INDEX

study time

Sr. No.	Date	Title	Page No.

Advance Java Programming

OOPS :- Object-oriented Programming is a methodology or Paradigm to design a program using class & objects.

→ It simplifies the software development and maintainance by providing some concepts.

Class :- Class is a user defined data types which defines its properties and its functions.

→ Class is a logical representation of data.

Eg- Human being is a class and the body parts of human being are its properties and the action performed by the body is known as functions.

Object :- Object is a runtime entity. It is an instance of the class. An object can be represent a person place or anything. An object can operate on both data member and member functions.

★ Features of Java —

1. **Platform independent** → Java Can run on any Platform like windows, mac, linux etc.. without modification.

2. **object oriented** → Java is based on OOPs Concepts like encapsulation, inheritance and Polymorphism.

3. **Simple and familier syntax** → Java ~~has~~ syntax is based on C++ syntax and is reletavily easy to learn.

4. **Robust security** → Java has in-built security features such as memory management and data type checking.

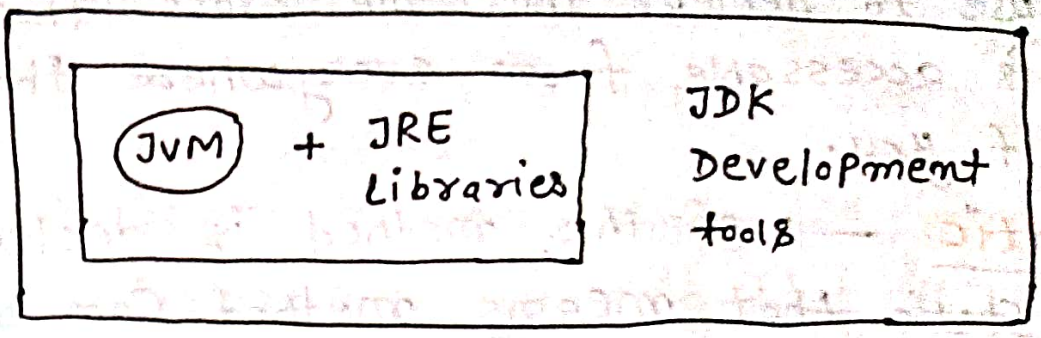
5. **Large Community** → Java has a massive Community of developers, ensuring extensive resources and ~~lib~~ libraries.

Path set :-

C:/> Program files > Java > JdK > bin > Javac

my Computer $\xrightarrow{RC}$ Properties > Adv. setting > Environment variable > System variable > Path set.

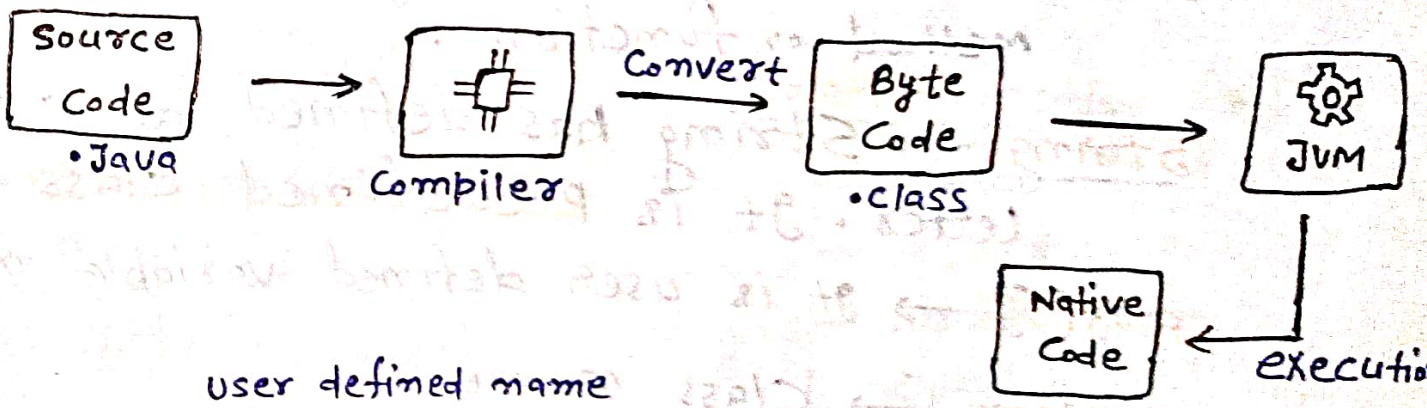
☆ How Java run Code



JDK – Java development kit

JRE – Java Runtime environment

JVM – Java vartual machine.



☆

```
                 user defined name
                 ↓
Class Test {                  keyword
                              ↗
Access specifiers → Public static void main(String arg[]){
                        System.out.println("Welcome");
  } }
```

```
cmd ↵
C:\User\HP > D: ↵
D:\> cd Java-File ↵
D:\Java-File > Javac Test.Java ↵
D:\ Java-File > Java Test ↵
```

Access specifiers → This **Pubic** keyword use to declear that the class that method is accessable from everywhere that mean if you.

Static → If this method is declear as Static that means method Can be called without creating Object.

Void → void is a keyword that define that null return type.

main → That is the signature of main Method or function.

String → String has defined as capital letter. It is **predefined class**.

arg[] → It is user defined variable name.

System → Class name

out → Variable of the System Class

. → Assuciation sign of class

print → Method of class.

ln → New line

Stack Memory – Local variable
Heap memory – object
Date – global variable
Text – Statement

```
class Test {
    public static void main(String arg[]) {
        int a=5;
        int b=12;
        String name = "Chandan";
        System.out.println (a+b) ; // 17
        System.out.println(name + a+b); //chandan512
        System.out.println(name+(a+b)); //Chandan17
    }
}
```

Data Types :-

Premetive (8)

- Boolean (1 byte)
- Byte (1 byte)
- Short (2 byte)
- long (4 byte)
- char (2 byte)
- int (4 byte)
- float (4 byte)
- double (8 byte)

Non-Premetive (4)

- Class
- object
- Array
- String

Condition

```
Class Test {
    public static void main(String arg[]) {
        int marks = 45;
        String name = "Chandan";
        if(marks ≥ 40)
            system.out.print(name + " Pass");
        else
            system.out.print(name + " Fail");
    }
}
```

output :- Chandan Pass

Loop

```
for(initialization ; Condition check; updation){
    --------
}
```

++, --

Q. Print table of 2.

```
Class Table2{
    public static void main(String arg[]){
        int count = 2;
        for(int i=01; i≤10; i++){
            system.out.println(count * i);
        }
    }
}
```

Q. print Pattern

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

```
class Pattern {
    public static void main(String arg[]) {
        for(int i=0; i<4; i++) {
            for(int j=0; j ≤ i; j++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
```

Array in Java :-

→ Array is a non-primitive data type.

→ This data type is able to store multiple value of a single data type.

int a[5] X in case of c language right but in Java wrong.

deceleration of array in Java

```
datatype VariableName[];

int arr[];
String ~~int~~ name[];
float b[];
```

create object; arr = new int[5];

name = new String[6];

Q. write a program to print an array

```
class arrayPrint1 {
    public static void main(String arg[]){
        int arr[];
        arr = new int[5];
        arr[0] = 2;
        arr[1] = 8;
        arr[2] = 4;
        arr[3] = 0;
        arr[4] = 5;
        for (int i=0; i<5; i++){
            System.out.print (arr[i]+ " ");
        }
    }
}

class arrayPrint2 {
    public static void main(String arg[]){
        int arr[];
        arr = new int[5];
        for (int i=0; i<5; i++){
            arr[i] = i * 5+1;
        }
        for (int i=0; i<5; i++){
            System.out.print (arr[i] + " ");
        }
    }
}
```

Q. write a Program to find a large number

```
Class largeNumberFind {
    public static void main(String arg[]){
        int arr[] = new int[5];
        for(int i=0; i<5; i++){
            arr[i] = i*3+2;
        }
        int max = a[0];
        for(int i=1; i<5; i++){
            if(arr[i] > max)
                max = a[i];
        }
        System.out.println(max);
    }
}
```

Q. write a Program to Copy of an array to the another array with any loop.

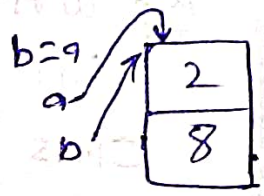
```
Class CopyArray {
    pubic static void main(string arg[]){
        int a[] = new int[5];
        int b[] = new int[5];
        for(int i=0; i<5; i++){
            a[i] = 5*i+1;
        }
        b = a;
        for(int i=0; i<5; i++){
            System.out.print(b[i]+" ");
        }
    }
}
```

Check:- if value change in array b then always change in a[].
due to pass by reference.

Q. write a program to copy of an array to the another array without any loop and both array diffent in size.

```
Class copyArrayDifferentSize {
    public static void main(String arg[]) {
        int a[]= new int[5];
        int b[]= new int[2];
        for(int i=0; i<5; i++) {
            a[i] = i*3+7;
        }
        b=a;
        for(int i=0; i<5; i++) {
            System.out.print(b[i]+" ");
        }
    }
}
```

Access to a[] Not size value of b




```
int a= new int[2];
int b= new int[5];
for(int i=0; i<2; i++) {
    a[i] = i*2+3;
}
b = a;
for(int i=0; i<5; i++) {
    System.out.print(b[i]+" ");
}
```

ERROR:
Array Index Out of Bound Exception.

Q. write a Program to extend array size.

```
class extendArraySize {
    public static void main(String arg[]) {
        int a[] = new int[5];
        int b[] = new int[5];
        for(int i=0; i<5; i++) {
            a[i] = i * 2+3;
        }
        for(int i= 0; i<5; i++) {
            b[i] = a[i];
        }
        a = new int[7];
        for(int i=0; i<5; i++) {
            a[i] = b[i];
        }
        for (int i= 0; i<7; i++) {
            System.out.print(a[i] + " ");
        }
    }
}
```

output:- 3 5 7 9 11 0 0

Q. write a program to pass a array through a method.

```
class arrayPassthroughMethod {
    static void printArray(int a[]){
        for(int i=0; i<a.length; i++){
            System.out.print(a[i] + " ");
        }
    }
    public static void main(String arg[]){
        int a[];
        a = new int[5];
        for(int i=0; i<5; i++){
            a[i] = i*2+3;
        }
        printArray(a);
    }
}
```

output: 3 5 7 9 11

Q. Anonymous Array create through a method.

```
class arrayAnonymous {
    static void printArray(int a[]){
        for(int i=0; i<a.length; i++){
            System.out.print(a[i] + " ");
        }
    }
    public static void main(String arg[]){
        printArray(new int[] {1,2,3,4});
    }
}
```

Q. write a program to return a array through a method.

```
class methodArrayReturn {
    Static int [] returnArray () {
        return new int[] {2,3,4,5};
    }
    public static void main (String arg[]) {
        // int a[] = new int [returnArray().length];
        // a = returnArray();
        // for (int i=0; i< a.length; i++) {
        //     System.out.print(a[i] + " ");
        // }
        int a[] = returnArray();
        for (int i=0; i<a.length; i++) {
            System.out.print( a[i] + " ");
        }
    }
}
```

2D Array

Q. write a Program to print a 2D Array.

```
Class arrayPrint2D {
    public static void main(String arg[]) {
        int a[][] = {{1,2,3},{4,5,6}};
        for(int i=0; i<2; i++) {
            for(int J=0; J<3; J++) {
                System.out.print(a[i][J] + " ");
            }
            System.out.println();
        }
    }
}
```

output:- 1 2 3
4 5 6.

```
Class arrayPrint2D {
    Public static void main(String arg[]) {
        int arr[][] = new int[3][3];
        for(int i=0; i<3; i++) {
            for(int J=0; J<3; J++) {
                arr[i][J] = i * J+2;
            }
        }
        for(int i=0; i<3; i++) {
            for(int J=0; J<3; J++) {
                System.out.print(arr[i][J] + " ");
            }
            System.out.println();
        }
    }
}
```

```
int a[][] = new int[2][3]; ✓
int a[][] = new int[2][ ]; ✓
int a[][] = new int[ ][3]; ✗
int a[][] = new int[ ][ ]; X
```

Q. write a Program to Print array when some value of coloum are not same in size

```
class PrintArrayNotSameSize2D {
    public static void main(String arg[]) {
        int a[][] = new int[3][ ];
        a[0] = new int[3];
        a[1] = new int[2];
        a[2] = new int[4];

        for(int i=0; i<3; i++) {
            for(int j=0; j<a[i].length; j++) {
                a[i][j] = i*j+2;
            }
        }
        for (int i=0; i<3; i++) {
            for (int j=0; j<a[i].length; j++) {
                System.out.print(a[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

3D Possible

```
int arr[][][] = new int[2][3][4]
```

Command Line Argument

Passing the argument during the execution

```
class passArgument {
    public static void main(String arg[]) {
        for (int i=0; i<arg.length; i++) {
            System.out.println(arg[i]);
        }
    }
}
```

```
C:\> Javac passArgument.Java ↵
C:\> Java passArgument 52 Ram Patna ↵
```

Output :- 52
Ram
Patna

arg[0] arg[1] arg[2]

String value

```
class passArgument2 {
    public static void main(String arg[]) {
        int b = Integer.parseInt(arg[0]);
        ~~float c = Integer f~~
        float c = Float.parseFloat(arg[1]);
        System.out.println(a+b);
    }
}
```

```
C:\> Javac passArgument2.Java ↵
C:\> Java passArgument2 5 8.2 ↵
```

Output :- 13.2

→ Because command line Argument return String type array.

```
int b = Integer.parseInt(arg[0]);
float c = Float.parseFloat(arg[0]);
double d = Double.parseDouble(arg[0]);
```

Q. Pattern print through Command Line.

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

```
class printPatternCL {
    public static void main(String arg[]) {
        int size = Integer.parseInt(arg[0]);
            for(int i=0; i<size; i++){
                for(int j=0; j<=i; j++){
                    System.out.print( i+1 + " ");
                }
                System.out.println();
            }
    }
}
```

```
C:\> Javac printPatternCL.Java
C:\> Java printPatternCL 4
```

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

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

```
C:\> printPattern 4 "*"
C:\> PrintPattern "+" 4
```

"$"
"@"

ERROR :- NumberFormatException

Q. Write a Program to Calculate the average using Command Line.

```
class calculateAverage {
    public static void main (String arg[]) {
        int size = arg.length;
        double sum = 0;
        for (int i=0; i<size; i++)
            sum += Double.parseDouble (arg[i]);
        System.out.println (sum/size);
    }
}
```

C:\> Javac CalculateAverage.Java ↵
C:\> Java CalculateAverage 4. 8 7 5 2 ↵

How to get Name of Array Class in Java

```
class getClassNameArray {
    public static void main (String arg[]) {
        int arr[] = {1,2,3,4};
        Class c = arr.getClass();
        String d = c.getName();
        System.out.println(d);
    }
}
```

output: [I

```
// system.out.println(arr.getClass().getName());
```

8. Calculate Matrix Addition

```
class matrixAddition {
    ~~public static void printA~~
    static void setArrayValue(int arr[][], int val){
        for(int i=0; i<arr.length; i++){
            for(int J=0; J<arr[i].length; J++){
                arr[i][J] = val;
                val ++;
            }
        }
    }
    public static void main(String arg[]){
        int row_arr1 = 2;
        int Col_arr1 = 2;
        int row_arr2 = 2;
        int Col_arr2 = 2;

        int arr1[][] = new int[row_arr1][Col_arr1];
        int arr2[][] = new int[row_arr2][Col_arr2];

        setArrayValue (arr1, 2);
        Set Array Value(arr2, 3);
        if (row_arr1 == row_arr2 && Col_arr1 == Col_arr2){
            for(int i=0; i<arr1.length; i++){
                for(int J=0; J<arr1[i].length; J++){
                    int add = arr1[i][J] + arr2[i][J];
                    System.out.print(add + " ");
                }
                system.out.println();
            }
        }
        else{ System.out.println("Not addition Possible");}
    }
}
```

Q. Calculate Matrix Multiplication

```
class matrixMultiplication {
    Static void setArrayValue (int arr[] , int val) {
        for(int i=0; i< arr.length ; i++) {
            for(int J=0; J< arr[i].length ; J++) {
                arr[i][J] = val ;
                val ++ ;
            }
        }
    }
    public static void main(String arg[]) {
        int row_a1 =2;
        int Col_a1 = 3 ;
        int row_a2 = 3 ;
        int Col_a2 = 3 ;

        int arr1[][] =new int [row_a1] [Col_a1] ;
        int arr2[][] = new int [row_a2] [Col_a2] ;

        setArrayValue ( arr1 , 1 ) ;
        setArrayValue (arr2 , 1 ) ;
        if ( Col_a1 == row_a2 ) {
            for(int i=0; i< arr1.length ; i++) {
                for(int J=0; J< arr2[0].length ; J++) {
                    int val=0 ;
                    for(int K=0 ; K < arr1[0].length ; K++) {
                        val += arr1[i][K] * arr2 [K][J] ;
                    }
                    System.out.print (val + " ") ;
                }
                System.out.println () ;
            }
        }
    }
}
```

Classes

→ Class is a blueprint which defines some properties and behaviours.

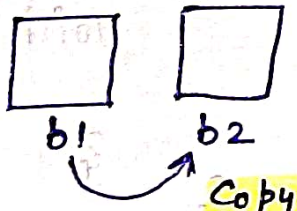
→ An object is an instance of a class which has those properties and behaviours attached.

```
Class Box{
    int l,b,h;
    void volume(){
        System.out.println(l*b*h);
    }
}
Class MBOX{
    public static void main(String arg[]){
        Box b1 = new Box();
        b1.l = 22;
        b1.b = 10;
        b1.h = 5;
        b1.volume();
    }
}
```

```
Javac MBox.Java ↵
Java MBox ↵
```

```
Class Box{
    int l,b,h;
}
Class MBox{
    public static void main(String arg[]){
        Box b1 = new Box();
        b1.l = 12;
        b1.b = 8;
        b1.h = 5;
        System.out.println(b1.l * b1.b * b1.h);
}
}
```

```
class Box{
    double l,b;
    void area(){
        System.out.println(l*b);
    }
}
Class MBox{
    public static void main(String arg[]){
        Box b1 = new Box;
        Box b2;
        b2 = b1;
        b1.l = 12;
        b1.b = 5;
        b1.area(); // 60
        b2.area(); // 60
    }
}
```



Copy by Reference

```
Class Box{
    double l,b;
    void area(){
        System.out.println(l*b);
    }
}
Class MBox{
    public static void main(String arg[]){
        Box b1 = new Box();
        Box b2;
        b2 = b1;
    // b1 = null; ← nullPointerException
    // b1 = b2;
        b1.l = 12;
        b1.b = 5;
        b1.area(); // 60
    }
}
```

if value change in any one variable then change value in both object variable.

```
class Box {
    double l, b, h;
    double volume() {
        return l*b*h;
    }
}

class MBox {
    public static void main(String arg[]) {
        Box b1 = new Box();
        Box b2 = new Box();
        Box b3;
        b1.l = 10;
        b1.b = 10;
        b1.h = 10;
        b2.l = 5;
        b2.b = 5;
        b2.h = 5;
        System.out.println(b1.volume()); // 1000
        System.out.println(b2.volume()); // 125
        b3 = b1;
        b3 = b2;
        System.out.println(b3.volume()); // 125
        System.out.println(b1.volume()); // 1000
    }
}
```

Shallow copy (Reference copy)

Deep Copy (Creating a new object)

Variable

Variable is a Container which is store a value in a memory.

Types of Variable :-

- → Instance variable
- → Local variable
- → Static variable

Instance Variable :- A variable which is decleared inside the class but outside of all the function/Method Called Instance variable.

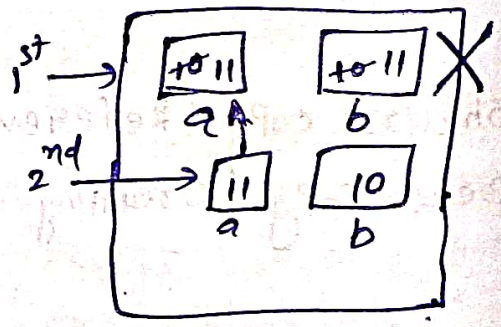
Local variable :- A variable which is decleared inside the body of the method or method Parameter Called Local variable.

- The Scope of this variable is only in the method.

Static Variable :- A variable which is decleared ~~with~~ by using the static keyword called static variable.

eg:- static int value;

```
✓ Class A{
    Static int a =10;
    Void fun(){
        int b=10;
        System.out.print(a+" "+b);
        ++a; ++b;
    }
}
```



fun() Called two times then

```
class A {
    static int b = 20; // static
    int c = 30; // Instance
    public static void main(String[] arg) {
        int a = 10; // Local
        A.ref = new A();
        System.out.println(a);
        System.out.println(A.b);
        System.out.println(ref.c);
    }
}
```

← by using direct class name

// if other class then use, direct b use in same class

Method overloading

→ Two or more methods can have the same name inside the same class if they accept different arguments, this features is known as Method overloading.

eg:- return type methodname (Para);
return type method-name (Para1, Para2);

→ Method name is same but signature of method are different.

```
void add (int a) { };
void add (double a) { };
void add (int a, double b) { };
double add (int a, int b) { };
```

→ Method overloading only focus on argument
→ You never focus on the return type.

```
void add (int a, int b) { };
int add (int a, int b) { };
```

Not Possible

→ ambigaity problem generate that means both are Same.

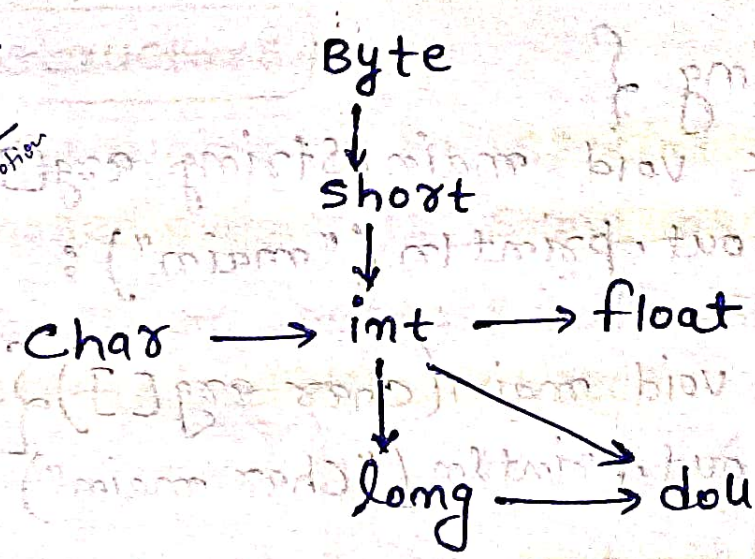
→ Method overloading work during Compilation time.

→ if function / Method signature not match then Compiler try to match simillar through Change type (data type Conversion) ~~known as type Casting~~.

```
class AM{
    void add (int a){
        System.out.println (a);
    }
    void add (int a, double b){
        System.out.println (a+b);
    }
}
Class Mo{
    public static void main (String arg[]){
        AM m = new AM();
        m.add (10);          // 10
        m.add (10, 10.9);    // 20.9
        m.add (10, 10);      // 20
    }
}
```

Change double to int

implict
explict
typ promotion




```
byte b=127;
int a=12;
b = a;X
a = b;✓
```

type Casting
b=(byte)a;

Conversion

```
class AM {
    void sum(int a, long b) {
        system.out.println(a+b);
    }
    void sum(long b, int a) {
        system.out.println(a+b);
    }
}
class MO {
    public static void main(String arg[]) {
        AM m = new AM();
        m.sum(20,20);
    }
}
```

→ Compiliation error try to conversion in int but both are same

reference to sum is ambiguous

(double int / int double) → ambiguous

5f	6
6	5f

← Problem solve

```
class overloading {
    public static void main(String arg[]){
        System.out.println("main");
    }
    public static void main(char arg[]){
        System.out.println("Char main");
    }
    public static void main(int arg[]){
        System.out.println("int main");
    }
}
```

→ only run main(string arg[])
Method overloading doing but ~~no any~~
Not run other main function.

Constructors

→ Constructor is a special type of method whose name is same as class name.

→ The main purpose of constructor is initialize the object.

→ Every java class has a Constructor.

→ A Constructor is automatically called at the time of object creation.

→ A Constructor never contain any return-type including void.

→ If a class does not have a Constructor, the Java Compiler automatically creates a default Constructor during run-time.

→ A Constructor can not be abstract or static or final.

→ The default Constructor initializes instance variables with default values.

→ A Constructor can be overloaded but can not be overridden i.e if we make Constructor then default Constructor not available.

→ If one time Constructor make then default Constructor not called.

→ <u>Default Constructor</u> :- A Constructor in which user not passing any parameter known as default Constructor.

```
class A {
    int x=10;
    int y=20;
    String name="abc";
}
```

```
A ref = new A();
```

Constructor

x = 10 y = 20
name = "abc"

```
class A {
    void A() {
        System.out.println("This is Method");
    }
    A() {
        System.out.println("This is Constructor");
    }
}
```

method

Constructor

```
class Main {
    public static void main(String arg[]) {
        A ref = new A();   // Constructor Call
        ref.A();           // Method Call
    }
}
```

Constructor

- default Constructor
- Parametrized Constructor
- Copy Constructor

Default Constructor

```
Class Student{
    int id;
    String name;
    Student (){
        system.out.println(id + " " + name);
    }
}
class MStudent{
    public static void main(String arg[]){
        Student s = new Student();
    }
}
```

output: 0 null

Parametrized Constructor

```
class Student{
    int id;
    String name;
    Student (int a, String n){
        id = a;
        name = n;
    }
    void display(){
        System.out.println(id + " " + name);
    }
}
Class MStudent{
    public static void main(String arg[]){
        Student s = new Student();
    }
}
```

output: Compilation error No suitable Constructor found. (Not run default Constructor)

Java ~~Const~~ support the Constructor overloading

```
Class A{
    A() {
        System.out.println("Constructor Call");
    }
    A (int age){
        System.out.println("your age :"+ age);
    }
}
Class AMain{
    public static void main(String arg[]){
        A a1 = new A();
        A a2 = new A(5);
    }
}
```

output:- Constructor Call
your age : 5.

Copy Constructor
[Passing through object]

→ when we pass object reference to the Constructor then it is called copy Constructor.

→ Copy data from one object to another.

```
Class Student{
    int id;
    String name;
    Student (int a, String n) {
        id = a;
        name = n;
    }
```

```
Student (Student s) {
        id = s.id ;
        name = s.name ;
    }
    void display() {
        system.out.println(id + " " + name) ;
    }
}
Class MStudent {
    public static void main(String arg[]) {
        Student s1 = new Student (24, "Abc") ;
        Student s2 = new Student (s1) ;
        s1.display() ;
        s2.display() ;
    }
}
```

output :- 24 Abc
24 Abc.

copy Constructor [without Passing object]

```
Class Student {
    int id ;
    String name ;
    Student (int i, String n) {
        id = i ;
        name = n ;
    }
    Student () { }
```

```
        void display(){
            System.out.println(id + " " + name);
        }
    }
    class MStudent {
        public static void main(String arg[]){
            Student s1 = new Student (24, "Abc");
            Student s2 = new Student ( );
            s2.id = s1.id;
            s2.name = s1.name;
            s1.display ( );
            s2.display ( );
        }
    }
```

this Keyword

1. this keyword refers to the current class object inside a method or Constructor.

```
class A {
    void show ( ){
        System.out.println (this);
    }
    public static void main(String arg[]){
        A r = new A( );
        System.out.println (r);
        A.show( );
    }
}
```

② this keyword always refer the current class instance variable.

[whenever the name of instance and local variable both are same then our runtime environment (JVM) gets confused that which one is local variable & which one is instance variable, to avoid this problem we shoud use 'this' keyword.]

```
class Student {
    int roll;
    String name;
    Student(int roll, String name) {
        this.roll = roll;
        this.name = name;
    }
    void display() {
        System.out.println(roll + " " + name);
    }
}
class MStudent {
    public static void main(String arg[]) {
        Student ref = new Student(24, "Abc");
        ref.display();
    }
}
```

```
// roll = roll
//name = name
```

output

```
0
null
```

③ this keyword is used to call the current class constructor.

```
class A {
    int a;
    A() {
        system.out.println("This is default Constructor");
    }
    A(int a) {
        this();
        this.a = a;
        System.out.println(a);
    }
}
class MA {
    public static void main(String arg[]) {
        A a1 = new A(5);
    }
}
```

output :- This is default constructor
5

④ It also called parametrized constructor of its own class.

```
class A {
    A() {
        this(10);
    }
    A(int a) {
        -----
    }
}
```

Called on main

```
A();
```

(5.) this keyword is used to call current class method.

```
class A{
    void A(){
        System.out.println("Method invoked");
    }
    void P(){
        this.A();
    }
}
class MA{
    public static void main(String arg[]){
        A a1 = new A();
        a1.P();
    }
}
```

output:- Method invoked.

(6) this keyword used to pass the argument in method call.

```
class A{
    void A(A obj){
        System.out.println("Method invoked");
    }
    void P(){
        A(this);
    }
}
class AM{
    public static void main(String arg[]){
        A a1 = new A();
        a1.P();
    }
}
```

pass current class ref.

output:- Method invoked

⑦ this keyword is used to pass the argument in Constructor Call.

```
Class A {
    B b1;
    A(B b1) {
        this.b1 = b1;
    }
    Void msg() {
        System.out.println(b1.data);
    }
}
Class B {
    int data=50;
    B() {
        A a1 = new A(this);
        a1.msg();
    }
}
Class MAB {
    public static void main(String arg[]) {
        B b1 = new B();
    }
}
```

Inheritance

When we construct a new class from existing class in such a way that the new class access all the features & properties of existing class called inheritance.

→ In Java 'extends' keyword is used to perform inheritance.

→ It provides code reusability.

→ We can't access private members of class through inheritance.

→ A sub class contains all the features of super class so, we should create the object of sub class.

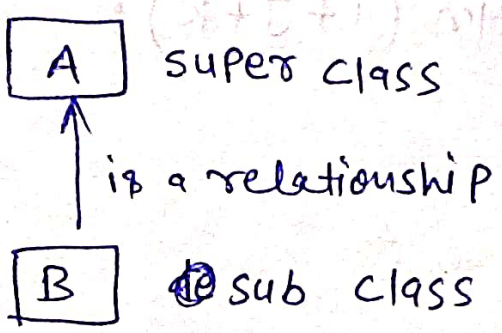
→ Method overriding only possible through inheritance

→ The class which is being inherate is called base / super / parent class. Parent class also known as stand alone class.

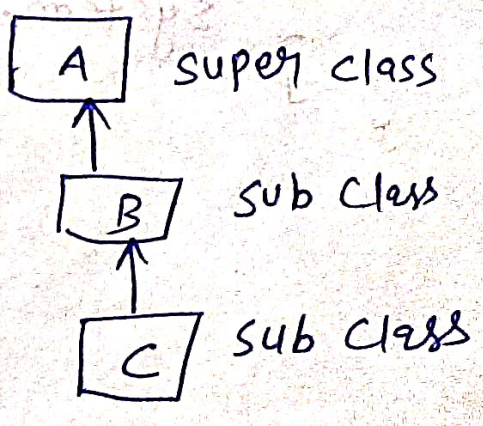
→ A class which inherit the base class known as child / sub-class / derived class.

Types of inheritance —

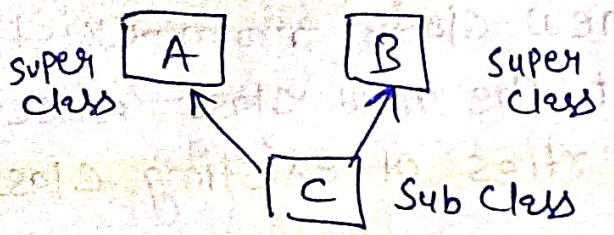
1. Single inheritance



2. Multi-level inheritance

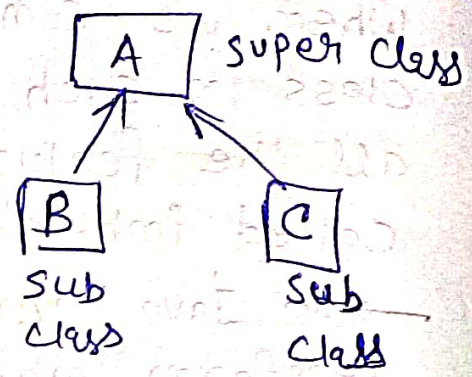


③ Multiple inheritance



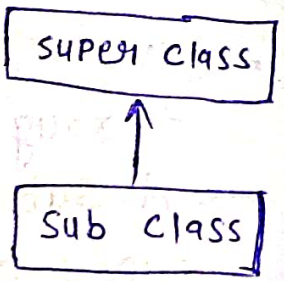
✗ Not support Java

④ Hierarchical inheritance



① Simple inheritance

which contain only one super class and only one sub class is called simple inheritance.




```
class A {
    int i, j;
    void show() {
        System.out.println(i + " " + j);
    }
}
class B extends A {
    int k;          // int k=40;
    k = 40;
    void add() {
        System.out.println(i+j+k);
    }
}
```

```
class AB {
    public static void main(String arg[]) {
        B b = new B();
        b.i = 20;
        b.J = 12;
        b.show();
        b.add();
    }
}
```

output

20 12

72

```
class AB {
    public static void main(String arg[]) {
        B b = new B();
        A a = new A();
        a.i = 20;
        a.j = 30;
        a.show();
        b.add();
    }
}
```

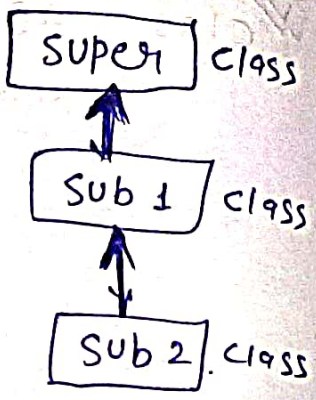
output

20 30

40

B create object and
A create ~~obj~~ another objec
So, a.i = 20 but b.i = 0 :
Simillar a.J = 30 but b.J = 0 .

2. Multi-level inheritance



In multi-level inheritance we have only one super class and multiple sub classes.

```
class A {
    int age;
    Void displayAge () {
        system.out.println (age);
    }
}
class B extends A {
    int marks;
    void displayMarks () {
        system.out.println (marks);
    }
}
Class C extends B {
    String name;
    void displayName () {
        System.out.println (name);
    }
}
Class MABC {
    public static void main(String arg[]) {
        C ref = new C();
        ref.age = 24;
        ref.marks = 90;
        ref.name = "Rahul";
        ref.displayAge();
        ref.displayMarks();
        ref.displayName();
    }
}
```

output

24
90
Rahul.

(3.) Multiple inheritance

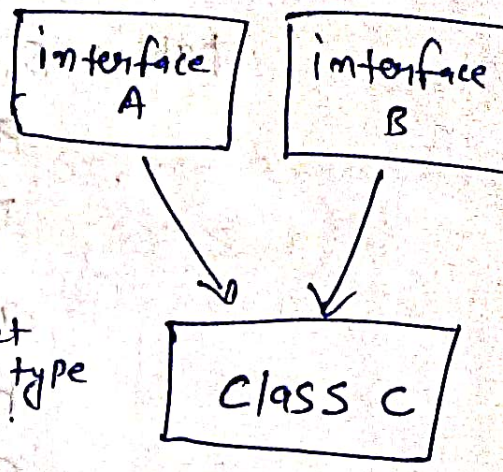
→ Not support in Java.

→ we can achieve multiple inheritance through interfaces.

→ Same method call then error.

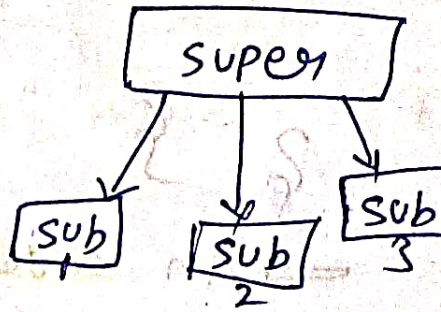
Class C extends A, B ✗ Java does not support this type of syntax!

Class C implements A, B ✓



(4.) Hierarchical inheritance

A inheritance which contain only one super class and multiple sub class and all sub class directly extends super class called hierarchical inheritance




```
Class Animal {
    void eat() {
        System.out.println("Animal eating");
    }
}
Class Dog extends Animal {
    void bark() {
        System.out.println("Dog barking");
    }
}
Class Babydog extends Animal {
    void weep() {
        System.out.println("Babydog weeping");
    }
}
```

```
class MAnimal {
    public static void main(String arg[]) {
        Babydog a = new Babydog();
        Dog b = new Dog();
        a.eat();
        b.bark();
        a.weep();
        // b.weep(); X
        // a.bark(); X
        b.eat();
    }
}
```

→ An object of sub class can be assign to the reference of super class.

It means that subclass object will be converted to superclass object.

A ← B

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

```
class Person {
    String name;
    int age;
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    void display() {
        System.out.println("This is Person Class");
        System.out.println(name + " " + age);
    }
}
```

```
class Student ~~Person~~ extends Person {
    String branch;
    int s_id;
    Student (String name, int age, String branch,
                                    int s_id) {
        super (name, age);
        this.branch = branch;
        this.s_id = s_id;
    }
    void Print () {
        System.out.println ("Student class");
        System.out.println(name + " " + age + " " +
                                branch + " " + s_id);
    }
}

class MS {
    public static void main (String arg[]) {
        Person P1 = new Person ("Abc", 18, "MCA",
                                    ~~MO~~);
        P1.display ();
    }
}
```

<u>output</u>:- This is Person Class
Abc 18

→ P1. Print (); compilation error

→ Both method are same then sub class method call

→ if different method then output given by super class method.

Method overriding

→ If a method of same name, signature and return type is implemented in sub-class that already defined in the super-class is called Method overriding.

→ Method overloading · Compile time Static Polymorphism.
Method overriding run time dynamic Polymorphism.
↳ (object involves)

```
Class Bank {
    void IntrestRate() {
        System.out.println("Bank");
    }
}
Class SBI extends Bank {
    void IntrestRate() {
        System.out.println("RI = 9 %");
    }
}
Class MS {
    public static void main(String arg[]) {
        SBI b = new SBI();
        b.IntrestRate();
    }
}
```

output – RI = 9 %

→ But `Bank b1 = new SBI();`

output :- Bank

→ In case of overriding first search on sub-class then search on super-class

→ super class reference can refer the object of the class.

```
class Bank {
    Private int m = 10;   // It Can not access
}                         outside of this class.

class Bank {
    static void IntrestRate() {
        system.out.println("This is Bank");
    }
}
class SBI extends Bank {
    static void IntrestRate2() {
        System.out.println("RI = 9 %");
    }
}
class MBank {
    public Static void main(String arg[]) {
        Bank s = new SBI();
        SBI.IntrestRate2();
    }
}
```

output :- RI = 9 %

static not make object
static is class properties
not method overriding

Static Keyword

→ Only a single copy of the static variable is created and shared among all the instances of the class.

→ Because it is a class-level variable, memory allocation of such variables only happens once when the class is

Static can be used four different ways –

1. static ~~keyword~~ variable
2. Static method
3. static block
4. Nested class.

→ static keyword used for single initilization.

→ when declear static then memory allocated to the class level memory (sharable by all object).

① Static variable

```
class student {
    String name;
    Static string branch = "MCA"
    int roll;
    student (string name, int roll) {
        this.name = name;
        this.roll = roll;
    }
    void display () {
        system.out.println(name+" "+roll+" "+
                                branch);
} }
```

```
class MStudent {
    public static void main(String arg[]) {
        Student s1 = new Student("ABC" 123);
        Student s2 = new Student("XYZ" 231);
        s1.display();
        s2.display();
    }
}
```

Student.branch

direct access

→ output:- ABC 123 MCA
XYZ 231 MCA

Q: Static keyword variable allow one time initilizatio

```
class A {
    static int Count = 0;
    A() {
        Count++;
        System.out.println(Count);
    }
}
class MA {
    public static void main(String arg[]) {
        A a1 = new A();
        A a2 = new A();
        A a3 = new A();
    }
}
```

→ output 1
2
3

Count=1
Count=2
Count=3

class level memory

② Static Method

→ Static method is accessable by the Class name rather than object.

→ Static method is only entital to access the Static variable.

```
class Student {
    String name;
    int roll;
    Static String branch = "IT";
    Student(String name, int roll) {
        this.name = name;
        this.roll = roll;
    }
    void display() {
        System.out.println(name+" "+roll+" "+branch);
    }
    Static void Change() {
        branch = "CSE";
    }
}
Class MStudent {
    public static void main(String arg[]) {
        Student s1 = new Student("ABC", 213);
        Student s2 = new Student("XYZ", 415);
        s1.display();   // ABC 213 IT
        s2.display();   // XYZ 415 IT
        Student.change();
        s1.display();   // ABC 213 CSE
        s2.display();   // XYZ 415 CSE
}}
```

→ Static method only access static variable.

```
class A {
    // int a = 20;
    static int b = 10;
    Public static void main (String arg[]) {
        // System.out.println(a);
        System.out.println (b); // 10
    }
}
```

Compilation error

'a' Cannot access because main is Static & instance variable is non-Static so, Static Method not access non-Static variable.

(3.) Static Block

→ It is used to initilized the Static variable.

→ The block is always access before the main or After the main.

```
class A {
    Static {
        system.out.println ("Static block before main");
    }
    Public static void main (String arg[]) {
        System.out.println ("This is main");
    }
    Static {
        system.out.println ("Static block after main");
    }
}
```

output:- Static block before main
Static block after main
This is main.

→ static block use outside of class then compilation error show.

```
class c {
    public static void main(String arg[]) {
        System.out.println("This is main");
        static {
            System.out.println("static block");
        }
    }
}
```

output:- Error : class interface, enum or record expect

Static keyword not used in main method because main is allready static method and in static method not allowed any static variable or keyword. static class level.

Abstraction

- Abstract class
- Interface

→ A class which contains the abstract keyword in its declaration is called abstract class.

```
abstract class A {
    void A() { }
    — — —
}
```

→ We can't create object for abstract class. (but create references.)

i.e An abstract class can't be instance created

— The use of abstract class through instance.

```
abstract class animal {
}
class Dog extends animal {
}
class Demo {
    public static void main(string arg[]) {
        animal r = new Dog();
    }
}
```

create reference

→ An abstract class may ~~have~~ or may not contain abstract method.

→ It can have abstract & non-abstract method.

→ If once the declare class abstract then we also be an abstract.

→ To use an abstract class, you have to inherite it from sub classes.

→ If a class contain partial implementation then we should declare a class as abstract

```
abstract class A {
    void A() { }
    abstract void B();
}
```

→ only implemenation not declearation called abstract Method

```
abstract class Animal {
        void eat () {
            System.out.println("Animal eat");
    }       }
Class Dog extends Animal {
        Void bark () {
            System.out.println("Dog bark");
        }
        Public static void main(String arg[]){
            Dog d1 = new Dog ();
            d1. eat ();
            d1. bark ();
        // Animal ref = new Animal();
    }   }
```

Can't create object of abst class.

```
abstract class Animal {
        void eat () {
            System.out.println("Animal eat");
        }
    class abstract void walk ();
}
class Dog extends Animal {
        void bark() {
            System.out.println("Dog bark");
        }
        void walk() {
            System.out.println("Dog walk");
        }
}
```

```
class Main {
    public static void main (String arg[]){
        Dog ref =new Dog();
        ref.eat();
        ref.bark();
        ref.walk();
    }
}
```

→ Animal eat
Dog bark
Dog walk.

If abstract method not emplement in sub class.

```
abstract class Animal {
        abstract void walk();
}
class Dog extends Animal {
        void bark() {
            system.out.println("Dog bark");
        }
        public static void main (String arg[]){
            Dog ref = new Dog();
            ref.bark();
        }
}
```

→ Compilertime error.
walk() must be override in sub class.

abstract Dog extends Animal and new sub class Create then ~~imp~~ override walk();

```
abstract class Animal {
    void eat() {
        System.out.println("Animal eat")
    }
    abstract void walk();
}

abstract class Dog extends Animal {
    void bark() {
        System.out.println("Dog bark");
    }
}

class Dog1 extends Dog {
    void walk() {
        System.out.println("Dog1 walk");
    }
}

class Main {
    public static void main(String arg[]) {
        Dog1 ref = new Dog1();
        ref.eat();
        ref.bark();
        ref.walk();
    }
}
```

→ Animal eat
Dog bark
Dog1 walk.

Abstract class may have a Constructor.

```
abstract class Parent {
    Parent() {
        System.out.println("This is Parent");
    }
    Void sayHello {
        System.out.Println("Hello");
    }
    abstract void greet();
}
Class Child extends Parent {
    void greet() {
        System.out.println("Good Morning");
    }
}
~~class main {~~
    ~~public static void main(String arg[]) {~~
class Child2 extends Parent {
    void greet() {
        System.out.println("Good evening");
    }
}
Class Main {
    public static void main(String arg[]) {
        Child ref = new Child();
        ref.greet();
    }
}
```

→ This is Parent
Good morning.

```
abstract class Bike {
        Bike () {
            system.out.println("This is bike");
        }
        void getGear {
            system.out.println("Gear change");
        }
}
class Honda extends Bike {
        Honda () { // super ();
            System.out.println("This is honda");
        }
        public static void main (String arg[]) {
            Honda ref = new Honda ();
            ref.getGear ();
        }
}
```

Not write compulsorily

output →

```
This is bike
This is honda
Gear change.
```

→ Abstract class have a Static Method:

```
abstract class Bike {
        Bike ( ) {
            System.out.println ("This is bike.");
        }
        Static void getGear ( ) {
            System.out.println ("Gear change");
        }
        abstract void run ( );
}
```

```
class Main3 extends Bike {
    void display(){
        System.out.println("This is display");
    }
    void run(){
        System.out.println("Bike run");
    }
    static void getGear(){
        System.out.println("Bike Gear Change.")
    }
    public static void main(String arg[]){
        Main3 ref = new Main3();
        ref.display();
        Bike.getGear();          // Main3.getGear()
        ref.run();               // getGear();
    }
}
```

override but not change in output

output →

```
This is bike.
This is display.
Gear Change
Bike run.
```

↓ output

Bike Gear Cha

Interface

→ Interface is the blueprint of Class.
→ Interface achieve 100% abstraction.
→ multiple inheritence supported by Java through interface.

```
Class A {
    int i;
    void add() {
    }
}
```

```
interface A {
    int i;
    void add();
}
```

```
interface A {
    static int i;
    abstract void add();
}
```

Computer read

wrong

interface X
↑ extends
Class A

interface X
↑ extends
interface A

is a

interface A extends X

interface X
↑ implements
Class A

class A implements X

```
interface A {
    int i = 20;
    void add();
}

~~inte~~class X implements A {
    Public void add() {
        System.out.println("This is X Class");
    }
}

class Main2 {
    Public static void main(String arg[]) {
        X ref = new X();
        ref.add();
    }
}
```

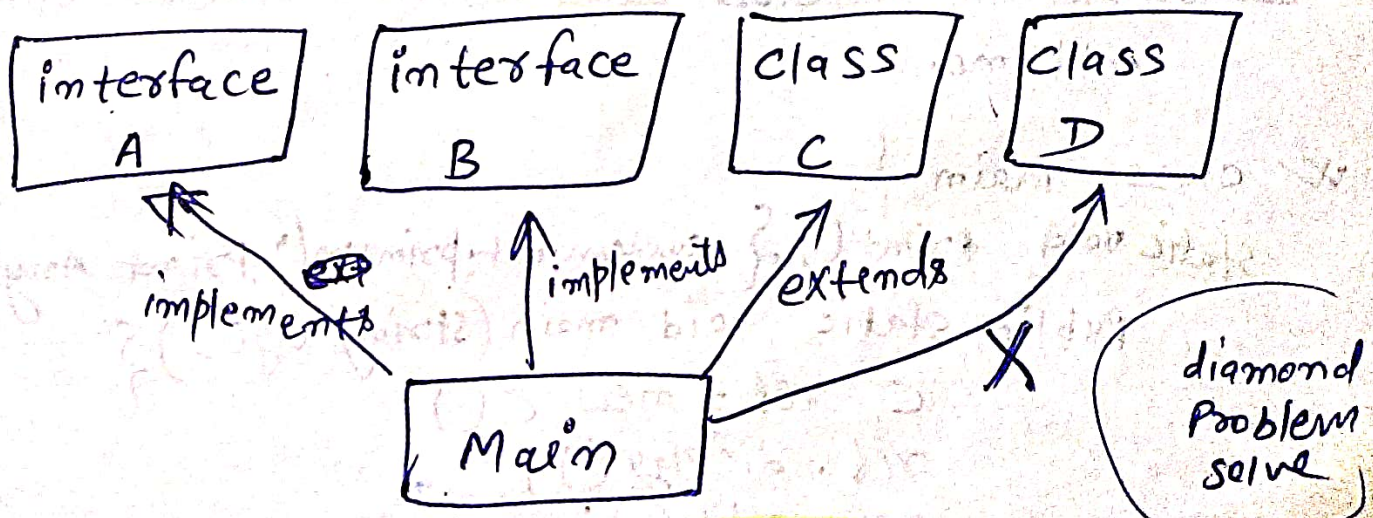
→ This is X Class

```
class Main3 {
    Public static void main(String arg[]) {
        X ref = new X();
        ref.add();
        System.out.println(X.i);
        ref.i
    }
}
```

prefer this type

but both are accessable

→ This is Class
20



class Main extends C implements A, B

~~class main extends C implements~~

```
class A {
    int i=20;
    void display() {
        system.out.println("This is A Class");
} }

interface B {
    void add();
}

Class C extends A implements B {
    Public void add() {
        System.out.println("Add no");
    }
}

Class Main {
    public static void main(String arg[]) {
        C ref = new C();
        ref.display();
        ref.add();
} }
```

→ This is A Class
Add no.

```
class Main {
Static void print() { system.out.println("Print Anything"); }
    Public static void main(String arg[]) {
        C ref = new C();
        ref.display();
        ref.add();
        Print();
} }
```

```
interface A{
    int i=20;
    void add();
}
interface X extends A{
    void add2();
}
class main implements X{
    public void add(){
        System.out.println("This is normall add");
    }
    public void add2(){
        System.out.println("This is add2");
    }
    public static void main(String arg[]){
        main ref = new main();
        ref.add();
        ref.add2();
    }
}
```

```
class Main extends A implements C,D
class Main extends A implements C
class Main implements C,D,E
```

```
interface A{
    int i=20;
    void square(int i);
}
```

07/10/24

MultiThreading

→ Thread is predefined class

Using default Constructor.

```
class myth extends Thread
{
    Public void run()
    {
        System.out.println("This is 1st thread");
    }
    Public static void main(String arg[])
    {
        myth th = new myth();
        th.run();
        myth th1 = new myth();
        th1.run();
    }
}
```

→ user defined body.

→ Thread is activated when you are able to Call the Start method.

→ th.start()

this ~~is auto~~ internally Calls run method and then th & th1 will share the stack pointer.

```
Class myth extends Thread {
    Public void run() {
        for(int i=0; i<10; i++) {
            System.out.println(i);
        }
    }
    Public static void main(String arg[]) {
        myth th = new myth();
        myth th1 = new myth();
        th.run();
        th1.run();
    }
}
```

o/p –

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

th.start();
th1.start();

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

→ No sequence will be maintained in output because both the objects share the same memory. (Stack)

* String Can also be Passed through default Constructor.

```
Class myth extends Thread {
    Public void run(){
        for(int i=0; i<10; i++){
            System.out.println(i);
    }   }
    Public static void main(String arg[]) {
        myth th, th1;
        th = new myth("First String");
        th1 = new myth();
        th.start();
        th1.start();
    }
}
```

String s = th.getName()
SOP(a)
to get thread name

4.) * Using Runnable Interface.

```
Class myth implements Runnable {
    Public void run(){
        for(int i=0; i<10, i++){
            System.out.println(i);
    }   }
    Public static void main(String arg[]) {
        myth th = new myth();
        Thread t = new Thread(th);
        t.Start();
        System.out.println(t.getName());
    }
}
```

Thread (Runnable R)

thread Name get.

* Basic Methods of Threads

1. Run () –

2. Start () – To internally Call the run() method.

3. Public void sleeping (long miliseconds) – This method is used to Pause the thread for Perticular milisecond.
   ↓
   always through exception (without this unusual stop)

4. Public void Join () – This method is used to wait for a method to die.
   - In this method without Parameter Pass after 1000 ms automically thread will die i.e will reach the termination state.

5. Public void Join (long milliseconds)

6. Public int getPriority () – If multiple instance is created. which instance will be given how ~~man~~ much priority
   Generally : 5 (all)

7. Public int setPriority (int Priority) – If we want to lower the Priority of any instance.

8. Public String getName () – To get Thread name

9. Public String setName () – To set Thread name

10. Public boolean isActive () – This method is used to return true or false.

if active → True or 1
if inactive → false or 0

Before Start : false
After Start : True.

08/10/12

Thread Synchronization

↳ only applicable where common/same memory (Resource) is shared.

→ we need to create the class and share among threads.

→ Threads allready share resources but Programmatically it is not Possible

Eg:- (1)

```
Class table {
    void Printt(int n) {
        for (int i=0; i<10; i++) {
            system.out.Println(i * n);
        }
    }
}
class thread1 extends Thread {
    table t;
    thread1 (table t) {
        this.t = t;
    }
    Public void run() {
        t.Printt(5)
    }
}
class thread2 extends Thread {
    table t;
    thread2 (table t) {
        this.t = t;
    }
    Public void run() {
        t.Printt(50);
    }
}
```

t is made local to access Printt method of table

```
class thread3 extends Thread {
    table t;
    table3 (table t) {
        this.t = t;
    }
    Public void run() {
        t.Print(500);
    }
}

class TMain {
    Public Static void main (String arg[]) {
        table t = new table();
        thread t1 = new thread1 (t);
        thread t2 = new thread2 (t);
        thread t3 = new thread3 (t);
        t1.Start();
        t2.Start();
        t3.start();
    }
}
```

output :- Not Predefined

Note :- Two ways of Synchronization

1. By synchronize the method (sharable method).
2. By synchronize the block.

Keyword - "synchronized"
- If method, Synchronized void Print(int n)
- If Block, then

```
Class table {
    void Printt (int n) {
        Synchronized (this) {
            for (int i=1; i<10; i++) {
                SOP(i+n);
```

Note :- If there is no common resource we don't need synchronization

Exception

→ Exception is an "Abnormal Event" that occurs during Runtime.
↳ if can disturb the flow of the Program.

Error : That Can't be rectify by User, Compile time error Can be rectified.

```
Class Number {
    Public void run() {
        int a, b;
        a = Integer.ParseInt(arg[0]);  → a=10
        b = Integer.ParseInt(arg[1]);  → a = 0
        System.out.println(a/b);
    }
}
```

→ If a=10, b=0, then exception occurs and if in any line exception occurs, Rest of the lines (even through they are Corrected) are not executed.

NOTE : "Exception is a class".

- Throwable → if Comes from object (superclass of Java)
  - Exception
    - → I/O Exception
    - → SQL Exception
    - → Class-Not-found Exception
    - → Runtime Exception
      - → Arithmatic Exception
      - → Null Pointer Exception
      - → Number Format Exception
      - → Index Out of Bound Exception → Array Index Out of Bound Exception, String Index Out of Bound Exception
  - Error
    - → StackOverflow Error
    - → VirtualMemory Error
    - → OutofMemory Error

Categories of Exception

There are two Categories of Exception.

(i) Checked Exception : checked exception are those class which directly inherit the throwable class except Runtime exception and Error.

i.e I/O Exception
i.e SQL Exception
i.e ClassNot found Exception } during Compile time

(ii) Unchecked Exception : [checked during Runtime]

The class which inherit the runtime Exception class are unchecked Exception.

i.e Arithmatic
Null Pointer
Numberformat
IndexoutofBound — Array, String.

09/10/24 * By using Byte Code we Can achieve Portability and robutness.

Keyword :- These 5 keywords are used in exception handling.

1. try
2. Catch } These two are always used in Pair.
3. finally
4. throw
5. throws

```
* Class exception {
    Public static void main(String arg[]) {
        int a, b;
        a = Integer.ParseInt(arg[0]);
        b = Integer.ParseInt(arg[1]);
        System.out.println(a/b);  ---> this line maybe create exception if a=21 b=0
        System.out.println("we are Indian");
        System.out.println(a+b);
    }
}
```

suppose Arithmetic exception occurs then,

```
* Class exception {
    Public static void main(String arg[]) {
        int a,b;
        a = Integer.ParseInt(arg[0]);
        b = Integer.ParseInt(arg[1]);
        try {
            system.out.println(a/b);
            system.out.println("we are indian");
            system.out.println(a+b);
        }
        catch(ArithmeticException e){
            System.out.println(e);  ---> There Can be multiple statements
        }  <---- only goes to Catch if exception occurs
    }
}
```

output Java exception 23 0 ↵

Java.lang.ArithmeticException

In try { } → There may be multiple lines of Code inside this

→ if any exception occurs it is the responsibility of Programmer to throw the same exception object.

↓

If we are unable to identify the exception under Catch(Exception e).

1 try → 1 or multiple Catch

1 Catch → 1 try

superclass of exception

↓

output Java.lang.ArithmaticException.

* NullPointerException

in array = a.length;
in string = a.length();

```
class exception {
    public static void main(String arg[]) {
        String s = null;        // "null" → No exception will occurs.
        try {
            System.out.println(s.length());
        }
        Catch(NullPointerException e){
            System.out.println(e);
        }
    }
}
```

output :- Java.lang.NullPointerException

* NumberFormatException

```
Class exception {
    public static void main(String arg[]){
        String s = "null";
        try {
            System.out.println (Integer.ParseInt(s));
        }
        Catch (NumberFormat~~Format~~Exception e){
            System.out.println (e);
        }
    }
}
```

output:- Java.lang.NumberFormatException: for input String: "null".

* ArrayIndexoutofBoundsException

```
Class exception {
    public static void main(String arg[]){
        int a[];
        a = new int[5];
        try {
            a[6] = 10;
        } Catch (ArrayIndexoutofBoundException e){
            System.out.println(e)
        }
    }
}
```

output:- Java.ArrayIndexoutofBoundsException: Index 6 out of bounds for length 5.

~~try {~~
~~System.out.println(a.length~~

ex

```
class exceptiom {
    public static void main(String arg[]){
        int a[] ;
        a = new int[5] ;
        String s= null ;
        try {
            system.out.printtln ( s.length() );
            a[6] = 10 ;
        }
        Catch( Exceptiom e) {
            System.out.printtln (e.) ;
}  }  }
```

output:- Java.lang.NullPointerExceptiom.

Note:- In try block, the first statement which creates exceptiom, only that Catch block will be executed, and rest of the lines creating exceptiom will be skipped (ignored) i.e not execute.

```
# class Exceptionmain {
    public static void main(String arg[]) {
    int a[];
    a = new int[5];
    String s = null;
    try {
        System.out.println(s.length());
    }
    Catch (ArithematicException e) {
        System.out.println(e);
    }
    Catch (NumberFormatException e) {
        System.out.println(e);
    }
    Catch (NullPointerException e) {
        System.out.println(e);
    }
}
}
```

output :- Java.lang.NullPointerException

```
# class ExceptionMain {
    public static void main(String arg[]) {
    int a[] = new int[5];
    String s = null;
    try {
        System.out.println(s.length());
    }
    Catch (Exception e) {
        System.out.println(e);
    }
```

```
catch(NullPointerException e){
        System.out.println(e);
}
catch(NumberFormatException e){
        System.out.println(e);
}
catch(ArithmeticException e){
        System.out.println(e);
}
}
```

output :- CompilationError
exception _____ has already been caught.

But, `catch(Exception e){ }`
→ this catch used as last catch statement then No Compilation Error show.
→ Exception handle by above Exception catch statement.

Nested try Catch

```
# class student { public static void main(String arg[]){
    int a = 10; int b = 0;
    try {
        System.out.println("welcome to outer try");
        System.out.println(a/b);
        try { }
        catch(Exception e){
            system.out.println(e);
        }
    }
    catch(Exception e){
        system.out.println(e);
    }
}}
```

```
# Class Student { public static void main(String arg[]){
    int a = 10;
    int b = 0;
    try {
        system.out.println("welcome outer try");
        try {
            system.out.println(" Inner try");
            System.out.println(a/b);
        }
        Catch(Exception e) {
            System.out.println(e);
        }
    }
    Catch(Exception e) {
        System.out.println(e);
    }
}}
```

finally :- ~~It used to execute code~~

→ It used with exceptions - try catch statements.

→ finally block is the block that always execute.

→ The finally block in Java is used to put important codes such as clean up code, closing the file or closing the connection.

Three case are possibles where finally block can be used

1. when an execution does not rise
2. when the execution rises and handles by the catch block.
3. when exception rise and not handled by the catch block.

```
# class Main1 {
    public static void main(String arg[]){
        int a= 4 ;
        int b= 0 ;
        try {
            System.out.println(a/b);
        }
        catch(Exceptiom e) {
            System.out.println(e);
        }
        finally {
            System.out.println("Always execute");
        }
        System.out.println("End of Code");
    }
}
```

<u>output</u>:- Java.lang.ArithmeticException: / by Zero
Always execute
End of code.

```
# class Main1 {
    public static void main(String arg[]){
        int a= 4 ; int b= 0;
        try {
            system.out.println(a/b);
        }
        finally {
            system.out.println("Always execute");
        }
        system.out.println("End of Code");
    }
}
```

<u>output</u>:- ~~finally~~ Always execute
Error ____

→ finally block contains the code which is always executed whether the exception is handled or not.

→ It is used to execute code containing instructions to release the system resources, close ~~the~~ a connection etc...

The throw keyword

→ The throw keyword is used to generate/throw an exception explicitly by the programmer.

```
# class ThrowMain{
        public void validate (int age){
            if (age < 18) {
                try {
                    throw new ArithmeticException("Not
                                                valid voter");
                }
                catch (ArithmeticException e) {
                    System.out.println (e);
            } }
            else {
                System.out.println ("Valid voter");
            }
        }
}
class Main2{
        public static void main(String arg[]){
            ThrowMain ref =new ThrowMain();
            ref.Validate();
}  }
```

output:- Java.lang.ArithmeticException: Not valid voter.

The throws exception

→ The Java throws keyword is used to declare an exception. This gives an information to the programmer that there might be an exception so its better to be parepared with a try Catch block.

```
public void validate (int age) throws IOException{
        // Code
}
```

import java.io.*;
Packge impo

```
# public class Main2 {
    public static int divide (int a , int b) throws Arithmetic
                                                Exception{
        return a/b;
    }
    public static void main (String arg[]) {
        try {
            int result = divide(10, 0);
            System.out.println(result);
        }
        Catch (ArithmeticException e) {
            System.out.println(e);
        }
    }
}
```

<u>output</u>:- Java.lang. ArithmeticException: / by zero.

Package :- Package is the Collection of Classes & interfaces but Constant is that Package always try to Contain simillar types of classes.

There are two types of Package in Java –

1. offered by Java JDK (Systematic Package)
2. User defined Package.

E:\Java\mypack\A.Java

```
package mypack;
public class A {
    public void display () {
        System.out.println("A class");
    }
}
```

mypack
A.Java
A.Class

E:\Java\Amain.Java

```
import mypack.*;
class Amain {
    public static void main(String arg[]) {
        A a1 = new A();
        a1.display(); // display method Can't be access
    }
}
```

Error removed by add public Class & Pubic metho

1. Runtime Path set
2. set Path in system variables.

-classPath.

~~those~~ Copy location ~~to~~ : E:\Java

set or add CLASSPATH (E:\Java)

Access modifers :- It is used to Control class, Method, Variables, Constructor etc...

Four types of Access modifers -

- → default (no-modifer)
- → public
- → private
- → protected

Default - with in this Class & Package is accessable

Public - In this Package & Class accessable from anywhere.

private - limition within this class. (same class).

protected - Just like default only with in Package is accessable but also sub-class accessable.

```
→ package mypack;
  public class A {
      int a = 10;
      private void AA() { }
      public void display() { }
  }

→ import mypack.*;
  class APack {
      public static void main(String arg[]) {
          A a1 = new A();
          // a1.AA();  ←—— Can't access because private.
          a1.display();
  }
}
```

Scanner class → Accept input through User's.

```
import java.util.*;
//import java.util.Scanner;
Scanner s = new Scanner(System.in)
// java.util.Scanner s = new java.util.Scanner(system.in)
int a = s.nextInt();
String b = s.nextLine(); //next();
Float f = s.nexFloat();
```

Source input.

```
import.util.*;
class ScannerClass {
    public static void main(String arg[]) {
        Scanner sc = new Scanner(system.in);
        System.out.print("Enter the Number:");
        int a = sc.nextInt();
        System.out.print("Enter the name:");
        String b = sc.next();
    }
}
```

→ javac ScannerClass.java ↵
java ScannerClass ↵
Enter the Number : 44 ↵
Enter the name : chandan ↵

write a Program to find number is a Prime or not

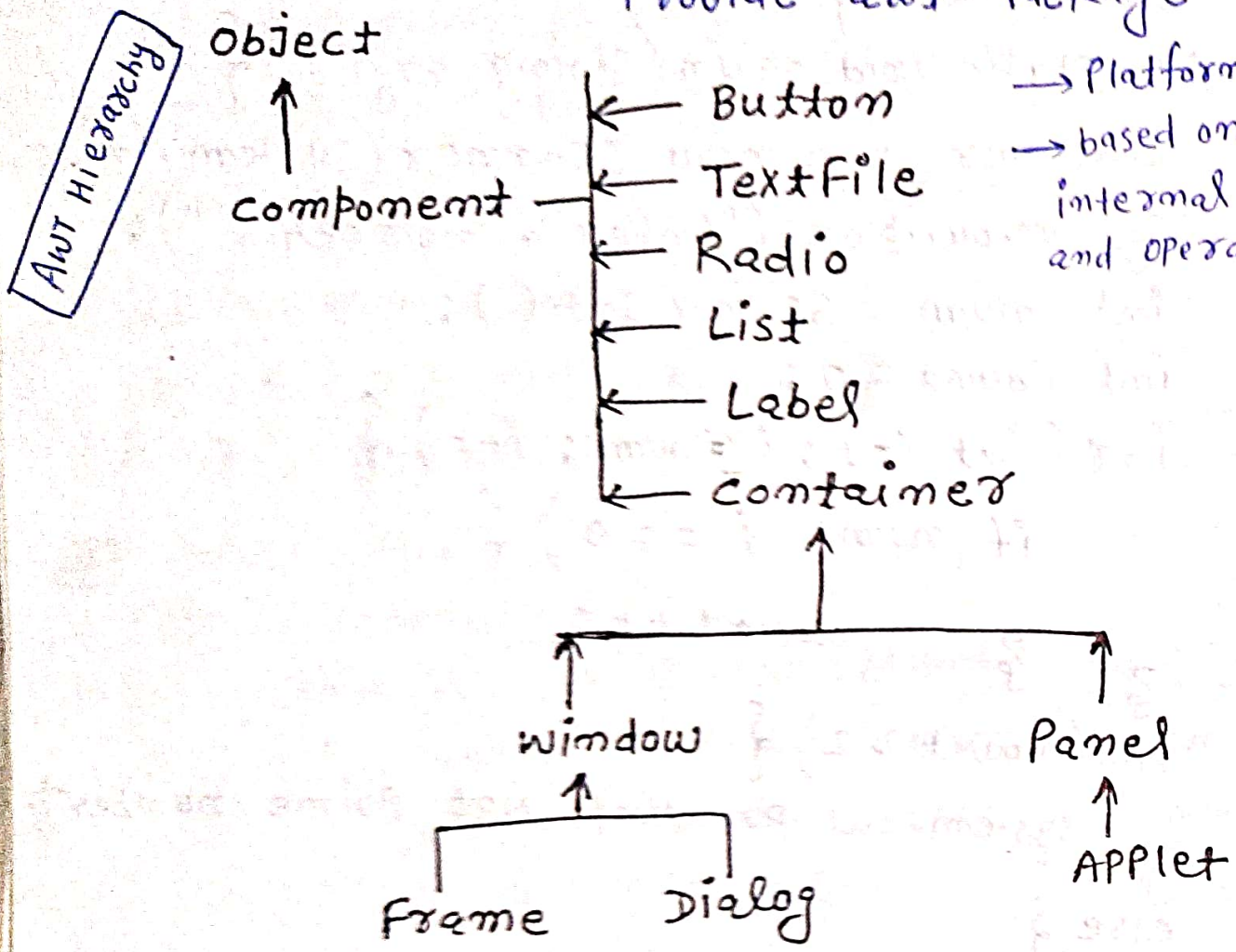
```
import java.util.Scanner;
class PrimeCheck {
    public static void main(String arg[]){
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number : ");
        int num = sc.nextInt();
        int Count = 0;
        for (int i=1; i<=num; i++) {
            if (num % i == 0) {
                Count ++;
            }
        }
        if (Count > 2) {
            system.out.println("Not Prime number");
        }
        else {
            system.out.println("Prime number");
        }
    }
}
```

OR

```
int Count = 0;
for (int i=2; i ≤ num/2; i++) {
    if (num % i == 0) {
        ~~brea~~ Count ++;
        break;
    }
}
if (Count == 0) system.out.println("Prime number");
else system.out.println("Not Prime number");
```

⊛ AWT (Abstract window Tool Kit) :-

→ java.awt.*; → To design GUI afflication Java Provide awt Package.



→ Platform dependent
→ based on system internal architecture and operating system

Container :- Container is a class which Contain the other Component. It has been inheriated by two sub-class
1. window
2. Panel

window :- window is a Container which does not have border or menu bar.

Panel :- It is a Container which does not have title bar.

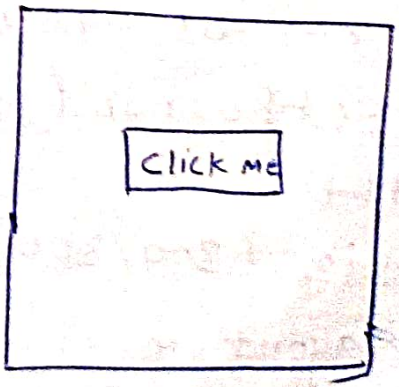
Frame :- It is also a Container which has border, menu bar or title bar.

Frame

1. public void add (Component c);
2. public void setSize (int x , int y); ← in pixel
3. public void setLayout (Layout Manager); ← null
4. public void setVisible (true); ← pass boolean value

Using Frame class -

```
import java.awt.*;
class awtMain extends Frame {
    awtMain () {
        Button b = new Button("Click Me");
        b.setBounds (50, 70, 50, 20);
                     x   y   w   h
        add (b);
        setSize (300, 300);
        setLayout (null);
        setVisible (true);
    }
    public static void main(String arg[]) {
        awtMain aw = new awtMain();
    }
}
```



Using by Association –

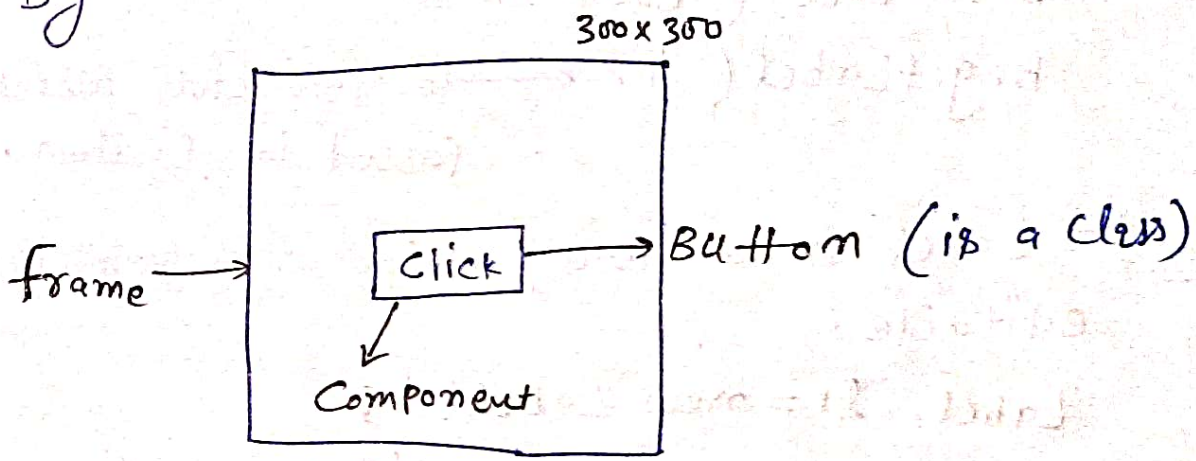
```
import java.awt.*;
class awtMain {
    awtMain() {
        Frame f = new Frame (" This is title");
        Button b = new Button ("Click Me");
        b.setBounds (50, 70, 50, 20);
        f.add (b);
        f.setSize (300, 300);
        f.setLayout (null);
        f.setVisible (true);
    }
    public static void main(String arg[]) {
        awtMain aw = new awtMain();
    }
}
```

```
import java.awt.*;
class awtMain extends Frame {
    awtMain() {
        TextField t1 = new TextField ();
        t1.setBounds (100, 120 ,200, 30);
        add (t1);
        Button b1 = new Button("Click Me");
        b1.setBounds (100, 160, 50, 20);
        add(b1);
        setSize(300, 300);
        setLayout(null);
        setVisible (true);
    }
}
```

```
Public static void main (String arg[]) {
        awtMain  aw = new awtMain ();
}
}
```

AWT
Event handling — Everthing will be done under Constructors.

* There are two ways to write "awt Class".
  1. By Inheriting the frame class
  2. By Association.



Textfield — (Editable field) we use default Constructor Can also use Parameter, Textfield (int Co

TextArea — For Multiple rows.

11.11.24

Event — when state of an object is changed.

Component — Button();

By default — Pass argument

```
Button b = new Button();
Button b1 = new Button("Click Me");
```

Method — void setLabel(String name);

→ only use for by default Button

```
b.setLabel("Login In");
b.getLabel();
```

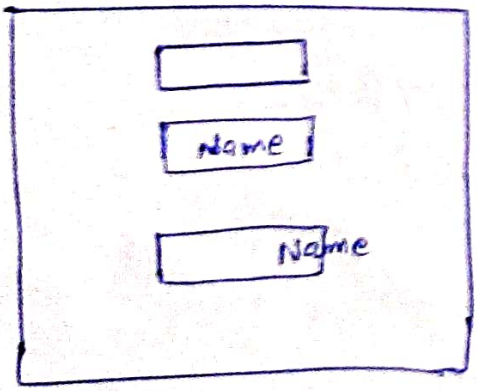
← to retreive which is passed in Button.

Label :- Label is a field / Component which is not editable.

```
Label l1 = new Label();
Label l2 = new Label("Name :");
Label l3 = new Label("Name", Label.RIGHT);
```

three static variables are used

0 — Label.LEFT
2 — Label.RIGHT
1 — Label.CENTER



Method for Label :

① void setText (String text) ; ← when default label is used.

② String getText () ; ← Always return the string

③ void setAlignment (int alignment) ; ← if earlier alignment was not used or want it again.

④ int getAlignment () ;
↑ for retrieving the alignment.

Notes – By default alignment is **LEFT**

TextField :– Always single row

① TextField ();
② TextField (String str);
③ TextField (int Cols) ; ← size of Column.
④ TextField (String str, int Cols);

TextField have some methods for setting "echo character
↑ Password character.

* void setEchoChar (char c);

```
TextField tf = new TextField ();
tf. setEchoChar('*');
char c = tf.getEchoChar ();
boolean b = tf. echoCharIsSet();
```

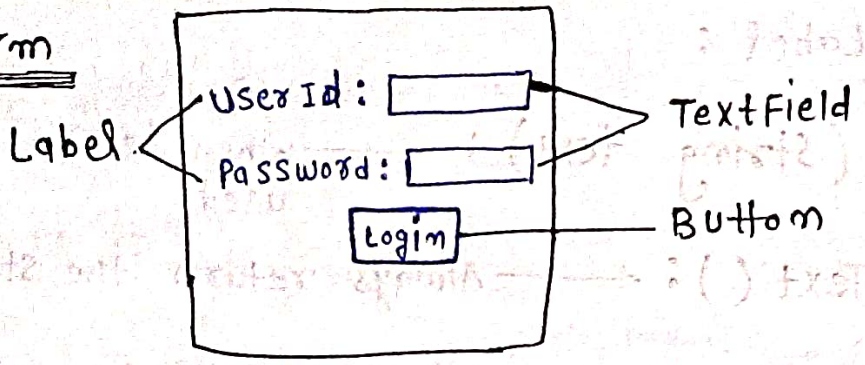
use only single character whatever character will be passed will apper in *.

← will return the Character

→ If nothing is passed, will return null.

↑ will return true/false, whether echoChar is set or not.

Login Form




```
import java.awt.*;
Public class LoginForm extends Frame {
    LoginForm ( ){
        Label userIdLabel = new Label("User Id : ", Label.LEFT);
        UserIdLabel.setBounds (100, 150, 80, 30);
        add (UserIdLabel);

        TextField UserId = new TextField ( );
        UserId.setBounds (200, 150, 150, 30);
        add (UserId);

        Label PasswordLabel = new Label ("Password : ", Label.LEFT
        PasswordLabel.setBounds (100, 200, 80, 30);
        add (PasswordLabel);

        TextField Password = new TextField ( );
        password.setBounds (200, 200, 150, 30);
        add (password);
        password.setEchoChar ('*');

        Button loginBtn = new Button (" Login");
        loginBtn.setBounds (210, 240, 80, 30);
        add (loginBtn);

        setSize (600, 600);
        setLayout (null);
        setVisible (true);
    }
```

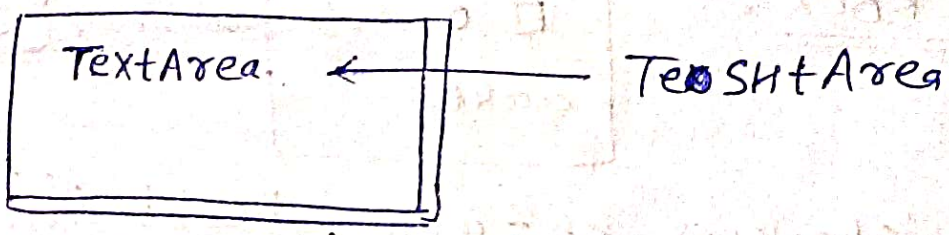
```
public static void main(String arg[]){
        LoginForm ref = new LoginForm();
    }
}
```

TextArea :-

```
TextArea t1 = new TextArea();
TextArea t2 = new TextArea(String str);
TextArea t3 = new TextArea(int cols, int rows);
TextArea t4 = new TextArea(String str, int cols
                                    int rows);
```

Methods –

```
TextArea ta = new TextArea();
void append(String s);  ← to add String after
                          TextArea words/Strings
ta.append("Hello");
System.out.println(ta.getColumns()+" "+ta.getRows());
void insert(String s, int Position);
ta.insert("SH", 2);
```



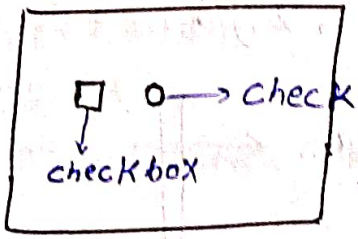

```
void replaceRange(String s, int start, int end);
ta.replaceRange("Sm", 2, 5);
```

Exclude ↑ ↑ include.

```
setColumns();
setRows();
```

```
import java.awt.*;
Public class Address extends Frame{
    Address ( ){
        ~~setSize(~~
        Label l1 = new Label ("Enter Address : ");
        l1.setBounds (100, 100, 150, 20);
        add (l1);

        TextArea ta = new TextArea ( );
        ta.setBounds(100, 130, 210, 100);
        add (ta);

        setSize (400, 400);
        setLayout (null);
        setVisible (true);
    }
    public static void main (String arg[ ]){
        Address ref = new Address( );
    }
}
```

CheckBox :-

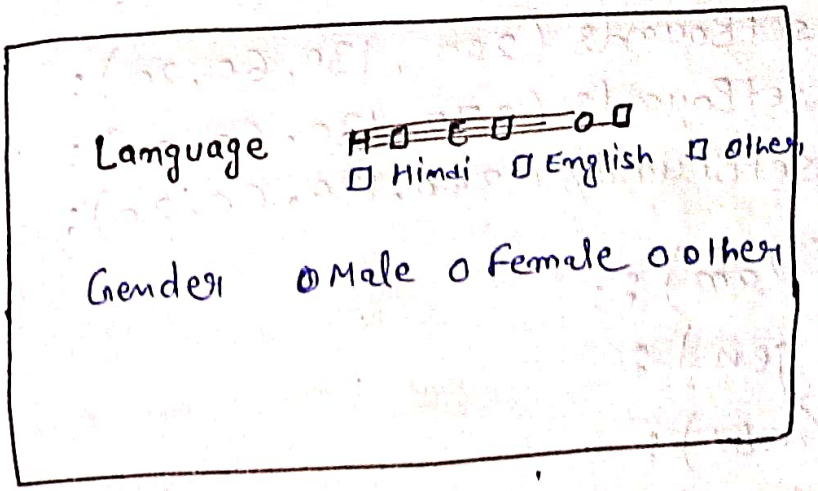


→ checkboxGroup [Just like Radio button]

```
Checkbox cb = new Checkbox( );
Checkbox cb2 = new Checkbox (String s);
Checkbox cb3 = new Checkbox (String s, boolean val);
Checkbox( );
Checkbox ("English");
Checkbox ("English", false);
```

```
boolean getState ( );
cb.setState ( );
```

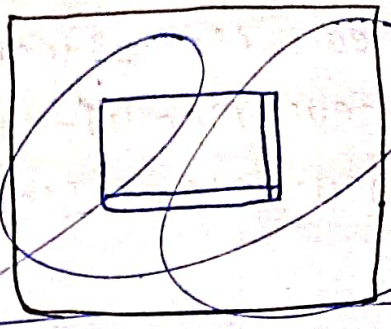
```
CheckboxGroup cbg = new CheckboxGroup();
Checkbox cbg1 = new Checkbox("M", cbg, false);
Checkbox cbg2 = new Checkbox("F", cbg, false);
```




```
import java.awt.*;
Public class awtMain extends Frame {
    awtMain() {
        Label lam = new Label("Language ");
        Label gen = new Label("Gender ");

        Checkbox cb1 = new Checkbox("Hindi", false);
        Checkbox cb2 = new Checkbox("English", false);
        Checkbox cb3 = new Checkbox("Other", false);

        CheckboxGroup cbg = new CheckboxGroup();
        Checkbox cbg1 = new Checkbox("Male", cbg, false);
        Checkbox cbg2 = new Checkbox("Female", cbg, false);
        Checkbox cbg3 = new Chekbox("Other", cbg, false);

        lam.setBounds(100, 100, 80, 20);
        gen.setBounds(100, 130, 80, 20);
```

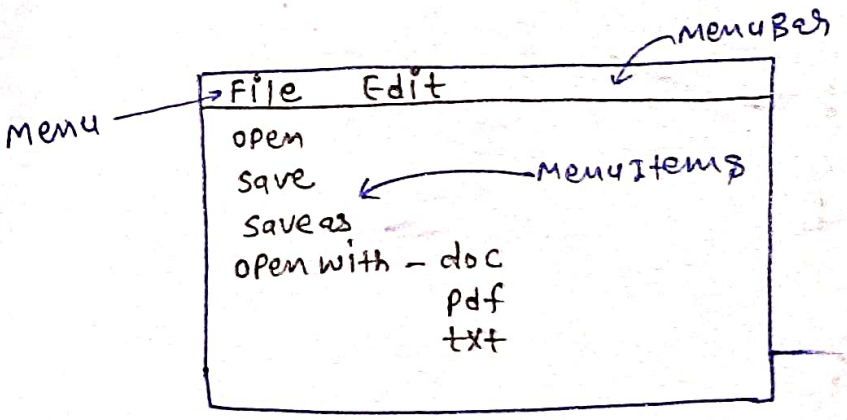
```
cb1.setBounds(200, 100, 60, 20);
cb2.setBounds(270, 100, 60, 20);
cb3.setBounds(340, 100, 60, 20);

cbg1.setBounds(200, 130, 60, 20);
cbg2.setBounds(270, 130, 60, 20);
cbg3.setBounds(340, 130, 60, 20);

add(lan);
add(gen);
add(cb1);
add(cb2);
add(cb3);
add(cbg1);
add(cbg2);
add(cbg3);
setSize(600, 600);
setLayout(null);
setVisible(true);
}
public static void main(String arg[]){
    awtMain ref = new awtMain();
}
}
```

TextArea :-



MenuBar :-

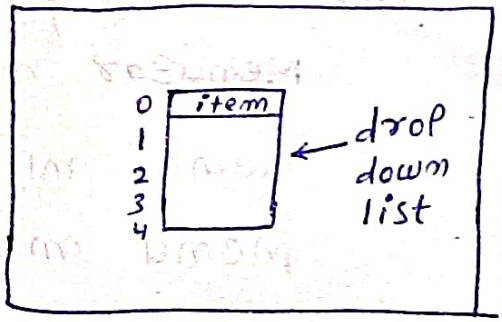
```
MenuBar mb = new MenuBar();
Menu m1 = new Menu(String s);
MenuItem mt = new MenuItem(String s);
m1.add(mt);
mb.add(m1);
```




```
import java.awt.*;
class awtMain extends Frame {
  awtMain() {
    MenuBar mb = new MenuBar();

    Menu m1 = new Menu("File");
    Menu m2 = new Menu("Edit");

    MenuItem mt1 = new MenuItem("open");
    MenuItem mt2 = new MenuItem("Save");
    MenuItem mt3 = new MenuItem("Save as");
    Menu m3 = new Menu("open with");
```

```
MenuItem mt4 = new MenuItem("doc");
MenuItem mt5 = new MenuItem("Pdf");
MenuItem mt6 = new MenuItem("txt");
m3.add(mt4);
m3.add(mt5);
m3.add(mt6);
m1.add(mt1);
m1.add(mt2);
m1.add(mt3);
m1.add(m3);
mb.add(m1);
mb.add(m2);
setMenuBar(mb);
setSize(400,400);
setLayout(null);
setVisible(true);
}
Public static void main(String arg[]){
        awtMain ref = new awtMain();
}
}
```

Choice :-

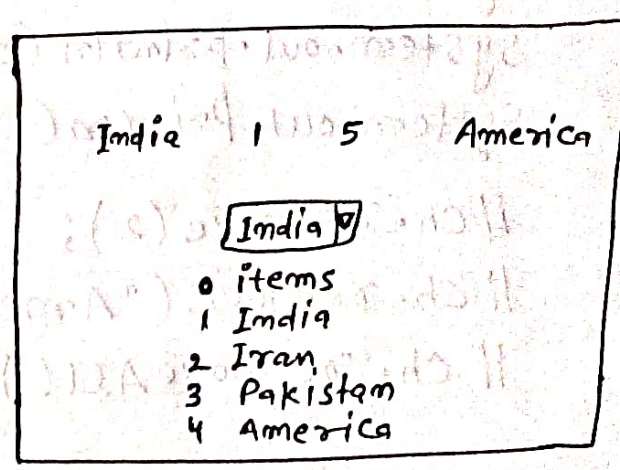



```
Choice c = new Choice();
void add(String s);
void select(int position);
String getSelectedItem();
int getSelectedIndex();
String getItem(int index);
```

index starting with zero (0).

```
int getItemCount();
void insert(String text, int Position);
void remove(String text);
void remove(int position);
void removeAll();
```

```
import java.awt.*;
Public class AwtMain extends Frame {
  AwtMain(){
    Choice ch = new Choice();
    ch.setBounds(100, 150, 100, 100);
    add(ch);
    ch.add("India");
    ch.add("Iran");
    ch.add("Pakistan");
    ch.add("America");
    ch.insert("items", 0);

  //ch.select(2);
    ch.select("India");

    Label l1 = new Label();
    Label l2 = new Label();
    Label l3 = new Label();
    Label l4 = new Label();

    l1.setBounds(100, 100, 50, 20);
    l2.setBounds(150, 100, 50, 20);
    l3.setBounds(200, 100, 50, 20);
    l4.setBounds(250, 100, 50, 20);
```



```
l1.setText(ch.getSelectedItem());
l2.setText(String.valueOf(ch.getSelectedIndex()));
l3.setText(String.valueOf(ch.getItemCount()));
l4.setText(ch.getItem(4));

add(l1);
add(l2);
add(l3);
add(l4);

System.out.println(ch.getSelectedItem());
System.out.println(ch.getSelectedIndex());
System.out.println(ch.getItemCount());
System.out.Println(ch.getItem(4));

//ch.remove(2);
//ch.remove("America");
//ch.removeAll();

setSize(400, 400);
setLayout(null);
setVisible(true);
}
Public static void main(String arg[]){
        AwtMain ref = new AwtMain();
}}
```

List :-