
 return false;
public static void main (String args []){
 Scanner sc = new Scanner (System.in);
 System.out.println (" Enter a range");
 int a = sc.nextInt ();
 int b = sc.nextInt (); int skip=1;
for (int i = a; i<= b; i++) {
    if (isStrong (i)) {
        if (skip % 2 == 1) {
      System.out.println (i);
  } skip++;
}
```

Diamond pattern →

```
class diamond {
  public static void main (String args[]) {
    int n=7 ; sp=n/2 , st=1;
  for (int i=1; i<=n; i++){
      for( int j=1 ; j<sp ; j++){
          System.out.println(" ");
      for (int k=1 ; k<=st; k++) {
          System.out.println("*");
    if (i<=n/2) {
        sp--;
      st+=2;
    else {
        sp++;
        st-=2;
}}}}
```

Pyramid pattern →

```
class pyramid {
  public static void main (String args[]){
     int n=3 ; sp=n-1 ; st=1;
  for (int i=1; i<n ; i++){
     for (int j=1; j<=sp ; j++) {
         System.out.println(" ");
     }
  for (int k=1; k<=st; k++){
      System.out.println("*");
  }
  if (i<=n/2) System.out.println();
      sp--;
      st+=2;
}}
```

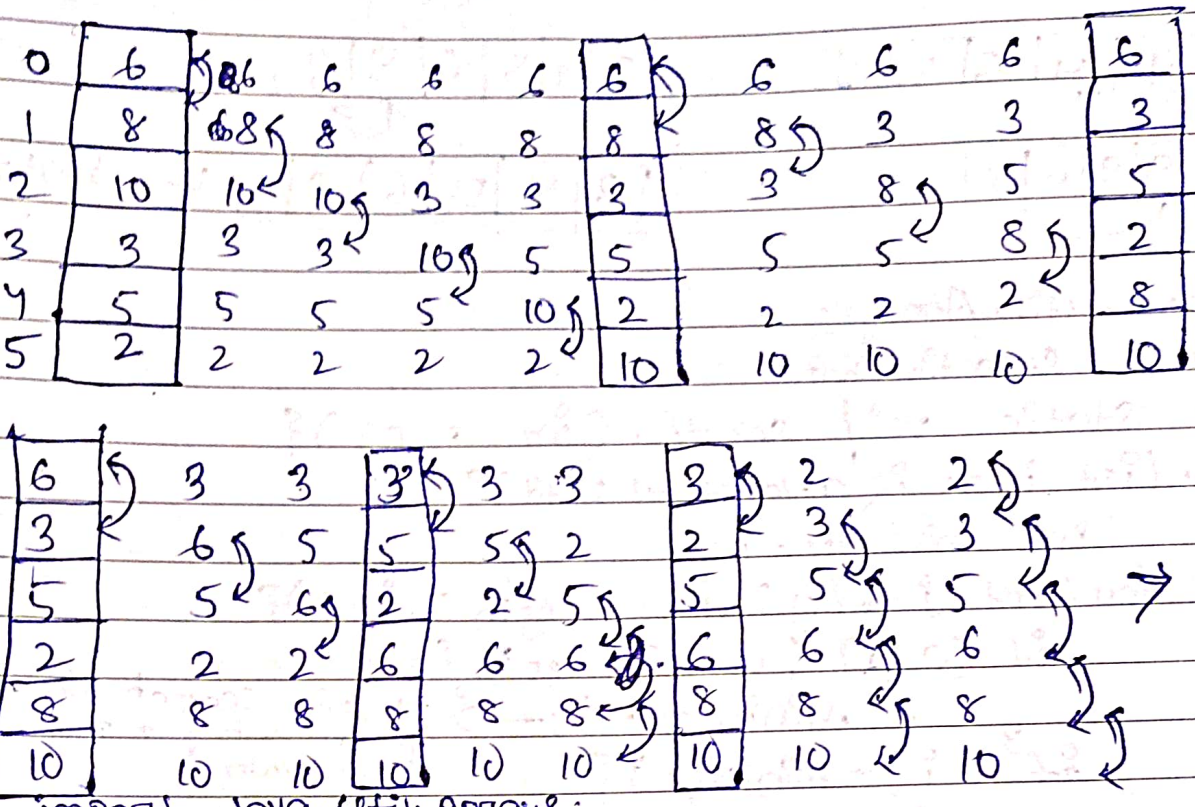
Number pyramid pattern →

```
class nopyramid {
public static void main(String args[]){
   int n=5; int st=1; int sp=n-1;
 for(int i=1; i<=n; i++){
   for(int j=1; j<=sp; j++){
     System.out.println(" ");
   int p=1;
   for(int k=1; k<=st; k++){
       if(k<=i){
          System.out.println(p);
       p++;
   else{
       System.out.println(p-1);
       p--;
   System.out.println();
   sp--;
} } st+=2;
}
```

```
    1
   1 2 1
  1 2 3 4 5
 1 2 3 4 5 6 7
1 2 3 4 5 6 7 8 9
```

Sorting algorithm →

Bubble Sort :-

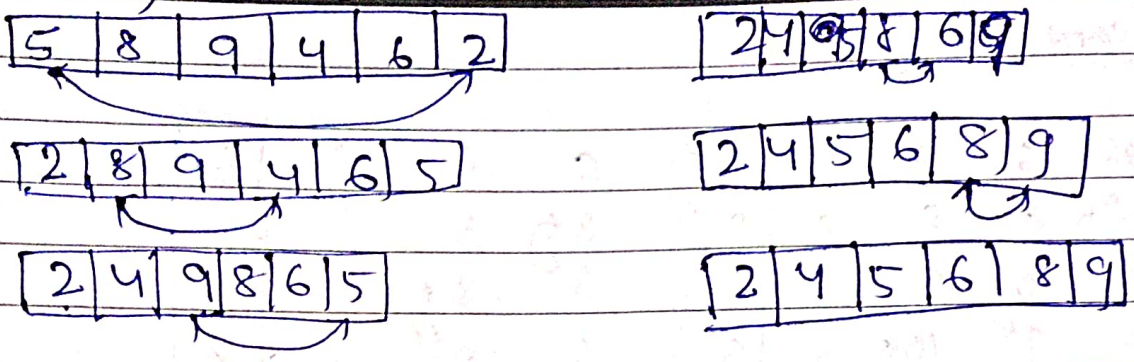


```
import java.util.Arrays;
class Bubblesort{
    public Steric void main(String [] args){
        int a [] = {3,2,5,6,8,10};
        for(int i=0; i<a.length-1; i++){
            for(int j=0; j<a.length-1-i; j++){
                if(a[j] > a[j+1]){
                    int temp = a[j];
                    a[j] = a[j+1];
                    a[j+1] = temp;
                }
            }
        }
        Sop(Arrays.toString(a));
```

```
import java.util.Arrays;
class Bubble{
    PSVM(--->){
int [] a={5,8,1,2,4,6};
SOP(Arrays.toString(a));
sort(a);
SOP(Arrays.toString(a));
```

```
pulic Steric void sort(int [] a){
for(int i=0;i<a.length-1;i++){
 for(int j=0; j<a.length-1-i; j++){
  int temp=a[j];
  a[j] = a[j+1];
  a[j+1] = temp;
```

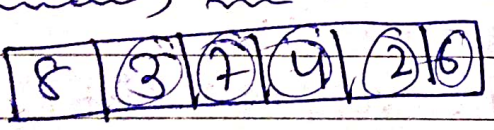
Selection Sort:-

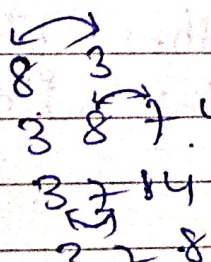


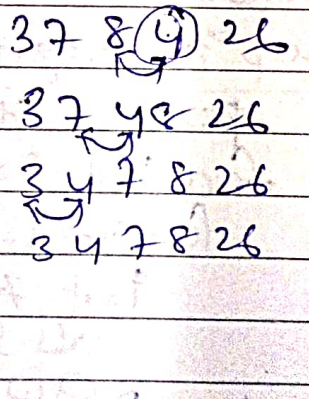

```
import java.util.Arrays;
public class selectionSort{
    public static void selection(int a[]){
        for(int i=0; i<a.length-1; i++){
            int minindex=i;
            for(int j=i+1; j<a.length; j++){
                if(a[j]<a[minindex]){
                    minindex=j;
                }
            }
            if(i!=minindex){
                int temp=a[minindex];
                a[minindex]=a[i];
                a[i]=temp;
            }
        }
    }
    psvm(--){
        int[] a={4,1,3,6,8,7,9,5};
        Sop(Arrays.toString(a));
        Sort(a);
        Sop(Arrays.toString(a));
    }
}
```

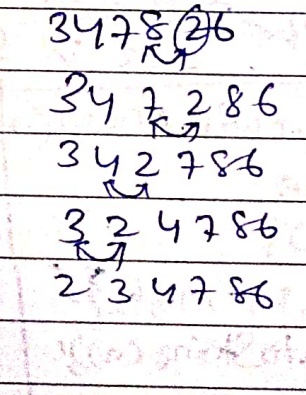
→ It will search min number in an array & print the minindex.

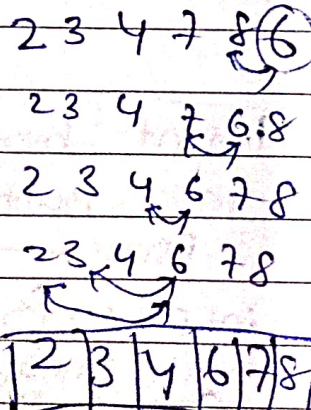
Insertion Sort:-





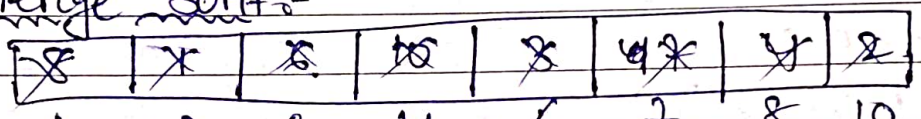


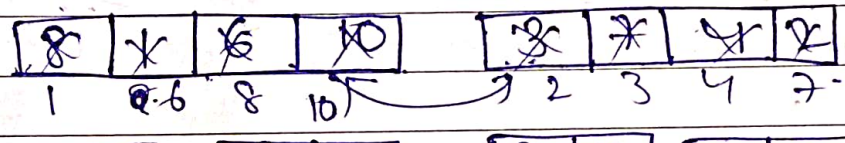


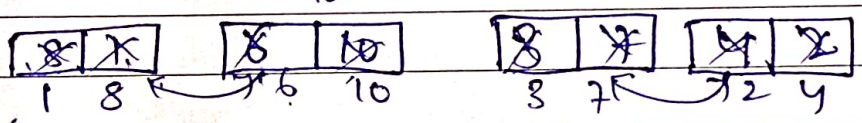


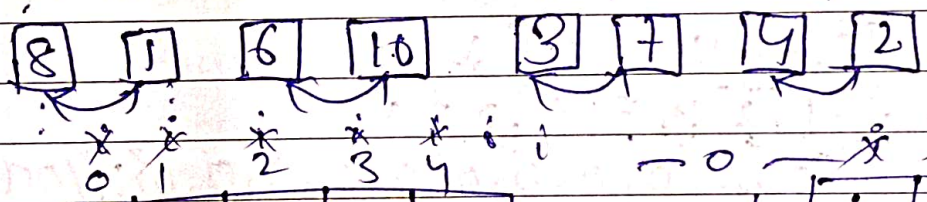
```
import java.util.Arrays;
class InsertionSort {
    public static void sort(int[] a){
        for(int i = 1; i<a.length; i++){
            int key = a[i];
            int j = i-1;
            while(j>0 && a[j]>key){
                a[j+1] = a[j];
                j--;
            }
            a[j+1] = key
        }
    }
    psvm(String[] args){
        int[] a = {4,1,3,6,8,7,9,5}
        sop(Arrays.toString(a));
        sort(a);
        sop(Arrays.toString(a));
    }
}
```

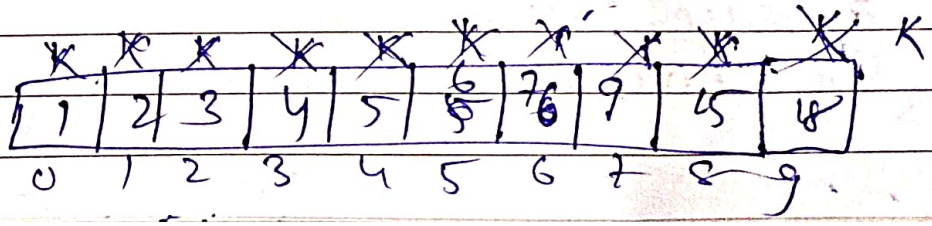
Merge sort:-











```
import java.util.Arrays;
Class Mergesort{
    Static void merge(int [] a, int b[], int [] c){
      int i=0; j=0; k=0;
    while (i<a.length && j<b.length){
      if (a[i]<b[j]){
          c[k] = a[i];
          i++;
        } k++;
        else{
            c[k] = b[j];
            j++;
            k++;
        }
    }
    while (i<a.length){
          c[k] = a[i];
          i++;
        k++;
    while (j<b.length){
          c[k] = b[j];
          j++;
        k++;
}
psvm(---){
    int a[] = {3,7,9,15,18};
    int b[] = {1,2,4,5,6,7};
    int c[] = {0,0,0,0,0,0,0,0,0,0,0,0};//or
    sop(Array.toString(a));
    merge(a);
    sop(Array.toString(a));
}
```

int c[] = new int[a.length + b.length];

⑥ Printing occurrence of each elements in given array
i/p = {7, 2, 2, 7} o/p = 2 time 7
2 time 2

⑦

```
class Change{
    psvm(———){
        int [] a = {0,5,0,7,9,0,8}; , int [] b = {0,0,0,0,0,0,0};
        for(int i=0; i<=a.length; i++){
            if (a[i] != 0){
                int temp = a[a.length-1];
                a[a.length-1] = a[i];
                a[i] = temp;
            if (a[i] > 0){
                b[i] = a[i];
            }
        }
```

o/p
{5,7,9,8,0,0,0}

⑤ Unique elements in given array i/p {7,1,7,2,2} o/p = 1 is unique element

② To remove duplicate element in given array

②③ i/p = {0,5,0,7,9,0,8} o/p = {0,0,0,5,7,9,8}

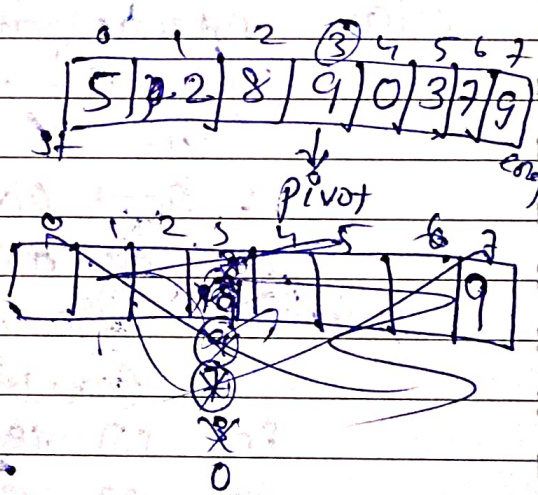
④ print matching subset in given array & count i/p = 7,3,2,7 o/p = 2 matching subset

```
class mergesort{
    psvm(—){
        int a[] = {9,2,6,7,9,0,4};
        sort(a);
        sop(Arrays.tostring(a));
    }
public static void sort(int [] a){
    if (a.length == 1) return;
    int [] left = new int [a.length/2];
    int [] right = new int [a.length - left.length];
    System.arraycopy(a, 0, left, 0, left.length);
    System.arraycopy(a, left.length, right, 0, right.length);
    sort(left);
    sort(right);
    merge(left, right, a);
}
```

o/p
{0, 2, 4, 6, 7, 9, 9}

Quick Sort :-

```
class Quick_sort{
    public static void sort(int []a, int st, int end){
    psvm(---){
        int a[] = {5,2,8,9,0,3,7,9};
        sort(a, 0, a.length-1);
    } sop(Arrays.toString(a));
public static void sort(int []a, int st, int end){
    if(st >= end)
        return;
    int i = st;
    int j = end;
    int pivot = (st+end)/2;
while(a[i] < a[pivot])
    i++;
while(a[j] > a[pivot])
    j--;
if(i <= j){
    int temp = a[i];
    a[i] = a[j];
} a[j] = temp;
i++;
j--;
sort(a, st, j);
} sort(a, i, end);
}
```



O/P

{0,2,3,5,7,8,9,9}

Linear search:-

O/P
index → 1

```
class Keysearch {
    psvm ( — ) {
        int a [] = {2,3,7,8}
        Scanner sc = new Scanner (System.in);
        sop ("Enter a key");
        int key = sc.nextInt ( );
    sop (search (a, 3));
public static int search (int [] a, int key) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] == key)
            return i;
    }
    return -1;
}
```

O/P
{2,3,6,7,9,9}

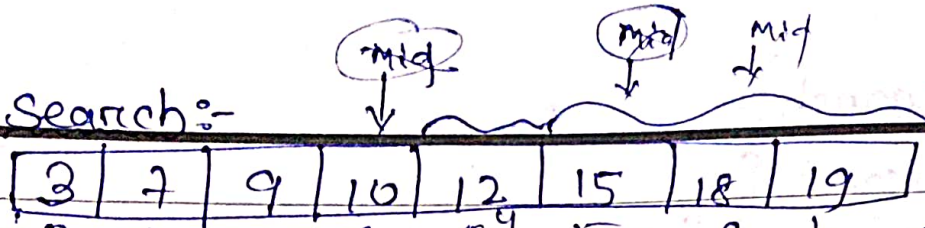
```
import java.util.Arrays;
class test {
    psvm ( — ) {
        int [] a = {3,6,9,2,7,9}
        Arrays.sort(a);
        sop (Arrays.toString(a));
    }
```

30-09-24

Quick sort →

```
public static void sort (int [] a, int st, int end) {
    if (st >= end) return;
    int i = st; int j = end; int pivot = a[(st+end)/2];
    while (i <= j) {
        while (a[i] < pivot) i++;
        while (a[j] > pivot) j--;
        if (i <= j) {
            int temp = a[i];
            a[i] = a[j];
            a[j] = temp;
            i++;
            j--;
        }
    }
    sort (a, st, j);
    sort (a, i, end);
}
```

Binary search:-



Key = 18

take 10 — 10 < 18 so it will be in right side

take 15 — 15 < 18 " " " " "

take 18 — 18 < 18 ✓

```
class Binarysearch{
    public static void main (String args [])
    int [] a = {2,4,6,7,8,9,12,13,14};
     sop (search (a, 13));
    } sop (search (a, 11));
  Static int search (int [] a, int key){
     int st = 0, end = a.length - 1;
    while (st <= end) {
        int mid = (st + end) / 2;
       if (a [mid] == key) return mid;
    else if (key < a [mid]) end = mid - 1;
    } else st = mid + 1;
    return -1;
} }
```

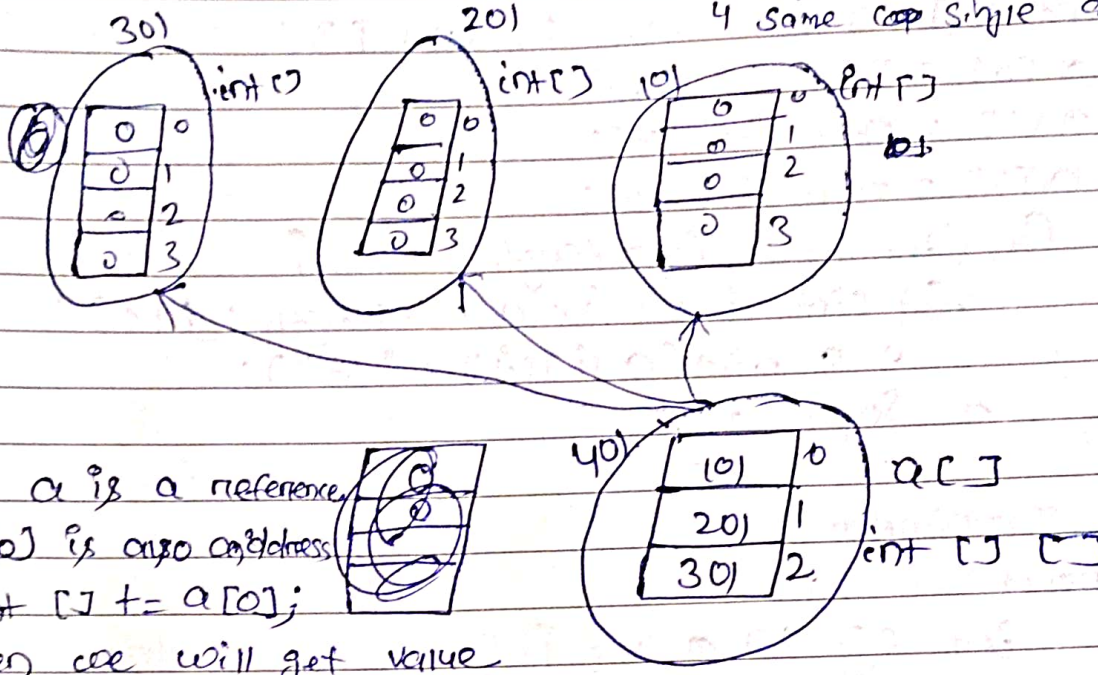
Here we are returning element so do not need any int type in method

```
class BinarysearchUsingRecursion{
    static int search (int [] a, int st, int end, int key){
      if (st > end) return -1;
       int mid = (st + end) / 2;
       if (key == a [mid]) return mid;
       else if (key < a [mid]) return search (a, st, mid-1, key);
       else return search (a, mid+1, end, key);
}
psvm (---){
    int [] a = {2,4,6,7,8,9,12,13,14};
    } sop (search (a, 0, a.length-1, 13));
}
```

Two Dimentional Array :- (Array of single Dimentional Array)

Syntax → `int [] [] a = new int [3] [4]`

3 single array size which having 4 same size single array.



Here a is a reference,
a [0] is also an address
So `int [] t = a [0];`
then we will get value
by t [0] or a [0] [0].

To print a 2D-Array →

```
int [] [] a = new int [3] [4];
for (int i=0; i<a.length ; i++) {
    for (int j = 0; j<a[i].length ; j++) {
        sop( a [i] [j] + "\t" );
    } sop( );
}
```

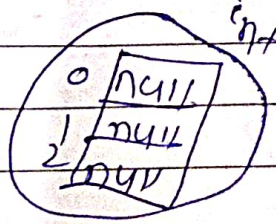
— o —

OR!! By using for each loop —

```
for (int [] t : a) {
    for (int n : t) {
        sop (n + "\t");
    } sop ( );
}
```

— o —

`int [] [] new int [3] [] →`



01-10-24

```
class TwoDimenArray{
  psvm(--){
  int [][] a=new int [3][];
  a[0]=new int [5];
  a[1]=new int [4];
  a[2]=new int [3];
  for(int i=0; i<a.length; i++){
    for(int j=0; j<a[i].length; j++){
      }Sop(a[i][j]+"\t");
  }  }  }Sop();
```

o/p

```
0 0 0 0 0
0 0 0 0
0 0 0
```

```
class TwoDimArr{
  psvm(--){
    int [][] a={
                {1,2,3},{4,5,6},{7,8,9}
               };
  for(int i=0; i<a.length; i++){
    for(int j=0; j<a[i].length; j++){
      }Sop(a[i][j]+"\t");
  }  }Sop();
```

o/p

```
1 2 3
4 5 6
7 8 9
```

```
class SumOfAll_theElement{
    psvm(--){
      int [][] a={{1,2,3},{4,5,6},{7,8,9}};
    for(int i=0; i<a.length; i++){
      for(int j=0; j<a[i].length; j++){
        }int sum+=a[i][j];
    }
    Sop(sum);
  }
}
```

o/p

```
45
```

```
class A {
  psvm(--){
  int [][] a={{1,2,3},{4,5,6},{7,8,9}};
  int sum=0;
  for(int i=0;i<a.length;i++){
  sum+=a[i][i];
  sop(sum);
}
}
```

O/P
1+5+9= 15

```
for(int i=0;i<a.length;i++){
  for(int j=0;j<a[i].length;j++){
    if(i==j){
      sum+=a[i][j];
}}
sop(sum);
```

```
class Diagonalsum{
  psvm(--){
    int [][] a={{1,2,3},{4,5,6},{7,8,9}};
    int sum=0;
    int len=a.length;
    for(int i=0;i<len;i++){
      sum+=a[i][i];
      sum+=a[i][len-1-i];
      if(i==len/2) continue;
      sum+=a[i][len-1-i];
}
sop(sum);
}
```

If we take 4 array

1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
O/P → 20

O/P

1+3+5+7+9
=25.

```
if(len%2!=0 && i==len/2) continue;
```

```
class Transposeofmatrix{
  psvm(--){
    int [][] a={{1,2,3},{4,5,6},{7,8,9}};
  int len=a.length;
  for(int i=0;i<len;i++){
  for(int j=0;j<len;j++){
    soprint(a[j][i]+"\t");
  }
  sop();
}
}
```

i/p
1 2 3
4 5 6
7 8 9

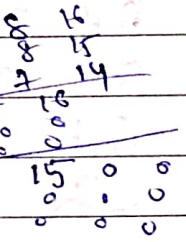
O/P
1 4 7
2 5 8
3 6 9

[row] [column] = [sum ...]

Multiply two matrix :-

02-10-24

```
Class MatrixMultiply {
    psvm (---){
        int [][] a = { {1,2,3}, {3,1,2}, {2,2,1} };
        int [][] b = { {3,1,3}, {3,2,2}, {2,1,4} };
        int [][] c = multiply(a,b);
    for (int [] temp : c){
        for (int n : temp){
            sopint(n+"\t");
        } sop();
    }
    static int [][] multiply (int [][] a, int [][] b){
        int len = a.length;
        int [][] c = new int [len][len];
    for (int i=0; i<len; i++){
        for (int j=0; j<len; j++){
            for (int k=0; k<len; k++){
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    return c;
}
```



Pascal Triangle →

```
class pascalTriangle{
    psvmc(--){
        int [][] a = pascalTriangle(5);
    for (int [] temp : a){
        for (int n : temp){
            sopint(n+"\t");
        } sop();
    }
    static int [][] pascalTriangle(int size){
        int [][] a = new int [size][];
        for (int i=0; i<a.length; i++){
            a[i] = new int [i+1];
```

O/P

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

```
for (int j=0; j<a[i].length; j++){
    if (j==0 || i==j){
        a[i][j]=1;
    }else{
        a[i][j]= a[i-1][j-1] + a[i-1][j];
    }
}
return a;
```

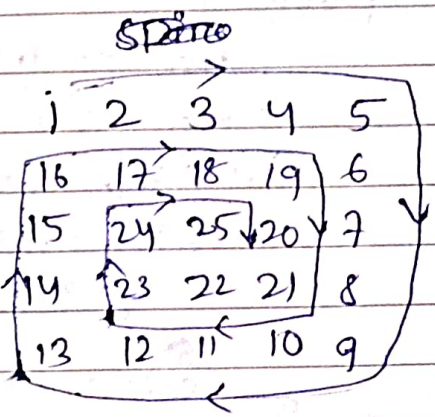
O/P

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

=

Spiral pattern :-

```
class Spiral{
    static int[][] Spiral(int size){
    int[][] a = new int[size][size];
    int r=0, c=-1;
    char m='r';
    for(int i=1; i<=size*size; i++){
        switch(m){
            case 'r':
                c++;
                a[r][c]=i;
                if(c==size-1-r) m='d';
                break;
            case 'd':
                r++;
                a[r][c]=i;
                if(r==c) m='l';
                break;
            case 'l':
                c--;
                a[r][c]=i;
            // if(c==0 || a[r][c-1]!=0) m='u';
                if(r+c==size-1) m='u';
                break;
```



```
case 'd':
    n--;
    a[n][c] = i;
if (n==1 || a[n-1][c] != 0)  n='n';
}
return a;
psvm ( ——— ) {
    int [][] a = {1,2,3,4,5,6,7,8,9,10,11,12,13};
    int [][] a = spiral (5);
for (int [] temp : a){
    for (int n : temp){
        sop(n+"\t");
    }
    sop( )
}
```

03-10-24

```
class Demo{
    psvm( ——— ){
    String [] a = new String [4];
    a[0] = "Java";
    a[1] = "Python";
    a[2] = "Java";
    a[3] = "C";
    for (String s:a){
        sop(s);
    }
}
```

creating objects
creating an array

```
public class Demos{
    psvm ( ——— ){
    Book [] a = new Book[3];
    a[0] = new Book(3);
    a[1] = new Book(10);
    a[2] = new Book(30);
    for (Book b:a){
        sop(b);
    }
}
```

```
public class Book {
    int pages;
    Book (int p) {
        pages = p;
    @override
    public String toString(){
    return "Book Pages= "
    +pages+"]";
}
```

```
interface Car {
    void start();
}
class BMW implements Car {
    @Override
    public void start() {
        Sop("BMW starts");
    }
}
class Benz implements Car {
    @Override
    public void start() {
        Sop("Benz starts");
    }
}
public class Demo6 {
    psvm(--) {
        Car[] a = new Car[4];
        a[0] = new BMW();
        a[1] = new Benz();
        a[2] = new Benz();
        a[3] = new BMW();
        for(Car c : a) {
            c.start();
        }
    }
}
```

Op:
BMW starts
Benz starts
Benz starts
BMW starts

Comparable Interface:-

→ It is in the lang package (java.lang.Comparable)
→ ~~It will compare the arguments~~
→ It is used to compare two objects
→ It provides a method compareTo() with one argument.
→ By compareTo() compares current instance/object with given instance/parameter.
→ It returns —
  (i) +ve number → If current instance is bigger than given parameter
  (ii) -ve number → If current instance is smaller than given parameter
  (iii) 0 → If both are equal.
→ The return type is "int" not a boolean.
→ By defaultly the compareTo() is abstract method ends with ";".

Ex:

java.lang.Comparable
int compareTo(Object arg);

A a = new A();

Compares → with → arg & a

~~74 — 80~~
80 − 74

```
public class BOOK implements comparable{
    int pages;
    BOOK (int p){
        pages = p;
    public int compareTO(Object arg){
// return this.pages - ((BOOK)arg).pages;
or
    BOOK bb = (BOOK) arg;//Downcasting
    if (this.pages == b.pages) return 0;
    else if (this.pages > b.pages) return 1;
    else return -1;
    @override
    public String toString(){
    return "BOOK [pages" +pages+"]";
}
public class Demo7{
    PSVM (— ){
    BOOK b1 = new BOOK(30);
    BOOK b2 = new BOOK(20);
    Sop (b1.compareTO(b2)); //10    //1
```

bcz always it gives the integer value so 1-6 = -6

ASCII value

So 74 − 80 = −6

```
    String s1 = "Java";
    String s2 = "Python";
    Sop (s1.compareTO(s2)); // -6
```

```
    String s1 = "a";
    String s2 = "B";
    Sop (s1.compareTO(s2)); // 31
    Sop (s1.compareToIgnoreCase(s2)); // -1
```

bcz capital B lies first then small a.

↳ If we have 2 different character

04-10-24

```
class Student implements comparable{
    String name;
    int marks;
    Student (String s, int m){
        name = s;
    } marks = m;
    public int compareTo (Object arg){
        Student s = (Student) arg;        sorting according to marks
        //return this.marks - s.marks;
    } return name.compareTo(s.name);      <- sorted according to name
    @override
    public String toString(){
    } return "Student [name = "+name+", marks = " +marks+"]";
}
```

```
class Demo8{
    PSVM (——){
    Student s = new Student ("Newton", 102);
    Student s1 = new Student ("Rakesh", 104);
    SOP (s.compareTo(s2));
    }
}
```

In java.util.lang package there are some ~~no~~ class with methods —

```
public class Arrays{
    psv sort (byte [] a)
    psv sort (short [] a)
    psv sort (String [] a)
    psv sort (int [] a)
    psv sort (Object [] a)
    PSV.sort (Boolean [] a) X -> It is not present in Arrays class
```

These are already exist in Arrays class.

```
class Test1{
    PSVM (——){
    Student [] arr = new Student [3];
    arr [0] = new Student ("shivam", 89);
    arr [1] = new Student ("nishi", 49);
    arr [2] = new Student ("Mahesh", 108);
    Arrays.sort (arr);
    for (Student s : arr){
        SOP (s);
    }
}
}
}
```

o/p	if marks sorted
Mahesh	108
shivam	89
nishi	49

o/p	if name sorted
Mahesh	108
nishi	49
shivam	89

obj1 & obj2 comparing —

ASC	Desc
+ve → obj1 > obj2	+ve → obj1 < obj2
-ve → obj1 < obj2	-ve → obj1 > obj2
0 → obj1 = obj2	0 → obj1 = obj2

```
class Test2{
    psvm(--){
    String [] arr={"Java", "python", "html", "c"};
    Arrays.sort(arr);
    for(String s:arr){
    }}sop(s);
}
```

↑ it have a comparable type

O/P
c
html
Java
python

05-10-24

```
class Account{
    psvm(--)
    double balance;
public Account(double b){
    balance=b;
public int compareTo(Object arg){
    Account a=(Account)arg;
    if(bal>a.bal) return 1;
    if(bal<a.bal) return -1;
}else 0;
@Override
public String toString(){
}return "Account [bal=" +bal+ "]";
```

or
if(bal<a.bal) return 1;
if(bal>a.bal) return -1;
print Decending order

It is in util package

java.util.Comparator
int compare(Object arg1, Object arg2);

If we will implement the comparision from one class to another class then we will use Comparator.

In this arg1 & arg2 will compare.
If arg1 > arg2 return 1;
If arg1 < arg2 return -1
If arg1 = arg2 return 0.

It will print or arrange according to their sites.

String is a comparable type

(Sort a string according to their site) ↓

```
Java.util.Comparator
int compare(Object arg1, Object arg2);
```

↑

```
LenComparator
@Override
public int compare(Object a, Object b)
{ String s1 = (String) a;
  String s2 = (String) b;
  return s1.length() - s2.length();
}
```

assume these a,b are String

```
class LenComparator implements comparator{
  public int compare(Object a, obj b){
    return ((String) a).length() - ((String) b).length();
}
public class test{
    psvm (--){
    String [] arr = {"Java", Python", "Sql", "c"};
    Arrays.sort(arr, new LenComparator());
    for(String s: arr){
      sop(s);
}}}
```

o/p
c
Sql
Java
Python

o/p

```
import java.util.Comparator;
public class emp implements comparable{
    String name;
    double sal;
    int id;
    static int num=100;
public emp(String n, double s){
    name = n;
    sal = s;
    id = num++;
}
@Override
public String toString(){
return name+"\t"+sal+
    "\t"+id;
public int compareTo(Object arg
{
return id-((emp)arg).id;
}}
```

```
import java.util.Arrays;
import java.util.comparator;

public class test1{
    psvm (--){
    emp [] a = new emp [3];
    a[0] = new emp ("Ra", 34999);
    a[1] = new emp ("Gopal", 24929);
    a[2] = new emp ("Sita", 12493);
// Arrays.sort(a); ← It will sort according to id.
//Arrays.sort(a, new Salcomp()); ← It will sort according to sal.
    Arrays.sort(a, new Namecomp());
for (emp e:a){
    sop(e);
}}}

class Salcomp implements comparator{
    public int compare(Object arg1, Object arg2){
        emp e1 = (emp) arg1;
        emp e2 = (emp) arg2;
    if (e1.sal > e2.sal) return 1; } we will do this bcz it is a double type.
    if (e1.sal < e2.sal) return -1
    }else return 0;
}

class Namecomp implements comparator{
    public int compare(Object arg1, Object arg2){
        emp e1 = (emp) arg1;
        emp e2 = (emp) arg2;
    return e1.name.compareTo(e2.name);
}
```

↖ use this bcz it is a string type

O/P

Gopal	24929
Ra	34999
Sita	12493

→ comparator is an interface used to compare two objects.
→ It provides a method compare() with two arguments
→ compare() compares 1st arg. with 2nd arg. & returns ⟶

+ve number → If 1st arg. > 2nd arg.
-ve number → If 1st arg < 2nd arg.
0 → If 1st arg = 2nd arg.

→ comparator is used to sort otherthan comparable implementation.

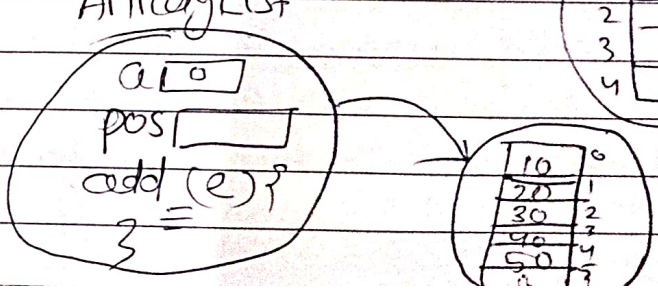
:- Data Structure :-

06/10/24

→ Data structure is a way of organising in a ~~struct~~ structured way. so that accessing data becomes easy & fast.

Ex-1

```
public class ArrayList {
    private object [] a;
    private int pos = 0;
    public ArrayList() {
    } a = new object [5];
public void add (object e){
    if (pos >= a.length) increase();
    ~~a[pos++]~~ a[pos++] = e;
void increase() {
    object [] temp = new Object [a.length];
for (int i=0; i< a.length; i++){
    } temp [i] = a [i];
    a = temp;
}
```



EX2

```
public class ArrayList {
    private Object [] a;
    private int pos;
public ArrayList (int initialsite){
    a = new Object [initialsite];
public void add (Object e) {
    if (pos >= a.length) increase ();
    a [pos ++] = e;
public void add (int index, Object e){
    if (index <= -1 || index >= size ()){
    throw new IndexoutofBoundException ();
    if (pos >= a.length) increase ();
    for (int i= size() -1; i>index; i--){
    a [i+1] = a [i];
    a [index] = e;
    pos ++;
private void increase (){
    Object [] temp = new Object [a.length + 3];
    for (int i=0; i<a.length; i++){
    temp [i] = a [i];
    a = temp;
public int size (){
    return pos;
public Object get (int index){
    if (index <= -1 || index >= size ()){
    throw new IndexOutofBoundException ();
    return a [index];
public void remove (int index){
    if (index <= -1 || index >= size ()){
    throw new IndexOutofBoundException ();
```

Exceptions are runtime errors

```
for (int i = index+1 ; i< size(); i++){
    a [i-1] = a [i];
}
a [--pos] = null;
```

```
public class test {
    psvm ( ) { ArrayList l = new ArrayList ( );
        l.add (10);
        l.add (2,70);
        l.add (80);
        l.remove (1);
    for (int i=0; i< i.size(); i++ ){
        int n = (Integer) l.get(i);
        Sop(n);
    }
}
```

Linked List :-

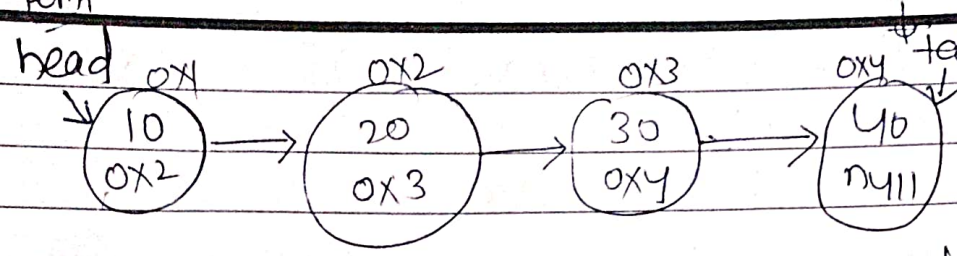
creating a node

```
class Node {
    Object ele;
    Node next;
Node(Object ele){
    this.ele = ele;
Node (Object ele, Node next){
    this.ele = ele;
    this.next = next;
public class test{
    Static Node head = null;
    psvm ( ){
        head = new Node (10);
        head.next=new Node(20);
        head.next.next = new Node(30);
        display ( );
```

07-10-24

```
pubic Static void display
{ Node temp = head;
while (temp != null){
  Sop(temp.ele);
  temp = temp.next;
}}
```

display or nodes




```
public class LinkedList {
    Node head;
    int count = 0;
public void add (Object ele){
    Node n = new Node (ele);
  if (head == null ){
      head = n;
      count ++;
   } return;
 Node temp = head;
while (temp.next != null) {
   } temp = temp.next;
  temp.next = n;
} count ++;
public int size (){
   } return count;
public void display (){
    Node temp = head;
    while (temp != null) {
        Sop (temp.ele);
   } } } temp = temp.next;
}
public class Test{
    psvm( — ){
LinkedList L = new LinkedList ();
L.add (10);
L.add (20);
L.add (30);
```

```
L.Reverse();
L.display ();
L.add (40);
L.add (50);
L.size ();
L.display ();
Sop (L.get (3)); //4
Sop (L.get (6)); // Except
L.Addfirst (5);
Sop (L.display ()); // 5 10 20 30 40 50
L.Remove (1);
L.display (); // 5 20 30 40 50
L.addAtposition(1,33);
L.display();
```

```
public Object get
        (int index)
if (index<0 || index>size()
throw new IndexOutOfBoundExce
 Node temp=head;
 for (int i=0; i<index; i++
 { temp = temp.next;
 }
return temp.ele;
}
public void Addfirst (Objec
 Node n = new Node(ele)
   n.next = head;
   head = n;
   count ++;
}
```

After addfirst() write this →

```
public void Remove(int index){ if(index<0 || index>size()-1) throw new IndexOutOfBoundException;
    if(index==0){
        head = head.next;
    } count--; return;
Node temp = head;
for(int i=0; i<index-1; i++){
    } temp = temp.next;
Node prev = temp;
prev.next = prev.next.next;
} count--;
```

```
public void addAtPosition(Object ele, int index){ Node n = new Node(ele);
    Node temp = head;
    for(int i=0; i<=index; i++){
        } temp = temp.next;
    Node prev = temp;
        n.next = prev.next;
        prev.next = n;
    }   count++;
```

08-10-24

```
public void Reverse(){
    Node current = head;
    Node prev = null, next = null;
    while(current != null){
        next = current.next;
        current.next = prev;
        prev = current;
    } current = next;
    head = prev;
}
```

if i/p 10 20 30 40

O/P

40
30
20
10

Double Linked List :-

```
public class Node{
    Object ele;
    Node next;
    Node prev;
public Node(Object e, Node n, Node p){
    ele = e;
    next = n;
}} prev = p;

public class DoubleLinkedList{
    Node head = null;
    int count = 0;
public void add(Object e){
    if (head == null){
        head = new Node(e, null, null);
        count ++;
    } return;
    Node curr = head;
    while (curr.next != null){
        curr = curr.next;
    }
    curr.next = new Node(e, null, curr);
    count ++;
} return;
public int size(){
    return count;
}
public void add(int index, Object e){
    if (index <= -1 || index > size()){
    } throw new IndexOutOfBoundException();
    if (index == 0){
        Node n = new Node(e, head, null);
        head.prev = n;
        head = n;
```

```
        count ++;
    } return;
    Node * curr = head;
    for (int i=1 ; i<index ; i++){
    } curr = curr.next;
    Node n = new Node (e, curr.next, curr);
    curr.next.prev = n;
    curr.next = n;
    } count ++;
public Object get (int index){
    if (index <=-1 || index >=size()) {
    } throw new IndexOutOfBoundException();
    Node curr = head;
    for (int i=1 ; i<=index ; i++){
    } curr = curr.next;
    } return curr.ele;
public void remove (int index){
    if (index <=-1 || index >=size()){
    } throw new IndexOutOfBoundException();
    if (index ==0){
        head = head.next;
        head.prev = null;
        count --;
    } return;
    Node curr = head;
    for(int i=1; i<index ; i++){
    } curr = curr.next;
    curr.next = curr.next.next;
    if(curr.next != null){
    curr.next.prev = curr;
    }
    count --;
}
}
```

```
public void display(){
for(int i=0; i<l.size();
    i++){
Sop(l.get(i)+" ");
}
}
```

```
public class Test{
    public static void main(String [] args){
    DoubleLinkedList l = new DoubleLinkedList();
    l.add(10);
    l.add(20);
    l.add(30);
    l.add(40);
    l.remove(3);
    l.display();
}
```

09-10-24

Queue :- (FIFO)

```
public class QueueUsingArray{
    Object [] a;
    int pos;
  QueueUsingArray(){
      a = new Object [5];
    pos = 0;
  public void add(Object e){
      if(pos >= a.length) increase();
    a[pos++] = e;
  private void increase(){
      Object [] temp = new Object [a.length+3];
      for(int i=0; i<a.length; i++){
        temp[i] = a[i];
    a=temp;
  public int size(){
      return pos;
  public boolean isEmpty(){
    return size() == 0;
```

```
public Object poll(){
    if (isEmpty()) return null;
    Object ele = a[0];
  for (int i=1; i<size(); i++){
    a[i-1] = a[i];
    pos--;
    a[pos] = null;
  } return ele;
 public Object peek(){
    if (isEmpty()) return null;
  } return a[0];
}

public class Test2{
    psvm(--){
     QueueUsingArray q = new QueueUsingArray();
     q.add(10);
     q.add(20);
     q.add(30);
     q.add(40);
    while(!q.isEmpty()){
     } } sop(q.poll());
    }
```

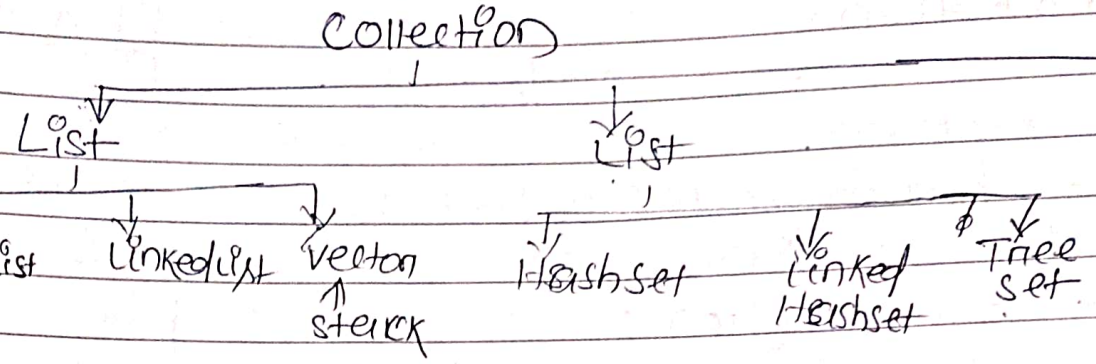
Stack :- ~~import java.util.EmptyStackException;~~

```
public class Stack{
    Node head = null;
     int count = 0;
    public void push(Object e){
      if(head == null){
        head = new Node(e, null);
        count++;
      } return;
```

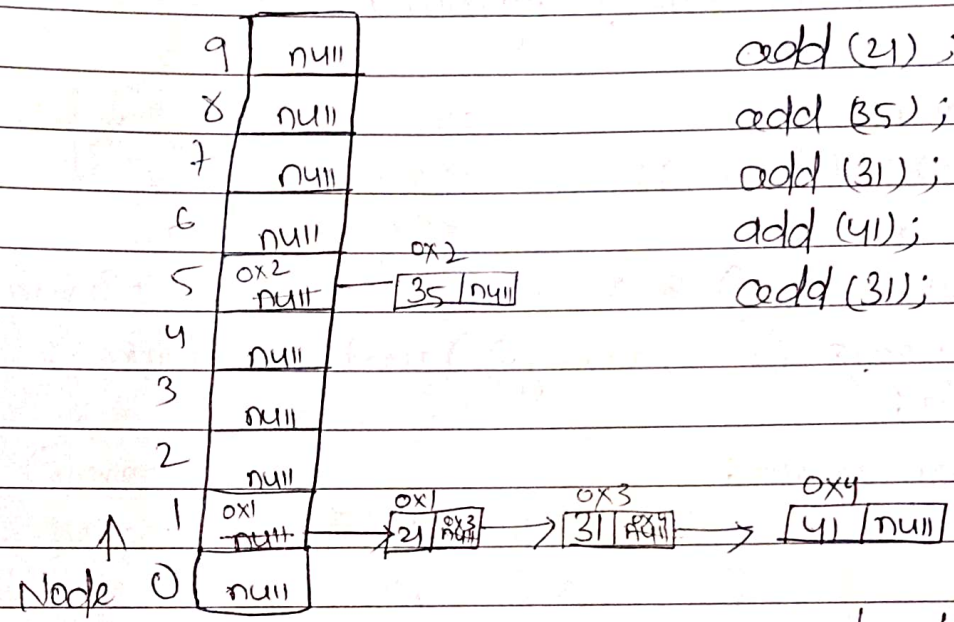
```
        head = new Node(e, head);
    } count++;
    public boolean isEmpty(){
        return count == 0;
    public int size(){
        return count;
    public Object pop(){
        if (isEmpty()){
            throw new EmptyStackException();
        Object ele = head.ele;
        head = head.next;
        count--;
        return ele;
    public Object peek(){
        if (isEmpty()){
            throw new EmptyStackException();
        return head.ele;
}


public class Test3 {
    psvm(---){
        Stack s = new Stack();
        s.push(10);
        s.push(20);
        s.push(30);
        while(!s.isEmpty()){
            sop(s.pop());
        }
    }
}
```

10-10-24



Hashset :- (Defaut capacity is 10)



add (21); 21 % 10 → 1
add (35); 35 % 10 → 5
add (31); 31 % 10 → 1
add (41); 41 % 10 → 1
add (31); we can't add. duplicate value.

(For Searching we can use hashset). very quickly loo

Resolving the collision → ① open addressing ② Separate chaining

For object type (Non-primitive type) –

```
package com.jsp.Hashing;
class Node{
    Object key;
    Node next;
public Node(Object key, Node next){
    this.key = key;
    this.next = next;
}
}
```

For int type (primitive)

→ int key

→ int key

Already present in Object class

```
Class HashsetImp {
    Node [] ann = new Node [10];
    int ccount = 0;
    public boolean add ( Object key) {
        int hc = key.hashcode();
        hc = Math.abs(hc);
        int index = hc % ann.length;
        if (ann[index] == null){
            ann[index] = new node (key, null);
            count ++;
        } return true;
        Node cann = ann[index];
        Node prev = null;
        while (cann != null) {
            if (key.equals (cann.key)) return false;
            prev = cann;
        } cann = cann.next;
        prev.next = new Node (key, null);
        count ++;
    } return true;
public int size () {
    } return count;
} public void display() {
    for (int i=0; i<ann.length; i++){
        if (ann[i] != null){
            Node cann = ann[i];
            while (cann != null) {
                Sop (cann.key);
                cann = cann.next;
            }
        }
    }
}
```

Absolute return value

For int type

int key

int index = key % ann.length

if (key == cann.key

```
public class Driver {
    psvm (——){
    HashSetImp hs = new HashSetImp();
    hs.add("Java");
    hs.add("c");
    hs.add("python");
    hs.add("c");
    sop(hs.size()); // 3
    hs.display(); // Java
                  // python
}}
```

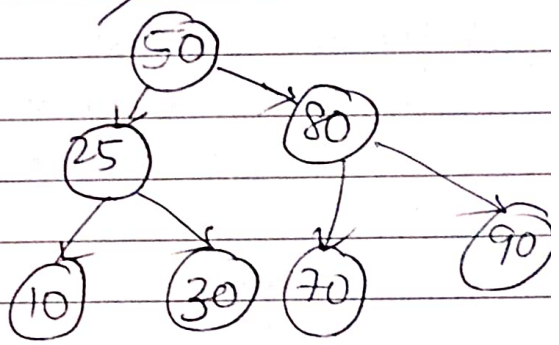
```
hs.add(10);
hs.add(20);
hs.add(10);
Sop(hs.size()); // 2
hs.display(); // 10
              // 20
}}
```

```
public class Student{
    String name;
    int marks;
public Student(String name, int marks){
    this.name = n;
    this.marks = m;
}
@override
public int hashcode(){
    return marks;
}
@override
public boolean equals(Object arg){
    if(!(arg instanceof Student)) return false;
    Student s = (Student) arg;
    return name.equals(s.name) && marks == s.marks;
}
@override
public String toString(){
    return "Student [name="+name+",marks="+marks+"]";
}}
public class Test{
    psvm(——){
    Student s1 = new Student("A", 98);
    Student s2 = new Student("B", 98);
    Student s3 = new Student("A", 98);
```

```
Sop (S1.equals(S3));   // true
Sop (S1.hashCode());   // 225679204
Sop (S3.hashCode());   // 225679204
HashSetImp S = new HashSetImp();
S.add (S1);
S.add (S2);
S.add (S3);
Sop ("Size = " + s.size());  // 2
```

16-10-24

Traversing in Binary Search Tree →



① Breadth-first traversing

(i) Level order Traversing.

50 25 80 10 30 70 90

② Depth-first Traversing

(i) pre-order Traversing

<root> <left> <right>

50 25 10 30 80 70 90

(ii) post-order Traversing

<left> <right> <root>

10 30 25 70 90 80 50

(iii) In-order Traversing

<left> <root> <right>

10 25 30 50 70 80 90

(i) Level-order traversing →

```
public void LevelOrder(){
    Queue q = new LinkedList();
    if (root != null) q.add(root);
while (!q.isEmpty()){
    Node n = (Node) q.poll();
    Sop(n.key + " ");
    if (n.left != null) q.add(n.left);
    if (n.right != null) q.add(n.right); } Sop(); }
```

```
BinarySearchTree b =
    new BinarySearchTree();
b.add(50);
b.add(25);
b.add(80,10,30,70,90);
b.levelOrder();
```

(i) Pre-order Traversing —

```
public void preOrder() {
    preOrder(root);
    Sop();
}
private void preOrder(Node n) {
    if (n == null) return;
    Sop(n.key + " ");
    preOrder(n.left);
    preOrder(n.right);
}
```

b.preorder();

(ii) Post-order Traversing —

```
public void postOrder() {
    postOrder(root);
    Sop();
}
private void postOrder(Node n) {
    if (n == null) return;
    postOrder(n.left);
    postOrder(n.right);
    Sop(n.key + " ");
}
```

b.postOrder();

(iii) In-order Traversing —

```
public void inOrder() {
    inOrder(root);
    Sop();
}
private void inOrder(Node n) {
    if (n == null) return;
    inOrder(n.left);
    Sop(n.key + " ");
    inorder(n.right);
}
```

b.inOrder();

— o —

Remove an element →

```
public void remove(int K) {
    root = removeNode(root, K);
}
```

```
b.inorder();
b.remove();
b.inorder();
```

```
private Node removeNode(Node n, int K){
    if (n==null) return null;
    if (K<n.key){
        n.left = removeNode(n.left,K);
    }
    else if (K>n.key){
        n.right = removeNode(n.right,K);
    }
    else{
        if (n.left==null && n.right==null){ // if there is no nodes
            n=null;
        }
        else if (n.left==null){ // if there is a right node present
            n=n.right;
        }
        else if (n.right==null){ // if there is left node present
            n=n.left;
        }
        else{ // if there are left & right node is present.
            Node maxNode = getmaxNode(n.left);
            int temp= maxNode.key;
            maxNode.key = n.key;
            n.key = temp;
            n.left = removeNode(n.left,K);
        }
    }
    return n;
}
Node getmaxNode(Node n){
    if (n.right==null) return n;
    return getmaxNode(n.right);
}
```

15-10-24

Tree Datastructure →

```
public class BinarySearchTree{
    private Node root = null;
    private int count = 0;
    private boolean flag;
    public boolean add(int K){
```

```
flag = true;
root = addNode(root, K);
} return flag;
private Node addNode(Node n, int K){
  if(n == null){
  n = new Node(K);
  count++;
  } return n;
if(K < n.key){
 } n.left = addNode(n.left, K);
else if(K > n.key){
 } n.right = addNode(n.right, K);
else{
 } flag = false;
}
```

—o—

Q:- what is hashing?

Ans:- Hashing is a process of generating index of fixed size from an i/p using hash() method.

Q:- what is hash table?

Ans:- In hashing we use a special structure called hash table.

→ Hash table maps a key to an index generated by hash() func.

Q:- what is collision in hashing?

Ans:- Sometimes hash() method can generated same index for more than one key. This state is called collision in hashing.

→ There are 2 ways of resolving collision in hashing —
① Separate chaining ② Open addressing.

→ In separate chaining a linked list is created in the identified index.

→ In open addressing, if already key is stored in identified index then we check for next free memory

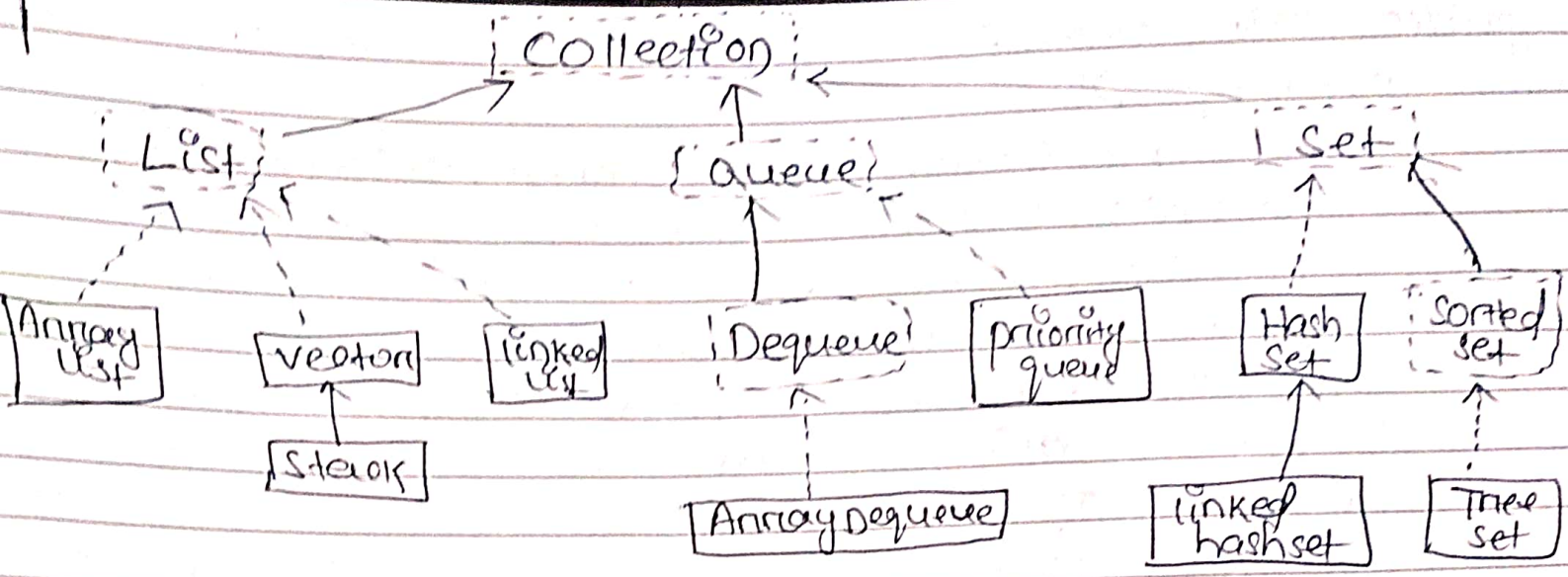
location in hashtable.
→ We can find the next free memory creation in 3 ways
① linear probing ② Quadratic probing ③ Double hasing.
→ In linear probing, free memory location is reached sequentially from the identifying index.
→ In quadratic probing ith sequence (i^2) is added each time for the indexfied index until we use find empty memory location.
→ In Double hashing we use a 2nd hashfun to identify free memory location.
Q:- contract b/w equals() & hashcode() methods?
→ If 2 objects are equal then they should have same hashcode.
→ If 2 objects ~~are~~ have same hashcode then they may or may not be equal.

Collection Framework:- 17-10-24
→ It is a readymade architecture which is having predefined classes & interfaces.
Diff. b/w Array & collection →

Array	collection
→ It will stores homogeneous only type data.	→ It can store both homogeneous & heterogeneous type data.
→ Array size is fixed size	→ collection is dynamic size (size is not fixed).
→ Array does not have any ~~func~~ pre-defined func^{n}s.	→ collection having pre-defined func^{n}s.

Q:- convert no. to string.
① i/p → 7293 . output → Seven thousand two hundred ninety Three
② To check given string is balanced string or not
{[]} ✓ {][} ✗ {[()]} ✓

→ Interface → class → extends ⇢ implements



Some important methods of collection interface —
→ add(), addAll(), remove(), removeAll(), retain(), retainAll(), clear(), size(), iterator().

List :-
→ It is a pre-defined sub-interface of collection interface
→ List present in java.util package.
→ List introduced at JDK 1.2.
Specification of List interface →
→ List follows insertion order.
→ List will allow to store "duplicate" elements.
→ List will allow to store "null" values.
We can't create obj. for list but subclass we can create

```
new ArrayList();
new Stack();
List l = new Vector();
new LinkedList();
```

```
public class Demo{
  psvm(--){
ArrayList a = new ArrayList();
a.add(10);
a.add(20);
a.add(30);
sop(a);//10,20,30
sop(a.contains(20));true
sop(a.isEmpty());//false
```

```
ArrayList a1 = new ArrayList();
a1.addAll(a);
sop(a1);//10,20,30
a1.add(500);//
sop(a1);//10,20,30,500
a1.removeAll(a);
sop(a1);//500
}
```

```
public class Demo1{
    psvm(--){
        List a = Arrays.asList(10,30,20,50,70,60,40);
        Sop(a); // 10,30,20,50,70,60,40
        Collections.sort(a);
        Sop(a); // 10,20,30,40,50,60,70
        List al = Arrays.asList(10,20,30,50,70,60,40);
        ArrayList al = new ArrayList();
        al.add(10);
        al.add(20);
        al.add(30);
        al.add(40);
        ArrayList a2 = new ArrayList();
        a2.add(10);
        a2.add(22);
        a2.add(34);
        a2.add(340);
        a2.retainAll(a1);
        Sop(a2); // 10,40
    }
}
```

ArrayList :- 17-10-24

→ It is an implementing sub class of List interface.
→ It present in java.util package.
→ It is introduced at JDK 1.2 version.
→ ArrayList initial capacity is 10.
→ Incremental capacity is 50% of its previous size.
→ ArrayList internally stores the element in the form of array of the type object.
→ It having 3 constructors —
→ Arraylist implements interfaces —
  (i) Serializable interface (ii) Random access
  (iii) Clonable interface
→ It is a multi threaded class

Arraylist specification —

→ All List specifications are applicable for Arraylist.

Vector :-

→ It is the implementing sub class of List interface.
→ It is present in java.util package.
→ Vector introduced at JDK 1.0 (Legacy class)
→ Its initial capacity is 10.
→ Incremental capacity is current capacity * 2 (10→20→40...)
→ Vector implements clonable, Serializable, random access interface
→ It having 4 overloaded constructors.
→ Its methods are Synchronized.
→ It is a single threaded class.

Arraylist

→ It implementing sub class of List.
→ It introduced at JDK 1.2.
→ Current capacity = $\left(\frac{cc * 3}{2}\right) + 1$ (incremental capacity)
→ Having 3 constructors.
→ This methods are not synchronized methods.
→ It supports multithreading.
→ It performs faster

Vector

→ It implementing sub class of List.
→ It introduced at JDK 1.0.
→ cc
→ incremental capacity = cc * 2.
→ Having 4 constructor.
→ This methods are synchronized methods.
→ It supports single threaded.
→ It performs slower.

18-10-24

Generics :- < >

```
public class Test {
    psvm(——){
    ArrayList<Integer> a = new ArrayList<Integer>();
    a.add(10);
    a.add(20);
    a.add(30);
    // a.add("JSP"); CTE
    sop(a); // 10, 20, 30
}
```

mandatory if we want only integer type

not mandatory

→ By default these are the non-primitive datatype

→ Hashset initial capacity is 16% & load factor at full ratio is 75%

Generic:-

→ Generic types are important features in collection.
→ These are introduced at JDK 1.5 version.
→ Generics will not support for array.
→ Generics are used to define which type of data can be stored in a collection.
→ Generic types represented by using "< >".
→ It supports for a non-primitive type.

Uses & advantages:-

→ Generics are used to achieve type safetyness.
→ Using generic types we can avoid unnecessary downcasting.

Stack:-

→ It is an implementing subclass of List interface.
→ Stack present in java.util package.
→ Stack introduced at JDK 1.2 version.
→ It follows the order Last In First Order (LIFO)
→ It is not index based collection.
→ It allows to store duplicate elements.
→ Stack can allows to store null values.

```
public class Test{
    psvm(--){
    Stack s = new Stack();
    s.push(10);
    s.push(20);
    s.push("Jsp");
    s.push("Qsp");
    s.push(null);
    sop(s.pop()); //null
    sop(s.pop()); // Qsp
    sop(s.size()); //3
    sop(s.peek()); //Jsp
}}
```

Assignment Ans

BalancedString

```
public class BalancedString {
    psvm(——){
    sop(isBalanced("{[()]}"));
public static boolean isBalanced(String s){
    Stack<Character> stack = new Stack<Character>();
for(int i=0; i<s.length(); i++){
    char c = s.charAt(i);
if(ch=='{' || ch=='[' || ch=='('){
    stack.push(ch);
else if(ch=='}' || ch==']' || ch==')'){
    if(stack.isEmpty()){
        return false;
    char ch2 = stack.pop();
    if(!ispair(ch2, ch)){
    return false;
    return stack.isEmpty();
public static boolean ispair(char ch1, char ch2){
 if(ch1=='{' && ch2=='}' || ch1=='[' && ch2==']' ||
        ch1=='(' && ch2==')'){
        return true;
    else
        return false;
```

—o—

LinkedList :-

→ It is an implementing subclass of List interface.
→ It is present in java.util package.
→ It is introduced at JDK 1.2 version.
→ It will stores the data in the form of nodes. Every node connected to the next node.
→ It have 2 overloaded constructors.

→ It implements Clonable & Serializable interfaces.
→ It priserves insertion order.
→ It allows to store duplicate elements.
→ null can be insert.
→ It doesnot have any initial capacity. It is contineous block of memory.
→ It doesnot have any shift operations.

```
public class Test2{
    psvm (——){
    List<String> l = new LinkedList<String>();
    l.add("Newton");
    l.add("Binesh");
    }} Sop(l); // Newton, Binesh.
```

ArrayList	LinkedList
→ It stores the data in the form of "Object" type array.	→ It stores the data in the form of "nodes".
→ Initial capacity & incremental capacity is applicable for arraylist.	→ NO initial capacity & NO incremental capacity for linkedlist.
→ It having 3 overloaded constructors	→ It having 2 overloaded constructors
→ It has a shift operation	→ It donot have shift operation.

Comparable<>	Comparator<>
→ Single Sorting sequence can achieve.	→ Multiple Sorting sequence can achieve
→ It is in java.lang.package	→ It is in java.util package.
→ It effects orginial class	→ It not effects original class
→ It have one parameter in the method.	→ It have 2 parameter in the method.
→ It is used for default sorting order (Natural sorting order)	→ It supports customized Sorting order
→ It having compareTo().	→ It having Compare().

19-10-24

Queue:-

→ Queue is an implementing a sub interface of collection interface.
→ Queue present in java.util package.
→ It is introduced at JDK 1.2 version
→ It follows the order First In First Out.
→ It is having implementing subclasses those are —
  → LinkedList → ArrayDequeue → priority Queue
→ Queue allows to stores duplicate values.
→ We can insert "null" values in Queue.

Queue

→ It is a single ended Queue.
→ In this, we can add or remove element from only one end.

Dequeue.

→ It is a double ended queue.
→ In this, we can add or remove elements from both ends.

priority Queue → It will adds the elements based on priority.

```
import java.util.ArrayDeque;
import java.util.Queue;
class Test {
    psvm (—— ){
    Queue<Integer> q = new ArrayDequeue<Integer>();
    q.add(10);                    //or// LinkedList<Integer>();
    q.add(20);
    q.add(30);
    q.add(40);
    Sop(q.poll()); // 10
    Sop(q.peek()); // 20
    Sop(q); //(20, 30, 40)
    }
}
```

Assignment ans — ② 76 → Seventy six.

Number to word →

```
public class NumToWord{
    PSVM(--){
    int num=77657678;
  pw(num/10000000," crone");
  pw((num/100000)%100," Lakhs");
  pw((num/1000)%100," thousand");
  pw((num/100)%100," hundred");
  pw(num%100,"");
}
public static void pw(int num,String s){
  String[] one={"","one","Two","Three","Four","Five","Six",
"seven","eight","nine","Ten","eleven","Twelve","Thirteen","Fourteen",
"Fifteen","Sixteen","seventeen","eighteen","ningteen"};
  String[] two={"","twenty","thirty","fourty","fifty","sixty",
"seventy","eighty","ninety"};
  if(num<=19)
    Sop(one[num]);
  else
    Soprint(two[num/10]+" "+one[num%10]);
  if(num!=0)
    Soprint(s);
}
```

O/P

Seven crone seventy six lakhs fifty seven thousand six hundred seventy eight.

—o—

Set :-

→ Set is a sub-interface of collection interface.
→ Set present in java.util package.
→ It introduced at JDK 1.2 version.
→ It having implementing sub classes those are — hashset, linked hashset, Tree set.

Specification :-

→ Set not priserving insertion order.
→ Set will not allow to store duplicate elements.
→ Set will allow to store only one null value.

List

→ It preserves insertion order
→ It will allows duplicates elements
→ It will store null values.

Set

→ It not preserves insertion order.
→ It will not allow duplicate elements
→ It will store only one null value

HashBased collection:-

→ Hashset → LinkedHashset → HashTable
→ Hashmap → LinkedHashmap.

Q:- Find length of string without using any string inbuilt func.

```
public class Test4{
    psvm(---){
    Scanner s = new Scanner ("Java is very easy");
    s.useDelimiter(" ");          // s.useDelimiter(" ");
    int count = 0;
    while (s.hasNext()){
        count ++;
        s.next();
    }
    Sop (count);  // 17 (characters)   Sop (count); // 4 (words)
}
```

Same

21-10-24

Hashset:-

→ It is implementing sub class of set interface.
→ It is present in java.util package.
→ It is introduced at JDK 1.2 version.
→ Hashset stores the elements based on hashcode.
→ It is having 4 overloaded constructors.
→ Hashset implements clonable & serializable interfaces.
→ It is not a index-based collection.
→ It will not preserves insertion order.
→ It will not allow to store any duplicate elements.
→ It will allow to store only single null value.
→ In hashset, search operations will happen based on hashcode.
→ Its initial capacity is 16%
→ Load fector or fill ratio is 75%

→ Collection will not store any primitive value
→ Collection is pure object oriented

```
public class Test {
    psvm ( ——— ) {
    Set <String> s = new HashSet <String> ();
    S.add ( "Ram" );
    S.add ("sita");
    S.add ("Laxman");
    S.add ("null");
    S.add ("Ravan");
    S.add ("Ram");
    S.add (null);
    Sop (S); // null, Ravan, Laxman, sita, Ram.
```

sita, Laxman

LinkedHashSet:-

→ It is ~~same~~ implementing sub class of Set interface
→ It is introduced at JDK 1.4 version.
→ It preserves the insertion order
→ It is present in java.util package.

public c+

Remove duplicate values in given array.—

```
public class Test2 {
    psvm ( ——— ) {
    int [] a = {10, 20, 30, 20, 10};
HashSet <Integer> s = new LinkedHashSet <Integer> ();
    for(int n : a)
        S.add(n);
    } Sop(S); // 10, 20, 30
```

Tree Set:

→ It is implementing sub class of Set interface.
→ It is present in Java.util package.
→ It is introduced at JDK 1.2 version.
→ It is not a hashbased collection.
→ It is developed using balanced tree.
→ It will stores only homogeneous type data.
→ It will not store single null value also.

→ Mainly
Sort
publi
3 3
Q: Rem
Stri
Iterat
→ It
→ It
→ It
list
→ It
→ By
in
→ met
→ The

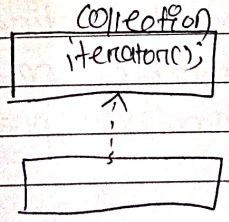
→ Mainly tree set is used for default sorting order.
for num → [0-9]
for char → [A-Z, a-z]

Sort the element without using any fun() the we use tree set

```
public class Test3{
    psvm(---){
    Set<String> s = new TreeSet<String>();
    s.add("Newton");
    s.add("Alin");
    s.add("Riya");
    }}
    sop(s); // Alin, Newton, Riya.
```

Q:- Removing the ~~words in~~ duplicate words in a given String without using split() method (use delimeter)

Iterator:-
→ It is an interface present in java.util package
→ It is introduced at JDK 1.2 version.
→ It is used to iterate any type of collection like List, Queue, Set.
→ It is also known as universal cursor.
→ By using iterator we can iterate a collection only in forward direction.
→ methods of Iterator → public boolean hasNext();
public Object next();
public void remove();
→ The return type is of Iterator.
Iterator iterator()



iterator() is an abstract method in iterator and collection interface

all of the interface which are implements the collection they will override iterator().

```
public class Test4{
    psvm(---){
    List<Integer> a = new ArrayList<Integer>();
    for(int i=10; i<=20; i++){
        a.add(i);
    Iterator<Integer> i = a.iterator();
    while(i.hasNext()){
        sop(i.next()); // 10,11,12,13,14,15,16,17,18,19,20
```

```
public class Test4{
    psvm(---){
    Set<String> s = new HashSet<String>();
    s.add("Ram");
    s.add("Sita");
    s.add("Ravan");
    Iterator<String> i = s.iterator();
    while(i.hasNext()){
        String str = i.next();
        if(str.equals("Ravan"));
        i.remove();
    sop(s); // Ram, Sita
```

List Iterator :-

- It is a Sub-interface of iterator.
- It is present in java.util package
- It is introduced at JDK 1.2 version.
- By using list iterator, we can iterate only list type collection not set & Queue.
- By using list iterator, we can iterate a collection in both forward & backward direction.
- List Iterator methods → hasNext(), next(), remove(), hasPrevious(), previous(), nextIndex(), previousIndex().

```
public class Tests{
  psvm(--){
List<Integer> a= Arrays.asList(10,20,30,40);
ListIteration<Integer> L = a.listIteration(); or// a.listiteration(a.size());
while(L.hasNext()){
  Sop(L.next()); // 10,20,30,40)
  Sop(=====
while(L.hasprevious())
  Sop(L.previous()); //30,20,10
```

// 10,20

24-10-24

```
public class Test{
  psvm
  Map <ch,st> m = new LinkedHashmap()
  for( i=0; i<=
pubic class OccaranceOfchar{
  psvm(--){
  String s="banana";
Map <ch,String> m = new LinkedHashMap<character, String>();
for(int i=0; i<s.length(); i++){
  char ch = s.charAt(i);
  if(!m.containKey(ch)){
    m.put(ch, i+"");
  else{
    String str = m.get(ch);
    m.put(ch, str+","+i);
  Sop(m);
}
```

dp

b=0

a=1,3,5

n=2,4

1=I 10=X 400=CD

4=IV 40=XL 500=D

5=V 50=L 900=CM

9=IX 90=XC 1000=M

100=C

Number → Roman number

```
pubiic class IntToRoman{
    psvm(--){
        Int num = 1225;
    } Sop(convert(num));
pubiic static String convert(int num){
    Map<Integer, String> m = new LinkedHashMap<Integer, String>();
    m.put(1,"I");
    m.put(4,"IV");
    m.put(5,"V");
    m.put(9,"IX");
    m.put(10,"X");
    m.put(40,"XL");
    m.put(50,"L");
    m.put(90,"XC");
    m.put(100,"C");
    m.put(400,"CD");
    m.put(500,"D");
    m.put(900,"CM");
    m.put(1000,"M");
int[] a = {1000,900,500,400,100,90,50,40,10,9,5,4,1};
String s = "";
for(int value : a){
    while(num >= value){
        st = m.get(value){
    } } num -= value;

Return s;
} }
```

ip 1225

op MCCXXV

Exception Handling :-

→ **Exception** :- It is an interuption, it will occurs during execution of a program.

→ When exception will occurs in a program abnormal termination will happen.

Errors :-

→ Errors are mistakes done by programmer (Syntax mistake)

→ We can not handle errors, we have to debug the errors.

Exception handling keyword →

① try ② catch ③ throw ④ throws ⑤ finally

→ In java all exceptions are classes.

```
Sop (10/0); → // Arithmatic exception
Sop (10/0.0) → // infinity
Sop (10.0%0.0) ; // NAN (not a number)
Sop (10.5 /0) ; // infinity
```

Syntax :-

```
try {
    ≡
}
catch ( ) {
    ≡
}
```

Ex:-

```
pubic class Demo {
    psvm (— ) {
        try {
            SOP (10/0);
        }
        catch (ArithmeticException e) {
            SOP ("Handled");
        }
    }
}
```

O/P
Handled

what is exception handling or why exception handling?

→ Exception handling is a mechanism used to continue the flow of an execution.

→ For one try block we can write "n" no. of catch block.

```
pubic class Demo{
    psvm(--){
    try{
      Sop(10/0); // internally it will create ArithmeticException a = new ArithmeticException();
    catch(ArithmeticException e){
        Sop("Handled");
        Sop(e); // it will print the reference.
        Sop(e.getMessage()); // why the exception //out// /by zero
    }}} Sop e.printStackTrace(); // reference, why all will print.
```

try:-

→ try is a keyword used to handle the keyword.
→ Inside the try block, we have to write risky lines of codes
→ Inside this, we should not declare multiple stmts. if exception will occur the remaining stmts will not execute
→ try block always associated with catch block.
→ without catch we can't write try block.
→ we should not declare multiple try blocks for one catch block.

```
public class Demo{
    psvm(--){
    try{ int[] a = {1,2,3};
    try{
        Sop(a[5]/0);
    catch(ArithmeticException | ArrayIndexOutOfBoundException e){
        Sop(e);
        Sop("Handled");
    }}
```

O/P

ArrayIndexOutOfBoundException
Handled.

```
public class Demo{
    psvm(--){
    String s = null;
    try{
        sop(s.length());
    }catch(NullPointerException e){
        sop(e);
    }}sop("Handled");
```

O/P
NullPointerException
Handled.

—o—

Note —

→ If two objects are equal according to equals() method, then their hash code must be same.

→ If two objects are not equal according to equals() method, their hascode are not required to be different.