Basic program

★ TreeSet

```
import java.util.TreeSet;
public class TreeSetDemo {
    public static void main (String [] args) {

        TreeSet<String> treeset = new TreeSet<>();

        treeSet.add ("C");
        treeset.add("B");
        treeset.add ("A");

        System.out.println (treeset);
    }
}
```

O/P :- [A, B, C]

* Map

1) Hashmap

```
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

    public class HashMapDemo {
        public static void main (String [] args) {

Map<String, Integer> map = new HashMap<>();

map.put ("a", 10);
map.put ("b", 20);
map.put ("c", 30);

system.out.println ("Size of map is :-" + map.
                                    size());

system.out.println (map);

if(map.containskey ("d")) {
   Integer  a = map.get ("d");
   System.out.println ("value for key " + "\" d1'
                              is :- " +a);
    }
}
```

ComparableDemo :- 2 class

```
import java.util.ArrayList;

class ComparableDemo {
  public static void main (String args[]) {

ArrayList<Student> list = new ArrayList<student>();
 Student.John = new student (3, "John", 21);
 student Jane = new Student (1, "Jane", 18);
Student tom = new student (2, "Tom", 20);

   list.add(john);
   list.add(jane);
   list.add(tom);

   Collections.sort(list);

   System.out.println("students after sorting :");
   list.forEach(Student → system.out.println
          (Student.getName()));
   }
}
```

O/P :- students after sorting : Jane
Tom
John

Student :- 1 class

```
* class Student implements Comparable<Student> {

    private int rollNumber;
    private String name;
    private int age;

    public student (int rollNumber, String name,
                                    int age) {
    this.rollNumber = rollNumber;
    this.name = name ;
    this.age = age;
}
    public void setName (String name) {
        this.name = name;
}
    public void
```

class 3 :- Age Comparator

```
import java.util.Comparator;

public class AgeComparator implements Comparator
                                    <Student> {

  @override
    public int compare (Student o1, Student o2) {
if (o1.getAge() < o2.getAge()) return -1;
if (o1.getAge() > o2.getAge()) return 1;
      else return 0;
  }
}
```

→ x ←

Digit Program

```
import java.util.Scanner;
 class odd
{
   p s v main(String [] args) {  int n, sum=0;
   Scanner sc= new Scanner (System.in);
   System.out.println("enter number");
    n=sc.nextInt();
   while (n>0)
    {
      sum= sum + n %10;
      n=n/10;
    }
System.out.println("sum of digit
                  =" +sum);
}
}
```

O/P :- 2+4+5 = 11

n=245 last digit cancelled

sum=0

Dry run

sum = sum+n%10
= 0 + 245 %10
= 0+5
= 5

sum = sum+n%10
= 5+4
= 9

Notes

Input

BufferedReader → class ← Scanner

Scanner → util pkg

package :- Container for class

→ It is collection of classes

io

```
import java.io.*;
```

eg :-
```
class Add
{
    p s v main(String [] args) {

    int a, b, sum;        → object
    BufferedReader br = new BufferedReader
                        ( new InputStreamReader(Syste
System.out.println (" Enter 1st no" );
    a= Integer.parseInt ( br.readLine() );
System.out.println (" Enter 2nd no ");
    b= Integer.paseInt (br.readLine ());
    sum = a+b;
    System.out.println ("Addition = ", + Sum);
    }
}
```

scanner

```
import java.util.Scanner;
  class Add
{
   p s v main(String[] args) {

    int a, b, sum;
   Scanner sc = new Scanner (System.in);
   System.out.println ("Enter First No.:");
     a = sc nextInt ();
   System.out.println ("Enter Second No:;");
     b = sc nextInt ();
    sum = a+b;
    System.out.println ("Addition =" +sum);
   }
}
```

* Control Statement

1) Selection control statement — ① If ② switch
2) Looping statement — ① while ② do-while ③ for

odd / even

```
Import java.util.Scanner;

class check {
  p s v main(String [] args) {

  int a;
  Scanner sc = new Scanner (System.in);
  S.out.println ("Enter no to check");
   a = sc nextInt();
   if (a % 2 == 0)
     S.out.println ("Even");
   else
     S.out println ("odd");
   }
}
```

* Switch Statement

switch statement is used to perform some task according to the choice of the user.

a. case - To check the value has been passed
b. break - To break the case after execution
c. default - statement will be executed when no choice

* control statement :- are used to control the flow of execution of statement in a programming language.

1. selection statement
   a. if
   b. switch.... case
2. looping statement
   a. while
   b. do-while
   c. for

Loop :- loop is execute a single statement or a block statement

Every loop contain three part :
1. initialization - where a start करना है
2. condition - कहां तक
3. Increment / Decrement - बढ़ाना / घटाना

Q. program to print 1-10

```
import java.util.Scanner;
 class pro1 {
  p s v main(String [] args) {
  Scanner sc= new Scanner(System.in);
   int i,n;
   sout (" Enter no. upto which you want to print ?")
    n=sc.nextInt();
    i=1;
   while ( i<=n)
   { sout (i) ;
     i= i+1 ;
} }
```

Q. Reverse print

```
import java.util.Scanner;

class Reverse {
  p s v main(String [] args) {

  Scanner sc= new Scanner (System.in);
  int n, rev=0;
  sout ("Enter No. to find Reverse:");
   n = sc.nextInt();
  while (n>0)
  {
     rev =(rev * 10) + n % 10;
      n= n/10;
   }
  sout("Reverse = " + rev);
  }
}
```

Note :- Dry Run :- How to execute statement

* How to print star shape printing

```
import java.util.Scanner;

 class pyramid
{
   p s v main(String [] args) {
    int n , i;
Scanner sc= new Scanner (System.in);
```

```
sout("Enter No. of rows :");
n = sc.nextInt();
for (i=1; i<=n; i++)
{
  for (j=1; j<=i; j++)
  {
    sout ("* ");
  }
  sout ();
}
}
```

"11" :-

```
*
* *
* * *
* * * *
* * * * *
```

* passing argument to function in class & object

* Static Members

1. Static data members
2. Static member function

- static data member are not the part of the any object on it is the part of the class
- Although static members are not the part of object, instead all the objects can us its value
- static data members are kept seperately and hence its value is unique for all the objects.

eg

```
class Student
{
  int roll;
  String name;
  static String cname;

  Student (int x, String y)
  {
    roll = x;
    name = y;
  }
  void display()
  {
    System.out.println ("Roll"="+ roll + "Name="
                        +name + " College Name="
                        + name );
  }
}
```

parameterized constructor

class name :- Demo.java

```
class Demo
{
   p s v main(string [] args)
{
   Student aa = new Student (101, "Ram");
   Student bb = new Student (102, "shyam");
    aa.display ();
    bb.display ();
   }
}
```

② Static Member function :-

- Static member are not the part of any object It is the part of class
- It can be invoked directly using the class name
- Static member functions can used only static data members

eg :-

```
Class student
{
    int roll;
   String name;
   static string Cname = "xyz";
   Student (int x, String y)
   {
     roll = x;
   name = y;
   }
    void display () {
   {
```

```
System.out.println(" Roll= " + roll + " Nam = "
                    + name + " College Name= "
                      + cname);
  }
  static void change()
  {
    cname = "ABC";
  }
}
class demo
{
  p s v main(String args[])
{
  Student aa = new Student(101, "Ram");
  Student bb = new Student(102, "Shyam");
  aa.display();
  bb.display();

  Student.change();
  aa.display();
  bb.display();
  }
}
```

Constructor

a) constructors are special member function which is used to initialize the data members of a class
(data members → variable)

b) A constructor has the same name as the class name.

c) A constructor has no return type not even "void".

d) Constructors are automatically invoked (call) as soon as object of its class is created

* Types :- 1> Default
2> parameterized

1> Default :- A constructor having no argument

2> Parameterized :- A constructor having parameter / argument

↓

A constructor having arguments in it is called parameterized constructor

eg
```
import java.util.Scanner;

Class add                                   → Constructor overloading
{
    int a, b;
    add (int x, int y)
    {
        a = x;
        b = y;
```

OR

```
add ()
{ a=0; b=;
add (int x, int y)
{ a=x;
b=y;
```

```
void input()
{
  Scanner sc = new Scanner(System.in)
  System.out.println("enter 1st no:");
   a = sc.nextInt();
  System.out.println("enter 2nd no:");
   b = sc.nextInt();
}
  void display()
{
   System.out.println("a=" + a + b=" + b);
}
  void output()
{
   int c;
   c = a+b;
   system.out.println("Addition = " + c);
}
   p s v main(String [] args)
   {
     add aa = new add();
     aa.display();
     aa.input();
     aa.output();
   }
}
```

default

of parameterized

```
add bb = new add(5,10);
bb.display();
bb.output();
}
```

OR

```
import java.util.Scanner;

class add
{
   int a,b;
   add (int x, int y)
   {
     a=x;
     b=y;
   }
   void output()
{
     int c;
     c=a+b;
    System.out.println("Addition =" +c);
   }
   p.s v main(String [] args)
{
     int m, n;
    Scanner sc= new Scanner (System.in);
   System.out.println ("Enter 1st no:");
    m = sc.nextInt ();
   System.out.println ("Enter 2nd no:");
    n= sc.nextInt();
     add aa = new add (m,n);
        aa.output();
   }
}
```

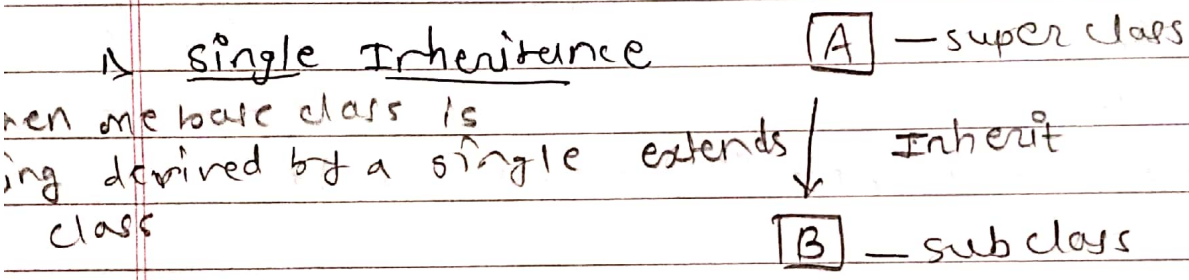
Inheritance

- It is always better to use a pre-existing resource rather than creating all over again.
- Inheritance provides an way using which we can use the pre-existing to resource our current program / application.

Benefit : Reusability

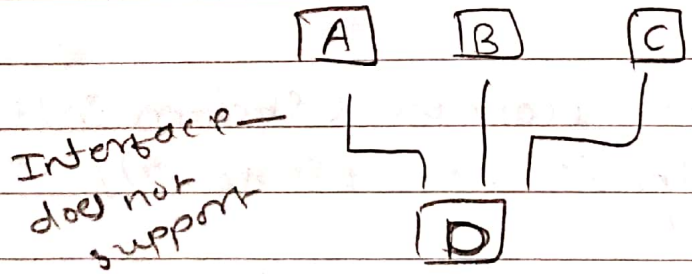
Types of Inheritance :-

1> Single Inheritance

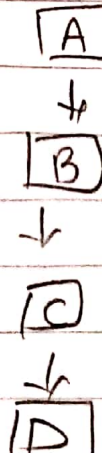


hen one base class is
ing derived by a single
class

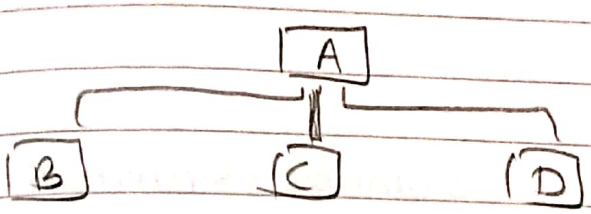
2> Multiple Inheritance



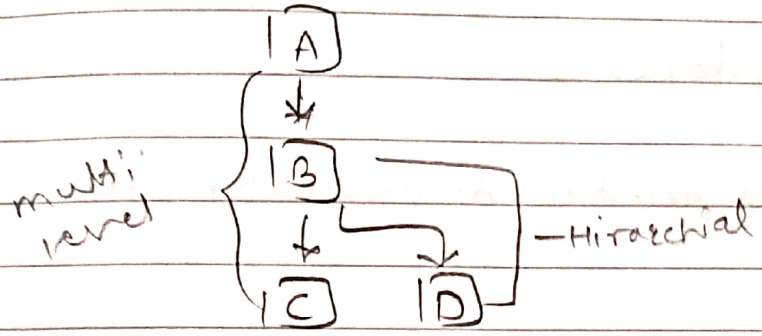
3> Multilevel Inheritance



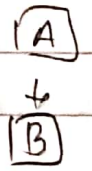
4> Hierarchical Inheritance



5> Hybrid Inheritance



1> Single Inheritance eg :-




```
import java.util.Scanner;
class A
{
    int a, b;
    void input()
    {
    Scanner sc= new Scanner (System.in);
    System.out.println ("Enter First Number");
     a=sc.nextInt();
    System.out.println("Enter Second Number");
     b= sc.nextInt();
    }
     void add()
    { System.out.println("Addition=" + (a+b));
    }
}
```

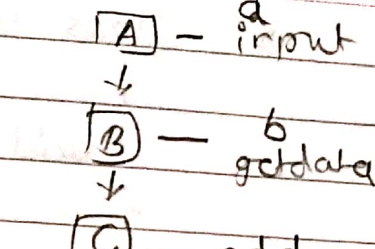
derived class

```
class B extends A
{
  int c;
  void getdata()
  {
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter NO:");
    c=sc.nextInt();
  }
  void display()
  {
    System.out.println("Addition="+(a+b+c));
  }
}
```

class demo.java → program name

```
{
  p s v m(String args[])
{
    B aa=new B();
    aa.input();
    aa.getdata();
  }
}
```

2) Multi-level Inheritance



— when a class is inherited by another class and again this another class is being inherited by another class, then this is called multi-level inheritance

eg import java.util.Scanner;

```
class A
{
    int a;          → function
    void input()
    Scanner sc = new Scanner (System.in);
    System.out.println ("Enter Number");
    a = sc.nextInt();
  }
}
class B extends A
{
  int b;
  void getdata ()
  {
    Scanner sc = new Scanner (System.in);
    System.out.println ("Enter NO:");
    b = sc.nextInt();
  }
}
```

↳ Class C extends B
{
void add()
{
System.out.println("Addition =" +(a+b));
}
}

↳ class Demo.java → class name
{
p.s.v. main(String [] args)
{
C aa = new C();
aa.Input();
aa.getdata();
aa.add();
}
}

Hierarchical Inheritance :-

- Hierarchical inheritance is a type of inheritance in which one class is derived by multiple classes.

↳ eg import java.util.Scanner;

class A
{
int a, b;
void input()
{

```
    Scanner sc= new Scanner (System.in);
  System. out. println ("Enter 1st NO:");
    a = sc. nextInt();
  System out. println ("Enter 2nd NO:");
    b= sc. nextInt();
  }
}
```

↳
```
class B extends A
{
   void add ()
   {
     system.out.println("Addition = " +(a+b));
   }
```

↳
```
  class C extends A
{
   void substract()
  {
    System. out. println ("sub=" + (a-b));
  }
}
```

↳ Class demo.java → class name
```
{
   p s v main(String [] args)
  {
      B aa = new B ();
      c bb = new C ();
      aa. input();
      aa. add ();
      bb. inpute();
      bb. subtract();
  } }
```

Method Overloads overriding

- When there is same name function in base and derived class, then when calling this function, the derived class function will be call and not the base class function. we can say that the derived class function has overriden base class function & this is called method overriding.

eg

```
↳ class A
{
    void message()
    {
    System.out.println("Welcome to Base class");
    }
}

↳  class B extends A
{
    void message()
{
    System.out.println("Welcome to Derived class");
    }
}

↳    class Demo.java → class name
{
     p s v main(String[] args)
     {
        B aa = new B();
       aa.message();
     }
}
```

* Final variable, final method, final class

1> Final variable:-(can't change the value)

- A final variable is a variable whose value can't be changed during the execution of the program

eg ↳ class Demo
{
p s v main(String [] args)
{
int a=5;
final int b=10;
a=10;
System.out.println("a ="+a+" b="+b);
}
}

2> Final method :- A final method prevent the method overriding

eg ↳ class A
{
final void display() ← method
{
System.out.println("Base class");
}
}

↳ class B
{
void display1()
{
System.out.println("Derived class");
}
}

↳ class Demo.java → class name
{
p s v main(String args[])
{
B aa = new B();
aa.display();
} aa.display1();
}

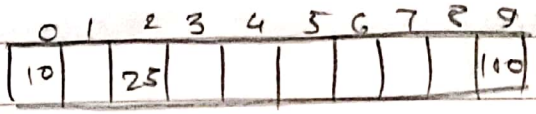
3> Final class :-
A final class cannot be inherited

eg ↳ final class A
{
void display()
}

Array

- Array is a collection of similar types of data. That means if we want to store multiple values of similar type than we want use the concept of array.

@ =



Index = 0 to n-1 (n = size)

↓

every time starts with 0

value = a[0] = 10; , a[2] = 25; a[9] = 100;

Syntax :- int a[] = new int[10];

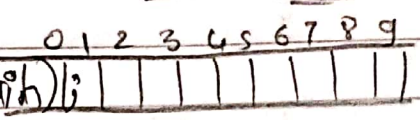
eg
```
import java.util.Scanner;

class Demo
{
 public static void main (String args[])
 {
  int a[] = new int[10];
  Scanner sc = new Scanner(System.in);
  for (int i=0; i<10; i++)
  {
   System.out.println ("enter no: ");
   a[i] = sc.nextInt();
  }
  for (i=0; i<10; i++)
   System.out.println (a[i]);
 }
}
```

Input

output

Dry Run



Q. Write a Java program to accept an array from user and find sum of elements of the array.

```
import java.util.Scanner;

class SumofArray.java --does
{
  p s v main(String[] args)
{
    int size;
System.out.println("Enter of size Array:");
size = sc.nextInt();
int a[] = new int[size];
for (i=0; i<size; i++)
{
 System.out.println ("Enter No:");
  a[i] = sc.nextInt();
}
for(i=0; i<size; i++)
  sum = sum+a[i];
System.out.println ("Sum="+sum);
}
}
```

Input

output

Array
① One Dimensional
② Two Dimensional

0	1	2	3	4
3	4	2	8	1

a

size = 5

5+1 = 18

Dry Run

⇒ sum = sum+a[i]
= 0+3
= 3
⇒ sum = sum+a[i]
= 3+4
= 7

★ Searching :- ⇒ Linear search / sequential
⇒ Binary search

⇒ Linear Search :- (one by one)

0	1	2	3	4	5	6	7	8	9
5	3	2	8	1	9	7	6	15	25

a

Very slow search key = 15

eg

```
import java.util.Scanner;

class Linear
{
    int a[] = new int[10];
    void getdata()
    {
        Scanner sc = new Scanner(System.in);
        int i;
        for (i=0; i<10; i++)
        {
            System.out.println("Enter No:");
            a[i]=sc.nextInt();
        }
// }
//  void search()
//  {
//    int key
    System.out.println("Enter No to search:");
    key =sc.nextInt();
}

    void search()
    {
        int i, flag=0; pos;
        for(i=0; i<10 && flag==0; i++)
        {
            if (a[i]==key)
            {
                flag=1;
                pos= i+1;
            }
        }
```

Input

i = 0 1 2 3 4

flag = 0 1

pos = 4

```
    if (flag == 1)
    System.out.println ("No. found at " + pos);
  else
     System.out.println("No. Not found at");
  }
}
```

↳ Class Demo
```
{
   p s v main (String [] args)
  {
   linear aa = new linear ();   (object)
   aa. getdata();
   aa. search();
  }
}
```

2) Binary Search (very fast)

- pre-condition :- All the elements should be in sorted order

Syntax

eg
```
class Binary
{
   int i=0;
       j=9;
     flag=0;
  while (i<=j && flag==0)
{
   mid = (i+j) /2;
   if (a[mid] == key)
  {
```

```
flag = 1 ; pos = mid + 1;
}
  if (a[mid] > key)
{   j = mid - 1;
  }
if (a[mid] < key
  i = mid + 1;
}
if ( flag == 1 )
  sop(" Number found" + pos);
else
    sop (" Number Not found "+pos);
```

Program

eg

```
import java.util.Scanner;

class Binary_search
{
  int a[] = new int[10];
  int key;
   void getdata()
{
  Scanner sc = new Scanner (System.in);
  int i;
 for(int i=0 ; i<10; i++)
 {
  System.out.println ("Enter No:");
   a[i] = sc.nextInt();
 }
  System.out.println("Enter No. to search");
  key = sc.nextInt();
 }
```

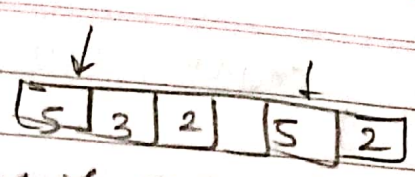
```
void search ()
{
    int i, j, mid, flag, pos
    i=0;
    j=9;
    flag = 0;
   while (i<=j && flag==0)
   {
     mid = (i+j) / 2;
    if (a[mid] == key)
    {
      flag = 1, pos = mid+s;
    }
    if (a[mid] > key)
      j = mid-1
   if (a[mid] < key)
      i = mid+1
   }
   if (flag==0)
   System.out.println("Not found");
else
    System.out.println("No found at" + pos);
 }
}
```

↳

```
class Demo
{ p s v main(String [] args)
{
   binary_search aa = new binary_search ();
      aa.getdata();
      aa.search();
   }
}
```

* Frequency Number (same no. count in array)

Same number count in given array



freq of 5 = 2
" 2 = 2
" 3 = 1
" 10 = 0

```
import java.util.Scanner;

class Frequency
{
  int a[] = new int[10];
  int key;
return type ---- void getdata()   ---- function
  {
    Scanner sc = new Scanner(System.in);
    int i;
    for(i=0; i<10; i++)
  {
    System.out.println("Enter NO:");
    a[i] = sc.nextInt();
  }
    System.out.println("Enter No. to find Freq.":");
      key = sc.nextInt();
  }
    void count()
{
  int i, count=0;
  for(i=0; i<10; i++)
{
  if(a[i]==key)
```

	0	1	2	3	4	5	6	7	8	9
a	5	3	2	8	2	7	8	2	1	3

key =

```
        count = count + 1 ;
    }
    System.out.println("frequency=" +count);
    }
}
```

↳
```
class Demo
{
    p s v main(String [] args)
    {
        frequency aa = new frequency ();
        aa.getdata();
        aa.count();
    }
}
```

✱ Selection Sort :-

execution part - 3 2 9 7 6 8 5 4

i j

2 3 9 7 6 8 5 4

since 2 < 3 - condition

```
import java.util.Scanner;

class Selection
{
    int a[]=new int [10];
    void getdata ()
    {
        Scanner sc=new Scanner(System.in);
```

```
    int i;
for (i=0; i<10; i++)
{
  System.out.println("Enter No:");
   a[i] = sc.nextInt();
 }
}
 void sort()
{
  int i, j, t
  for (i=0; i<9; i++)
  for (j=i+1; j<10; i++)
  {
    if (a[i] > a[j])
    {
      t = a[i];
      a[i] = a[j];
      a[j] = t;
    }
  }
}
  void putdata()
  {
    int i;
   for(i=0; i<10; i++)
   System.out.println(a[i]);
  }

↳ class Demo
{
  p s v main(String[] args)
{
```

```
{
  Selection aa = new Selection();
    aa.getdata();
    sop("Unsorted Array");
    aa.putdata;
  sop("sorted Array is:");
    aa.sort();
    aa.putdata();
  }
}
```

* **Bubble Sort :-**

eg

```
import java.util.Scanner;

  class Bubble
{
  int a[] = new int[10];
    void getdata()
{
  Scanner sc = new Scanner(System.in);
  for(i=0; i<10; i++)
  {
    System.out.println("Enter NO:");
    a[i] = sc.nextInt();
  }
}
  void putdata()
{
for(i=0; i<10; i++)
  System.out.println(a[i]);
}
```

```
void sort()
{
  int i, j, t
 for(i=0; i<10; i++)
for(j=0; j<9-i; j++)
{
   if(a[j] > a[j+1])
   {
     t=a[j];
     a[j]=a[j+1];
     a[j+1]=t;
   }
}
}
```

↳

```
class Demo
{
   p s v main(string[] args)
   {
     Bubble aa = new Bubble();
     aa.getdata();
     SOP("unsorted Array is :");
     aa.putdata();
     SOP("sorted Array is :");
     aa.sort();
     aa.putdata();
   }
}
```

* Insertion Sort :-

eg

```
import java.util.Scanner;

class Insertion_sort
{
  int a[] = new int[10];
  void getdata()
  {
    int i;
  Scanner sc= new Scanner (System.in);
  for (i=0;  i<10;  i++)
{
  System.out.println(" Enter NO :");
   a[i] = sc.nextInt();
  }
}
  void putdata()
{
  int i;
  for (i=0;  i<10;  i++)
    System.out.println(a[i]);
}
  void Sort()
{
  int i , val. j
for ( i=1 ;  i<10;  i++)
  {
    val = a[i];
    j = i-1;
while ( val<a[j])
```

```
{
    a[j+1] = a[j];
    j--;
    if(j==-1)
    break;
}
 a[j+1] = val;
 }
 }
}
```

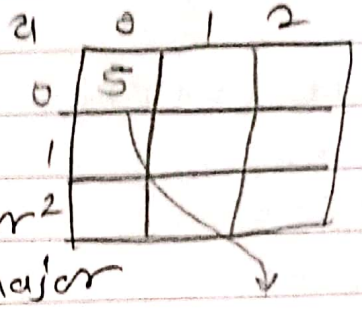
↳ class Demo

```
{
    p s v main(String [] args)
{
  insertion_sort aa = new insertion_sort();
   aa.getdata();
  sop("unsorted Array is :");
   aa.putdata();
 aa.sort();
  sop("sorted Array :");
   aa.putdata();
  }
}
```

2D Array

Matrix - starts with 0

Addressing — 1) Row major
2) column major



a[0][0] = 5

row column

eg

```
import java.util.Scanner;

class Twodim
{
  int a[][]=new int[][];
  void getdata()
  {
    Scanner sc=new Scanner(System.in);
     int i,j
  for(i=0; i<3; i++)
  for(j=0; j<3; j++)
  {
    System.out.println("Enter No:");
     a[i][j] = sc.nextInt();
  }
}
  void putdata()
{
  int i, j;
for (i=0; j<3; i++)
{
  for (j=0; j<3; j++)
{
  System.out.println(a[i][j] + "  ");
}
```

Input

output

```
System.out.println();
  }
 }
}
```

↳
```
class demo
{
  p s v main(String[] args)
{
  twodim aa = new Twodim();
   aa.getdata();
   aa.putdata();
  }
}
```

* Addition Subtraction of Matrix

eg

```
import java.util.Scanner;

class Addition
{
  int a[][]=new int[3][3];
  int b[][]=new int[3][3];
  int c[][]=new int[3][3];
   void getdata()
  {
    int i, j;
    i=j=0;
   for(i=0; i<3; i++)
   for(j=0; j<3; j++)
  {
    System.out.println("Enter NO:");
```

```
    a[i][j] = sc.nextInt();
  }
   for(i=0; i<3; i++)
  for(j=0; j<3; j++)
    {
  {
     System.out.println("Enter NO:");
     b[i][j]= sc.nextInt();
    }
  }
   void add()
  {
     int i, j;
     i=j=0;
   for (i=0; i<3; i++)
   for (j=0; j<3; j++)
     c[i][j] = a[i][j] + b[i][j];
  }
   void putdata()
  {
      int i, j;
       i=j=0;
   for (i=0; i<3; i++)
   {
     for (j=0; j<3; j++)
     System.out.print(c[i][j]+"  ");
     System.out.println();
   }
  }
}
```

```
class demo
{
  p s v main(String [] args)
{
  addition aa = new addition ();
    aa.getdata();
    aa.add();
    aa.putdata();
  }
}
```

* String

String is a class

String is a sequence of character

	0	1	2
s	R	a	m

```
import java.util.Scanner;

class string_input
{
  public static void main(String [] args)
{
  String str = new String();   // create string object first
  Scanner sc = new Scanner(System.in);
  System.out.println("Enter Name:");
   str = sc.nextInt();
  System.out.println("Hello Mr./Ms/Mrs" + str);
 }
}
```

* String function () :- ① length()
② charAt()

⇒ length() :- This function returns the length of the given string

eg :- String s = "shyam";
System.out.println("length=" + s.length());

o/p :- length = 5

⇒ charAt() :- This function returns the character at the given index

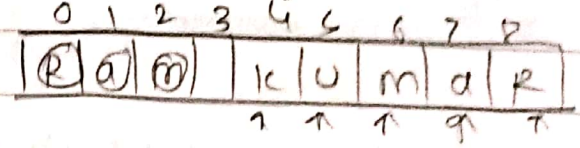
eg :- String s = "shyam";
System.out.println("character 0 index=" + s.charAt(0));

eg import java.util.Scanner; → length()

```
class Demo
{
    p s v main(String[] args)
{
    String s = new String();
  Scanner sc = new Scanner(System.in);
  System.out.println(" Enter String : ");
  s = sc.nextInt();
System.out.println("The string is : " +s);
SOP("The length of the String is : " +len);
for(i=0; i<length; i++)
 SOP("s.charAt(i));
}
}
```

* Count Total vowel (a, e, i, o, u) & consonant (other string) in a String

consonant



vol = a x x (3)
con = b x x b b (5)

eg

```
import java.util.Scanner;

class Demo
{
  public static void main(String args[])
  {
    String s = new String(); int i, vol=0, con=0;
    Scanner sc=new Scanner(System.in);
    sop("enter String:");
    S= sc.nextLine();
    s = s.toUpperCase();
  for(i=0; i<s.length(); i++)
  {
    char c=s.charAt(i);
  if (c!=' ')
{
if(c=='A'||c=='E'||c=='I'||c=='O'||c=='U')
  vol++;
else
  con++;
}
}
  System.out.println("vowels=" + vol + " con=" + con);
}
}
```

Write a java program to check whether a string is palindrome or not ? eg

S= | M | A | D | A | M |

eg

```
import java.util.Scanner;

class palin
{
  p s v main(String [] args)
{
  String s = new String();
  Scanner sc = new scanner(system.in);
  System.out.println("enter string to check :");
  s = sc.nextLine();
  int i, j, flag;
  i=0;
  j= s.length()-1;
  flag =0;
  while( i<j && flg==0)
{
  if (s.charAt(i)! = s.charAt(j));
    flag=1;
  i = i+1;
  j = j-1;
  }
 if (flag ==0)
   system.out.println ("palin");
else
  system.out.println("Not palin");
 }
}
```

* Equality of two strings

① equals
check for equality of two strings but [case sensitive]

Str 1 = "Ram" → [Equal]
Str 2 = "~~shyam~~ RAM"
Not equal

② Equals Ignore Case
checks for equality of two string but the checking is [non-case sensitive]

eg

```
import java.util.Scanner;

class Demo
{
    p s v main(String [] args)
    {
      String str1 = new String();
      String str2 = new String();
    Scanner sc = new Scanner(System.in);
    SOP("enter 1st String:");
     str1 = sc.nextLine();
     sop("enter 2nd string:");
      str2 = sc.nextLine();
    if ( str1.equals(str2)).
     sop("Equal String");
   else
      SOP("Not Equal string");
   }
}
```

True / False

★ CompareTo () :- compare two strings

when

0 → s1 = s2

pos + → s1 > s2

neg - → s1 < s2

eg

```
import java.util.scanner;

class demo
{
  p s v main(string [] args)
{
  String s1 = new String ();
  String s2 = new String ();
 Scanner sc = new scanner (system.in);
 System.out.println ("Enter 1st string:");
  s1 = sc.nextline ();
  system.out.println ("enter 2nd string:");
  s2 = sc.nextLine();
 if (s1.(compareTo (s2) == 0)
  sop (" equal string");
 else if (s1.compareTo(s2)>0)
   sop(" Not equal"); → (" s1 is greater than s2");
}}
}
  else: sop (" s2 is lesser than s2");
}
}
```

* LowerCase & UpperCase

1) toLowerCase :

Converts the string into lower case

2) toUpperCase :

converts the String into upper case

```
String str = new string ();
str = "ram"
```

RAM - Upper
ram - Lower

Q- Write a program to convert a given String into upper case / Lower case

```
import java.util.scanner;

class Demo
{ p s v main(string [] args) {
String str = new String ();
Scanner sc = new Scanner (system.in);
sop("Enter string in lower case:");
str = sc.nextLine();
sop(" String in uppercase = " + str. toUpperCase()
sop("enter String in Upper case:");
str1 = sc.nextline();
sop (" string in lowercase = " + str1 to LowerCase());
}
}
```

Reverse string

```
import java.util.Scanner;

class Reverse
{
   p s v main(String []args)
{
 String str = new String();
Scanner sc = new Scanner(System.in);
Sop("Enter String :");
str = sc.nextLine();
 int i,j,k;
 i= str.length()-1;
 while(i>=0)
{  j=i;
while(str.charAt(j)!=' && j>0)
          j--;
 if(j==0)
     k=0;
 else
     k=j+1;
  Sop(str.substring(k, i+1);
    i=j;
 }
}
}
```

StringBuffer : (mutable - changable)

1. setCharAt (n, 'character') n = index

This function replace nth index character with given character.

```
StringBuffer s = new StringBuffer ("codeitup");
s.setCharAt (3, 'U');
```

2. Append () = appends the second string in first string

Str1.append (Str2)

```
StringBuffer str1 = new StringBuffer ("codeitup");
StringBuffer str2 = new StringBuffer ("java");
str1.append (str2);
sop("str");
```

3. insert :

str1.insert (n, str2)

```
StringBuffer str1 = new StringBuffer ("codeitup");
StringBuffer str2 = new StringBuffer ("java");
str1.insert (4, str2);
```

eg
```
import java.util.Scanner;

Class SBExample
{
  p s v main(String [] args)
{
 StringBuffer str = new StringBuffer();
 Scanner sc = new scanner (System.in);
 sop(" Enter string :");
 str.append (sc.nextLine());
 Str. setcharAt (3, 'u');
  sop(Str);
  str. insert (3, "ABCD");
   sop(Str);
   str. setlength (20);
   sop(Str);
  }
}
```

★ Vector - (Class)

vector is a class in java present in java.util pkg. It is used to store objects of different types & size in any number.

* import java.util.*;

```
class VectorExample
{
  public static void main(String args[])
  {
    Vector list = new Vector();
    list.addElement("Ram");
    list.addElement("Shyam");
    list.addElement("seeta");
    SOP("vector elements are :"+list);
  // }
// }
    list.AddElementAt("Ravi", 1)
    SOP("vector elements are:" + list);
  }
}
```

Interface

An interface is basically a kind of class. Like class, interface also contains methods and functions but with a difference that it can only contain abstract methods and final fields.

1. Abstract methods: This is a method that is declared without an implementation (without braces & implementation)
2. Final fields: A final field can not change its value.

Interface is used to implement the concept of multiple inheritance in java.

Syntax :- interface Interface_name
{
variable declaration;
method declaration;
}

Extending Interface

In the same way we extend a class, we can also extend an interface. It means than an interface can extend another ex interface.

eg
```
interface abc
{
  int code = 101;
  string str = "Anup";
}
interface xyz extends abc
{
  void display();
}
```

Package

A package is basically collections of different classes. In other words we can say that package is a way to group variety of classes and/or interfaces together. The grouping is usually done on the basis of functionality.

Benefits :

1. The classes contained in a package can be used in any program that import that classes.
2. packages provide a way to "hide" classes.
3. Package also provides a way for separating "design" from "coding".

* Types : 1) API packages - Built-in Pckg
2) User Defined Packages

Java API Package :- Java API (Application Programming interface) packages provide way to group diff classes into diff packages.

Includes : lang, util, io, awt, net, applet

Package Name	what it contains?
Java.lang	language support classes. string, thread etc.
Java.util	language utility classes such as vector, random, number, scanner etc.

java.io	input/output support classes
java.awt	set of classes to implement GUI
java.net	classes for networking
java.applet	classes for creating and implementing applets.

Using System package :

There are two ways to access the classes stored in the package. The first approach is use to fully qualified class name of the class that we want to use.

for example, if we want to refer to the class color in the awt package, then we may do so as follows:

java.awt.Color

if we want to refer the scanner class in the util package, then we may do so as follows : java.util.Scanner

* Naming Conventions :-

packages can be names using the standard java naming rules. This makes it easy for users to distinguish package name for class name. pkg class method

double y = java.lang.Math.sqrt(x); args

Here, lang is the package, Math is the class and sqrt is the function .i.e. the method name.

* Creating User Defined Package

For creating user defined package, we need to first define the structure of the package.

Let's go for the following structure:

1. First decide the name of the package and then create a sub-directory with the same name inside bin directory of java.

2. Now, here you have to create a program in which you have to type the code viz.

eg
```
package anand;
public class myclass
{
public void myfunction()
{
System.out.println("Yahoo I created my package...");
}
}
```

eg
```
import anand.*;
class demonstration
{
p s v main(String args[])
{
myclass aa = new myclass();
aa.myfunction();
}
}
```

Multi-threading

Modern operating system has the capability to execute several programs simultaneously. ie. at the same time - This ability is known as multitasking. In system's terminology, it is called "multi-threading".

multithreading is a conceptual programming paradigm where a program is divided into two more subprograms which can be implemented at the same time in parallel.

Thread :- A thread is similar to program that has a single flow of control. It has a beginnig, a body and an end. All the program which we have done till now are called as single threaded program as they have single flow of execution

Java has a unique property that it supports multithreading that is, java enables us to use multiple flows of control in developing program and each flow of control is considered as a seperate.

A program that contains multiple flow of control is knows as multithreaded program.

Multithreading	Multi-tasking
1) It is a programming concept	1) it is an operating system concept
2) It supports execution of multiple parts of the same programs simultaneously	2) It supports execution of multiple programs simultaneously
3) The processor switches betn diff parts / threads of the program	3) The processor switches betn multiple programs
4) It is highly efficient	4) It is less efficient as compared to multithreading
5) It helps in developing efficient programs	5) It helps developing efficient os.

* Creating Threads

This run() is invoked by calling the start() method. now remember this that if you want this run() method to behave like threaded program you need to call the start() method. if you will call it by its name viz. run(), it is not going to show any multithreading property.

- create thread by two ways :

1) By creating a thread class :
2) By Implement the Runnable Interface :

1) Extending Thread class :

In order to create a threaded program we have to extend the java.lang.Thread class. This given the access to all the thread methods directly. steps are :

1. Declare the class as extending the thread class.
2. Implement the run() method
3. Create a thread object and call the start() methods.

Class :

2. By converting a class to thread : for this you need to implement the "Runnable" interface.

eg

```
class ThreadTest
{
  p s v main(String [] args)
  {
   new A().start();
   new B().start();
   new C().start();
  }
}
Class A extends Thread
{
```

```
public void run()
{
  for(int i =1; i<=5; i++)
{
   sop("i=" +i);
}
}
}
class B extends Thread
{
   public void run()
{
   for (int j=1; j<=5; j++)
{
   sop("j =" +j);
}
}
}
class C extends Thread
{
   public void run()
{
   for (int k=1; k<=10; k++)
{
   sop(" k=" +k);
}
}
}
```

* Life cycle of a Thread

Thread priority

In java every thread is assigned a priority and based on this priority the threads are executed. Usually all the threads have equal priority and hence they share the processor on a first come first serve basis.

Java provides us the following methods related to Thread priority:

a. SetPriority()
b. getpriority()

eg
```
class ThreadPriority
{
  p s v main(String [] args)
  {
    A aa = new A();
    B bb = new B();
    C cc = new C();
   cc.setPriority(Thread.MAX_PRIORITY);
   bb.setPriority(aa.getPriority()+1);
   aa.setPriority(Thread.MIN_PRIORITY);
   SOP("Thread A started...");
    aa.start();
   SOP("Thread B started...");
```

```
        sop("C Thread started ---");
        cc.start();
        sop("Main Thread Ended...");
    }
}
class A extends Thread
{
    public void run()
    {
        sop("Thread A started");
        for(int i=1; i<=5; i++)
        {
            sop("From Thread A : i ="+i);
        }
        sop("Thread A Ends Here...");
    }
}
class B extends Thread
{
    public void run()
    {
        sop("Thread B started");
        for(int j=1; j<=5; j++)
        {
            sop("From Thread B : J ="+j);
        }
        sop("Thread B Ends Here...");
    }
}
```

↳ class C extends Thread

```
{
    public void run()
{
    sop("Thread Started");
    for(int k=1; k<=5; k++)
{
    sop("From Thread C : k=" +k);
}
    sop("Thread C Ends Here....");
}
}
```