CORE JAVA

11/08/22

Programming Language :-

Def^{n} :- Programming language is a set of instruct-ions to perform a particular task, by the machine.

Programming language is classified into
1) High level programming language.
2) Low level programming language
3) Middle level programming language

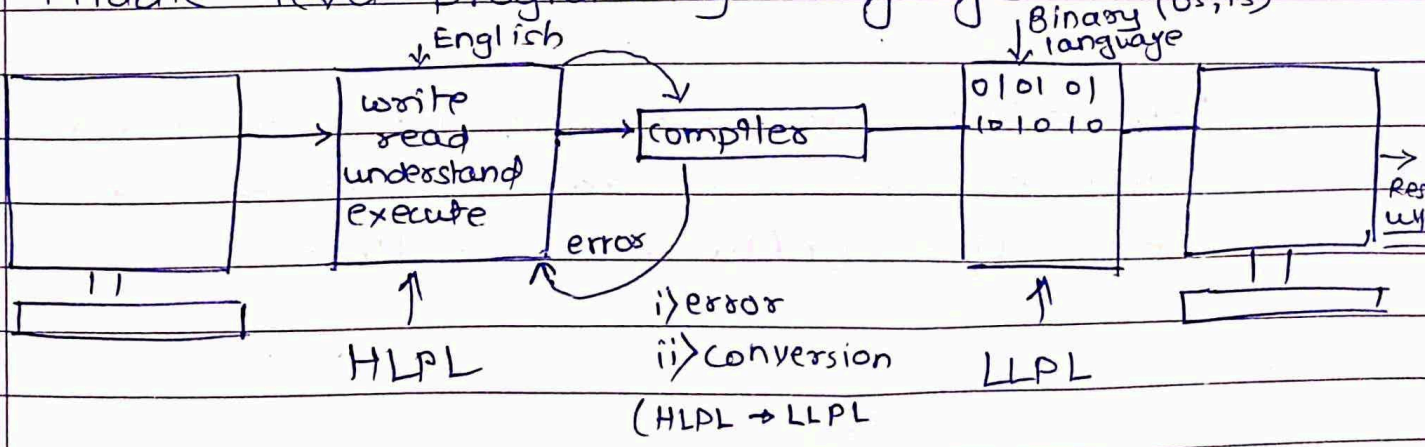
1) High level programming language :-

The language which is writable, readable, understandable, executable by the programmer such languages are high level programming langu-age. (HLPL).
eg. C, C++, Java, Python etc.

2) Low level programming language :-

The language which is understandable & executable by the machines such languages are called as low level programming language (LLPL).
eg. binary language (0,1)

3) Middle level programming language :-



What is Java?
What are the characterestics of Java?
What are the features of Java?

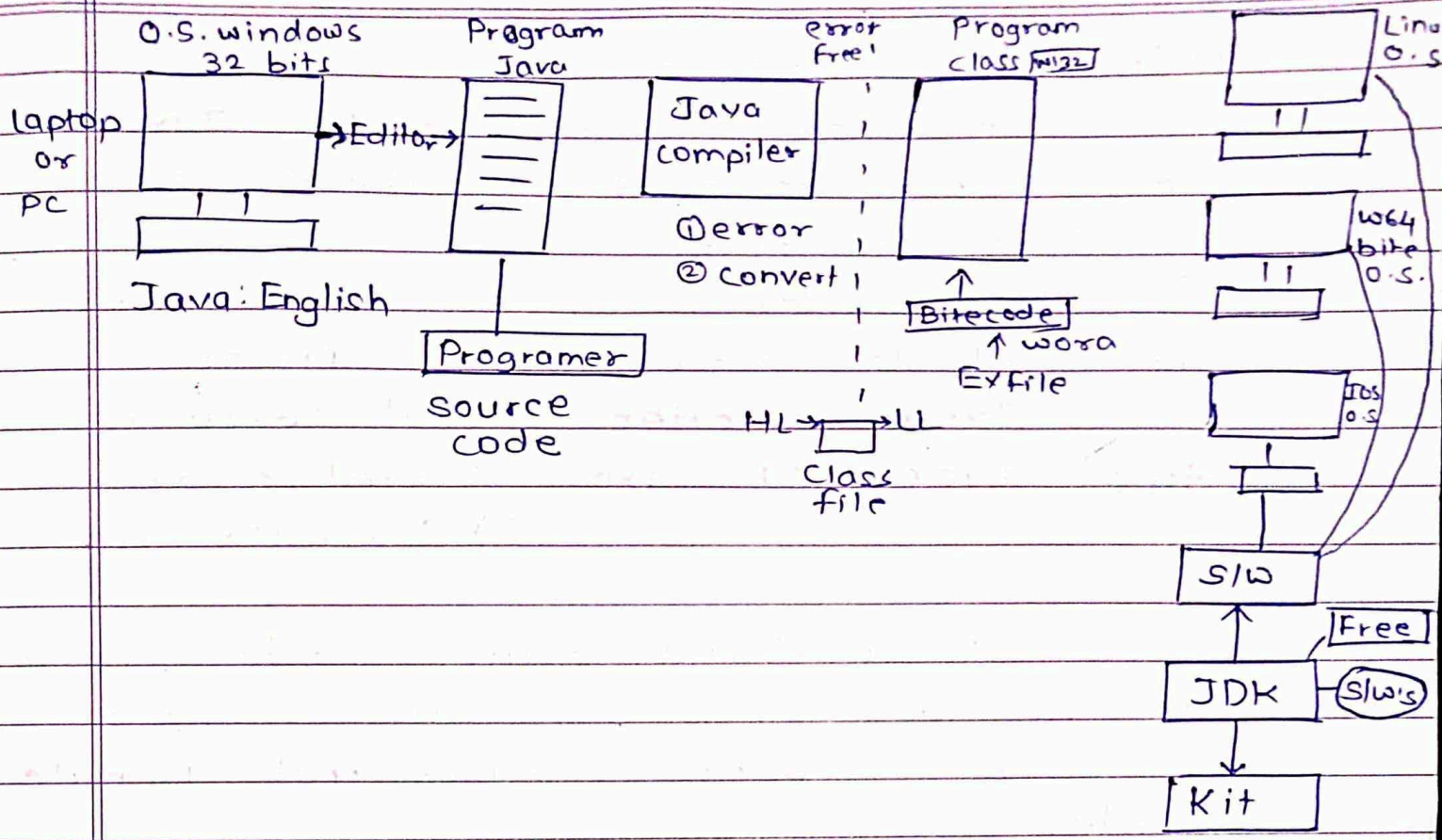
1) Java is high level programming language
2) Java is easy to understand.
3) Java provides high security
4) Java object oriented programming language.
5) Java is platform independant
6) Java consists of inbuilt library.

12/08/22.

* Why Java is called as platform independent
→ A source code which is created using java it is not directly converted into machine level programming language (low level programming language).
- Instead of it is converted into an intermediate language for bite code.
we can execute this bytecode in any platform (O.S.) which has JDK software.
- Hence Java is called platform independent but it is totally dependent on JDK softwa-re.

Note :- We can not execute a Java source code in any platform which has no JDK software installed.

JDK - JAVA DEVELOPMENT KIT



13/08/22

* Steps to execute a Java program.
  i) Create a source code.
  ii) Compile source code
  iii) Execute bite code.

i) Create a source code :-
- We can create a java source code ~~to be~~ with the help of editors
- An editor is a software which provide a platform to develope a Java source code.
- eg. Notepad, notepad ++, Editplus, There are some advanced editors like visual studio, Eclipse, etc.

ii) Compile Source code :-

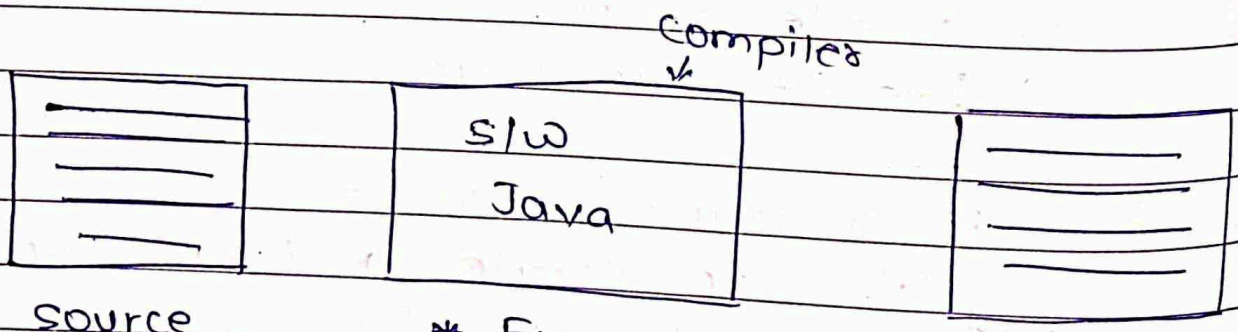
In order to compile the Java source code we need to take the help of Java compilier.

In order to access Java complier we need to install JDK software.

Java compiler it will check for two types of error i) Syntax error
ii) Semantic error

If a Java source code consist of any one error in Java compiler does not convert source code to byte code.

If the Java source code consist of 0 error (No syntax and semantic error) then the compiler will convert source code to byte code.



iii) Execute bite code :-

We can execute Java bite code with the help of JVM (Java virtual machine).

16/08/22

* Structure of Java :-

```
class
{

    public static void main (string [] args)
    {
      main_method
    }

}
```

```
class program_name / class_name
{
    public static void main (string [] args)
    {
      main_method
    }
}
```

class block

* Steps to execute a simple Java programe :-

1) Create a source code and save the program with same name as class-name.
2) In order to compile and run the Java program we need to use a common software CMD [command prompt]
3) Command to compile the Java source code (Java C - class-name.Java).
4) Command to run Java source code (Java class_name).

19/08/22

~~Key words &~~

TOKENS :-

- Tokens are the smallest ~~un~~element of java programming.
- Tokens are classified into
  - i) Key words
  - ii) Identifiers
  - iii) Literals
  - iv) Seperators
  - v) Operators

i) Key words :-

- Key words are predefined and reserved words of Java.
- We can identify the keywords of java in two ways
  - a) They are highlighted in blue colour and
  - b) They are written in terms of lower case.

  eg. class, public, static, void, etc.

Keywords in JAVA :-

1) Abstract
2) Assert
3) Boolean
4) Break
5) byte
6) Case
7) catch
8) char
9) class
10) continue
11) default
12) do
13) double
14) else
15) enum
16) extend
17) final
18) finally
19) float
20) for
21) if
22) implements
23) import
24) instanceof
25) int
26) interface
27) long
28) native
29) new
30) null
31) package
32) private
33) protected
34) public
35) return
36) short
37) static
38) strictfp
39) super
40) switch
41) synchronized
42) this
43) throw
44) throws
45) transient
46) try
47) void
48) volatile
49) while
50) Const
51) Go to
52) false
53) true.

ii) Identifiers :-

- Identifier it an name given by programmer to elements of java.
- Elements of java are
  - i) Class - name
  - ii) Variable - name
  - iii) Method - name
  - iv) Package - name
  - v) Interface - name

08/22

* What are the rules of identifier?
* What are the class naming conventions?

1) The first letter of class name can be written in terms of other upper case or lower case. But this rule is not mandatory it totally depend on company standards.

eg.

Program.java , program.java

2) We cannot start a class name with a numerical value.

eg.

1program.java : CTE

3) In order to use a numerical value we need to follow alphanumeric rule.

ex.

Program1.java , P1.java

3) We cannot use a keyword as class name.

ex.

static.java : CTE

4) We cannot define a space between the class name

ex. Program one.java : CTE

In order to fill the space we can use 2 special characters in our class name.

ex. Program_one.java , Program$one.java

5) If a program name consist of more than one word (multiple words) the first letter of second word has to written in terms of upper case.

— iii) Literals :- Literals are different types of values we use in java programming.

- Values
  - Numbers
    - Integer → 10, 20, 30
    - float → 10.11, 20.11
  - Character → 'A', 'b', '$'
  - words/string → "Dog", "Cat"
  - Boolean → True, false } key words.

iv) Seperators :-

ex :-

';', ',', ':', '{ }'

* Printing statements of Java :-

There are two printing statements in java :-

i) System.out.println()

ii) System.out.print()

i) System.out.println() :—

It is used to print the value & move the cursor to next line.

ex.

```
Class ProgrammOne
{
    public static void main (string[] args)
    {
        System.out.println (1);
        System.out.println (2);
    }
}
```

	O/P
Println (1)	_1_
Println (2)	_2_
	_

- for println statement passing a value is not mandatory.

23/08/22

ii) System.out.print() :-

It is used to print the value and move the cursor on the same line.

for system.out.print() passing a value is mandatory.

ex.

```
Class ProgrammOne
{
    public static void main(string[] args)
    {
        System.out.print (1);
```

```
        system.out.print(2);
        }

    }
```

* Printing all the values of java with the help of printing statement.

```
S.O.pln(100);
S.O.pln(10.25);
s.o.pln('A');
s.o.pln('$');
s.o.pln("hello");
s.o.pln(true);
s.o.pbn(false);
```

24/08/22

* Datatypes :-
  - Datatypes are the type of values we use in java programming.
  - Datatypes depends on literals.
  - Datatypes is classified into two types
    i) Premitive Datatypes
    ii) Non premitive Datatypes.

i) Premitive Datatypes :-
  - Premitive datatype are the type of premitive values we use in java programming.

1 bit = 8 bits = 1 byte

- Premitive datatypes are classified into

PTV	PTDT	Size
i) Integer	byte	1 byte
	short	2 byte
	int	4 byte
	long	8 byte
ii) Float	float	4 byte
	double	8 byte
iii) Char	char	2 byte
iv) Boolean	boolean	1 bit

ii) Non Premitive Datatype :-

- Non premitive datatype are the type of non premitive values we use in java programming
- ex. String.

★ Variable :-

Variable its a name given to a block of memory to store a value

Syntax :-

```
data_type variable_name = value;
```

ex.

```
int wallet = 100;
```

Variables are classified into two type
i) Global Variables.
ii) Local Variables.

ex.

```
class
{
  // Global Variable : class block

  public static void main (string [] args)
  {
    // Local Variables : method block
  }

}
```

25/08/22

* Local variable :-
  - Local variables are declared inside method block.
  - Local variables does not consists of default values.
  - In order to access local variable initializing a value is mandatory.
  - We can declare more than one local variables inside the method block. But variable name has to be different.

Note :- Inside the method block we cannot declare more than one local variable with same variable name.

ex. 1)

```
class P1
{
    public static void main (string[] args)
    {
        int wallet = 11 ;
        S.O. pln (wallet);
    }

}
```

O/P = 11

ex. 2)

```
class P2
{
    p. s v m (string[] args)
    {
        int// variable name has tobe different
        string bag = "mango" ;
        string bag1 = "apple" ;

        S.O. pln (bag);
    }

}
```

O/P = mango

Note :- In order to access a variable declaration of datatype and variable-name has to be done first.

ex. 1)

```
class P1
{
    P s v (String[] args)
    {
        int wallet ;
        wallet = 1500 ;

        S.o.pln (wallet) ;      // 1500
    }

}
```

ex. 2)

```
class P1
{
    P.s v (string [] args)
    {
        wallet =1500 ;  // CTE
        int wallet ;

        S.O.pln (wallet) ;
    }

}
```

27/08/22

* Plus operator :-

Plus operators are we can use two ways.

i) Addition
ii) Concatination

int + int = int
int + char = int
char + int = int
char + char = int
Ascii + Ascii = value
int + float = float
float + float = float
float + char = float

Boolean + int, float, char, Boolean } CTE

string + boolean, int, char, float, string } string

```
class P2
{

    Public static void main (string[] args
    {

        int a = 12 ;
        int b = 13 ;
        char c = 'd' ;

        S.O.pln (a+b+c) ; // O/P = 125
    }

}
```

* Operators :-

Operators are used to perform a ~~task~~ particular task.

Operators are classified into three types

i) Unary operator
ii) Binary operator
iii) Ternary operator.

i) Unary operator :-

a) Increment operator.
b) Decrement operator.
c) Cast operator.

ii) Binary operators :-
   a) Arithmatic operator.
   b) Relational operator
   c) logical operator.

iii) Ternary operator :-

i) Unary Operators :-
The operator which take only one value at a time such operators are called as unary operator.
ex. Increment operator
Decrement operator
Cast operator.

ii) Binary Operations :-
The operator which take two values at a time such operator are called binary operator.
ex. Arithmatic operator
Relational operator
logical operator

iii) Ternary operator
The operator which take value at a time such operator are called as ternary operator.
ex. Conditional operator.

i) Increment Operator :-

Increment operator is used to increment the value which is present inside the variable. (Increment operator it is denoted ~~it is~~ denoted by '++').

Increment operator is classified into two types :-

a) Post increment operator.
b) Pre increment operator.

a) Post increment operator :-

Syntax :-

```
variable_name++;
```

ex.

```
a++;
```

The increment operator which is suffixed to its variable_name is called as increment operator.

* Steps to perform post increment operator :-

i) Use the value which is present inside the variable (Either to print or store).

ii) Increment the value which is present inside the variable by '1' and update

ex. 1

```
class P1
{
    public static void main (String[] args
    {
        int a = 1;

        System.out.println (a);      // 1
        System.out.println (a++);    // 1
        System.out.println (a);      // 2
    }
}
```

ex. 2

```
class P2
{
    public static void main (String[] args)
    {
        int a = 1;
        int b = a++;
        int c = b++;

        S.O.pln (a);      // 2
        S.O.pln (b);      // 2
        S.O.pln (c++);    // 1
        S.O.pln (c);      // 2
    }
}
```

```
class P1
{
    public static void main (string[] args)
{
int a = 1234567
int b = 23456778
int c = 123
    s.o.pln (b++ + a++ + a + b++ + a + a++ + b++ + a);
             2   + 1   + 2 + 3 + 2 + 2 + 4 + 3
            = 19

    s.o.pln ( c++ + a++ + b++ + a++ + a + b + a++);
              1 + 3 + 5 + 4 + 5 + 6 + 5
             = 29

    s.o.pln (b++ + a++ + c + c++ + b++ + a)
              6 + 6 + 2 + 2 + 7 + 7
             = 30

    }

}
```

ii) Pre increment operator :-

Syntax :-

```
++variable-name;
```

ex.

```
++a;
```

The increment operator which is prefixed to its variable name is called as pre increment operator.

Steps to perform pre increment operators:-

i) Increment the value ~~present~~ which is present inside the variable by '1' & update first.

ii) Use the updated value which is present inside the variable

```
Class P4
{
    public static void main (String[] args)
    {
        int a = 1;

        s.o.pln (a);      // 1
        s.o.pln (++a);    // 2
        s.o.pln (a);      // 2
    }
}
```

1 + 4 + 2 + 4 + 3 + 3 + 3 + 5 + 4 = 29

```
Class P3
{
    public static void main (String[] args)
    {
        a = 1
        b = 3
        res = ++a + ++++b + ++a + ++ b ;
                  2      4      3      5     //14
        s.o.pln (res) ;     //14

    }

}
```

```
Class P4
{
    Public static void main (String[] args)
    {
    int  a = 1 2 3 4
    int  b = 2 3 4 5
    int  c = 3 4 5

        res = a++ + ++ +c + b++ + ++b + c++ +
                                ++a + a++ + ++b + c++;

        = 1 + 4 + 2 + 4 + 4 + 3 + 3 + 5 + 5
        = 31
    }

}
```

* Decrement operator :-

Decrement operator is used to decrement the value which is present inside the variable. (Decrement operator is denoted by '--').

Decrement operators are classified into two types :-

a) Post Decrement operator
b) Pre Decrement operator

a) Post decrement operator :-

The decrement operator which is suffixed to its variable-name is called as post decrement operator

Syntax :-

variable-name --;

ex.

a--;

* Steps to perform post decrement operator :-

i) Use the value which is present inside the variable (Either to print or to store.).

ii) Decrement the value ~~by~~ which is present inside the variable by '1' and update.

ex. 1)

```
class P1
{
    public static void main (string[] args
    {
        int a = 10

        system.out.println (a);     //10
        System.out.println (a--);   //10
        System.out.println (a);     //9
    }
}
```

ex. 2)

```
class P2
{
    public static void main (string[] args)
    {
        int a = 10;
        int b = a--;
        int c = b--;

        s.o.pln (a); // 9
        s.o.pln (b); // 9
        s.o.pln (c--); // 10
        s.o.pln (c); // 9
    }
}
```

ex. 3)

```
class P3
{
    public static void main
    {
        int a = 10 9 8 7 6 5 4
        int b = 20 19 18 17 16 15 14
        int c = 10 9 8

s.o.pln (b-- + a-- + a + b-- + a + a-- + b-- + a);

// (20 + 10 + 9 + 19 + 9 + 9 + 18 + 8) = 102

s.o.pln(c-- + a-- + b-- + a-- + a + b + a--);

// (10 + 8 + 17 + 7 + 6 + 16 + 6) = 70

s.o.pln (b-- + a-- + c + c-- + b-- + a);

// (16 + 5 + 9 + 9 + 15 + 4) = 58

    }
}
```

* b) Pre Decrement operator :-

Decrement operator which is prefixed to ~~with~~ its variable-name is called as pre Decrement operator.

Syntax :-

```
-- variable_name;
```

2.

ex.

```
--a;
```

Steps to perform pre decrement operator

i) Decrement the value present inside the variable by '1' and update.

ii) Use the updated value which is currently present inside the variable.

ex. 1)

```
class P1
{
    public static void main (String [] args)
    {
        int a = 10;

        S.O.pln (a);      // 10
        S.O.pln (--a);    // 9
        S.O.pln (a);      // 9

    }

}
```

01/09/22

* Type Casting :-

It is a process of converting one datatype into another datatype is called as type casting.

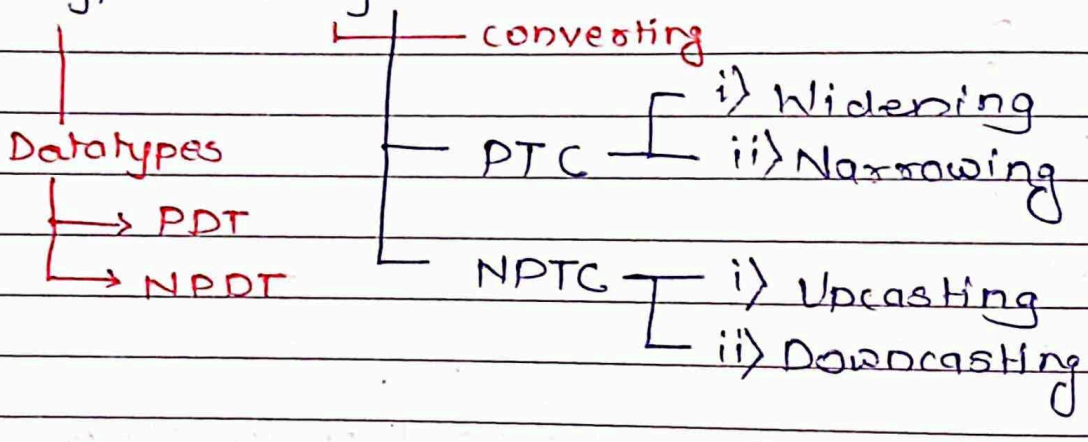
Type casting depends on data types.

Type casting is classified into two types.

i) Premitive type casting
ii) Non premitive type casting

D → D

Type casting :-



i) Premitive Type Casting :-

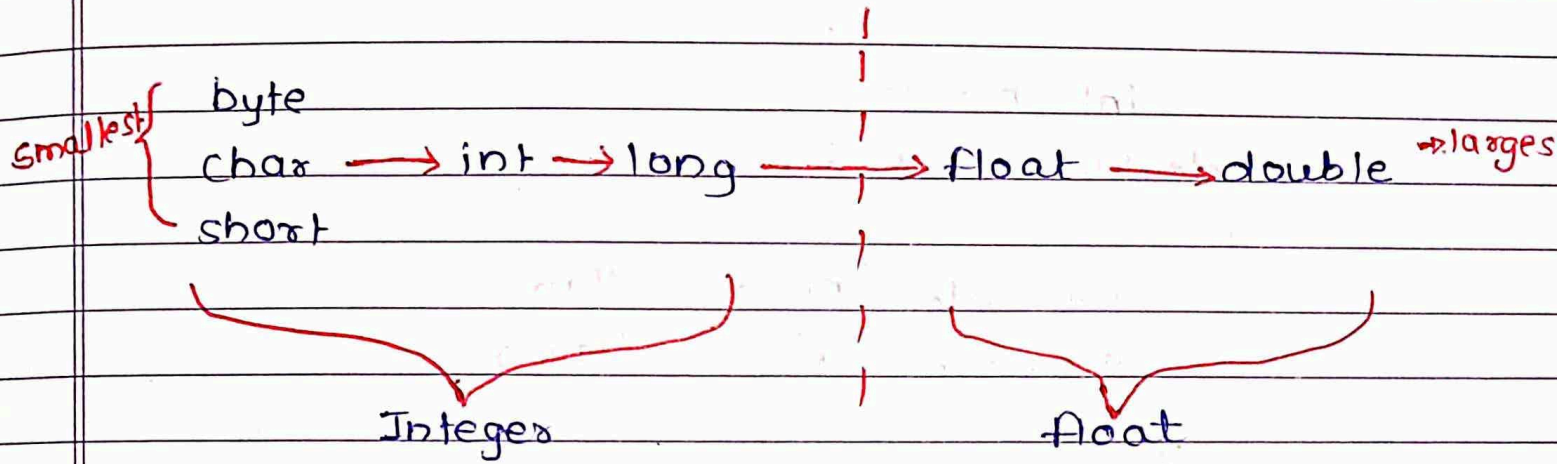
Premitive type casting is a process of converting one premitive datatype into an another premitive datatype. is called Premitive Type Casting.

Premitive type casting is classified into two types.

a) Widening
b) Narrowing

a) Widening :-

It's a process of converting smallest premitive datatype into largest premitive datatype is called as Widening.



ex. 1)

```
class P1
{
    public static void main (string [ ] args)
    {
    char n = 'a' ;
    int m = n ;

        s.o.pln (n) ;     // a
        s.o.pln (m) ;     // 97
    }
}
```

```
class P2
{
    public static void main (String[]args)
    {

        int n = 10;
    double a = n;

        s.o.pln (n);    //10
        s.o.pln (a);    //10.0

    }

}
```

Widening process is done by the compiler automatically hence widening is also called as Auto Widening.

Advantages :-

i) Addition of data. (Decimal values).

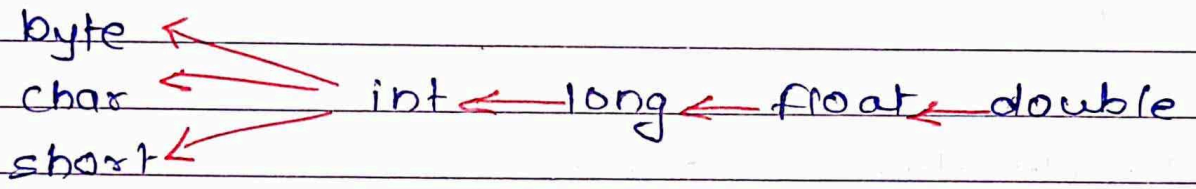
* Narrowing :-

Narrowing is a process of converting a largest premitive datatype into smallest premitive datatype is called as narrowing.

We cannot perform narrowing process automatically.

In order to perform narrowing we need to take the help of cast operator.

Cast operator its an unary operator it takes one value at a time.



ex. 1)

```
class P4
{

    public static void main(String [] args)
    {

        int a = 100;
        char b = (char) a ;
       // char b = a ---- CTE
        s.o.pln (a);          // 100
        s.o.pln (b);          // d
    }

}
```

```
class P4
{
    public static void main (String[]xy)
    {
  double i = 10.1234;
    int j = (int) i;

    s.o.pln (i);    //10.1234
    s.o.pln (j);    //10
    }

}
```

Disadvantage of Narrowing :-

1) There is a loss of data in Narrowing.

* Arithmatic Operator :-

It is an binary operator which take two values at a time.

```
class P6
{
    public static void main (String [] args)
    {
    int n = 10;
    int m = 2;

    s.o.pln (n+m);    // 12
    s.o.pln (n-m);    // 8
    s.o.pln (n*m);    // 20
    s.o.pln (n/m);    // 5
```

```
        s.o.pln(n%m); //0
    }

}
```

Arithmatic operator writtens output either in integer or float format.

* Relational operator :-

It is an operator which takes two values at same time.

The output type of relational operator is boolean (condition (True / false)

```
class P5
{
    public static void main (string[] args)
    {
        a = 10
        b = 10

        s.o.pln (n>m);
        s.o.pln (n<m);
        s.o.pln (n>=m);
        s.o.pln (n<=m);
        s.o.pln (n==m);
        s.o.pln (n!=m);
    }

}
```

03/09/22

Logical operator :-

Logical operator it takes 2 values at time.

Logical operator is classified into 3 types.

i) AND operator (&&)
ii) OR operator (||)
iii) NOT operator (!)

AND operator :-

Truth Table :-

Statement 1	st 2	O/P
True	True	True
True	false	false
false	True	false
false	false	false

OR operator :-

Truth table :-

st 1	st 2	O/P
True	True	True
True	false	True
false	True	True
false	false	false

iii) NOT Operator :-

Truth table

I/P	O/P
True	False
False	True

```
class P3
{
    public static void main (string [] args)
    {
      // AND operator
      s.o.pln (True && True);  // True
      s.o.pln (True && False); // False
      s.o.pln (False && True);
      s.o.pln (False && False;

      // OR operator
      s.o.pln (True || True);
      s.o.pln (True || False);
      s.o.pln (False || True);
      s.o.pln (false || False);

      // NOT operator
       s.o.pln (! False);
       s.o.pln (! True);
```

Turnary operator :-

i) Conditional operator :-

Conditional operator it is an turnary operator it takes 3 values at time.

Syntax :-

condition ? Task 1 : Task 2

ex. ⇒ If the condition is true the compiler transfers the control to task 1.

If the condition is false the compiler control is transfer the control to task 2.

ex. 1)

```
class P4
{
    public static void main (string[] args)
    {
    int n = 12
    int m = 13

     s.o.pln (n>m ? n : m); // 13
    }

}
```

```
class P5
    {
        public static void main (String [] args)
        {
    int  n = 12
    int  m = 15
    int a = n > m ? n : m ;  //store the result

        s.o.pln (a);   // 15
    }

    }
```

1) Maximum of 2 Numbers?

→

```
class P1
{

    psvm (string [] args)
    {
        int  a = 12;
        int  b = 05;

    s.o.pln (a < b ? b : a);

    }

}
```

05/09/22

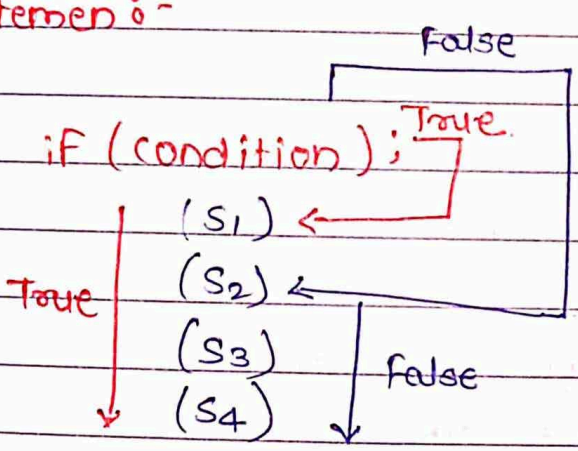
* Decision Making Statements :-

Decision making statement it is used to check a condition before performing a particular task.

Decision making statement is classified into five types :-

1) If statemen
2) If Else statement
3) Else if Ladder
4) Nested if
5) switch

1) If statemen :-




```
class P1
{
    psvm (string [] args)
    {
        if (false)
        {s.o.pln ("vadapav");
        }
        s.o.pln ("Go back to home");
    }
}
```

op false :- Go back to home

```
class P2
{
    psvm (string [] args)
    {
        if (True) (False)
        {
            s.o.plo ("sugar packet");
            s.o.plo ("ice cream.");
        }

        s.o.plo ("Go back to home");
    }

}
```

o/p :-

False ⇒ Go back to home

True ⇒ Sugar packet
ice creame
Go back to home

06/09/22

If else statement :-

Else block use to perform task for false condition.

Else block is not mandatory.

```
if (condition)
{
    statements
}
Else
{
    statements
}
```

```
class P3
{
    psvm (String[] args)
    {
        String shop = "closed";

        if (shop == open)
        {
            s.o.pln ("salt");
            s.o.pln ("chilly powder");
        }
        else
        {
            s.o.pln ("Ask neighbour house");
        }
        s.o.pln ("go back to home");
    }

}
```

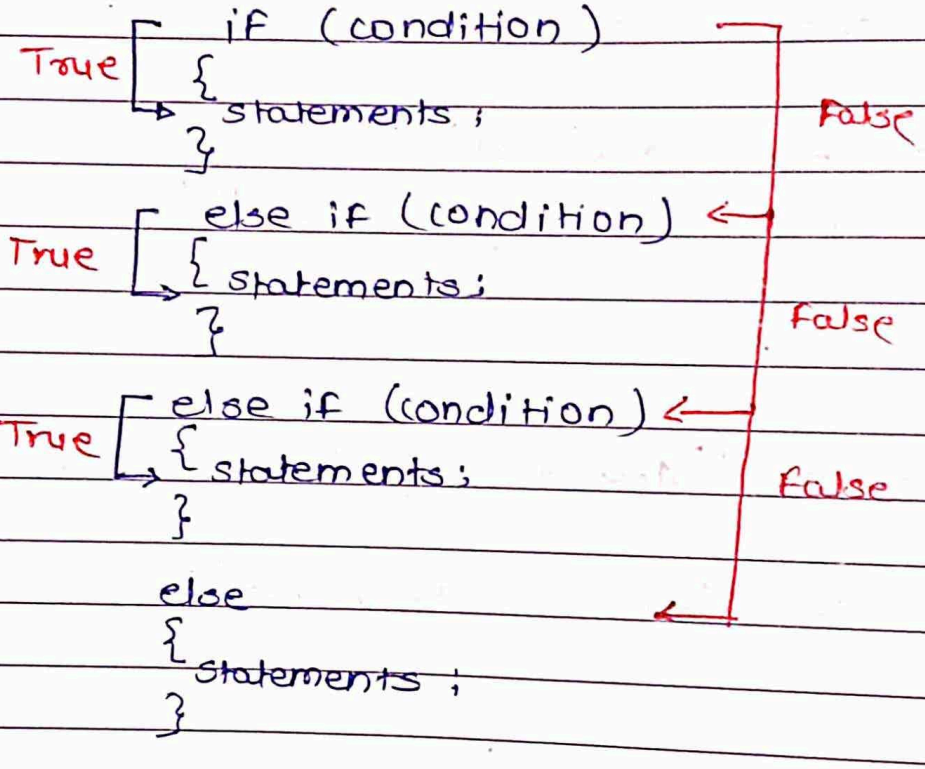
O/P = if (condn is true)
salt
chilly powder
go back to home

O/P = (condn is false)
Ask neighbour house
go back to home.

Else if Ladder :-

Verifying more than one condition is called as Else if Ladder.

Syntax



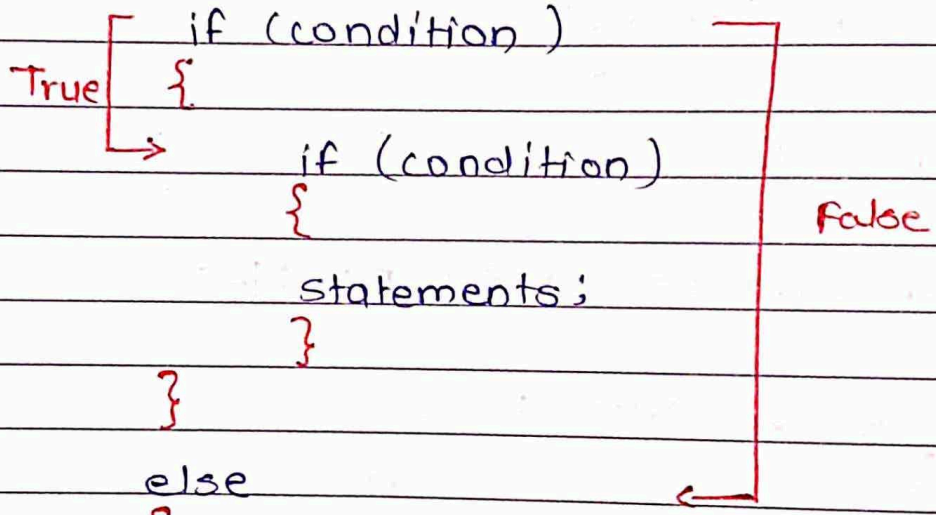
For else if ladder else block is not mandatory.

* Write a program to check a given no. is
  i) If the no. is divisible by 7 and if it is not an even no. print "Hi".
  ii) If the no. is divisible by 3 or 7 print "Bye".
  iii) If the no. is even no. & not divisible by 7 print "Good bye";

Nested if ~~statements~~ condition :-

An if condition inside another if condition is called as nested if condition.

Syntax :-



★ Write a program to check the given no is positive or negative.

★ Write a program to check the given no is even or odd.

★ Write a program to check whether the given no. is divisible by 3 and 7.

★ Write a program to check the given character is vowel or not using else if ladder.

5) Switch :-

Switch it is an decision making statement, inside switch we cannot pass cond^n

Inside switch we can pass an expression

The output of an expression it is always a value

Syntax :-

```
                          ← switch value
Switch (expression)
{
case value : Task ;
case value : Task ;
case value : Task ;
case value : Task ;
default    : Task ;
}
```

Inside switch we can also pass differe -nt types of values. (Literals)

In switch the comparison of condition happens between switch value and case value.

ex.

```
class P5
{
    ps v m (string[] args)
    {
        int var = 3;
```

```
s.o.pln ("main begin");

Switch (var %.2)
{
case 0 : {
          s.o.pln ("even");
        }
case 1 : {
          s.o.pln ("odd");
         }

}
s.o.pln ("main end");
    }
  }
}
```

* break Keywords :-

break is a keyword is used to stop execution. (particular task).

ex.

```
class P5
{
    psvm (string [] args)
    {
        int var = 2 ;
        s.o.pln ("main begin");
```

```
Switch (var %. 2)
{
 case 0 : {
          s.o.pln ("even");
          break;
           }
 case 1 : {
          s.o.pln ("odd");
           }
 }
 s.o.pln ("main end");

 }

}
```

1) WAP to check the given character is vowel or not using switch.

* Looping Statements :-

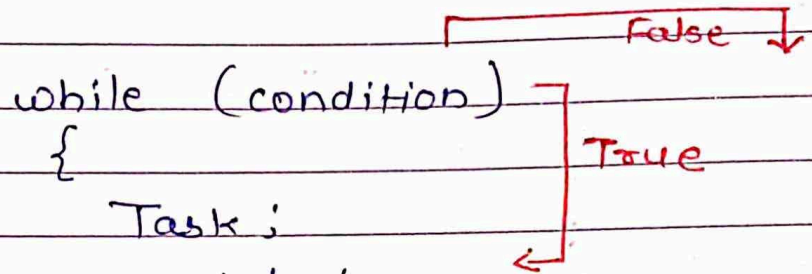
It is used to verify condition more than once before performing a particular task.

looping statements is classified into 3 types :-

a) while loop
b) do while loop
c) for ~~whi~~ loop

* While Loop :-

While loop is looping statement.



ex.

```
class
{
    psvm (String[] args)
    {
        int a = 0;
        while (a<3)
        {
        s.o.pln ("$");
        a++;
        }

    }
}
```

* do while :-

```
do
{
Task;
update;
```

```
    }
    while (condition);
```

ex:-

```
class P2
{
    psvm (String [].args)
    {
        char ch = 'a';

        do
        {
            s.o.pln (ch);
            ch++;
        }
        while (ch<='e');

    }

}
```

Note :- For false condition do while loop gets executed only once.

ex. :-

```
class P3
{
    psvm (String [] args)
    {
        char ch='a';
```

```
        do
        {
        s.o.pln (ch);
        ch++;
        }
        while (ch>'e');
    }
}
```

* for loop statements :-

```
for (~~initializing~~ initialization; condn;
                                update)
{
   Task;
}
```

True

ex. :-

```
class P3
{
    psvm (String[]args)
    {
        for (int n=0; n<0; n++)
        {
        s.o.pln (n)
        }
    }
}
```

* Control transfer statements :-

Control transfer statement is used to transfer the control out of the block. Control transfer statement is classified into two types :-

1) break
2) continue.

1) break :-

break is an control transfer statement in java.

break keyword is used to stop the execution ~~of part~~ and transfer the control to out of the block. (to the end of ~~the~~ the particular block).

```
class P4
{
    ps v m (string [] args)
    {

        int a=0;
        s.o.pln ("main begin");
        while (a<5)
        {
         s.o.pln (a);
         if (a==2)
          {
          s.o.pln ("if block");
          a++;
          break;
          }
        s.o.pln (a);
        a++;
         }
        s.o.pln ("main end");

    }
}
```

O/P

```
main beg
0
0
1
1
2
if block
main end
```

2) Continue :-

Continue is an control transfer statement & is keyword in java

Continue (keyword) is used to transfer control to the beginning of loop condition.

Continue keyword is not meant to stop the execution of the program.

```
Class P4
{
    psvm (String[] args)
    {
      int a=0;

      s.o.pln ("main begin");

      while (a<5)
      {
        s.o.pln (a);
        if (a==2)
        {
        s.o.pln ("if block");
        a++;
        continue;
        }
      s.o.pln (a);
      a++;
      }
      s.o.pln ("main end");

    }

}
```

O/P

```
main beg
0
0
1
1
2
if block
3
3
4
4
main end
```

NOTE :- break :-

1) break keyword always has to be defined at the end the block.

2) We cannot use break keyword in between the block. (We get CTE } called as unreachable statement).

ex.

```
if (a==2)
{
  S.o.pln ("if block");
  break ;  // CTE = unreachable statement
  a++;
}
```

continue :-

1) Continue key word always has to be defined at the end of the block.

2) We cannot use continue keyword in bet^n the block. (We get CTE called as unreachable statement).

ex.

```
if (a==2)
{
S.o.pln ("if block")
continue ; // CTE = unreachable statement
a++ ;
}
```

3) We cannot define both break & continue keyword together inside single block (We get CTE called as unreachable statement).

* Methods :-

Methods is a set of instructions to perform a particular task.

In order to create a method we need to take the help of method heade

Syntax :- (Method header)

modifier return-type method-name(fo
-al argumen

We can identify a method with the help of method ~~sig~~ signature

Method signature consists of two elements :- 1) Method-name
2) Formal arguments / parameter

Syntax of method signature :-

Method_name (formal arguments)

ex.

```
class P5
{
    // Method
    public static void demo()
    {
        s.o.pln("demo method");
    }
    public static void main (string[] arg
```

```
{
    s.o.pln("main begin");
    demo()
    s.o.pln("main end");
}
}
```

* Method naming convention :- (Rules of Method name)

1) If a method name consists of a single word the entire word has to be written in terms lower case.

ex.

demo(), main(), test(), add(), etc.

2) If a method name consists of more than 1 word the first letter of second word ~~shoud~~ has to be written in terms of upper case.

ex.

demoOne(), testOne(), addOne(), etc.

We can create two types of methods

1) No argument method

ex.

demoOne(), test(), etc.

2) Parameterised method

datatype ↓ variable-name ↑

ex. demo(int a, int b). test(int a, int b)

* Method call statement :-

It is a process of ~~call~~ transfering the control from calling method to called method is called method call statement.

Method call statement is classified into two types of methods.

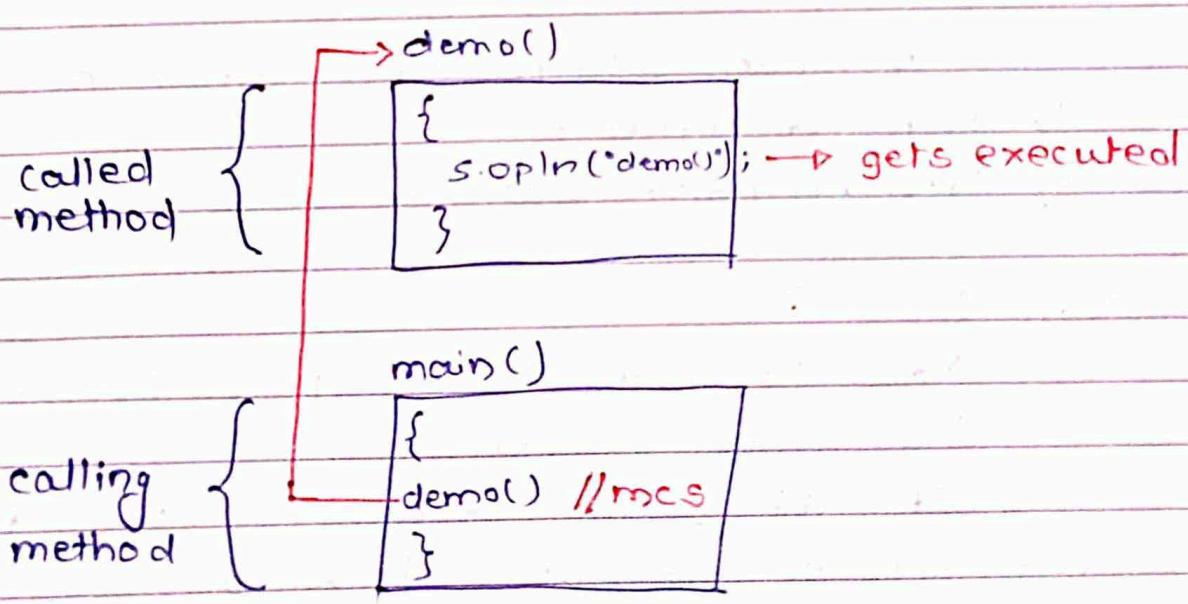
i) Calling method
ii) Called method.

i) Calling Method :

The method which tries to call another method is called as calling method.

ii) Called Method :-

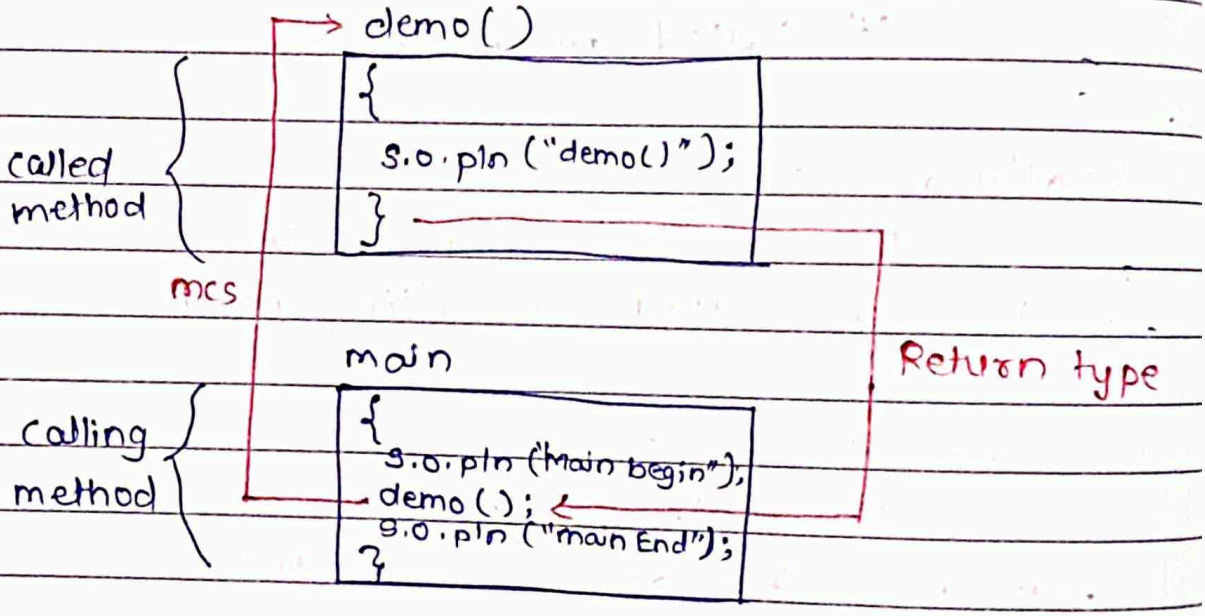
The method which comes under execution is called as called method.



Return type :-

It is a process of transfering control from called method to calling method is called as return type method.

To perform return type method call statement is mandatory.



We can return two types of data

i) Premitive datatype

ii) Non premitive datatype

Void :-

Void is just keyword in java.

Void means nothing.

If the return type is void it does not return any value back to programmer.

But it will just transfer control from called method to calling method.

ex.

```
class P2
{
    p s int recieve (int a)
    {
    s.o.pln (a);
    int res = a*2;
    s.o.pln (res);
    return 500;
    }
     p s v m
    {
    s.o.pln ("start");
    int res = sagar (2000); //500 (return value)
    s.o.pln ("continue");
    s.o.pln (res);
    }
}
```

```
→ Sagar (int a)
  {
   sopln(a)
   int res = a*2
   s.o.pln (res);
   return 500;
  }
  main
  {
   s.o.pln ("start")
   int res = sagar (2000);
   s.o.pln ("continue");
   s.o.pln (res);
```

500 (Reduce value)

```
class A5
{
    static int a = 1;
    static int b = 2;
    p s v m (String[] args)
    {
    s.o.pln (a)
    s.o.pln (b)
    }
}
```

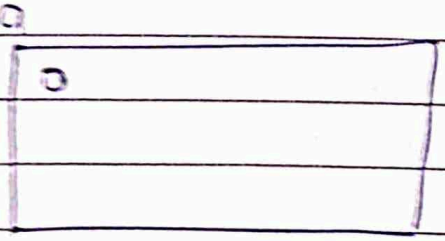
Loading Process of class :-

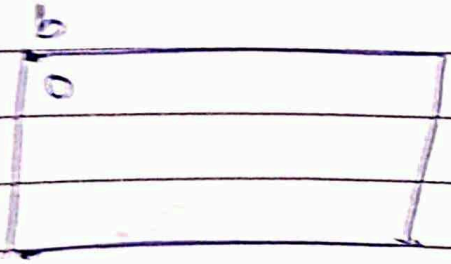
1) Search
2) Memory
3) Execution

} member

Load { top ↓ bottom — 1st search static v 1 more ↓ NOT present

comp → 1st static variable - Loading initializing after





19/9/22

NOTE:- We can recieve the return ~~to~~ value in two ways :-

1) Storing the return value & accessing the value present in the variable.

2) Printing the return value directly (we need to a write method call statement inside printing statement.)

ex.

```
class P3
{
    public static string vowel (char var)
    {
    string res = "constant";
    if (var == 'a' || var == 'e' || var == 'i'
        || var = 'o' || var == 'u')
    {
      res = "vowel";
    }
    p s v m (string [] args)
    {
    string res = vowel ('k') //consonant :string
    s.o.pln (res);   // printing
    s.o.pln (vowel ('a')); //vowel :printing
    }

}
```

Recurssive Method :- It is a process of a method calling itself is called as recurssive method.

ex.

```
class P4
{
    p s v demo ()
    {
     S.O.pln ("demo()");
     demo ();
    }
    ps v m (string []args)
    {
    demo ();
    }
}
```

In the above program the demo method it is being called itself multiple times such that it provides an infinite loop.

In order to stop infinite loop which occures during recurssive method we need to use base condition.

Base condition :- Base condition is used to stop the infinite loop which occures in recurssive method.

ex.

```
class P4
{
    p s v demo (int a)
    {
     if (a==5)
     {
```

```
        return 0;
        }
        s.o.pln (a++);
        demo (a);
    }
    p s v m (string []args)
    {
        demo (0);
    }
}
```

√(early binding)

Method Overloading :- Class consists of more than one method with same name & diff^{n} arguments is called as method overloading.

Inside the class block we cannot declare more than one method with same name & same arguments (we get CTE).

ex.

```
class P5
  {
    p s v test ()
    {
    s.o.pln ("test ()-1");
    }
    p s v test (int a)
    {
    s.o.pln ("test ()-2");
    }
    p s v m (string []args)
    {
    test (12) ;    // mcs
```

```
    }
}
```

Early binding [verify]

① same name class
② args → 2 = test
→ no args Data
↓ int

test ()
{ }
→ test (int a) 12
{ }
main
{ test (12); }

compile t
select → R
→ test (int
↓
Run time
executed
test ( ) =

ex.

```
class P5
{
    ps v test (double a)
    {
    s.o.pln ("test (double)");
    }
    psv test (int a)
    {
    s.o.pln ("test (int)");
    }
    psvm (string[] args)
    {
    test ('a'); mcs // char → int (auto widening)
    }
}
```

ex.

```
class P5
{
    p s v test (char n)
    {
    s.o.p ln ("test (char)");
    }
    p s v m (string[] args)
    {
    test ((char) 97.123); //mcs//double → char
                          //(Narrowing)
                ↓
            cast operator
    }
}
```

Global variable :-

Global variables are declared inside the class block.

Global variables are classified into two types

i) Static global variable
ii) Non static global variable.

Static global variable :-

A variable which is declared inside the class block & prefixed with static keyword is called as static global variable.

Inside class block we can declare more than one static global variable but variable name has to be different

Syntax :-

```
Static datatype var-name = value;
```

ex.

```
class A1
{
static int n = 12;
static char m = 'a';
}
```

Static variable :-

Static variable consists of default valu
(depends on datatype).

We can access a static variable in two ways

1) Directly by the variable name.
2) With the help of class name as referance.

ex.

```
class A2
{
    static int a;
    p s v m (string [] args)
    {
        // printing

        s.o.pln (a);    //0
        s.o.pln (A2.a);  //0

        // storing or intializing a value.

        a=100;
        s.o.pln (a);    // 100

        a=200;
        s.o.pln (A2.a);  //200

    }
}
```

Rules for accessing static variable :-

1) If a class consists of a static variable & local variable with same variable name after declaration of local variable we cannot access static variable directly. (priority is given for local variable first.)

⇒ On such case to access static variable we need to take the help of class name as refferance.

```
class A3
{
    static int a;
    {
    public static void main (String [] args)
    {
        a = 100
        s.o.pln (a); //100

        int a = 50; // local variable
        s.o. pln (a); // 50

        a = 25;
        s.o. pln (a); //25

        s.o. pln (A3.a); //100
        A3. a = 200;
        s.o. pln (A3.a); //200
    }
}
```

2) We can access a static variable from one class to another class. with the help class name as referance.

ex.

```
class A3
{
    static int a = 100;
}

class A4
{
    static int n = 150;

    psvm (string[] args)
    {
        s.o.pln (n);        //150
        s.o.pln (A3.a);     //100

        A3.a = 300;

        s.o.pln (A3.a);     //300;
    }

}
```

Static block :-

A block which is prefixed with static keyword is called static block.

Static block are declared inside the class block.

Syntax :-

```
static
{
  statement;
}
```

We can declare more than one static block inside class block.

All the static blocks get executed in top to bottom order

ex.

```
class A6
{
    static
    {
        s.o.pln ("static block");
    }
    static
    {
        s.o.pln ("static block");
    }
    p s v m (string [] args)
    {
```

```
        S.O.println("Main Method");
    }
}
```

o/p

SB

SB

MM

Can we access or initialize static variable in side static block (in how many ways).

We can initialize a static variable from static block before the execution of main method begins. Hence static block is also known as static ~~initio~~ initializer block.

A programmer cannot call static block because static block does not consists of any name.

Static block does not return any value back to the programmer because it does not consists of return type.

Static block gets executed before the execution of main method begins.

ex.

```
class A7
{
    static int a=10;
```

```
static
{
 a = 10;
 S.O.pln (a);
}
A7.a = 10;
S.O.pln (A7.a);
}
 p s v m (string [] args)
{
 S.O.pln (a);
}
}
```

NOTE :- Inside static block we can declare local variable also. After declaring local variable if it consists of same variable name of static variable later to access a static variable from static block we need to take the help class name as referance.

ex.

```
class A7
{
    static int a;

  static
 {
  a = 10;
```

```
S.O.pln (a)   // 10

int a=25;
S.O.pln (a);   //25
S.O.pln (A7.a); //10
}

{  p s v m (string[]args)
{
   S.O.pln (a); //10
}
}
```

Static Method :-

A method which is prefixed with static keyword is called as static method.

Static methods are declared inside the class block.

We can access static method in two ways.

1) Directly
2) Class name as referance.

ex.

```
class A8
{
   Static public void test()
   {
   S.O.pln ("A8 - test()");
   }
```

```
    p s v m (String[] args)
    {
    test();
    A8.test();
    }
}
```

In order to access a static method from one class to another class we need to take the help class name as referance.

ex.

```
class AG
{
    p s v test()
    {
    s.o.pln("A8-test()");
    }
}
```

ex.

```
class Ag
{
    p s v m
    {
    s.o.pln A8.test();
    }
}
```

Can we access a static method from static block

Yes we can with help of directly & class name as reference. (Both no argument & parameterised methods).

ex.

```
class A8
{
    static public void test (int a)
    {
    s.o.pln ("test()"+a);
    }
    static
    {
    test (12);        // test() 12
    A8.test (23);     // test ()23
    }
    p.s.vm (String [] args)
    {
    }
}
```

Can we access a static variable from ~~static~~ programmer created static method.

Yes we can with help of variable name directly & class name as reference.

```
class A10
{
    static int a;

    public static void test ()
    {
     a=15;
     s.o.pln (a);   // 15

     int a=25;      // local variable.
      s.o.pln (a); //25

      A10.a=35;
      s.o.pln (A10.a); //35
    }
    p s v m (string[] args)
    {
    test ();
    }
}
```

1) Initialize a static variable from parameterized static method.
2) Can we access a static method and static variable from one class to another class. with the help static block.
3) Can we initialize a static variable from one class to another class.
4) Can we access one static method from another programmer created static method.

Can we access static variable & static method from one class to another class with the help of static method.

* Loading process of class :-

During the loading process of a class the static variables gets memory allocation first.

Once after the loading process of static variables is completed static method get loaded.

Once after both static variable & static methods loading process is completed static blocks of the class will start getting executed in top to bottom order.

Once after the execution of static blocks is completed then the execution of main method begins

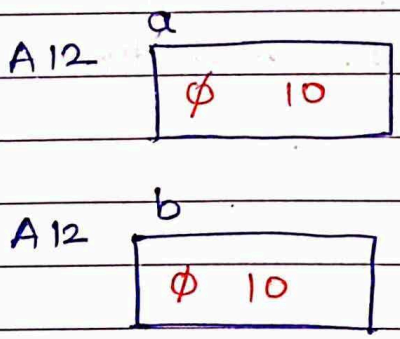
```
class A12
{
    static ~~a = 10~~ int a = 10;
    static int b = 10;

    p s v demo()
    {
        s.o.pln ("demo() - A12");
    }
    Static
    {
        A12.demo();
```

```
    S.O.pln ("SIB");
    }
    public static void main (string[] args)
    {
      S.O.pln (a);
    }
}
```

Searching Memory → SV
SM
SB

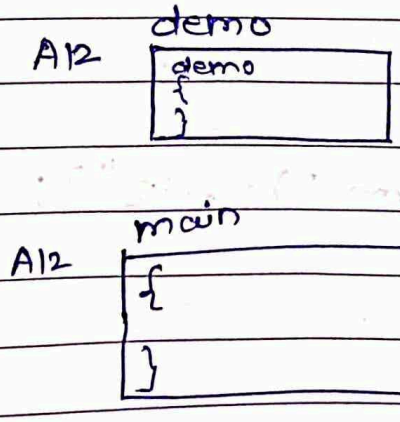
→ call → execution

1) SV



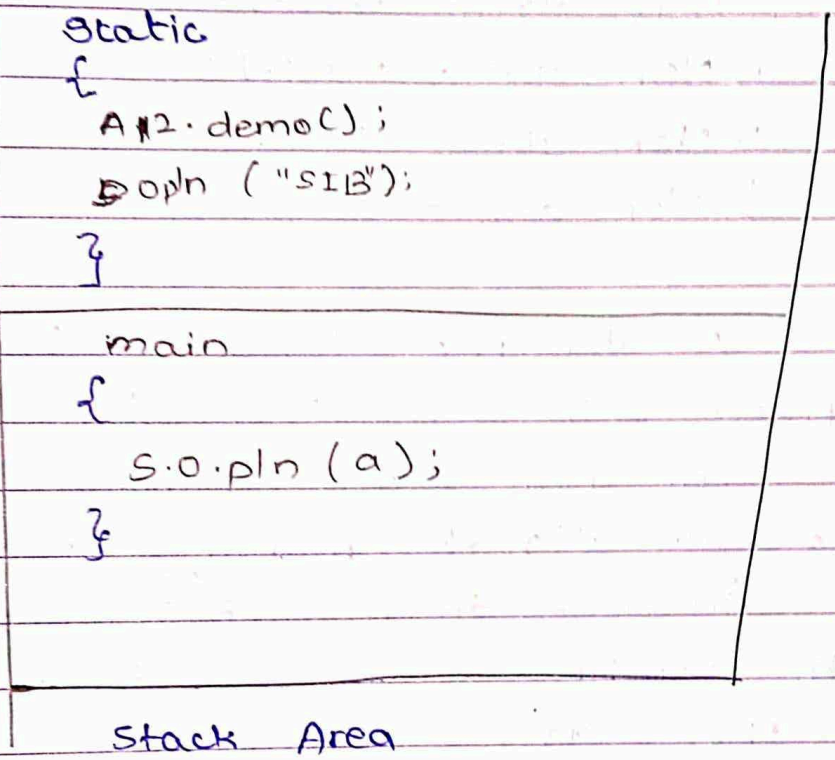
Static area

2) SM



3) SB

execution



Non static

Non static members are declared inside class block.

Non static members are classified into

i) Non-static variable
ii) Non-static block
iii) Non-static Method

All the non static members are not prefixed with static keyword.

For non static members the compiler does not allocate memory automatically.

In order to allocate memory for non static members we have to create object for the class.

Before creating an object for a class we need to create object reference variable first.

Object reference variable is used to store address of the object which is created for the class.

Syntax :-

Store address ↓

```
class.name  variable.name;
```

↑ object reference variable

ex.

```
P1 ref;
```

We can create an object for a class with the help of constructors.

Constructors :-

Constructor is used to create an object for a class. with the help new key-word

Syntax :-

Create an object

```
new  class_name ();
```

constructor of the class

We can create more than one object for the class but reference variable name has to be different.

ex.

```
class P1
{
    p s v m (string [] args)
    {
    P1 ref = new P1();
    P1 ref1 = new P1();

    S.O.pln (ref);    // P1@15db9742
    S.O.pln (ref1);   // P1@6d06d69c
    }
}
```

Non static variable :-

Non static variables are declared inside the class block. & not prefixed with static keyword.

We can declare more than one non-static variable ~~but~~ inside class block but the variable name has to be different.

In order to load non static variable object creation is mandetory.

We can access ~~ob~~ and initialize non-static variable with the help of object referance variable.

ex.

```
class P1
{
   int a ;
   char a1 = 'A';

    p s v m (string[]args)
   {
     P1 ref = new P1();

    s.o.pln (ref.a);
    ref.a = 50;
    s.o.pln (ref.a);
    s.o.pln (ref.a1);
    }
}
```

Non static variable consists of default values.

NOTE :-

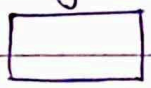
We can create an object from one class to another class.

```
class P2
{
    int a = 16;
}


Class P3
{
    int b = 15;

    psvm (String [] args)
    {
    P2 obj = new P2();

    s.o.pln (obj);
    s.o.pln (obj.a);

    P3 obj1 = new P3();

    s.o.pln (obj1.b);
    }
}
```

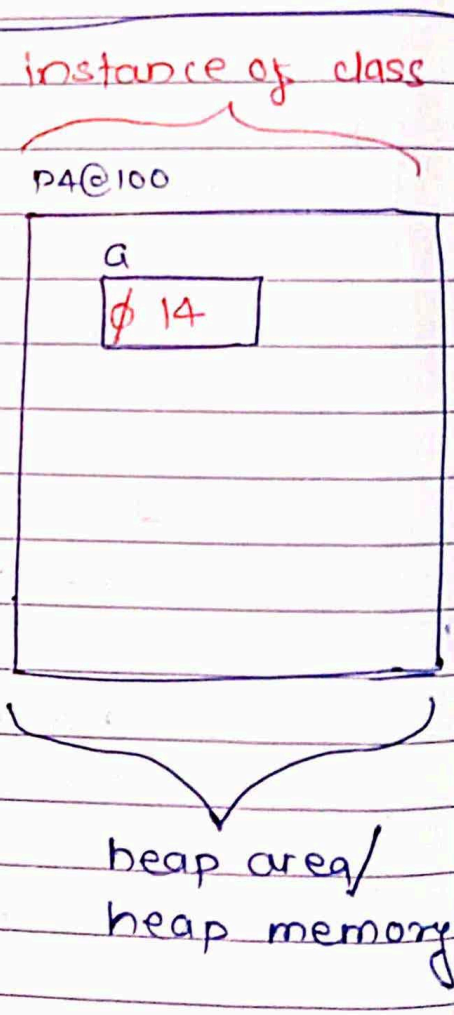
instance = object of class
object = block of memory

```
class P4
{
    int a = 14;

    p s v m (String []args)
    {
    P4 obj = new P4()

    s.o.pln (obj);      // P4@100
    s.o.pln (obj.a);    // 14
    }
}
```

```
main
{
obj creation → P4 obj = new P4()
```



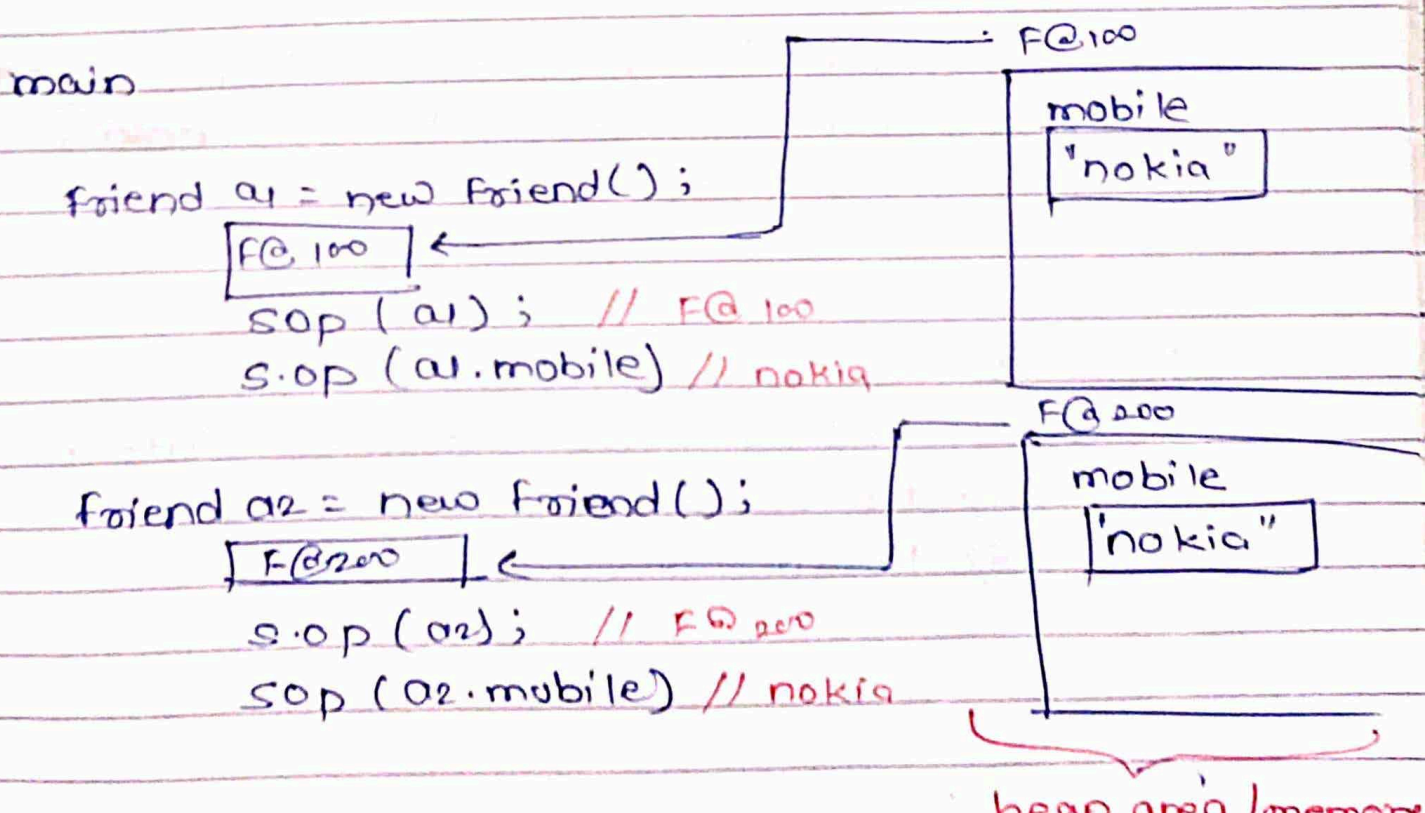
```
sop (obj);
sop (obj.a);
}
```



P4 → NSM
1) NSV
2) NSM
3) NSB

new = block of memory

```
class Friend
{
    String mobile = "nokia";

    psvm (String[] args)
    {
        Friend a1 = new Friend();

        S.O.pln (a1);
        S.O.pln (a1.mobile);   // nokia

        Friend a2 = new friend();

        S.O.pln (a2);
        S.O.pln (a2.mobile); // nokia
    }
}
```



```
class Friend
{
    String mobile = "nokia";

    p s v m (String []args)
    {
    Friend a1 = new Friend();

    s.o.pln (a1);          Friend@15db9742
    s.o.pln (a1.mobile);   // Nokia

    Friend a2 = new friend();

    s.o.pln (a2);          Friend@6d06d69c
    s.o.pln (a2.mobile);   // nokia

    a2.mobile = "Iphone-14";

    s.o.pln (a2.mobile);   // iphone-14
    s.o.pln (a1.mobile);   // nokia
    }
}
```

Types of constructors:-

- Constructors are classified into two types
  - i) No argument constructor
  - ii) Parameterised constructor

i) No argument constructor :- The constructor which does not takes any values such constructors are called as no arguments constructors.

No ~~The~~ args constructors are declared inside the class block.

ex.

```
class P5
{
    P5()
    {
    S.O.pln("P5()");
    }

    p s v m (string[] args)
    {
    P5 obj = new P5();
    }
}
```

ii) Parameterised Constructor :- The constructor which which takes arguments & value such constructor are called as parameterised constructor.

Parameterised constructor are declare inside class block.

```
class P5
{
    P5(int a, int b)
    {
    S.O.pln(a); //
```

```
        S.O.pln (b);    //2
    }

    p s v m (String[] args)
    {
    p5 obj = new p5(1,2);
    }
}
```

Default constructor

When a programmer fails to define atleast one constructor inside the program. the compiler will automatically add ~~default~~ constructor of no argument type such constructors are called as default constructor.

ex.

```
class P6
{
    p s v m (string [] args)
    {
    P6 obj = new P6()
    }
}
```

converted to

```
class P6
{
    // default constructor
    P6()
    {
    }   // added by compiler automatically

    p s v m (string [] args)
    {
    P6 obj = new P6()
    }
}
```

Constructor Overloading :-

Class consists of more than one constructor with same name class name & diffn arguments is called constructor overloading.

ex.

```
class P7
{
   P7()
   {
   S.O.pln("P7()");
   }
   P7(int a)
   {
   S.O.pln("P7(int)");
   }
   p s v m (String[] args)
   {
   P7 obj = new P7(); //"P7()"
   }
}
```

```
class P7
{
   P7(double a)
   {
   S.O.pln("P7(double)");
   }
   P7(int a)
   {
   S.O.pln("P7(int)");
   }
```

```
    p s v m
    {
    P7 obj = new P7('b');  // same data = NP
    }                      //widening -> SP -> LP
}                          // Ascll -> 98 int
```

∴ OP = 98

// "P7 (int)"

~~Using~~
~~Non static variables~~

Using Non static variable inside no argument constructor :-

We can access and initialize non static variable directly from no-argument constructor.

ex.

```
class P7
{
   int a = 50;

   P7()
   {
   s.o.pln (a); //50
   a = 123;
   s.o.pln (a); //123
   }
   p s v m (string [] args)
   {
   P7 obj = new P7();
   s.o.pln (obj.a); //123
   }
}
```

```
class P8
{
    int a = 50;

    P8 ()
    {
    S.o.pln (a); //50
    }
    p s v m (string [] args)
    {
    P8 obj = new P8 ();
    S.o.pln (obj); // P8@100
    }
}
```

Load
Static → SV
SM
SB → T
B

```
main
{
P8 obj = new P8();
```

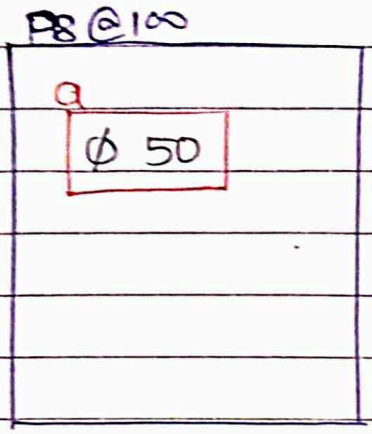
P8@100

```
S.o.pln (obj);
}
```

```
P8()
{
S.o.pln (a);
}
```

New keyword = block of memory

constructor → const. Nature → ~~block of~~ Load → NSM
Executed



Constructor
↓
P8

C → C.N. → Load = NSM
C → Constructor execution

Loading of NSM ↑ For this object creation is mendatory
- NSV
- NSM
- NSB → Top ↓ Bottom

class block

Non static members loading completed

a; directly → local	static → class block	NSM → class block
No	No	Yes

Note:-

Inside constructor to access NSV after declaration of local variable. If it consists of same variable name of NSV.

Later to access NSV we need to take the help "this" keyword.

```
class P8
{
  int a=50;

  P8()
  {
  S.o.pln (a); //50
  int a= 25; // local : directly
  S.o.pln (a); //25
  S.o.pln (this.a); //50
  }
  p s v m (string[]args)
  {
  P8 obj = new P8();

  S.o.pln (obj); // P8@15db97-42
  }
```

Note :-

We cannot use this keyword inside static members (static methods, static blocks).

In order to access NSM from static Members we need to create an object & access with the help object referance variable.

Using / accessing Non static variable inside parameterised constructor :-

To ~~the~~ ~~can~~ initialize a non static variable ~~inside~~ during the execution of constructor we need to take the help of parameterised constructor.

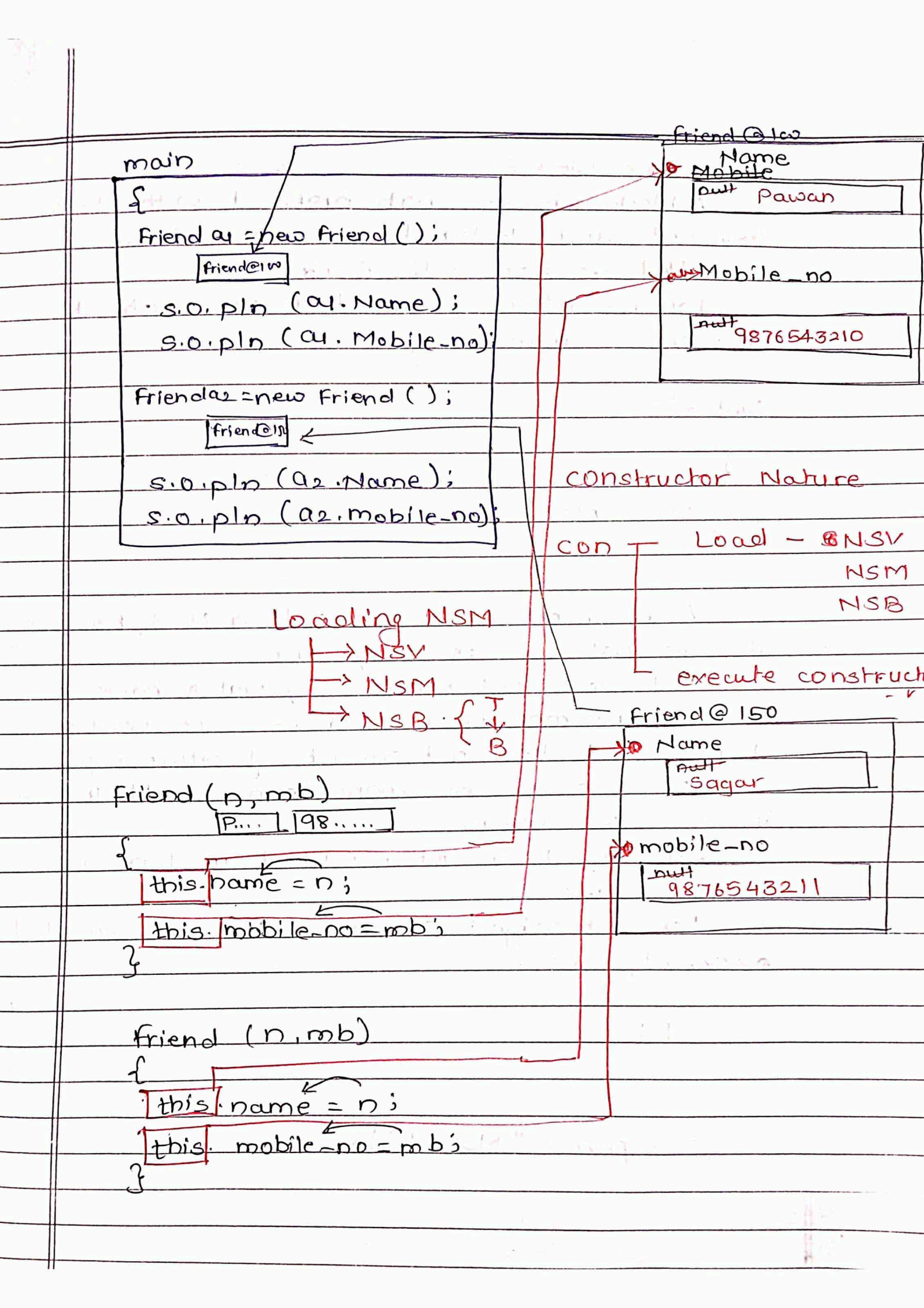
```
class P8
{
~~P8(int n)~~ int a;

  P8(int n)
  {
  s.o.pln (a); // 0
  s.o.pln (n); // 15
  ~~a~~this.a = n;  // passing value
  s.o.pln (a); // 15
  }
  public static void main (string [] args)
  {
  P8 obj = new P8 (15);

  s.o.pln (obj.a); // 15
  }
}
```

"this" keyword :-

"this" keyword points to current object based on object creation.

We can use "this" keyword only inside non-static members. (Constructors, NSB, NSM).

```
class Friend
{
    String Name;
    String Mobile_no;

    Friend (~~int~~ String n, String Mb)
    {
     this.Name = n;
     this.Mobile_no=mb;
    }
    psvm (string []args)
    {
    Friend a1 = new Friend ("Pawan", "9876543210");
    s.o.pln (a1.Name); // Pawan
    s.o.pln (a1.Mobile_no); //9876543210

    friend a2 = new Friend ("Sagar", "9876543211");
    s.o.pln (a2.Name); // Sagar
    s.o.pln (a2.Mobile_no); //9876543211
    }
}
```


main
{
Friend a1 = new Friend ();
friend@100
s.o.pln (a1.Name);
s.o.pln (a1.Mobile_no);
Friend a2 = new Friend ();
friend@150
s.o.pln (a2.Name);
s.o.pln (a2.mobile_no);
friend@100
Name
Mobile
null
Pawan
Mobile_no
null
9876543210
constructor Nature
con
Load - NSV
NSM
NSB
execute construct
Loading NSM
NSV
NSM
NSB
T
B
friend (n, mb)
P....
98.....
{
this.name = n;
this.mobile_no = mb;
}
friend @ 150
Name
null
Sagar
mobile_no
null
9876543211
friend (n, mb)
{
this.name = n;
this.mobile_no = mb;
}

Non static Block :-

A block which is not prefixed with static keyword and declared inside the class block is called as Non-static Block.

For execution of non static block object creation is mandatory.

Syntax :-

```
{
  statements;
}
```

A programmer cannot call a NSB because it does not consists any name.

NSB does not return any value back to the programmer because there is no return type or it does not consists of return type.

We can declare more than one NSB inside class block.

All the NSB get executed in top to bottom order.

For each object creation NSB gets executed only once.

```
class P9
{
  {
    S.o.pln ("NSB-1");
  }
```

```
    {
      S.O.pln ("NSB-2");
    }
    p s v m (String[] args)
    {
      Pg obj = new Pg();
      Pg obj2 = new Pg();
    }
}

class Pg
{
  {     // during object creation
  S.O.pln ("NS - block");
  }

  static
  {     // before execution of main method begins
  S.O.pln ("SB - block");
  }

  Pg()
  {     // after object creation
   S.O.pln ("Pg()");
  }

  p s x m (String[] args)
  {
  Pg obj = new Pg();
  }
}
```

1) Can we access (print) a NSV from NSB
2) Can we initialize a NSV from NSB. (How many ways)

Instance Initializer block :-

We can initialize a NSV from a NSB during object creation.

Because of this nature NSB is also called as Instance Initializer block.

```
class P10
{
    int a;
    {
this.a = 15; //during object creation : IIB
    }

    p s v m (string [] args)
    {
    P10 obj = new P10();

    s.o.pln (obj.a); // 15.
    }
}
```

Non static Methods:-

A method which is declared inside the class block and not prefixed with static key-word such methods are called as non-static ~~keyword~~ Method

for NSM object creation is mandatory.

We can access a NSM with the help object reference variable.

We cannot declare more than one NSM inside ~~with same name~~ class block with same name and same arguments.

```
class P11
{
    public void test ()
    {
    s.o.pln ("test ()"); // test ()
    return (100);
    }

    psvm (string []args)
    {
    P11 obj = new P11 ();
    int n = obj. test(); // return type using NSM
    s.o.pln (n);    // 100
    }
}
```

1) Can we access a NSV from NSM

→ Yes.

```
class P12
{
    int String s1;

    public void test()
    {
    // printing
    System.o.pln (s1); //null
    s.o.pln (this.s1); //null
    //initializing
    s1 = "mango";
    s.o.pln (s1);      //mango
    this.s1 = "mango- icecream";
    s.o.pln (this.s1); //mango-icecream
    }

    ps vm (string [] args)
    {
    P12 obj = new P12();
    obj. test();
    }
}
```

★ Can we access a NSV from a NSParame-terised ~~constructor~~ Method.

★ Can we initialize a non static variable from a parameterised NSM.

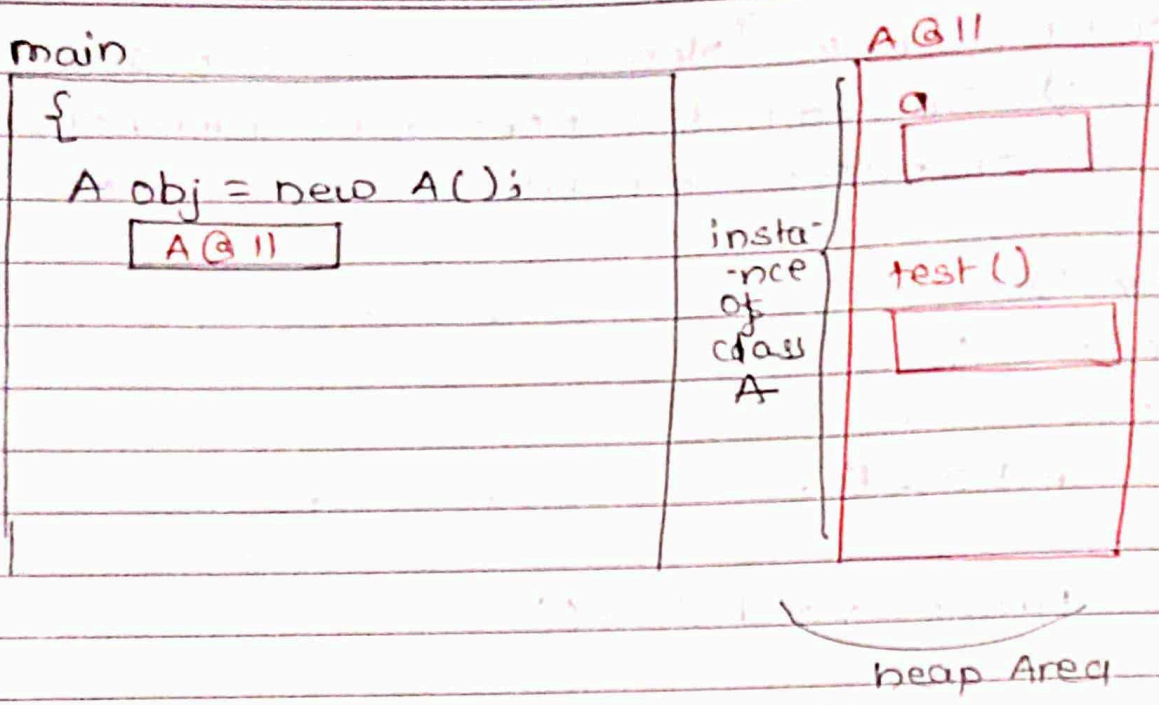
★ Can we access a non static Method ~~from~~ inside non static block, No argument constr-uctor, Parameterised constructor. (In how many ways).

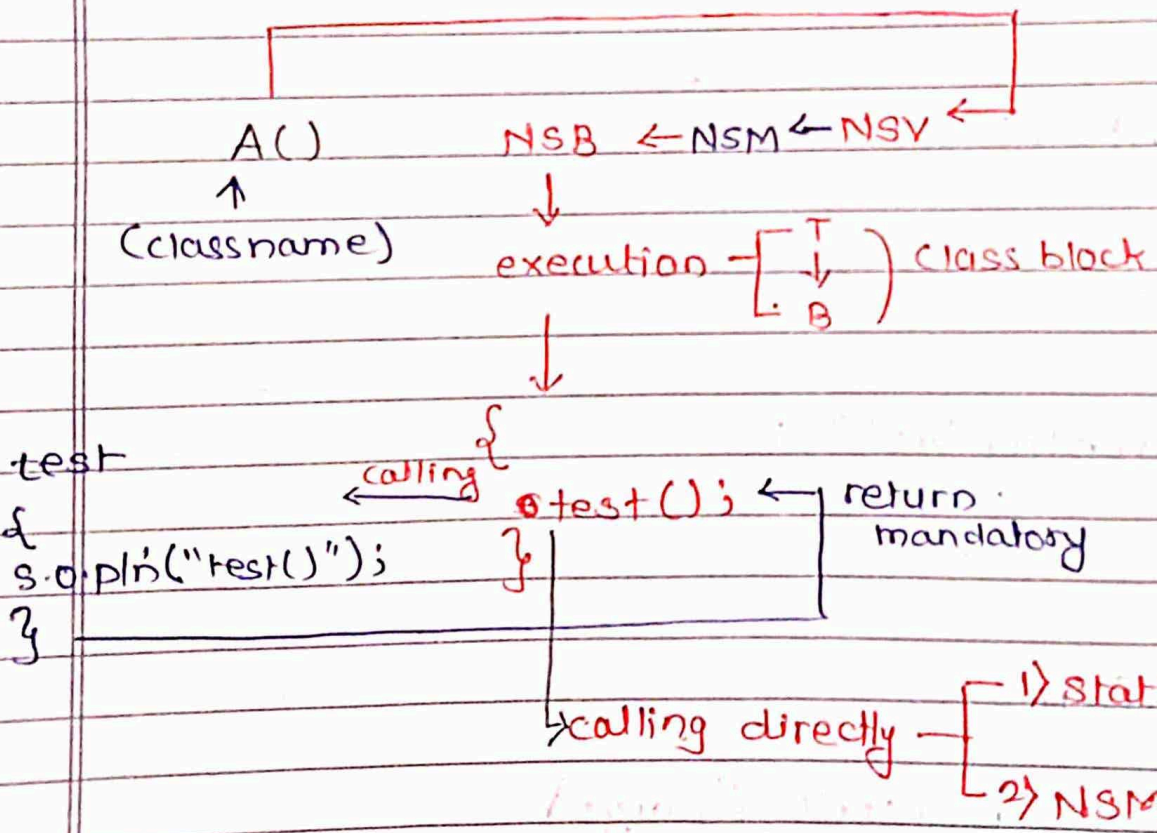
Loading Process of Non-static Members :-

⇒ To load the Non-static members object creation is mandatory.

ex 1)

```
class A
{
    int a = 12;

    public void test()
    {
    s.o.pln ("test()");
    }
    {
    test();
    }

    A()
    {
    s.o.pln("A()");
    }
}

class Demo
{
    psvm (string[] args)
    {
    A obj = new A();
     s.o.pln (obj);
    s.o.pln (obj.a);
    obj.test();
    }
```


main
{
A obj = new A();
A@11
A@11
a
insta-
-nce
of
class
A
test()
heap Area


A()
(classname)
NSB ← NSM ← NSV
execution
T
B
class block
test
{
s.o.pln("test()");
}
calling
{
test();
}
return
mandatory
calling directly
1) staticM
2) NSM

→ To load NSM object creation is mandatory.

i) During object creation NSV gets loaded first

ii) Once after loading process of NSV gets completed then NSM gets loaded next.

iii) Once the loading process of both NSV & NSM gets completed then All the NSB ~~starts~~ gets executed in top to bottom order.

iv) Once after the execution of NSB is completed then later constructor will get executed.

Accessing / Calling of SM :- i) Direct ways.

1) Static Variable :- From Static Method
From static block
From NSM
From NSB
From constructor
} i) Directly
ii) Class-name as referance

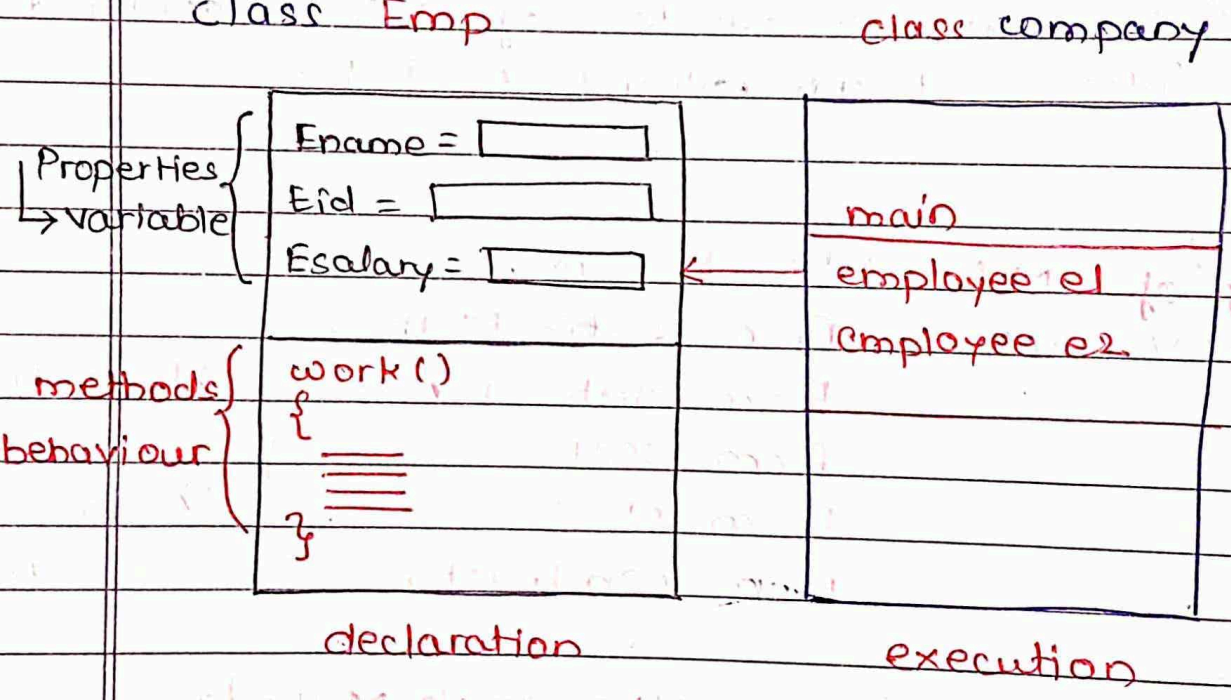
2) Static Method :- From another static Method
From static block
From NSM
From NSB
From constructor
} i) Directly
ii) class-name as referance

Accessing NSM :- Direct ways

1) ~~Sta~~ Non Static Variable :- Static method
Static block
NSM
NSB
constructor
} objectref. variable } object creation is mandatory
} i) Directly
ii) this keyword

2) Non static

Method :- Static Method } obj. ref. } object
Static block } variable } creation
NSM
NSB } 1) Directly
constructor } 2) this keyword.



ex.

```
class Emp
{
  String Ename;
  int Eid;
  double Esalary;

  Emp (string n, int id, double s)
  {
    this. Ename = n;
    this. Eid = id;
    this. Esalary = s;
```

```
    }
    public void work ()
    {
    s.o.pln (this.Ename + " : Emp is working ");
    }

    public void details ()
    {
    s.o.pln (this.Ename);
    s.o.pln (this.Eid);
    s.o.pln (this.Esalary);
    }
}

class Emp
{
    p s v m (string[] args)
    {
    Emp e1 = new Emp("Rahul", 1, 10000);
    Emp e2 = new Emp("Sagar", 2, 11000);

    e1.details ();
    e2.details ();
    e1.work();
    }
}
```

Object Oriented Programming :-

Object oriented programming its a concept where we take all the real time examples into a programatical worlds

An object it is an real time example which consists of two elements

1) Properties.
2) Behaviours.

The properties of an object is converted into variables.

The behaviour of an object is converted into methods.

Since variables & methods are the members of class hence the class is called as blue print of an object.

Object oriented programming supports following pillers

i) Encapsulation
ii) Inheritance
iii) Polymorphism
iv) Abstraction

i) Encapsulation :-

It is a process of hiding the data and providing binding between properties & behaviour of an object.

Advantages of Encapsulation :-

- Data Hiding

ii) Inheritance :-

It is a process of aquiring the properties and behaviours from parent to child is called as Inheritance.

Advantages

- Reusability

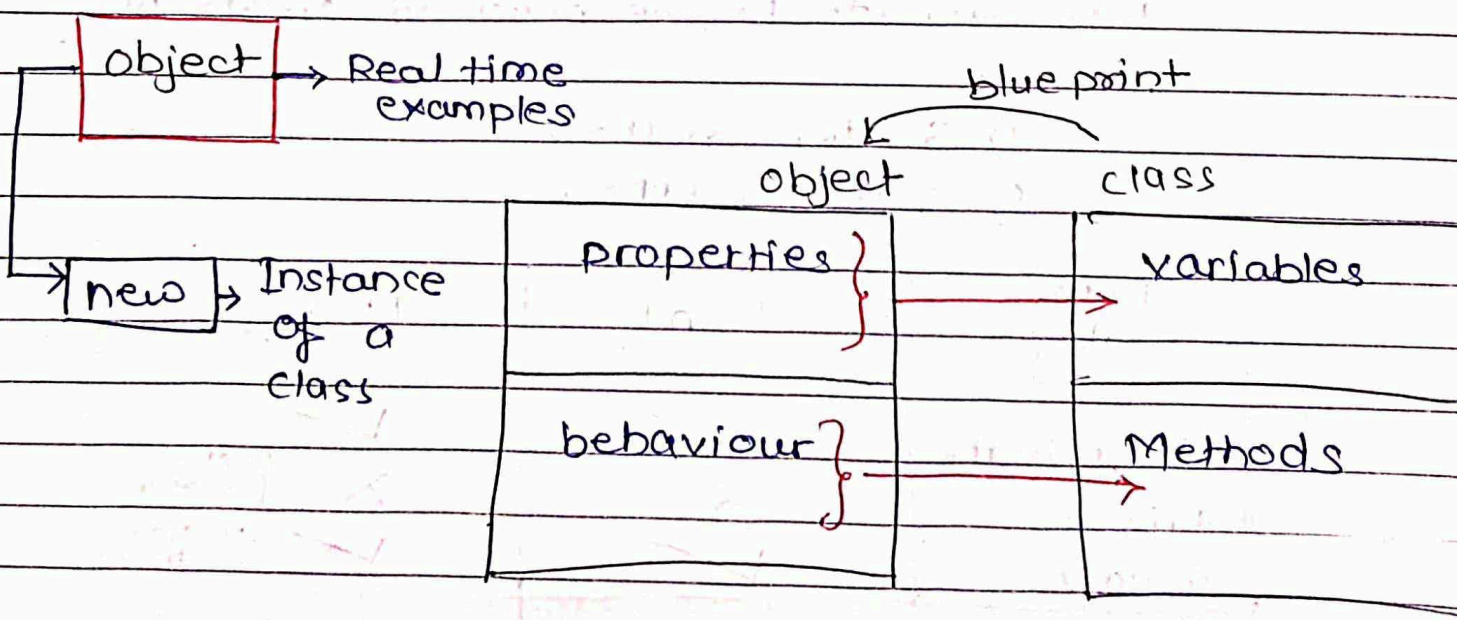
iii) Polymorphism :-

Poly means many, morphism means different forms

An object performing or exhibiting more than one form called as polymorphism.

iv) Abstraction :-

Its a process of hiding the method implementation is called as abstraction.



1) Encapsulation :-

It is a process of ~~achie~~ ~~the data~~ achieving the data hiding is called as Encapsulation.

We can perform data hiding with the help of private keyword.

Data Hiding :-

Hiding the data from outside world is called as data hiding.

Private keyword :-

If we prefix a variable with private keyword. we cannot access the value present inside the private variable from one class to another class. (for printing the value and for modifying the value which is present inside private variable.

Even after hiding the data with the help of private keyword we should be able to print & modify the private data for that we need to create two methods

1) Getter method
2) Setter method

Data salary	Print	Modify	
Not private	✓	✓	
Private	×	×	Data hiding
Private	✓	✓	encapsulation

1) Getter Method :-

Getter Method is used to access or print a private data.

We need to call a getter method from printing statement.

In order to call a method from printing statement return value is mandatory.

Method header for getter method :-

```
modifier return type get attribute_name()
{ return private variable name ;
}
```

2) Setter Method :-

Setter method is used to modify the private data.

Method header for setter method :-

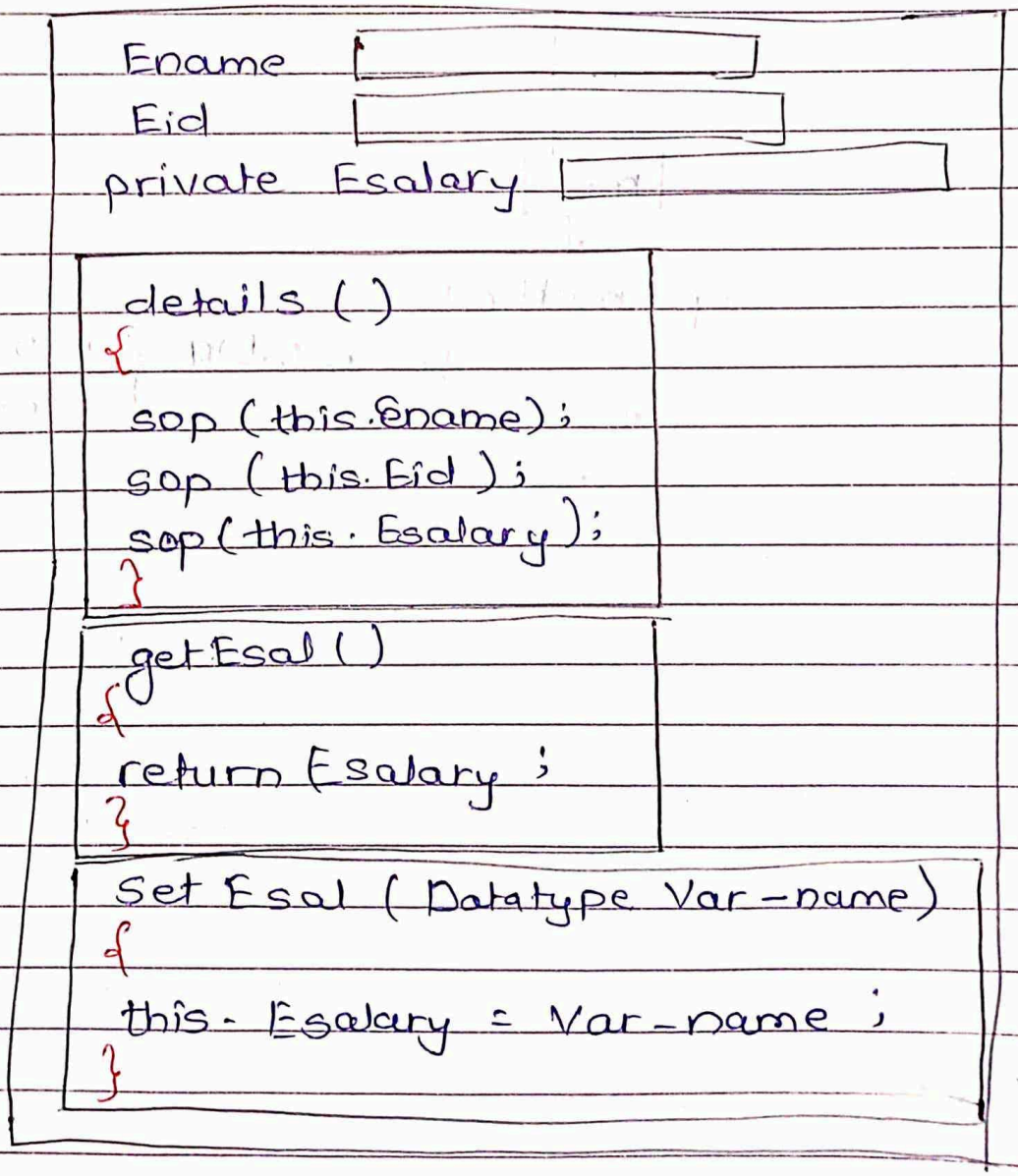
```
                    void
modifier return type set attribute_name ( datatype variablename)
{
  this.private variable name = Variable name ;
}
```

```
class Emp
{
   String Ename;
   int Eid;
private double Esalary;

  // gNsm : Getter method for print Esalary
```

```
public double getEsal()
{
return this.Esalary;
}
// NSM & SETTER for modify the salary
public void setEsal(double n)
{
this.Esalary = n;
}

Emp(string n, int id, double sal)
{
this.Ename = n;
this.Eid = id;
this.Esalary = sal;
}

public void details()
{
s.o.pln(this.Ename);
s.o.pln(this.Eid);
s.o.pln(this.Esalary);
}
}

class Cmp
{
psvm
{
Emp e1 = new Emp("shubham", 1, 10000);
```

```
// s.o.pln (e1.Esalary) ;- variable CTE
   s.o.pln (e1.getEsal ()) ; //method

// e1.Esalary =12000 ; //variable CTE
   e1.setEsal (12000); //method


      s.o.pln (e1.getEsal());
   }
}
```

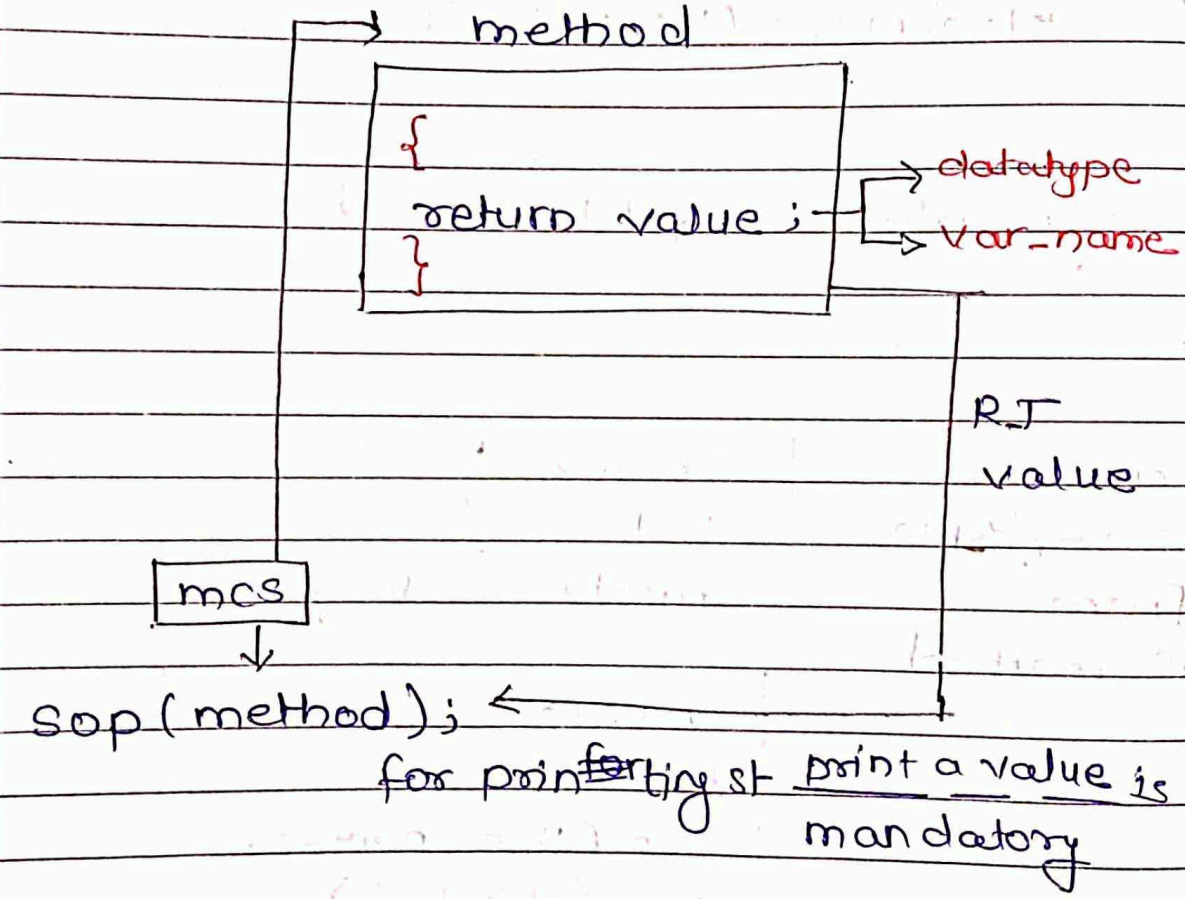
★ Accessing NS variable from NS method



~~method~~

~~mcs~~

~~sop(method)~~



for printing st print a value is mandatory

Relationship :-

Relationship is classified into two types
i) Is a relationship
ii) Has a relationship.

i) Is a relationship :-

Is a relationship it behaves like a parent & child.

ex. Laptop is electronic device.
Cricket is a sport.
Car is a vehicle.

ii) Has a relationship :-

Has a relationship behaves like a complete whole object & a part of it.

ex. Car has a engine.
Application has a set of programs.
Team has a players.
Bank has a lockers.
Electronic devices has hardware parts.

Inheritance :-

It is a process of aquiring the properties & behaviours from super class to sub class / parent class to child class is called as inheritance.

Parent class is also called as super class.

Child class is also called as sub class.

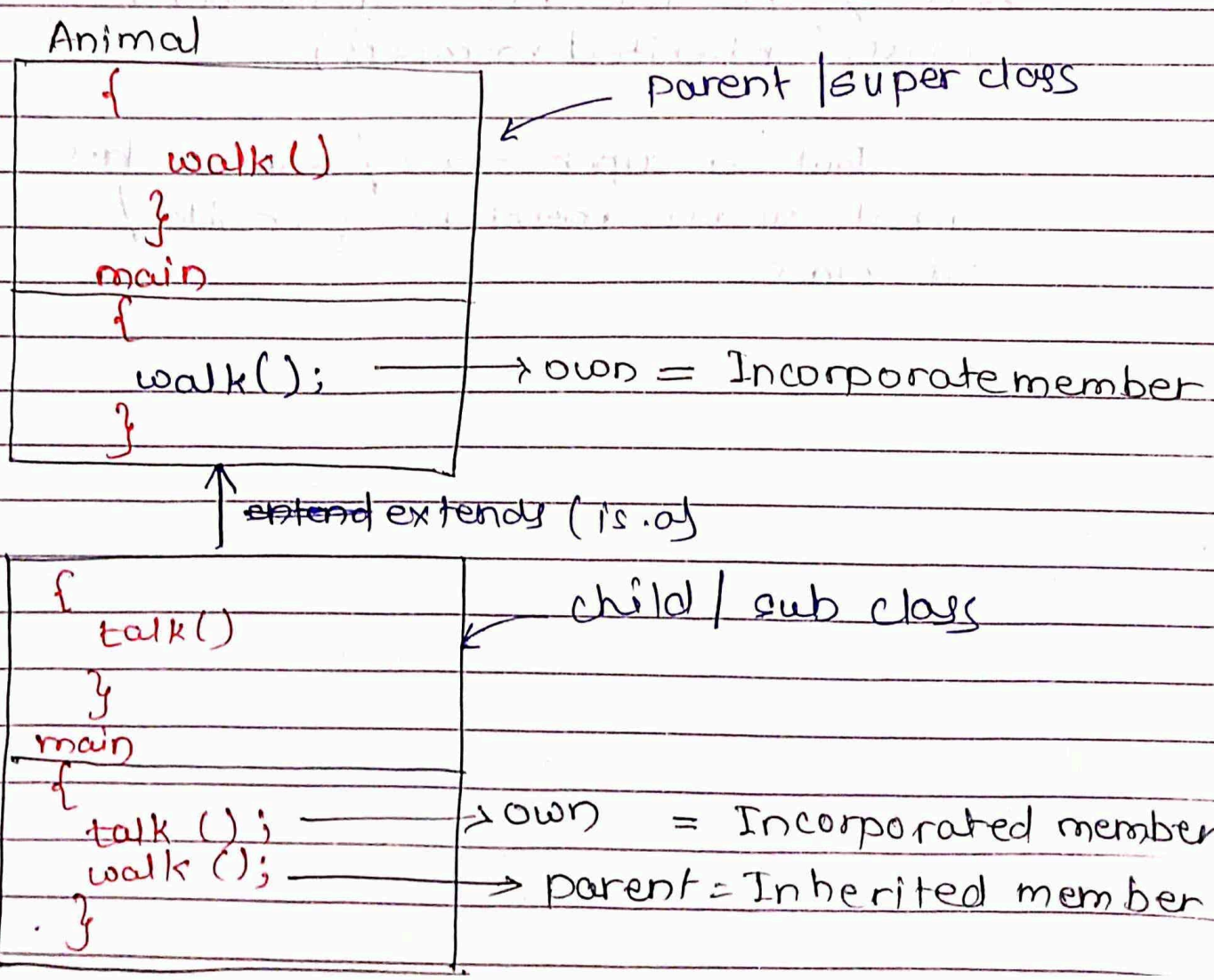
Inheritance is process of aquiring the properties & behaviours from super class to sub class is called as inheritance.

We can achieve inheritance with the help of extends keyword.

Extends key-word represents is a relationship.

```
class Animal
{
    public static void walk()
    {
        S.o.pln("All animals can walk");
    }
    psvm (String[] args)
    {
        walk(); // member of own class
        talk(); // member of child class : CTE
    }
}
```

```
class cat : extends Animal
{
    public static void talk ()
    {
        s.o.pln ("mee--ow meeow");
    }
    psvm (string [] args)
    {

        talk (); //member of own class
        walk (); //member of parent class
    }
}
```



Sub class cat can access two members easily

1) The members of its own class (Incorporated members).

2) The members of its parent class (Inherited members).

If a class is behaving like a parent class or ~~child~~ super class (class Animal).

It can access two members easily.

1) The members of its own class (Incorporate members).

And 2) The members of its parent class. (Inherited members)

But a super class / parent class cannot access members of child / sub class.

* Loading process of static members using inheritance

1) During the loading process parent class static members gets loaded first.

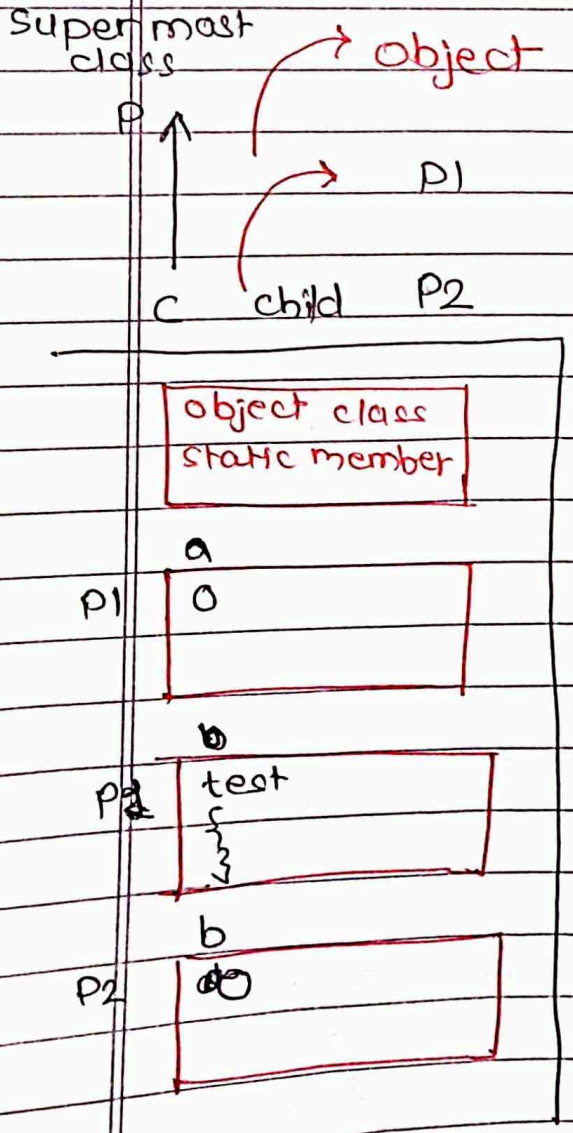
2) Once after loading process of static member of parent class is completely loaded. static members of its child class gets loaded second.

ex.

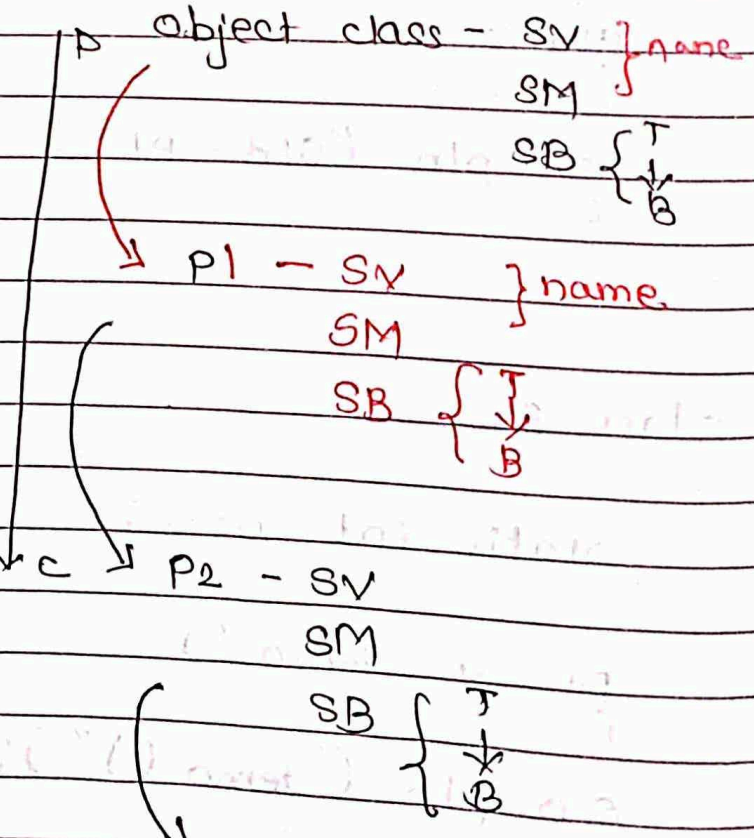
```
class P1
{
    static int a = 1;
    {
    public static void test()
    {}
        s.o.pln ("test()");
    }
    static
    {
     s.o.pln ("SIB - P1");
    }
}

class P2
 {
    static int b= 2;

    p's v demo()
    {
    s.o.pln ("demo()");
    }
    static
    {
```

```
        S.O.pln ("SIB-P2");
        }
        ps v m
        {
         s.o.plo  (b);
         s.o.pln  (a);
         test();
         demo();
         }
    }
```

① Searching Relation
child & Parent



② Loading static member



P2 demo
{
}

Static
area

[NP] → Not present

[P] → Present

child class P2

```
{
  S.o.pln(b);
  S.o.pln(a);
  test();
  demo();
}
```

S.o.pln(b); → directly [NP] → local [NP] → static of own class [P]

S.o.pln(a); → directly [NP] → local [NP] → static own [NP] → static parent [P]

test(); → own [NP] → parent [P]

demo(); → own class [P]

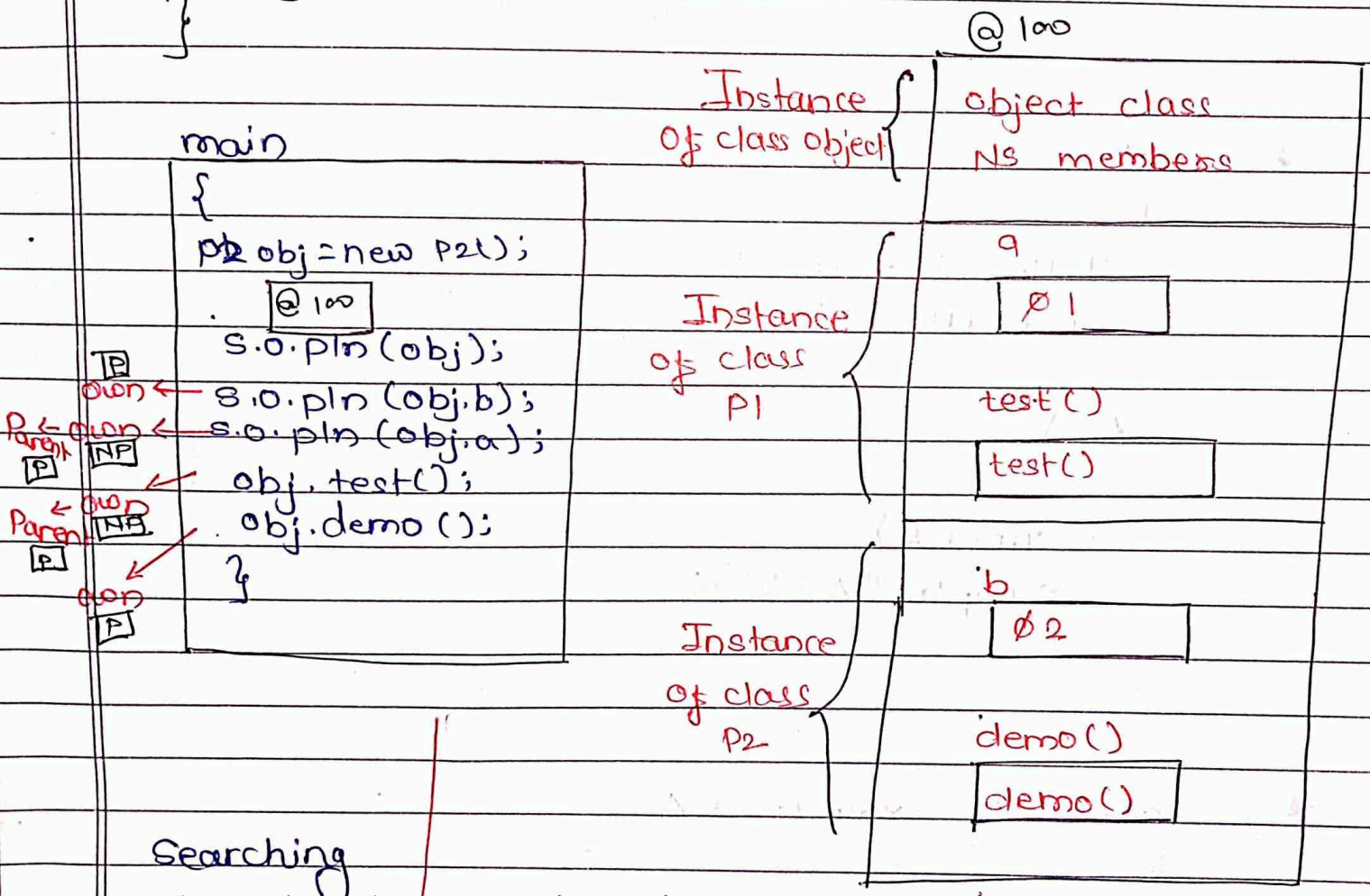
O/P :-

SIB - P1
SIB - P2
2
1
test()
demo()

Loading process of Nsm using inheritances:

```
class P1 // extend compiler automatically extends
{
  int a=1;

  public void test()
  {
    s.o.pln("test()");
  }

  {
  s.o.pln("IIB-P1");
  }
}

class P2 extends P1
{
  int b=2;

  public void demo()
  {
  s.o.pln("demo()");
  }

  {
  s.o.pln("IIB-P1");
  }

  public static void main
  {
```

```
P2 obj = new P2();
}
S.O.println (obj);
S.O.println (obj.b);
S.O.println (obj.a);
obj.test();
obj.demo();
}
}
```



Searching

parent object

child P2 → P1 → object

Loading

parent object → P1 → P2 → child

object { NSV, NSM, NSB, const } → B

P1 { NSV, NSM, NSB, const } → B

P2 { NSV, NSM, NSB, const } → B

// O/P

IIB - P1

IIB - P2

2

test()

demo()

Constructor chaining :-

It is a process of constructor calling another constructor is called as constructor chaining.

Constructor chaining is classified into two types.

1) ~~Supercall~~ Supercall statement [super()]

2) Supercall statement it is used to call the constructor of its parent class.

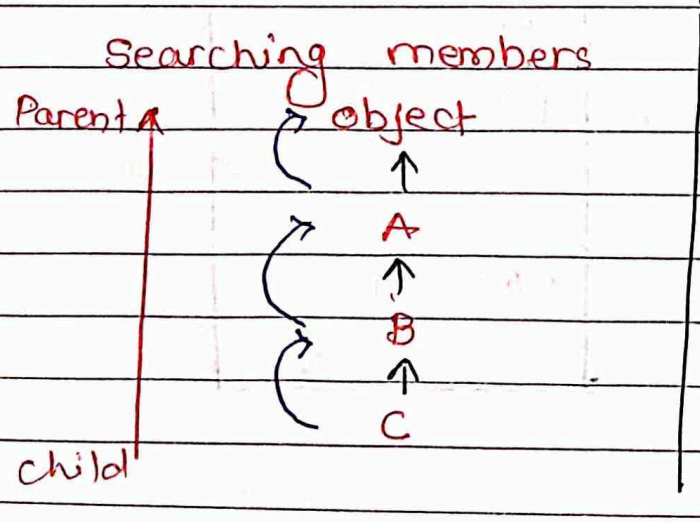
★
```
class A
{
  static int a = 1;
   int a1 = 1;

  A()
  {
    super()
    s.o.pln("A()");
  }
}
```

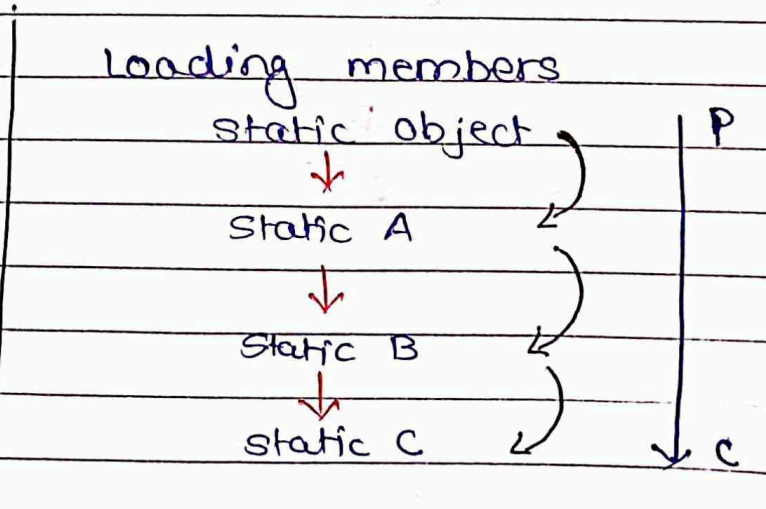
★
```
class B extends A
{
  static int b = 2
  int b1 = 2;

  B()
  {
    super();
```

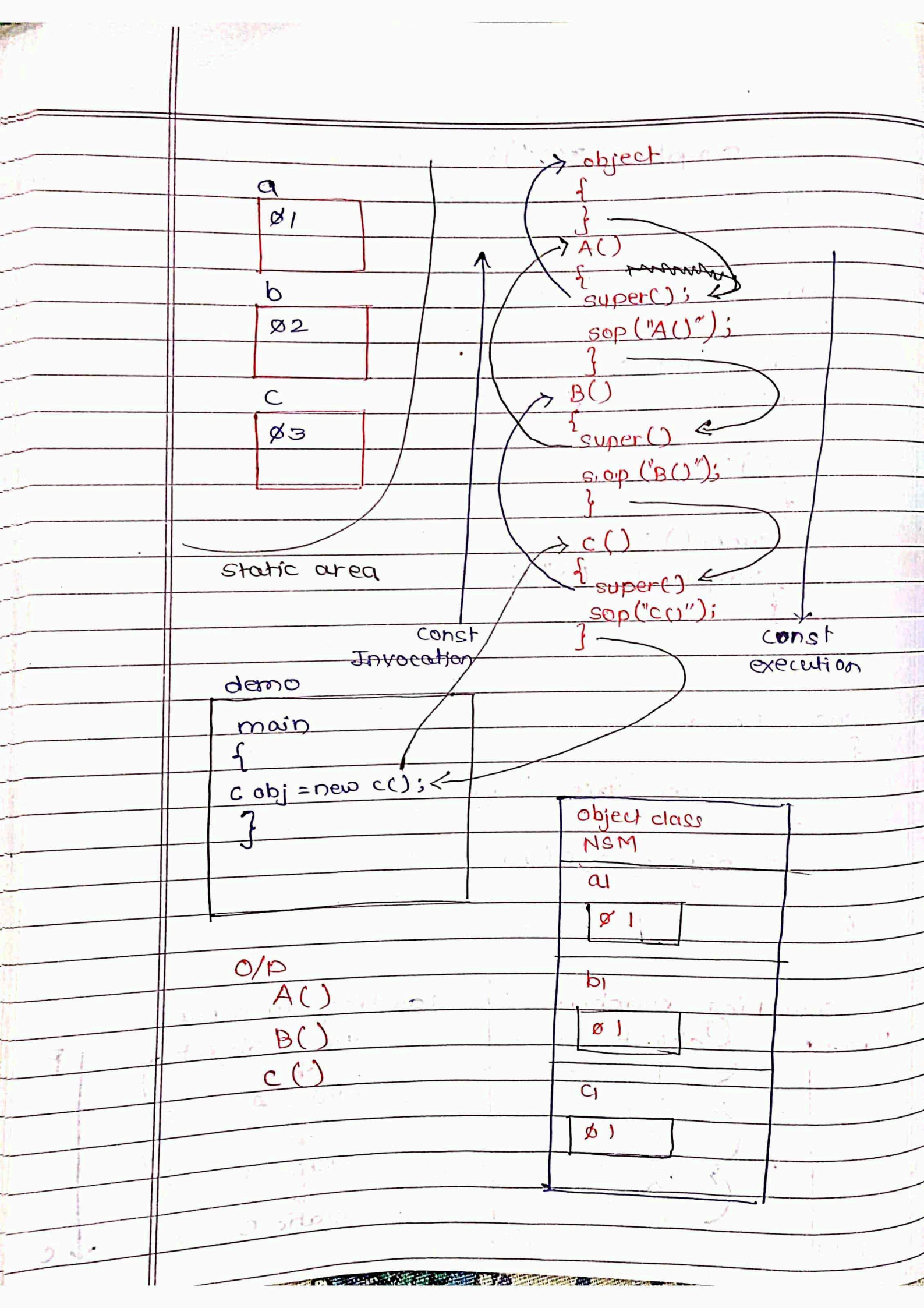
```
        S.o.pln ("B()");
        }
    }
```

★
```
class C extends B
{
static int c = 3;
 int c1 = 3;

 c()
 {
  super();
  s.o.pln ("C()");
 }
}
```

★
```
class Demo
{
    p s v m (string [] args)
    {
     C obj = new C();
    }
}
```






a
ø1
b
ø2
c
ø3
static area
object
{
}
A()
{
super();
sop("A()");
}
B()
{
super()
s.o.p ("B()");
}
C()
{
super()
sop("C()");
}
const
Invocation
const
execution
demo
main
{
c obj = new C();
}
O/P
A()
B()
C()
object class
NSM
a1
ø 1
b1
ø 1
c1
ø 1

Important points of super call statements :-

1) Supercall statement always has to be declared as first statement inside the constructor.

2) We cannot write super call statement in between the constructors.

ex.

```
A()
{
 super(); // works
 s.o.pln("A()");
}

A()
{
 s.o.pln("A()");
 super(); // Not works & CTE
}
```

3) Compiler by default automatically adds super call statement of no argument type as the first statement inside all the constructors of java. (No argument constructor, Parameterised constructors).

4) To call parameterised constructor of parent class writing super call statement ~~of parame-~~ & passing a value is mandatory inside child class constructor.

ex.

```
class A
{
    A(int a)
```

```
        {
          S.O.pln ("A()");
        }
    }

class B extends A
{
    B()
    {
      super (123); //mandatory
      S.O.pln ("B()");
    }
}
```

5) To call a no argument constructor of parent class writing a super (No args type) call statement is not mandatory inside the child class constructor.

ex.

```
class A
{
  A()
  {
    S.O.pln ("A()");
  }
}

class B
{
  B()
  {
```

```
        super (); // not mandatory
        S.O.pln ("B ()");
        }
    }
```

★ Constructor overloading with the help of super call statements :-

ex.

```
Class A
{
  A( int a)
  {
   S.o.pln ("A()");
  }
  A (double a)
  {
   S.o.pln ("B()");
  }
}

Class B
{
   B()
   {
   super ((int)123.12); // Narrowing :- 123
   S.o.pln ("B()");
   }
}
```

This call statement :-

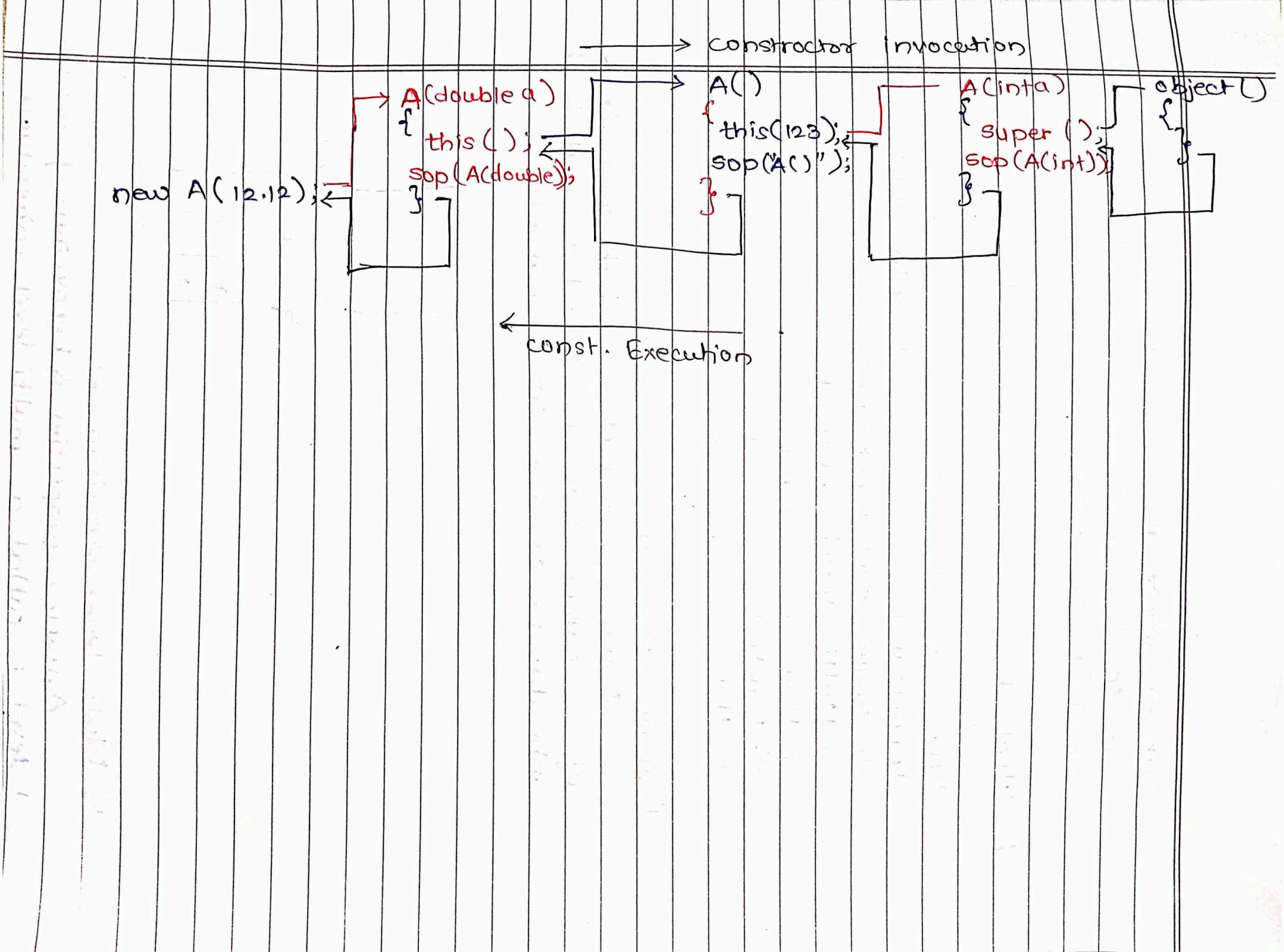
'this call statement' is used to call the constructor of own class within the class.

'this call statement' is not added by the compiler automatically.

We can use this call statement to call no argument constructor and parameterised constructors of its own class

'this call statement' always has to be written as first statement inside constructor.

```
class A
{
   A()
   {
     this (123);
     S.O.pln ("A()");
   }
  A(double a)
   {
    this ();
    S.O.pln ("A(double)");
   }
 A (int a)
   {
   S.O.pln ("A(int)");
   }
 ~~S.O.~~ psvm (string [] args)
   {
  A obj = new A (12.12)
   }
}
```


constructor invocation
new A(12.12);
A(double a)
{
this();
sop(A(double));
}
A()
{
this(123);
sop("A()");
}
A(int a)
{
super();
sop(A(int));
}
object()
{
}
const. Execution

* Can we write this call & super call statement inside single constructor. (Explain with an example).

* What are the types of inheritance

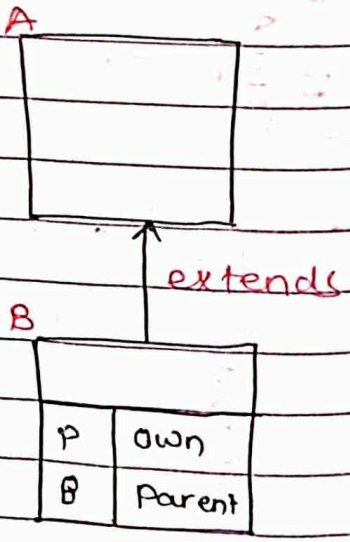
* Types of inheritance :-

Inheritance is classified into 05 types :-

1) Single level inheritance.
2) Multi level inheritance
3) Hierarchical inheritance
4) Multiple inheritance
5) Hybrid inheritance.

1) Single level inheritance

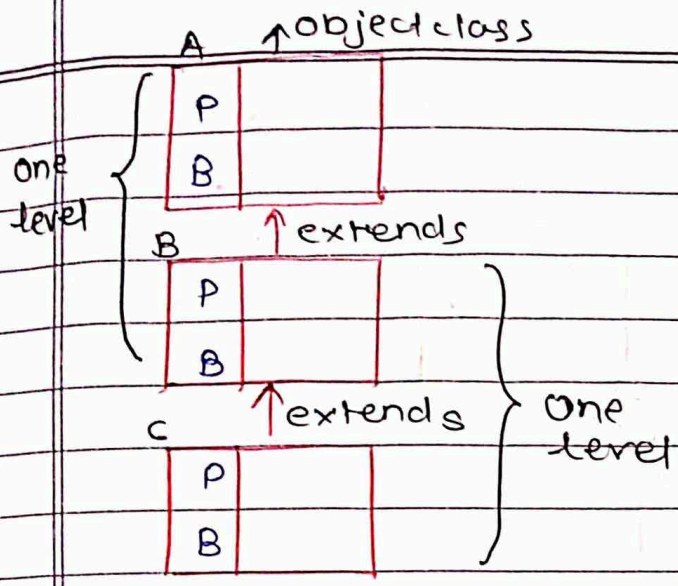
Aquaring properties & behaviour ~~from~~ of its parent class is called as single level inheritance. or one level inheritance.



Class B extends class A

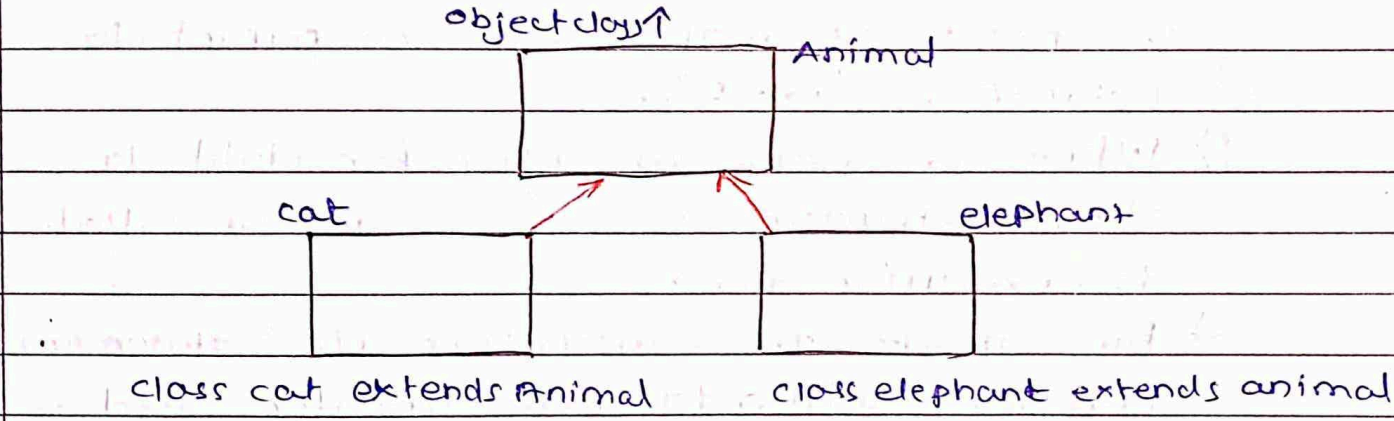
2) Multi level inheritance :-

Aquaring properties & behaviours of more than 1 level is called as multi level inheritance.



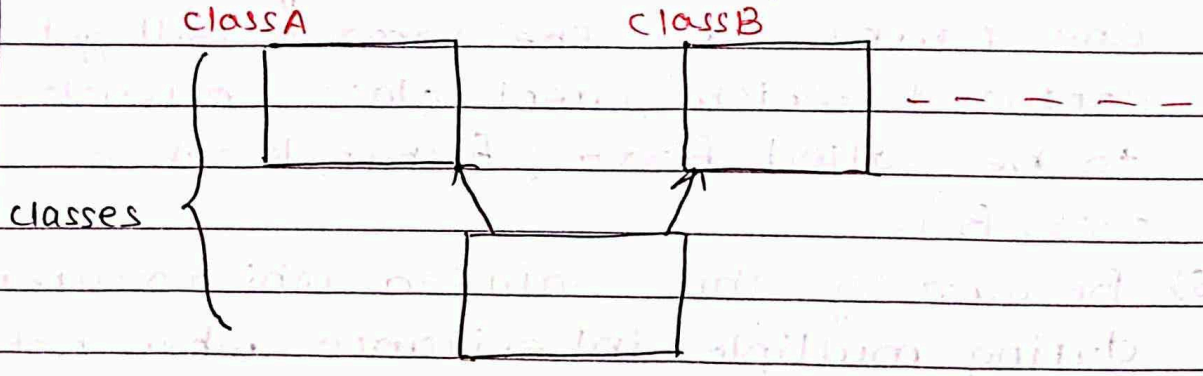
3) Heerarchical inheritance :-

A parent class consists of more than 01 child class is called as hierarchical inheritance

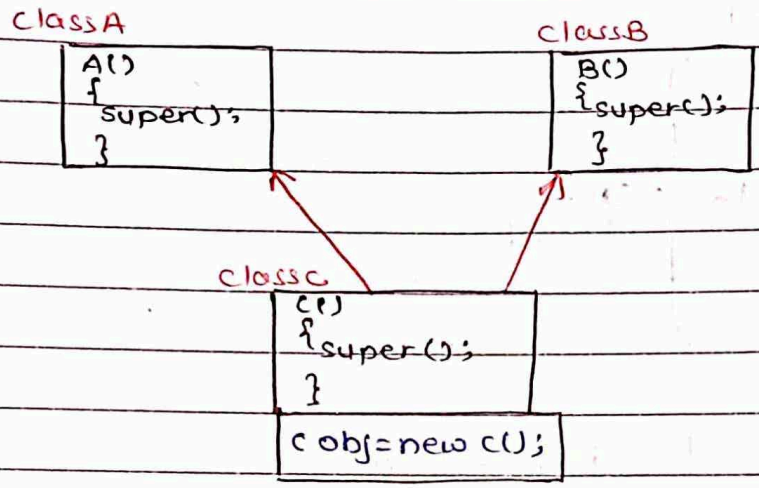


4) Multiple inheritance :-

Subclass consist of more than one super class is called as multiple inheritance.



* Diamond Problem :-

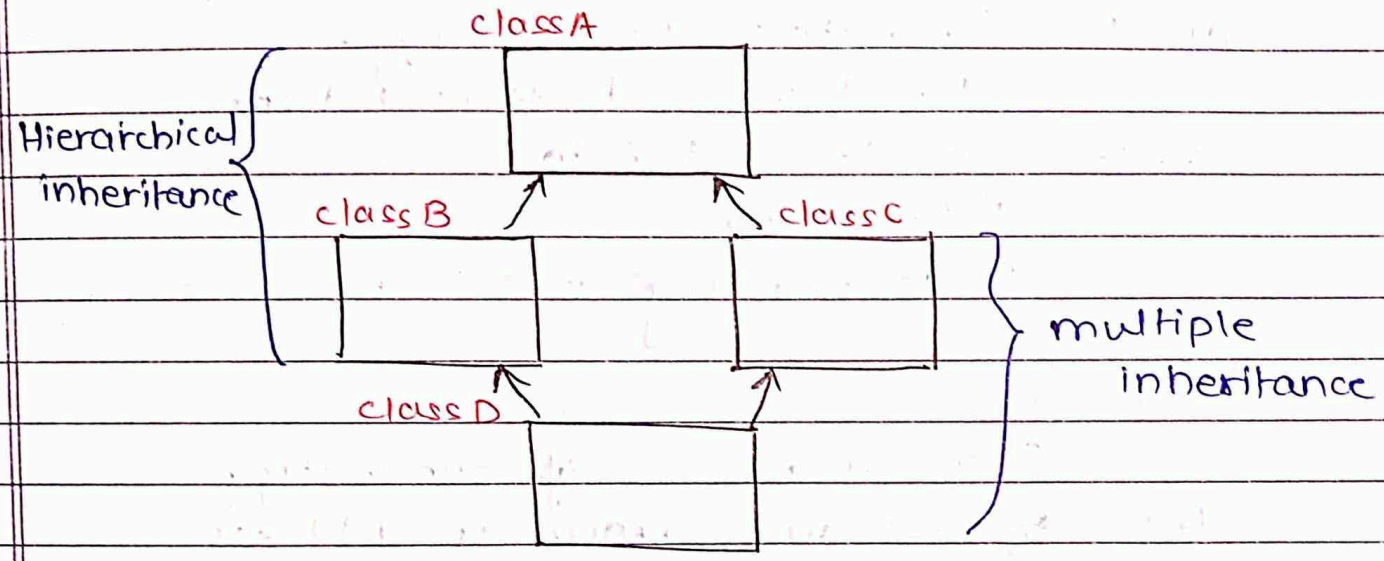


1) Here in the above statement the child class C consists of more than one parent class (class A & class B).
2) When we create an object for child class C, the constructor of class C will be called for execution first.
3) But inside the constructor first statement compiler will add automatically called as supercall statement.
4) The supercall statement is used to call constructor of parent class
5) But the sub class C consists of more than one parent class. The compiler will get confused which parent class constructor to be called first (Either class A or class B).
6) Because of this confusion which occures during multiple inheritance when both parent & child are of class type. Hence we cannot perform multiple inheritance in java (with the help of classes)

7) This problem which is occured due to multiple inheritance. it is also called as diamond problem.

5) Hybrid inheritance :-

It is a combination of Hierarchical and multiple inheritance.



Derivede type casting (Non premitive type casting) :-

It is a process of converting the referance from one class to another class. is called as derived type casting.

Derived type casting depends on non-premitive type casting.

Based on that derived type casting is classified into

1) Upcasting
2) downcasting.

1) Upcasting :-

It is a process of converting the referance or address from child class to parent class or sub class to super class is called as upcasting.

To perform upcasting we need to create an object for child class.

We need to create an object referance variable for parent class & pass the referance or address of its child class.

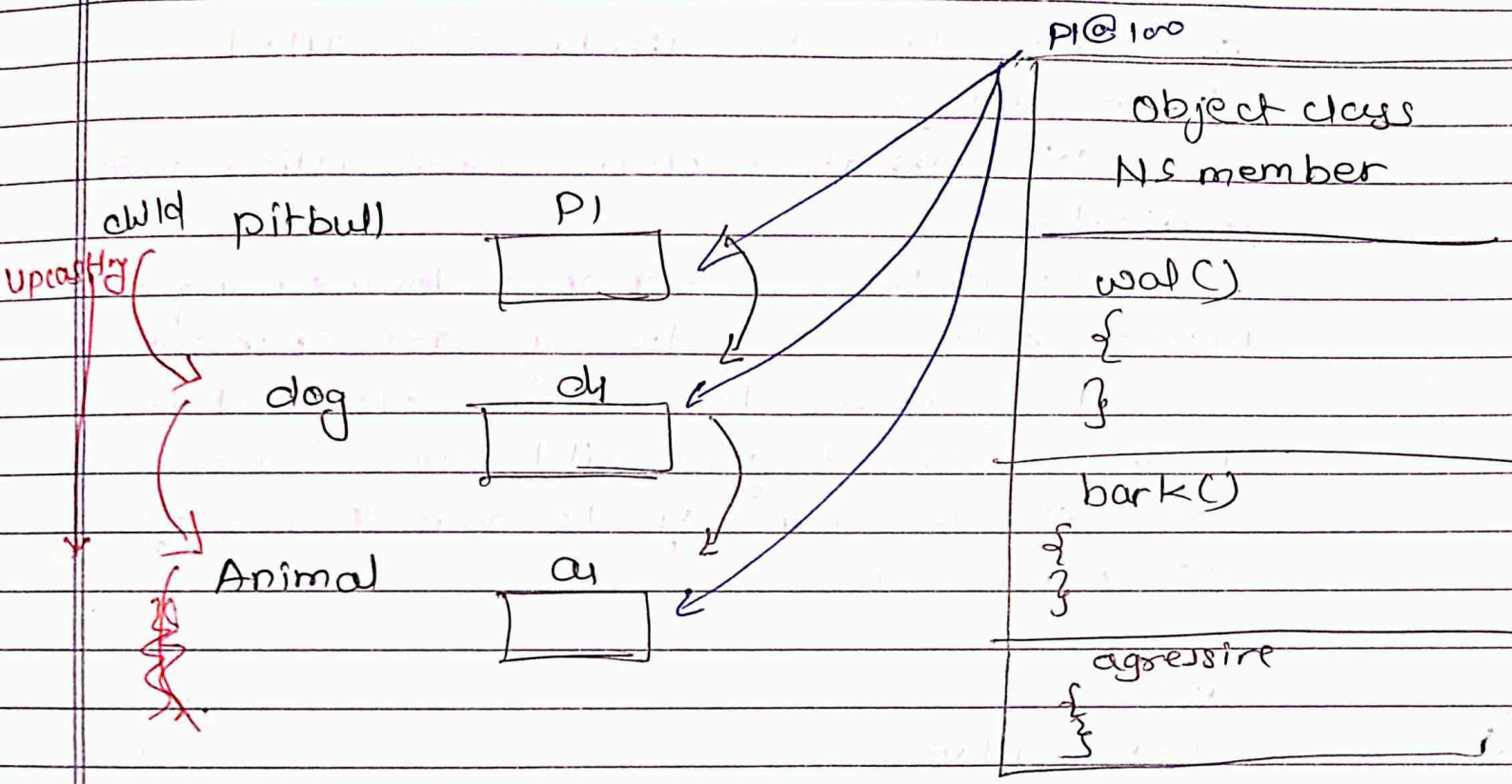
Syntax :-

Parent_class_name ref.var.name = ref. var. of child class

If we create an object for child class we can load the members of its own class & members of its parent class.

```
class Animal
{
  public void walk()
  {
  S.O.pln("All animals walk");
  }
}

Class Dog extends Animal
{
  public void bark()
  {
   S.O.pln("bow bow");
  }
}

class PitBull extends Dog
{
  public void Agressive()
  {
  S.O.pln("they are dangerous");
  }
}
```

```
class Planet
{
    psvm
    {
PitBull Pl = new Pitbull ();
Pl.agressive ();  // own class memb
Pl. bark ();  // Parent class member
Pl. walk ();  // Parent class member
S.O.pln ();

// upcasting from child to parent
// upcasting from PitBull to Dog
Dog d1 = Pl;
d1.bark();
d1. walk ();
d1. agressive';
S.O.pln ();

//upcasting from child to parent
//upcasting from from dog to animal
Animal a1 = d1;
a1. walk ();
a1. bark ();
a1. agreesive();
    }
}
```


p1@100
object class
N.S member
wal()
{
}
bark()
{
}
agressive
{
}
child pitbull
p1
Upcasting
dog
d1
Animal
a1

Down casting :-

It is a process of converting the referance from parent class to child class or super class to subclass is called as down casting.

We cannot perform downcasting auto-matically.

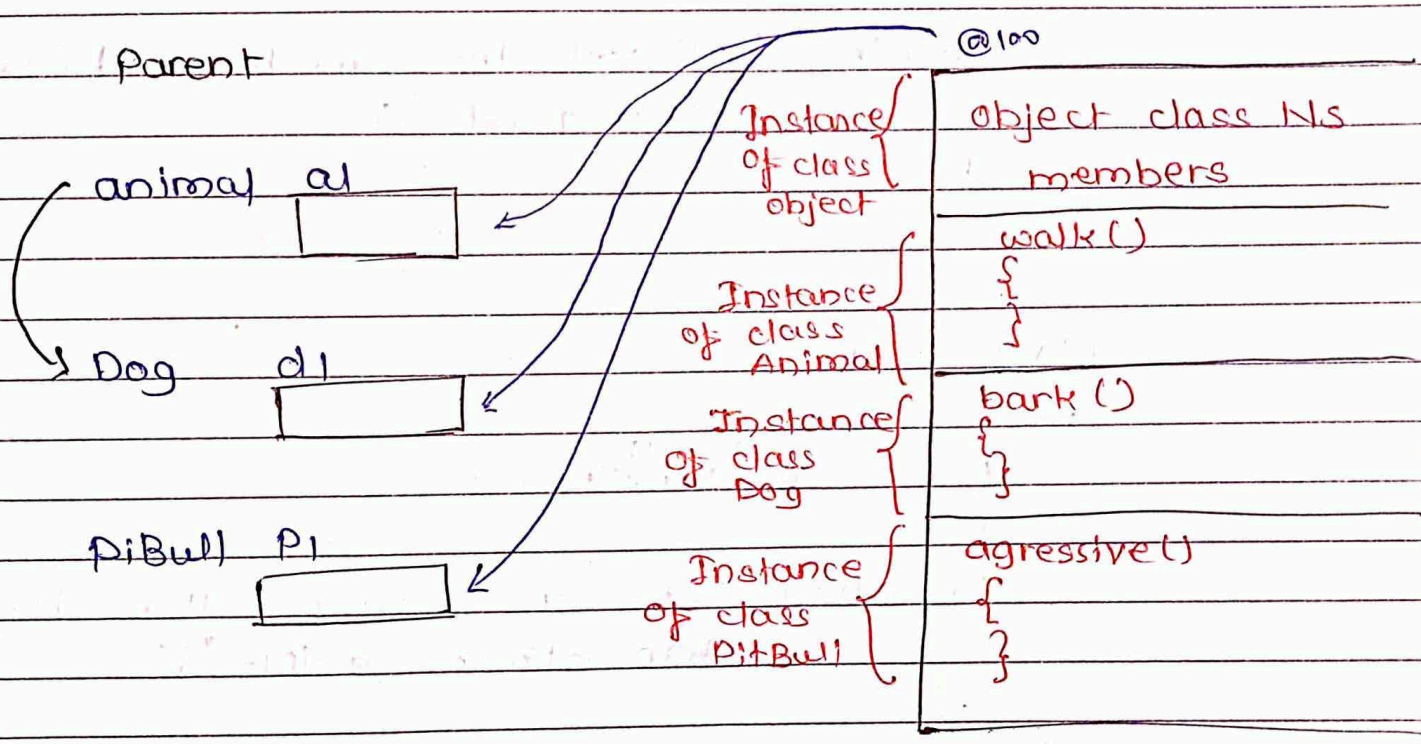
In order to perform downcasting we need to take the help of cast operator.

To perform downcasting we need to create a object for child class. (So we can load the members of its own class & of its parent class).

ex.

```
class Planet
{
    psvm (String[] args)
    {
        Parent class ref. = obj creation for child class
        Animal a1 = new PitBull();
        a1.walk();
        // a1.bark(); : CTE
        // a1.agressive(); : CTE
        s.o.pln();

        Dog d1 = (Dog)a1; //down casting from parent to child
        d1.bark;
        d1.walk;
        // d1.agressive; : CTE
```

```
PitBull p1 = (PitBull)d1; //downcasting from parent to child
        p1.bark;
        p1.walk;
        p1.agressive;
    }
}
```



NOTE :-

1) In order to perform upcasting & downcastin we need to create an object for child class.
2) To perform down casting cast operator is mandatory.
3) To perform upcasting cast operator is not mandatory (optional to use).

Create two technical examples for upcasting & down casting. using multilevel inheritance & hierarchical inheritance. & write & explain. (also make a document & post

→

```
class vehicle
{
    p v transport ()
    {
    s.o.pln ("all vehicles can be used for
                transport");
    }
}

class car
{
    p v ~~light vehicles~~ mileage ()
    {
    s.o.pln ("cars ~~are~~ giving better milage
                than other vehicles");
    }
}

class swift ~~affordable~~
{
    p v affordable ()
    {
    s.o.pln ("swift cars are affordable");
    }
}
```

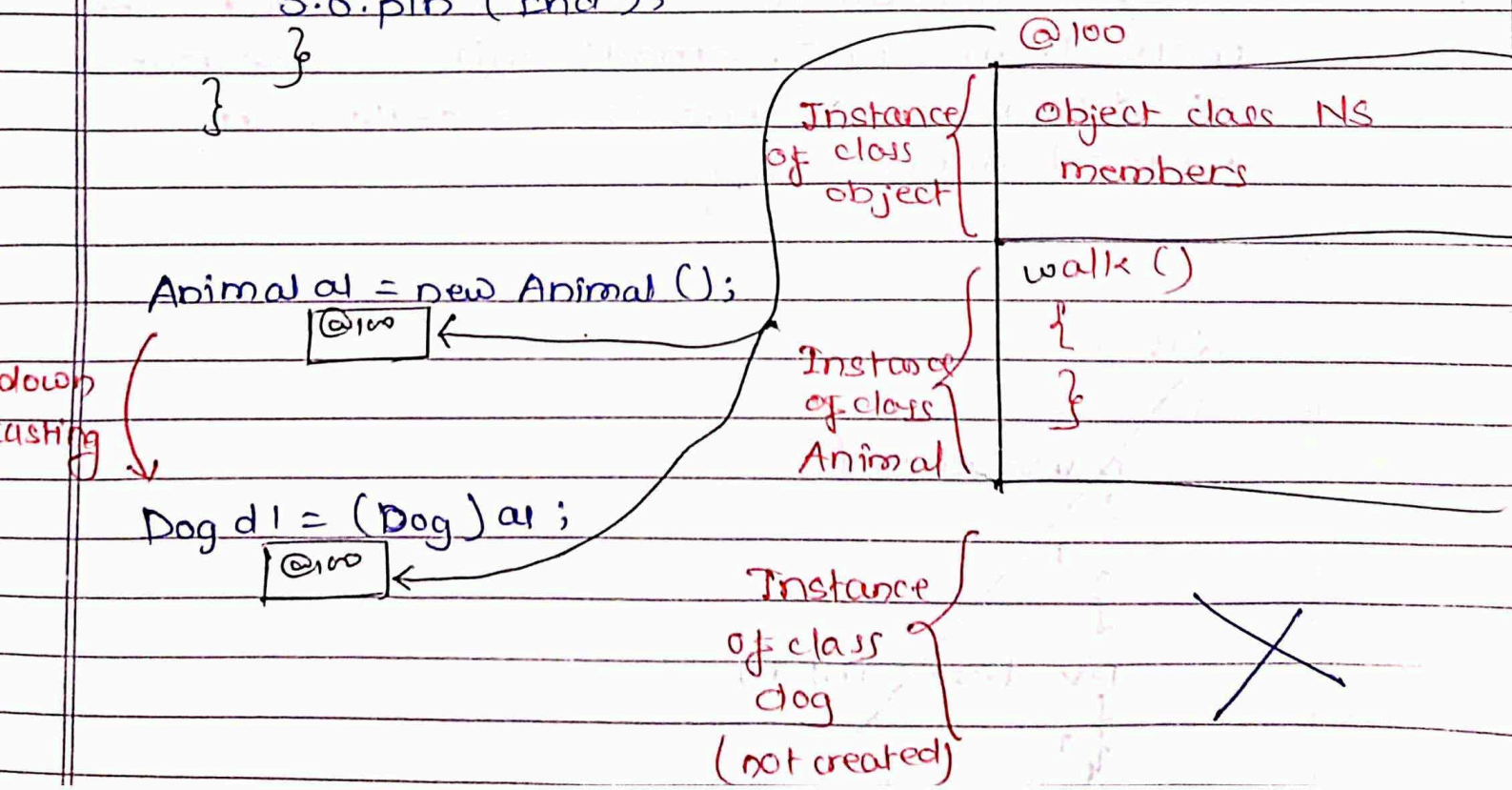
* Class cast exceptions :-

If we create an object for parent class & if we try to downcast it. To its child class. ~~since~~

Since instance of child class is not created inside the object we get an ~~ex~~ run time / execution time ~~exce~~ exception called as class V cast exception.

We get class cast exception during run time.

```
class Planet
{
    p~~s~~v m (string [] args)
    {
    s.o.pln ("start");
    Animal al = new Animal ();
    al.walk ();
    // Downcasting :- Animal to dog.
     Dog d1 = (dog) al ; //Exception: classcastexception
     d1. bark ();
     s.o.pln ("End");
    }
}
```



Polymorphism :-

An object performing or exibiting more than one form is called as polymorphism.

Polymorphism is classified into two types.

1) Compile time polymorphism.
2) Run time polymorphism.

Compile time polymorphism :-

Its a process of binding between method call statement and method implementation during compile time is called as compile time polymorphism.

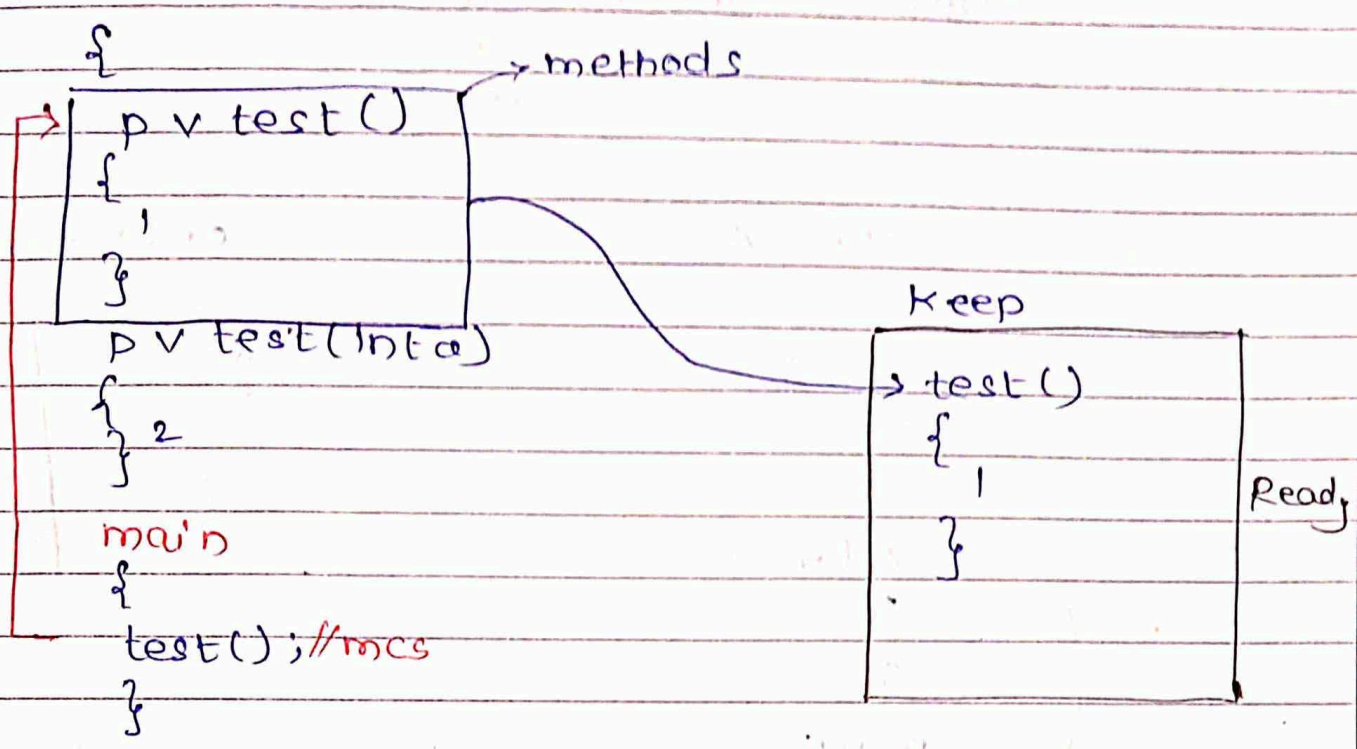
Compile time polymorphism is also called as early binding.

Compile time polymorphism is classified into two types.

i) Method overloading.
2) Constructor overloading.

Method overloading :-

~~In~~ the class consists of ~~there are~~ more than one methods with same name & different arguments is called as method overloading.



No members in ~~en~~ this entire java will ~~exec~~ get executed during compile time.

Enter → C → javac A.java → CTs

Enter → IR → java A → RTs

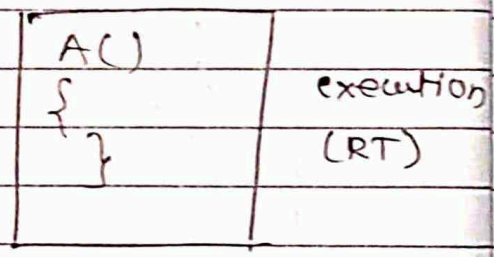
Constructor Overloading :-

~~In~~ A class ~~there~~ consists of ~~are~~ more than one custructor be present with same class name & different arguments is called as constructor overloading.

object :- object creation : RT { NS (member)
↓
execution

```
class A
{
   A()
   {
   }
   A(int a)
   {
   }
   main
   {
      A obj = new A();
   }
}
```

CT : keep it ready



CT : javac A.java

RT :- java A.

Run time polymorphism :- It is a process of binding method call statement & method implemetation during runtime. is called as run time polymo-rphism.

During runtime object gets created.
During runtime non static members gets loaded.

Runtime polymorphism is classified into two types :-

i) Derived type casting
ii) Method overriding.

i) Derived type casting :- Its a process of converting the referance from one class to another class is called as derived type casting.

Referance means ~~object~~ address of the object.

Derived type casting is classified into two types

1) Upcasting
2) Downcasting.

To perform derived type casting there should be 'is a' relationship between classes & object creation has to be done for child class

ii) Method overriding :-

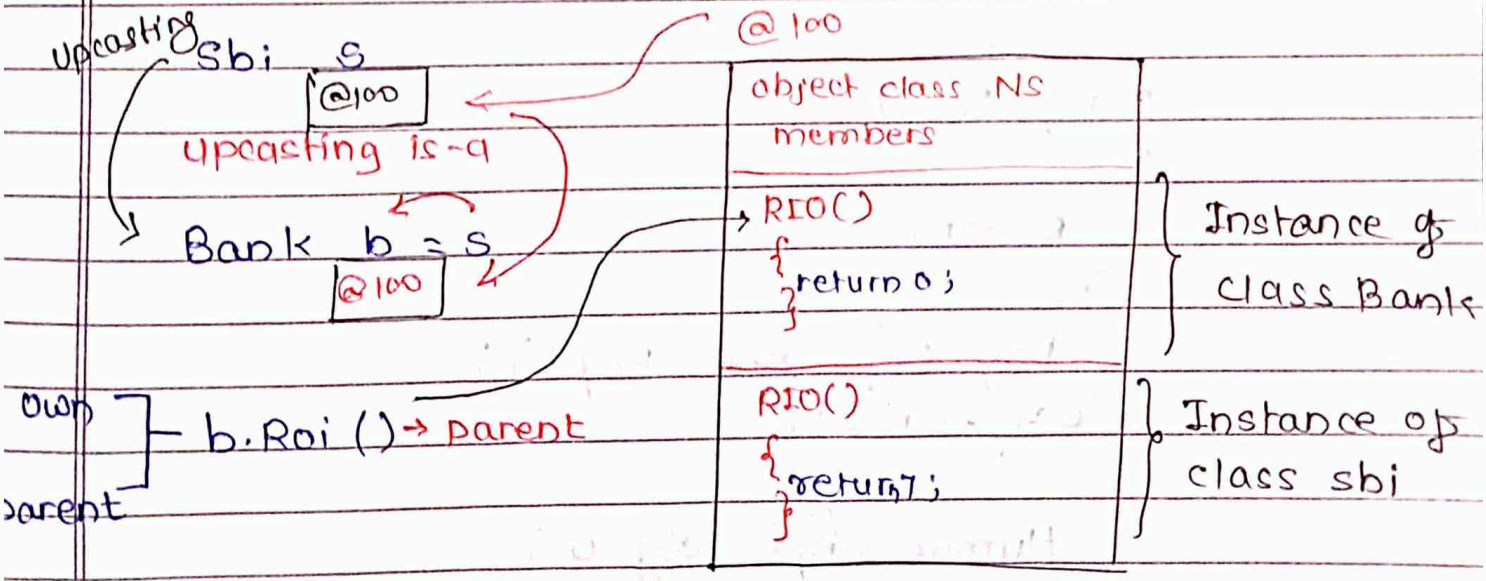
It is a process of overriding parent class method implementation by child class method implementation is called as method overriding.

Rules to perform method overriding :-

1) There should be is-a relationship (parent & child)
2) Both method header of parent class & child class has to be same. (modifier return_type method_name (formal args)).
3) To perform method overriding we need to create an object for child class.

```
class Bank
{
  public int Roi()
  {
   return 0;
  }
}

class SBI extends Bank
{
    public int Roi()
    {
    return 7;
    }
}
```

```
class city // has-a : execution
{
   public static void main
   {
   // object creation for child class
   // It will load members of its own & parent class
   Sbi s = new Sbi();
   s.o.pln (b.Roi());

   // verify is-a relationship i- upcasting
   // ref-variable for parent class = pass ref of child class
   Bank b = s;
   s.o.pln (b.Roi());
   }
}
```



Q. Create 2 real time example for method overriding & explain in detail & write in notebook (Today).

Abstraction :-

It is a process of hiding implementation is called as abstraction. (Implemetation means method body).

We can perform abstraction with the help of abstract keyword.

Abstract Methods :-

A non static method which is terminated by ~~semi~~ (;) semicolon & prefixed by abstract keyword. Such methods are called as abstract methods.

Abstract methods does not consists of any implementation inside its own class.

We can define / create more than one abstract methods inside the class block.

Method header of abstract methods :-

abstract modifier return-type methodname (formal args);

ex.

1) abstract public int Roi ();
2) abstract public int loan ();

Abstract class :-

A class prefixed with abstract keyword is called as abstract class.

If a class consists of atleast one abstract method then it is mandatory to make class as abstract.

Making class abstract it depends on the member of the class.

Syntax :-

```
abstract class abstract class name
{
   members
}
```

ex.

```
abstract class Bank
{
abstractpublic void ROI (); //abstract-method
abstract p v loan (); //abstract-method
}
```

NOTE :- We cannot use static method to perform abstraction.

We cannot prefix static method header with abstract keyword.

Concrete Methods :-

The method which consists of implementation is called concrete methods.

Concrete methods are classified into two types :-

1) Static concrete method.
2) Non static concrete method.

ex.

```
~~class~~ abstract class Bank
{
abstract public void Roi (); //abstract method
```

```
public static void Loan ()//concrete static method
{
s.o.pln ("bank provid Loan");
}
public void Fd(): // concrete Non static method
{
s.o.pln ("Bank provid Fd");
}
```

Q. Can we declare only concrete methods inside abstract class?

→ Yes. Inside abstract class we can define concrete methods also. (concrete means method implemetation)

ex.

```
~~class~~
abstract class Bank
 {
                //concrete methods.
  ~~abstract~~ public static void Loan ()
    {
    s.o.pln (" Bank provid Loan");
    }
   public void Fd()
    {
    s.o.pln (" Bank provid Fd");
    }
 }
```

Q Can we create an object for abstract class?

Q. Can we create an object for abstract class?

Q. Can we declare NSV, SV, ~~sta~~ SB, NSB & constructor (no args & parameterised) inside abstract class?

Q. Can we create an object referance variable for abstract class?

1)

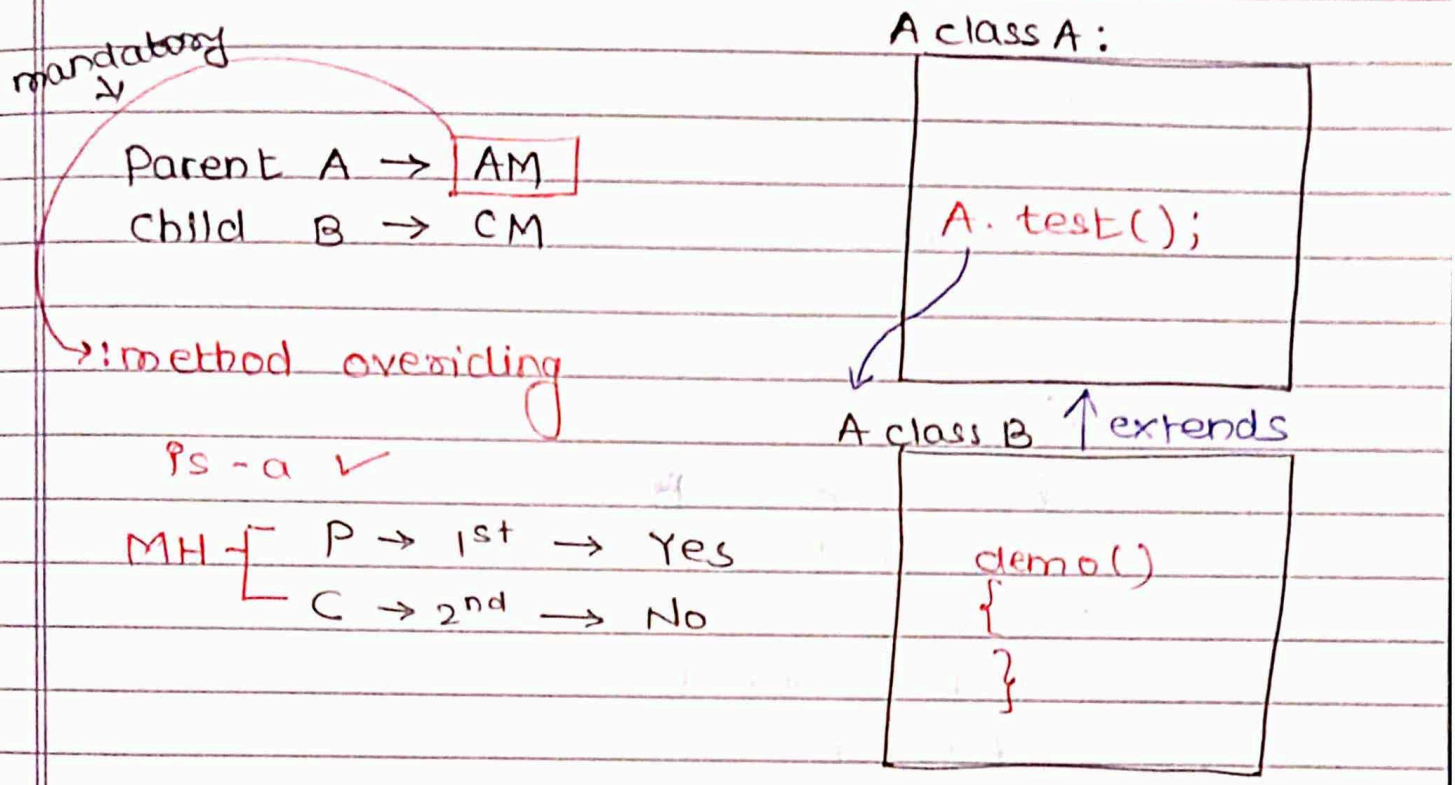
```
abstract class ABC
 {
   abstract public void Rain();
 }

class XYZ
 {
   p s v m
   {
     ABC m = new ABC(); // CTE
       m.Rain();
   }
 }
```

Explanation :-

We cannot create object for abstract class because class ABC has abstract method & in which the no implementation is there.

Inheritance using abstraction process:-

NOTE 1:- If a parent class consists of abstract method & if we are not overriding from its child class. Then it is mandatory to make child class as abstract.



NOTE 2:- If a parent class consists of abstract method & if we are overriding from its child class. Then it is not mandatory to make child class as abstract.

ex.

```
abstract class A
  {
    abstract public void test();
  }
```

```
class B extends A
{
    public void test()
    {
    S.O.pln ("test()");
    }
}

class demo
{
    psv m
    {
    A a = new B();
      a.test; //own

    B b = new B();
    Aa = b;
    // upcasting
    a.test;
    }
}
```

Paren A → AM

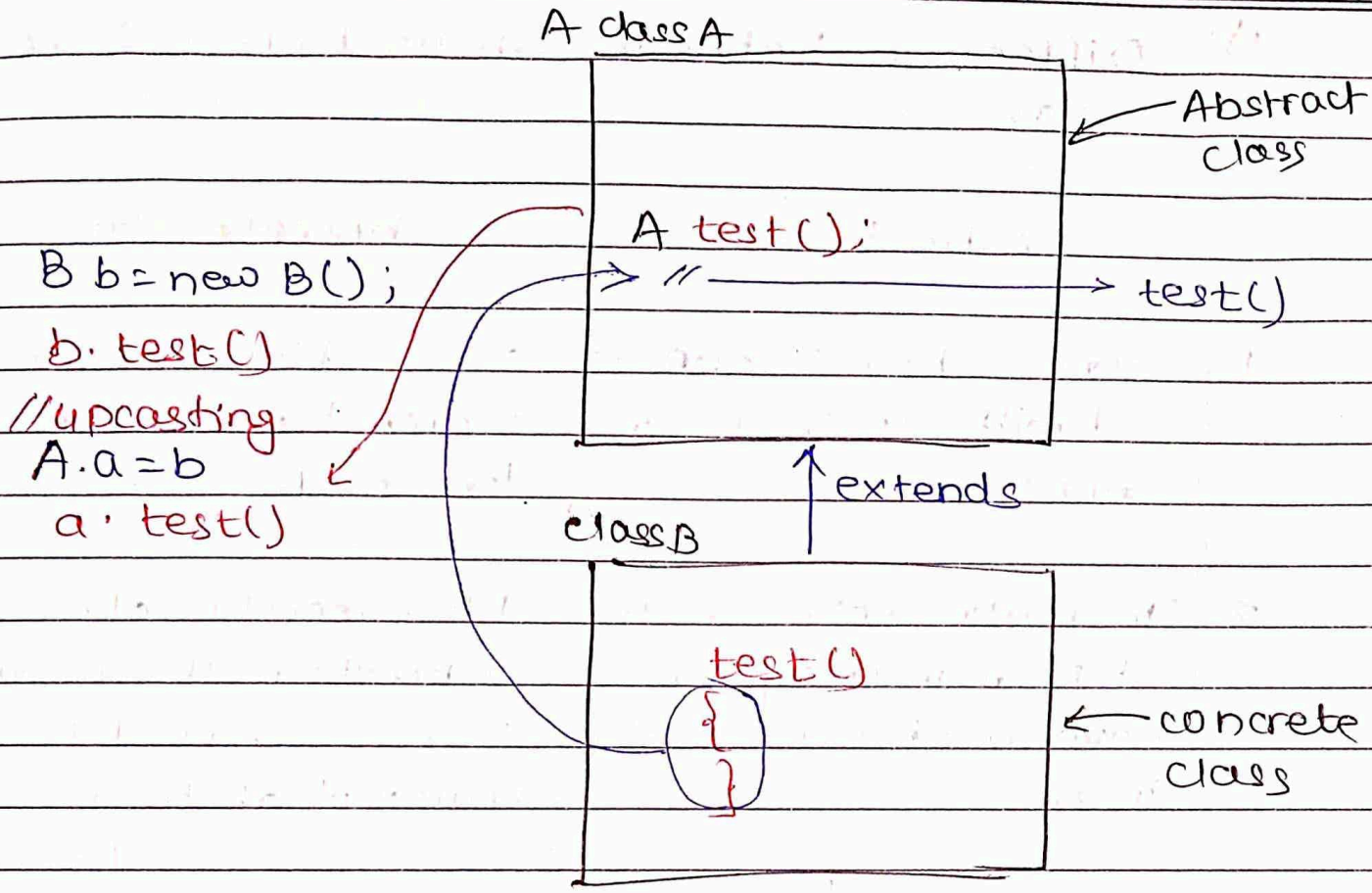
child B.

method overriding

is-a ✓ (extends)

MH [same] p → 1st → yes

c → 2nd yes



Concrete class :-

~~The class which~~

If a class consists of more than one concrete method & if object creation is possible such class are called as concrete class.

Q. WAD between abstract class & concrete class.

Q WAD between abstract method & concrete method.

Q. When it is mandatory to make class as abstract.

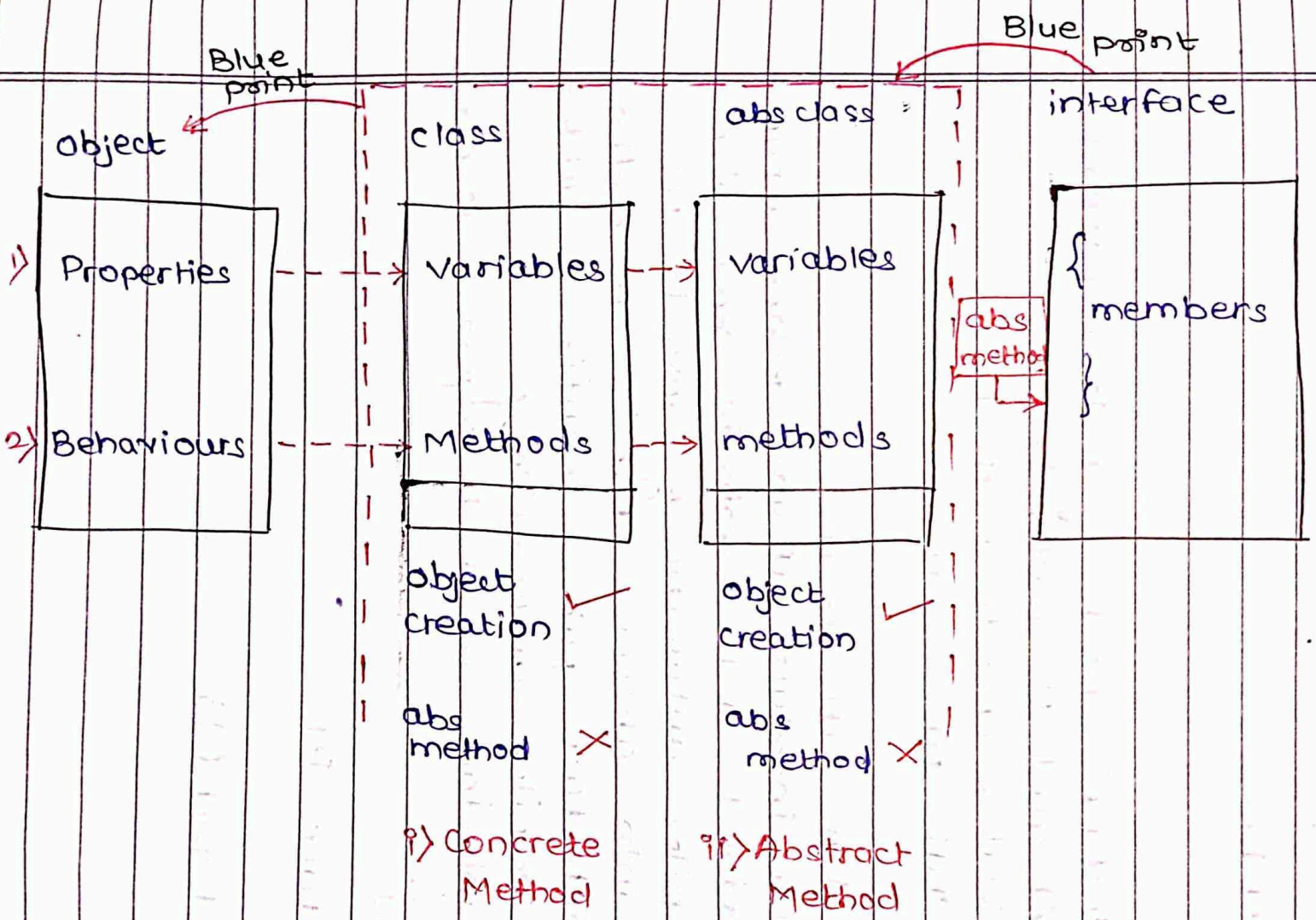
Q Create two example for method overriding process. using abstraction. with the help of two types of inheritance. i) multilevel inheritance ii) heirarchical in

NOTE :- All the questions has to be solved with real time examples & written in notes.

Interface :-

With the help of interface we can perform 100 % abstraction.

The internal nature of interface is abstraction process.



Interface is an blue print of the class

~~Structio~~

Structure of interface :-

```
interface interface_name
{
 members
}
```

Interface consists of following members.

1) Abstract NS method.
2) Static concrete method.
3) Static final variable.

Abstract Non static method :-

Inside interface if we declare a non static method and terminate semi-colon. The compiler automatically converts NS method into abstract public NS method.

SP⇒ In interface public modifier is added by the compiler automatically.

In interface abstract keyword is not mandatory.

ex.

```
interface PI
{
 void test();
}
to
```

```
interface P1
{
 abstract public void test();
}
```

Static Concrete Method :-

If we declare a static method without a modifier inside interface. The compiler automatically adds public modifier in interface.

ex.

```
interface P1
{
   static void test()
   {
    s.o.pln ("test()");
   }
}
```

to

```
interface P1
{
   public static void test()
   {
     s.o.pln ("test()");
   }
}
```

Note :- Inside interface we can declare more than one static method & abstract Ns method.

Static final Variable :-

In interface if we declare a static variable, the compiler converts static variable into final static variable.

Since final keyword is added by the compiler automatically initializing a static variable is mandatory for programmer in interface.

Final keyword :-

If we prefix ~~final~~ keyword variable with final keyword we cannot modify the value which is present inside the final variable. (because the value becomes constant & it is not changable).

```
interface P1
{
 static int a = 100;
}
```

to

```
interface P1
{
final static int a = 100;
}
```

Note :- If we declare a non static variable inside interface the compiler automatically converts NS variable into final static variable.

ex.

```
interface P1
{
 static int a=100;
}
```

to

```
interface P1
{
 final static int a=100;
}
```

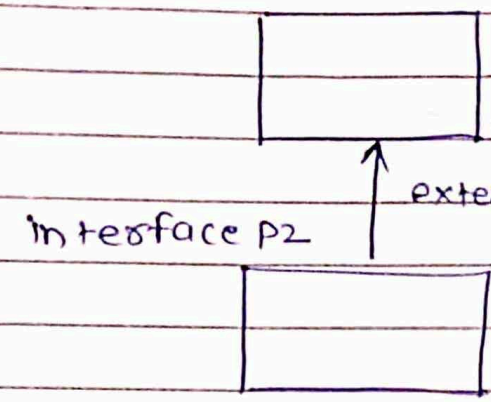
Note :- Interface does not support a non-static variable.

Inheritance using interface :-

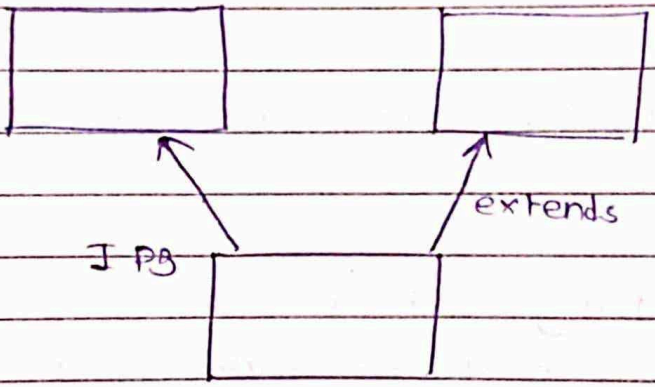
We can perform inheritance with the help of extends keyword in interface.

```
interface P1
{
}

interface P2 extends P1
{
}
```

interface P1



I P1 I P2




```
interface P1
{
}

interface P2
{
}

interface P3 extends P2, P1
{}
```

With the help of interface we can achieve multiple and hybrid inheritance. because interface does not support constructors.

* Concrete class :-

{

static { variables
methods
blocks

NS { variables
methods
blocks
constructors { 1) default
2) No ~~ag~~ arguments
3) ~~parameterise~~

Object Creation ✓
abstract method ✗

}

* Abstract class :-

{

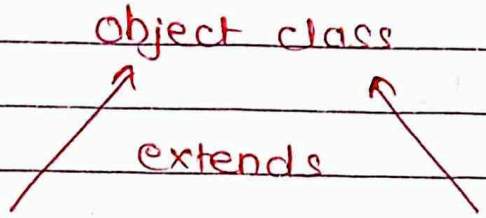
static { variables
methods
blocks

~~NS~~
NS { variables
methods
blocks
constructors { default
No args
parameterised

object creation ✗
abstract method ✓

}

Inheritance using interface & class :-

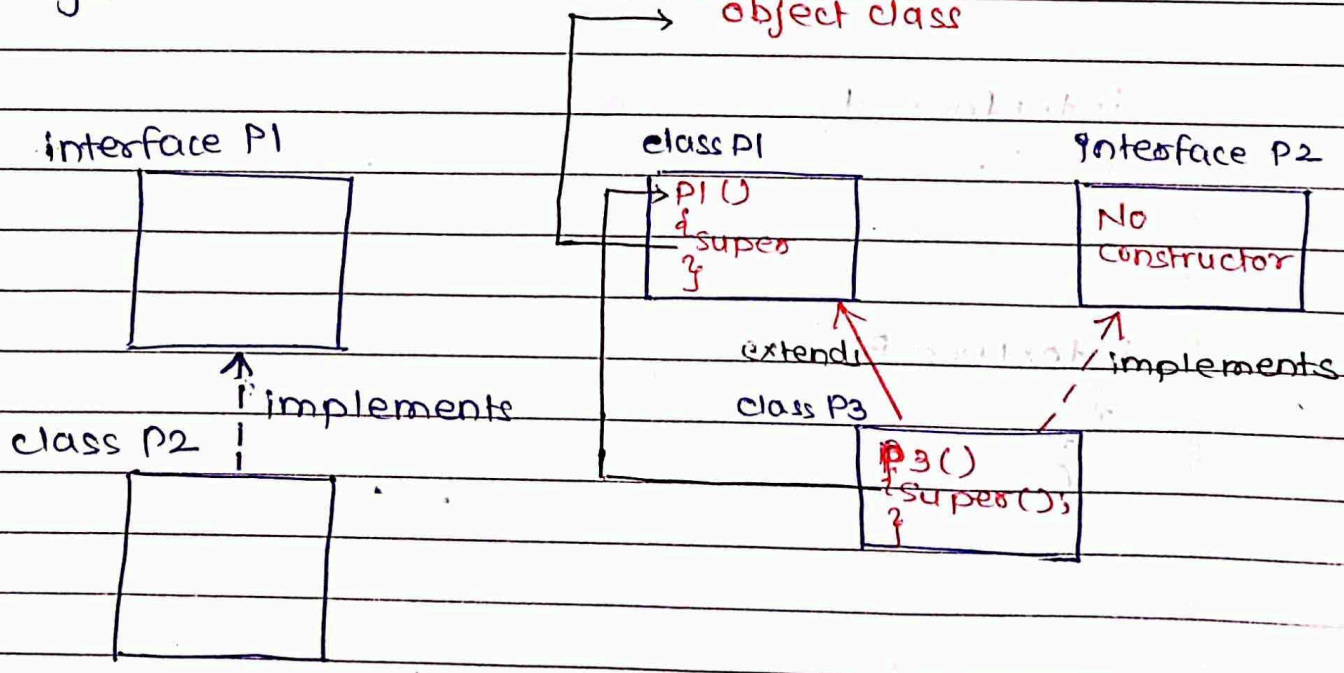


concrete class
{
}
- object creation ✓
- abstract method ✗

abstract class
{
}
- object creation ✗
- abstract method ✓

We can perform inheritance between interface & class with the help of implements keyword.

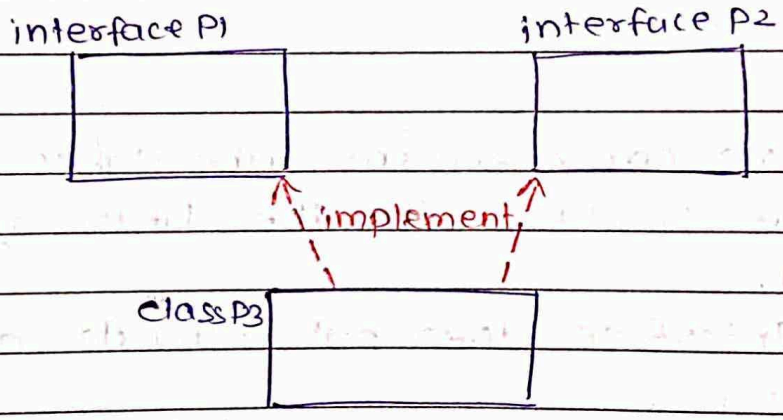
Interface does not extends or implements object class.



```
class P1
{
}
```

```
interface P2
{
}
```

```
class P3 extends P1 implements P2
{
}
```




```
interface P1
{
}
```

```
interface P2
{
}
```

```
class P3 implements P2, P1
{
}
```

Object class :-

Object class is an supermost parent class in java.

Object class is an predefined and inbuilt class in java.

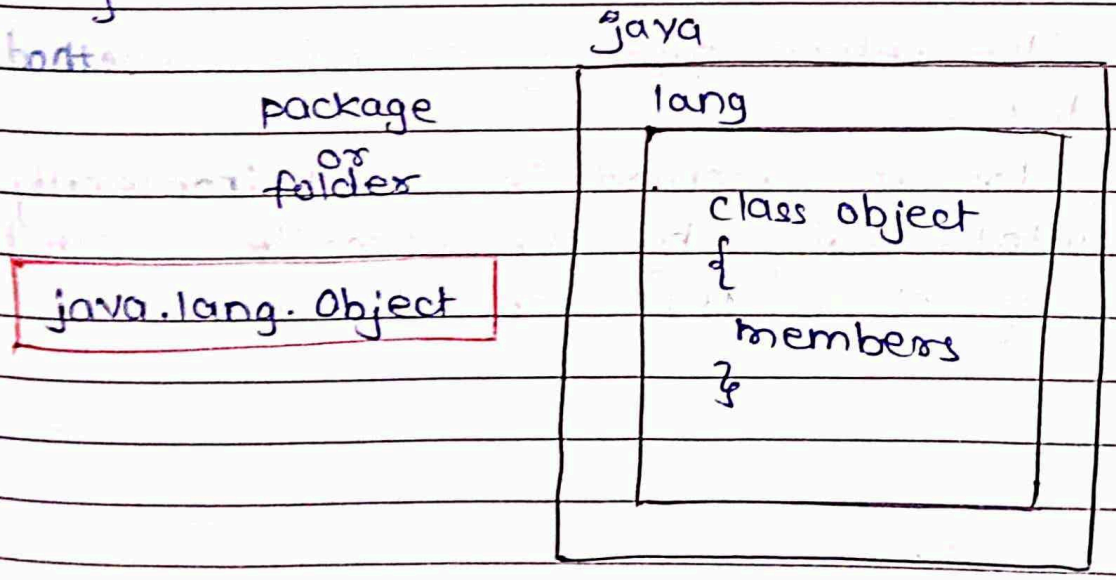
Object class is an concrete class in java.

We can create more than one object for object class.

Object class it is present inside [java.lang package (folder).]

fully qualified name of object class [java.lang.object]

```
class A
{
    public static void main
    {
        object var = new object();
        S.O.PLN (var); // java.lang.object@15db9742
    }
}
```



Object class ~~doe~~ consists of following methods :-

1) Public String tostring()
2) Public int hashcode()
3) public boolean equals (object obj)
4) Protected void finalize()
5) Protected object clone()
6) final public void wait()
7) public final void wait (long miliseconds)
8) public final void wait (long int)
9) public final void notifyAll()
10) public final void notify()
11) public final class getClass().

1) toString Method :- (toString())

toString method of an object class is used to convert the address of an object into string format.

toString method is an non static method which is present inside object class.

The return type of toString method is string.

tostring method is called implecitly or automatically by the compiler with the help of object referance variable.

Method header of tostring method:-

[public String toString()]

```
class P1
{
    p s v m
    {
        P1 obj = new P1();

        s.o.pln(obj); // P1@15db9742
        s.o.pln(obj.tostring); // P1@15db9742
    }
}
```

2) hashCode Method:- (hashCode())

hashCode is used to converts the address of an object into integer format.

hashCode method is an non static method which is present inside object class.

The return type of hashcode method is in int format.

hashCode method is not called by compiler automatically.

We can access hashcode method of object class with the help of object referance variable.

Method header of hashCode method:-

[public int hashCode()]

```
class P1
{
  psvm
  {
   P1 obj = new P1()

   s.o.pln (obj); // P1@15db9742
   s.o.pln (obj.hashCode()); 366712642
  }
}
```

3) Equals (object obj) :-

equals method is used to compare the address or refernce of two different object.

It is an non static method which is present inside object class.

The return type of equals method is a boolean condition (True/false).

```
class P1
{
  psvm
  {
   P1 obj = new P1();
   P1 obj1 = new P1();
   s.o.pln (obj); // P1@15db9742
   s.o.pln (obj1); // P1@6d06d69c
   s.o.pln (obj.equals(obj1)); //False
  }
}
```

```
class P1
{
    psvm
    {
        P1 obj = new P1();
        P1 obj1 = obj;

        S.o.pln (obj); // P1@15db9742
        S.o.pln (obj1); // P1@15db9742
        S.o.pln (obj.equals(obj1)); // True
    }
}
```

Q. Can we override toString() method of object class.

Yes. ~~we~~

So we can create our own address by overriding to string method of object class.

```
Class Emp
{
    int Eid;

    Emp (int n)
    {
        this.Eid = n;
    }

    public String toString()
    {
        return "WIPRO" + Eid;
    }
}
```

```
Class Cmp
{
  p s v m
  {
    Emp e1 = new Emp(123);
    Emp e2 = new Emp(124);

    s.o.pln (e1); // WIPRO123
    s.o.pln (e2); // WIPRO124
  }
}
```

Q. Create two examples for overriding tostring method of object class.

Q. Can we override hashcode method of object class.

Q1)

~~Class Student~~

i) {

```
Class vehicle
 {
   ~~public~~ int RegNo;

   Vehicle ( ~~int~~ int Rn)
   {
     this.RegNo = Rn;
   }

   public string toString ()
   {
   return "MH1300"+ RegNo;
   }
}
```

```
class SolapurRTO
{
   psvm
   {
      Vehicle V1 = new Vehicle (1100);
      Vehicle V2 = new Vehicle (1234);
      Vehicle V3 = new Vehicle (2244);

      S.O.pln (V1);
      S.O.pln (V2);
      S.O.pln (V3);
   }
}
```

ii)

```
Class Student
{
   int Sid;
   String Sname;

   Student (int id, String name)
   {
      this.Sid = id;
      this.Sname = name;
   }
   public String toString ()
   {
      return "The student details are :" + Sid +
                                          Sname;
   }
}
```

```
class College
{
    p s v m
    {
     Student s1= new (101, "Yash");
     Student s2= new (102, "Sonu");
     Student s3 =new (103, "Prasad");

     s.o.pln (s1); (s1);
     s.o.pln (s2); (s2);
     s.o.pln (s3); (s3);
    }
}
```

Q.2) Yes, we can override hashcode() method of object class from child class.

But while execution we have to call hashCode method from printing statement.

ex.

```
Class Employee
{
 int Scontact;

 Employee (int con)
 {
   this.Scontact = con;
 }

 public int hashCode()
 {
   return scontact;
 } }
```

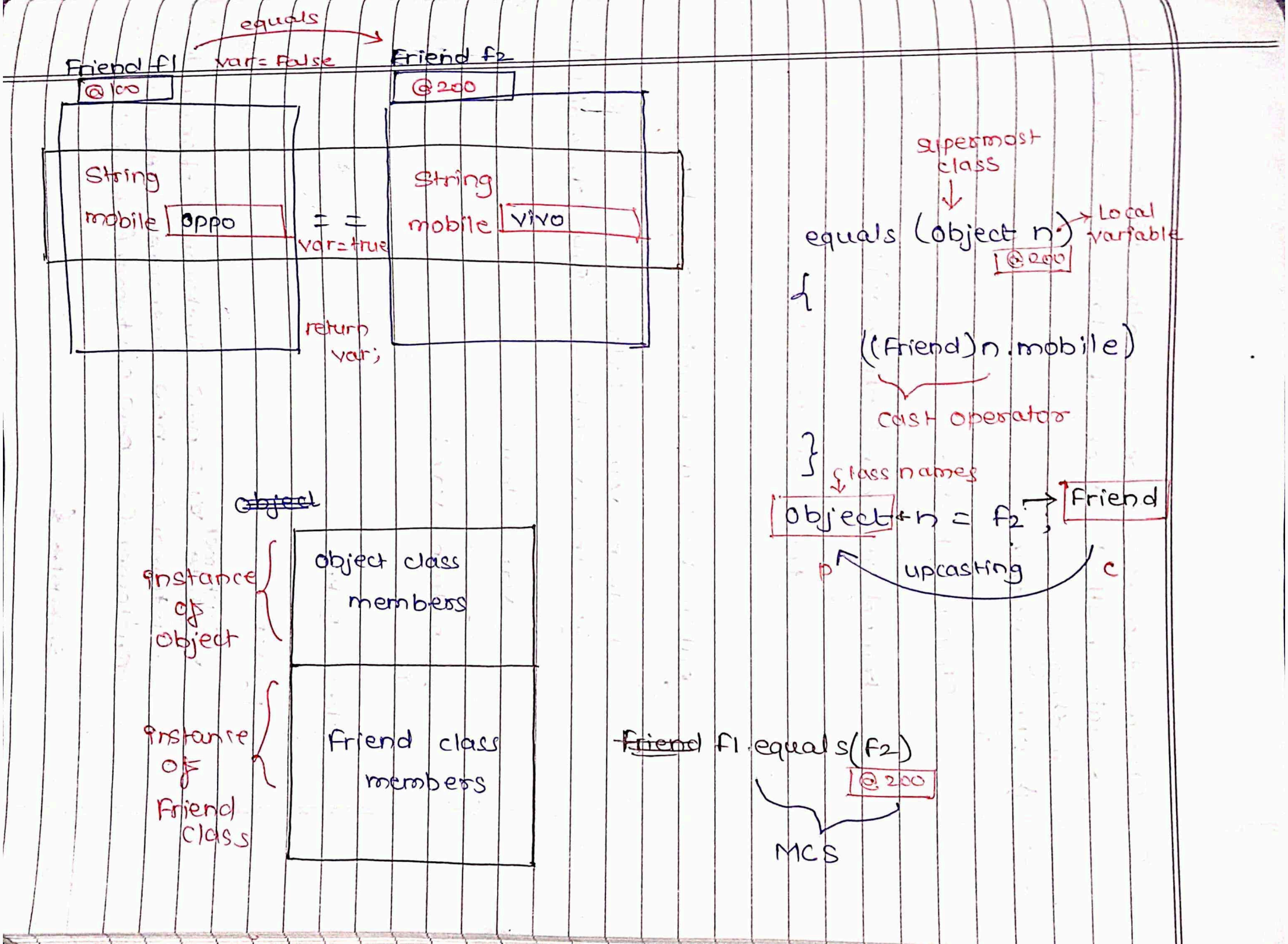
```
class Company
{
    psvm
    {
        Employee e1 = new Employee(9876543210);
        Employee e2 = new Employee(9876543211);
        Employee e3 = new Employee(9876543221);

        s.o.pln (e1.hashCode());
        s.o.pln (e2.hashCode());
        s.o.pln (e3.hashCode());
    }
}
```

* Can we override equals method of object class?

Yes.

By overriding equals method of object class we can compare the members of two diff^n object.

```
Class Friend
{
  String mobile;

  Friend (String m)
  {
   this.mobile = m;
  }

  public boolean equals (object n)
  {
     boolean var = false;
     IF (this.mobile ==((Friend)n).mobile)
     {
       var = true};
      }
    return var;
  }
```


equals
var = false
Friend f1
@100
Friend f2
@200
String
mobile
oppo
==
var = true
String
mobile
vivo
return
var;
supermost
class
equals (object n)
@200
Local
variable
{
((Friend)n.mobile)
cast operator
}
class names
object
n = f2;
Friend
p
upcasting
c
object
instance
of
object
object class
members
instance
of
Friend
class
Friend class
members
Friend f1.equals(f2)
@200
MCS

Exceptions :-

Exception is an abnormal situation that occures during the execution of a program

Whenever an exception occures the execution of a program ~~stop~~ is forcefully stopped.

Whenever an exception occures two things will takes place :-

1) It ~~throughs~~ throws an throwable type of object (address of a particular exception class).
2) Compiler will search for an exception handling mechanism (we need to create referance variable for particular exception class).
3) Exception is classified into types.
   a) Checked exception.
   b) Unchecked exception.

a) Checked exception :-

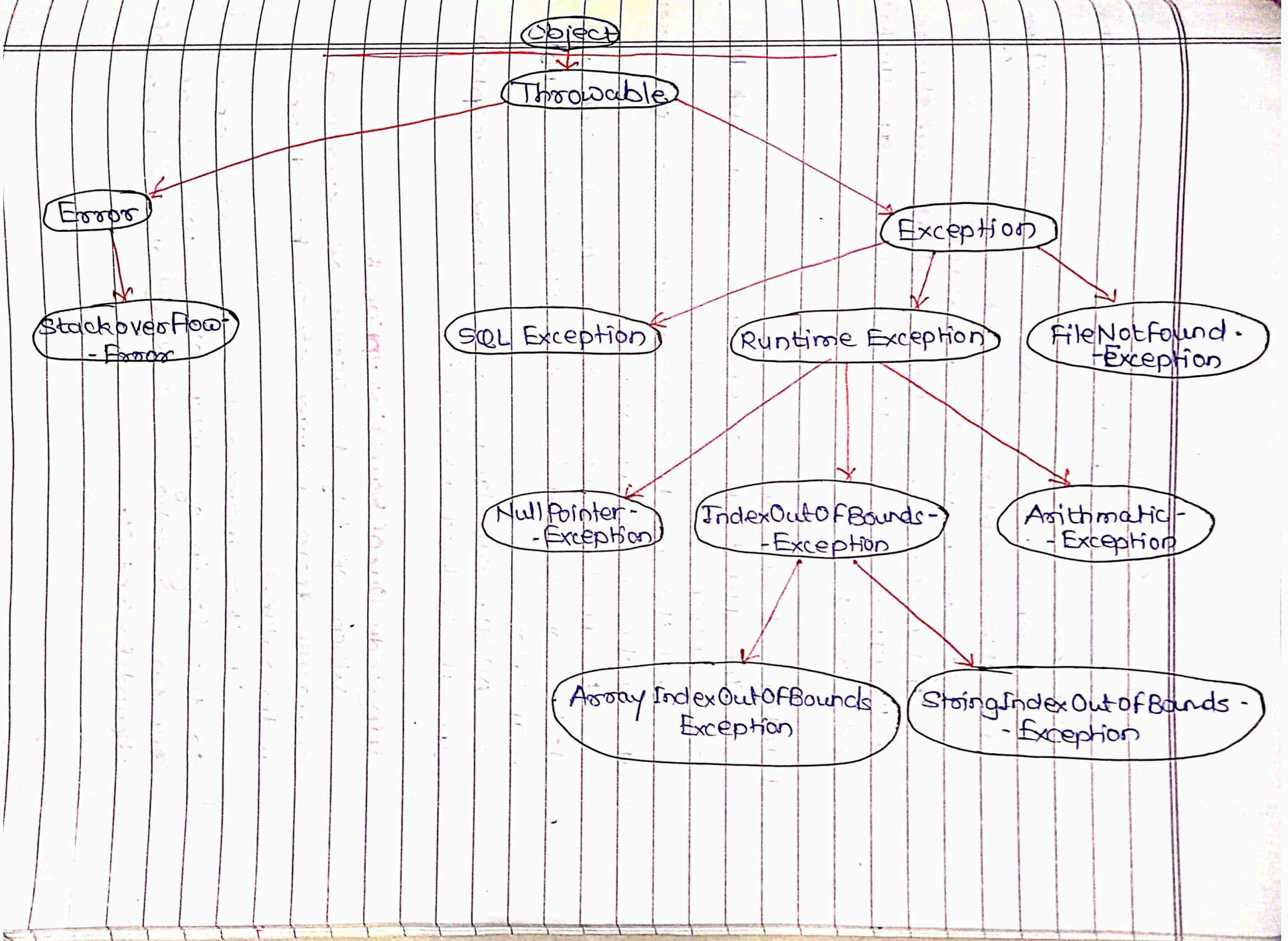
Checked exceptions are compiler aware exception.

For checked exception declaration and handling the exception is mandatory.

b) Unchecked exception :-

Unchecked exception are compiler unaware exception.

For unchecked exception declaration and handling the exception is not mandatory.


Object
Throwable
Error
Exception
StackOverFlow - Error
SQL Exception
Runtime Exception
FileNotFound - Exception
NullPointer - Exception
IndexOutOfBounds - Exception
Arithmatic - Exception
ArrayIndexOutOfBounds Exception
StringIndexOutOfBounds - Exception

Unchecked Exception :-

Unchecked exceptions are compiler unaware exception.

For unchecked exception declaration & handling the exception is not mandatory. (It's only for compiling the program.)

The runtime exception and its subclasses & the error class & its subclasses are unchecked exceptions.

Runtime Exception :-

The run time exceptions is a subclass of exception class.

All the runtime exceptions & its subclasses are unchecked exception.

For runtime exception & for its subclasses declaration & handling the exception is not mandatory

All the subclasses of runtime exception are unchecked exception.

StringIndexOutOfBoundException :-

When we try to access or extract a character from a given string based on index value.

If the index value is greater or equals to the length of given string we get an run time exception called as stringIndexOutOfBoundsException.

Length ():-

It is used to count the no. of elements which is present in a given string.

CharAt ():-

It is used to extract a character from a given string based on its index value.

ex.

```
Class A
{
  psvm (String[] args)
  {
    // index-value = 0
    String s1 = "Hello";
    System.s.o.pln (s1);
    s.o.pln (s1.length());

    s.o.pln (s1.charAt (1));
    s.o.pln (s1.charAt (5)); // exception
  }
}
```

ArithmaticException :-

~~It is used~~ If we try to devide anything by zero we get an exception called as ArithmaticException.

It is an uncheaked exception

```
Class B
{
  psvm (String[] args)
  {
```

```
        S.O.pln(1); //1
        int a=1;
        int b=0;
        int res = a/b; // ArithmaticException
        S.O.pln(2);
    }
}
```

Q. To handle an exception what is required?

→ Exception handling mechanism.

Exception handling mechanism:-

The mechanism which is used to handle an exception. is called as exception handling mechanism.

Exception handling mechanism consists of two block :-

1) try block.
2) catch block.

Syntax :-

```
try {
{
  statements;
}
catch(class_name var_name)
{
  statements;
}
```

try block :-

Any statement which is responsible for an exception has to be written inside try block.

try block is used to throw an throwable type object catch block.

catch block :-

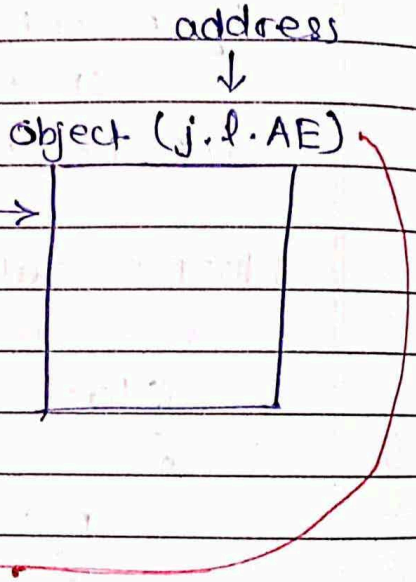
It is used to catch or store the throwable type object which is given by try block.

If the exception is caught then execution of the program will continue.

NOTE :- Catch block is used to handle an exception.

```
class B
{
  psvm
  {
    int a=1;
    int b=0;
    int res=
    // syntax :-
    try
    {
      int res = a/b; // object creation -> addres -> catch block
    }

    catch (ArithmeticException n)// ref var<add.
    {
      s.o.pln(n);
    }
```

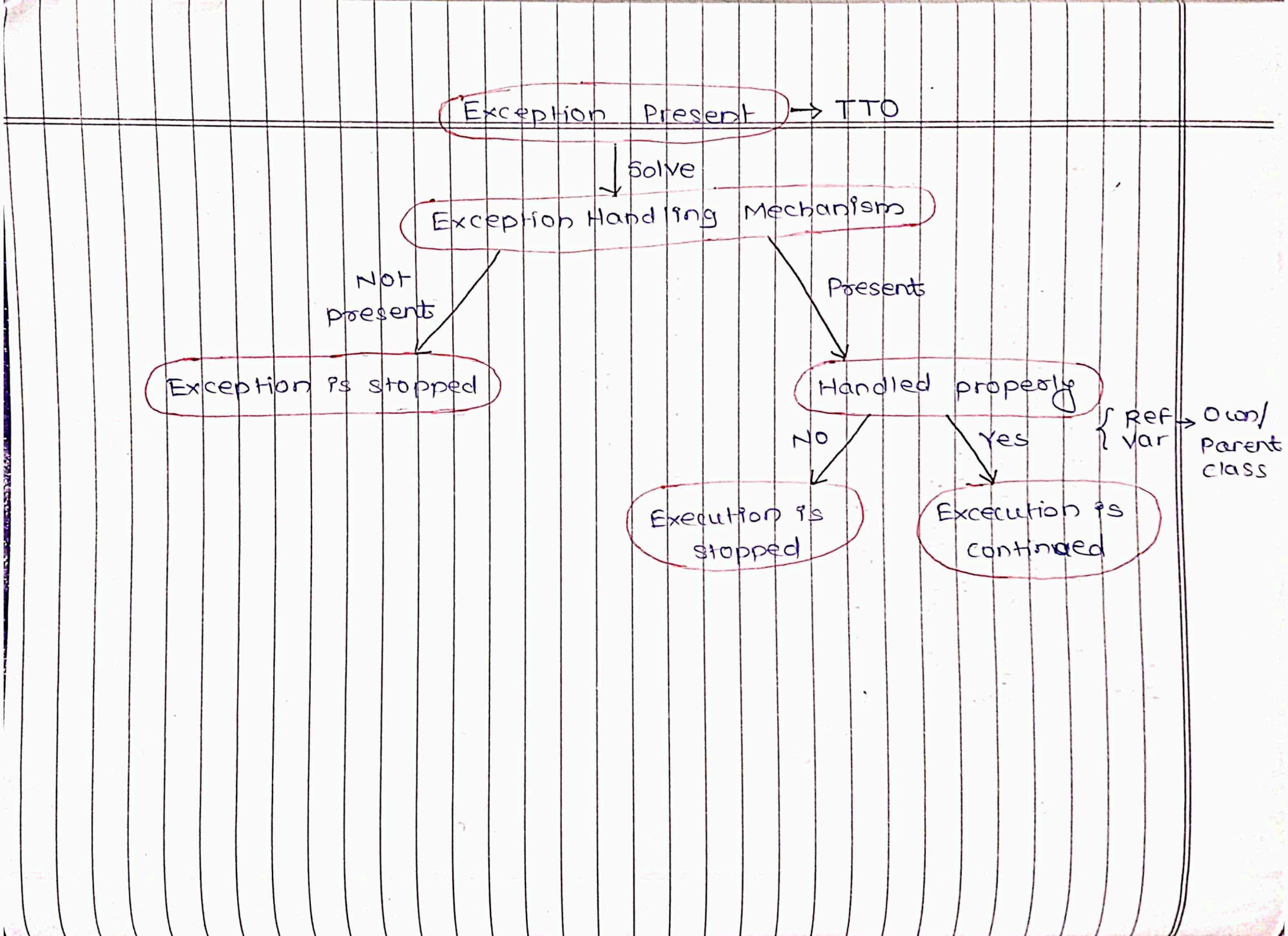
```
        S.O.pln (2);
        }
}

main
{
  sop (1);
   a = 1;
   b = 0;

try
{
   int res = a/b;
}

catch (ArithmeticExc. n)
{
   S.op (n);
}

S.O.ptn (2);
}
```



SP Upcasting :-

```
class B
{
  psvm
  {
   int a = 1;
   int b = 0;
```

```
try
{
  int res = a/b; // J.l.AE // child
  }
        // parent
catch (Exception n) // upcasting
{
  s.o.pln (n);
  }
  s.o.pln (2);
}
}
```


Exception Present → TTO
Solve
Exception Handling Mechanism
Not present
Present
Exception is stopped
Handled properly
Ref Var → Own/ Parent class
No
Yes
Execution is stopped
Execution is continued

Finally Block :-

If a program consists of a certain task to be executed even if the exception which is present inside the program is handled or not handled such task are written inside finally Block.

Syntax :-

```
finally
{
  statements
}
```

1) To use finally block, try & catch block is mandatory.
2) We cannot use finally block without try & catch block (atleast try block is required.).

Syntax :-

01)
```
try
{
  statements;
}
catch
{
  statements;
}
finally
{
  statements;
}
```

02)
```
try
{
  statements
}
finally
{
  statements
}
```

ex.

```
class B
{
 public static void main (string [] args)
  {
    s.o.pln ("Main Begin");
    int a=1;
    int b=0;

    try
    {
     int res = a/b;
    }
   catch (Exception n)  // Parent class referance variable
    {
     s.o.pln (n);
    }
    finally
    {
    s.o.pln ("Finally block execution");
    }
    s.o.pln ('Main End");
  }
}
```

	Exception occured	catch block Present	Handled	Finally block executed	Execution of prog. Cont.
1	✓	✓	✓	✓	✓
2	✓	✓	×	✓	×
3	✓	×	×	✓	×

* Try with multiple catch block :-

A try block consists of more than 01 catch block is called try with multiple catch.

Syntax :-

```
try
{
 statements;
}
catch (class_name var_name)
{
 Statements;
}
catch (class_name var_name)
{
 statements;
}
catch (class_name var_name)
{
 statements;
}
```

ex,

```
class B
{
 public static void main (String[] args)
 {
  S.o.pln ("main begin");
  int a = 1;
  int b = 0;
 try
 {
  int res = a/b;
 }
```

```
catch ( NullPointerException n)
{
  s.o.pln (n);
 }
 catch (Arithmetic Exception n)
 {
   s.o.pln(n);
 }
 s.o.pln( "Main End");
 }
}
```

* Nested try & catch :-

Syntax :-

```
try
{
  try
  {
   statements
  }
  catch (class_name var_name)
  {
   statements
  }
  statements
 }
catch (class_name var_name)
{
  statements;
 }
```

* Stack over-flow Error :-

1) Stack Overflow Error is an unchecked excepti
2) Stack Overflow Error is a sub class of error class.
3) For stack overflow Error, declaration and handl the exception is not mandatory (for compilin the program).

```
class B
{
  static int a = 1;
  p s v m (string [] args)
  {
    s.o.pln (a++);
    main (null); //mes.
  }
}

// string : NPTD
// []     : Array
// args   : Variable-name
```

Q. Can we handle stack overflow error with the help of EHM? Explain.

Q. What is the memory size of stack?

Q: Can we declare / write try & catch block inside finally block.

→
```
class Add
{
    public static void main(string [] args)
    {
        S.o.pln ("start"); //start
        int m = 'a';
        int n = 0;

        try
        {
        }
        finally //we cannot execute finally block without try block.
        {
         try
         {
           int res = m/n;
         }
         catch (Exceptions)
         {
           S.o.pln ("Catch block"); //catch block
         }
        S.o.pln ("End");  //End
    }
}
```

Yes, we can write / declare try & catch block inside finally block. But one more try block is needed outside of the finally block for execution of finally block.

Q. Can we handle exception inside finally block

→

```
class Add
{
  psvm
  {
    s.o.pln ("Start");
    int m = 'a';
    int n = 0;
   // to declare try & catch block inside finally
    block one try block out side finally block
    is needed.
    try
    {
    }
    finally
    {
      try
      {
        int res = m/n;
      }
      catch (Exception s)
      {
        s.o.pln ("Catch block");
      }
    }
    s.o.pln ("End");
  }
}
```

Yes, with the help of try & catch block we handle exception inside finally block.

Q. Can we handle stack overflow error with the help of EHM? Explain.

→

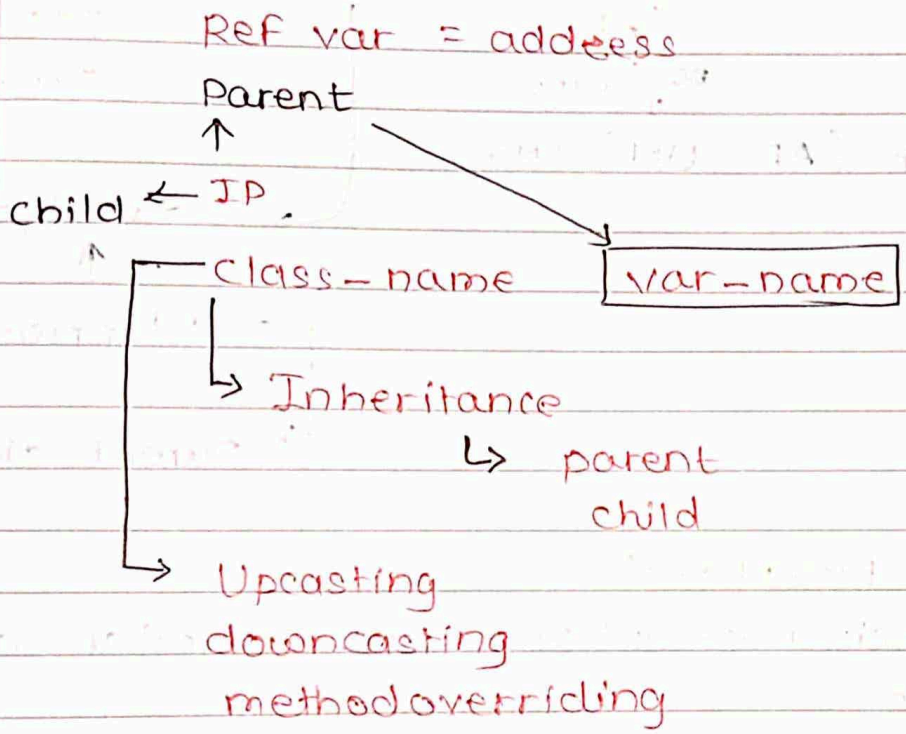
```
class start
{
  static int a=1;
  psvm
  {
    try
    {
      s.o.pln (a++);
      main (null);
    }
    catch ( Error m)
    {
      s.o.pln (m);
    }
  }
}
```

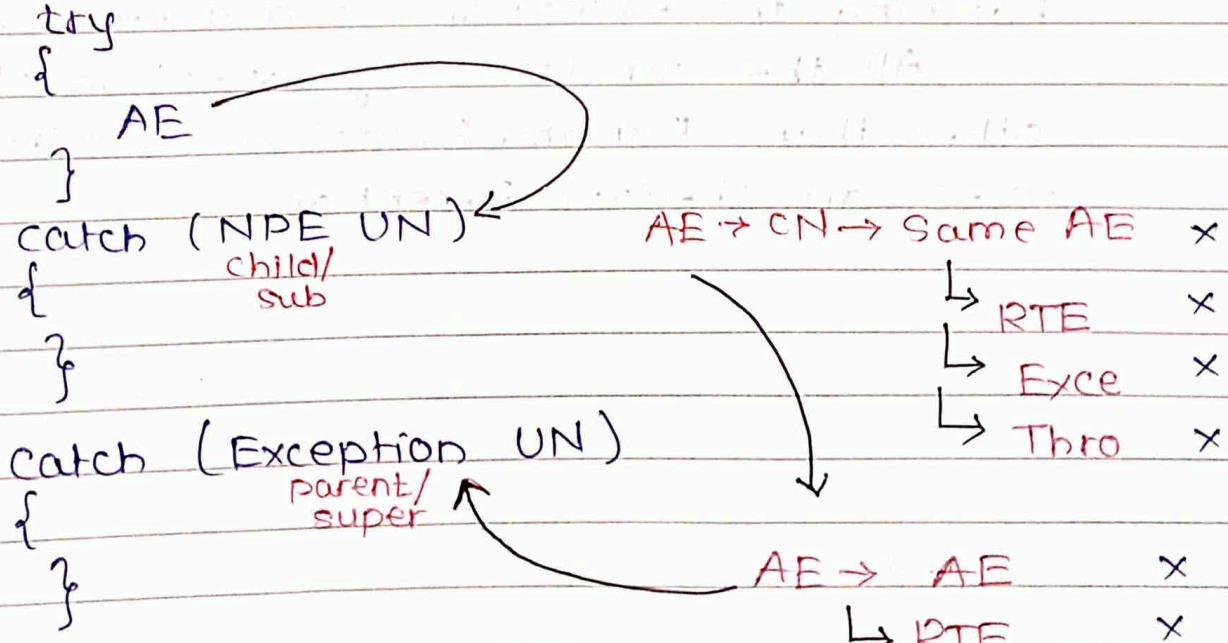
Q. What is the size of memory size of stack?

→ The memory stack of stack is by default ~~512 KB~~. 320k for 32 bit & 1024k for 64 bit.

Q. WAP for nested catch

NOTES- In try with multiple catch block the referance variable inside the catch block always has to be written in child class to parent class ordered (Upcasting).





upcasting

```
try
{
   AE
}
P  catch (Exception UN)
{                          AE
}                      DC / same
C  catch (AE UN) → NO
{
}
```

Down casting

AE → same AE ✗
RTE ✗
Exce ✓

Sametype
↑
1st priority
Parent class

Checked Exceptions:-

Checked exceptions are compiler aware exception.

For checked exception declaration and handling the exception is mandatory.

All the subclasses of exception class other than RuntimeException and its subcla-sses are checked Exception.

FileNotFoundException :-

```
import java.io.*;
class P4
{
    psvm(string args) throws FileNotFoundExcep
                                -tion
    {
    FileOutputStream obj = new FileOutputStream("C:
                        \\basicprog\\abcd.txt");

    }
}
```

```
// Path : Address : String format
// Inbuilt class :- Java package
// * : : All the classes for io package.

// must be caught : we need to use HSM.
// declare to be thrown : aware : keyword throws
// FileNotFoundException : compiler : line 6/st.
```

'throws' keyword :-

'throws' keyword is used to declare an exception.

We can usee 'throws' keyword with the help of method header.

'throws' keyword is used to ignore an exception.

With the help of 'throws' keyword we can declare more than one exception.

Syntax :-

modifier return-type method-name (FA) throws Exce1, Exce2, ... Excen

Q Can we handle FileNotFound exception with the help of exception handling mechanism ex. with program.

→

```
import java.io.*;
class star
{
  psvm (string args[])
  {
    try
    {
      FileOutputstream O = new Fileoutputstre
                           ("D:\\newFold\\newabc
    }
    catch ( FileNotFoundEx  f)
    {
      s.o.pln ("catch block");
    }
  }
}
```

Yes, we can handle.

'throw' keyword :-

'throw' is an keyword in java.

With the help of throw keyword we can create or throw an exception by creating an object for a particular exception class.

We can use 'throw' keyword inside the method.

With the help of 'throw' keyword we can throw only one exception at a time.

ex.

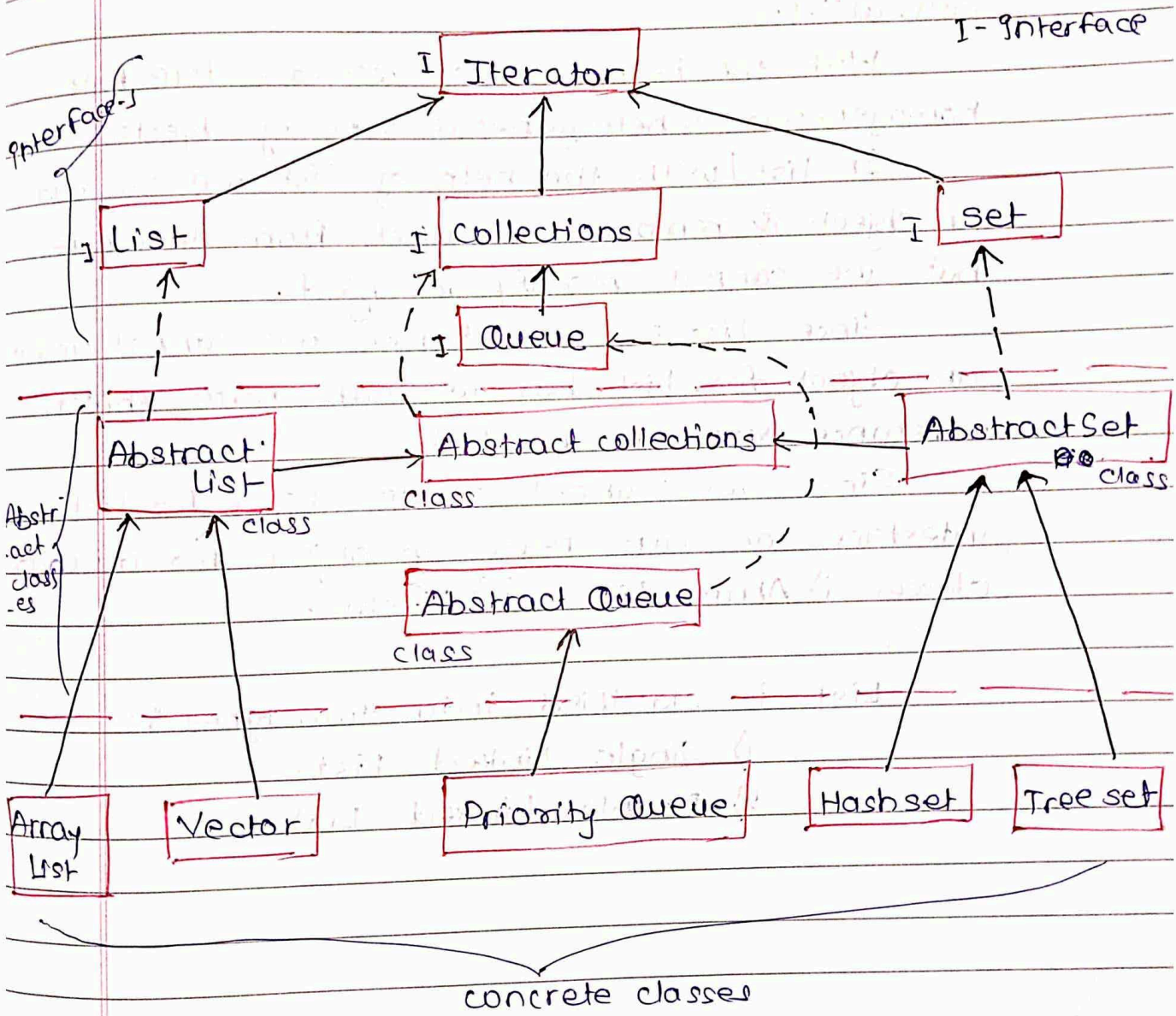
```
class P5
{
  public static void vote (int age)
  {
   s.o.pln ("vote and go home");
   If (age >= 18)
   {
    s.o.pln ("vote and go home");
   }
  else // Create exception means create an object
                                  for exception classes
  {
   throw new ArithmeticException ("be in home");
  }
  }
  public static void main (String[] args)
  {
   s.o.pln ("welcome");
   vote (18);
   s.o.pln ("bye bye");
  }
}
```

* Collection Framework :-

Collection framework is a structure / paradiagram to maintain a group of objects.

We can store more homogeneous & hetero-geneous

Collection framework classes :-



All the inbuilt collection framework classes are present inside java.util package.

List :-

List is an interface in java.

List interface is a child of iterator interface.

List interface is present inside java.util package.

The fully qualified name of List interface is java.util.List.

With the help of list we can store both homogeneous & hetrogeneous type of objects.

In list / with the help of list we can add an object & remove an object from anywhere but we cannot modify an object.

Since list is an interface we cannot create an object for List, but we can create object referance variable for List.

Since we cannot create object for List interface we can create an object for its child classes i) Array List ii) Vector.

List is classified into two types :-

1) Single Linked List.
2) Double Linked List.

Array List :-

Array list is a concrete class in java.
Array list is present inside java.util package.
The fully qualified name of array list is java.util.ArrayList.

With the help of array list we can add remove and read an object.

With the help of array list we can store both homogeneous & hetrogeneous type of object.

We can create an object for arraylist.

In order to create an object for array list we need to import array list from util package.

For array list we can create more than one object.

```
import java.util.ArrayList
class P1
{
  P.S.V.M
  {
    ArrayList obj = new ArrayList();
  }
}
```

Vector :-

Vector is a concrete class in java.

Vector is present inside java.util packa-ge.

The fully qualified name of vector is java.util.Vector.

With the help of vector we can ~~store~~ add remove & read an object.

With the help of a vector we can store both homogeneous & hetrogeneous type of object.

We can create an object for vector.

In order to create an object for vector we need to import vector from util package.

for vector we can create more than 1 object.

```
import java.util.Vector;
class P2
{
    p s v m (string [] args)
    {
    Vector obj = new Vector();
    }
}
```

Array List	Vector
1) Array List is not synchro-nized	Vector is synchro
2) ArrayList increments 50 % of current array size if the number of elements exeeds from its capacity.	Vector increments 10 100 % means doubles the array size if t total no. of element exceeds than its capa
3) ArrayList is not a lagacy class. It is introduced in JDK 1.2	Vector is a legacy class.
4) ArrayList is fast because it is non synchronized.	Vector is slow because It is synchronized i.e. in a multithreading environment, it holds the other threads in runnable or non runn -ble state until curren thread releases the lock of the object.
5) ArrayList uses the iterator interface to traverse the elements	A vector can use the Iterator interface or Enumeration interface to traverse the element.

Programs of Array list :-

1)

```
import ~~clas~~ java.util.ArrayList;
class P1
{
  psvm (String[] args)
  {
    ArrayList obj = new ArrayList();
    s.o.pln (obj);

   s.o.pln ( obj.isEmpty());
    obj.add ("Pizza");
    obj.add (12);
    obj.add (false);

    s.o.pln (obj);
   s.o.pln (obj.isEmpty());
   s.o.pln (obj.add (12));

    s.o.pln (obj);
   s.o.pln (obj.size());
  }
}
```

O/P = []
true
[Pizza, 12, false]
false
true
[Pizza, 12, false, 12]
4

size
~~isEmpty~~ () :- (what is size () ?)

'size' is a Non static method of ArrayList class.

'size' is a concrete method.

It is used to count the size of array list.

The return type of 'size()' is integer value.

'size()' is accessed with the help of object referance variable.

add () :- (What is add () ?)

add() is non static concrete method of ArrayList class.

add() is used to add the ~~value~~ object or element to the arrayList.

We can add object in the arrayList in two ways :-

1) By passing value inside add method. It will add the newly added object at the end of the ArrayList.

2) By using index value. We can pass the index value & then put comma & pass the value for object. It will add the object in between the ArrayList 目.with respect to the index value. in the List.

ex. 1) directly

```
obj.add("pizza");
obj.add(12);
```

O/P :- [Pizza, 12]

2) With index value :-

obj.add(1, false);
1, false

O/P :- [Pizza, false, 12]

The return type of add() is boolean value (true / false).

As add is NS concrete method we have to access it with the help of object referance variable.

~~size() :-~~

isEmpty :- (What is isEmpty()?)

'isEmpty' is a NS concrete method of arrayList class.

It is used to check whether the arrayList is empty or not.

If the ArrayList is empty it will return the value false.

If the ArrayList is not empty then it will returen value true.

The return type of this method is boolean value.

As it is an NS concrete method so we can access with the help of object referance variable.

2)
```
import java.util.ArrayList;
class P2
{
  psvm (String[] args)
  {
   ArrayList obj = new ArrayList ();
    obj.add (1);
    obj.add (false);
    obj.add ("Oreo");

    S.o.pln (obj.contains ("oreo");
  }
}
```

★ What is contains()?

→ Contains() is a non static concrete method of arrayList.

It is used to verify whether the passed value is present inside the Arraylist or not.

If the value is present inside Arrayli then the value returns the value true.

If the value is not present inside Arraylist then it will return value as false.

The return type of contains() is boolean value.

We can access the contains() with the help of object referance variable.

3)
```
import java.util.ArrayList;
class P3
{
    ps vm (string []args)
    {
     ArrayList a1 = new ArrayList ();
     a1.add(1);
     a1.add("oreo")

    s.o.pln (a1);

    ArrayList a2 = new ArrayList ();
     a2.add (1);
     a2.add ("monaco");

     s.o.pln (a2);
     s.o.pln (a2.size());
     a2.add (a1); // adding the entire arraylist

     s.o.pln (a2.size());
     s.o.pln (a2.size());
    }
}
```

O/P :-

[1, oreo]

[1, monaco]

2

[1, monaco, [1, oreo]]

3

★ What is addAll() ?

→ 'addAll' is a non-static concrete method of arrayList.

It used to merge all the objects which present inside one object of arrayList into the object of another arrayList.

ex.

```
a2.addAll (a1);
```

O/P :-

[1, ~~oreo~~ monaco, 1, oreo].

The return type of addAll() is a boolean value.

We can access this method with the help of object referance variable.

4)

```
import java.util.ArrayList;
class P4
{
  psvm (string[]args)
  {
    ArrayList a1 = new ArrayList();
    a1.add (1);
    a1.add ("Oreo");
    s.o.pln (a1);

    ArrayList a2 = new ArrayList();
    a2.add (1);
    a2.add ("monaco");
    s.o.pln (a2);
```

```
        s.o.pln (a1.size());
        s.o.pln (a2.size());

        a2.addAll (a1); //adding all the object
                          from one arraylist to
                          another.

        s.o.pln (a2);
        s.o.pln (a2.size());
    }
}
```

O/P :-

```
[1, oreo]
[1, monaco]
2
2
[1, monaco, 1, oreo]
4
```

5)

```
import java.util.ArrayList;
class P5
{
    psvm (string [] args)
    {
     ArrayList al = new ArrayList();
      //index_value: 0
        al.add ("hello");
        al.add (true);
        al.add (123);
        s.o.pln (al);
        al.add (1, 'a');
        s.o.pln (al);
    }
}
```

o/p = oreo
oreo
)
true
true

* What is get()?

→ get() is a non static concrete method of arraylist.

~~get() is used to add the object or element.~~

get() is used to access only particular object from arraylist with the help of index value.

It is accessed with the help of object referance variable.

7)

```
import java.util.ArrayList
class P7
{
    psvm
    {
    ArrayList al = new ArrayList;

    al.add ("oreo");
    al.add (1);
    s.o.pln (al.get (0));
    s.o.pln (al.get (1));
    s.o.pln (al.get (2));

    }
}
```

* remove ( ) :-

It is used to remove an object which is present inside the ArrayList.

With the help of remove method we can remove object in 2 ways.

1) Directly passing the value.
2) With the help of index value.

```
import java.util.ArrayList;
class P10
{
    psvm (String [] args)
    {
    ArrayList al = new ArrayList ();
     al.add(1);
     al.add(true);
     al.add("hello");

     s.o.pln (al);
     al.remove (true)

     s.o.pln (al);

     // remove (value)
    }
}
```

O/P:

[1, true, hello]

[1, hello]

```
import java.util.ArrayList;
class P11
{
  psvm (String [] args)
  {
   ArrayList al = new ArrayList();
   al.add(1);
   al.add(true);
   al.add(true);
   al.add("hello");

   s.o.pln (al);
   al.remove (2); //index-value
   s.o.pln (al);
   // remove(index-value)
  }
}
```

O/P :-

[1, true, true, hello]

[1, true, hello]

```
import java.util.ArrayList;
class P12
{
  psvm (String [] args)
  {
   ArrayList al = new ArrayList();
   al.add(1)
   al.add(true);
   al.add(true);
   al.add("hello");
```

```
        s.o.pln (al);
        al.remove (1); // considered as index value.
        s.o.pln (al);
    }
}
```

O/P:-

[1, true, true, hello]

[1, true, hello]

```
import java.util.ArrayList;
class P13
{
    psvm (String [] args)
    {
    ArrayList al = new ArrayList();
     al.add(1);    //Integer
     al.add(true); //Boolean
     al.add(true);
     al.add("hello"); //String

     s.o.pln (al);
     al.remove (Integer.valueOf(12)); //to remove integer obj
      // Passing the value)
     s.o.pln (al);

     // remove (class_name.value of (value));
    }
}
```

removeAll():-

It is used to compare one ArrayList with another ArrayList and remove same type of object.

```
import java.util.ArrayList;
class P14
{
  psvm (String [] args)
  {
   ArrayList a1 = new ArrayList ();
   a1.add(1);    // Integer
   a1.add(true); // Boolean
   s.o.pln (a1);

   ArrayList a2 = new ArrayList ();
   a2.add ("hello"); // String
   a2.add (true);  // Boolean
   s.o.pln (a2);
   a2.removeAll (a1);
   s.o.pln (a2);
  }
}
```

O/P :-

[1, true]
[hello, true]
[hello]

retainAll() :-

It is used to compare one ArrayList with another ArrayList and remove different type of object. by keeping same type of object.

```
import java.util.ArrayList;
class P15
{
  psvm (String[] args)
  {
    ArrayList al = new Arraylist();
    al.add (1);
    al.add (true);
    s.o.pln (al);

    ArrayList a2 = new ArrayList();
    a2.add ("hello");
    a2.add (true);
    s.o.pln (a2);
    a2.retainAll(al);
    s.o.pln (a2);
  }
}
```

O/P :-

[1, true]
[hello, true]
[true]

Q. Write diff^n bet^n removeAll() & retainAll() metho

RemoveAll	RetainAll
Remove all method used to compare one arraylist object with another arraylist object & remove the same type of object	RetainAll method used to compare one arrayli object with another arraylist object & remo the different type of object.
It retains different types of object	It retains same type of object.
Return type is boolean value	Return type is boolean value.

Q Can we make arrayList as homogeneous? possible?

With the help of generics we can make ArrayList as homogenous such that is won't allow a programmer to store different type of objects.

```
import java.util.ArrayList;
class P15
{
  p.s.v.m
  {
   ArrayList<Integer> al = new ArrayList<Integer>();
    al.add(1);
    a1.add(2);
    a1.add(3);

    for (int index=0; index<al.size(); index++)
    {
      s.o.p.ln (al.get(index));
    }
  }
}
```

O/P :-

1
2
3

Important points of ArrayList :-

ArrayList consists of index value (starts with 0 & gets ended by size() - 1).

ArrayList maintains insertion order (FIFO: First I/P & First O/P).

In arraylist we can store duplicate type of objects.

Set Interface :-

Set is an interface in java.

for set interface we cannot create an objec

For set interface we can create object referance variable.

set interface is present inside java.util package.

The fully qualified name of set interface is java.util.Set.

for set interface we cannot create an object but we can create an object to its child class.

HashSet :-

HashSet is a concrete class in java.

Hashset is present inside java.util package.

The fully qualified name of HashSet class is java.util.Hashset.

With the help of Hashset we can add, remove & modify object.

HashSet does not maintain insertion order.

In Hashset duplicate objects are not allowed.

For Hashset we can create object by importing from util package first.

```
import java.util.HashSet;
class P16
{
    psvm
    {
        HashSet h1=new HashSet();
        h1.add(1);
        h1.add(true);
        h1.add("hello");
        h1.add('a');
        h1.add(1);
        h1.add(false);
        h1.add(true);

        s.o.pln(h1);
    }
}
```

O/P :-

[1, a, false, hello, true]

Treeset :-

TreeSet is a concrete class in java.

TreeSet is present inside java.util package.

The fully qualified name of TreeSet is java.util.Treeset.

Treeset does not allow duplicate objects.

In treeset the objects are automatically sorted in ascending order.

Since treeset has a nature of sorting the objects. ~~Hence Treeset is homogeneous.~~

In side treeset we need to pass homogeneous objects (same type).

For treeset we can create an object by importing from util package.

```
import java.util.Treeset;
class P16
{
    p s v m
    {
    Treeset t1 = new Treeset ();
    t1.add (4);
     t1.add (4);
     t1.add (3);
     t1.add (2);
     t1.add (2);
     t1.add (1);
     t1.add (3);
     t1.add (4);

     S.o.pln (t1);
    }
}
```

O/P:-

[1, 2, 3, 4]

Q. Why do we get classCastException in Treeset.

Treeset has a nature of sorting all the object which is present in ascending order.

Because of this the objects which is present insid treeset ~~not~~ are always should be comparable type.

But if we try to pass different type of object inside treeset we get an runtime exception called as ClassCastException.

```
import java.util.Treeset;
class p16
{
    psvm
    {
     Treeset t1 = new Treeset();
     t1.add(1);
     t1.add(2);
     t1.add(true);
     t1.add(1);

     s.o.pln(t1);
    }
}
```

Q.1) Can we handle ClassCastException of treeset with help of ~~try~~ EHM? Explain with program & write.

2) Verify all the methods of arraylist using vector.

3) Verify all the methods of arraylist using Hashset. & treeset. & check which method supports & which does not support.

4) WAP to remove duplicate objects of arraylist using Hashset.

* WAP to sort & Reverse the object which is present inside ArrayList.

We can sort the objects which is present inside arraylist in ascending order with the help of sort method which present inside collections class.

In order to access sort method we need to import collections class from util package first.

Sort method is a static method which is present inside collections class.

We can get the objects of arrayList in descending order also. but with the help of reverse() of collections class.

Reverse() is a static method which is present inside collections class.

We can access both sort & Reverse method with the help of class name as referance.

ex.

```
import java.util.ArrayList;
import java.util.Collections;
class P17
{
  psvm
  {
    ArrayList<Integer> al = new ArrayList<Integ

    al.add(1);
    al.add(3);
    al.add(2);
```

```
        S.O.pln (a1);
        // Sort : Ascending order
        Collections .sort(a1);
        S.O.pln (a1);
        // reverse : Descending / Reverse ???
        Collections.reverse(a1);
        S.O.pln (a1);
    }
}
```

O/P

[1, 3, 2]
[1, 2, 3]
[3, 2, 1]

* Advanced for loop :-

With the help of referance of an object we can ~~create an object~~ read the no. of objects present inside arraylist.

We need to create a referance variable, in order to use advanced for loop.

Advanced for loop is ~~called~~ also called as for Each loop.

Syntax :-

```
for (class_name ref-var : ref-var)
{
    ~~refer~~ statements;
}
```

```
import java.util.ArrayList;
class P18
{
    psvm
    {
```

```
ArrayList al = new ArrayList();
al.add(1);  // Integer
al.add('a'); // Character
al.add(true); // Boolean

    // obj : j.l.Integer
for ( Object obj : al) // j.l.Integer: Upcasting
{
  S.O.println (obj.toString(-)); // method overriding
    }
  }
}
```

Q. Read the objects of arraylist without using index value & without using ~~obj~~ advance for loop.

→ Yes, we can read the objects of arraylist with the help of supermost interface in java. (Iterator interface).

ex.

```
/*
import java.util.ArrayList;
import java.util.iterator;
class pig
{
   psvm
   {
      ArrayList al = new ArrayList();
      al.add(123); //Integer
      al.add('d'); //character
      al.add('false'); //Boolean

      //al:arraylist : concrete class
      //obj: Iterator : interface

      //Upcasting
      Iterator obj = al.iterator(); // NS-method

      while(obj.hasNext())
      {
         s.o.pln(obj.next());
      }
   }
}
```

hasNext ():-

It is used to check or verify whether the current object is holding / pointing the address of next object or not.

next ():-

It is used to read or print the value present inside the current object.

Q.1) Can we sort & reverse the object of hashset & treeset. → No

2) Read the objects of Hashset & Treeset with the help of index value.

3) Remove duplicate objects of ArrayList using Treeset.

4) Verify advanced for loop & Reading the objects with the help of superMost interface using Hashset & Treeset?

1) Remove duplicate value with the help of Hashset & Treeset

```
import java.util.*ArrayList;
class P15
{
  ps vm
  {
    ArrayList obj = new ArrayList();
    obj.add (125);
    obj.add (112);
    obj.add (123);
    obj.add (125);
    obj.add (132);
    obj.add (123);
    obj.add (135);
    obj.add (122);
    obj.add (113);
    S.o.pln (obj);

    Hashset al = new Hashset (obj);
    S.o.pln (al);
```

```
        TreeSet a2 = new Treeset(obj);
        S.O.pln (a2);

        }
    }
```

1) i)
```
import java.util.TreeSet;
import java.util.Collections;
class P16
{
    psvm
    {
     TreeSet a1 = new Treeset();
     a1.add (10);
     a1.add (14);
     a1.add (11);
     a1.add (12);
     S.O.pln (a1);
     collections.sort (a1);

     S.O.pln (a1);
    }
}
```

ii)
```
import java.util.HashSet;
import java.util.Collections;
class P17
{
    psvm
    {
```

```
        HashSet a1 = new HashSet();
        a1.add (10);
        a1.add (14);
        a1.add (11);
        a1.add (12);
        s.o.pln (a1);
        Collections .sort (a1);

        s.o.pln (a1);
    }
}
```

→ No, we cannot sort, or reverse the object of treeset & HashSet class.

2)

```
import java.util. HashSet;
import java.util. Treeset;
class P18'
{
  psvm
  {
   Treeset a1 = new Treeset;
    a1. add (10);
            (14);
            (11);
            (12);
   s.o.pln (a1);
   s.o.pln (a1.get (0));
```