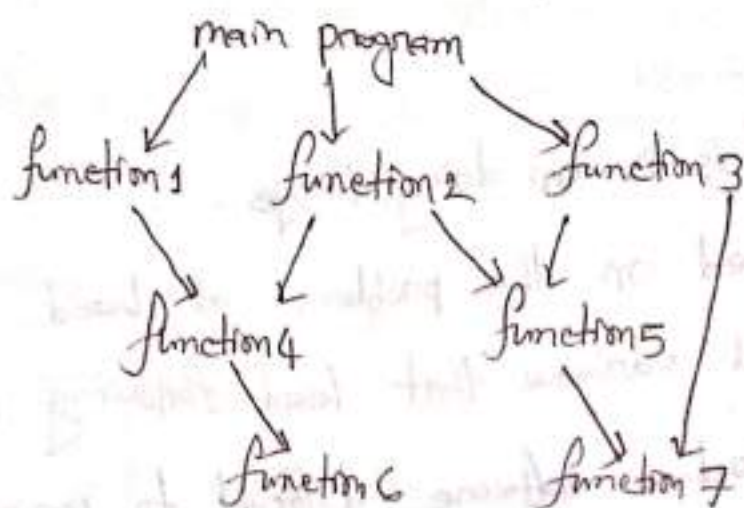


Tree

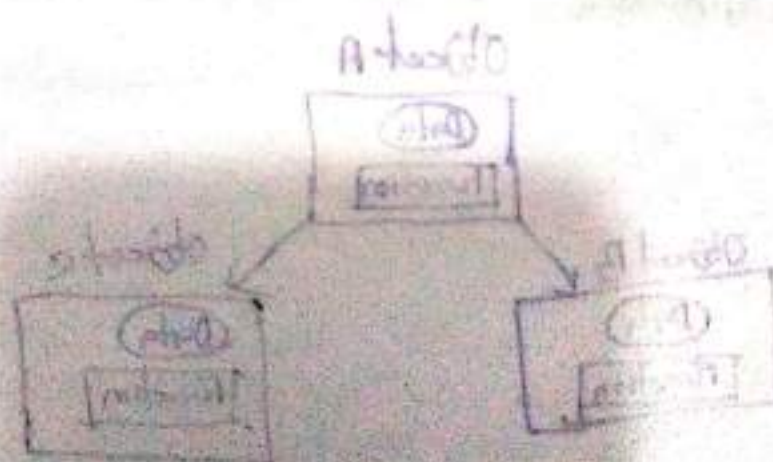
main program

Pop : —

- A program in procedural is a set of instruction.
- The program are divided into sub program called function



what do we have to do to solve this problem

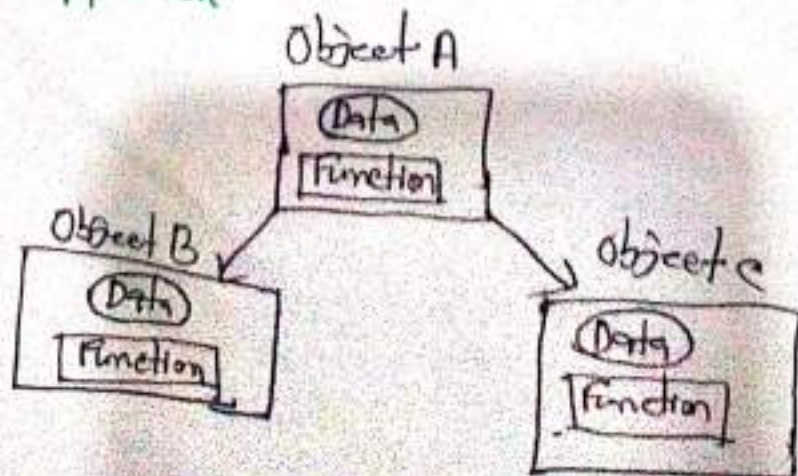


- \* uncontrolled modification
- \* problem of enhancement & maintainability
- \* code Reusability.

### \* Major characteristic:—

- more emphasis is on doing things.
- It is based on the problem at hand.
- used global variable that lead following problem.
- It will produce software difficult to maintain
- The risk of unwanted access & modification increases.
- It will not result in reusable software.
- It employs top down approach in program design
- Works well in small system.

### \* OOP → OOP approach





रमाशारी

Java

Oop: - Object oriented programming is a programming paradigm in which program are organization around data on object rather than function & logic.

- Oop returns to type of computer programming in which programmer define not only the data structure but also the types of operation that can be applied to the data structure.
- The data structure becomes an object that include both data & function.
- programmers can create relationship between object
- object can inherit the characteristics from other object.

## Oop advantages

- code reusability
- data redundancy
- code maintenance
- security
- Design benefit
- polymorphism flexibility.

## \* Basic feature of oop:-

- class
- object
- Inheritance
- Encapsulation
- polymorphism
- Abstraction



## \* Object:-

- Object is an instance of class
- The object is an entity which has state & behaviour.
- Object has state & behaviour.

State:- represent data (value) of object  
i.e; balance.

behaviour:- Represent functionality of an object.  
i.e; deposit - withdraw.

Example:-

Pen, state - color, brand, size

behaviour - writing

Software development:-

state : data

behaviour : method function.

## Class: →

6

- A class is a blueprint/prototype from which objects are created.
- A class is a blueprint, idea & prototype for creating object.
- A class is a group of objects which have common properties.

Example: → sketch (prototype) of house

- It contains all details about floor, door, windows etc.
- Based on this description, we can create house.
- Here house is the object.

```
class student {  
    String name;  
    int ID;  
    int age;  
}
```

Object: →

```
classname objectname = new classname()  
student object1 = new student()  
object1.name = "Rohit";  
object1.ID = 50;  
object1.age = 20;
```



## Encapsulation :-

- The wrapping up of a data & function into a single unit is called encapsulation.
- The data is not accessible outside the class.

**Abstraction :-** Defines the representation of essential features without including background details or explanation.

**Inheritance :-** Refers to a process in which object of one class acquires the property of another class.

**Polymorphism :-** It means the ability to take more than one form.

programming with java Balagurusamy (common)

Java How to program Deitel & Deitel

The complete reference Java - H. Schilt

## Java!—

Basic feature of Java/properties.

- Object oriented
- compiled & interpreted.
- platform Independent
- Distributed.
- High performance
- simple & familiar.
- multithreading (একটি আধিক-প্রকল্প একই সময়ে  
করতে পারে)

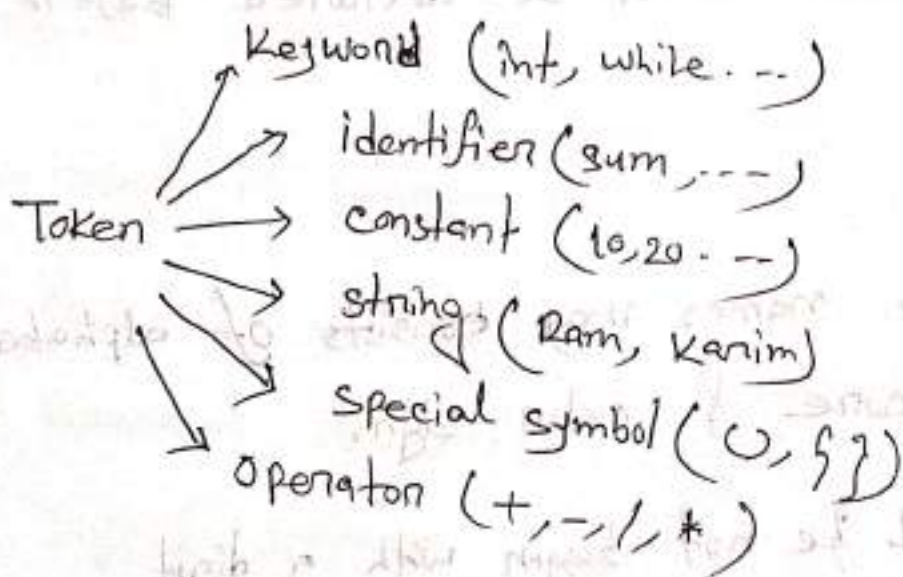
## Application:→

- mobile Application
- desktop " (Swing, Awt, JavaFx)
- web based " (JEE; servlet, JSP...)
- scientific "
- gaming "
- cloud based Application
- Business Application (Banking software)



- Variable
- Token
- Data type
- Identifier
- Operator
- control statement (if, else, switch)
- loop statement (for, while, do-while)

**Token:** - Token is basically smallest element that can be identified by the compiler.



**Identifier:** - Identifier is a sequence of character.

- Identifier are programmer designed token.
- Identifier used for naming class, method, variable, object, package.

**Keyword:-** Reserved word / identifier.

int, float, double, if, switch...

Keywords are the words and phrases that people type into search engines to find what they're looking for.

**Variable:-** A variable is an identifier that denotes a storage location used to store data value.

→ Their value changes during the execution of a program.

→ every variable must be declared before used.

**Rules:-** Variable names may consist of alphabet, digit, underscore & colon sign.

→ They must not begin with a digit.

→ uppercase & lowercase are distinct.

→ It should not be a keyword.

→ No space is allowed.

→ Variable name can be any length.



**Data types:** - Data types specifies the size & type of values that can store in variable.

→ primitive data type (integer, character, Boolean, floating)

→ Non primitive data type (class, interface, array)

**Operator:** - An operator is a symbol that tells the computer to perform certain mathematical or logical operation.

→ used in program to manipulate data & variable.

→ Arithmetic operator (+, -, /, \*)

→ Relational operator (==, !=, >, <, >=, <=, ---)

→ logical operator (&&, ||, !)

→ Assignment operator (=, +=, -=, \*=, ---)

→ Increment / decrement operator (++ , --)

→ conditional operator (? :)

→ Bitwise operator ( $\&$ ,  $|$ )

→ special operator ( $\cdot$ )

### Control statement

The branching structure which controls the program flow is called control structure.

### The if-else statement

- \* The if-else statement contains a condition.
- \* It is logically evaluated.
- \* If it evaluates to ~~TRUE~~ condition, then statements under if part is executed.
- \* If the condition evaluates to FALSE state the statements under else part is executed.
- \* Else clause is optional.
- \* The statements under if or else part can be single or multiple statements.
- \* If multiple statements are used, they must be enclosed with the braces.



If - else

pass - fail

\* release  
4

① import java.util.Scanner;

```
public class passorfail {
```

```
    String check(int n)
```

```
    {  
        if (n >= 3)
```

```
            return "pass";
```

```
        else
```

```
            return "fail";
```

```
    }
```

```
    public static void main (String args[]) {  
        int n;
```

```
        Scanner z = new Scanner (System.in);  
        n = z.nextInt();
```

```
        passorfail ob = new passorfail ();
```

```
        System.out.println (ob.check(n));
```

```
    }
```

## leap-year

② Import java.util.Scanner;

int check (int y)

public class leapyear {

int check (int y) {

if ( $y \% 4 == 0$  &  $y \% 100 != 0$ )

return "Leap Year";

else if ( $y \% 400 == 0$ )

return "Leap Year";

else

return "Not Leap Year";

public static void main (String args[]) {

int y;

Scanner z = new Scanner (System.in);

y = z.nextInt();

leapYear ob = new leapYear();

System.out.println ("The year is " + ob.check (y));

}



## Positive - negative

```
import java.util.Scanner;
```

```
public class PositiveNegative {
```

```
    void check (int n)
```

```
    {  
        if (n >= 0)
```

```
        {  
            System.out.println("positive");
```

```
        }  
        else
```

```
            System.out.println("negative");
```

```
    }
```

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

```
    int n;
```

```
    Scanner z = new Scanner (System.in);
```

```
    n = z.nextInt();
```

```
    PositiveNegative ob = new PositiveNegative();
```

```
    ob.check(n);
```

```
}
```

```
import java.util.Scanner;
```

```
public class ifelse2 {
```

```
    void check (char ch)
```

```
    {
```

```
        if ((ch >= 'a' && ch <= 'z') || (ch <= 'A' && ch <= 'Z'))
```

```
            System.out.println (" Not Alphabet ");
```

```
        else
```

```
            System.out.println (" Not Alphabet ");
```

```
    }
```

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

```
    char ch;
```

```
    Scanner z = new Scanner (System.in);
```

```
    ch = z.next().charAt (0);
```

```
    ifelse2 if2 = new ifelse2 ();
```

```
    if2.check (ch);
```

```
}
```



## minimum number

```
import java.util. Scanner;
```

```
public class minimum {
```

```
int check ( int a, int b, int c)
```

```
{ if ( a < b && a < c)
```

```
    return a;
```

```
else if ( b < a && b < c)
```

```
    return b;
```

```
else
```

```
    return c;
```

```
}
```

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

```
{ int a, b, c;
```

```
Scanner z = new Scanner (System.in);
```

```
a = z.nextInt();
```

```
b = z.nextInt();
```

```
c = z.nextInt();
```

```
minimum ob = new minimum ();
```

```
System.out.println ("ob.check");
```

## for-loop

```
① public class pattern {  
    void print ()  
    {  
        for (int i=1; i<=5; i++)  
        {  
            for (int j=1; j<=i; j++)  
            {  
                System.out.print ("x ");  
            }  
            System.out.print ("\n");  
        }  
    }  
}
```

```
public static void main (String[] args) {  
    pattern ob = new pattern();  
    ob.print();  
}
```



```

2) public class pattern2 {
    void print()
    {
        for (int i=1; i<=5; i++)
        {
            for (int j=1; j<=i; j++)
            {
                System.out.print(i);
            }
            System.out.print("\n");
        }
    }
}

```

```

public static void main (String[] args) {
    pattern2 ob = new pattern2();
    ob.print();
}
}

```

② Import java.util.Scanner;

public class evenNumbersSum {

int sum (int n)

{

int s = 0;

for (int i = 1; i <= n; i++)

{

if (i % 2 == 0)

{

s = s + i;

}

} return s;

}

public static void main (String[] args) {

int n;

Scanner z = new Scanner (System.in);

n = z.nextInt();

evenNumbersSum ob = new evenNumbersSum();

System.out.println ("Sum = " + ob.sum (n));

}



## while loop

① import java.util.Scanner;

```
public class ReverseOfNumber {
```

```
    int reverse(int n)
```

```
    {
```

```
        int n, sum = 0;
```

```
        while (n > 0)
```

```
        {
```

```
            n = n / 10;
```

```
            sum = sum * 10 + n;
```

```
            n = n / 10;
```

```
        }
```

```
        return sum;
```

```
    }
```

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

```
        int n;
```

```
        Scanner z = new Scanner (System.in);
```

```
        n = z.nextInt();
```

```
        ReverseOfNumber ob = new ReverseOfNumber();
```

```
        System.out.println ("Reverse of the number is " + ob.reverse(n));
```

```
    }
```

2) import java.util.Scanner;

public class palindrome {

    int reverse(int n)

    {

        int r, sum = 0;

        while (n > 0)

        {

            r = n % 10;

            sum = sum \* 10 + r;

            n = n / 10;

        }

        return sum;

    }

public static void main (String[] args) {

    int n;

    Scanner z = new Scanner (System.in);

    n = z.nextInt();

    palindrome ob = new palindrome();

    if (n == ob.reverse(n))

        System.out.println ("palindrome");

    else

        System.out.println ("Not palindrome");

}



③ Import java.util.\*;

```
public class doeven {
```

```
    private static Scanner sc;
```

```
    int evenNum (int x, int y) {
```

```
        int i = x; sum = 0;
```

```
        do {
```

```
            if (i % 2 == 0) {
```

```
                sum += i;
```

```
            }  
            i++;
```

```
        } while (i <= y);
```

```
        return sum;
```

```
    }
```

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

```
        int a, b;
```

```
        Scanner sc = new Scanner (System.in);
```

```
        a = sc.nextInt (a);
```

```
        b = sc.nextInt (b);
```

```
        doeven obj = new doeven ();
```

```
        System.out.println (obj.evenNum (a, b));
```

```
    }
```

## do-while

① Import java.util.\*;

public class dosum {

private static Scanner sc;

int sum (int n) {

int i = 1, sum = 0;

do {

sum += i;

i++;

} while (i <= n);

return sum;

public static void main (String[] args) {

int a;

sc = new Scanner(System.in);

a = sc.nextInt();

dosum obj = new dosum();

System.out.println (obj.sum(a));

}

}



② Import java.util.\*;

```
public class doodd {  
    private static Scanner sc;
```

```
    void odd (int a) {
```

```
        int i=1;
```

```
        do {
```

```
            if- (i%2!=0) {
```

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

```
            }
```

```
            i++;
```

```
        } while (i<=a);
```

```
    }
```

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

```
        int b;
```

```
        sc = new Scanner (System.in);
```

```
        b = sc.nextInt();
```

```
        doodd obj = new doodd ();
```

```
        obj.odd (b);
```

```
    }
```

```
}
```

③ Import java.util.\*;

public class doSum {

public static Scanner sc;

int sum (int n) {

int i=1, sum=0;

do {

sum += i;

i++;

}

while (i <= n);

return sum;

}

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

int a;

sc = new Scanner (System.in);

a = sc.nextInt();

doSum obj = new doSum();

System.out.println (obj.sum(a));

}

}



## Switch

```
1) import java.util.*;  
public class vowel {  
    private static Scanner sc;  
    void letter (char x) {  
        switch (x) {  
            case 'a':  
            case 'e':  
            case 'i':  
            case 'o':  
            case 'u':  
            case 'A':  
            case 'E':  
            case 'I':  
            case 'O':  
            case 'U':  
                System.out.println ("vowel");  
                break;  
            default:  
                System.out.println ("consonant");  
        }  
    }  
}
```

```

public static void main (String [] args) {
    char ch;
    sc = new Scanner (System.in);
    ch = sc.next ().charAt (0);
    vowel obj = new vowel ();
    obj.letter (ch);
}
}

```

② Import java.util.\*;

```

public class weak {
    private static Scanner sc;

    void day (int n) {
        switch (n) {
            case 1:
                System.out.println ("Saturday");
                break;
            case 2:
                System.out.println ("Sunday");
                break;
            case 3:
                System.out.println ("Monday");
                break;
        }
    }
}

```



case 4:

```
system.out.println("Thursday");  
break;
```

case 5:

```
system.out.println("Wednesday");  
break;
```

case 6:

```
system.out.println("Thursday");  
break;
```

case 7:

```
system.out.println("Friday");  
break;
```

default:

```
system.out.println("Invalid");
```

```
}  
public static void main (String[] args) {
```

```
    int a;
```

```
    sc = new Scanner (system.in);
```

```
    a = sc.nextInt();
```

```
    week obj = new obj();
```

```
    obj.days(a);  
}
```

③ Import java.util.\*;

public class month {

private static Scanner sc;

void days(int n) {

switch(n) {

case 1:

System.out.println("31 days");

break;

case 2:

System.out.println("28/29 days");

break;

case 3:

System.out.println("31 days");

break;

case 4:

System.out.println("30 days");

break;

case 5:

System.out.println("31 days");

break;

case 6:

System.out.println("30 days");

break;



```
case 7:  
    System.out.println("31 days");  
    break;
```

```
case 8:  
    System.out.println("31 days");  
    break;
```

```
case 9:  
    System.out.println("30 days");  
    break;
```

```
case 10:  
    System.out.println("31 days");  
    break;
```

```
case 11:  
    System.out.println("30 days");  
    break;
```

```
case 12:  
    System.out.println("31 days");  
    break;
```

```
}  
}
```

```
public static void main (String[] args) {  
    int a;  
    Scanner sc = new Scanner(System.in);
```

a = se.nextInt();

month obj = new month();

obj.days(a);

}



```

* public class BankAccount {
    void show() {
        System.out.println("Hello world");
    }
    public static void main (String[] args) {
        BankAccount ob = new BankAccount();
        ob.show();
    }
}

```

\* Write a Java program that initialize a variable  
balance, account number, account holder name

→ credit to balance 10000tk using method name  
creditAccount();

→ debit from account 5000tk using method name  
DebitAccount();

→ show the balance with all information using  
method showBalance();

```
public class BankAccount {
```

```
    int balance = 0;
```

```
    String name = "Rahim sarkar";
```

```
    int accNumber = 1234;
```

```
    void creditAccount () {
```

```
        balance = balance + 10000;
```

```
    }
```

```
    void debitAccount () {
```

```
        balance = balance - 5000;
```

```
    }
```

```
    void showBalance () {
```

```
        System.out.println("Name = " + name + "
```

```
Account Number = " + accNumber + " Balance  
= " + balance);
```

```
}
```



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

```
    BankAccount ob = new BankAccount ();
```

```
    ob.showBalance ();
```

```
    ob.creditAccount ();
```

```
    ob.showBalance ();
```

```
    ob.debitAccount ();
```

```
    ob.showBalance ();
```

### Static Keyword

- Static keyword used in Java mainly for memory management.
- Static member belongs to class instead of specific instance/object.
- Static member can access without creating object.

→ static keyword in java can be used in

- variable
- method
- nested class
- Block

# Differentiate of static and normal variable

```
public class staticExample {
```

```
    static int a = 20;
```

```
    int b = 25;
```

```
    void show() {
```

```
        a++;
```

```
        b++;
```

```
        System.out.println("Non static variable b = " + b);
```

```
        System.out.println("Static variable a = " + a);
```

```
    }
```



```
public static void main (String [] args) {  
    staticExample ob1 = new staticExample ();  
    ob1.show();  
    staticExample ob2 = new staticExample ();  
    ob2.show();  
    System.out.print ("A=" + a + " B=" + ob1.b);  
}
```

```
# static method;  
    static int b=25;  
    static void show () {  
        }  
}
```

```
ob1.show();  
}
```

### nested-class


```
class Test {
```

```
    int x = 20;
```

```
    static class Hello {
```

```
        void show () {
```

```
            print (x);
```

```
        }   
           
        class innerclass () { }   
    }   
}
```

```
public static void main (String[] arg) {
```

```
    Test.Hello object = new Test.Hello();
```

```
    object.show();
```

```
}
```

= Test.Innerclass ob = new Test.Innerclass();



## H.W

```
public class BankAccount3 {  
    int balance;  
    String name = "Rahim";  
    int accNumber = 1234;  
  
    void creditAccount (int b)  
    {  
        balance = balance + b;  
    }  
  
    void debitAccount (int b)  
    {  
        if (balance < b)  
            System.out.println("Your account balance is not  
            adequate");  
        else  
            balance = balance - b;  
    }  
  
    void showBalance ()  
    {  
    }  
}
```

```
System.out.println("Account number: " + accNumber);
```

```
System.out.println("Name: " + name);
```

```
System.out.println("Balance: " + balance);
```

```
}
```

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

```
BankAccount ob = new BankAccount ();
```

```
ob.showBalance ();
```

```
ob.creditAccount (37821);
```

```
ob.showBalance ();
```

```
ob.debitAccount (7372);
```

```
ob.showBalance ();
```

```
ob.debitAccount (939180);
```

```
ob.showBalance ();
```

```
}  
}
```



## Program 10

```
public class BankAccount1 {  
    private static String mastery;  
    float balance;  
    String name;  
    int accountNo;  
    void creditAccount (float c) {  
        balance = balance + c;  
    }  
    void debitAccount (float d) {  
        balance = balance - d;  
    }  
    void showBalance (float b, String n, int a) {  
        System.out.println ("Name = " + n + " Account Number = " +  
            a + " Balance = " + b);  
    }  
}
```

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

```
    BankAccount ob = new BankAccount ();
```

```
    ob.showBalance (10000, "Master", 1234);
```

```
    ob.creditAccount (10000);
```

```
    ob.debitAccount (5000);
```

```
}  
}
```



## Java class

Program structure:-

- Documentation section
- package statement (oop important sect-)
- import statement (import java.util \*)
- ⇒ interface statement (public <sup>variable</sup> static <sup>final</sup>)  
method → abstract method
- class Definition
- main method Declaration

\* public <sup>access modifier</sup> class <sup>keyword</sup> Hello <sup>class name</sup>  
public <sup>object create</sup> static <sup>start at run any</sup> void <sup>class</sup> main (String args[])

System.out.println ("Hello world");

<sup>class</sup> <sup>object</sup> <sup>method</sup>

}

}

- The keyword public is an access modifiers
- The keyword static means that a method is accessible & usable without object creation
- void means main method does not return any value.

Eclipse      Textpad & Run Kit 2011  
 ———  
 APS

JDK = Java Development Kit

JRE → Java Runtime Environment

एक नाम class नाम (और नाम दिए नाम भाबर  
 निष्कर्ष 2 है ।

main function में 2 class को भाबर 3 नाम  
 file save करवा करके class भाबर ।

\* Import java.util.Scanner;

public class Text {

public static void main (String [] args) {

int a=5;

Scanner z = new Scanner (System.in);

a = z.nextInt();

System.out.println(a);

System.out.print ("The value of a is " + a);



## Constructor

Theory:-

```
class student {  
    int stdID;  
    String name, mobile;  
    student (int id, String name, String mob) {  
        stdID = id;  
        name = nm;  
        mobile = mob;  
    }  
    void show () {  
        S.out.p (---)  
    }  
}
```

```
class person {  
    P.S.V.M (---) {  
        student st = new student ("Rahul", "017...")  
        st.show();  
        student st = new student ();  
    }  
}
```

- \* default constructor
- \* no arg constructor
- \* copy constructor (object copy to)

```

Student (Student st) {
    name = st.name;
    Student st = new Student (10, "nam" "mobile");
    Student st2 = new Student (st)
  
```

\* write a java program to calculate sum of two integer using implementing copy constructor.

```

Ans: public class sum {
    int a, b;
    sum (int p, int q)
    {
        a = p;
        b = q;
    }
  
```



```
sum (sum su)
```

```
{
```

```
    a = su.a;
```

```
    b = su.b;
```

```
    int sum = su.a + su.b;
```

```
    System.out.println("Int sum = " + sum);
```

```
}
```

~~int a = 10;~~

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

```
    sum ob = new sum (10, 20);
```

```
    sum ob2 = new sum (ob);
```

```
}
```

```
}
```

```
or class sum {
```

```
    int a = 10, b = 20;
```

```
    sum (int x, int y) {
```

```
        a = x;
```

```
        b = y;
```

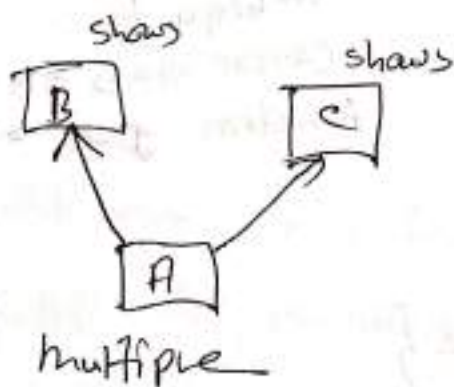
```
    }  
    sum (sum ob) {
```

```
        int a = ob.a + ob.b;
```

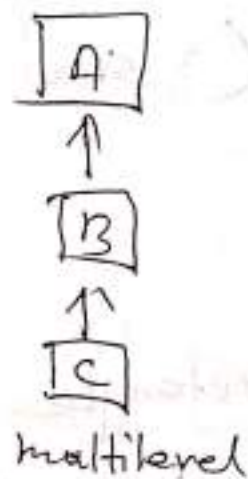
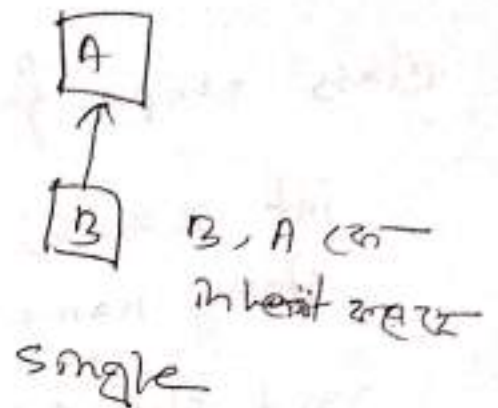
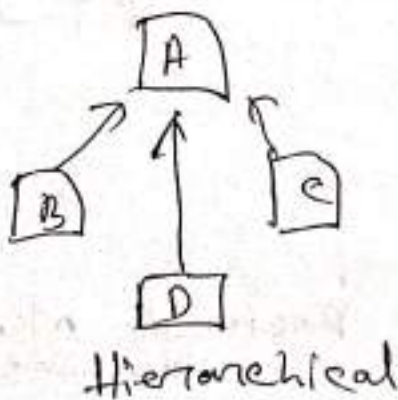
```
}
```

# Inheritance

- single inheritance
- multiple "
- multilevel "
- Hierarchical "
- Hybrid "



Java support only it





Single :

```
class shape {
```

```
    int x=20;
```

```
    String name = "Rohit";
```

```
    void show () {
```

```
        SOP ("show from shape")
```

```
    }
```

```
A a = new A ();
```

```
a.show ();
```

ambiguity

error class error

function not found

```
class Rectangle extends shape {
```

```
    void display () {
```

```
        SOP ("sam from display Rectangle")
```

```
    }
```

```
    void show () {
```

```
        SOP ("sam from show Rectangle")
```

```
    }
```

```

main ()
{
    Rectangle Rec = new Rectangle();
    Rec.show();
    Rec.display();
    cout << "x=" << Rec.x << " Name=" << Rec.Name;
}
}

```

Multilevel Inheritance.

\* WAP to implement multilevel inheritance.

```

class A
{
    int x=20;
    void show()
    {
        cout << "Sam from A";
    }
}

```



```
class B extends A {
```

```
    void display ()
```

```
    {
```

```
        sout ("sam from B");
```

```
    }
```

```
}
```

```
class C extends B {
```

```
    void print ()
```

```
    {
```

```
        gout ("sam from C");
```

```
    }
```

```
}
```

```
main ()
```

```
{
```

```
    C ob = new C();
```

```
    ob.print();
```

```
    ob.display();
```

```
    ob.show();
```

```
}
```

Hierarchical: —

```
class A {  
    }  
class B extends A {  
    }  
class C extends A {  
    }
```

- \* Define a class shape with instance variable radius, length, and width,
  - use constructor to initialize the variable
  - create three subclass circle, Rectangle, square.
  - calculate the area in each class using method and print it (use inheritance)



## polymorphism

polymorphism: It means that ability to make more than one form.

type - 2 प्रकार:

- compile time polymorphism = method overloading
- Run time polymorphism = method overriding

\* Method overloading:

① Number of parameter (sum(int x)  
sum(int x, int y))

② changes of data type of parameter (sum(int x),  
sum(float x))

③ datatype sequence (sum(int x, float y), sum  
(float y, int x))

③ Datatype sequence

```
class Test {  
    static void sub (int x, float y) {  
        float c = x + y;  
        float (c);  
    }  
}
```

```
static void sub (float y, int x) {
```

```
    float c = y + x;
```

```
    float (c);
```

```
}
```

```
main () {
```

```
    sub (3, 4.5f);
```

```
    sub (4.5f, 5);
```

```
}
```

method overloading:

- method name must same as parent class & class child class.
- Return type must be same for all.
- final method can not be overridden
- static " " " " " "
- private " " " " " "



```
class person {  
    void show() {  
        s.o.p ("gsm from person class show");  
    }  
}
```

```
class student extends person {  
    void show() {  
        s.o.p ("gsm from student class show");  
    }  
}
```

```
public static void main (String args[]) {  
    student st = new student ();  
    st.show();  
}
```

```
    person pt = new person ();  
    pt.show();  
}
```

```
    person pt2 = new student ();  
    pt2.show();  
}
```

```
}  
}
```

Super class ଏବଂ ଏହାର subclass ଏକ Reference ସ୍ୱରୂପ  
ପାରିବ ବିଷୟ - subclass ଏକ ଏହାର super class ଏକ -  
reference ପ୍ରାପ୍ତ କରିବେ ନା ।

\* super keyword → special meaning.

- super & used to refer parent class object.
- used to invoke parent class variable.
- used to " " " " constructor
- " " " " " " method.

```
class person {  
    int n=20;  
    void show () {  
        S.o.p("I am from person show");  
    }  
}
```

```
class student extends person {  
    int n=10;  
    void show () {  
        S.o.p("I am from student show");  
    }  
}
```



```

void display () {
    S.O.P ("x = " + x);
    show ();
}

```

```

void display () {
    S.O.P ("x = " + super.x);
    super.show ();
}

```

```

public static void main (String args[]) {
    Student st = new Student ();
    st.display ();
}

```

output x=10

I am from student show.

constructor:

```

class person {
    int n=20;

```

```

    person (int a) {
        person ();
        S.O.P ("I am from person");
    }

```

```
class student extends person {
```

```
    int n=10;
```

```
    student(int p) {
```

```
        n=p;
```

```
        super(p);
```

```
    }
    void display() {
```

```
        S.OP("X=" + super.n);
```

```
    }
```

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

```
    student st = new student(100);
```

```
    st.display();
```

```
}
```

### Final - Keyword

→ final keyword used to restrict the user

→ variable → final static int n; → static { n=100; }

→ method → final method (subclass override not int)

final int n;  
blank final variable.

final as class name  
initializing the variable

to have 1 time

constant final variable

int 24, final 4

int 24,

(24)



### Abstraktion

- \* normal method a body ~~ना~~ (normal class  $\rightarrow$  concrete class)
- \* Abstract a a body ~~ना~~ just declaration ~~है~~ definition ~~है~~ !

- Abstraction refers to the representation of essential features without including background details.
  - Abstraction is a process of hiding the implementation details and showing only functionality to user.
  - Abstract class (0-100%) (Object create করতে পারবে না, abstract method থাকবে)
  - Interface (100%) (abstract method → method এ পরিণত হবে, বা normal method ও থাকবে)
- ★ abstract class person

```

★ abstract class person {
    abstract void display ();
    void show ();
}
}

```

Abstract class वर गरी पळवुला  
Abstract method यावर  
अवगुणतक inherit करण  
अस implementाशन  
कराई द्या ।

{ \*abstract class normal class ko implementation deta hai  
↓  
Sub class ko abstract class se inherit method implementation deta hai (karna compulsory hai)

```
class student extends person {  
    void display () {  
        sop ("Rahim");  
    }  
}
```

```
student z = new student ();  
z.show();
```

```
class stuff extends student {  
    method implementation deta hai aur class  
    ko method hai  
}
```



Interface :- An interface in java is a blueprint of a class.

A java interface contains static constants and abstract methods. The interface in java is a mechanism to achieve abstraction.

→ There can be only abstract methods in the java interface not the method body.

Interface class or interface  
final class or final

```
Interface shape interface name {  
    int n=20;  
    void show();  
    void display();  
}
```

```
interface circle {  
    int n=10;  
    void sum();  
}
```

```
class Rectangle implements shape, circle {  
    void show();  
    void display();  
    void sum();  
}
```

interface or interface  
(2) constant variable  
constant variable  
variable (public static  
final int x=20;  
method or method  
abstract method ..

```
public static final int n = 20;  
public abstract void show();
```

② Class and Interface का मही समझ  
Abstract class and normal class का मही समझ

- \* Interface का मतलब एक interface class है  
implementation करता है।
- \* double method का मतलब है एक interface implementation  
कराएगा है।
- \* Interface use करके multiple inheritance support  
कराएगा है।

```
class A {  
    void show();  
}  
class B {  
    void show();  
}  
class C extends A, B {  
}
```

Java support करेगा न।



- \* Interface এ কখন object create করা যায় না
- \* একটি interface আরেকটি interface কে extends inherit করতে পারে।

```
Interface shape {
    int x=20;
    void show();
}
```

```
interface square {
}
```

```
interface circle extends shape, square {
    int x=30;
    void sum();
}
```

- \* Java ফোঁস multiple inheritance support করে না।  
interface এর সমস্যা -

```
interface Rectangle {
    void show();
}
```

```
interface circle {  
    void show();  
}
```

```
class shape implements Rectangle, circle {  
    void show() {  
    }  
}
```

```
public static void main() {  
    shape p = new shape();  
    p.show();  
}
```

### ★ package :

- Package in Java is a mechanism to encapsulate a group of classes, sub package & interfaces.
- package keyword used create package in Java.
- a class can have only package declaration.
- Every class is part of some package.



## ★ Advantage: -

- preventing naming conflict
- Better organization.
- providing controlled access
- Reusability.

## ★ Package IET;

```
class Student {  
    int x = 20;  
    void show();  
}
```

```
class Employee {  
    void display();  
}
```

```
package IET 21;
```

```
{  
    import IET;  
    import IET.*;  
    import IET.Student;  
    (IET.Student st = new IET.Student())  
    class person extends Student {  
        default void show();  
        public ... {  
            Student st = new Student();  
            st.show();  
        }  
    }  
}
```

```
sub package IET 21;
```

## ★ Access modifiers:

→ Default → (same package, Access modifier use  
at class)

→ public → unrestricted access everywhere

→ protected → same package & subclass of another package.

→ private → (same package) - only in class  
access (same class)

package ICT;

class Person {

default (protected) void show() {

access  
restricted

package ICT2;

import ICT;

class student extends Person {  
}



★ Differentiate <sup>between</sup> class and Interface ?

class	interface
<p>① A class describes the attributes and behaviors of an object.</p> <p>② A class may contain abstract methods, concrete methods</p> <p>③ member of a class can be public, private, protected or default.</p>	<p>① An interface contains behaviours that a class implements.</p> <p>② An interface contains only abstract methods</p> <p>③ All the members of the interface are public by default.</p>

## Differentiate abstract class and normal class

Abstract class	concrete class
<p>① An abstract class is declared using abstract modifier.</p> <p>② An abstract class cannot be directly instantiated using the new keyword.</p> <p>③ An abstract class may or may not contain abstract methods.</p> <p>④ An abstract class cannot be declared as final.</p> <p>⑤ Implementation of interface is possible by not providing implementations of all of the interface's methods. For this a child class is needed.</p>	<p>① A concrete class is not declared using abstract modifier.</p> <p>② A concrete class can be directly instantiated using the new keyword.</p> <p>③ A concrete class cannot contain an abstract method.</p> <p>④ A concrete class can be declared as final.</p> <p>⑤ Easy implementation of all the methods in the interface.</p>



## Exception handling

compile time error  $\rightarrow$  ; न मिल ए error न

Run time error  $\rightarrow$  Array size क न मिल ,

int a = 5/0 ;

Sop (a) ; Arithmetic exception

String s = null ;

Sop (s.length) ; / null point

checked exception

unchecked exception

throws exception object { try  $\rightarrow$  that statement causes an exception } exception object creator  
catch  $\rightarrow$  that statement handle the exception } exception handler

```
try {  
    a = 5/0 ;  
    Sop (a) ;  
}  
catch (Exception e) {  
    Sop (e) ;  
}
```

Example:

```
public class ExceptionHandling {  
    public static void main (String[] args) {  
        try {
```

```
            int a = 5/0; // String s = null;  
            sop(a);      sop(s.length());
```

```
        }  
        catch (Exception ob) {  
            sop(ob);  
            sop(ob + "Hello");
```

```
        }  
        sop("Hello");  
    }  
}
```

try block error  
catch block error  
or error

```
* catch (ArithmeticException e) {  
    sop(e);
```

```
}  
catch (NullPointerException ee) {  
    sop(ee);  
}
```



```

catch (Exception a) {
    sop(a);
}

```

```

* public class ExceptionHandling {
    public static void main (String [] args) {
        try {
            String s = null;
            sop(s.length());
        } catch (ArithmeticException ob) {
            sop(ob);
        } catch (NullPointerException n) {
            sop(n);
        }
        sop("Hello");
    }
}

```

finally = 100  
 finally block zarant  
 execute ho exception  
 at manjara 1

```
catch (ArrayIndexOutOfBoundsException ar) {
```

```
    sop(ar);
```

```
catch (Exception e) {
```

```
    sop(e);
```

```
finally {
```

```
    sop("I am from finally");
```

```
}
```

try এর সমস্ত  
catch ও finally  
block পিছরে  
রই।

এগাম

```
try {
```

```
    int a[] = new int[5];
```

```
    a[6] = 7;
```

```
finally {
```

```
    sop("I am from finally");
```

```
sop("Hello");
```

অস্বাভাবিক condition  
exception



throw एक keyword होता है। इसका उपयोग exception  
(अपवाद) को डालने के लिए किया जाता है।

throws यह keyword एक method को call करने के लिए उपयोग की जाती है।

Java

```
String name = textFieldName.getText();  
text.setText(name);
```