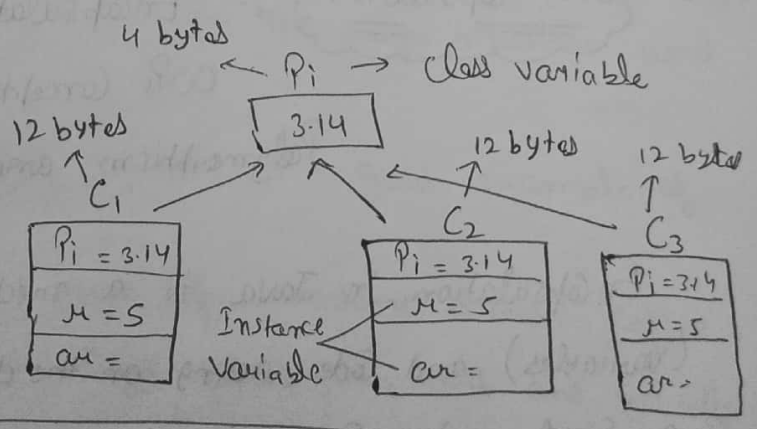
★ Static Members :- Static variables and methods

```
Class CIRCLE
{
    public :
    Static float Pi ; || class variable
    float r, ar ; || Instance variables

    Void Area ()
    {
        Cout << Pi *r *r ;
    }
    Static Void fun () { ——— } || Static method.
};

float CIRCLE :: Pi = 3.14 ; || creation of static variable
                              it will take value '0' by
                              default.
```

belongs to class

```
int main ()
{
    CIRCLE C1, C2, C3 ;
    C1. r = 5 ;
    C2. r = 10 ;
    C3. r = 20
    Cout << C1. Pi ;
    Cout << C2. Pi ;
    Cout << C3. Pi ;
    C1. fun () ;
    C2. area () ;

    CIRCLE :: fun () ;
}
```



Need :- Here we can see that Pi value is same for C1, C2 & C3 and it occupy 4 bytes for every object, and if there are 100 objects, then it occupy 400 bytes, so to avoid this, we have to make it class variable, so that it will static for all.

Scanned with CamScanner

⇒ **Static Variables :-**

- Static variables are known as class variables.
- By default it takes "0" value.
- It belongs to class instead of instances object.
- Its creation takes place outside the class using Scope resolution (::) operator.
- It can be used by duo instance methods and static methods.

⇒ **Static Methods :-**

- Static Method is called using class Name with (::) operator.
- But It can use only static variables.
- It belongs to class.

☆ **ENC- Capsulation** :- Encapsulation is one of the four fundamental OOPs Concepts. The other three are inheritance, Polymorphism and abstraction.

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data ([redacted] methods) and together as a single unit. In Encapsulation, the variables of a class will be Hidden from other classes, and can be accessed only through the methods of their current class, therefore, it is also known as Data - Hiding.

Scanned with CamScanner

To Acheive Encapsulation in Java :-

- Declare the variables of a class as Private.
- Provide Public setter and getter methods to modify and view the variables values.

★ **Abstraction** :- objects in an OOP Language provide an abstraction that hides the internal implemantation details. Similar to the Cofee machine your kitchen, you just need to know which methods of the object are availaible to call and input (which) parameters are needed to trigger a specific operation. But you don't need to understand how this method is implemanted and which kinds of actions it has to perform to create the expected result.

Example :- • I'm Cofee addict. So, when I wake up in the morning, I go to my kitchen, switch on the Cofee machine and make Cofee.

- Making Coffee with a Coffee machine is a good example of abstraction.

- You need to know how to use your Coffee machine to make Coffee. You need to provide water and Coffee beans, switch it and get Coffee.
- The thing you don't need to know is how the Coffee machine is working internally to brew a fresh cup of delecious Coffee. you don't need to know the ideal temperature of the water or the amount of ground Coffee you need to use.

Scanned with CamScanner

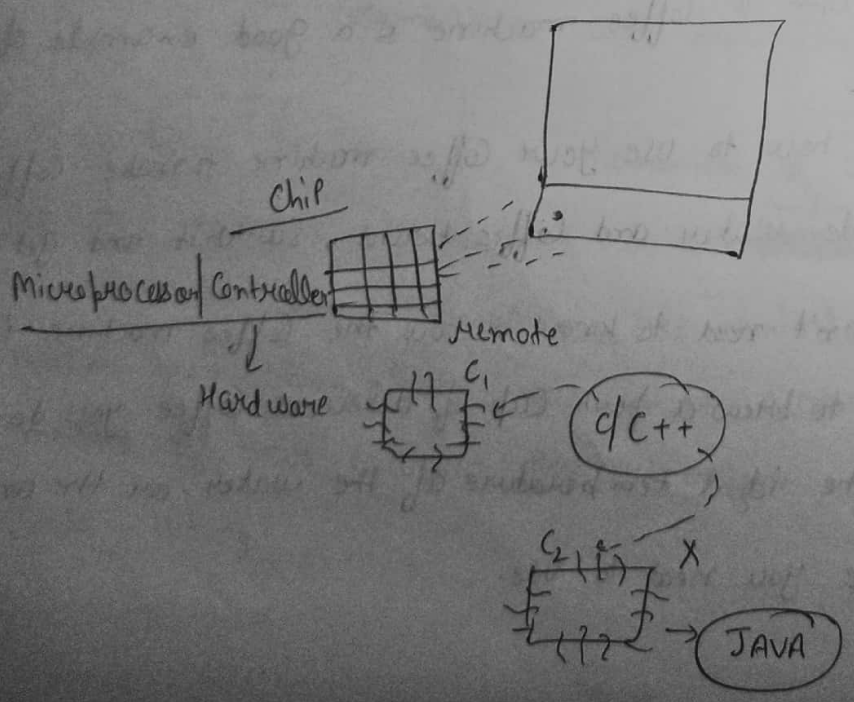
JAVA

- In 1991 By James Gosling
  ↓
  Green Team

- was developed for electronic Equipments

Why, Need ? → Earlier c/c++ was used
→ Hardware dependent Language

As we know, in our TV Remotes, there is a microcontroller called chip, which is used to control TV, as earlier these chips were developed in c/c++ as we can say these hardwares, they are having more speed but not reliable and once developed hardware can't be used for other, So, this is the reason of Reliability and Reusability, Java is introduced with some new features.



+ Added new features
(Platform independent)

Scanned with CamScanner

★ **JDK** :- Java Development kit used for Developing Java applications.

JDK 16 ⇒ JAVA 16 , C:/Pf/Java/JDK 16/Bin >

Versions :-

	1.0	1.5
Major	1.1	1.6
	1.2	1.8
Archieve changes	1.5	1.10
		1.11
		1.16

+ new classes (Libraries are added)

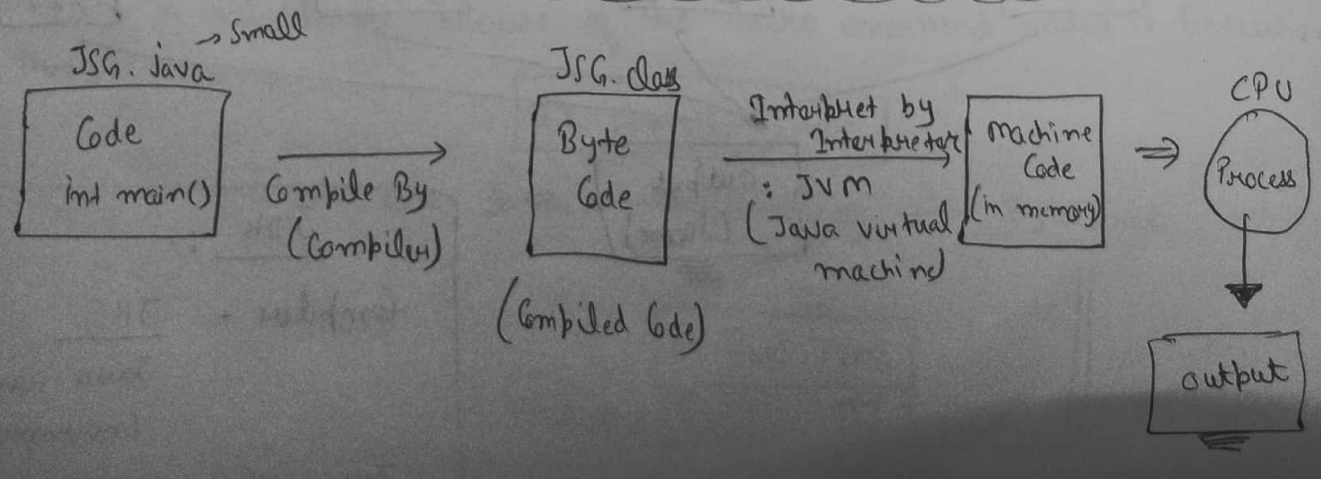
Standard Platform :- Java 2

This kit include following tools :-

(i) Java Runtime Enviromant (JRE)
(ii) Compilar (Javac)
(iii) Interpreter / Loadar (Java)
(iv) Archievar (Jar)
(v) and other tools.

→ ① Java Compilar ⇒ Javac.exe
+
② Java interpretar ⇒ Java.exe

⇒ **Compilation and Execution process of program** :-



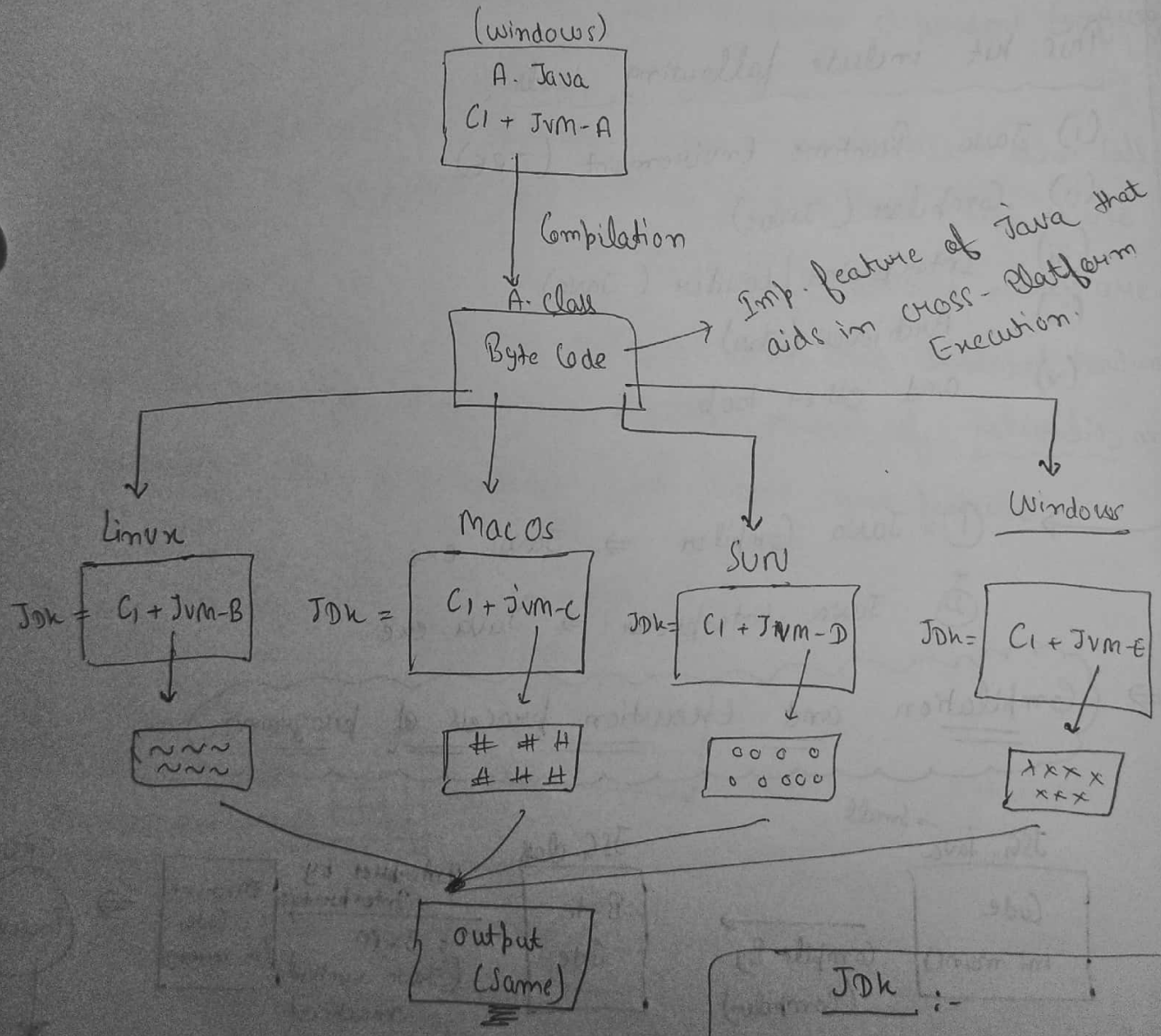
Scanned with CamScanner

* Features of Java :- PLATFORM INDEPENDENCE

"Write once Run Anywhere".

Compile any OS — Independent — OS

It allows a Java Program to run on any operating System like Windows, Linux, Mac or Solaris.



JDK :-

Compiler + JRE

Java Run time Enviromant

JRE = Class Lib. + JVM.

Scanned with CamScanner

⇒ "JVM Provides a Platform - Independent Way of Executing Code. Programmers can Concentrate on writing software, without having to be Concerned with how or where it will run". But Note that JVM itself is not Platform independent. It only helps Java to be executed in a platform - Independent Way".

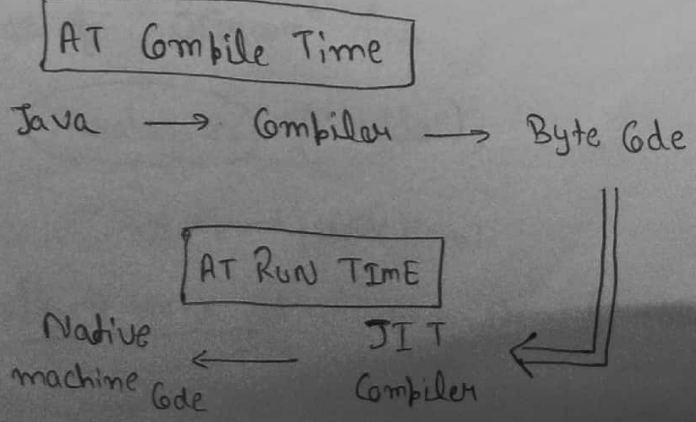
H.W

[redacted]

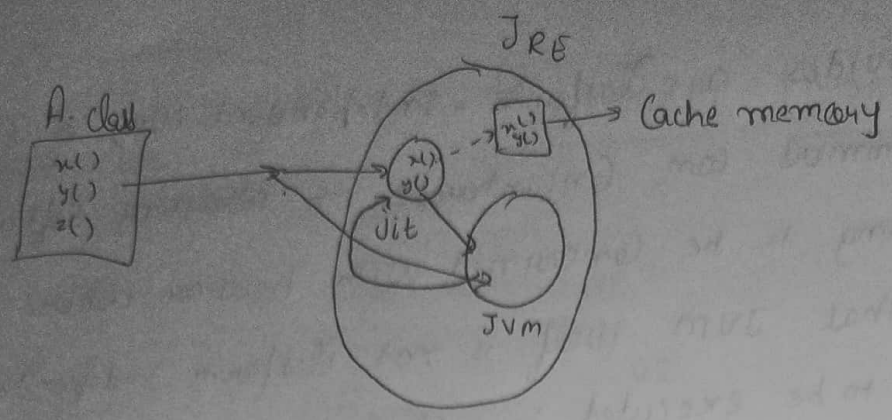
Q - Write a note on Jit Compiler?

A - The Just-In-time (Jit) Compiler is an essential part of JRE, that is responsible for performance optimization of Java based applications at run time. Compiler is one of the key aspects in deciding performance of an application for both parties, i.e. the end user and application Developer.

The JIT Compiler is able to perform certain Simple optimizations while Compiling a Series of bytecode to native machine Language. Some of these optimizations are data-analysis, reduction of memory accesses by register allocation, etc. The greater is the degree of optimization done, the more time a JIT Compiler Spends in the Execution Stage. Therefore, it Cannot afford to do all the optimizations that a Static Compiler is Capable of, because of the extra overhead added to Execution time.



Scanned with CamScanner



Now if user wants to access method x(), than it will process this request to JVM than JVM to Jit Compiler and Jit Compiler will perform some optimizations and return this code to JVM but before this, it will saved to Cache memory also, so when the requests method x() again, it will directly call from Cache memory, this is how it reduces memory access by register allocation.

Scanned with CamScanner

Class name must be same as file name. — JSG.java → (Small Letters)

```
Class JSG
{
  public static void main (String args[])
  {
    System.out.print ("Hello In");
    System.out.println ("Programmers");
    System.out.println ("How are you");
  }
}
```

- public → Accessible every where
- static → belongs to class
- void → No return
- main → method name
- String → A Libarary Class
- args[] → Array of objects
- System → A Libarary Class
- out → Object inside System
- print → method

Hello
Programmers
How are you → output

In C++ :-

```
Cout << "Hello";
```

<< → operator overloaded function

Cout → an object of "Ostream" class

Cout.operator << ("Hello");

operator << → Used to print | write | show

Internally it maybe as :- Cout.show ("Hello");

Scanned with CamScanner

18/05/21 ☆ Basics :-

```
Class JSG2
{
    public static void main (String args [])
    {
        int, a = 5, b = 10, Sum; // Decleration-cum-initialization
            Sum = a+b;
        System.out.println (Sum); // 15
        System.out.println ("Sum =" + Sum); // Sum = 15
        System.out.println (Sum + " is Sum"); // 15 is Sum
        System.out.println ("Sum =" + a+b); // Sum = 510
                                  ①   ②
                                  Priorities
        System.out.println ("multiply =" + a*b); // multiply = 50
                                  ②   ①
                                  Priorities
        System.out.println ("Sum =" + (a+b)); // Sum = 15
        // - - - - - - - - - - - - - - - -
            int x, y;
                x = 2, y = 3; // Error (Comma Initialisation is not allowed)
            System.out.println (x * y); // Error
        // - - - - - - - - - - - - - - - -
            float r = 10, ar;
                ar = 3.14 * r * r; // Error (found float return double)
                        Use this if don't want to use double
            System.out.println (ar); 314
```

X int x = 5.2; X

Scanned with CamScanner

float x = 2.0 ; X → need 8 bytes space

4 bytes

① double x = 2.0 ; ✓

8 bytes

② float x = 2.0f ; ✓ → floating Constant 4 bytes

Now, 3.14 → double Constant → need 8 bytes space, it is not a floating Constant

```
int z ; // must be Initialized
System.out.println(z); // Error
```

// - - - - - - - - - - - - - -

```
int rate , unit = 100;
if (unit <= 200)
{
  rate = 5;
  System.out.println(rate); // Error
}
```

Conditional Initialization is not allowed

if we use blocks, than it will print 5, bcz, than it gets value or else it will not enter in Condition (i.e don't print anything).

// - - - - - - - - - - - - - -

```
int r, u = 100;
if (u == 100)
    r = 2;
else
    r = 3;
System.out.println (r);
}
}
```

it will surely work bcz if it will not enter into 'if-Condition', it will surely enters into else part.

Scanned with CamScanner

☆ Data types :-

'A' → $(65)_{10}$

DATA TYPES

Boolean Type

→ boolean — 1 bit
Size not Specified in Java

Range :- True/false

Default :- false

```
boolean bl;
System.out.println(bl);//false
```

Numeric Type

Integral Type

→ Character Type

→ Integer Type

Floating type

→ float — 4 bytes
0.0f → default

→ double — 8 bytes
0.0 → default

Ref. type

Classes

Default :- NULL

Character Type

→ char → 2 bytes (16 bits)

Range :- 0 to $2^{16}-1$
0 to 65535

```
byte b1 = 127; ✓
byte b2 = -128; ✓
byte b3 = 128; ✗ ERROR
```

Integer Type

→ byte → 1 byte (8 bits)
Range :- -2^{7} to $2^{7}-1$

→ Short → 2 bytes (16 bits)
Range :- -2^{15} to $2^{15}-1$
i.e. -32768 to 32767

→ int → 4 bytes (32 bits)
Range -2^{31} to $2^{31}-1$

→ long → 8 bytes (64 bits)
Range :- -2^{63} to $2^{63}-1$

☆ Default value for byte, Short, int, long = 0

Scanned with CamScanner

⇒ Wrapper class

int ⇒ Integer

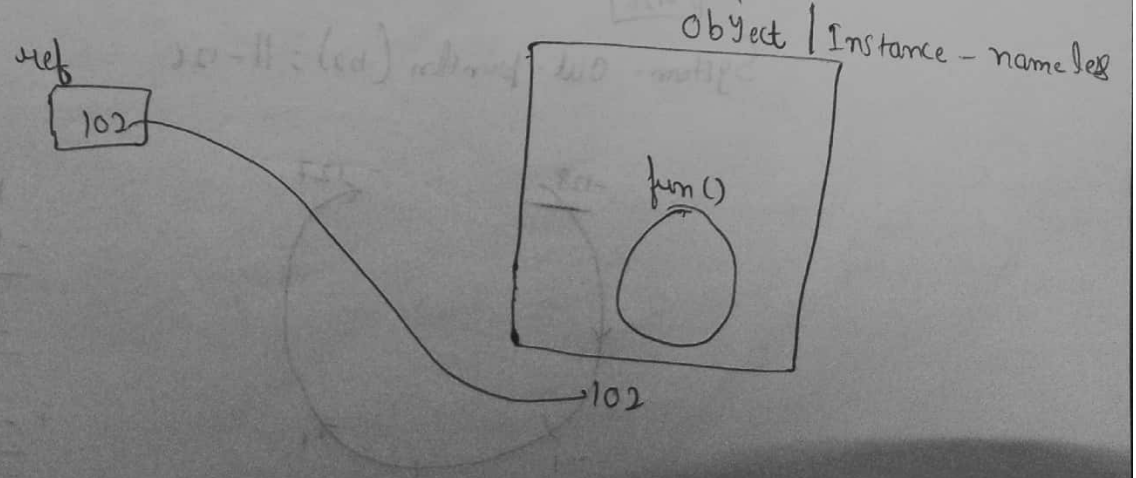
float ⇒ Float

Char ⇒ Character

```
System.out.println(Float.MAX_VALUE);
```

Class | Static variable

```
Class EXP
{
    public Static Void main (Strings args[])
    {
        EXP ref; // Decleration of ref variable
        ref = new EXP(); Creation of object dynamically
        ref.fun();
    }
    Void fun () // Instance method
    {
        System.out.println("Hello");
    }
}
```



Scanned with CamScanner

☆ Type Casting :-

byte :- -128 to 127

```
byte b1 = 50;
byte b2 = 50*2;
```

→ Constant

→ Constant

```
int x=2;

Short s1 = x; // Invaid (Error)
```

[ ] → 4 bytes

2 byte

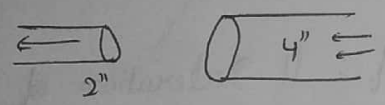
// Explicit type Casting

```
Short s1 = (short) x;
```

→ Explicit Type Casting

Called "Narrowing"

Narrowing can be Acheived Using Explicit type Casting




```
long l1 = x; // widening (auto)
```

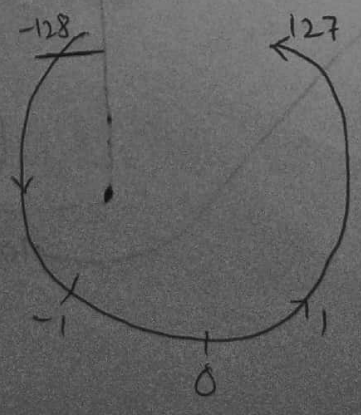
Its safe.

→ int (4 bytes wale)

```
byte b3 = 130; // Error (Found int reduced byte)
byte b3 = (byte)130;
```

[-126]

```
System.out.println(b3); // -126
```




```
  130
 -127
 ----
    3
   -1
 ----
    2
+ -128
 ----
 -126
```

Scanned with CamScanner

Imp #

```
byte b5 = 50;
byte b6 = 2 * b5 ; // Error
System.out.println(b6); // 100 found byte reduced int
```

Implicitly

```
byte b6 = (int)2 * (int) b5 ; // Error
```

[100]$_4$ bytes

Solution :- `byte b6 = (byte) (2 * b5) ;`

[100]$_1$ [100]$_4$ Bytes

Imp #

```
byte b7 = 2 , b8 = 3;
b7 += b8; // it will run
b7 = b7+b8 ; // Error
System.out.println(b7); // 5
```

Implicitly `b7 = (byte) (b7+b8);`

Scanned with CamScanner

★ **Input System** :- we will do this using multiple classes

```
import java.util.Scanner;

class InputProcess ——→ Compile → InputProcess.class → main
                                 Product.class
                                 → Error (if we run this)
                                 Java InputProcess
{
    public static void main(String args[])
    {
        Scanner cin = new Scanner(System.in);
        System.out.println("Enter the price :");
        float price = cin.nextFloat(); // Enter
        cin.nextLine(); // eating the Enter
        System.out.println("Enter product name :");
        String item = cin.nextLine();
        System.out.println("Enter the Quantity :");
        int Qty = cin.nextLine(); // Enter
        System.out.println("your Product " + item);
        Product ref = new Product();
        float bill = ref.getbill(Price, Qty);
        System.out.println("Bill = " + bill);
    }
}

class Product
{
    float getbill(float Price, int Qty)
    {
        return Price * Qty;
    }
}
```

Scanned with CamScanner

Home Work

Q- Write a Program for volume of Cylinder and Cone using Classes and method with Arguament and Return.

A-

```
import [redacted]java.util.Scanner;

class Volume
{
    public static void main (String args[])
    {
        Scanner Cin = new Scanner (System.in);
        System.out.println (" Enter Radius of Cone:");
        float rCO = Cin.next Float ();
        System.out.println (" Enter Radius of Cylinder:");
        float rCy = Cin.next Float ();
        System.out.println (" Enter Height of Cone:");
        float hCO = Cin.next Float ();
        System.out.println (" Enter Height of Cylinder:");
        float hCy = Cin.next Float ();
        Cylinder Cref = new Cylinder();
        Cone Cref1 = new Cone ();
        float VCy = Cref.Area [redacted] (rCy, hCy);
        float VCO = Cref1.Area [redacted] (rCO, hCO);
        System.out.println (" Volume of Cylinder = " + VCy);
        System.out.println (" Volume of Cone = " + VCO);
    }
```

Scanned with CamScanner

```
class Cylinder
{
    float Area (float r, float h)
    {
        return 3.14f * r * r * h;
    }
}

class Cone
{
    float Area (float r, float h)
    {
        return (3.14f * r * r * h)/3;
    }
}
```

Q- Write a program for Reverse of number.

A-

```
import java.util.Scanner;

Class Reverse
{
    public Static void main (String[] args [])
    {
        Scanner Cin = new Scanner (System.in);
        System.out.println (" Enter the number : ");
        int num = Cin.nextInt ();
        NumRev ref = new NumRev();
        int RN = ref.Result (num);
        System.out.println (" Reversed number is " +RN);
    }
}
```

Scanned with CamScanner

```
Class NumRev
{
    int Result (int num)
    {       int r, rev = 0;
            While (num != 0)
            {
                [redacted]

                r = num % 10;
                rev = rev * 10 + r;
                num = num/10;
            }
            return [redacted] rev;
    }
}
```

Scanned with CamScanner

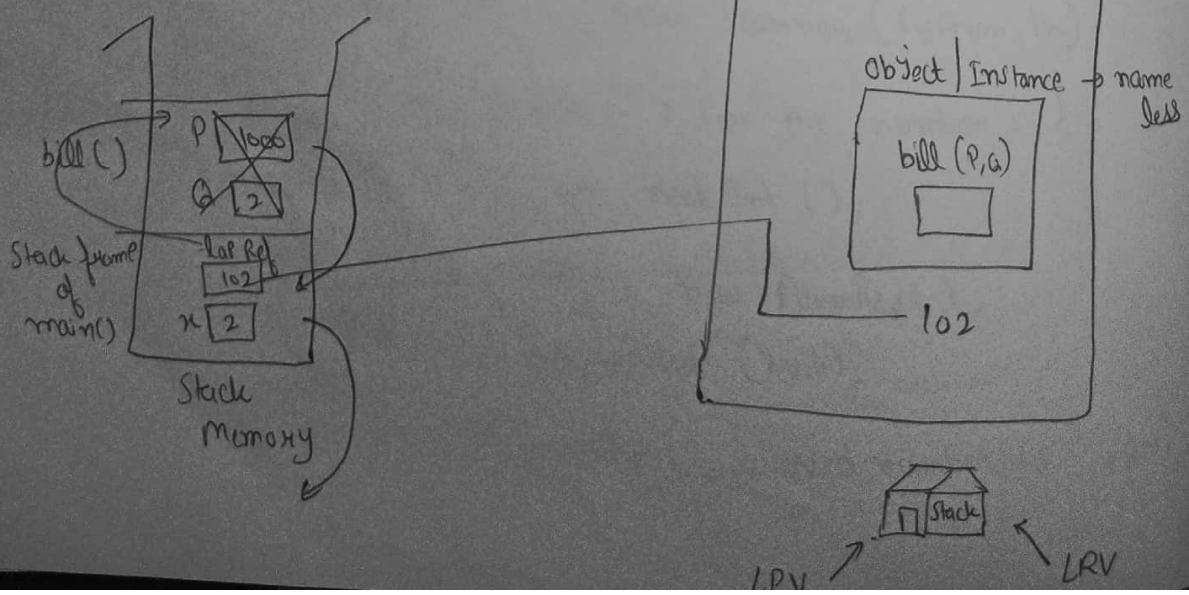
☆ LPV and LRV :- Local Primitive Variables
Local Reference Variables

```
Class LPVLRV
{
    public static void main (String args[])
    {
        int x = 2; // Local Primitive variable (LPV)
        Product lap Ref; // Local Reference variable (LRV)
        lap Ref = new Product (); // creation of object
        Ref. bill (1000, 2);
        // Its frame will be created with Local Variables
    }
}
Class Product
{
    Void bill (int P, int Q)   // Instance method
    {
        System. out. Println (P * Q); // 2000
    }
}
```

Out of main (ie destroyed from Stack)

LPV

Now Its frame will destroyed from Stack



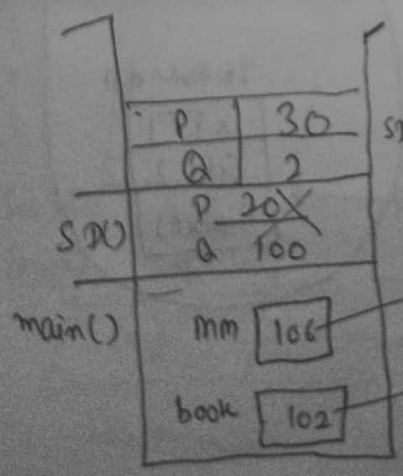
Scanned with CamScanner

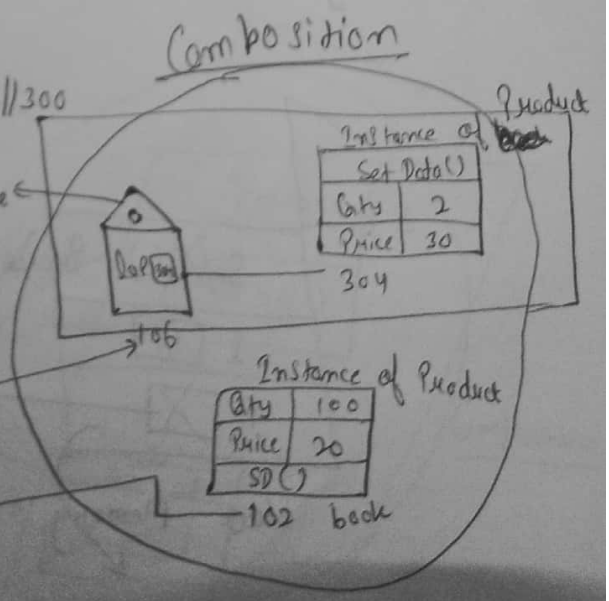
☆ Composition :- IRV, IPV :-

- Instance Primitive Variables
- Instance Reference Variables

```
class Product
{
    int Qty, Price ; || Instance Primitive Variables (IPV) with default value zero(0)
    Void Set Data (int Q, int P)
        {
            Qty = Q ;
            Price = P ;
        }
}

Class MALL
{
    Product lap = new Product () ;   → Instance Reference Variable (IRV) ⇒ Composition
}

Class CUSTOMER
{
    public static void main (String args[])
    {
        Product book = new Product () ;
        book . Set Data (100, 20) ;
        System . out . println (book . price * book . Qty) ; || 2000
        || - - - - - - - - - - - - - - - - - -
        MALL mm = new MALL() ;
        System . out . println (mm . lap . Qty) ; || 0
        mm . lap . Set Data (10, 30) ;
        S.O.P( mm . lap . Qty * mm . lap . Price) ; || 300
    }
}
```

This line demonstrates System.out.println Hence this shows Java based on Composition





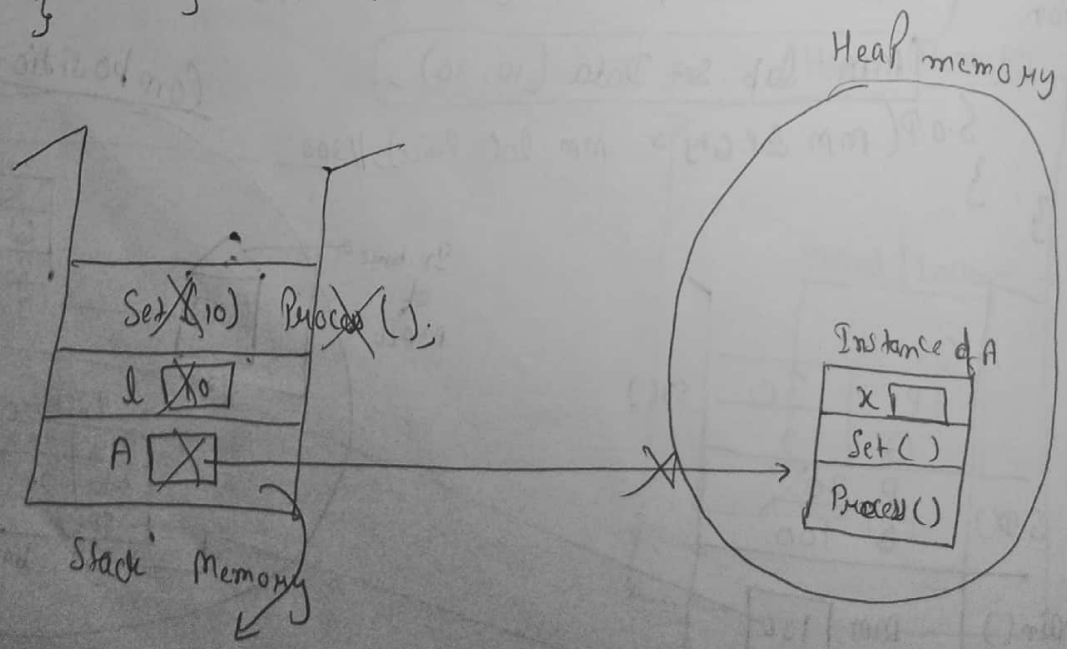
Scanned with CamScanner

Home Work

Q- Find Local Primitive Variable and Local Instance Variable and Draw Stack & Heap Memory.

A-

```
Class A
{ int x; // Instance Primitive Variable.
  void set (int y) → LPV
  {
    x = y;
  }
  void Process ()
  {
    int z = x * x;
    System.out.println (z);
  }
  public static void main (String args[])
  {
    A ref = new A ();
    int l = 10; // LPV
    ref.set (l);
    ref.Process ();
  }
}
```

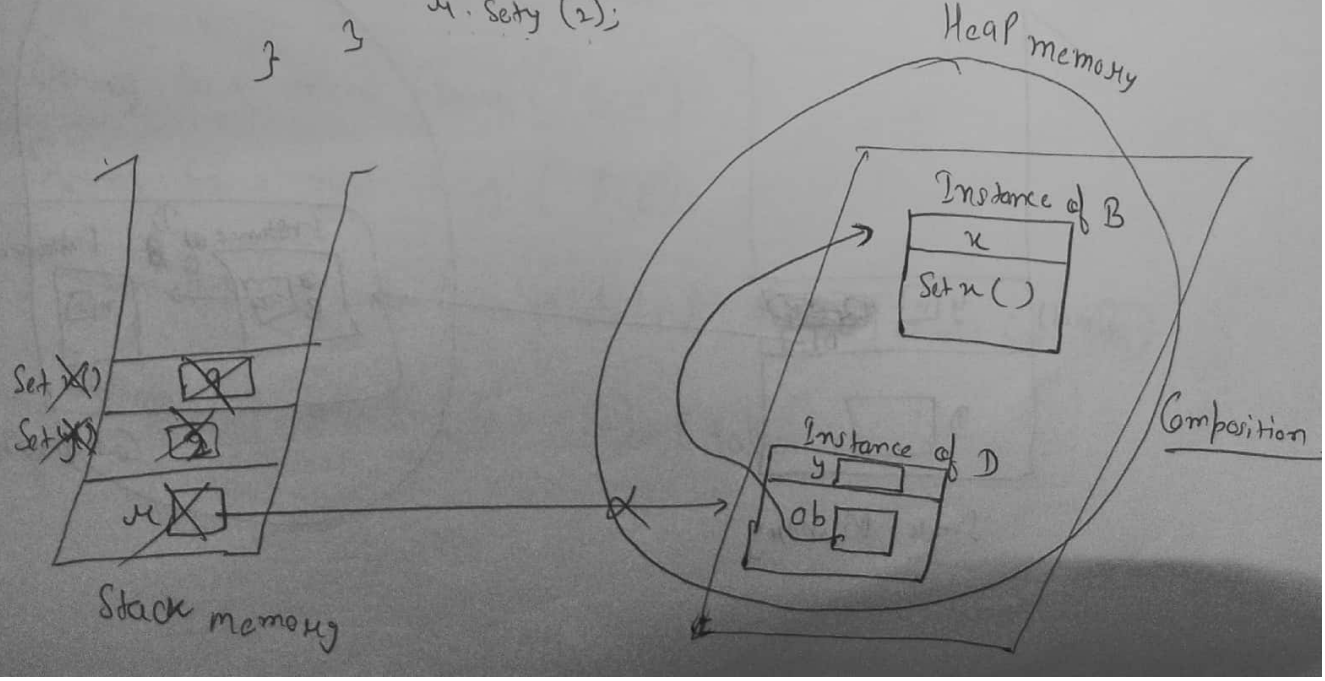


Scanned with CamScanner

Q- Draw Stack and Heap

A -

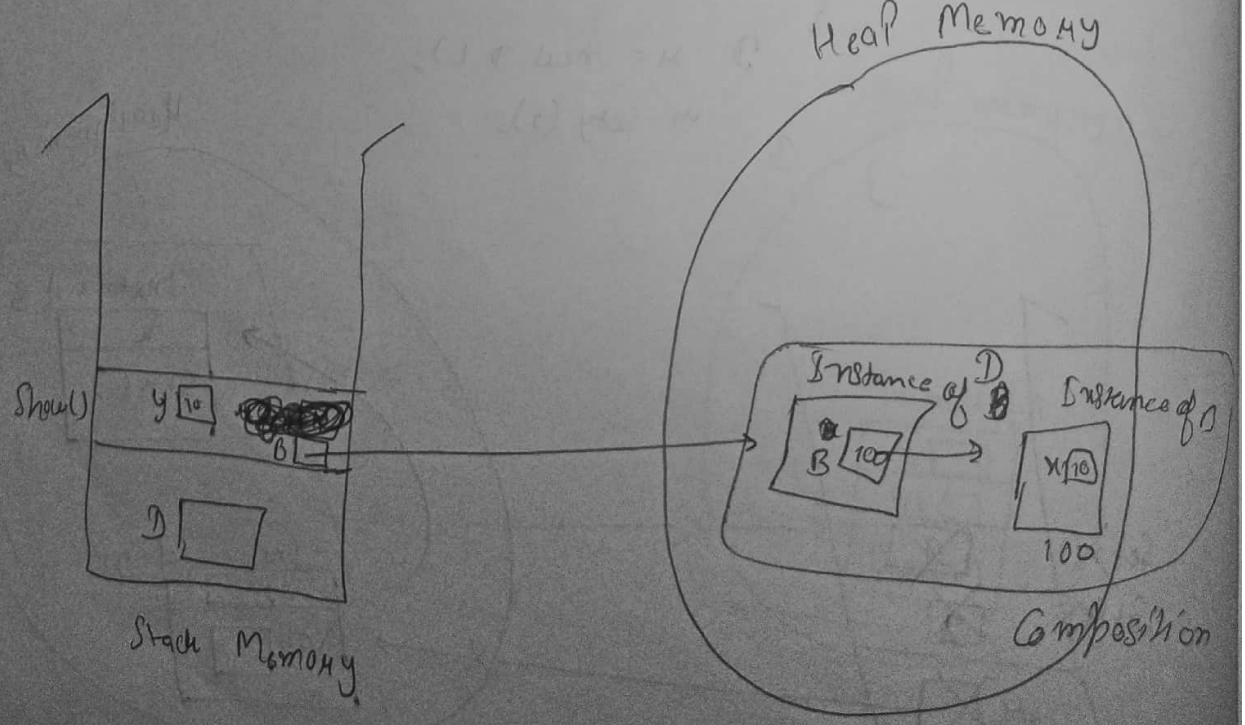
```
Class B
{
    int x ; || Instance Primitive Variable
    Void setx (int x1)
        { x = x1;
        }
}
Class D
{
    int y ; || Instance Primitive variable
    Void sety (int y1)
        {
            y = y1;
            B ob = new B() ; || local Reference Variable
            ob.setx(a);
        }
    System.out.println (y + " " + ob.x);
}
Class Demo
{
    public static void main (String args[])
        {
            D u = new D ();
            u.sety (2);
        }
}
```



Scanned with CamScanner

Q- Class B

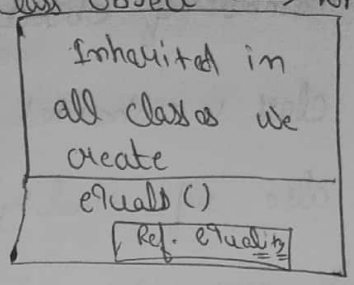
```
{
  int x; // Instance Primitive Variable
}
Class D
{
  int y; // Instance Primitive variable
  B ref = new B(); // Instance Reference Variable
  Void show (int t)
  {
    y = t;
    ref.x = t;
  }
}
Class demo
{
  public Static Void main (String args[])
  {
    D ref2 = new D();
    ref2 = show(10);
    System.out.println (ref2.ref.x); // 10
    System.out.println (ref2.y); // 10
  }
}
```



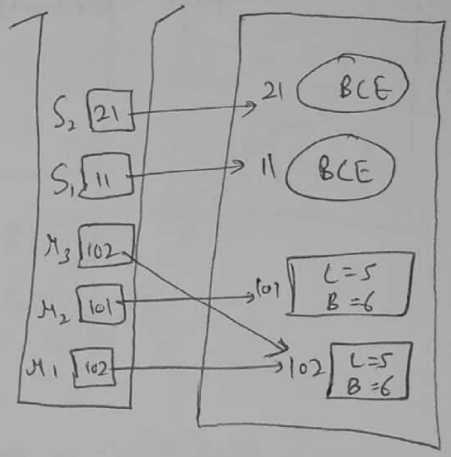
Scanned with CamScanner

Important for interview Purpose :-

Class object → known as Universal class




```
Class RECT
{ int L=5, B=6;
}
Class UC
{
  public static void main(String args[])
  {
    RECT M1 = new RECT();
    RECT M2 = new RECT();
    System.out.println(M1.equals(M2));  // ~~true~~ False
      ×  M1.show();  // Error
    RECT M3 = M1;
    System.out.println(M3.equals(M1));  // True
    //- - - - - - - - - - - -
    System.out.println(M1.L == M3.L);  // True
    System.out.println(M1 == M2);  // False
    //- - - - - - - - - - - -
    String S1 = new String("BCE");
    String S2 = new String("BCE");
    System.out.println(S1.equals(S2));  // true [checks contents]
    System.out.println(S1 == S2);  // False
  }
}
```



Scanned with CamScanner

- Equals () :- It is reference Equality operator.
- It is the member of "object" class called as universal class
- The "object" class is inherited in all classes implicitly.
- == is also ref. equality operator.
- String class over-rides equals () method of "object" class to Compare Contents of both objects.

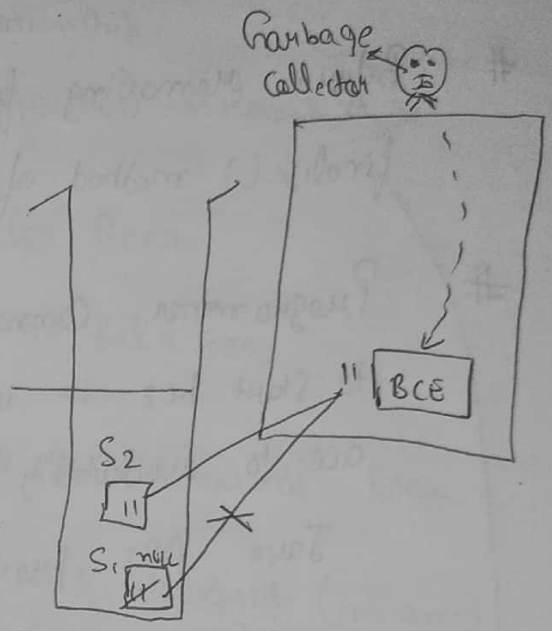
Scanned with CamScanner

★ **Garbage Collection** :- It is the process of getting memory free from those objects which are not ~~in~~ use.
Un-reachable.

```
Class GC
{
  public static void main (String args[])
  {
    String S1 = new String ("BCE");
    String S2 = S1;
```

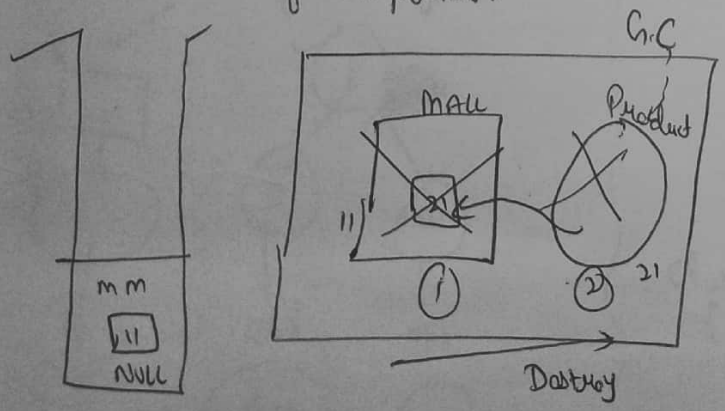
1000 — GC

```
    S1 = NULL; // Now object "BCE" becomes un-reachable and is eligible for garbage collection.
    S2 = NULL; → G.C
  }
}
```



A process run by JVM at random Intervals.

In Case of Composition

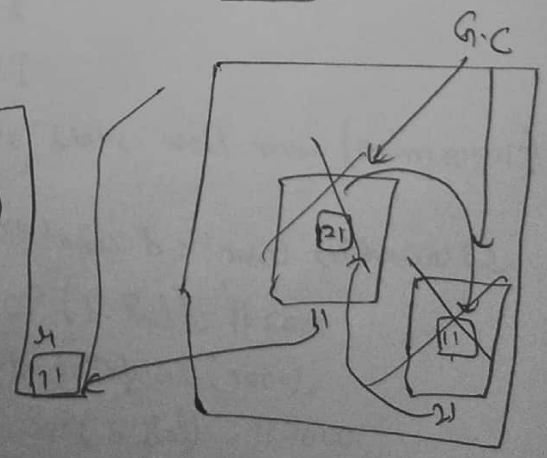


System.out.println(mm.lab.price);

mm = NULL;

→ G.C

Even cyclic Dependency Cannot avoid Garbage Collection if There is no Waris on stack.



Scanned with CamScanner

Garbage Collection is a process run by daemon thread called Garbage Collector. Garbage Collector finds the un-referenced objects and re-claim their memory.

Before removing from memory, Garbage Collector calls finalize() method of that object for clean up operation.

Programmer cannot force the garbage collection process to start bcz it is run by JVM at random intervals acc. to availability of memory in heap.

Java API provide two methods:-

- System.gc()
- Runtime.gc()

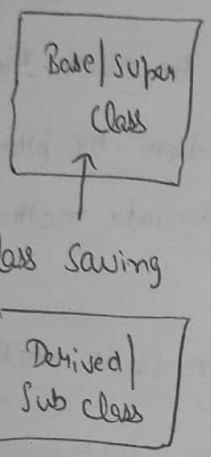
<u>Syntax</u> :- Protected void finalize throws Throwable {}

Scanned with CamScanner

★ **Inheritance** :- The process of creating new classes from existing class is known as Inheritance. It implements "IS A" relationship (Inheritance) where as Composition implements "HAS A" relationship.

(Composition) → ✗ Reusability ✗

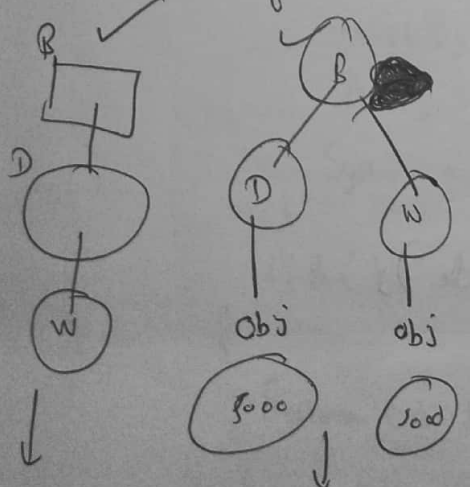
Class Account



"Saving" IS AN "Account"

Base class — Derived class

⇒ re-usability of methods/ Properties of base class



Multi level — Heirarchial

Note :- Multi level Inheritance is not allowed in Java.

```
Class BANK
{
    int Bal = 5000;
};
Class Deposit extends BANK
{
    Void doDeposit (int amt)
    {
        Bal = Bal + amt;
    } SOP (Bal);
};
Class Withdraw extends Deposit
{
    void doDraw (int amt)
    {
        Bal = Bal - amt;
        System.out.println(Bal);
    }
};
Class SBI
{
    public static void main (String args[])
    {
        Withdraw b = new withraw();
        SOP (b.Bal); // 5000
        b.doDeposit (2000);
        SOP(b.Bal) ; // 7000
        S.doDraw (6000);
        SOP (b.Bal) ; // 1000
    }
}
```

Scanned with CamScanner

★ **Method over-riding** :- when base class and derived class duo containing the methods with same name and prototype is known as method-overriding and over-riding is used when the behaviour of base class method doesn't suit derived class.

Note :- • No multiple Inheritance in Java — No Ambiguity Error.

• A final method of super class cannot be over-ridden.

• A Public method cannot be over-ridden by protected or private method but a protected or a private method can be over-ridden by public method.

• A static method cannot be over-ridden as non-static method, however it must qualify as static.

```
class RECT || super class
{
    void Area() || over-ridden method
    {
        System.out.println("Area of Rectangle");
    }
}

class Circle extends RECT
{
    void Area() || over-riding method
    {
        System.out.println("Area of circle");
    }
    void RArea()
    {
        super.area();  super represents base class always
    }
}
```

Scanned with CamScanner

```
class Method Over-riding
{
    public static void main (String args[])
    {
        Circle cir = new circle();
        cir. Area(); // Area of circle
        cir. RArea(); // Area of Rectangle
    }
}
```

<u>Note</u> :- Whenever you create object of Subclass, an object of parent Class is created implicitly. only instance methods and Constructors Can use this " Super" key word.

☆ Constructor and Destructors :-

```
class LIB
{ float Pi ; // Instance Primitive variable
    LIB (float P) // Parametarized Constructor
    {
        this(); // Used to call its Default Constructor.
        Pi=P;
        System.out.println ("Para. Const. of Base class :" + Pi);
    }
    LIB()
    {
    System.out.println ("Default of Base");
    }
}
```

Scanned with CamScanner

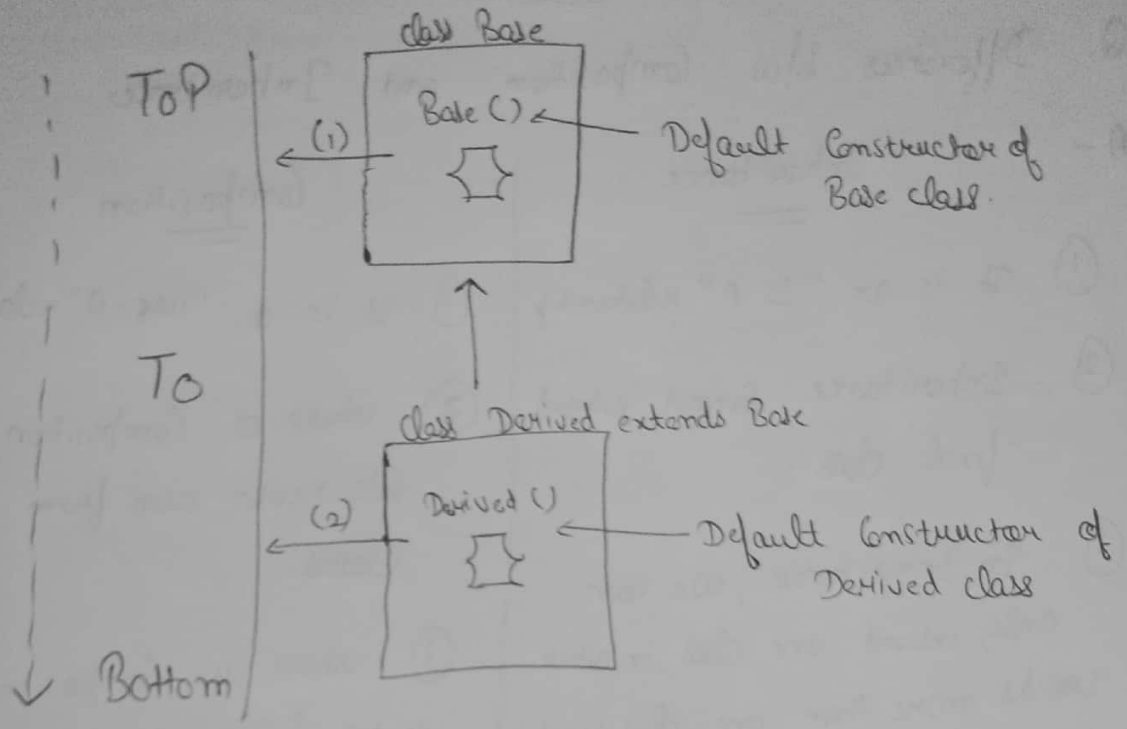
```
Class CIRCLE extends LIB
{
    CIRCLE ()
    {
        super (3.14f); // it calls Parameterized Constructor
        System.out.println ("Const. of Derived class =");
    }
    CIRCLE (int s, float Pi) // Parameterized Const.
    {
        Super (Pi); // it must be the first line in Constructor
        System.out.println ("Area =" + s*s);
    }
}

Class Inh Const
{
    public static void main (String args[])
    {
        CIRCLE ref = new CIRCLE (10, 4.14f);
    }
}
```

Note :- • Always Derived class Constructor called by main but "Always Base Class Constructor is called". (Default).

• When there is no Parameterized Constructor in a class, then a Default Constructor is provided by Compiler implicitly.

Scanned with CamScanner

Implementation of Constructors :-



Imp

Q- why Constructor of Base class Executes first

A- Sometimes derived class Constructors use variables of base class, which are initialized by Constructor of base class and used by Constructor of derived. So to provide accurate data to assist the working of Sub class Constructors, Contructor of Base class executes first.

```
class DATA
{
  float Pi, // Pi=0 → by default not initialised
  DATA ()
  {
    Pi = 3.14f;
  }
}
Class CIRCLE extends DATA
{ float Ar;
  CIRCLE ()
  {
    Ar = Pi * 10 * 10;
    System.out.println (Ar);
  }
}
```

otherwise

```
class DEMO
{
  public static void main (String[] a-)
  {
    CIRCLE C = new CIRCLE();
  }
}
```

Scanned with CamScanner

Home Work

Q- Differances b/w Composition and Inheritance.

A-

Inheritance	Composition
① It is an "IS A" relationship.	① It is a "HAS-A" relationship.
② Inheritance cannot extend final class.	② where as Composition allows Code reuse even from final Classes.
③ In Inheritance, we can only extend one class, in other words more than one class can't be extended as Java do not support multiple inheritance.	③ where as Composition allows to use functionality from Different Classes.
④ In inheritance, we define the class which we are inheriting (super class) and most importantly it cannot be changed at runtime.	④ Whereas in Composition we only define a type which we want to use and which can hold its different implemantation also it can change at runtime. Hence, Composition is much more flexible than inheritance.

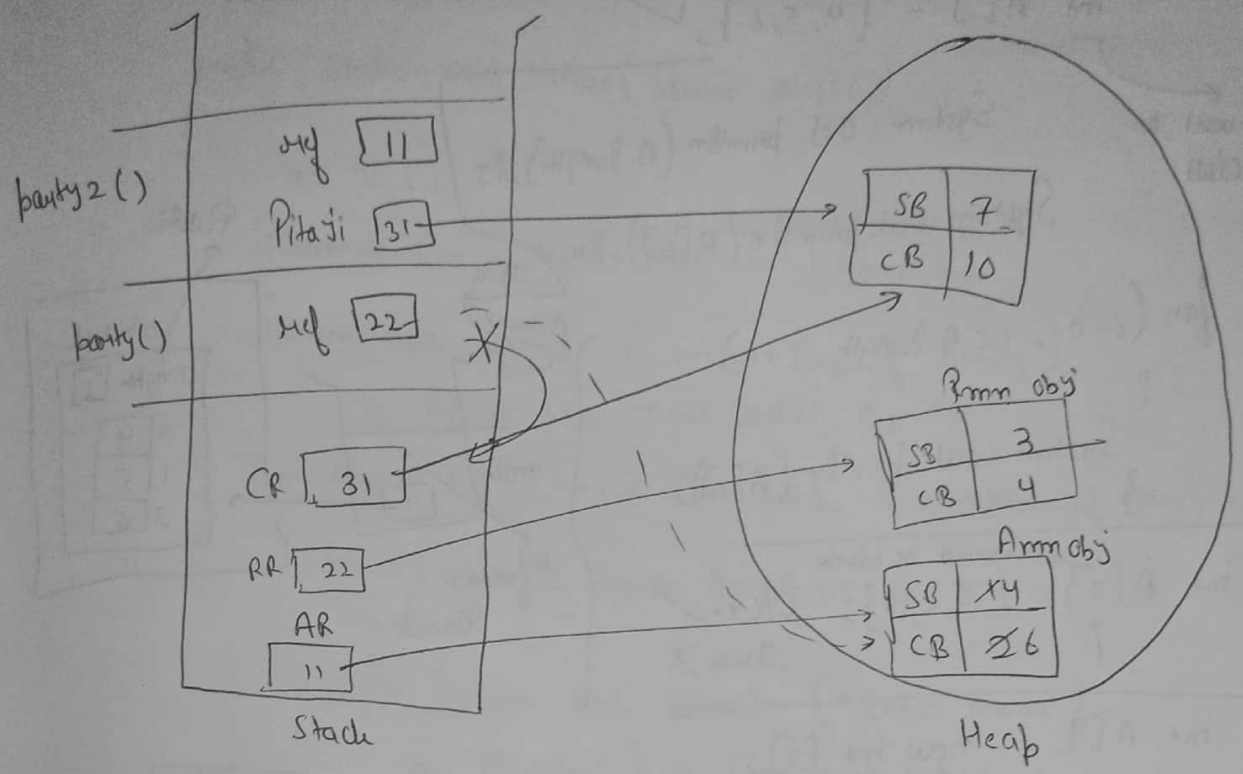
Scanned with CamScanner

★ PORO :- Passing Object Returning Object

```
Class BANK
  {
    int SB, CB ; // IPV

    Void Set Bal (int s, int c)
     {
        SB = S;
        CB = C;
     }
    Void show ()
    {
       System.out.println (" SB = " + SB + " CB = " + CB);
    }
    Void Party (BANK ref)
      {
        SB = SB + ref.SB;
        CB = CB + ref.CB;
      }
    BANK Party2 (BANK aref)
      {
        BANK Pitaji = new BANK();
        Pitaji.SB = SB + aref.SB;
        Pitaji.CB = CB + aref.CB;
        return (Pitaji); // Returning object / Reference
      }
  }
```


Scanned with CamScanner

[Passing Reference Returning Reference] (PRRR)




```
Class PcRo
{
    public static void main (String args[])
    {
        BANK AmnRef = new BANK();
             AmnRef.SetBal (1,2);
        BANK RmnRef = new BANK();
             Rm Ref.SetBal (3,4);
```

Involving object ← `AmnRef.party (RmnRef);` || Passing object | Reference

```
             AmnRef.show(); || 4-6
             RmnRef.show(); 3-4
        BANK chmnRef = RmnRef.party2 (AmnRef);
```

31

```
             chmnRef.show(); || 7-10
    }
}
```

Scanned with CamScanner

☆ **ARRAYS** :- An object

C/C++ → Memory

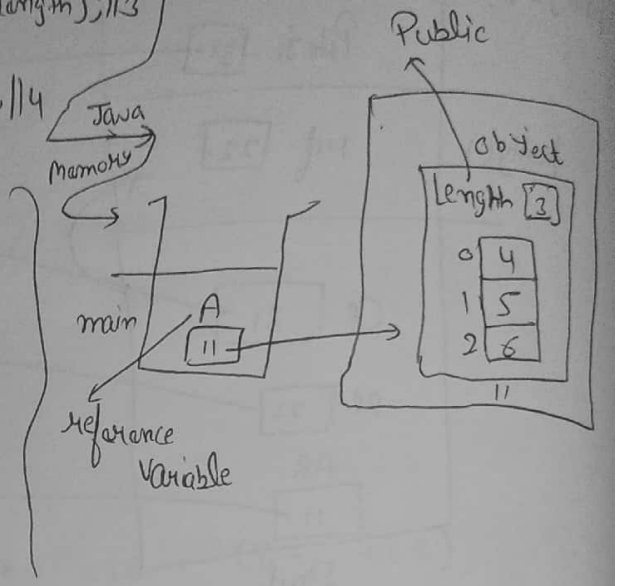
0	1	2
4	5	6

```
int A[] = {4, 5, 6};
```

Convert to class

```
System.out.println(A.length);//3
System.out.println(A[0]);//4
```

Java → Memory




```
for (i=0 ; i<A.length ; i++)
{
    System.out.println(A[i]);
}
```

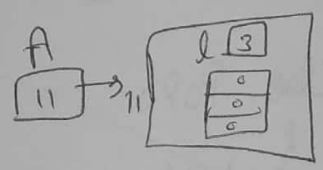
```
int A[3] = {5, 6, 7};
```

it should be blank

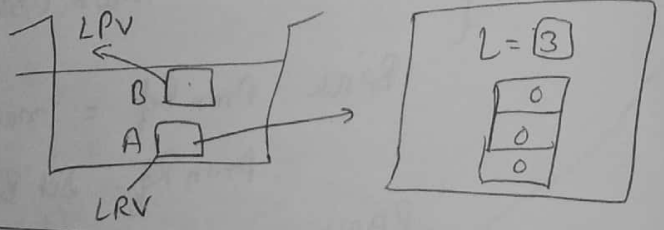
C/C++ ✓
Java ✗

```
int A[] = new int [3];
System.out.println(A[0]);//0
```

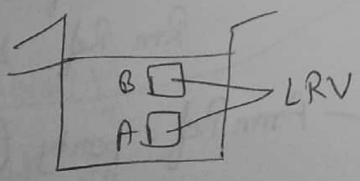
→



```
int A[], B;
A = new int [3];
```



```
int [] A, B;
A = new int [3];
B = new int [4];
```



```
int [] A;
int A[];
```

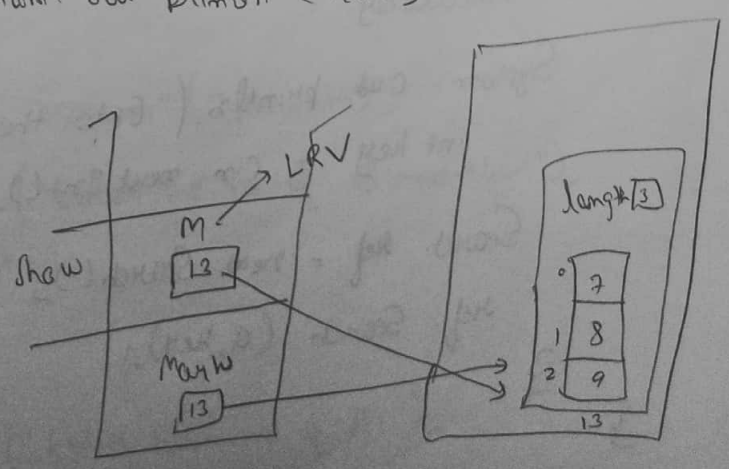
```
int A[];  // Not initialized
X A = {1, 2, 3}; X → not allowed;
A = new int [] {4, 6, 8};
```

Anonymous Array

Scanned with CamScanner

|| Input and output passing Array as Arguement.

```
class ARY
{
    public Static void main(String args[])
    {
        int N, i;
        Scanner Cin = new Scanner (System.in);
        System.out.println ("Enter Size of Array");
                N = Cin.next Int(); //3
        int Marks [] = new Int [N]; || creation of Array obied
        for (i= 0; i< Marks.length; i++)
        {
            System.out.println ("Enter marks");
                Marks[i] = Cin.next Int();
        }
        Show (marks); || Passing Array as Arguement
    }
    Static void show (int [] m)
    { int i;
        for (i=0; i< m.length ; i++)
        {
            System.out.println (m[i]);
        }
    }
}
```



Scanned with CamScanner

Home Work

Q- Write a program for Searching an element by passing Array as Arguement.

A-

```
import Java.Util.Scanner;

class Search_in_Array
{
  public static void main (String.args[])
  {
     int [] a;
     int n, i;
     Scanner Cin = new Scanner (System.in);
     System.out.println ("Enter the No. of Elements");
     n = Cin.nextInt();
     a = new int [n];
     System.out.println ("Enter Elements of Array:");
     // for (i=0; i< a.length ; i++)
     // {
          a[i]= Cin.nextInt();
     // }
     a = new Search().getArray (n, cin); // creating Anonymous
object, one time use, object created, used and becomes unreachable
immediately.
     System.out.println ("Enter the key you want to Search:");
     int key = Cin.nextInt();
     Search ref = new Search();
     ref.Search (a, key);
  }
}
```

Scanned with CamScanner

```
class Search
{
  Void Search (int [] m, int key)
  {
    int i;
    for (i=0; i< m.length ;i++)
    {
      if (m[i] == key)
        System.out.println ("Element is found");
    }
  }
  int [] get Array (int n, Scanner Cin)
  {
    int Ary [] = new int [n];
    for (int i=0 ; i< ary.length ; i++)
    {
      System.out.println ("Enter Element at " +(i+1) + " Position:");
      Ary [i] = Cin.nextInt();
    }
    return Ary;
  }
}
```

Q- Write a program for Sum of two Arrays by passing Array as Argument.

A-

```
import Java.util.Scanner;
Class Sum_of_Two_Arrays
{
  public static void main (String args [])
  {
    int [] a,b;
    int n,i;
    Scanner Cin = new Scanner (System.in);
    System.out.println ("Enter the no. of Elements :");
      n = Cin.nextInt();
    // a = new int [n];
    // b = new int [n];
    System.out.println ("Enter Elements of A:");
      a = getArray (n, cin);
```

Scanned with CamScanner

```
System.out.println("Enter Elements of B:");
  b = getArray(n, cin);
System.out.println("The Sum of two Arrays :");
  int d[] = Sum(a,b);
  for (i=0 ; i< d.length ; i++)
   {
      System.out.println(d[i]);
   }
}

Static int [] getArray (int n, Scanner cin)
   {
      int Ary[] = new int[n];
      for (int i=0 ; i< Ary.length ; i++)
       {
          System.out.println("Enter the element of Array at :"+(i+1)+"
          Ary[i] = cin.nextInt();
       }
      Return Ary;
   }

Static int [] Sum (int [] m, int [] n)
{
   int [] c = new int [m.length];
      for (int i=0 ; i< c.length ; i++)
       {
          c[i] = m[i] + n[i];
       }
   } return c;
}
```

Scanned with CamScanner

☆ 2-D Array :- It is a collection of 1D Arrays

[We cant see 2-D Array in Java, so we can say No 2-D Array in Java].

```
int A[][] = { {1,2,3},
              {4,5,6}
            };
```

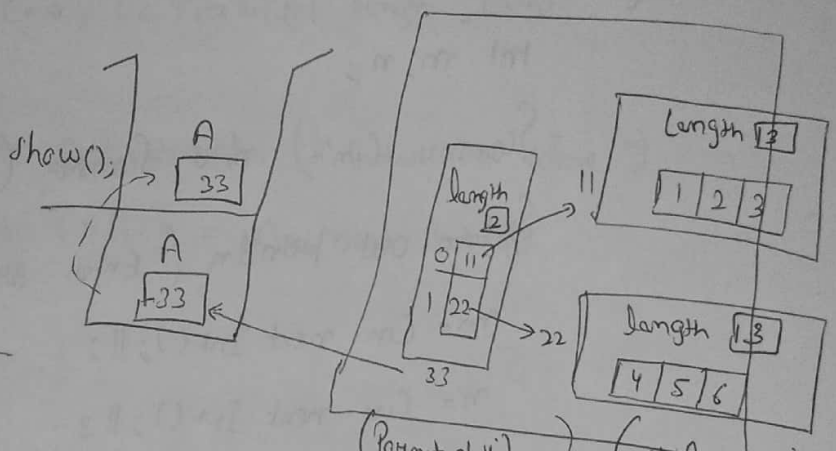
C++ → Memory

A	0	1	2
0	1	2	3
1	4	5	6

// Initialization

```
System.out.println(A[0][0]);//1
```

In Java → Memory →



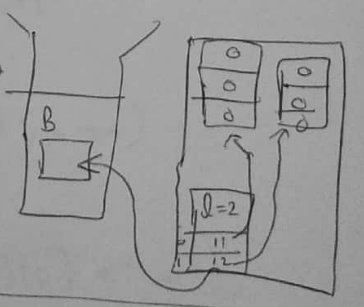
(Parent obj) → array of references (array of pointers)

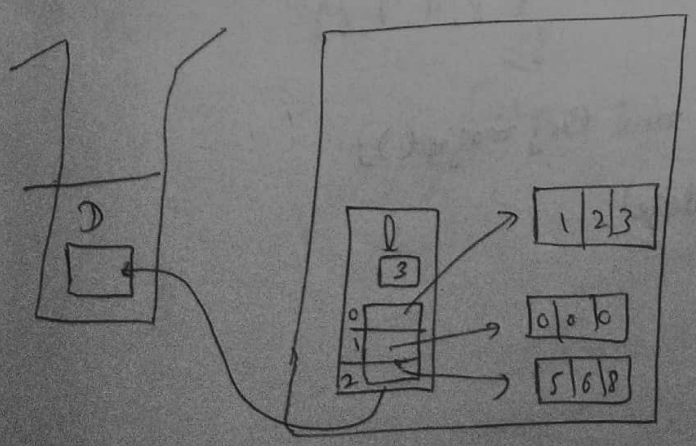
(Child obj) → Array of Integers/values

// Memory allocation

```
int [] B[] = new int [2][3];
```

```
int [][] D;
D = new int [] { {1,2,3},
                 new int [3],
                 {5,6,8}
               };
```






```
System.out.println(D.length);//3
System.out.println(D[0].length);//3
```

Scanned with CamScanner

|| Input for 2-D Array

```
import Java.util.Scanner;
class twod1
{
    public static void main (String args[])
    {
        int m, n;
        Scanner Cin = new Scanner (System.in);
        System.out.println ("Enter order of matrix:");
         m = Cin.nextInt(); || 2
         n = Cin.nextInt(); || 3
        Ary magic ref = new Ary magic();
        int Ary[][] = ref.Create and Return 2D Array (m,n, cin);
          ref.show(Ary);
        /* Jagged Array
          int College [][] = {
                                {5,7,9},
                                new int[2],
                                {8,9,7,6}
                              };
          Ary magic ref = new Ary magic();
            ref.show (College);
    }
}
```

Scanned with CamScanner

```
Class Arymagic
{
    int [][] create and return 2D Array (int m, int n, Scanner Cin)
    {
        int [] Pitaji [] = new int [m] [n];
        for (int r = 0 ; r < Pitaji . length ; r++)
        {
            for (int c=0 ; c< Pitaji[r]. length ; c++)
            {
                System. out. println ("Enter the value:");
                Pitaji [r][c] = Cin. next Int();
            }
        }
        return Pitaji ;
    }

    Void show(int [] ary[])
    {
        // no. of rows = ary. length
        for (int r=0; r < ary. length ; r++)
        {
            for (int c=0 ; c < ary[r]. length ; c++)
            {
                System. out. print (ary [r][c] + "_");
            }
            System. out. println();
        }
    }
}
```

Scanned with CamScanner

Q- Write a function for the input of Jagged Array

```
Enter no. of classes : 3
Enter no. of students : 2 (in 1st)
Enter no. of students : 1 (in 2nd)
Enter no. of students : 3 (in 3rd)
```

5	6	
1		
2	8	7

A -

```
int [] [] getArray (int m, Scanner Cin)
{
    int Ary [] [] = new int [m] [];
    for (int r = 0 ; r < Ary. length ; r++)
    {
        int a [] = Array (Cin);
        Ary [r] = a;
        for (int c = 0 ; c < a.length ; c++)
        {
            Ary [r] [c] = Cin. nextInt();
        }
    }
    return Ary;
}

Static int [] Array (Scanner Cin)
{
    System. out. println ("Enter the no. of Columns Want :");
        int c = Cin. nextInt();
        int P [] = new int [c];
    Return P;
}
```

Scanned with CamScanner

Q- Write a program for Sum of two matrices by passing 2-D Array as Argument.

A-

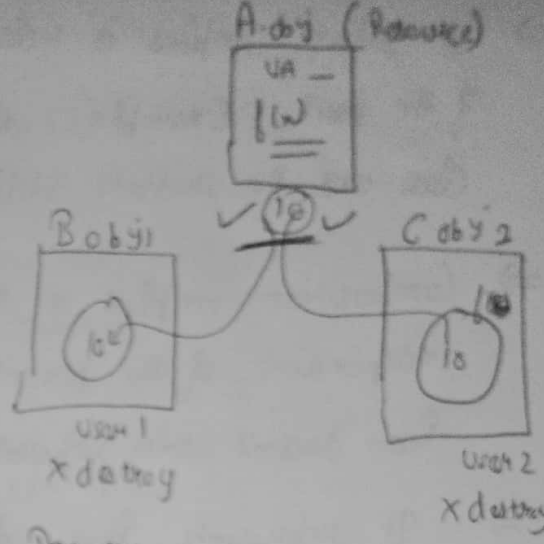
```
Class Sum_of_Two_matrices
{
  public Static void main (String args [])
  {
      int m, n;
      Scanner Cin = new Scanner (System.in);
      System.out.println ("Enter order of Matrix:");
       m = Cin.nextInt();
       n = Cin.nextInt();
      Array ref = new Array();
        int [][] A = new Array().getArray (m, n, Cin);
        int [][] B = new Array().getArray (m, n, Cin);
    int [][]c=ref.Sum (A, B);
          ref.show (c);
  }
}
Class Array
{
    int [][] getArray (int m, int n, Scanner Cin)
    {
        int Ary [][] = new int [m][n];
          for (int r = 0 , r < Ary.length ; r++)
            {  for (int c=0 ; Ary[r].length ; c++)
                {
                  Ary [r][c] = Cin.nextInt ();
                }
            }
            return Ary;
    }
```

Scanned with CamScanner

```
int [][] Sum (int [][]m, int [][]n)
{
    int i,j;
    int [][] Ary = new int [m.length] [m[0].length];
        for (i=0 ; i< m.length; i++)
        {
            for (j=0 ; j< m[0].length ; j++)
            {
                Ary [i] [j]= m[i][j] + n[i][j]
            }
        }
        return Ary;
}

void Show ( int [][]m)
{
    int i,j;
    for(i=0 ; i< M.length; i++)
    {
        for (j=0 ; j< m[i].length ; j++)
        {
            System.out.print (m[i][j]);
        }
        System.out.println();
    }
}
```

Scanned with CamScanner

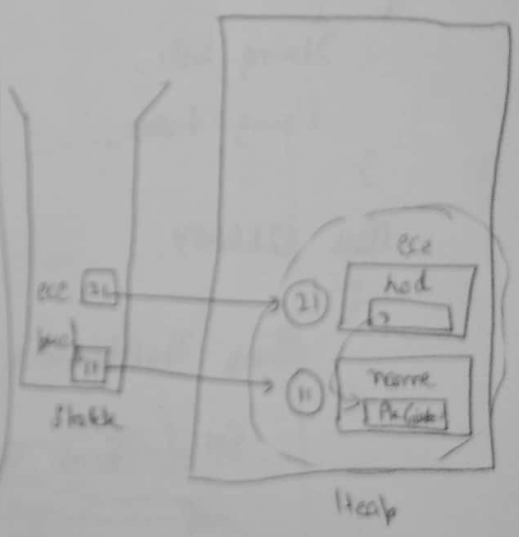
★ **Aggregation** :- → Resource sharing (object)



The resource is not dependent on user object for its existance and value but other two objects depends on it for their values but not for existance

```
class PROFESSOR
{
    String name;
}
class DEPARTMENT
{
    PROFESSOR hod; // IRV
    void SetHod(PROFESSOR prof)
    {
        hod = prof; // Aggregation
    }

class COLLEGE
{
    public static void main(String args[])
    {
        PROFESSOR prof = new PROFESSOR();
            prof.name = "Pk Gorkade";
        DEPARTMENT ece = new DEPARTMENT();
            ece.SetHod(prof); // OR ~~ece.hod = prof;~~
        System.out.println(ece.hod.name);
                // Pk Gorkade
            ece = null;
        System.out.println(prof.name); // Pk Gorkade
    }
}
```

We will do this bcz of future preparation



Scanned with CamScanner

⇒ Aggregation implies a relationship where the child can exist independent of the part. Example : class (parent) and student (child). Delete the class and the students still exist.

⇒ Composition implies a relationship where the child cannot exist independent of the parent. Example : House (parent) and Room (child). Room doesn't exist separate to a House.

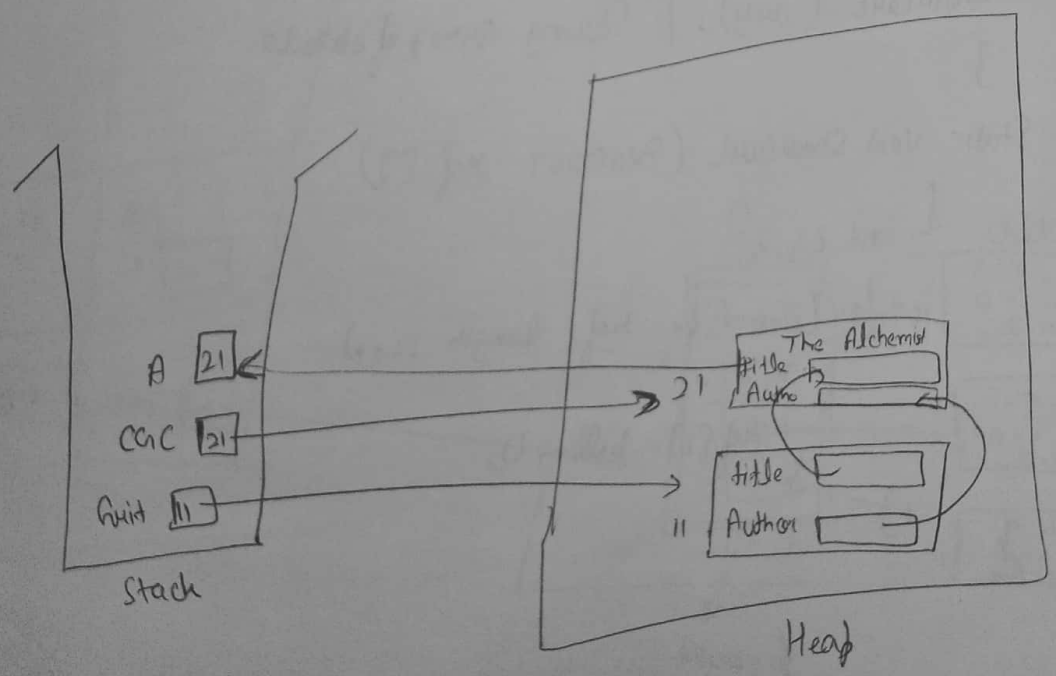
⇒ It represents Part - of relationship.

// Program for Aggregation of our choice

```
Class Book
{
    String title;
    String Author;
}
Class LIBRARY
{
    Book TheAlchemist;
        Set ( Book GRIT)
        {
            The Alechemist.title = GRIT.title;
            The Alchemist. Author = GRIT.Author;
        }
LIBRARY Get ( )
        {
            return The Alchemist;
        }
```

Scanned with CamScanner

```
Class Motivation
{
    public static void main (String args[])
    {
        Book GRIT = new Book();
            GRIT.title = "Positive thinking"
            GRIT.Author = "Prince Wadhwa"

        LIBRARY CGC = new LIBRARY();
                CGC.Set (GRIT);
        LIBRARY A = CGC.Get ();
            System.out.println ( A.title + "_" A.Author);
    }
}
```



Scanned with CamScanner

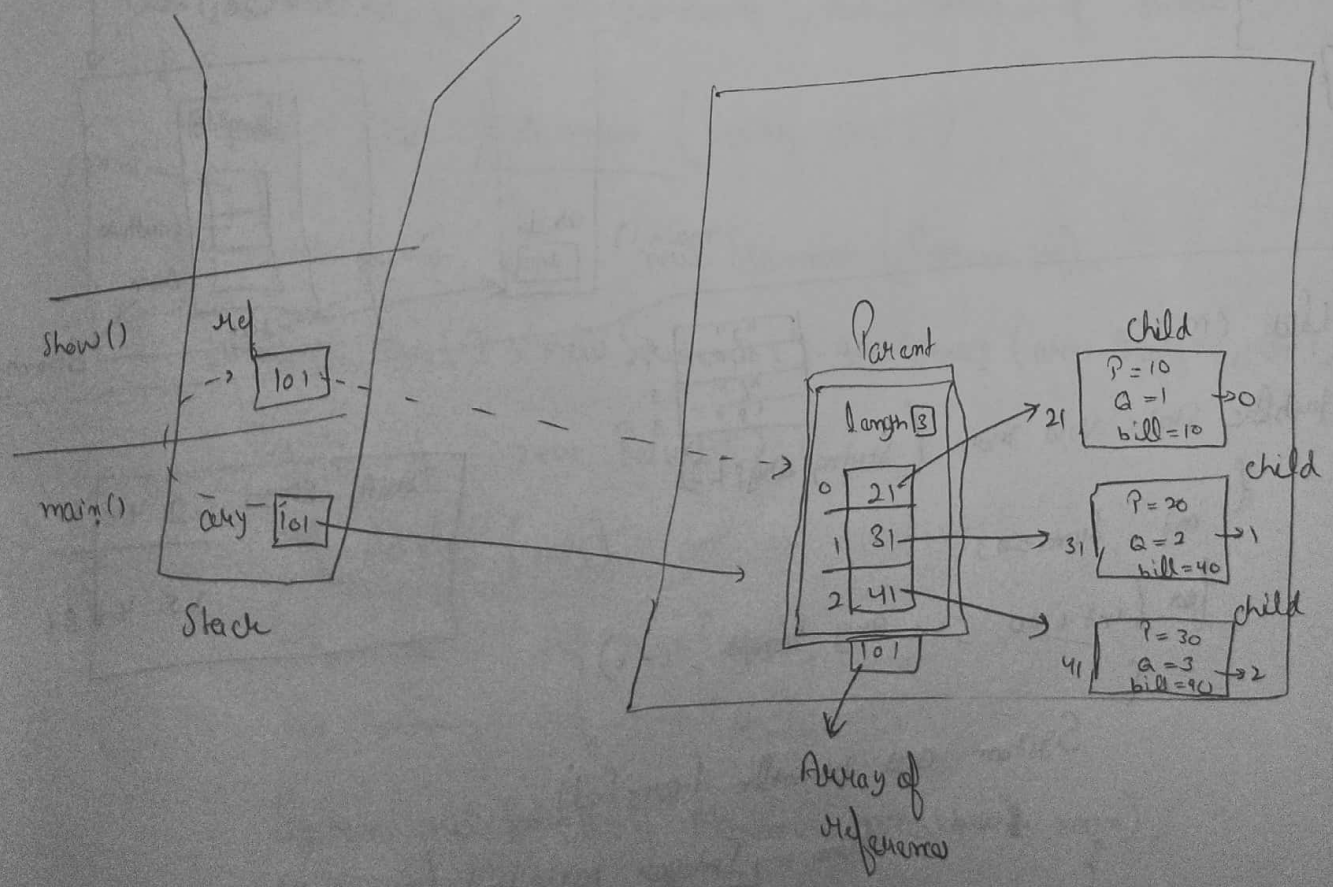
★ **Array of objects** :- Array of objects and Passing it as Argument

```
Class A00
{
  public Static void main ( String args[] )
  {
    int i;
    PRODUCT ary[] = new PRODUCT [3]; // creating array of refarances (Parent ob
    for (i=0 ; i< ary. length ; i++)
    {
      ary[i] = new PRODUCT(); // creation of object (child ob
      ary[i]. Set Data ((i+1)*10, i+1);
      ary [i]. billing ();
    }
    ShowCart (ary); // Passing Array of objects
  }
Static void ShowCart (PRODUCT ref [])
  { int i;
    for (i=0 ; i< ref. length ; i++)
    {
      ref[i]. billing ();
    }
  }
}
```

array of objects

Scanned with CamScanner

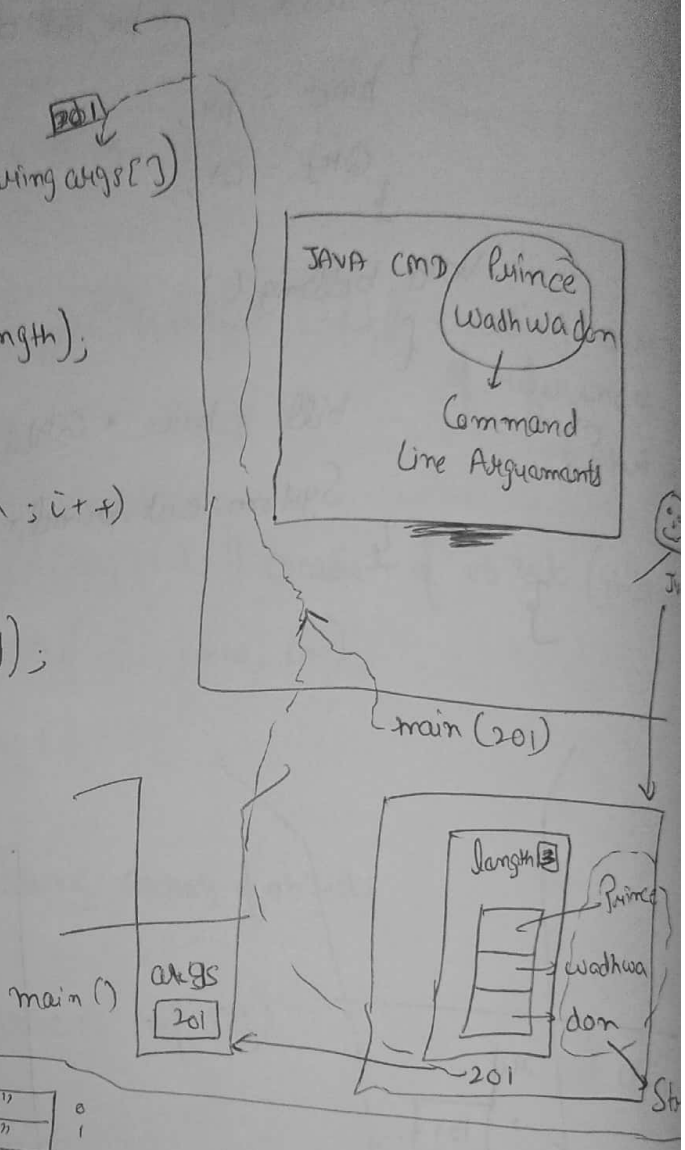
```
class PRODUCT
{
    int price, Qty, bill;
    Void SetData (int pr, int Qt)
    {
        price = pr;
        Qty = Qt;
    }
    Void billing ()
    {
        bill = price * Qty;
        System.out.println ("Billing =" + bill);
    }
}
```



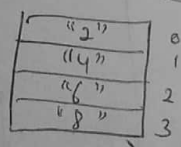
Scanned with CamScanner

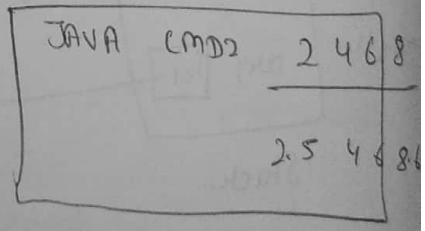
★ Command Line [redacted] Arguamants :-

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



```
class CMD2
{
  public static void main (String args[])
  {
    int sum=0;
    for (int i=0 ; i< ary.length ; i++)
    {
      System.out.println (ary[i]);
      sum = sum + Integer.parseInt (ary[i]);
    }
    System.out.println ("Sum =" + Sum);
  }
}
```

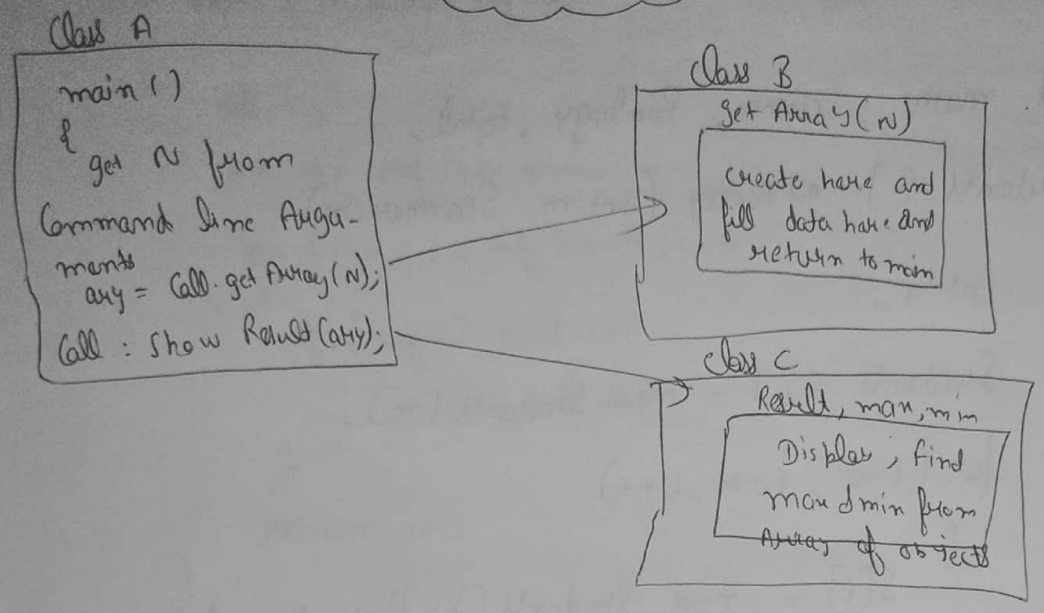




→ Convert str to Integer

Home Work

Q-



A -

```
import Java. Util. Scanner ;

Class Command_line_ Arguemants _ And _ Array _ of _ objects
{
    public Static void main (String args[])
    {
        Scanner Cin = new Scanner (System. in);
    Students ary[] = new Students (). get Array (args. length, Cin);

    Result ref = new Result ();
    ref. display (ary);
    int max = ref. max (ary);
    int min = ref. min (ary);
    System. out. println ("Maximum marks : " + max);
    System. out. println ("Minimum marks : " + min);
    }
}
```

Scanned with CamScanner

```
Class Students
{
  int maths , Science, Biology , total;
  Students [] getArray (int m, Scanner cin)
  {
    int i;
    Students S[] = new Students [m];
    for (i=0 ; i< m ; i++)
    {
      S[i] = new Students (); || creation of object
      S[i].maths = (i+1) * 20;
      S[i].Sceince = (i+1) * 15;
      S[i].biology = (i+1) * 10;
      S[i].total = S[i].maths + S[i].Science + S[i].biology;
    }
    return S;
  }
}

Class Result
{
  Void Display (Students ref []) )
  {
    int i;
    for (i=0 ; i< ref.length ; i++)
    {
      System.out.println ("Total marks of " + (i+1) + "Students :
                                                ref [i].total);
    }
  }
```

Scanned with CamScanner

```
int max (Students ref[])
{
    int i;
    int m= ref[0].total;
    for (i=0 ; i< ref.length ; i++)
    {
        if (ref[i].total > m)
            m = ref[i].total;
    }
    return m;
}

int min( Students ref[])
{
    int i;
    int m = ref[0].length;
    for (i=0 ; i< ref.length ; i++)
    {
        if( ref[i].total < m)
            m= ref[i].total;
    }
    return m;
}
```

Scanned with CamScanner

☆ Final variables, final class and final methods :-

① **Final Method** :- It cannot be over-ridden in derived class

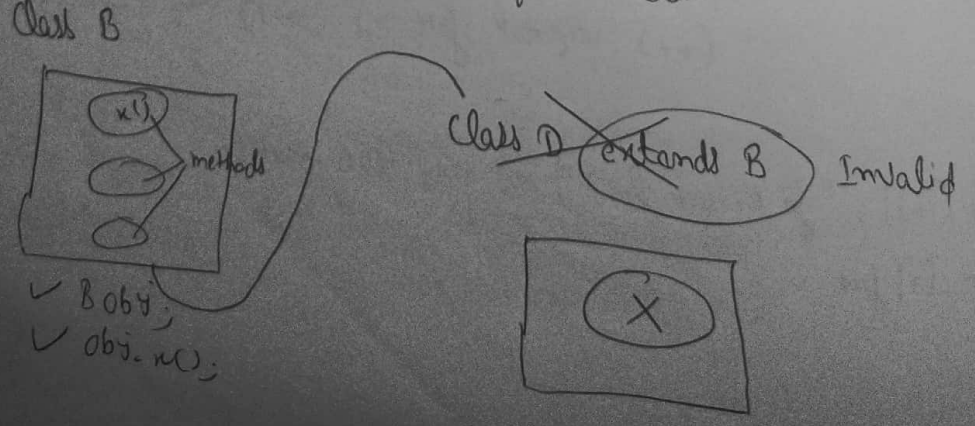
```
class B
{
final void fun ()   // final method
        {
        }
}
Class D extends B
{
    void fun ()   // Invalid
    {
    }
}
```

- A static method can also be declared as final to prevent from over-riding.

② **Final class** :- A final class cannot be inherited.

- It is not necessary for a final class to have final methods
- A final class cannot have abstract methods.

Imp Java.lang.String is a final class.



Scanned with CamScanner

③ Final Variable :-

In C/C++ :- Const int x = 5; that, value of x will Cant be changed.
x = 6; ✗ Invalid

In Java :- final int x = 5; || local Constant Variable
x = 6; || Invalid ✗

Instance final Variable

→ Instance Primitive final variable
→ Instance Reference final variable

(A)

```
Class Maths
{
    final double Pi = 3.14; || Instance Primitive final variable
    Void fun()
    {
        Pi = 4.54; || Invalid ✗
        System.out.println(Pi);
    }
}
```

(B)

```
class Salary
{
    final int Sal; || Final Instance primitive variable   [2000]
    Salary (int s) || Constructor
    {
        Sal = s; || Allowed here
    }
    Void fun ()
    {
        System.out.println(Sal); || 2000
        ✗ Sal = 3000; || Invalid
    }
class FV
{ public static void main (String args[])
    {
        Salary ref = new Salary (2000);
        ref.fun();
    }
}
```

```
Salary (final int s)
{
    S = S - 1000; || Invalid
    Sal = S;
}
```

final local primitive variable

[1000] → if "s" is not final

Imp Note :- We can change the value of final variable, Just only by Constructor other-wise it Can't be possible.

Scanned with CamScanner

Final Reference Variable :-

When a final variable is reference to an object, than this final variable is called reference final variable

```
final MEDIA M;
```

Here, the reference variable can't be re-assigned, but the internal state of the object pointed by it can be changed and this is not re-assigning. This property is called Non-transitivity and Non-transitivity can also applies to Arrays, As Arrays are also objects in Java and Arrays with final key word are known as final Arrays.

```
Class GFg
{
  public static void main (String [] args)
  {
    final MEDIA M = new MEDIA("Geeks");
    System.out.println (m); // Geeks
    m.append ("For Geeks");
    System.out.println (m); // Geeks for Geeks
  }
}
```

Imp Note :- Note the difference b/w C++ Const variables and Java final variables. Const variable in C++ must be assigned a value when declared. For final variables in Java, it is not nessesary. A final variable can be assigned value later, but only once.

Scanned with CamScanner

☆ Abstract Method and Abstract class :-

Opposite to final Method
Must be over-ridden

An abstract class can't be instantiated and → object can't be formed.
Can be used for inheritance only.

```
abstract Class shape   || Abstract class
  {
    abstract Void Area (int s); || Abstract Method (over-ridden)
                                || Can't have body(== Pure virtual type)
  }

  Class RECT extends shape
    {
      Void Area (int s) || over-riding Method
        {
          System.out.println("Area of Rect:" + s*s);
        }
      Void line () || Its own
        {
          System.out.println("=======");
        }
    }
  Class ACAM
  {
    public static void main (String args[])
    {
      ✗ Shape ref = new Shape (); || Invalid
        Shape ref ; || Valid ✓
        RECT rct = new RECT(); ✓
          rct.Area (10); || 100
          rct.line (); || =======
```

Base/Parent class

```
        Shape ref = new RECT();
          ref.Area (5); || 25
        ✗ ref.line(); || Invalid
        ((RECT) ref).line(); || ======
```

Polymorphism → Calling over-riding method of derived class using object of base class.

Down Casting

OR

```
RECT Sref = (RECT) ref;
Sref.line();
```

```
    }
  }
```

Scanned with CamScanner

- A class without any abstract method is called Concrete class. Concrete just means a non-Abstract class.

A class without any abstract method can also be declared using keyword abstract. It means that an abstract class does not necessarily have abstract methods.

- A class derived from Concrete class may overide any, none or all methods but if it is derived from abstract class, then it must over-ride the abstract methods, otherwise it will also become an abstract class.

Scanned with CamScanner

★ **Interface** :- It is the first Cousin of class.

> **Note** :- In Concrete and abstract Class, the Variables are not public nor static nor final by default.

```
~~class~~ RECT
{
    Void Area()
    {
    }
}
```

```
interface SHAPE → A user defined data type
{
    Void Area (float r); // are "Public" and "abstract" by default

    float Pi = 3.14f; // are Public, Static and final by default
}
```

Imp

> # While over-riding its Accessibility Specifier must be >= the Access specifiers of over-hidden method.

```
Class CIRCLE implements SHAPE   // Inheritance
{
    float ar;
    public Void area (float r)   // over-riding method
    {
        ar = Pi * r * r;
        System.out.println (ar);
    }
    Void fun()
    {
        System.out.println ("=======");
    }
}
```

```
Class Inter
{
    public Static Void main (String args[])
    {
        ~~SHAPE~~ sref = new ~~SHAPE()~~; // Invalid
        ✓ SHAPE sref; // Valid
        sref = new CIRCLE (); // Polymorphism
        sref.area (10); // 314 ✓
        sref.fun(); // Invalid
        ((CIRCLE) sref).fun(); // =======
        System.out.println (SHAPE.Pi); // 3.14
    }
}
```

Scanned with CamScanner

interface A

fun1();
fun();

interface B

fun2();
fun();

interface C extends A, B

fun1();
fun();
fun2();

No Ambiguity

Class all implements C

fun1()
fun2()
fun()

Interface

C : implements

Interface

Interface : extends

Class

Class : extends

Class implements interface
Interface extends interface
Class extends class

Scanned with CamScanner

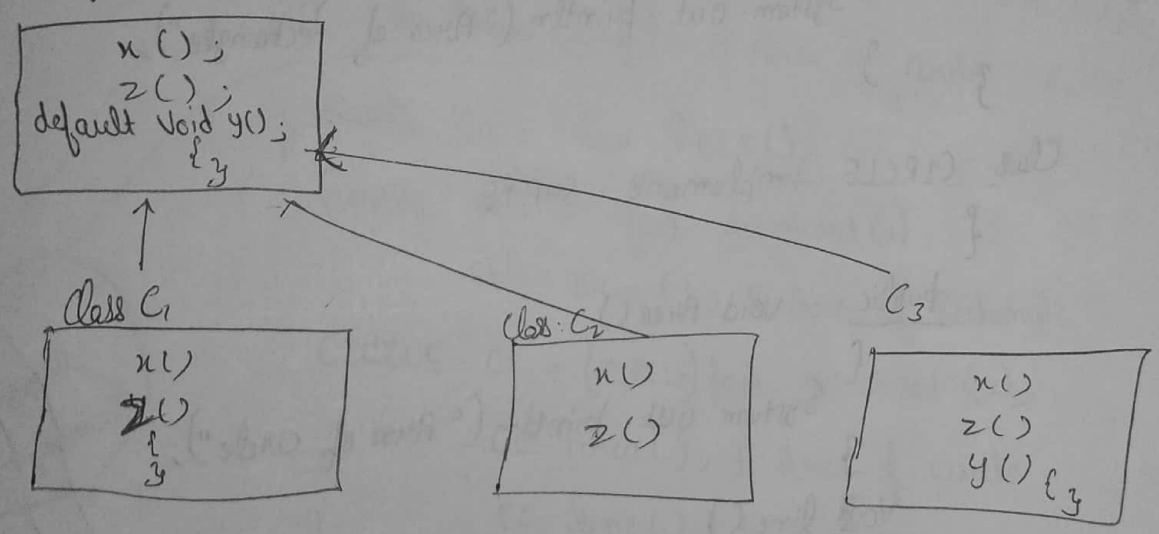
interface A

```
x = 2;
```

interface B

```
x = 5;
```

```
Class AB implements A, B        // A class can implement multiple interfaces
{
    Void show()
    {
        System.out.println (x); // Ambiguity Error
        System.out.println (A.x); //2
        System.out.println (B.x); //5
    }
}
```



This figure shows if a class implements an interface, then it have to over-ride the methods declared in an interface, But if we make it Default then we can give body to that function in interface and there is no Compulsion to over-ride this Default method, and we can also change its body in derived class also (i.e. we can over-ride it too depending upon ourselves totally).

Scanned with CamScanner

★ Polymorphic Arguments and Polymorphic Return :- PAPR

```
interface SHAPE // we can use abstract class as well here
  {
     void Area();
  }

Class RECT implements SHAPE
  {
    public void Area()
        {
          System.out.println("Area of Rectangle");
        }
  }

Class CIRCLE implements SHAPE
  {
    public void Area()
        {
          System.out.println("Area of circle");
        }
    void line()
        {
        }
  }

Class FACTORY
  {
    void doDraw (SHAPE ref)
        {
          ref.Area();
        }
    SHAPE getobject (int what)
        {
          if (what == 1)
              return new RECT();
          else if (what == 2)
              return new CIRCLE();
```

Polymorphic return

UpCasted

Scanned with CamScanner

```
        else
            return null;
        }
    }

class PAPR
{
    public static void main (String args[])
    {
        RECT rref = new RECT();
        CIRCLE cref = new CIRCLE();
        FACTORY fact = new FACTORY();
        fact.doDraw(rref); // Area of Rectangle
        fact.doDraw(cref); // Area of circle
        // SHAPE Sh = new RECT();
        SHAPE shP1 = fact.getObject(1);
            shP1.area(); // Area of Rectangle
        CIRCLE cc = (CIRCLE) fact.getObject(2);
            cc.Area(); // Area of circle
            cc.line(); // ====
    }
}
```

Polymorphic Arguments

Scanned with CamScanner

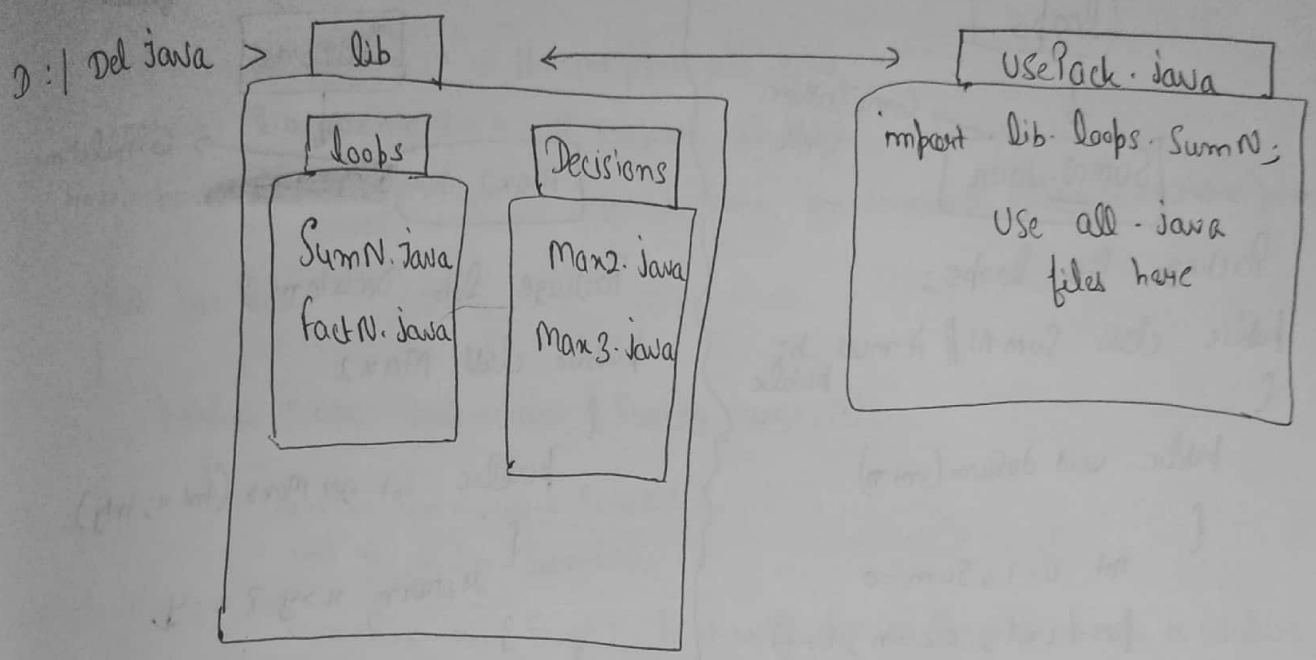
Explation of PAPR :-

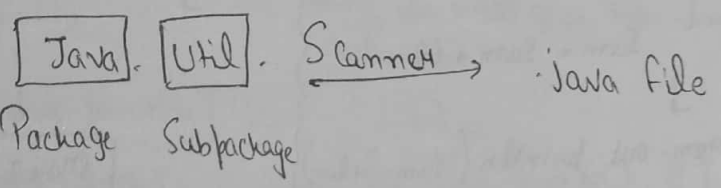
(1) Sabse Pehle humne '2' classes bnayi RECT and CIRCLE Aur dono me void area naam ka method bnaya with body, then FACTORY class bnayi And us mein ek method bnaya do Draw ka, Jo ki ek object receive krega, Aur usi object ke Area method ko Call krega, Hamne Pehle to RECT and CIRCLE ka object bnaya, and Ab agar humko dono ka Area ka Call karwana hai, to '2' do draw function bnane padenge ek mein RECT ka object receive hoga and dusre mein CIRCLE ka object receive hoga but if we have 100 different objects, To kya ham, 100 bar alag function bnayenge? Nahi Toh isliye hamne us function mein receive krne ke liye SHAPE interface ka ref variable lgga diya, SHAPE humne interface bnaya, Jis mein Area() naam ka method bnaya, Jo ki over-ride ho jayega Jab baki classes implament krengi shape ko, To ab jab hum RECT ka object pass krengh, To waha SHAPE ka ref variable receive krega Aur jab Area() method Call hoga to vo ussi object ka hoga, jis class ka pass kiya. (ie. Polymorphism, Derived class RECT ka object pass krke Uska over-riding method Call karna, using ref variable of base class).

(2) Now, If we want to return an object from function, Toh again Data type kya rakhengh, Taki agar RECT ka chahiye toh uska aajaye and agar CIRCLE ka chahiye To uska aajaye. Toh iske liye hamne firse SHAPE ka use kiya, then Jab Suppose humka RECT ka chahiye, To jab vo return hoga, tab vo SHAPE mein upcast ho chuka hoga, Toh usko CIRCLE ke variable mei Store krne ke liye, humne down Casting ki, Taki agar koi method aur hai jaisi void line(), vo bhi hum call kar sake.

Scanned with CamScanner

★ **Packages** :- It is a folder / Directory Containing .java files, which can be reused from other .java files.





Java → Package

Util → Sub Package → it may Contain Compilation unit and Sub package of its own.

Scanner → .java file | Compilation unit | translation unit

only one class can be declared public in a Compilation unit, otherwise Compiler will show error message, If a Compilation unit Contains additional non Public classes, they will be hidden from the outside world but can be used as Support classes to the main public class.

Scanned with CamScanner

Creating Packages

lib → Package

Sub folder → loops → SumN.java → Compilation unit

```
Package lib.loops;
public class SumN // it must be public
{
    public void doSum(int n)
    {
        int i=1, Sum=0
        for (i=1; i<=n; i++)
        {
            Sum = Sum + i;
        }
        System.out.println("Sum=" + Sum)
    }
}
```

↓

Fact N.java → Compilation unit

```
Package lib.loops;
public class FactN
{
    public void factorial(int N) // it also must be public
    {
        int i=1, fact=1;
        for(i=1; i<=n; i++)
        {
            fact = fact * i;
        }
        System.out.println(fact);
    }
}
```

Sub-folder → Decisions → Max2.java [redacted] → Compilation unit

```
Package lib.Decisions;
public class Max2
{
    public int getMax2(int x, int y)
    {
        return x>y ? x : y;
    }
}
```

↓

Max3.java → Compilation unit

```
Package lib.Decisions;
public class Max3
{
    public int getMax3(int x, int y, int z)
    {
        return x>y ? x>z ? x : z : y>z
                                y : z;
    }
}
```

Scanned with CamScanner

Using Packages

```
// import lib.loops.SumN; // Single-type-import Declaration
// import lib.loops.FactN; // Single-type-import Declaration

import lib.loops.*; // Import all files
import lib.decisions.*; // Import all files
import lib.loops.Fun; // Type-import-on demand declaration (higher priority)

Class UsePack
{
    public Static void main (String args[])
    {
        SumN Sn = new SumN();
        int m = Sn.Sum(5);

        Fun f = new Fun(); // it will give ambiguity error to solve
        f.doFun();          this, we will give type-import on demand
                            declaration.

        System.out.println("Sum = " + m);

        lib.loops.Fact fn = new lib.loops.FactN(); // fully-qualified path
            int facto = fn.factorial(5);

        System.out.println("Factorial = " + facto);

        Max2 p = new Max2();
            int max1 = p.getmax2(2,4);

        Max3 n = new Max3();
        int max2 = n.getmax3(2,4,6);

        System.out.println("max1 = " + max1);

        System.out.println("max 2 = " + max2);
    }
}
```

Scanned with CamScanner

Note :- ⇒ [redacted] Single - Static - import Decleration

Using import statement to access the static members of a class/interface from Packages is often called Static import. Using a Single static import decleration, you can avail only one Static member. It means that for multiple Static members, you will need multiple Single - static - import decleration

Syntax :- import Static Packagname . TypeName . Static MemberName ;

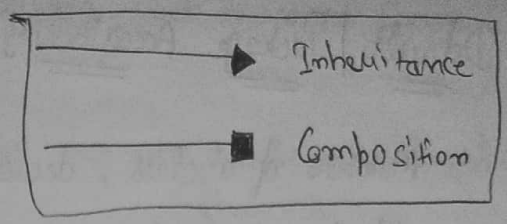
⇒ Static - import - on - Demand Decleration

It avails all Static members of a type to the Compilation unit.

Syntax :- import static PackageName . type Name . * ;

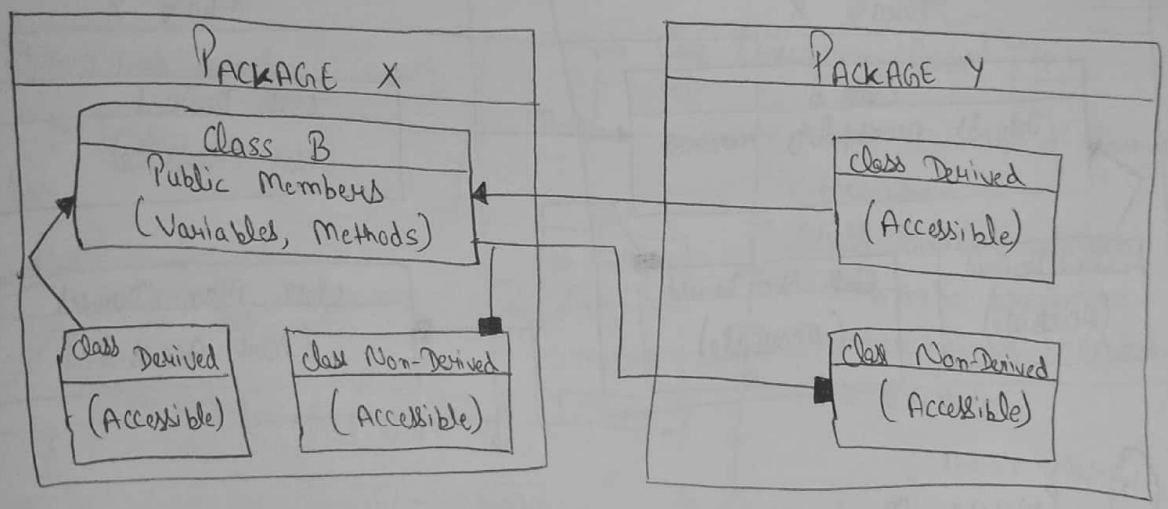
Scanned with CamScanner

☆ Accessibility Specifiers :-



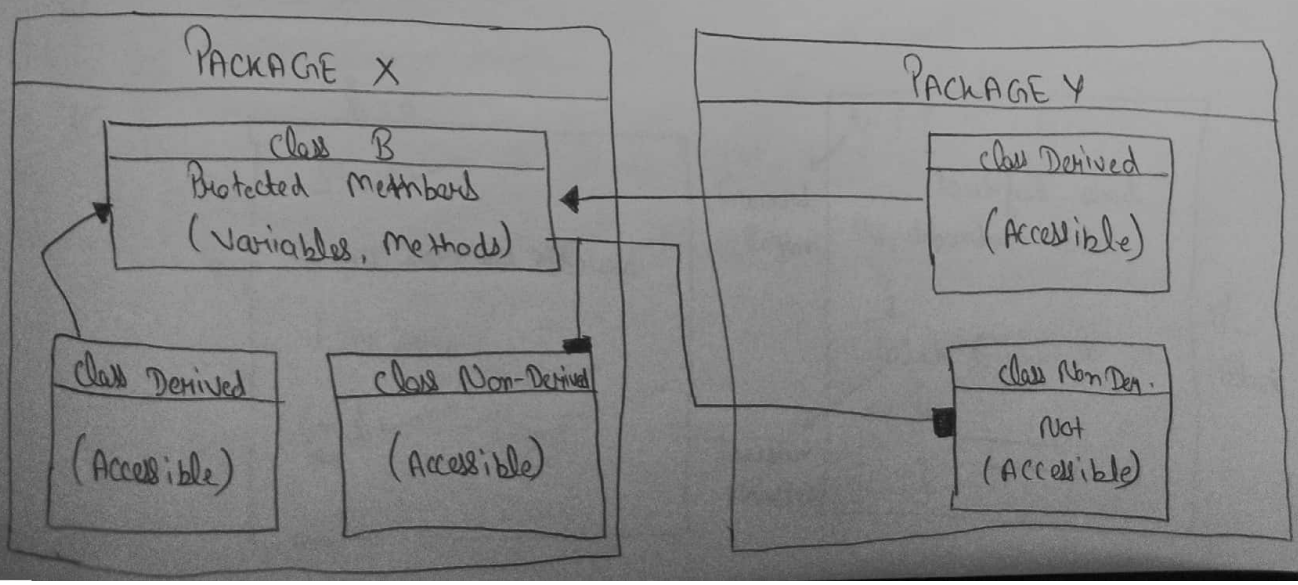
① Public Members :-

- Accessible everywhere whether in same package or another package.
- Accessible in derived and non-derived classes whether of same package or another.



② Protected Members :-

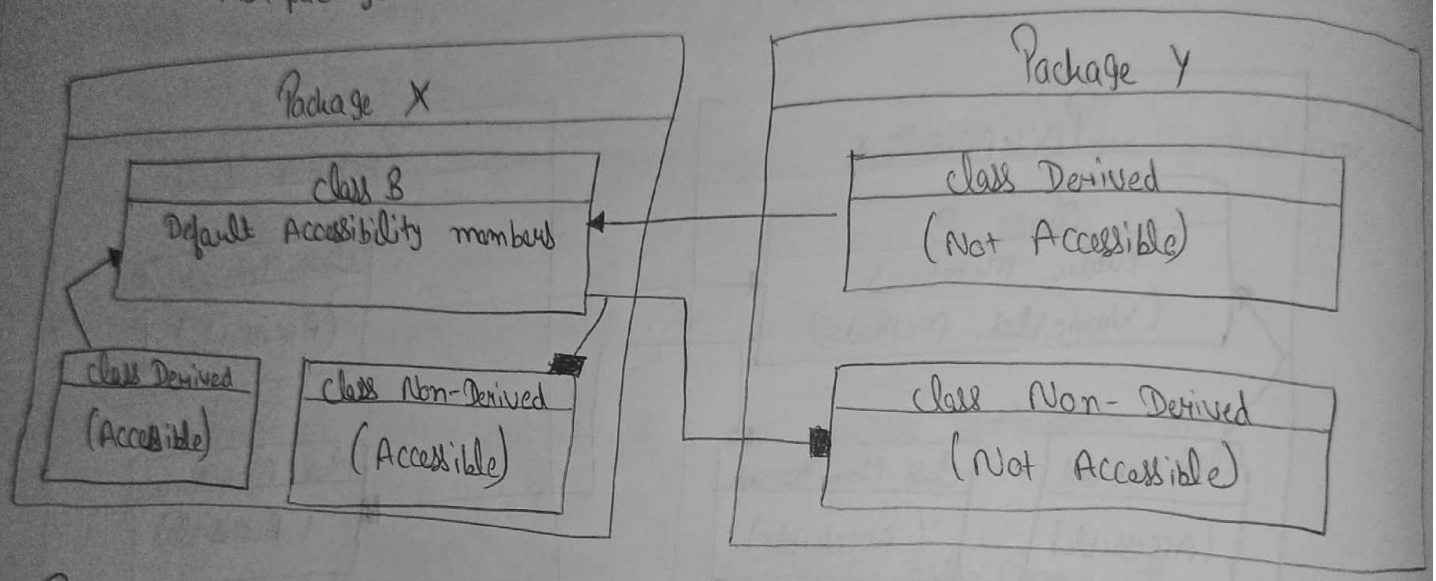
- They are Accessible in all classes in same package, where class is derived or not.
- But for other packages, these are accessible in derived classes only.



Scanned with CamScanner

③ Default / Package Accessibility Members :-

- Members of a class, declared without any accessibility specifier, have default Accessibility.
- They are accessible in derived and non-derived classes of same package only. Even if the class is public, those members are not visible in other packages.



④ Private Members :-

- The private members of a class have more restrictive access as compared to other accessibility specifiers.
- They are accessible only with in own class, Even not Accessible in derived class of same package.

Scanned with CamScanner

Exception Handling :- Dealing | working with Runtime Errors

[JVM] Runtime Errors (Interpretation time)

Avoiding | manage ⟹ to develope reliable Softwares

[Exception is an object] bcz when Exception is thrown, an Exception object is created.

Class: Throwable

Class: Exception

Class: Runtime Exception

Inheritance occurs here

"is A"

```
class EH
{
  public static void main (String args[])
  {
    System.out.println ("Start");
      int l = 0;
    try
      {
        int d = 5/l;
        System.out.println ("div = " + d);
      }
    Catch (ArithmeticException exp)
      {
        System.out.println (" B.S Div by Zero not Allowed");
        System.out.println (exp.getMessage());
      }
    System.out.println ("End");
  }
}
```

Exception thrown (202) → Exit (crash)

202 → Class Lib (Exception Classes)

JVM → Array Index out of Bounds Exception
→ Null Pointer Exception
→ Arithmetic Exception
→ Number format Exception
→ IO Exception
⋮ 1000's more

Object
get Message()
Return Dm
D. Message
/ By zero

202

l = 0

Start ✓

Div by Zero not Allowed

/ By Zero

End ✓

Guarded Region

```
try {
    Doubtful and
  its dependent Code
  }
  Catch (     )
    {
      ____
      ____
    }
```

Exception Handler

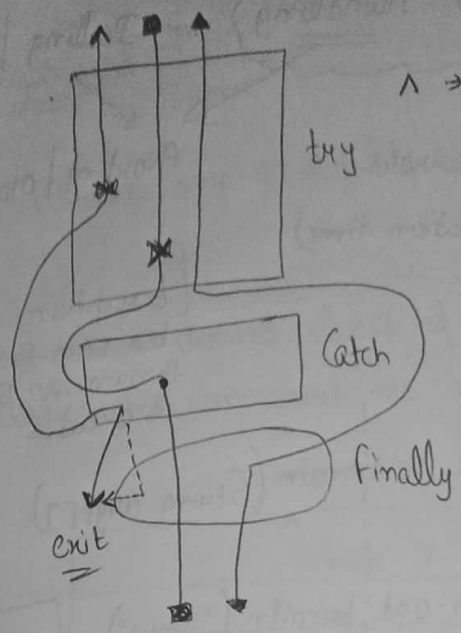
T/C Block

Scanned with CamScanner

■ → Exception thrown but handled.

∧ → Exception thrown and can't handle

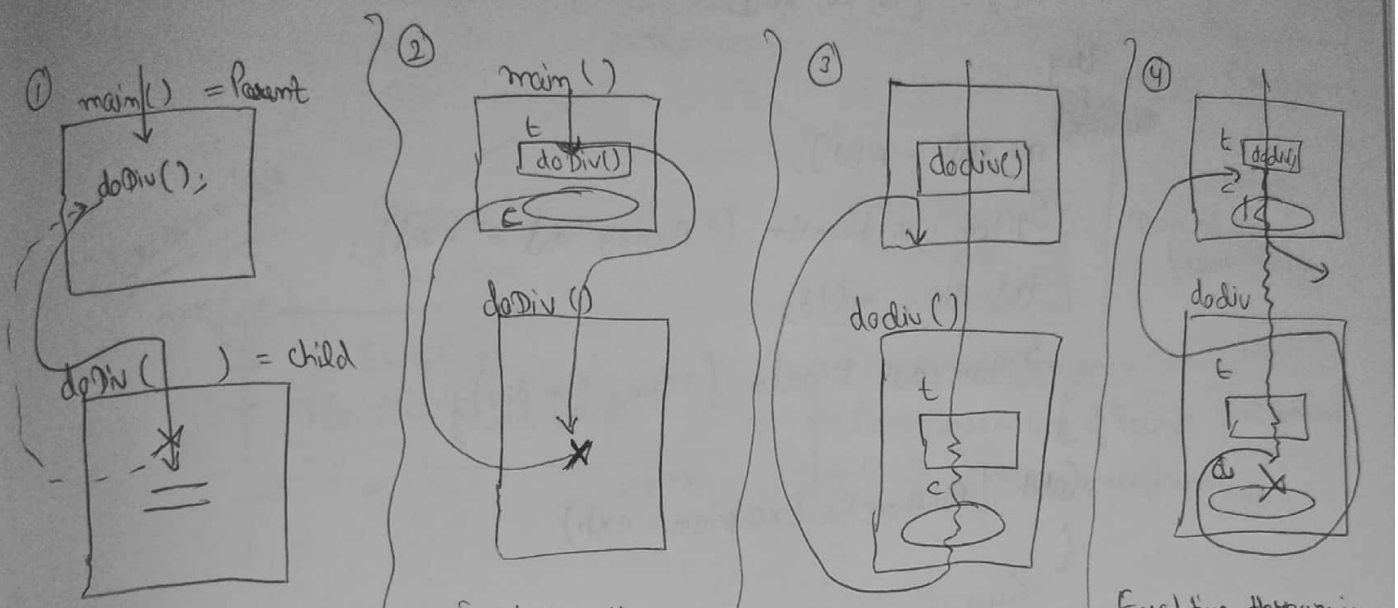
* Multiple Catch :-




```
Class Multi
{
    public Static void main(String args[])
    {
        int A[] = {6, 8, 10};
        System.out.println("Start");

        int l = 2 ; or l = 6

        try {
            int val = A[l];
            System.out.println("Array Val = " + val); // 10
            int dv = val/2;
            System.out.println("Div = " + dv); // 5
            String s = null;
                System.out.println(s.length()); ] Exception X will be X
                                                  unhandled
        }
        Catch (Arithmetic Exception exp)
        {
            System.out.println("Div error");
        }
        Catch (ArrayIndex OutofBounds Exception exp)
        {
            System.out.println("Invalid index");
        }
        Catch (Exception exp)
        {
            System.out.println(exp.getMessage()); // Polymorphism
        }
        finally
        {
            System.out.println("finally Block executed");
        }
    }
}
```

Scanned with CamScanner

⇒ Now there will be four Situation can occur, as we know we always make functions that will called from main, so where to handle the Exceptions.



Exception thrown in do Div but handled [redacted] no where

Exception thrown in do Div but handled in main

Exception thrown in doDiv() but Also handled in do div()

Exception thrown in do div() but not handled by by dodiv() but then handled by main ()

```
Public class MultiCatch
{
  Public Static void main (String args[])
  {
    MultiCatch mc = new MultiCatch();
    try
    {
      mc.div(10);
    }
    Catch (ArrayIndexOutofBoundsException exp)
    {
      System.out.println(" Invalid Index :" + exp.getMessage());
      exp.printStackTrace();
    }
  }
```

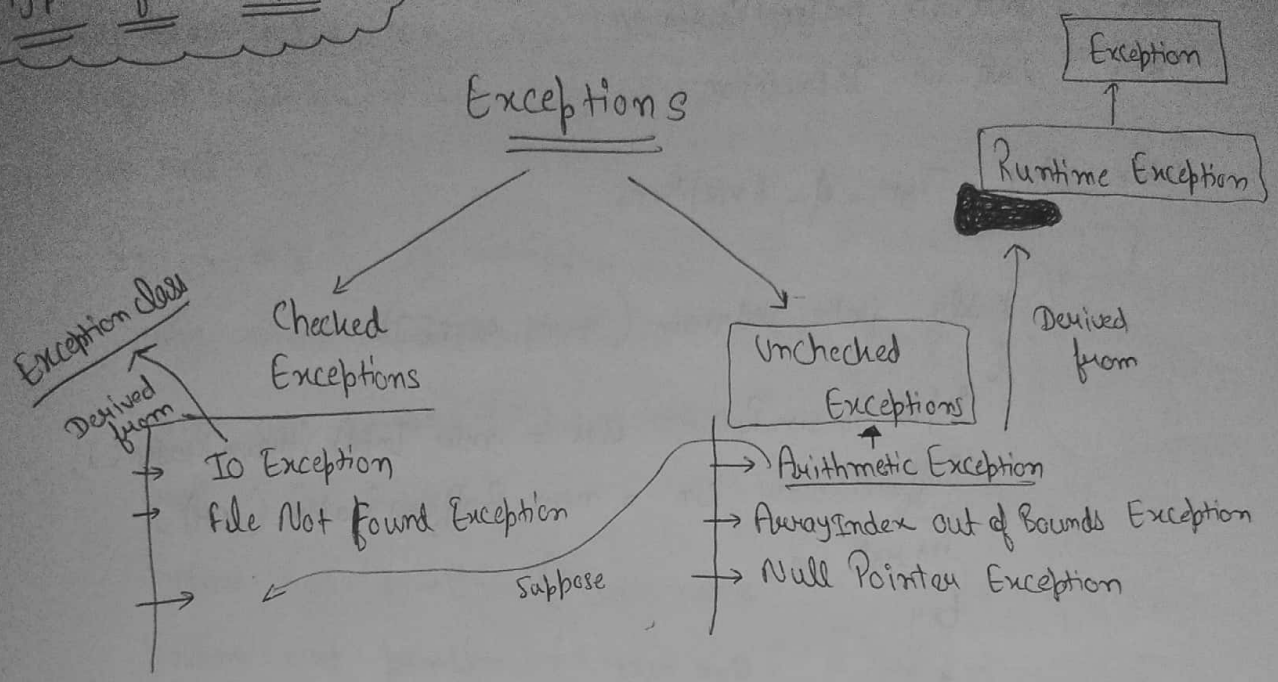
Scanned with CamScanner

```
void doDiv (int d)
{
    int A[] = {10, 20, 30};
    try
    {
        int val = A[d];
        System.out.println ("Array val =" +val);
        int div = val/d;
        System.out.println ("Div = " + div);
    }
    Catch (Arithmetic Exception exp)
    {
        System.out.println (exp.getMessage());
    }
    System.out.println ("******");
}
}
```

<u>Note</u> :- Stack trace is used to know about the point of failure that occurred during execution of program. Java provides printStackTrace() and getStackTrace() methods to print all the stack trace elements on a run time stack.

Scanned with CamScanner

* Types of Exceptions :-



In Case of checked Exceptions, the method which can throw checked Exception must be called within try block or pass the exception using throws clause, otherwise it will be detected at Compile time.

```
class CHKD || We Suppose if Arithmetic Exception is checked to understand this topic
{
  public static void main (String args [])
  {
    int A[] = {8};
    try
    {
      System.out.println (A[10]);
      int x = 5/0;
      System.out.println ("x = " + x);
    }
    Catch (-)
    {
      ==
    }
    System.out.println (" end");
  }
}
```

Scanned with CamScanner

```
import    java.io. InputStreamReader;
import    java.io. BufferedReader;
import    java.io. IOException;

Public class  Type_of_Exceptions
  {
    */ public static void main (String args[])
      {
        InputStreamReader isr = new InputStreamReader();
        BufferedReader cin = new BufferedReader (isr);
        int roll = 0;
        try
        {
        System.out.println ("Enter Name :");
        String name = cin.readline();
        System.out.println (" Enter roll No :");
        roll = cin.read();
        System.out.println ("your name =" + name);
        System.out.println (" your roll =" + roll);
      } Catch (Exception exp)
        {
          exp.getStackTrace();
        }*/

    public static void main (String args[])
    {
      Type_of_Exceptions chk = new Type_of_Exceptions();
      try
        {
          chk.doInput();
        } Catch (Exception exp)
          {
            exp.printStackTrace();
          }
    }
```

Scanned with CamScanner

```
void doInput () throws IOException // means try catch will be in main
{
  InputStreamReader isr = new InputStreamReader (System.in);
  BufferedReader cin = new BufferedReader (isr);
  int roll = 0;
  System.out.println ("Enter name :");
  String name = cin.readLine ();  it may through IoException
  System.out.println ("Enter roll No.");
  roll = Cin.read();
  System.out.println ("your name = " + name);
  System.out.println ("your roll = " + roll);
}
}
```

Scanned with CamScanner

☆ Creating our own Exceptions :-

We can create checked Exception as well as unchecked Exception.
We will learn by creating unchecked Exception.
We know there is not any Float Arithmetic Exception, we will handle this by creating our own Exception.

```
class FloatArithmeticException extends Exception
{
  FloatArithmeticException (String msg)
    {
      Super (msg); // Calling Parametrized Constructor of base class
    }
}

public class FloatDiv
  {
    public Static void main (String args[])
      {
        int d = 0;
        try
          {
            if (d == 0)
              throw new FloatArithmeticException ("Div by zero is
                                                  invalid in float also");
          else
            {
              double x = 5.0/d;
              System.out.println (x);
              System.out.println ("End");
            }
      }
```

Scanned with CamScanner

```
Catch (FloatArithmeticException exp)
  {
    System.out.println(exp.getMessage());
  }
  System.out.println("End Again");
}
}
```

Throwable

```
String DM; // private variable
Throwable (String msg)
  {
    DM = msg;
  }
  getMessage()
  {
    return DM;
  }
```

Exception : Throwable

```
Exception (String msg)
  {
    Super (msg);
  }
```

<u>Note</u> :- If we handle checked Exception, than we have to inherit Runtime Exception rather than Exception.

Scanned with CamScanner

☆ String class :-

```
class StringClass
{
    public static void main (String args[])
    {
        */
            String S1 = "BCE"; // Using String Constant for creation of
                                  object
            String S2 = new String ("BCE");
            System.out.println (S2.length()); // 3
            String S3 = new String (); // Creates a (0) length String
            System.out.println (S3.length()); // 0
            byte ary1[] = { 65, 67, 68 };
            String S4 = new String (ary1);
            System.out.println (S4);
            byte ary2[] = {'S', 'S', 't'};
            String S5 = new String (S2);
            System.out.println (S5.length());
            String S6 = new String (S2); // Copy Content of S2 - new object
            System.out.println (S6);
            String S7 = S6; // Just reference Copied
            System.out.println (S7); // BCE */
    // - - - - - - - - - - -
```

Scanned with CamScanner

```
*/ String S1 = new String ("SST");
   String S2 = new String ("SST"); || new object Created with Same Contents
   || Reference Equality operator
   System.out.println (S1 == S2); || False
   System.out.println (S1.equals (S2)); || True
   String S3 = "real"; || object created at String Constant pool
   String S4 = "real"; || no new object created
   System.out.println (S3 == S4); || True
   System.out.println (S3.equals (S4)); || True
   String S5 = "re"; String S6 = "al";
   S5 = S5 + S6; || new object will be created on heap not on pool
   System.out.println (S3 == S5); || false
/*
||- - - - - - - - - - - - - - - - - - - -
|| String class objects are immutable - Can not be changed
/*
   String S1 = new String ("Banglore");
   String S2 = S1.toUpperCase();
   String S3 = S1.toLowerCase();
   System.out.println (S1); || Banglore
   System.out.println (S2); || BANGLORE
   System.out.println (S3); || banglore
*/
|| - - - - - - - - -
```

Scanned with CamScanner

```
/* String S1 = new String ("Bangloreglo");
boolean b1 = S1.endsWith (".jpg"); // whether it ends with it or not
System.out.println (b1); // false
System.out.println (S1.Starts with ("Ban")); // True (whether it starts
                                                 with it or not)

int Pos = S1.Index of ("glo");
  if (Pos == -1)
     System.out.println ("Not found");
  else
     System.out.println (Pos);
  int Pos1 = S1.indexof ("glo", Pos+1); // find from Pos+1 index
     System.out.println (Pos1);
*/
// - - - - - - - - - - - - - - - - - -

*/ String S1 = new String ("BCE");
System.out.println (S1.equalsIgnoreCase ("bce")); // True
System.out.println (S1.equals ("bce")); // false
String list = "C++, Java, NodeJs, ReactJs";
String [] aryref = list.split (",");
// Using for each loop
  for (String ref : aryref)
    {  System.out.println (ref); //  C++
    }                                Java
                                     NodeJs
                                     ReactJs
String S = "aman Raman aman"
   int Pos = S.lastIndexof ("aman"); // //
   System.out.println (Pos);
*/
```

Scanned with CamScanner

Homework

Q. Input a String and Substring from user and find out position/Index of each occurance and Count also

A-

```
import Java.io. InputStreamReader;
import Java.io. BufferedReader;
import Java.io. IoException;

public class StringHomework 1
{
  public static void main (String args [])
  {
    InputStreamReader isr = new InputStreamReader (System.in);
    BufferedReader cin = new BufferedReader (isr);
    try
    {
      System.out.println ("Enter a String");
      String S = cin.readLine();
      System.out.println ("Enter a Sub-String");
      String Sub = cin.readLine();
      int Count =0;
      int check =1;
      int k = 0;
      for (int i =0 ; i< S.length () ;i++)
      {
        if (S.charAt(i) = = Sub.charAt(k))
        {
          try
          {
```

Scanned with CamScanner

```
while (k < Sub.length())
{
    if (S.charAt(i) == Sub.charAt(k))
    {
        check++;
        k++;
        i++;
    }
}
if (check-1 == Sub.length())
{
    System.out.println(Sub + " is found at" + (i - check+1) + " index");
    Count++;
    k=0;
    check=1;
    i = i-check+1;
}
} Catch (StringIndexOutOfBoundException exp) {
}
} }

// *** EASY WAY ********
int index =0, easyCount =0;
while (index != -1)
{
    index = S.indexOf(Sub, index+1);
    if (index == -1)
        System.out.println(" ");
    else
    {
        System.out.println("Found at" + index);
        easyCount++;
    }
}
System.out.println("Found at " + easyCount + " Times");
System.out.println("Found" + Count + " Times");
} Catch (IOException exp)
{
}
} }
```

Scanned with CamScanner

Q- Input a File Name with path and find out File name and parent folder.

A-

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IoException;

public class StringHomeWork2
{
    public static void main (String args[])
    {
        try
        {
            InputStreamReader isr = new InputStreamReader (System.in);
            BufferedReader cin = new BufferedReader (isr);
            System.out.println ("Enter a filename");
            String path = cin.readLine();
            int pos = path.lastIndexOf ("/"); or ("\\\\") → for("\")
            System.out.println (pos);
            System.out.println ("File name is");
            for (int i = pos+1 ; i< path.length (); i++)
            {
                char a = path.charAt (i);
                System.out.print (a);
            }
            System.out.println ();
            System.out.println ("Parent Folder :");
            int pos1 = path.lastIndexOf ("/", pos-1);
            for ( int i= pos1 +1 ; i < pos ; i++)
            {
                char b = path.charAt (i);
                System.out.print (b);
            }
```

Scanned with CamScanner

// - - - - - - - - BEST WAY - - - - -

```
String file = Path.Substring(Pos+1);
System.out.println(file);

System dir = path.Substring(pos1+1, pos);
System.out.println(dir);
}
 Catch (IOException exp)
 {
 }
}
}
```

Scanned with CamScanner

★ Some more String Concepts and Buffered String :-

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;

public StringClass2
{
    public static void main (String args[])
    {
        InputStreamReader obj = new InputStreamReader (System.in);
        BufferedReader cin = new BufferedReader (obj);
        Work ref = new Work();
        try {
            String fil = ref.doinput (cin);
            ref.process (fil);
        }
        Catch (IOException exp)
        {
        }
    }
}

class work
{
    String doinput (BufferedReader cin)
    {
        System.out.println ("Enter file name and path:");
        String s = cin.readLine();
        return s;
    }
```

Scanned with CamScanner

```
void process (String fil)
{
    int index = fil.lastIndexOf ("\\");
    int index2 = fil.lastIndexOf ("\\", index -1);
    String filname = fil.SubString (index +1);
    String folder = fil.SubString (index2 +1, index);
    System.out.println ("FileName :" + filname);
    System.out.println ("Folder :" + folder);
}
}
```

Scanned with CamScanner

★ **Threads** :- It is an independant sequential path of Execution within a program. They are ligth weight as compared to independant process.

It is important to keep in mind that " Threads running in parallel" does not mean that they are actually running at same time. Since all the threads are running on a single Processor, thus according to CPU Scheduling, CPU switches among different threads randomly. JVM handles the switching process so rapidly that it appears that the are running con-currently. Like other threads, main is also a thread So it is involved in the switching process. As the threads finish their Jo they come out from switching process.

```
class Thread1
{
    public static void main (String [] args)
    {
        a ref1 = new a ();
        b ref2 = new b ();

        ref1. Start ();
        ref2. Start ();
    }
}
```

Scanned with CamScanner

```
class a extends Thread
{
    public void run()
    {
        for (int i=0; i<5; i++)
        {
            System.out.println (i);
            try
            {
                Thread.sleep (1000);
            }
            Catch ( InterruptedException e)
            {
                e.printstacktrace ();
            }
        }
        System.out.println (" i wala thread finished");
    }
}

class b extends Thread
{
    public void run()
    {
        for (int j=0 ; j<5 ; j++)
        {
            System.out.println (j);
            try
            {
                Thread.sleep (1000);
            }
            Catch (InterruptedException e)
            {
                e.printstacktrace();
            }
        }
        System.out.println (" j wala thread finished");
    }
}
```

Scanned with CamScanner

⟹ Suppose we have to do for 100 objects then can we make 100 classes, now, we will use Constructors for best approach acc. to Nature of Task that if we want to print integers we can use this approach.

```
class MThread extends Thread
{
    int SP, EP;

    public void run ()
    {
        for (i= SP ; i<= EP ; i++)
        {
            System. out. println (get Name () + " : " + i);
            try
            {
                Thread. Sleep (1000);
            }
            Catch (Exception exp)
            {
                exp. print Stack trace ();
            }
        }
        System. out. println (Thread. Current Thread () + " Stopped ");
    }

    MThread (int S, int E, String name)
    {
        SP = S;
        EP = E;
        Set Name (name);
        start (); // go to ready Queue (it means CPU Scheduling)
    }
}
class SST
{
    public static void main (String args[])
    {
        MThread th1 = new MThread (1, 5, "A");
        MThread th2 = new MThread (6, 10, "B");
    }
}
```

Scanned with CamScanner

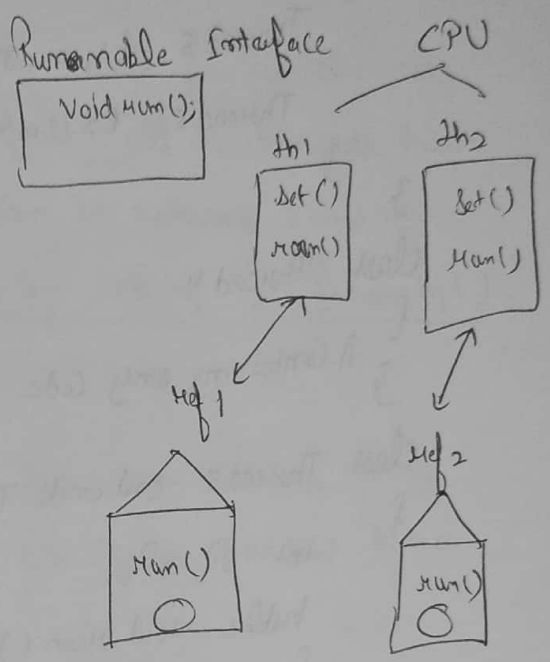
Now creating Threads Using Interface (i.e. Runnable)

X class A extends B, Thread X

{ multiple Inheritance is not Allowed, ⇒ use interface

}

```
Class B implements Runnable
{ public void run()
  { for (i=1; i<5; i++)
    {
      S.O.P (Thread.CurrentThread() + " :" + i);
      try { — }
        Catch () { — }
    }
```




```
Class C implements
{ ————
  for (int i = 6; i<=10; i++)
    {
      ————
    }
  ————
}
```

Previous Code

```
class R Inter
{ public static void main (String args[])
  {
    B ref1 = new B();
    C ref2 = new C();
    Thread th1 = new Thread (ref1);
      th1.SetName ("B");
      th1.Start ();
    Thread th2 = new Thread (ref2);
      th2.Set Name ("C");
      th2.Start ();
  }
}
```

Polymorphic Argument

class Thread

Thread (Runnable ref)

run ()

⇒ BEST Approach for this

```
public class Runnable - Interface
{
    public static void main (String args[])
    {
        Threads t1 = new Threads (1, 5, "A");
        Threads t2 = new Threads (6, 10, "B");
    }
}
class Thread4
{
}  // Containing any code

class Threads extends Thread4 implements Runnable
{
    int SP, eP;
    public void run ()
    {
        for (int i=SP; i<=eP; i++)
        {
            System.out.println (Thread.currentThread().getName()+
                                ":" + i);
            try
            {
                Thread.Sleep (1000);
            }
            Catch (InterruptedException e)
            {
                e.printStacktrace();
            }
        }
        System.out.println (Thread.currentThread() + ": stop");
    }
    Threads( int s, int e, String name)
    {
        SP = s;
        eP = e;
        Thread ref = new Thread (this);
        ref.Start ();
        ref.Setname (name);
    }
}
```

Scanned with CamScanner

Thread Priority :- We know threads in the ready state can be choosen by the Scheduler, the order in which threads are executed, cannot be predicted in advance, But sometimes, we can influence the scheduler by specifying the priority of the thread.

The thread having higher Priority run more frequently as compared to low priority threads. By default, all the threads have same priority, The priority of thread can be between 1 and 10.

"We can also get the priority of thread by calling getPriority()."

Note :- As we done programs earlier, we saw one thing that Thread.currentThread() gives output like this, Thread[A, 5, main]

Actually, it is the currently executing thread object in which

A → thread name

5 → thread priority

main → parent thread

and it is returned by currentThread() method.

Scanned with CamScanner

★ **Join() Method** :- Thread class Provides Join() Method. It will block the calling thread from becomming Runnable till the other thread is Alive.

As main bhi ek thread hai to CPU isko bhi baki threads ke saath hi process krega and we want ki jo thread run kr rha hai pehle vo run ho jaaye. Uske baad main thream finish ho. To do that, we will use Join method.

```
Class Joining_ok
{
  public static void main (String args[])
  {
    System.out.println ("Main thread Started");
      Task thrd = new Task ("Real Java");
       thrd.start();
       try
        {
          thrd.join();
        }
       Catch (Exception ex)
        {
          ex.printStackTrace();
        }
       System.out.println ("Main thread finished");
  }
}

Class Task extends Thread
{
  Task (String name)
   {
     SetName (name);
   }
  public void run ()
   {
     for (int i=1 ; i<=2 ; i++)
      {
        System.out.println ("thread→ "+getName()+ ":"+i);
      }
     System.out.println (getName() + " : thread finished");
   }
}
```

OUTPUT :-

Main thread Started
thread → Real Java : 1
thread → Real Java : 2
Real Java : thread finished
Main thread finished.

Scanned with CamScanner

Daemon Thread :- A Daemon Thread is simply a thread that serves other threads only. As Daemon thread just exists to serve user threads, so if there are no more user threads then all Daemon threads are terminated by the jvm automatically.

Daemon threads are suitable for background tasks like garbage collection process. The Garbage Collection runs from time to time and get the memory free from those objects that no longer have valid references. That is why Java programmers does not need to worry about memory management. Hence garbage collector is useful thread, which is marked as a Daemon thread.

(Hum ye bhi keh sakte hai ki child thread depend hogga parent thread ke behaviour par)

```
Class AlphabetService extends Thread
{
    // this process will take 2600 milliseconds to complete
    public void run()
    {
        System.out.println("Printing Started By Daemon Thread");
        for (char ch = 65; ch<=90; ch++)
        {
            System.out.println(ch);
            try
            {
                Sleep(100);
            }
            Catch (Exception ex)
            {
                ex.printStackTrace();
            }
        }
        System.out.println("Printing Completed");
    }
}
```

Scanned with CamScanner

```
public class DaemonThread
{
    public static void main (String args[])
    {
        System.out.println (" Main thread (non-Daemon) Started");
        AlphabetService ServiceThread = new AlphabetService();
            ServiceThread.SetDaemon (true);
            ServiceThread.Start();
            try
            {
                Thread.sleep (1000);
            }
            Catch (Exception ex)
            {
            }
        System.out.println (" Main thread (non-Daemon) Stopped");
    }
}
```

Now, If we want ki jo child thread hai vo parant thread pe depend kre, Like main bhi ek thread hai, Now we want Jab main thread finish ho Jaaye, tab child thread bhi finish ho Jaaye, Uske baad vo na chale then we will use setDaemon () method.

Scanned with CamScanner

Synchronized Method | Synchronization Block :-

We know thread allows multiple activities to proceed concurrently, while execution multiple threads may share common resources. Thus multiple threads compete for the common resources, called race condition. However, there are critical situations, where it is desirable that only one thread at a time has an access to a shared resource to avoid the race condition and it is acheived by Synchronization.

Synchronization works with locks. Every object in Java has built in lock, that only comes into play when the object has synchronized method or synchronized block. There is only one lock per object, if one thread picked the lock, no other thread can pick it until the first thread releases the lock.

1st way :- Using Synchronized with function

```
class Resource
{
    Synchronized void printTable (int tbl) || Synchronized method
    {
        for (int i=1 ; i<=5 ; i++)
        {
            int Value = tbl * i;
            System.out.println (tbl + "*" + i + "=" + Value);
            try
            {
                Thread.sleep (100);
            }
            Catch (Exception ex)
            {
                ex.printStackTrace();
            }
        }
    }
}
```

Scanned with CamScanner

```
Class User implements Runnable
{
    Resource res;
    Thread th;
    int tbl;
    User (Resource res , int tbl)
        { this.res = res;
          this.tbl = tbl;
          th = new Thread (this);
          th.start();
        }
    public void run()
        {
          res.printTable (tbl);
        }
}

public class SynchronizedMethod
{
    public static void main(String args[])
    {
        Resource res = new Resource();
        User thrd1 = new User (res, 5);
        User thrd2 = new User (res, 10);
    }
}
```

2nd Approach :- • Using Synchronized with object

```
public class Syncmeth
{
    public static void main (String args[])
    {
        resource res = new resource();
        Users ref1 = new Users (res, 5);
        Users ref2 = new Users (res, 10);
    }
}
```

Scanned with CamScanner

```
class Resource
{
    public void dofun (int tbl)
    {
        System.out.println ("table of" + tbl);
        for (int i = 1; i<=10; i++)
        {
            System.out.println (tbl + "*" + i + "=" + (tbl*i));
        }
        System.out.println (Thread.currentThread() + "finished \n");
    }
}

class Users implements Runnable
{
    Resource reuse;
    int tbl;
    Users (Resource res, int t)
    {
        Thread ref = new Thread (this);
        reuse = res;
        tbl = t;
        ref.start();
    }
    public void run()
    {
        Synchronised (reuse) || Synchronized methods of lib class
                                as we cannot change them
        {                       as we are not creating lib
            reuse.dofun (tbl);  classes.
        }
    }
}
```

Scanned with CamScanner

☆ Files and Streams :-

- Sometimes our program needs to read data from file and write data to a file. Java offers various classes in a single java.io package to facilitate their input/output operations.
- Java treats I/O in the form of streams.
- The data flow b/w a program and I/O devices through these streams.

☆ File class :- It is offered by Java.io package and by using this we can work file system on our machine. Not only files but directory also.

Three Constructors →
1. File (String path Name)
2. File (String parent, String child)
3. File (File parent, String child).

Program :-

```
import java.io.File;
import java.io.IoException;
import java.util.Date;
public class FileClass
{
    public static void main (String args[]) throws IOException
    {
        /*
            File Fi = new File("/home/Prince/Desktop");
            System.out.println (Fi.exists()); // whether this file
                                                 exists or not
                                                 returns boolean
        */
```

Scanned with CamScanner

```
/* File obj1 = new File("/home/Prince/Princy.com");
System.out.println(obj1.getAbsolutePath());
System.out.println(obj1.getCanonicalPath());
```

both will return Path of file chahe exist kre chahe nhi

```
if (obj1.exists())
{
    System.out.println(obj1.getName());
    System.out.println(obj1.getParent());
    System.out.println(obj1.getPath());
}
else
{
    System.out.println("File Not found");
}
*/
```

will return name of file

will return name of parent folder

will return path of file

```
/*
    File dir = new File("myFile.txt");
    System.out.println(dir.getCanonicalPath());
    System.out.println(dir.getAbsolutePath()); // it show file name
                                                with current folder path
*/

File dir = new File("/home/Prince/Desktop/REAL_JAVA");
File Ary[] = dir.listFiles(); // returns array of objects of dir File
int fCount=0, dCount=0;
for(File ref: Ary)
{
    // System.out.println(ref.getPath());
    if(ref.isFile())
    {
        Date dt = new Date(ref.lastModified());
System.out.println(ref.getName() + dt.toLocaleString() + "\t" +
    "Bytes");
```

Scanned with CamScanner

```
else if (ref.isDirectory())
    {
        d Count++;
        System.out.println("                    <DIR>" + ref.getName());
    }
}
System.out.println("Files = " + f Count + "    Directories = " + d Count);
}
}
```

Home Work

Q- Enter a path and find its all files and sub folders and their files and so on.

A -

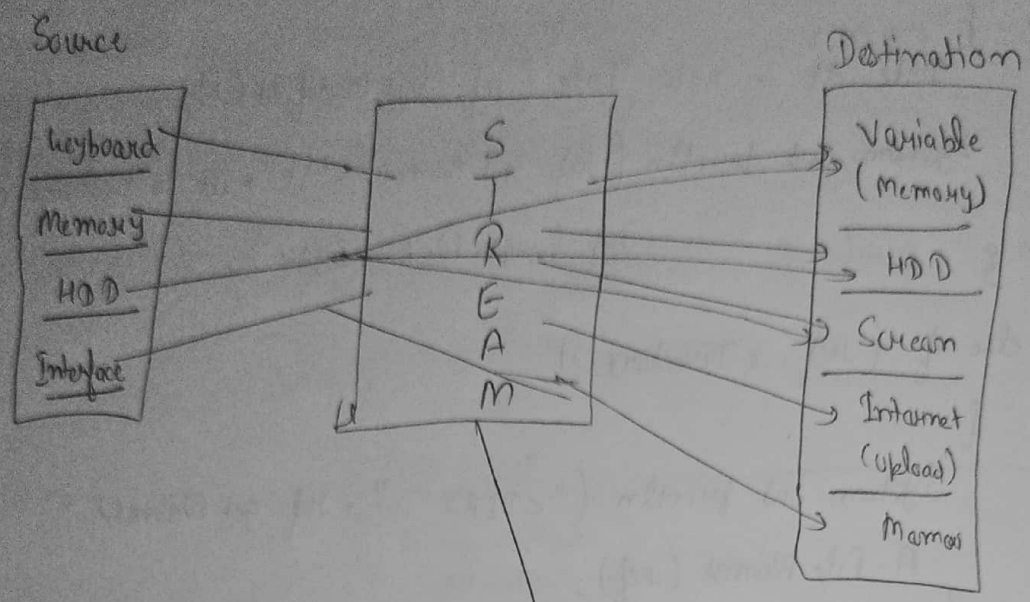
```
import java.io.Exception;
import java.io.File;
import java.io.Date;
import java.io.Scanner;
public class All Files of System
{
    public static void main (String args[]) throws IoException
    {
        file dir = new file("/home/Prince/Desktop/REAL_JAVA");
        file [] ary = dir.list Files();
        All files of System A = new All Files of System();
        System.out.println("Files of :" + dir.get Name());
```

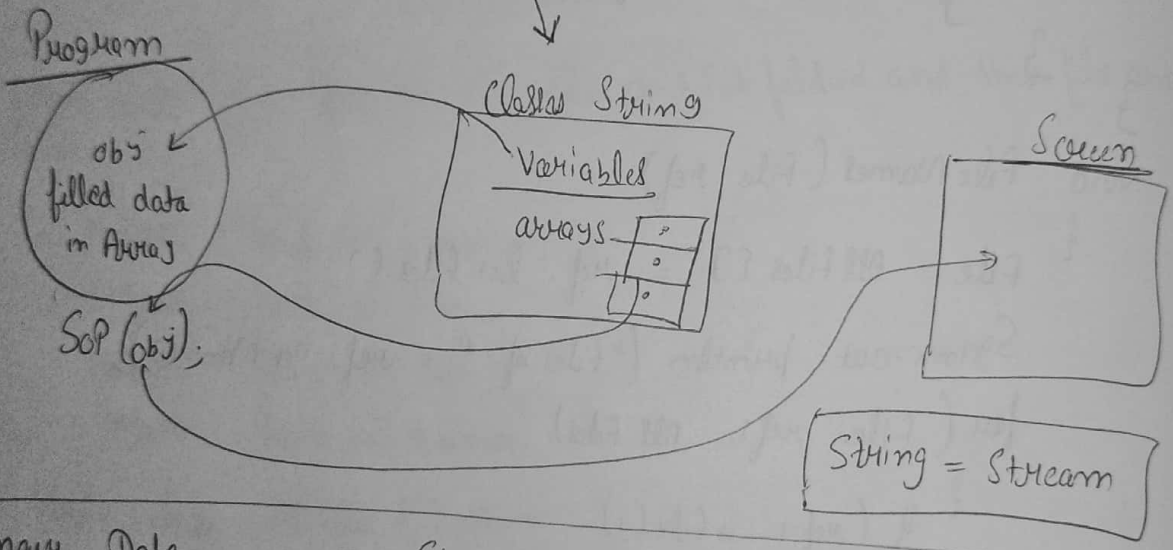
Scanned with CamScanner

```
for ( File ref : arg)
{
    if (ref. isFile())
    {
      Date dt = new Date (ref. lastModified());
      System. out. println (ref. getName() + "\t" + dt. toLocalestring() + "\t" +
                                ref. length() + " Bytes");
    }
    else if (ref. isDirectory())
    {
      System. out. println ("<DIR>  " + ref. getName() + "  <DIR>");
      A. FileNames (ref);
    }
  }
}
void FileNames (File ref)
{
  File AllFiles [] = ref. listFiles();
  System. out. println ("Files of :" + ref. getName());
  for ( File ref1 : AllFiles)
    {  if (ref1. isFile())
       {
         Date dt = new Date (ref1. lastModified());
  System. out. println (ref1. getName() + "\t" + dt. toLocaleString() + "\t" +
                                ref1. length() + " Bytes");
       }
    else if (ref1. isDirectory)
    {
      System. out. println ("<SUB DIR> of" + ref. getName() + "  " +
                                ref1. getName() + " <SUB DIRS>");
      FileNames (ref1);
    }
  }
}
}
```

Scanned with CamScanner

☆ **Stream** :- A way to transfer data b/w Source and destination

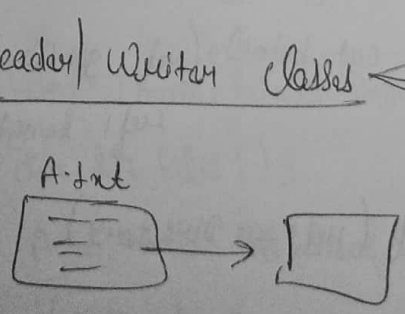




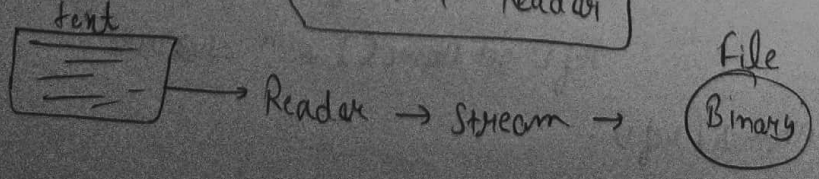
Binary Data → Stream Classes

text Data → Reader/Writer Classes

Copy (Movies, images, Songs)



Stream + Reader



Scanned with CamScanner

Types of Streams :- ① Byte Streams
② Character Streams

* Byte Streams → Input Streams
→ output Streams

* Methods in Input Stream classes : ① int read() throws IO Exception
② int read(byte [] arrays) throws IO Exception
③ int read (byte [] array, int Start index, int N) throws IO Exception

* Methods in Output Stream classes :- ① Void Write (int) throws IO Exception
② Void Write (byte [] array) throw IO Exception
③ Void Write (byte [] array, int Start index, int N) throws IO Exception
④ Void close () throws IOException
⑤ Void flush() throws IO Exception

* File Input Stream Class :- Constructors :-
① File Input Stream (String File Name)
② File Input Stream (File file object)
③ File Input Stream (FileDescriptor desc object)

it throws File Not Found Exception.

* File output Stream Class :- Constructors :-
① File output Stream (String file Name)
② File output Stream (String FileName, boolean append)
③ File output Stream (File fileobject)
④ File output Stream (FileDescriptor fdobject)

Scanned with CamScanner

☆ Program to Copy Binary Data :- Here will Copy .JPG & .mP3 file.

```
import java.io.*;

public class BinaryDataConversion
{
    public Static void main (String args[])
    {
        try
        {
            CopyImage();
        }
        Catch (IoException e)
        {
            e.printStackTrace();
        }
    }
    Static Void CopyImage () throws IoException
    {
        File Src = new File ("download.jPeg"); // take file from current folder
        File des = new File ("wahegwru.jPeg");
        FileInputStream fis = new FileInputStream (Src);
        File Output Stream fos = new File OutputStream (des);
        int c, Count =0;
        try
        {
            while ((c= fis.read()) != -1)
            {
                fos.write (c);
                Count ++;
            }
            Catch (FileNotFoundException e)
            {
                e.printStackTrace();
            }
        System.out.println (Count + " Byte Copied");
            fis.close();
            fos.close();
        }
    }
```

Scanned with CamScanner

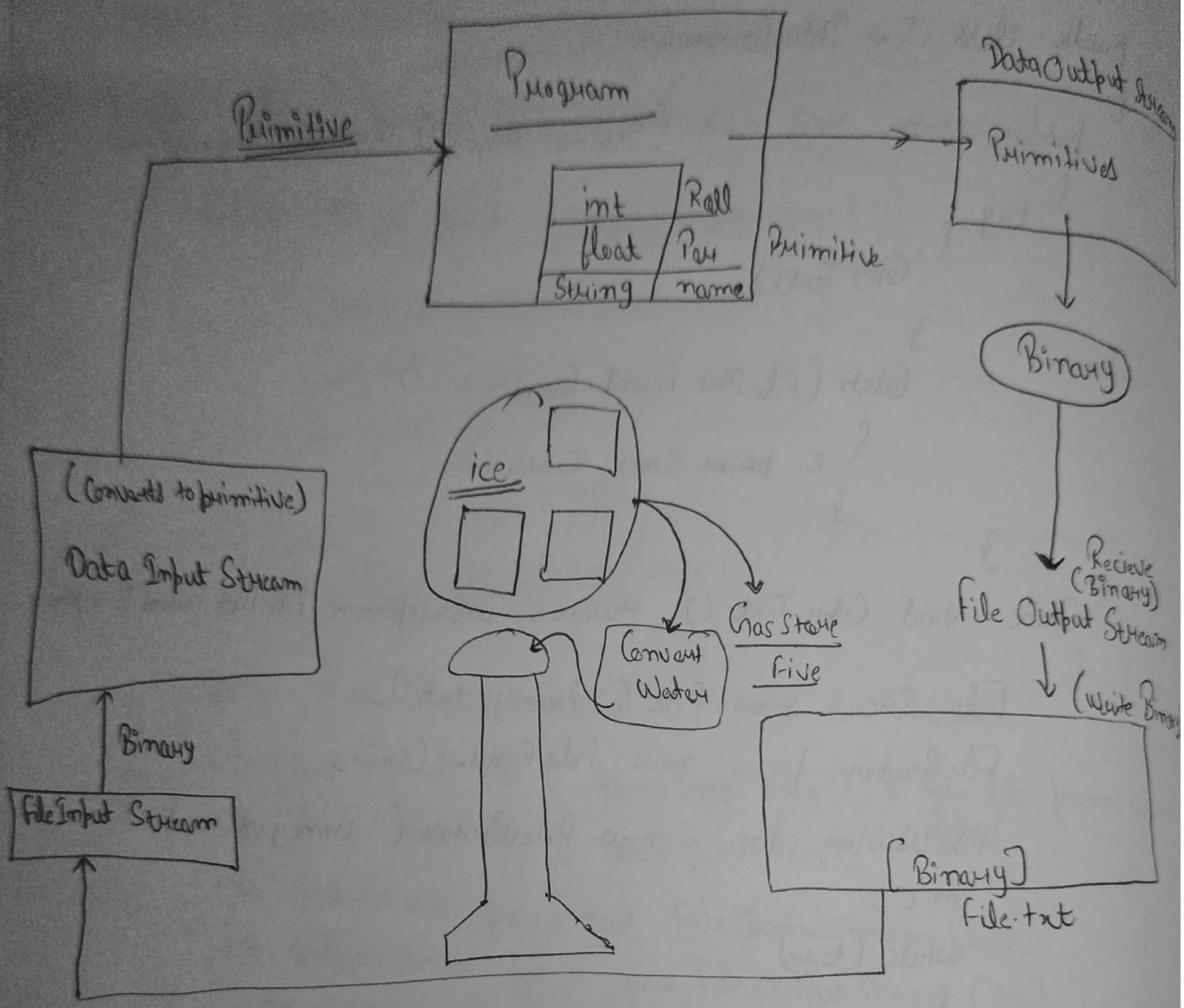
Program to Copy Text data

:- Here we will copy text data from one file to another. (Uses Reader & Writer classes).

```
import. java.io.*;
public class Text Data Conversion
{
 public Static Void main (String args[]) throws IoException
 {
  try {
       Copy Text();
      }
      Catch (File Not Found Exception e)
       {
        e. print Stack Trace ();
       }
  }
 Static Void Copy Text () throws IoException, File Not Found Exception
 {
   File Src = new File ("Primey. txt");
   FileReader fis = new File Reader (src);
   File Writer fos = new File Writer ("Princy 2. text");
   int c;
   While (true)
   {
       c = fis .read ();
       if (c == -1)
           break;
       fos. Write (c);
       System. out. print ((Char) c);
       System. out. ("File Copied");
   }
   fis. close();
   fos. close();
  }
}
```

Scanned with CamScanner

★ Program for Binary To Primitive To Binary :-



- Constructor of DataInputStream (InputStream object)
- Constructor of DataOutputStream (outputStream object)

Scanned with CamScanner

```
import java.io.*;
public class DataInOutStream
{
  public static void main (String args[])
  {
    int ProductId = 1;
    String ProductName = "Laptop";
    int Quantity = 2;
    float price = 100000;
    int ProductId2 = 2;
    String ProductName2 = "Bike";
    int Quantity2 = 5
    float Price2 = 2400000;
    try
    {
      FileOutputStream fos = new FileOutputStream("Product.txt", true);
                                  // opening file in
                                     append mode
      DataOutputStream dos = new DataOutputStream (fos);
      // Write 1st Record
        dos.WriteInt (ProductId);
        dos.WriteUTF (ProductName);
        dos.WriteInt (Quantity);
        dos.WriteFloat (price);

      // Write 2nd Record
        dos.WriteInt (ProductId2);
        dos.WriteUTF (ProductName2);
        dos.WriteInt (Quantity2);
        dos.WriteFloat (Price2);

        dos.flush();
        dos.close();
```

Scanned with CamScanner

```
FileInputStream fis = new FileInputStream ("Products.txt");
DataInputStream dis = new DataInputStream (fis);
// Reading Records from file

try
{
    while (true)
    {
        ProductId = dis.readInt();
        System.out.println(" Id : " + ProductId);
        ProductName = dis.readUTF();
        System.out.println(" Name : " + ProductName);
        Quantity = dis.readInt();
        System.out.println(" Quantity : " + Quantity);
        price = dis.readFloat();
        System.out.println(" Price : " + price);
    }
}
Catch (Exception ex)
{
}

dis.close();
}
Catch (IOException e)
{
}
}
}
```

Scanned with CamScanner

* Object Input and Object output Stream or SERIALIZATION AND DESERIALIZATION

We will use this, when we want to convert the primitive data into Binary data and Binary into primitive data, but as we saw Earlier, we also doing this same, but we have to write and read for every variable and if we have 100 variables, then?, So, we will make an object containing all these variables and then perform conversion on it.

```
import java.io.*;

class Company implements Serializable
{
    String name, Website;
    Company (String n, String ws)
    {
        name = n;
        Website = ws;
    }
    String getCompileDetails()
    {
        return "Company Name = " + name + " Website Name = " + Website;
    }
}
class Product implements Serializable
{
    String ProductName;
    Company Comp;
    Product (String Pn, String nm, String ws)
    {
        Comp = new Company (nm, ws);
        ProductName = Pn;
    }
```

Scanned with CamScanner

```
    Void ShowProduct Details ()
    {
      System.out.println ("Product =" + ProductName + ", " +
                                  Comp.get CompileDetails ());

    }

}
class Serializable_Process
  {
    Void SerializableObject (Product P)
    {
      try {
          FileOutputStream fos = new FileOutputStream ("file.txt");
          Object output Stream oos = new ObjectOutput Stream (fos);
          oos.writeObject (P);
          oos.close();
          fos.close();
          System.out.println (" object Serialized Succefully");
      }
      Catch (Exception e)
      {
      }
    }
    Void deSerializable object ()
    {
      try {
          System.out.println (" --- Deserializing Start --- ");
          FileInputStream fis = new File Input Stream (" file.txt");
          ObjectInput Stream ois = new objectInput Stream (fis);
          Product P = (Product) ois.read object ();
          P.ShowProduct Details ();
          ois.close();
          fis.close();
          System.out.println (" object Deserialized Successfully");
      }
      Catch (Exception e)
      {
      }
  }
}
```

Scanned with CamScanner

```
public class Serialization Vs Deserialization
{ public Static Void main (String args[])
  {
      Serialize Process sP = new SerializeProcess();
      Product Pro = new Product ("Laptop", "Dell", "www.dell.com")
       sP. Serializable object (Pro);
       sP. Deserializable object ();
  }
}
```

⇒ Now if sometimes we want that, the data we stored in a file will not be stored (i.e. some data : like Password), then we will use 'transient' key word.

```
import java. io. *;
class Login implements Serializable
{
   String Uid;
   transient String Pwd;
   String time of Login;
   Login (String u, String Pw, String tol)
   {
      Uid = u;
      Pwd = Pw;
      time of Login = tol;
   }
   Void Show Details ()
   {
      System. out. println (" User id =" + Uid + " Password =" + Pwd + "time of login"
                                                                  + time of login);
   }
}
```

Scanned with CamScanner

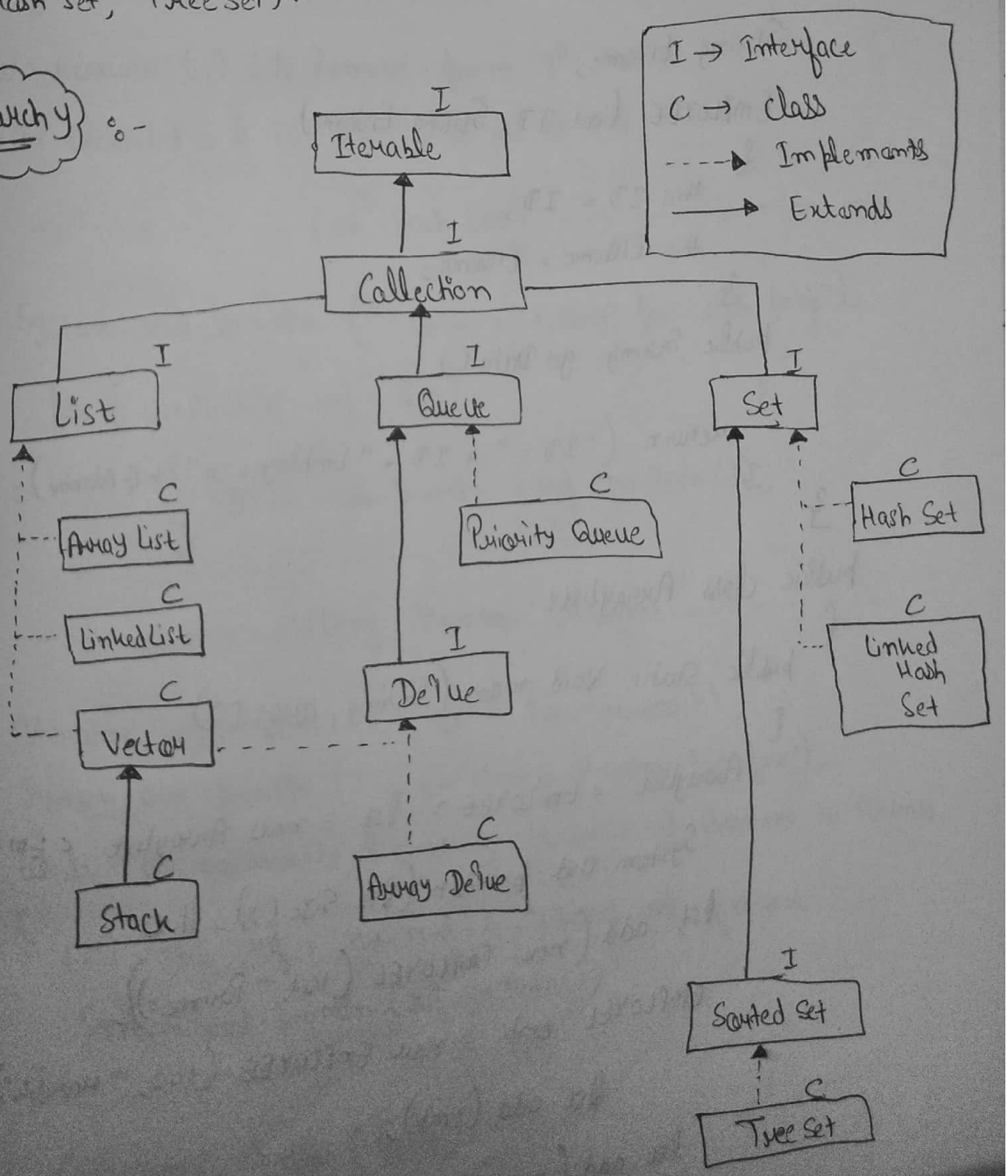
```
Class Serialize User
{
    Void Serializeobject (Login User)
    { try {
      FileOutputStream fos = new FileOutputStream ("Login.txt");
      ObjectOutputStream oos = new ObjectOutputStream (fos);
       oos. writeObject (User);
       oos. close();
       fos. close ();
      } Catch (Exception e)
       {
       }
    }
  Void Deserializableobject ()
   { try {
        FileInputStream fis = new FileInputStream ("Login.txt");
        ObjectInputStream ois = new ObjectInputStream (fis);
        Login P = (Login) ois. readObject ();
         P. showDetails();
        ois. close ();
         fis. close ();
       }
       Catch (Exception e)
        {
        }
    }
}

public class TransientKeyWord
{
  public static void main (String args[])
   {
     Serialize User SP = new Serialize User ();
       Login User = new Login ("Prince", "Wadhwa", "15:34 Am");
       SP. Serializableobject (User);
       SP. Deserializableobject ();
   }
}
```

Scanned with CamScanner

* Collection Framework :- The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.

- Java Collections Can achive all the operations that you perform on a data Such as Searching, Sorting, insertion, manipulation & Deletion.
- Java Collection means a Single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (Array List, Vector, LinkedList, Priority Queue, Hash Set, Linked Hash Set, Tree Set).

⇒ Hierarchy :-



Scanned with CamScanner

⇒ ARRAY LIST :-

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.ListIterator;

class EMPLOYEE
{
    int ID;
    String EName;
    EMPLOYEE (int ID, String EName)
    {
        this.ID = ID;
        this.EName = EName;
    }
    public String getData()
    {
        return ("ID =" + ID + " Employee =" + EName);
    }
}
public class Arraylistt
{
    public static void main (String args[])
    {
        Arraylist <EMPLOYEE> lst = new Arraylist <EMPLOYEE>();
        System.out.println (lst.size()); // 0
        lst.add (new EMPLOYEE (101, "Prince"));
        EMPLOYEE emp = new EMPLOYEE (102, "Hardik");
            lst.add (emp);
        lst.add (new EMPLOYEE (103, "Rishabh"));
```

Scanned with CamScanner

```
System.out.println (lst.size ()); //3
lst.add (new EMPLOYEE (104, "Saksham"));
System.out.println (lst.size()); //4
for (int i=0 ; i<lst.size (); i++)
  {
    EMPLOYEE ref = lst.get (i) ;
      System.out.println (ref.getData());
  }
lst.add (0, new EMPLOYEE(105, " Sanyam")); //Inserting at 0th index
System.out.println (lst.size()); //5

lst.remove (1) ; // Remove from 1st index
// lst.clear() ; // To delete/clear all objects

//- - - - - - - - - - FoR-Each LooP - - - - - - - - - - -

System.out.println (" === Using for-each Loop");
for ( EMPLOYEE ref : lst)
      {
        System.out.println (ref.getData());
      }

//- - - - - - - - - - Using Iterator Interface - - - - - -

Iterator <EMPLOYEE> itr = lst.iterator();
System.out.println (" === Using iterator Interface");
while (itr.hasNext()) // checking existence of element in Array
{
  EMPLOYEE ref = itr.next() ; //gives ref of object
    System.out.println (ref.getData());
}

Collections.shuffle (lst);
```

Scanned with CamScanner

```
||- - - - - - - - - - - Using - ENUMERATION - - - - - - - -

Enumeration <EMPLOYEE> enm = Collections. enumeration (lst);

System. out. println ("=== Using Enumeration interface");

while (enm. hasMoreElements())
{
    EMPLOYEE ref = enm. nextElement();
    System. out. println (ref. getData());
}

||- - - - - - - - - Using - ListIterator - - - - - - - - -

ListIterator <EMPLOYEE> litr = lst. listIterator();

    litr. next(); || going to 1st index
    litr. next(); || ——— 2nd ———
    litr. next(); || ——— 3rd ———
    litr. next(); || ——— 4th ———

System. out. println ("=== Using ListIterator ==");
while (litr. hasPrevious()) || going back
{
    EMPLOYEE ref = litr. previous();
    System. out. println (ref. getData());
}
}
}
```

Scanned with CamScanner

MORE FUNCTIONS OF ARRAYLIST

```
import Java.Util.*;

class ArrayList Program 2
{ public Static void main (String args[])
  {
     ArrayList Program Ar = new ArrayList Program ();
     Ar. Learn ArrayList ();
  }
}

class Array List Program
{
   public void Learn ArrayList ()
   {
      int max = 10;
      int Counter = 0;
      List <object> listA = new ArrayList <objects> ();
      List <object> list B = new ArrayList <objects> ();
      System.out.println ("-- Storing 10 integer objects");
      for (int i = 0 ; i < MAX-1 ; i++)
       {
         list A. add (new Integer (i));
       }

      System.out.println (" Storing Strings ");
      list A. add (" Amain");
      list A. add (" Raman");
      listA. add (" Elon Musk");
      listA. add (" Jeff Bezos");

      System.out.println (" Retrieve objects using an Iterator : ");
```

Scanned with CamScanner

```
Iterator <object> i = listA.iterator();
    while (i.hasNext())
    {
        object ob = i.next();
        System.out.println(ob + " , ");
    }
  int loc = listA.indexOf("Aman");
System.out.println("In location of String Aman is " :"+loc);

  List <object> listSub = listA.subList(19, listA.size());
  System.out.println("New Sublist" + listSub);
/* // Collections.sort(listSub); ⟶ For sorting Plz Plz check from our Computer.
  System.out.println("New Sorted List :" + listSub);
  System.out.println("original list :" + listSub); */
  Collections.reverse(listSub);
  System.out.println("New Reversed list :" + listSub);
  System.out.println("Is Sub-list Empty ?" + listSub.isEmpty());
  System.out.println("Are Lists A and B equal ?" + listA.equals(listB
  System.out.println("**** List A (Before) :" + listA);
  Collections.shuffle(listA, new Random());
  System.out.println("**** ListA (After) :" + listA);
  System.out.println("-- Converting An [redacted] list to an [redacted] Array --");
    object[] obyArray = listA.toArray();
      for (int j=0; j < obyArray.length; j++)
    {
      System.out.println(", Element [" + j + "] =" + obyArray[j
    }
  }
}
```

Scanned with CamScanner

Sets :- It is derived from Collection Interface. Hash Set and Linked Hash Set are the Concrete implamantations of Set interface.

- Set interface implamantation do not allow duplicates. Its add() and addAll() methods will not store duplicates.
- A Hash Set Collection does not maintain insertion order of the elemants. However, Linked Hash Set store elemants in order.

Where to use :- Like Hamare Pass 2 Lists hai, ek hai Web ke batch ki and dusri hai DotNet ke batch ki and there are Some Common Students among both batches and if we Want ki humko vo Students chahiye Jo dono batches mei ho (like union) or humko vo bache chahiye Jo sirf Java ya Sirf web ka rhe hai (like Subtraction from one another) ya vo Students Jo dono batches mei hai, To Yaha hum Use kate hai Sets. or We Can Say Jab 2 lists par (ek saath) operations karne ho.

```
import java.util.*;

public class Set_All Methods
{
    public Static void main (String args[])
    {
        String JavaBatch[] = {"Diya", "Tanvi", "Prince", "Rishabh"};
        String doNet[] = {"Hardik", "Diya", "Sakshram"};

    HashSet<String> jb = new HashSet<String>(Arrays.asList(JavaBatch));
    HashSet<String> dn = new HashSet<String>(Arrays.asList(doNet));

        System.out.println("Java Batch": + jb);

        System.out.println("DotNet Batch:" + dn);
```

Scanned with CamScanner

```
HashSet <String> dnCopy = new HashSet <String> (dn);
        dnCopy.retainAll(jb); // Like Intersection
System.out.println("Retain All:" + dnCopy);

HashSet <String> doCopy2 = new HashSet <String> (dn);
        dnCopy2.removeAll(jb); // Like (dn - jb)
System.out.println("Remove All();" + dnCopy2);

HashSet <String> All = new HashSet <String> ();
        All.addAll(jb);
        All.addAll(dn);   // Like Union
System.out.println("addAll():" + All);
    }
}
```

Scanned with CamScanner

MaPS :- A Map defines mapping from keys to values. The Pair is Called Entry. A map doesnot allow keys. In other words keys are unique but values can be Repeated.

- A map is not a Collection, because Map interface doesnot extend Collection interface.

- The Class Hash Map and Hash Table do not Maintain Insertion order as Linked Hash Set do. The Hash Map Class allows null key and it is not thread-Safe, where as Hash Table do not allow null key and it is thread Safe too.

```
import. java.util.*;

public class HashMap1_rand
{
    public Static void main (String args[])
    {
        Random rnd = new Random();
        Hash Map <Integer, Integer> m = new Hash Map <Integer,
                                                Integer>();
        for (int i=1; i<=1000; i++)
        {   int r = rnd.nextInt(10); // This will
            // randomly choose no. from (0-9) every time
            Integer occr = m.get (r);
            if (occr == null)
                occr = 1;
            else
                occr = occr + 1;
                m.put (r, occr);
        }
        System.out.println (m);
    }
}
```

It accepts two Arguments as objects one is key and other is value

★ **Design Patterns** :- Design Patterns are a Cornerstone of today's Software Development. Using Design Patterns, Creation and maintainability of Java applications Can be improved and their Cost Can be reduced.

Some Definitions of Design Pattern :-

(i) Design Patterns are recurring solutions to design Problems, you See over and over.

(ii) Design Patterns Constitute a Set of rules describing how to accomplish Certain tasks in the realm of Software development.

(iii) Patterns identify and specify abstractions that are above level of Single Classes and instances.

Note :- Design Patterns are based on the following principles of Object Oriented Design (OOD):

- Program to an Interface, not an implementation.
- Prefer object Composition over inheritance.

⇒ **4 types of Design Patterns** :- (i) Factory Pattern
(ii) Abstract factory Design Pattern
(iii) Adapter Design Pattern
(iv) Singleton Design Pattern

✓ We already done factory Pattern in Polymorphic Arguement and Polymorphic return.

- Further, we will do Singleton Design Pattern, bcz these two are most imp. for future (interviews and Development).
- Abstract factory Design Pattern is an advanced pattern/version of factory pattern and ~~is also known as~~

Scanned with CamScanner

Singleton Design Pattern :- Singleton design Pattern is a Creational design Pattern. A Singleton is simply a class which can be instantiated exactly once. There are two ways to implemant Singleton Design Pattern, both use private Constructors and a public static field of class type that contains this feild to provide access to sole instance.

Where to use :- It is used where we need to make a single object of any Particular class. Like Example in Databases, where we want that our project can have single connection with Database, Here we use Singleton Design Patterns.

1st Approach ⇒ In this a final & Static field of Same class type is used.

```
class UserLibrary
{
  public static final UserLibrary lib = new UserLibrary();
  private UserLibrary ()
  {
  }
  public double getSquare (double v)
    {  return v * v;
    }
  public double getPi ()
    {  return 3.14;
    }
  public double getCylinderVol (double r, double h)
    {
      return getPi() * getSquare (r) * h;
    }
}
```

Scanned with CamScanner

```
public class Singleton_Design Pattern
{ public static void main (String args[])
  {
    UserLibrary lib = UserLibrary.lib;
      double sq = lib.getSquare(10);
      double cylval = lib.getCylinderVal(10,20);
    System.out.println("Square = " + sq + " Cyl = " + cylval);
  }
}
```

2nd Approach ⇒ In this, a public & static factory method is used to return the reference of UserLibrary class object which is declared as private, static & final.

```
Class UserLibrary
{
  private static final UserLibrary lib = new Library();
  private UserLibrary()
  {
  }
  public static UserLibrary getInstance()
  {
    return lib;
  }

  public double getSquare(double v)
  {
    return v*v;
  }
  public double getPi()
  {
    return 314;
  }
  public double getCylinderVal(double r, double h)
  {
    return getPi() * getSquare(r) * h;
  }
}
```

Scanned with CamScanner

```
public class Singleton_Design Pattern
{
  public static void main (String args[])
  {
    UserLibrary lib = UserLibrary.getInstance();
    double sq = lib.getSquare(10);
    double cylval = lib.getCylinderVal(10, 20);
    System.out.println("square = " + sq + " cyl = " + cylval);
  }
}
```

Scanned with CamScanner

☆ NESTED CLASSES :-

Java allows us to define classes inside other classes. Nested classes enable us to logically group classes that are only used in one place, write more readable and maintainable code & increase encapsulation.

Types of Nested classes :-

- Static Nested classes.
- Non-Static Nested classes.
- Local classes
- Anonymous classes.

Scope :- ★ bounded by Scope of its enclosing class, Nested class doesn't exist independently of class outer class.

☆ A Nested class has access to members, including private members of class in which it is nested. However, reverse is not true, i.e. the enclosing class doesnot have access to members of nested class.

☆ A Nested class can be declared private, public, protected or package private (by default)

⇒ **Static Nested class :-** As due to this, Static Nested class can access only static members of its outer class, it can access to non-static members by reference obj.

Syntax :- Outer class. Static Nested class nested object = new outer class. Static Nested class()

→ **Some key Points :-**

- They only have access to static members in enclosing class.
- As with static members, these belongs to their enclosing class or not to a instance of class.

Scanned with CamScanner

Program :-

```
class OuterClass
{
    Static int Outerx = 10;
    int outery = 20;
    private Static int outerprivate = 30;
    Static class StaticNestedClass
    {
        Void Display()
        {
            System.out.println("outer x =" + outerx);
            System.out.println("outer y =" + outerprivate);
        }
    }
}
public class StaticNestedClassDemo
{
    public Static void main (String args[])
    {
        OuterClass.StaticNestedClass nestedobj = new OuterClass.
                                                 StaticNestedClass();
        nestedobj.Display();
    }
}
```

OUTPUT

```
outer x = 10
outer private = 30
```

⇒ **Non-Static Nested Class** :- Also known as inner class.

- It has access to all variables and methods of outer class including its private data members and methods and may refer to them directly. But reverse is not there. Outer class cannot directly access members of inner class. Inner class can be declared private, public, protected or default access.
- They can only define non-Static members.

⇒ **Local Class** :- This is a special type of inner class, in which the class is undefined inside a method or scope block.

- They cannot have access modifiers in their declaration.
- They have access to both static & non-Static members in enclosing context.
- They can only define instance members.

Scanned with CamScanner

Program for Local class :-

```
public class BCE
{
    Void bti ()
    {
        class Local
        {
            Void run()
            {
                System.out.println ("Inner Run Method");
            }
        }
    }
    Local local = new local();
        local.run();

public Static void main (String args[])
    {
        BCE bce = new BCE();
        bce.bti();
    }
}
```

Program for Non-Static Nested class :-

```
class outer
{
    public Void Display ()
    {
        Inner in = new Inner();
        in.show();
    }
    Class Inner
    {
        public void show()
        {
            System.out.println (" Inner class");
        }
    }
}
```

Scanned with CamScanner

```
public Test
{ public static void main (String args[])
    {
        Outer ot = new Outer();
        ot.display();
    }
}
```

→ Another Syntax :-

```
Outer ot = new Outer();
Outer.Inner innerobj = ot.new Inner();
```

⇒ **Anonymous Class** :- It can be used to define an implementation of an interface or an abstract class without having to create a reusable implamantation.

- Cannot have Access modifiers.
- Access to both static and non-static members in enclosing Context.
- They can only define instance members.
- They are the only type of nested classes that cannot define Constructors or extend other classes or interface.

Program :-

```
interface Salary
{
    void Calc (int months, int amount);
}
class Outer
{
    class Inner
    {
        void hi()
        {
            System.out.println ("hi");
        }
    }
}
class Employee implements Salary
{
    public void Calc (int month, int amount)
    {
        System.out.println (months * amount);
    }
}
```

```
class AnonInn class
{
    public static void main (String args[])
    {
        //- - - - - - - - - - - - -
        Outer.Inner inRef = new Outer().new Inner();
        inRef.hi();
        //- - - - - - - - - - - - -
        new Employee().Calc(2, 2000);
        //- - - - - - - - - - - - -
        Salary emp = new Salary()
        {
            public void Calc (int month, int amount)
            {
                System.out.println (month * amount);
            }
        };
        emp.Calc (5, 10000);
        emp.Calc (7, 100000);
        //- - - - - - - - - - - - -
        new Salary()
        {
            public void Calc (int mon, int am)
            {
                System.out.println(mon * am);
            }
        }.Calc (22, 3333);
    }
}
```

Scanned with CamScanner