Date / /
Page No.

* **LANGUEGE :-**
→ Any medium to communicate is called language.

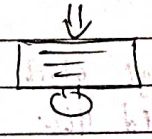
* **PROGRAMMING LANGUAGE :-**
→ A language ie used to communicate to the computer is called programming language.
→ Programming language is in the form of set of instruction.

Types of Programming Language :-
(i) Low Level Language / Binary Language
(ii) Assembly Language / Mid Level Language
(iii) High Level Language

* **LOW LEVEL LANGUAGE :-**
→ Low level language is the first programming language which was used to communicate with computer.
→ The instructions of LLL is in the form of '0' and '1', so it is also called as binary language.

Ex:- 0 ⟶ off ⟶
1 ⟶ ON ⟶

Eg. Program :-
01011011 10011011 10111001
11000011 ------ ------



A computer will directly understand low level language.

Date / /
Page No.

Date / /
Page No.

* ASSEMBLY LANGUAGE :-

→ Assembly language is the mid level programming language where predefined words will use to make programming easy.

→ AL code was easier than LVL but computer can't understand assembly languag directly.

eg. Program :-

```
ADD   1.81    1.81C
DIV   1.111   1.81b
PUSH  1.81C   1.311
POP   1.11b
CALL  LDR@11C ....
```

Predefined words — address of memory block

⇓ MUL

[Assembler]

⇓ LVL

→ A computer can't understand assembly language code directly so assembler is used as a translator which converts assembly language into LVL.

* HIGH LEVEL LANGUAGE :-

→ A language which is written in human understandable form is called HLL.

→ HLL programming is written by using simple english in order to make programming easy.

Date / /
Page No.

eg. Program :- C-language — 1972

```
int main()
{
    int age = 23;
    if (age >= 18)
    {
        printf("You are Eligible for vote");
    }
    else
    {
        printf("You are not eligible");
    }
    return 0;
}
```

⇓ HLL

[Compiler or Interpreter]

⇓ NVL

→ A computer can not understand HLL directly so compiler or interpreter is used in between as a translator which converts high level language into next level language (NLL).

* COMPILER :-

→ Compiler is a translator which reads complete code at a time and at it generates next level code if the provided code is correct.

Date / /
Page No.

* **INTERPRETER:-**
→ Interpreter reads code line by line and generates next level code for one line at a time.
→ If Interpreter comes across an error then any further execution will be suspended from that line.

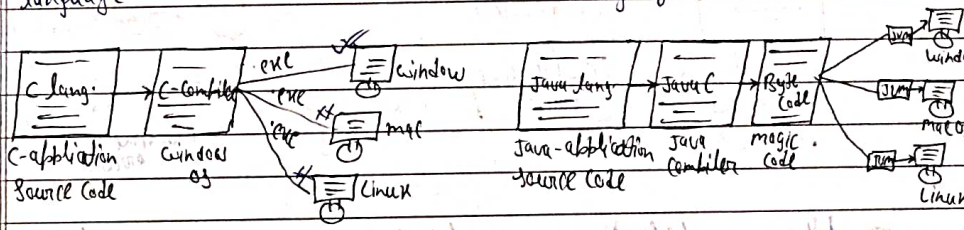
=> Compiler provides faster execution than an Interpreter.

* **JAVA C [JAVA COMPILER]:-**
→ It is the name of Java compiler.

High Level Language
- Procedure Oriented Language [C]
- Object Oriented Language [C++, Java, Python, Java Script etc.]

* **Difference Between Programming Procedural and Object Oriented Language:**

Procedural Language	Object Oriented Language
→ Procedure oriented language (C) is not a secure programming language.	→ But object oriented language (Java) is a secure programming language.
→ Procedure oriented language (C) is platform dependent programming language.	→ But object oriented language (Java) is platform independent programming language.
→ Procedure oriented language does not use class and object concept to design it's program.	But in object oriented language class and object concept is used to design a program.
→ A procedure oriented language does not support the object oriented concept such as encapsulation, inheritance, polymorphism and abstraction.	→ But an object oriented language supports all the object oriented concepts.
→ C-language is a platform dependent language.	→ But Java is platform independent language.



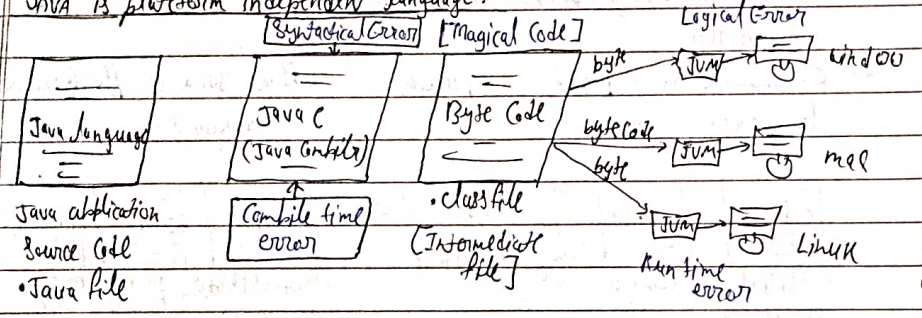
* When an application is build by C-programming language then the first step is compilation.
* C-compiler generates platform specific code which can execute only on a particular platform so C-language is called platform dependent language.

JAVA:-
→ Java is a high level, object oriented, platform independent and secure programming language which supports object oriented concepts such as encapsulation, inheritance, polymorphism and abstraction.
→ Java was released by Sun Micro System in 1995 which was developed under James Gosling.

1995 → Java beta version
1996 → Java 1.0.0 (JDK 1.0.0)
1997 → Java 2.0.0 (JDK 2.0.0)
⋮
March, 2025 → Java 20.0.0 (JDK 20.0.0)

→ JDK-8.0.0 is the most stable version of JDK.
→ In 2010 oracle bought Sun Microsystem.

* Java is plateform independent language :-



→ When an application is developed by Java programming language the first step is compilation.
→ JavaC is the Java compiler which takes Java file and if it correct then compilation will be successful and next level code will be generated which is called byte code or •class file or magical code.
→ Byte code is intermediate code which can not run directly on plateform or a computer.
→ JVM (Java Virtual Machine) comes between as translator which converts byte code into low level code and in this manner Java becomes plateform independent so it is called WORA language.

WORA. => Write Ones Run Anywhere

→ Java is plateform independent ~~but~~ but JVM is plateform dependent because it is different for different system.

* Compile Time Error :-

→ If a Java program will have any syntactical type error then it is identified by compiler by compile time and give an error it is called compiled time error.

* Run Time Error :-

→ If a Java program will have any logical type error then it is identified by JVM at run time. When JVM gives an error which is called Run time error or exception.

* JVM (JAVA VIRTUAL MACHINE) :-

→ JVM stands Java Virtual Machine.
→ JVM converts byte code into low level code and to perform this task JVM has some components which works together such as :-

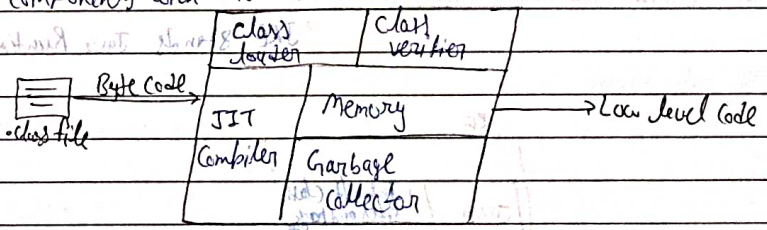


fig: JVM

* Class Loader :-

→ It loads class file into JVM.

* Class Verifier :-

→ It reads class file line by line and generates low level code.

* JIT Compiler :-

→ JIT stands 'Just In Time' compiler. It helps class verifier at run time so that over all efficiency of JVM can be improved.

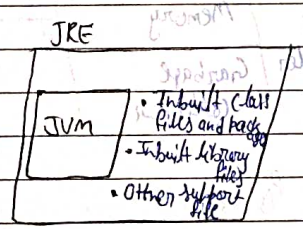
Date / /
Page No.

* **Memory Block:-**
  → Memory block is responsible for each task

* **Garbage Collector:-**
  → It is responsible to delete unwanted members from memory so that smooth execution of the program can be mandate.

→ JVM is called virtual because it does not exist alone.
→ It needs a body for implementation which is provided by JRE.

* **JRE (JAVA RUN TIME ENVIRONMENT):-**
  → JRE stands Java Runtime Environment.



→ JRE provides implementation for JVM, so JVM will be physically available inside JRE.
→ JRE combines JVM along with inbuilt class file, packates and library files which is needed by JVM at run time.
→ By using JRE we can only run an application but we can not do development work.

* **JDK (JAVA DEVELOPMENT KIT):-**
  → JDK stands Java Development kit.
→ JDK combines JRE along with all the development tools which are needed by a program.

Date / /
Page No.

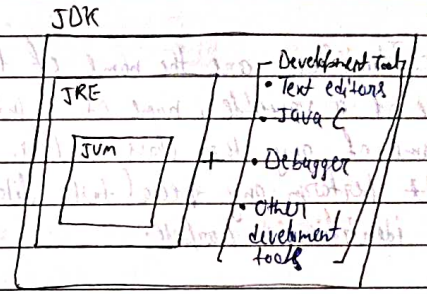
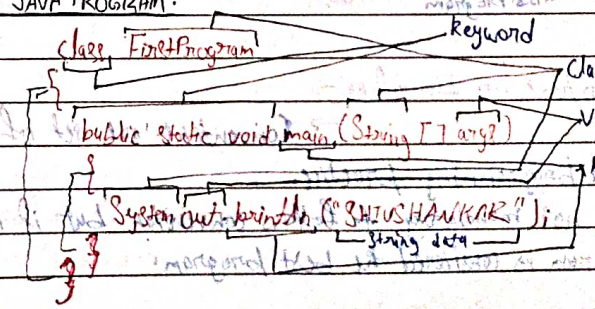


fig:- JDK

→ By using JDK we can run an application as well as we can group development work.
→ JDK is plateform dependent because the JVM inside the JDK is plateform dependent.

* **KEY WORDS:-**
  → Keywords are predefined words or reserved words which are used to perform some specific task.
→ Java has 49 keywords and all are written in lower case letters.
→ eg:- if, else, while, do, for, break, pass, continue, switch, case, byte, long, int, short, char, float, double, boolean, try, catch, finally, throw, throws, final, extends, class, public, static, void, etc.

* **STRUCTURE OF JAVA PROGRAM:-**



Date / /
Page No.

* IDENTIFIERS:-

→ Identifiers are the name of the class, name of a method, name of a variable, name of a interface, name of a package or name of any other class members.

→ Identifiers don't perform any special task like keywords. It is used only for identification purpose.

* Rules for Identifiers:-

(i) An identifier can be written by using alphabets (A-Z or a-z), Numbers (0-9) and special character ($, _).

(ii) An identifier must not with numbers.

(iii) An identifier must not have any space.

(iv) Any keyword can't be used as a identifier.

eg:- (i) FirstProgram123 ✓✓
(ii) 123FirstProgram ✗
(iii) _FirstProgram ✓✓
(iv) _123FirstProgram ✓✓
(v) First$Program ✓✓
(vi) First Program ✗
(vii) public ✗✗
(viii) Public ✓✓
(ix) public123 ✓✓
(x) firstProgram ✓✓

* Conventions for Identifier:-

→ Conventions are set of guidelines for good programming practise.

→ It is not mandatory to follow convention but if it is followed then program is considered the best program.

Date / /
Page No.

(a) Convention for a Class Name:-

→ If a class name has only one word then the first character should be in uppercase.

eg:- class Test { }

→ If a class name has multible words then first character of each word should be upper case letter.

eg:- class PrimeNumber { }, class SumOfNumber { }

→ There are some library files stored inbuilt classes and there names stored in first letter in capital i.e some name start with capital letter.

(b) Conventions for method name:-

→ If a method name has only one word then it should be in lower case letter.

eg:- test(), demo(), factorial(), multiply(), etc.

→ If a method will have multiple method then the first word should be in lower case letter and then from second word the first character of each word should be in uppercase.

eg:- findFactorial(), ~~sum~~ sumOfNumber(), etc.

(c) Convention for variable Name:-

→ If a variable name has only one word then it should be in lower case letter.

If it has multible words then there should be underscore (_) between two words.

eg:- int age = 18; String color = "Red";
int age_of_employee = 24; String[] color = {"Red", "Green", "Blue"}
int[] age = {12, 18, 30, 16, 42, 58, 68}

Date / /
Page No.

* **Literals:-**
  → Literals are different types of values that are used to writing the Java program.

* **Types of Literals:-**
  (i) Single Values Literals,
  (ii) Multi Values Literals.

(a) Numeric Type Literals
(b) Character Type Literals
(c) Boolean Type Literals

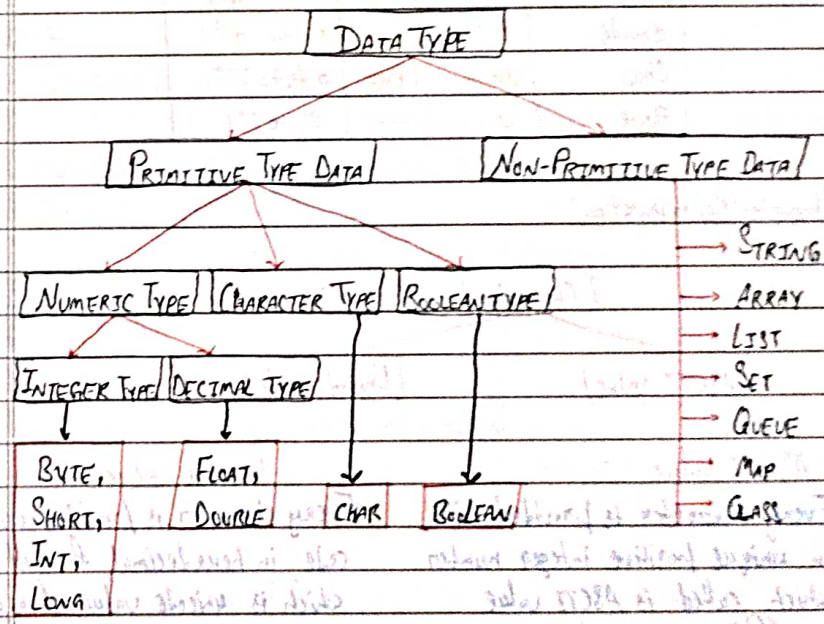
(a) String
(b) Array
(c) List
(d) Set
(e) Queue
(f) Map
(g) Class ⇒ Multivalued literals, etc.

* **DATATYPE:-**
  → In Java different types of data or literals are categorized based on it's ~~ty~~ type and based on it's size under datatype.
→ On its type basis Java has 2 types of data such as:-
(a) Primitive Type Data,
(b) Non-Primitive Type Data.

* **PRIMITIVE TYPE DATA:-**
  → The single valued type of data in Java are called primitive types of data.
→ In Java there are total eight (8) primitive type of data and it is available in the form of keywords.

Date / /
Page No.

* **NON-PRIMITIVE TYPE DATA:-**
  → The multivalued types of data in Java are called non-primitive type data.
→ The class type data in Java is called non-primitive data.
→ A non-primitive data can be predefined class or it can be a user defined class.



eg:-

```
class Employ {
    String name;
    int id;
    double salary;
    int age;
}
```

Date / /
Page No.

* **SIZE:-**

Data Type	Default Value	Size	Range
Boolean	false	1 ~~Byte~~ Bit	(-128 to 127) 2^{8}
Short	0	2 Byte	-2^{15} to $2^{15}-1$ -32768 to 32767
Int	0	4 Byte	-2^{31} to $+2^{31}-1$
long	0l or 0L	8 Byte	-2^{63} to $+2^{63}-1$
float	0.0f or 0.0F	4 Byte	-2^{31} to $+2^{31}-1$
double	0.0	8 Byte	-2^{63} to $+2^{63}-1$
Char	'\u0000'	2 Byte	0 to 65,535
Byte	0	1 Byte	-128 to 127

Formula for Range:-
$= -2^{n-1}$ upto $2^{n-1}-1$
where n = no. of bits
like 1 byte = 8 bits
$\Rightarrow -2^{7}$ to $2^{7}-1$
$\Rightarrow -128$ to 127

* **Character Representation:-**

Character
- ASCII value
- Unicode value

ASCII value

→ Every character is provided with an unique positive integer number which called is ASCII value

ex:- '@' = 64
'A' - 'Z' => 65 - 90
'a' - 'z' => 97 - 122
'0' - '9' => 48 - 57

Unicode Value

Every character is provided with a code in hexadecimal format which is unicode value of a char.

ex:- $\underbrace{\backslash u0041}_{\text{hexadecimal no.}}$ => F

* **VARIABLES:-**

→ Variables in Java is a block of memory or a container which is used to store a value.

ex:- int age = 22;

Date / /
Page No.

(a) **Declaration of the Variable:-**

→ Providing datatype to a variable is called variable declaration.

Syntax:-

```
Datatype Var_Name;
```

ex:-

```
int x;
byte y;
```

(b) **Initialization of a variable:-**

→ Assigning value to an already declared variable is called initialization of a variable.

Syntax:-

```
Var_Name = Value/Expression;
```

eg:-

```
x = 12;
y = 40;
```

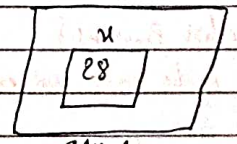
(c) **Declaration and Initialization of Variable:-**

Syntax:-

```
data_type Var_Name = Value/Expression;
```

ex:-

```
int x = 28;
char y = '8';
```



RAM Area

* **Declaration of all data type:-**

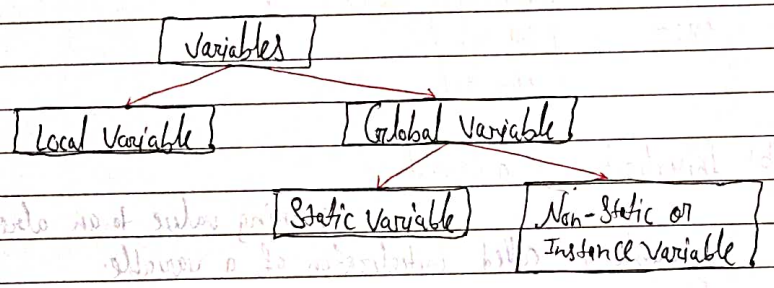
→ byte a = 120;
→ short b = 255;
→ int c = 578;
→ long d = 123456789010L;
→ float e = 12.5869f;
→ double f = 35.6732;
→ char g = '@';
→ String h = "Shivshankar Mishra";
→ boolean i = true;

Date / /
Page No.

* Types of Variable:-

→ Variables of Java is of two types:-

(i) Local Variable,
(ii) Global Variable.



* LOCAL VARIABLE:-

→ If a variable is not directly inside class block then such variable is called a local variable.

→ In other words if a variable is inside method block, constructor block or for any block after class block then it is called a local variable.

eg:-

```
Class Program1 {
public static void main(String[] args)
{
    int a = 30;
    int b = 48;   } Local Variable
}
}
```

Date / /
Page No.

* GLORAL VARIABLE:-

→ If a variable directly inside a class block then such value is called global variable.

eg:-

①

```
Class Program1
{
    int x = 12;
    int y = 40;   } Global Variable
    public static void main(String[] args)
    {
    }
}
```

②

```
class Test {
    int a = 10;
    int b = 20;
    public static void main(String[] args)
    {
        int c = 30;
        {
            int d = 40;
        }
        int e = 50;
    }
    int f = 60;
    {
        int g = 70;
    }
    int h = 80;
}
```

Local varibal	Global var.
c	a
d	b
e	f
g	h

Date / /
Page No.

* **Static Variable:-**

→ If a global variable is declared with the keyword static then it called a static variable.

→ A static variable can be accessed inside the main method directly or on class name as reference.

* **Non-Static Variable:-**

→ If a global variable is not declared with the keyword static then it is called a non-static variable.

→ A non-static variable can be accessed only on object only on object reference inside main method.

eg:-

```
class Program1
{
    static int x = 30;   → static variable
    int y = 40;          → Non-static variable
    public static void main (String [] args)
    {
        S.o.p (x);
        S.o.p (Program1.x);   } static variable is being accessed directly or on class reference
        Program1 P1 = new Program1();
        S.o.p (P1.y);
    }
}
```

Program1 object → Program1 P1 = new Program1();

object reference → S.o.p (P1.y);

object of Program1 class

non-static variable is being accessed on object reference.

* **Properties of a Variable:-**

→ A global variable will always have default ~~valy~~ value.

Date / /
Page No.

eg:-

Datatype	Defaul Value
String	null
Any non-primitive	null

→ A local variable will not have any default values.

→ A local variable must be initialized in order to use it.

→ We can not declare same variable with inside same class more than once.

eg:-

```
class Program1
{
    int a = 12;
    int b = 30;
    int a = 40;   ✗ ERROR
    public static void main (String [] args)
    {
        int x = 40;
        double y = 60.5;
        String x = "Mohan";   } Error not allowed in
        int y = 61;           } same block declaration
    }                           of a variable more than one
}
```

```
{
    int a = 40;
}
{
    int a = 50;   ✓ Allowed in different block
}
```

→ We can have local variable and global variable with same name.

eg:-

```
class Project1
{
    static int x = 50;
    public static void main (String[] args)
    int x = 60;
    S.o.p (x); } }   => x = 60
```

→ If local variable and global variable exist with same name then preference given to a local variable and global variable getting hidden which is called variable hiding.

eg:-

```
class program
{
  int x=30;
  public static void main(String[] args)
  {
    int x= 60;
    S.o.p(x); -> local x
    program z = new program();
    z.Sop(z.x); -> global non static of x
  }
}
```

→ We can change the value of a local variable only by using it's variable name.

eg:-

```
Class Program
{
  public static void main(String[] args)
  {
    int x= 30;
     x= 60;
    S.o.p(x); ---> x=60
     x= 85;
    S.o.p(x); ---> x=85
     x= 350;
    S.o.p(x); ---> x=350
  }
}
```

→ A local variable is created when control of JVM enters to block and it is deleted when control is out of the block.

eg:-

```
class program
{
  static int x = 10; ---> *
  public static void main(String[] args)
  {
    int x= 20;
    S.o.p(x); ---> x=20
  }
  S.o.p(x); ---> x=10
}
```

* COMMENT LINE :-

→ Comment line in any programming language is used to write comments or messages to help user to understand code easily.

\# Types of Comment:-

(i) Single line Comment:-

```
// It is single line comment
```

(ii) Multiline Comment:-

```
/*
  It is multiline comment
*/
```

* TYPE CASTING :-

→ Type casting is a process to convert one type of data into another type of data.

→ It is of two types:-
(i) Primitive Type Casting,
(ii) Non-Primitive Type Casting.

* **Primitive Type Casting:-**
→ Converting one primitive data into another primitive data is called Primitive type casting.
→ It is of two types:-
(i) Widening,
(ii) Narrowing.

* **Non-Primitive Type Casting:-**
→ Converting a non-primitive data into another non-primitive data is called non-primitive type casting.
→ It is of two type:-
(i) Upcasting,
(ii) Downcasting.

Widening
byte < short < int < long < float < ~~decimal~~ double
Narrowing

* **Widening:-**
→ Widening is a process to convert a smaller type data into bigger type.
→ In widening there will be no data loss so it is done by compiler automatically.

eg:-

```
{ int x = 238;
  ~~int y~~ double y = x;
      sob(x);  => 238
      sob(y);  => 238.0
  char z = 'a';
  int a = z;
  sob(a); }  => 64 (ASCII value)
```

* **Narrowing:-**
→ Narrowing is a process to convert bigger type of data into smaller type of data.
→ In narrowing there will be data loss so it is not done by compiler automatically.
→ It is achieved by using typecast operator.

eg:-

```
{
  int x = 480;
  char y = (char) x;
      sob(x);   => 80
      sob(y);   => 0
  double a = 23.58;
  int b = (int) a;
      sob(a);   => 23.58
      sob(b);   => 23
}
```

* **CONCATENATION:-**
→ Concatenation is a process to attach any type of data with string.
→ ('+') operator is used as a concatenation operator.

eg:-

```
{ int a = 28;
  int b = 20;
  int c = a+b;
  sob("The sum is:" + c);  => The sum is: 48
  sob("The sum of a and b is:" + c);  The sum of a and b is: 48
  sob("The sum of " + a + " and " + b + " is: " + c);  The sum of 28 and 20 is: 48
}
```

Date / /
Page No.

* OPERATORS:-

→ Operators are predefined symbols used to perform some operations.

* Operands:-

→ Values needed by an operator are called operands.

* Types of Operator:-

(i) Unary Operator:- Only one operand needed.
(ii) Binary Operator:- Two operands needed.
(iii) Ternary Operator:- Three operands needed.

1. ARITHMETIC OPERATOR:-

→ Arithmetic operators are binary operator which are used to perform arithmetic operation. They are:-
('+', '-', '/', '*', '%')

→ $\frac{int}{int}$ ⇒ int, $\frac{double}{int}$ ⇒ double, $\frac{double}{double}$ ⇒ double, $\frac{int}{double}$ ⇒ double.

→ ('+') operator performs two operations in Java:-
(i) Addition,
(ii) Concatenation.

* USER INPUT:-

→ To take input from user we have to follow three steps in Java:-

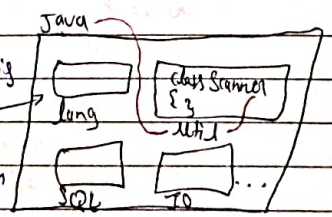
Step 1:-

import java.util.Scanner;
OR
import java.util.*;

we have imported only scanner class of util package

Here, we imported all of the classes of util package.

lang package is by default available to every program in Java



Date / /
Page No.

→ We have to import Scanner class of util package by using keyword import.

→ The import state must be the first statement of the program.

Step 2:-

→ We have to create object of scanner class where we want to ~~tell~~ take user input.

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

object creation of Scanner class to take input from system

Step 3:-

→ We have to use different methods of scanner class for different types of input.

Data type	Methods of Scanner Class
Byte	nextByte()
Short	nextShort()
int	nextInt()
long	nextLong()
float	nextFloat()
double	nextDouble()
boolean	nextBoolean()
String (Firstwords)	next()
String (Completely)	nextLine()
Char	next().charAt(index)

eg:-

```
import Java.util.Scanner;
Class Program
{
    public static void main (String[] args)
    {
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt();
        int b = sc.nextInt();
        int c = a+b;
        System.out.println("The sum of "+a+" and "+b+
                " is: "+c);
    }
}
```

→ for implementation of String, char datatype method use this:-

```
String d = sc.next();
String e = sc.nextLine();
Char f = sc.charAt(1);
```

we also can use:-

```
char g = sc.next().charAt(5);
char h = sc.nextline().charAt(5);
```

→ If we want to take string or input immediate int or other type of datatype then we use:-

```
int a = sc.nextInt();
sc.nextLine();
String b = sc.nextline();
```

* COMMAND LINE ARGUMENT :-

→ int[] x = {18, 20};
Sop(x[0]); => 18
Sop(x[1]+100); => 120

→ String[] x = {"Ram", "Mohan"};
Sop(x[0]); => Ram
Sop(x[1]); => Mohan

* ASSIGNMENT OPERATOR:-

* COMMAND LINE ARGUMENT:-

→ Command line argument is a mechanism where we can pass arguments to main method in the form of string.

→ (String[] args) of main method is used to accept multible argument from command line in the form of string.

→ Various operations can be perform from command line argument to get the fruitful result.

* ASSIGNMENT OPERATOR:-

→ Equals to is used as assignment operator.

→ when there will be assignment operator then reads the statement from right to left.

→ First right side calculation will be performed and then it will be assign to left side variable.

eg:-

```
int x = 12;
int y = x+30;
```

* COMPOUND ASSIGNMENT OPERATOR:-

→ Combination of two operators are called compound assignment operator. It also called ~~shortcut~~ shorthand operator.

→ They are:- (+=), (-=), (*=), (/=), (%=).

eg:-

```
int a = 12;
a += 30;   // a = a+30
Sop(a);  => 42
a -= 2;
Sop(a); => 40
a *= 5;
Sop(a); => 200
```

Date / /
Page No.

* RELATIONAL OPERATOR :-

→ Relational operator is used to compare two number and after comparision it gives boolean result.

They are :- ('<', '>', '<=', '>=', '==', '!=')

'=' => Assignment Operator

'==' => Relational operator to compare two numbers / operands.

eg:-

```
int a=40;
int b=24;
int c=1900;
Sop(a+b>100); => ~~true~~ false
Sop(a+b>2*b); => true
Sop((a+b)/4==a); => false
Sop((a+b)%4==0); => true
Sop(c%100!=0); => false
```

* Increment / Decrement :-

→ Increment / decrement operator is ~~either~~ either increase or decrease the variable by 1 of memory.

(a) Post Increment :-

x++;

first display the variable → Then increase the value of x by 1 of memory.

(b) Post Decrement :-

x--;

first display → Then decrease

Date / /
Page No.

(c) Pre Increment :-

++x;

first increase the value of x of memory by 1 → then display the value of x.

(d) Pre Decrement :-

--x;

first decrease → then display

eg:-

①
```
int x=40;
Sop(x++); => 40
Sop(++x); => 42
Sop(x--); => 42
Sop(--x); => 40
Sop(x); => 40
```

②
```
int a=24;
Sop(a++); => 24
Sop(a++); => 25
Sop(++a); => 27
Sop(--a); => 26
Sop(a++); => 26
Sop(++a); => 28
```

③
```
int a=24;
a++;
int b=a++;
int c=++a;
Sop(a); => 27
Sop(b); => 25
Sop(c); => 27
```

a [24|25|26|27]
b [25]
c [27]

a [30|31|32] b [30|31|32] c [63]

④
```
int a=30;
int b=a++;
b++;
int c=a++ + ++b;
Sop(a); => 32
Sop(b); => 32
Sop(c); => 63
```

Date / /
Page No.

* int a = 24; a [24 | 25 | 26] b [30 | 29 | 28] c [53 | 54 | 53] d [1]

int b = 30;

int c = a+++ --b; //53

c++; //54

int d = a++ +b - - - - - c; //1

Sop(a); => 26

Sop(b); => 28

Sop(c); => 53

Sop(d); => 1

* int a = 16; a [16 | 17 | 18] b [20 | 21 | 22] c [20 | 21] d [106]

int b = 20;

int c = 30;

int d = a++ + b++ + ++a + c+++ ++b; ~~=>106~~ //106

Sop(d); => 106

* int a = 18; a [18]

a = a++; //18

a = a++; //18

a = a--; //18

a = a++; //19

a = a++; //18

Sop(a); => 18

* <u>NUMBER REPRESENTATION</u>:-

→ Any integer number can be represented at the following ways:-

(a) Binary Representation,

(b) Octal Representation,

(c) Decimal Representation,

(d) Hexadecimal Representation.

Date / /
Page No.

* BINARY REPRESENTATION:-

→ If a number is written only by using two digit 0 and 1 then such representation is called binary.

→ In Java if a number start 0b then such number is considered in binary number.

→ A Binary number is written in 2's complement form.

eg:- 0b01101011 ✓✓

0b01432112 ✗✗

Binary no:- 0, 1 ⇒ decimal

10 ⇒ 2

11 ⇒ 3

100 ⇒ 4

101 ⇒ 5

$101 \Rightarrow 1\times2^2 + 0\times2^1 + 1\times2^0$

$= 4 + 0 + 1$

Binary (101) = (5) Decimal

* OCTAL REPRESENTATION:-

→ If a number is written by using 8 digits from 0 to 7 then such representation is called octal representation.

→ If a number in Java start with 0 then such number is considered octal in number.

→ An octal number is written in 8's complement form.

eg:- 0536 ✓✓

05981 ✗✗

Octal no:- 0, 1, 2, 3, 4, 5, 6, 7

10 ⇒ 8

11 ⇒ 9

12 ⇒ 10

13 ⇒ 11

14 ⇒ 12

15 ⇒ 13

$0862 \Rightarrow 0\times8^3 + 8\times8^2 + 6\times8^1 + 2\times8^0$

$342 = 3\times8^2 + 4\times8^1 + 2\times8^0$

$\Rightarrow 3\times64 + 4\times8 + 2\times1$

$\Rightarrow 192 + 32 + 2$

Octal(342) ⇒ (226) Decimal

Date / /
Page No.

* DECIMAL REPRESENTATION:-

→ If a number is re written by using 10 digits like 0 to 9 then such representation is called decimal.

→ In Java a number is not prefix with anything then such number is considered decimal no.

eg:- 1234 ✓✓

01234 ✗✗

Decimal no:- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

10 ⇒ 10

11 ⇒ 11

12 ⇒ 12

13 ⇒ 13

* HEXADECIMAL REPRESENTATION:-

→ If a number is written by using 16 digits from 0 - 9 and A - F then such representation is called hexadecimal representation.

→ If a number in Java is prefixed with 0x then such number is called hexadecimal number.

→ A Hexadecimal number is written in 16's complement form.

eg:-

0x701239 ✓✓

0xA002F ✓✓

0xAGH001 ✗✗

Hexadecimal number:- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Decimal	Hex
9 ⇒	9
10 ⇒	A
11 ⇒	B
12 ⇒	C
13 ⇒	D
14 ⇒	E
15 ⇒	F

$123 = 1\times16^2 + 2\times16^1 + 3\times16^0$

$= 256 + 32 + 3$

Hexadecimal(123) ⇒ (291) Decimal

* LOGICAL OPERATOR:-

→ Logical operator ~~two gives~~ takes boolean ~~operator~~ operand and boolean results.

They are logical (and, or, not).

(a) Logical AND (&&),
(b) Logical OR (||),
(c) Logical NOT (!).

* Logical AND:-

→ Logical AND takes two boolean operands and if both are true it gives true otherwise return false.

table (&&)

OP1	OP2	&&
0	0	0
0	1	0
1	0	0
1	1	1

0 => false
1 => true

table (&)

OP1	OP2	&
0	0	0
0	1	0
1	0	0
1	1	1

&& => fast because it can skip 2nd operand if 1st operand is false

& => slow because executes both operand every time.

eg:-

```
int x = 12;
int y = 20;
sop(x++>10 && y++>16); => true
sop(x);  => 13
sop(y);  => 21
```

```
int x = 12;
int y = 20;
sop(x++>12 && y++>10); => false
sop(x);  => 13
sop(y);  => 20
```

```
int x = 12;
int y = 20;
sop(x++>12 & y++>10); => false
sop(x);  => 13
sop(y);  => 21
```

* LOGICAL OR (||):-

→ Logical operator two boolean operand at least one is ~~try~~ true the it return true otherwise it return false.

OP1	OP2	||
0	0	0
0	1	1
1	0	1
1	1	1

0 → false
1 → true

(||, or)

OP1	OP2	||
0	0	0
0	1	1
1	0	1
1	1	1

0 → false
1 → true

(|, or)

OP1	OP2	|
0	0	0
0	1	1
1	0	1
1	1	1

|| => fast and efficient because it skips 2nd operand if 1st operand is true.

| => slow because it checks both operand in every case.

eg:-

```
class Program
{
    public static void main(String[] args)
    {
        int x=12;
        int y=20;
        sop (x++>10 || y++>12);  => true
        sop (x);  => 13
        sop (y);  => 21
    }
}
```

Date / /
Page No.

* **LOGICAL NOT (!):-**

→ Logical not take boolean operands and it changes the boolean state of the operand.

(not (!))

OP1	NOT (!)
0	1
1	0

→ 0 → false
1 → true

eg:-

```
int x=12;
int y=60;
sop (!((a+b)/8 != y));  => true
sop (!((a+b)%8 != 0));  => true
sop (!(a+b%8 != 0));  => false
```

* **CONDITIONAL OPERATOR:-**

Syntax:-

OP1 ? OP2 : OP3
true false

→ Conditional operator is a ternary operator where operand one should be a boolean operand.

→ If operand one is true then operand two will be executed otherwise operand three will be executed.

eg:-

```
sop (true ? "Monday" : "Friday");  => Monday
sop (12+3>20 ? 12+10*3 : 180-30*4);  => 60
String x=sop (false ? "shiv" : "Shankar");  => Shankar
```

Date / /
Page No.

* **Program for ODD and Even:-**

```
import java.util.Scanner
class OddEven
{
  public static void main (String [] args)
  {
    Scanner sc = new Scanner (System.in);
    int n = sc.nextInt ();
    String x = (n%2 == 0) ? "Even no" : "Odd no";
    System.out.println (x); }
  }
```

* Write a java program to take an input and print whether the last digit of the number is 6 or not.

```
import java.util.Scanner
Class Lastdigit
{
  public static void main (String [] args)
  {
    Scanner sc= new Scanner (System.in);
    System.out.println ("Enter the no:");
    int n= sc.nextInt ();
    String x= (n%10==6) ? "Last digit is 6" : "Last digit is not 6";

    System.out.println (x);
  }
}
```

Date / /
Page No.

* Write a Java program to take an input and print whether last digit of the number is even or last digit of the number is odd.

```
import Java.util.Scanner;
~~Class C~~
class LastOddEven
{
    Scanner sc= new Scanner(System.in);
    System.out.println("Enter the no:");
    int n =sc.nextInt();
    ~~int n=~~ int p= n%10;
    String x= (p%2==0)?"Last digit is even":
                        "Last digit is odd";

    Sop(x);
  }
}
```

* WAJP to check whether the number entered by user is a three digit number or not a three digit number.

```
import Java.util.Scanner;
class DigitNumber
{
  public static void main(String[] args)
  {
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter the no:");
    int n= sc.nextInt();
    String x= (n>=100||n<=999)?"The number is of
                three digit":"The number is of not
                three digit";
    Sop(x);
  }
}
```

Date / /
Page No.

* Valid triangle or not?

```
import Java.util.Scanner
class ValidTriangle
{
  public static void main(String[] args){
```

* Flow Control Statement :-
→

Logic for second big no between three no.

```
int big = (a>b)?(a>c?a:c):(b>c?b:c);
int Secondbig = (big==a)?(b>c?b:c):((big==b)?(a>c?a:c):(a>b?a:b));
```

→ Flow control statement is used to change the flow of execution of a program.

They are : 'if statement, if-else statement, else-if statement and nested if-else statement'.

Date / /
Page No.

* If Statement:-

→ If statement is used to check a condition and when condition is provided true the block will be executed otherwise if block will not be executed.

Syntax:-

```
                false
if (condition) true
{
    // if block statement
}
// outside if block statement
```

eg:-

```
class IfState
{
    public static void main (String [] args)
    {
        System.out.println("Main code start");
        if (true){
                System.out.println(" if block code");
                }
        if (false){
                System.out.println("Second if block code");
                }
        System.out.println("End of program");
    }
}
```

Date / /
Page No.

* If-Else Statement:-

→ If-else statement is used to check a condition and when condition is found true then if block will execute otherwise when condition is found false then else block will execute.

Syntax:-

```
                false
if (condition){ true
        // If block code
        }
else {
        // Else block code
        }
// outside of if-else statement code
```

eg:-

```
class IfElseStatement
{
    public static void main (String [] args)
    {
        System.out.println("Main code start");
        if (age>18){ age=25;
                System.out.println("Eligible for vote");
                }
        else {
                System.out.println("Not eligible for vote");
                }
        System.out.println("End of code");
    }
}
```

Date / /
Page No.

* WASP to take three items price from user if total price is greater than or equal to 10000 then there should be 40% discount otherwise there will be 10% discount and ask user how much to pay.

Program:-

```
import java.util.Scanner
class ShoppingDisc
{
  int price1, price2, price3, price;
  Scanner sc = new Scanner(System.in);
  System.out.println("Enter first item price: ");
  price1 = sc.nextInt();
  System.out.println("Enter Second item price: ");
  price2 = sc.nextInt();
  System.out.println("Enter Third item price: ");
  price3 = sc.nextInt();
  price = price1 + price2 + price3
  if (price >= 10000){
        price = (price*40)/100 + price;
        System.out.println("Your total price
        is greater or equal to 10000 so you get
        40% discount so you have to pay
        only amount: " + price);
        }

  else{
        price = price - (price*10)/100;
        System.out.println("Your total price is less than
        10000 so you get only 10% discount therefore
        you have to pay only amount: " + price);
        }
  }
}
```

Date / /
Page No.

* IF-ELSE-IF STATEMENT:-

→ Else-if statement is used to check multiple condition and when a condition found true then a particular block will execute.

→ If no condition will be found true then else block executes.

Syntax:-

```
if (condition1)
{
    //Statement of if block 1
}
else if (condition2)
{
    // Statement of if block2
}
else if (condition3)
{
    //Statement of if block 3
}
else
{
  // Statement of else block
}
```

* Take 4 input from user and print the biggest number by using else-if.

Program:-

```
import java.util.Scanner;
class BiggestNo
{
    public static void main(String [] args)
    {
        Scanner sc = new Scanner(System.in);
```

```
System.out.println("Enter first digit: ");
int a = sc.nextInt();
System.out.println("Enter the Second digit:");
int b = sc.nextInt();
System.out.println("Enter third digit:");
int c = sc.nextInt();
System.out.println("Enter forth digit: ");
int d = sc.nextInt();
if (a=b && b=c && c=d){
                System.out.println("All are equal");
                }

else if (a>b && a>c && a>d)
{
  System.out.println("The biggest no is a: "+a);
}
elseif (b>c && b>d)
{
  System.out.println("The biggest no is b: "+b);
}
else if (c>d)
{
  System.out.println("The biggest no is c: "+c);
}
else
{
  System.out.println("The biggest no is d: "+d);
}
}
```

* WAJP to take 3 ~~digits~~ input from user and print second biggest.

Program:-

```
import java.util.Scanner;
class Second_Biggest
{
  public static void main(String[] args)
  {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter first digit: ");
      int a = sc.nextInt();
    System.out.println("Enter Second digit: ");
      int b = sc.nextInt();
    System.out.println("Enter third digit: ");
      int c = sc.nextInt();
    if (a==b && b==c){System.out.println("All are equal");
    else if (a>b && a>c)
      { int d = b>c ? b:c;
        System.out.println("Second biggest is: "+d); }
    else if (b>c && b>a)
      { int d = c>a ? c:a
        System.out.println("Second biggest is: "+c);}
    else { System.out.println(c>a ? a:b);}
    }}
```

* NESTED IF STATEMENT:-

→ If one statement inside another if statement then it is called nested if.

Syntax:-

```
if(condition){
        if(condition2){ [ ; ; ]
        else } //else body }}
else { //else body }
```

Date / /
Page No.

* Math class in Java:-

Syntax:- Math.function_Name (value);

→ There are many method in math class.
Math class are stored in Java.lang package.

Function_Name	Syntax
Power	Math.pow (a,b)
Square Root	Math.sqrt (a)
Maximum	Math.max (a,b)
Minimum	Math.min (a,b)
Ceil value	Math.ceil (a)
Floor value	Math.floor (a)
Exponential value	Math.exp (a)
Logrithmic value	Math.log (a)
Trigonometric value	Math.sin (a)
Trigonometric inverse value	Math.sinh (a)

* WAJP to take character input and output and print alphabet, number, character or special character.

Program:-

```
import Java.util.Scanner;
class Keychecker
{
    public static void main (String [] args)
    {
        Scanner sc = new Scanner (System.in);
        char p = sc.next().charAt(0);
        if (p>='A' && p<='Z' || p>='a' && p<='z')
        {
            System.out.("This is an alphabet");
        }
```

Date / /
Page No.

```
        else if (p>='0' && p<='9')
        {
            System.out.("It is number");
        }
        else
        {
            System.out.("It is special character");
        }
    }
}
```

* Loop :-

→ Loop is a block of instruction which executes repetedly until the loop condition will be false.

→ Loop is of two types:-

(i) Entry Control Loop;
(ii) Exit Control Loop.

* Entry Control Loop:-

→ If the loop condition is at entry point and after loop condition there is loop body then it is called entry control loop.

It is of two type:-

(i) While Loop;
(ii) For Loop.

* Exit Control Loop:-

→ If the loop condition is at exit point but before loop condition there is loop body then it is called exit control loop

(i) Do-While Loop.

Date / /
Page No.

* While Loop:-

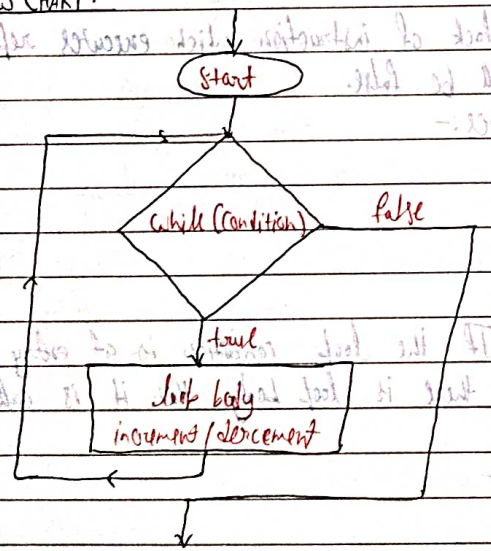
→ While ~~for~~ loop is an entry control loop which execute repetedly untill while condition will be false.

Syntax:-

```
while (condition)
{
    // loop body
    // increment / decrement
}
// outside loop statement
```

(incre / decre) true false

Flow Chart:-



* eg:-

```
class Print_Serial_Number
{
    public static void main (String [] args)
    {
        System.out.println ("Main method Start");
        int i = 1;
        while (i<=10)
        {
            System.out.println("Java Program " +i);
            i++;
        }
        System.out.println ("Program End");
    }
}
```

Date / /
Page No.

* Programs:-

1→ WAJP to print all the even numbers upto 100.
2→ WAJP to print all the numbers upto 200 which is divisible by 7.
3→ WAJP to print all the numbers upto 200 which ends with 7.
4→ WAJP to print all the numbers upto 200 which is divided by 7 or ends with 7.
5→ WAJP to print all the numbers upto 1000 which is divided by 7 and also ends 7
6→ WAJP to count how many numbers are upto 100 which is divided by 7 and ends with 7
7→ WAJP to take an input from user and print all the factors of the number (number itself is not includ
8→ WAJP to take an input and print how many factors of that number (number itself is not includ
9→ WAJP to take an input and print whether the given number is a prime number or not.
10→ WAJP to take an input and print how many digits are in that number.
11→ WAJP to take an input from user and print only even digits of the number.
12→ WAJPTTI and print that is it's of the number which is greater or equals to five.
13→ WAJPTTI and print how many digits in the number is even.
14→ WAJPTTI and print factorial of that number.
15→ WAJPTTI and print power of that number.

Date / /
Page No.

* Do-While Loop :-

→ Do-while loop is exit control loop where first body loop executes and then condition is checked at ending (exit) point.

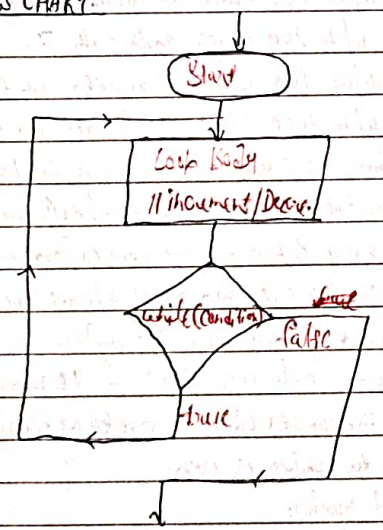
→ In do-while-loop the first execution of takes place without any condition.

→ From Second execution loop will be executed as per the condition.

Syntax:-

```
do{
    // loop body
    // increment/decrement
  } while (condition);
// outside do-while loop
```

Flow Chart:-



Date / /
Page No.

Ex:- WAP to calculate power of number.

Program:-

```
import java.util.Scanner
class PowerOfNumber
{
    public static void main(String[] args)
    { int pow=1; int x;
      Scanner sc= new Scanner(System.in);
      System.out.println("Enter the the values");
      do{
        System.out.println("Enter the number: ");
        int a= sc.nextInt();
        System.out.println("Enter the power: ");
        int b= sc.nextInt();
      do{ pow = pow * a;
        i++;} while(i<=b);
      System.out.println("The power of "+a+"to"+b+" is: "+pow);
      System.out.println("Enter 1 for other number power or press
                          any other number to exit");
      x= sc.nextInt();
      } while(x==1);
```

* For Loop:-

→ For loop is an entry control loop which executes until the condition of for loop will be true.

Syntax:-

```
for (initialization; condition; Inc/Dec)
{
  // loop body
}
```

eg:-
```
for (int i=1; i<=10; i++)
{ sop(i); }
```

* INFINITE LOOP :-

(i)
```
while (true)
{
}
```

(ii)
```
do {
} while (true);
```

(iii)
```
for (; ;)
{
}
```

(iv)
```
for (int i=1; ; i++)
{
}
```

(v)
```
for (int i=1; true; i++)
{
}
```

* BREAK KEYWORD :- Break is a keyword which is used to terminate a loop or a switch case block.

→ ~~With JVM~~ When JVM reads break keyword then immediately control comes out of JVM comes out of the block.

Syntax:-
```
{
break;
}
```

eg:-
```
for(int i=1; ; i++)
{
if(i>=15){ break; } sop("Java program:"+i);
}
```

* WASP to take three input from user and print LCM of the three number.

Program:-
```
import Java.util.Scanner;
class LCMOFDigit
{ public static void main (String[] args)
  {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the first two number: ");
    int a = sc.nextInt();
    System.out.println("Enter the second number: ");
    int b = sc.nextInt();
    System.out.println("Enter the third number: ");
    int c = sc.nextInt();
    for (int i=1; i<0; i++)
    {
      if(i%a==0 && i%b==0 && i%c==0)
      {
        System.out.println("The LCM of "+a+","+b+"and"+c+"is: "+i);
        break;
      }
    }
  }
}
int big = (a>b?a:b)>c?(a>b?a:b):c;
for (int i=0; ; i+=big)
{
```

* CONTINUE KEYWORD :-

→ Continue is a keyword which is used inside the loop to skip some of the execution from loop.

→ When JVM reads continue keyword then immediately JVM goes for next execution of loop.

Syntax:- { continue; }

* WAJP to print all the numbers upto 5000 which does not contain 4.

* **SWITCH CASE:-**

→ Switch case statement is used to match switch value with case value.

→ When a case value will match with switch value then the particular case statement executes and after that switch case will terminated by using break keyword.

→ If switch value will not match with any case value then default statement executes.

Syntax:-

```
Switch(value/expression)
{
  case value1:
        // case statement
        // break;
  case value2:
        // case statement
        // break;
  ::::
  default:
        // default statement
}
// outside switch case;
```

Points:- We can pass only int, string or character type value at switch.

→ Switch type value and case type value should be same type.

→ Every case should have unique values.

→ We can pass only one value at case.

→ Break keyword is not mandatory but if break keyword is not used then multiple case statement will get executed once switch and case will match.

* **METHODS:-**

→ A method in Java is a block of instruction which is used to perform some specific task.

Points:- A method in Java is a class member so it should be design directly inside class block.

→ We can design multiple methods inside a class either before main method or after main method.

ex:-

```
class Programs
{
  public static void test()
  {
    System.out.println("This is test method");
  }
  public static void main(String[] args)
  { System.out.println("This is main method"); }
  public static void demo()
  {
    System.out.println("This is demo method");
  }
}
```

→ A method only executes when it is called by it's method signature.

→ We can call a method again and again repeatedly.

→ We can call programatically any method but the main method only call by JVM at runtime.

→ A static method can be called directly inside same class but in other class it will be called by class name as reference.

→ A non-static method recall only using object reference either in same class or in different class.

→ Any method can call other method but it will not execute untill it is called from main method.

* Structure of a method :-

→

PUBLIC STATIC VOID MAIN (STRING[] ARGS)

- PUBLIC → access modifier
  - → Public (Least Restricted)
  - → Private (Most Restricted)
  - → default
  - → Protected
- STATIC → non-static modifier
  - → Static
  - → Final
  - → Abstract
  - → Synchronized
  - → Native
  - → Strictfp
- VOID → Return type
  - → int
  - → Boolean
  - → char
  - → String
  - → Long, etc
- MAIN → Method Name
- (STRING[] ARGS) → Parameter/Argument

(i) Access Modifier :-

→ Access modifier is used to set the accessibity or visibility to a method.

→ It can be public, private, protected or default.

(ii) Non-access Modifier :-

→ Non-access modifier of a method is used to change the behaviour of a method.

→ It can be static, final, abstract, synchronized, native, strictfp.

(iii) Void :-

In Java a method can return any type of value either primitive or non-primitive.

→ If a method does not return anything then it's return type should void.

(iv) Method Name :-

→ A method should be given name of the convention of the method.

→

(v) Parameter/Argument :-

→ A method can accept different argument at it's parameter

→ It is doesn't take any argument then it should be the blank.

* Syntax Of a Method :-

Method-declaration :-

access-modifier non-accessmodifier return-type method-Name (Argument/Parameter){

// method body

}

Method Signature :-

→ Method name along with it's argument is called method signature.

Method Declaration :-

→ Access modifier, non-accessmodifier, return type along with method signature is called method declaration.

→ Method Defination :-

→ Method declaration along with method body is called method defination.

Date / /
Page No.

* Types of Method :-

→ Method is of two type :-

(i) No-Argument Method,
(ii) Parameterized Method.

* No-Argument Method :-

→ If a method does not take any argument and it's parameter then it is called no-argument method.

Example :-

```
Public static void test ()
{
}
```

No-argument method

* Parameterized Method :-

→ If a method takes different argument in it's parameters then it is called a parameterized method.

Syntax :-

```
access_modifier non-access_modifier return_type method_Name (datatype var1,2,-)
{
  //method body
}
```

Example :- Method design :-

```
Public static void test (int x, int y, double z, String b)
{
}
```

arg1 arg2 arg3 arg4

Formal Argument

Parameterized Method

Method Call :-

```
test ();  //ERROR
test (10, 35, 8.5, "Mohan");
```

Actual Argument

Date / /
Page No.

Formal Arguments :-

→ The argument which are declared at the time of method design is called formal arguments.

Actual Arguments :-

→ Arguments which are passed at the time of method call ~~design~~ is called actual argument.

Compile time binding :-

→ At the time of compilation compiler checks whether arguments at method ~~both~~ design and method call matches or not.

→ If actual arguments and formal argument will match then binding will be successful otherwise will give an error.

→ This process in Java is also called early binding.

* Return Statement :-

→ In Java a method can return any type of value either primitive or non-primitive.

→ If a method doses not return anything then it's return type should be 'void'.

Points :-

→ To return a value from method we have to declared the datatype at method declaration.

→ To return a value from method we have to use return keyword at the last line of method.

→ The return statement must be the last statement. If anything is written after return statement then it will give error called unreachable statement.

→ The return statement is also called control transfer statement.

→ When JVM reads return statement then JVM has to go back to from the method with return value.

→ One method can return only one value at a time. We can't return multiple value from a method.

example:-

```
class Program1
{
    public static void main (String [] args)
    {
        int a = 32;
        Test();
    }
    public static int Test()
    {
        int b = 40;
        int c = 2;
        return c*b;
    }
```

→ If a method has return type then we can store the returned value in some type of variable.

→ We can call a method directly inside System.out.println only if the return type of the method is not void.

```
{ System.out.println (Test()); }
```

void type not allowed

* WAJP to check whether a number is armstrong number or not.

Program:-

```
import Java.util.Scanner;
class CheckArmstrong
{
    public static void main (String[] args)
    {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        if (findSum(n) == n)
        {
            System.out.println (n + " is Armstrong Number");
        }
        else {
            System.out.println (n + " is not a Armstrong Number");
        }
    }
    public static int findSum (int n)
    {
        int sum = 0;
        int x = CountDigit (n);
        while (n > 0)
        {
            int rem = rem % 10;
            sum += findPower (rem, x);
            n /= 10;
        }
        return sum;
    }
    public static int CountDigit (int n)
    {
        int count = 0;
        while (n > 0)
        {
            count++;
```

```
        n = n/10;
        }
    return count;
    }
public static int findPower(int a, int b)
{   int pow = 1;
    for(int i=1; i<=b; i++)
    {
        pow *= a;
    }
    return pow;
}
```

* Object Oriented Programming [OOPS]:-

→ Object Oriented programming is a mechanism to solve real time problem with the help of concept class and object.

The four main pillers of object oriented programming is:

(i) Encapsulation,
(ii) Inheritance,
(iii) Polymorphism,
(iv) Abstraction.

```
Employ e1 = new Employ();
```

non-primitive data → Employ; return variable → e1; object → new Employ()

non-primitive always returns to an object.

```
class EMPLOY
{
    String name;
    int eid;
    double salary;
}
```

Employ is not primitive data

* - CLASS:-

→ Class in Java is a blueprint or master copy or prototype from which an object is created.

→ Class is a logical entity which will not have any physical instance

→ Class in Java is a non-primitive type data which does not occupy any space. Memory allocation is not needed in class.

→ By using one class we can create multiple objects.

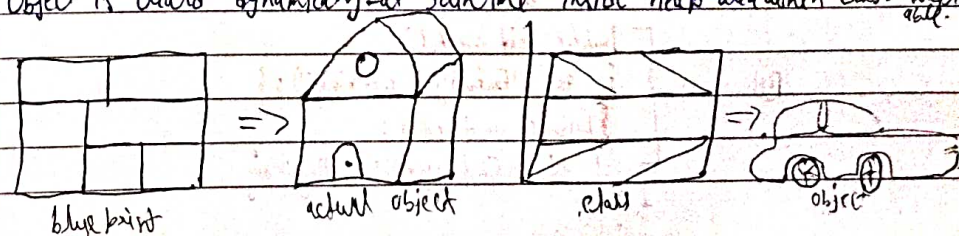
* OBJECT:-

→ Object in Java is the physical implementation of a class

→ Object is a real world entity which requires memory allocation.

→ Whenever an object is created then memory is provided to the object inside heap area. Object will have all property defined by class.

→ Object is created dynamically at runtime inside heap area when class exists.



Date / /
Page No.

* **Characteristics of a class:-**
→ Every class in Java will have two type of characteristics such as:-
(i) States,
(ii) Behaviours.

* **States:-**
→ States of a class is description ~~on~~ or details about a class.
→ State ~~fatty~~ forms global variable of a class.
Example:- name, price, height, color, age, salary, address, etc.

* **Behaviours:-**
→ Behaviours are the actions perform by the object.
→ Behaviours will form methods of class.
Example:- start(), stop(), drive(), work, eat(), run(), bark(), jump(), sleep(), etc.

* **Design a class/Blueprint for emplay:-**
→

```
Class Emplay
{
    String name;
    int eid;
    double salary;
    int age;

    public void work()
    { sop ("Emplay is working"); }
    public void eat()
    { sop ("Employee is eating"); }
}
```

States → name, eid, salary, age
Behaviours → work(), eat()

Date / /
Page No.

HW

* Design a class for:- Laptop, Car, Browser, Weapon.

CW

* **GLOBAL VARIABLE:-**
→ If a variable is directly declare inside class block then such variable is called a global variable.
Global variable is of two types:-
(i) Static Variables,
(ii) Non-Static Variables / Instance Variables

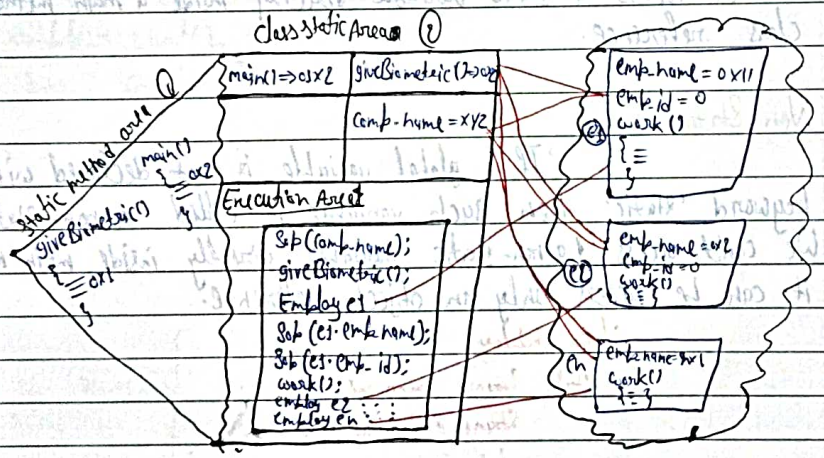
* **Static Variables:-**
→ If a variable declared with the keyword 'static' then it is called static variable.
→ We can access a static variable directly inside a main method or on class reference.

* **Non-Static Variable:-**
→ If a global variable is not declared with the keyword 'static' then such variable is called a non-static variable.
→ We can't access a non-static variable directly inside main method, it can be access only on object reference.

```
Class Emplay
{ static String companyname = "xyz";
    String emp_name;
    int eid;
    public static void giveBiometric()
    { sop ("Emplay should give biometric"); }
    public void work()
    { sop ("Emplay is working"); }
```

Date / /
Page No.

```
public static void main (String [] args)
{   set (comp_name);
    set (emp.compname);
    Employ.giveBiometric();
    Employ e1 = new Employ();
    set (e1.emp_name);
    set (e1.eid);
    e1.work;
    Employ e2 = new Employ();
    Employ e3 = new Employ();
    :
    Employ en = new Employ();
}
}
```



Date / /
Page No.

* Loading Process when a program starts Execution :-

→ When a program starts execution then first all the static methods are loaded in ~~class~~ static method area.

→ Once all static methods are loaded then next all the static members are loaded in class static area.

→ After that main method starts it's execution in execution area.

→ If any object is created inside main method then object memory allocation takes place in heap area.

* Difference between Stack Area and Heap Area :-

Stack Area	Heap Area
• Main methods get executed in stack area.	• Object gets created in heap area dynamically at run time.
• All static members are loaded in class static area.	• All non-static members are loaded in heap area.
• This area is fixed in size	• This area is dynamic in size
• Stack area member will provide faster execution	• Heap area members provide slower execution.

* Important points about static and non-static members :-

→ A static method can access any other static member directly.

→ A non-static method can access any other non-static member directly and

we don't have to create object.

→ All the details of static members are always shared with objects. So a non-static method can access details of any static members directly and we don't have to creat object.

→ A static method can not access any non-static members directly. It can be accessed only through object reference.

* Object Creation in Java :-

```
class Employ
{
    String name;
    int eid;
    Public void work()
    { sop("Employ is working");}
    Public Static void main (String[] args)
    {
        Employ e1 = new Employ();
    }
}
```

new is a keyword which creates an employ object in heap area.

Constructor call for employ class which finds all the non-static members inside object

```
Employ e1 = new Employ();
```

- Employ → non-primitive data-type of employ.
- e1 → variable name or reference variable
- new Employ() → object / Instance of Employ class

→ In Java object is created with the help of new keyword and constructor of a the class.

NEW Keyword :-

→ new is a key word in Java which is used to create an empty object inside heap area.

→ Whenever new key word will be used then a new object is created inside heap area and object is provided with a memory address in hexadecimal form.

Constructor :-

→ A constructor is used to load and initialize ~~of~~ all the non-static member inside object.

→ The name of a constructor must be same as the class name.

* Initiallization of an object :-

→ Assigning value to the non-static member inside object is called initiallization of an object. We can initialize an object in the following ways:

(i) By using reference of the object,
(ii) By using a non-static method,
(iii) By using constructor.

* By using reference of the object :-

→ Whenever an object is created then all the details of the object is stored on object reference.

→ We can use this reference to initialize the object.

Example :-

```
Mobile m1 = new Mobile();
m1.name = "Samsung";
```

```
m1.price = 23456;
m1.color = "blue";
Mobile m2 = new Mobile();
m2.name = ~~23567~~ "Nokia";
m2.price = "34567;
m2.color = "black";
```

Initialize by reference of object

Homework

1. Design a class for rectangle and print it's area and perimeter.
2. Design a cuboid and print it's volume and it's total surface area.

* By using Non-Static Method :-

→ Whenever an object is created then all the non-static members are loaded inside object.

→ We can use a parameterized non-static method to initialize the object as below.

```
→ class Car
  { String name;
    double price;
    String color;
    Public void setValue(String n, int p, String c);
    { name = c;
      price = p;
      color = c; }
    Public void displayCar()
    { Sop("Car name is:"+ name);
      Sop("Car price is:"+ price);
      Sop("Car color is:"+ color);
    }
  }
```

```
Class CarDriver
{ Public static void main(String [] args);
  { Car c1 = new Car();
    c1.setValue("maruti", 28567, "white");
    Car c2 = new Car();
    c2.setValue("Honda" 34567.0, "Red");
    c1.displayCar();
    c2.displayCar();
  }
}
```

* Variable Hiding :-

→ If a local variable and global variable exists with the same name then preference is given to a local variable and global variable becomes hidden which is called variable hiding.

→ If variable hiding is for a static member then we can use class reference to access ~~the~~ a global static variable.

→ If variable hiding is called a non-static member then we can use 'this' keyword to access global non-static variable.

Home Work

1. Design class for mobile, ~~zoo~~ fruit and vehicle where set and display the details by using non-static method.

* This Keyword :-

→ 'this' is keyword which is used to refer global non-static member.

→ this keyword holds the reference of current object under execution.

* Initialization by using Constructor :-

→ Constructor is a block of instruction which is used to load and initialize all the non-static members inside the object.

Difference between Method and Constructor :-

→ A constructor will have name same as the class name but method name can be any thing.

→ A constructor will never have return type but the method have always return type.

Date / /
Page No.

→ A constructor will never be static, final, abstract or synchronized but a method can be static, final, abstract or synchronized.

→ A constructor as well as method will have access modifier.

→ A constructor is called only when object is created together with 'new' keyword but a method can be called directly or on class reference or on object reference.

→ If in a class there is no constructor then compiler adds a constructor which called default constructor. But in Java there is no default method added by compiler.

* **Types of Constructor:-**

Constructor
- → Default Constructor
- → User Defined Constructor
  - → No-argument C.
  - → Parameterized C.

Default Constructor:-

→ When there will be no constructor designed by programmer in Java file then at the time of compilation compiler adds a constructor by itself in class file which is called default constructor.

→ A default constructor added only if there will be no constructor defined by user.

* **Advantages of default Constructor:-**

→ With the help of default constructor we can create object even incase there is no user defined constructor designed by programmer.

User-Defined Constructor:-

→ A constructor designed by programmer is called user-defined constructor.

Date / /
Page No.

→ **No-argument Constructor:-**

→ If a user-defined constructor takes no-argument at it's parameter then it is called a no-argument constructor.

Ex:-

```
class Car{
    String name;
    Car(){System.out.("Car object created");}
}
```

↓ No-argument constructor

```
class CarDriver{
    public static void main(String[] arg)
    { Car C1 = new Car();   // car object created
      Car C2 = new Car();   // car object created.
    }
}
```

* **Parameterized Constructor:-**

→ A user-defined constructor will take different types of argument at it's parameter then it is called parameterized constructor.

→ A parameterized constructor is used to ~~study~~ initialize the non-static members of an object.

```
class Fruit{
    String name;
    int price;
    double weight;
    Fruit (String name, int price, double weight)
    { this.name = name;
      this.price = price;
      this.weight = weight;
    }
```

```
class FruitDriver{
    public static void main (String[] args)
    { Fruit F1 = new Fruit ("Mango, 120, 8.3);
      Fruit F2 = new Fruit ("Grapes", 90, 3.8);
    }
}
```

Date / /
Page No.

HomeWork:-

1- WAJP to designing a class for employee where there will be details such as name employee id and salary. Initialize this details with the help of parameterized constructor. After initialization display the details of employee ~~thro~~ throw a method. Design one more method to get salary after leave deduction.

* Constructor Overloading:-

→ Constructor overloading is a mechanism to have multiple constructors with different arguments.

→ We can achieve overloading in three ways:-

(a) By changing the number of argument at parameter.

Ex:-

```
Employ (String name)
{ this.name = name; }
Employ (String name, String dob)
{ this.name = name;
  this.dob = dob; }
```

(b) By changing the datatype of argument at parameter.

Ex:-

```
Employ (String name)
{ this.name = name; }
Employ (double salary)
{ this.salary = salary; }
```

(c) By changing the sequence of argument at parameter.

Ex:-

```
Employ (String name, double salary)
{ this.name = name;
  this.salary = salary; }
Employ (double salary, String name)
{ this.salary = salary;
  this.name = name; }
```

Date / /
Page No.

* Multiple constructor with the same datatype are not allowed.

* Constructor Chaining:-

→ Constructor chaining is a process to connect or join one constructor with another constructor.

→ Constructor chaining is achieved via help of 'this()' statement.

this():-

→ this() statement in java is used to call one constructor from another constructor.

Syntax:-

this (variable 1, variable 2, ----);

→ A constructor call must be first statement of constructor so this() statement should be used as the first statement inside the constructor.

HomeWork:-

1→ Design a class of student and initialize the details of student with the help of constructor chaining and after initialization display the details of student by calling display method.

* Copy Constructor:-

→ A constructor which is use to copy the details of another constructor is called copy constructor.

```
class Fruit
{ String name,
  int price,
  double weight
  Fruit () { }
  Fruit (String name, int price, double weight)
  { this.name = name;
    this.price = price;
    this.weight = weight; }
  Fruit (Fruit p)
  { this.name = p.name;
    this.price = p.price;
    this.weight = p.price;
  }
  public void display () {
    Sop (name + "+ name, "+ price + ": price and weight: "+ weight);
  }
}
```

```
class FruitDriver {
  Fruit f1 = new Fruit ("Mango",
                        120, 4);
  Fruit f2 = new Fruit (f1);
  f2.display ();
  f1.display ();
}
```

* Method Overloading :-
OR :- Static Binding :-
OR :- Compile Time Polymorphism :-
OR :- Static Method dispatch :-
OR :- Early Binding :-
OR :- Compile Time Binding :-

→ If there are multiple methods inside a class with same name with different argument then it is called method overloading.

We can achieve method overloading by three ways :-

(a) By changing number of arguments at parameter :-

Ex :-

```
Public static void test (int x)
{ }
Public static void test (int x, int y) { }
```

(b) By changing the datatypes of arguments at parameter :-

Ex :-

```
Public static void test (int x) { }
Public static void test (double x) { }
```

(c) By changing the sequence of datatypes at argument :-

Ex :-

```
Public static void test (int x, double y) { }
Public static void test (double p, int q) { }
```

→ Method overloading is only decided by the method signature where method name must be same but arguments must be different.

→ In method overloading non-access modifier and return type doesn't play any role. It is decided by only method signature.

Homework :-

* Design a class and copy the object details by using copy constructor :-
(i) Shirt, (ii) Laptop

*→ The advantage of method overloading is that it improve code readability and it decreases the complexity of code.

→ If method overloading is used then we don't have to remember name. With the same name, different tasks can be performed.

→ Example:- 'print()' and 'println()' are the example of overloaded methods available ~~at~~ in PrintStream class of Java.

→ It is also an example of polymorphism because in method overloading we can get different output with the same name of method.

→ It is example of compile time polymorphism because the execution of method is decided by compiler at compile time and same method executed by JVM at run time.

* Method Chaining:-

→ Method chaining is a process to connect or join one method with another method.

→ Method chaining is achieved by (.) dot operator.

Syntax:-

```
m1().m2().m3()------.mn();
```

Example:-

```
Class Employ{
String name;
int sal;
int eid;
Employ(){ }
Employ(String name, int sal, int eid)
{ this.name = name;
  this.sal = sal;
  this.eid = eid; }
Public void dPName(String name){ sop(name); }
Public void dPSal(int sal){ sop(sal); }
Public void dPEid(int eid){ sop(eid); }
}
```

```
Class EmployDriver{
public static void main(String[] args)
{ Employ e1 = new Employ("Ram", 5000, 1);
  Employ e2 = new Employ("Hari", 10000, 2);
  e1.dPName().dPEid();
  e2.dPEid().dPSal();
}
```

* Can we overload main method of Java or not?:-

→ Yes we can overload main method where we can have multiple main methods inside a class with different arguments.

→ In this case at run time JVM calls the main method where argument is string array (String[]).

→ Other main methods will only execute when it is called.

* Relationships / Association:-

→ If one class is connected with other class or one object is connected with another object then it is called relationship or association.

→ Relationship is of two types:-

(i) Has-A Relationship.

(ii) Is-A Relationship [Inheritance].

* Has-A Relationship:-

→ If one object is completely or partially dependent on other object then it is called has-a relationship.

Has-A Relationship is of two types:-

(i) Composition

(ii) Aggregation

* Composition:-

→ If Has-A Relationship is a strong type relationship where one object is completely dependent on another object then it is called composition.

→ In composition there will be main object and dependent object.

→ The dependent object will not have any existance without main object.

Ex:- Car Has-A Engine

main object → dependent object.

→ Mobile Has A Screen

Laptop Has A Processor

ORganization Has A CEO

* Aggregation :-

→ If one object is partially dependent on another object then it is called aggregation.

→ In aggregation one object ~~pl~~ is exists without other object but in composition if main object deleted then dependent object will be also deleted.

Ex:-

Student Has-A Address,
Mobile Has-A Cover,
Car Has-A Music Player, etc.

* Explain "system.out.println" :-

→

* Aggregation Program :-

→ Student Has-A Address

```
class Address
{ String city;
  String state;
    int pin;
Address() { }
Address (String city, String state, int pin)
{ this.city = city;
  this.state = state;
  this.pin = pin; }
}
```

```
class Student
{ String name;
  int rollno;
  Address ad = new Address();
  Student (String name, int rollno,
           Address ad)
  { this.name = name;
    this.rollno = rollno;
    this.ad = ad; }
}
```

```
class Driver
{ Public static void main(String[] args)
  { Address a1 = new Address ("Noida", "UP", 201301);
    Student s1 = new Student ("Mohan", 103, a1);
      s1 = null; // Here object will deleted
                 by address will remain same.
    Address a2 = new Address ("Kanpur", "UP", 501307);

    Address a3 = new Address ("Gorakhpur", "UP", 501305);
    Student s2 = new Student ("Sohan", 508, a2);
  }
}
```

* Four Pillars of Object Oriented Programming (OOPs):-

(i) Inheritance (Is-A Relationship) → very strong pillar
(ii) Polymorphism
(iii) Abstraction
(iv) Encapsulation

* Inheritance :-

→ Inheritance in Java is use to represent parent-child relationship where a child class will inherit the properties from a parent class.

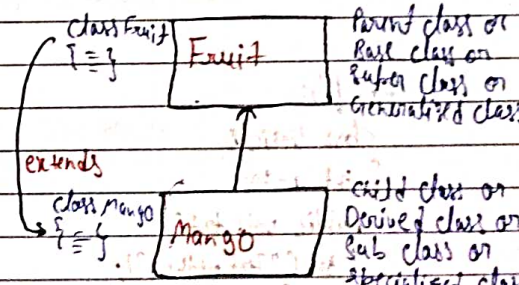
→ Inheritance is also use to represent Is-A-Relationship.

→ Inheritance is unidirectional so only a ~~parent~~ child class inherit the property from a parent class.

→ In Java "extends" keyword is use to achieve inheritance

→ A child class always extends parent class to inherit the properties.

eg:- Mango Is A Fruit,
Car Is A Vehicle,
Chrome Is A Browser,
Roti Is A Food,
Rectangle Is A Shape,
Gun Is A weapon.



→ The main advantage of inheritance of its code reusability

→ Once a class is designed then it can be reuse again and again.

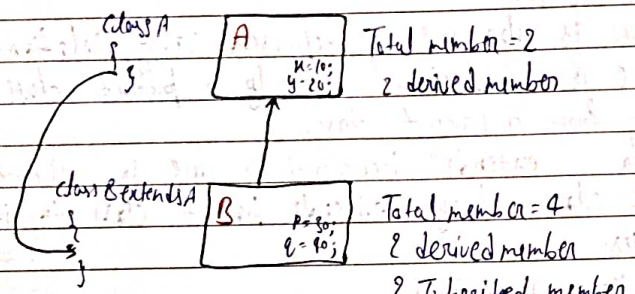
* Types of Inheritance :-

→ Inheritance in Java is of five types:-

(i) Single level Inheritance,
(ii) Multi level Inheritance,
(iii) Hierarchical Inheritance, (हाइरेर्किकल)
(iv) Multiple Inheritance, ✗ not supported in Java with class but supported by Interface.
(v) Hybrid Inheritance

* Single Level Inheritance :-

→ If there are one parent class and one child class then it is called single level inheritance.



Example :-

```
class Vehicle
{ int a=10;
  public void test()
  { sop("Test method");
    sop(a);
    int b=4;
    sop(b); }
}
class Driver
{ public static void main(String[] args)
  { vehicle v1 = new Vehicle();
    sop(v1.a);
    v1.test();
    sop("=========");
    Car c = new Car();
    sop(c1.x);
    sop(c1.a);
    c1.test(); }}
```

```
class Car extends vehicle
{ int x=50;
  public void demo()
  { sop("demo method");
    int y = 60;
    sop(x);
    sop(y);
    sop(a);
    test(); }
}
```

Output :- 10
Test method
40
10
==========
50
10
demo method
60
50
10
Test method
40
10
Test method
40
10

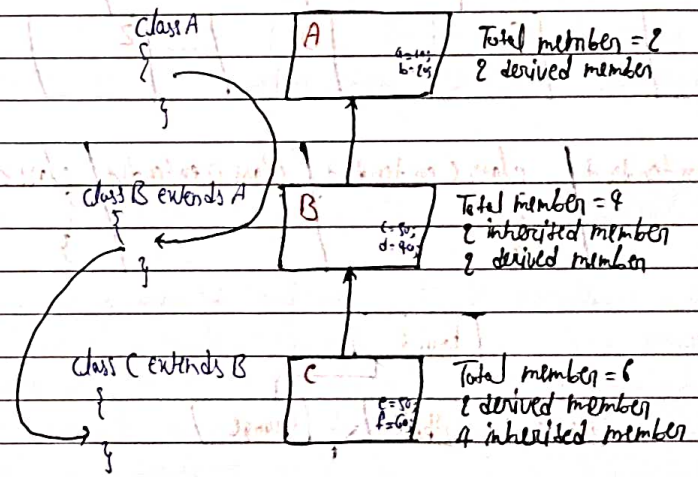
Date
Page No.

* Super Keyword :-

→ In Java "super" is a keyword which is use to refer the global members of immediate parent class.

* Multi Level Inheritance :-

→ If there are a parent class and a child class and child class is parent class another class then it is called multilevel inheritance.



Example :-

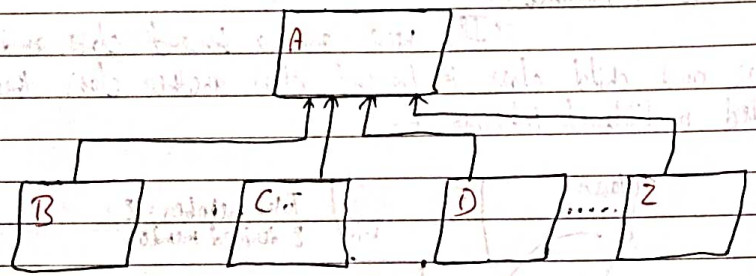
class A	class D extends C	class B extends A	class C extends B
{ int x=20; int b=30; public void xoA() { return x; } }	{ int x=120; int s=140; public void demo(){ sop("demo method"); sop(s); sop(a,b,p,q); sop(this.x); sop(super).x); sop(xoB()); sop(xoA()); } ~~sop(x)~~ }	{ int x=40; int q=50; public void xoB() { return x; } }	{ int x=60; int r=90; }

class Driver
{ Public static void main(String[] args)
{ D d = new D();
d.demo();
}
}

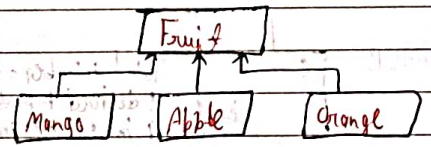
Date / /
Page No.

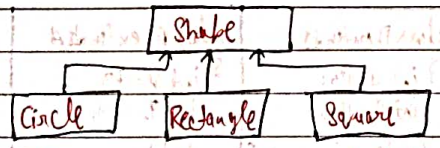
* **Hierarchical Inheritance :-**

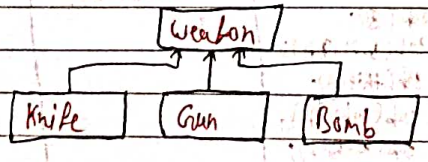
→ When there is a parent class and multiple child class then such inheritance is called hierarchical inheritance.



⇒ class B extends A { } | class C extends A { } | class D extends A { } | class Z extends A { }







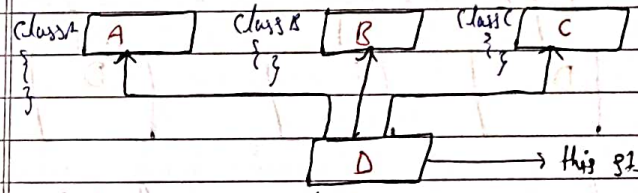
Date / /
Page No.

* **Multiple Inheritance :-**

→ When there is multiple parent class and one child class then it is called multiple inheritance.

→ Java does not support multiple inheritance with class.

→ Multiple inheritance supporting in Java by using interface.



→ this structure is not allowed in Java with class.

class D extends A, B, C { }

Note:- One child class can't have multiple parent class.

* Why Java doesn't support multiple inheritance with class.

Answer:- In case of multiple inheritance JVM comes from across ambiguity problem where it will be not able to decide which parent class member should be executed. Because of this ambiguity problem Java doesn't support multiple inheritance.

→ The problem of this multiple inheritance is also called diamond shape problem in Java.

→ Resolve:- Diamond shape in Java will resolve with the help of interface.

* **Hybrid Inheritance :-**

→ The combination of two or more than two inheritance is called hybrid inheritance.

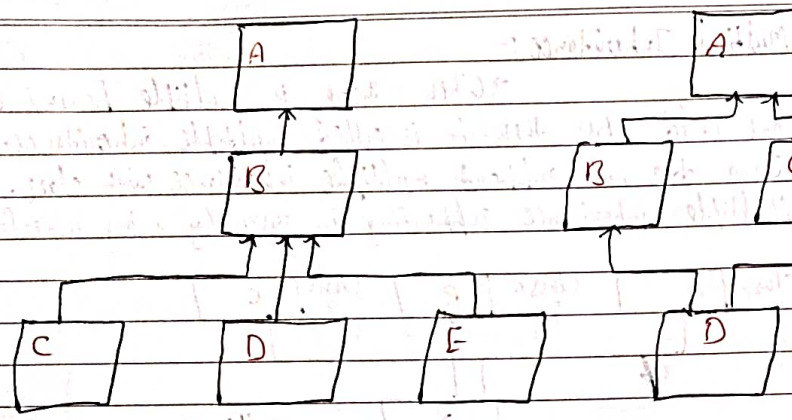


fig:- Hybrid Inheritance

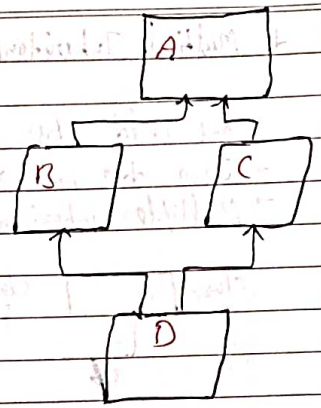


fig:- Hybrid but not allowed

* **Super Call Statement [super()]:-**

→ super() call statement in Java is used to call constructor of immediate parent class.

→ super() statement must be the first statement of constructor.

Syntax:- super(var-name1, var-name2, ----);

* **Why this() and super() statement can't be use together?:-**

→ this() and super() are both the constructor call which must be the first statement of the constructor.

→ If they are used together so one will be first statement and other will be second statement which will give compile time error.

→ So they can not be used together.

* **Non-Primitive type Casting:-**

→ Converting a non-primitive data into another non-primitive data is called non-primitive type casting.

→ In other words when a class type data is converted into another class type data then it is called non-primitive type casting.

→ A class type data will be converted into another class type data only if they will have inheritance or Is-A Relationship.

Non-primitive type casting is two type:-

(a) Upcasting / Generalization.

(b) Down casting / Specialization.

* **Note:-** Object class is the supermost parent class of every class in Java. When ever we design a class then that class by defau extends object class.

```
class Object                    class Car extends Object
{                               {
    //// methods                      // All the // methods of object class will
}                                       be accessible in every class
                                }
```

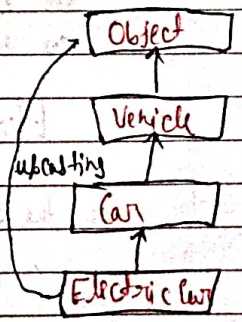
* **Upcasting / Generalization:-**

→ When a child type object is stored on parent reference then it is called upcasting.

→ In otherwords when a child type data is converted into parent type then it is called upcasting.

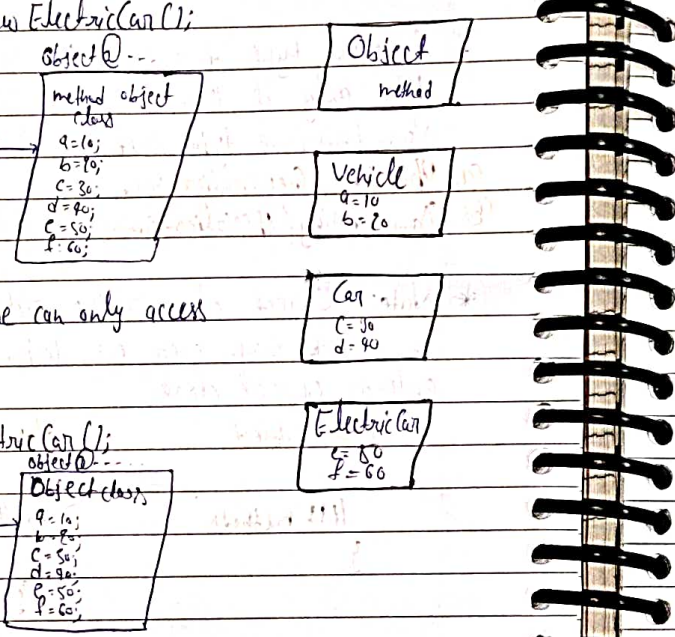
Ex:-

```
ElectricCar e1 = new ElectricCar();
Vehicle v1 = e1;
OR/ Vehicle v1 = new ElectricCar();
    Object o1 = new ElectricCar();
    Object o2 = new Vehicle();
Vehicle ~~Object~~ v1 = new Car();
    Car c1 = new ElectricCar();
```



→ On upcasting we can access the details of only present member we can not access the details of child members.

Object o1 = new ElectricCar();



an object reference we can only access the details of object.

Vehicle V1 = new ElectricCar();

on vehicle reference we can access the details upto electric car.

Car c1 = new ElectricCar();

Here we can access the details up to car class or car reference.

ElectricCar e1 = new ElectricCar();

on the electric car reference we can access all the reference.

public class Vehicle	public class Car extends Vehicle
{ a=10; b=20; }	{ c=30; d=40; }

```
public class ElectricCar extends Car
{ e=50;
  b=60;
}
```

```
public class Driver
{ public static void main (String [] args) {
  Object o1 = new ElectricCar();
  sop ("=====");
  Vehicle v1 = new ElectricCar();
  sop (v1.a);
  sop (v1.b);
  sop ("=====");
  Car c1 = new ElectricCar();
  sop (c1.a);
  sop (c1.b);
  sop (c1.c);
  sop (c1.d);
  sop ("=====");
  ElectricCar e1 = new ElectricCar();
  sop (e1.a);
  sop (e1.b);
  sop (e1.c);
  sop (e1.d);
  sop (e1.e);
  sop (e1.f); }}
```

Date / /
Page No.

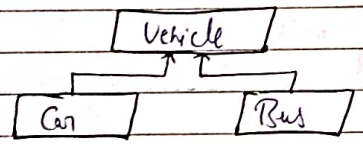
* Down Casting / Specialization:-

→ Down casting is a mechanism to convert a parent type reference into child reference.

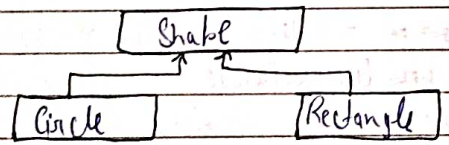
→ In other words if a parent type reference is stored in child reference then it is called down casting.

→ Down casting can not be achieved directly. Down casting is achieved with the help of type cast operator.

Ex:-

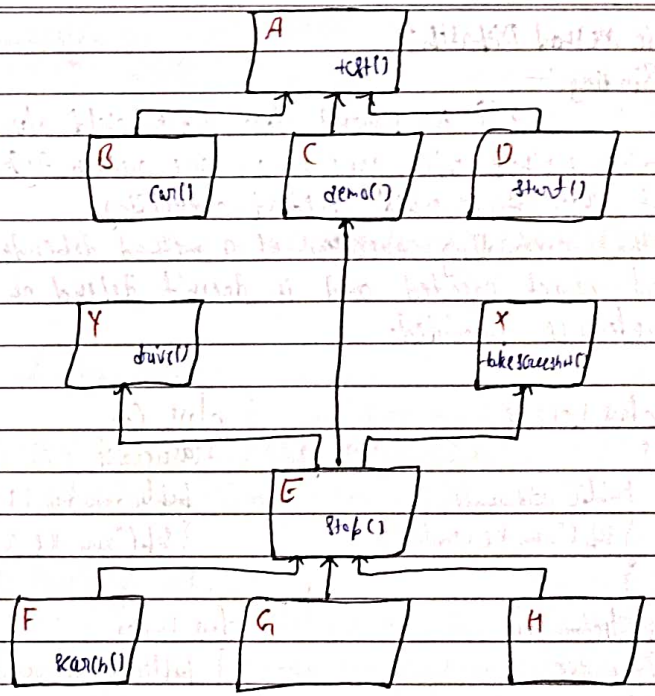



```
Vehicle v1 = new Car();     → up casting
Car c1 = (Car) v1;          → Down casting
Vehicle v2 = new Bus();     → up casting
Bus b1 = (Bus) v2;          → down casting
```

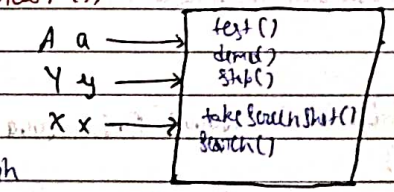



```
Shape s1 = new Circle();        → up casting
Circle c1 = (Circle) s1;        → down casting
Shape s2 = new Rectangle();     → up casting
Rectangle r1 = (Rectangle) s2;  → down casting
```

Date / /
Page No.




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



If I want to use drive method then we have to get reference of Y through downcasting

```
Y y = (Y)a;
X x = (X)a;
```

If we want to take screenshot then we have to generate the reference of X through downcasting.

* Method Overriding / Run Time Binding:-

Run Time Polymorphism:-

Dynamic Polymorphism:-

Date / /
Page No.

Dynamic Method Dispatch :-

Late Binding :-

→ If in parent class and in child class there are non-static method with same name, same return type, same argument then it is called method overriding.

→ In method overriding execution of a method depends on the type of object created and it doesn't depend on the type of reference provided.

Ex:-

```
class Vehicle
{
    public void run()
    { sop("run the vehicle"); }
}

class Electric Car
{ @ override
  public void run()
  { sop("run the electric car"); }
}

class Car
{ @ override
  public void run()
  { sop("run the car"); }
}

class Driver
{ public static void main (String [] args)
  { Vehicle v1 = new Vehicle();
    v1.run();
    Vehicle v2 = new Car();
    v2.run();
    Vehicle v3 = new ElectricCar();
    v3.run();
    Car c1 = new Electric car();
    c1.run();
  }
}
```

Here compiler does the binding → Javac Driver.Java

But there at runtime JVM will change the method binding as per the object created in case of method overriding. → Java Driver

Date / /
Page No.

* Polymorphism :-

→ Polymorphism in Java is a mechanism where a member behaves differently with the same name.

Same variable => Different Value
Same method => Different Implementation

* Types of Polymorphism :-

→ In Java there are two types of polymorphism :-

(i) Compile Time Polymorphism / Static Polymorphism
(ii) Run Time Polymorphism / Dynamic Polymorphism

* Compile Time Polymorphism :-

→ If the execution of a member is decided by compiler at compile time ~~then~~ and same gets executed at run time then it is called compile time polymorphism.

What Compiler Sees/Binds => Same gets executed

Different ways to achieve Compile Time Polymorphism :-

→ Method Overloading,
→ Constructor Overloading,
→ Method Shadowing,
→ Variable Shadowing,
→ Operator Overloading → It is not supported in Java.

* Run Time Polymorphism :-

→ If the execution of a member is decided by JVM at runtime then such polymorphism is called run time polymorphism.

What Compiler Sees/Binds ⇒ Same gets not executed

Different Ways to achieve Run Time Polymorphism :-

→ Method Overriding,

→ Upcasting.

* Method Shadowing :-

→ If in parent class and child class there are static methods with same name, same return type and same argument then it is called method shadowing.

→ In method shadowing execution of a member depends as per the reference given and it does not depend as per the object created.

Ex:-

```
class A                          class B extends A                class C extends B
{ public static void Start()     { public static void Start()     { public static void
  { Sop("Start method of A");}     { Sop("Start method of B");}     Start() { Sop ("
}                                }                                  Start method of C");
                                                                   }}

class Driver
{
public static void main (String [] args)
{
  A a = new B();
  a.Start();   // Start method of A
  A a1 = new C();
  a1.Start();  // Start method of A
  B b1 = new C();
  b1.Start();  // Start method of B
}
}
```

* Variable Shadowing :-

→ If a parent class and a child ~~cl~~ child class there are variables either static or not static with the same name then it is called variable shadowing.

→ In variable shadowing execution of a member or variable will depend as per the reference given. It will not depend as per the object created.

* Why operator overloading is not allowed in Java ?:-

→ In Java at the place of operator overloading we have a more powerful concept called method overloading.

→ If operator overloading was allowed then it could have increase only complexity of the code so Java does not support operator overloading.

* Which type of details from parent class will be non-inherited in child class ?:-

→ If the parent class will have any private members then it will not be inherited in child class.

→ A child class can not inherit the constructors of parent class.

→ If child class will inherit the constructor of parent class there it break the rule of constructor because child class have a constructor with different name. A child class will not inherit the the constructor of parent class but the parent class constructor can be called in child class by using super call statement.

Date / /
Page No.

* Encapsulation :-

→ Encapsulation is a mechanism to restrict direct access of data members and provide indirect and validated access.

→ In otherwords encapsulation is a mechanism to wrap data members and method inside a single class.

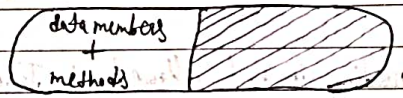


fig:- Encapsulation

* Steps to achieve Encapsulation :-

1→ We have to make all data members 'private'. A private member is visible or accessible only inside the same class but outside the class the private member is not visible so can not be accessible.

2→ To provide the access of private members we have to design public type of 'getters()' and 'setters()' method.

(a) Design of getters() :-

→ A getters() is a public type method which is used to get or read or access the details of private members. For every private data there should be getters() method.

Ex:-

```
public String getName()
{ // validation
  return name; }

public double getSalary()
{ // validation
  return salary; }

public int getEmpId()
{ // validation
  return EmpId; }
```

Date / /
Page No.

(b) Design of setters() :-

→ setters() are public type methods which is used to set or write or update the details of private members. For each private data there should be it's own setters()

Ex:-

```
public void setName (String name)
{ // validation
  this.name = name; }

public void setSalary (double s)
{ // validation
  this.salary = s; }

public void setEmpId (int id)
{ // validation
  this.empId = id; }
```

3→ In an encapsulated class we can design constructor to initialize the object but after that any operation on object will be performed through getters() and setters().

* Data Hiding :-

→ Data hiding is a mechanism to restrict direct access of data members and provide indirect and validated access of data members by using public getters() and setters() method.

* Project Work *

* Create a project of facebook where there should be multiple users and details should be access only through validation.
* Create ATM machine.

Date / /
Page No.

* **Abstraction:-**

→ Abstraction is a mechanism to show the functionality but ~~hide~~ hide the implementation details.

→ In otherwords abstraction is used to show what it does to user and to hide how it does from user.

→ Abstraction is achieved in two ways:-

(i) By using Abstract Class (to achieve 0% to 100% abstraction),

(ii) By using Interface (To achieve 100% abstraction).

* **Abstract Class:-**

If a class in Java is declared by using the keyword 'abstract' then such class is called abstract class.

Ex:-

```
class Vehicle
{
}
```
→ Normal class

```
abstract class Vehicle
{
}
```
→ An abstract class.

* **Property of an Abstract Class:-**

→ If a class is abstract then we can not create object of abstract class.

→ Inside an abstract class we can have concrete method and abstract method.

(a) **abstract method:-** If a method in Java is declared by using abstract modifier then such method is called abstract method. An abstract method do not specify method body.

→ An abstract method is only allowed inside an abstract class.

Ex:-

```
public abstract void test();
```

Date / /
Page No.

* **Note:-** A static method can never be abstract because static method always requires method body.

→ Ex:- `public static abstract void test();` ✗ Error.
illegal combination

→ Only a non-static method will be abstract method.

Ex:- `public abstract void start();` ✓ Allowed

(b) **Concrete Method:-** If a method has method body and it is not declared with the keyword abstract then such method is called a concrete method. A Concrete method can be static or non static method.

Ex:-
```
public void drive()
{ Sop("drive method"); }
```
→ Concrete method

```
* abstract class Vehicle
{
    public abstract void Start();
    public abstract void test();
    public void accelarate()
    { Sop("accelarate the vehicle"); }
    public void stop()
    Sop("stop the vehicle")
}
```
abstract method

Concrete method

$$\% \text{ abstraction} = \frac{\text{The no. of abstract method}}{\text{Total no. of non-static method}}$$

→ An abstract class can be extended by another abstract class.
→ In the end there should be a normal class which will provide body to all the abstract methods inherited.
The normal class will extend abstract class and it will provide body by using the concept method overriding.

* Final Keyword:-
→ 'final' is a keyword in Java which is used to restrict modification of a member.
→ 'final' can be a variable, a method or a class.

* final variable:- If a variable in Java is declared with the keyword 'final' then it is called a final variable.
→ If a variable is final then we can not change or update the value.

Ex:-
```
final int x = 40;
x = 50;   ✗ Error, Reason:- A final variable can't be assigned.

final int y;   // final blank variable
y = 50;   // Only once can be assigned.
y = 60;   // ✗ Error
```

→ If a final variable is only declared then it is called a blank final variable. We can assign a blank final variable only once.

* final method:- If a method in Java is declared with the keyword 'final' then such method is called a final method.

Ex:-
```
public final void test()
{ Sop("test method"); }
```
} final method

* Properties of a final method:-
1→ A final method is inherited by child class.

Ex:-
```
public class A
{ public final void test()
  { Sop("test method for A"); }
}
```

```
class B extends A
{ public void test()
  { Sop("test for B"); }
}
```
✗ Error, Reason:- final method from parent class can't be overridden in child class.

```
class Driver
{ public static void main(String[] args)
  { B b = new B();
    b.test();
  } // test for B
}
```

\# Which type of method can not be overridden from child class?:-
→ A final type method of parent class can not be overridden in child class.

* final class:- If a class in Java is declared with the keyword 'final' then such class is called a final class.

Ex:-
```
final class VariableName
{
}
```
⇒ final class

→ If a class is final then it can not be extended by any other class.
→ The child class of a final class is not possible.

Ex:-
```
class Car extends VariableName
{
}
```
⇒ ✗ Error:- A final class can't be have any child.

Date / /
Page No.

* **Why String class of Java is final class ?:-**

→ String class in Java is a predefined class which contains many methods to perform different task.

→ String class is made final so that no class can extends String class to override or change it's methods.

* **Interface :-**

→ Interface is a mechanism which is used to achieve complete abstraction or 100% abstraction.

→ We can have interface in Java by using the keyword 'interface'.

Ex:-

class Vehicle { //A class }	→ Class	interface Vehicle { //An interface }

⇓

Interface

Properties of Interface :-

1→ We can't create object of an interface.

2→ An Interface can't have constructors.

3→ Inside an interface a variable is always public, static and final type variable.

4→ Inside an interface a non-static method is by default always abstract.

5→ Inside an interface we can have static and default methods since Java 8 update in 2014.

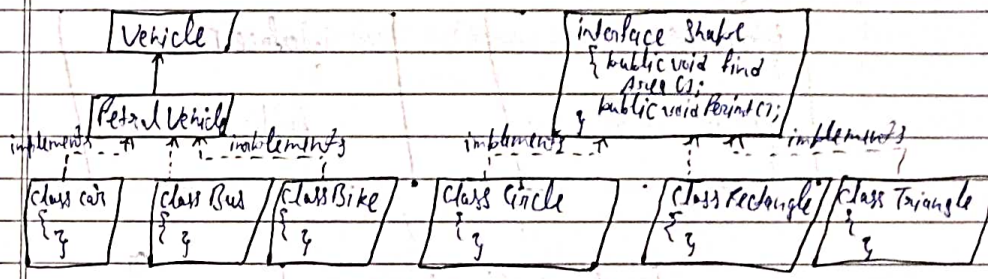
6→ Since Java 9 update we can have private static methods inside an interface.

7→ One interface can be extended by another interface by using the keyword 'extends'.

Date / /
Page No.

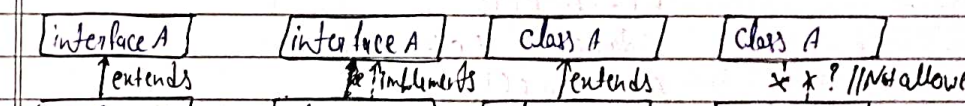
8→ In the end there should be a class which will always implement interface to provide body to all the abstract methods of interf

9→ A class implements interface by using the keyword 'implements' then it will override the abstract method of interface.

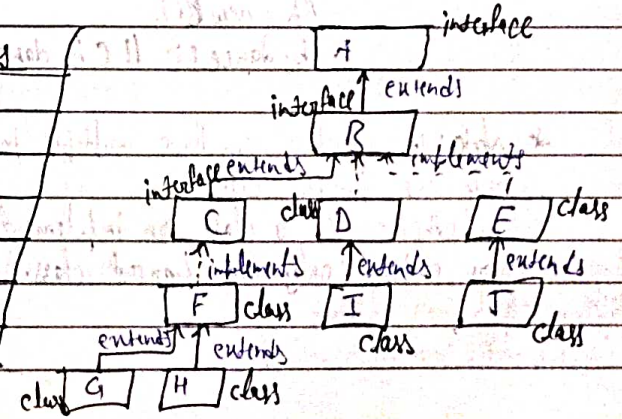



```
interface Vehicle
{
  public void start();
  public void drive();
}
```

```
interface PetrolVehicle extends Vehicle
{
  public void stop();
}
```



Interface	class
A	D
B	E
C	F
	G
	H
	I
	J



Date / /
Page No.

* How multiple inheritance is allowed by Interface in Java? :-

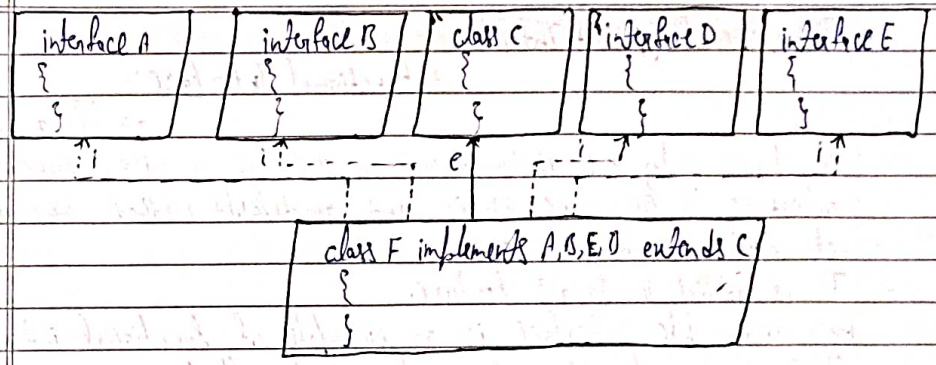
→ With help of interface the ambiguity problem of JVM is resolved so interface supports multiple inheritance.

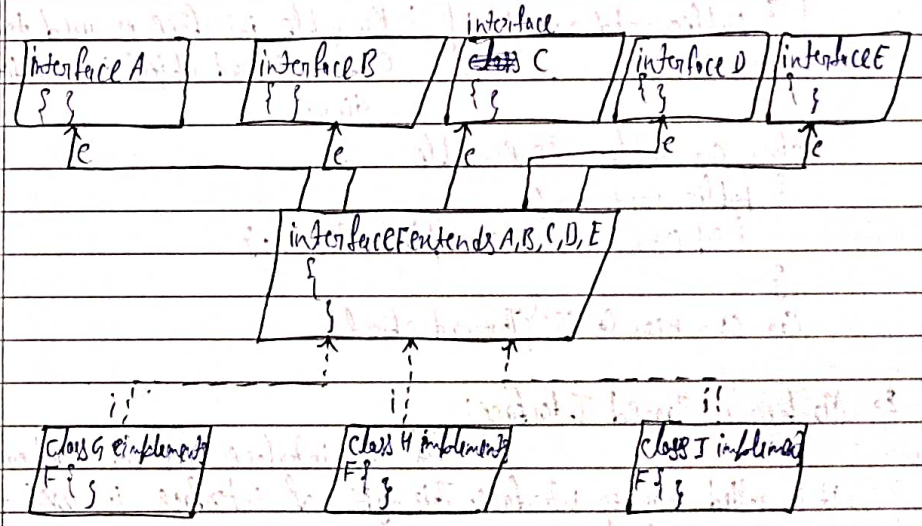
```
interface A
{
    public void dance();
}

interface B
{
    public void dance();
}

class C implements A,B
{
    public void dance()
    { sop("C is dancing");}
}
```

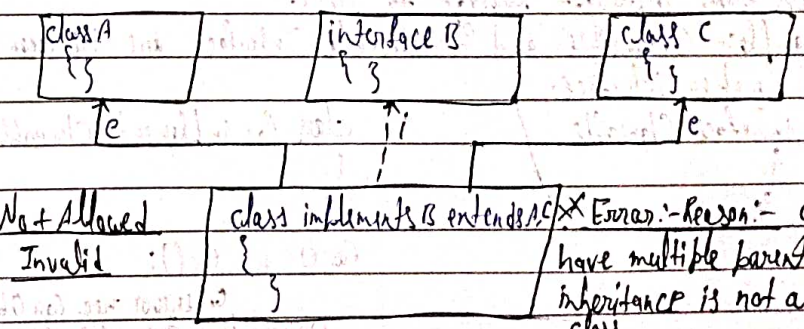
```
C c = new C();
c.dance();  // C is dancing
A a = new C();
a.dance();  // C is dancing
B b = new B();
b.dance();  // C is dancing
```

* Note :- A class can have multiple parent interface but it will have only one parent class.
In other words a class can implements multiple interface but it can extend only one parent class.

Date / /
Page No.







Not Allowed
Invalid

✗ Error :- Reason :- one class can't have multiple parent class multiple inheritance is not allowed with class

Date / /
Page No.

* Some Special Types of Interface :-

1> Functional Interface :-

→ If a interface have only one abstract method then it is called functional interface. It can have static and default methods but abstract method must be one.

→ It is added in Java 8 feature.

Ex:- Runnable Interface is an example of functional interface which have only one abstract method called run.

```
interface Runnable
{ public void run();
}
```

⇒ functional interface

:- Runnable interface is used to create thread object in multithreading.

```
class Car implements Runnable
{ public void run()
  { System.out("run method is implemented"); }
}
Car C1 = new Car();
```
} Thread object.

2> Marker or Tagged Interface :-

→ If an interface has no members then it is called marker interface. This type of interface is used to pass some important message to JVM.

Example:- Cloneable and Serializable Interface are the example of marker interface.

```
interface Cloneable
{
}
```

```
class Car implements Cloneable
{
}
Car C1 = new Car();
```

Whenever new Car Object is created its clone is create at the time of creation.

Date / /
Page No.

* What is difference between class and abstract class :-

Class V/S Abstract Class

→ A class is declared by using keyword 'class' but abstract class is 'class' declared by using 'abstract' keyword

→ We can create object of class but we can't create object of abstract class.

→ A class will have only concrete type method but abstract class can have concrete as well as abstract methods.

→ A class may or may not be extended by another class but an abstract class must be extended by a class to provide body to abstract methods.

* Difference Between class and Interface :-

Class V/S Interface :-

→ A class is declared by using keyword 'class' but an interface is declared by using keyword 'interface'.

→ We can create object of an class but we can't create object of interface.

→ A class can have constructor but an interface can not have construct

→ A class will have concrete method but an interface will have abstract method.

→ A class is extends another class but a class implements interface.

→ Multiple inheritance is not supported by class but it is supported by interface.

Date / /
Page No.

* Exception Handling :-

Exception :-

→ Exception is an unexpected situation and because of exception program gets terminated ~~upon~~ abruptly or suddenly.

→ When there will be exception then program will not maintain the normal flow of execution.

Exception Handling :-

→ Exception Handling is a mechanism by which a program will maintain the normal flow of execution even in case there will exception in the program.

→ Some examples of exceptions :-

1→ ArithmaticException,
2→ StringIndexOutOfBoundsException,
3→ ArrayIndexOutOfBoundsException,
4→ NullPointerException,
5→ ClassCastException.

→ Exception is a class in Java which is available in ~~J~~ Java.lang package.

→ The topmost class in exception hierarchy is 'Throwable'.

→ Throwable ~~cl~~ class has two child class which are exception and error.

* What is difference between Exception and Error ? :-

→ Exception and Error are both child class of throwable in Java.lang package.

Date / /
Page No.

→ Exception is a situation which can be handled programmatically.
Ex:- FileNotFound, InterruptedException, SQLException.

→ Error is a situation which can not be handled programmatically.
Ex:- StackOverFlowError, MemoryError, IOError, SQLError.

Exception of Java is of two types :-
(a) Checked Exception / Compiled Time Exception,
(b) Unchecked Exception / Run Time Exception

* Checked Exception OR Compiled Time Exception :-

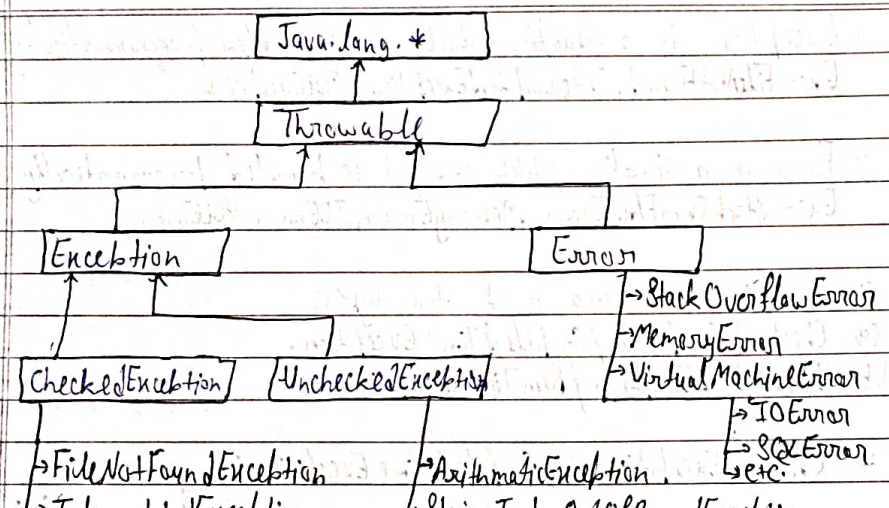
→ If an exception is checked by compiler at compile time whether the exception is handled or not then such exception is called checked exception.

→ If a checked exception is not handled at compile time then it gives an error at compile time.
Ex:- ClassNotFoundException, NoSuchMethodException.

* Unchecked Exception OR Run Time Exception :-

→ If an exception is not checked by compiler at compile time whether it is handled or not then it is called unchecked exception.

→ Unchecked Exception is identified or checked directly at run time by JVM.
Ex:- ClassCastException, NullPointerException.



* How we can Handle Exception in Java ?:-

→ Exception in Java can be handled by two ways:-

(i) By using try catch block,

(ii) By using throws keyword.

* Try-Catch Block:-

→ The try-catch block in Java is used to handle the exception. In try-catch block 'try' and 'catch' are the keyword which are used to handle the exception.

'try' Keyword:- 'try' keyword represents the try block in exception handling.

→ Inside try{} block we should write the statement which may throw exception.

Syntax:-

```
try
{ // try block }
```

→ Whenever an exception will be generated in try block then internally object of exception is thrown from try block. When exception object is thrown from try block then immediatly comes out of try block with the exception object.

The program will get normal flow only when this exception object will be caught by catch block.

Ex:-

```
try
{ sop(10/0);   throw new ArithmeticException();
}
catch (Exception e)
{
}
```

'catch' block:- 'catch' block is used to catch the exception object thrown from 'try' block. If in try block exception will not occure then catch block is not execute.

→ Catch block is only execute when there will be exception in try block

* Points related to try-catch block:-

1→ A try block should be followed by atleast one catch(){} block to catch the exception.

2→ If there will be exception in try block then any of the next statement of

try block will not execute.

3→ We can have multiple catch ()\{block\} with one try\{block\}.

Ex:-
```
try
{ sop (a[n]);
  sop (a[1]/a[3]);
}
catch (ArithmeticException e)
{ sop("It is Arithmetic Exception"); }
catch (ArrayIndexOutOfBoundException e)
{ sop ("Index is out of range"); }
catch (Exception e)
{
  sop ("Other type Exception");
}
```

Child type exception

parent type exception

4→ When multiple catch block is used with one try block then hierarchy must be child type Exception to parent type Exception.

```
try{
    sop (a[n]);
    sop (a[1]/a[2]);
}
catch (Exception e){ }   → parent type exception
Error X catch (ArithmeticException e){ }
Error X catch (ArrayIndexOutOfBoundException e){ }
```

} child type exception

Reason:- this two catch block will be came unreachable.

5→ The details of exception is stored on the exception refrence so we can get the details of exception through the exception reference.

Ex:-
```
try{ sop (a[n]);
     sop (a[1]/a[2]);
}
catch (Exception e)
{ sop (e); }
```

Exception reference

this will point the details about exception

6→ By using exception method 'printStackTrace()' we can get all the details of exception.

Ex:-
```
catch (Exception e)
{ e.printStackTrace(); }
```

This method will give complete hierarchy of exception which has occures.

7→ We can declare multiple exception at one catch ()\{block\} by using pipeline (|).

Ex:- catch (ArithmaticException | NullPointerException e) { }

8→ Inside one program we can have multiple try catch block to catch multiple exceptions.

Ex:-
```
try{ sop (a[n]); }
catch (ArrayIndexOutOfBoundException e){ }
catch (Exception e){ }
try { sop (a[1]/a[2]); }
catch (Exception e) { }
```

'finally' block :- 'finally' is a block which is used with try-catch block.

→ In the end of try-catch block finally block return and it will execute always whether there is exception or no exception.

Ex:-

```
try { }
catch (Exception e) { }
finally {
        // This block always executes
        }
```

→ Use of Finally block :- finally { } block is use for cleaning operation, for closing resource, for closing connections, for closing file connection and for terminating threads.

'throws' keyword :- 'throws' keyword is used to handle the exception through method declaration.

→ By using 'throws' keyword we specially handle the exception of method.

→ When a method gives any exception then it can be handled by 'throws' keyword at method declaration.

→ By using one 'throws' keyword we can delare multiple exceptions.

Ex:- public static void test() throws ArithmaticException, FileNotFoundException

→ ~~Ex~~ When this method is called then it should be called in try-catch block.

Ex:-

```
try
  { test(); }
catch (Exception e) { }
```

'throw' keyword :- By using ~~through~~ k 'throw' keyword we throw the object exception.

Ex:-

```
public static void ~~main~~ test() throws ArithmeticException, Exception
  {
      throw new ArithmeticException();
  }
```

→ By using 'throw' keyword we can throw predefind exception or custome exception.

* Difference Between 'throw' and 'throws' keyword :-

1→ By using 'throw' keyword we throw the exception object but by using throws keyword we declared the exception at method declaration.

2→ One 'throw' keyword can throw only one exception object but by using one 'throws' keyword we can declared multiple exceptions.

3→ # 'throw' keyword is used inside method but 'throws' keyword is used at method declaration.

4→ 'throw' keyword is followed by instance of exception but 'throws' keyword is followed by name of exception.

* User defined Exception :-

→ An exception which is design by programmer is called user defined or custom exception.

Ex:- InsufficientFundException, UnderAgeException

* Steps to create CustomException :-

1→ We have to create the class for which we want to throw exception.

2→ We have to provide all the properties of exception to our class because

we can throw an object exception only when it has properties of exception.

3→ All the custom exception is of checked type exception so we must handle it at compile time at method declaration by using 'throws' keyword.

* How many exceptions can be thrown from try block?:-

→ One try{} block can have possibility for many exception but at a time only one exception can be thrown from a try{} block.

* How many finally{block} can be designed after a try-catch{}{block}?:-

→ Only one finally{block} can be design after one try catch block.

* Can we write any statement between try-catch{block}?:-

→ We can not write any statement between try-catch{block}.

* Will a program run if catch{block} is not used with try{block}?:-

→ If catch{block} is not used with try{block} then in that case try{block} can be designed with finally block.

Ex:- try{}
finally{}

→ This design will only run successfully if there will be no exception. If there will be any exception then finally{block} can not catch that exception so program will not maintain the normal flow.

* If there will be any exception in catch{block} then will program execute normally?:-

→ If catch{block} gives any exception then finally{block} will execute finally but other statement after finally{block} will not execute so program will not maintain the normal flow.

* Is it possible that finally{block} will not execute completely in som scenario?:-

→ In some scenarios finally{block} may not execute completely:-

(i)→ If system.exit(1) will execute before finally{block} then finally{block} will not execute because system.exit(1) immediatly terminates the execution of the program.

(ii)→ If there will be exception in finally{block} itself then finally{block} will not execute completely.

* Object class:-

(a) What is Object class?
(b) Explain some methods of Object class.
(c) Explain the difference between final, finally and finalize().

→ Object class is the supermost parent class of every class in Java. Whenever we design a class then by default it extends to object class of Java.

Ex:-

```
class Object
{
    // 11 methods
}
```

```
class Car extends Object
{
    // 11 methods of object class is
    inherited by car class
}
```

Date / /
Page No.

* Some important methods of Object class :-

(a) Public String toString();
(b) public native int hashCode();
(c) public boolean equals(Object);
(d) protected void finalize();
(e) public final native getClass();

* public String toString() :- 'toString()' is method available in Object class in Java. We can override toString() to get the useful information about the object.

Ex:- @Override
public String toString()
{ return "Name: "+name+"\t Engine Id: "+ennot+"Price: "+price;

* public native int hashCode() :- 'hashCode()' is a method of object class. Whenever an object is created then internally object is provided with any unique integer number which is called hashcode.
→ We can get this code by calling the method hashCode().
→ We can Override hashCode() to get some important result.

* public final native getClass() :- By using this method we can get the details about class. This method is a final method so we can not override this.

* protected void finalize() :- 'finalize()' is method in Object class in Java. Whenever an Object is created then it will have many connections such as system connection, database connection, network connection, etc.
When any object is assigned 'null' then garbage collector takes responsibility to

Date / /
Page No.

delete that object.
→ Before deleting that Object, garbage collector internally calls finalize() to close all the connections of object so that a program should run smooth.
→ We can call garbage collector programatically at system.gc().

System.gc()
↑
Here only garbage collector calls finalize method.

→ We can override finalize() to provide the logic that which should be detached.

* public boolean equals(Object) :- 'equals(Object)' method is available in object class of Java which is used to compare internal details of two objects.
→ We can override equals(Object) method to give the logic as per which details we want to compare.

* What is difference between (==) and equals(Object) method :-

→ (==) :- This is a relational operator which is used to compare two primitive values.

→ equals(Object) :- It is a method of object class which is used to compare two non-primitive data or two objects internal details.

* 'Static' Keyword :-
Static is a keyword which is used to memory management purpose.

A static member is loaded in class static area and it is shared with all the objects. Inside same class an static member can be access directly and inside a different class we can access the static member by class name as reference.

→ static can be a variable, a method, a block/static initializer block, static class.

Static Variable:- If a global variable in Java is declared static then it is called a static variable.

Ex:-

```
public class Program1 { static int x=12;
{ public static void main(String[] args)
  { sop(x);      // 12
    x=92;
    sop(x);      // 92
    x=60;
    sop(x);      // 60
    Program1.x=120;
    sop(x);      // 60
    sop(Program1.x);  // 120
  }
}
```

Static method:- If a method is declared with the static keyword then it is called a static method.

A static method is loaded in class static area only once and shared with all the objects. Inside same class we can call a static method directly but inside different class we can call a static method by using class name as reference.

Static block / Static Initializer block:- If a block in Java is declared by using keyword 'static' then such block is called static block.

→ A static block is loaded in class static area and it gets executed at the same time.

→ A static block will get executed before main method.

→ Static block is also used to initialize static members so it is called static initializer block.

Ex:-

```
static
{
  // static block
}
```

→ Inside a class we can have multiple static block and it will get executed in order, it is written inside the program.

Static class:- If a class in Java is declared by using the keyword 'static' then such class is called a static class.

Ex:-

```
class A          // outer class
{
  class B        // Inner class
  {
    // only an inner class can be static
  }
}
```

→ An outer class can never be a static class only two non-access modifier :- 'abstract' and 'final' is allowed for an outer class.

→ A static class will be only an inner class.

Ex:-

```
class A
{   static class B
    {
      //Static class
    }
}
```

→ Static class or inner class is used to achieve higher level security.

Ex:-

```
class A
{ static int x=30;
  static class B
  { static int y=40;}
}
```

```
class Driver
{ public static void main(String[] Args)
  { sop(A.x);
    sop(A.B.x);
  }
```

* **Non-Static Block:-** If a block in Java is not declared with static then such block is called non-static block.

Syntax:-

```
{
  //Non-Static block
}
```

→ A non static block is loaded and executed only one when object is created. It will get executed at that time when it will be loaded inside the object.

Note:- A static block loaded and executed only once per multiple objects.

A non-static block will load and execute every time when object is created.

Execution Sequence for Different type of members:-

1→ All static block gets executed then→

2→ All non-static block will get executed.

3→ Now constructor will execute it's own statement.

4→

Constructors with parent child Relation:- A child class constructor always calls parent class no-argument constructor internally by using super() call statement.

→ User can change this call by making super() call explicitely.

Ex:-

```
class A
{ A()
  { sop("A no-arg constructor");}
  A(int x)
  {sop("A parameterized Constructor");}
}
```

```
class B extends A
{ B()
  { super();   Implicite call
    sop("B no-arg constructor");}
  B(int x)
  { super();   Implicite call
    sop("B parameterized Constructor
  }
}
```

```
{ B b = new B();}
```

Other Code:-

```
class A
{ A(){sop("A const.");}
  {sop("A non-static");}
  static{sop("A static");}
}
```

```
class B extends A
{ B(){sop("B const.");}
  {sop("B non-static");}
  static{sop("B static");}
}
```

```
class C extends B
{ C(){sop("C const.");}
  {sop("C non-static");}
  static{sop("C static");}
}
```

```
class Driver
{
public static void main(String[] Args)
{
  C c = new C();
}
}
```

Output:- A static
B static
C static
} → static block
A non-static
A Const.
B non-static
B Const.
C non-static
C Const.

→ Flow will be parent to child.

* Packages:-

→ Package is a physical folder which contains a group of similar types of files.

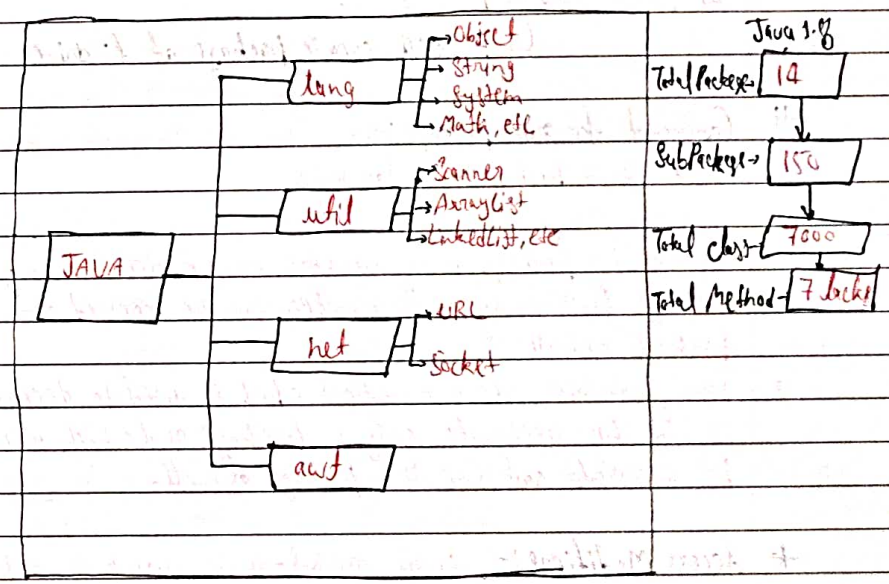
→ The advantage of creating package is:-

(i) Easy access,
(ii) Easy Maintinance,
(iii) Easy debug,
(iv) Organized files, etc.

→ In Java a package is created by using a keyword 'package'.
There are two type of package:- inbuilt package, user defined package.

Inbuilt Package:- Java provides a large amount of package and libraries to perform various type of task.

Ex:- Java.lang, Java.util, Java.io, Java.sql, Java.awt, etc.

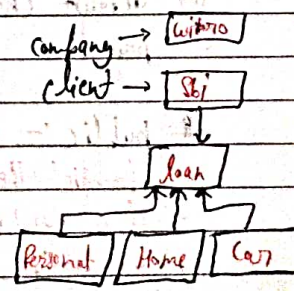


User Defined Package:- A package is designed by programmer is called user-defined package.

→ A package is created by 'package' keyword followed by name of the package.

Ex:- package abc;
keyword — name of package

*→ package com.wipro.sbi.loan.homeloan;
Domain Name — Company Name — Client Name — Project Name — Module Name



Command to compile and create package:-

→ Javac -d . ~~Fok~~ Programs.Java
↳ It will create packag at same location

→ Javac -d F:\\ Programs.Java
↳ It will create package at f: drive

Command to run package:-

→ Java packageName.class Name

→ By creating package many members can be accessed only inside same package but many other member can be accessed out side the package as well.

→ Java provides access-modifiers which is used to decide which member should be accessible only in package and which member should be accessible out side the package as well.

* **Access Modifiers:-** Access modifiers is used to set or change the accessibility of a member.

→ Java provides four access modifiers:-

(i) public, — Less Restricted
(ii) protected,
(iii) default (no keyword)
(iv) private — Most Restricted

public:- If a member in Java is declared by using the keyword 'public' then that member is called a public type members.

→ If a member is public then it can be accessed inside the package or outside the package.

→ public members can be class, variables, method, constructor or interface.

protected:- If a member in Java is declared as protected then it is called a protected member.

→ A protected member is accessible inside the same package and outside the package only by the child class.

→ protected can be variable, methods, constructor.

→ A class or interface can not be protected.

→ 'protected' keyword is not applicable for class or interface.

default:- If a member is not declared by any access modifiers then it is called a default member.

→ A default member is accessible only inside the same package.

→ A default members can be class, variables, method, constructors or interface.

private:- If a member is declared by using keyword 'private' then such member is called a private members.

→ A private member is accessible only inside the same class.

→ It can not be accessed by any outside class.

→ private members can be a variable, method, and constructors.

→ A class or interface can not be private.

Table:-

access modifier	Inside same class	Other class same package	Other package child class	Other package normal class
Public	✓	✓	✓	✓
Protected	✓	✓	✓	X
Default	✓	✓	X	X
Private	✓	X	X	X

Date / /
Page No.

* **Can a constructor be private? If so then what is it's use?:-**
→ Yes a constructor can be private. A private constructor is used to design singleton class.

Singleton Class:- A class for which only one object creation is allowed such class is called a singleton class.

* **Rule for Overriding:-**
→ A child class can override parent class method only if the method in child class will be of same visibility or more visibility.

* **'instanceOf' keyword:-** 'instanceof' keyword is used to check whether the given reference is the instanof given type or not.
→ If it is the instance then it returns true otherwise it will return false.

Ex:-

```
Car c1 = new Car();
sop(c1 instanceof Car); //true
```

given reference — given type

* **'native' keyword:-** 'native' is a keyword which is used to communicate or provide communication of Java code with other language code (C or C++).
→ When a Java method communicates or implements code of other language then that method should be declared by using keyword 'native'.

Date / /
Page No.

* **'strictfp' keyword:-** 'strictfp' is a keyword in Java which is used to keep scientific values same on different operating system (32 bits/64 bits).
→ 'strictfp' stands strict floating point which can be used eighter with a method or with a class.
→ It is not applicable for a variable.
→ 'strictfp' keyword is now considered outdated.

Ex:-

```
strictfp class Test
{
}
```

```
class Driver
{ public ~~void~~ strictfp void main(String[]a
  {double x = 1.9438e+48;
   double y = 2.6791e+18;
   double z = x+y;
  }
}
```

* **'transient' keyword:-** 'transient' is a keyword which is used in serialization process.
→ transient keyword is used only for a variable.

* **'Volatile' keyword:-** 'volatile' keyword is used in multithreading to allow access to multiple threads.
→ It is used only for a variable if a variable will be declared volatile
→ then thread will get most upto-date value.

* **'enum' keyword:-** 'enum' is a keyword which is used to declare constant values.
→ enum is used to represent a collection of constant values.
→ ~~Ex:-~~ enum should be public and static.

Ex:-
```
public enum Day{MONDAY,TUESDAY}
public enum Month{JANUARY }
```

Date / /
Page No.

→ enum can be declared inside of the class and outside of the class.
→ enum is similar as class but it is used to define only constant values.

Ex:- public enum Day {MONDAY, TUESDAY, WEDNSDAY}
for (Day x : Day.values())
{ sop(x); }

* Difference Between array and Collection Framework:-

→ Array is in fixed in length but collection framework will provide mechanism where it can grow or shrink the size as per the requirement.
→ In array only homogeneous types of elements can be store but in collection framework we can store heterogeneous type of elements.
→ Array does not have any inbuilt support of methods so the code designed by array will increase in length and ~~ent~~ complexity. But the collection framework provide the support many inbuilt methods which helps to keep code simple and more readable.
→ The memory management of array is not good but collection framework provides a very good memory management mechanism.
→ The performance of array is ~~good~~ good but collection framework is slow in performance.

\# To over-come the above mentioned drawbacks of array, java provides collection framework.

Date / /
Page No.

Date / /
Page No.

JAVA

* WRAPPER CLASS :-

→ In Java for every primitive data there is an inbuilt class which is called wrapper class.

→ All the wrapper class is available in Java.lang package.

→ Wrapper class provide many feature which is not available with primitive data.

→ With the help of wrapper class we can convert a primitive data into non-primitive and non-primitive data back into primitive type.

Primitive Data		Wrapper Class
byte	→	Byte
short	→	Short
int	→	Integer
long	→	Long
float	→	Float
double	→	Double
char	→	Character
boolean	→	Boolean

Boxing / Auto Boxing :- Boxing is a process to convert a primitive type data into non-primitive or into wrapper class type. Boxing will be done automatically by compiler so it is also called Auto Boxing.

Example :-

Primitive data is converted ~~into non-primitive or wrapper class~~ into object

```
int x = 385;
Integer y = x;
```
⇒ Boxing

```
double p = 18.597;
Double q = p;
```
⇒ Boxing

Date / /
Page No.

Unboxing/Auto UnBoxing :- Unboxing is a process to convert a non-primitive data back into primitive type. It is done by compiler automatically so it is also called auto-unboxing.

Ex:-

```
Integer a = 598;
int b = a;
```
⇒ Unboxing

```
Double k = 59.356;
double b = k;
```
⇒ Auto-Unboxing

Advantage of Wrapper Class :- → With the help of wrapper class we can convert any data into string type, wrapper class provides toString() method which is used to convert any data into string type.

→ Once data is converted into string then we can apply all the method of string to perform various task.

Ex:→

```
int x = 3897;
String b = Integer.toString(x);
Sop(b.length()); // 4
```

```
double y = 1567.32983;
String k = Double.toString(y);
Sop(k.length()-1); // 9
```

→ With the help of wrapper class we can convert a string value in there original form.

→ Every wrapper class provide parse method which converts string data into it's original form.

Wrapper Class		Method
Byte	→	parseByte()
Integer	→	parseInt()
Double	→	parseDouble()

Date / /
Page No.

Ex:-

```
String s1 = "325";
Integer x = Integer.parseInt(s1);   //No conversion
int y = Integer.parseInt(s1);       // unboxing
```

```
String s1 = "53246";
int x = Integer.parseInt(s1);
```
=> ERROR

Reason:- String can't be cast to Integer.

→ With the help of wrapper class we can perform downcasting to convert object data into it's original form.

Ex:-

```
Object x = 325;
sop((Integer)x + 100);
```

```
Object y = 538.62;
double a = (Double)y;
```
=> Now we can use all properties of double

```
Object z = "Mohan";
String b = (String)z;
```
=> Now we can use all properties of String.

→ When primitive data is not allowed then we have to use wrapper class at the place of primitive data.

* COMMAND LINE ARGUMENT :-

→ Command line argument in Java is a feature which is used to provide arguments to main method from command line.

→ The argument which is passed from command line to main method is accepted by main() in the form of String[]

Ex:- Javac program.Java

Java program mohan sohan 12 18.5

Command line argument in String form which is accepted by String[] args

Date / /
Page No.

* COLLECTION FRAMEWORK :-

→ Collection Framework is a group of many interface and classes in the form of different datastructure implementation to achieve CRUD operations efficently.

→ CRUD operations is used to achieve many functionality such as creation of collection of element, read or access it, write or updated, sort the data, search the data, delete or remove from the data, etc.

→ Collection Frameworks provide two different hierarchy which are:-

(a) Collection Hierarchy,
(b) Map Hierarchy.

Collection Hirarchy :-

→ Collection Hirarchy in Java provide a group of classes and interfaces which are used to store a group of individual objects.

18	Mohan	16.5	true	John345
object	object	object	object	object

collection

Map Hirarchy :-

→ Map Hirarchy in Java provides many classes and interfaces which are used to store key-value pair.

Key	Value
12	Mohan
15	Sohan
6	John
8	Jack
18	David

→ All the classes of collection hirarchy aswell as map hirarchy available

Date / /
Page No.

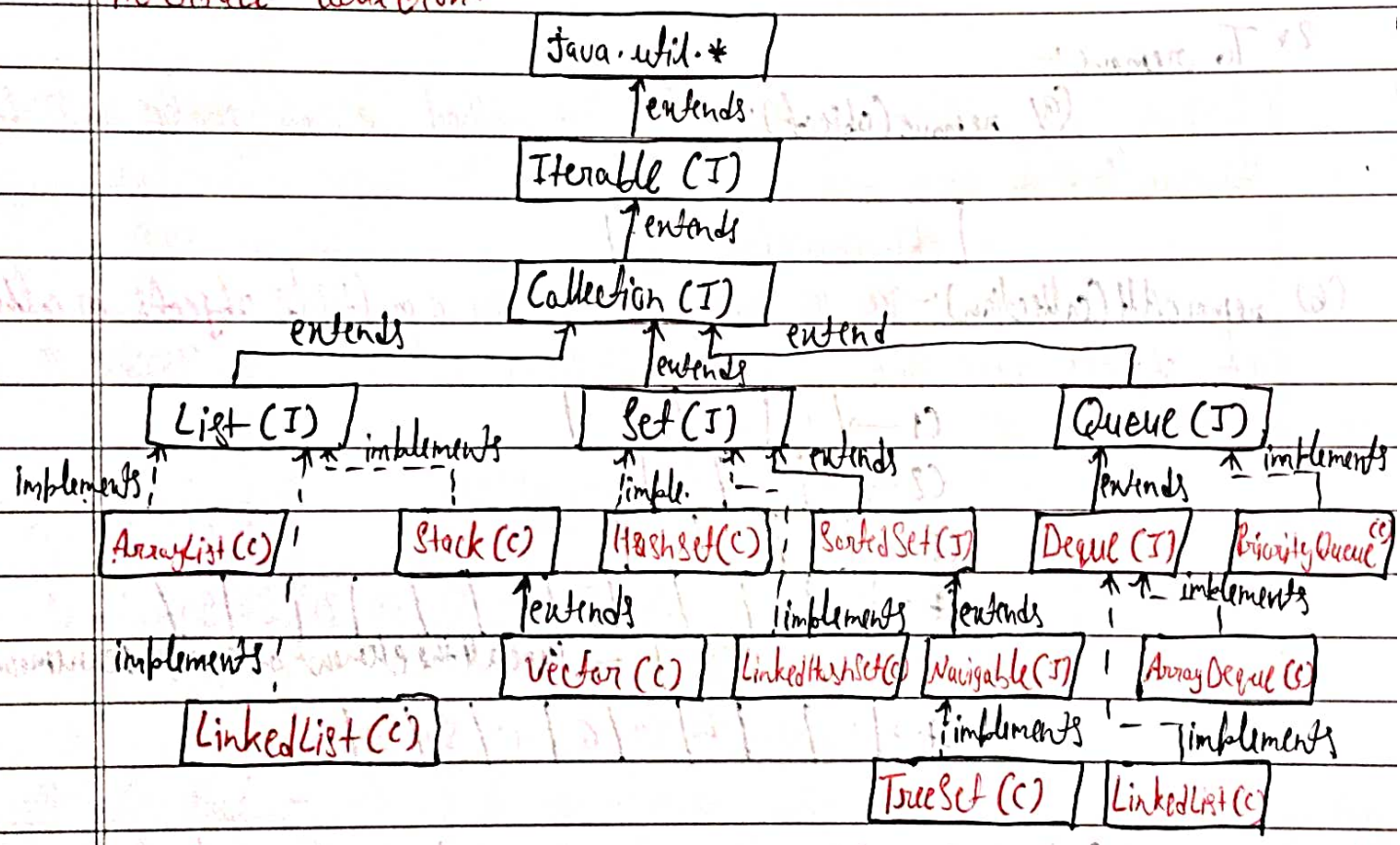
of 'Java.util' package.

→ So to use this classes we must import Java.util package.

* COLLECTION HIRARCHY:-

→ Collection hierarchy provide many classes and interfaces to support different data-structure implementation to achieve CRUD operations efficiently.

Collection Hierarchy is available in 'Java.util' package with the topmost interface 'Collection'.



* Collection Interface:-

→ It is the topmost interface of collection hierarchy.

→ Collection interface provide many methods to perform different types of task.

Date / /
Page No.

* Methods of Collection Interface:-

1. To add:-

(a) add(object):- By this method we can add one object at a time into collection.

```
obj.add(object);
```

(b) addAll(collection):- By this method we can add a collection of elements into another collection.

```
obj.addAll(collection);
```

2. To remove:-

(a) remove(object):- By this method we can remove one object from collection.

```
obj.remove(object);
```

(b) removeAll(collection):- By this method we can remove multiple objects or collection of objects at a time.

C1 → | 10 | 20 | 30 |

C2 → | 40 | 50 | 60 |

C3 → | 70 | 80 | 90 |

C4 → | 100 | 120 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |

```
C4.removeAll(C1)
```
⇒ Here all the element of C1 will be removed.

C4 → | 100 | 120 | 40 | 50 | 60 | 70 | 80 | 90 |

(c) retainAll(collection):- This metho will retain a particular collection and all the other elements will be removed.

```
C4.retainAll(C2);
```
→ Here all the element of C2 will retain and other will be removed.

C4 → | 40 | 50 | 60 |

(d) clear():- By this method a collection will be completely cleared and all the element will be removed.

```
C4.clear();
```
// C4 → [] ⇒ It will return empty array.

Date / /
Page No.

3→ To Search:-

(a) contains(object):- This method will search whether a particular object is available or not. If elements are available the it returns true otherwise it returns false.

```
c1.contains(20);    // true
```

(b) containsAll(Collection):- By this method we can search whether a collection of element are available or not.

```
c4.containsAll(c1);    // true
c4.containsAll(c2);    // false
```

Note:- The return type of all the methods such as add(), addAll(), remove(), removeAll(), is always boolean type it returns true or false.

* Access the element of Collection one by one:-

We can access the elements of a collection one by one in three ways:-

(a) for loop,
(b) foreach loop,
(c) iterator().

for loop:- for loop can not be used by collection because collection does not provide indexing.

→ for loop will used with list where list provide indexing.

for each loop:- We can access the collection elements one by one by using for each loop.

Syntax:-
```
for (datatype varname : Array/CollectionName)
{
    // loop body }
```

Date / /
Page No.

Ex:-
```
for (object x: c1)
{ sop(x); }
```

* Take a collection of elements and print ~~sum~~ sum of all the numbers.

```
for(Object x: c1)
{ sum = sum + (Integer)x; }
sop("Sum is: "+ sum);
```

* WAJPT take user input and search how many times this number is available into collection.

```
int n = sc.nextInt();
int count = 0;
for(Object x: c1)
{ if(n==(Integer)x)
  {
    count++;
  }
}
sop("Total Occurance is: "+ count);
```

* WAJP to take a collection and print how many integer number are available into collection.

```
for (Object x: c1)
{ if(x instanceOf Integer)
  { count++; }}
sop("Integer number is:" count);
```

* Take a collection of elements and count how many string elements are there along with the print length of each string.

```
int count=0;
for(Object x: c1)
{ if(x instanceof String)
```

Date / /
Page No.

```
{ count ++;
  String p = (String)x;
  System.out.println("length of "+x+" is: "+p.length());
  }
}
System.out.println("Total Occurance of String"+count);
```

* Take a collection and remove all the other types of elements ~~of~~ then an integer type.
* Take a collection and :-
(i) print only prime number,
(ii) print palindrom number,
(iii) print only perfect number,
(iv) print only strong number,
(v) print the numbe which has exactly three ~~n~~ digit.

Access through iterator():- iterator is a method which is used to create cursor object by which we can iterate elements one by one.
→ iterator() creates a cursor object inside collection. This cursor object can be used to access collection elements one by one.

Iterator :- Iterator provide some methods which is used to access the collection element.
(a) hasNext():- This method checks whether the element is ~~a~~ available at current position or not. If element is available then it returns true otherwise it return false.

(b) next():- next() method will access the element from current position and after that it will send the cursor to the new position.
→ By Iterator the cursor can be used only in forward direction but ListIterator will provide a cursor which can be used for forward access

Date / /
Page No.

as well as backward.

→ ListIterator is only available for List interface but Iterator is available for all the collection type so the Iterator cursor is also called universal cursor.

Ex:-

```
Iterator itr = c1.iterator();

while (itr.hasNext())
    { sop(itr.next()); }
```

* **Concurrent ModificationException:-** When foreach loop and Iterator is operating on collection then add or remove operation will change the size of collection which will return problem to cursor.
So in that scenario JVM will an exception called 'ConcurrentModificationException'.

* **Other Methods:-**

(i) **size():-** By this method we can get the size of collection.

(ii) **equals(Collection):-** By this method we can compare whether two collection are equals or not.

(iii) **isEmpty():-** By this method we can check whether a collection are empty or not.

(iv) **toArray():-** By this method we can convert a collection into array. The converted array will be the type of object array (Object[]).

Ex:- c1 → | 12 | 18 | Mohan | Sohan | 48.5 |

```
Object [] p = c1.toArray();
```

p → | 12 | 18 | Mohan | Sohan | 48.5 |

```
for (int i=0; i<p.length; i++)
    { sop(p[i]); }
```

Date / /
Page No.

* **Generics:-** generics in Java is parameterized collection.

→ If a collection is declared with datatype as the argument then such collection is called generics.

Syntax:- `Collection <datatype> c1 = new ArrayList <datatype>();`

Ex:-

```
Collection <Integers> c1 = new ArrayList <Integers>();
Collection <String> c2 = new ArrayList <String>();
Collection <Double> c3 = new ArrayList <Double>();
```

← parameterized or Generics

→ Generics in Java introduce in Java 1.5 version.

→ Generics is very much recommended to be used collection because of some important advantages.

Advantage of Generics:-

1→ Generics helps to achieve type safty.

2→ By the use of generics we can identify errors at compile time instead of run time.

3→ By the use of generics objects will be in it's own form so we don't have to perform downcasting operations.

* **LIST:-** List interface is child interface of collection.
List interface has all the properties of collection and along with that it has some special features.

Features of List:-

1→ List has indexing so elements in list we added, removed or searched based on the index.

2→ List allows duplicate values so we can add same value multiple times in the list.

3→ List maintains insertion order so we can access the element in the same order in which it was added.

Date / /
Page No.

4→ List provides a method called get of(Index) by which we can get the element available at the given index.

5→ In list we can add multiple null elements.

Methods of List:-

1→ To add:-

(a) add(int index, Object o):- By this method we can add an element at given index.

(b) addAll(int index, Collection c):- By this method we add a collection from the given index.

2→ To Remove:-

(a) remove(index):- By this method we can remove an element from the given index.

3→ To Search:-

(a) get(index):- By this method we can search that a given element at which index.

(b) indexof(Object):- By this method we can search a given object is available at which index.

4→ Run a loop through List:- In list we can run a normal for loop to access the element one by one by using get of(index) method.

(a)
```
for(Integer x: l1)      => Foreach Loop
{ sop(x); }
```

(b)
```
for(int i=0; i<l1.length(); i++)      => For Loop
{ sop(l1.get(i)); }
```

(c) ~~Ite~~ Iterator:-

→ listIterator():- listIterator() is a method which is used to create a cursor object of ~~li~~ ListIterator. Which can help in accessing

Date / /
Page No.

forward as well as backward direction.

* **Object creation of ListIterator :-**

ListIterator lit = l.listIterator(); object creation of ListIterator.

Methods of ListIterator cursor :-

(i) **hasNext():-** By this method we can check whether the element is available at current position or not

(ii) **next():-** This method will give the element from the current position of cursor and after that it will send the cursor to the next position.

(iii) **hasPrevious():-** This method will check whether the element is available at previous position or not. If there is element it returns true otherwise it will returns false.

(iv) **previous():-** This method will access the elements from previous position of cursor ~~at~~ and after that cursor will go one position backward.

(v) **nextIndex():-** By this method we can get ~~a~~ current position of cursor.

(vi) **previousIndex():-** By this method we can get the previous index of cursor. If there is no previous index then it will return -1.

* **ArrayList :-**

→ ArrayList is a child class of List interface, ArrayList provides implementation to all the abstract methods of List interface.

Properties of ArrayList :-

→ ArrayList internally uses dynamic array to store the array which can grow or shrink.

Date / /
Page No.

→ In ~~array~~ Arraylist insertion order will maintain so element will access in same order ~~when~~ in which it was added.

→ We can add duplicate in Arraylist.

→ We can store multiple null values into Arraylist.

→ In Arraylist all the elements will be stored in continuous block of memory. so searching an element will be very fast by using index.

15	18	16	17	30	40
0	1	2	3	4	5

Continuous block of memory to store elements

→ If our operation will search operation where duplicate elements are allowed then ~~array~~ Arraylist will be recommended.

→ In Arraylist modification operation will be slow because when element is added or removed in between then all the subsequent elements will eighter shift to right side or left side in the array.
This shift operation is slow process so Arraylist is not recommeded if our frequent operation add or remove.

a1 →

10	12	18	30	...	...	40
0	1	2	3			1000

a1.add(1,50);

a1 →

10	50	12	18	30	...	40
0	1	2	3	4		1000

a1.remove(12);

a1 →

10	50	18	30	...	...	40
0	1	2	3			1000

Constructors of Arraylist:-

```
Arraylist a1 = new Arraylist();
```

Date
Page No.

→ By this constructor a default array will created with the default capacity 10.

→ It will grow it's size one by one when elements will be added in a ArrayList.

→ When 70% of a capacity is filled then it is called 'load factor'. At this point a new ~~array~~ ArrayList is created which will be approximatly 50% bigger in size.

→ Once new ArrayList is created then all the elements from old Array List will be copied into new ArrayList and after that old Array List will be deleted.

* <u>Second Constructor</u>:-

→ By this constructor we can declare the size of Array List at the time of ArrayList creation.

→ By providing the size we can avoid the repeated reallocation.

```
ArrayList al = new ArrayList(n size);
```

* <u>Third Constructor</u>:-

By this constructor we can directly pass a collection to ArrayList. In this manner all the elements of other collection will be stored directly into ArrayList.

```
ArrayList al = new ArrayList(Collection c);
```

<u>Conclusion</u>:- ArrayList is best choice where we want duplicate element with very fast search operation.

→ ArrayList is fast in search because it has index and moreover it implements "random access interface" for faster search operation.

→ ArrayList is slow for insertion and deletion operation so ArrayList is not best choice for this requirement.

Date ___/___/___
Page No. _____

* LinkedList :-

→ LinkedList is a class which implements 'List' as well as 'Queue' Interface.

→ LinkedList internally uses node as data structure implementation to store the elements.

Node :-

Node is a block of memory which stores the object value and along with that node address.

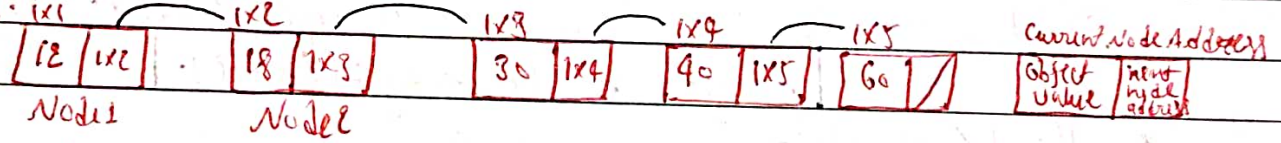
Node is of two types:-

(a) Singly Linked Node

(b) Doubly Linked Node

Singly Linked Node:-

Singly Linked Node is a block of memory which contains object value along with that address of current node and next node.



→ In Singly Linked Node we can traverse the elements only in forward direction

Doubly Linked Node:-

Doubly Linked Node is a block of memory which contains object value along with that current node address, previous node address and next node address.

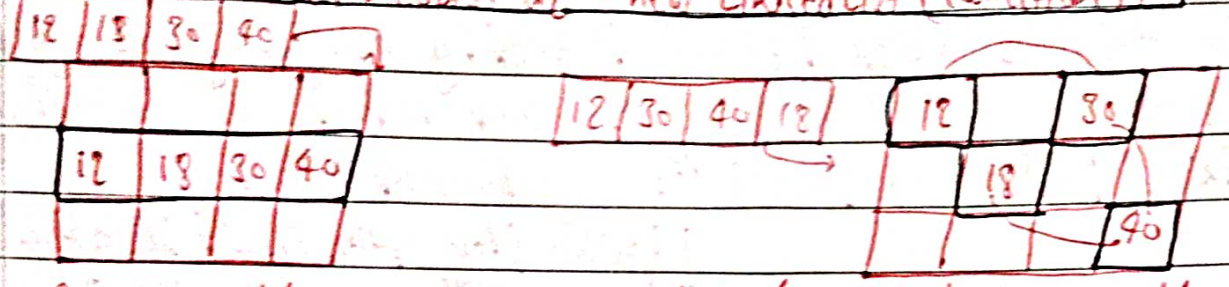
In doubly linked node we can traverse in forward direction as well as in backward direction.

LinkedList internally uses doubly linked node to store the object.

Date
Page No.

Constructors of LinkedList :-

```
LinkedList l1 = new LinkedList();
LinkedList l2 = new LinkedList(collection);
```



ArrayList implements => Continuous Block of Memory.

LinkedList implements => Random Block of Memory.

Features of LinkedList :-

→ LinkedList internally uses doubly linked node to store the object.

→ Duplicates are allowed inside LinkedList.

→ We can traverse LinkedList from both the ends.

→ LinkedList provide fast insertion or deletion operation.

→ LinkedList is slow for search operation because elements are randomly on different memory location.

* Synchronization :-

→ If a resource is being used by only one thread at a time and remaining all thread are in wait then such environment or mechanism is called synchronization.

→ Synchronization is achieved by keyword synchronized on method declaration.

Points :- 1→ In synchronization performance will be slow.

2→ In synchronization thread safty will be achieved.

Ex:-

```
public void synchronized book () { // synchronized method
                                   // ticket booking should be
                                   synchronized }
```

Date / /
Page No.

* **Non-Synchronization:-** If a resource is being allow to be used by multiple thread at a time simultaneously then it is called non-synchronization.

→ In Java, if a method is not declared by using keyword synchronized then such method is non-synchronized method.

Ex:- public void play(){ // non-synchronized method
// hotstar play option should be allowed
// for multiple user at a time}

→ Performance will be fast in non-synchronized environment.

→ Thread safty will not be achieved in non synchronized environment.

* **Vector:-** Vector is a child class of list interface.

→ Vector class is similar as ArrayList with few difference.

→ Vector class also uses dynamic array to store the objects.

* **Difference between ArrayList and Vector:-**

→ ArrayList is not synchronized but Vector is synchronized.

→ ArrayList provide fast performance but Vector is slow in performance.

→ ArrayList does not provide thread safty but Vector provide thread safty.

→ Arraylist grows by 50% in size when load factor is reach but Vector grows by 100%.

→ Vector and stack is lagacy class in Java but Arraylist is not a lagacy class because it was added in Java 1.2 version.

* **Constructors of Vector:-**

```
Vector v1 = new Vector();
```

Date / /
Page No.

→ By this constructor a dynamic array created internally and when 70% of capacity is filled then a new Arraylist will created with double in size.

* **Second Constructor :-**

Vector v2 = new Vector (int initialCapacity);

→ By this constructor we can provide initial capacity to Vector.

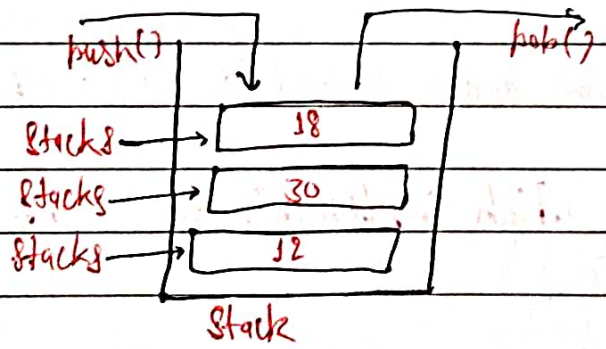
* **Third Constructor :-**

Vector v3 = new Vector (collection);

→ By this constructor we can directly pass any type of collection into Vector.

* **STACK :-** Stack is a class of List interface where a linear data structure is used to store the element the main feature of stack is LIFO (Last In First Out) mechanism.

→ In Stack elements will be always pushed on top of the stack and elements will be removed as well top of the stack.



Special Method of Stack :-

(a) **Push (element e) :-** By this method we can push element top of stack.

(b) **Pop (element e) :-** This method will take out the element from top of the stack.

Date / /
Page No.

(c) peek():- This method will return the element which is available at top of stack without removing it.
(d) empty():- This method will check wether the stack are empty or not.
(e) search(Object o):- this method will search the element from top of the stack. If the element is found then it will give the respective position from top. If element was not found then it will return -1.

Constructor of Stack:-

```
Stack s1 = new Stack();
```

By this constructor an empty stack will be created and it will grow one by one when element will be pushed into stack.

* SET:-

→ Set interface is available in collection framework of Java.

Features of Sets:- Set is a collection of only unique element.

→ Duplicate values are not allowed in set.
→ Set may or may not maintain insertion order of the element
→ Set does not provide index so elements can not be added or removed as per the index.
→ As set does not have index so get(index) method is not available in set.
→ The different class which implements set interface are:-
(a) HashSet,
(b) LinkedHashSet,
(c) TreeSet.

Assignment:- 1→ Store multiple item in a list and access all the item.
2→ Access all the handbag.
3→ Access all the handbag which is of green color.
4→ Access all the item which is kept 'nearby'.
5→ Access all the remote which is of white color.

Date / /
Page No.

HashSet :- HashSet is a class which implements set interface of collection framework.

→ In HashSet elements are store on a mechanism called Hashing.

→ In Hashing Hash function is used to generate unique key for each object internally.

→ Based on the unique key elements will be stored on a table called Hash Table.

→ The default size of HashTable is 16 and it will grow as per the requirement.

	12		
			18
	30		50
40			

Features of HashSet :-

→ In HashSet elements are stored based on a mechanism called Hashing.

→ Only unique elements will be store on HashSet

→ Insertion order of the element will not be maintain in HashSet.

→ HashSet provides very fast search operation.

→ HashSet is not 'synchronized'.

Key	112158	112115	111210		151114
obj, val	18	15	30		40

fig:- Hash Table

Constructor of HashSet :-

1→ HashSet hs1 = new HashSet();

2→ HashSet hs2 = new HashSet(int capacity);

3→ HashSet hs3 = new HashSet(collection);

Date / /
Page No.

* Linked Hash Set:-
* Question:- Remove all the duplicate element from an array.

```
int [] a = {12, 15, 18, 50, 30, 12}
HashSet hs = new Hashset();
for (int x:a)
  { hs.add(x); }
Sop(hs);
Sop int [] b = new int[hs.size()];
Sop (b.length);
Iterator itr = hs.iterator();
for (int i=0; i< b.length; i++)
  { b[i] = (Integer) itr.next(); }
Sop ("Element of b which is unique of a ");
for (int y:b)
  { Sop (y); }
```

* Question:- Take an array and count how many unique elements are in array.
* Take a string and print all the unique characters.
* Take a string and count how many unique elements/characters are in string.
* For array as well as string count and print all the duplicate element/characters

```
String S1 = "abcadbc";
int count = 0;
HashSet hs = new HashSet();
for (int i=0; i< hs.length(); i++)
  { if (hs.add(S1.charAt(i)))
      { count.++; }
  }
```

Date / /
Page No.

* LinkedHashSet:-

→ LinkedHashSet is a class which implements set interfac

Features of LinkedHashSet:-

→ In LinkedHashSet the mechanism is used to store the element is hashing with LinkedList implementation.

→ LinkedHashSet is fast in search operation but slower than HashSet

→ Duplicate is not allowed in LinkedHashSet.

→ Insertion order of the elements are maintained in LinkedHashSet.

→ LinkedHashSet has not synchronized.

Constructors of LinkedHashSet:-

→ `LinkedHashSet lhs = new LinkedHashSet();`

→ `LinkedHashSet lhs = new LinkedHashSet(collection);`

* Tree Set:-

→ TreeSet is class which implements set interface of collecti

Properties of TreeSet:-

→ In TreeSet elements are always stored in ascending order or in sorted manner.

→ Duplicate is not allowed in TreeSet.

→ In TreeSet elements are always stored in sorted manner so we must add similar type of data in TreeSet.

If it is not similar type then it will give an exception called 'class CastException'.

Constructor of Tree Set:-

→ `TreeSet ts = new TreeSet();`

→ `TreeSet ts = new TreeSet(collection);`

Date / /
Page No.

* Question:- Take an array and sort all the elements.

=>
```
int [] a = {15, 23, 12, 45, 67, 29, 56, 89, 28};
TreeSet ts = new TreeSet();
for (int i=0; i< a.length; i++)
{ ts.add(a[i]); }
Sop(ts);
```

* Sort all the characters of a string.

=>
```
String s1 = "tabfreiohnpq"; String s2 = "";
TreeSet<String> ts = new TreeSet<String>();
for (int i=0; i< s1.length(); i++)
{ ts.add(+s1.charAt(i)); }
Sop(ts);
for (Character x: ts)
{ s2 = s2+x; }
Sop(s2);
```

Assignment:-

\# Project 1:- Store ~~search~~ Shirt type object into HashSet with the details of Shirt brand name, size, price, color. Now perform the below operations.

(i) Access all shirts.

(ii) Access all the Shirts with brand name "Arrow";

(iii) Access all the Shirts which is in price range 1500 upto 3000.

(iv) Access all the shirts which is of brand 'LP' and in white color.

(v) Access all the shirts which is of black colour and price less than equals 2500.

\# Project 2:- Store Shirt, T-Shirt, Jeans object into HashSet. Now perform the ~~bt~~ below actions:-

(i) Access all clothes.

Date / /
Page No.

(ii) Access all the shirt with price >= 1800:-
(iii) Access all the T-shirt of green color and size: 42.
(iv) Access all the Jeans which is of Wragler and size: 32 and color: blue and price <= 3000.

* QUEUE:-

→ Queue is an Interface in collection framework the main mechanism of Queue is FIFO (First In First Out).
→ The first element entered will be removed first from the Queue.

* Assignment *

* We have given an alpha-numeric string. Perform below operation on it:
(i) Print all the non-numeric character.
(ii) Print all the numeric character.
(iii) Print the sum of all the numeric character in string.
and String is :- "ab42cd36xy4"

(i) =>

```
String s1 = "ab42cd36xy4";
List<String> l1 = new ArrayList<String>();
for(int i=0; i<s1.length(); i++)
  { ~~s1.add~~ l1.add(s1.charAt(i)); }
for(Character x: l1)
  { ~~sop(x);~~ } if(x<=48 && x>57){ sop(x); } }
```

(ii)

```
String s1 = "ab42cd36xy4";
List<~~Integer~~ Character> l2 = new ArrayList<~~Integer~~ Character>();
for(int i=1; i<=s1.length(); i++)
  { l1.add(s1.charAt(i); }
for(Character x: l1)
  { if(x instanceof Integer){ sop(x); } }
```

Date / /
Page No.

(iii)

```
String s1 = "ab 43 cd 8 d"; int sum=0;
List<~~Integer~~ Character> l1 = new ArrayList<~~Integer~~ Character>();
for(int i=0; i<=s1.length(); i++)
  { l1.add(s1.charAt(i)); }
for(character x : l1){ if(x>=48&&x<=57){
  { sum += ~~(Integer)~~(x-48); }
System.out.(sum is: " + sum);
```

* Take a ~~Sta~~ String input and perform below operations :- s1 = "abc aabca";

(i) Each ~~charact~~ character is how many times in string.

```
String s1 = "abcaabca";
HashSet hs = new HashSet();
for(int i=0; i<s1.length; i++)
  { hs.add(s1.charAt(i)); }
~~int count=0;~~
for(Character x : hs){ int count=0;
  { for(int i=0; i<s1.length(); i++)
      { if(x==s1.charAt(i))
          { count++; }
      } System.out.(" x+" occurance: " + count);
  }
```

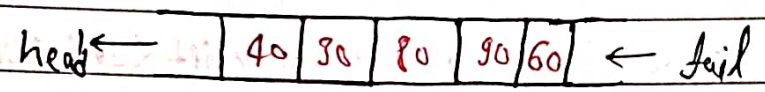
* Take an array and count the occurance of each number.

→

```
int[] a = {12, 34, 45, 12, 20, 34, 12, 34, 20, 60, 34};
Set<Integer> s1 = new HashSet<Integer>();
for(Integer i=0; i<a.length; i++)
  { s1.add(a[i]); }
for(Integer x : s1)
  { if(x==a[i]){ count++; }}
System.out.("Occurance of " + x + " is: " + count);
```

Date / /
Page No.

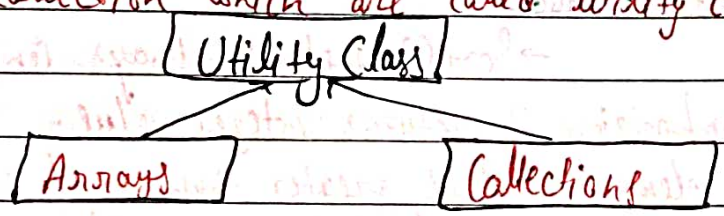
* **Queue:-** To store elements in queue linear data structure is used.
→ Elements in queue always added ~~rate~~ from tail and removed from head.



Linear Data Structure

→ The differente classes which implements queue is:-
(a) LinkedList,
(b) PriorityQueue, etc.

* **Utility Class:-** Java provides some classes which is used to support Array and Collection which are called utility class.



* **Arrays:-** Arrays is a Utility Class which is used to support arrays to perform various type of task such as sorting and searching.

Ex:- Arrays class in 'Java.util' package. So we have to import 'Java.util' package to use it.

```
int[] a = {12, 30, 145, 50, 60}
Arrays.sort(a); // array is now sorted
                [12, 30, 50, 60, 145]
Arrays.toString(a); // it gives string representation of
                    Array.
                    "[12, 30, 50, 60, 145]"
```

Date / /
Page No.

* **Collections:-** Collections is a Utility Class in 'Java.util' package which is use to perform various operation on Collection such as sorting and Searching.

Ex:-

```
List<Integer> l1 = new ArrayList<Integer>();
    l1.add(30);
    l1.add(45);
    l1.add(10);
    l1.add(60);
Collections.sort(l1);
sop(l1); //[30, 30, 45, 60]
```

Working of sort() method:-

→ Sort() method always compares two elements and after comparision it returns integer value.

→ If the first element will be greater than next element then elements will be swaped. Otherwise if they are equal or first element is smaller than next element then elements will not be swaped.

→ Comparator and Comparable Interface uses this mechanism of sort() method to achieve the sorting.

a	>	⇒ +ve	⇒ swapped
	<	⇒ −ve	⇒ not swapped
	==	⇒ 0	⇒ not swapping

ArrayList a →

12	40	10	30	28	45
10	40	12	30	28	45
10	12	40	30	28	45
10	12	30	40	28	45
10	12	30	28	40	45
10	12	28	30	40	45

Date / /
Page No.

* Sorting for a user-defined class on the basis of different details is called custom sorting.
Ex:- sorting on price, roll no., etc.

* Java provide two important interface to achieve custom sorting:-
(a) Comparable Interface,
(b) Comparator Interface.

Comparable Interface:- Comparable Interface is available in 'java.lang' package which is used to achieve custom sorting for user-defined class.

→ By using comparable interface we can achieve single sorting sequence for user-defined class.
Ex:- sorting on age, salary, name or etc.

Structure of Comparable Interface:-

```
public interface Comparable
{ public int compareTo(Object); }
```

To achieve sorting through Comparable we have to implement comparable interface for our class where we want to achieve custom sorting.
The class which implements comparable interface should provide body to the ~~abstr~~ abstract method of comparable interface.

```
class Car implements Comparable<Car>
{ public int compareTo(Car c) { // provide body } ← the returned
}                                                  int value will
                                                   be used by sort()
                                                   to achieve sorting
```

Date / /
Page No.

* Comparator Interface :-

→ Comparator is an interface in Java.util package which is use to achieve multiple sorting sequence for user defined class.

Ex:-
```
interface Comparator <Type>
{
  Public int compare (type t1, type t2);
}
```

→ When a class will implement Comparator Interface then that class must implement the abstract method of comparator and from that method int value will return which will be used by sort ().

Ex:-
```
class Car {
String name;
int carno;
double price;
}
```

```
import Java.util.*;
class NameComparator implements Comparator<Car>
{ public int compare (Car c1, Car c2)
  { return c1.name.compareTo(c2.name); }
}
```

```
import Java.util.*;
class CarnoComparator implements Comparator <Car>
{ public int compare (Car c1, Car c2)
  { return c1.carno - c2.carno; }
}
```

```
import Java.util.*;
class PriceComparator implements Comparator<Car>
{ public int compare (Car c1, Car c2)
  { return (int)(c1.price - c2.price); }
}
```

```
class CarDriver { public static void main(String [] args){
  List l = new ArrayList ();
  { Collections.sort (l, new PriceComparator ());
  Collections.sort(l, new NameComparator ()); }}
```

Date / /
Page No.

* Difference Between Comparable and Comparator Interface:-

→ Comparable interface is in 'Java.lang' package but comparator is in 'Java.util' package.

→ Comparable interface is used to achieve single sorting sequence but comparator is used to achieve multiple sorting sequence.

→ To achieve ~~ata~~ sorting through comparable we need to design one class but to achieve sorting through comparator interface we need to design multiple classes.

→ In Comparable Interface we need to override compareTo() but in Comparator Interface we need to override compare().

* Map Interface:-

→ Map is an interface available in 'Java.util' package which is used to store key and value pair.

Entry →	16	Jack
	32	John
	18	Sohan
key →	12	Mohan ← value

fig: Map

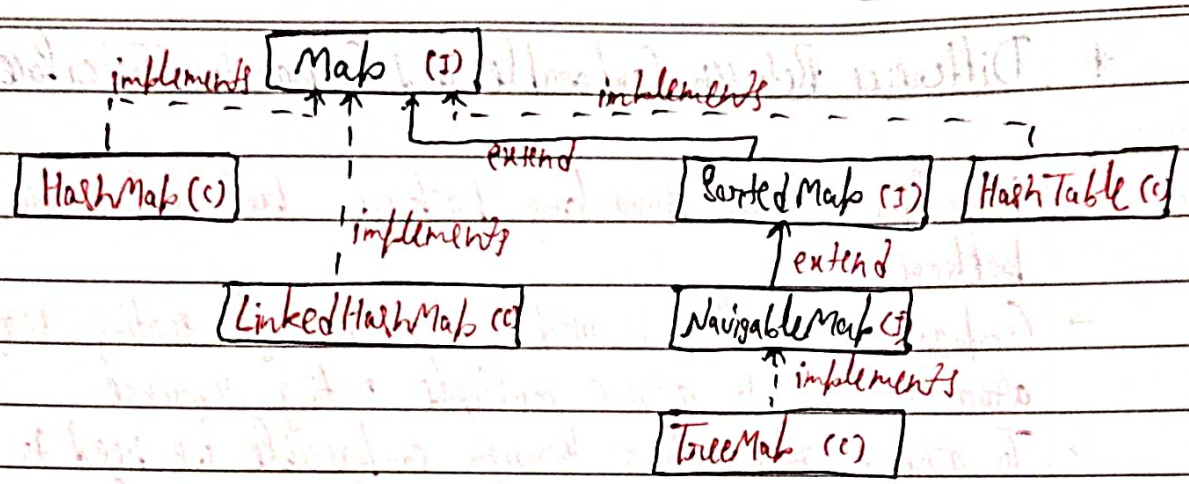
→ In map key and value both will be in object form. Inside a Map all keys must be unique but values can be duplicate.

→ One key will refer only one value.

Different classes which implements Map is:-

(a) HashMap,
(b) LinkedHashMap,
(c) TreeMap,
(d) HashTable

Date / /
Page No.



HashMap :- HashMap is ~~an inter~~ class which implement map interface.

→ HashMap based on a mechanism called Hashing.

→ In HashMap insertion order will ~~no~~ not maintain it will provide very fast search operation.

LinkedHashMap :- LinkedHash map is a class which implement Map interface.

→ In LinkedHashMap elements are stored based on an algorithm Hashing with LinkedList implementation.

→ In LinkedHashMap ~~#~~ insertion order of the elements are maintained so elements can't be access in the same order entered.

TreeMap :- In TreeMap elements are always sorted based on it's key.

→ When our requirement will be sorted Map then treemap should be used for such scenario.

HashTable :- This class implements map interface.

→ HashTable is similar as HashMap but HashTable is synchronized while ~~Hah~~ HashMap is not synchronized.

Date ___/___/___
Page No. ___

* **Methods of Map:-**

(a) put (Object) (k:v):- By this method we can add. By the entry into Map.

(b) remove (Key):- By this method we can remove an entry as per provided key: value.

* **Constructors of HashMap:-**

Normal → `HashMap hm1 = new HashMap();`

Generics → `HashMap<Integer, String> hm1 = new HashMap<Integer, String>();`

* Create a Map and access:-
(i) All the keys,
(ii) All the values only,
(iii) All the entries.

(i) We can ~~acce~~ access all the keys by using a method called keySet().

Ex:-
```
for (Integer x: hm1.keySet())
{ sop(x); }
```

(ii) We can access all the values by using a method called values().

Ex:-
```
for (String x: hm1.values())
{ sop(x); }
```

(iii) Program:-
```
for (Map.Entry<Integer, String> x: hm1.entrySet())
{ sop("Key is: " + x.getKey() + "value is:" + getValue()); }
```