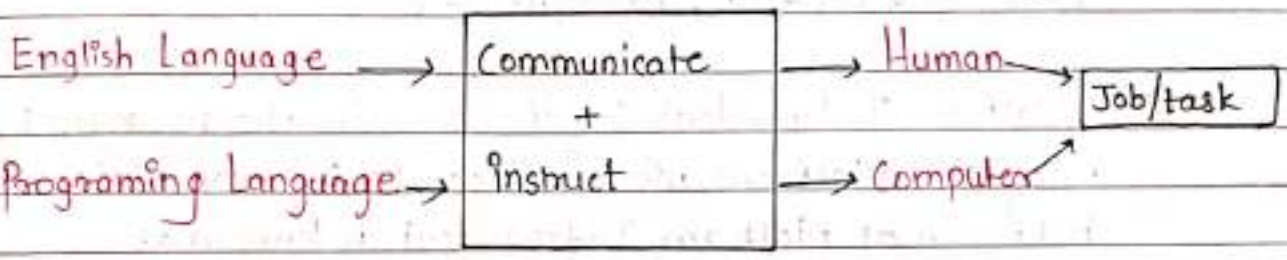
18/10/2022
Tuesday

CORE JAVA

Page No.
Date

* Programing language :- It is a Language which is used to communicate and instruct to computer to perform task/Job.



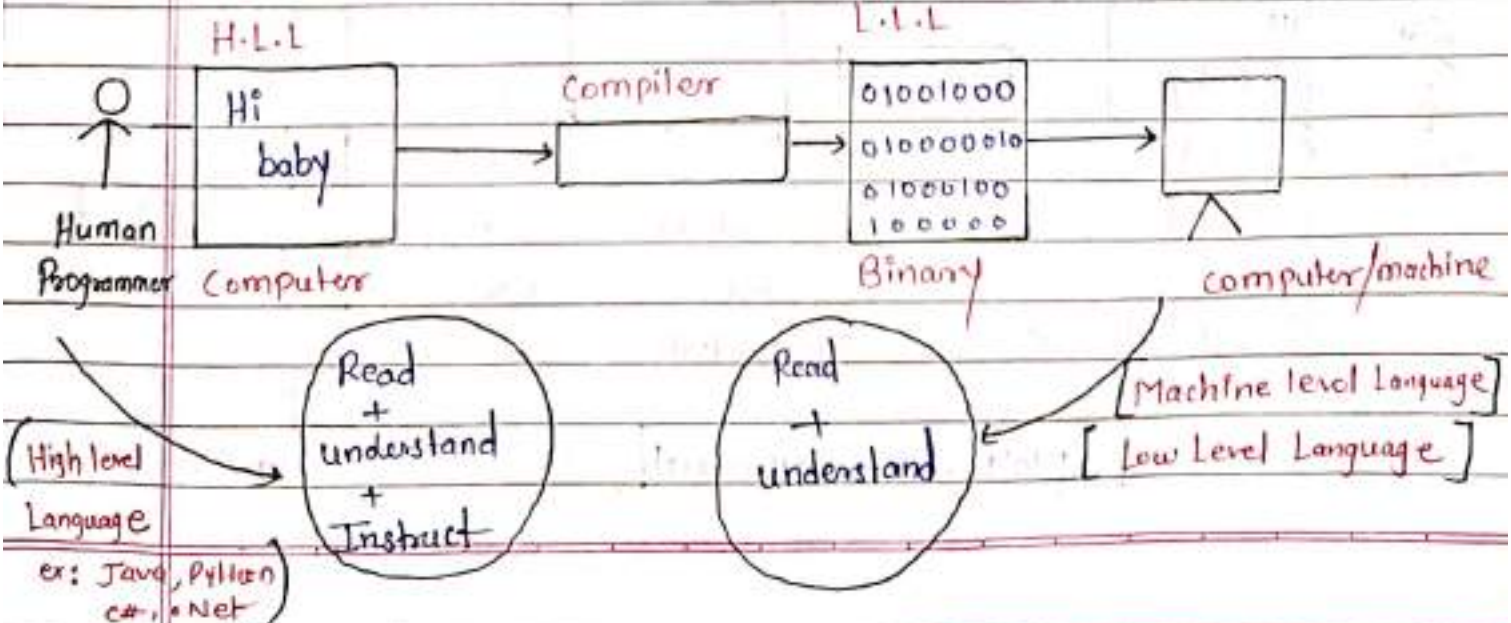
19/10/2022
Wednesday

* Low Level Language :- The Language it is a easily Readable, and understanding by the machine and/or computer is known as low Level Language it is also called as machine level Language.

example :- Binary Number → 0 , 1
off on

* high level Language :- The Language it is a easily Readable, understandable and instuct by the programmer or human is known as high level Language.

example :- Java, .Net, Python, c#



ASCII :- American Standard Code For Information Interchange.

A = 65
B = 66
C = 67
D = 68
E = 69
F = 70
G = 71
H = 72

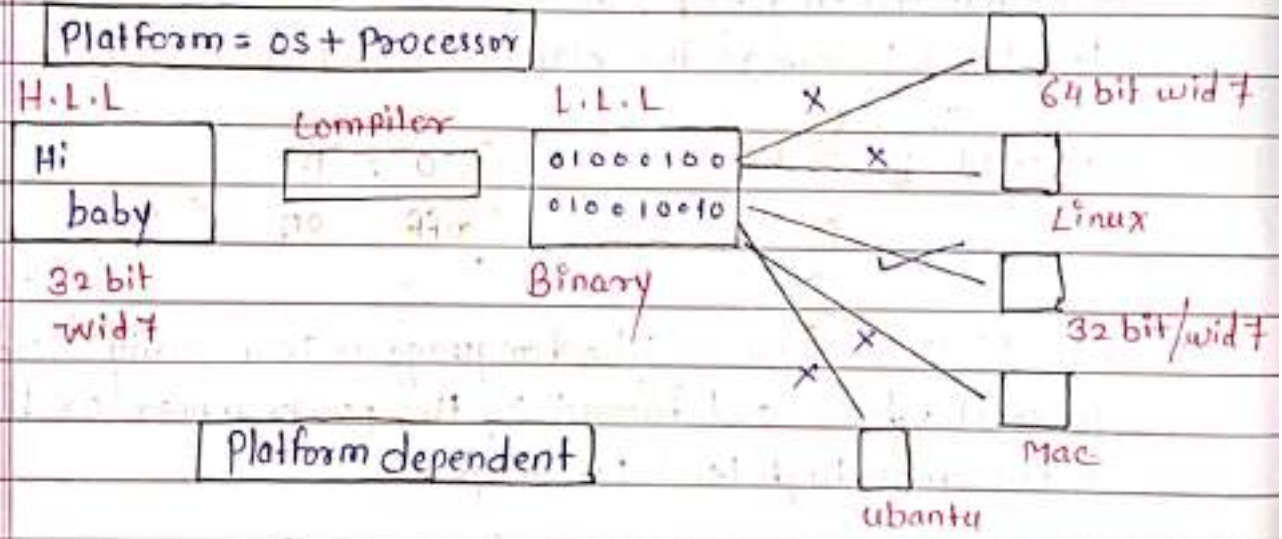
Binary 0 1

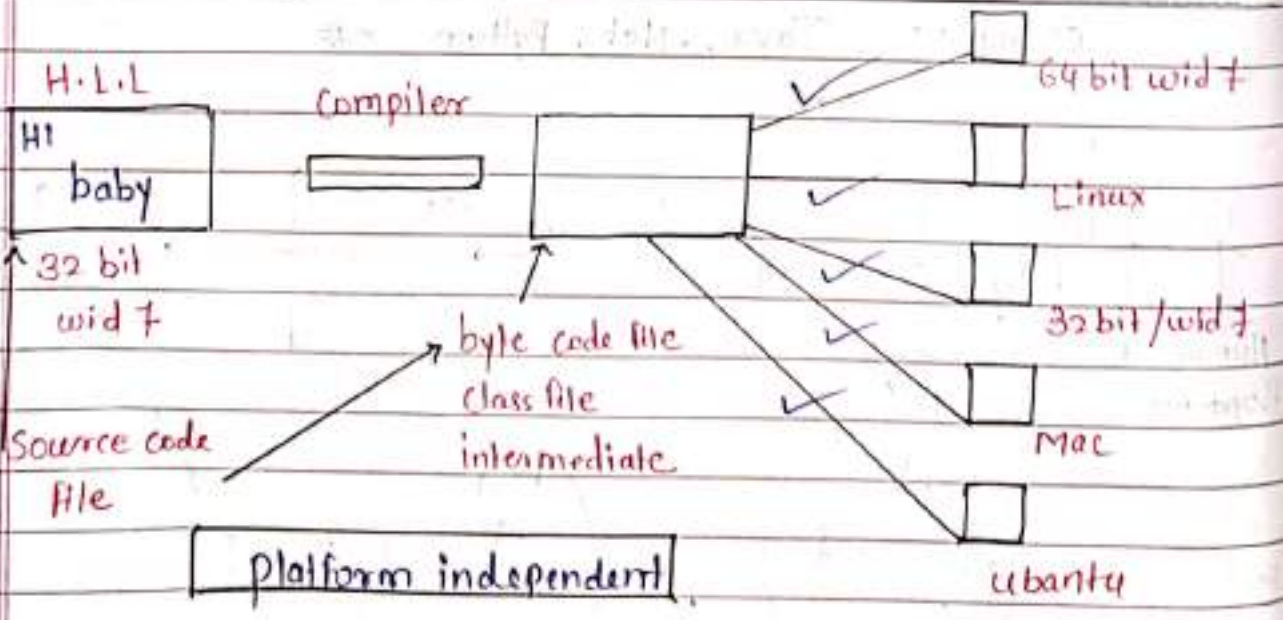
2	72	0
2	36	0
2	18	0
2	9	1
2	4	0
2	2	0
	1	1

R ↓ L

* Platform dependent :- If we write program by using one platform we can execute in only in the same platform but we can not execute in different platform. is known as platform dependent, example :- C

* Platform Independent :- If we write the program by using one platform we can execute in any platform is known as platform Independent is known as- example :- Java





20/10/2022
Thursday

Page No.
Date

I.M.P

(Q) * Explain Why Java is Platform Independent ?

→ Becuse in Java source code file is Converted into byte code file which can be execute by any Platform which has JVM

I.M.P

(Q) * Difference Between JVM, JDK, JRE ?

→ ① JVM :-

i) it is stand for Java Virtual machine
ii) JVM takes byte code as input and generated binary as output.
iii) JVM is physically does not exists.
iv) JVM is platform dependent but JVM makes Java as platform independent.
v) JVM consist of interpreter and JIT

Interpreter :- it is check and execute the byte code Simultaneously but line by line.
but one thing happened the Interpreter is the more time to execute, that's way Installing on software is known as JIT.

JIT :- 1) it is a used to increase speed of execution.
2) JIT identify some set of program which are Having same function and Compies the at ~~onces~~ once then it is give to interpreter.
3) interpreter directly execute the byte code Becuse JIT already ~~the~~ check and compile the byte code.
4) it is stand for Just in time compiler

Page No.
Date

② JRE :-

1) It is stand for Java Run Time environment
2) it is provides and environment to run Java Program.
3) JRE is a physically exists.
4) JRE consist of JVM and inbuilt libraries → (class file)

③ JDK :-

1) It is stand for Java Development kit
2) It is provide and environment to develop as well as to run Java program.
3) JDK is also physically exists.
4) JDk consist of development tools and JRE.

Diagram of JVM, JDK, JRE

Sundra

Hi baby

Source code

Compiler

* Byte code
* class file
* Intermediate Language file

input

Byte code

JVM

Java P.I ← P.D

JIT

Interpreter

output binary

1) execute
2) one by one
3)
4)
5)

JRE

JVM + Libraries

JDK

(DE) Development Envionment + JRE Java Run Time Environment

* Feature Of Java / Buzz words / Advantage / why

① Simple :- it is easy to readable, understandable and instruct by the programer that's way Java is the Simple Language

② P.ID :- If we write one program any platform we can execute this program any different platforms.
That's way Java is the Platform Independent Language.

③ WORA :- Write Once Run Anywhere.

~~④ High Performance :-~~

④ High Performance :- Becuse of JIT (Just in time) compiler, compilation happend fast inside JVM.

⑤ Dynamic :- Automatically allocated memory and Automatically deallocated memory
for example :- Automatically add memory and Automatically remove memory.

⑥ Robust :- Because it check error two times.
1st it is Compiler time and
2nd it is Runtime.

(7) Secured :- Because of JRE only Source code is execute.

(8) OOPL :- Object Oriented programing language.

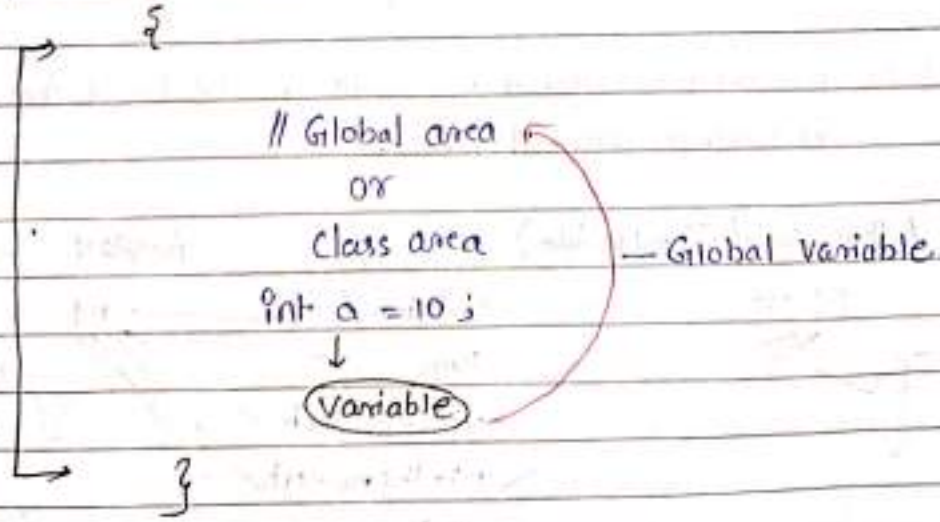
(9) Multi-Threaded Language :- In Java we write one program, which can execute multiple task.

(10) Free and open :- It is Source is open and free to use Software and easily available.

*** types of Variable
1. Local Variable
2. global Variable

1. Global Variable :- Variable create inside Global area is known as Global Variable.

example :-



② Local Area :- Variable create inside Local Area is known as Local Area. Variable.

example :-

```
Class P2
{
    Public static void main (String [] args)
    {
        // Local Area
        int a = 10;
    }
}
```

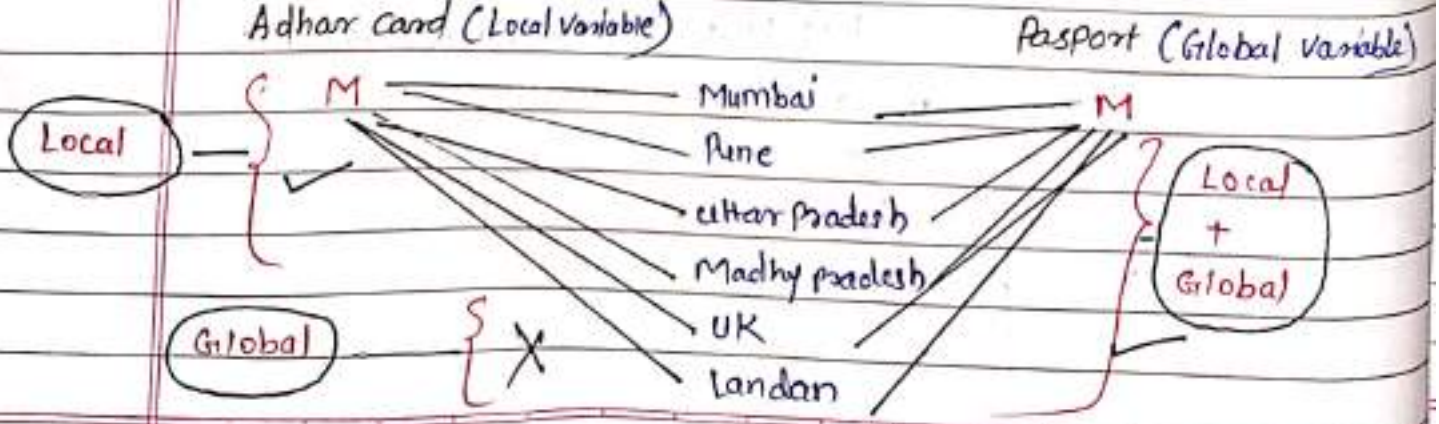
main block

a → Variable → Local Variable

Rules to used Local Variable

1) Local variable are not assigned with default value.
2) we can not use Local variable without initialization
3) we can not create two variable with same name.
4) we can not use Local variable inside global Area.
5) we can use Local variable only inside Local Area.

Note :- we can use Global variable inside Local Area as well as Global Area.

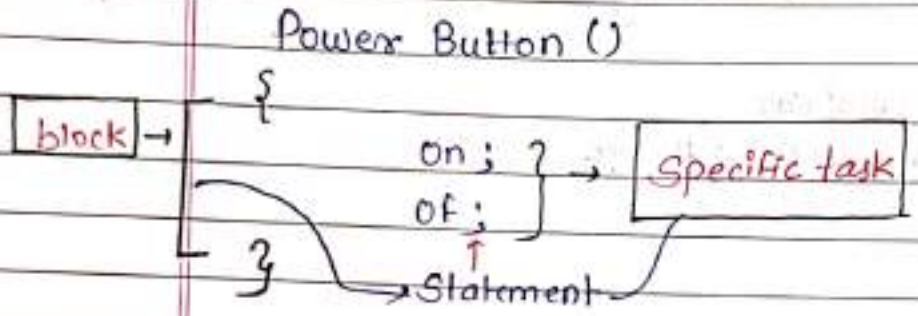


```
class P1
{
    Public static void main (String [] args)
    {
        int a ; // Local var
        System.out.println(a) ; // CTE
    }
}

class P2
{
    public static void main (String [] args)
    {
        int a = 10 ; // Local var
        double a = 20.9 ; // CTE
    }
}

class P3
{
    public static void main (String [] args)
    {
        int a = 10 ; // Local var
    }
        System.out.println(a) ; // CTE
    }
```

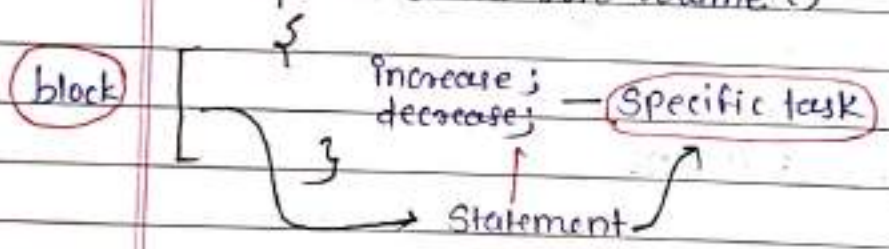
→ output :- Error

* Methods → (Functions) → ( )

→ A block of statements used to perform specific task



Q] Create the Method for Volume button



Note :- * Methods gets executed only when it is called
* one Methods when we call any number of times.
* we can create any number of method for a class inside global Area.
* we cannot create method inside another method.

```
class P2
{
   pubic static void ~~main~~ sheela ()
   {
    System.out.println (" hi bady ...");
   }
    public static void main (String [] args)
   {
      System.out.println (" main method begins ");
      Sheela ();
      System.out.println (" main method ends ");
   }
```

output :-

Main method begins
Hi bady
Main method ends

25/10/2022
Tuesday

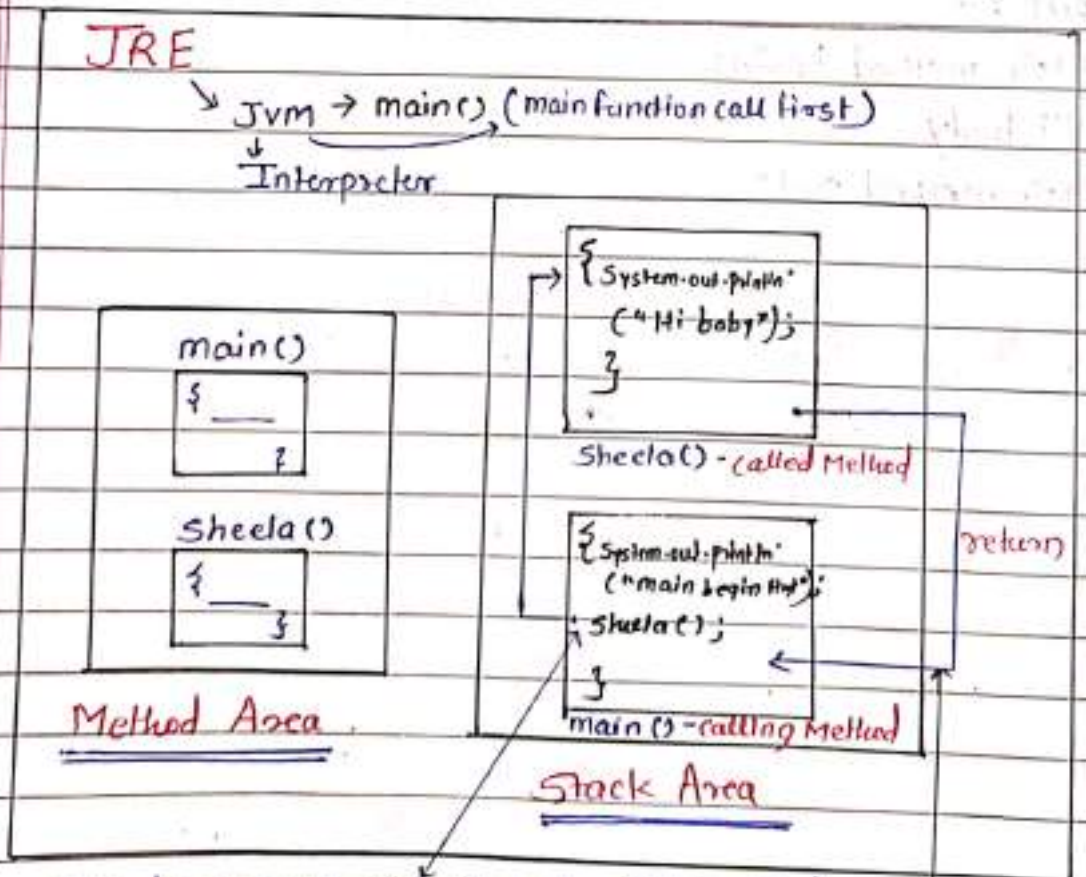
Page No.
Date

* In Java four types of Area.
(1) class static area
(2) Method area
(3) Heap area
(4) Stack area

1) **Method area** :- it is used to store all methods is known as Method Area.

2) **Stack area** :- it is used to execute all Java programs is known as stack area.

Fig 1.



Methods gets execute only when it is called
Automatically to add memory and Automatically to remove memory after it resumes where it stops.

Page No.
Date

Q) Create a class consist of add method to perform addition of two Numbers.

→

```
class P2
{
    public static void add ()
{
    int a = 10;
    int b = 20;
    int c = a+b;
      System.out.println (c);
}
   public static void main (String [] args) // calling Method
    {
      System.out.println ("main method begins");
      add ();
    }
}
```

is not mandatory

Page No.
Date

Q] Create a calculator which Consist of 4 Method to perform.
- Addition of 4 Numbers
- Subtraction of 5 Numbers
- Multiplication of 6 Numbers
- divition of 2 Numbers.

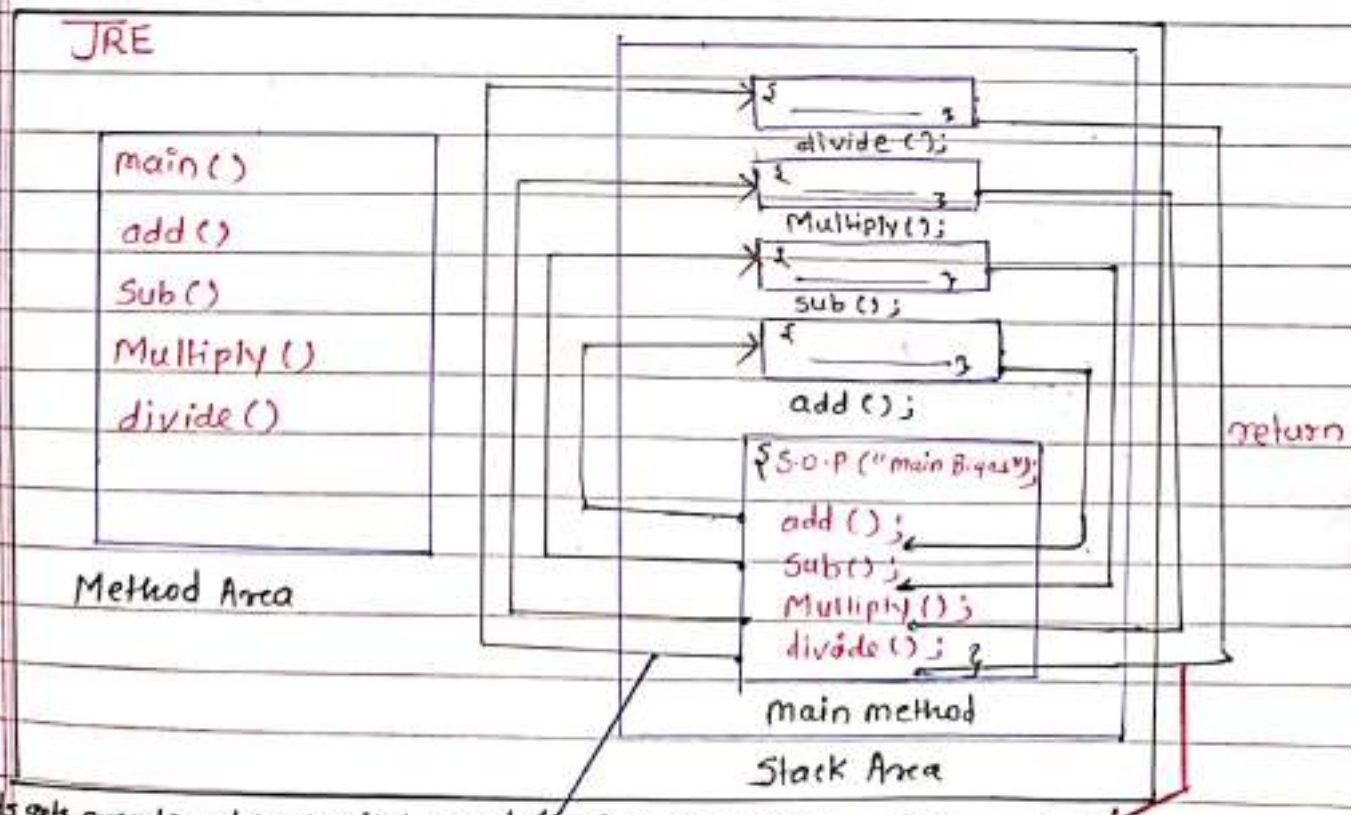
→

```
Class P1
{
  Public Static Void add ()
{
  int a=10, b=20, c=30, d=40;
  int e=a+b+c+d;
 System.out.println(e);
}
  Public Static Void Sub()
{
 int a=10, b=20, c=30, d=40, f=50;
 int e= a-b-c-d-f;
 System.out.println(e);
}
  public Static Void mulliply ()
{
   int a=10, b=2, c=4, d=12, e=14, f=11;
   int G= a*b*c*d*f*e;
  System.out.println(G);
}
  Public Static Void divide ()
{
 int a=100;
 int b=20;
 int c= a/b;
System.out.println(c);
}
```

```
public ~~satic~~ static void main (String [] args)
{
  System.out.println ("Karan");
  add ();
  Sub ();
  multiply ();
  divide ();
}
}
```

Output :- Karan
100
÷130
147, 840
5

Fig 2.



26/10/2022
Wednesday

Page No.
Date

* Types of Methods
1) no argument- Method
2) Parameterized Method

1) no argument Method :- The method which does not have Any formal argument is called as no argument Method.

2) Parameterized Method :- The Method which has formal argument is called as parameterized Method.

* Formal argument (FA) :- Variable create inside the Method is known as Formal argument (FA).

* Actual argument (AA) :- data passed inside method calling Statement is known as Actual argument (AA).

Variable create inside the method → FA

0, 1, 4, 3, 7, 8, 9, ... n

① Public Static Void add()
{
int a = 40;
int b = 50;
int c = a+b;
System.out.println(c);
}
Public Static Void main(String[]args)
{
add();
}
}

output:- 90

② Public Static void add(int a, int b)
{
int c = a+b; (10+20)
System.out.println(c);
}
Public Static Void main (String[]args)
{
add(10,20);
add(30,20);
add(40,40);
add(50,50);
}
}

output:- 30
50
80
100

a 10 int
b 20 int
c 30 int

AA → data passed inside method calling statement

Methods
- no argument
- parameterized

Page No.
Date

Q] Create class consist of multiply method to perform multiplication of 4 Numbers.

→

```
class P1
{
  Public static void multiply (Int a, int b, int c, int d)
  {
    Int Sum = a*b*c*d;
    System.out.println (Sum);
  }
    Public static void main (String [] args)
  {
    Multiply (10, 2, 3, 4);
  }

}
```

Output :- 240

→

```
class P2
{
  Public static void multiply (double a, double b, double c, double d)
{
    double Sum = a*b*c*d;
  System.out.println (Sum);
}
  Public static void main (String [] args)
{
  multiply (10.2, 2.4, 3.5, 4.5);

}

}
```

output :-

I.M.P

Note:- return :- 1) return is the control transfer statement,
2) return statement is use to return maximum one data, from called method to calling method.

Rule :- 1) one method can have only one return statement.
2) return statement should always be the last statement in method.

→

```
class P2
{
    Public static void int add (int a, int b)
{
    int c = a+b;
    return c;
}
    public static void main (String[] args)
{
    System.out.println ("mB");
    System.out.println (add (10,20));
    System.out.println ("end");
}
}
```

Output :- mB
30
end

Page No.
Date

27/01/2022
Thursday

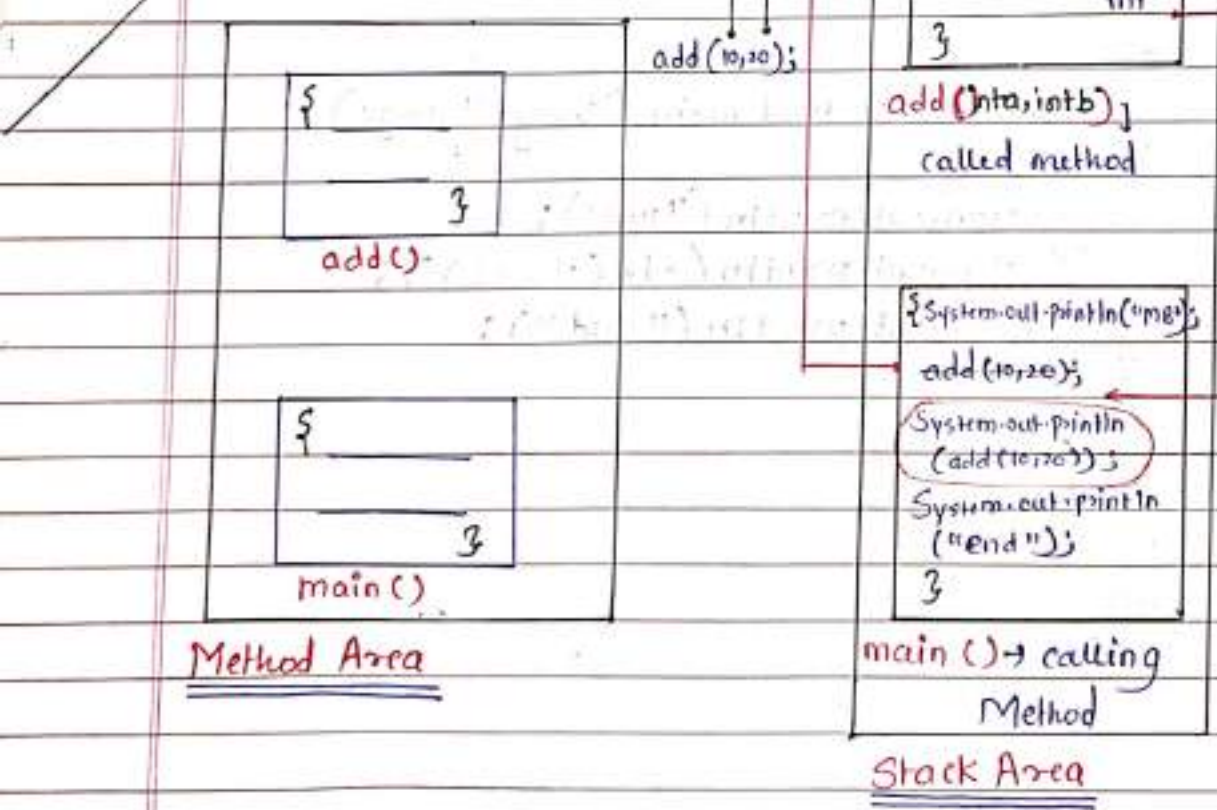
(Return type of method)

Public static void add (int a, int b)

Public static int add (int a, int b)

```
{
    int c = a+b;
    return c;
}
```

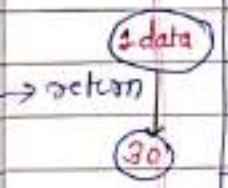
JRE



* Void

Note :

example :



example1 →

```
{
pub
{
in
}
pub
{
}
```

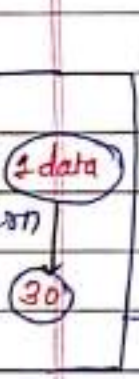
7/2/2022
Thursday

Page No.
Date

* **Void** :- 1) void is Return type of statemen method
2) void Represent nothing
3) if any method is having void as Return type which means the methods is not Returning any data.

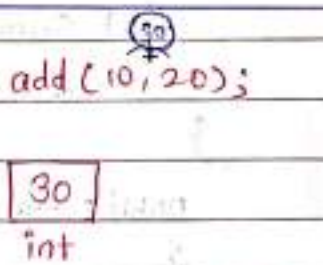
Note :- 1) if any method is having void as Return type then it is not mandatory to write Return Statement.
2) if any method is having other than void as Return type then it is mandatory to write Return Statement.

example :- `int a = 30;`



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

`int a = add(10,20);`



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

example 1) :-

```
class P2
{
 public static void int add(int a, int b)
 {
  int c = a+b;
  return c;
 }
 Public static void main(String[] args)
 {
  System.out.println("MB");
  int a = add(10,+20);
  System.out.println(a);
  System.out.println("end");
 }
}
```

output :- MB
30
end

Q] Create a class consist of Subtract Method to Peaform
- Subtraction of 3 Numbers.
- Subtraction of 4 Numbers.
- return data from each method
- Store the data which is Return the print

→

```
class P2
{ public static int sub (int a, int b, int c)
  {
     int Sum = a-b-c;
     return sum;
  }
  Public static int sub (int a, int b, int c, int d)
  {
     int Sum = a-b-c-d;
     return sum;
  }
  Public static void main (String [] args)
  {
     int a = sub (100, 80, 10);
     int b = sub (100, 20, 30, 10);
    System.out.println (a);
    System.out.println (b);
  }
}
```

output:- 10
40

I.M.P

* Method Overloading :-

Class is having more than one method with same name but different formal argument.
either differening length or differnting data type.

Example:- Q] Create Calculator with addition of 2 Number, 3 Number, 4 Number, 5 Numbers, return data for each method, store the return data and print.

→

```
class P2
{
    public static int add (int a, int b)
    {
        int sum = a+b;
        return sum;
    }
    public static int add (int a, int b, int c)
    {
        int sum = a+b+c;
        return sum;
    }
    public static int add (int a, int b, int c, int d)
    {
        int sum = a+b+c+d;
        return sum;
    }
    public static int add (int a, int b, int c, int d), int e)
    {
        int sum = a+b+c+d+e;
        return sum;
    }
```

```
public static void main (String[] args)
{
    int a = add (10,20);
    int b = add ( 10,20,30);
    int c = add (10,20,30,40);
    int d = add (10,20,30,40,50);

    System.out.println(a);
    System.out.println(b);
    System.out.println(c);
    System.out.println(d);
}
}
```

Output:- 30
60
100
150

28/10/2022
Friday

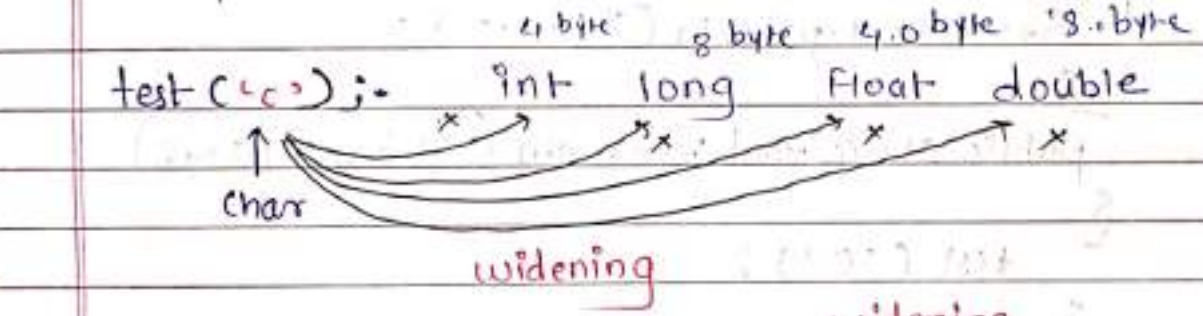
Page No.
Date

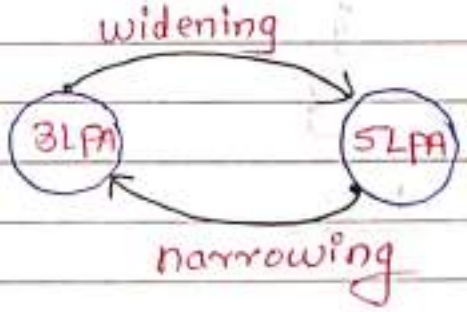
* Rules to call Parameterized Method.

(1) the length of Formal Argument and Actual Argument be same.

(2) The coresponding datatype of formal Argument and Actual Argument should be same.

- If same datatype is not available then compiler will try to do widening
- Supose widening is also not posible then we get Compile time error.





* Widening :- Smaller data convert into bigger data this process called as widening

Example :- Widening

```
class P1
{
  public static void test (double a)
  {
    System.out.println ("double");
  }
  public static void test (long a)
  {
    System.out.println ("long");
  }
  public static void main (String[] args)
  {
    test ('c');
  }
}
```

Output :- long

* **Method Recursion** :- method is calling it self is known as method recursion.

or

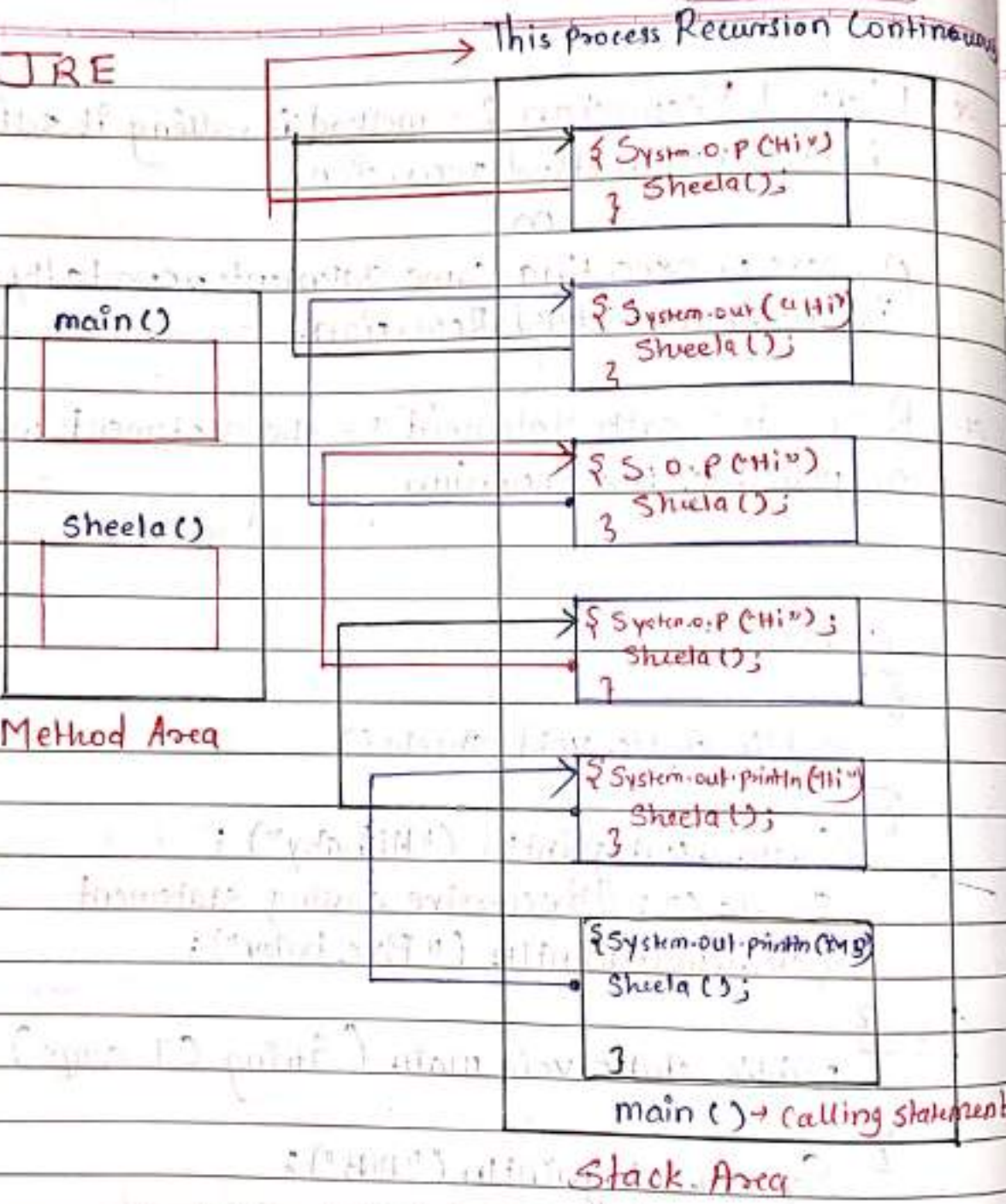
process of executing same statement repeatedly is known as ~~Method~~ Recursion.

* **Recursive call statement** :- the statement which is responsible for Recursion

```
class P5
{
   public static void sheela()
{
   System.out.println("Hi baby");
    Sheela(); //Recursive calling statement
   System.out.println("Bye baby");
}
   public static void main(String[] args)
{
   System.out.println("MB");
    Sheela(); // Method calling statement
    System.out.println("ME");
}
}
```

Page No.
Date

Output:-



output:- MB
Hi
Hi
Hi
Hi
Hi
Hi
⋮
Stackoverflow error

Page No.
Date

* Fundamental of OOPS

members
↓
Var | method

Global member

Static members
1) Static Variable
2) Static Method
3) Static Block/
4) Static Initializer Block (SIB)

non static members
1) non-Static Variable
2) non-static method
3) non-static Block
4) Instance Initializer Block (IIB)
5) Constructor

example :-
```
class P1
{
    // Global Area
    Static int a = 10;
        ↓
    Variable

    Public Static void Savita()
        ↓
    Method
    {
    }
}
```

29/10/2022
Saturday

Page No.
Date

* Global Members :-

→ Any members declared inside global Area are known as Global members.

* Static Members :-

→ Any members declared inside global Area prefixd with Static keyword is known as static member.

* Static Variable :-

→ Any Global variable prefixd with static keyword is known as static variable.

* Static Method :-

→ Any method declared inside Global Area Prefixd with Static keyword is known as static method.

example :-

```
class P1
{
    static int a ; // Global Variable (Global variable have defaul
    public static void main (String [] args)
    {
        System.out.println(a);
    }
}
```

output :- 0

(Note :- Global variable have default value that's way Global variable can declared without ~~Initiliation~~ Initialization.)

* Difference Between Local Variable And Global Variable

(*)	Local Variable	global variable
(1)	Local The variable declerd inside Local Area is known as Local variable.	(1) The variable decleard inside Global Area is known as Global Variable.
(2)	Local variable can not decleard without Initialization.	(2) global variable can decleard without Initialization.
(3)	local variable don't have defaut value	(3) global variable ~~can~~ have default value.
(4)	local variable decleard in Local Area only	(4) global variable decleard in global Area as well as local Area.
(5)	Local variables are Stored inside Stack Area.	(5) global variable are Store inside class static Area.

* **global variable**

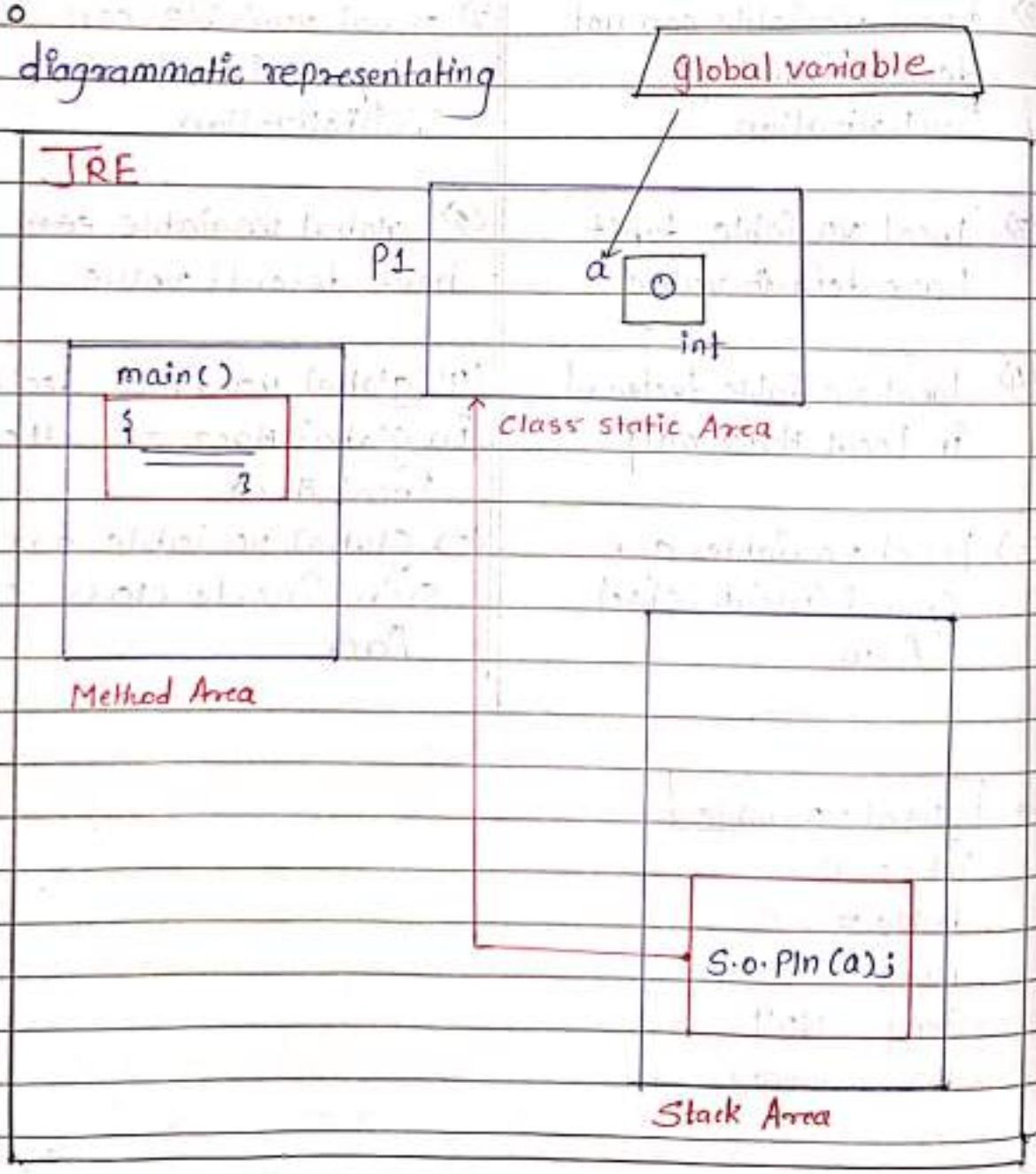
int = 0
double = 0.0
float = 0.0
String = Null
char = empty

Page No.
Date

Forexample:-
```
class P1
{
    static int a; // Global variable have default value
    public static void main (String [] args)
    {
        System.out.println (a);
    }
}
```

Output :- 0

diagrammatic representating



Output :- 0 (int → default value is 0)

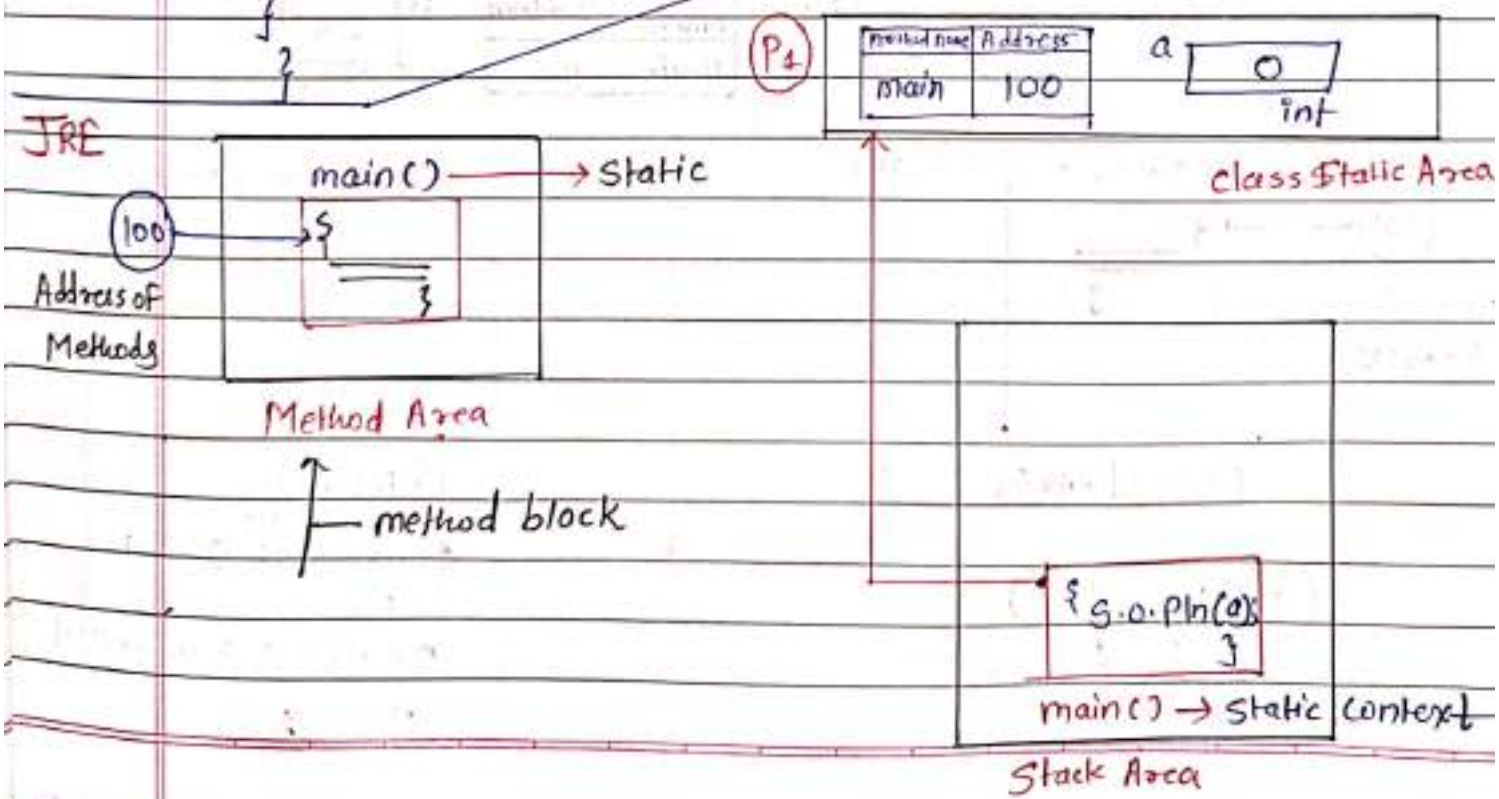
Page No.
Date

* Static Method :-

→ Any method declared inside global Area prefixed with Static keyword is known as static method.

Note :- ① all the Method block are stored inside Method Area
② Address of static method will be store inside class static Area.
③ class static Area is use to store all the static members
④ Static method block is also known as static context
⑤ every static context will be pointing towards class static Area.

example :-
```
class P1
{
    Static int a; // Global Variable / static Variable
    public Static void main (String [] ~~mat~~ args) // Static Context
    {
        System.out.println (a);
    }
}
```



Page No.
Date

Q] * Can we create Local and global variable with same name.

→ Yes

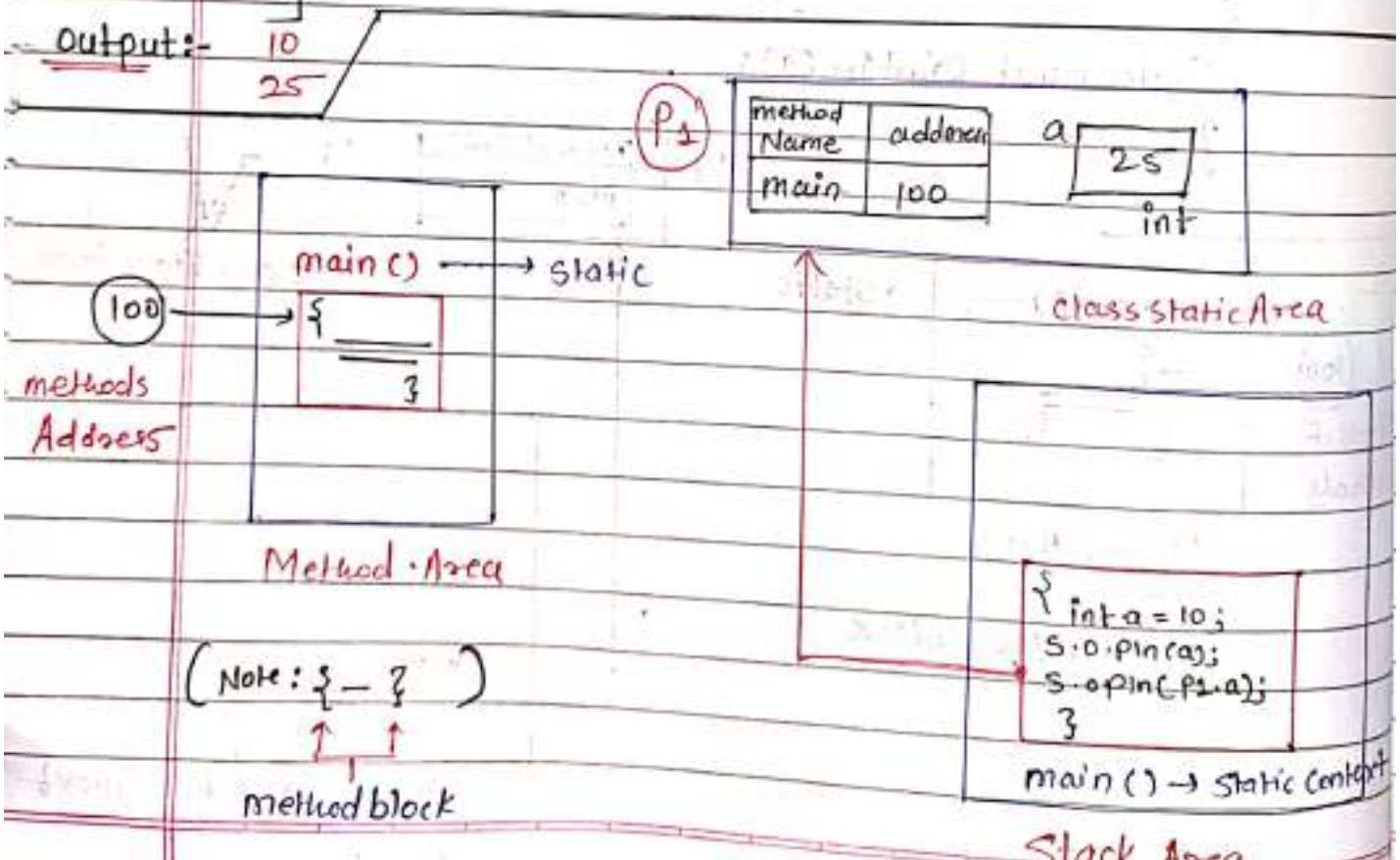
1) Always high priority given to local variable.
2) If we want to use static variable then we must considerd class name.

Syntax :- class Name . variable name ;

example:-

```
class P1
{
    static int a = 25; // Global variable / (Static variable)
    public static void main (String [] args) // Static Context
    {
        int a = 10; // Local variable
        System.out.println (a);
        System.out.println (P1.a); // class Name. variable name;
    }
}
```

output:-
10
25



* Static Block :- it is block which executes before main method but only once.

characterstics of static block

- Static block does not have any name.
- Static block does not have any return type.
- it ~~doetnt~~ does not have any formal Argument
- it gets executed implicitly only once.
- Static block can not be called explicitly.

```
class P2
{
  static
  {
    System.out.println ("welcome to Instragram");
  }
  public static void main (String [] args)
  {
    System.out.println ("Enter Username");
    System.out.println ("Enter password");
  }
    static
    {
      System.out.println ("From meta");
    }
    }
```

Output:- Welcome to Instagram
From meta
Enter Username
Enter Password

Q] how to call any method before main method ?

→ by using static block

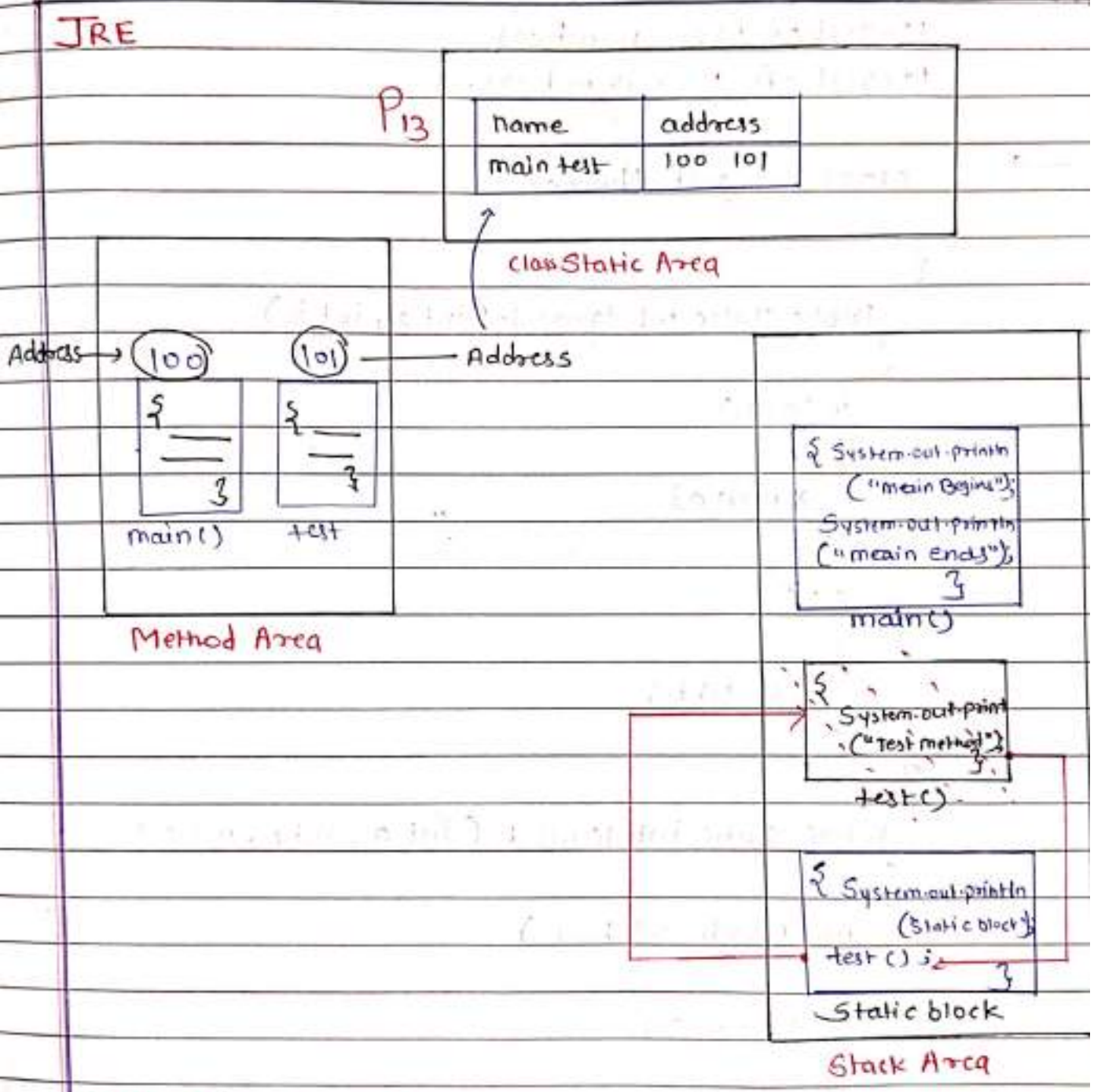
For example:-

```
class P13
{
  static
  {
    System.out.println("Static Block");
    test(); // Method calling statement
  }
  public static void main(String[] args)
  {
    System.out.println("main Begins");
    System.out.println("main ends");
  }
  public static void test()
  {
    System.out.println("Test method");
  }
}
```

Output:-

```
Static Block
Test method
main Begins
main ends
```

diagrammatic representating



Output :- Static Block
Test method
main Begins
main ends

Q] write a Java program to find largest of
two numbers
largest of tree numbers
largest of four numbers.

→

```
class Largest_Number
{
  Public static int largest (int a, int b)
  {
    if (a>b)
    {
      return a;
    }
    else
    {
      return b;
    }
  }
  public static int largest ( int a, int b; int c)
  {
    if (a>b && b>c)
    {
      return a;
    }
    else if (b>c)
    {
      return b;
    }
    else
    {
      return c;
    }
```

```
}
public static int largest (int a, int b, int c, int d)
{
  if (a>b && b>c && c>d)
  {
    return a;
  }
  else if (b>c)
  {
    return b;
  }
  else if (c>d)
  {
    return c;
  }
  else
  {
    return d;
  }
}
public static void main (String[] args)
{
  int i = largest (10,20);
  System.out.println (i);
  int m = largest (10,20,30);
  System.out.println (m);
  int n = largest (10,20,30,40);
  System.out.println (n);
}
}
```

Page No.
Date

Q)

```
class Task
{
    Static String a;   // Static Variable
    Static boolean b;

    public static void sheela()
    {
        System.out.println ("Sheela B");
        System.out.println (a);
        System.out.println (b);
        a = "leela";
        b = true;
        System.out.println ("Sheela E");
    }
    static
    {
        System.out.println ("SIB-1");
    }
    public static void main (String [] args)
    {
        System.out.println ("MB");
        sheela ();
        System.out.println (a);
        System.out.println (b);
        System.out.println ("ME");
    }
```

Page No.
Date

```
Static
{
    System.out.println("SIB-2");
    Sheela();
}
}
```

Output :- SIB1
SIB2
Sheela B
Null
False
Sheela E
MB
Sheela B
leela
True
Sheela E
leela
True
~~MB~~
ME

static → method
↘ var
↘ block

Page No.
Date

class Static Area

Task

name	address
main()	100
sheela()	200

leela
a Null String
b False boolean
True

```
{ System.o.println("SheelaB");
  System.o.println(a);
  System.o.println(b);
  a = "leela";
  b = true;
  System.o.println("SheelaE");
}
```
Sheela() → static context

```
{ System.o.println("MB");
  Sheela();
  System.o.println(a);
  System.o.println(b);
  System.o.println("ME");
}
```
main() → static context

```
{ System.out.println("SheelaB");
  System.out.println(a);
  System.out.println(b);
  a = "leela";
  b = true;
  System.out.println("SheelaE");
}
```
Sheela() → static context

```
{ System.out.println("SIB-2");
  Sheela();
}
```
Static block 2

```
{ System.out.println("SIB-1");
}
```
Static block 1

100 Address

200 Address

main() Sheela()

Method Area

Stack Area.

output:-

SIB1	leela
SIB2	True
SheelaB	ME
Null	
False	
SheelaE	
MB	
SheelaB	
leela	
True	
SheelaE	

Page No.
Date

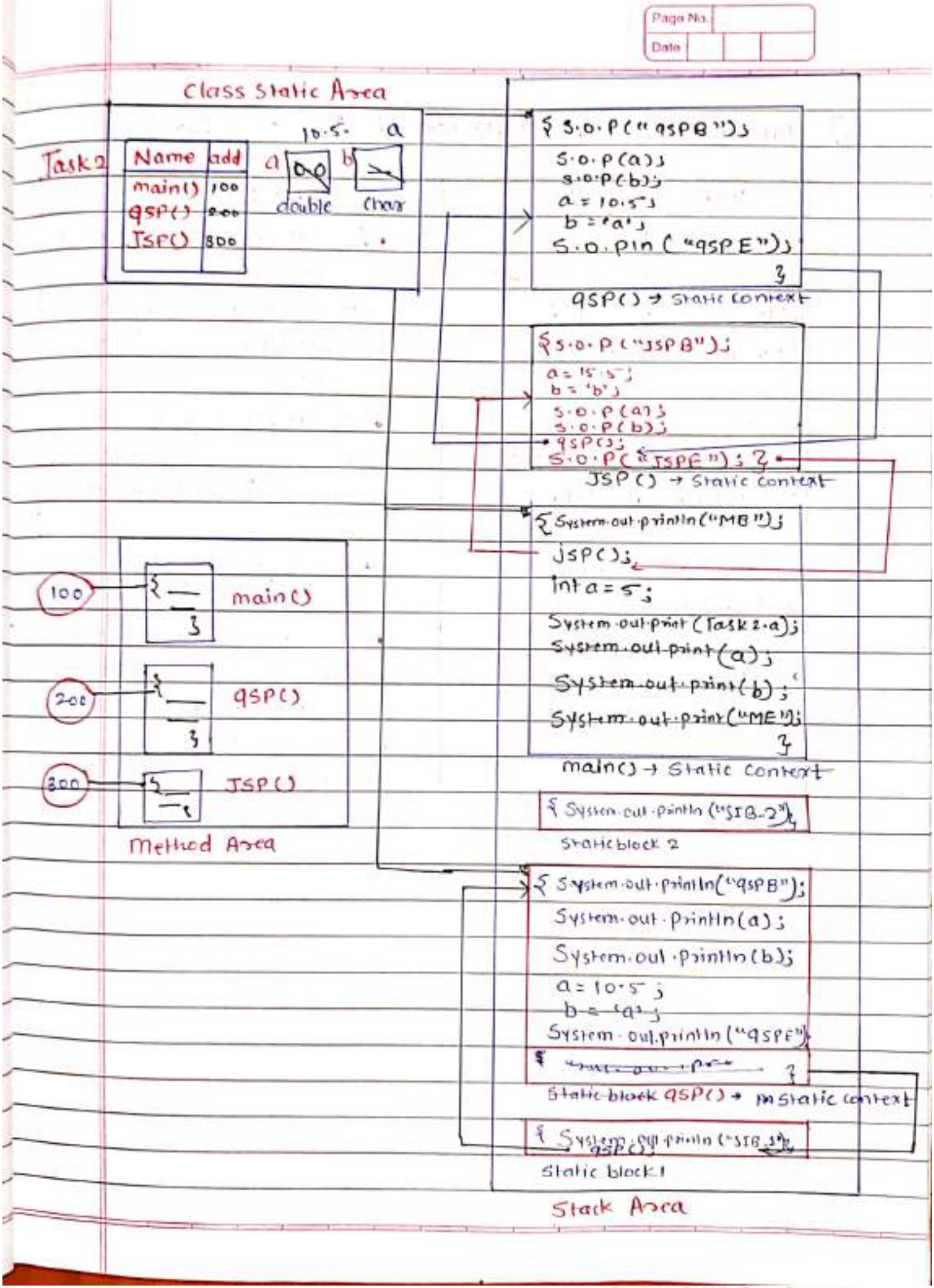
Q] class Task2

```
{
    static double a;
    static char b;
    public static void qsp()
    {
        System.out.println("qsp B");
        System.out.println(a);
        System.out.println(b);
        a = 10.5;
        b = 'a';
    System.out.println("qsp E");
    }
    public static void Jsp()
    {
        System.out.println("Jsp B");
        a = 15.5;
        b = 'b';
        System.out.println(a);
        System.out.println(b);
        qsp();
        System.out.println("Jsp E");
    }
    static
    {
        System.out.println("SIB-1");
        qsp();
    }
```

```
public static void main (String [] args)
{
    System.out.println ("MB");
    jsp ();
    int a = 5;
    System.out.println (Task2.a);
    System.out.println (a);
    System.out.println (b);
    System.out.println ("ME");
}
Static
{
    System.out.println ("SIB-2");
}
}
```

Output:-

SIB_1	10.5
qSPB	5
0.0	a
—	ME
qSPE	
SIB_2	
MB	
JSP.B	
15.5	
b	
qSPB	
15.5	
b	
qSPE	
JSPE	



Prgm No.
Date

Q] difference between Static Method And Static block

→

Static Method	Static block
• Any method declared inside Global Area prefixed with static keyword is known as Static Method.	• Any static block declared inside Global Area ~~or class Area~~ prefixd with Static keyword is known Static' block
• static method have name, Formal Arrgument, and return type data	• Static block don't have name, formal Argumen and return type data.
• Method get execute only when it is call	• block execute directly in stack Area.
• Static method execute after main method	• block it is execute Before main method but only once.
	• it is also called as static Initializer block

Non-Static Members

Page No.
Date

* **Non-Static members :-**
→ ~~any member declared inside global Area not prefixd with static keyword is known as non-static members.~~ object

* **Non-Static Variable :-** ~~any~~ global Variable not prefixd with Static keyword is known as non-static Variable.

* **non-Static method :-** any method declared inside global Area not prefixd with Static keyword is known as non-static method.

* **Non-Static block :-** any block decleard inside global Area not prefixd with static keyword is known as Non-Static block
It is also called as Instance Initializer Block.

Non-static members

1. non-static variable
2. non-static method
3. non-static block / Instance Initializer Block

* **Non-Static members :-** any member declared inside global Area not prefixed with static keyword is known as non-static members.

Page No.
Date

6] Difference between static Variable and non-static Variable

Static Variable	Non-static Variable
1) A variable declared inside global Area prefixed with Static keyword is known as Static Variable.	1) A variable declared inside a global Area but it's not prefixed with static keyword is known as non-static ~~Keyword~~ Variable.
2) Static Variable have same memory for every object.	2) Non-static Variable will have different memory for each every object.
3) Static variable are stored inside class Static Area	3) non-static Variable are stored inside object.
4) static Variable will have 3 ways to execute 1) Directly 2) with the help of classname 3) object Referrence.	4) non-static Variable It is only ~~one~~ way to execute 1) Object Reference classname ref = new classname(); (ref.a);
5) static variable ~~have~~ inside object creation is not mandatory	5) non-static Variable have object creation is mandatory.

Q] Difference between static method and non-static method.

→

Static method	Non-Static method
1) A method declard inside global Area but it prefixed with static keyword is known as static method	1) A method declared inside global Area but it's not prefixed with static keyword is known as Non-Static method
2) A static method is also known as static context	2) A non-static method is also known as non-static context.
3) It is stored inside class static area	3) It is stored inside object
4) Every static context will be pointing toword to class static Area.	4) Every non-static context will be pointing toward object.
5) Static method there are 3 ways 1) directly 2) with help of classname 3) object reference	5) non static there are only one way it's 1) Object Reference.
6) Object creation is not mandatory.	6) Non-static have object creation is mandatory.
7) Static context will don't have "this"	7) non-static context will have default "this"

Page No.
Date

Q1 Difference between Static block and Non static block.

Static block	non static block
1) The block which is declared inside global Area prefixed with static keyword is known as static block	1) The block which is declared inside global Area not prefixed with static keyword is known as non-static block.
2) when we have more than one static block they will get executed in top to bottom order	2) when we have multiple no. of non static block then they will get executed in top to bottom order.
3) object creation is not mandatory	3) Object creation is mandatory
4) Static block is a block which is executed before main method.	4) Non-static block is a block which is executed after main method.
5) Static block to execute Some set of statement before main method but only once.	5) non-static block to executed some statements mandatory it is only when we create object but only once for one object
6) static block does not have any name, formal Argument and return datatype	6) non-static also does not have any name, formal Argument and return datatype.

Q] difference between primitive datatype and non primitive datatype.

→

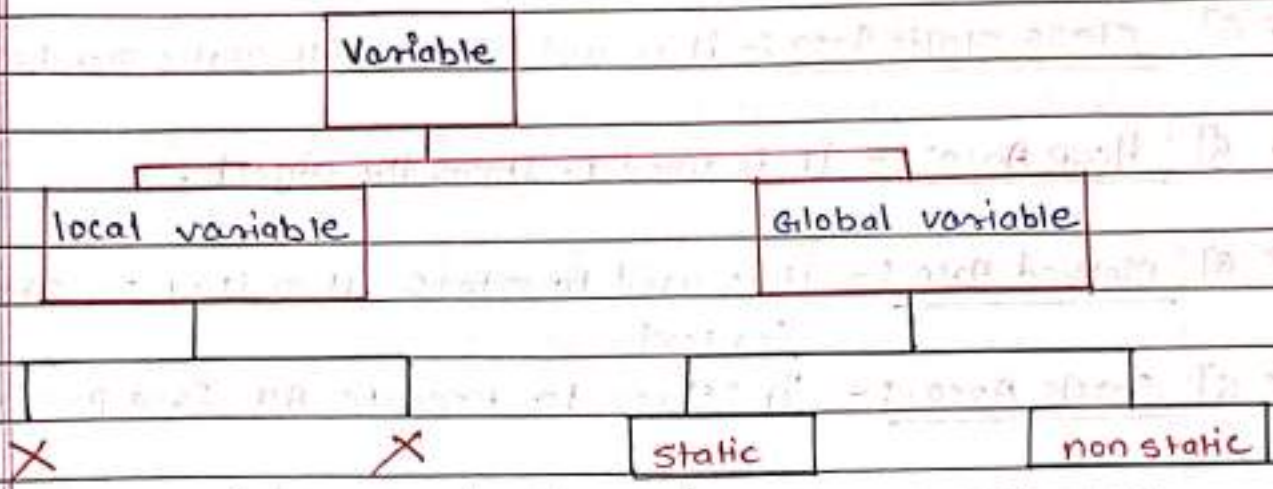
Primitive datatype	non-primitive datatype
1) primitive datatype is also known as inbuilt datatype. example :- class, string	1) non-primitive datatype it is also known as user defined datatype. example :- int, long, double
2) Primitive datatype have exact size example :- int, float, double	2) non-primitive datatype does not have exact size example :- class, string
3) Primitive data structure will contain some value i.e it cannot be null	3) non-premitive data can consist of a null value
4) start with a lowercase latter ex. byte, short, int, long float, double, char, boolean	4) start with a uppercase latter ex. class, String

Q] difference between primitive variable and reference variable

Primitive variable	reference variable
1) A variable create using data with premilive datatype is called as premilive variable	2) A variable create using with non-premilive datatype is called as non-premilive variable.
2) premilive variable are stored to the data	2) non-primitive variable are stored the object address or reference.
3) primitive datatype is also called as primitive variable.	3) non-primitive datatype is also called as reference variable or non-primitive variable.

Q] Difference between Constructor and method

method	Constructor
1) method is a block of Statement used to perform specific task	1) Constructor it is used to store all the non-static members.
2) method have Return type data	2) constructor doesn't have any type of data.
3) method name can be anything	3) Constructor name is similar to class name.
4) method we can execute 3 ways with the help of 1) directly 2) className 3) object Reference	4) Constructor we can execute only one way with the help of 1) ~~Obje~~ a new Unary operator.
5) method is static method and also non-static method	5) Constructor is only non-static member.
6) method is created by programer	6) Constructor is created by compiler



Note :-

* We cannot use non-static member inside static context
  1. Directly
  2. className (with the help of className.a).

* Non-static member will be stored inside object hence it is mandatory to create object.

* Four Area In JAVA *

Page No.
Date

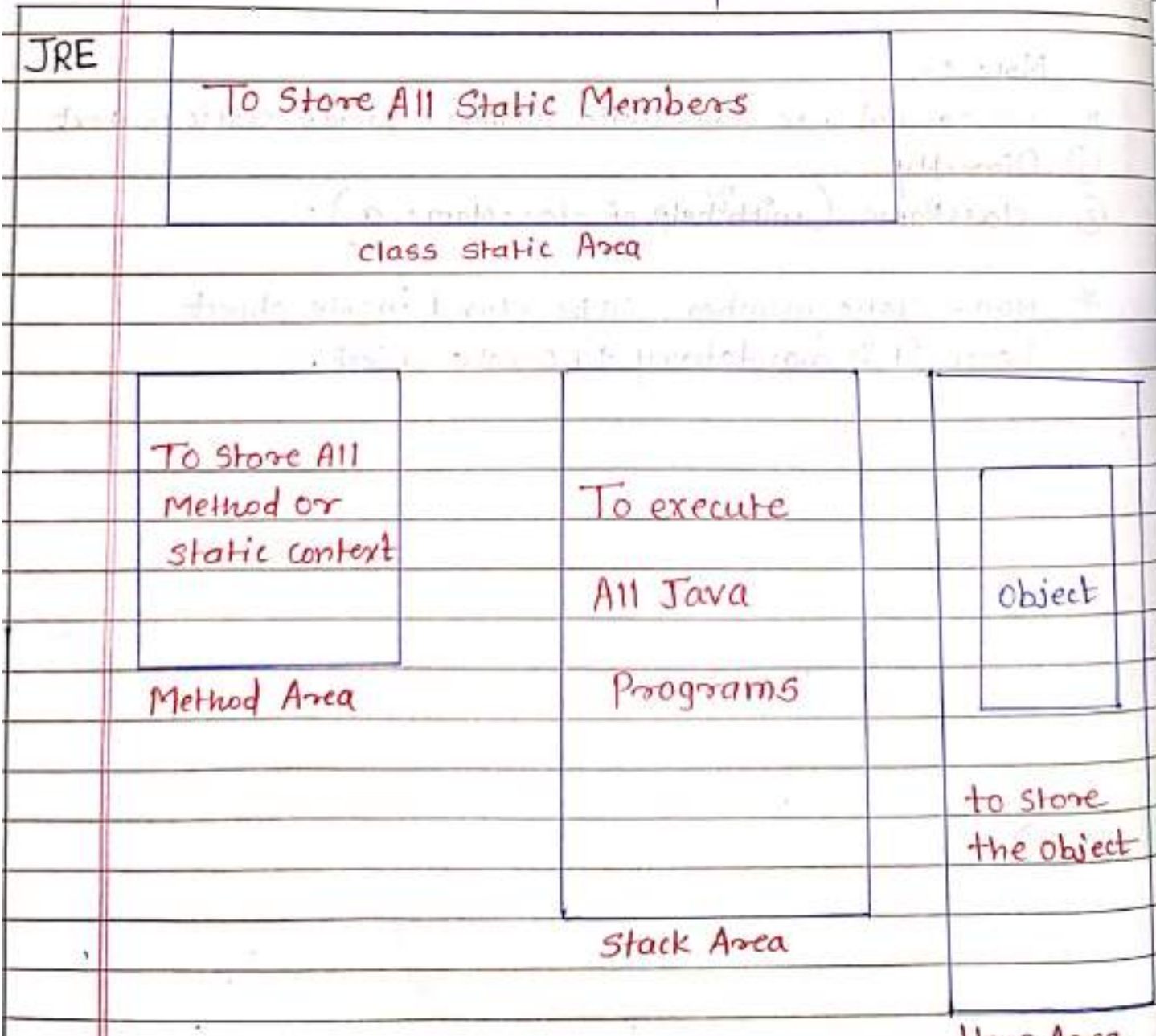
* Q] Class static Area :- it is used to store All Static members

* Q] Heap Area :- it is used to store the object.

* Q] Method Area :- it is used to store all method or Static Context.

* Q] Stack Area :- it is use to execute All Java program

Note :- This four Area Automatically created inside JVM



Q] What is an object?

→ A block of memory created inside heap Area during run time. is known as object.

Q] How to create object?

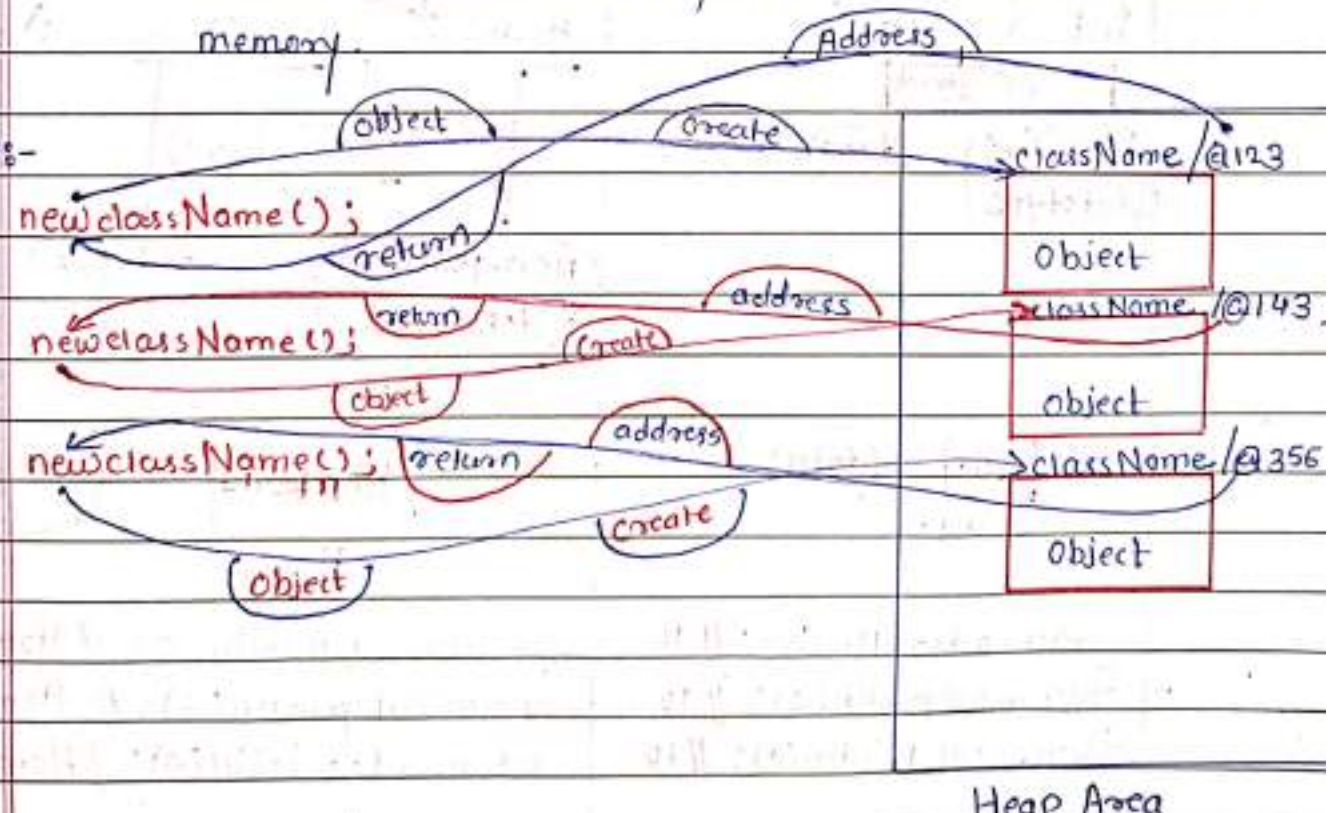
→ Syntax :- new className ();

* New

→ 1) new is keyword
2) new is unary operator
3) new operator create a block of memory inside Heap Area during Run time. and it returns address of an object.

Note :- we can create any numbers of object for a class
Whenver we write New keyword it creates new block of memory.

Example :-

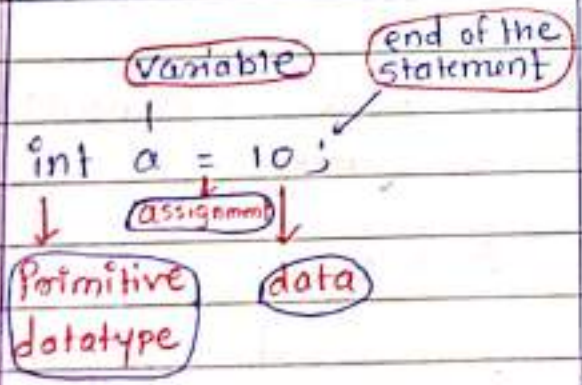


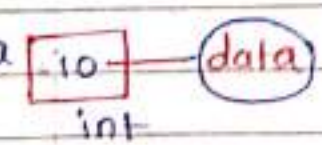
Example:-

```
class Home
{
    public static void main (String [] args)
    {
        System.out.println (new Houme ());
        System.out.println (new Home ());
        System.out.println (new Home ());
    }
}
```

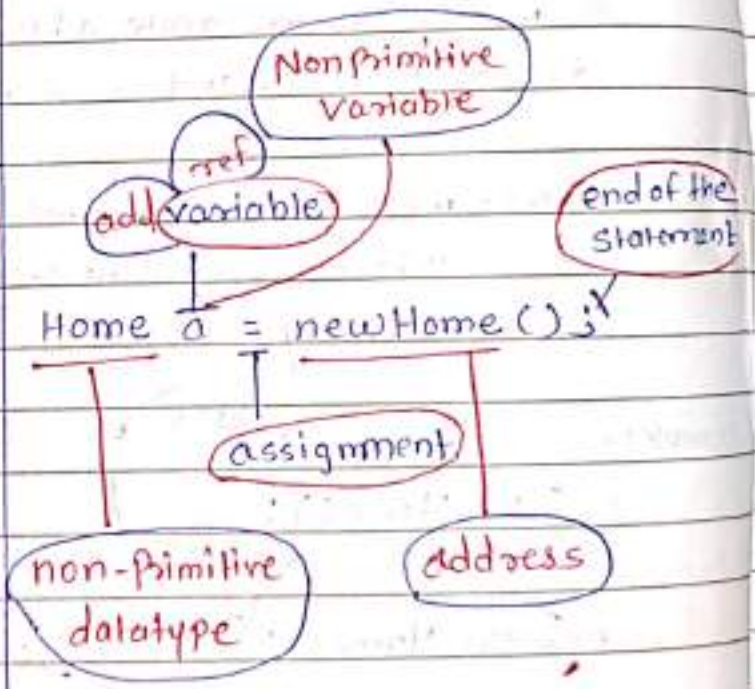
Output:-

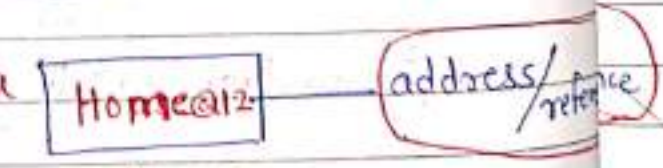
Home @ 123
Home @ 143
Home @ 356






```
System.out.println (a); // 10
System.out.println (a); // 10
System.out.println (a); // 10
```






```
System.out.println (a); // Home@12
System.out.println (a); // Home@12
System.out.println (a); // Home@12
```

Page No.
Date

example:-
```
Class Home
{
    public static void main (String [] args)
    {
        Home ref = new Home ();
        System.out.println(ref);
        System.out.println(ref);
        System.out.println(ref);
    }
}
```

Output:-
```
Home@12
Home@12
Home@12
```

Q] Create bike class.
Create at least 3 bike objects
Store address for each bike then print.

→

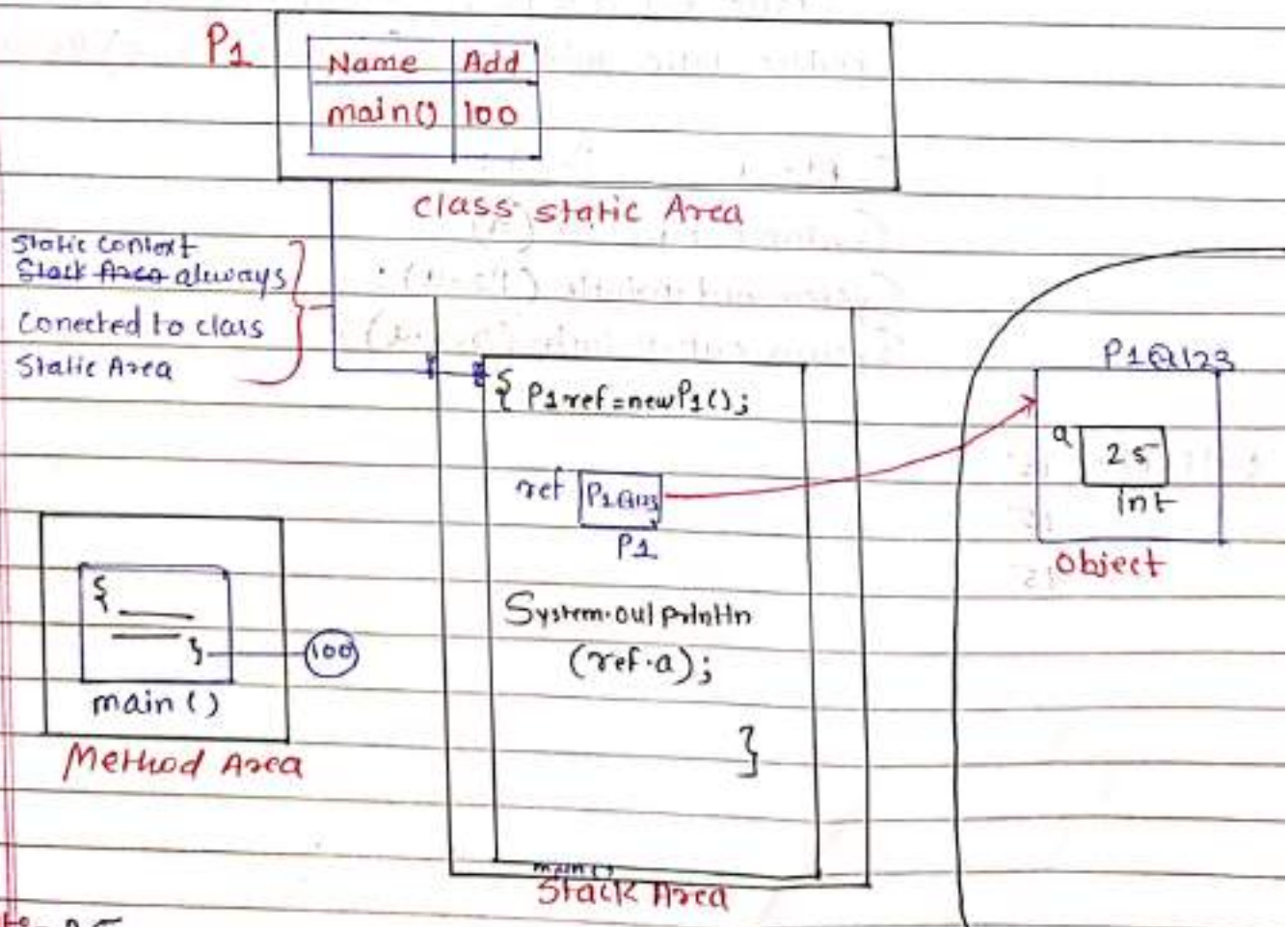
```
class Bike
{
    Public static void main (String [] args)
    {
        Bike ref1 = new Bike ();
        Bike ref2 = new Bike ();
        Bike ref3 = new Bike ();

        System.out.println (ref1);
        System.out.println (ref2);
        System.out.println (ref3);

    }

}
```

Output:-
```
Bike@123d
Bike@143e
Bike@786M
```

n.Pg] How to use non-static members inside static context?

→ with the help of (Object reference)

mple:-
```
class P1
{
    int a = 25; // Global Variable (non-static variable)
    public static void main (String [] args) //static context
    {
        P1 ref = new P1 ();
        System.out.println (ref.a);
    }
}
```



Output:- 25

Imp → Q] How ~~to~~ many ways can we use static members inside static context.

→ with the help of 3 ways

(i) Directly (a);

(ii) className (P2.a);

(iii) objectreference (ref.a);

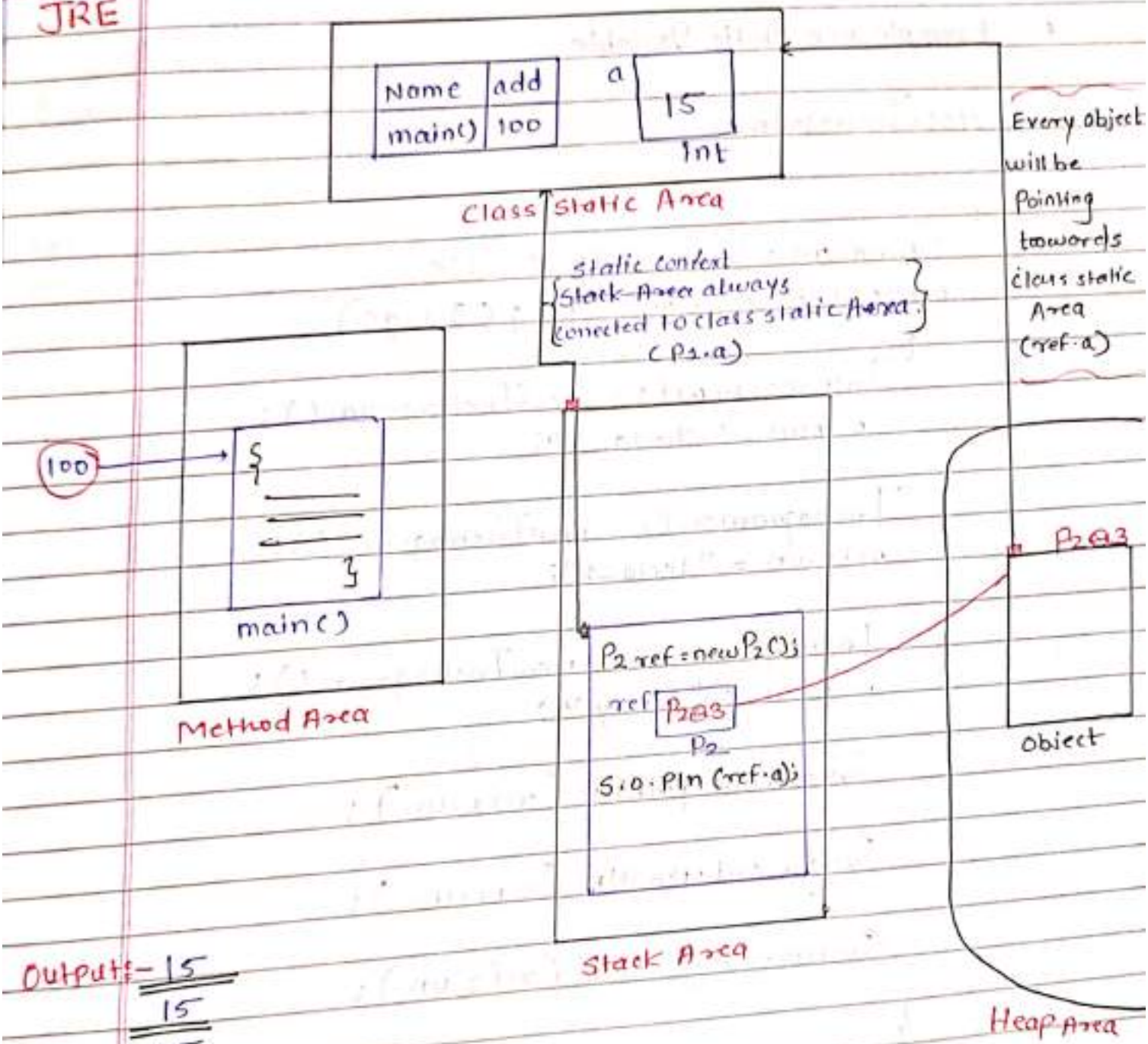
Note:- Every object will be pointing towards class static Area

example:-

```
class P2
    {
        Static int a = 15; // Global Variable (Static Variable)
        public static void main (String[] args) // static context
        {
            P2 ref = new P2 ();
        System.out.println (a);
        System.out.println (P2.a);
        System.out.println (ref.a);
```

output:- 15
15
15

JRE



Output:- 15
15
15

Page No.
Date

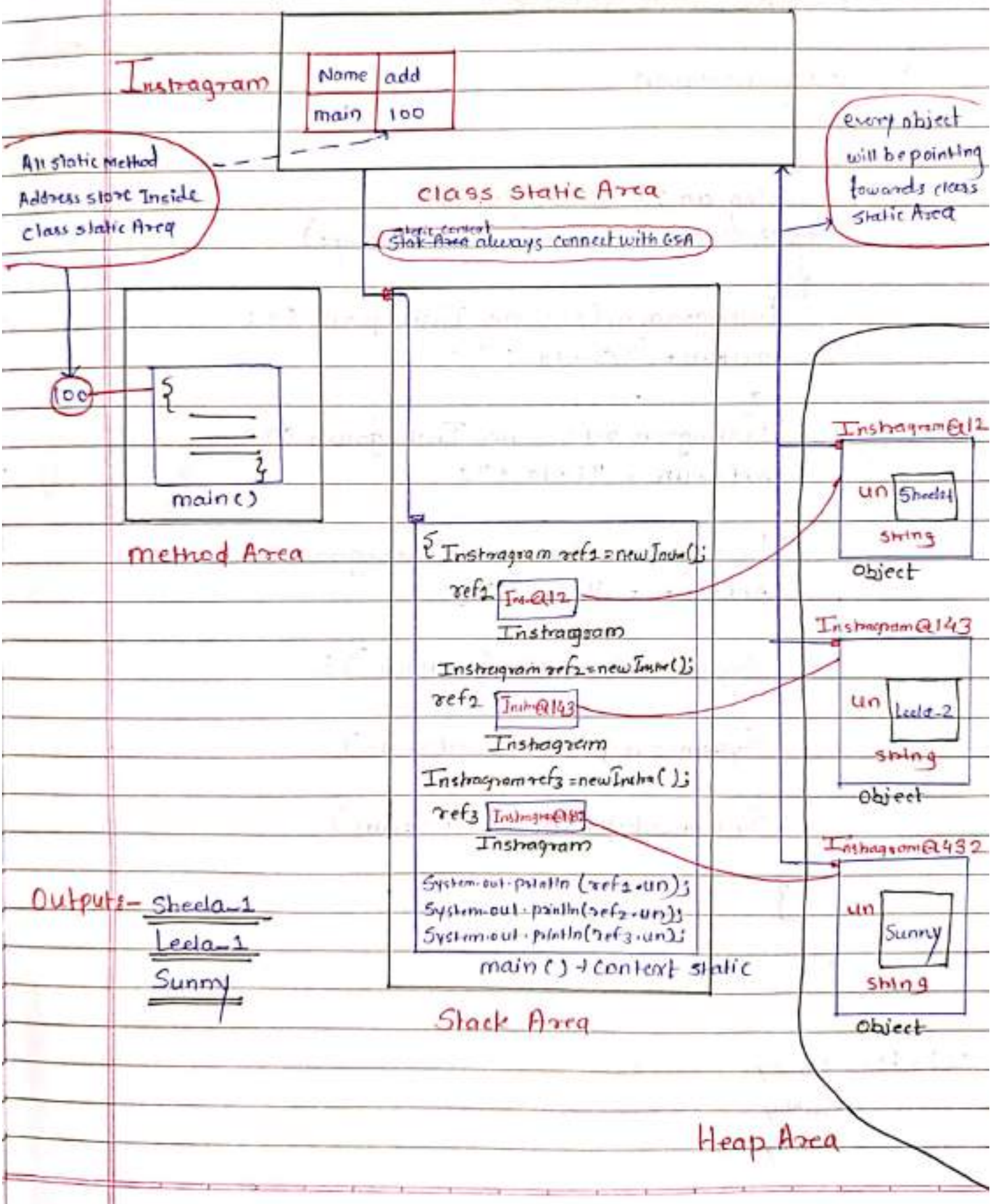
* Example non-static Variable

→

```
class Instagram
{
  String un; // non-static variable
  public static void main (String [] args)
  {
    Instagram ref1 = new Instagram ();
    ref1.un = "Sheela_1";

    Instagram ref2 = new Instagram ();
    ref2.un = "Leela_1";

    Instagram ref3 = new Instagram ();
    ref3.un = "Sunny";

    System.out.println (ref1.un);

    System.out.println (ref2.un);

    System.out.println (ref3.un);

  }

}
```

Output:-
Sheela_1
Leela_1
Sunny

JRE



Output:- Sheela_1
Leela_1
Sunny

* Example Static Variable

→

```
class Instagram
{
Static String un; // Static Variable
    public static void main (String [] args)
    {
      Instagram ref1 = new Instagram ();
      ref1.un = "Sheela_1";

      Instagram ref2 = new Instagram ();
      ref2.un = "Leela_1";

      Instagram ref3 = new Instagram ();
      ref3.un = "sunny";

      System.out.println (ref1.un);

      System.out.println (ref2.un);

      System.out.println (ref3.un);
    }

}
```

Output:- Sunny
Sunny
Sunny

JRE

every object will be pointing toward class static Area

Page No.
Date

Instagram

name	add
main	100

un ~~Null~~ ~~Sheela-1~~ ~~Leela-1~~ Sunny

String

Class Static Area

All static Method Address Store to Inside Class Static Area

Stack Area always Static content concern with C.S.A

100

{ ? }

main ()

Method Area

{ Instagram ref1 = new Insta();

ref1 Insta@123

Instagram

ref1.un = "Sheela-1"

Instagram ref2 = new Insta();

ref2 Insta@432

Instagram

ref2.un = "Leela-1"

Instagram ref3 = new Insta();

ref3 Insta@567

Instagram

ref3.un = "Sunny"

System.out.println(ref1.un);

System.out.println(ref2.un);

System.out.println(ref3.un);

main ()

Instagram@123

Object

Instagram@432

Object

Instagram@567

Object

Output- Sunny
Sunny
Sunny

Stack Area

Heap Area

Page No.
Date

* non-static method :- any method declared inside global Area not prefixed with static keyword is known as non-static method.

- all the method block are store inside Method Area. either it can static method block or non-static method block
- non-static method block is also known as non-static Context
- every non-static context will be pointing toward object.
- address of non-static methods will be stored inside object.

* this

1) this is keyword
2) this is a referance variable / Address variable.
3) this will have address of current object.

Note :-

- we can not use non-static variable "this" inside static context.
- this is non-static referance variable hence we can not use "this" inside static context.

* what is current object ?

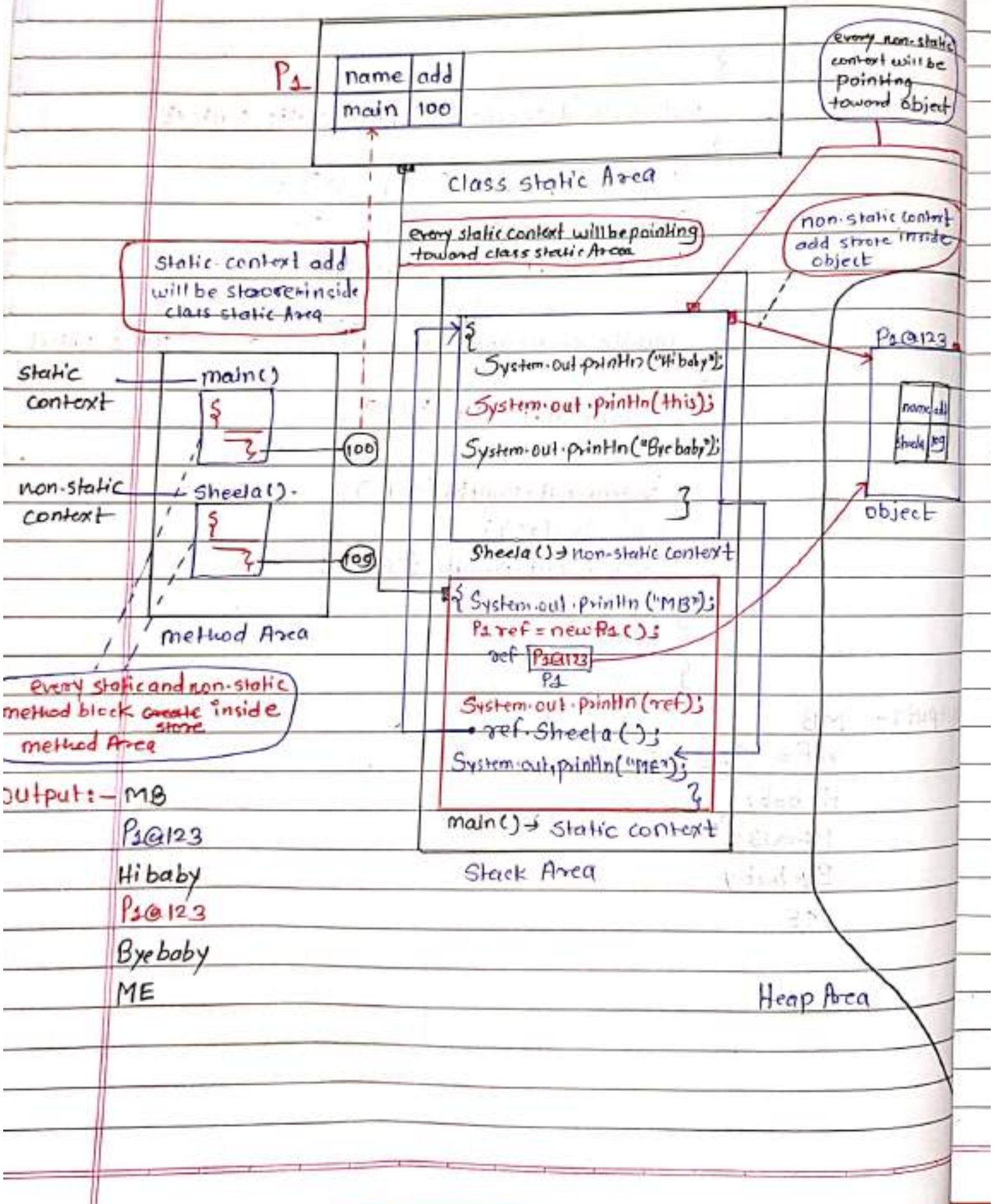
→ To call a method currently which address we use same address will be stored inside "this"

→

```
class P1
{
  public void sheela () // non-static context.
  {
    System.out.println ("Hi baby");
    System.out.println ( this );
    System.out.println ("Byebaby");
  }
    public static void main (String[] args) // static context
  {
    System.out.println ("MB");
    P1 ref = new P1 ();
    System.out.println (ref);
     ref.sheela();
    System.out.println ("ME");
  }

  }
```

Output:- MB
ref = P1@132
Hi baby
P1@132
Bye baby
ME

Static member ← method, variable, block



Output:- MB
Ps@123
Hi baby
Ps@123
Bye baby
ME

Q] How many ways can we use static members inside non static context.

→ we can use 3 ways
1) directly (a)
2) className.(a)
3) this.(a)

Q] How many ways can we use non-static members inside non-static context.

→ we can use 2 ways
1) directly (a)
2) this.(a)

Q] How many ways can we use non-static members inside static me context.

→ we can use only one way
1) object reference (ref.a)

Example:- Static members inside non-static context

```
class P1
    {
        static int i = 10; // static variable
        public static void sheela() // non-static context
    {
        System.out.println("directly : " + i);
        System.out.println("className : " + P1.i);
        System.out.println("this ref : " + this.i);
    }
        public static void main(String[] args) //static conte
    {
        P1 ref = new P1();
        ref.Sheela();
    }
}
```

Output :- 10
10
10

Example:- use non-static members inside non-static context

```
class P12
{
    int i = 10; // non - static variable
public void sheela() // non-static context
{
    System.out.println("Directly : " + i);
    System.out.println("this : " + i);
}
    public static void main (String [] args) // static context
    {
        P12 ref = new P12 ();
        ref.sheela();
    }
}
```

Output:- 10
10

Q] Why do we need this?

→

When we have non-static variable and local variable with same name, the high priority is given to a local variable.
If we want to use non-static variable, we need this keyword.

- this will use inside only non-static context

Example:-

```
class P13
{
    static int i = 10; // non-static variable
    public void sheela() // non-static context
    {
        int i = 20;
        System.out.println(i); // 20 (high priority to local variable)
        System.out.println(this.i); // 10
    }
    public static void main(String[] args) // static context
    {
        P13 ref = new P13();
        ref.sheela();
    }
}
```

Output:-
20
10

Page No.
Date

* **Non-Static block :-** Anonymous / IIB :- Instance Initializer block

* **Non-Static block :-** Any block declared inside global Area but not prefixed with static keyword is known as non-static block.

* **Characteristics of non-static block**

1) non-static block does not have any name.
2) non-static block does not have any return type.
3) It does not have any formal Argument.
4) non-static block get executed implicitly (Automatically) whenever we create object.
5) non-static block can not be called by programer.
6) non-static block get execute after main method.

Q] Why do we need non-static block ?

→ to execute some statement manadatory whenver we ~~ever~~ create object but only once ~~from~~ for one object.

Q] Why do we need static block ?

→ to execute some statements before the execution of main method but only once.

Page No.
Date

Example:- non-static block

```
class Instagram
{
    {
        System.out.println("A/c has been created Successfully");
    }
    public static void main (String[] args)
    {
        Instagram ref1 = new Instagram();
        Instagram ref2 = new Instagram();
        Instagram ref3 = new Instagram();
    }
}
```

output:- ~~Sys~~ A/c has been created Successfully
A/c has been created Successfully
A/c has been created Successfully

Note:- { to execute some set of statement
manadatory whenver we create object
but only once for one object
and multiple for multiple object. }

example:- static block

```
class Instagram
{
    static
    {
        System.out.println("A/c has been created");
    }
    public static void main(String[] args)
    {
        Instagram ref1 = new Instagram();
        Instagram ref2 = new Instagram();
        Instagram ref3 = new Instagram();
    }
}
```

Output:- A/c has been created

Note :- (to execute some set of statements before main method but only once)

* Constructor

→ Constructor is the special non-static members because

(1) constructor name must be as same as class name.

(2) constructor is a similar to method but constructor doesn't have any return data type.

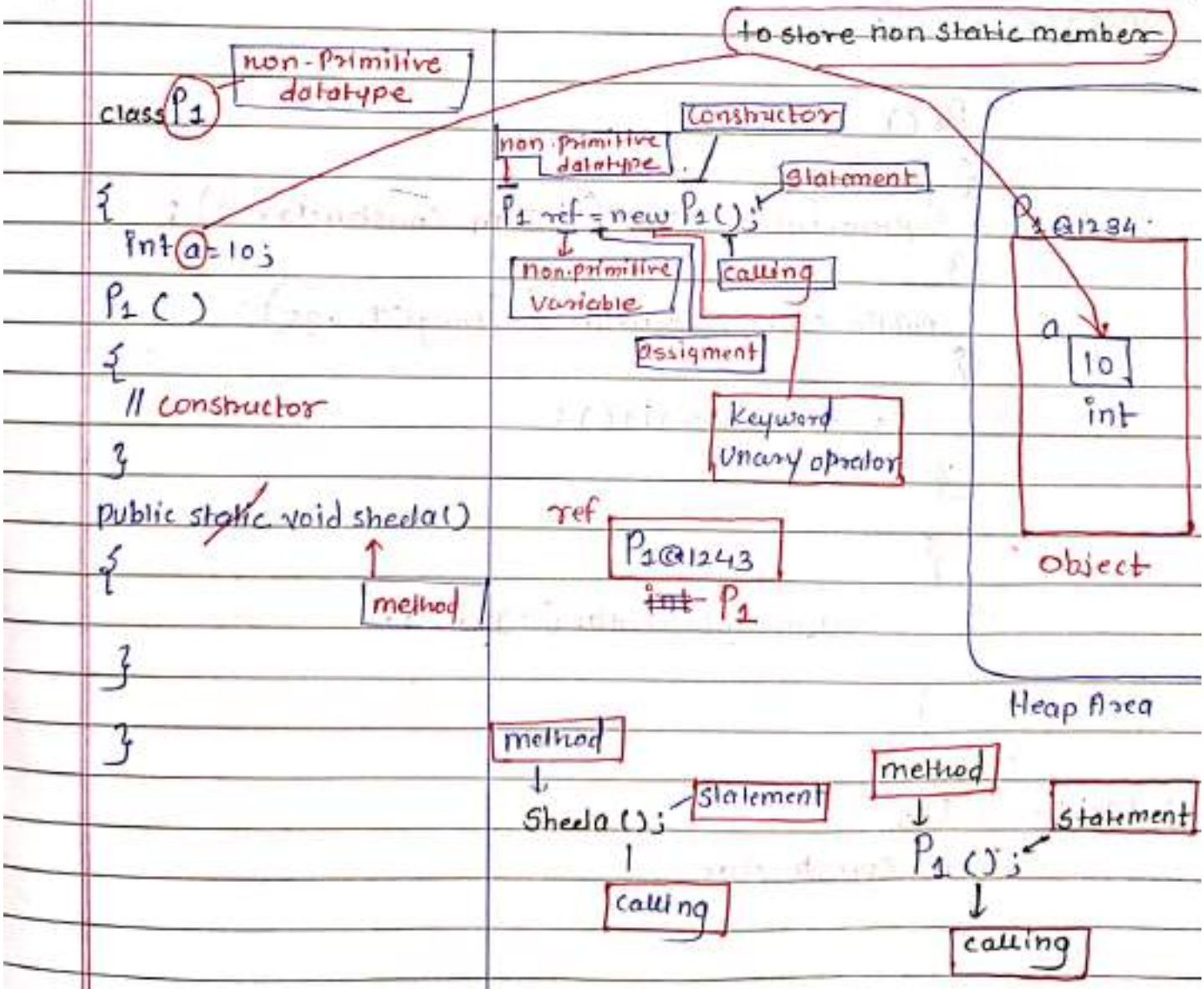
Q] why do we need a constructor?

→ it is used to store all the non-static members inside object.

Q] Difference between Method And Constructor

Method	Constructor
(1) method is a block of statement ~~to~~ used to perform specific task	(1) Constructor it is used to store all the non-static members
(2) method is created by programer	(2) constructor is created by compiler
(3) method have any Return typedata	(3) Constructor doesn't have any Return typedata
(4) method is static method and also non-static method	(4) constructor is only non-static member

(5) We can execute the method 3 ways with the help of 1) directly (sheela();) 2) className (P12.sheela();) 3) object Reference (ref.sheela();)	(5) Constructor we can execute only one way with the help of (1) obj ~~R~~ new keyword / Unary operator
(6) method name can be Anything	(6) Constructor name is similar to class name.



* Every constructor have 3 things

(1) PLI (Pre-Loading Instruction/statement)
:- To load all the non-static member inside object.

(2) IIB (Instance Initializer Block)

(3) UWS (User written statements)
:- present inside constructor

Example:-
```
class P1
{
  P1()
  {
   System.out.println("I m constructor");
  }
   public static void main (String[]args)
  {
     P1 ref = new P1();
  }
     {
       System.out.println("IIB");
     }
}
```

Output:-
```
IIB
I m constructor
```

* Types of Constructor

(1) no-Argument Constructor
(2) parameterized Constructor

⇒ no-Argument constructor :- The constructor don't have any formal Argument (FA) is known as no Argument constructor.

⇒ Parameterized constructor :- The constructor in which has any formal Argument (FA) is known as parameterized constructor.

example :-

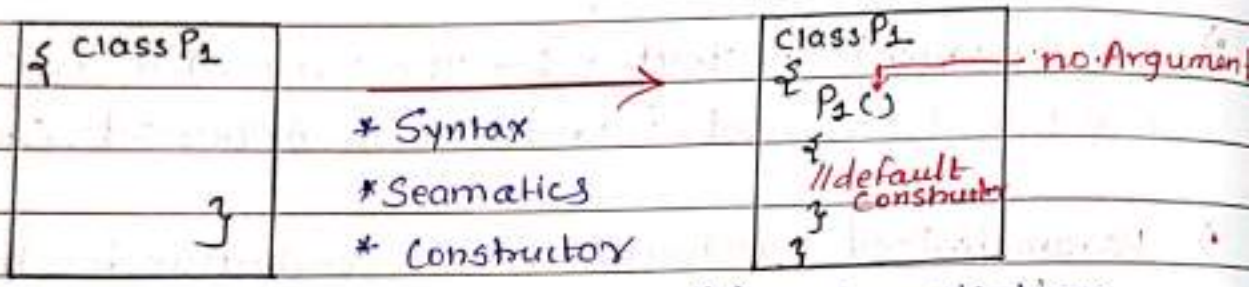
```
class P2
{
  P2 () // no-Argument Constructor
  {
    System.out.println ("No Argument Constructor");
  }
    P2(int a) // Parameterized Constructor
    {
      System.out.println ("Parameterized Constructor");
    }
}
```

Note :- Formal argument : variable declared inside the ~~method~~ constructor is known as formal Argument (FA)

Actual argument :- data passed inside ~~method~~ constructor calling statement is known as Actual argument (AA)

* default Constructor

→ If class is not having any constructor by default compiler will create no-argument constructor during compile time.



Before compiler Compilation

After compilation

* Constructor Overloading

→ If class is having more then one method Constructor with the same name but different formal Argument either it is different length or different datatype.

Constructor Overloading (example)

```
class P3
{
    P3()
    {
    System.out.println("No-Argument Constructor");
    }
    P3(int a)
    {
    System.out.println("One Formal Argument Constructor");
    }
    P3(int a, int b)
    {
    System.out.println("two formal Argument constructor");
    }
    P3(int a, int b, int c)
    {
    System.out.println("three formal Argument Constructor");
    }
    P3(int a, int b, int c, int d)
    {
    System.out.println("Four formal Argument Constructor");
    }
    public static void main(String[] args)
    {
      P3 ref = new P3(10,20,30);
      P3 ref = new P3(10);
      P3 ref = new P3(10,20);
      P3 ref = new P3(10,20,30,40);
      P3 ref = new P3();
    }
}
```

Output:- three Formal Argument Constructor
one formal Argument constructor
two formal Argument Constructor
four formal Argument Constructor
No - Argument Constructor.

* Why do we need Parametrized Constructor
→ TO Initialize non-static variable during object Creation

```
class P4
 {
   int i ;
  P4 (int i) // non-static variable
   {
      this.i = i;
   }
  public static void main (String [] args)
   {
     P4 ref = new P4 (40);
     System.out.println (ref.i);
   }
 }
```

Output:- 40

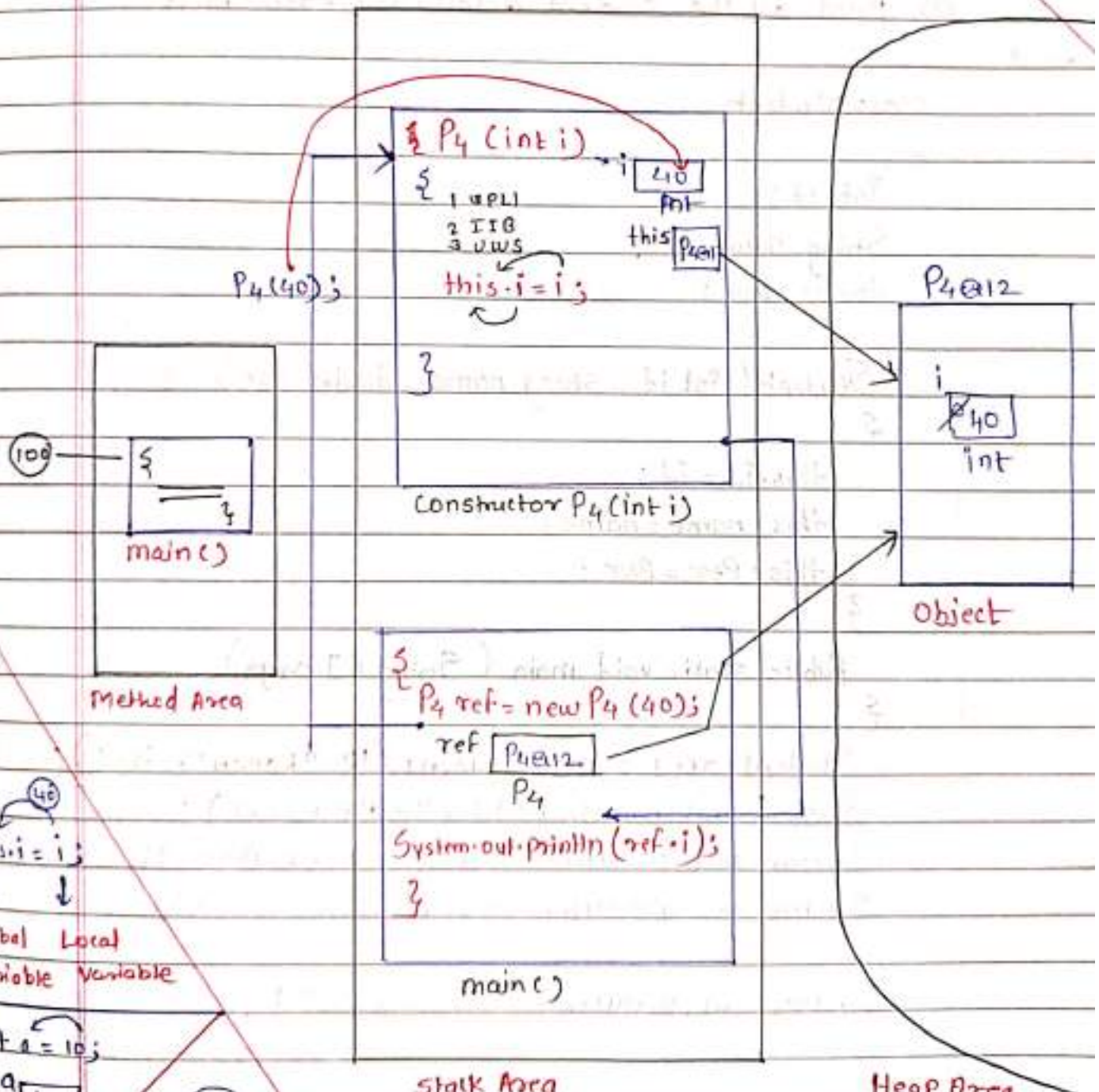
JRE

Page No.
Date

i = this.i ; ✗
this.i = this.i ; ✗
i = i ; ✗
this.i = i ; ✓

P4	name	add
	main()	100

Class Static Area



this.i = i ;

Global Variable | Local Variable

int a = 10 ;

a 10 int

int b = a ;

b 10 int

Scanned with CamScanner

Page No.
Date

Q] Create Student class consist of ~~sut~~ student-id, student-name and student-percentage.

(1) Create 3 Parametrized Constructor to Initialize Student Id, name, percentage.

(2) create at least two student object Initialize Unique data

(3) Print all the student details for each object

⟶

```
class Student
{
  int id;
  String name;
  double per;

  Student (int id, string name, double per)
  {
    this.id = id;
    this.name = name;
    this.Per = Per;
  }

  Public static void main (String [] args)
  {
   Student ref1 = new Student (10, "Karan", 68.7);
   System.out.println ("Id :" + Ref1.id);
   System.out.println ("name :" + ref1.name);
   System.out.println ("Per :" + ref1.per);

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

```
Student ref2 = new Student (20, "Aniket", 8.7);
System.out.println ("Id :" + ref2.id);
System.out.println ("name :" + ref2.name);
System.out.println ("Per :" + ref2.Per);

}
}
```

Output:-
```
Id : 10
name: karan
Per : 69.7
==========
Id : 20
name: Aniket
Per : 8.7
```

Page No.
Date

Q] Create employe class consist of employe Id employe name and employe salary.

- Create no-argument constructor
- Create one parametrized constructor to Initialize employe id
- Create two formal Argument constructor to Initialize employe id and employe name.
- Create three formal Argument constructor to Initialize ~~emp~~ employe -id, employe name, employe Salary.
- Create object foreach constructor & print employe deals for every object.

→

```
class Employe
{
    int id;
    String name;
    double salary;

    Employe()
    {
        System.out.println("no argument constructor");
    }
    Employe (~~int~~) (int id)
    {
        this.id = id;
        System.out.println("one formal Argument constructor");
    }
    Employe (int id, String name)
    {
        this.id = id;
        this.name = name;
        System.out.println("two formal Argument constructor");
    }
```

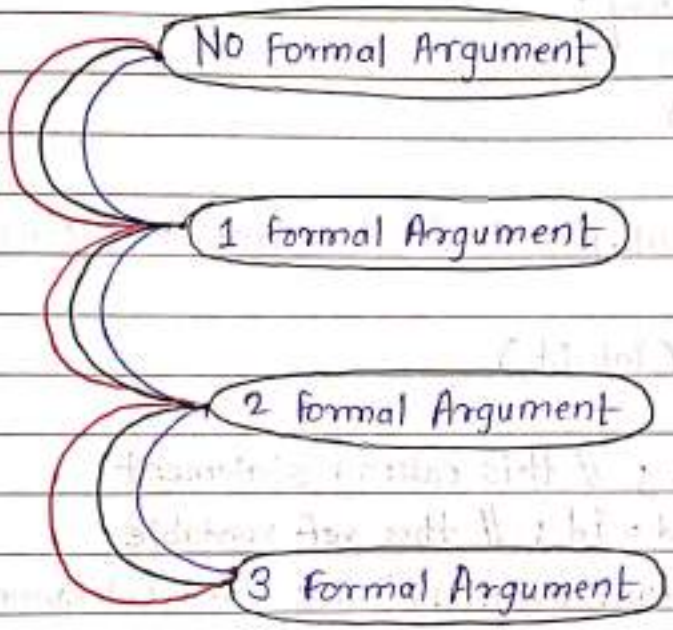
```
Employe (int id, String name, double salary)
{
    this.id = id;
    this.name = name;
    this.salary = salary;
System.out.println("three formal Argument constructor");
}
public static void main (String[] args)
{
    Employe ref1 = new Employe();
    System.out.println("id :" + ref1.id);
    System.out.println("name :" + ref1.name);
    System.out.println("Salary :" + ref1.salary);

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

    Employe ref2 = new Employe (10);
    System.out.println("id :" + ref2.id);
    System.out.println("name :" + ref2.name);
    System.out.println("Salary :" + ref2.salary);

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

    Employe ref3 = new Employe (10, "Karan");
    System.out.println("id :" + ref3.id);
    System.out.println("name :" + ref3.name);
    System.out.println("Salary :" + ref3.salary);

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

```
Employe ref4 = new Employe(20, "Aniket", 20.4);
System.out.println("Id:"+ref4.id);
System.out.println("Name:"+ref4.name);
System.out.println("Salary:"+ref4.salary);

}

}
```

Output:- no Argument Constructor
id : 0
name: Null
Salary: 0.0
==========
one formal Argument Constructor
id : 10
name: null
Salary: 0.0
==========
two formal Argument Constructor
id : 20
name: Karan
Salary: 0
==========
three formal Argument Constructor
Id : 20
name: Aniket
Salary: 20.4

* Constructor Chaining

→ One Constructor is calling another Constructor is known as Constructor chaining.



* Difference between this. and this();

	this.	this();
1.	this is referance Variable	this calling statement
2.	To access non-static Variable	To call another Constructor inside constructor
3.	this will have address of current object	is also known as Constructor calling statement

* Example for constructor chaining

```
class Employe
{
  int id;
  String name;
  double salary;

  Employe()
  {
    System.out.println("No-Argument constructor");
  }
  Employe(int id)
  {
    this(); // this calling statement
    this.id = id; // this ref variable
    System.out.println("one formal Argument Constructor");
  }
  Employe(int id, String name)
  {
    this(id);
    this.name = name;
    System.out.println("Two formal Argument Constructor");
  }
  Employe(int id, String name, double salary)
  {
    this(id, name);
    this.salary = salary;
    System.out.println("three formal Argument Constructor");
  }
```

```
public static void main (String [] args)
{
  Employe ref4 = new Employe (12, "Karan", 25000);
  System.out.println("id :"+ref4.id);
  System.out.println ("name :"+ref4.name);
  System.out.println ("salary :"+ref4.salary);
}
}
```

Outputs- No argument Constructor
One Formal Argument constructor
Two formal Argument Constructor
Three formal Argument constructor
id : 12
name: Karan
Salary: 25000.0

* How to achieve Constructor chaining

→ Constructor chaining can be achieve in two ways
this calling statement
Super calling statement

→ this calling statement :- It is used to call the constructor of same class.

* Rules for this calling statement (this();)

(1) this calling statement must be first the statement in the Constructor.

(2) one constructor can have only one This calling statement.

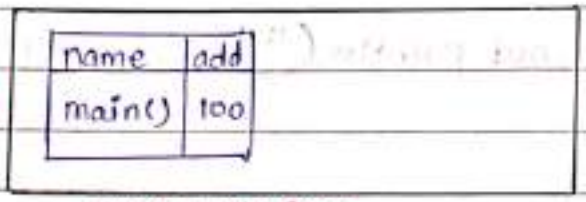
(3) If we have (n) numbers of constructor then we can have only (n-1) this calling statement

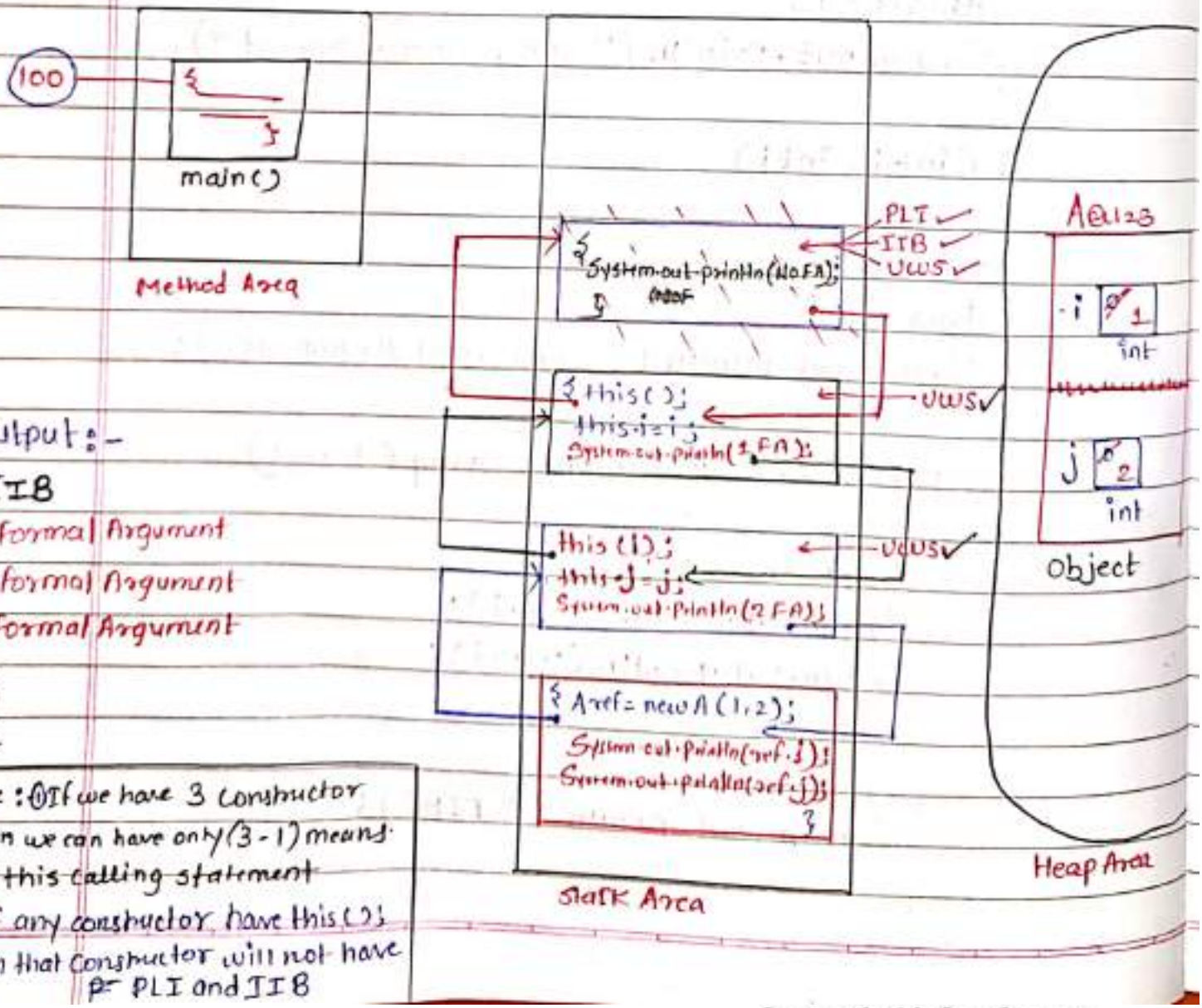
(4) Constructor recurssion is not possible.

Note :- If any constructor has this calling statement then that Constructor will not have PLI and PIIB

```
class A
{
  int i;
  int j;
  A()
  {
    System.out.println("No Formal Argument constructor");
  }
A(int i)
  {
    this();
    this.i = i;
   System.out.println("one Formal Argument");
  }
  A(int i, int j)
  {
    this(i);
    this.j = j;
    System.out.println("two Formal Argument");
  }
    public static void main(String[] args)
    {
      A ref = new A(1,2);
      System.out.println(ref.i);
      System.out.println(ref.j);
    }
      {
        System.out.println("IIB");
      }
    }
```

Output:- IIB
No formal Argument constructor
one formal Argument
two formal Argument
1
2





Output:-
IIB
NO formal Argument
1 formal Argument
2 formal Argument
1
2

Note: ① If we have 3 constructor then we can have only (3-1) means 2 this calling statement
② If any constructor have this(); then that constructor will not have PLI and IIB

14/11/2022

Page No.
Date

* OOP → Object Oriented programing

→ object oriented programming is a design model / Paradigm which helps programmer to co-relate real world senarios to the programming world senarios in the form of object.

* OOP'S → Object Oriented programming Principles.

→ ① Encapsulation
② Inheritance
③ Polymorphism
④ Abstraction

* OOPL → Object Oriented programing Language.

→ Any programming Language which follows object oriented programming principle is known as object oriented programing Language.

Or

Short definition → Any (PL) which follows (oop's) is known as (oopL)

Note :- 1) In Java state of an object will be Represented as Variable.
2) behaviours of an object will be Represented as method.

Real world
Physically exist

Object

Sheela
(Human Bing)

→ Name
→ age
→ Hight
→ weight
→ Gender

Properties / States → Variable

```
talk()
{
}
eat()
{
}
walk()
{
}
```

behaviours ⇒ methods

OOP

Design model / paradigm

Physically does not exist

Programming world

Emp@12

id	10
name	Sheela
sal	25000

States → Variable

name	add
mecinal()	
Sheela()	200

behaviours → methods

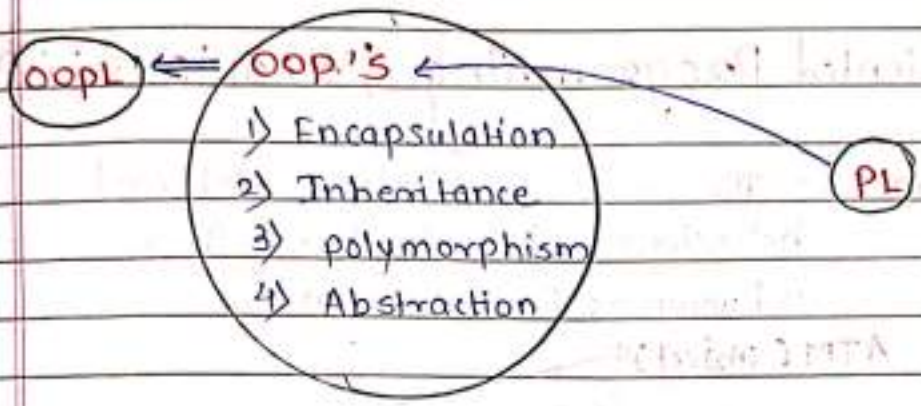
Object

offline	online
Real word	Programing word

Shoes

Programmer

Scanned with CamScanner



Q] what is class ?

→

1) class is blue print of Real world object
2) class is used to create non-primitive datatype.
3) class is keyword

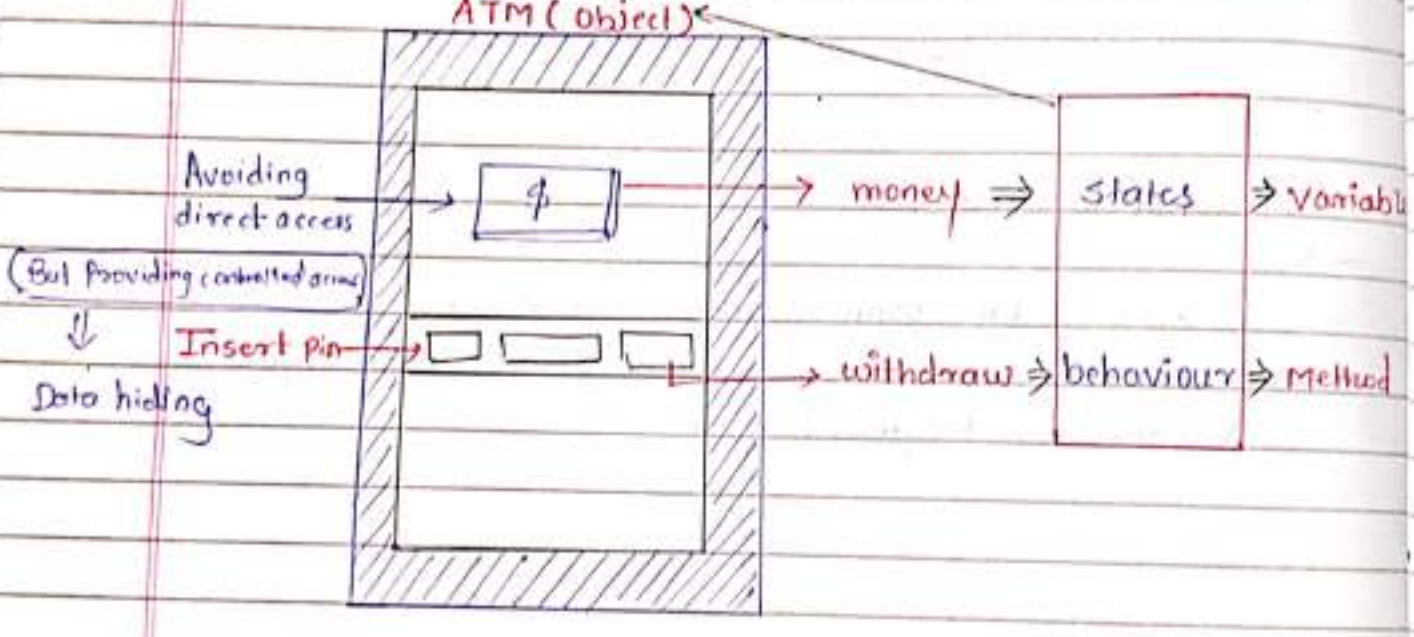
→ variable

→ methods

15/11/2022

Object Oriented Programming Principle (OOP's)

* Encapsulation :- The process of binding state and behaviours of an object together is known as Encapsulation



* It is not a concept of the oopt, Basic and Imp question for interview.

1) can we create a class without main method
→ yes we can create class without main method we can compile but we cannot execute.

2) can we create multiple classes in one Java file.
→ yes we can create multiple classes all the classes will get compile.

3) what is driver class
→ The class which has main method is known as driver class

4) Can we create object of one class inside another class
→ Yes we can create object of one class inside another class

Output

example:-

```
class ATM
{
    double money = 85000 ; // states
     public void withdraw() // bihaviours
    {
     System.out.println ("u can withdraw the cash");
    }
}

class SBIbank
{
    public static void main (String [] args)
   {
     ATM ref = new ATM();
  System.out.println ("total amount:" + ref.money);
    ref.withdraw ();
   }
}
```

Output:- total amount : 85000.0

u can withdraw the cash

Q] why do we need Encapsulation ?

→ To achieve data hiding

Q] what is data hiding ?

→ The process of avoiding direct Access but providing control access is known as data hiding.

Q] How to achive data hiding ?

→ by using private Access specifiers.

example

```
class ATM
{
  private double money = 85000; // states
  public void withdraw() // behaviour
  {
    System.out.println("u can withdraw cash");
  }

class SBIbank
{
  public static void main (String[] args)
  {
    ATM ref = new ATM();
    System.out.println("Total amt:" + ref.money); //CTE
    ref.withdraw();
  }
}
```

16/11/2022

Page No.
Date

Q] what happends when we make any data as private ?

→ ① we can use only inside the class where it is declared but we cannot use inside different class.

② If we want to use private data inside different class then we must create getter and setter method.

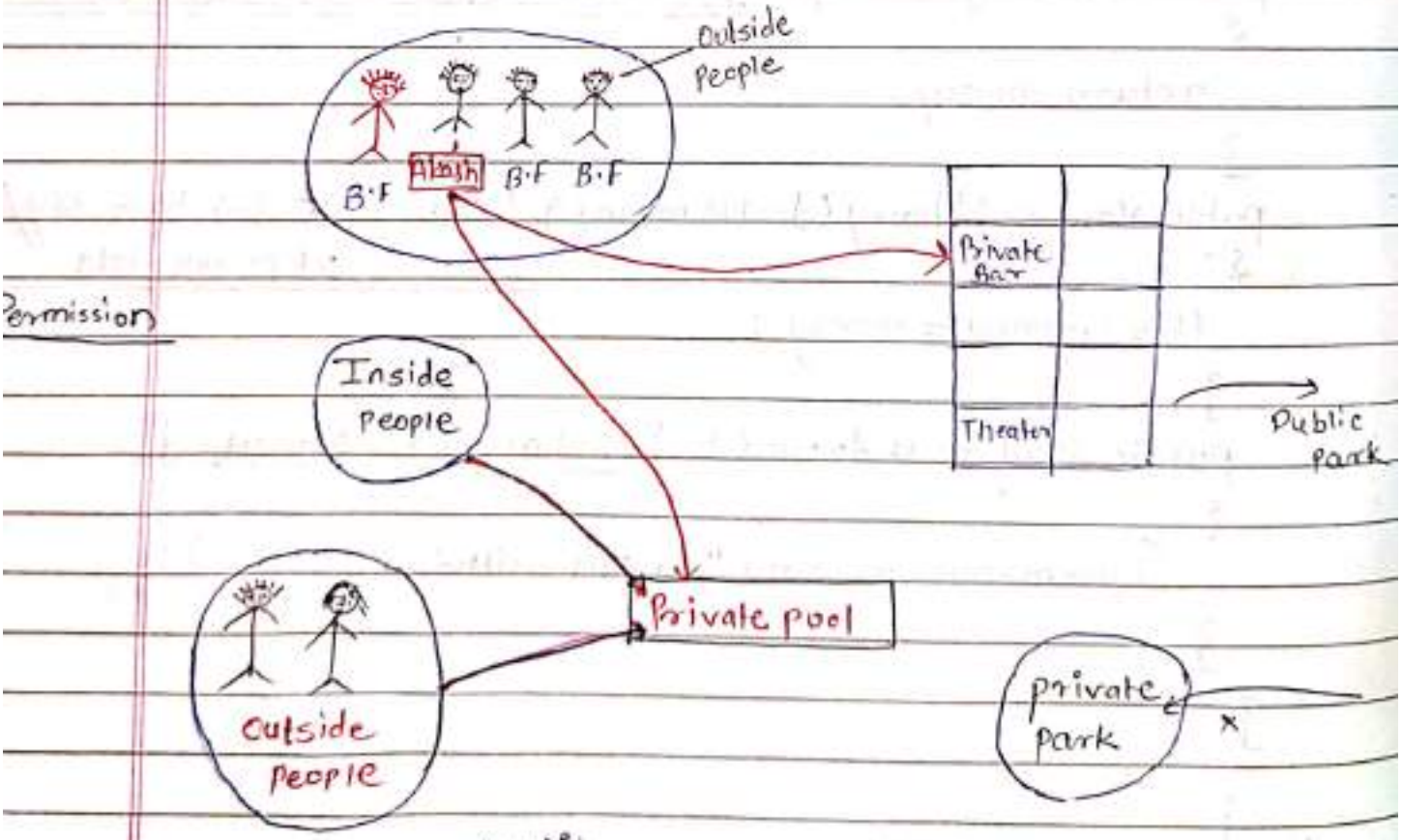
getter method → get private data
setter method → set private data

```
class ATM
{
   private double money = 85000; // states --> data

public double getMoney() // permission -> getter method to get private data
{
   return money;
}
public void setMoney(double money) // setter method -> to modify/
{                                  set private data.
   this.money = money;
}
public void withdraw() // behaviour --> method
{
   System.out.println("u can withdraw cash");
}
}
```

↳ continuous Next page

```
class SBIbank    // Driver class
{
   public static void main (String[] args)
   {
      ATM ref = new ATM ();
      ref.setMoney (40000);
   System.out.println (" Total amt : " + ref.getMoney ());
      ref.withdraw ();
   }
}
```

Output :- Total amt :- 40000.0
we can withdraw cash.



camel case
methodName ()
{
}

lower
getVar ()
{
}

Page No.
Date

Q) Create employee class consist of employee id, employee name, and salary
i) Create parameterized constructor To Initialized to all the data.
ii) achieve data hiding for salary
iii) Create at least two employee object
iv) print all the details of employee & create a method to increase salary.

→
```
class Employee
{
    int id;
    String name;
    private double sal;

    public double getSal() // getter method → get private data
    {
        return sal;
    }
    public void setSal(double sal) // setter method → set private data
    {
        this.sal = sal;
    }

    Employee(int id, String name, double sal) // constructor
    {
        this.id = id;
        this.name = name;
        this.sal = sal;
    }
}
```

```
class Employee Driver    // Driver class
{
  public static void main (String [] args)
  {
    Employee ref1 = new Employee (10, "Aniket", 25000);
    System.out.println (ref1.id);
    System.out.println (ref1.name);
    System.out.println (ref1.Sal);
    ref1.setSal (40000);
    System.out.println (ref1.getSal());

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

    Employee ref2 = new Employee (20, "karan", 30000);
    System.out.println (ref2.id);
    System.out.println (ref2.name);
    ref2.setSal (50,000);
    System.out.println (ref2.getSal());

  }
}
```

output:-

Page No.
Date

* Advantages of Encapsulation

1) It is used to achieve Data hiding
2) It is used to achieve design Flexibility

i) Data can be made only Readable
Q. How to make data only Readable?
a) Data must be ~~pefiti~~ prefix with private access specifier.
b) Create only getter method.

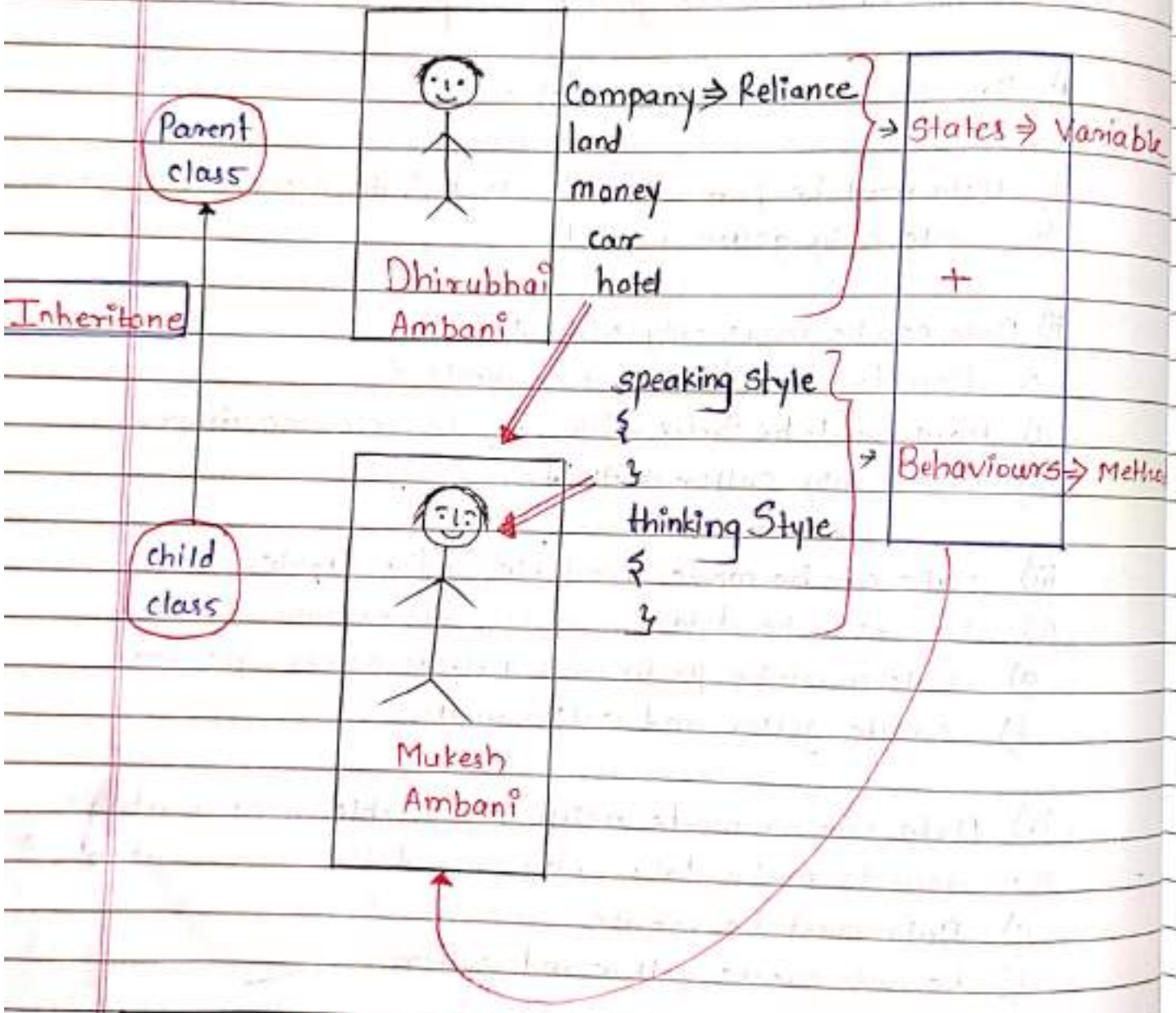
ii) Data can be made only writable
Q. How to make data only writable?
a) Data must be Prefix with private access specifier.
b) Create only setter method.

iii) Data can be made readable and writable.
Q] How to make data readable and writable?
a) Data must be prefix with private access specifier.
b) Create getter and setter method

iv) Data can be made neither readable nor writable
Q. How to make data neither readable nor writable?
a) Data must be private.
b) Do not create getter and setter.

17/11/22

* Inheritance :- the process of child class aquiring all the states and behaviour of parent class is known as Inheritance.



Note :- child class is also known as sub-class or derived class
parent class is also known as super-class or Base class

Page No.
Date

Q] How to achieve Inheritance ?

→ i) by using extends keyword
ii) by using implements

Syntax :- Sub-class extends Super class
child class extends parent class

example :-

```
class Dhirubhai
{
  String company = "Reliance Inds" ; // states ⇒ Variable
 public void thinkingStyle () ;// behaviours
  {
    System.out.println ("Dhirubhai thinkingStyle.") ;
  }
}

class Mukesh extends Dhirubhai // Subclass extends super class
{
}

class India // driver class
{
  public static void main (String [] args)
  {
    Mukesh ref = new Mukesh () ;
   System.out.println (ref. company) ;
     ref . thinkingStyle () ;
  }
}
```

Output :- Reliance Inds
Dhirubhai thinking Style.

Note :- ① with the help of extends keyword the class can inherit one class but it cannot inherit multiple classes.

Q] Create car class consist of car colour, car brand, car price
① Initialized all the data
② create your Favourite car inherit car class
③ create car driver to drive your Favourite car.

→
```
class Car
{
  String colour;
  String brand;
  double price;

public void carstyle()
 {
  System.out.println("unige car style");
 }
}
```

```
class Favourite_car extends Car
{
    Favourite_car (String colour, String brand, double price)
    {
        this.colour = colour;
        this.brand = brand;
        this.price = price;
    }
}

class Expert_Driver
{
    public static void main (String[] args)
    {
        Favourite_car ref1 = new Favourite_car ("red", "kiya", 40000);
        System.out.println (ref1.colour);
        System.out.println (ref1.brand);
        System.out.println (ref1.price);
        ref1.carstyle ();

    }
}
```

Output:-
red
kiya
40000
unige car style.

Q] Can we Inherit static members

→ yes, we can Inherit static members

* Inheritance with Respect to Static member

→ ① always Super class will Looded First
② once the loading process of super class is complited then sub-class will be looded.

```
class P1
{
  static
  {
    System.out.println("SIB from Super-class");
  }
}

class P2 extends P1
{
  static int i=15;
  static
  {
    System.out.println("SIB from sub-class");
  }
}

class Driver2
{
public static void main(String[] args)
{
P2 ref = new P2();
System.out.println("i:"+ref.i);
}}
```

15/11/2022

Page No.
Date

SIB from Super class

output :- SIB from Sub - class
IS

Q] Can we inherit non-static members.

→ yes we can Inherit non-static members.

* Inheritance with respect to non-static members.

→ 1) Sub class object always will have members of sub class as well as members of Super class.

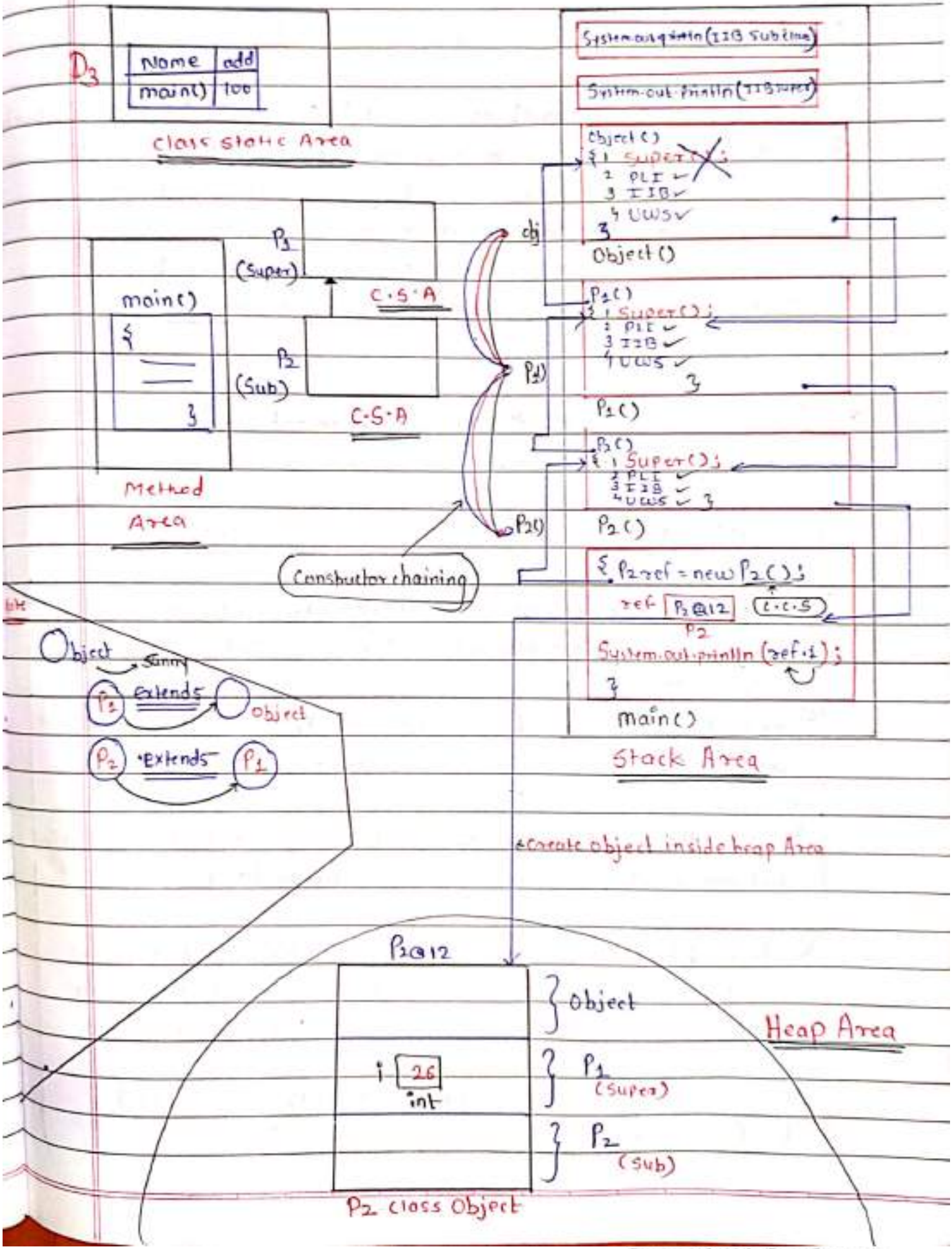
2) Super class members will be stored into Subclass object ~~with~~ with the help of Constructor chaining by using Super calling statement (super();)

* Super calling Statement (super();)

→ ① it is a constructor calling statement
② it is used to call constructor of Super class
③ it is used to achieve constructor chaining between Sub class and Super class

```
class P1
{
  int i = 26; // non-static var
  {
    System.out.println (" IIB - Super class");
  }
}
  class P2 extends P1
  {
    {
      System.out.println ("IIB - Sub class");
    }
  }
  class Driver3
  {
    public static void main (String [] args)
    {
      P2 ref = new P2 ();
    System.out.println ("i :" + ref.i);
    }
  }
```

Output :-
```
IIB - Super class
IIB - Sub class
26
```



* Rules For Super calling Statement

1. Super calling statement must be 1st Statement inside Constructor
2. One constructor can have only one super calling statement
3. One constructor can have either this calling statement or Super calling statement but it can not have both.
4. If programmer as not Return this calling statement then compiler will create Super calling statement

* Difference between this calling statement and Supercalling Statement.

this();	super();
1) this(); is also known as Constructor calling statement	1) super(); is also known as Constructor calling statement
2) this(); is used to call constructor from of some class	2) super(); is used to call constructor from of super class.
3) inside this(); inheritance not manadatory	3) inside super(); inheritance are mandatory
4) this(); is used to achieve Constructor chaining within Same class.	4) super(); is used to achive Constructor chaining within Sub-class and super-class
5) this(); will be called explicity by programmer	5) super(); will be added implicity by compiler

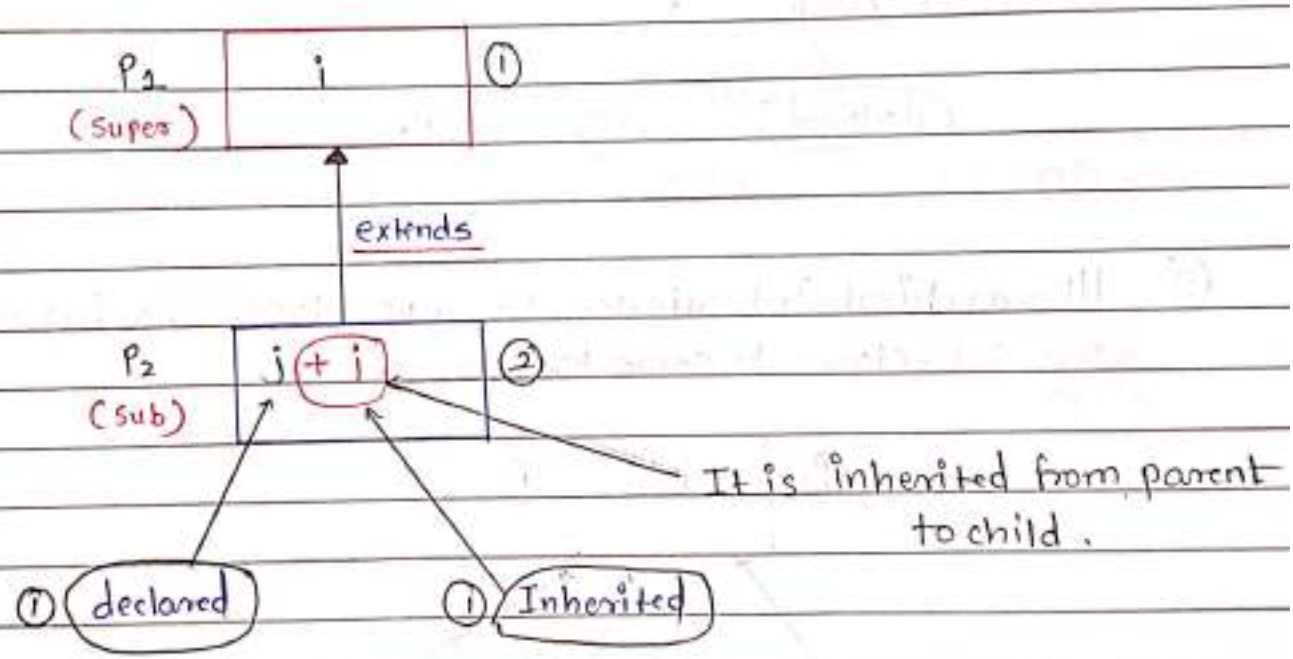
Page No.
Date

* Types of Inheritance

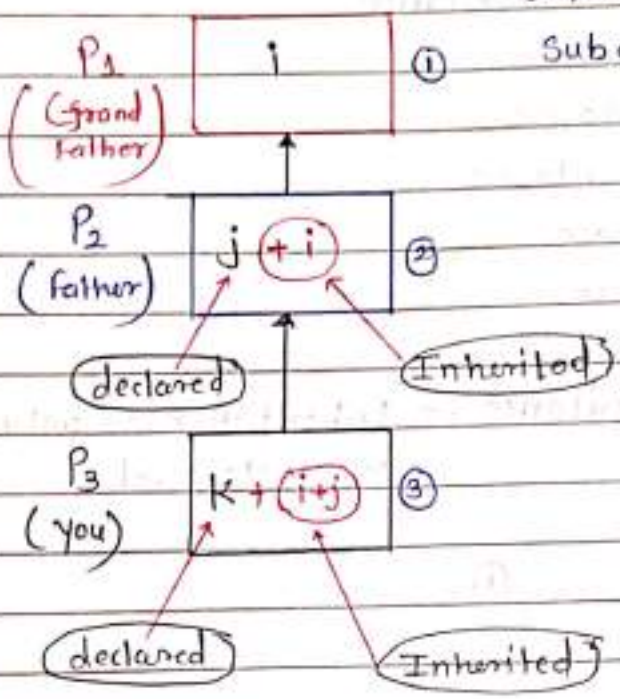
there are 5 types of Inheritance.

1. Single level Inheritance
2. Multi-level Inheritance
3. Hierarchical Inheritance
4. multiple Inheritance
5. Hybrid Inheritance

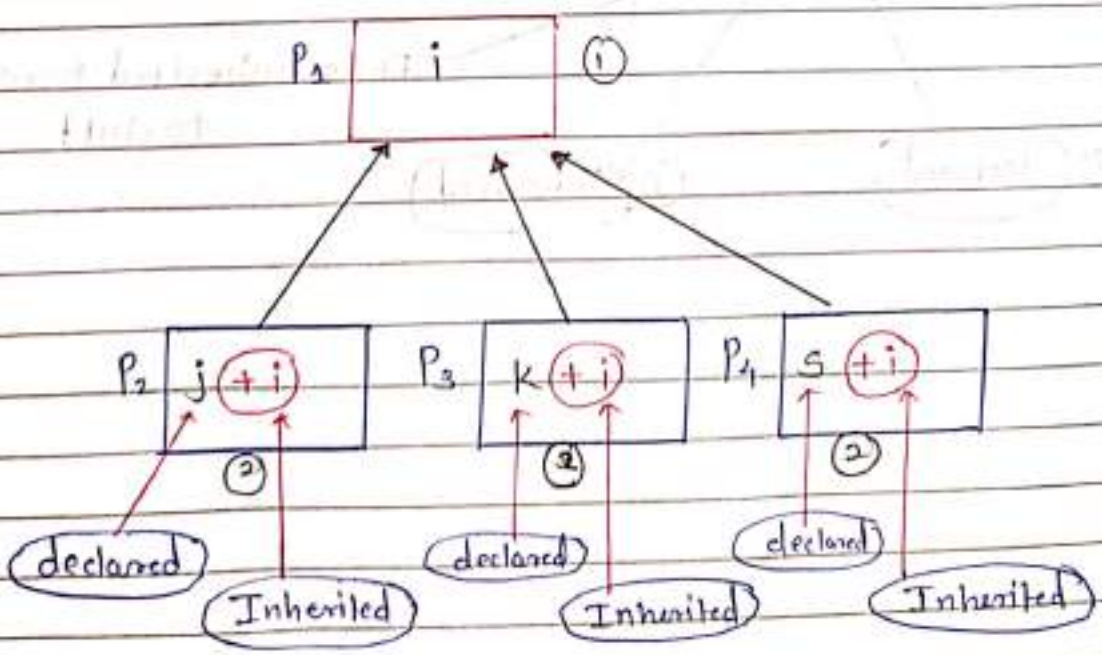
→ ① Single level Inheritance :- Inheritance of only one level or single level



② Multi-level Inheritance :- Inheritance of more than one level
or
Super class is having more than one sub class at different level



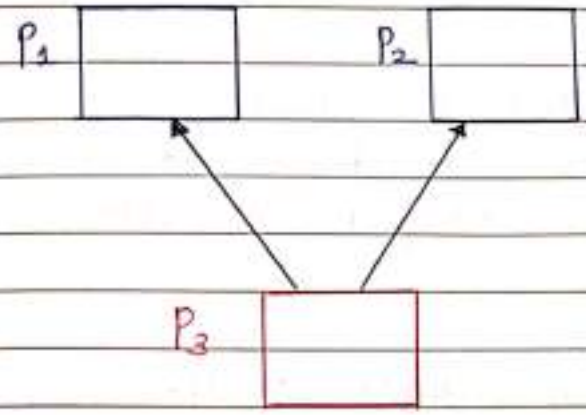
③ Hierarchical Inheritance :- Super class is having more than one sub-class at same level.



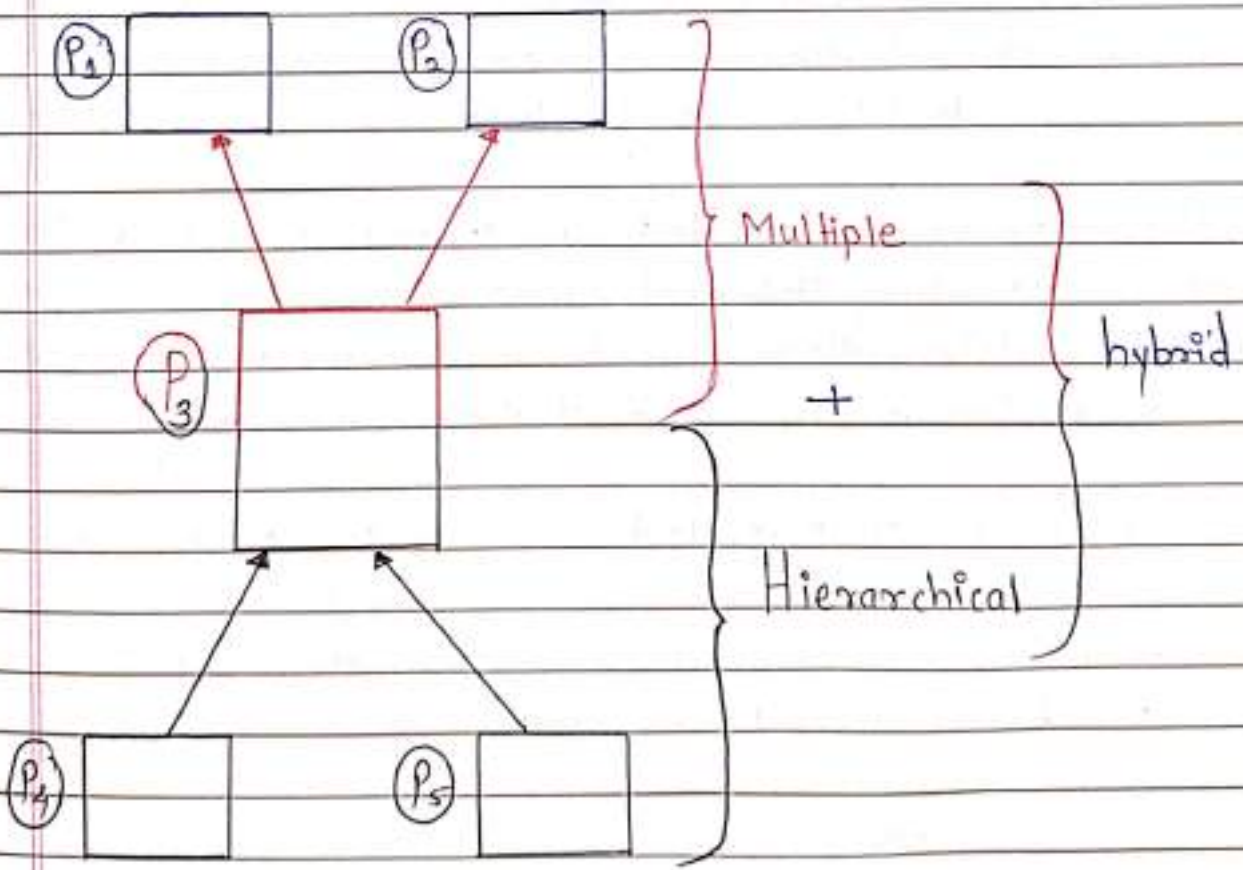
at same Level

Page No.
Date

④ Mulliple Inheritance :- Sub class is having more than one super-class at same level.



⑤ Hybrid Inheritance :- Combination of Hierarchical and multiple Inheritance.



19/11/2022

Page No.
Date

* Does Java Support multiple Inheritance ?
or
Can we achieve multiple Inheritance ?

→ yes. Java support multiple Inheritance
with the help of class it is not possible.
but with the help of Interface it is possible.

* Diamond problem ?

→ the confusion arises in multiple inheritance is known as dimond problem.

* Reason for dimond problem :-

1) Confusion with respect to class Loading process (C.L.P) :-

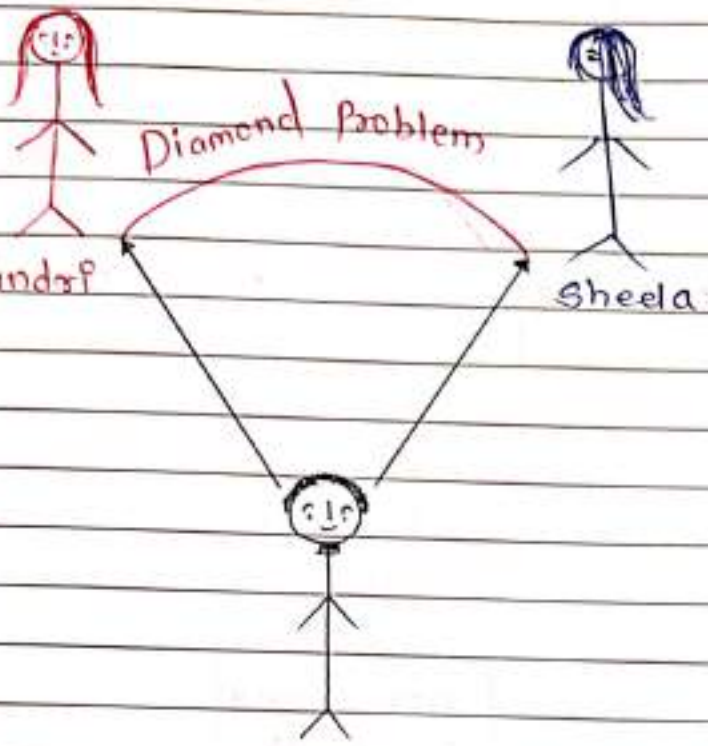
→ Always super class should be loaded first but in multiple inheritance compiler gets confused which super class to be loaded first.

2) Confusion with respect to object Loading process (O.L.P) :-

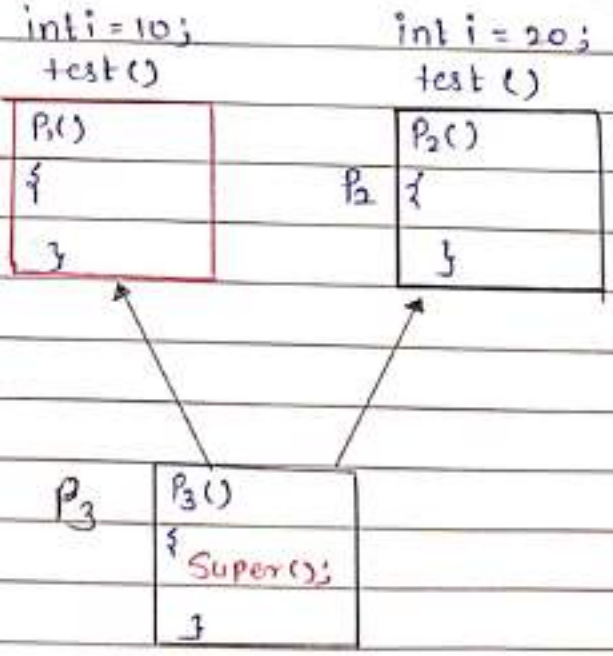
→ super calling statement use to call constructor of super class but in multiple inheritance it will gets confused which super class constructor to be called.

3) Confusion with respect to accessing the members :

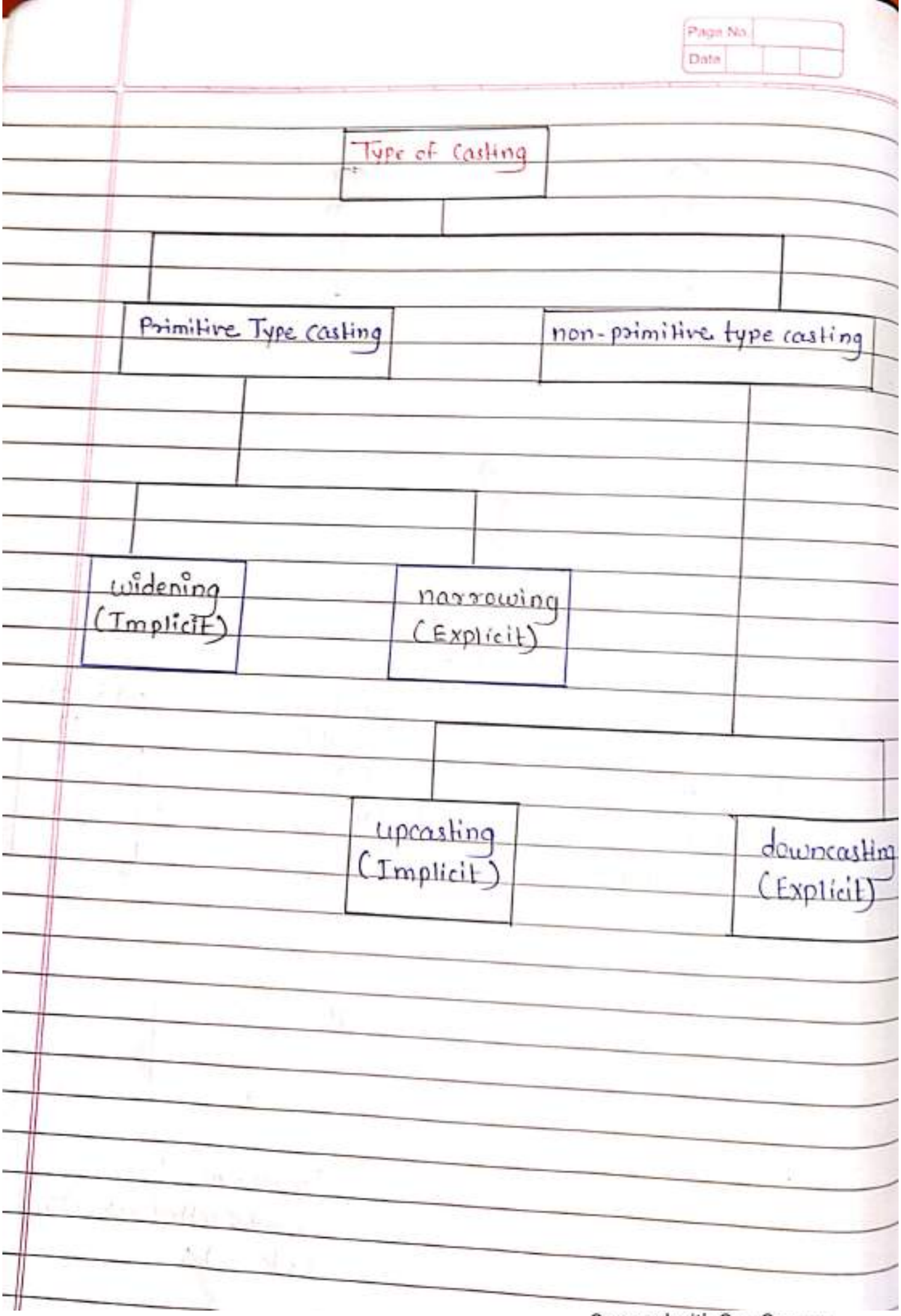
→ in multiple inheritance if super class is having members with same name then compilers get confused which members to be used.



1) C.L.P
2) O.L.P
3) Accessing members




```
P3 ref = new P3();
System.out.println(ref.i); // 10 or 20
ref.test();
```



* Non-primitive type casting

The process of converting one non-primitive datatype into another non-primitive datatype is known as Non-primitive type casting

- up-casting

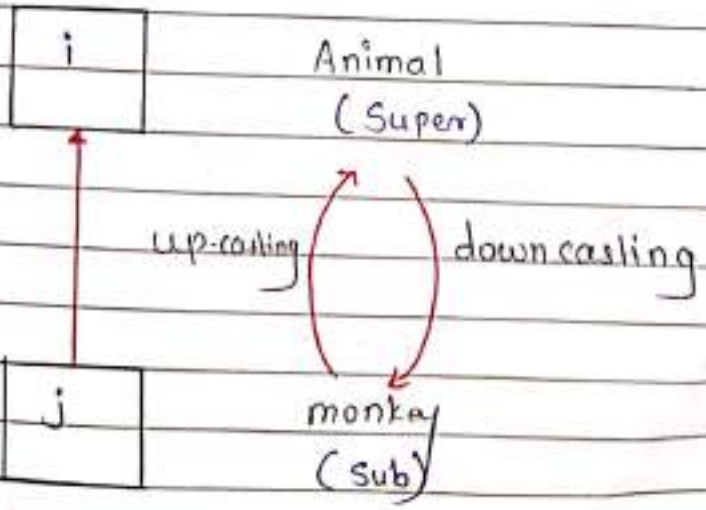
process of converting Sub class type into Super class type is known as up-casting.

Note :- up-casting will be done Implicitly by the compiler

- down-casting

the process of converting Super class type into Sub class type is known as down-casting.

Note :- down-casting will be done explicitly by the programmer by using type cast operator.



Page No.
Date

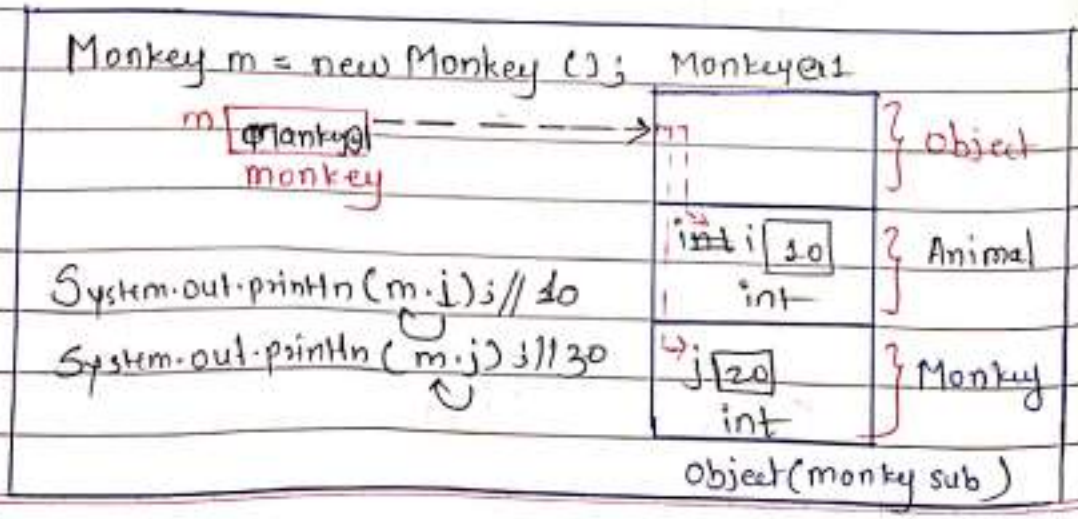
Note :- 1) with help of sub class object referance we can use member of Subclass as well as member of Super class

2) with the help of Super class object referance we can use only member of Super class but we can not use member of Sub class

example :-

```
class Animal
{
  int i = 10;
}
class Monkey extends Animal
{
  int j = 20;
}
class Driver1
{
  public static void main (String [] args)
  {
    Monkey m = new Monkey (); // sub - class
    System.out.println ("i :" + m.i); // 10
    System.out.println ("j :" + m.j); // 20
  }
}
```

Output :- 10
20

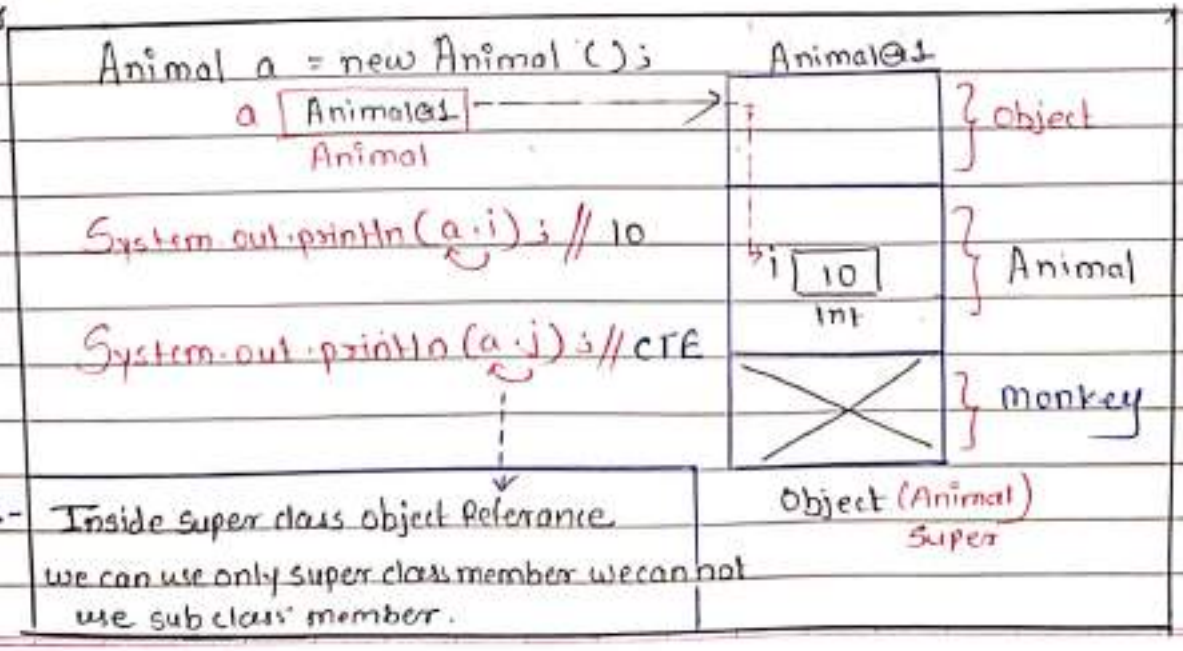


2) with the help of Super class object Referance we can use only member of Super class but we can not use member of sub class.

example:-

```
class Animal
{
  int i=10;
}
class Monkey extends Animal
{
  int j=20;
}
class Driver2
{
  public static void main (String[] args)
  {
    Animal a = new Animal (); // Super-class
    System.out.println ("i :" + a.i); // 10
    System.out.println (" j :" + a.j); // CTE
  }
}
```

Output :-

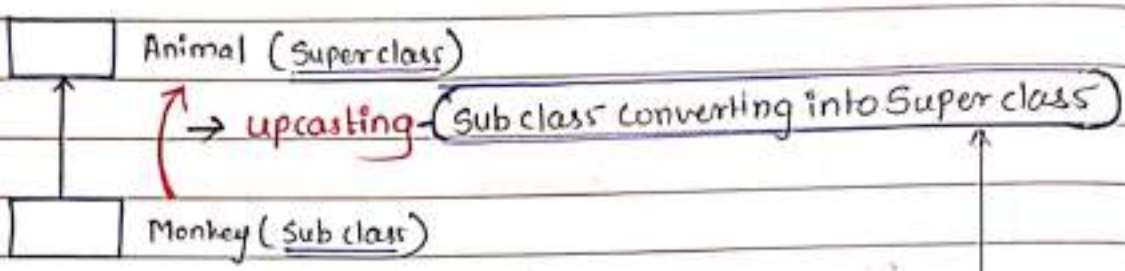


Note :- Inside super class object Referance we can use only super class member we can not use sub class' member.

Page No.
Date

* Drawback of UPCASTING

When we perform up casting we cannot use members of Sub class, but we can use only members of super class

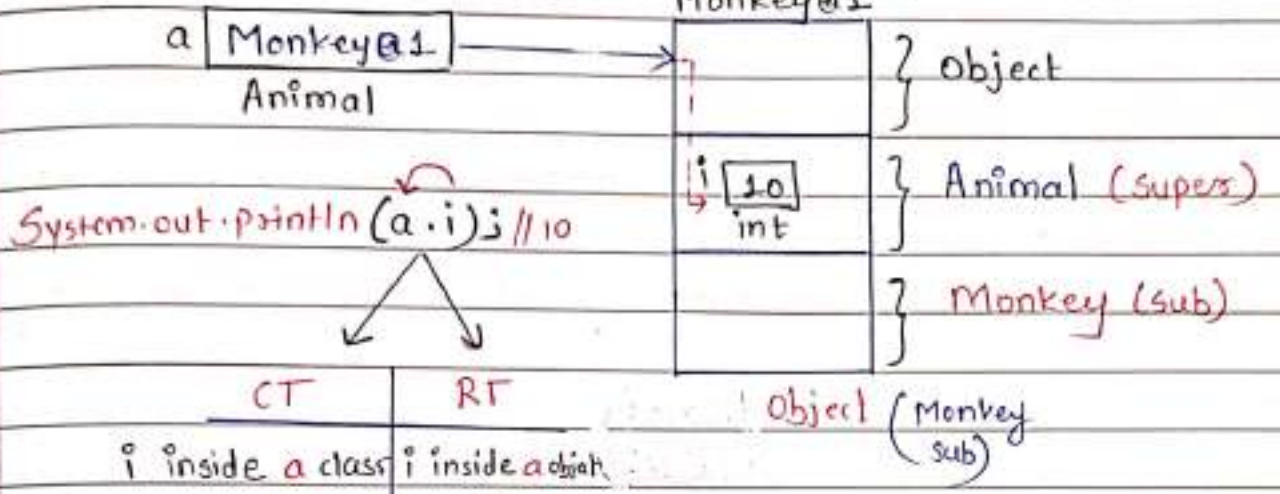


example:-

```
class Animal          // Super class
{
  int i = 10;
}
Sub class // class Monkey extends Animal
{
  int j = 20;
}
class Driver3
{
  public static void main(String[] args)
  {
    Animal a = new Monkey(); // upcasting
    // Super class a = new sub class();

    System.out.println(a.i); // 10

    System.out.println(a.j); // CTE
```

(Super) (Sub)

Animal a = new Monkey(); // upcasting



System.out.println(a.i); // 10

CT	RT
i inside a class ✓	i inside a object ✓

Object (Monkey sub)

System.out.println(a.j); // CTE

CT	RT
j inside a class ✗	

j is sub class variable (member)
we can not use sub class member inside super class
That's why we get compile time error.

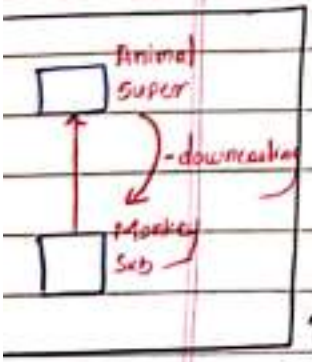
Page No
Date

* Why do we need Downcasting? and Advantages of Downcasting

→ when we perform upcasting we can not use members of sub class, To use members of sub-class than we should go for down casting (datatype var = (datatype) var;)

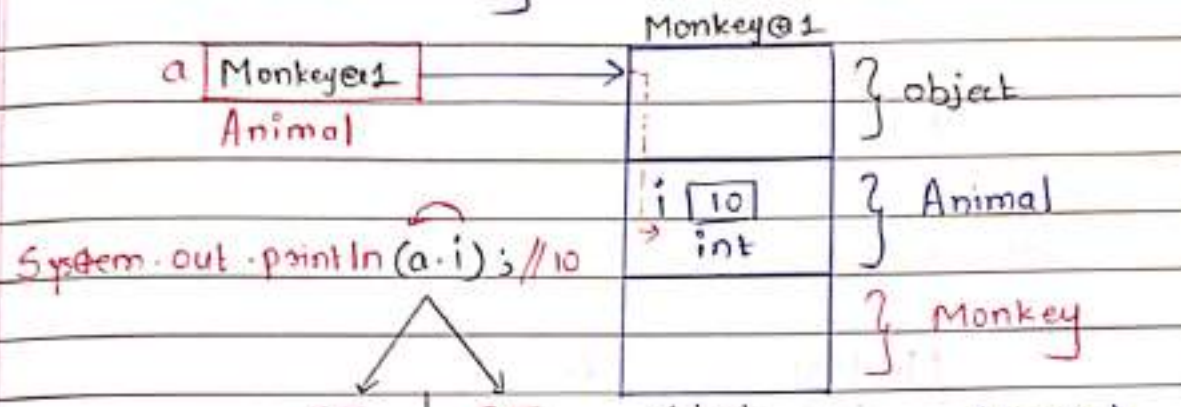
example:-

```
class Animal
{
  int i = 10;
}
class Monkey extends Animal
{
  int j = 20;
}
class Driver4
{
  public static void main (String[] args)
  {
    Animal a = new Monkey (); // upcasting
    System.out.println (a.i); // 10

comment // System.out.println (a.j); // CTE

    Monkey m = (Monkey) a; // downcasting
    System.out.println (m.j); // 20
  }
}
```

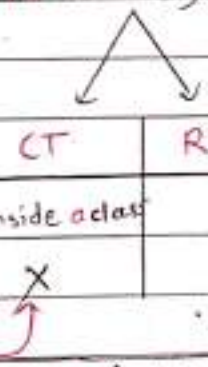


(Super) (Sub)

Step 1 `Animal a = new Monkey();` // upcasting



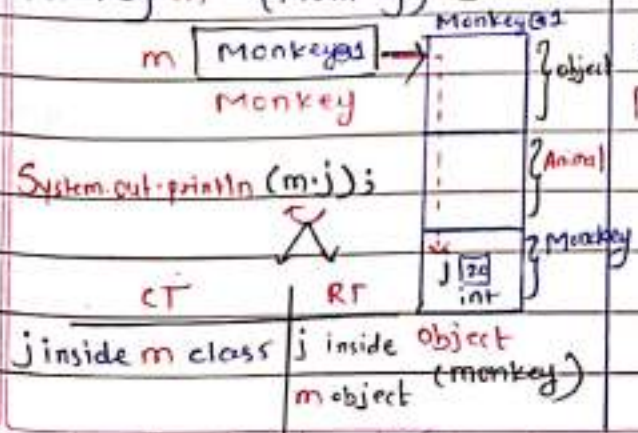
Step 2 `System.out.println(a.j); // CTE`

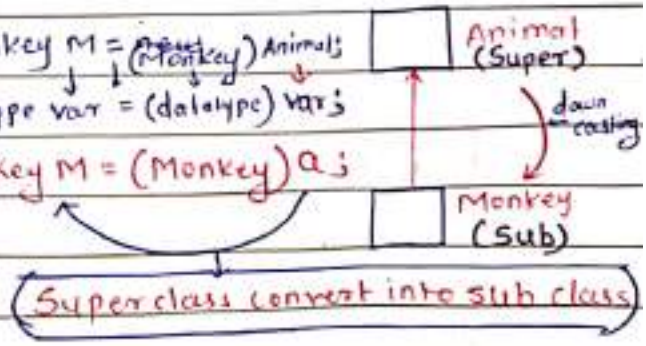


j is sub class member (we can not use sub class member inside superclass That's why we get compile time error)

If we want to use Sub-class member inside Superclass than we should go for downcasting (Syntax:- datatype var = (datatype) var;)

Step 3 `Monkey m = (Monkey) a;`





Q) what happens if we perform downcasting without upcasting ?

⇒

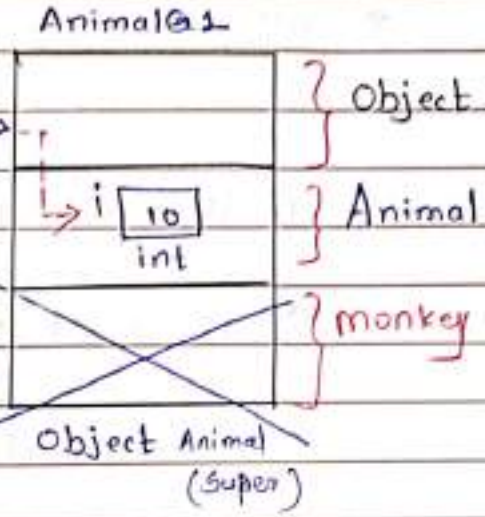
```
class Animal                                    // super class
{
 int i = 10;
}
class Monkey extends Animal                     // Sub class
{
  int j = 20;
}
 class Driver5
{
 public static void main (String[] args)
 {
  Animal a = new Animal ();                     // Super class
 System.out.println (a.i); // 10

  Monkey m = (Momkey) a                         // down-casting
 System.out.println (m.j); // CTE
}
}
```

Output:

```
10
// Compile Time error class Cast Exception //
```

> Note :- when we do downcasting without upcasting then we get classCast Exception

Animal a = new Animal();



a | Animal@1 → Animal

System.out.println(a.i); // 10

CT	RT
i inside a class ✓	i inside a object ✓

Monkey m = (Monkey) a;

m | Monkey@1 → monkey

System.out.println(m.j);

CT	RT
j inside m class ✓	j inside m object ✗

* Why do we need upcasting & Advantages of upcasting?

→ 1) it is used to achieve Generalization.

2) To avoid code-redundancy.

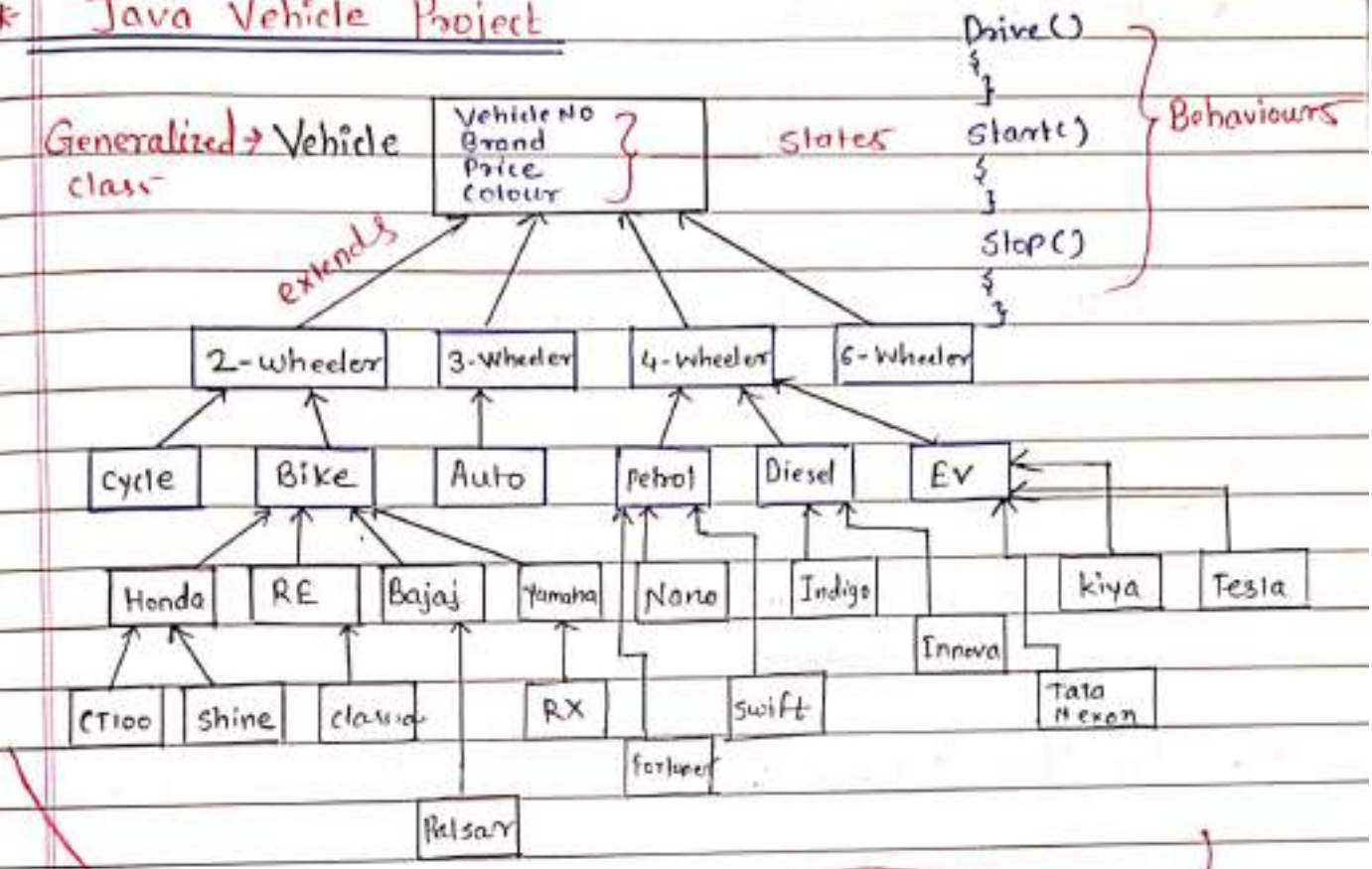
3) To restrict sub class members

* what is Generalization?

⇒ to creating generalized (Super most class) and specialized (sub class) class. this process is called as Generalization.

Generalized class → Super class
Specialized class → Sub class

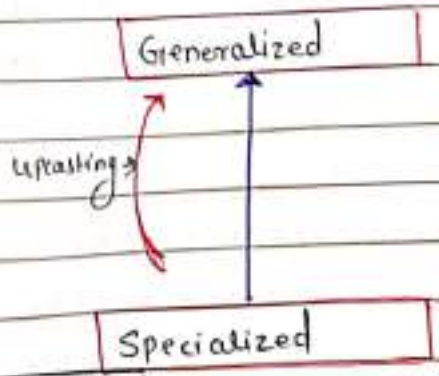
* Java Vehicle Project




```
Vehicle v = new Vehicle();
          = new Twowheeler();
          = new Threewheeler();
          = new fourwheeler();
          = new sixwheeler();
          = new Bajaj();
          = new Yamaha();
          = new Honda();
          = new Kiya();
```

Upcasting

Upcasting



* Note:- if we create referance variable for Super class we can store address of Sub-class with the help of upcasting

* Polymorphism

same name but different behaviours is known as polymorphism.

OR

Ability of an object to ~~execute~~ exhibit more than one form.

* Types of Polymorphism

1) • Compile Time Polymorphism / Compile Time Binding / Static Binding

2) • Run Time Polymorphism / Run Time Binding / Dynamic Binding

1) * Compile Time Polymorphism (CTP) :-

→ Binding happens during Compile time.

there are 4 types of example

1) variable shadowing
2) Method Shadowing
3) Method Overloading
4) Constructor Overloading

* Compile Binding

→ which variable to used is decided by compiler during Compile Time Based on

① reference Variable type ✓

② Not based on object creation ✗

* **Variable Shadowing** :- Sub class and Super class is having static variable and non-static variable with same name is known as Variable Shadowing

* **Rules To Achieve Variable Shadowing**

1) Inheritance is Manadatory
2) Sub class and Super class must have variable with same name

Example:- ~~Non~~ Static Polymorphism

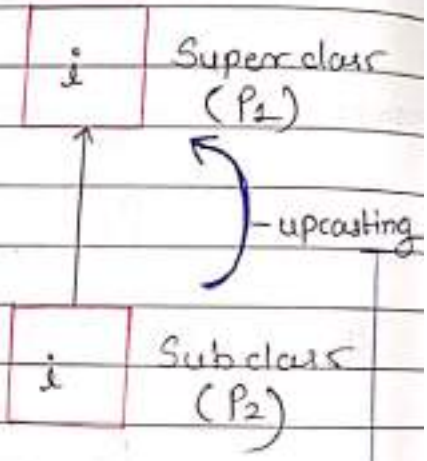
```
class P1
{
static int i = 10;
}
class P2 extends P1
{
static int i = 20;
}
class Driver1
{
   public static void main (String[] args)
{
   P1 ref1 = new P1 ();   // Super class
System.out.println (ref.i);

  P2 ref2 = new P2 ();   // sub class
System.out.println (ref2.i);

P1 ref3 = new P2 ();  // upcasting
System.out.println (ref3.i);
}
}
```

output :- i : 10
i : 20
i : 10

example 2) Non - Static polymarphism

→

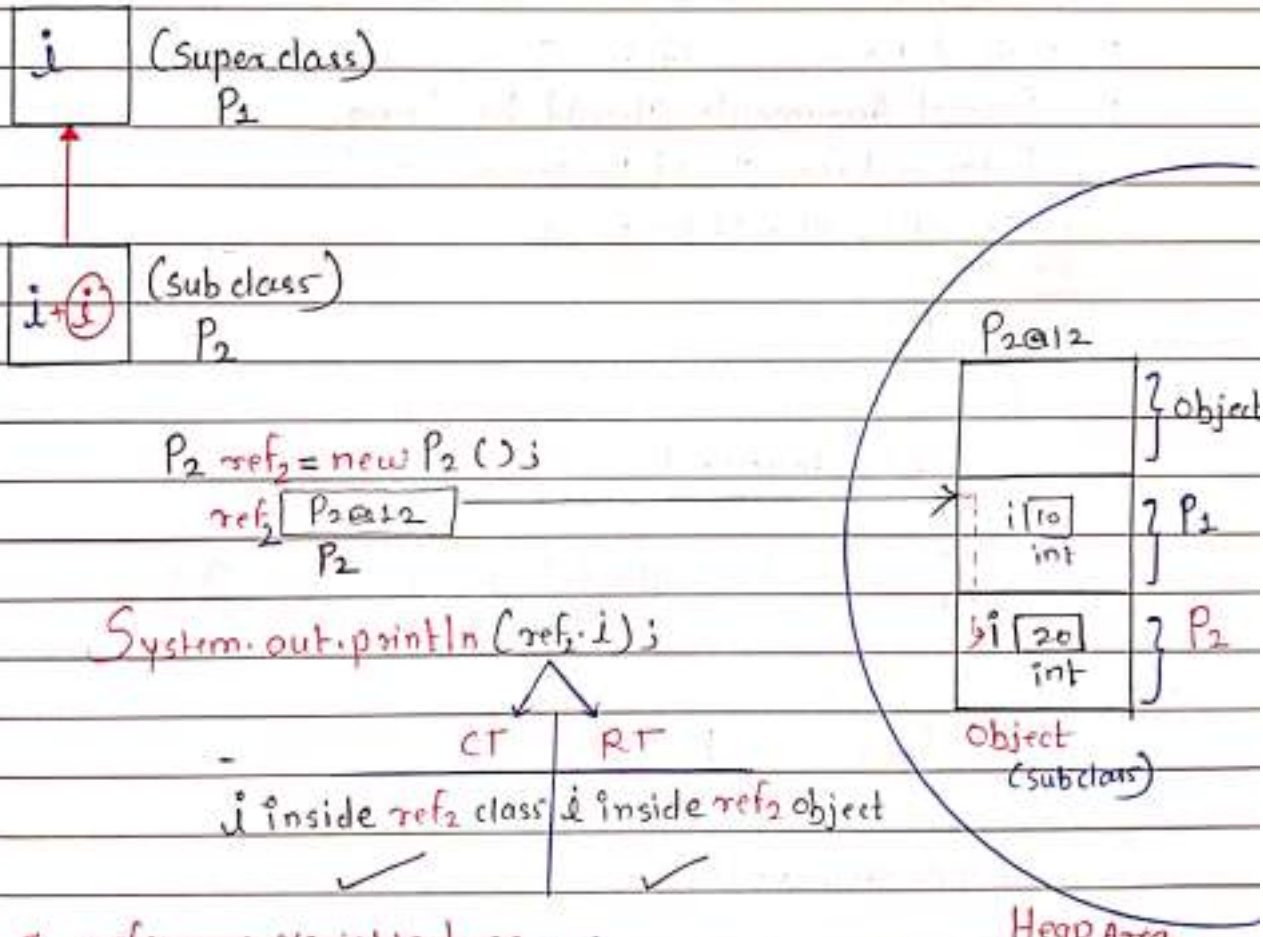
```
class P1
{
 int * i = 10;
}
class P2 extends P1
{
 int i = 20;
}
class Driver2
{
  public static void main(String[] args)
  {
    P1 ref1 = new P1();          // Super - class
   System.out.println(ref1.i);

    P2 ref2 = new P2();          // Sub - class
   System.out.println(ref2.i);

    P1 ref3 = new P2();          // upcasting
   System.out.println(ref3.i);
  }
}
```



output :-
10
20
10

Note :- inside Variable shadowing Variable name should be same
that's why, which variable to be used is decided by
Compiler during compile time. based on
1) reference variable type
2) Not based on object creation

Diagrammatic Representation



* reference variable type ✓
* object creation X

* Method Shadowing :- Sub class and Super-class is having static method with same name and same formal Argument is known as Method Shadowing

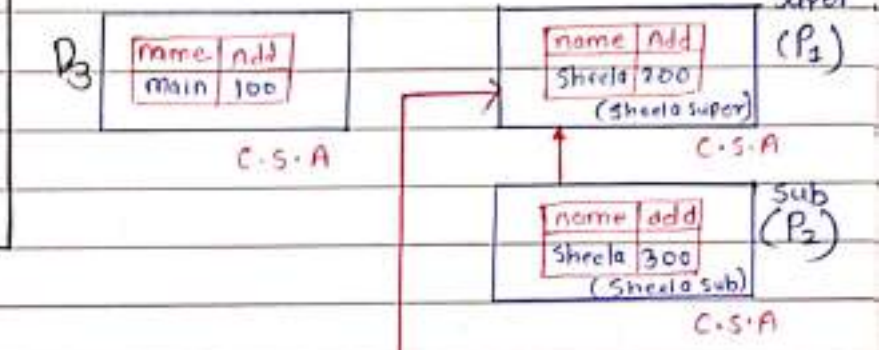
* Rules To Achieve Method Shadowing

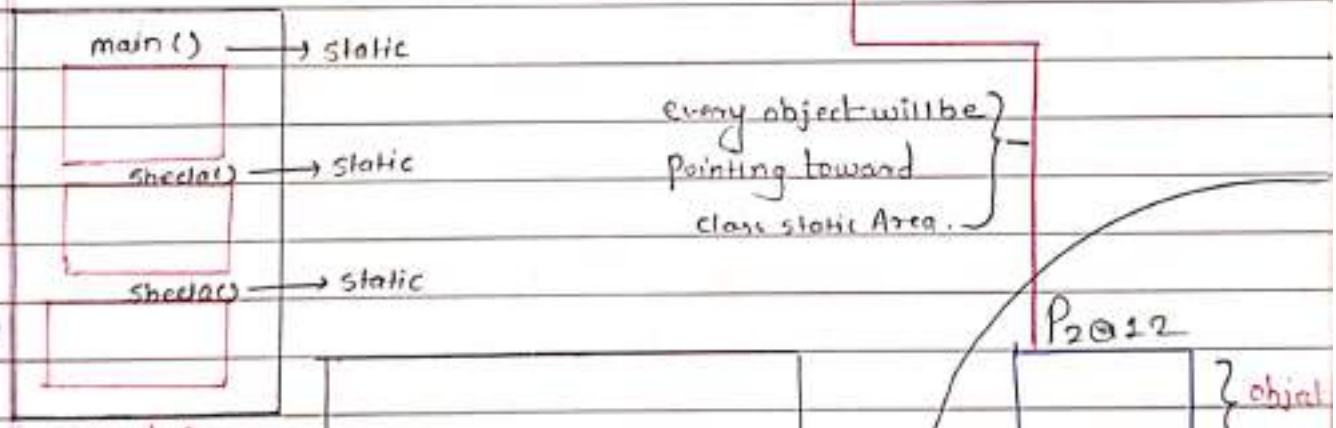
1) Inheritance is Manadatory
2) Sub-class and Super class must have static method with same name and same formal Argument.
3) Method Name should be Same
4) Formal Argument should be Same
5) Return type should be same
6) Access Specifier should be same.

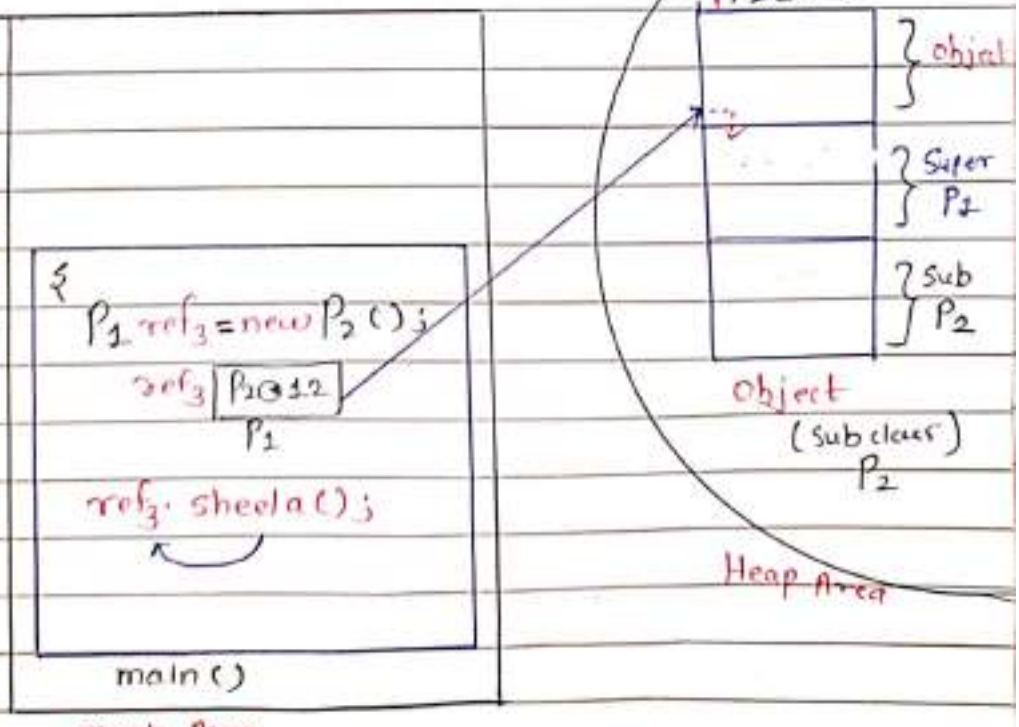
→
```
class P1
{
  public static void Sheela ()
  {
    System.out.println ("Sheela Super");
  }
}
class P2 extends P1
{
  public static void sheela ()
  {
    System.out.println ("Sheela Sub");
  }
}
class Driver3
{
  public static void main (String [] args)
  { P1 ref1 = new P2 (); // Super-class
    ref1.Sheela ();
```

```
P2 ref2 = new P2();   // Sub-class
ref2.sheela()

P1 ref3 = new P2();   // up casting
ref3.sheela();
}
}
```

→ Sheela Super
Sheela Sub
Sheela Super







Note:- inside method shadowing method some name should be That's why which method to be called is decided by compiler during compile Time. based on
1) reference var type ✓
2) Not based on object ✗

output:- Sheela Super

2) * Run Time Binding (RTB) :-

→ binding happens during Run time.

there are only one example.

- Method Overraiding

* Run time Binding :- which method to be called is decided during Run time based on

1) object creation ✓

2) not based on ref variable type. ×

* Method Overraiding :- Sub class non-static method Overraides Super class non-static method is known as Method Overraiding.

OR

Sub-class and Super class is having non-static method with the same name. And same formal Argument is known as Method overraiding.

Page No.
Date

* Rules to achieve Method Overraiding

1) Inheritance is manadatory
2) Sub class and super class should be have non-static method
3) Method name should be same
4) Return type should be same
5) Access Specifier should be same.

Q. Examples for Compile time polymorphism

→ Variable Shadowing
Method Shadowing
Method overloading
Constructor overloading

Q. Examples for Run time polymorphism.

→ method Overraiding

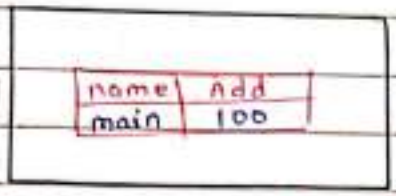
Page No.
Date

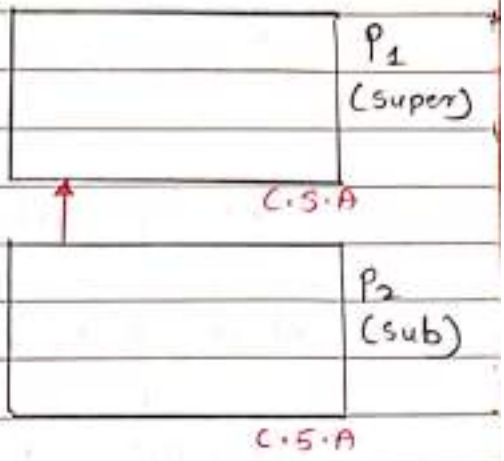
Example :- Method Overraiding

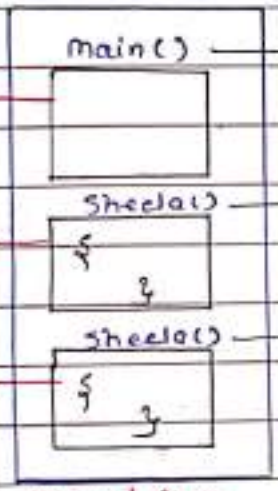
```
class P1
{
   public void sheela()
   {
    System.out.println("Sheela Super");
   }
}
 class P2 extends P1
{
   public void sheela()
   {
    System.out.println("Sheela sub");
   }
}
  class Driver4
{
   public static void main(String[] args)
   {
     P1 ref1 = new P1();   // Super class
       ref1.Sheela();

     P2 ref2 = new P2();   // Sub class
       ref2.sheela();

     P1 ref3 = new P2();   // upcasting
       ref3.sheela();
   }
}
```

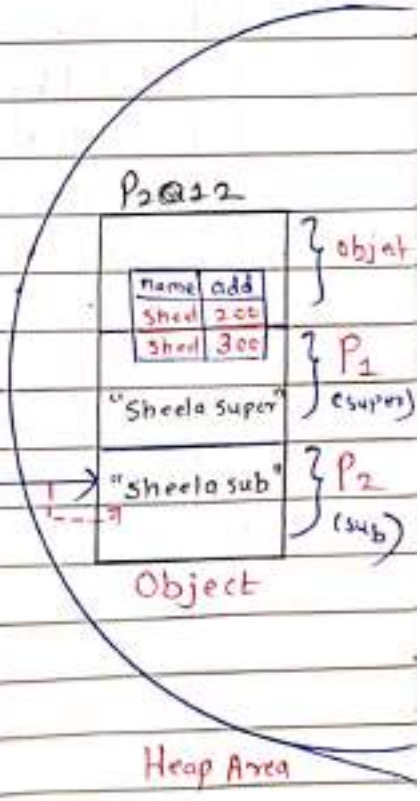
Output :- Sheela Super
Sheela sub
Sheela sub

Diagrammatic Representation

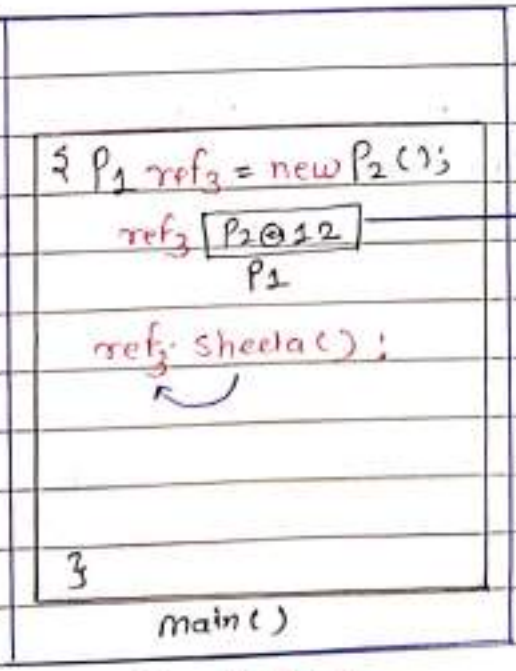








Note:- Inside method overriding which method to be called is decided during run time based on
1) object creation ✓
2) not based on ref variable type ✗



Output :- sheelasub

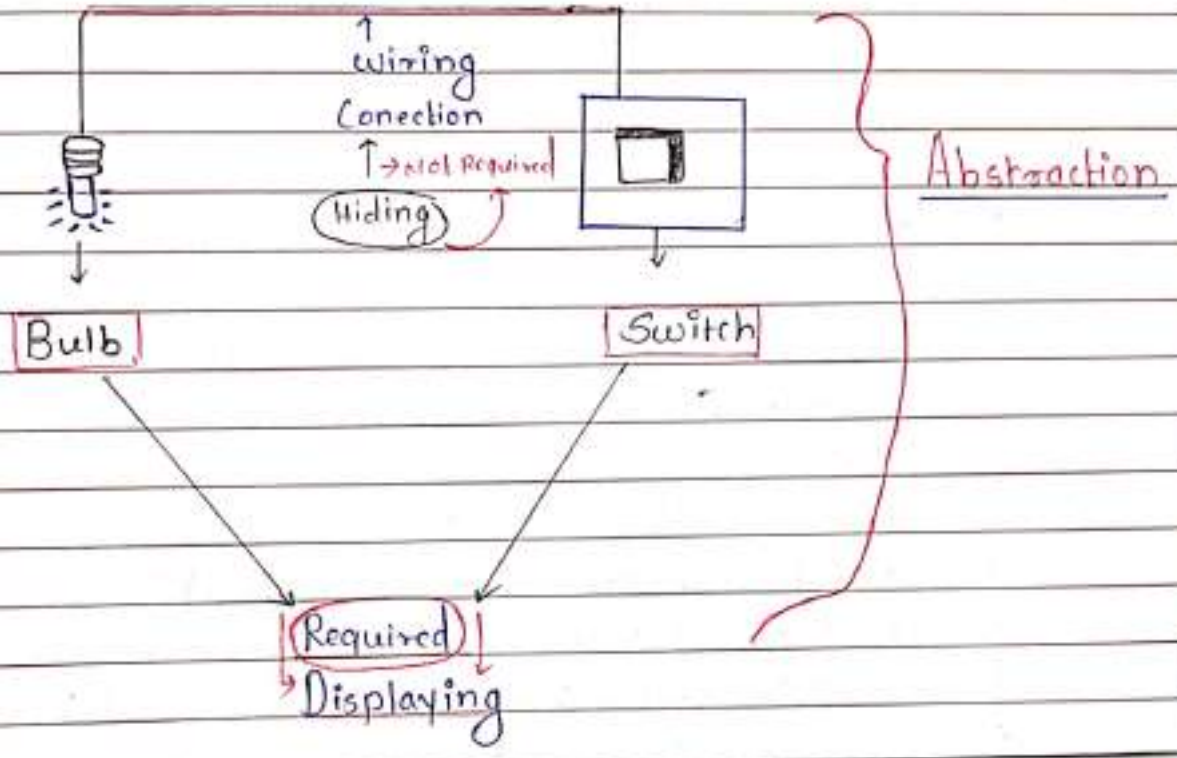
Page No.
Date

Q. difference between Method Overloading And Method overraiding

	Method OVerloading	Method OVerraiding
1)	class is having More than one method with same name but different Formal Argument either differ in length or differ in datatype	1) Subclass and Super class having non-static method with same name, same formal Argument.
2)	Formal Argument are different	2) formal Argument is same.
3)	Method can be static or non-static	3) Method should be non-static
4)	Inheritance is not mandatory	4) Inheritance is mandatory
5)	which method to be called is decided by compiler during compile time based on reference variable type.	5) which method to be called is decided during Run time based on object creation.
6)	Widening happens in method overloading	6) widening does not happens in method overraiding
7)	it is an example for Compile time polymorphism.	7) it is an example for Run time polymorphism.
8)	return type can or cannot be same	8) Return type must be same.
9)	It is occurs within same class	9) It is occurs within two class / different class
10)	upcasting is not Mandatory	10) upcasting is mandatory
11)	Object creation is Not Mandatory	11) object creation is Mandatory.

* ABSTRACTION

→ The process of Displaying the things which are Required and Hiding the things which are not Required is known as Abstraction.



1) Concrete Method

→ the method which has method body or method Logic, method Implimentation is known as Concrete method.

2) Concrete class

→ A class which has Concerete method is known as Concrete class.

1) Abstract Method

→ The method which does not have any method body or method logic, method Implimentation is known as abstract method.

2) Abstract class

→ The class which has abstract method is known as abstract class.

Note: ① we can not create object for abstract class but we can create ref variable.

② we can create object for concrete class Also we can create ref variable.

Q. How to Create Abstract Method.

→ ① method must be prefixed with abstract keyword
② method should ends with (;)
③ method should not have body, logic, and Implimentation
④ we can not overraide static method.
we can make only non-static method as abstract.

Q. why can't we Create object of abstract class?

→ Because ~~in~~ abstract class is having abstract method and abstract method does not have any body, logic, Implementation.

Page No.
Date

Q when do we make class as abstract

→ ① if the class is having at least one abstract method than the class must be abstract.

② if any class is Inheriting abstract method then the inherited class also should be abstract

③ if we don't want to make Inheriated class as abstract then we must overraide abstract method.

Q. why do we need abstract class ?

⇒ we need abstract class to achieve Abstraction In which we can create abstract method to hide implementation body / logic.

Q) difference between Abstract and concrete

Concrete method	Abstract method
① ~~method must be prefixed with concrete~~	① method must be prefixed with Abstract keyword
② method should have body, logic, Implentation	② method should don't have any body, logic, Implementation
③ we can create object and also Referance for concrete class	③ we can not create object but we can use only referance
④ method does not have any Prefix keyword	④ method should ends with(;)
⑤ can be static and ~~so~~ non-stati	⑤ we need abstraction ~~because~~ To hide the logic from the user. or implemention.
⑥ we cannot achieve abstraction we cannot hide implementation using concrete method	⑥ we can make only non-static method, as abstract.

```
abstract class Amaz
{
    abstract public void login();
}

class Amazon extends Amaz        // Concrete class
{
    public void login()
    {
        System.out.println("Providing logic/body/Implemetation");
    }
}

class Driver1
{
    public static void main (String[] args)
    {
        Amaz ref = new Amazon();    // upcasting
        ref.login();                // Method Overraiding
    }
}
```

output :- providing logic/body/Implementation

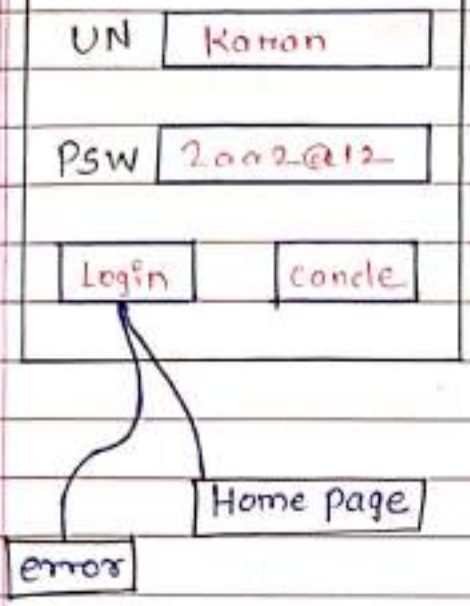
Q. Why do we need Overraiding?

→ because to provide logic

Page No.
Date

Java + DB ⇒ JDBC
(Java Database Connectivity)

Login (Amazon)



Concrete	abstract
Object ✓	Object X
reference ✓	reference ✓

Concrete
// complete class
class Amazon
{
// concrete
// complete method
public void login (String UN, String psw)
{
IF (UN == UNDB && PS == PSDB)
{
HM;
} body logic implementation
else
{
Error;
}
}
}

class block
Method block

abstract
// incomplete class
abstract class Amaz
{
// abstract
// incomplete method
public void login ();
abstract

X X X
Body logic implementation

}

user
costumer

Q. why do we need Abstraction
→ To hide logic from the user

Concrete class

* What Are The Members Can We Have In Concrete Class ?

→ class P1

{

1. Variable
   - Local variable
   - Global variable
     - Static Variable
     - Non-Static Variable

2. Method
   - Concrete method
     - Static Method
     - Non-Static Method
   - Abstract method

abstract method ⇒ non-static method

X X X X X

Not Possible inside normal class

3. Blocks
   - Static Initialization Block (SIB)
   - Instance Initialization Block (IIB)

4. Constructor

}

Note :- Concrete class can have only concrete method but it can not have abstract method.

abstract class

Page No.
Date

* What Are the Members can we have in abstract class

→ abstract class P1

{

1. variable
   - Local variable
   - Global Variable
     - Static Variable
     - non-static Variable

2. Method
   - Concrete Method
     - static method
     - Non-static method
   - abstract method

abstract method ⇒ non-static ✓ ✓ ✓

3. Blocks
   - Static Initialization Block (SIB)
   - Instance Initialization Block (IIB)

4. constructor

}

Note :- Abstract class can have abstract method as well as Concrete method

* Example :- abstract class

```
abstract class P1
{
    abstract public void test (); // abstract Method

    Public void sheela ()  // concrete Method
    {
     System.out.println ("Concrete Method");
    }
}
```

Q. Why Do Wee Need abstract class ?
OR
Advantages of abstract class ?

→ * to achieve abstraction in which we can create abstract method to hide method body, Method logic, or method Implementation.

* DrawBack Abstract class
we can not achieve 100% abstraction by using abstract class
because abstract class can have concrete method.

Q. why cannot we achieve abstraction 100% ?
→ because abstract class can have concrete method. that's why we can not achive 100% abstraction

Q. How to achieve 100% Abstraction ?
→ by using Interface.

Page No.
Date

* Interface

- interface is a Keyword
- interface is use to create non-primitive datatype.
- interface is use to achieve 100% Abstraction
- interface is use to achieve Multiple Inheritance.

* Why Do We Need Interface ?

OR

What is Advantages of Interface ?

there are two Advantages

1) To achieve 100% Abstraction.
2) To achieve Multiple inheritance

Note :- we can not create object for interface but we can create referance variable for interface

Note :- in interface by default all the method are Prefixed with abstract and public.

example :-

interface I1	abstract interface I1
{	{
void test(); →	abstract public void test();
}	}
Befor Compilation	After compilation

Page No.
Date

Q. How to Achieve 100% Abstraction

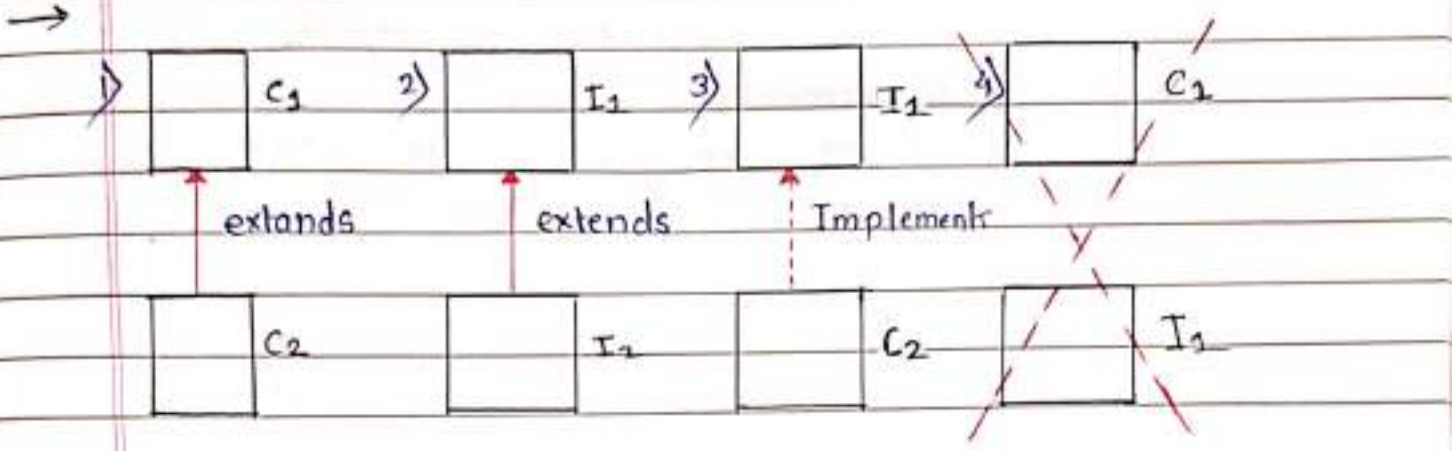
→

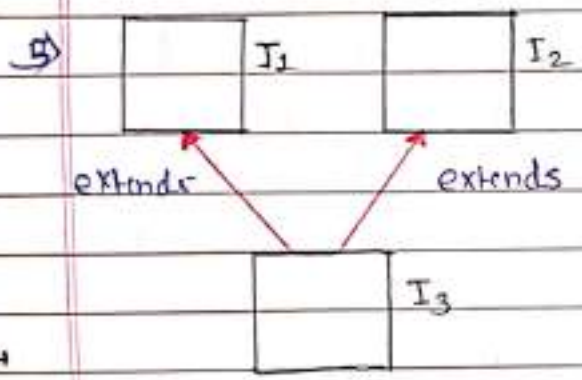
```
interface Amaz
{
    void login();                          // abstract method
}

class Amazon implements Amaz               // implementing class
{
    public void login()                    // concrete method
    {
      System.out.println("providing logic/body/implementation");
    }
}

class Driver1
{
    public static void main(String[] args)
    {
      Amaz ref = new Amazon();             // upcasting
      ref.login();                         // Method overriding
    }
}
```

↑ which method to be called is decided by during Run time base on object creation

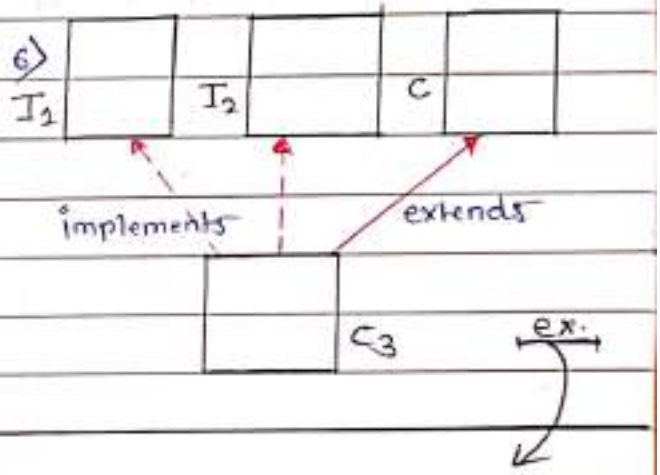
output:- providing logic/body/implementation

* Inheritance with Respect to Interface





ex.



ex.

Note:- (1) interface can not inherite class

(2) one interface can inherite multiple interface with the help of extends

```
interface I1
{
}
interface I2
{
}
interface I3 extends I1 , I2
{
}
```

```
interface I1
{
}
interface I2
{
}
class C
{
}
class C3 extends C implements I1,I2
{
}
```

Note:- A class can extends only one class but it can implements multiple Interface

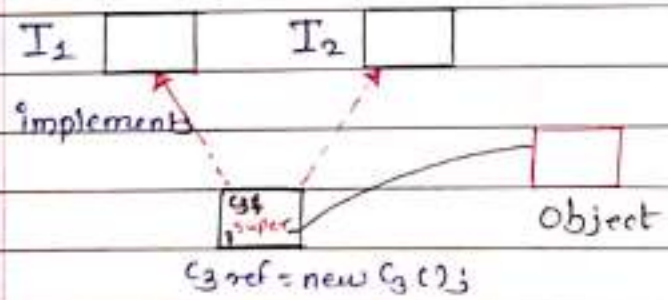
Note :- ① A class can extends only one class but it can implements multiple interface

② always we should write extends first then implements

③ one interface can inherite multiple interface with the help of extends

Q. why multiple inheritance is possible in interface why it is not possible in class

→ because 1) constructor are not allowed
2) concrete method are not allowed
3) Variable are not allowed.
4) blocks are not allowed
5) No confusion with Respect to loading process Since programmer specify which interface to be implemented first.



class C3 implements I1, I2

{

}

1) C.I.P ✗
2) O.I.P ✗
3) A.M ✗

class C3 implements I1, I2, I3

Page No.
Date

Q. example program to multiple Inheritance.

```
interface I1
{
  void test();
}
interface I2
{
  void test();
}
class C3 implements I1, I2
{
  public void test()
  {
    System.out.println("providing logic/body/Implementation");
  }
}

class Driver3
{
  public static void main(String[] args)
  {
    I2 ref = new C3();          // upcasting
    ref.test();                 // Method overraiding
  }
}
```

output :- providing logic/body/Implementation

* Difference Between abstract class And interface

	abstract class		interface
1)	we can not achieve 100% abstraction by using abstract class	2)	we can achieve 100% abstraction by using interface
2)	abstract class can have abstract method & concrete method.	3)	interface can not have concrete method, but it can have abstract method
3)	variable are allowed	3)	variable are not allowed
4)	constructor are allowed	4)	constructor are not allowed
5)	A class can inherit abstract class using extends	5)	A class can inherit interface by using implements
6)	we can not achieve multiple inheritance	6)	we can achieve multiple inheritance.
7)	A class can implements one or more interface.	7)	interface can extends one or more interface but it can not inherit class
8)	Blocks are allowed	8)	Blocks are not allowed.
9)	Abstract class must be prefixed with abstract keyword.	9)	Need Not be prefixed with keyword.
10)	abstract class can inherit one class	10)	Inherit n no. of interface.
11)	In abstract class explicitly we need to add abstract keyword for a method.	11)	By default abstract will be prefixed.
12)	In abstract class explicitly we need to add public access Specifier.	12)	By default public will be prefixed.

Page No.
Date

* Object class

→ i) it is a super most class of java.

ii) It is inbuilt class which is defined in java.lang package Hence we don't have to import it.

iii) Object class has many methods which are required to perform someaction in all the classes.

Note :- In object class there are 11th non-static method are present.

1) toString()

2) equal (Object obj)

3) hash code ()

4) clone ()

5) finalize ()

6) wait ()

7) wait (long)

8) wait (long, int)

9) notify ()

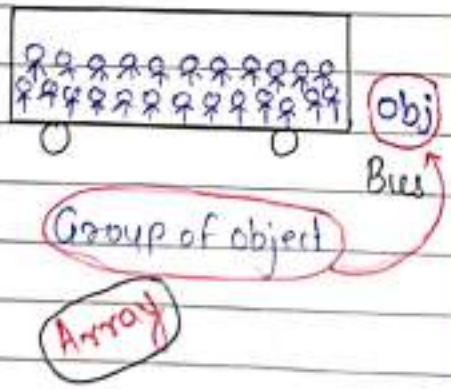
10) notifyAll ()

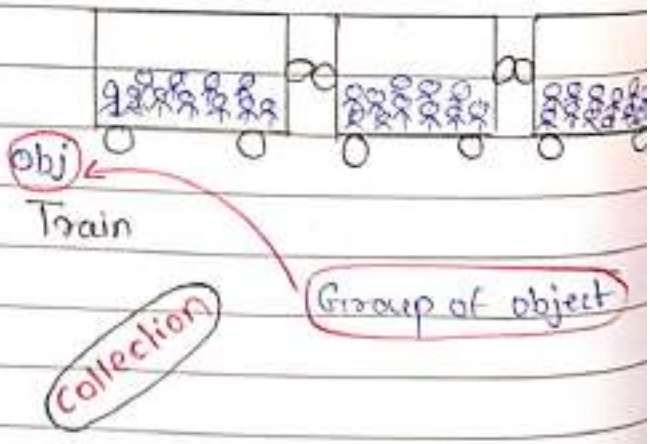
11) get class()

* Difference between Array And Collection

→

Array	Collection
① Size is fixed	① Size is not fixed
② & can store same type of object/elements (Homogenous)	② Same as well as different type of elemen object (Heterogenous)
③ we don't have in-built or ready-made methods to perform CRUD, sorting & Searching	③ we have inbuilt or ready-made methods to perform CRUD, Searching & Sorting
④ To find no of elements or Object in array we have variable length	④ To find no of elements or object in collection we have method size()



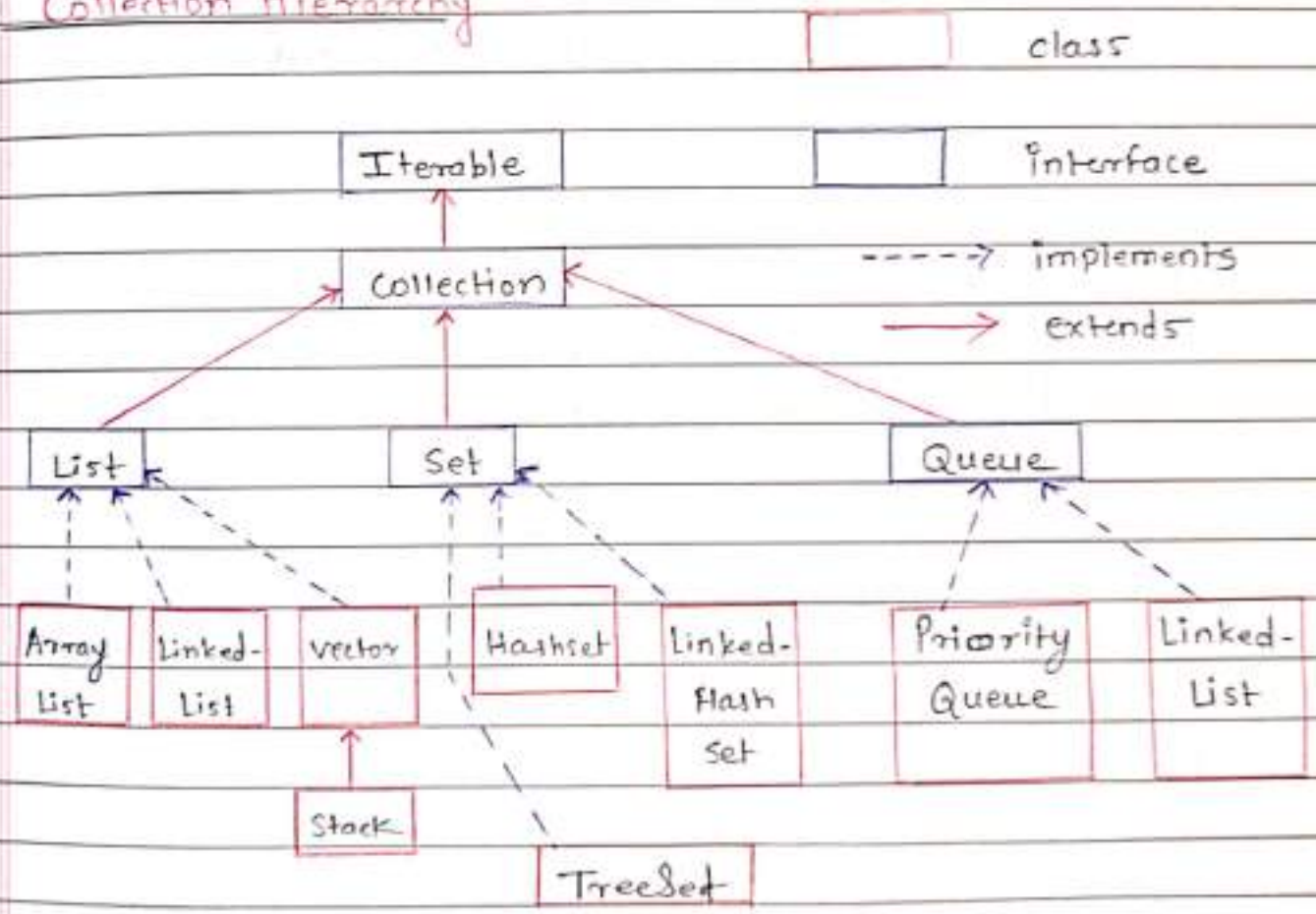


* **Collection** :- Group of object present inside an object.

* **Framework** :- ① Framework is structure or hierarchy
② Framework consist of many inbuid classes and interfaces.

* **Collection Framework** :- it is set of classes and interfaces which provides a mechanism to perform CRUD operation, Searching, and sorting operation on group of object.

* **Collection Hierarchy**



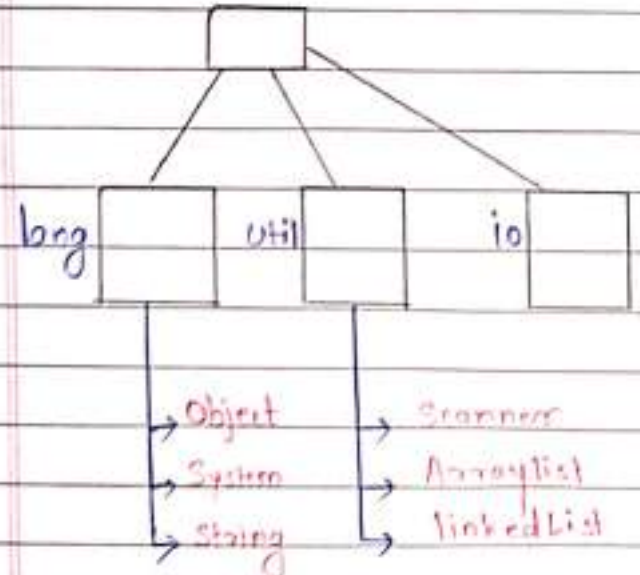
Page No
Date

* Characterstics of List

1) It is heterogenous allowed
2) Size is not-fixed (Growable type)
3) List maintain index
4) List maintain insertion order (FIFO / LILO)
5) Random Access are allowed
6) Duplicate object are allowed
7) Null object are allowed.

Arraylist :- ① it is a implementing class of List interface.
② characterstics of Arraylist

Note :- all the classes and interfaces present in collection hierarchy by default present inside java.util package. Hence it is manadatory to input import.



Page No.
Date

Syntax :- import package name . className
import java.util . Arraylist

* import java.lang . String
import java.lang . System

Note :- if any class is present inside lang package then it is not manadatory to import by default it will be imported.

* Inbuilt non-static members of Array list

1) To add an object :-
- add (object)
- addAll (collection)
- add (int index . object)
- addAll (int index . collection)

2) To remove objects :-
- remove (object)
- remove (int index)
- removeAll (collection)
- clear ()

3) To search object :-
- contains (object)
- containsAll (collection)

4) To access objects
- get (int index)
- iterator ()
- listIterator ()
- For each loop

5) Other methods :-
- size ()

Page No.
Date

Program ⇒ add (object), size ()

```
Package Collection;
import java.util.Arraylist;
public class C1
{
    public static void main (String[] args)
    {
        Arraylist a = new Arraylist ();
        System.out.println (a);
        a.add (10);
        a.add (20);
        a.add ("hi");
        a.add (10);
        a.add (10);
        a.add (null);
        a.add ('c');
        a.add (false);
        a.add (15.5);
        a.add (null);
        System.out.println (a);
        System.out.println (a.size());
    }
}
```

output :- []

[10, 20, hi, 10, 10, null, c, false, 15.5, null]

9

Program C2

```
package Collection;
import java.util.ArrayList;
public class C2
{
    public static void main (String[] args)
    {
        ArrayList s = new ArrayList ();
        s.add ("dosa");
        s.add ("vada");
        s.add ("samber");
        s.add ("uttapa");
    System.out.println ("South :" + s);

    ArrayList m = new ArrayList ();
        m.add ("vadapav");
        m.add ("puranpoli");
        m.add ("daal mundi");
        m.add ("popti");
        m.add ("jawala");
    System.out.println ("soMaharashtrian :" + m);
// s.add ("Biryani");
    s.add (0, "Biryani");
    System.out.println ("South :" + s);

// m.addAll (s);
    m.addAll (3,s);
System.out.println ("maharashtrian :" + m);

// m.remove ("daal mundi"); // by using object
    m.remove (s); // To remove only one object // by using index.
System.out.println ("maharashtrian :" + m);
```

// To Search

```
System.out.println ("Search : " + m.contains ("dosa"));
System.out.println ("Search : " + s.contains ("jawala");
System.out.println ("Search : " + m.containsAll (s));

    m.clear ();   // To clear all the object
System.out.println (" maharashtrian : " + m);
}

}
```

output:-

```
South : [ dosa, vada, samber, uttapa ]
maharashtrian : [ vadapav, puran poli, daal mundi,
                  popti, jawala ]
South : [ biryani, dosa, vada, samber, uttapa ]
Maharashtrian : [ vada pav, puran poli, dal mundi,
                  biryani, dosa, samber, uttapa, popti,
                  jawala ]
Maharashtrian : [ vada pav, Purn poli, daal mundi,
                  Biryani, dosa, samber, uttapa,
                  popti, jawala ]
Search : true
Search : false
Search : false
maharashtrain : [ ]
```

Q How to Access Collection ?

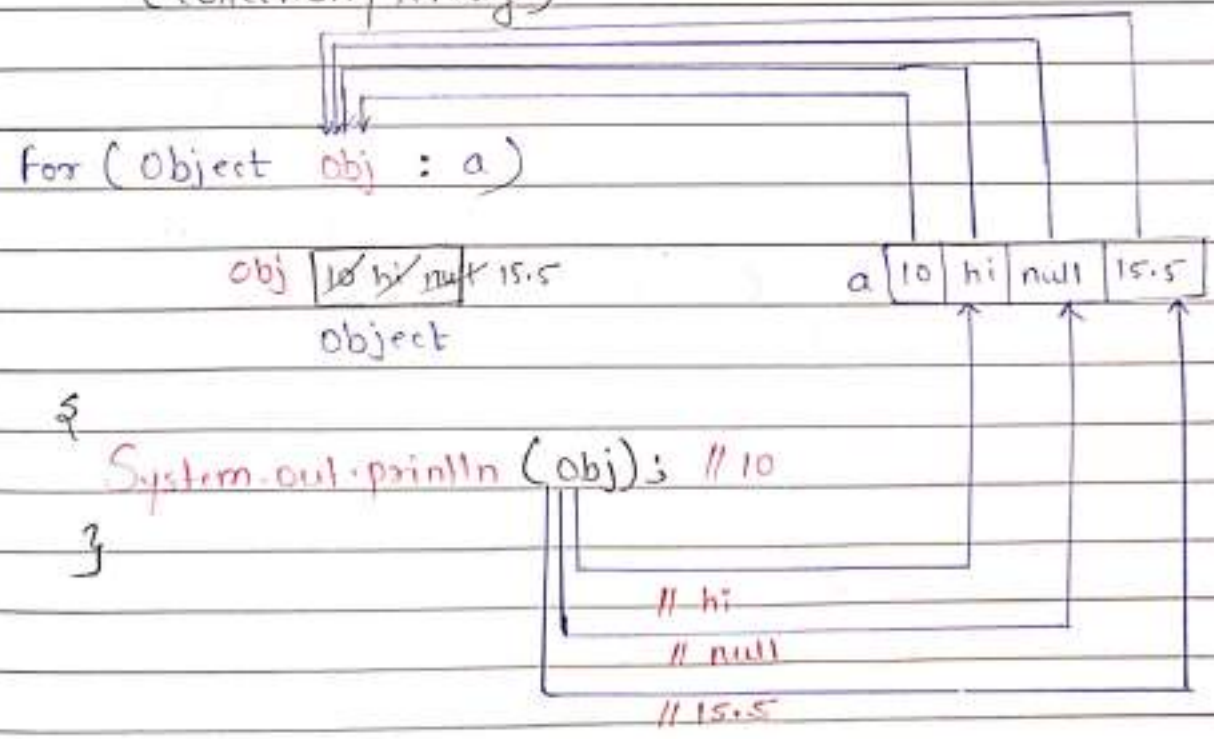
→ 1) by using get (int index)

2) by using for each loop

Syntax :- For each loop

```
For (Object obj : collection_ref)
{
  System.out.println (obj);
}
```

Note :- 1. it has only 2 Segments
2. Each segments have been seperated by colon (:)
3. No updation and No condition segments
4. For-each loop is used to access only non-primitive (Collection / Array)



```
package collection;
import java.util.Arraylist;
public class C3
{
    public static void main (String[] args)
    {
        Arraylist a = new Arraylist ();

        a.add (10);
        a.add ("hi");
        a.add ( null);
        a.add ( 15.5);
    System.out.println (a);

System.out.println ("==== get (index) ====");

    for (int i = 0; i<a.size(); i++)
    {
      System.out.println (a.get(i));
    }
      System.out.println ("==== for each ====");
      For ( Object obj : a)
        {
          System.out.println (obj);
        }
      }
    }
```

output:- [10, hi, null, 15.5]

==== get (index) ====

10

hi

null

15.5

==== for each ====

10

hi

null

15.5

* Difference between For loop and for each loop?

For loop	For each loop
1. It has '3' segments	1. It has only '2' segments
2. Each segment have been seprated by semicolon (;)	2. each segment have been seprated by (:) colon.
3. updation & condition are required.	3. No updation & no condition segments.
4. For loop is used to access primitive (collection/Array)	4. For each loop is used to access only non-primitive (collection/Array)

* iterator ()

iterator interface has two methods

① next ();

② hasNext ();

1. next ();

It is used to get current object & moves curser to the next object in forward direction.

2. hasNext ();

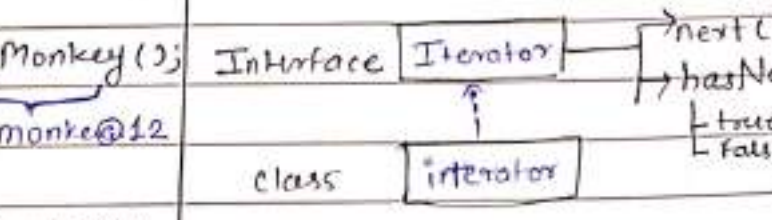
It is used to check whether the collection has object or not
It returns true if there is next object
else it returns false.

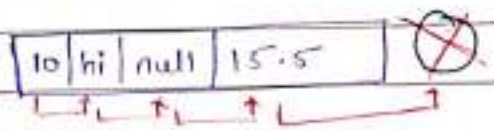
```
class Monkey
{
  Public sheela()
  {
    return new Monkey();
  }
}
Monkey i = a.sheela();
```

monkey@12 (new Monkey())
monkey@12 (a.sheela())

```
iterator()
{ return new interator(); }
Iterator i = a.iterator();
```

Iterator@12 (a.iterator())





1) System.out.println(i.next()); // 10
2) System.out.println(i.next()); // hi
3) System.out.println(i.next()); // null
4) System.out.println(i.next()); // 15.5
5) System.out.println(i.next()); // Exception RTE

Page No.
Date

ListIterator()

ListIterator interface has four methods

1. next();
2. hasNext();
3. previous();
4. hasPrivious();

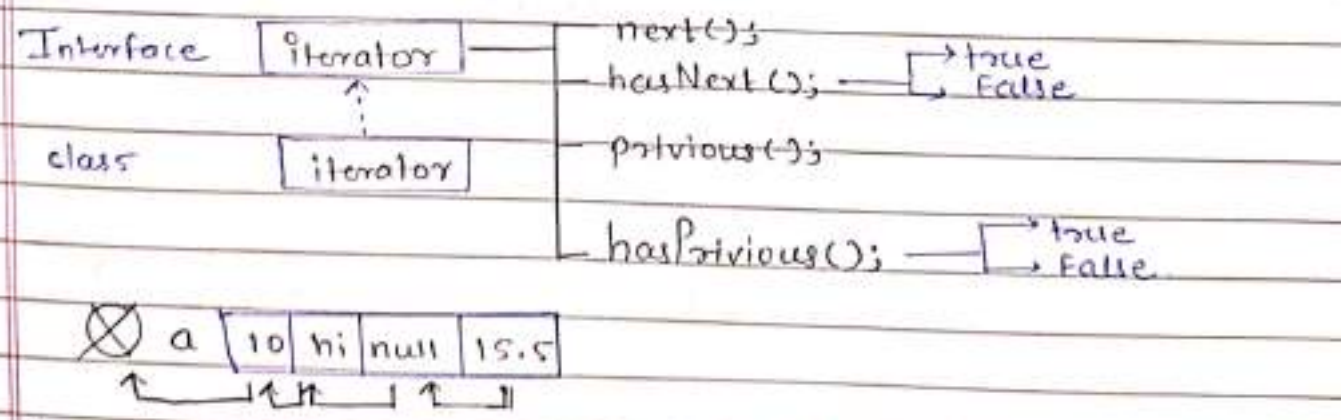
③ previous();

It is used to get current object & mouse cursor to the privious object in backward direction.

④ hasPrivious();

It is used to check whether the collection has privious object or not.

It returns true if there is privious object else it returns false.



1) System.out.println (i.hasNext()); // 10
2) System.out.println (i.hasNext()); // hi
3) System.out.println (i.hasNext()); // null
4) System.out.println (i.hasNext()); // 15.5
5) System.out.println (i.hasNext()); // false

(i.hasPrivious()); // 15.5
(i.hasPrivious()); // null
(i.hasPrivious()); // hi
(i.hasPrivious()); // 10
(i.hasPrivious()); // false

```
package collection;

import java.util.Arraylist;
import java.util.Iterator;
import java.util.ListIterator;

public class C4
{
  public static void main (String[] args)
  {
    ArrayList a = new ArrayList ();
    a.add (10);
    a.add ("hi");
    a.add ( null );
    a.add ( 15.5);
System.out.println (a);

System.out.println ("====iterator====");
  Iterator i = a.iterator ();
while ( i.hasNext ())
   {
     System.out.println ( i.next ());
   }
System.out.println ("====listIterator=====")
                                  Forward

  ListIterator j = a.listIterator ();
while ( j.hasNext ())
   {
     System.out.println ( j.next ());
   }
```

```
System.out.println ("==== listIterator() Backward ====");
while (j.hasPrivious())
  {
    System.out.println (j.privious());
  }
 }
}
```

output:- [10, hi, null, 15.5]

==== iterator ====

10

hi

null

15.5

==== listIterator() forward ====

10

hi

null

15.5

=== listIterator() backward ====

15.5

null

hi

10

Q How to sort list or Array list

→ by using static method of collections class i.e sort sort(in);

```
package collections;
import java.util.ArrayList;
import java.util.Collections;
Public class Cs {
Public static void main (String[] args)
{
    ArrayList a = new ArrayList();
        a.add(10);
        a.add(-5);
// a.add("hi"); ClassCastException
// a.add(null); NullPointerException
        a.add(200);
        a.add(-100);
        a.add(0);
        a.add(150);
System.out.println("Before sorting :"+ a);
    Collections.sort(a);
System.out.println("After sorting ASC :" + a);
    Collections reverse(a);
System.out.println("after sorting DESC:"+ a);
}
}
```

Output :-

Page No.
Date

* Difference between collection and Collections

Collection	Collections
1. Collection is a interface	1. collections is a class
2. Collection interface has many sub interfaces such as list, set, Queue and their implementing classes.	2. collections class has static method to sort list i) sort (list) ii) reverse (list)

* Difference between Array List and Vector

Array List	Vector
1. it is present frome since JDK 1.2	1. it is present from since JDK 1.0
2. it is not legacy	2. it is legacy class
3. initial capacity is 10 Increment capacity is increase by 15% from current capacity $Ic = \left(cc \times \frac{3}{2}\right) + 1$	3. initial capacity is 10 Increment capacity is 100% of current capacity. $Ic = cc \times 2$
4. Array List is faster than Vector.	4. vector is slow

```
package collection
import java.util.vector.
import java.util.collections;
public class C5
{
    public static void main (String[] args)
    {
        vector a = new vector ();
        a.add (10);
        a.add (20);
        a.add (-5);
        a.add (-6);
    System.out.println (a);
```

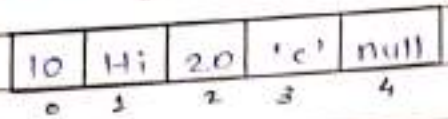
output :- [10 20 -5 -6]

Page No.
Date

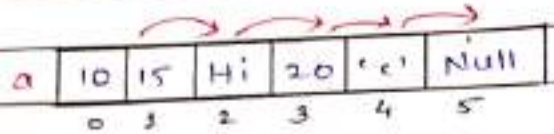
* Difference between ArrayList and LinkedList

ArrayList	LinkedList
1. Array list follows Array data structure.	1. Linked List follows Doubly LinkedList Data Structure.
2. object is stored in the form of continuous block of memory	2. object is stored in the form of node.
3. It has initial and increment capacity	3. It has does not have any initial and increment capacity
4. Arraylist is an implementing class of list interface	4. Linked list is an implementing class of List and Queue interface
5. Shifting operation is present	5. Shifting operation is not present.
6. Array list is not good for adding and removing operation	6. Linkedlist is good for adding and removing operation because there is no shifting operation
7. It is used to search and access object	7. It is not suitable to search and access object

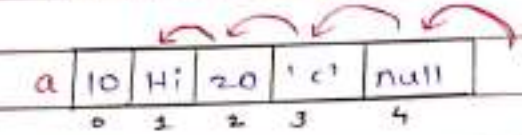
For Arraylist



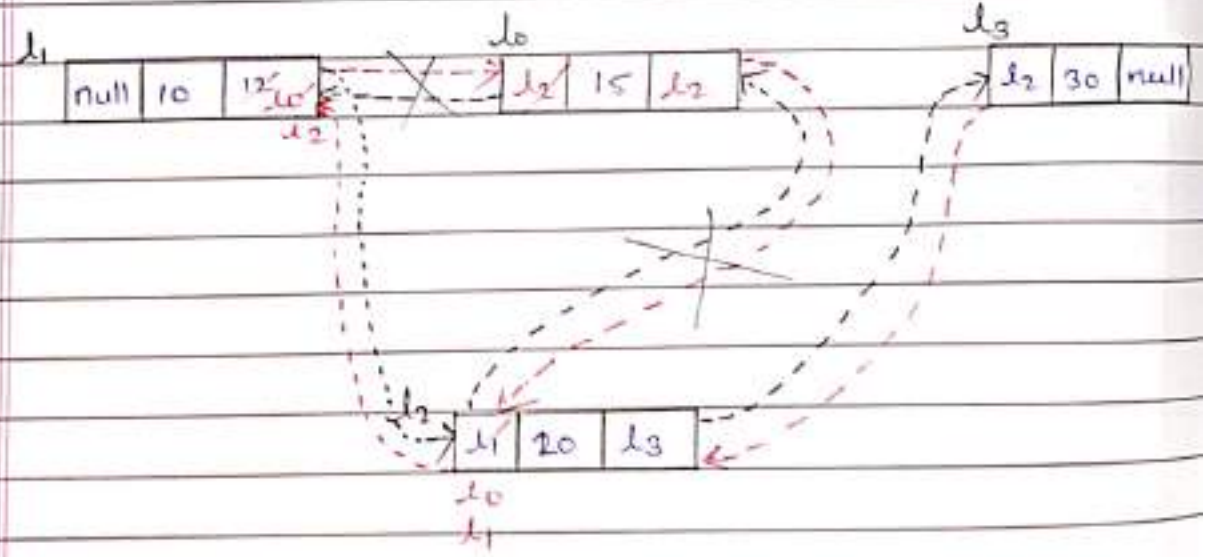
a.add (1, 15);



a.remove (2);



In Linkedlist (NODE)



(10, 15, 20, 30)

0 1 2 3

Page No.
Date

```
package collection;

import java.util.LinkedList
import java.util.collections

Public class C5
{
   public static void main (String[] args)
   {
      LinkedList a = new LinkedList();
      a.add(10);
      a.add(20);
      a.add(15);
   System.out.println(a);
   }

}
```

output :- ~~Listlink~~ Linkedlist [10, 20, 15]

Page No.
Date

* Difference between List And Set

List	Set
1. Duplicate object are allowed	1. Duplicate object are not allowed.
2. insertion order are allowed (list maintain insertion order)	2. insertion order Not allowed.
3. List maintain Index.	3. Index are not allowed.
4. Any no. of Null object allowed	4. only one null object are allowed
5. Random Access are allowed	5. Random Access are Not allowed.
6. It can be add and Remove operation by using index.	6. it can not be add and Remove operation because does not have index.
7. List can be Accessed in 4 ways 1) get (index) 2) for each loop 3) iterator () 4) listIterator ()	7. Set can be Accesed in 2 ways 1) for each loop 2) iterator ()

Page No.
Date

* Difference between HashSet, LinkedHashSet, TreeSet

HashSet	LinkedHashSet	TreeSet
1) It does not maintain Insertion order.	1) It is maintains Insertion order	1) By default sort object in ASC order.
2) Hetrogenous are allowed.	2) Hetrogenous are allowed	2) (Hetrogenous object are not allowed) - Hel (only Homogenous object are allowed)
3) one null object are allowed	3) one null object are allowed.	3) one null object are also Not allowed. (NullpointerExcepHon)

Program Set

```
package collection;
import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.TreeSet;
public class Cs
{
    public static void main(String[] args)
    {
        HashSet h = new HashSet();
        h.add(10);
        h.add("hi");
        h.add(10);
        h.add(Null);
        h.add('c');
        h.add("hi");
        h.add(15.5);
        h.add(Null);

    System.out.println("HashSet :" + h);

      LinkedHashSet L = new LinkedHashSet();
      L.add(10);
      L.add("hi");
      L.add(10);
      L.add(Null);
      L.add('c');
      L.add("hi");
      L.add(15.5);
      L.add(Null);

System.out.println("LinkedHashSet :" + L);
```

```
TreeSet t = new TreeSet();
    t.add(10);
    t.add(-5);
//  t.add("hi");  classCast Exception
//  t.add(Null);  NullpointerException
    t.add(100);
    t.add(-200);
    t.add(25);
    t.add(0);

System.out.println("TreeSet ASC :" + t);

System.out.println("TreeSet DESC :" + t.descendingSet());

}

}
```

Output :- [Null, hi, 10, c, 15.5]

[10, hi, 10, Null, c, 15.5]

[-200, -5, 0, 10, 25, 100]

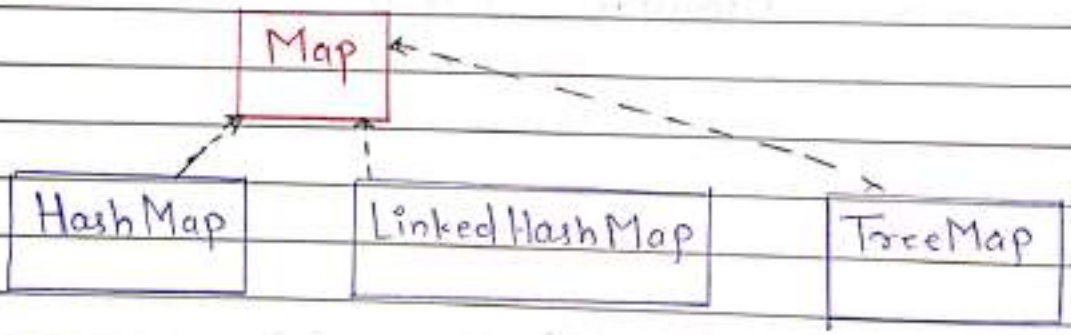
[100, 25, 10, 0, -5, -200]

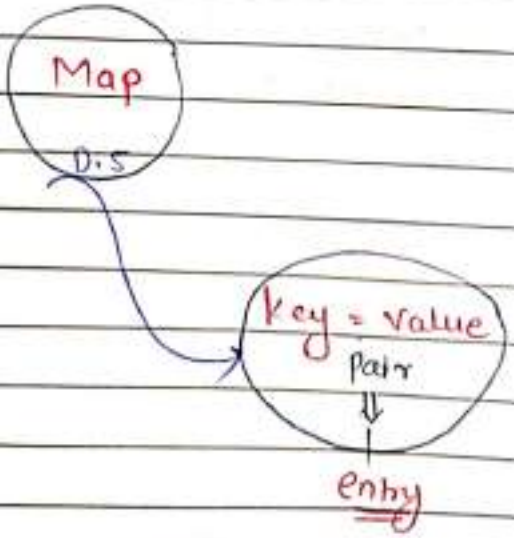
Page No.
Date

* **Map :-** 1) map is data Structure in which object will be Stored in the form of key and value pair.

2) Map is an interface which has Implementing classes. Such as HashMap, LinkedHashMap, TreeMap.

Note :- 1) key must be unique
2) value can be duplicate
3) key and value pair is known as Entry





Unique Key empId	Duplicate value Ename
1	Sheela
2	Sundri
3	Sheela.

Page No.
Date

* Difference between HashMap, LinkedHashMap, TreeMap.

HashMap	LinkedHashMap	TreeMap
1) HashMap does not maintain Insertion order.	1) It is maintain insertion order.	1) By default sort object in ASC order based on key.
Hetrogenous key are allowed.	Hetrogenous key are allowed	Hetrogenous key are Not allowed only Homogenous key are allowed.
only one Null key are allowed	only one Null key are allowed.	one null key are also Not allowed.

Program Map

Page No.
Date

```
package Collection;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.TreeMap;
public class M11
{
    public static void main (String[] args)
    {
      HashMap h = New HashMap();
      h.put (1, "Sheela");
      h.put (2, "Sundri");
      h.put (3, "Sheela"
      h.put("leela", -5);
      h.put (Null, 'c');
      h.put (true, 5.5);
// h.put(1, "Sunny"); // old value replaced by new value
when key is same

System.out.println ("HashMap :" + h);

LinkedHashMap L = new LinkedHashMap
L.put (1, "Sheela");
L.put (2, "Sundri");
L.put (3, "Sheela");
L.put ("leela", -5);
L.put (Null, 'c');
L.put (true, 5.5);

System.out.println ("LinkedHashMap :" + L);
```

```
TreeMap t = new TreeMap();
t.put(10, "sheela");
// t.put(null, 'c'); NullPointerException
// t.put("hi", 10); classCastException
  t.put(15, Null);
  t.Put(0, 10.9);
  t.put(-25, true);
  t.put(30, "sheela");

  System.out.println("TreeMap ASC :" + t);
  System.out.println("TreeMap DESC :" + t.deseendingMap
                                          ().());
}
}
```

Output :- HashMap : {leela = -5, null = c, 1 = Sheela, 2 = sundri
3 = sheela, true = 5.5}

LinkedHashMap : {1 = sheela, 2 = sundri, 3 = sheela,
leela = -5, null = c, true = 5

TreeMap ASC : {-25 = true, 0 = 10.9, 10 = sheela,
15 = null, 30 = sheela}

TreeMap DESC : {30 = Sheela, 15 = null, 10 = Sheela,
0 = 10.9, -25 = true}

Page No.
Date

* Input inbuilt non-static method of Map interface

1) To add an Entry
   - put (key, value)
   - putAll (Map)

2) To remove objects
   - remove (key)
   - clear()

3) TO search key and value
   - containskey (key)
   - containsValue (value)

4) To access / get entry
   - entrySet()
   - keySet()
   - values()

5) To sort map
   - TreeMap() ---> ASC
   - descendingMap() ---> DESC

```
package Collection;
import java.util.Hashmap;
Public class M2
{
  public static void main (String[] args)
{
    HashMap h = new HashMap();
    h.put (1, "sheela");
    h.put ("leela", -5);
    h.put (null, 'c');
    h.put (true, 5.5);

System.out.println ("Map :" + h);
  // converting Map into Collection/set

System.out.println ("set :" + h.entrySet());
System.out.println ("keys :" + h.keySet());
System.out.println ("values :" + h.values());

    h.remove ("leela");
System.out.println ("map :" + h);
}
```

output :- Map : { leela = -5, null = c, 1 = sheela, true = 5.5 }

Set : [leela = -5, null = c, 1 = sheela, true = 5.5]

Keys : [leela, null, 1, true]

Values : [-5, c, sheela, 5.5]

Map :- { null = c, 1 = sheela, true = 5.5 }

* **Queue** :- Queue is an interface which has implementing classes such as priority Queue, LinkedList.

Queue works on FIFO/LIFO principle.

```
package Collections;
import java.util.LinkedList;
public class P7
{
    public static void main (String[] args)
    {
        LinkedList L = new LinkedList ();
            L.offer (10);
            L.offer (20);
            L.offer ("hi");
            L.add (10);
            L.add ("hi");
            L.add (Null);
            L.add ('c');

        System.out.println ("linkedlist : " + L);
    }
```

Note :- offer () – to add add () → also used to add

Page No.
Date

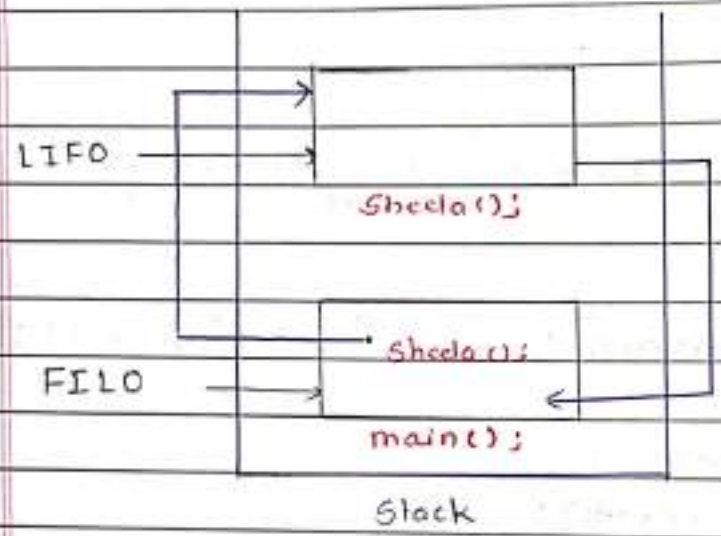
Priority Queue :- priority Queue works on priority

```
package collection;
import java.util.priorityQueue;
public class Cg
{
    public static void main (String[] args)
    {
        PriorityQueue P = new PriorityQueue ();
        P.offer (10);
        // P.offer(null);
        // P.offer("hi");
        P.offer (25);
        P.offer(-5);
        P.offer(15);
    System.out.println("Priority Queue : "+ P);
}
```

output :- PriorityQueue : [-5, 15, 10, 15]

* Stack

Stack works on FILO/LIFO (First input last output / Last input First output)




```
package collection;
import java.util.Stack;
public class C10
{
    public static void main (String[] args)
    {
        Stack s = new Stack();
        s.push(10);
        s.push(5);
        s.push("hi");
        s.push('c');
        s.push(-5);

        System.out.println("Stack:" + s);
        System.out.println("Peek:" + s.peek());
        System.out.println("Stack:" + s);
        System.out.println("Pop:" + s.pop());
        System.out.println("Stack:" + s);
```

Page No.
Date

output :- Stack :- [10, 5, hi, c, -5]
Peek :- -5
Stack :- [10, 5, hi, c, -5]
Pop :- -5
Stack :- [10, 5, hi, c]

* Interview Questions

1) How to remove duplicate object from list.

2) How to convert list into Set

3) How to Convert Set into list

4) How to Convert Map into list

5) How to convert Collection into Array or Array list into Array.

Answer :-

```
package Collection;
import java.util.Arraylist;
import java.util.HashSet;
public class C;
 {
   public static void main (String [] args)
   {
     ArrayList a = new ArrayList ();
      a.add (10);
      a.add ("hi");
      a.add ('10');
      a.add (Null);
      a.add ("hi");
       a.add (null);
System.out.println ("list :" + a);

// Converting List into Set
HashSet h = new HashSet (a);
System.out.println ("set :" +h);
```

// Converting Set into list

```
ArrayList l = new ArrayList (h);
System.out.println ("List :" + l);

// Converting Map into List
// 1. entrySet () --> Map into Set
// 2. Set into list
// convert ArrayList into Array
    Object [] i = a.toArray ();
}
}
```

Output :- List : [10, hi, 10, null, hi, Null]
List : [Null, hi, 10]
List : [Null, hi, 10]

Generic Collection (Homogeneous)

Syntax :- ArrayList <non-primitive-datatype>
a = new Arraylist <non-primitive-datatype> ();

Example :- ArrayList <Integer> a = new ArrayList <Integer> ();

OR

Arraylist <Integer> a = new ArrayList ();