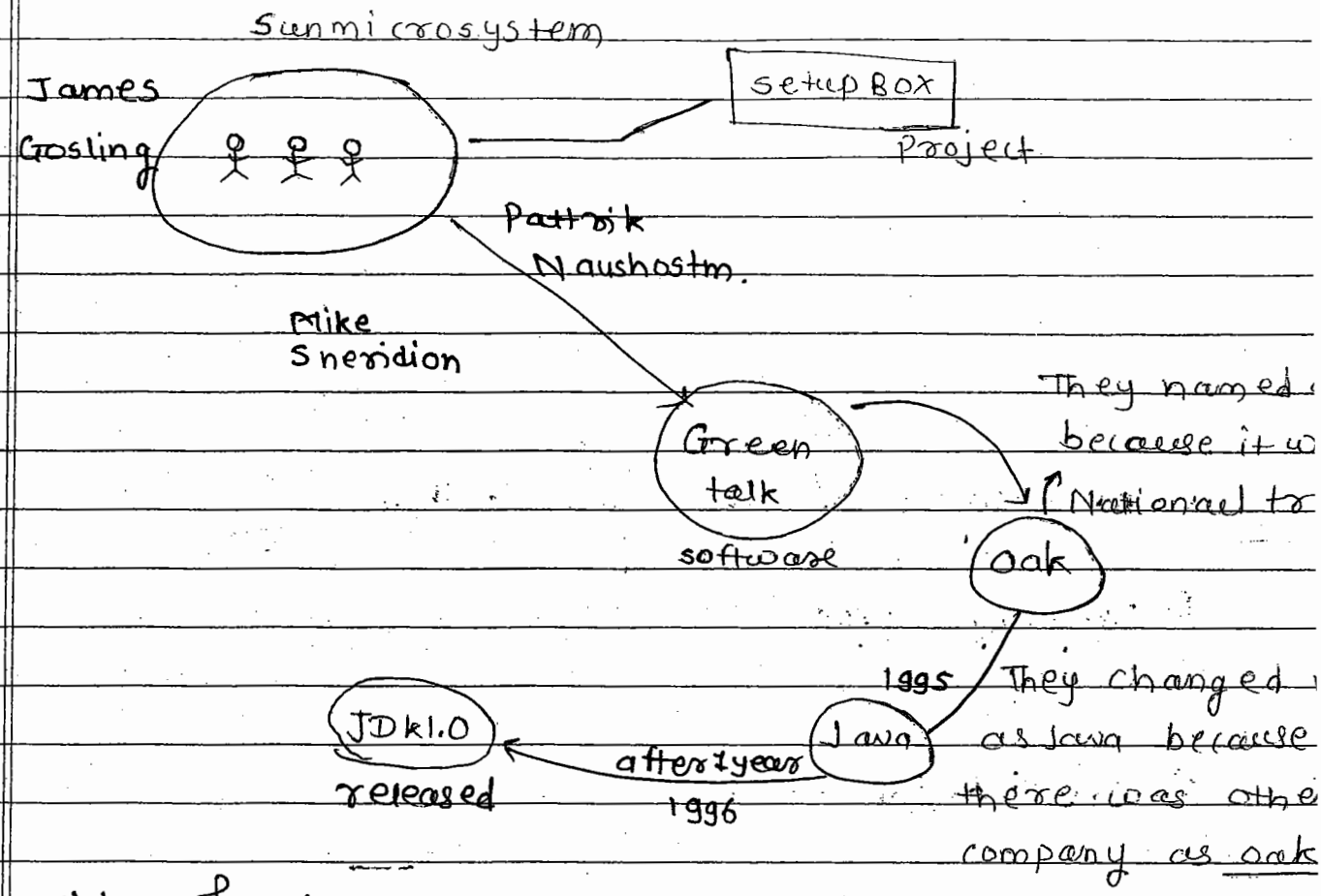


* Edition of Java.—

Java SE (standard edition) Java platform.
Java EE (Enterprise edition)
Java ME (Micro edition)

- Java is a class based object-oriented language
- Any real world entity is object.

* 5 Types of programming language.—

1) Procedural Programming language
2) Functional programming language
3) scripting programming language
4) logic programming language

Page No.
Date

5) Object- oriented programming language.

- The first version of JDK the release in the year 1996 - JDK 1.0

- Latest version available in the market is JDK 20 '2023-03-21'.

* Advantages of Java over python:-

1) language Type.- General purpose language follows write once run anywhere.

2) compilation. - compile easily on any platform without hassal. It has the flexibility. That gives is one edge.

3) speed. - Java is 25 times faster than python in popular game.

* What is language &
→ Human language has syntax. a set of rules for connecting word together to make a statements & quetion. It is a medium through which people communicate with one another. It is made for communication

* Qualies which are connected with languages.
→ 1) Discretness. - (word of sound) word means collection of character.

Page No.
Date

I) Grammer. - Arranging the words in a proper manner of a set of structure rules created for that particular language.

2) Displacement. - It is an ability to think about the past & future.

* Programming language. -
The language which is used to command a comp to do specific task is known as programming language.

* History of PL. -
- The first high-level programming language was "Plankalkul" created by kanrad zuse between 19 & 1945.

- The first high-level language to have an associat compiler was created by carado Bohm in 1951, for his ph.D thesis.

- The first commercially available language was for forTran (formula Translation), developed in 1956 (first manual appeared in 1956, but first develope in 1954) by a term led team led by John Backus at JBM.

* What is application. -
A computer program with an interface enabling people to use the system (laptop, pc, mobile)

as a tool to accomplish a specific task.

* Types of application —

1) Standalone Application.
2) client-Server application.
3) Web application.
4) Hybrid application.

1) Standalone application. —
- Application which is used by Single User.
- Which does not require any internet connection.
- Which does not depends on any other external application for its running / working.
- This application requires a memory from the local Storage (RAM).

Ex. — Calender, Calculator, Notepad, compass, Camera, gallary, point, Setting, Microsoft office, word, Powerpoint, Snake game, etc.

2) client-Server application. —
- In a client-server application it consist of a client program that consumes services provided by the Server.
- The client request the server by Sending the request and the request is processed inside the server. the server sends back response to the client.

Ex. — Banking application (Paytm, Google Pay)

Page No.
Date

3) web - application. —

- A web application is a software that runs in our browser.
- The application which can be access through web browser are known as web application.

Ex.- Goverment websites, University websites

4) Hybrid application. —

- It is the combination of client - server & web application. so it is known as hybrid application.

Ex. — Instragram, facebook, whatsapp, Telegram.

Page No.
Date

* Binary language. —

- Binary language of computer is defined as combination of zeros & one (0 & 1).
- Binary language consist of zeros & one's. Typically in group of 8 or 16 character.

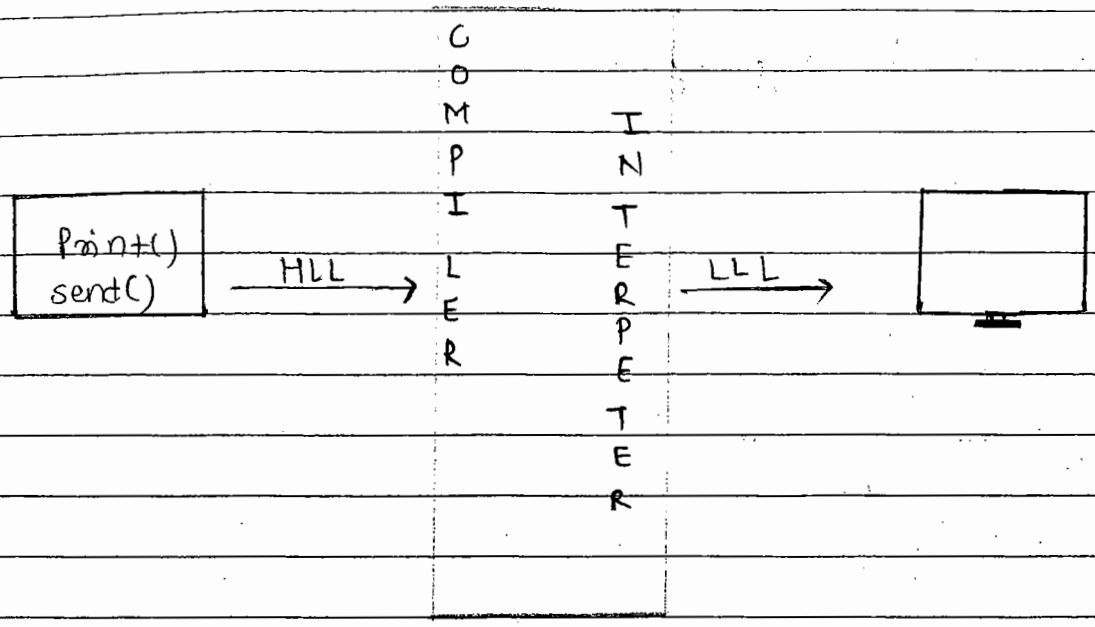
- Any language which made up of 2 digit called as binary.

* Level of programming language. —

1) Low - Level programming language.
2) Mid - Level programming language.
3) High - Level programming language.

1) High level language. —
The language which is easily Understandable, readable, instructible by a human is called as high - level language. language which is similar to human language.

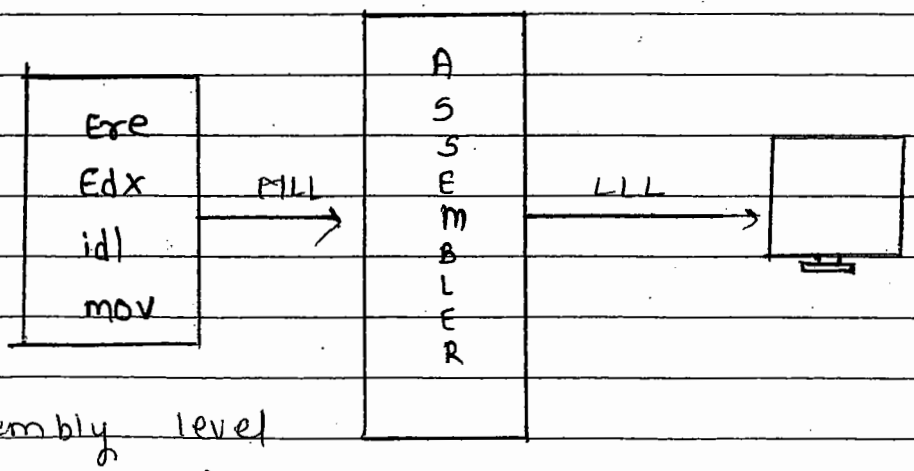
- language which is similar to human language.



Ex. – Java

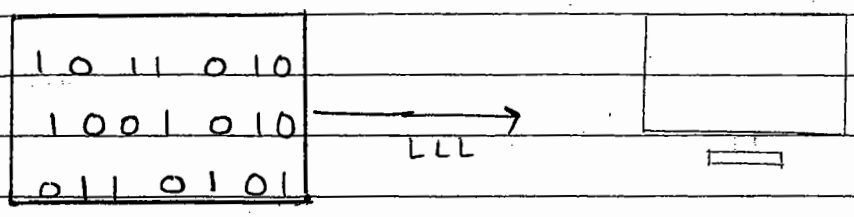
* Mid-level language. – (Pen emonics)

The language which contains predefined set of words which is understandable to computer within intermediate software (Assembler) known as mid level language.



(Assembly level language.)

3) Low level language.—
The language which is easily understandable to computer is known as low level language.



(Binary)

e.g. Binary (consist of 1)

* Assembler.—

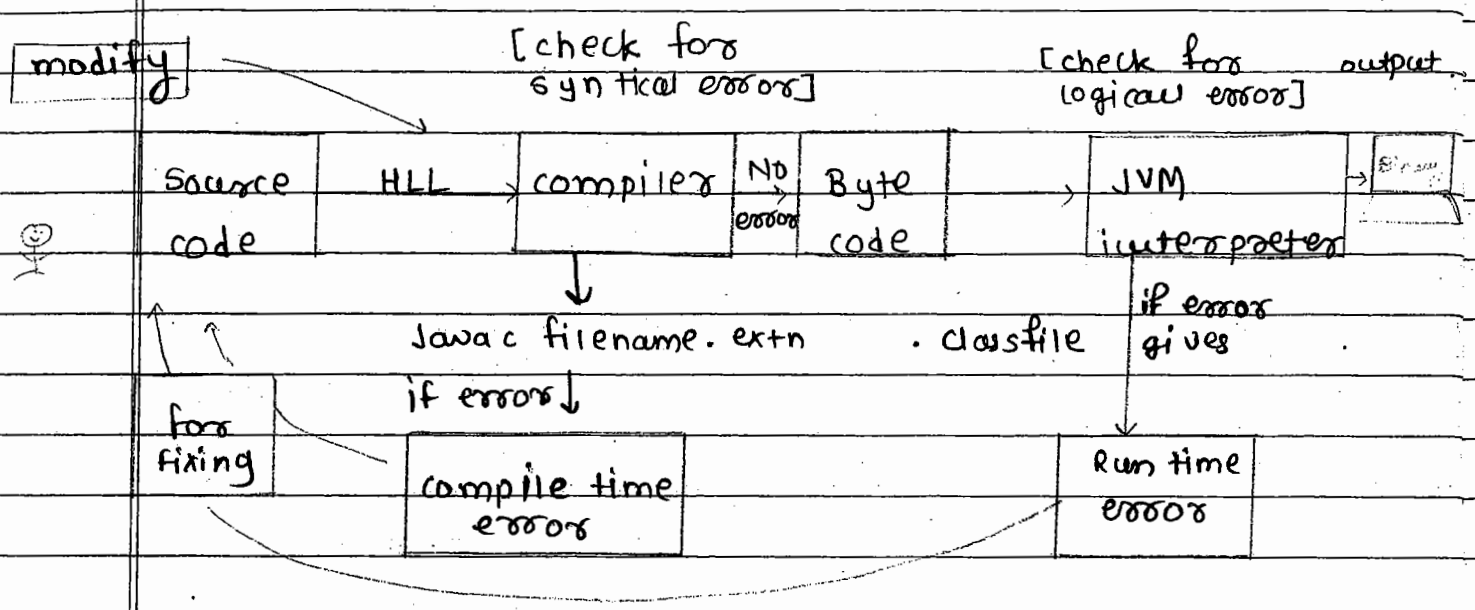
- Assembler is an inter mediate softwar which is used to convert assembly language to Machine Understandable language.

Compiler	Interpreter.
1) It convert HLL to LLL.	1) It also convert HLL to LLL (with the help of JIT Just-in-time compiler).
2) It reads entire program at a once.	2) It read single line of code or instruction at a time.
3) It generate an intermediate file which is byte code (.class file)	3) It does not generate any kind of file.
4) Compilation is done before execution.	4) It does not generate anny

4) Compilaution is done before execution.	4) Compilaution & execution takes place simultaniously
5) compiler is faster.	5) Interpretor is slower.
6) memory requirement is more because of byte code.	6) memory requirement is less because it does not generate any intermed file.
7) Display all error at a same time.	7) Display errors line by line.
8) Error detection is difficult (debugging)	8) Error detection (debugging) is easy.
9) It consume less time.	9) It consume more time.
10) Compiler check for synh-ctal error / mistakes.	10) Interpreter checks for logical mistake.
11) Error given by compiler is known as compiler time error.	11) Error given by interpretor is known as run time error.

* What is programming?

programming is a process of creating a set of instruction that tell a computer how to perform task.

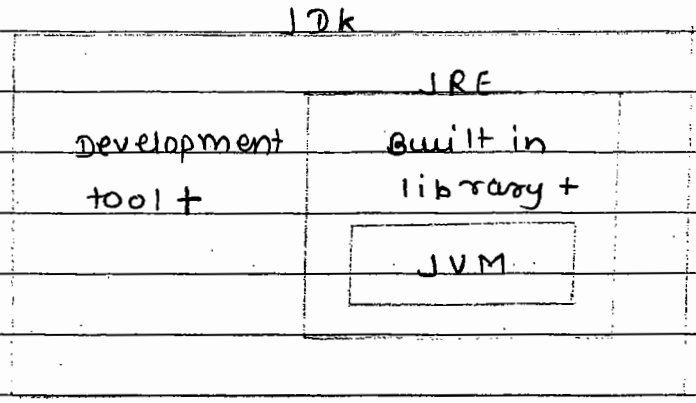


Both file saved in a same location.

Flow of code.

Page No.
Date

* JDk architecture. —



* JDk. —

- JDk stands for Java Development kit. It is a package provided by oracle community. basically we are using JDk to develop & execute a Java applica

It consists of development tools serveral developm tools & utilitizer with JRE.

* JRE. —

- JRE stands for Java Runtime Environment It is used only to execute (run) the Java applications.

- In case if we want to just run a java application we can use a JRE.

- JRE consists of JVM as well as some built in libraries to run a Java code.

Page No.
Date

* JVM :-

- The JVM stands for Java Virtual Machine is a software program that interprete Java byte code.

- This is one of Java's main advantage. a Java byte code can run on different hardware platform & operating system (platform Independency).

- Java Source code is compile into byte code & that byte code is interpreted by the JVM.

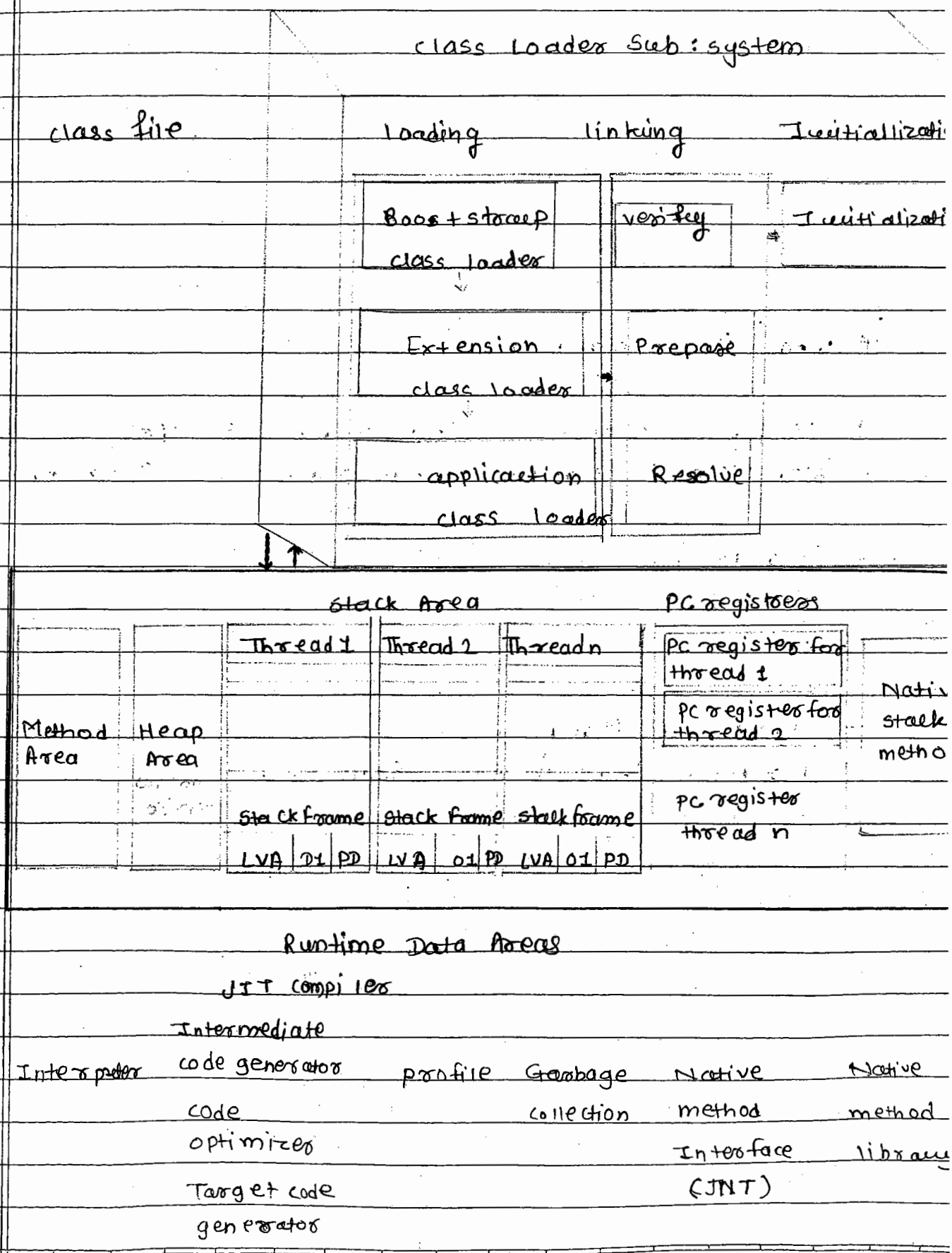
- The JVM executes the byte code along with the code in the library.

- We can execute a byte code on any platform which is having a JVM, which is an interpreter.

- It translate a individual instructions in the byte code into machine understandable language one at a time rather than the whole program as a single unit.

Page No.
Date

* JVM Architecture. —



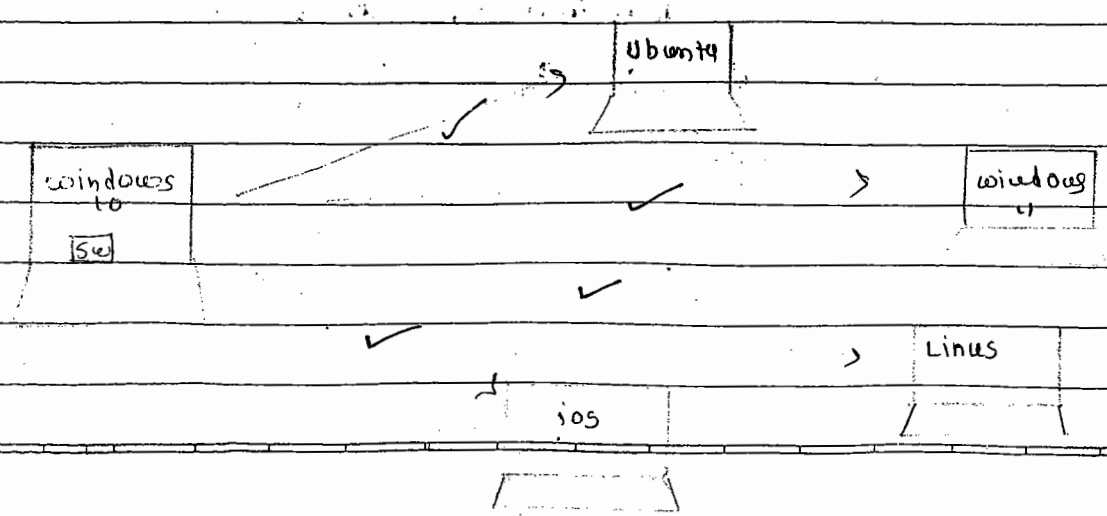
* Source code.-
Source code is a set of Instructions commands & statements which is return by a programmer by using programming language.
OR
The source code is a collection of computer Instructions return using a human readable programming language.

* What is a platform.

1) Is Java platform Independent ? → Yes.
2) JDK JRE is platform Independent ? → Dependent.

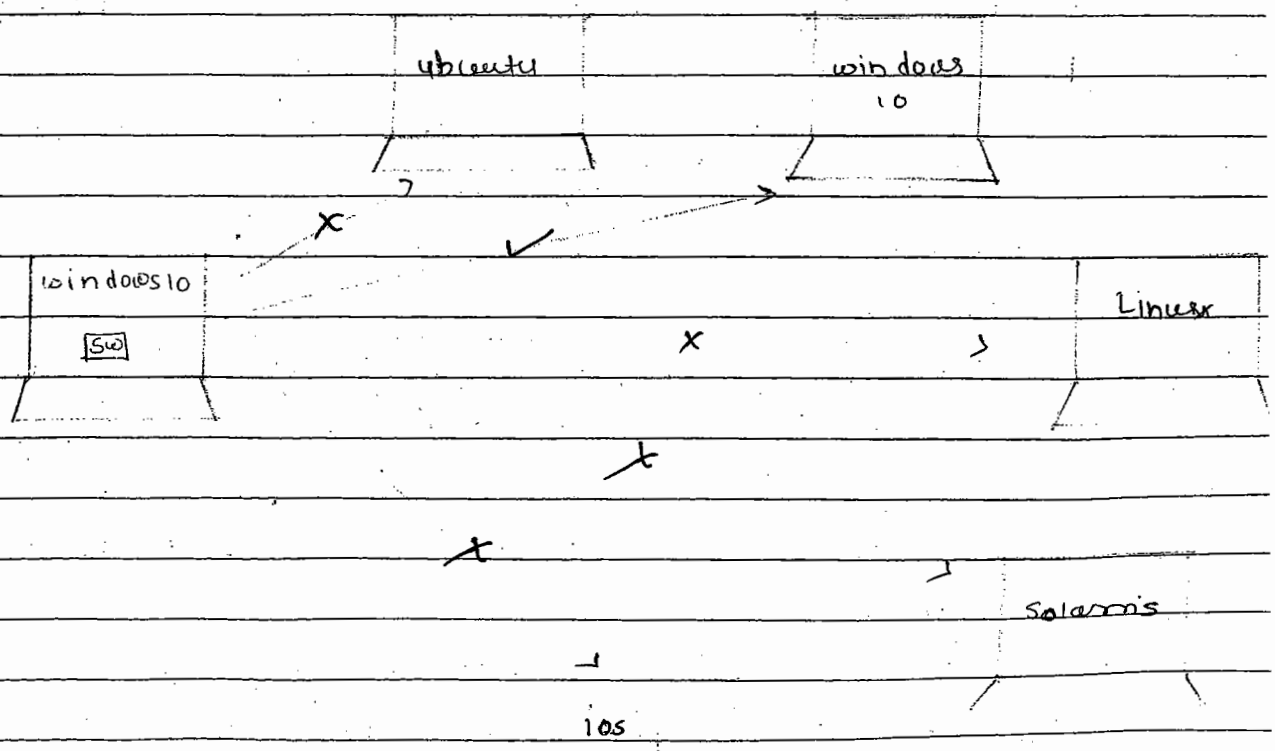
- platform.
A platform is a basic Hardware & software on which application can run.

- Types of Platform.-
1) Platform Independent Software.-
A software developed on 1 platform can be used on any platform such kind of software or called as platform Independent software.



Page No.
Date

2) Platform dependent software. —
software developed on one platform can be used o
that specific platform, such software are called
platform dependent software.



Page No.
Date

* What is Java?
(1995)

- Java is a high level, object oriented programming language. Initially developed by "sun micro systems". (currently owned by "oracle ~~Corporation~~").
- It was designed to be platform Independent that is Java program can run on any device or operating system that has a JVM. (Interpreter).
- These platform Independencuy is achieved through a WORA (Write Once Run Anywhere). principle/approach.

- Features of Java. —

1) Platform Independent. — Java's Bytecode can be executed on any platform which has a JVM in it.
2) Object oriented. — Java follows the object oriented programming principles such as Encapsulation, Inheritance, ploymorphism and abstraction.
3) Rowbust & secure. — Java has a feature like memory management, Strong type checking & exception handling to ensure a rowbust and secure program.
4) multithreding. — Java supports multithreading allowing multiple tasks to be executed concurrently.
5) dynamic. — Java supports dynamic memory allocation & Garbage collection, by simply find memory management.
6) High performance. — while Java Programs not be fast be as compile language like C++ but Java's

performance and speed ~~mt~~ is much faster th
any other programming language, thanks to JVM

7) Portable. – Java's "Written Once Run Anywhere" capability makes it highly portable.

Tokens.

- Tokens are the smallest unit of Java programm language.
- There are different types of Tokens.–

1) keywords
2) Identifier
3) operator
4) Separator
5) comments
6) literals (Data

1) keywords. – compiler aware / predefined words a known as keywords.
- keywords should be return in lowercase always.
- There are 52 + keywords in Java.

* robust. –

Robust is a quality of Java programming that ensure they can handle unexpected events & errors without crashing or causing harm.

* keywords :-

abstract	double	instanceof	static
assert	else	int	strictfp
boolean	enum	interface	super
break	extends	long	switch
byte	false	native	synchronized
case	final	new	this
catch	finally	null	throw
char	float	package	throws
class	for	private	transient
const	goto	protected	true
continue	if	public	try
default	implements	return	void
do	import	short	volatile
			while.

Note :-

- "$" is used as a class name. but "_" is not used as a Identifier.
- "_" is used as a Identifier before the JDK 8 but after JDK 8 "_" is used as a keyword.
- special chars are not allowed except $ & _
- keywords are not allowed
- should not start with a number
- number of any length - characters special allowed

rules are mandatory to follow if not followed we get compile time error because rules are checked by the compiler.

Page No.
Date

* conventions — are nothing but industry standards followed by the programmer (culture).
- It is not mandatory to follow but it is highly recomand
- conventions are not checked by the compiler.

1) * conventions for naming a [class]. — Single
- class name should be in a Upper case. — Multi Word
- eg. — ① class Program1
{
}

② class program1
{
}

- 1) e.g. — Single 2) Single Word Namming.
- ~~ILoveM~~ If class name has multiple words every first ~~characters~~ are in upper case.
- e.g. — ILove My Country Bharat ✓
ilove My country bharat ✗

- Note — these style of ~~writing~~ writing the class name called as a pascal casing.

- conventions of [Interface name] ~~class name~~ is same for ~~interface name~~ class name.

2) conventions for [method] name. —
If the method name is single word the ~~first~~ character must be in lower case & rest all other ~~characters~~ also in lower case.

Page No.
Date

- Ex. — class Demo addition()
  } main(string [] args)
  public static void ~~main~~

e.g. — ① word
② iLove MyCountryBharat.

- If method name has multiple ~~name~~ word, the 1st character of the 1st word must be in upper case from the 2nd word every character must be in uppercase.

- Note — these type of naming method is called as a camel casing.

- conventions for a variable is same as method conventions

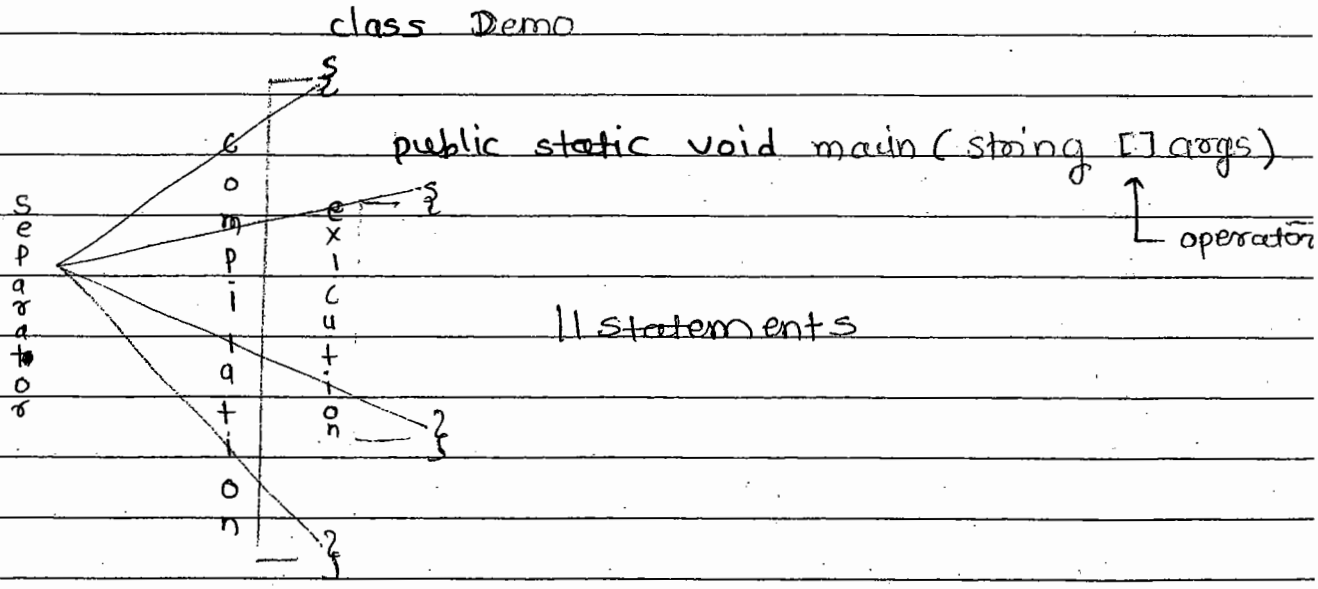
* Literals. —

2) * Seperator. —
It is a symbol which is used to separate any block, statement or any Java members.
- In Java we have 5 separators they are —
{ } Blocks / curly Braces.
( ) Paranthesis.
; End of statements
. Reference.
, multiple variable.

Page No.
Date

Rules of Identifier. —

1) we cannot use keywords in place of Identifie
2) only two special characters are allowed which are " $ ", " _ "
3) It cannot start with a number.
4) charechter spaces are not allowed.
5) It can be a a any length.



* Printing Statements in Java. —

— print & println are the two methods which belon to ~~print stream~~ Print Stream class
— Print Stream class is present in Java.io Packedge.

— System. Out. print [argument]
It is a printing statement which prints the value

and move the cursor to next letter.

- System.out.Println[arguments] —
It is a printing statement which prints the value and move the cursor to next line.

Print	Println
1) The cursor remains on the same line after printing the argument.	1) The cursor moves to the next line after printing the argument.
2) The print() only works with arguments, otherwise it will cause a syntax error.	2) The println() method can work without arguments.
3) Basically it performs a single task of printing the argument.	3) It perform multiple task such as printing the argument & moves the cursor to the next line.

* Java is Imperative type of paradigm.

3) Literals (Data) —

Data

- Single Value Data (primitive Data)
  1) Number
  2) charecter
  3) Boolean
- multi value I (Non - primitive I
  1) array
  2) class
  3) String
  4) etc

Number

- Integer (-2, -1, 0, 1, 2)
- Decimal (-2.0, -1.0, 0.0, 1.0, 2.0)

charecters

- Alphabates
  - upperrase (A - Z)
  - lowerrase (a - z)

  ASCI I 65 to 90 ASCI I 97 to 122
- Digits (0 - 9)

  ASCI I (48 - 57)
- Special charecters

  rest other than alphabates and digits all are special charecters

Page No.
Date

- charecters are always enclosed with single quotes (' ').
  Ex. — 'A', 'z', '0', '9', '#', '@'

- ASCII Value of characters are rang from 0 to 127

* boolean. —
  Boolean is used to represent logical values it has two keyword words True & False.

* String. — string is a class.
- group of characters that are enclosed with inge
(" ") are double quotes are known as
- strings. string is a non-primitive data type.
- length of string can be any thing.
- they must be enclosed within double quotes
- (" ") and these strings are case sensitive.
- Ex. — "Hello Mansi".
- string is a predefined built in class present in Java.lang package

Datatype.

- Datatype defines the type of data and the size of data (which we wants to store)

- In Java datatype is broadly classified into two chategorey
  1) Primitive data type
  2) Non-Primitive data type.

Page No.
Date

* primitive data type. —
The data type which is used to creat a container (variable) to store a primi- data value. (single value data.)
Ex. — Integers, decimals, character, boolean

* Non-Primitive data type —
The data type which is used to create a container to store Non primitive data value (multi value data).
Ex. — String, class, Array, etc.

* ASCII. —
(American Standard Code for Information Interchange).

Datatype

primitive data type		Non primitive data type
Integer —	byte	Array
	short	String
	int	class, etc.
	long	
Decimal —	Float	
	double	
character —	char	
Boolean —	Boolean	

primitive data type	Non-primitive data type
1) primitive data types are predefined.	1) Non-primitive ~~datatyper~~ needed to be defined by the user.
2) Primitive data type stores single value data	2) Non-primitive data stores multivalued data.
3) Primitive data types cannot store null value	3) Non primitive data can store null value.
4) The size of primitive data depends on their data type.	4) The size of non-primitive data is not fixed.
5) Primitive data type stores the value.	5) Non-primitive data type stores the reference.

- System is a predefined built in class present in Java.lang package
- The classes which are used for language purpose are present (Defined) in lang packadge.
- Programmer does not needs to import them.
- out is an variable which is used as an reference for print Stream class.

primitive data	datatype	size (byte)	default value	length	range
Integer	byte	1 byte (8 bits)	0	3	-128 to 127
	short	2 byte (16 bits)	0	5	-32768 to 32767
	int	4 byte (32 bits)	0	10	-2147483648 2147483647
	long	8 byte (64 bits)	0l0L1l0l	19	-9.12×10^{18} to 9.12×10^{18}
Decimal	float	4 byte	0.0	39	-1.4×10^{45} to 3.40×10^{38}
	double	8 byte	0.0d~~0.0~~	309	-4.9×10^{324} to 1.79×10^{308}
	char	2 byte	~~to 0000~~ blank		0 to 65535
	boolean	1 bit	false		

undefined..

Page No.
Date

* Normal Java Code. —

filename

```
class Demo
{
    Public static void main (string [] args)
    {
        System.out.println("    ");
    }
}
```

class block

mainmethod, array operator, class name, variable name

method block.

Q: Write a java program to print my personal info (Name, Native, Education, University, CGPA, skills).

```
class PersonalInfo
{
    public static void main (String [] args)
                                Next line.
    {
        System.out.println("Name: Nikita Jagtap");
        System.out.println("Native: Pune");
        System.out.println("University: SPPU");
        System.out.println("CGPA: 6.78");
        System.out.println("skills: SQL, Python, HTML");
    }
}
```

Q: Write a program to find addition of 2 numbers.

Page No.
Date

```
class Addition
{
  public static void main (String [] args)
  {
    System. out. print ln ("Addition of two numbers
    System. out. print (1 5 6 7 6 3 + 5 6 7 5 5 4 6 3
  }
}
```

Note: No (inverted)" " comma because we are performing addition.

Q: Write a program to substraction of two numbers.

```
Class Substraction
{
  public static void main (String [] args)
  {
    System. out. print ("substraction of two numbers
    System. out. print ( 9876543 - 43525);
  }
}
```

Q: Write a program to do multiplication of two numbers

→

```
Class Multiplication
{
  public static void main (String [] args)
  {
    System. out. print ("Multiplication of two no:");
    System. out. print (5438 * 12);
  }
}
```

Q: Write a program to display Hello world.

```
class Demo
{
  public static void main (String [] args)
  {
    System.out.print ("Hello world");
  }
}
```

Q: Write a program to display My first java program.

→

```
class MansiPatil
{
  public static void main (String [] args)
  {
    System.out.print("My first java program");
  }
}
```

* Variables :—

Variables are used to store values to be use
Latter in a program. They are called variabl
because their values can be changed.

- variables is a place or a container to sto
the data.

- Variable declaration. —

syntax. — datatype variable_name;
Ex. — int a;

- variable Initiallization. —

syntax. — Variable_name = values;
Ex. — int a = 10;

- Variable Utillization. —

system.out.println (a);
O/P. — a = 10;

- Variable declaration and Initialization in a single value.

syntax. — datatype variable_name = values;
Ex. — int a = 10;

Page No.
Date

- Variable Re-initialization

Ex -- int y=80; int y=100;
system.out.println(y);

o/p -- y=100;

- copying a value from one variable to another.-

→ class variable Example
{
public static void main (String[]args)
{
~~int~~ byte a=10;
byte b=a;
System.out.println("A"+a);
system.out.println("B"+b);
}
}

* Special Operator. — (+)

Based on the value it will perform two operation.

- Additional Operator.
- Concatination.

	Operand 1		Operand 2		output datatype
addi.	number	+	number	→	number
add	number	+	character	→	number
add	character	+	character	→	number
conc.	number	+	string	→	string
	character	+	string	→	string
	String	+	string	→	String

Page No.
Date

```
{
    public static void main (String [] args) {
        system.out.println ("hello"); // hello
        System.out.println (10);          10
        system.out.println (10 + 10);     20
        system.out.println ("10+10");     1010
        System.out.println ("number is:"+10);   number is 10
        System.out.println (1-'a');       -96
        System.out.println (1+'0');       49
        system.out.println ('b'+'a');
        system.out.println ("hello"+10);
        system.out.println (10+"10");
        System.out.println (10+10+"10");
        System.out.println ('a'+10+"a");
        system.out.println (10+"10"+10);
        system.out.println (10+'a'+"10"+'1');
        system.out.println ('a'+"a"+'a');
        system.out.println (10+"10"+10*2);
        system.out.println (10+"20"+100/2);
    }
```

- Declaraution of multiple variable on ~~seeingl~~ statement. -

```
{
    public Sectatic void main(String [] args)
    {
        byte a,b,c,d = 10;
        a = 10;
        b = 20;
        c = 30;
        ~~d 10~~ S.O.P.(a);
                 S.O.P.(b);
                 S.O.P.(c);
```

```
        S.o.p.(d);
}
```

output — 10
20
30
10

- Variables can be classified into two categories.—

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

* Global Variable. —

- Any variables which is declared outside method and inside the class is called as Global Variables.
- The scope of the Global variables is from the beginning of the class till the end of the class.
- It can be classified into static and non-static.
- It will have default values.
- Once the global variables is declared immediately in the next line we cannot initialized or re-initialized, if so we will get

compile time error.

* Local Variable. —

- Any variables which is declared within the meth is called as local variables.
- The scope of the local variables is from the beginning of the method till the end of the meth or within the method.
- It cannot be classified into static and non-static
- It will not have default values.
- Local variables should be initialized before utilization.

* A Java program to Print default values for data types.

```
{
    static byte a;
    static short b;
    static int c;
    static long d;
    static float e;
    static double f;
    static char g;
    static boolean h;
    static string i;
    public static void main(String [] args) {
        System.out.println("byte -" + a);
        S.o.p. ("short - "+b");
        S.o.p. ("int - " + c);
```

```
s.o.p. ("long -" + d);
s.o.p. ("float -" + e);
s.o.p. ("double -" + f);
s.o.p. ("char -" + g);
s.o.p. ("Boolean -" + h);
s.o.p. ("string -" + i);
}
}
```

— output. —

```
0
0
0
0
0.0
0.0
space
false
null
```

Page No.
Date

* primitive data type

- Byte.

size - ~~size~~ - 1 byte or 8 bits
Range - max value = +127
min value = -128

sign block.

1	2	3	4	5	6	7	8

- formula to find max value,

$$\boxed{2^{n-1} - 1}$$ (n means no. of bits)

Ex. - $2^{8-1} - 1$
$2^{7} - 1$
$128 - 1 = 127$

max value = 127.

- formula to find min value

$$\boxed{-2^{n-1}}$$

Ex. - -2^{8-1}
max value - 128

- conversion of a decimal number to Binary. -

1) 127

2	127	1
2	63	1
2	32	0
2	16	0
2	8	0
2	4	0
2	2	0
	1	

0 1 1 1 1 1 1 1

2) 57
4
17
10

2	57	1
2	28	0
2	14	0
2	7	1
2	3	1
	1	

1 1 1 0 0 1

- Byte Example. —

```
public static void main(String[] args) {
    byte a = 0;
    byte b = -128;
    byte c = -129; //CTE (Possible lossy conversion from int to byte)
    byte d = 127;
    byte e = 128; // CTE ( Possible lossy conv. from int to byte)
}
```

Q: Write a java program to find max & min value using wrapper class variable. —

→ class ByteExample
{
public static void main (String [] args) {
System.out.println (" Max : "+Byte.MAX_VALUE
System.out.println (" Min : "+Byte.MIN_VALUE
}
}

O/P. → MAX : +127
MIN : -128.

* Assignment. —

1) Class Cospideas
{
public static void main (String [] args) {
byte staff = 25;
byte classRooms = 10;
byte JavaTrainer = 3;
byte subjects = 8;
byte ac = 40;
byte camera = 24;
~~byte PC~~
System.out.println ("No of staff members : "+ staff);
S.o.p. ("No. of classrooms : "+ classRooms);
S.o.p.

```
System.out.println("No of staff meembers :"+ staff);
S.o.p.("No of classrooms : " + classRooms);
S.o.p.("No of Javatrainins :" + javaTrainers);
S.o.p.("No of subjects : " + subjects);
S.o.p.("No of AC : " + ac);
S.o.p.("No of camera :" + camera);

}
}
```

o/p. → No of classrooms:
No of Javatrainins:
No of subjects:
No of AC :
No of camera:

2) Class India

```
{
    public static void main(string[] args) {
        byte noofStates = 29;
        byte unionTeri = 8;
        byte cabinateMinisters = 45;
        byte cm = 29;
        byte pm = 1;
        byte armyHeadQuarters = 7;
        byte wonderInIndia = 1;

S.o.p.("Number of states :" + noOfStates);
s.o.p.("Number of UnionT :" + unionTeri);
S.o.p.("Number of cabi.Mini." + cabinateMinister);
```

```
s.o.p. ("Number of cheief Minisiter:" + cm);
s.o.p. ("Number of PM:" + pm);
s.o.p. ("Number of Army Head quuters:" + armyHeadquuters
s.o.p. ("Number of Horeld Wonder in India:" + wonderIn Indi
}
}
```

3) G20. —

→

O/P. → Number of states : 29
Number of UnionT : 8
Number of cabi.Mini.: 45
Number of cheeif Minister: ~~29~~ 1
Number of Army Head quters: 7
Number of World Wonder in India: 1

Page No.
Date

3) G20. —

→ class G.

```
{
    public static void main (String [] args) {
        byte noofMetting = 18 ;
        byte condIn India = 1 ;
        byte noof Days = 2 ;
        byte participetedContries = 20;
        byte noOfCities = 50 ;
        byte noOf Events = 119 ;

    S.O.P.(" No of metting :" + noofMetting );
    S.O.P.(" coducted in India:" + condInIndia);
    S.O.P.("No of days : " + noOf Days );
    S.O.P.(" Participeated Countries:" + participetedContries);
    S.O.P.(" No of cities :" + noOf Cities );
    S.O.P.(" No of Events :" + noOfEvents); }
}
```

o/p →

```
No of metting : 18
conducted in India : 1
No of days : 2
Participeated Countries : 20
No of cities : 50
No of Events : 119 .
```

Page No.
Date

4) Netaji Bose. —

→ class Bose

```
{
    public static void main(String [] args){
        byte age = 48;
        byte booksWritten = 125;
        byte publishedNewsppr = 1;
        byte lifeBasedMovies = 5;
        byte familyMembers = 13;

        s.o.p. (" Age :" + age);
        s.o.p. (" Books Written:" + booksWritten);
        s.o.p. (" Published Newsppr:" + publishedNewsppr);
        s.o.p. (" life based movies:" + lifeBasedMovies);
        s.o.p. ("nofamily Members:" + familyMembers);
    }
}
```

O/P. →

```
Age : 48
Books Written : 125
Published Newsppr : 1
life based movies : 5
nofamily Members : 13
```

Page No.
Date

5) William shakesphere.—

→ Class Shakesphere

```
{
    public static void main(String[] args) {
        byte noOf Play = 39;
        byte noOf Sonetnets = 122;
        byte noOf Poems = 3;
        byte noOf Famous Work = 20;
        byte age = 51;

        s.o.p. (" No of Play :" + noOf Play);
        s.o.p. (" No of Sonnets :" + noOf Sonnets);
        s.o.p. (" No of Poems :" + noOf Poems);
        s.o.p. (" No of Famous Work :" + noOf Famous Work);
        s.o.p. (" Age :" + age);
    }
}
```

o/p. → No of Play : 39
No of Sonnets : 122
No of Poems : 3
No of Famous Work : 20
Age : 51.

Page No.
Date

2) short

2 bits | 16 bits

Range min Range - -32768
max Range - 32767

- formula for ~~max min~~ value. —

$$\boxed{2^{n-1} - 1}$$

$2^{16-1} - 1$

$2^{15} - 1$

$32768 - 1$

$\underline{\underline{32767}}$

- formulla to find min value —

$$\boxed{-2^{n-1}}$$

-2^{16-1}

-2^{15}

$\underline{\underline{-32768}}$

Page No.
Date

```
class ShortRange1
{
    public static void main (String [] arg s) {
        short a = 0;
        short b = -32768;
        short c = -32769; // CTE(
        short d = 32768; // CTE(
        short e = 32767;
    }
}
```

```
class short Range
{
    public static void main (String [] args) {
        System. out. println ("max value : "+short .MAX_VALUE);
        System. out. println ("min value : "short. MIN_VALUE);
    }
}
```

O/P. → MAX : 32767
MIN : -32768

1) college. —

→ class college

```
{
    public static void main (String [] args) {
        short staff = 250;
        short noOfBenches = 2000;
        short students = 4500;
        short window = 450;
        short vehicles = 1622;
        s.o.p (" staff member :" + staff);
        s.o.p ("No of Benches :" + noOfBenches);
```

```
s.o.p.(" No of Students:" + students);
s.o.p.(" No of windows:" + windows);
s.o.p.(" No of vehicles:" + vehicles);
    }
}
```

O/P. → Staff Members : 250
No of Benches : 2000
No of Students : 4500
No of windows : 450
No of vehicles : 1622

2) Indian Army. —

→

```
class IndArmy
{
    public static void main (String [] args) {
        short coastGaurds = 11,000;
        short headQuarters = 7;
        short baseCamp = 113;
        short defenceTrainingCenter = 142;
        short navyOfficers = 10,000;

        s.o.p.("Coast Gaurds:" + coastGaurds);
        s.o.p.("No. of Head Quarters:" + headQuarters);
        s.o.p.("No. of Base Camp:" + baseCamp);
        s.o.p.("Defence Trainning Center:" + defenceTrainCe
        s.o.p.("No. of Navy Officers:" + navyOfficers);
    }
}
```

Page No.
Date

O/p. → Coast Guards: 11,000
No of HeadQuarters: 7
No of Base camp: 113
Defence Trainning Center: 142
No of Navy officers: 10000

3) TCS. —

→
```
class TCS
{
    public static void main(String[] args)
    {
        short ~~bindia~~branchesInIndia = 150;
        short retirementAge = 65;
        short ~~#~~ no.OfBoardDirectors = 9;
        short workingHours = ~~12;~~ 9;
        short proLangUsed = 13;

        S.O.P.(" Branches In India:" + branchesInIndia);
        S.O.P.(" Retirement Age:" + retirementAge);
        S.O.P.(" No. of Board of Directors:" + no.OfBoardDirectors);
        S.O.P.(" Working Hours:" + workingHours);
        S.O.P.(" Prog. Lang. Used:" + proLangUsed);
    }
}
```

O/p. → Branches In India: 150
Retirement Age: 65
No of Board of Directors: 9
Working Hours: 9

Page No.
Date

prog. lang. Used = 13.

4) AirIndia. —

→ class AirIndia

```
{
    public static void main (String[] args)
    {
        Short noOfEmployee = 8400;
        Short noOfAircraft = 700;
        short noOfFlightsPerDay = 6500;
        Short noOfPilot = 1600;
        short noOfCourtCases = 600;

        S.o.p. (" No of Employee:" + noOfEmployee);
        S.o.p. (" No of Aircraft:" + noOfAircraft);
        S.o.p. (" No of Flights/day:" + noOfFlightsPerDay);
        S.o.p. (" No of Pilot:" + noOfPilot);
        S.o.p. (" No of court Cases:" + noOfCourtCases);
    }
}
```

O/P. → No of Employee: 8400
No of Aircraft: 700
No of Flights/day: 6500
No of Pilot: 1600
No of court Cases: 600

Page No.
Date

5) PMPML. —

→ class Pmpml.

```
{
    public static void main (string[] args)
    {
        short noOfBuses = 2079;
        short noOfBusStop = 7693;
        short passangerCapacity = 54;
        short noOfDriver = 2296;
        short noOfConductors = 4776;

        s.o.p. ("No of Buses:" + noOfBuses);
        s.o.p. ("No of BusStop:" + noOfBusStop);
        s.o.p. ("Passanger Capacity:" + passangerCapacity);
        s.o.p. ("No of Driver employee:" + noOf Driver);
        s.o.p. ("No of conductor employee:" + noOf conductor);
    }
}
```

o/p. → No of Buses: 2079
No of Busstop: 7693
Passanger capacity: 54
No of Driver employee: 2296
No of Conductor employee: 4776

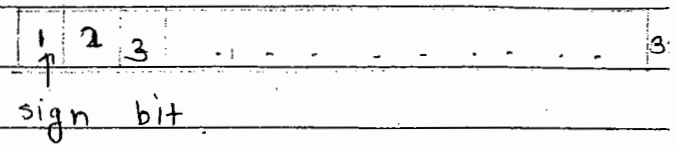
* int (Note). —

- length of digit we can initialize in int is 10
- If we try to store a integer value more than int's range we will get compile time error (exc integer number is too long).

3) int

- Int is default datatype for integer number.

- size - 4 bytes = 32 bits



- Range - -2147483648 to 2147483647.

- Java pgm for int

```
class IntExample
{
    public static void main (String [] arg) {
      int a = 0;
      int b = -2147483648;
      int c = -2147483649;
      int d = 2147483647;
      int e = 2147483648;
    }
}
```

- Java pgm to find the range of int using wrapper class methods. -

```
class IntRange
{
    public static void main (String[] args)
    {
        System.o.p.("min value:" + Integer.MIN_VALUE);
```

```
        S.o.p("Max value :" + Integer.MAX_VALUE);
    }
}
```

O/p. → Min value.— -2147483648
Max value.— 2147483647.

1) Doctors.—

1)
```
class Int
{
    public static void main (string [] args)
    {
        int noOf Doctors = 153513;
        int noOf FemaleDoc = 60786;
        int noOf Hospi = 2440;
        int noOf Beds = 231435;
        int noOf Policies = 870;

        System.o.p ("No of Doctors : " + noOf Doctors);
        S.o.p. ("No of Female Doctors :" + noOf FemaleDoc);
        S.o.p. ("No Of Gourment Hospitals :" + noOfHospitals);
        S.o.p. ("No Of Beds:" + noOfBeds);
        S.o.p. ("No Of Policies :" + noOf Policies);
    }
}
```

O/p. → No of Doctors: 153513
No of Female Doctors: 60786
No of Gouvment Hospitals: 2440
No of Beds: 231435
No of Policies: 870

Page No.
Date

- Isro. -

2)
```
class index
{
    public static void main (String[] args)
    {
        int Employee = 167865;
        int research = 8457218;
        int rockets = 12248;
        int satelightLaunched = 150;
        int scientist = 324543;

    s.o.p ("No of Employee :" + ~~noof~~ employee);
    S.O.p. ("No of Research :" + research);
    S.O.p. ("No of Rockets :" + rockets);
    S.O.p. ("No of satelight :" + satelightLanched);
    S.O.p. ("No of scientist :" + scientist);
    }
}
```

O/P → No of Employee : 167865
No of Research : 8457218
No of Rockets : 12248
No of Satelight : 150
No of Scientist : 324543

- Cosmatics Brands

3)
```
class intexa
{
    public static void main (String [] args)
    {
```

```
int cosmaticsBrands = 1389700;
int cosmaticsCompany = 3542234;
int employees = 23434481;
int activeCases = 45987;
int onlineBuisness = 18765;

s.o.p("No of cosmaticsBrand:" + cosmaticsBrands);
s.o.p.("No of cosmatics company:" + cosmatics Company);
s.o.p.("No of Employees:" + employees);
s.o.p.("No of Active Legal cases:" activeCases);
s.o.p("No of online cosmatics shopes:" + onlineBusiness);
}
}
```

O/p. ⟶ No of cosmatics Brand: 1389700
No of cosmatics company: 3542234
No of Employees: 23434481
No of Acutive Legaul cases: 45987
No of online cosmaticsshops: 18765

— Flipkart. —

4)
```
class intexam
{
    public static void main(String [] args)
    {
        int noOfPhones = 213952;
        int noHomeAppli = 22341268;
        int offerPerDay = 432564;
        int cellsPerHouse = 18932;
```

```
int noOfBook = 1878924;

S.o.p. ("No of Phones:" + noOfPhones);
S.o.p. ("No of Home Appliences:" + homeAppli);
S.o.p. ("No of offers/Day:" + offerperDay);
S.o.p. ("cells/Houre:" + cellsperHoure);
S.o.p. ("No of Books:" + noOfBook);
}
}
```

O/P → No of phones : 213952
No of HomeAppliences : 2234 1268
No of offers/Day : 432564
cells/Houre : 18932
No of Books : 1878924

laptop.-

5)
```
class interamp
{
    public static void main(String [] args){
        int dell = 3645344;
        int lenovo = 764436;
        int macbook = 645343;
        int asus = 545343;
        int hp = 432435;
        S.o.p. ("Dell Users:" + dell);
        S.o.p. ("Lenovo Users:" + lenovo);
        S.o.p. ("Macbook Users:" + macbook);
        S.o.p. ("Asus Users:" + asus);
        S.o.p. ("HP Users:" + hp);
    }
}
```

Page No.
Date

4) long -

- Size - 8 bytes = 64 bits.
- Range - max = 9.12×10^{18}
  min = -9.12×10^{18}

1	2	. . . . .	64

- class longRange
{
    public static void main (String [] args) {
        S.o.p("MAX :" + long.MAX_VALUE);
        S.o.p ("MIN :" + Long.MIN_VALUE);
    }
}

- class LongExample
{
    public static void main (String [] args){
        byte a = 12L
    short b = 32767L;
    int c = 2147483647;
    int d = 2147483647L;
    long e = 2147483647;
    Long f = 12345678901234567890L;
    long g = 12345678901234567890L;
    Long h = 12345678901234567890L;
    }
}

Page No.
Date

1) Dominoz Pizza. —

```
Class longex
{
    public static void main (String [] args) {

        long employee = 315106429L;
        Long cust = 1000000000L;
        Long PizzaPerDay = 4000000L;
        Long PizzaPerYear = 1200000000L;
        Long franchise = 87094508L;

        s.o.p ("No of Employee:" + employee);
        s.o.p. ("No of Customers:" + cust);
        S.O.P. ("No of sells / Day:" + pizzPerDay);
        S.O.P. ("No of sells / Year:" + PizzaPerYear);
        s.o.p. ("No of Frenchise:" + franchise);
    }
}
```

O/P. → No of Employee: 315106429
No of customers: 100000000
No of sells / Day: 4000000
No of sells / Year: 1200000000
No of Frenchise: 87094508

Note. —

1) If we try to store a Integer value which exceeds the capacity of int we will get a compile time error (Integer number too large.)

2) To over come these error we specify with a char 'l'/'L' at the last.
These character is just for the JVM's Understanding given by a programmer to specify that the number which we are Initiallizing is a long number.

Page No.
Date

2) Human body :-

```
class longexa
{
    public static void main (String [] args)
    {
        long bloodVessales = 680000L;
        Long noOf Cells = 40000 000 000L;
        Long noOf Mussels = 600L;
        Long noOf White Cells = 1100000000 L;
        Long noOf Red Cells = 3000000000000L;

s.o.p.("No of Blood vessales:" + bloodVessales);
s.o.p.("No of cells :" noOf cells);
s.o.p.("No of Mussels:" noOf Mussels);
s.o.p.("No of White cells:" noOf White Cells);
s.o.p.("No of Red cells:" noOf Red Cells);
    }
}
```

O/P. ⟶ No of Blood Vessales : 680000
No of cells : 40000000
No of Mussels : 600
No of White cells : 110000000
No of Red cells : 3000000000

Note. —

4) long can hold upto 19 digits.

Page No.
Date

3) Spotifey :—

```
class longexam
{
    public static void main(String[] args)
    {
        Long songs = 1000000000L;
        long artist = 1423467900L;
        Long users = 4 5623 90078L;
        long podcast = 4764001 0632;
        Long primium = 1 2 5 67 348L;

s.o.p. ("No of Songs on Spotifey:" + songs);
S.O.p. ("No of Users:" Artist :"+ artist);
S.o.p. (" No of Users :" +users);
s.o.p. (" No of Podcast :"+podcast);
S.o.p. (" No of primium Users":+primium);
    }
}
```

o/p. → No of Songs on Spotifey: = 1000 0000
No of Artist: = 1423467900
No of Users: 4 5 6 2 3 9 0 0 7 8
No of Podcast: 4 7 6 4 0 0 1 0 6 3 2
No of Primium Users: 1 2 5 6 7 3 4 8

5) Float.

size = 4 bytes = 32 bits.

Range = max = 3.40×10^{38}
MIN = -1.4×10^{45}

— Write a java program to find range of float using wrapper class method —

→
```
class FloatExample
{
    public static void main(String [] args)
    {
        S.O.P.("max :" + Float.MAX_VALUE);
        S.O.P.("min :" + Float.MIN_VALUE);
    }
}
```

O/P → max: 3.40×10^{38}
min: -1.4×10^{45}

```
- class Error
  {
      p.s.v.m.(S.[]args)
      {
          float a = 127;
          float b = 32768;
          float c = 2147483647;
compiletimeerror// float d = 2147483648;
          float e = 1234567890;
compiletime error // float f = 12345678901;
          float g = 12345678901L;
          float h = 1234567890123456789L;
compiletime error // float i = 1234567890123456789 0L;
          float j = 1234567890123456789 0f;
          float k = 123.....39f;
compile time error// float l = 123.....40f;
          float m = 1.1F;
          float n = 1.1f;
      }
  }
```

* Note. —

- Float can hold upto 39 digits at its +ve range.
- If we try to store to a number exceeding the capac of int or long we need to suffix that number with a character 'f'/'F'
- If we try to store a number without suffixin it will & it exceeds the capacity of int we will get a compile time error (integer number too large) & if we try to store an integer number which is exceeding the capacity of float we will get a compi time error (floating point number is to large).
- If we try to store a decimal number without suffixing with character 'f'/'F' we will get a compile time error (possible lossy con from float to double)

Page No.
Date

6) Double. —

Size. — 8 byte → 64 bits.

Range. — Min — ~~1.79 × 10^308~~ -4.9×10^{324} (4.9E324)
Max. — 1.79×10^{308} (1.79E308).

E = Exponential.

```
* class FloatExample
  {
      ps v.m.(s[]args)
      {
          S.o.p.("max range :" + Double.MAX_VALUE);
          S.o.p.("min range :" + Double.MAX_MIN_VALUE);
      }
  }
```

O/P. →

* Note. —
- It is a default data type for decimal number.
- Insight Double container we can store any other number data type (byte, short, int, long, float).
- It is not mandatory to specify at the last of a decimal number because it's a default data type.
- Max^m length of ~~integer~~ a number for integer is 309 for double.

Float	Double
1) Size = 32 Bits	1) Size = 64 Bits.
2) Float is a single-precision data type.	2) Double is a double-pre on data type.
3) It has low accurecy.	3) It has high accurecy.
4) Float takes 4 bytes.	4) Double takes 8 bytes.
5) value of Float endes with "F/f".	5) value of Double endes with "D/d".
6) It is a Integral type.	6) It is a floating poi type.
7) Default value is 0.0	7) Default value is 0.0

Page No.
Date

```
* class FloatExample
{
    public static void main (String [] args)
    {
        double a = 127;
        double b = 32768;
        double c = 2147483647;
        double d = 2147483648; //
        double e = 1234567890;
        double f = 12345678901; //
        double g = 12345678901L;
        double h = 123456789012345678 9L;
        double h = 12345678901234567890L; //
        double i = 12345678901234567890f;
        double j = 123456789012345678901234 5678
                   901234567890f; //
        double k = 12345678901234567901234567 8
                   01234567890f;
        double l = 1234567890123456789 0d;
        double m = 1.1F;
        double n = 1.1;
        double o = 1234567890123456789 01234567 89
                   0123456789 0d;
        double o = 12345678 9 . . . . 309d;
        double o = 12345678 . . . . 310.D; //
    }
}
```

Page No.
Date

Float. -

1) Human body. —

```
class Float

    p.s.v.m. (s[]a)
    {
        float temp = 95.24f;
        float height = 5.4f;
        float weight = 55.20f;
        float eyeNo = 1.28f;
        float hemo = 10.48;

        s.o.p. ("Body Temp :" + temp);
        s.o.p. ("Height :" + height);
        s.o.p. ("Height :" + weight);
        s.o.p. ("Eye Number :" + eyeNo);
        s.o.p. ("Hemoglobine :" + hemo);
```

O/P: → Body Temp = 95.24
Height = 5.4
Height = 55.20
Eye Number = 1.28
Hemoglobine = 10.48

Page No.
Date

2) Distance.—

```
class FloatExample
{
    public static void main (String [] args)
    {
        float indiaAme = 1248932.80f;
        float indieuro = 6729338.98f;
        float indiCan = 9386248.11f;
        float indiGer = 657893.27f;
        float indiCol = 768902218.92f;

        S.o.p("Distance from India to America:"+ IndiaAme);
        S.o.p("Dist from India to Europe:"+ indieuro);
        S.o.p("Dist from India to Canada:"+ indican);
        S.o.p("Dist from India to Germani:"+ indiGer);
        S.o.p("Dist from India to Colombia:"+ indicol);
    }
}
```

O/P → Distance from India to America = 1248932.80
Distance from India to Europe = 6729338.98
Distance from India to Canada = 9386248.11
Distance from India to Germani = 657893.27
Distance from India to Colombia = 768902218.92

Page No.
Date

* Double. —

1) Solar Weights. —

```
class doublee
{
    p.s.v.m (S[]args)
    {
        double mars = 639284234698800.90;
        double sun = 248364123456789011 13.14
        double moon = 1764329021207689.22;
        double stars = 97640010639021207 6.98;
        double earth = 90212076 89 97640010 63.9
        s.o.p.("Weight of Mars:" + mars);
        s.o.p.("Weight of sun:" + sun);
        s.o.p.("Weight of Moon:" + moon);
        s.o.p.("Weight of star:" + stars);
        s.o.p.("Weight of Earth:" + earth);
    }
}
```

O/p. → Weight of Mars = 639284 2346 98800.90
Weight of sun = 2483641 2345 6789011 3.1
Weight of Moon = 1764 329021 207689.22
Weight of Star = 97640 0106 3902 1207 6.9
Weight of Earth = 90212078 8997 640010 63.

2) Natural type. —

```
class doubleex
{
    p.s.v.m.(S[] args)
    {
        double trees = 6498902120768990712 7.23d;
        double birds = 97640010 63 93098155 97.90d;
        double animals = 904942164 5902120.20d;
        double river = 93098155 97 9021.20 93 13d;
        double mountains = 106315 5972298.30d;
    }
}
```

O/P. →

```
S.o.p.("No of Bo trees :" + trees);
S.O.p.("No of Birds :" + birds);
S.o.p.("No of Animals :" + animals);
S.o.p.("No of Rivers :" + river);
S.o.p.("No of mountains :" + mountains);
```

O/P. → No of trees = 6498902120768990712 7.23;
No of Birds = 97640010 63 93098155 97.90;
No of Animals = 904942164 5902120.20.
No of Rivers = 93098155 97 9021 20 93 13
No of mountains = 106 3 15 15 97 2298.30

Page No.
Date

Combine datatype. —

1) Bank. —

```
class bank
{
    p.s.v.m(s.[] args)
    {
        byte boardMembers = 97;
        short branches = 5467;
        int profitDay = 123456789;
        long acco = 1296786405 11;
        float profitYear = 212789345601 34.15f;
        double money = 90212078899764001063904 9.8

    S.o.p("No of Board Members:"+ boardMembers);
    S.o.p("No of Branches:"+ branches);
    S.o.p("Profit/Day:"+ profitDay);
    S.o.p("~~Profit/Year:~~"+"Accounts:"+ acc);
    S.o.p("Profit/Year:"+ profitYear);
    S.o.p("Total Money:"+ money);
    }
}
```

O/P → No of Board Members = 97
No of Branches = 5467
Profit/Day = 123456789
Accounts = 1296786405 1
Profit/Year = 212789345601 34.15
Total Money = 90212078899764001063904 9.8

Page No.
Date

2) Classmate combine —

```
class Classmate
{
    p.s.v.m.(S[]a.)
    {
        byte boardMembers = 28;
        short emplo = 2814567;
        int sellsPerDay = 62345834;
        long sellsPerYear = 6345676789 10L;
        float custoShare = 33345678 90.23f;
        double income = 9021207 8 9 9 7 6 4 0 0 1 0 6 3.97d;

        s.o.p.("No of Board Members:"+ boardMembers);
        S.o.p.("No of Employee:" + emplo);
        s.o.p.("No of cells/day:"+ sellsPerDay);
        s.o.p.("No of cells/year:"+ sellsPerYear);
        S.o.p.("No of customer shares :"+ custoshare);
        s.o.p.("Income:" + income);
    }
}
```

O/P → No of Board Members = 28;
No of Employee = 2814567
No of cells/day = 62345834
No of cells/Year = 6 3 4 5 6 7 6 7 8 9 0 L;
No of customer shares = 3 3 3 4 5 6 7 8 9 0 . 2 3 f;
Income = 9 0 2 1 2 0 7 6 8 9 9 7 4 7 0 0 1 0 6 3 . 9 7 d;

Page No.
Date

* Precision.—

* W.A.P. to check the precision value of float & double. —

→ class precision
{
p.s.v. m- (s. []a)
{
float a= 513.0f;
double b = 513.0;

s.o.p.("a");
s.o.p.(b);

o/p. → 1.6666666
1.666666666667

7) Char

size → 2 bytes.
Range → min = 0
max = 65535.

* W.A. java program to find range of char data-type using Wrapper class method.

→

```
class CharExample
{
    p.s.v.m.(s.[] args)
    {
        S.o.p.("min:" + (int) character.MIN_VALUE);
        S.o.p.("max:" + (int) character.MAX_VALUE);
    }
}
```

O/P → min: 0
max: 65535.

*

*

```
{
    p.s.v.m(s[] args)
    {
        char a = 'a';
        char b = 65;
        char c = 65535;
        char d = 65536; //
    int ~~char~~ e = 1;
```

```
        char d = e;
        char f = 'a';
        int g = f;
    }
}
```

Note. —

- char is a short for character in Java
- most of the time, the actual data store in the char data type dosen't take a more than 8 bits.
- The reason Java allows 16 bits is so that a characteres in all human languages can b represented in it.
- When we store a character it will be con rted into its corresponding ASCII value and then that ASCII value is converted in its binary form.
- It means we can store a ASCII value inside c char container directly

Ex. — char a = 127; //

- characters must be ~~enclo~~ always enclosed within single quotes (' ').

8) Boolean.

size 1 bit

~~Range~~

- It can contain Only two possible values. i.e. True or False.
- The values for Boolean data type are key words, It means we have to specify them in lower case without any quotes.

- class BooleanExample
{
    psvm.(S[]args)
    {
boolean a = true;
boolean b = false;
boolean c = 0; // CTE (cannot contain int to Boolean)
boolean d = 1; // CTE
boolean e = "true";
boolean d = 't'; // CTE
S.o.p.(a);
}
}

Page No.
Date

* ~~String~~

* Comment. —

- The Java comments are the ~~comments~~ statements in the program that are not executed by the compi and the Interpriter.

- Why do we need to use comment?
→ 1) comments are used to add details of the code for easy Understanding of the given task performed.
2) It makes easy to maintain the code and find errors easily.
3) comments can be used to provide an information about any Java components or statements.

Types of comments. —
① Single line comment. —
- is used to comment only one line of the code.
- Single line comment starts with two forward slashes (//)
- Any text or a statement in front of (//) is not executed.
Ex. — // Mansi.

② multi line comments. —

- multiline comment is used to comment multiple line code.
- It can be used to explain a complex code or too comment multiple line of a code at a time.
- multiline comments are placed between forward slash & Astric (*) and Astric forward slash (*/)

Ex.- /* Mansi
Patil */

* Scanner Class.—

- Reading input from the console enables the program to accept the input from the user.
- To read the data dynamically from the user we use a built in class called as 'Scanner'.
- scanner class is used to read data dynamically during execution of a program.
- scanner is built-in class present in java.util package.
- Java uses system.out to refer to the standard output device & system.in to the standard input device
- By default, the output device is the display monitor & the input device is the keyboard.
- To perform console output, you simply use the println method to display a primitive value or a string to the console.

Page No.
Date

~~Operator.~~

- Console input is not directly supported in Java, but you ~~co~~ can use the scanner class to create an object to read input from system.in.

```
import java.Util.scanner;
class UserInput
{
    p.s.v.m.(S.[]args)
    {
// object creation & init of variable of scanner class
Scanner sc = new Scanner(System.in);
```

scanner variable (non-primitive)

constructor of Scanner class.

object creation (keyword)

```
int num1 = sc.nextInt();
int num2 = sc.nextInt();
int op = num1 + num2;
S.o.p.("Addition of 2 num is : " + op);
  }
}
```

O/P → Addition of 2 num is : 10
20
30

import
class PersonalInfo
{
_____ ( )
{
scanner sc = _____
- print Enter your name : sc.nextline();
- string name : sc.next
- " " age : sc.nextFloat();
- s.o.p. enter 4 D Branch : sc.next.

Page No.
Date

* steps to use a Scanner class.

step 1. → Import Scanner class.
We can input class with the help of keyword called import java.util.Scanner;
Import statement must be the first statement of a java program.

Step 2 :→ Create the object of Scanner class with the help of a scanner constructor.
Scanner input = new Scanner (System.in);
The scanner object gives a scanner object reference which should be stored inside a scanner type variable (input).

Step 3 :→ with the help of Scanner object reference use the method of scanner class to read data dynamically from the User.

Page No.
Date

* methods of scanner class objects-

Data type	Method Name	Return type
byte	nextByte();	byte
Short	nextShort();	short
int	nextInt();	int
long	nextLong();	long
float	next Float();	float
double	next Double();	double
char	next().charAt(0);	char
boolean	next Boolean();	boolean
String	next(); / nextLine();	string

Q: WA Java program to take i/p from used to prin- the given information. (Nam, Native, YOP, Branch University, CGPA, do you have any backlog).

```
import java.util*;
class Scannerexample {
Public static void main (String [] args) {
Scanner sc = new Scanner (System.in);
system.out.print ("Enter your name:");
String name = sc.nextline();
System.out.print ("Enter your Native:");
String ~~name~~ Nativeplace = sc.next();
System.Out.print ("Enter your YOP:");
int YOP = sc.nextInt();
System.out.print ("Enter your Branch:");
String Branch = sc.next();
```

Page No.
Date

```
System.out.print("Enter your University:");
        string University = sc.next();
System.out.print("Enter your C.G.P.A");
        Float CGPA = sc.nextfloat();
System.out.print("Do you have any backlog?:");
        boolean backlog = sc.next Boolean();

System.out.println("Your name is:" + name);
S.o.p ("Your Native is:" + nativeplc);
S.o.p. ("Your YOP is:" + YOP);
S.o.p ("Your Branch:" + branch);
S.o.p. ("Your University:" + university);
S.o.p. ("Your CGPA:" + cgpa);
S.o.p. ("Your backlog:" + backlog);
```

O/P. ⟶

Note — If the scanner class fails to read the data or ~~at~~ skip that statement we can fixed that problem using multiple ways —

1) Scanner variable. nextline();
2) Scanner variable. reset();
3) Creating a new scanner object & using that reference variable for the further inputs.

Note —
— We can create a class with a predefined class but it is not recomended

```
* import java.util.Scanner;
class charecterDate
{
    public static void main(string[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your Name:");
        String name = ~~s.c~~ sc.next();
        System.out.print("Enter a index value to fetch the char");
        byte indx = sc.nextByte();
        char ch = name.charAt((indx-1));
        System.out.println("charecter is:"+ch);
    }
```

O/P → } Enter your Name : mansi
Enter a index value to fetch the char 2
character is : a

32762

Date

- Operator —
Operators are the special Symbol which is used to perform operation on Operends.

- Operent - variables or Literals.

- classification of operators —

a) based on the no of Operants —

1) Unary operator. Increment Decrement
2) Binary operator.
3) Ternary operator.

b) classification based on their operation —

1) Arithmatic Operators. +, −, *, /, %
2) Compound assignment Operator += , −= , *= , /= , %=
3) Assignment operator =
4) logical operator. &&, !, ||
5) Relational operator == , != , < , > , >= , <=
6) conditional Operator. OP1 ? OP2 : OP3
7) Bitwise Operator.
8) Miscellaneous operator.

1) Arithmatic Operators —
1) Addition +
2) Substraction −
3) multiplication *
4) division /

Page No.
Date

5) modulous %.

1) WAJP to take input from the user to perform ad

```
import java.util.*;
class Addition
{
        p.s.v.m(s[]args)
        {
            Scanner sc = new Scanner (System.in);
            System.out.print("Enter value 1:");
            int num1 = sc.nextInt();
            System.out.print("Enter value 2: ");
            int num2 = sc.nextInt();
            int op = num1 + num2;
            System.out.print("Addition is :"+ op);
        }
}
```

O/P → Enter value 1 : 10
Enter value 2 : 15
Addition is : 25.

2) WAPJP to show the result of $\frac{9.5 * 4.5 * 2.5 * 3}{45.5 * 3.5}$

→

```
import java.util.*;
class MultiPli
{
        psvm(s[]a)
        {
            Scanner no = new Scanner (system.in);
```

```
double a = 9.5;
double b = 4.5;
double c = 2.5;
double d = 3;
double e = 45.5;
double f = 3.5;
double g = a*b*c*d/e*f;
S.o.p ("9.5*4.5*2.5*3/45.5*3.5:"+g);
}
}
```

O/P. → 9.5*4.5*2.5*3/45*3.5 : 24.663461538461537.

3) WATP to display the area & perimeter of a circle that has a radius of ~~5.5~~ using the following formula-

→
```
import java.util.Scanner;
class CircleRadius
{
        psvm(s[]a)
    {
        Scanner sc= new Scanner (System.in);
        final double pi = 3.14;
        S.o.p. ("Enter Radius :");
        double radius = sc. nextDouble();
        double perimeter = 2* pi* radius;
        double area = pi* radius*radius;
        S.o.p("Radius of circle is :" + radius + "cm");
        S.o.p.("perimeter is : "+area+" cm2);
        S.o.p.("area is :"+ area+" cm2");
    }
}
```

Page No.
Date

O/P. → Enter Radius : 8
Radius of a circle is : 8.0cm
perimeter is : 200.96 cm2
area is : 200.96cm2

4) Area of a Rectangle —

→
```
import java.util.Scanner;
class Rectangle
{   ps.vm.(s[]a){
    Scanner sc = new Scanner (System.in);
    System.out.print ("Enter width :");
    Float num1 = sc.nextFloat ();
    System.out.print("Enter Height :");
    float num2 = sc.nextFloat ();
    Float op = num1 + num2;
    System.o.p("Area of Rectangle is:" + op);
  }
}
```

O/P → width : 4
Height : 5
Area of Rectangle is 20.0

5) Area of Δ. —

→
```
import java.util.Scanner;
class Triangle
{
    p s v m (s [] args)
    {
```

Page No
Date

```
        Scanner sc = new Scanner(System.in);
        System.o.p.("Enter Base:");
        Float num1 = sc.nextFloat();
        S.o.p.("Enter Height :");
        Float num2 = sc.nextFloat();
        Float op = num1 + num2/2;
        System.out.print("Area of Triangle is:" + op);
    }
}
```

O/P → Enter Base : 5
Enter Height : 6
Area of Triangle is : 15.0

11) WAP that reads a celsius deg. in a double value from the user, then converts it to Fahrenheit & displays the result.

→

```
import java.util.*;
class Degree
{
    p s v m(s[]a)
    {
    Scanner sc= new Scanner(System.in);
    System.out.print("Enter celsius Degree: ");
    double celsisius = sc.next Double();
    double Fhr = 9.0/5* celsius +32;
    System.out.println(" Result of celsius to
     Fahrenheit value conversion is : " +fhr);
    }
}
```

O/P. → Enter celcious Degree : 43
Result of celcius to Fahrenheit value conversion is 109.4

12) that a program that reads in the radius & length of cylinder & computers the area & volume using the follo formulas. —

→
```
import java.util*;
class Cylinder
{
        psvm (s[]a)
        {
        final double pi = 3.14;
        Scanner sc = new Scanner (System.in);
        System.out.print ("Enter radius: ");
        double radius = sc.nextDouble ();
        System.out.print ("Enter height : ");
        double length = sc.nextDouble ();
        double area = pi* radius* radius;
        double volume = area* length;
        System.out.println ("Area of cylinder : " + ar
                                    +"cm"));
        s.o.p("volume of cylinder: " + volume+"c
        }
}
```

O/P. → Enter radius : 13.5
Enter height : 12.5
Area of cylinder : 572.265 cm
volume of cylinder : 7153.3125 cm

Page No.
Date

9) Assume a runner run 14 kilometers in 45 min & 30 s
write a p. to d. the avg. speed of miles/hr

→
```
import java.util.*;
class speed
{
        psvm(s[]a)
        {
         Scanner sc= new Scanner(System.in);
         S.O.P.("Enter kilometers:");
         double kilometers = sc.nextDouble();
         double miles = kilometers/1.6;
         double formula = (45.5*60.0+30.0)/
                                (60.0*60.0);
         double milesperHour = miles/formula;
         S.O.P("Avg speed in miles/hour:"+milesPerHour+" miles/hr");
        }
}
```

O/P → Enter kilometers:
Avg speed in miles/hour: 11.538461538461538

6) WAJP to solve the eqn $(a+b)^2$ using Scanner class —

→
```
import java.util.Scanner;
Class Equation
{
```

```
public static void main( String [] args)
{
  Scanner sc= new Scanner (system.in);
  S.o.p.( "Enter 'a' value :");
   int a = sc.nextInt();
  System.o.p(" Enter 'b' value :");
  int b = sc.nextInt();
  int equ = a*a+2*a*b+b*b;
   S.o.p. (" [(a+b)2] = " + equ);
  }
}
```

O/p → Enter 'a' value : 2
Enter 'b' value : 3
[(a+b)2) = 25.

8) First Digit.

→
```
import java.util.*;
class First
{
    psvm(S[]a)
    {
      Scanner sc = new Scanner (system.in);
      S.o.p.( "Enter a 3 digit no: ");
      in no = sc. next Int();
      int a = no/100;
      S.o.p. ("First Digit is :" +a);
    }
}
```

O/P → Enter 3 digit number : 123
first digit is : 1

10) population Projection—

→
```
class population
{
    psvm (S[] a)
    {
      s.o.p. ("current population is 312032486");
      double intpop = 312032486;
      double number of-second-InYear = 3153600;
      double birthPerYear = numberOfSecondInYear/7;
      s.o.p. ("birthPerYear :" + birthsPerYear);
      double deathsPerYear = numberofsecondInYear/13;
      s.o.p. ("deathsPerYear :" + deathsPerYear);
      double immigrantsPerYear = numberofsecondsInYear/
                                                45;
      s.o.p. ("immigrantsPerYear :" + immigrantsPer
                                        Year);
      double ratePerYear = birthsPerYear + immigrant
                                  perYear;
s.o.p ("ratePerYear :" + ratePerYear);
s.o.p (— 11 — 2 popl^n is :" + (int pop + (2 * ratePerYear)));
s.o.p. (— 11 — 3 popl^n is :" + (int pop + (3 * ratePerYear)));
s.o.p. (— 11 — 4 popl^n is : + (int pop + (4 * ratePerYear)));
s.o.p. (— 11 — 5 popl^n is : (int pop + (5 * ratePerYear)));
    }
}
```

O/P → current population is - 312032486
birth Per year : 4505142.857142857
death Per Year : 2425846.1538461 54
immigrant Per Year : 700800.6
rate Per Year : 2780096.7032967033

End of year 1 popl^n is : 3.1481258270329 67E8
End of year 2 pop^n is : 3.175926794065934E8
End of year 3 pop^n is : 3.2037277610989 01E8
End of year 4 pol^n is : 3.2315287281318 68E8
End of year 5 popl^n is : 3.2593296951648 35E8

7) last digit of a Number -

→
```
import java.util.*;
class lastDigit
{
  public static vm(S[]a)
  {
    scanner sc = new scanner(system.in);
    S.o.p.("Enter a number:");
    int num = td.. sc.nextInt();
    int num2= num % 10;
    s.o.p.l("Last Digit is :"+num2);
  }
}
```

O/P → Enter a Number : 123
Last Digit is 3.

2) Assignment Operator. —

- Assignment Operator is a binary operator.
- Assignment operator is used to stored value into a variable.

Syntax :— variable = value / expression / variable.

Note. — Assignment operator work from Right to left.

Rule 1 :— An assignment operator should have a variable in LHS else we get a compile time error.

Example. — 10 = 10 ✗
int a = 10 ✓

Rule 2. — The type of value to be stored should be same as the type of variable, if it is not same there are two possibilities.

3) Relational Operator. —

Relational operators are the operators used to create a relationship and compare the values of two operands.

For example :— there are two no. 5 & 15 & we can get the greatest number using the

greater than operator (>) that returns 1 as the

the return type of relational operator is a Boolean value. (True or False).

- Types of relational Operator (Theory)

~~one program related to == operator~~

1) Equal To Operator (==). —
It is used to ~~co~~ compare both operands and returns 1 if both are equal or the same, a 0 represents the operands that are not equal

syntax :— Opr 1 == Opr 2;

2) Not Equal To operator (!=)

The Not Equal To Operator is the opposite of the Equal To operator and is represented as the (!=) Operator. The Not Equal To Operator compares two operands and returns 1 if both operands are not the same; otherwise, it returns 0.
syntax : Opr1 != Opr2;

Page No.
Date

3) Less than Operator (<) —
It is used to check whether the value of the left operand is less than the right operand, and if the statement is true, the operator is known as the Less than Operator.
Syntax: A<B;

4) Greater than Operator (>) —
The operator checks the value of the left operand is greater than the right operator, and if the statement is true, the operator is said to be the Greater Than Operator.
Syntax: A>B;

5) Greater than Equal To Operator (>=)
The operator checks whether the left operand's value is greater than or equal to the right operand. If the statement is true, the operator is said to be the Greater than Equal to Operator.
Syntax: A>=B;

6) Less than Equal To Operator (<=) —
The operator checks whether the value of the left operand is less than or equal to the right operand, and if the statement is true, the operator is said to be the Less than Equal to Operator.
Syntax: A<=B;

Page No.
Date

* Example. —

```
class Demo
{
    psvm(S[]a)
    {
    S.o.pln.(10==10);
    s.o.p.(10==10.00);
    S.o.p.(10==10.001);
    S.o.p.(97==95+2);
    S.o.p.(97=='a');
    // S.o.p.(true==1); // CTE
    s.o.p.(true==true);
    s.o.p.(true==false);
    // S.o.p.(true=="true"); // CTE
    S.o.p.("mansi"=="mansi");
    S.o.p.("Mansi"=="mansi");
    }
}
```

O/P. → True
True
False
True
True
True
False
True
False

Page No.
Date

15) * Sum of digits —

```
Class Demo
{
        psvm (S[]a)
        {
        Scanner sc = new Scanner (System.in);
        S.o.p ("Enter a number:");
        int num = sc.nextInt();
        int op = 0;   // 0 3
        int rem = num % 10;
        op = rem;
        num = num / 10;   // 123/10 = 12.
        rem = num % 10;   // 12 % 10 = 2
        op = op + rem;   // 3+2
        num = num / 10;   //
        op = op + num;
        S.o.p. ("sum of digit is : "+ op);
        }
}
```

o/p. → Enter a number = 123
sum of digit is : 6.

* reverse number. —

```
import java.util.scanner;
class revnum
{
        psvm(S[]a)
        {
        Scanner sc = new Scanner (System.in);
```

Page No.
Date

```
s.o.p.("Enter 3 digit Number :");
int num = sc.nextInt();
int rev = 0;
int rem = num % 10;
rev = rem;
num = num / 10;
rem = num % 10;
rev = rev * 10 + rem;
num = num / 10;
rev = rev * 10 + num;
s.o.p.("The Number is :" + rev);
}
}
```

O/P. → Enter 3 digit Number : 321
The Number is : 123

~~Why can't we~~

* WAJP related to Relational Operator. —

```
{
    S.O.pln(10 > 12);   // False
    S.O.pln (10 > 10);  // False
    S.O.pln (10.01 > 10.00);  // true.
    // S.O. pln (true > false);  //CTE (Bad operands)
    // S.O.pln (true < false);
    // S.O.pln (true >= false);
    S.O. pln (10 >= 10);
    S.O.pln ('a' != 97);
    S.O.pln ('a' != 'b');
    S.O.pln (1 != 0);
    S.O.pln (0 != 0);
    S.O.pln (2 > 0);
    S.O.pln (true == !false);
    S.O.pln (true != false);
    S.O.pln (true != true);
}
```

O/P. → False
false
true
~~EF~~

Page No.
Date

* Formula to ~~store the~~ find the data type to store the resultant Output. —

output operant = max data type
data type

int
operant 1
data type
operant 2
data type

Ex. – 1

```
byte a = 1;
byte b = 1;
int c = a+b;  ✓        byte c = a+b;  X
```

Ex. – 2

```
short a = 1;
short b = 1;
short c = a+b;  X      int c = a+b;  ✓
```

Ex. – 3

```
int a = ~~24~~ 2147483647;
int b = 2147483647;
int c = a+b;  // output = -2.
```

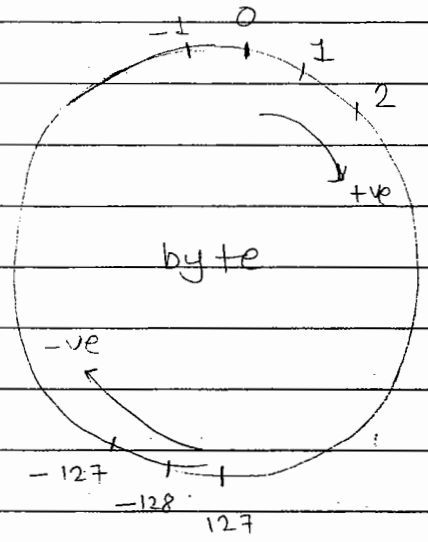
Ex. – 4

```
float ~~short~~ a = 123.....39f;
float b = 123.....39f;
float c = a+b;  ✓
int c = a+b;  X
double c = a+b;  ✓
```

Page No.
Date

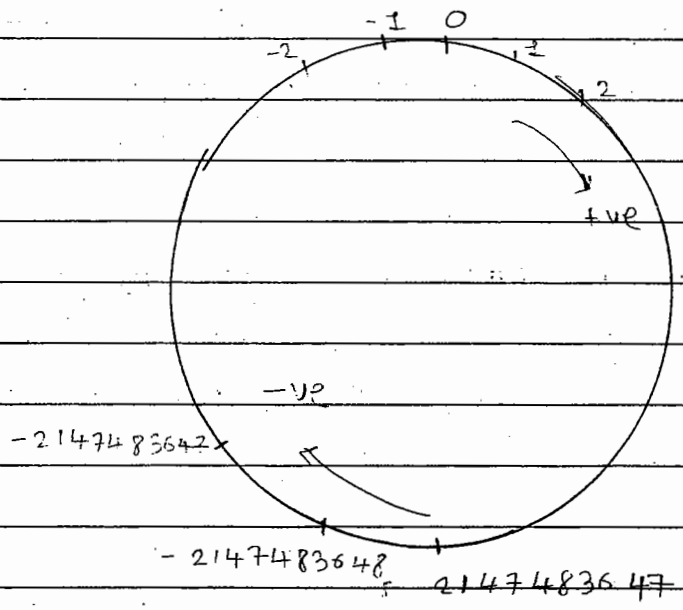
* Data Representaction. —

1) byte. —




```
byte a = 127;
a + = 1;
S.O.P. (a); // -128.
```

2) int. —




```
int a = 2147483647;
a + = 2;
S.O.P. (a) = -2147483647
```

$a + = 3$

when there is data loss use compound assignment operator

* Compound Assignment Operator. —

Compound assignment operator is a binary operator. compound assignment operator is used to a value into a variable by updating the older value from it.

Different types of compound operator :
$+=$, $-=$, $/=$, $*=$, $\%=$

Compound assignment operator, perform operation specified by the additional operator and then assign the result to left operator

Syntax. — variable += expression
It is very similar to var = var + expression.

* Compound Assignment Operator —

Operator	Name	Example	Equivalent
+=	Addition assignment	i += 8	i = i+8
-=	Subtraction assignment	i -= 8	i = i-8
*=	Multiplication assignment	i *= 8	i = i*8
/=	Division assignment	i /= 8	i = i/8
%=	Reminder assignment	i %= 8	i = i%8

Page No.
Date

* Increment and Decrement Operators. —

- This are Unary Operator.
Increment Operator (++)

pre Increment are classified as:
1) pre Increment Operator.
2) Post Increment Operator.

1) pre Increment Operator. —
If a variable is prefix with increment Operator then it is known as pre-increment operator.

- Syntax : ++variable.
- operation:
- It updates (adds one value to the old value by 1)
- Then it uses the updated value.

2) Post Increment Operator. —
If a variable is suffixed (after) with increment Operator then it is known as post increment

Page No.
Date

13) (convert feet into meters)
Write a program that reads a number in feet, converts it to meters, and display the result. on foot is 0.305 meter.

→
```
import java.util.Scanner;
class feet
{
    p.s.v.m.(S[]a)
    {
        Scanner sc = new Scanner (System.in);
        s.o.p.("Enter a value of feet : ");
        double num = sc.nextDouble();
        double op = num * 0.305;
        s.o.~~pln~~ pln.("Feet value in meter is : " + op)
    }
}
```

o/p → Enter a value of feet : 43
feet value in meter is : 13.115.

14) (convert pounds into kilograms)
WAJP that converts ~~into~~ pound into kg. The P prov the user to enter a number in pounds, convert it to kg, & display the result. One pound is 0.454 kg.

→
```
import java.util.Scanner;
class pound
    {
        p.s.v.m(S[]a);
```

```
{
        Scanner sc = new Scanner(System.in);
        S.o.p("Enter number in pounds :");
        double num = sc.nextDouble();
        double op = num * 0.454;
        S.o.pln.("value is :" + op);
    }
}
```

O/P. → Enter number in pounds : 87
value is : 39.49800

15) (Financial application: calculate tips)
WAP that reads the subtotal & the gratuity rate, then computes the gratuity & total.

→
```
import java.util.Scanner;
class tips
{
    psvm(S[]a)
    {
        Scanner sc = new scanner(System.in);
        S.o.p.("Enter the subtotal :");
        float subt = sc.nextFloat();
        S.o.p.("Enter the tip rate :");
        Float tipr = sc.nextFloat();
        float op = subt + (subt * tipr / 100);
        S.o.pln.("The tip is : "+op" and total
        is : "+op);
    }
}
```

Page No.
Date

O/P. → Enter the subtotal : 34
Enter the tip rate :12
The tip is : 38.08 and total is : 38.08

16) ~~(5~~ Sum the digits in an integer
Write a prog. that reads an integer betn 0 & 10 and adds all the digits in the integer.

→

17) Find the number of years.
WAJP that prompts the user to enter the minutes

→
```
import java.util.Scanner;
class year
{
        psvm(S[]a)
        {
            Scanner sc = new Scanner(System.in);
            s.o.p.("Enter the number of minutes :");
            double num = sc.nextDouble();
            double op = num/1440;
            S.o.pln(op + " minutes is approximately
        }
}
```

O/P. → Enter the number of minutes : 2735
minutes is approximately. 1.8993055

Page No.
Date

18) Physics : acceleration.
Average acceleration is defined as the change of a velocity divided by the time taken to make the change, Formula -

$$a = \frac{V_1 - V_0}{t}$$

WAJP that prompts the user to enter the Starting velocity V_0 in meters/sec. the ending velocity V_1 in met./sec.

→
```
import java.util.Scanner;
class acceleration
{
        p s v m(S[]a)
        {
            Scanner sc = new Scanner(system.in);
            S.o.p.("Enter V0 : ");
            Float num1 = sc.nextFloat();
            S.o.p.("Enter V1 :");
            Float num2 = sc.nextFloat();
            S.o.p.("Enter t: ");
            Float a = num1 - num2 / num3;
            S.o.pln.("The average acceleration is :"
                                    " a);
        }
}
```

O/P. → Enter V_0 : 34
Enter V_1 : 24
The average acceleration is : 32.9999

Page No.
Date

19) Calculating energy.

WAIP that calculates the energy needed to heat wat from an initial temperature to a final temperature. Your ~~w~~ prog. should prompt the user to enter the amount of water in kg & the initial & final tem of the water.

→
```
import java.util.Scanner;
class energy
{
        p.s.v.m.(S[]a)
        {
            Scanner sc = new scanner(System.in);
            S.o.p("Enter the amount of water in kG:"
            Float num1 = sc.nextFloat();
            S.o.p.("Enter the initial temperature:");
            Float num2 = sc.nextFloat();
            S.o.p.("Enter the final temperature:");
            Float num3 = sc.nextFloat();
            Float op = num1*(num-num2)*4184;
            S.o.pln("The energy needed is:"+op);
        }
}
```

O/P. → Enter the amount of water in kG: 22
Enter the initial temperature: 34
Enter the final temperature: 56
The energy needed is: 2025056.0

Page No
Date

20) Physics : finding runway length —
WAJP that prompts the user to enter v in meters/sec. & the acceleration a in m/s^2. & displays the mini. runway length.

→
```
import java.util.Scanner;
class runway
{
    p.s.v.m.(string[] args)
    {
        Scanner sc = new Scanner(System.in);
        s.o.p.("Enter the Acceleration :");
        Float acc = sc.nextFloat();
        s.o.p.("Enter the speed :");
        Float speed = sc.nextFloat();
        Float op = speed* 2/2* acc;
        s.o.pln("The mini runway length for this
        airplane is :" + op);
    }
}
```

O/P → Enter the Acceleration: 23.4
Enter the speed: 123
The mini runway length for this airplane is : 2878.2

21) Financial application : compound value —
WAJP that prompts the user to enter a monthly saving amount & displays the account value after the sixth month.

Page No.
Date

```
→ import java.util.Scanner;
class compValue
{
    psvm(s[]a)
    {
        Scanner sc = new Scanner(system.in);
        s.o.pt("Enter the monthly Saving Amount:");
        double amt = sc.nextDouble();
        double first = ~~sc.next~~ amt*(1+0.00417);
        double second = (amt*first)*(1+0.00417);
        double Third = (amt*second)*(1+0.00417);
        double fourth = (amt*third)*(1+0.00417);
        double fifth = (amt*~~third~~ fourth)*(1+0.00417);
        double sixth = (amt*fifth)*(1+0.00417);

s.o.pln("After first month the amount is: + first);
s.o.pln("After second month the amount is: + second);
s.o.pln("After third month the amount is: + third);
s.o.pln("After forth month the amount is: + forth);
s.o.pln("After fifth month the amount is: + fifth);
s.o.pln("After sixth month the amount is: + sixth);
    }
}
```

O/P. → Enter the monthly Saving Amount: 23
After first month the amount is: 23.09
After second month the amount is: 533.421
After third month the amount is: 12319.84
After fourth month the amount is: 284538.0258
After fifth month the amount is: 6971364.63
After sixth month the amount is: 1.917787

Page No.
Date

22) Computing BMI
WAJP that prompts the user to enter a weight in pounds & height in inches & Displays the BMI.

→
```
import java.util.Scanner;
Class bmi
{
    public s v m(s[]a)
    {
        Scanner sc = new Scanner(system.in);
        S.o.p("Enter weight :");
        Float we = sc.nextFloat();
        S.o.p.("Enter height :");
        Float he = sc.nextFloat();
        Float op = we/he*he;
        s.o.pln.("The BMI is: "+op);
    }
}
```

o/p → Enter weight: 45
Enter height: 564
The BMI is: 45.0

23) Geometry : area of a hexagon. —
WAJP that prompts the user to enter the side of a hexagon & displays its area.

→
```
import java.util.scanner;
class hexagon
{
```

```
p.s.v.m. (s.[]a.)
{
    Scanner sc = new scanner (system. in);
    S.o.p.l("Enter the side :");
    Float side = sc.nextFloat();
    float op = 3*2+1.141*side*side;
    S.o.pln.("The area of Hexagon is : " op);
}
}
```

O/P → Enter the side: 4.5
The area of Hexagon is : 29.10525

24) Financial application: calculate interest
WATP that reads the balance & the annual interest rate & displays the interset of the next month.

→
```
import java.util.Scanner;
class interest
{
    p s v m (s[]a)
    {
        Scanner sc = new scanner (System. in);
        system.out.print ("Enter Balance :");
        float bal = sc.nextFloat();
        s.o.p.("Enter interest");
        float inte = sc.nextFloat();
        Float op1 = bal* (inte/1200);
        s.o.pln. ("The interest is : "+op1);
    }
}
```

O/P. → Enter ballance : 345
Enter interest rate : 232
Enter interest : 66.7

25) Cost of driving —
WAJP ~~a~~ that prompts the user to enter the distance to drive, the fuel efficiency of the car in miles/gallon, & the price/gallon & the displays the cost of the trip.

→
```
import java.util.Scanner;
class driving
{
    p s v m (s[] a)
    {
        Scanner sc = new scanner (system.in);
        S.o.p("Enter the driving distance :");
        Float dis = sc.nextFloat();
        s.o.p.("Enter miles per gallon :");
        Float gal = sc.nextFloat();
        S.o.p("Enter the price per gallon :");
        Float pri = sc.nextFloat();
        Float op = (dis/gal)* pri;
        S.o.pln ("The cost of the driving is :
          $" top);
    }
}
```

O/P. → Enter the driving distance:
Enter miles per gallon:
Enter price per gallon:

Page No.
Date

The cost of the driving is :

* Increment & Decrement Operators. —
This are Unary Operator. —

* Increment Operator (++)
pre increment are classified as :
1) pre increment operator. —
2) post increment operator. —

1) Pre increment operator. —
If a variable is prefix with increment operator then is known as pre-increment operator.
Syntax : ++variable
operation : It updates (adds one value to the old value by 1).
: Then it uses the update value :

2) Post Increment Operator. —
If a variable is suffixed (after) with increment operator then it is known as post increment operator.

Syntax : variable++
- In post increment operator, the given value first is used (substituted)
- After using (substituted) it will then immediately update the variable with an increase of one value to the variable.

Page No.
Date

* Decrement Operator. —
These are unary operator.

Decrement Operator (--)
Decrement Operator are classified into:
1) Pre Decrement Operator. —
2) Post Decrement Operator. —

* Decrement Operator. —
If the variable is prefix (before) with variable then it is known as decrement operator.
- Syntax: --variable
- Operation: first it updates (subtract) one value to the subtracts value by 1.

pre ~~increment~~ decrement Operator. —
If a variable is ~~prefix~~ with ~~increment~~ decrement op. then it is known as pre-Decrement operator.
syntax : ++variable.
Operation
it updates (adds one value to the old value by 1)
Then it uses the updated value.

Page No.
Date

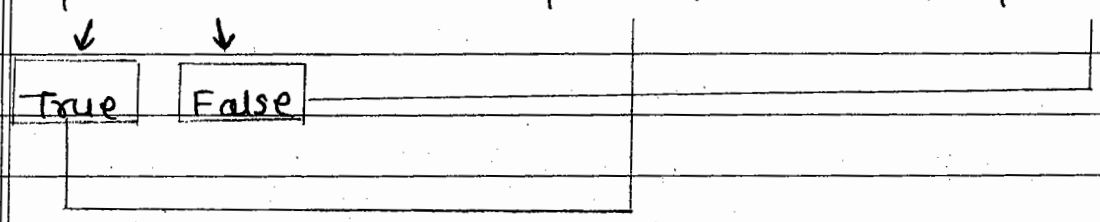
* Increment And Decrement Operator. —

	Operator	Name	Description	Example Cassume i
1)	++ var	pre increment	Increment var by 1, & use the new var value in the statement.	int j = ++i; // j is 2, i is 2
2)	var ++	Post Increment	Increment var by 1, but use the original var value in the statement.	int j = i++; // j is 1, i is 2
3)	-- var	Pre decrement	Decrement var by 1, and use the new var value in the statement.	int j = --i; // j is 0, i is 0
4)	var --	Postdecrement	Decrement var by 1, and use the original var value in the statement.	int j = i--; // j is 1, i is 0.

* Conditional Operator.—

- It is used for decision making purpose.

```
int num = 16
(num % 2 == 0) ? Even / odd.
```



- The condition (operand 1) must be of boolean type.
- If operand 1 written true the control goes to operand 2.
- If operand 1 returns false the control goes to operand 3.
- The result of conditional operator depends on the type of operand 2 & operand 3.

Page No.
Date

```
Class Demo2
{       p s v m (S[]a)
        {
            int a = 2;
            s.o.pln.(a);
            s.o.pln.(a++);
            s.o.pln.(a);
            s.o.pln.(a++);
            s.o.pln.(a);
        }
}
```

O/P → 2
2
3
3
4

```
Class Relational Operator
{
        psvm(string[] args)
        {   int a = 2;
            s.o.pln.(a);          // 2
            s.o.pln.(a++);        // 2
            s.o.pln.(++a);        // 4
            s.o.pln.(a);          // 4
            s.o.pln.(a++);        // 2 (3) 4
            s.o.pln.(a++);        // 4 5
            s.o.pln.(++a);        // 3 7
            s.o.pln.(a++);        // 3 7
```

```
        s.o.pln (a);      //8.
    }
}
```

o/p: →

* class IncrementOperator

```
{
    p s vm(s[] a)
    {
        int a = 2;
        int b = a++;      //2 (3)
        int c = a;        //3
        s.o.pln (a);      //3
        s.o.pln (b);      //2
        s.o.pln. (c);     //3
    }
}
```

* {

```
    p s vm(s[] a)
    {
        int a = 2;
        int b = ++a;      //3
        int c = ++b;      //4
        s.o.pln.(a);      3
        s.o.pln (b);      2
        s.o.pln. (c);     3
    }
}
```

Page No
Date

```
*   int a = 2;
    int b = 3;
    int c = 4;
    s.o.pln.(a++ + ++c + ++b + b++ + a+b+c); //2
    s.o.pln.(a + ++b + a++ + a); //11 ~~//15~~
    s.o.pln.(++a + a + ++b + a+b); //17
    s.o.pln.(a++ + ++a + b++ + ++b + a + b); //2
  }
}

*   ass IncreDecre
{
    p s vm(s[]a)
    {
        int a = 1;
        int b = ++a; //2
        int c = b--; //2(1)
        #
s.o.pln.(++a + a-- + --b + b++ + c-- + a+b+c)
        // 3 + 3 + 0 + 0 + 2 + 2+1+1
    }
}
o/p. → 12

*   import java.util.Scanner;    // Max Number -
class MaxNumber
{
    p s vm(s[]a)
    {
        Scanner sc = new Scanner(system.in);
        s.o.p.("Enter num 1 :");
```

```
        int a = sc.nextInt();
        S.o.p.("Enter num 2 : ");
        int b = sc.nextInt();
        int max = a>b ? a:b;
        S.o.pln.("Max number is :" +max);
    }
}
```

O/P → Enter num 1 : 8
Enter num 2 : 7
max number is : 8.

* Min Number. —

```
import ~~util.ja~~ java.util.Scanner;
class minNum
{
        psvm(s[]a)
        {
        Scanner sc = new Scanner(System.in);
        S.o.p.("Enter num1 :");
        int num1 = sc.nextInt();
        S.o.p.("Enter num2 :");
        int num2 = sc.nextInt();
        int OP = num1<num2 ? num1 : num2;
        S.o.p.("The min num is :" + OP);
        }
}
```

O/P → Enter num 1 : 8
Enter num 2 : 2
The min num is : 2.

Page No.
Date

* Even Odd number. —

```
import java.util.Scanner;
class evenodd
{
        p s vm(s[] a)
        {
            Scanner sc = new Scanner( system.in);
            s.o.p. ("Enter num 1 :");
            int a = sc.nextInt();
            s.o.p. (" Enter num 2 :");
            int b = sc.nextInt();
            int op =(a%2==0) ? " num is odd": "num is e
            s.o.p (" The num op ");
        }
}
```

o/p. → Enter num = 2
num is even
Enter num = 9
num is odd.

* character program. —

```
import java.util.Scanner;
class chare
        {
            p s vm (s[] a)
            {
                scanner sc = new Scanner (system.in);
                s.o.p. (" Enter char value :");
```

```
char = sc.nextcharAt(0);
string op = ((ch >= 65 && ch <= 90)) || ((ch >= 97 && ch <= 122));
        ? is alphabet : Not alphabet;
s.o.p.(op);
```

o/p. → Enter value : E
is alphabet

Enter value : m
is alphabet

Enter value : 2
Not alphabet.

Completed

Srikant
13/10/2023

JSPIDERS JSPIDERS

1) WAJP to check the char. is in upper case or in lower case. —

```
import java.util.Scanner;
class upperlower
{
        p s v m(s[]a)
        {
        scanner sc = new Scanner (system.in);
        s.o.p.("Enter the char : ");
        char ch = sc.next().charAt(0);
        string op = ((ch >=65 && ch <=90) || (ch >=
        97 && ch <=127)) ? ((ch >=65 && ch <=90) ? "
        Upper case" : "Lower Case") : "Not an Alphab
        s.o.pln(op);
        }
}
```

o/p. ⟶ Enter the char : A
uppercase.

Enter the char : m
lowercase.

2) WAJP to check the char is vowel or consonants

```
import java.util.Scanner;
class vocon
{
        p s v m (s[] a)
        {
```

```
        Scanner sc = new Scanner(System.in);
        S.o.p. ("Enter char : ");
        char ch = sc.next().charAt(0);
        string op = (ch == 'a' || ch == 'e' || ch == 'o' || ch == 'i' ||
                    ch == 'u') ? "The char is vowel" : "The
                    char is consonant";
        S.o.p. (op);
    }
}
```

o/p. → Enter char : a
The char is vowel

Enter char : M
The char is consonant.

3) * WAJP to check the no. is +ve or -ve.

```
import java.util.Scanner;
class posneg
{
        p s vm (S[] a)
        {
            Scanner sc = new Scanner(System.in);
            S.o.p. ("Enter Number :");
            int a = sc.nextInt();
            int op = (a > 1) ? "-ve" : "+ve";
            S.o.p (op);
        }
}
```

Page No.
Date

O/P. → Enter Number : -1
-ve.

Enter Number : 8
+ve.

4) WAIP to ~~d~~ check max no among 3 digits.

```
import java.util.Scanner;
class maxno
{
        p s v m (s[] a)
        {
          Scanner sc = new scanner (System.in);
          s.o.p. ("Enter num 1 :");
          int a = sc.nextInt();
          s.o.p. ("Enter num 2 :");
          int b = sc.nextInt();
          s.o.p. ("Enter num3 :");
          int c = sc.nextInt();
          ~~s.o.p. (:~~
          int op = a>b ?((a>b)?a:b):((b>c)?b:c);
          s.o.p. ("The max no is :" + op);
        }
}
```

O/P. → Enter num 1 : 8
Enter num 2 : 7
Enter num 3 : 2
The max no. is : 8

Page No.
Date

5) WAJP to check mini no. among 3 digits.

```
import java.util.Scanner;
class minadigit
{
        p s v m (S [] a)
        {
          Scanner sc = new Scanner (system.in);
          s.o.p. ("Enter num 1 :");
          int a = sc.nextInt();
          s.o.p. ("Enter num 2 :");
          int b = sc.nextInt();
          s.o.p. ("Enter num 3 :");
          int c = sc.nextInt();
          int op = a<b?((a<b)?a:b):((b<c)?b:c);
          s.o.p. ("The mini no is :" + op);
        }
}
```

O/P → Enter num 1 : 10
Enter num 2 : 12
Enter num 3 : 16
The mini no is : 10.

Page No.
Date

6) WAJP to check the 1st charecter ~~of~~ the ~~year~~ your name is a vowel or a consonants.

→
```
import java.util.Scanner;
class name
{
        p s v m(s[]a);
        {
        Scanner sc = new scanner(System.in);
        s.o.p.("Enter your name :");
        ~~Scanner sc~~
        char ch = next().charAt(0);
        string op = (ch =='a'|| ch =='e'|| ch =='i'||
                    ch =='o'|| ch =='u')?" Name
        starts with vowel":" Name Start wit
        consonaret";
        s.o.p.(op);
        }
}
```

o/p → Enter your name : Mansi
Name starts with consonant.

Enter your name : isha
Name starts with vowel.

Page No.
Date

7) WAJP to check the Number is +ve, -ve or 0.

→
```
import java.util.Scanner;
class number
{
        p s v m(s[]m)
        {
        Scanner sc = new scanner(system.in);
        S.o.p. ("Enter a Number :");
        int a = sc.next Int();
        string op = (a>0)?"Number is +ve" :
        (a<0) ?" Number is -ve" : "Number is
        zero";
        s.o.p. (op);
        }
}
```

O/P → Enter a number : 2
Number is +ve.

Enter a number : -8
Number is -ve

Enter a number : 0
Number is 0.

Page No.
Date

* Logical Operators. —

logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output based on a certain logic.
All digital systems can be constructed by only the basic logic gates. These basic gates are called the AND gate, the OR gate, and the Not gat

- class LogicalAndOperator

```
{
    p s v m(S[] a)
    {
        s.o.pln(true && true);      // True
        s.o.pln (false && true);    // false
        s.o.pln (true && false);    // false
        s.o.pln (false && false);   // false
    }
}
```

- class LogicalOrOperator

```
{
    p s v m(S[] a)
    {
        s.o.pln(true || true);      // true
        s.o.pln (false || true);    // true
        s.o.pln (false true || false); // true
        s.o.pln (false || false);   // false
    }
}
```

```
- class LogicalNotOperator
{
    p s v m(S[] a)
    {
        s.o.pln(!true);        // false.
        s.o.pln(!false);       // true.
        s.o.pln(!(!false));    // false.
    }
}
```

* And Truth Table.—

Input 1	Input 2	output
True	True	True
True	false	false
false	True	false
false	false	false

* OR Truth Table.—

Input 1	Input 2	output
True	True	True
True	false	True
false	True	True
false	false	false

Page No.
Date

* Control Flow Statements. —

The Statements which are use to control the execution flow of the Java program ~~Statements~~, are known as control flow statements

Types of control flow statements —

1) Conditional Statements / Decision making statement
2) looping statement.
3) Branching statement / Control transfer statem

* Decision makering statements / conditional Statem

The statements which help us (programmer) to decide wether to execute a set of Instruction or not.

Types of conditional statements. —

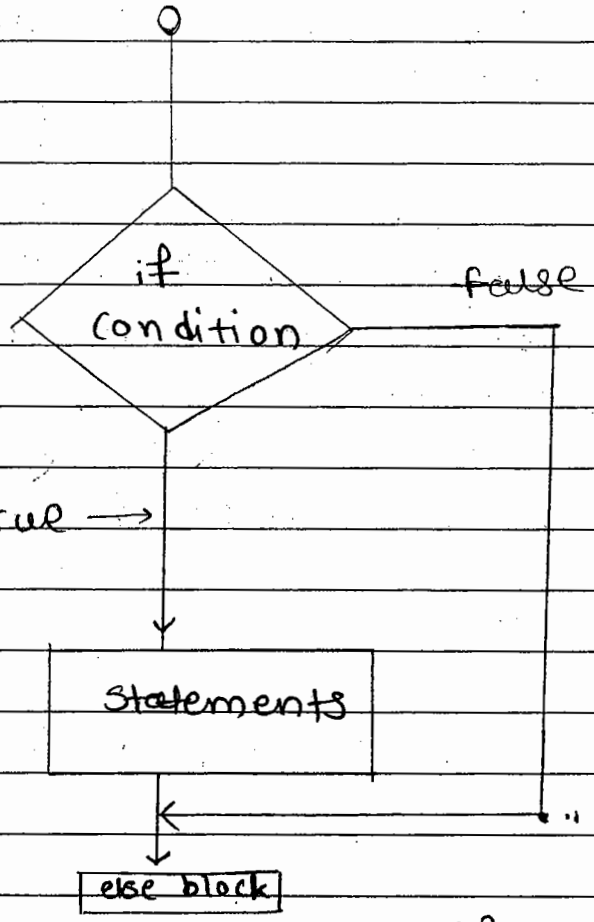
1) Simple if
2) if else statement.
3) if else if ladder.
4) switch statement.

1) Simple if. —

If statement is the control flow statement and helps the programmer in decision makeing.

Syntax — if (condition)
{
(statements);
}

- Flowchart of if statement —



- The statements inside the if block is executed if the condition is evaluated to true.
- If the condition is false the execution of statements inside the if block are skipped.

Note. — The condition for an if statement must be ~~of~~ Boolean type.

```
* class IfExample
  {
      p s v m (s[]a)
      {
          s.o.pln("Execution starts");

          if (true) {
                  s.o.pln.("hi from if block");
                  }
          s.o.pln. ("Execution Ends");
      }
  }
```

O/P. ⟶ Execution starts.
hi from if block.
Execution false.

```
* class IfExample
  {
      p s v m(s[]a)
      {
          s.o.pln ("Execution starts");
          if (false) {
                  s.o.pln. ("hi from if block");
                  }
          s.o.pln. ("Execution Ends");
      }
  }
```

O/P → Execution Starts
Execution Ends.

* class IfExample
{
p s v m (s[]a)
{
s.o.pln. ("Execution starts");

if (false)
s.o. pln. ("hi from if block");
s.o.pln. ("Something");
s.o.pln. ("Something else");
s.o.pln. ("Execution ends here");
}
}

O/P → Execution starts
Something
something else
Execution ends here.

* <u>Note</u> :— Blocks (curly Braces) are optional in if statement.

- If we are not using the Blocks only one statement after the condition will be considered as a part of if.
- And we cannot declare a variable but we can do the Initialization of a variable.

Page No.
Date

```
* class IfExample
{
        p s v m (S[] a)
        {
            if (true)
                int a = 10;
                s.o.pln.("hello world");
            }
}
```

o/p. ⟶ Compile Time Error : variable declaret is not allowed here.

```
* class IfExample
{
        p s v m(S[] a)
        {
            int a;
            if (true)
            {
                a = 10;
                s.o.pln("hello");
            }
            s.o.pln(a);
        }
}
```

o/p ⟶ hello
10.

```
* class IfExample
{
```

```
psvm(s[]a)
{
    int a;
    if (false)
    {
        a=10;
        s.o.pln("helloo");
    }
    s.o.pln(a);
}
}
```

o/p → CTE : variable

* WAIP to check if the number is a multiple of 5, the ~~if else~~ program displays "HiFive". If the number is divisible by 2, it displays "HiEven".

→
```
class IfElse5
{
    psvm(s[]a)
    {
        if (5%.5 == 0)
        {
            s.o.pln("HiFive");
        }
        s.o.pln("HiEven");
    }
}
```

o/p. → HiFive
HiEven

Page No.
Date

* WATP if statement that assigns 1 to x if y is grea than 0.

→ class IfXY

```
{
    P s v m(S[] a)
    {
        int x;
        int y=2;
        if (y>0) {
                x=1;
                s.o.pln. ("Successfull");
            }
        }
    }
```

O/p. → successfull.

* WATP if statement that increases pay by 3% if score is greater than 90.

→ class IfElse

```
{
    p s v m(S[]a)
    {
        int a = 190;
        if (a>=90) {
                int op= a+3%.a;
                s.o.pln. (op);
            }
            s.o.pln. ("No Increment");
    }
}
```

O/p. → 193
No Increment

Page No.
Date

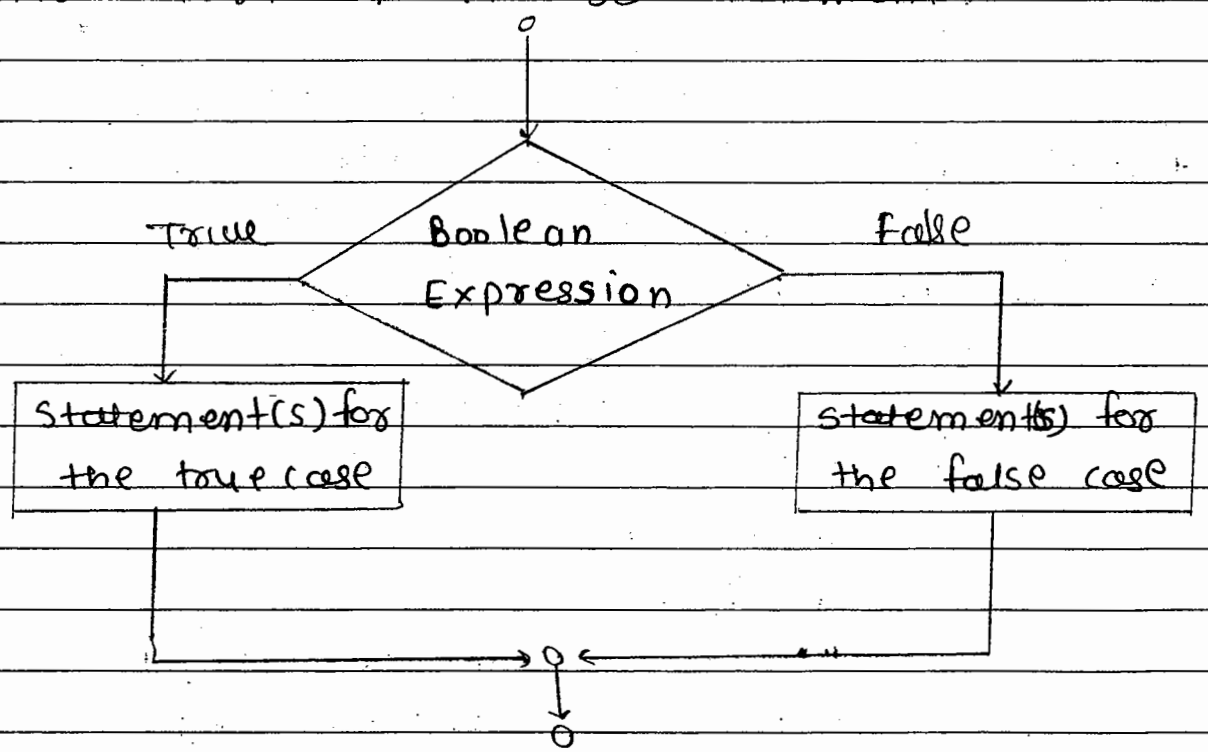
2) If else Statements. —

If statement is the control flow statement and helps the programmer in decision making.

Syntax : —

```
if (condition)
{
    (statements);    // for the true case
}
else
{
    (statements);    // for the false case
}
```

* Flowchart of if else Statement



* Characteristics. —

The statements inside the if block gets executed if the condition is returns true and the execution of else block is skipped ~~to true~~.

- Else ~~bolo~~ block gets executed only when the condition is false.

Note — The condition must be of boolean type
- we cannot execute both the block at a time (i.e. only one block can be executed at a time.)

* WAJP to check if the user entered % is eligible for appearing the interview or not?

→
```
class IfElseInterview
{
    p s v m(S[]a)
    {
        if (60<=60) {
                    s.o.pln. ("Eligible");
                    }
                    else
                    {
                     s.o.pln. ("Not Eligible");
                    }
        }
}
```

O/p. → Eligible.

* WAJP to check if the user entered no. is Even or ODD

→
```
class IfElseEvodd
{
    p s v m(s[]a)
    {
        int a = 8;
```

```
        if (a%2 == 0)
        {
            s.o.pln.("The number is Even");
        }
        else
        {
            S.o.pln.("The Number is Odd");
        }
    }
}
```

o/p. → Even.

* WAJP to check if the user given no. is ~~+ve~~ ~~Even or Odd?~~ +ve -ve no.?

→ class IfElseEvenOdd

```
{
    p s v m(S[]a)
    {
        int a = 2;
        if (a%2 >= 0)
        {
            s.o.pln.("The no. is ~~Even~~ +ve");
        }
        else
        {
            s.o.pln.("The no. is ~~Odd~~ -ve");
        }
    }
}
```

o/p → The no. is ~~even~~ +ve

Page No.
Date

* WAJP to check if the user given no. is +ve no.

→ class IfElseposiNegi

```
{
    p s v m(s[]a)
    {
        int a = -10;
        if (a>0){
         s.o.pln.("the Number is +ve");
        }
        else
        {
         s.o.pln.("the Number is -ve");
        }
    }
}
```

o/p. → the Number is -ve.

* class IfExample

```
{
    p s v m(s[]a)
    {
        if (true)
        {
            s.o.pln.("hii");
        }
        else {
            s.o.pln.("byee");
        }
    }
}
```

Page No.
Date

O/p. → hii

* class IfExample
{
    p s v m(S[] a)
    {
        if (false){
            s.o.pln. ("hii");
        } else {
            s.o.pln. ("byee");
        }
    }
}
O/p. → byee.

* class IfExample
{
    p s v m(S[] a)
    {
        if (false) {
            s.o.pln. ("hiii");
        }
        s.o.pln. (" hi in between");
        else {
            s.o.pln. ("byee");
        }
} }
O/p. → hi in between.
byee.

Page No.
Date

```
* class IfExample
{
    p s v m (s[] a)
    {
        if (false){
                s.o.pln. ("hii");
                }
                else{
                    s.o.pln. ("byee");
                    }
                }
        }
```

o/p. → byee.

```
* class IfExample
{
        p s v m (s[] a)
        {
            if (false)
            s. o.pln. ("hii");
            s.o.pln. ("byeeee");
            else
            s.o.pln. ("byee");
            }
}
```

o/p. →

Page No.
Date

_____ .equals (_____)

* WAIP to check if the boy is eligible for marrying his girlfriend?. (21+, 1000000)

→ class IfElseMarge
{
    p s v m(s[]a)
    {
        int a = 22;
        int b = 10000000;
        if (a>= 21 && b == 10000000) {
            s.o.pln. ("Eligible"); }
        } else {
            s.o.pln. (" Not Eligible ");
            }
    }
}

O/p → Eligible.

* WAIP that increases the pay of employee by 3% if his monthly target score is greater than 90, Otherwis increases pay by 1%.

→ class Demo {
{
    p s v m (s[]a)
    {
        int a = 100;
        if (a > = 90) {
            int op = a+3%.a;
            s.o.pln. (op);
        } else
        { int op = a +1%.a;
            s.o.pln. (op);
    }
}

O/p → 103.

Page No.
Date

* WAJP to check if the user entered number is Even or odd ?

→ class IfElseEvenodd
{
p s v m (s [] a)
{
int a = 2;
if (a % 2 == 0) {
s.o.pln.("The No is even");
}
else {
s.o.pln.("The No. is odd");
}
}
}

o/p → The No is even.

* WAJBP to check if the user given number is +ve or -ve

→ class IfElsePoNe
{
p s v m(s[] a)
~~if~~ { int a = -9;
if (a >=0) {
s.o.pln("The Number is +ve");
}
else {
s.o.pln.("The Number is -ve
}
}
}

Page No.
Date

by

O/P. → The Number is -ve.

- byte code. —
- bytecode in Java is a set of Instructions for the Java virtual Machine (JVM) ~~that~~ is responsible for interpreting them.
- ~~upon~~ after compiling a Java program Java bytecode is generated that can be executed on any platform using the JVM. It is present in .class file.

- System. — built-in class ~~in~~ present in Java.lang package.

Out. — is a variable used as a Reference for print stream class

Println. — method ~~of~~ print Stream Class. these class is present in Java.io package.

- Java is a Imperative language Because it specifying each & every thing.

- The classes which are used for language purpose present in lang package. programmer does not need to import ~~to~~ them.

Page No.
Date

- can we store a -ve value in char?
→ No, The range of char datatype is 0 to 65535. If we try to store a -ve value it will give a CTE (possible lossy conversion from int to byte but, we can do that by Type casting (conversion of one datatype into another).
- can we run java file without compilation
→ Yes, But JVM has to present in that system.

- to get last digit of a number % 10.
- to remove last digit of a number / 10.

- only 1 ternary operator is there in Java — Conditional operator. op1 ? op2 : op3 ;

- { } braces are not mandatory in control flow statements except switch.

```
= if(true) {
        int a;          → Declaration
        int a=10;       → Initialization
    }

= int a;
= if(    )
        int a=10;
```

- Why we can't store byte inside byte container?
→ cause, size of byte is 8 bit and processor run on 32 bits.

Page No.
Date

- How we can store int + int inside int container cause size is 32 bit.

- It is mandatory to give filename as class name.
→ No, it was not mandatory it is just for our understanding, if we give a different filename & classname, we can compile a file using javac filename.java command it will generate a .class file which is create by class name if you try to run the same file you error file not found and if you run file by using class name it will be run and this is very hectique task for programmer & this looks like bad programming so we have give a filename save as class name.

- Why there is 52+ keyword
→ cause as JDK version updates new keywords all added but there are only 52+ keywords which all use for n follow.

Page No.
Date

1) Ugly numbers:—
Ugly number are +ve no. whose only prime factors are 2, 3 or 5. first 10 ugly nos – 1, 2, 3, 5, 6, 8, 9, 10
(1 is added by convention.)

2) Happy numbers. —
Starting with any positive integer, replace the no by the sum of the squares of its digits, & ~~repeat~~ repeat the process until the no. equals to 1.

3) cube no. —
The cube of no. n is third power.

4) cyclic no. —
It is an integer in which cyclic permutations of the digits are successive multiple of the number.

5) ~~Pl~~ pallindrome no. —
no. is a number that is the same when written forwards or backwards. for same after reverse. e.g. – 545, 15

6) Hamming no. —
no. with having prime ~~factontuyrs~~ only 2, 3 & 5.

7) Armstrong no — 1 plus perfect. — 1 plus perfect.
No. "m" that is equal to the sum of ~~cubes~~ m of its digits.

8) Lucky no. —
By applying the process ~~in on a sequencial~~ there still remains numbers in the seq. are called Lucky no.

Page No.
Date

9) Fibonacci Series —
next no. is the sum of previous two nos.

10) prime number —
The number which is divided by 1 or itself only.
e.g. — 2, 3, 5, 7, 11, 17, etc.

11) factorial number —
The product of all +ve descending integers.
e.g. — $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$.
$4! = 4 \times 3 \times 2 \times 1 = 24$.

12) Tech Number —
- If the number has an even no. of digits and the no. can be divided exactly into two parts from the middle.
- After equally dividing no, sum of the nos & find the square of the sum.
- If we get the no itself as a square then it is a tech no. otherwise not.

2025 — 20 | 25 — 20 + 25 — $(45)^2$ → 2025

1312 — 13 | 12 — 13 + 12 — $(25)^2$ → 625

2025 = 2025 tech no.

1312 ≠ 1312 not a tech no.

Page No.
Date

13) Sunny number. —
If the no. next to the given no is a perfect square
e.g. = N = 80 80 + 1 = 81 perfect sq. of 9 = ✓
N = 10 10 + 1 = 11 not a perf sq. of 10 =

(N + 1) = perfect sq. of any NO.

14) Automorphic Number. —
if & only if the square of the given no. end with the same no. itself.
e.g. 25 $(25)^2$ = 625
76 $(76)^2$ = 5776.

15) Krishnamurthy Number. —
- If the factorial sum of all the digits is equal that number.
- also known as strong number.

145
1! + 4! + 5!
1 + 24 + 120
145

120
24
5
144

16) Peterson number. —
If the sum of factorials of each digit is equal to the sum of number itself.

17) Fascinating Number. —
- multiply the no. by 2 or 3 separately
- write the obtaining result with the no.
- after concatination it contains all the 1 to 9 digit exactly once.
- e.g. = 327 × 2 = 654 } concate = 327654981
327 × 3 = 981

Page No.
Date

18) Keith Number —

e.g. — 742

separate the No.: 7 4 2

add them 742

7 + 4 + 2 = 13

4 + 2 + 13 = 19

2 + 13 + 19 = 34

13 + 19 + 34 = 66

19 + 34 + 66 = 119

34 + 66 + 119 = 219

66 + 119 + 219 = 404

119 + 219 + 404 = 742

78 + 87 = 165

11 (x + y)

11 (8 + 7)

88 + 77

165

87 + 78 = 165

11 (x + y)

11 (8 + 7)

88 + 77

= 165 — (1)

11 (x + y)

11 (7 + 8)

77 + 88

= 165 — (2)

88 + 77 19) Mystry Number —

165 — if the no. can be expressed by the sum of two no.

— these two nos must be reverse of each other.

87 + 78 — It lies between 22 to 198

78 — Mystry No. are multiple of 11.

165

20) Neon number —

A +ve integer whose sum of digits of its square is equal to the number itself is called as neon number.

e.g. →

9

↓

$9^2 = 81$

↓

8 + 1 = 9

9 = 9

It is a neon number.

Page No.
Date

21) Natural Numbers. —
all +ve integers from 1 + ∞.

22) Spy Numbers. —
A +ve integer is called as a spy number if the sum & the product of its digits are equal.
e.g. — 1124.
$1 + 1 + 2 + 4 = 8$
$1 \times 1 \times 2 \times 4 = 8$.

23) Buzz no. —
No that ends with 7 or divisible by 7.

24) Duck No. — Number that contain 0 but not at begginning or mid or end.

25) Evil No. —
A particular no's binary code should contain even 1 in it.
Ex — 15 → 1111 It is a Evil no.
16 → 10000 It is not a Evil no.

26) Emirp No. —
Suppose we tak a ~~no~~ prime no. & after reversing that we get another prime no.
e.g. → 13 → is a prime no
reverse it
31 → It is a prime no.

Page No.
Date

27) Bouncy no. —

$1 \xrightarrow{+1} 2 \xrightarrow{+1} 3 \xrightarrow{+1} 4$ it is a bouncy no

$1 \xrightarrow{+1} 2 \xrightarrow{+1} 3 \xrightarrow{+4} 7$ not a bouncy no.

28) smith / Joke number. —
any no sum & its prime factor sum is equal then it is a smith / Joke no.

e.g. → 8 5

$8 + 5 = \underline{\underline{13}}$

85's prime factors are 5 & 17

$\therefore 5 + 1 + 7 = \underline{\underline{13}}$

85 is a smith no.

29) strontio no. —
four digit no when get multiply by 2 then it give the same digit at the hundread place & at the Tense place. $1221 \times 2 = \underline{\underline{14}}\,\underline{\underline{41}}$

30) Xylem & pholem no. —
If the 1st digit's & last digit's addition matches with the middle digit addition it is xylem no. otherwise it is a pholem no.

e.g. → 12 / 48

$1 + 2 / 3 \quad 4 + 8$

0

e.g. → 1 2 3 4 8

$1 + 8 = 9$ = $2 + 3 + 4 = 9$

$5 \neq 9$ — problem

Page No. 9 = 9

Date

e.g. → 7 1 2 9 2

$7+2 = 9 \neq 1+2+9 = 12$

31) ISBN Number — International standard Book number.

1 2 5 9 0 6 0 9 7 7

$= (1\times10)+(2\times9)+(5\times8)+(9\times7)+(0\times6)+(6\times5)+(0\times4)+(9\times3)+(7\times2)+(7\times1)$

$= 10+18+40+63+0+30+27+14+7$

$= 88+93+27+21$

$= 209 \% 11$

Remainder = 0 — It is a ISBN no.

32) Autobiographical No. —

$\underline{2\ 1\ 3\ 4\ 5} = 5$ $\quad 5 \neq 15$

$2+1+3+4+5 = 15$

$\underline{2+1+2+0+0} = 5$

length should be less than 10.

Page No.
Date

4 4 < 5 4 % 1 == 0

a → 4 → 1, 2, 4 4 % 1 == 0

b → 5 → 1, 5 5 % 1 == 0

4 5 a % i == 0

int gcd = 0; 4 <= 4 b % i == 0

```
for(int j = 4; j <= 4; j++) {
    if(a % j == 0 && b % j == 0) {
        gcd = j;
    }
}
```

4 % 4 == 0 5 % 4 == 0

a = 10
a == 10

S.O.P(gcd) // 1

* final is a keyword → makes the members constant → cannot change them values → used for variable, methods, blocks.

* we can make class as abstract, even if the class does not have any method init. e.g. → abstract class Demo { ... }

* If we want to stop the Inheritance of the class then make that class final.

* If the class is not abstract then it is a concrete.

* Explicit call is done in constructor chaining.

→ In Interface to create a default method or default block.

* In method overloading only method name must be same & formal arguments must be diff. other remaining things dosen't matter.

Page No.
Date

* can we print this keyword?

→ No, we can't print the this keyword. but we can print the reference of the obj.

- It prints = A@Hexadecimalvalues

A → class name
@ → common in all

- alpha numeric value → combination of char(A-Z, a-z) & Numbl

* can we create ~~class another~~ 2 classes with same name?

→ No, compiler get's confused which class he hase to be compile. || Package

* can we create ~~mu~~ object inside object?

→ No, we can not create but we can use the obj referen

* Source code → Human Understandable language (program that we write in sublime.)

* mid level lang = byte code.

Source code → .classfile → JVM

. Java class file : byte code
↓
present in
mid level lang.
(consist of 0, 1, char, special char).

* Java is 100% Independent (platform). & JVM is platform dependent. JVM is diff for MAC & Windows. but the byte code which Java created is run on any platform which has JVM on it.

* Public ++
protected
default
Private --

p s v [illegible] arguments

method signature

Java execution starts from main method & ends on main method

in program, 1st JVM finds for main method

* P.S.V.M. (S[]a)

If we remove static & make method as a non-static then can we run these?

→ No, Because we can't create obj without static method cause, we can't create the obj without static method.

* which one is faster recursion or loop?

→ - recursion is faster cause loop has many further process to follow/do like loop has to ~~follow~~ run the blocks, conditions, etc.)

& recursion does not have any ~~proc~~ task further to do.

* When we create an object of a class A inside the class B then we can access all the properties of the class A inside class B. (cond^n = the class A must not be the private)

* Why we need Upcasting?

to access only parent class block.

* Why we need Downcasting?

to avoid extra memory allocation.

* Static → non static ✗

* non static → static ✓

Page No.
Date

1) for (; true;)
{
s.o.pln ("Hi;")
}
s.o.pln ("bye");

CTE

unreachable stat

2) for (; false;)
{
s.o.pln ("Hi;") ← unreachable state
}

CTE

3) disadvantage of loop → if loop goes in a infinite loop then remaining statements are unreachable

4) default value of cond^n of for loop is "true"
— " — while loop is "nothing"
not any value

If we not written a cond^n in while loop = error

— entry control loop — while loop
— exit — " — — double

Page No.
Date

[illegible]

If prog [illegible] able to give [illegible] no
then [illegible] give [illegible] zero
at computation [illegible] for For loop)

any single ~~single~~ digit no is divided by
10 it returns 0 in java

$$\begin{array}{r} 0.9 \\ 10 \overline{)\,9\,5} \\ -9\,0 \\ \hline 5 \end{array}$$

0.9 is 0 in java (take as int)

- WAJP count
- WAJP reverse a num
- —"— to check whether the user no. entered no. is pallindrome or not
- WAJP to find factorial (Scanner class)
- WAJP to find factors (—"—)
- —"— no is prime or not (—"—)
- Fibonnoci series
- first 10 no of series 18 24 84 294

18 - 1 = 17

1) reverse no (while loop)
2) Print 2 to A
3) scanner enter no = 2

Java script
Java → method Inside method (is not possible) Search

8 16 24

24 40 64 104 176

24 × 2 = 48 - 8
40 × 2 = 80 - 16
64 × 2 = 128 - 8

Because range is ...

Page No.
Date

long

cannot pass float in switch } not ~~co~~ recommend
double } 1.0000
Boolean } 2.0000000

fall through =

→ Because boolean has only 2 ~~execution~~ possible value
~~so~~, then use if else

string is a class & non primitive
- we can pass ~~long~~ string & class in switch

- final is a keyword. ~~to make variable~~ & non access modifier
- we cannot assign final variable
- case block & default block is not mandatory.
- if we not create case & block then it will

when programer knows the no of iteration to use for loop otherwise use while loop

— Unary operator - ~~Unary~~, Increment decrement, new & type cast operator
new is a keyword & operator
~~obj. are also~~

case 1 label ←

switch block ←

to remove last number = Defa

- Type of reln operator (Tuesday) ..
- one prog. related to = = operator.
- 13 prog. Tuesday + 12 prog (last Sat) = 25 prog on (Thursday)
- ~~2 pr~~ WAP to find sum of digit for 3 digit no. Take the input from the user
- WAP to reverse a 3 digit no. entered by user.

- Why can't we store byte + byte inside a byte container
- How we can store int + int inside an int container

- one prog related to relational operator
- 5 prog on increment & decrement (pluses)

a = 2   2   2

b = a++   2   a = 3

c = a   → 4   4

3

4

2 + 3 + 3 3 / 2 + 3 + ~~3+~~ + 4

11   13

a = 2

b = 3

3 + 3 + 4 + (3+4)

a = 2   17

b = 3

2(1) + 4 + ~~a~~ 3(1) + ~~a~~ 5 + 4 = 5

a   a   b   b   a   b

23

= ~~[illegible]~~ using

we have pass switch

Page No.
Date

- In Java we can not create Duplicate case labels = can not write boolean in switch

type casting

Implicit		Explicit	
widening (primitive)	upcasting (non-primitive)	Narrowing (primitive)	Down Casting (Non-primiti

Type casting — conversion from one datatype into another.

-||-||-

1) Duck No.
2) perfect No
3) krishnamurthi No

a = 2
b = 3
c = 4

2(3) + 5 + 4 + 4(5) + 3 + 5 + 5
a c b b

2(3) + 5 + 4 +

a = 1
b = ++a (2)
c = b-- 2

2 + 2(1) + 1 + 2 + 2(1) +
a a b b c

a + b + c

2 + 2(3) 3 + 4 2 + 2 + 1 + 2 + 2 + 2 + 1
a + b

~~1(2)~~
~~2(3)~~

2 + ~~2(3)~~ +

2 + ~~2~~ 1(0) + 1 + 1(2) + ~~2(3)~~ + 0 + 2 + 1
2(1).

a a b b c

10

Page No.
Date

1) – declarative and Imperative programming.
2) – Research on ASCII (why ~~use~~ we use ASCII?)
3) – Unicoad. total Unicoad in Java = 65535.
3) → Unicoad is a international encoding standard for use with different languages and scripts, by which each letter, digit or symbol is assigned a unique numeric value.

1) → Declarative. – programming is a type of programming in that declares what program todo be executed. e.g. – React.

elaborate every term
Imperative. – It is a type of a programming that describes how the todo program executed.
e.g – C, C sharp, Java

2) → (American Standard Code for Information Interchange)
is the most common character encoding format for text data in computers & on the internet.

– Java is a ~~bas~~ class base prog. lang.

– Range of char data type is 65535.

– What is coatlean. & why we use Java ~~w~~ not coatlean

– diff betn float & double. –
– Write a java program to check the precision value of double (f - 5/3 of) d - 5/3.0

Page No.
Date

Write a 2 program for float 2 for double.
- two programs for combine data types. classmate

* object are store inside hip memory

1) WA Java prog. to take input from a user to print th given information. (Name, Native, YoPassOut, Branch, University), (CGPA). Do you have any backog.).

1) WAJP to take 2 no. input from the User to find add. of it
2) solve the eqn to display the below (photo).
3) WAJP to find perimeter & area of a circle using user input.
4) WAJP to find area of a Rectangle using Scanner class.
5) WAJP to find area of Δ using scanner class
6) WAJP to solve the following equation using scanner cla
$(a+b)^2 = a^2 + 2ab + b^2$
7) WAJP to find last Digit of any User Entered No.
8) WAJP to find 1st Digit of a 3 digit no.
9) Photo ~~ner~~
10) photo popn.
11) —"— ceiling F
12) cylinder

* Why we can't store byte inside byte container.
① Because byte Since its 8 bit & process works on
② 32 bits! — min
③ —"— either a no. is even or odd.
④ we can't store int inside the int size is 32 bits.

Page No.
Date

- [illegible] specified [illegible] only [illegible] is [illegible] [illegible], [illegible]

- int a = 1_50; — is correct
  output = 150

- declarative / [illegible] Shortcut way to tell any work
  (Imperative) longcut — " —

(1) WAJP to check a char is upper case or low case using input

(2) — " — vowel or cons

Entered by user

(3) WAJP to check a No. is +ve or -ve (scanner)

(4) — " — max no. among 3 (scanner)

(5) — " — min — " — (scanner)

30
30

(6) WAJP to check the 1st char of your name is a vowel or a consonant (scanner)

(7) WAJP to check the No. is +ve, -ve or 0

153 1) 1 to 100 4) 100 to 1

2) A to Z 5) Z to A

125 × 27 3) a to z 6) Alternative characters (A D G ....

1 × 125 × 28 = 153 7) ASCII characters with their corresponding Asci values.

8) even no's bet^n ~~using~~ user entered range.

snum for (int i = snum; i <= enum
enum

28

(9) odd no's — " —

(10) WAJP to find count of digit.

(11) WAJP to reverse a number.

12) WAJP no. is even or odd without using modulus operator.

Control Flow statements. —

The statements which are use to control the execution flow of the java program ~~statements~~, are known as control flow statements.

Types of control flow statements. —

1) Conditional Statements | Decision making Statement
2) Looping statement.
3) Branching statement | Control transfer Statement
   Break, return, continue.

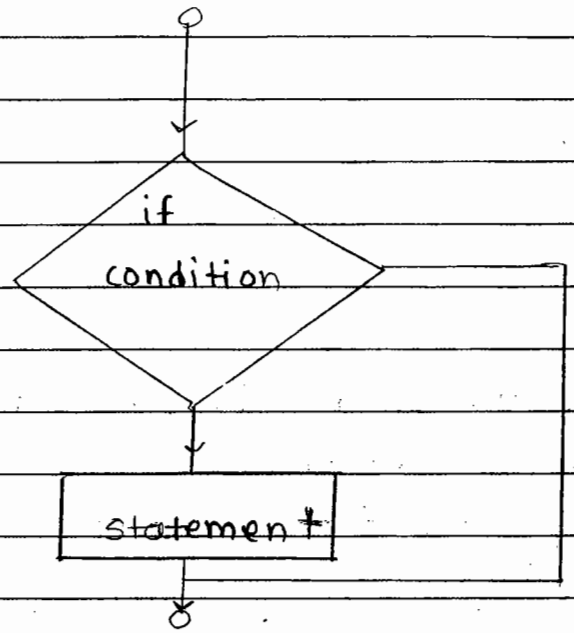
Decision making Statements | conditional Statements.
The statements which help us (programmer) to decie wether to execute a set of Instruction or not.

Types of Conditional Statements. —
1) Simple ~~of~~ if
2) if else statement.
3) if else if ladder.
4) Switch statement.

1) Simple if. —
If statement is the control flow statement and helps the programmer in decision making.

syntax. —
```
if (condition)
{
    (statement)
}
```

- Flow chart of if statement -



- The statements inside the if block is executed if the condition is evaluated to true.
- If the cond^n is false the execution of statements inside the if block are skipped.

- Note — The condition for an if statement must be of Boolean type.

```
* class Demo
  {
      p s v m (S [] a)
      {
          S.o.pln ("Execution starts");
          {
          if(true) {
                  S.o.pln ("hi from if block");
              }
```

```
        s.o.pln. (" Execution Ends");
        }
}
```

o/p → Execution starts
hi from if block.
Execution ~~false~~ Ends.

```
* class Demo
{
        p s v m (s[]a)
        {
            s.o.pln.(" Execution starts");
            if (~~true~~ false.) {
                    s.o.pln.(" hi from if block");
                }
            s.o.~~pln~~ln. ("Execution Ends");
            }
    }
```

o/p → Execution starts
Execution Ends.

```
* class Demo
{
        p s v m (s[]a)
        {
            s.o.pln. (" Execution starts");
            if (~~false~~ True.) s.o.pln.(" hi from if block");
                s.o.pln. (" something");
                s.o.pln. (" something else");
                s.o.pln. (" Execution ends here");
        }  }
```

o/p → Execution starts
Something
something else
~~Execution ends here~~

Note. —

- { } (Blocks) are optional in if statement.
- If we are not using blocks only one statement after the condition will be considered as a part of if.
- And we cannot declare a variable but we can do the Initialization of a variable

- class Demo

```
{
    p s v m(s[]a)
    {
      if (true)
       int a = 10;
       s.o.pln. ("hello world");
    }
}
```

O/P. → CTE (variable declaration is not allowed here).

- class Demo

```
{
    p s v m(s[]a)
    { int a;
      if(true)
      {
          a = 10;
          s.o.pln. ("hello");
      }
      s.o.pln. (a);
```

```
        }
    }
```

O/P → hello
10.

- class Demo

```
{
    p s v m (s[]a)
    {
        int a;
        if (false)
            {
                a = 10;
                s.o.pln. ("hello");
            }
            s.o. pln. (a);
    }
}
```

O/P → CTE (variable a might have been initialized)

* WAJP to check if the no is a multiple of 5, the program displays "HiFive". If the number is divisible by 2, it displays "HiEven".

→

```
import java.util.Scanner;
class Demo
{
    p s v m (s[]a)
    {
        Scanner sc = new Scanner (System.in);
        s.o.p. ("Enter a no :");
```

```
        int num = sc.nextInt();
        if (num % 5 == 0)
        {
            S.o.pln (" HiFive");
        }
        ~~et~~ if (num % 2 == 0)
        {
            s.o.pln (" HiEven");
        }
    }
}
```

o/p → Enter a no : 12
HiEven

Enter a no : 25
HiFive.

* WAJP on if statement that Assigns 1 to x if y is >0.

→

```
class Demo
{
    p s v m (S[]a)
    {
        int x;
        int y=2;
        if (y>0) {
                x= 1;
                s.o.pln.(" succesfull");
                }
    }
}
```

O/P. → Successful.

* WAJP an if statement that Increases pay by 3%. if score greater than 90.

→ Class Demo

```
{
    p s v m(S[]a)
    {
        int a = 190;
        if (a>=90)
        {
            int op = a+3%.a;
            s.o.pln.(op);
        }
        s.o.pln.("No Increment");
    }
}
```

O/P → 193
No Increment.

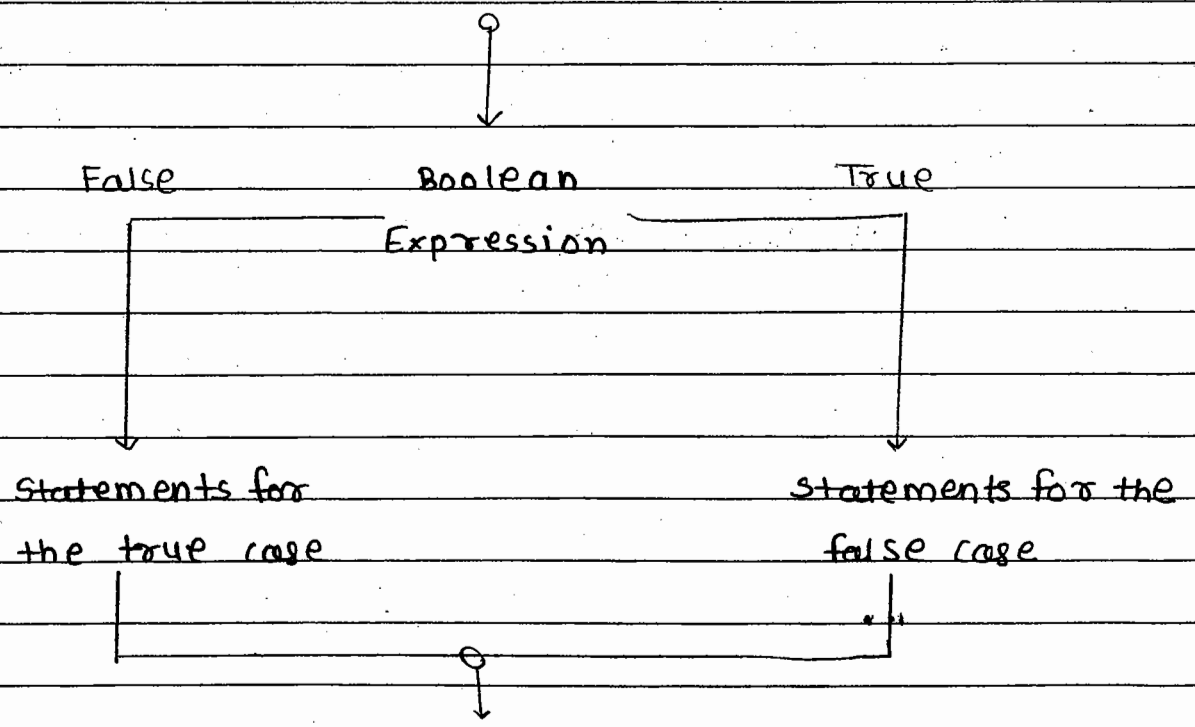
2) If else statement.—

If statement is the control flow statement and helps the programmer in decision making.

Syntax : if (condition)

```
{
    (statements);      // for the true case
}
else
{
    (statements);      // for the false case
}
```

* Flow chart of if else statement.—



* characteristics —

- The statements inside the if block gets executed if the condition is returns true and the execution of else block get skipped.
- Else block gets executed only when the condition is false.

* WAJP to check if the user entered % is eligible for appearing the interview or not?

→ ~~class If~~

```
import java.util.Scanner;
class IfElse1
{
        p s v m(s[]a)
        {
            Scanner sc = new Scanner (system.in);
            int a = sc.nexInt();
            S.o.pln.("Enter a %.");
            if (a >= 60)
            {
                S.o.pln ("Eligible");
            }
            else
            {
                S.o.pln ("not Eligible");
            }
        }
}
```

O/p → Enter a %. : 60

Eligible

* WAJP to check if the user entered no. is Even or odd?

→ class Demo

```
{
        p s v m(s[]a)
        {
            int a = 8;
            if (a % 2 == 0)
            {
                s.o.pln. ("The no. is Even");
            }
            else {
                s.o.pln(" The no. is odd ");
                }
        }
}
```

O/p → Even.

* WAJP to check if the user if the user given no is +ve or -ve.

→ class Demo

```
{
        p s v m (s []a)
        {
            int a = 2;
            if (a >= 0)
            {
                s.o.pln. (" The no is +ve");
            }
```

```
else {
        s.o.pln.("The no. is -ve");
    }
  }
}
```

o/p → The no. is +ve.

/ * WAJP to check if the user given no. is +ve no.

→

```
class Demo
{
    p s v m(s[]a)
    {
        int a =-10;
        if (a>0) {
                s.o.pln.("the no. is +ve");
                }
                else{
                s.o.pln.("the no. is -ve");
                }
    }
}
```

o/p → The No. is -ve.

* class Example

```
{
    p s v m(s[]a)
    {
    if (true) {
            s.o.pln.("hii");
            }
```

```
        else{
                S.o.pln.("byee");
            }
        }
    }
```

o/p. → hii.

```
* class IfExample
  {
      p s v m (s[] a)
      {
          if (false){
                  s.o.pln.("hii");
                  }
                  else{
                      S.o.pln.("byee");
                      }
                  }
              }
```

o/p. → byee.

```
* class Demo
  {
      p s v m(s[] a)
      {
          if (false){
                  S.o.pln.("hii");
                  }
          S.o.pln.("hi in between");
          else{
```

```
        s.o.pln("byee");
        }
    }
}
```

o/p → hi in between
byee.

* class Demo

```
{
    p s v m(s[]a)
    {
        if (false){
                s.o.pln.("hii");
            }
        else {
                s.o.pln.("byee");
            }
    }
}
```

o/p → byee.

* class Demo

```
{
    p s v m(s[]a)
    {
        if (false) {
                s.o.pln.("hii");
                s.o.pln("byee");
            }
        else {
```

```
            s.o.pln.("byee");
        }
    }
}
```

Olp. → byeee
bye

* WAIP to check if the boy is eligible for marrying his gf. ? (21+, 10000000, Ambani).

→

* WAJP that increases the pay of employee by 3% if his monthly target score is greater than 90, others increases pay by 1%.

→ class Demo

```
{
        p s v m(S[]a)
        {
            int a = 100;
            if (a >=90)
            {                    = a + 3% a;
                int op = a + 3% a; * 1.03 * a;
                S.o.pln. (op);
            } else
            {                    = a + 1% a;
                int op = a + 1.01 * a;
                s.o.pln. (op);
            }
        }
}
```

o/p. →

* WAJP to check if the user entered no. is Even or odd

→ class Demo

```
{
        p s v m (S[]a)
        {
            int a = 2;
            if ( a % 2 == 0) {
                                s.o. pln. ("even");
                            }
```

```
        else {
                s.o.pln.("odd");
            }
    }
}
```

o/p. ⟶ even.

* WAJP to check if the user given no. is +ve or -ve

⟶

```
class Demo
{
    p s v m(S[]a)
    {
        int a=-9;
        if (a>=0)
        {
            s.o.pln.("+ve");
        }
        else {
            s.o.pln.("-ve");
            }
        }
    }
}
```

o/p. ⟶ -ve.

17110123

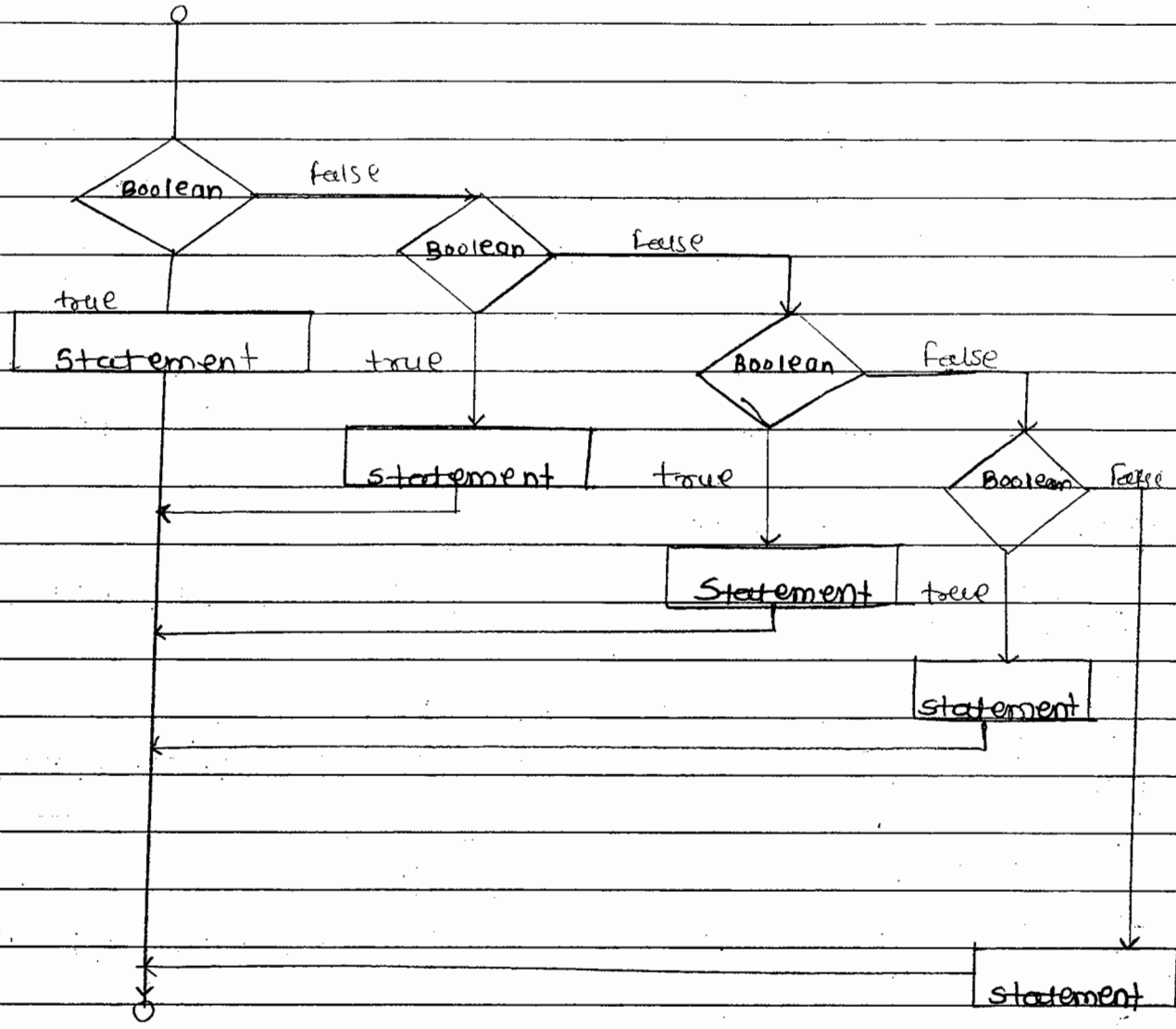
* if else if Ladder. —

If else if ladder is a control flow statement which helps the programmer in decision making. (wether to execute a block or not).

If we have to check more than one condition we make use of a if else if ladder.

syntax. —

```
if (condition1){
        s.o.pln.("Hii");|stateu
    }
else if {
        statement; // cond^n
    }
else if {
        statement; // cond^n
    }
else {
        statement;
    }
```



* Characteristics —

- using an else block is not mandatory
- Else block get executed when all the conditions are false
- In ~~if~~ else if ladder only one block can get executed even though multiple if conditions are true
- The first Block which evaluates to true gets executed.

Note. —

- In if else if ladder also curley Braces are optional
- ~~w~~ If we are not using curley Braces only one Statement is considered as a part of the above statement / block. and by mistakenly if we are writting a more than one State it will break the ladder and will cause a compile time error. (else without if)
- And that one Statement cannot be a declarat statement.

Ex. —

```
class Example
{   int i;
    p s v m(s[]a)
    {
        if (true)
            int i=10;   — D
        else if (false) {
                s.o.pln. ("from else if b
                }
    }
}
```

o/p. → CTE (else without if.)

Variable declaraction is not allowed here.

~~int~~ a = 10; ✗ int i; → D.

a = 10; ✓ int i = 10; → I

```
* class elseif
{
        p s v m(s[] a)
        {
          if (false)
            {
                s.o.pln.("from if block");
            }
          else if( false)
            {
                s.o.pln.("from if block 1");
            }
          else if( true)
            {
                s.o.pln.("from if block 2");
            }
          else if(true)
            {
                s.o.pln.("from if block 3");
            }
          else {
                s.o.pln.("from else block");
}       }   }
```

o/p → from if block 2.

```
* class ifelse
{
        p s v m(s[]a)
        {
          if (false) { s.o.pln.("from if block"); }
```

```
else if (false) {
                s.o.pln.("from if block 1");
            }
else if (false) {
                s.o.pln ("from if block 2");
            }
else if (false) {
                s.o.pln.("from if block 3");
            }
        else {
                s.o.pln ("from else Block");
            }
        }
    }
```

o/p → from else Block

* Note. →

* class ifelse2

```
{
    p s vm (s[] a){
    {
        if (false)
                s.o.pln.("from if block");
        else if (false)
                s.o.pln.("from else if block 1");
        else
                s.o.pln.("from else block");
```

O/P. → from else block.

* p s v m (S[]a)

```
{
    if (false)
        s.o.pln. ("from if block");
        s.o.pln. ("hello");
    else if (false)
        s.o.pln. ("from else if block");
    else
        s.o.pln. ("from else block");
    }
}
```

O/P → CTE (else without if).

1) * WAP to check wether the number is +ve, -ve, 0.

```
import java.util.Scanner;
class IfElseLadder
{
    p s v m (string[] args)
    {
        scanner sc = new Scanner (System.in);
        S.o.p(" Enter Digit :");
        int num = sc.nextInt();
        if (num>0) {
            s.o.p.(" positive No");
        }
        else if (num==0)
        {
```

```
            S.o.p. ("It is a 0");
        }
        else
        {
            s.o.p. ("Negative No.");
        }
    }
}
```

o/p → Enter Digit : 0
It is a 0

Enter Digit : 2
Positive No.

Enter Digit : -8
Negative No.

2) WAIP to check wether the ~~number~~ student is Pass/Fa

```
import java.util.Scanner;
class Demo
{
        p s v m(S[]a)
        {
            Scanner sc = new Scanner (System.in);
            S.o.p. ("Enter your marks");
            double marks = sc.nextDouble ();
            if (marks >=75){
                S.o.pln(" First class with distn.'
            }
            else if (marks>=60)
            {
            s.o.pln (" First class.");
            }.
```

```
else if (marks >=50){
                    s.o.pln.("second class");
                }
else if ( marks >=35) {
                    s.o.pln. (" passed");
                }
else if (marks ==35){
                    s.o.pln. ("Just passed");
                }
            else {
                    s.o.pln.("Fail");
                }
        }
}
```

O/P. ⟶ Enter your marks : 99.99
First class with distinction.

3) WAJP to check that your name starts with vowel or consonants.

```
import java.util.Scanner;
class Demo
{
        p s v m (S[ ]a)
        {
        Scanner sc= new scanner (system. in);
        S.o.pln (" Enter your name ");
        Char ch = sc. next() charAt(0);
        if ( ch ==' a') {
                    s.o.pln("Your nam starts
```

```
                    with vowe");
            }
        else if (ch =='e')
        {
            s.o.pln(" Your name starts with vowe
        }
        else if (ch == 'i')
        {
            s.o.pln.(" your name starts with vowe
        }
        else if (ch == 'o')
        {
            s.o.pln. (" your name starts with vowe
        }
        else if (ch == 'u')
        {
            s.o.pln(" your name starts with vowel
        }
        else {
            s.o.pln. (" Your Name starts with conso.
            }
    }
}
```

o/p. → Enter your name : Mansi
Your name starts with Consonants.

Enter your name : isha
your name starts with vowel.

4) WAP to check the Leap year.

```
import java.util.Scanner;
class Demo
{
        p s v m (s[]a)
        {
            Scanner sc = new Scanner(system.in);
            s.o.p.("Enter year : ");
            int yr = sc.nextInt();
if(yr%.4==00) if (yr%.400==0||(yr%.4==0&&yr%.100!=0))
            {
                s.o.pln("Leap year");
            } else
            {
                s.o.pln.("Not a Leap year");
            }
        }
}
```

O/P → Enter year : 2023 Enter year : 2012
Not a leap year Leap year.

5) WAJP to check Driving licence Eligibility.

```
import java.util.Scanner;
class Demo
{
        p s v m (s[]a)
        {
            Scanner sc = new scanner (system.in);
```

```
s.o.pln.("Enter your age:");
int age = sc.nextInt();
if (age >= 18)
{
  s.o.pln.("R U done with your RTO Test ?.");
  boolean test = sc.nextBoolean();
  if (test == true)
  {
    s.o.p.ln.("Enter your RTO test marks");
    double marks = sc.nextDouble();
    if (marks >= 60)
    {
      s.o.pln.("your Eligible");
    }
    else
    {
      s.o.pln.("Reappear for the RTO Test");
    }
  }
  else
  {
    s.o.pln.("Age is not eligible");
  }
}
}
```

o/p → Enter your age : 22
R U done with your RTO Test : True
Enter your RTO Test marks : 70
You are eligible.

6) WAIP to check Blood Donation Eligibility.

```
import java.util.Scanner;
class Demo
{
        p s v m (s[]a)
        {
            Scanner sc = new Scanner(system.in);
            s.o.p ("Enter your age :");
            int age = sc.nextInt();
            s.o.pln. ("have you done smoking or
            drinking since last 24 hourse ?");
            boolean sd = sc.nextBoolean();
            s.o.pln. ("enter your weight");
            double we = sc.nextDouble();
            if (age>20&&sd==false||we>=50)
            {
                s.o.pln. ("You are eligible");
            }
            else if (age<=20&&(sd==true&&we<50))
            {
                s.o.pln. ("You can't ~~do~~ or not
                        eligible for blood Donation.");
            }
}       }
```

O/p. → Enter your age : 22
have you done smoking or drinking since last 24 hourse : False
Enter your weight : 65

You are eligible for Blood Donation.

7) WAJP to check the eligibility of Army Cadic

```
import java.util.Scanner;
class Army
{
    p s v m(S[] a)
    {
      Scanner sc = new Scanner(System.in);
      S.o.pln.("enter your age :");
      short age = sc.nextShort();
      S.o.pln.("enter your height :");
      double height = sc.nextDouble();
      S.o.pln.("enter your weight :");
      double weight = sc.nextDouble();
      if (age >=18 && (height >=162 && weight >5
      {
        S.o.pln.("you can join Army :");
      }
      else if (age <18 && (height >=162 && weight >5
      {
        S.o.pln.("your age is not ~~fit~~ eligible");
      }
      else if (age <=18 && (height <162 && weight >
      {
      S.o.pln.("your height is not eligible");
      }
      else if (age >=18 && (height >=162 && weight <=55))
      {
```

18/10/23.

```
        S.O.ln. ("you are not eligible");
        }
    }
}
```

O/P → Enter your age : 12
not Eligible.

Switch Statement —

switch is a Decision Statement
It is a control flow statement and helps the programmer in Decision making.

characteristics. —

1) We can pass byte, short, int, char and String type data only inside switch and case block.

2) We cannot pass long, float, double and Boolean type case block.

3) Inside switch block we can have any number of case blocks but only one default block

4) Using a default block is not mandatory.

5) we cannot pass a variable inside case, if it is not a final variable.

6) we can pass expressions inside switch and case.

7) we can pass a variable inside switch.

Syntax. —

```
switch (literal, exp, var)
{
    case (literal, exp):
    {
        statement 1; break;
    }
    case (literal, exp);
    {
        statement 2; break;
    }
    case (literal, exp);
    {
        statement 3; break;
    }
}
```

* Control Flow in switch statement. —

1) The values given to switch is composed with the values given with case using equality operator

2) If a case is evaluated to False, the control goes to the next case block. if all the cases are evaluated to FALSE then default blocks get executed

3) If a case is evaluated to TRUE;

a) The statement inside the case blocks get executed.

b) if break statement is not used the case block, the statement of the below case block also get executed.

* Break —

Break is a keyword

1) Break is a control transfer statement.

2) It is used only inside switch and loop statements.

* Using Break Inside switch —

1) Break will transfer the control outside the switch block.

2) We can use a break statement in all the case blocks and it is Optional.

1) *
```
class Demo
{
    p s v m (s[] a)
    {
        byte a = 1;
        Switch (a)
        {
            case 1 : s.o.pln ("case 1"); break;
            case 2 : s.o.pln ("case 2"); break;
            case 3 : s.o.pln ("case 3"); break;
            default : s.o.pln ("from default");
```

```
        }
    }
}
```

o/p. → case 1.

2) * class Demo

```
{
        p s v m (s[]a)
        {
         byte a = 1;
         switch (a)
         {
            case 1 : s.o.pln. ("case 1"); break;
            case 2 : s.o.pln. ("case 2"); break;
            case 3 : s.o.pln. ("case 3"); break;
            case 128 : s.o.pln. ("case 3"); break;
            default : s.o.pln. ("from default");
         }
        }
    }
```

o/p. → CTC (possible lossy conv. from int to byte).

3) * class Demo

```
{
        p s v m (s[]a)
        {
         byte a = 1;
         switch(a +0)
         {
```

```
        case 1 : s.o.pln. ("case 1"); break;
        case 2 : s.o.pln. ("case 2"); break;
        case 3 : s.o.pln. ("case 3"); break;
        case 128 : s.o.pln. ("case 3"); break;
        default : s.o.pln. ("from default");
    }
}
```

o/p. → case 1

4)
```
class Demo
{
    p s v m (s[]a)
    {
        boolean a = true;
        switch (a)
        {
            case true : s.o.pln. ("case 1"); break;
            case false : s.o.pln. ("case 2"); break;
            default : s.o.pln. ("from default");
        }
    }
}
```

o/p. → CTE (cannot pass Boolean)

5)
```
class Demo
{
    p s v m (s[]a)
    {
        String a = "Ramesh";
```

```
switch(a)
 {
     case "Ramesh": s.o.pln ("case 1"); break;
     case "suresh": s.o.pln.("case 2"); break;
     case "Mangesh": s.o.pln. ("case 3"); break;
     case "Ganesh": s.o. pln. ("case 4"); break;
     default : s.o.pln. ("from default");
    }
  }
}
```

Output. — case 1.

6)
```
class Demo
{
    p s v m (s[]a)
    {
      String a = "Ramesh";
      switch (a)
      {
        case "Ramesh": s.o. pln. ("case 1");
        case "suresh": s.o. pln. ("case 2");
        case "mangesh": s.o.pln. ("case 3");
        case "Ganesh": s.o. pln. ("case 4"); break;
        default : s. o. pln. ("from default");
      }
    }
}
```

O/P ⟶ case 1
case 2
case 3
case 4

7)
```
class Demo
{
    p s v m (s[]a)
    {
        byte a = 7;
        switch (a)
        {
            case 1 : s.o.pln.("case 1"); break;
            default : s.o.pln.("from default");
            case 2 : s.o.pln. ("case 2"); break;
            case 3 : s.o.pln. ("case 3"); break;
            case 4 : s.o.pln. ("case 3"); break;
        }
    }
}
```

O/P → from default
case 2

8)
```
class Demo
{
    p s v m (s[]a)
    {
        byte a = 7;
        byte b = 4;
        switch (a)
        {
            case 1 : s.o.pln.("case 1"); break;
            case 2 : s.o.pln ("case 2"); break;
            case 3 : s.o.pln ("case 3"); break;
            case b : s.o.pln.("case 3"); break;
```

```
default : s.o.pln.("from default");
    }
}  }
```

O/P → CTE (constant variable required).

9)
```
Class Demo
{
    p s v m(s[]a)
    {
        byte a = 7;
        final byte b = 4;
        switch (a)
        {
            case 1: s.o.pln.("case 1"); break;
            case 2: s.o.pln.("case 2"); break;
            case 3: s.o.pln.("case 3"); break;
            case b: s.o.pln.("case 3"); break;
            default : s.o.pln.("from default");
        }
    }
}
```

O/P → from default.

10)
```
Class Demo
{
    p s v m (s[]a)
    {
        byte a = 7;
        final byte b = 4;
```

```
switch (a)
{
    case 1:{
            s.o.pln.("case 1");
            break;
            }
    case 2:{
            s.o.pln.("case 2");
            break;
            }
    case 3:{
            s.o.pln.("case 3");
            break;
            }
    default:{
            s.o.pln.("from default");
            }
}
```

O/P. → from default.

11) WAJP to check the Name is consonant or vowel using switch case.

```
class Demo
{
    p s v m (S[]a)
    {
        char a = 'a';
```

```
switch (a)
{
        case 'a';
        case 'e' :
        case 'i' :
        case 'o' :
        case 'u' : S.O.pln. (" It is vowel");
        default : S.O.pln. (" It is consonant");
    }
}  }
```

o/p. ⟶ It is vowel.

— Note. —

1) In switch blocks are mandatory or we will get a compile time error.
2) We cannot write any statement inside a switch other than case and default block.
3) default block is optional in switch
4) we cannot create more than one default block we if we try to create more than 1 default it will be a duplicate so, compiler will give us a compile time error (Duplicate Default label).
5) we can specify default block anywhere inside a switch it means start, between, end
It will always exicute when all of the conditions evaluate to false.
6) case blocks are also optional in switch but it is not recommanded.

* fall through condition in Switch.—

this condition Occurs in switch statement when there is no break keyword mention for the perticular case and cause execution of all the cases till a break statement Occurs or exit from the switch statement.

Note.— This fallthrough condition has its own advantages & disadvantages & ~~its~~ it totally depend upon the type of operation we want to perform in our program.

```
* import java.util.Scanner;
class Demo
{
    p s v m (S[]a)
    {
        Scanner sc = new Scanner(System.in);
        s.o.pln.("Enter a month :");
        String month = sc.next();
        Switch (month.toLowerCase());
        {
          case "oct":
          case "nov":
          case "dec":
          case "jan": s.o.pln.("it's a winter"); break;
          case "feb":
          case "mar":
          case "Apr":
```

```
case "may": s.o.pln.(" Its a summer"); break;
case "Jun":
case "Jul":
case "Aug":
case "sep": s.o.pln.(" It's a mansoon"); break;
    }
}  }
```

o/p. → Enter a month : Mar
Its a summer.

```
* import java.util.scanner;
class Demo
{
    p s v m (S [] a)
    {
        Scanner sc = new Scanner(system.in);
        s.o.pln.(" Enter a number :");
        byte num = sc.nextByte();
        switch (num)
        { case "1" : s.o.pln.("Jan"); break;
          case "2" : s.o.pln.("Feb"); break;
          case "3" : s.o.pln.(" Mar"); break;
          case "4" : s.o.pln.(" Apr"); break;
          case "5" : s.o.pln.("May"); break;
          case "6" : s.o.pln.("Jun"); break;
          case "7" : s.o.pln.("Jul"); break;
          case "8" : s.o.pln.(" Aug"); break;
          case "9" : s.o.pln.(" sep"); break;
          case "10" : s.o.pln.("Oct"); break;
          default : s.o.pln.(" you entered wrong No.");
    }  }
}
```

* Final.

- final is a keyword and a non access modifier in java which is used in different members like class, methods, variables.
- If we are prefering final keyword to a variable it becomes a constant expression.
- It means we cannot do reintiallization of that variable.

```
- class Demo
{
        p s v m (S[]a)
        {
            final int dob = 2000;
            s.o.pln ("date of birth :"+dob*);
            dob = 2004;
            s.o.pln ("date of birth :"+dob);
        }
}
```

o/p → CTE (

switch	Ladder
1) Expression inside switch statement (case label) decides which case needs to be executed.	1) expressions inside ~~Ladd~~ if statement decides to whether to execute statements inside if block or not.
2) the switch checks only for equality.	2) It checks for equality as well as logical expressions

3) switch block is mandatorwy.	3) Blocks are not mandatory.
4) switch is faster in execution.	4) Ladder is slower in execution.
5) we cannot pass long, float, double & boolean in ~~double~~ switch.	5) We can use all datatypes for a conditional expression in if.
6) we cannot create duplicate case label in switch.	6) we can create sim ar conditions in ladd
7) we can use break statement.	7) we cannot use break statement in ladder it is not present in de a loop.
8) we can create a default block in switch.	8) we cannot use defau block in ladder.

* Loop Statement. —

loop statements are control flow statement which helps the programmer to execute the set of instructions repeatedly.

* Types of loop. —

1) for loop.
2) while loop.
3) do while loop.
4) for each loop/ or Enhanced loop /or advanced ~~loop~~ for loop.

* parts of a loop. —

1) Initiallization.
2) Condition.
3) updation.

1) Initialization. — —
It has to be done before the starting of the loop.

2) Condition —
- Condition is used to control the loop.
- If condition is true the loop continuoues.
- If the condition is false, the loop stops or gets turminated.

3) update. —

- An update statement should be executed in eve cycle or iteration of the loop.
- The update statement must help the condition become false after desired number iteration.

* ~~for loop~~ —
* Extra. —

```
int a = 2
switch (a)
{
    case 1:
    case 2:
    ⋮
    case n : break;
    default :
}
```

switch block

case label

- Range is depends on data that we pass in switch.
- where is a key word start using ~~after~~ from JDK 17.
- In Java we cannot create Duplicate.
- cannot pass Boolean Because, Boolean has only two possible ~~out~~ values (true & false), so we can use if or if else.

For Loop. —

- for is a loop statement used to execute the set of statements repeatedly.

- syntax. — (for (initialization ; condition ; update)
  {
  (statement);
  }

i value gets updated by 1 in every iteration.

control → 1 → 2 5 8 11 → 4 7 10

```
for (int i=1; i<=3; ++i)
{
    S.O.P.(i);
}
```

i 1 <= 3 (true)
ii 2 <= 3 (true)
iii 3 <= 3 (true)
iv) 4 <= 3 (false) → loop gets terminates because condition become false.

control goes inside block for execution statements because cond^n evaluates true.

3 6 9

op → iteration 1 i → 1
iteration 2 i → 2
iteration 3 i → 3 ...

1) WAJP to display 1 to 100

```
Class Demo
{
    p s v m(S[]a)
    {
        for (int i=1; i<=100; i++)
        {
            s.o.p (i+" ");
        }
    }
}
```

O/P → 1 2 3 4 5 6 7 8 100

2) WAJP to display the char A to Z.

```
Class Demo
{
    p s v m (S[]a)
    {
        for (char c=65; c<=90; c++)
        {
            s.o.p(c+" ");
        }
    }
}
```

O/P → A B C D E F G H I ..... Z

3) WAJP to display a to z.

```
class Demo
{
    p s m v (S[]a)
```

```
{
        for (char c=97; c<=122; c++)
        {
            s.o.p.(i+"  ");
        }
}   }
```

O/P → a b c d e f g h i ..... z

4) WAJP to display 100 to 1

```
class Demo
{
        p s v m (s[]a)
        {
            for (int i=100; i>=1; --i)
            {
                s.o.p(i+"  ");
            }
        }
}
```

O/P → 100 99 98 97 96 95 1

5) WAJP to display Z to A.

```
class Demo
{
        p s v m (s[]a)
        {
            for (char c=90; c>=65; --c)
            {
                s.o.p.(c+"  ");
            }
        }
}
```

O/p → Z ~~X~~ ~~Y~~ Y X W ..... A

6) WAJP to display Alternate characters

```
Class Demo
{
        p s v m (s[] a)
        {
            for (char c='65; c<=90; c+=2)
            {
                s.o.p.(c+"  ");
            }
        }
}
```

O/p → A C E G I K M O Q S U W Y

7) WAJP to display characters with their ASCII values.

```
Class Demo
{
        p s v m (s[] a)
        {
            for (char c='A'; c<='Z'; c++)
            {
                s.o.p.(c+"  "+(c+0)+"  ");
            // s.o.p.(c+"  "+(c*1)+"  ");
            // s.o.p.(c+"  "+(int)c+"  ");
            }
        }
}
```

O/p → A 65 B 66 C 67 D 68 ..... Z 9

8) WAJP to display the even no.s with betn user entered the range.

```
import java.util.Scanner;
class Demo
{
        ps vm (S[]a)
        {
          Scanner sc = new Scanner(system.in);
          s.o.p.("Enter the Range :");
          int fnum = sc.nextInt();
          int Lnum = sc.nextInt();
          for (int i = fnum; i<= Lnum; i+=2)
          {   if(i%2==0)
              {
                s.o.pln(i+" ");
              }
          }
        }
}
```

O/p. → Enter the Range : 10
20

10 12 14 16 18 20

10) WAJP to display the odd nos betn user entered Range.

→

```
import java.util.Scanner;
class Demo
{
        p s v m (S[]a)
        {
          Scanner sc = new Scanner(system.in);
          s.o.p.("Enter the Range :");
          int fnum = sc.nextInt();
          int Lnum = sc.nextInt();
          for (int i = fnum; i<=lnum; i++)
```

```
        {   if(i % 2 = 0)
            s.o.p.("i+" "");
        }
    }
}
```

O/P. → Enter the Range : 10
20

11 13 15 17 19

12) WAJP to display the even odd nos. without using mod

```
class Demo
{
        p s v m(s[]a)
        {
            int num =8;
            s.o.pln(((num/2)*2 == num)?"Even":"Odd
        }
}
```

O/P. → Even.

11) WAJP to find count of the digit.

```
import java.util.Scanner;
class Demo
{
    p s v m(s[]a)
    {
        Scanner sc = new Scanner(system.in);
        s.o.pln("Enter the Number :");
        int num = sc.nextInt();
        int count = 0;
```

```
int i = 1; num1 != 0; num1 /= 10)
for ( ; num1 != 0 ; num1 /= 10)
{
    count++;
}
{
    S.o.pln.(count);
}
}  }
```

O/P. ⟶ Enter the Number : 1235 4.

12) WAJ P to print the reverse Number.

```
class Demo
{
    p s vm (S [Ja)
    {
        int num = 1234;
        int rev = 0;
        for (int i=num; i != 0 ; i /= 10)
        {
            int rem = i % 10;
            rev = rev * 10 + rem;
        }
        {
            s.o.pln.(rev);
        }
}  }
```

O/P. ⟶ 1234

4321

3) WAIP to check wether the user no entered is pallindrome or not.

```
import java.util.Scanner;
class Demo
{
    p s v m (S[]a)
    {
        Scanner sc = new Scanner(system.in);
        s.o.pln.(" Enter a Digit : ");
        int num = sc.nextInt();
        int rev = 0;
        for (int i=num; i!=0; i/=10)
        {
            int rem = i%10;
            rev = rev*10+rem;
        }
        {
            s.o.pln.((rev)?" A pallindrome":"not a
                        pallindrome");
        }
}   }
```

O/p. → Enter a digit : 121
A pallindrome.

4) WAIP to find factorial number (using Scanner clas

```
import java.util.Scanner;
class Demo
{
    p s v m (S[]a)
```

```
{
    Scanner sc = new Scanner (system.in);
    s.o.p. (" Enter a digit :");
    int num = sc. nextint();
    int fact = sc. 1;
    for (int i=1; i<=num; i++)
    {
        fact * = i;
        ~~s.o. pln ( fact);~~
    }
        s.o. pln. (fact)
}
```

O/p ⟶ Enter a digit : 5

~~4 3 2~~ 120

i=1 f=1

1×1=1

1×2=2

i=3×2=6

i=4×6=24

i=5×24=120

i=6

factorial ⟶ 3 = 3×2×1

Factor ⟶ 12 = 1, 2, 4, 3, 6, 12

5) WAJP to find the User Entered Number's ~~is~~ factor

```
import java.util.Scanner;
class Demo
{
    p s v m(s[] a)
    {
        Scanner sc = new Scanner (system.in);
        s.o.p ("Enter a ~~Dt~~ Number :");
        int num = sc.nextInt();
        for (int i=1; i<=num; i++)
        {
            if (num%i==0)
            {
                s.o.pln.(i+" ");
            }
        }
    }
}
```

o/p. → Enter a Number : 30
1 2 3 5 6 10 15 30

6) WAJP to find the number is prime or not (Using Scanner class).

```
import java.util.Scanner;
class Demo
{
    p s v m (s[] a)
    {
        Scanner sc = new scanner (system.in);
        s.o.p.("Enter ~~the~~ a Number :");
        ~~scan~~ int num = sc.nextInt();
```

```
int count = 0;
for (int i=2; i< num; i++)
{
    if (num % i == 0)
    {
        count ++;
    }
}
s.o.pln.(count>0?" not a prime no ":" prime
                no.");
}    }
```

O/P. → Enter a number : 8
not a prime no.

7) WAP to print the fibonacci series.

```
Class Demo
{
    p s v m (S[]a)
    {
        long num1 = 0;
        long num2 = 1;
        s.o.pln.(num1+"  "+num2+"  ");
        for (int i=1; i<=10; i++)
        {
            long op = num1 + num2;
            s.o.p.(op+"  ");
            num1 = num2;
            num2 = op;
        }
}    }
```

o/p. → 0 1

0 1 2 3 5 8 13 21 34 55 89.

1)
```
p s v m (s[]a)
{
    int i=1;
    for (; i<=10; )
    {
        s.o.p. ("hello");
        i++;
    }
    s.o.pln. ("byee");
}
}
```

o/p. →

2)
```
p s v m (s[]a)
{
    for (; true ; ){
            s.o.pln. ("hello");
            }
            s.o.pln. ("byee");
}
}
```

o/p. →

3)
```
p s v m (s[]a)
{
    for ( ; false ; ) {
                s.o.pln. ("hello");
            }
            s.o.pln. ("byee");
}
```
O/P →

4)
```
p s v m (s[]a)
{
    boolean a = true;
    for ( ; a ; ) {
                s.o.pln. ("hello");
            }
            s.o.pln. ("byee");
}
```
O/P →

5)
```
p s v m (s[]a)
{
    int b = 0;
    boolean a = true;
    for ( ; a ; ) {
                s.o.pln ("hello");
                b++;
                if (b == 1000)
                {
                    break;
                }
            }
```

```
            s.o.pln.("byee");
    }       }
```

o/p. →

6)
```
p s v m (s[]a)
{
    int b = 0;
    boolean a = true;
    for(;a;){
            s.o.pln("hello");
            b++;
            if (b==1000)
            {
                a = false;
            }
        }
        s.o.pln.("byee");
    }
}
```

o/p →

7)
```
p s v m (s[]a)
{
    final boolean a = true;
    for(;a;){
            s.o.pln.("hello");
        }
        s.o.pln.("byee");
    }
}
```

o/p →

8) p s v m (S[]a)

```
{
        final int a=1;
        final int b=4;
        for (; b>a; ) {
                s.o.pln.("hello");
                }
                s.o.pln.("byee");
}
```

o/p →

Breaking statement

1) When we know the number of iteration we go for for loop.
2) The main disadvantage of using loops are unreachable statement.
   i) if condition is true (constant value) statement after the loop will be unreachable.
   ii) if condition is false (constant value) statement inside the loop will be unreachable.

3) These situation or condition is applicable for for loop | for for while | do while.
3) Blocks are optional and only one statement is considered as a part of that loop.
4) If a programmer fails to provide cond^n which is true & it is create an infinite loop.

* Squareroot Nuber —

```
class Demo
{
    p s v m (s[]a)
    {
        int num=100;
        int sqrt =0;
        for (int i=1; i<=num; i++){
        if (i * i == num) {
                    sqrt = i;
                    break;
        }        }
        s.o.pln. (sqrt==0?"not a perfect":" squareot is "+sqrt)
}   }
```

While loop. —

- It is a looping statement which help the programer to execute the set of instructions Repetedly until the cond^n become false.
- We use it when we dosen't know the number of iteration.

Syntax. —

```
Initialization
while (conditions)
    {
        Statements;
        Update;
    }
```

1)
```
Class Demo
{
    p s v m (S [] a)
    {
        int i=1;
        while (i<=10)
        {
            S.o.p.(i);
            i++;
        }
    }
}
```

o/p ⟶ 1
2
3
⋮
10

(2)
```
class Demo
{
    p s v m(s[]a)
    {
        boolean a = false;
        while (a)
        {
            s.o.pln. ("hello");
        }
    }
}
```

O/p → CTE.

(3)
```
class Demo
{
    p s v m (s[]a)
    {
        while (true)
        {
            s.o.pln. ("hello");
        }
        s.o.p. ("bye");
    }
}
```

O/p. → CTE (Unreachable statement bye)

1) WAJP to print a to z.

```
class Demo
{
    p s vm (s[]a)
    {
        char c=65;
        while (c<90)
        {
            s.o.p.(c+" ");
            c++;
        }
    }
}
```

o/p → a b c d e f ..... z

2) WAJP to display reverse number
input

```
class Demo
{   p s v m (s[]a)
    {   Scanner sc= new Scanner (system.in);
        s.o.p. ("Enter your Number :");
        int num = sc.nextInt();
        int rev =0;
        while (num!=0);
        {
            int rem = num %10;
            rev = rev * 10 +rem;
            num /=10;
        }
        s.o.p. ("After Reversing No is:" + rev);
    }
}
```

3) WATP to display table (User input).

```
import java.util.Scanner;
class Demo
{
    Scanner sc = new Scanner(System.in);
    S.o.pln("Enter a No : ");
    int num = sc.nextInt();
    int op;
    int i = 1;
    while (i<=10)
    {
        op = num * i;
        S.o.pln(num+" * "+i+" = "+op);
        i++;
    }
}
}
```

O/P → Enter a No. 2

2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20

Imp: * Difference

i	for loop	while loop
1)	When we know the no of iteration we use for loop	1) While loop is used when the number of iteration is not known.
2)	Initialization can be done inside the loop or before starting of the loop.	2) Initialization needs to be done before starting of the loop.
3)	the iterates infinite no. of times if the cond^n is not specified.	3) If the condition is not specified it shows a compilation error.
4)	for loop is faster Because	4) While loop is slower Because checks cond^n for every iteration

- Note. ⟶ if for loop no. is of iteration is defined then we can say the loop will execute at a faster rate. Because, a number of iterations are known by the virtual machine and to improve the performance it skills the condition.
- There is negligible difference between for loop & while loop execution.

* Do while loop. —

- It is a looping statement and helps the programmer to execute the set of Instructi Repeatedly.
- It give first priority to the statement which needs to be present inside the do block.

Syntax. —

```
class Demo
{
    p s v m (S[]a)
    {
        do Initialization
        do
        {
            // statements;
            update;
        } while (Condition);
    } }
```

* Defination of do-while loop. —

1) do while loop is a loop statement where the work is done first and then condition is checked i.e. the control first goes to the do block and execute the statements.
2) After the execution of the statement the control to while to check for the condition.

3) If the condition is true, the loop continues
4) If the condition is false, the loop terminates
5) The minimum number of execution of a do-while loop is 1
6) The minimum number of execution of a while loop is 0.

* WAJP to print 1 to 10.

```
class Demo
{
    p s v m (S[]a)
    {
        int num = 1;
        do
        {
            S.O.p.(i)
            i++;
        }
        while (i<=10);
    }
}
```

o/p. → 1
2
3
4
5
6
10.
7
8
9
10.

```
import java.util.Scanner;
Class Password
{
    p.s.v.m(s[]a)
    {
     scanner sc = new scanner (system.in);
     int Pin1 =123;
     int count =3;
    do
    {
        s.o.p. ("Enter your Pin:");
        int Pin2 =sc.nextInt();
        if (Pin1== Pin2)
        {
            s.o.p. ("phone has been Unlocked");
            break;
        } else
        {
            s.o.p. ("Incorrect Pin" + (count-1) +"
                    Attempts left:");
        }
         count --;
        }
    while (count != 0);
    }
}
```

O/p. ⟶ Enter your Pin : 123
phone has been Unlocked.
Enter your pin : 321
incorrect pin 2 Attempts left.

* Difference between while and do while loop.

while loop	do while loop
1) In while loop cond^n is evaluated first & then loop body is executed ~~first~~.	1) In do while loop body is executed first & then condition is checked.
2) condition is checked at the entry that's why it is also known as "entry control loop."	2) condition is checked at exit that's why it is also known as "Exit control loop."
3) No semicolon is used as a part of syntax.	3) Semicolon is used as a part of syntax.
4) It allows initialization of counter variable before entering loop body	4) It allows initialization of counter variable before & after entering loop body.
5) This loop should be used when condition is more important.	5) This loop should be used when the process is important.
6) minimum number of iteration is '0'. If condition evaluates to false.	6) minimum no. of iteration is '1'. even if cond^n is false.

* Do while loop. —

```
class Demo
{
    p s v m (S[]a)
    {
        int i;
        do{  i=1;
                s.o.pln.(i);
                i++
    }  }  } while (i<10);
```

* Continue. —

- the continue keyword is used to end the current iteration in a loop & continues the next iteration.
- when a continue statement is encount the control is directly jumpes to the begining of the loop for the ne iteration insted of executing the statemen of the current iteration.
- It is a branching statement.

* program for continue. —

1) print even no. —

```
class Demo
{
    p s v m (S[]a)
    {
        for (int i=1; i<=100; i++)
        if (i%2==0)
```

```
    {
        continue;
    }
        s.o.p.(i);
    }
}
```

O/p. → 0, 2, 4, 6, 8, ..., 100.

2) WAIP skip that no which is divisible by 5.

```
class Demo
{
    p s.v.m.(S[]a)
    {
        for (int i = 1; i<=100; i++)
        {
            if (i%5==0)
            {
                continue;
            }
            s.o.p.(i+" ");
        }
}}
```

O/p. → 1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, ..., 99.

QSPIDERS
25 OCT 2023
QSPIDERS
25 OCT 2023

reverse the no & count.

1)
```
class Demo
{
    public static void main (String [] args)
    {
        ~~int~~ int num;
        count = 0;
        ~~for ( int i=1; i<=~~

        for( int ~~int~~ i= num; i<= num; i++)
        {
            count ++;
            s.o.p.(count);
        }
    }
}
```

local | Global

2)
```
class Demo.
{
    public static void main (String [] args)
    {   int i = 123;
        rem = 0;
        int rev;

        For ( int ; i!=0; i % 10==0)
        {   rev++
            ~~s.o.pln(rev);~~
        }
        s.o.pln. (rev);
    }
}
```

123
123/10
=12

rev
+
rev
i /
int % 10==0

32

rev
123

123 | 2

```
* class Demo
{
    p s v m (s[]a)
    {
      for(char i='a'; i<='f'; i++)
      {
        for(char j='a'; j<=i; j++)
        {
          s.o.p.(j+" ");
        }
        s.o.pln();
      }
}  }
```

o/p. → a
a b
a b c
a b c d
a b c d e

```
* class Demo
{
    p s v m (s[]a)
    {
      for(int i=1; i<=5; i++)
      {
        for (int j=1; j<=i; j++)
        {
          s.o.p. (j+" ");
        }
        s.o.pln();
```

```
        }
}   }
```

o/p. ⟶

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

* class Demo

```
{
    p s v m (s[]a)
    {
        char c='a';
        for (int i=1; i<=5; i++)
        {
            for (int j=1; j<=i; j++)
            {
                s.o.p.(c+" ");
                c++;
            }
            s.o.pln.();
        }
}   }
```

o/p. ⟶

```
a
b c
d e f
g h i j
k l m n o
```

* ~~Demo~~ class Demo

```
{
    p s v m (s[] a)
    {
        int c=1;
        for(int i=1; i<=5; i++)
        {
            for (int j=1; j<=i; j++)
            {
                s.o.p.(c+" ");
                c++;
            }
            s.o.pln.();
}   }   }
```

O/p ⟶ 1
2 3
4 5 6
7 8 9 10
11 12 13 14 15

* Class Demo

```
{
    p s v m (s[] a)
    {
        int num=2;
        for (int i=1; i<=5; i++)
        {
            for (int j=1; j<=i; j++)
            {
                s.o.p.(num+" ");
                num+=2;
            }
```

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

O/p. → 2
4 6
8 10 12
14 16 18 20
22 24 26 28 30.

```
* class Demo
  {
      p s v m (s[]a)
      {
          int num = 1;
          for (int i=1; i<=5; i++)
          {
              for (int j=1; j<=i; j++)
              {
                  s.o.p. (num+" ");
                  num t=2;
              }
          }
  }   }
```

O/p. → 1
3 5
7 9 11
13 15 17 19
21 23 25 27 29.

```
* class Demo
{
    p s v m (s[]a)
    {
        for(int i=1; i<=5; i++)
        {
            for(int j=1; j<=1; j++)
            {
                s.o.p.("*"+" ");
            }
            s.o.pln();
        }
}   }
```

Olp. →

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

```
* class Demo
{
    p s v m (s[]a)
    {
        for(int i=5; i>=1; i--)
        {
            for(int j=1; j<=i; j++)
            {
                s.o.p.("* ");
            }
            {
```

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

O/p. →

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

* class Demo

```
{
    p s v m (s[]a)
    {
        int a=0;
        int b;
        for (int i=1; i<=5; i++)
        {
            b = a+i;
            a = a+i;
            for (int j=1; j<=i; j++)
            {
                s.o.p.(b--+" ");
            }
            s.o.pln();
        }
} }
```

} } O/p. →

```
1
3 2
6 5 4
10 9 8 7
15 14 13 12 11
```

* Armstrong Number. —

```
import java.util.scanner;
class Demo
{
    p s v m (s[] a)
    {
        scanner sc = new Scanner (system.in);
        s.o.pln. (" Enter a Number:");
        int num = sc.nexIInt();

        int dup = num;
        ind dup1 = num;
        int count = 0;
        int sum = 0;
        while (dup > 0)
        {
            count ++;
            dup1 = 10;
        }
        while (dup1 > 0)
        {
            int rem = dup1 % 10;
            int pow = 1;
            for (int i = 1; i <= count; i++)
            {
                pow * = rem;
            }
            sum + = pow;
            dup1 / = 10;
```

```
    }
    s.o.pln (sum==num ?" It is ~~palind~~ Armstrong"
                        " Not a Armstrong no.");
    }
}.
```

O/P → 153
It is a Armstrong no.

* Automorphic Number. —

```
import Java.util.Scanner;
class Automorphic
{
    p s v m (s[]a)
    {
      Scanner sc=new Scanner (system.in);
      S.O.pln ("Enter a Number :");
      int num = sc.next Int();
      int dup = num;
      int sqr = num*num;
      int count =0;
      int div = 1;
      while (dup>0)
      {
          count ++;
          dup/ =10;
      }
      for ( int i= 1; i<= count ; i++)
      {
          div * = 10;
      }
```

```
s.o.pln.((num ==(sqr % div))?" Automorphic":
                                " Not an automorphic");
    }
}
```

O/p. ⟶ Enter a Number : 25
Automorphic

* Sunny Number. —

```
import java.util. Scanner;
class SunnyNO
{
    p.s.v.m. (S[]a)
    {
      Scanner sc= new Scanner (system.in);
      s.o.pln. ("Enter a Number : ");
      int num = sc. nextInt();
      int b = num + 1;
      int sqrt =0;
      for (int i=1; i<=num/2 ; i++)
      {
          if( i*i == b)
          {
              sqrt =i;
              break;
          }
      }
      if( sqrt >0)
      {
          s.o.p.("It's a sunny Number");
      }
```

```
else
 {
    s.o.pln. ("## It is not a sunny Nuber");
 }
} }
```

o/p. ⟶ Enter a Number : 80
It is a sunny Nuber.

* Neon Nuber. —

```
import java.util.Scanner;
class Neon
{
    p s v m (S[] a)
    {
        Scanner sc=new Scanner (System.in);
        S.o.pln. ("Enter a Number :");
        int num=sc.nextInt();
        int op = num * num;
        int sqrt=0;
        int sum= 0;
        while (num! = 0)
        {
            int rem = num%10;
            sum += rem;
            num /=10;
        }
        pln. (sum == num ? "Neon No" : "Not a Neon No
    }
}
```

O/p. → Enter a Number : 9
Its a Neon Number.

* WAJP to find prime factors of a Name number.

```
import java.util.Scanner;
class primeFactors
{
    p.s.v.m.(s.CJA)
    {
        Scanner sc = new Scanner (System.in);
        S.o.pln. ("Enter a Number :");
        int num = sc.nextInt();
        for (int i=1; i< num; i++)
        {
            if (num % i == 0)
            {
                int num2 = i;
                int count = 0;
                for(int j=2; j<num2; j++)
                {
                    if (num2 % j == 0)
                    {
                        count ++;
                        break;
                    }
                }
                if (count == 0 && num2 > 1)
                {
                    s.o.p(num2+" ");
                }
```

```
        } }
    }
```

O/p. ⟶ Enter a Number : 20
25.

* Perfect Number. —

```
class perfect
{
    p s v m (s[]a)
    {
        int num = 6;
        int sum = 0;
        int i = 1;
        while ( i<= num/2)
        {
            if (num % i == 0)
            {
                sum = sum + i;
            }
            i++;
        }
        if ( sum == num)
        {
            s.o.p. (num + " It is a perfect Number
        }
        else
        {
            s.o.p. (num + " It is not a perfect No");
        }
```

```
    }
}
```

O/p. → 6 is a perfect Number.

*

```
impor
class krish
{
    p s v m (S[]a)
    {
        Scanner sc=new scanner(System.in);
        s.o.pln. ("Enter a number :");
        int num = sc. nextInt();
        int sum =0;
        for ~~fact=1;~~ (int i=num; i>0; i/=10)
        {
            int fact =1;
            int digit =i%10;
            for (int j=1; j<=digit; j++)
            {
                fact *=j;
            }
            sumt=fact;
        }
        if(sum ==num)
        {
            s.o.p. ("krishnah Nuber");
        }
        else {
```

```
            pln. (" Not a kaprekar No ");
        }
    }
}
```

O|P. ⟶ Enter a Number : 143.
It is a kaprekar Number.

Duck Number. —

```
class DuckNo
{
    p s v m (s[]a)
    string a = "120";
    if (a. charAt (0) == '0')
    {
        s o.pln. (" Not a Duck No ");
        system. exit(0);
    }
    else
    {
        boolean b= false;
        for (int i= 0; i<=a. length(); i++)
        {
            if (a. charAt (i) == '0')
            {
                b = true;
                break;
            }
        }
        if (b)
```

```
    {
        pln. ("It is a Duck No");
    }
    else
    {
        pln.("It is not a Duck No");
    }
  } }
}
```

O/p. → Enter a Number : 1203
It is a duck Number

31 /10/23

* Methods :—

Methods are the block of codes or set of Instructions which is design to perform a specific task. Method gets executed when ever it is called.

* Syntax. —

access modifier | non-access modifier | return type | method name (formal argument)

method block/ body.

```
{
    // statement.
}
```

implementation.

* Ex. :—

```
public static void mansi ()
{
    s.o.pln. ("Hello World");
}
```

* Advantages. —

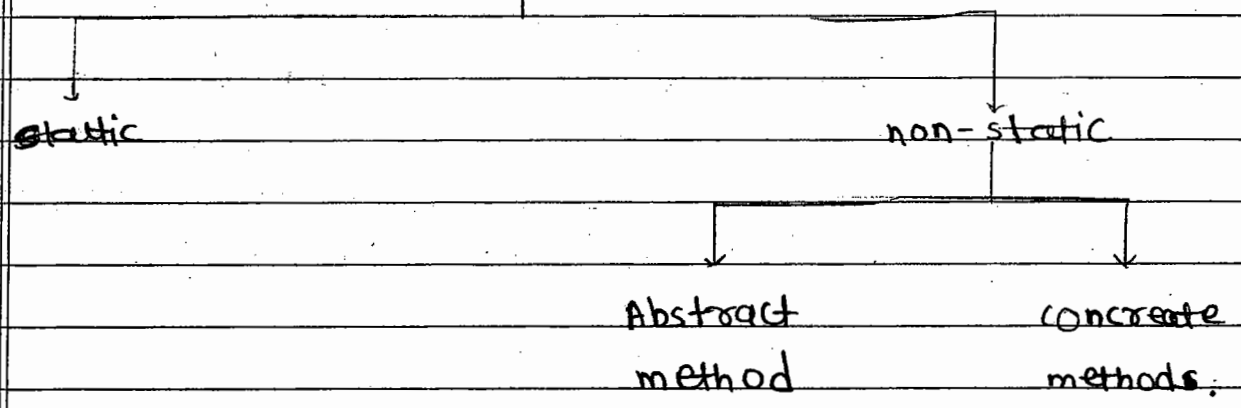
1) It allows code reusability. (Define once & use multiple times)

2) ~~2)~~ We can break a complex program into a smaller peice of code.
3) It is used for code modularity.
4) It increases code readability.

* Classification / types of method. —

1) Based on formal Arguments. —
   i) No argument method. —
   ii) Parameterised method. —

2) Based on Non-Accesed modifier. —



3) Types of methods. —
   1) Buildin method or predefined method.
   2) User defined

* Terminologyies of Method. —

method defination.

method declaration.

access modifier | non access modifier | return type | method name (formal argume

method signaturse.

method body/block.

{

// statement.

}

* Method Signaturse. —
It is a combination of method name and formal argument.

* method declaration. —
It is a combination of access modifier, non access modifier, return type & method signaturse.

* method body. —
A method is a place where all of the actions take place.
Basically It countains the Implementation

* method defination. —
It is a combination of method body & method decl^n.

* modifiers —
modifiers are the keywords which modifies (changes) the behaviour ~~or~~ of Java members or components.

There two types of modifiers —
1) access modifier / access specifiers
2) Non access modifier.

1) Access Modifier. — = talks about Range
Access modifiers are keywords that can be used to control the visibility of feilds, methods and constructors in a class.
There are 4 access modifier in Java.
1) Public. — wide Range
2) protected.
3) default. / Packedge level modifier.
4) private.

* Non-access Modifier. —
~~modifi~~ non-access modifiers are keyword words which alters the behaviour of Java members or components.

* Types of Non-access Modifier. —
1) static
2) final
3) abstract
4) synchronized
5) volatile

06/10/23.

6) transient.

7) strictfp. strick floating point

Return type. –

- Return types specifies either the type of va is Return (Data type) or Nothing is Return after the execution of method.
- If the method is not written any thing th return type should be void.
- If the method is writening something ~~the write type~~ that must be a data type Insted of void.
- It might be primitive or Non-primitive ~~data~~ type.

Method Name. –

- A method Name is an Identifyer which is used to Identify a method or to call a metha
- We have to follow the rules of Identifiers a well as conventions to write a method Nam (camel case).

Formal Arguments –

- Formal Arguments are the variables creaated in the method declaraction statement.
- Formal arguments are the local variables of that method.
- Formal Arguments are ~~closed~~ used if the metha wants to accept any Input ~~data~~ at the time of calling.

1)
```
class Demo
{
    p s v m (s[]a)
    {
        s.o.pln ("main starts");
        ramesh (10);   // method calling state
        s.o.pln. ("main ends");
    }
    p s v ramesh (int a)
    {
        s.o.pln ("Ramesh starts");
        s.o.pln (a +" from main");
        s.o.pln (suresh () +" from suresh");
        s.o.pln ("Ramesh ends");
    }
    p s int Suresh()
    {
        s.o.pln. ("Suresh starts");
        int b = 20;
        return b;
    }
}
```

o/p → main starts
ramesh starts
10 from main
suresh starts
20 from suresh
ramesh ends
method ends

2)
```
class Demo
{
    public s v m (S[] a)
    {
        int op = m1("Ramesh", 9.9, false);
    }
    p s string m1(string name, double cgpa, boolean
                  bcklg)
    {
        s.o.pln.("My name is "+ name);
        s.o.pln.(" I have scored "+ cgpa+" cgpa");
        s.o.pln.(bcklg);
        return "Good Evening";
    }
}
```

o/p. ⟶ my name is ramesh
I have scored 9.9 CGPA
false

3)
```
class Demo
{
    p s m v m(S[]a)
    {
        s.o.pln (add(20, 10));
        s.o.pln (sub(20,10));
        s.o.pln (div(20, 10, 2));
    }
    p s int add (int a, int b)
    {
```

```
        int c = a + b;
        return c;
    }
    p s int sub (int a, int b)
    {
        int c = a-b;
        return c;
    }
    p. s. double div (int a, double b)
    {
        double c = a/b;
        return c;
    }
  }
}
```

o/p. ⟶ 30
10
9

4)
```
class Demo
{
    p s v m (s [] a)
    {
        m1();
    }
    p s v m1()
    {
        s.o.pln. ("m1()");
        m2 ();
    }
```

```
p s v m2()
{
    s.o.pln ("m2()");
    m2();
}
}
```

O/p → m2()
m2()
m2()
StackOverflowError

5)
```
class Demo
{
    static char ch = 'A';
    p s v m (s[]a)
    {
        m1();
        s.o.pln ("exe ends");
    }
    p s v m1()
    {
        if (ch==91)
        {
            return;
        }
        s.o.pln (ch++);
        m1();
    }
}
```

o/p. →

6)
```
Class Demo
{
    static int fact = 1;
    p s v m( s[]a)
    {
        Factorial(5);
    }
    p s v factorial(int num)
    {
        if(num==0)   // num== → true.
        {
            s.o.pln (fact);
            return; // going back to the caller method.
        }
        fact = fact* num ; // 1*5   5*4   20*3   60*2
                                  120*1   120*0
        factorial (--num); // 4   3   2   1   0
    }
}
```

o/p. → 120.

AIP to find factors using recursion

Class Demo

```
P.s.v.m.(s[]a)
{
    int num1 = 10;
    factors (num1, 1);
}
P.s.v. factors( int n, int i)
{
        if ( i<=n)
        {
            if ( n % i == 0)
            {
                s.o.p. ( i+" ");
            }
            factors (n, i+1);
        }
}
```

→ true.

ack to the c

5 5*4 20*3

*1 120*0

4 3 2

IP → 1, 2, 5, 10.

07/10/23

*

- A method gets executed only when it is called.
- When we are not calling a method anywhere in the program, that created method gets compiled but does not gets executed.
- A method must be called using a method called Statement.

~~Class Demo~~

```
class Demo
{                                          calling method
    P s v m (s[]a)
    {
        m1()  <------------ actual method
    }                                      method call statement
    P s v m1()  <---------- method signature                called m
    {                       <------------ formal argument
        S.O.P.("Hello from m1()");
    }
```

* method called ~~method~~ statement. —
- the statement which is used to invoke (called) invoke a method is called as method calling statement
- If we are ~~ust~~ calling a method inside itself it is called as Recursion.

* Actual Argument. —
- Arguments passed in the method called statement are known as actual argument.
- Actual arguments are the data which is used to pass or store to the variables of formal arguments.

* Note. —
We should not specified any data type for the actual arguments the length of actual and formal argument must be same.

- The sequential Order of the data which we passed or specifying must be compartible.

* Calling method :-
- The calling method is the method that contains the actual call (method called statement).

* called method :-
- The method which is called by another method is called as called method.

* Return :-
- Return is a keyword in Java and it is a branching statement
- Return statements stops the execution of a current method & the frame inside the stack gets distroid and the control is transfered back to the calling method.
- Return statement must be always at the last of the block if we have any statement after the return statement, we will get a compile time error. (# CTE → Unreachable statement.)

* class Demo
{
    P S V m (S [] a)
    {
        string msg = m1(10);
        S.o.pln (msg);
    }

```
p s string m1 (int num)
{
    for (int i=1; i<=20; i++)
    {
        if (i==num)
        {
            return "I have stopped the execution";
        }
    }
    return null;
}
}
```

O/p. →

```
class Demo
{
    p s v m (s[] a)
    {
        S.o.p/n (m1('a'));
    }
    Public static string m1 (char ch)
    {
        if (ch=='b')
        {
            return "value is b";
        }
        else if (ch=='d')
        {
```

```
        return "value is d";
    }
    else if
    {
        return "value is a";
    }
    return null;
  }
}
```

O/P. ⟶

```
* P S V m (S[]a)
{
    string opm1 = m1(10,20);
    system.out.pln.(opm1);
    string opm2 = m2(opm1);
    System.out.println(opm2);
    string opm3 = m3(opm2);
    S.o.pln.(opm3);
  }
  P.S. string m1(int a, int b)
  {
      S.o.pln.("from m1()");
      return "hello";
  } P.S. string m2(String msg)
  {     return (msg + " world");
  } P.S. string m3(string opm2)
  {
      opm2 = "hello ramesh";
}. } . return opm2;
```

* Recursion —

- Recursion is a process where a method call itse to find the result.
- There must be a base condition which acts as a break point to stop the recursion.
- A method in java ~~which~~ that calls itself it is called as a recursive method.

* ~~Note~~ — Stack Overflow error —

If the base condition is not satisfied or defined then the stack overflow problem may arises and we will get a stackOverflow.

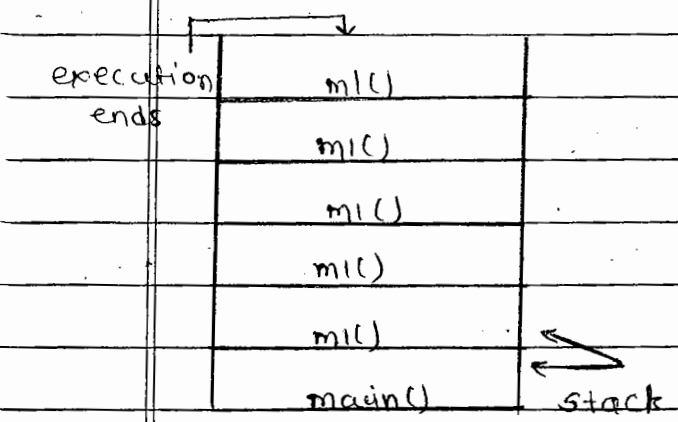
~~main ()~~

m1()
m1()
m1()
m1()
m1()
m1()
main()

```
main () ← Normal meth
call.
{
    m1();
}
m1(); ← recursive metho
call.
{
    s.o.pln. ("Hi");
}
```

dia 1

Stack Over Flow
Error




```
int count =1;
main()
{
    m1(); ← Normal method call
}
m1()
{
    s.o.p.(count++);
    if (count==3)
    return;
    m1(); ← recursive method call
}
```

dig. 2

Note —

- In diagram 1 there was no base condition defined to stop the recursion. so Inside stack for every method call stack is creating a new stack frame and after few calls the stack memory gets full & we get a stack overflow.
- To overcome these error in diagram 2 we have specified a base condition to stop the recursive call to avid the stack overflow error.

Imp. - In Java Recursion is just possible with methods.
we cannot make a recursive call to a constructor.

1) WAJP to find a fibanacci series using recursion meth

```
class fib
{
    static int num1 = 0;
    static int num2 = 1;
    static int op = 0;
    P s v m ( s[] a)
    {
        int count = 10;
        fib (count -1);
    }
    P s v m void fib (int count)
    {
        if ( count > 0)
        {
            op = num1 + num2;
            num1 = num2;
            num2 = op;
            s.o.p. (op + " ");
            fib (count -1);
        }
    }
}
```

o/p. ⟶ 1
2
3
5
8
13
21
34
35

2) WAJP to find Even numbers from 1 to 100 using recursion.?

3) WAIP to find the pallindrome no. using recursion.

```
class Demo
{
    Static int rev = 0;
    P s v m (S[]a) {
     int num = 121;
     int rev = palli (num, 0);
     S.o.pln. ( (rev = = num) ?" Pallindrome. " : " not pallinde
    }

     P . S . int palli (int num , int rev)
     {
          if ( num = =0)
          {
               return rev;
          }
          rev = ( rev * 10) + (num % 10);
          return palli (num /10, rev);
     }
}
```

O/P. → pallindrome.

8/10/23.

* Calling main method Explicitely —

```
class Demo
{
    Static int i=1;
    P s v m (s[]a)
    {
        m1();
    }
    P s v m1()
    {
        if (i==5)
        {
            return;
        }
        i++;
        s.o.pln. ("m1()");
        string [] args = new String[1];
        main (args);
    }
}
```

* Note. —

If we have to call main method explicitely we have to pass string array as an actual argument we have to specify a base condi to stop a Recursion

* Static. —

- static is a keyword and a non-access modifier in java.
- The static keyword in java is used to share the same variables and methods from a given class to some another cla
If we prefixed static modifiers to a java member hence, they are called as static members.
- Static members are loaded inside the class sta area.
- We can use a static member inside the sa class or it can be also excesed Outside t class using class name as a reference.
- We can used static members without Creatin an Instance (object) of a class.

- Static modifiers can be prefixed to ~~variable~~ ~~variou~~
  1) Variables.
  2) Blocks.
  3) methods.
  4) Nested classes

nent

* Static methods. —

- A method which Contains a Static modifier in its Declaration Statement is known as static method
- static methods body will be stored Inside a method area
- And the method signature and it's reference is stored Inside the class static area

- Static methods can be used Inside the same class directly with the help of method called statement.

```
* class Demo
{
        p s v m (s[]a)
        {
            m1();
        }
        p s v m1()
        {
            s.o.p("Hii from m1()");
        }
}
```

We can also use class name as a reference to call a method which is present inside the same class (which is not mandatory)

```
class Demo1
{
        p s v m (s[]a)
        {
            Demo1.m1();
        }
        p s v m1()
        {
            s.o.p("Hii from m1()");
        }
}
```

We can call a static method from different class within the same file or different file using class name as a reference.
(In this case class name as a reference is mandatory.)

* File 1 —

```
class Demo1
{
    p s v m(s[]args)
    {
        m1();
        Demo1.m1();
    }
}

p s v m1()
{
    s.o.p.("Hi from Demom1()");
}
}
```

O/P. → Hi from Demo m1()

File 2 —

```
class Demo1
{
    p s v m(s[]args)
    {
        Demo3.m2();
    }
}
```

```
class Demo3 {
    p s v m2()
    {
        s.o.p ("Hii from Demo3 m2()");
    }
}
```

o/p. → Hii from Demo3 m2()

- we can call a non-static member inside a static context directly without creating an instance of that class.

```
Class Demo1
{
    static String a = "Hi static";
    Static String b = "Hii non-Static";
    p s v m (S[] args)
    {
        s.o.p. (a);
        m1();
        m2();      // CTE
    }
    p s v m1()
    {
        s. o. p. ("from static m1()");
        s.o.p. (a);
    }
} p s void m2 ()
{
```

```
        s.o.p.("from non-static m2()");
        s.o.p.(a);
    }
}
```

✓* context. —

- A block in a class which belongs to a static member is known as static context.
- If the block belongs to non-static member i known as non-static context.

✓* Static - variable. —

- A variable declared in a class block & prefi with static keyword is known as static variable when a variable is declared as a static then a single copy of that variable is created and it is shared among all the member of that class
- static variables are essentially global variable.

Note. — We can create a static variable only as a class label. —

```
class Demo1
{
    static String a = "static variable.";
    static
    {
        s.o.p.(" 1"+ a);
    }
    p s v m (S[] args)
```

```
p s v m (s[] args)
{
    s.o.p. ("2" + a);
}
p s v m1()
{
    s.o.p. ("3" + a);
}
p s v m2()
{
    s.o.p. ("4" + a);
}
}
```

o/p → 1. static variable
2. static variable

A global variables are loaded inside the class static area & they will be provided with a default value.

```
Class Demo1
{
    static byte a;
    static ~~byt~~ short b;
    Static int c;
    Static long d;
    Static float e;
    static double f;
    static char g;
    static boolean h;
```

```
Static String i;
P S V m(S [] args)
{
    S.O.P. ("byte " + a);          byte -0
    S.O.P. ("short " + b);         0
    S.O.P. ("int " + c);           0
    S.O.P. ("float " + e);         0
    S.O.P. ("double " + f);        0.0f
    S.O.P. ("char " + g);          0.0d
    S.O.P. ("boolean " + h);       space
    S.O.P. ("String " + i);        false
}
}
```

* Scope of static Variable. —

1) class file1

```
{
    static string a = "Rawesh";
    p s v m (s [] args)
    {
        s.o.p. ("main ()");
        s.o.p(a);
    }
    p s v m1()
    {
        s.o.p. ("Ex 11 m1()");
        s.o.p(a);
        s.o.p. (Ex11a);
    }
}
```

class file 2

```
{
    p s v m2()
    {
        s.o.p ("Ex12 m2()");
        s.o.p(a);                    // CTE.
        s.o.p. (Ex11 a);
    }
}
```

class file 3 → new separate file

```
{
    p s v m2()
    {
        s.o.p. ("Ex13. m3 ()");
        s.o.p. (Ex 11a);
```

```
}
}
```

O/P → javac file1.java ← compilation is mandato
java file1.java
→ main();
Ramesh

2)
```
class file1
{
    static string a;
    p s v m(s [] args)
    {
        a = "Ramesh";
        s.o.p.(a);
        m1();
        file2.m2();
    }
    p.s.v.m1()
    {
        a = "Suresh";
        s.o.p(a);
    }
}
class file2
{
    p s v m2()
}
file1.a = "Ganesh";
    s.o.p.(file1.a);
}
}
```

o/p. → Ramesh
Suresh
Ganesh

3)
```
Class Example3
{
    Static String a = "I am global";
    P S V M (String [] args);
    {
        String a = "I am local";
        S.o.p.(a);
        S.o.p(Example3.a);
        a = "noo";
        S.o.p(a);
        S.o.p(Example3.a);
        Example3.a = "blue";
        S.o.p(Example3.a);
    }
}
```

o/p. → I am local
I am global
noo
I am global
bye

e.g. – public static String a = "Hello";

```
class StaticInit
{
    Static int a = m1();
    P.S.V.M. (S[] args)
    {
      S.O.P(a);
    }
    P S int m1()
    {
        return 100;
    }
}
```

- class Demo

```
{
    Static int a;
    P S V m(S[] a)
    {
        S.O.P(a);
    }
}
```

o/p. → 100.

* static block –

- If we need to perform certain task before the actual execution, we can use Static block
- It is a block created inside a class block

& it is prefixed with static modifier is known as static block.

* Syntax. — static {
// task
}

- Class StaticInit
{
static
{
S.o.p. ("Hii from static");
}
P S V M (S[] args)
{
s.o.p. ("Hii from main");
}
}

o/p. → Hii from static
Hii from main.

- Note. — Static block does not have any name
- Therefore, a programmer cannot call a static block explicitely for execution.
- Static block does not have any implicit return type
- Static block does not accept any input data (no formal argument)

```
Class StaticInit
{
    Static
    {
        S.o.p.("static block");
    }
    P s v m (S[]args)
    {
        s.o.p.("main()");
        m1();
    }
    static
    {
        S.o.p. ("Static block 2");
    }
    P s v m1()
    {
        S.o p. ("m1()");
    }
    Static
    {
        s.o.p. ("static block 3");
    }
}
```

o/p. → Static Block1
Static Block2
Static Block3
main()
m1()

```
class staticInit
{
    static
    {
        s.o.p. ("static block");
    }
    p s v m (S[]args)
    {
        s.o.p. ("main ()");
        m1();
        staticInit.m2();
    }
    static
    {
        s.o.p. ("static block 2");
    }
    p s v m1( )
    {
        s.o.p.("m1()");
    }
}
class staticInit1
{
    static
    {
```

O/p → 1. static variable
2. Static variable

* Scope of a static variable.—

```
//file 1
class Example1
{
    static String a = "Raush";
    p.s.v.m(S[]args)
    {
        S.o.p("main()");
        S.o.p.(a);
    }
    p.s.v m1()
    {
        S.o.p("Eg m1()");
        S.o.p(a);
        S.o.p.(Example1.a);
    }
}
```

O/p → 4.

Multiple classes. –

It is a design of having a more than one class in a single file or source drive. is known as multiple classes.

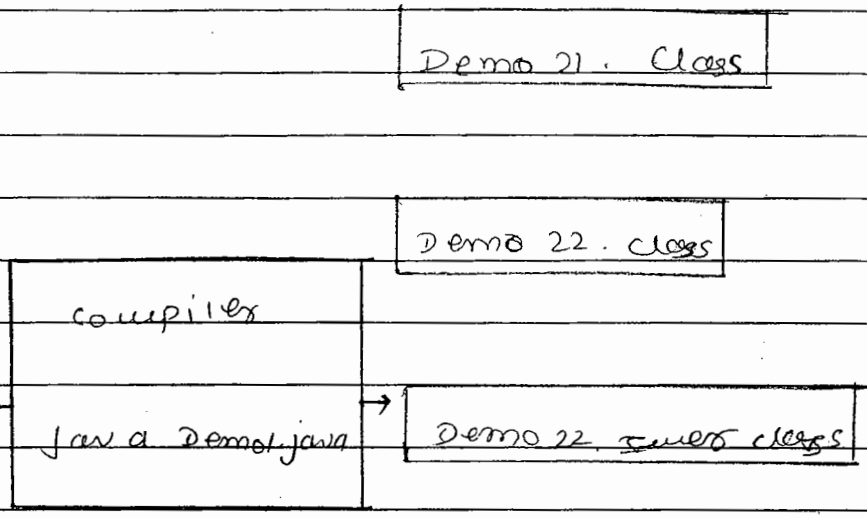
ex. –

File : (Demo1.java)

```
class Demo21
{
}
class Demo22
{
    class Innerclass
    {
    }
}
class Demo23
{
}
```

Note. – We can have more than one class in file and other compilation ~~n no. of~~ / byte codes for every different class / even for / inner class ~~n no. o~~ other compilation of that source file compiler will generate n no. of byte codes for every different class even for inner class.

```
class Demo21
{
}
class Demo22
{
  class innerclass
  {
  }
}
class Demo23
{
}
```



- In these case of multiple classes, file name can be anything but at time of execution we need to use respective class name in which main method is present is for the execution.

Public class —

- A class which is prefixed with public access modifier is known as a public class.
- we cannot have more than one public class inside a single java file.
- whenever we declare a class as public a file name should be same as class name (public class) or else we will get a CTE.

ex.—

file (Demo21.java)

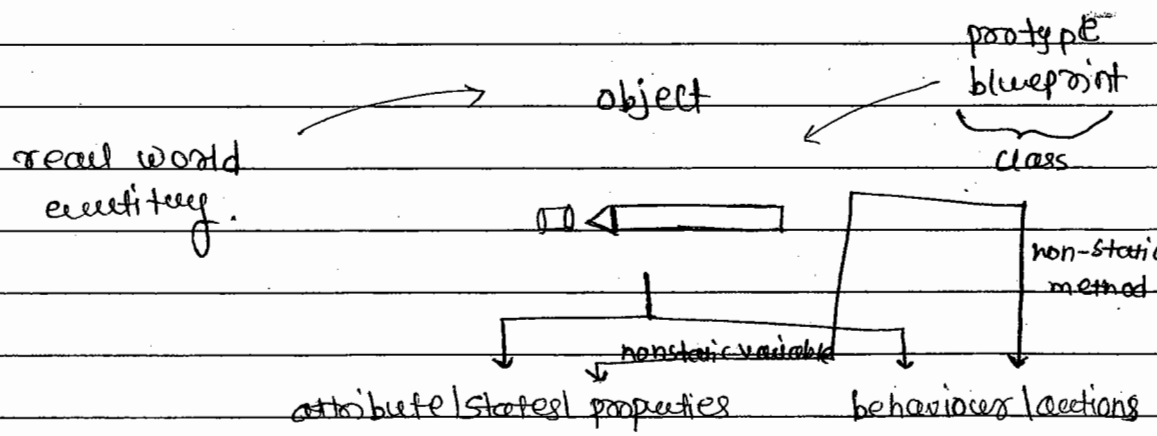
```
public class Demo21
{
}
class Demo22
{
}
class Demo23
{
}
```

21/10/23

Object —

An Object is a real world entitey that has a state & a specific behaviour.
Object is ~~being~~ ~~is~~ ~~are~~ been <u>created</u> for a specific purpose. in the <u>real world</u> an object can be <u>Imagined exactly</u> as it is (Pen, book, bag, marker, etc.)

Imp = Object is a block of memory which is created at the run time and It is stored inside the heap memory



use to define the object → color material shape size height weight ... write weapon tool

↑ tell us how to use the object.

Tree

Shape	birds use it as a shelte
color	use its fruits, ~~leav~~
type	leafs, ~~or~~ ~~w~~ wood.
height	

Object has two parts. —

1) States | attributes | properties. —

2) Behiover | Actions. —

- Attributes are used to define the object and behaviour tells us how to use it
- An Object is not just a product of a class, It is also contains the data members (variable) and member function (method) of that class

```
class cellphone
{
    String brand;
    String type;
    double price;
    double screensize;
    String model;
    int camera;
    int ram;
    int rom;

    p s v display phone()
    {
        S.o.pln. ("cellphone Info");
        s.o.pln. ("Brand : " + brand);
        s.o.pln. ("model : " + model);
        s.o.pln. ("price : " + price + " rs.");
        s.o.pln. ("screen size : " + screensize + "
                    inches");
        s.o.pln. ("camera resolution : " +
                    camera + " px");
        s.o.pln. ("Ram : " + ram + "gb");
        s.o.pln. ("memory : " + rom + "gb");
        s.o.pln.();
    }
}
class cellphoneDriver
{
    p s v m (s[]a)
    {
```

```
cellphone a = new cellphone();
a.brand = "Apple";
a.type = "smartphone";
a.price = 150000;
a.screensize = 6.2;
a.model = "iphone 14 pro max";
a.camera = 200;
a.ram = 12;
a.rom = 1024;
a.displayphone();
}
}
```

o/p. → cellphoneinfo
Brand : Apple.
type : Smartphone
Price : 150000
screensize : 6.2.
mode : iphone 14 promax
camera : 200
ram : 12
rom : 1024

* Note. —
- In Real world before creating an object blueprint is created.
- The blue print provides all the specification of that particular object.
- In java before creating an object a blue print must be created with the help of the class.

Ex.— Before manufacturing a car a blue-print of a car is created.
- Before constructing a house a prototype of it is created.

* Class
A class is not a real world entity, it is just a template / Blue print / prototype from which objects are created.
Class ~~is a group~~ has a group of different data types & methods.
a class in java can contain a data members, methods, constructor, nested class, blocks, etc.

* How to create an object in java. —

1) Using new keyword.

2) Using newInstance();

3) Using clone method();

4) Using deserialization.

5) Using newInstance Method of the Constructor class

new
It is a keyword & also an a operator which is used to create an object in java.

Ex/ syntex. —

class name = variable name = new class constructor call

class name → datatype

variable name → contain which holds the ref of the object

new → responsible to create object at the runtime

constructor call → used to initialized the non-static member of that class.

Ex. —

* Assignment. —

1) Train. —

```
class Train
{
        int double weight ;
        double int loadingCapacity;
                int NoofWheels;
                String type ;
                double length;

        Public void displaytrain()
        {
                s.o.pln.(" Train Info");
                s.o.pln.(" Weight:" + weight + " kg");
                s.o.pln.(" Loading Capacity of Train : +
                        loading Capacity);
```

```
s.o.pln.("No of wheels :"+ # no.of wheels);
s.o.pln.(" Type of a Train :"+ type);
s.o.pln.(" length of a Train :"+ length);
}
}
Class Train2.
{
    p s v m (S[]a)
    {
    Train a = new Train();
      a. weight = 229867.59 kg;
      a.loading capacity = 982798.33;
      a. Noofwheels = 58;
      a. type = Metro;
      a. length = 1923.22
      a. displayTrain();
} }
```

o/p. ⟶ Train Info

Weight : 229867.59kg.

loading capacity = 982798.33.

No of wheels = 58.

Type = Metro.

length = 1923.22

2) Factory. —

```
class Factory
{
        String type;
        double area;
        double profitlyear;
        int NoOfEmployee;
        int NoOfMachines;

        public void displayFactory()
        {
            s.o.pln.("Factory Info");
            s.o.pln.("Type :"+ type);
            s.o.pln.("Area :"+ area);
            s.o.pln.("Profitl Year :"+ profitlyear);
            s.o.pln.("No of Employee :"+NoOfEmploye
            s.o.pln.("No. of Machines :"+NoOfMachines
        }
}
Class Factory2
{
        P s v m(S[]a)
        {
            Factory a = new Factory();
            a.type = ClothFactory;
            a.area = 59.2294;
            a.profitlyear = 2983589g;
            a.NoOfEmployee = 1029;
            a.NoOfMachines = 359;
            a.displayFactory();
```

22/1/D 123.

}

}

o/p. → Factory Info
Type : cloth factory.
area : 59.2294
profit/year : 2943989
No. of Employee : 1029
No. of Machines : 359

Non-Static Memembers —
- A Non-Static members — are the members of the class which does not contain static keyword or a modifier.
- such as Non-static variable, non-static method & non-static Initializers.
- ~~Non~~

Non primitive data types
are multivalue
- data & their variable stores the reference of that object.
- Non primitive data type are also called as Reference type. Because they to refer to an object.
- Every class in java is a non primitive data type.

Ex — class Car.
class Marker.

- In these Example class is a non-primitive data type (key word). and car, Marker, etc are the non-primitive variables.
- with the help of that non-primitive data type we can create the reference variable which can store the address / Reference of an object which is created & stored in the heap area

- Ex. — class class name

keyword

~~variable name~~

non-primitive ~~data~~ type.

* Non-static variable. —

- A variable declared Inside the class block without prefixed ~~+~~ with the static modifier is known as a non-static variable
- when we create an object Inside that object a non-static variable gets memory and they are assighned with default value if no Initialization happens
- A non-static variable are present Inside the object which is stored in heap memory.
- Every time when we create a new object a non Static variable is created in that object.

Ex —

```
class Example2
{
    String a = "Ramesh";
    p s v m (S[]a)
    {
        Example2 a = new Example2();
        s.o.pln. (a.a);
    }
    Public void m1()
    {
        s.o.pln. (a);
        s.o.pln. ("Hello");
    }
}
```

o/p → Ramesh
Hello.

Note. —

- Memory is allocated for these variable whenever an object is created.
- Memory is allocated multiple times when multiple new object are created.
- Non static variables are also known as Instance variables Because they get memory while instance of a class is created.
- A non-static variable same as a regular variable (local variable).

- A static method or a block cannot use a non-static variable directly until and unless an object is created.
- We can access non-static variable using the reference of an object.

1) Ex. —

```
class Example2
{
    p s v m (s[]a)
    {
        Example3 ab = new Example3
        ab.a();
        s.o.p.(ab.a);
    }
}
class Example3
{
    string a = "Ramesh";
}
```

2) Ex. —

```
class Example2
{
    string name;
    string NativePalce;
    int yop;

    public void displayInfo()
    {
```

```
        S.o.pln.("Name :" + name);
        s.o.pln.("Native Place :" + nativeplace);
        s.o.pln.("YOP :" + yop);
    }
}
class Example3
{
        p s v m ~~SEO~~ (S[]args)
        {
            Example2 a = new Example2();
            a.Name = mansi;
            a.NativePlace = Malegaon;
            a.YOP = 2023;
            a.displayExample2();
        }
}
```

O/P. →

Non Static methods.—

methods which does not contain Static modi in its declaration is called as ~~static~~ Non stati methods.

like non-static variable non-static methods also needs to be called using object reference.

without an Instance of a class we cannot access non-static method inside a static contex

If we have a global and a local variabl with same name we can use this keyword t access the global variable.

A non static method declaration is stored insid the method area. A non-static method block is also known as non static context.

ex.—

```
class Example2
{
    p s v m (S[]a)
    {   Example2  a = new Example2();
        a.m1();
    }
    public void m1()
    {
        s.o.pln.("m1() non-static from Example2");
    }
}
```

O/p. → m1() non-static from Example2.

- 2 classes in same file. —

2)
```
class Example2
{
    p s v m (s[]a) {
    Example3 a = new Example3();
            a.m1();
            a.m2();
} }
class class Example3
{
    public void m1()
    {
        s.o.pln. ("m1() non-static from Example3");
    }
    public void m2()
    {
        s.o.pln. ("m2() non-static from Example3");
    }
}
```

- 2 classes with 2 files. —

3) i)
```
class Example2
{
    p s v m (s[]a) {
    Example2 a = new Example4();
        a.m1();
        a.m2();
}
}
```

ii)
```
class Example4
{
    public void m1()
    {
        s.o.pln. ("m1() non-static from Example4");
```

24/11/23

```
}
public void m2()
{
    s.o.pln.("m2() non-static from Example 4");
}
}
```

O/P. → m1() nonstatic from Example4.
m2() non-static from Example4.

Non-Static Initializers.—

- It gets executed during the loading process of an object.
- Non-static initializers are of two types

1) Non static declaration & initialization
2) Non static block

1) Non static Declaration & Initialization.—

Syntax :- Data type variable name = literal / Expression / variable

Ex. - int a = 10;
string b = "Hello";

2) Non-Static block.—

A block which is created in a class block without prefixing with static modifier is known as non-static block.

- Non static block does not have any name
- So programmer cannot called it explicitely
- It does not have ~~o~~ any return type
- & It it does not accept any kind of Input data.
- non static block will be executed at the time of object loading.
- It will get executed every time when we create a new object.

1) Ex. —

```
class Examples
{
    static
    {
      s.o.pln.("Static block 1");
    }
    P.s.v.m. (S[] a)
    {
        Example6 a = new Example6();
    }
    {
        s.o.pln.("non-Static block 1");
    }
}
class Example6
{
    static
    {
        s.o.pln. ("bye");
```

```
    }
    {
        s.o.pln ("non-static block Hello");
    }
}
```

O/p: →

2) Class Example7

```
{
    p s v m (S[]a)
    {
        S.o.pln.(Example8.a);
    }
}
```

Class Example8

```
{
    static string a = m1();
    static
    {
        s.o.pln ("Bypee");
    }
    p static string m1()
    {
        return "Ramesh";
    }
```

O/p. →

3)
```
class Example
{
    static
    {
        s.o.pln(a);
        a = "Ramesh";
        s.o.pln(a);
    }
    static String a = "suresh";
    p.s.v.m.(s[]a)
    {
        s.o.pln(a);
    }
}
```

o/p. →

4)
```
class Example9
{
    static String a = "suresh";
    static
    {
        s.o.pln(a);
        a = "Ramesh";
        s.o.pln(a);
    }
    p.s.v.m.(s[]a)
    {
        s.o.pln.(a);
    }
```

}

o/p. →

5) Class Example10

{ static {

s.o.pln. ("Hello");

int op = m1 (10,20);

s.o.pln. ("addition is :" + op);

System. exit (0);

}

P.s. int m1 (int a, int b)

{

return a+b;

}

}

o/p. →

6) Class Example11

{

Static

{

s.o.pln. ("static block 1");

} p.s.v.m. (S[]a){

Example12 a = new Example ();

a. m1();

} }

class Example12

{

static {

s.o.pln. ("Hi;!");

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

}

Public void m1()

{

s.o. pln. ("Bye");

} }

O/p. →

Starts the cons. chaining.

Constructor —

- A constructor is used in a creation of an Object.
- Typically it perform Operation required to Initiallize the member (Non Static Variables) of class before the methods and feilds are Invoked.
- A constructor is a block of code similar a method.
- It is called when an Instance of a class is created.
- At the time of calling a Constructor memo for the object is created.
- Every time when an Object is created using new ~~to~~ key word or operator at least one Constructor must be called.

Ex. —

```
class Example13
{
        // No argument Constructor
        Example3()
        {

        }
        p s v m (s[] a) ..
        {
            Example3 b = new Example3();
        }
}
```

Rules for creating a Constructor.—

1) Constructor Name must be same as class name.

2) A constructor must have no Explicit return type.

3) A Java constructor cannot be abstract, static, final, synchronized
It can be public, private, protected, default
(It depends on the type of the constructor)

Ex. —

```
class Demo
{
    Demo()
    {
        s.o.pln.("Constructor called")
    }
    p s v m(S[]a)
    {
        Demo a = new Demo();
        Demo b = new Demo();
        Demo c = new Demo();
        Demo d = new Demo();
    }
}
```

o/p. → Constructor called
Constructor called
Constructor called

constructor called.

* purpose of a Constructor. —

- It is used to Initiallized all the non-stat members of a class Inside an object.
- It is mandatory to Provide a Constructor by th programmer or by the Compiler.

Constructors can be classified into — 4 types. —

1) default Constructor.
2) No argument Constructor.
3) Parameterized Constructor.
4) Copy Constructor.

* Default Constructor. —

- A Default Constructor in Java is created by the compiler itself when the programmer dosen't create any constructor.
- It cannot accept any kind of Input data

Ex. —

```
Class Defaultconstructor
{
    Defaultconstructor()
    {
    }
}
```

→ constructor provided by the compiler (Default constructor)

- To call variables & methods we use [illegible]
- To call const. we use super() & this()

Note. —

- A default constructor can have default & public access modifier
- It cannot be private or protected because a top level class cannot be private or protected.

Ex. —

```
class ~~par~~ Parent
{
    Parent()
    {
        super();
        s.o.pln. ("Parent cons");
    }
}

class child extends Parent
{
    child()
    {
        super();
        ~~pl.~~ s.o.pln. ("Child cons");
    }
    P.s.v.m.(s[]a)
    {
        child a = new child();
    }
}
```

used to achieve a relationship betn two classes / creating a parent child class Relation.

used to invoke with parent class constructor.

o/p → Parent cons

Note. —

- A default constructors 1st statement will be always super call statement.

Difference between a constructor and a method. —

Constructor	Methods
1) A constructor is used to Initialize the states of the object	1) A method is used to expose the behaviour of a object
2) Constructor name must be same as a class name.	2) method name may or may be same as class name
3) constructor does not have any return type.	3) method must have a return type.
4) constructor may invoked Implicetly / Explicitely	4) methods are invoked explici
5) construction recursion is not possible.	5) method recursion is possib
6) Compiler provides a default constructor if we fails to add any constructor in a class.	6) If we fails to create any method compiler is ~~going~~ not going to provide it for us.

7)	super(): statement & this(): called statement can be used only from constructor.	7) we cannot use super called statement & this inside the method block.
8)	constructor cannot be abstract	8) methods can be abstract (no implientation).

* No Argument Constructor. —

- A constructor without any formal Argument is kno as a No Argument Constructor.

Ex. —

```
class Demo
{
    Demo()
    {
        super();
        S.o.pln.("No argument Constructor");
    }
    p.s.v.m.(S[]a)
    {
        ~~Example~~ Demo a = new Demo();
    }
}
```

o/p. → No argument constructor.

* Note. —

- A No argument Constructor have public, private, protected & Default Access modifier.
- We can have <u>super</u> call statement or <u>this</u> call statement as a first statement inside the constructor body.

Ex. —

```
class Demo
{
    public Demo()
```

```
{
    s.o.pln. (" public No argu. const");
}
protected Demo()
{
    s.o.pln. (" protected No argu. const");
}
Demo()
{
    S.o.pln. (" default No argu. const");
}
private Demo()
{
    s.o. pln. (" private No argu. const");
}
p s v m (S[]a)
{
    Demo a = new Demo();
}
}
```

o/p → Error, CTE(same const. name is used) instead of these we can use diff. formal arguments).

Parameterized Constructor. —

A constructor with formal Argument is known a Parameterized constructor.

Ex. —

```
Class student
{
    String name;
    String nativePlace;
    int age;
    String gender;
    double contact;

    student (String name, String nativePlace, int age,
            String gender, double contact)
    {
        this. name = name;
        this. ~~str~~ nativeplace = nativePlace;
        this. age = age;
        this. String = String;
        this. contact = contact;
    }
}
class studentDriver
{
    P. S. V. m. (S[]a)
    {
        student a = new student ("Mansi", "Malegaon
            23, "Female", 9021207689.)
```

called by [illegible] only

```
    }
}
}
EX.—
class Aeroplane
{
    string company;
    string type;
    int noEngine;
    int seatingCap;
    double travellingRange;

    Aeroplane (string company, String type, int seating-
    Cap, double travellingRange)
    {
        this.company = company;
        this.type = type;
        this.noEngine = noEngine;
        this.seatingCap = seatingcap;
        this.travellingRange = travellingRange;
    }
    Public void display()
    {
        s.o.pln.(this.company);
        s.o.pln.(this.type);
        s.o.pln.(this.noEngine);

    }
```

```
class AeroplaneDriver
{
    p.s.v.m.(S.[]a)
    {
        Aeroplane a = new Aeroplanett("

    }
}
```

o/p. →

Constructor Overloading —

A class having more than one constructor is known as constructor overloading.

Ex —

```
class Example17
{
    Example17()
    {
        S.O.Pln.("No argu. cons");
    }
```

```
Example17(int a)
{
    s.o.pln. (" 1 argu int const");
}
Example17(byte a)
{
    s.o.pln. (" 1 argu byte constr");
}
Example17 (int a, int b)
{
    s.o.pln. (" 2 argu const");
}
p.s.v.m (S[]a)
{
    Example17 a = new Example17();
    Example17 b = new Example(10);
    Example17 c = new Example(20);
    Example17 c = new Example(10, 20);
}
}
```

O/P. →

Ex. →

```
Class Example17
{
    Example17 (int a)
    {
        s.o.pln. (" 1 argu int cons");
    }
```

```
Example17(byte a)
{
    s.o.pln.("1 argu byte constr");
}
p s. v. m. (S[]a)
{
    byte dt = 10;
    Example17 b = new Example17(dt);
    Example17 c = new Example17(20);
}
}
```

O/p. →

15 ~~meth~~
16 cons. overloading

17 → static method overloading

methodoverloading

static
non-static

16 - cons. over
17 - method over

parent
child

17 →

Note.—
Constructor overloading in an type of compile ti
Polymorphism.

*** method Overloading —**
It is a design of having a more than one meth
in a class with same name & different formal
argument is known as method overloading.

*** Note —**
method overloading is also a type of compile time
polymorphism

Ex. —

1)
```
Class Demo
{
    P.s.v.m.str (S[]a){
      m1();
      m1(10);
      m1(20,30);
    }
    P.s.v.m1()
    {
      s.o.pln. ("m1() no argu");
    }
    P.s.v.m1(int a){
      s.o.pln. ("m1() 1 argu");
    }
    P.s.v.m1(int a, int b){
      s.o.pln. ("m1() 2 argu");
    }
}
```

O/P. →

Ex.

2)
```
class Demo{
      P.s.v.m.(s.[]a){
      Demo a = new Demo();
      a.m1();
      a.m1(10);
      a.m1(10 20, 30);
  } public void m1() {
      s.o.pln. ("m1() no argu");
  }
```

```
Public void m1(int a, int b)
{
    s.o.pln. ("m1() 2 argu");
}   }
```

o/p. →

method chaining —

- It is a design or way of calling one meth to another.

OR

- method chaining is the chain of methods beer called one after another.
- It is same as constructor chaining but th Only difference is of method and constructor

Ex. — parent child.

```
class methodchain
{
    P.s.v.m.(s[]a)
    {
        m1();
    }
    P.s.v.m1()
    {
        s.o.pln. ("m1()");
        m2(10);
    } p.s.v.m2(int a)
    {
```

```
    p. s. v. m2(int a)
{   s.o.pln.("m2(int a)"); }
    m3(); }
    p. s. v. m3 (char ch)
    {
        s. o. pln. ("m3(char ch)"); }
    }
```

* Constructor chaining :—

✓ It is a design or way of calling one constructor inside another constructor is known as constructor chaining.

- We can achieve constructor chaining using this() & super().

Ex-
```
class Parent
{
    Parent ()
    {
        super(); // object class constructor.
        s.o.pln.("Parent const");
    }
}
class child extends parent
{
    child ()
    {
        super();
        s.o.pln.("from child cons");
    }
    p.s.v.m.(S[]a)
    {
```

26/11/23.

```
        child a = new child();
    }
}
```

O/p. → parent const.
child const.

2)
```
class Demo
{
    Demo()
    {
        this(10);
        s.o.pln. (" No argu");
    }
    Demo (int a)
    {
        this (10, 20);
        s.o.pln. (" 1 argu const");
    }
    Demo (int a, int b)
    {
        s.o.pln. (" 2 argu const");
    }
}
class Demo2 {
    P s. v. m. (s. [] a)
    {
        Demo a = new Demo();
    }
}
```

O/P. → No argu
1 argu cons.
2 argu cons

1)
```
class parent
{
    Parent()
    {
        super();
        s.o.pln. ("no argu Parent const");
    }
}
class child extends Parent
{
    child()
    {
        this (10);
        s.o.pln. ("no argu child const");
    }
    child (int a)
    {
        super();
        s.o.pln. ("1 argu child const");
    }
    p.s.v.m. (s[]a)
    {
        child a = new child();
    }
}
```

o/p. → No argu parent const.

2)
```
class Parent
{
    Parent()
    {
        s.o.pln. ("no argu Parent const");
    }
    parent (int a)
    {
        this();
        s.o.pln. ("1 argu parent const");
    }
}
class child extends Parent
{
    child()
    {
        this (10);
        s.o.pln.("no argu child const");
    }
    child (int a)
    {
        super (10);
        s.o.pln.("1 argu child cons");
    }
    p.s.v.m.(string [] args)
    {
```

```
        child a = new child();
    }
}
```

o/p. →

3) class child extends parent

```
{
    String name = "child";
    p.s.v. m(S[]a)
    {
        child a = new child();
        s.o.pln.(a);
        s.o.pln.(this.s);
        a.m1();
    }
    void m1()
    {
        string name = "Hello";
    }
}
```

o/p. →

4) class parent

```
{
        String name = "Parent";
}
class child extends parent
```

```
{
    string name = "child";
    P.S.V.M.(S[] a)
    {
        child a = new child();
        a.m1();
    }
    void m1()
    {
        string name = "Hello";
        S.o.pln (super.name);
    }
}
```

o/p. →

5)

```
class child extends parent
{
    string name = "child";
    P.S.V.M.(S[] a)
    {
        child a = new child();
        a.m1();
    }
    void m1()
    {
        string name = "Hello";
        S.o.pln.(this.name);
        this.m2();
```

```
}
void m2()
{
        s.o.pln.("M2()");
  }
}
```

O/p →

6)

```
class parent
{
        String a = "Parent";
        Parent()
        {
            s.o.pln.("Parent cons");
        }
        void m1()
        {
            s.o.pln.("m1() parent");
        }
        int addition (int a, int b)
        {
            return a+b;
        }
    }
class child extends parent
{
        p s vm (s [] a)
        {
```

```
        child a = new child();
        a. interaction();
    }

    p. v. interaction()
    {
        s.o. pln. (super. a);
        super. m1();
        int op = super. addition(10, 20);
        s. o. pln. (op);
    }
}
```

o/p. →

7)

```
class parent
{
    string a = "parent";
    parent()
    {
        s.o. pln. ("parent const");
    }
    void m1()
    {
        s.o. pln ("m1() parent");
    }
    int addition(int a, int b)
    {
        return a+b;
    }
}
```

```
class child extends parent
{
    p.s.v.m (S[]a)
    {
        child a = new child();
        s.o.println(a.a);
        a.m1();
        s.o.println(a.addition(20,30));
    }
}
```

O/P. → parent
m1() parent
50.

```
long [][][] a = new long [3][][];
a[0] = new long [1][4];
a[1] = new long [2][1];
a[2] = new long [3][];
a[2][0] = new long[1];
a[2][1] = new long [4];
a[2][2] = new long [1];
int a [][][] = new int [2][][];
a[0] = new long [2][5];
a[0][0] = new long [1];
a[0][1] = new long [3];
a[1] = new long [1][4];
```

int int

~~double~~ [ ][ ][ ] a = new ~~double~~ [2][ ][ ];

a[0] = new long [2][ ];

a[0][0] = new long [1];

a[0][1] = new long [3];

a[1] = new long [1][4];

a[0][0][0] = 10

a[0][1][0] = 20

a[0] 20

40

50

60

70

80

this. —

- this is a reserved word in Java.
- this keyword is a non-static reference variable which holds the reference of current object which is under used.
- the this keyword refers to the current object in a method or a constructor.
- The most common use of this keyword is to eliminate the confusion between the class attribute and parameter (local variable) with same name.

Ex.—

```
class child
{
    class child extends parent
    {
        String name =" child";
        p.s.v.m (S[]args)
        {
            child a = new child();
            a.m1();
        }
        void m1() {
                String name =" Hello";
                s.o.pln.( this.name);
                this.m2();
            }
            void m2 (){
                s.o.p. ("m2 ()");
            }
    }
```

Super. —

- Super is a reserve keyword in Java which is used to invoke the fields (constructor & methods). and variables of parent class.
- Super keyword is used to represent an instance of a parent class which is created implicitely for each object of a child class.
- Ex. —

```
Ex.— class parent
{
    string a = "parent";
    Parent()
    {
        s.o.pln. ("Parent cons");
    }
    void m1()
    {
        S.o.pln. ("m1() parent");
    }
    int addition (int a, int b)
    {
        return a+b;
    }
}
class child extends parent
{
    P.s.v.m.(S[]a)
    {
        child a = new child();
        a.interaction();
```

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

O/P. →

* Difference between this called Statement & super call statement.

	this	super
1)	this is used to invoke the constructor of the same class	1) Super is used to invoke the constructor of the parent class.
2)	It should be the first statement in the constructor body	2) It should be the first statement in the constructor body.
3)	If No explicitely called the compiler automatically inserts a called to the no argument ~~compile~~ constructor of the same class (if available.)	3) If not explicitly called the compiler automatically inserts a called to the No argument constructor of the parent class (if available).
4)	It can be explicitelly called to invoke the specific constructor	4) It can be explicitelly called to invoke the

of the same class with argument.	specific constructor of the parent class with argument.

⇒ Note. — ⇒

- this called statement & super called statement must be used only inside a constructor.
- It should be the first statement.
- recursive call to a constructor is not possible (constructor calling itself).
- If we are not providing this() or super() then in these case compiler will provide the super()
- If a class has n no. of constructor we should use 'n-1' constructor calling statement or else recursion will happen.

~~Object 0~~

* Object Oriented Programming Concepts.—

- Object oriented programming is a core of Java programming language which is used for designing a program using classes and Objects.
- Oop's can be characterized as data controlling for accessing the code.
- The basic concept ~~for~~ of Oop's is to create Object, reuse them throughout the program and manipulate (modification) this Object to get an expected result.

* Various Oop's Concept in Java.—

1) Class
2) Object
3) Encapsulation
4) Inheritance.
5) Polymorphism.
6) Abstraction

Apart from these we use some various concept in Oop's like

1) Aggregation
2) Composition
3) Interface
4) Assosiation / Relationship.

* Advantages of Oop's ~~en~~ :—

* Reusability — When we say Reusability it me write once and ~~Run~~ use it multiple times that is Reusing some facilities (code) Rather than crea it again and again, which can be achieved by using Oop's concept
- we can use it n no. of times whenever & where it is required.

* Data Redundancy — It is one of the main advanta in Oop's this the condition which is created at th data stored — where same peice of code is held at more than one place.
- If we want to use some similar functionalities in multiple classes we can write a common class definati which will be shared among all that multiple classes.

* Easy code maintanance — It is easy to modify or maintain an existing code as new objects which will be created with a small difference from the existin one.

* Security — data hiding & abstraction are used to filter out limited exposure, which means we are providing only necessary data to view because we hav to maintain the security

* Easy Troubleshooting — Everything in OOP's is an Object. So we can use these object to interchange or

to remove the functionalities to meet ~~use~~ the users need

- basically trouble shooting is to find the problems and to set a process to resolve it.

* Principal of Oop's.

1) Inheritance.
2) Abstraction.
3) Polymorphism.
4) Encapsulation.

* Encapsulation. -

- It is a process of binding ~~and~~ or wrapping the data members (variables / Attributes) and the member function (methods) together ~~as a sing~~ representing as a single unit is known as Encapsulation.

OR

It is a process of binding or wrapping an attributes & behaviors of an object is known as Encapsulation.

Ex. — 1) Capsule / Tablate. 3) laptop
~~Data hidd~~ 2) car Engine. 4) Humanbodey.

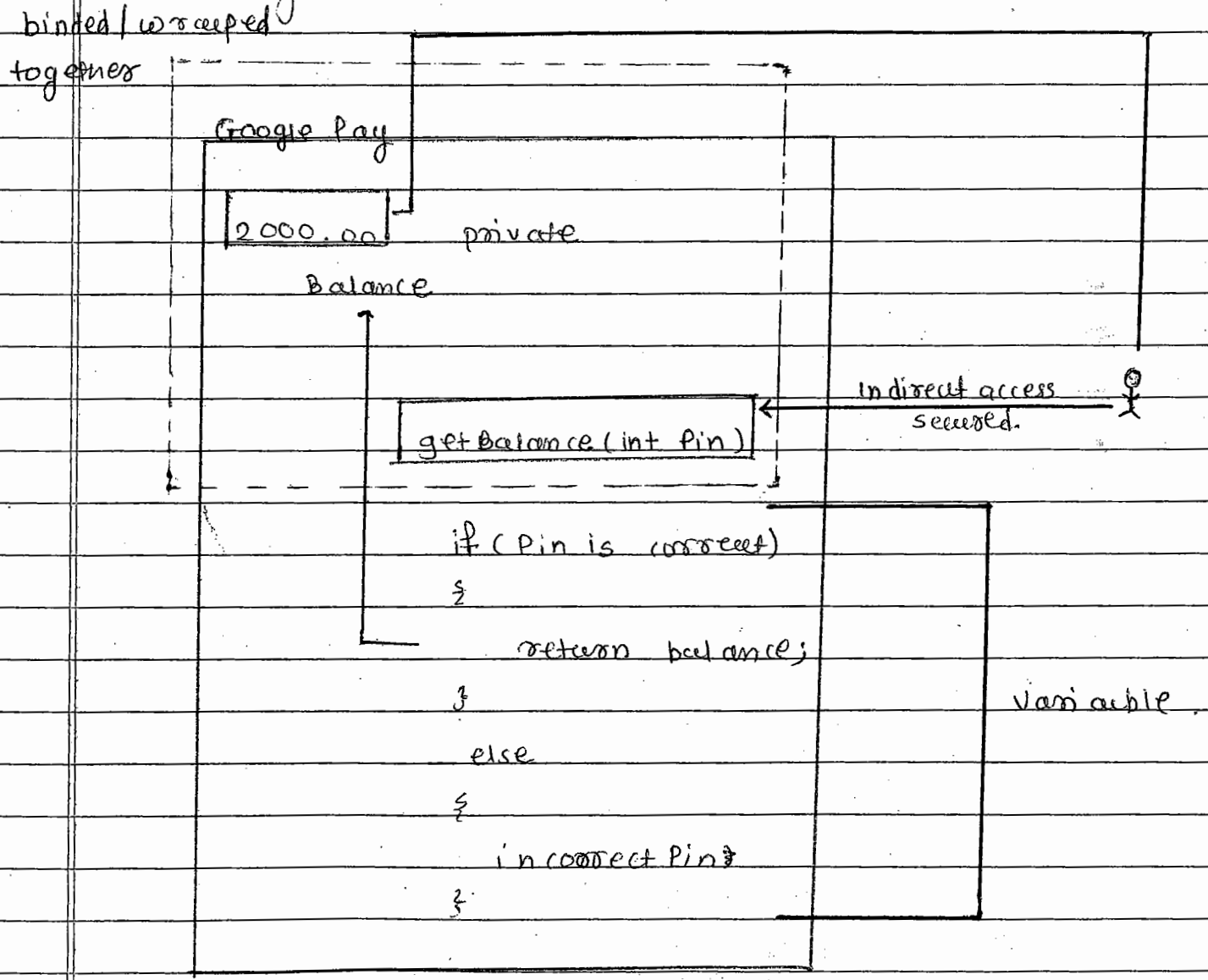
* Data hidding —

The process of restring the direct access to the data members (variables) & providing a secured & Indirect access to the data members with the help of member function like (Getters & setters)

of that class is known as Data hidding.

* Steps to acheive data hiding –

1) Declare the variable of a class as private.
2) Provide Public Getter & setter method to view only Read Only.



Ex — class mobile
{
String name;

```
private ~~pin~~ int pin;

Mobile (string name, int pin)
{
        this.name = name;
        this.pin = pin;
        s.o.pln. ("mobile obj created sucessfully");
}

public int getpin()
{
        return this.pin;
}
}
class mobileDriver
{
        p.s.v.m.(s[]a)
        {
                mobile a = new mobile ("Mani", 1234);
                s.o.pln. (a.name);
                s.o.pln. (a.getPin());
        }
}
```

o/p → mobile obj created successfully.

for the security purpose we should not give the direct access to the properties of a class but the user should be able to read or modified the data members with the helper methods (getters & setters)

* Why we need Encapsulation. —

1) we can acchieved security by hidding the data with the help of private modifier.
2) It help us to keep related field & methods together which makes our code cleaner and easy to read.

* Private modifier —

- If a member is Declared with private modifier it can be only accessible with in the same class
- we can use private modifier for a variable, method, constructor, inner class.
- Private access modifier is not allowed for a top level class & an Interface.

27/11/23.

* Bank Application —

```
class AccountHolder
{
    static int uniqueNumber = 342116;
    string name;
    string address;
    Private long contactNumber;
    private long adharNumber;
    Private string pancardNumber;
    Private double balance;
    Private int Pin;
    int accountNumber;

    AccountHolder( String name, String address, long
contactNumber, long adharNumber, String pancardNumber,
double balance, int Pin);
    {
        this.name = name;
        this.address = address;
        this.contactNumber = contactNumber;
        this.adharNumber = adharNumber;
        this.pancardNumber = pancardNumber;
        this.balance = balance;
        this.Pin = Pin;
        this.accountNumber = uniqueNumber++;
        S.o.pln.("Account created Successfully");
    }

    public string getName()
    {
```

```
        return this.name;
    }
    public void setName (string newName)
    {
        this.name = newName;
    }
    public string getAddress()
    {
        return this.address;
    }
    public void setAddress (string newAddress)
    {
        this.address = newAddress;
    }
    public long getcontact()
    {
        return this.contactNumber;
    }
    public void setcontact (long newcontact)
    {
        this.contactNumber = newcontact;
    }
    public long getAdhar()
    {
        return this.adharNumber;
    }
    public string getPancard()
    {
        return this.pancardNuber;
    }
```

```
public double getBalance()
{
    return this.Bbalance;
}
public void setBalance (double newBalance)
{
    this.balance = newBalance;
}
public int getpin()
{
    return this.pin;
}
public void setpin (int newPin)
{
    this.pin = newPin;
}
}
```

e 2. —

```
import java.util.Scanner;
class Bank
{
    static Scanner sc = new Scanner (System.in);
    Static AccountHolder obj;
    Static
    {
        S.o.pln. ("""" Welcome to laxmi-chit Fund""""");
    }
    Public static void main (String[] args)
    {
        for (;;)
        {
```

```
s.o.pln. (" choose an option : 1. Create Account
2. Exisisting Account - Holder");
s.o.pln. ("Enter Number :");
int option = sc.nextInt();
switch (option)
{
    case 1 :
}
scanner Scanner = new Scanner(system.in);
s.o.pln. ("***** Account creation *****");
s.o.pln. (" Please enter the details");
Scanner.reset();
S.o.p. (" Enter your name :");
String name = scanner.nextline();
scanner.reset();
s.o.p. ("Address :");
string address = scanner.nextline();
s.o.pt p. ("contact Number :");
Scanner.reset();
long contactNumber = scanner.nextLong();
s.o.p. (" Adhar number :");
Long adharNumber = scanner.nextlong();
s.o.p. (" Pan card Number :");
String pancard Number = scanner.next();
s.o.p.("opening deposit amount :");
scanner.reset();
double amount = scanner.nextDouble();
s.o.p ("Enter a new pin :");
int pin = scanner.nextInt();
obj = new AccountHolder (name, address,
```

```
contactNumber, adharNumber, panCardNumber, amount, pin)
break;
}
case 2:
{
    for( ; ;)
    {
        s.o.pln.("*** Features ***");
        s.o.pln.("1. Deposit. ,2. withdraw ,3. Account
Balance ,4. Account - Details. 5.                    6.Exit");
    s.o.pln.("Enter an option :");
    s.o.pln();
    int option2 = sc.nextInt();
    switch(option2)
    {
        case 1;
        {
            s.o.pln.("Amount Deposit Feature");
            s.o.p.("Enter the amount to deposit :");
            double depositAmount = sc.nextDouble();
            double newBalance = obj.getBalance() +
                                    depositAmount;
            obj.setBalance(newBalance);
            s.o.pln("Amount Deposited");
            break;
        }
        case 2:
        {
            s.o.pln.("Amount withdraw Feature");
            s.o.p("Enter the amount to withdraw :");
```

```
double withdrawAmount = sc.nextDouble();
s.o.p.("Enter your pin :");
int pin1 = sc.nextInt();
if (pin1 == obj.getpin())
{
    if (withdrawAmount < obj.getBalance())
    {
        s.o.pln.("Amount withdraw");
        double newBalance = obj.getBalance().withdrawAmount;
        obj.setBalance(newBalance);
    }
    else
    {
        s.o.pln.("Insufficient Funds");
    }
} else
{
    s.o.pln.("Incorrect pin try Again");
}
break;
}
case 3:
{
    s.o.pln.("Balance Feature");
    s.o.p.("Enter your pin :");
    int pin1 = sc.nextInt();
    if (obj.getPin() == pin1)
    {
        s.o.pln.("Account Balance is : "+obj.
```

```
getBaulance());
}
else
{
    s.o.pln.("Incorrect pin & try again");
}
break;
}
case 4:
{
    s.o.pln.("Account Details");
    s.o.pln.("Account Holder : "+obj.getName());
    s.o.pln.("Address : "+obj.getAddress());
    s.o.pln.("phone number : "+obj.getContact());
    s.o.pln.("Account Nuber : "+obj.acountNuber);
    break;
}
case 5:
{
    ~~s.e~~ system.exit(0);
}
default: {
            s.o.pln.("Enter an correct option");
            break,
        }
    }
    break;
    }
}
default : system.Out.println("Invalid Input");
```

```
break;
}
}
}}
```

o/p. →

Vaishnavi
28-11-23

28/11/23.

* Relationship. —

- Relationship in Java means different real connections between two or more classes. (object, Entities).
- It means that an instance of a class that has a reference to the instance of another class or the other instance of the same class
- In other words a relationship in which an Object of one class has a reference ~~n~~ of another class.

* Relationship is classified into 2 types. —

1) has - A - Relationship.
2) IS - A - Realationship.

1) has - A - Relationship. —

It is an Association between two or mor Objects such that they are dependent on each other is known as has - a - Relationship OR Association.

- When an Object of one class is connected as a Data member Inside another class it is called as has - A - Relationship.
- ~~I~~

It is classified into two types. —

1) composition. — (Tight bond).
2) Aggregation. — (Close bond).

Composition —

- It is an association between two Objects that is one cannot exist without another.
- They are like human and heart, car and engine, fish and water, AC and compressor, human and brain etc.

Ex.—

File 1 — class Car {

```
        string brand;
        string model;
        Engine engine;
        Car(String brand, String model, Engine engine){
        this.brand = brand;
        this.engine = engine; }
        Public void displayCar(){
        s.o.p. ("car Details");
        s.o.p. ("Brand :" + brand);
        s.o.p. ("Model : " + model); }
```

File 2. class Engine

```
{       int noOfPiston;
        double bhp;
        String engineType;
Engine(int noOfPiston, double bhp, string engine type) {
        this. noOfPiston = noOfPiston;
        this. bhp;
        this. engineType = engineType; }
        s.o.p. ("Engine Details");
        s.o.p. ("No of Piston :" + this. noOfPiston);
        s.o.p. ("Break Horse Power :" + Hbp);
        s.o.p. ("Type of Engine :" this. engineType); }
```

File 3 class CarDriver {

P.S.V.M. (SIJA)

```
Car a = new Car("Audi", "08", new Engine(4,340, V6));
    a.displayCar();
    a.engine.displayEngine(); } }
```

O/P. → car details

Brand : Audi

Model : 08

Engine Details

No of Piston : 4

Break hp : 3400

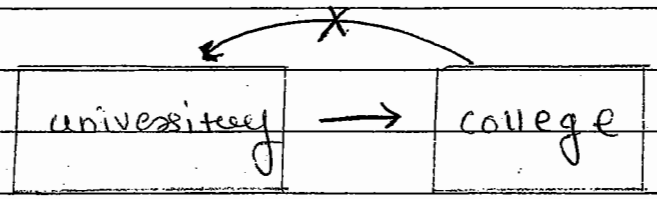
Engine Type : V6.

* Aggregation —

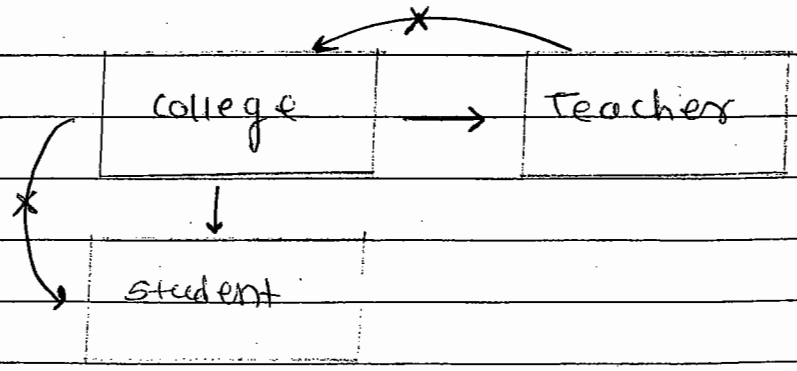
- It is a special form of has-A-Relationship where two objects are dependent on each other but both can survive Indivisually.
- It is a unidirectional Association that is one way relationship

Ex. — A university can have different colleges but Vice-Versa is not possible. It means a colleges can not have universities.

1)



2)



Composition. —

1) mother board & laptop
2) Maths & Number.
3) machine & Hardware.
4) plant & leaf
5) compressor & fridge.

Aggregation. —

1) Laptop & application.
2) seller & customer.
3) Flowerpot & flower
4) shoe & shoeless
5) juicer & fruits
6) Mirror & cupboard.

Ex. Aggregation. —

file 1. —

```
class Car1
{
        string brand;
        string model;

        Car1 ( string brand , string model)
        {
                this. brand = brand;
                this. model = model;
        }

        void display car()
```

```
}
    s.o.p ("Car Details:");
    s.o.p. ("Brand :" + this.brand);
    s.o.p. ("Model :" + this.model);
}
void createDriver()
{
    Driver1 a = new Driver1("Maui", 22, "female");
    a.display();
}
```

File 2

```
class Driver1
{
    string name;
    int age;
    string gender;

    Driver1(String name, int age, String gender)
    {
        this.name = name;
        this.age = age;
        this.gender = gender;
    }
    void displayDriver()
    {
        s.o.p. ("Driver Details");
        s.o.p. ("Name :" + this.name);
        s.o.p. ("Age :" + this.age);
        s.o.p. ("Gender :" + this.gender);
    }
}
```

file 3.

```
class DDriver
{
    p.s.v.m (S[]a)
    {
        car a = new car ("Honda", "city");
            a.display car();
            a.create Driver();
    }
}
```

O/p. → Car Details
Brand : Honda
model : city.

Driver Details
Name : Mansi
age : 22
gender : Female.

1/12/23.

* Library Application. —

File 1

```
class Library
{
    string name;
    double startTime;
    double endTime;
    double fees;
    string address;
    long contactNuber;
    Book book;
    Librarian librarian;
    shelf shelf;
    Student student;

    Library (string name, double startTime, double
endTime, double fees, string address, long contactNuber,
Book book)
{
    this.name = name;
    this.startTime = startTime;
    this.endTime = endTime;
    this.fees = fees;
    this.address = address;
    this.contactNuber = contactNuber;
    this.book = book;
}
public void displayLibrary()
{
    S.O.pln ("**** Library Details ****");
```

```
s.o.pln.("Name :" + this.name);
s.o.pln.("Address :" + this.address);
s.o.pln.("Fees :" + this.fees + "rs");
s.o.pln.("Timings :" + this.startTime + "am -" + this.endTime + " pm");
s.o.pln.("contact Number :" + this.contactNumber);
s.o.pln.();
}
public void addLibrarian()
{
    s.o.pln("Library Details");
    s.o.pln.("Name:" + this.name);
    s.o.pln.("Address :" + this.address);
    s.o.pln.("Fees :" + this.fees + "rs");
    s.o.pln.("Timings :" + this.startTime + "am -" + this.endTime + " pm");
    s.o.pln("contact Number :" + this.contactNumber);
    s.o.pln();
}
public void addLibrarian()
{
    String name = "Ramesh";
    String gender = "Male";
    Long contactNumber = 9089497491L;
    String address = "karvenagar";
    String shift = "Day";
    Librarian = new Librarian(name, gender, contactNumber, address, shift);
}
public void addself()
```

```
{
    String type = "Historic";
    int capacity = 150;
    shelf = new shelf (type, capacity);
}
public void addstudent()
{
    String name = "Suresh";
    String gender = "Male";
    long contactNumber = 9021456809L;
    String address = "kothrud";
    String department = "civil";

    student : new student(name, gender, contactNumber,
            address, department);
}
```

File 2 —

```
Class Book
{
    String name;
    String author;
    String type;
    String publication;
    String language;

    Book ( String name, String author, String type, String
    publication, String language)
    {
```

```
        this.name = name;
        this.author = author;
        this.type = type;
        this.publication = publication;
        this.language = language;
    }
    public void displayBook()
    {
        s.o.pln.("Book Details");
        s.o.pln.("Name : " + this.name);
        s.o.pln.("author : " + this.author);
        s.o.pln.("Type : " + this.type);
        s.o.pln.("publication : " + this.publication);
        s.o.pln.("Language : " + this.language);
        s.o.pln.();
    }
}
```

File 3. —

```
class student
{
    static int uniqueid = 1;
    String name;
    int id;
    String gender;
    long contactNumber;
    String address;
    String department;

    ~~String (str~~
    student(String name, String gender, long contactNumber,
```

```
String address, String department)
{
    this.name = name;
    this.id = uniqueId;
    this.gender = gender;
    this.contactNumber = contactNumber;
    this.address = address;
    this.department = department;
}
public void displayStudent()
{
    S.o.pln.("Student Details");
    S.o.pln.("Id : "this.id);
    S.o.pln.("Name :"+ this.name);
    S.o.pln.("Gender :"+ this.gender);
    S.o.pln.("Contact Number :"+ this.contactNumber);
    S.o.pln.("Address :"+this.address);
    S.o.pln.("department :"+ this.department);
    S.o.pln.();
}
}
```

File 4. –

```
class ~~Student~~ Librarian
{
    ~~static int uniqueId = 1;~~
    String name;
    String gender;
    Long contactNumber;
    String address;
    String shift;
```

```
Librarian (String name, String gender, long contact Number,
String address, String shift).
{
    this.name = name;
    this.gender = gender;
    this.contact Number = contact Number;
    this.address = address;
    this.shift = shift;
}
public void displayLibrarian()
{
    s.o.pln.("Librarian Details");
    S.o.pln.("Name :"+ this.name);
    s.o.pln.("Gender Number :"+ this.contactNumber);
    s.o.pln.("Address :" + this.address);
    s.o.pln.("shift :"+ this.shift);
    s.o.pln();
  }
}
File 5. —
class shelf
{
    String type;
    int capacity;
    shelf (String type, int capacity)
    {
        this.type = type;
        this.capacity = capacity;
    }
    public void displayShelf()
```

```
{
    s.o.pln.("shelf Details");
    s.o.pln.("Type of books :"+this.type);
    s.o.pln.("Capacity :"+this.capacity);
  }
}
```

File 6. —

```
Class LibraryDriver
{
    ~~pu~~ p.s.v.m(S[]a)
    {
        s.o.pln.("Welcome");
        Book book = new Book("~~Here~~ shriman","                ",
"Autobiography","Bhawarth publication","Marathi");
~~Book book = new Book(~~
Library library = new Library("ABCD",6,12,500,"Deccan-Pune",
9021007089L,book);
library.displayLibrary();
library.book.displayBook();
library.addLibrarian();
library.librarian.displayLibrarian();
library.addShelf();
library.shelf.displayShelf();
library.addStudent();
library.student.displayStudent();
  }
}
```

o/p. → Welcome

Name: ABCD

Address: Deccan-pune.

Fees: 500.0 rs.

Timing: 6.0 am – 12.0 pm

Contact Number: 9021207689

Book Details

Name: shriman

Author: Ranjit Desai.

Type: Autobiography

publication: Bharath publication.

Language: Marathi

Librarian Details

Name: Ramesh

Gender: Male

contact Number: 9021208899.

Address: karvegar

Shift: Day.

Shelf Details

Type of Books: Historic

Capacity: 150.

Student Details

ID: 1

Name: Suresh

Gender: Male

Contact Number: 9021207688.

12/12/23

Address : kothrud
Department : computer

* Inheritance —

- Inheritance in java is a mechanism in which, one Object aquires all the properties and behaviours of a parent Object.
- The idea behind Inheritance in java is that we can create new classes that are developed upon existing classes.
- When we inherit a class we can reuse the methods and feuilds of the parent class, moreever we can also add new members to that current class.
- Inheritance represents IS-A-Relationship which is also known as parent child relationship

- Why we we Inheritance in java?

→ 1) Reusability.
2) Time consumption is less.
3) code sharing.
4) To avoid data Redundancy.

Terminologies used in Inheritance —

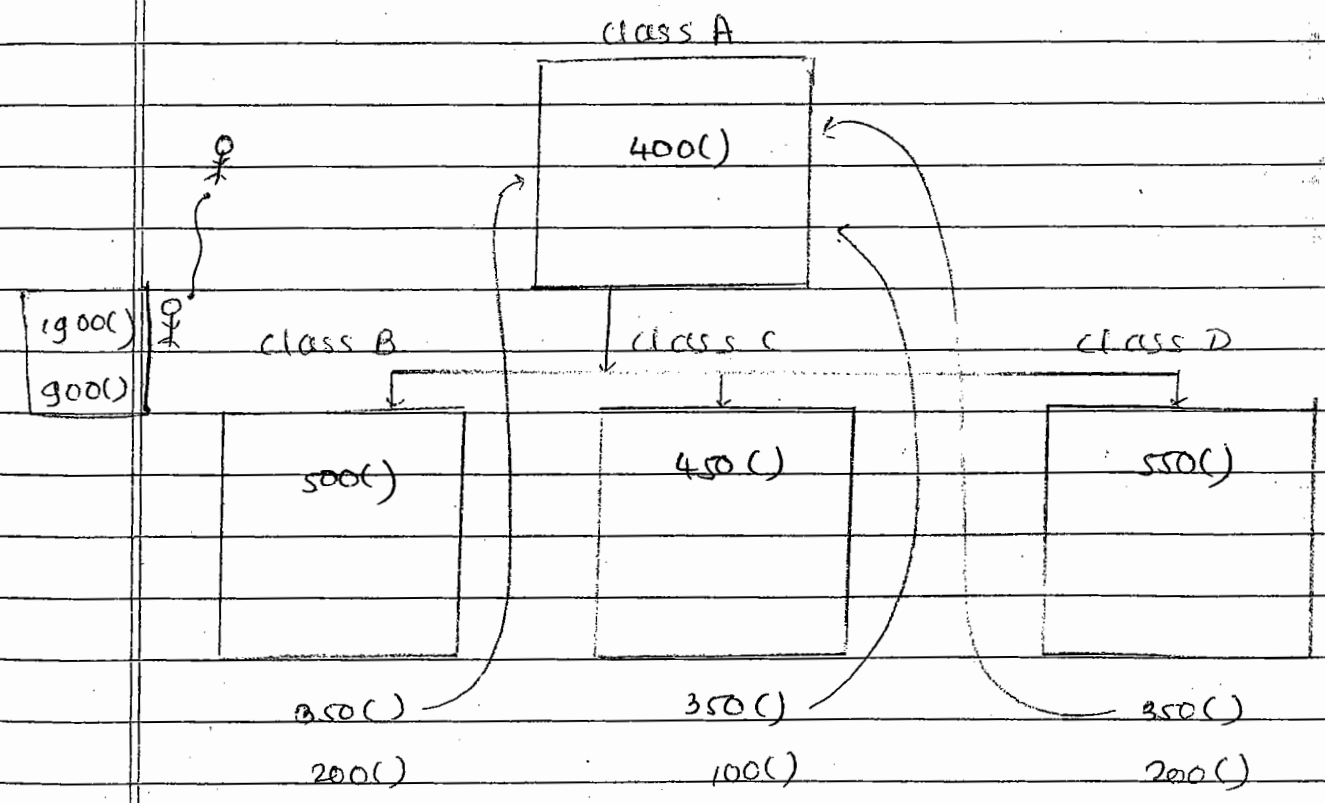
1) class —
A class is a blue print from which Objects are created.

2) child class —
A class which inherits the other class is known

as child class / derived class / extended class / sub class

3) parent class. —
A class from whose a sub class inherits the features (functionalities). is known as parent class / ~~super class~~ / base class / etc.

4) Reusability. —
Reusability is a mechanism which facilitates us to use the fields and methods of the existing class when we create a new class.



Cyclic inheritance
(Just a conceptual thing)

parent become a child &
child become parent } → It is not possible

Types of Inheritance —

1) Single level Inheritance —
2) Multilevel Inheritance —
3) Multiple Inheritance —
4) hirecical Inheritance —
5) Hybrid Inheritance —

```
Class Parent
{
    String a = "Parent Class Variable";
    public void m1()
    {
        s.o.pln. ("Parent m1()");
    }
    Public void m2()
    {
        s.o pln. ("Parent m2()");
    }
}
class child extends Parent
{
    String b = " child class variable";
    public void m3()
    {
        s.o.pln. ("child m2()");
    }
}
Class Driver1
```

Interface - can extends any class

```
{
    public static void main (String[] args)
    {
        Child obj = new Child ();
        Obj . m2 ();
        S.o.pln. (obj. b);
        obj . m1();
        obj . m2();
        S.o.pln. (obj.a);
    }
}
```

extends. —

- The keyword basically establish a relationship of Inheritance among classes.
- If a class Extends another class then we say that it has aquired all the properties & behavior of the parent class

Note. —

- We can use extends keyword in class as well as interface

1) one class can extends only one class.
2) one Interface can extends more than one interface.

Ex1 —

```
class Parent1{}
class Parent2{}
class Child extends Parent1, Parent2{}
```

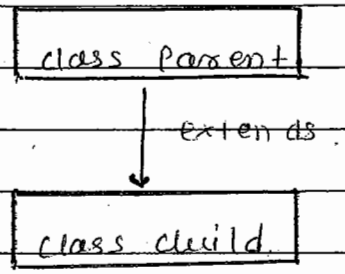
X error

Ex. 2. —

```
interface Parent 1 {}
interface Parent 2 {}
interface child extends Parent 1, Parent 2 {}
```

1) * Single level Inheritance. —

- Single level Inheritance consist one parent class and one child class



* Ex of single level Inheritance. —

1) shapes → square.

```
class shapes
{
        float area = 238.78f;
        float diameter mentions = 23;
}
Class Square extends shapes
{
        int size = 4;
        int height = 20;
        int width = 20;
}
class Square1
{
```

```
public static void main(s[]args)
{
    square a = new square();
    s.o.pln ("square info");
    pln. (a.area);
    pln (a. diamentions);
    pln. (a. size);
    pln. (a. height);
    pln. (a. width);
}
}
```

o/p. ⟶ square info
238.78
23
4
20
20

2) Airways Aeroplane.

```
class Airways
{
    string type = "Jet, Aeroplane, Helicopter, Figluter Jet";
    double weight = 1287694388.92d;
}
class Aeroplane extends Airways.
{
    float setting capacity = 234f;
    int noofEngines = 4;
}
class Aeroplane:1
```

```
}
    p. s. v. m (S[]args)
    {
        Aeroplane a = new Aeroplane();
        s.o.pln.("Aircrafts Info");
        s.o.pln.(a.type);
        s.o.pln.(a.weight);
        s.o.pln.(a.setting(capacity));
        s.o.pln.(a.noOfEngines);
    }
}
```

3) Fabrics silk. —

```
class Fabrics
{
    float length = 23425.80f;
    int price = 2500;
}
class silk extends Fabrics.
{
    string color = "red";
    string type = "Organza silk";
}
class
{
    p. s. v. m. (S.[]a)
    {
        silk a = new silk();
        s.o.pln. ("silk Info");
```

```
s.o.pln (a.length);
s.o.pln (a.price);
s.o.pln (a.color);
s.o.pln (a.type);
}
}
```

O/p. ⟶ Silk Info

length : 2342 5.80

price : 2500

color : red

Type : Organza Silk

Inheritance

diamond problem

ambiguity issues

1) constructors ambiguity

2) [illegible]

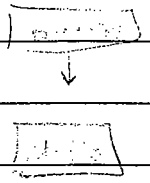
- [illegible]

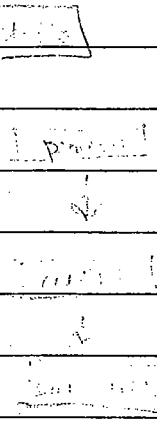
- [illegible]

obscure methods e.g. [illegible] pointer null

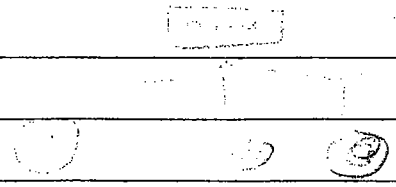
method which does [illegible] body

[illegible]

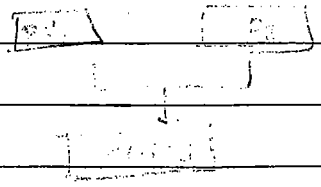
1) 

2) 

multilevel inheritance

3) 

Hierarchical inheritance

4) 

multiple inheritance

5/12/23.

2) Multilevel Inheritance —

- one class Extends another class and that extended class becomes the parent for another is known as multil Inheritance.
- It includes the involvement of more than one cla at different level.

Ex. — 1) Subject → Java → Inheritance

2) Airways → Aeroplane → Air India.

3) mobile → smartphone → Android

4) fabric → silk → Organzasilk

5) sport → cricket → virat kohli

Ex. —

```
class Human
{
    String name;
    Human (String name)
    {
        this. name = name;
    }
    void eat()
```

```
{
    s.o.pln.("Human can eat");
}
void think()
{
    s.o.pln.("Human can Think");
}
void talk()
{
    s.o.pln.("Human can Talk");
}
}
class UnderGraduate extends Human
{
    String comany;
    String branch;
    UnderGraduate(String name, String company, String branch)
    {
        super(name);
        this.company = company;
        this.branch = branch;
    }
    void branch()
    {
        s.o.pln.("Engineer Branch :"+ this.branch);
    }
    void work()
    {
        s.o.pln.("Engineer is working at - "+this.company);
    }
}
```

```
class PostGraduation extends UnderGraduate
{
        string subj;
        PostGraduation (string name, string subj, string company
                        string branch)
        {
                super(name, company, branch);
                this.subj = subj;
        }
        void stream()
        {
                s.o.pln. ("Subject for Mtech is : "+ this.subj);
        }
}
class Driver3
{
        P.s v.m. (s[]a)
        {
                PostGraduation a = new PostGraduation ("Mansi",
"Thermodynamics","TCS", "CS");
                s.o.pln. (a.name);          o/p → Mansi
                a.eat();                    Human can eat
                a.talk();                   Human can Talk
                a.think();                  Human can Think
                a.branch();                 Engineer Branch : CS
                a.work();                   Engineer is working at - TCS
                a.stream();                 Subject for Mtech is :
        }                                   Thermodynamics.
}
```

* Hirarcleical Inheritance. —

- Hirarcleical Inheritance in java is a type of Inheritance in which the same class is Inheritated by more than one class.

OR

- If multiple child classes are pointing towards the same parent class is known as Hirarcleical Inheritance.

1) Example. —

color

Blue | Red | Purple.

2)

sport

cricket | Tennis | Basketball.

3)

movies

Historical movies | comedey movies | Horror movies.

4)

Fabric

silk | cotton | wolvate.

5)

Defence

Army | Navy | Air Force

1) Ex. —

```
Class Movies
{
    void buget()
    {
        s.o.pln.("Buget of the Movie ");
    }
    void story()
    {
        s.o.pln.("story Line");
    }
    void directors()
    {
        s.o.pln.("ABC");
    }
}
class HistoricalMovies extends Movies
{
    void name()
    {
        s.o.pln.("subhedar");
        s.o.pln("Manikarnika");
    }
}
```

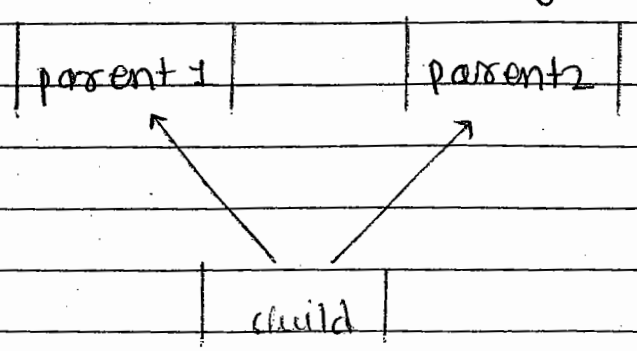
```
class ActionMovies extends Movies
{
    void actors()
    {
        s.o.pln.("Ranbir kapoor");
        s.o.pln.("Allu Arjun");
    }
}
class HorrorMovies extends Movies
{
    void ticketPrice()
    {
        s.o.pln.("220");
    }
}
class Driver2
{
    p.s.v.m.(s[]a)
    {
        Movies a = new Movies();
        a.buget();
        a.story();
        a.director();
        HistoricalMovies b = new HistoricalMovies();
        b.name();
        ActionMovies c = new ActionMovies();
        c.actors();
        HorrorMovies d = new HorrorMovies();
        d.ticketPrice();
    }
}
```

06/12/23.

Output. — Buget of the Movies
Story Line
ABC
Subhedar
Manikarnika
Ranbir kapoor.
Allu Arjun.
220.

4) * Multiple Inheritance. —

- Multiple Inheritance in Java is the capability of creating a Single class with multiple Super class (parent class).
- Unlike Some other programming lang. like C++, Java dosen't have a support for multiple Inheritance in classes
- Java dosen't support multiple Inheritance in classes because it can lead to diamond problem.
- So Insted of resolving that problem java has restricted multiple Inheritance in classes and because of which we can extend only 1 class to another



{ private String = "ABCD" ;
}
class B extends class P
{
}
class Driver
{ B a = new B();
}

can we
Ans, No!!!

* diamond problem. —

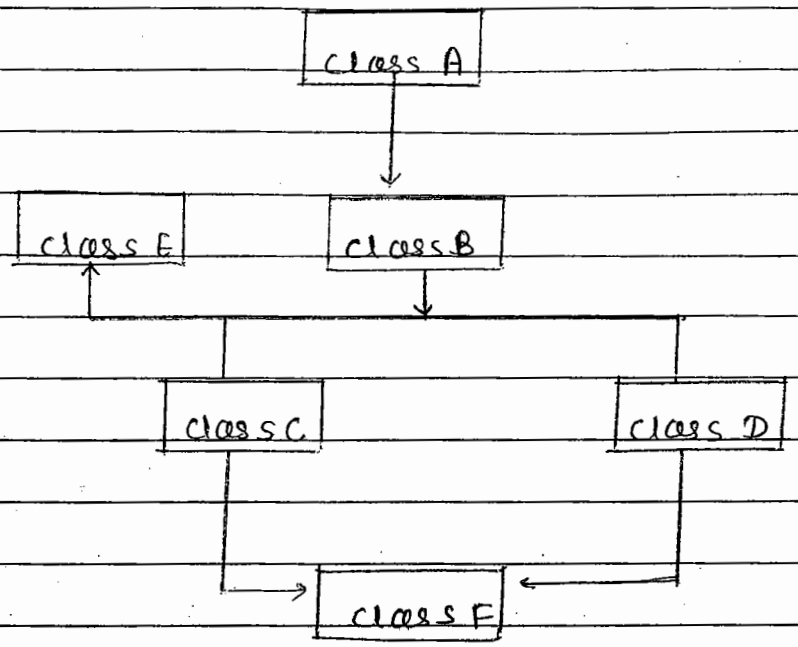
- diamond problem is an ambiguity (confusion) at the execution.
- consider we have a class child and that child class have two parent classes. Class Parent 1, class parent 2
- Now, If we try to create an object of a child class, the constructor in child class will make a call to the parent constructor using super call statement & there will be ~~an~~ an ambiguity, which parent constructor must be Invoked.
- If both the parent classes has same method signature again it will create an ambiguity for the child class which method must be Invoked

* Note. —

- To overcome these problem of multiple Inheritance we have Interface.
- Multiple Inheritance in Java can be achieved using Interface.

5) Hybrid Inheritance. —

- Hybrid Inheritance is a combination of more than one type of Inheritance
- It can be achieved through Interfaces only. Because, multiple Inheritance is not supported in classes in Java.
- Basically it is a combination of single, multiple, heirchical Inheritance.



* Ex. Multilevel. —

1) class sports {

```
        string rules;
        sports()
        {
            this. rules = "abcd";
        } void score() {
                s.o.pln. ("882");
            }
    }
class Cricket extends sports
{
    void stadium() {
            s.o.pln. (" Wankhed, Mumbai");
        }
        void ticketprice()
        {
            s.o.pln. (800);
        }
}
```

```
Class Viratkohali extends cricket
{
        void matchesPlayed()
        {
            s.o.pln.(150)
        }
        void centuries()
        {
            s.o.pln.(1000);
        }
}
class Driver4{
            P.s.v.m.(String a){
            viraatkohali a = new Viraatkohali();
            a.score();
            a.Stadium();
            a.ticketprice();
            a.matchesplayed();
            a.centuries();
            }
}
```

o/p. ⟶

2) class Mobile1 {

```
        void storage()
        {
            s.o.pln.("16GB");
        }
```

}

```
void battery()
{
        s.o.pln. ("xyz");
}
} class smartphone extends Mobile
{
    void company()
    {
        s.o.pln. ("mi");
    }
    void model()
    {
        s.o.pln. ("Note 5 Pro");
    }
}
class Android extends smartphone
{
    void color()
    {
        s.o.pln. ("Black");
    }
    void Ram()
    {
        s.o.pln. ("64GB");
    }
}
class Mobile
{
        p.s.v.m(S[]a)
    {
```

```
        Android a = new Android();
        a.storage();
        a.battery();
        a.company();
        a.model();
        a.color();
        a.Ram();
    }
}
```

o/p. →

* Hierarchical Example:—

1)
```
class Fabric1
{
    void price()
    {
        s.o.pln.("450/meter");
    }
}
class silk extends Fabric1
{
    void manufacturerAt()
    {
        s.o.pln.("Varanasi");
    }
```

```
}
class cotton extends Fabric
{
        void color()
        {
            s.o.pln. ("Red");
        }
}
class Velvet extends Fabric
{
        void length()
        {
            s.o.pln. ("20 meter");
        }
}
class Fabric
{
        p. s. v. m. (s[] a)
        {
            fabric a = new Fabric();
            a. price();
            silk b = new silk();
            b. manufacturerAt();
            cotton c = new cotton();
            c. color();
            Velvet d = new Velvet();
            d. length();
        }
}
```

O/P. → 456/meter

Varanasi

Red

20 meter

* Type Casting. —

- Type casting is a way of converting data from one data type to another data type.
- These process of ~~converse~~ data conversion is also known as type casting. OR Type conversion OR Type coersion.
- We can cast both primitive and non-primitive using ~~up~~ a type cast operator. [(datatype)]

* Note. —

we cannot perform type casting for boolean in Java

1)

Type Casting
- primitive
  - widenning
  - Narrowing
- Non - primitive
  - upcasting
  - downcasting.

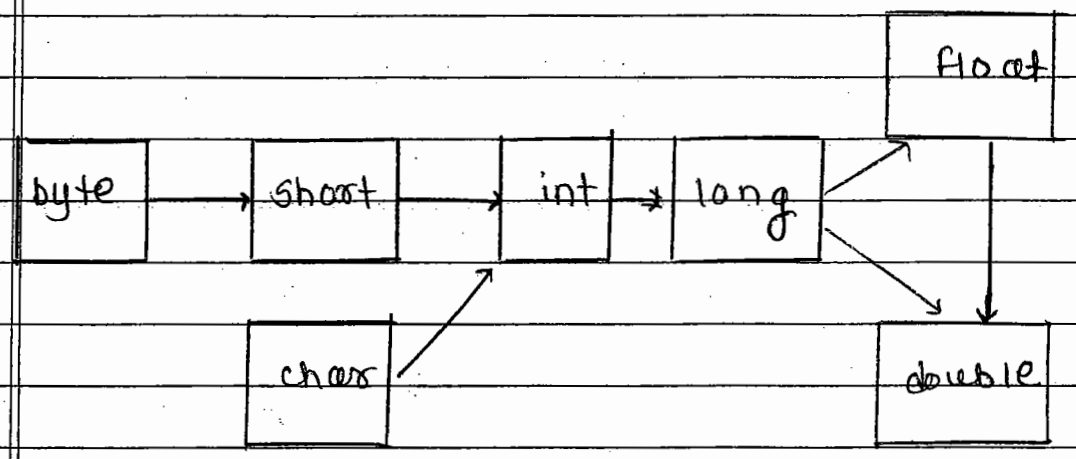
2)

Type casting.
- Implicit casting
  - widenning
  - ~~N~~ upcasting
- Explicit Casting.
  - Narrowing
  - downcasting.

1) Widening. —

- These type of conversion occurs implicitely when converting from a narrow range datatype to wider range datatype.
- Implicit casting is possible in two condition. —

1) It checks the compatibility of data.
2) If the targeted value which is to be converted should not be bigger than the storing variable's or container's datatype.



Ex 1). — {

~~short a =~~

```
byte   a = 127
short  b = a;
int    c = a;
long   d = a;
float  e = a;
double f = a;
char   g = a;  // CTE
```

}

2) {

```
short a = 32767;
byte  b = a;   // CTE
char  c = a;   // CTE
int   d = a;
long  e = a;
float f = a;
double g = a;
```

}

3) {

```
int   a = 2147483647;
byte  b = a;   // CTE;
short c = a;   // CTE.
char  d = a;   // CTE
long  e = a;
float f = a;
double g = a;
```

}

4) {

```
long  a = 21474g;
byte  b = a;   // CTE
short c = a;   // CTE
char  d = a;   // CTE.
int   e = a;   // CTE.
float f = a;
double g = a;
```

}

5) {

```
float   a = ~~99421;~~ 1.4x10^38
byte    b = a; // CTE
short   c = a; // CTE
char    d = a; // CTE
int     e = a; // CTE
long    f = a; // CTE
double  g = a;
```

}

6) {

```
double  a = 99218494219 8;
byte    b = a; // CTE
short   c = a; // CTE
int     d = a; // CTE
long    e = a; // CTE
float   f = a; // CTE
char    g = a; // CTE
```

}

7) {

```
char    a = 65535;
byte    b = a; // CTE
short   c = a; // CTE
int     d = a;
long    e = a;
float   f = a;
double  g = a;
```

}

Narrowing —

- It is a process of converting a higher data type into lower datatype.
- this process is also known as explicit conversion. because
- because it is done by the programmer explicitely. (forcefully).
- If the programmer does not perform the casting in these situation the we will get the CTE (possible lossy conversion)
- We need to use a type cast operator to perform the conversion.

1)
```
byte a = 127;
short b = a
int c = a
char d = (char) a;   // Narrowing.
long e = a;
float f = a;
double g = a;
~~char~~ h = a;
```

2)
```
short a = 32767;
byte b = (byte) a;   // Narrowing.
int c = a;
long d = a;
float e = a;
double f = a;
~~char~~
char g = a;
```

3)
```
int a = 2147483647;
byte b = (byte) a;   // Narrowing
short c = (short) a;   // Narrowing.
```

```
long d = a;
float e = a;
double f = a;
char g = a;
```

4)

```
long a = ~~9.12~~ 12 ..... 39;
byte b = (byte)~~(12 ... 39);~~ a; // Narrowing.
short c = (short) a;          // Narrowing.
int d = (int) a;              // Narrowing.
float e = a;
double f = a;
char g = a;
```

5)

```
float a = 19127 ..... 309;
byte b = (byte) a;
short c = (short) a;
int d = (int) a;
long e = (long) a;
double f = a;
char g = a;
```

6)

```
double a = 192764 .... 99;
byte b = (byte) ~~b;~~ a;
short c = (short) a;
int d = (int) a;
long e = (long) a;
float f = (float) a;
char g = a;
```

7) ~~7~~

```
char   a = 65535;
byte   b = (byte) a;
short  c = (short) a;
int    d = a;
long   e = a;
float  f = a;
double g = a;
```

09/12/23.

* Polymorphism —

- The directionary defination of polymorphism refer to a principle in biology to a speci has of diff forms or stages
- These principal can also be applied to object oriented and programming lang. like Java.
- polymorphism is derived from a greek word.
- poly means many and morph means form.
- polymorphism stands for one name & many forms.

poly morphism

many → forms

Ex:— polytechnique, polyhouse, polyclinic, polytwin, etc.
water → ices, vapour.

* polymorphism is classified into two types.
1) compile time polymorphism or static polymorphism
2) run time polymorphism or dynamic polymorphism

1) compile time polymorphism / static polymorphism.
If the binding is achieved at the compile time & Same behaviour is executed is known as compile time polymorphism.

OR

If the binding is achieved at the compile time is known as compile time polymorphism.

- argu. must be diff
- return things doesn't matter

- binding means an association of method called statement with the method defination.

Types of compile time polymorphism. —

1) method overloading.
2) constructor overloading.
3) method shadowing.
4) variable shadowing.
5) operator overloading.

Ex. — **Compile time polymorphism.** —

```
Class CompileTimePoly
{
    P. s. v. m(s[]a)
    {
        m1(10);
        byte a=1;
        m1(a);
    }
    P. s. string m1(int a)
    {
        s.o.pln. (" No argu");
        return null;
    }
    Privacte static void m1(byte a)
    {
        s.o. pln. ("byte argu");
    }
    Static void m1( int a, int b)
    {
```

binding active at compile time.

```
        s.o.pln. ("2 argu");
    }
}
```

o/p → No argu.
byte argu.
2 argu

method Overloading.—

In a single class when there is a more than 1 methods with ~~f~~ same ~~argument~~ Name and different formal arguments is known as a method Overloading.

Ex.—

```
Class CompileTimePoly
{
    compileTimePoly obj = new CompileTimepoly();
    {
        obj. m1(10);
        byte b = 1;
        m1(b);
        obj. m1(10, 20);
        m1(10, b);
    }
    public string m1(int a)
    {
        s.o.pln. (" int argu string return type");
        return null;
    }
    private static void m1 (byte a)
    {
        s.o.pln. ("byte argu return type is void");
    }
```

we in Interface to create the default method or creating a default block.

```
int m1 (int a, int b)
{
    ~~p~~ s.o.pln.(" 2 argu int, int with int return type"
    return 1;
}

protected static boolean m1 (int a, byte b)
{
    s.o.pln.(" 2 argu int, byte with boolean return type"
    return true;
}
}
```

o/p. → int argu string return type.
byte argu return type is void
2 argu int, int with int return type.
2 argu int, byte with boolean return type.

* Constructor Overloading —

A class having more than one constructor is known as a Constructor Overloading.

Ex. —

```
class CompileTimePolymorphism
{
    CompileTimePoly
    {
        this(10);
        s.o.pln.(" No argu default cons");
    }

    private CompileTimePoly(int a)
    {
        this (10, 20);
```

```
        s.o.pln("1 argu. private const");
    }
    public CompileTimePoly (int a, int b)
    {
        this (10, 20, 30)
        s.o.pln(" 2 gregu public const");
    }
    protected CompileTimePoly(int a, int b, int c)
    {
        s.o.pln(" 2 go argu public const");
    }
    Public static void main (String [] args)
    {
        CompileTimePoly obj = new CompileTimePoly();
    }
}
```

o/p. → No argu default cons.
1 argu private cons.
2 agrgu public cons.
2 argu public cons.

used to check instance of class

11/01/23

To access only parent class block

Upcasting —

- When the reference variable of super class refers to the object of subclass it is known as Upcasting, Implicit Casting or Generalization.
- In other words the subclass object type is converted into super class type is known as Upcasting.

Ex. —

```
class Parent
{
    void car()
    {
        s.o.pln.("Parent car");
    }
}
class child1 extends Parent
{
    void Bike()
    {
        s.o.pln.("~~Parent B~~ child1 Bike");
    }
}
class child2 extends Parent
{
    void Bicycle()
    {
        s.o.pln ("child2 Bicycle");
    }
}
class UpcastingExample
```

```
}

    p.s.v.m.(S[]a)
    {
        Parent a = new child1();
        a.bike();  // CTE (cannot find symbol) Because we
                   cannot invoke the members of child during
                   upcasting.
        a.car();
        a.bicycle();  // cannot find symbol.
        Parent b = new child2();
        b.bicycle();  // CTE (cannot find symbol).
        b.car();  <-
        b.bike();  // cannot find symbol.
    }
}
```

O/p. →

Note.—

- Converting a Sub class type into a super class type is called as Generalization.
- when we peform upcasting, the child behaves like the parent
- and also we cannot invoke the members of child class.

To avoid extra memory allocation

Downcasting —

When a subclass reference referce to a super class object type is known as downcasting or specialization.

In other words, when super class is converted in sub class type is known as downcasting.

Ex. —

```
class Downcast
{
    P.s.v.m.(S[]a)
    {
        object a = new string();
        stringbuffer b = (stringbuffer) a; // CTE
                                (class cast Exceptiion)
        String c = (string) a;
    }
}
```

Rules for downcasting. —

$$\boxed{A\ b = (c)\ d;}$$

Rules at Compiletime

The type of c & d must have some relation, either parent / child ~~or~~, child / Parent or they must be of same type otherwise CTE

c must be the same type of A or c must be the child of A.

Rules at runtime. —

1) The Underlying original object ~~of d b of d~~ must be either same type ~~of A~~ or it must be the derived type of c.

* ~~cast class~~ class cast Exception.—

- When we try to do downcast a reference variable to a subclass type and the object does not have an Instance of the sub class type then we will get a class cast Exception.
- In simple words it means when we try to perform downcasting without performing Upcasting we will get a class castException.

Note.—
It is mandatory to perform upcasting before downcasting.

Ex.—

Parent

child 1 child 2

subchild 1 subchild 2 subchild 3 subchild 4

```
class parent{}
class child1 extends Parent{}
class subchild1{} extends child1{}
class subchild2{} extends child1{}
class child2 extends Parent{}
class subchild3{} extends child2{}
class subchild4{} extends child2{}
```

```
p. s. v. m. (S[] args);
{
        parent a = new child2();
subchild 3 b = (subchild 3) a;
child 2 b = (subchild1) a;
subchild1 b = (subchild1) a;
child1 a = new subchild1();
subchild 2 b = (subchild 2) a;
child1 b = (subchild 2) a;
parent b = (child 1) a;
parent a = new child 1();
parent b = (subchild1) a;
child b = (subchild 1) a;
subchild 1 c = (child1) a;
child b = (child 1) a;
}
}
```

* Ex. – Amezone payment. —

```
File 1. - Class Gpay extends payment { string option = "Google pay"
class p pay extends payment { string option = "phone pay";
class paytm extends payment { string option = "paytm"; }
class payment
{
        p v chooseOption(Payment p)
        {
            if (p instanceof Gpay)
            {
                Gpay g = (Gpay)p;
```

```
        s.o.pln.(g.option);
    }
    else if (P instanceof ppay)
    {
        ppay py = (Ppay)P;
        s.o.pln.(Py.option);
    }
    else if (P instanceof paytm)
    {
        paytm pt = (paytm) P;
        s.o.pln.(Pt.option);
    }
}
}
```

1e 2 —

```
import java.util.*;
class AmazonDriver
{
    P.s.v.m.(S[]a)
    {
        Scanner sc = new Scanner(System.in);
        for (;;)
        {
            s.o.pln.();
            s.o.pln.("Welcome To payment module");
            s.o.pln.("select the payment option");
            s.o.pln.("1. Gpay 2. Ppay 3. paytm. 4. COD.
                    5. cancle");
            s.o.pln.("Enter a n option:");
            int option = sc.nextInt();
            payment payment;
```

```
switch (option)
{
    case 1: {
            payment = new Gpay();
            payment.chooseOption(payment);
            break;
        }
    case 2: {
            payment = new ppay();
            payment.chooseOption(payment);
            break; }
    case 3: {
            payment = new paytm();
            payment.chooseOption(payment);
            break; }
    case 4: { s.o.pln("keep your cash ready & u
            have choose COD");
            break; }
    case 5: {  s.exit(0);
            }
    default: { s.o.pln("Wrong Input");
                break;
}   }   }   }   }
```

— When we [illegible] parent class [illegible] then we use upcasting.

12/12/23.

- child class.

* Instance of —

- Instance of is an keyword & operator which is used to check wether an object is an instance of a specific type (specific type / class, subclass, interface).
- It is also known as a type comparision operator Because it compares the Instance with its type.
- The return type of Instance of ~~tru~~ operator is a boolean. (i.e. true or false).
- It is a binary operator.

3) method shadowing —

- If a subclass & a superclass haves a static method with same signature is known as method shadowing.

Q) Which method Implimentation get executed?

→ In method shadowing binding is done at the compiletime and the execution of method depends upon the reference type.

Ex. —

```
class Parent
{
        public static streing Study()
        {
                ~~P.S.V.~~ S.O.pln.("studying regularly");
        }
}
class child extends Parent
{
```

Protected
~~public~~ default
private --

```
public static String Study()
{
        S.o.pln.("I do");
}
}
class Driver
{
    P.S.V.M.(S[]a)
    {
        Child a = new Child();
        a.Study();
        parent.Study()
    }
}
```

O/P → Study regularly.
I do.

* Rules for method shadowing —

1) Is-A-Relationship is mandatory.
2) Method must be static.
3) Method signature must be the same.
4) Return type must be same for primitive & Return type can be co-variant for non-primitive.
5) We cannot Reduce the scope of Access modifier from Parent to Child.

1) class parent

```
{
    p. s. int study()
    {
        s.o.pln. ("study ");
        return 1;
    }
}
class child extends Parent
{
    P. s. byte study()
    {
        S. o.pln ("Yes");
        return 0;
    }
}
```

o/p → CTE. (does not have same return type int, byte)

2) class Parent

```
{
    P.S. boolean study()
    {
        s. o. pln ("study");
        return true;
    }
}
class child extends Parent
{
    P. s. boolean study()
```

```
{
        s.o.pln.("yes");
        return false;
    }
}
```

O/P. → ✓ Study
yes.

3)
```
class Parent
{
    p s. string Study()
    {
        s.o.pln.("Study");
        return null;
    }
}
class child extends Parent
{
    p. s. string Study()
    {
        S.o. pln ("Study");
        return null;
    }
}
```

O/P. → study
study

4)
```
class Parent
{
    P. s. object study()
    {
        s.o.pln. ("study");
```

```
        return null;
    }
}
class child extends Parent
{
    p. s. string study()
    {
        s.o.pln(" Yes");
        return null;
    }
}
```

O/p. → study
Yes.

5) class parent

```
{
    p. s. string study()
    {
        s.o.pln("study");
        return null;
    }
}
Class child extends Parent
{
    p. s. object studey()
    {
        s.o.pln(" Yes");
        return null;
    }
}
```

O/P → CTE (string, object).

6) class Parent

```
{
    P.s. String steady()
    {
        S.o.pln ("studey");
        return null;
    }
}
class child extends Parent
{
    p.s. StringBuffer study()
    {
        s.o.pln.("Yes");
        return null;
    }
}
```

O/p. → CTE (String stringBuffer)

7) class parent

```
{
    static int steady()
    {
        S.o.pln.("steady");
        return 0;
    }
}
class Child extends Parent
{
    P.s. int steady()
    {
```

```
        s.o.pln.("yes");
        return 0;
    }
}
```

O/P. → study
yes..

8) class Parent

```
{
    p.s. int study()
    {
        s.o.pln.("study");
        return 0;
    }
}
class child extends Parent
{
    static byte study()
    {
        s.o.pln.("yes");
        return 0;
    }
}
```

O/P. → CTE (cannot reduce the size of access modifiers)

9) class parent

```
{
    p.s. string study()
    {
        s.o.pln.("study");
        return null;
```

```
    }
}
class child extends parent
{
        static string study()
        {
            s.o.pln. ("study");
            return null;
        }
}
```

O/p. → CTEC cannot reduce the size of access modifiers).

10)
```
class parent
{
        static object study()
        {
            s.o.pln. ("study");
            return null;
        }
}
class child extends Parent
{
        Protected static string study()
        {   s.o.pln. ("~~study~~ yes");
            return null;
}   }
```

O/p. → study
yes.

* Variable shadowing —

- If a superclass and a subclass has a variable with same name then it is known as variable shadowing.

* Note —

A variable which is used to access the member depends on the reference type.

Ex

```
class parent
{
        string a = " Parent";
}
class child extends Parent
{
        String a = " child";
}
class Demo
{
        p. s. v. m (s[]a)
        {
                child obj = new child();
                s.o.pln. (obj.a);
                parent b = (child) obj;
                s.o.pln. (b.a);
        }
}
```

o/p. → parent.
child.

Ex. –

```
class Parent
{
        static string a = "Parent";
}
class child extends Parent
{
        static string a = "child";
}
class Demo
{
        p. s. v. m. (s[]a)
        {
                child obj = new obj ();
                S.o.pln (obj.a);
                s.o.pln. (Parent.a);
        }
}
```

o/p. → ~~Par~~ child.
parent.

* Operator Overloading. –

- Operator Overloading is used to specifiey a user define Implimentation for operation with one or both operants.
- So, we can not use operator overloading in Java there is only one example of operator overloading which is (+ operator).

Ex. – int ⊕ int → addition.

Ex. – int ⊕ string = string → concatination.

Ex. → string + string = string.

↓

concatination.

* Run time Polymorphism. —

- If the binding is achieved at the run time then it is known as Run time polymorphism or dynamic polymorphism.
- Run time polymorphism is achieved by method Overridding.

* method Overridding. —

If a sub class & a superclass have a non-static method with a same signature is known as a method Overridding.

Ex. —

```
class parent
{
        public void message()
        {
                s.o.pln.("sukanya");
        }
}
class child extends Parent
{
        public void message()
        {
                s.o.pln.("Shila");
        }
}
```

13/12/23.

```
class Driver
{
        P. s. v. m. (S[]a)
        {
            child obj = new child();
            obj. marriage();  // sheela.
            parent obj = (child) obj;
            obj. marriage(); // sushila.
        }
}
```

* Examples. method Overidding. →

1) ~~class Parent~~ RBI

```
{
        void RateOfInterest()
        {
            s.o.pln. ("8%.");
        }
}
class SBI extends RBI
{
        void RateOfInterest()
        {
            s.o.pln. ("8.29%.");
        }
}
class ~~Driver~~ HDFC extends RBI
{
        ~~p. s. v. m. (S[]a)~~ void rateOfInterest()
        {
```

```
            s.o.pln.("92%");
        }
    }
}

2) class SNJB
{
        void cutoffList()
        {
            s.o.pln.("98%");
        }
}
class Mechanical extends SNJB.
{
        void cutoffList()
        {
            s.o.pln.("80%");
        }
}
class computer extends SNJB:
{
        void cutoffList()
        {
            s.o.pln.("99%");
        }
}
3) calss Father
{
        void singing()
        {
```

```
            S.o.pln.("classical singing");
        }
    }
    class ~~extends~~ Child1 extends Father
    {
            void singging()
            {
                S.o.pln.("Rock Singing");
            }
    }
    class child2 extends Father
    {
            void singging()
            {
                s.o.pln.("Rap Singing");
            }
    }

4) class Parent
    {
            void Family Business()
            {
                s.o.pln.("Electronics shop");
            }
    }
    class child1 extends Parent
    {
            void Familiy Business()
            {
                s.o.pln.("Car showroom");
```

```
    }
}
class child2 extends Parent
{
        void FamilyBusiness
        {
            s.o.pln("Construction Business");
        }
}
```

* Rules for method Overriding. —

1) IS-A-Relationship is mandatory.
2) Method must be non-static.
3) Method signature must be the same.
4) Return type must be same for primitive & Return type can be covarient for non primitive.
5) we can not reduce scope of Access modifier from Parent to Child.

* How does JVM decides which method to call at run time?

→ - The JVM checks the actual type of the object which is calling the method, then on that actual type it calls the corresponding method.
- The actual type of an object is the type which is used with new keyword.

* Advantages of method overridding —

1) The main advantages of using method overridding is that it gives the child class the ability to change the behaviour of parent class method as per the requirement.

2) another advantage of overridding is that it provide multiple implimentation for the same method & one can call parent or child class method as per the need

* Ex. — upcasting & Downcasting —

```
class Father
{
    void singing ()
    {
        s.o.pln.(" Classical Singing");
    }
}
class child1 extends Father
{
    void Singing ()
    {
        s.o.pln.(" Rock Singing");
    }
}
class Driver4
{
    p.s.v.m.(S[]a)
    {
```

```
Father a = new Child();
a.singing();
Father b = (Father)a;
b.singing();
}

}
```

o/p. → Rock singeuing
Rock singing.

* Note. —

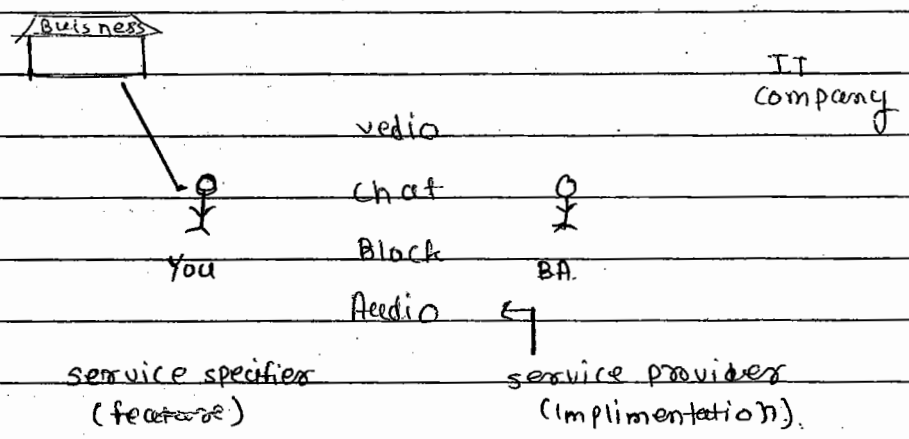
- If we are performing overridding the method present inside parent class is called as overridden method.
- And the method present inside the child class is known as Overridding method.

method in that same class
public void Demo
{
}

* Abstraction.—

The process of hidding Implementation details & showing only ~~the~~ the Important features is called as Abstraction.

It is the design process which helps the service specifier (customer) to provide the most essential features for of project without providing the Implementation details is known as abstraction.



Abstraction in Java can be achieved in two ways.—

1) Abstract class
2) Using an Interface.

* Abstract method.—

- method declared with an abstract keyword or non-access modifier in its method declaration & does not contain a method body (without any implimentation) is known as abstract method.
- In other words if we want a class to have a particular method but we want the actual ~~m~~ implimentation of the method to be determined by its child classes then we

- If then make a class as a first
- If class is not a abstract then it is concreate

14/12/23

declare the method in parent class as abstract.

* Note —

1) If we cannot have any implimentation (method body) for abstract method.
2) Semicolan is mandatory at the end of method declaration statement.
3) If a class contains a single abstract method in it then it is mandatory to declare the class as abstract
4) If we have any child class (derived class). It is mandatory the child class must provide the implimentation for the parent abstract method or the child class must be declared as abstract.

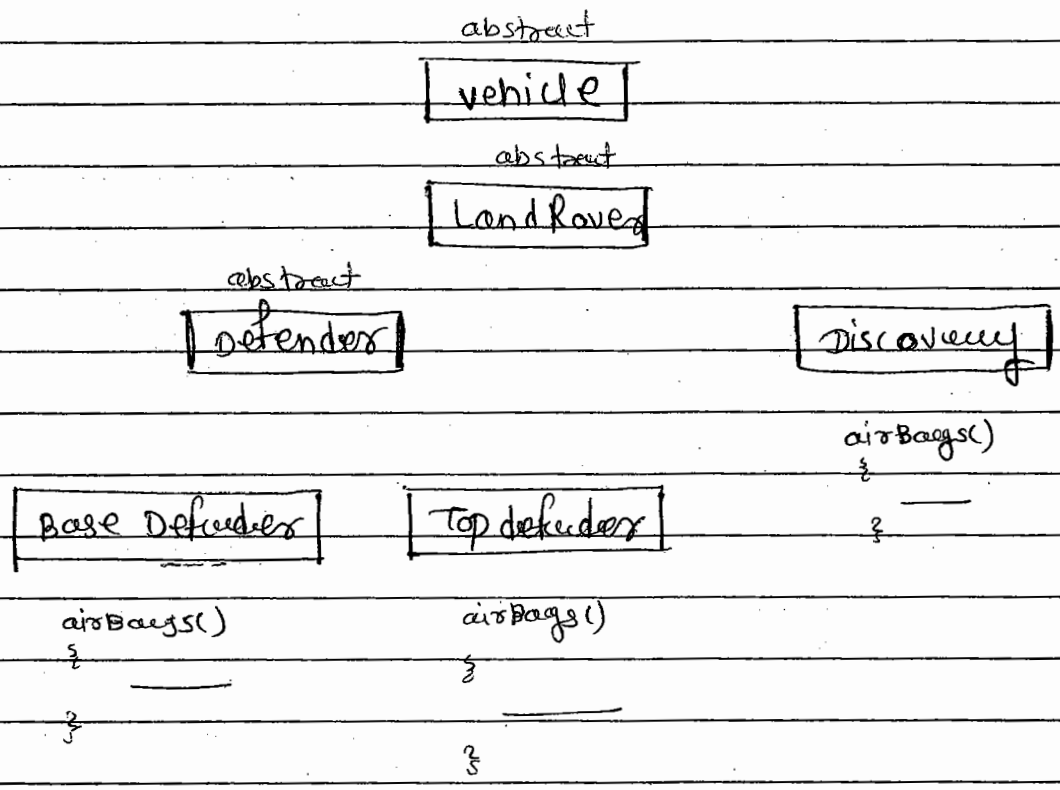
1) Ex. —

```
abstract
class parent
{
        public abstract void m1();
}
class child extends parent
{
        public void m1()
        {
            s.o.p(n. ("Impl for parent abstract method");
        }
}
class Driver
{
        p.s.v.m.(S[]a)
        {
```

```
        child a = new child();
        a.m1();
    }
}
```

op. ⟶ kuchli for parent abstract method.

2)



2) ~~class~~ ~~ca~~ abstract class Vehicle

```
{
    abstract int airBags();
}
abstract class LandRover extends vehicle { }
{

}
class Discovery extends LandRover
{
```

(illegal combination)

Final & abstract

```
        public int airBags()
        {
            return 6;
        }
}
abstract class Defender extends LandRover
{

}
class BaseDefender extends Defender
{
        public int airBags()
        {
            return 5;
        }
}
class TopDefender extends Defender
{
        public int airBags()
        {
            return 8;
        }
}
```

O/P.

3) abstract class Parent

```
{
        private abstract void m1();
}
class child extends Parent
{
```

- final is use for variable
- class
- method

```
        public void m1()
        {
            s.o.pln. (" Imp for Present abstract method");
        }
}
Class Driver
{
        p.s.v.m. st (s[]a)
        {
            child a = new child();
            a.m1();
        }
}
```

o/p → Impl for parent abstract method.

* final method —

- If a method is declared with final keyword or no access modifier that method is known as a final method or concreate.
- A final method cannot be overrided.

1) Ex. —

```
class Bike
{
        public final void discBrake()
        {
            S.o.pln. ("Bike 150cc");
            s.o.pln. ("disc brake is present");
        }
}
```

multithreading
volatile - multithreading

variables
variable
method

class
static FP - class
method

methods
Interface

Transient - serialization

synchronized - block
method

sto

```
class unicorn extends Bike
{
        p. s. void discBrake()
        {
                s.o.pln.("Bike is so cc");
                s.o.pln.("I dont have disc brake");
        }
}
class Driverg
{
        p.s.v.m(S[]a)
        {
                unicorn a = new unicorn();
                a.discBrake();
        }
}
```

CTE — discBrake() in unicorn cannot override discBrake() in Bike.

Because method is final.

* Concreate class. —

- A class which has implementation for all the methods present in it, is known as a Concreate class.
- If we want to stop the Inheritance of a class we need to declare the class as final.

```
class Maruti
{
        p. void ssc()
```

```
        s.o.pln. ("year of passout : 2016");
        s.o.pln. ("percetage : 85.80%");
    }

    public void HSC()
    {
        s.o.pln. ("year of passout : 2018");
        s.o.pln. ("Percetage : 77.80");
        s.o.pln. ("
    }
    public void BE()
    {
        s.o.pln. ("year of Passout : 2023");
        s.o.pln. ("CGPA : 7.97");
        s.o.pln. ("
    }
}
```

* Abstract class —

- A class which contains Abstract keyword in its declaration is known as abstract class.
- syntax —

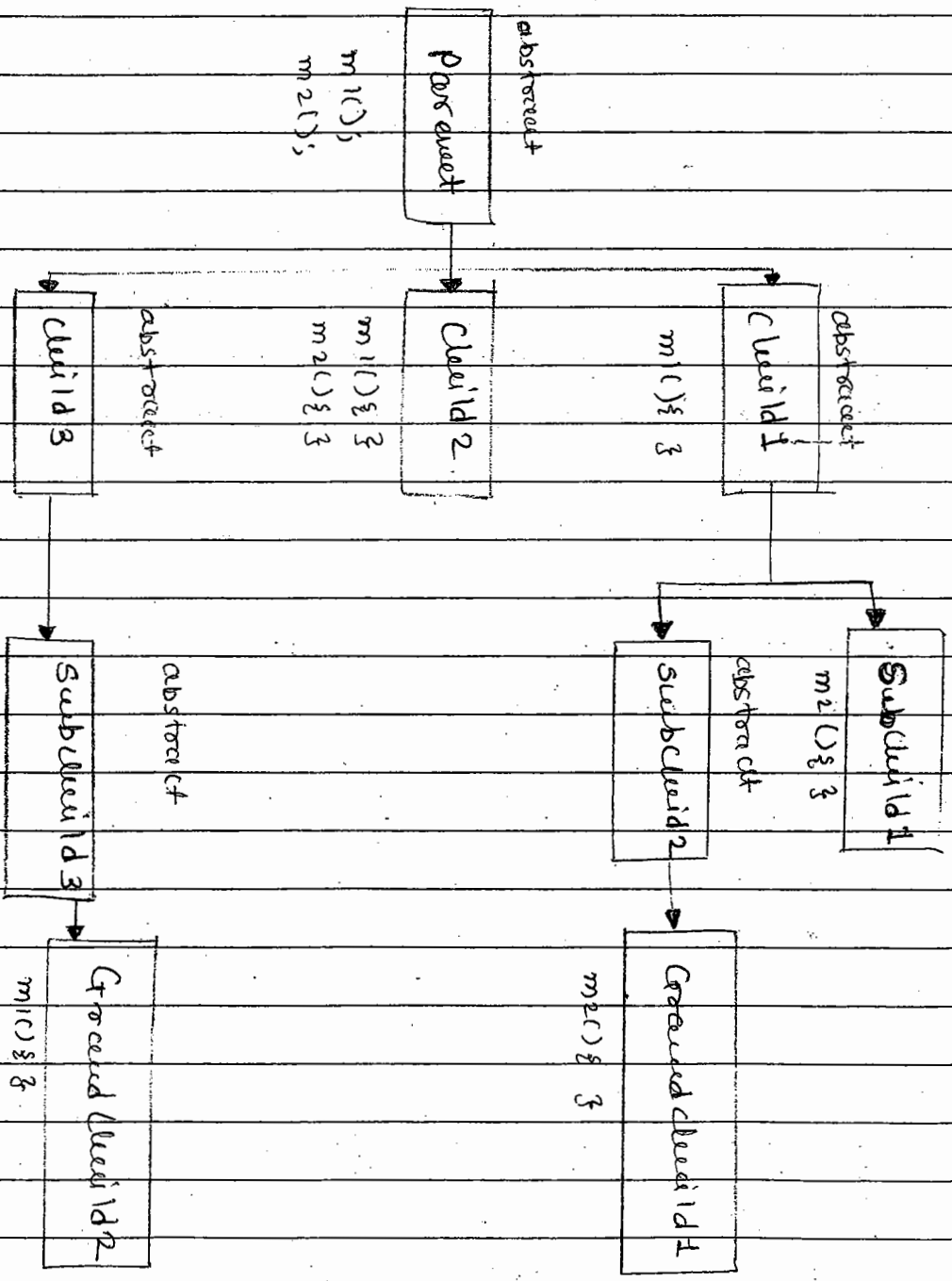
```
abstract class className
{

}
```

- A abstract class may or may not contain any abstract method in it.
- But if a class has ~~a sing~~ atleast a single abstract method in it, then it is mandatory to declare the

Class as abstract

- if a class is declared as abstract it cannot be Instenciated. (cannot create object)
- to use an ~~abstract class~~ members of abstract class we have to Inherit it from another class (child class), ~~it~~ by providing the implimentation to it's abstract method.



```
abstract class parent
{
        abstract void m1();
        abstract void m2();
}
abstract class child1 extends Parent
{
        void m1()
        {
                s.o.pln.("from child1");
        }
}
class subchild1 extends child1
{
        void m2()
        {
                s.o.pln.("from subchild1");
        }

}
abstract class subchild2 extends child1
{


}
class Grandchild1 extends subchild2()
{
        void m2()
        {
                s.o.pln.("from Grandchild1");
        }
}
```

```
class child2 extends Parent
{
    void m1()
    {
        s.o.pln("from child2m1");
    }
    void m2()
    {
        s.o.pln("from child2 m2");
    }
}
abstract class child3 extends parent
{

}
abstract class subchild3 extends child3
{

}
class Grandchild2 extends subchild2
{
    void m1()
    {
        s.o.pln("from Grandchild2 m1");
    }
    void m2()
    {
        s.o.pln("from Grandchild2 m2");
    }
}
```

```
class Abstracuet
{
    p. s. v. m. (S[]a)
    {
        subchild1  a = new subchild();
        a. ~~void~~ m2();
        Grandchild1  b = new Grandchild
        b. ~~void~~ m2();
        child2  c = new child();
        c. ~~void~~ m1();
        c. ~~void~~ m2();
        Grandchild  d = new Grandchild();
        d. ~~void~~ m1();
        d. m2();
    }
}
```

o/p. ⟶ from subchild1
from ~~sub~~ chilH1
from child2 m1
from child2 m2
from GrandchildH1d2 m1
from Grandchild2 m2

Note. —

we cannot use final modifier with abstract modifie for a method & class.

1) Ex. → class parent

```
{
        public abstract final void m1();
}
```

o/p → illigal combination of modifiers.

2) final abstract class Parent

```
{

}
```

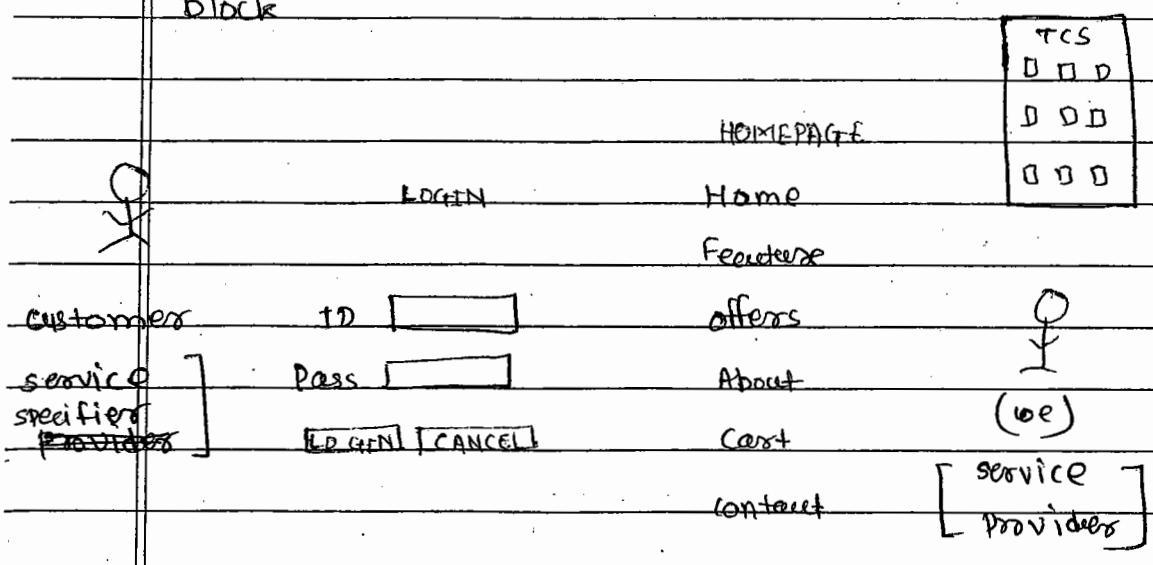
illigal combination of modifiers.

void m1();
}

15/12/23

* Interface —

- An Interface in java is an abstract type that is used to describe the behaviours (methods that a class must implement
- An Interface is reference type which is similar to a class
- Interface is a collection of abstract method
- Before version 1.8 Interface was used to achive 100% abstraction.
- But after JDK 1.8 an Interface can also contain Default and Static blocks.
- from version 1.9 we can also use private methods inside an Interface and also we can do create a nested Interface from 1.8 version 1.8 onwards.
- method body exists only for default and static block



* Declaration of an Interface

An Interface keyword is used to declare a Interface

& they are final

— cannot use const. in Interface

* Syntax. — Interface InterfaceName
{

}

* Ex. — Interface Demo
{

}

* Characteristics / properties of Interface. —

- An Interface is Implicitely abstract in Nature i.e. we don't need to provide abstract keyword in it's declaration.
- Each method present in Interface is also implicitely abstract.
- method present in Interface are by default public
- Global variable / variables declared inside an Interface are static final.

* Ex. —

Interface Demo
{

provided implicitely] static final String a = "Hello";
public abstract void m1();

nad use

It is not recommended to use main method in Interface
we cannot create an obj of Interface

* Interface is similar to a class in following ways.-

- An Interface can contain any number of methods
- Interface file is saved with .java extentention with the name of Interface matching with the file name.
- After compilation a byte code of Interface is generated. (.class extention)

* An Interface is different from a class ~~to~~ from follow ways —

1) We cannot Instanciate an Interface.
2) Interface dosen't contains any constructor.
3) all the ~~methods~~ non-static methods in Interface are abstract.
4) an Interface can consist of default & static block.
5) we cannot have Initializers(Static & non static blocks) Inside an Interface.
6) An Interface cannot be extended by a class.
7) One Interface can extend multiple Interface

Ex. —

1)
```
Interface Demo2
{
        public static void m1()
        {
                s.o.pln.("Hello from an interface m1()");
        }
}
class Driver11
```

```
{
        p.s.v.m.(s[]a)
        {
            s.o.pln.("Driver11");
            Demo2.m1();
        }
}
```

o/p. → Driver11
Hello from an interface m1()

2) Interface Demo2

```
{
        p.s.v.m1()
        {
            s.o.pln.("hello from an interface m1()");
        }
        p.s.v.m(s[]arge)
        {
            m1();
        }
}
```

o/p. → hello from an interface m1()
But it is not recommended to write main method in Interface. it ~~no~~ is not good programming practice

* Implementation of an interface. —
- A class Uses an Impliments keyword to implement an interface.
- the Impliments keyword also established a relation of parent and child.
- we cannot use Implements keyword for class
* the implements keyword appear in the class declaration like the following example. —

1) Ex. —

```
interface Interf
{
    void m1();
}
class implementation implements Interf
{
    public void m1() {
        s.o.pln. ("Imp for interface m1()");
    }
}
```

* Marker Interface. —
- An Interface which does not contain any abstract method, fields, Constants is known as Marker Interface or Tagged Interface or Empty Interface
- In simple word An Interface which is empty is known as marker Interface
* Ex. →

1) Random access 2) clonable 3) serializable

interface implements interface extends class extends

* Functional Interface. —

- An Interface which contains a single abstract method in it is known as a functional Interface
- It can have any number of default and static blocks in it

Ex. —

```
interface Functional
{
    int a = 10;
    void m1();
    public static void m1()
    {
        s.o.pln.("m1()");
    }
    public static void m2()
    {
        s.o.pln.("m2()");
    }
    public default void m3()
    {
        s.o.pln.("Hello");
    }
}
```

* Extending an Interface. —

- An Interface can Extend another Interface.
- Similarly in a way that a one class can extend another class but we can extend more than one Interface in Java Because of which we can achieve multiple Inheritance in Java in Interface
- when we extend one Interface to another Interfac

i.s. final _ static synchronized, static [illegible] → ✓ 16/12/23 it is possible
we can not inherit the final method

the extended interface becomes the child interface
- And the child interface can inherits the child or members from the parent interface.

Ex.—

```
interface Parent
{
    void m1();
}
interface child extends Parent
{

}
class Driver12
{
    p.s.v.m(s.[]a)
    {
    }
}
interface operations
{
    int addition( int a, int b);
    int substraction( int a, int b);
    int multiplication( int a, int b);
    int division( int a, int b);
    int modulus( int a, int b);
    int square (int a);
}
class arithmatic implements operations1n
{
    public int addition (int a, int b)
    {
```

```
s.o.pln(" Addition is :"+(a+b));
return a+b;
}

P: int substraction(int a, int b) {
        s.o.pln.("substraction is :"+(a+b));
        return a-b;
}

P. int multiplication (int a, int b) {
        s.o.pln.("multiplication is :"+(a*b));
        return a*b;
}

p. int division (int a, int b)
{
        s.o.pln.("Division is :"+(a/b));
        return a/b;
}
```

* multiple Inheritance in Interface. —

- Java class can extends only one class because of which multiple Inheritance is not possible.
- An Interface in Java can extend more than one parent Interface.
- extends keyword is used once and the parent Interfaces are declared and separated useing comma.

* Ex. —

```
interface Parent1
{

}
interface Parent2
{
}
interface Parent3
{
}
interface child extends Parent1, Parent2, Parent3
{

}
```

* multiple Inheritance is possible in Interface because

1) non-static methods are abstract, hence there is no ambiguity.
2) static methods of an Interface can not be Inherited that's why no ambiguity.

3) Interface does not have any cons. in it that's why there is no ambiguity (diamond Problem)

4) default methods with same signature present in multiple parent ~~class~~ interface will be inherited in child interface then it is mandatory to overide that default method in child interface.

i) In these case if we need to access the parent default method we need to use parent

- parent interface reference name. super. meth called statement;

Ex.- Parent1. super. m1();

1) Ex. —

```
interface Parant1
{
    void m1();
}
interface Parent2
{
    void m1();
}
interface child extends Parent1, Parent2
{
}
Class Imp implements child
{
    public void m1()
    {
        s.o.pln ("IMC");
    }
```

ex.-1) Public void m1();
by default

ex.-2) default void m1()
by default.

```
class Driver12
{
    public static void main(S[] a)
    {
        s.o.pln ("Hello");
        Imp obj = new Imp();
        obj.m1();
    }
}
```

o/p. ⟶

2)
```
interface parent
{
    p.s.v. m1()
    {
        s.o.pln ("Hello from parent");
    }
}
interface parent2
{
    p.s.v. m1()
    {
        s.o.pln ("Hello from parent2");
    }
}
interface Child extends Parent1, Parent2
{

}
```

= private
= default

```
class Driver12
{
    p.s.v.m.(s[]a)
    {
        s.o.pln.("Hello");
        Parent1.m1();
        parent1.m1();
    }
}
```

O/P. →

3)
```
interface Parent1
{
    default void m1()
    {
        s.o.pln.("Hi from parent1 m1()");
    }
}
interface Parent2 {
    default void m1()
    { s.o.pln.("Hi from Parent2.m1()"); } }
interface Child extends Parent1, Parent2 {
  default void m1() {
        s.o.pln.("Hello from child m1()");
        parent1.super.m1();
        Parent2.super.m1();
    }
}
```

interface are by default static/ ~~in~~ nature

Difference between abstract class &

abstract class	Interface.
1) An abstract class must be declared with abstract modifier.	1) An Interface must be declared with an Interface keyword
2) An abstract class can have abstract & concrete methods	2) An Interface can have only abstract method before 1.8 V.
3) abstract class does not support multiple inheritance.	3) Interface supports multiple inheritance
4) abstract class can have static, non-static, final & non-final, variable.	4) Interface can have only static, final variable.
5) abstract class can be extended by using extends keyword.	5) an Interface can be implimented using implements keyword.
6) It can have class members like public, private, protected & default.	6) members are public by default
7) An abstract class can have a constructor.	7) An Interface does not have any constructor.

- non-s -"- / is by default abstract in interface
- S.b X →

18/12/23.

- non static block X → ~~class interface~~ is abstract & we cannot create the object.

8) abstract class can have Initializers.

8) An Interface cannot have Initializers

* Object Class —

- Object class is derived in java.lang package
- Object class sits at the top of class Hiraclie
- Every class present in java has a direct or Indirect relation with the object class.
- Every class which we create's will inherit the members from object class.
- Object class is the super class for all classes such as string, stringbuffer, stringbuilder, exceptio wrapper class, etc.

* Constructor of Object class —

```
public class object
{
    public object
    {
    }
}
```

wait()
wait(long ms)
wait(long ms, int ns)

F = Finalize
E = Equals to
T = To string
] non-static

C = clone
H = Hashcode
] final native

G = get class] native fin

notify()
notify all()

* methods of object class —

- public final native java.lang.Class <?> getClass();
- public native int hashCode();
- public boolean equals (java.lang.object);
- protected native java.lang.object clone() throws

class ⓞ Hashcode
hexadecimal

- java.lang.CloneNotSupportedException;
- public java.lang.String toString();
- public final native void notify();
- public final native void notifyAll();
- public final void wait()
- public final void wait(long)
- public final void wait(long, int)

1) finalize()

the finalize method is called by the Garbage collector on an Object when the Garbage collector determines an object which does not have any reference.

* Note:—

- It is not recomended to Override the finalized method because it is debricated from JDK 1.9

2) wait()

- The wait methods in object class is an example of wait to method Overloading
- These methods are used in multithreading for interthread communication.
- It causes the current thread to wait for a specific period of time based on the arguments until the another thread invokeds them using notify(); or notifyall();

- native method are abstract.

3) notify()
notify wakes up a single thread that is waiting for its the execution of on that object.

4) notify all()
notify all wakes up(notified) all the threads which are waiting for the execution on that object

5) get class()
The get class method is used to get the run time class name of the perticular object.

Ex - get class()

```
import java.util.Scanner;
class Demo
{
    p.s.v.m(S[]a)
    {
        Scanner sc = new Scanner(system.in);
        S.o.pln.(sc.getclass());

        String name = "Maari";
        s.o.pln.(name.getclass());

        Demo obj = new Demo();
        s.o.pln.(obj.getclass());

        int[] arr = new int[10];
        s.o.pln.(arr.getclass());
    }
}
```

o/p. ⟶

* clone() —

The clone method is used to create an exact copy of an object if we try to create the clone of an object using new keyword it will take a lot of processing time that's why we use object cloning.

* Advantages of using clone method —

1) we don't need to write lengthy & repetative code.
2) It is easiest & most efficient way to create clone of an object.
3) fastest way to create an clone of an object (like array).

steps to clone an object. —

1) we need to implements clonable interface.
2) we need to overide the clone method & need to handle clone not supported exception.
3) Inside the clone method we can provide our own implimentation or we can use a statement

super.clone

Ex. —

```
class Student implements clonable
{
    string name;
    string ntuplace;
    long phon;
    string branch;

    Student(string name, string ntuplace, long phon,
            string branch)
    {
        this.name = name;
        this.ntuplace = ntuplace;
        this.phon = phon;
        this.branch = branch;
    }
    protected object clone() throws CloneNotSuppo
            rted Exception
    {
        return super.clone();
    }
}
class cloneExample
{
    p.s.v.m(s[]a) throws cloneNotSupportedExcepti
    {
        Student obj1 = new Student("Raumasi", "Malegaon
            gon 207889L, "Computer");
        student obj2 = (Student) obj1.clone();
```

hexadecimal int type

```
        s.o.pln.(obj1.name);
        s.o.pln.(obj2.name);
        obj1.name = "Patil";
        s.o.pln.(obj1.name);
        s.o.pln.(obj2.name);
    }
}
```

o/p → Mausi
Musi
Patil
Mausi

* To string () —
- The to string method is used to display a string representation of an object like class name@hashcode in (hexadecimal)
- It is always reomended & it is a good practice to overide to string method.

EX. —

```
class Student implements Cloneable
{
    string name;
    string ntuplace;
    long phno;
    string branch;
}
student (string name, string ntuplace, long phno,
        string branch)
```

```
{
        this.name = name;
        this.ntuplace = ntuplace;
        this.phno = phno;
        this.branch = branch;
}
public string toString()
{
        return ("my name is " + this.name + ", I belongs to " + this.
ntuplace + " and my phone no - " + this.phno + " branch is
" + this.branch);
}
Class CloneExample
{
        p.s.v.m(S[]a)
    {
        Student obj1 = new Student ("Mani", "Pune", 9021207689 1,
                "Computer");
        s.o.pln(obj1.toString());
        student obj2 = new student ("Pallavi", "Delhi", 29800001,
                "E&TC");
        s.o.pln(obj2);
        string name = new string ("Aushka");
        s.o.pln(name);
        stringbuffer sb=
    }
}
```

* equalls. —

- The equals method is used to compaire the reference of an object and returns true if references are the same or returns false if references are different.
- The equals methods works similar double equals (==) operator.
- It is highly recomanded to overide these equals method

* Ex. —

```
class student implements clonable
{
    string name;
    String ntuplace;
    long phon;
    string branch;

    students (string name, string ntuplace, long
    phon, string branch)
    {
        this.name = name;
        this. ntuplace = ntuplace;
        this. phon = phon;
        this. branch = branch;
    }
    p. boolean equals (object obj)
    {
        student s1 = (Student) obj;
        if ((this name == s1. name) && (this.ntuplace
==s1.ntuplace) && (this. phon == s1. phon) && (this. branch =
```

```
s1.branch))
{
    return true;
}
else
{
    return false;
}
}
}
class cloneExample2
{
    student obj1 = new Student ("Mausi", "Malegaon", 902198
                                "computer");
    Student obj2 = new student ("Gayatri", "Nashik",
                gau42111 0091, "Electrical");
    s.o.pln. (obj.equals (obj2));
}
```

o/p. → false

* Hash Code —

- The hash code method generates the unique hashcode value for every objects.
- The return type of hashcode method is int.

Ex. —

```
class student
{
    static int unique =100001;
    String name;
    String place;
    student (string name, string place)
    {
        this. name = name;
        this. place = place;
    }
    public int hashcode()
    {
        return unique ++;
    }
}
class objectOverRiding
{
    public static void main (s [Ja)
    {
        Student a = new student ("Mansi", "pune");
        student b = new student ("Ankit", "Pune");
        Student c = new Student ("Gaurishi", "Nasik");
        Student d = new student ("Anusheka", "pune");
    }
}
```

```
o/p. ⟶  s.opln.(a.hashcode());
         s.opln.(b.  —"—  );
         —"—  (c.  —"—  );
         —"—  (d.  —"—  );
}
o/p. ⟶
```

method create this exception

20/12/23

Exception Handling. —

Exception Handling is one of the most important feature of java programming that allows us to handle the exception which are caused because of some unwanted event.

Exception. —

- An Exception is unwanted event that occurs during the execution of a program that disturbs the normal termination of a program.
- when an Exception occurs, the program ~~execution~~ execution is terminated abnormally.
- In these case we get a system generated message (which is not at all understandable by our user).
- There is a good thing about exceptions, is that they can be handled.

Note. —

Exception is a class in java.

Why Exception Occurs. —

1) If we are writting a statement which is not logically correct.
2) If we try to open an non-existing file.
3) If we try to provide a bad input data
4) Because of bad network connection, etc.

* Default Exception Handling :—

1) Ex. —

```
class Demo
{
    p. s. v. m (S[]a)
    {
        s.o.pln.("Hello");
        m1();
    }
    p.s. void m1()
    {
        m2();
    }
    p. s.void m2()
    {
        s.o.pln.(101/0);
    }
}
```

o/p. ⟶ Hello

Exception in thread "main" java.lang.ArithmeticException: by zero at Demo.m2(ExceptionExample.java.14)

at Demo.m1(ExceptionExample.java.10)

at Demo.main(ExceptionExample.java.6)

IVpt

thread

main

```
class Demo
{
  p.s.v.m (S[]a)
  {
    s.o.p.("Hello");
    m1();
  }
  p.s.v.m1()
  {
    m2();
  }
  p.s.v.m2()
  {
    s.o.p.(1010);
  }
}
```

Throwable object

- Exception class name.
- Exception Description.
- Stack Entries.

Data structure

1st in Last out

m2()
m1()
main()

stack

Hello

1. stack frame for 1 thread

main
m1()
m2()

Stack

2) Ex. —

```
Class Demo
{
    p.s.v.m (S[]a)
    {
        s.o.pln.("Hello main");
        m1();
    }
    p.s.v.m1()
    {
        s.o.pln.("m1()");
        m2();
```

```
        s.o.pln.(1010);
    }
    p.s.v.m2()
    {
        s.o.pln.("m2()");
        m3();
    }
    p.s.v.m3()
    {
        s.o.pln.("m2()");
    }
}
```

Throwable object

- Exception class name.
- Exception Description.
- Stack Entries

Stack

m3()	m3()
m2()	m2()
m1()	m1()
~~m0~~ Hello main	main()

O/P. →

Hello main

m1()

m2()

Exception in thread "main" java.lang.ArithmeticException: / by zero at Demo.m1(ExceptionExample.java:12)

at Demo.m2(ExceptionExample.java:6)

3) Ex. —

```
Class Demo
{
p.s.v.m(S[]a)
{
  s.o.pln.("Hellomain");
   m1();
  s.o.pln.(10/0);
}
p.s.v.m1()
{
  s.o.pln.("m1()");
  m2();
}
p.s.v.m2()
{
  s.o.pln.("m2()");
}
p.s.v.m2()
{
  s.o.pln.("m2()");
}
```

Throwable object

Exception class name
Exception Description
Stack entries

m3() ←	m3()
m2() ←	m2()
m1() ←	m1()
Hellomain	main

o/p. → Hello main

Exception in thread "main" java.lang.ArithmaticException

/by zero

at. Demo.main.(Demo.java:14)

* Default Exception Handling in Java. —

1) At the time of execution, whenever an exception occur insight a method, ~~or~~ that method is responsible to create an exception object.

2) the object ~~is~~ created by the method is of throwable type. & that object is an exception object. ~~which~~

3) Object is transfered or given to the run time system (JVM).

4) That Exception object Includes. —

4.1) exception class name.

4.2) Exception Description.

4.3) Stack trasing (~~sta~~ method names where exception object flows or responsible).

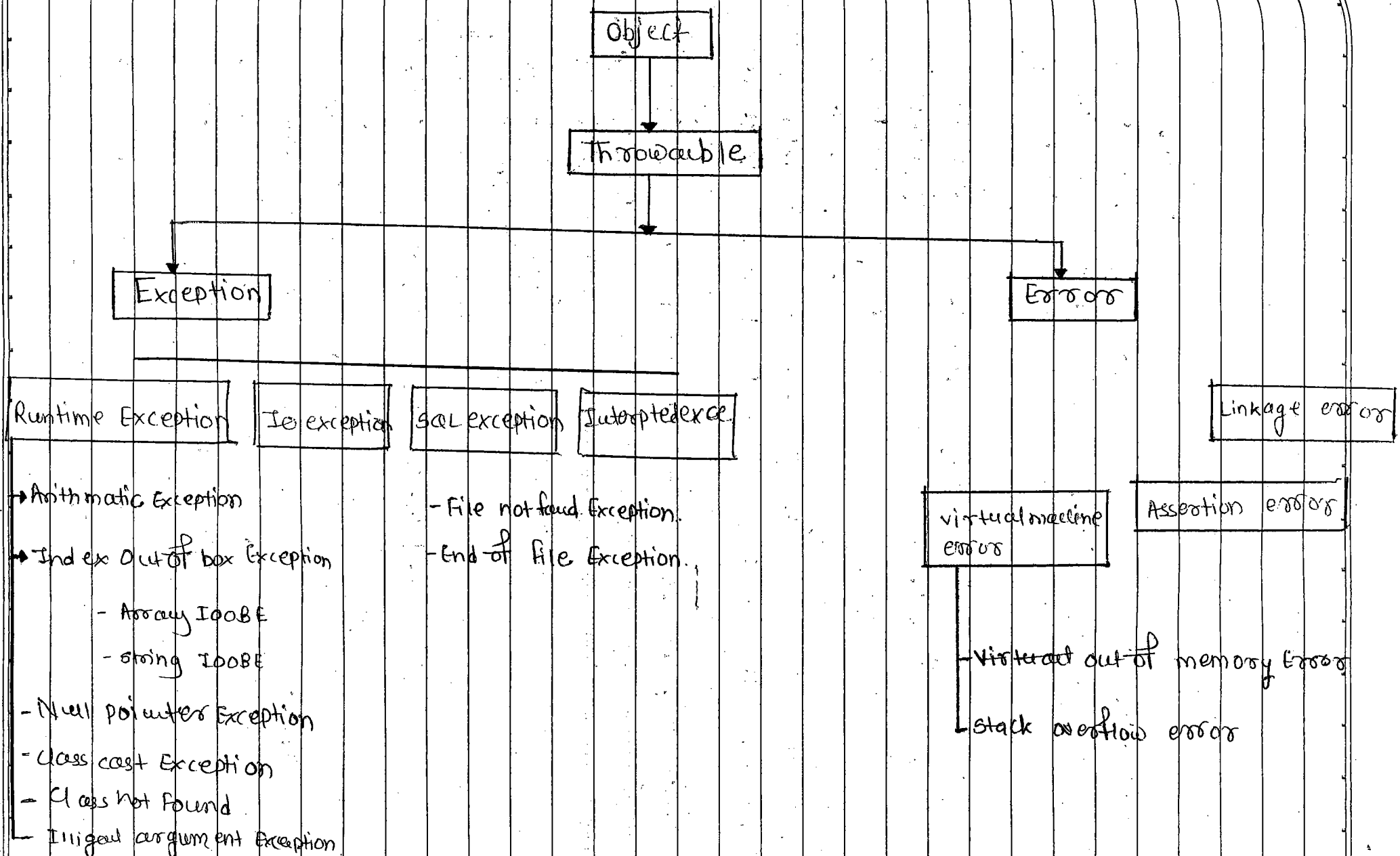
5) JVM will check wether the object which is created by the method has the exception handling code or not.

6) If the method does not have the Handling code JVM goes to the calling method & checks for it, respectively.

7) If ~~any of~~ none of the method has the exception handling code in that case JVM gives that object to <u>default Exception Handler.</u>

* Note. —

If the exception object is handled by the default exception handler always it will be abnormal termination.


Object
Throwable
Exception
Error
Runtime Exception
Io exception
SQL exception
Interrupted exce
Linkage error
Arithmatic Exception
Index Out of box Exception
- Array IOOBE
- string IOOBE
- Null pointer Exception
- Class cast Exception
- Class not Found
- Illigal argument Exception
- File not found Exception.
- End of file Exception.
virtual machine error
Assertion error
- Virtual out of memory Error
- Stack overflow error

21/12/23

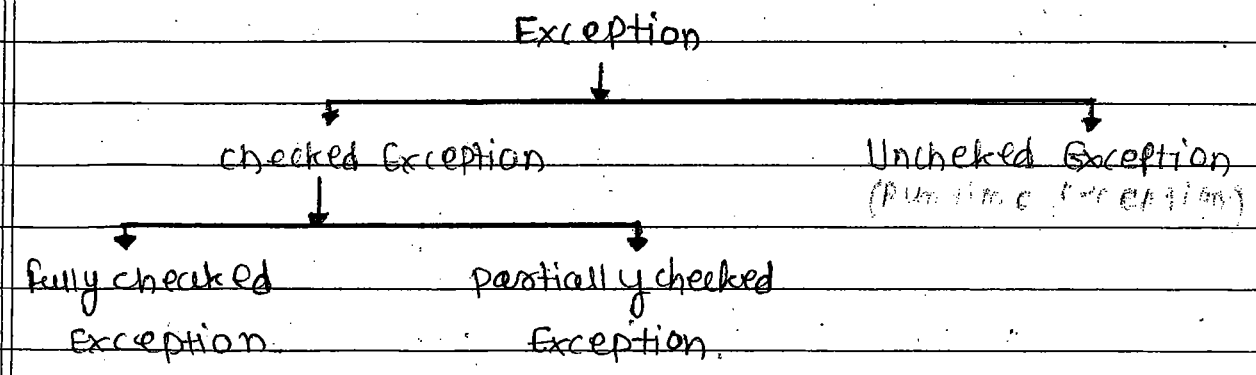
* Exception	Error
1) An issue in a program that prevents the normal flow of execution is known as Exception.	1) An Intigation of an unxpe condition that occurs du to lack of system resources is known as error.
2) It occurs because of an issue of an program	2) It occurs because of lack of system resourc
3) They are recoverable (can be handled)	3) They unrecoverable (cannot be Handled)
4) It can be classified as checked & uncheked exception.	4) It is classified as unchcked exception.
5) It is possible to handle exception in a program using keywords such as ~~throw~~, try, catch, finally & throws	5) There is no way to handle the error in a program.
6) Ex. — Arithmatic Exception. ~~Cls~~ class cast Exception. Null pointer Exception. class not found.	6) Ex. — Assertion Error. Linkage Error. virtual Machine Erro stack overflow Erro

* Exception Handling:-

- If a exception occurs which is not handled by the programmer then program execution get's terminated abnormally, to avoid this abnormal termination we need to create an alternative way for a normal termination of a program.

- The message which displayed when an exception occured is not user friendly.
- Our user is unable to understand what went wrong in order to let them (user) know the reason why this message or an unwanted event occured.
- We need to handle exceptions to display a user friendly warning message.
- It is also used to do some cleanup task before the program gets terminated abnormally. (cleanup task like - closing of object, closing connections, streams, servers).

Exceptions in java can be classified into



* Checked Exception.—

- The Exception cheked by the compiler at the Compile time for a normal termination of a progr at runtime is known as checked Exception.

Note.—

- Compiler is not aware about the exception but compiler ~~checks the programmer Handled has an exception~~ checks whether the programmer has handled the exception for that unwanted state

- All Exceptions other than run time Exception are known as checked exception

As compiler checks them during compile time for a smooth Execution or a normal Execution / Terminal

1) Ex.—

```
Class Demo
{
    p.s.v.m.(S[]a)
    {
        Thread.sleep(30000);
    }
}
```

o/p. ⟶ unreported exception Interrupted Exception

2) Ex.—

```
import java.io.*;
Class Demo
{
    p.s.v.m(S[]a)
```

```
}
    FileReader a = new FileReader("demo.txt");
}
}
```

* checked exception can be classified into two types.

1) fully checked Exception —

- In fully checked Exception both parent & child class is cheked by the compiler
- e.g. — Exception & io Exception.
  ~~Throwable~~ &
  Exception & Interrupted Exception.

2) Partially checked Exception. —

- In partially checked Exception only the parent class is checked by the compiler.
- e.g. — Exception & Runtime Exception
  Throwable & Exception.

* Uncheeked Exception. —

- The exceptions which are checked or caught at run time is known as Uncheked Exception
- The uncheked exception are also known as Run time Exception.
- This Exceptions are not checked at compile time so compiler is not aware about it.
- but it is the responsibility of a programmer to

handle them for a normal termination of a program

1) Ex. —

```
Class Demo
{
    p.s.v.m. (S[]a
    {
        s.o.pln. (1010);
    }
}
```

2) Ex. —

```
Class Demo
{
    static Demo obj;
    String name = "Manni";
    P.s.v.m (S[]a)
    {
        s.o.pln. (obj.name);
    }
}
```

o/p → Null pointer Exception.

3) Ex. —

```
import java.util.Scanner;
class Demo
{
    p.s.v. m.(S[]a)
    {
        Scanner sc = new scanner (s.in);
        int num = sc.nextInt();
        s.o.pln. (num);
    }
```

8.

o/p. → user → Mansi
InputMismatchException

* Exception Handling using try catch block —

* try —
- try is a keyword in java
- the try block contains the set of statements because of which an exception is occured might occured.
- The Number of statement in a try block must be minimum or less ~~less~~
- because the exception is occured the next statements after the exceptions will be ignored inside the try block.
- A try block is followed by the catch block or finally or catch and finally block which helps to handle the exception occured

* Syntax. —

```
try
{
    // statements because of which an exception
       might occured.
}
```

* catch —
- ~~try~~ catch is a keyword in java.
- A catch block is where we can handle exception.
- this catch block must be followed by another catch block or try block.

- A single try block can have several (multiple) catch blocks associated with it.
- by creating multiple catch blocks we can catch different exceptions in it.

* syntax. —

```
catch(exception class name variable)
{
    // except statements which are going
       to handle the exception.
}
```

combine syntax for try & catch block —

```
try
{
    // statements because of which
       an exception might occured.
}
catch(exception class name variable
{
    // statements which are going to handle
       the exception.
}
```

* Normal termination of a program by handeling the Exception. —

```
class ExceptionExample1
{
```

```
p.s.v.m.(S[]a)
{
    s.o.pln.("Execution Start");
    try
    {
        s.o.pln.(1010);
    }
    catch (ArithmaticException Handled i)
    {
        s.o.pln.("Arithmatic Excuption Handled");
    }
    s.o.pln.("Execution Ends");
    }
}
```

o/p. → Execution Start
Arithmatic Excuption Handled
Execution Ends

* Abnormal termination of a program. —

```
class Demo
{
    p.s.v.m(S[]a)
    {
        s.o.pln.("Execution Starts");
        s.o.pln.(10/0);
        s.o.pln.("Execution Ends");
    }
}
```

O/p. →

* multiple catch block. —

1) A single try block can have any number of catch block.
2) A Generic catch block (parent type) can handle all types of Exceptions Either it is runtime exception or any other type of Exception.
3) If no exception occurs in try block then the associated catch blocks are completely ignored.
4) corresponding catch block executed for the specific type of exception.

4.1) catch (ArithmaticException m)

this type of catch block can only handle arithmatic exception.

4.2) catch (NullPointerException mansi)

this type of Exception can only handle NullPointerException.

4.3) catch (RunTimeException patil)

this type of catch block can handle any type kind of Runtime exception like AE, NPE, CCE.

child ✓ ↓ x child.

* Control Flow of try & catch block. —

1) Ex. —

```
class Exception
{
    p.s.v.m.(s[]a)
    {
        s.o.pln. ("Execution starts");
        try {
            s.o.pln. (1010);
        }
        catch (ArithmaticException i){
        S.o.pln. ("Ari. Exce. handled");
        }
        try {
            Thread.sleep (1000); }
        catch (InterruptedException ie)
        {
            s.o.pln. ("Interr. Exce. handle");
        }
        s.o.pln. ("Execution Ends");
    }
}
```

o/p. → Execution starts
ArithmaticExceiption handled
Execution Ends.

2) class Exception
{
    p.s.v.m.(s[]a)

```
{
    s.o.pln. ("Execution starts");
    try
    {
        Thread. sleep(1000);
    }
    catch( ArithmeticException i){
        s.o.pln. ("Arithmetic Exception handle");
    } catch (InterruptedException ie)
    {
        s.o.pln. ("Interrupted Exception handle");
    }
    s.o.pln. ("Execution Ends");
}
```

o/p. ⟶ execution start.
execution end.

3) Class Exception

```
{
    p.s v. m.(s[]a)
    {
        s.o.pln. ("Execution Starts");
        try {
            s.o.pln. (1010)
        } catch ( ArithmeticException ae)
        {
            s.o.pln. ("ArithmeticException handle");
        }
```

```
catch (RuntimeException x)
{
    S.o.pln.("RuntimeException handle");
} catch(Exceptione)
{
    s.o.pln.("Exception handle");
} catch(Throwable t)
{
    s.o.pln.("Throwable handle");
}
s.o.pln.("Execution ends");
}
}
```

o/p. ⟶ Execution start
Arithm. Excep. Handled
Execution ends.

4) Class Exception

```
{
    p.s.v. m(S[]a)
    {
        s.o.pln.("Execution starts");
        try
        {
            s.o.pln.(10/0);
        }
        catch(RuntimeException x)
        {
            s.o.pln.("RuntimeExce handle");
        }
```

```
catch (ArithmeticException ae)
{
    s.o.pln. ("Arithmetic Exception handle");
}
s.o.pln. ("Execution Ends");
}
}
```

o/p. → error. - exce. prith has already been caught.

//catch (ArithmeticException c)

5) Class Exception

```
{
public s. v. m (S[]a)
{
    s.o.pln. ("Execution starts");
    try
    {  Thread. sleep (2000);
    }
    catch (Runtime Exception re)
    {
        s.o.pln. ("Runtime Exception handle");
    } catch (Arithmetic Exception ae)
    {
        s. o. pln. ("Arithmetic Exception handle");
    } catch (InterruptedException ie)
    {
        s. o. pln. ("InterruptedException handle");
    }
    s. o. pln. ("Execution Ends");
} }
```

o/p. → exception ArithmeticExc. has already been caught
// catch(Arith.Exc. m)

6) class Exception
{
p.s.v.m.(s[]a)
{
s.o.pln.("Execution starts");
try{
s.o.pln.("1010");
}
catch(RuntimeException re)
{
s.o.pln.("RuntimeException handle");
} catch(InterruptedException ie)
{
s.o.pln("InterruptedException handle");
}
s.o.pln.("Execution Ends");
}
}
o/p. → ~~error~~. Exception starts
1010
Execution Ends

7) class Exception
{
p.s.v.m(s[]a)

```
s.o.pln. ("Execution Starts");
try {
        Thread.sleep (2000);
    }
    catch (ArithmeticException ae)
    {
        s.o.pln. ("Arithmetic Exception handle");
    } catch (Exception ie)
    {
        s.o.pln. ("Interrupted Exception handle");
    }
    s.o.pln. ("Execution s ends");
  }
}
```

O/P. → Exec. Starts
Interrupte Exce.
Excep. ends.

8)
```
class Exception
{
    p.s.v.m. (S[] a)
    {
        s.o.pln. ("Execution Starts");
        try {
            s.o.pln. ("Inner try");
            try {
                s.o.pln. (10/0);
            }
            catch (NullPointerException npe)
            {
```

Exception occurred in a catch block it behave like a exception occurred in any other block

```
        s.o.pln.("NullPointerException handle");
      }
    }
    catch(Exception e)
    {
        s.o.pln.("Exception handle");
    }
    s.o.pln.("Execution Ends");
  }
}
```

O/p. ⟶ Exception starts
in nestry
Exception handle
Execution ends.

(11) Ex —

```
import java.io.*;
class Exception
{
    p.s.v.m.(s[]a)
    {
        try
        {
            s.o.pln.(10/0);
        } catch (RuntimeException re)
        {
            s.o.pln.("Runtime Exception handle");
        }
        catch (IOException io)
        {
```

RE handled

block → yes it is only a comment
= in if else we can write a comment.

```
        s.o.pln. ("IOException Handle");
        }
    }
```

O/p. → ~~error: Exception IOE has already been caught~~
error: exception IOException is never thrown in body of corresponding try statement.

2)
```
import java.io.*;
Class Exception
{
    p.s.v.m.(S[]a)
    {
        try {
            s.o.pln. ("outer try");
            try {
                s.o.pln. ("inner Try");
                s.o.pln (10/0);
            }
        catch (IndexOutofBoundsException iobs)
        {
            s.o.pln ("IndexoutOBound handle");
        }
    } catch( Arithmatic Exception ae)
    {
        s.o.pln. ("Arithm. Exec. ae");
    }
}
```

O/p. → outer try
Inner try
Arithm. Exec. ae

3)
```
import java.io.*;
class Exception
{
    p.s.v.m.(S[]a)
    {
      try{
            s.o.pln.("Outer Try");
          try{
                s.o.pln.("Inner try");
                s.o.pln.(10/0);
            }
            Catch (Exception i)
            {
                s.o.pln.("Inner catch");
                s.o.pln.("Exception handle");
            }
            s.o.pln.(10/0);
        }
        catch (ArithmeticException ae)
        {
            s.o.pln.("Outer catch");
            s.o.pln.("Arithmetic Exce. ae");
        }
    }  }
```

o/p. → Outer Try Exception handle
Inner Try Outer catch
Inner catch Arithmetic Exce ae

4)
```
import java.io.*;
class Exception
```

```
{
    p.s.v.m.(S[]a)
    {
        try {
            s.o.pln.("outer try");
            try {
                s.o.pln.("inner try");
                s.o.pln.(1010);
            }
            catch (Exception iobe)
            {
                s.o.pln.("Inner catch");
                s.o.pln.("Exception handle");
                s.o.pln.(1010);
            }
        }
        catch (ArithmeticExce ae)
        {
            s.o.pln.("outer catch");
            s.o.pln.("Arith. ae");
        }
    }
}
```

o/p. → Outer try
Inner try
Inner catch
Exception handle
Outer catch
Arith. ae

(5)

```
class Exception
{
    p.s.v.m.(s[]a)
    {
        try{
            s.o.pln.("Outer try");  1
            try{
                Thread.sleep(2000);
                pln.("abcd");  2
            }
            catch(InterruptedException ie)
            {
                s.o.pln.("Inner catch");
                s.o.pln.(1010);
            }
        }
        catch(ArithmaticException ae)
        {
            s.o.pln.("Outer catch");
            s.o.pln.("Arith m. Exce ae");
        }    }
}
```

O/p. → Outer try
(after 2 sec)
abcd.

6) class Exception

```
{
    p.s.v.m.(s[]a)
    {
        try {
            s.o.pln.("outer try");
            try {
                FileReader obj = new FileReader ("ramesh.tx
            }
            catch (FileNotFoundException ie)
            {
                s.o.pln.("Inner catch - ramesh");
                s.o.pln.("FileNotFound Excuption handle");
                s.o.pln.(1010);
            }
        }
        catch (ArithmeticException ae)
        {
            s.o.pln.("outer catch");
            s.o.pln.("Arithmetic exc: ae");
        }
    }
}
```

o/p. →

7) Class Exception

```
{
    p.s.v.m.(S[]a)
    {
        try{
            s.o.pln.("outer try");
            s.o.pln.(1010);
        try{
            s.o.pln.("inner try");
            s.o.pln.(1010);
        }
        catch(ArithmeticException ie)
        {
            s.o.pln.("inner catch");
        }
    }
    catch(ArithmeticException ae)
    {
        s.o.pln.("outer catch");
    }
}
catch(ArithmeticException ae)
{
    s.o.pln.("outer catch");
    s.o.pln.("ArithmeticExce. ae");
}
}}
```

O/P. →

(8)

```
class Exception
{
    p.s.v.m. (S[]a)
    {
        try {
            s.o.pln. ("outer try");
            try {
                s.o.pln. ("inner try");
                s.o.pln. (1010);
            }
            catch (ArithmeticException ie)
            {
                s.o.pln. ("inner catch");
                Thread.sleep (2000);
                s.o.pln. ("abcd");
            }
        }
        catch (Exception ae)
        {
            s.o.pln. ("outer catch");
            s.o.pln. ("Arithmetic Excep. ae");
        }
    }
}
```

o/p → outer try
inner try
~~Arithmetic Excep~~ inner catch
(pause for 2sec)
abcd

9)

```
class Exception
{
    p.s.v.m(S[]a) throws InterruptedException
    {
        try {
            S.o.p.ln.(1010);
        } catch (ArithmeticException ae)
        {
            S.o.p.ln.("ArithemeticException ae");
            S.o.p.ln.(1010);
        }
        Catch (Exception ae)
        {
            S.o.p.ln.("Arithmetic Exception 2");
        }
    }
}
```

O/p. → ArithmeticException ae
Exception in ~~main~~ thread "main"

* finally block —

- finally is a keyword in java.
- A finally contains all the crucial statement that must be executed wether an exception occurs or not
- finally block follows the catch block or try block
- The crutial statements such as closing a connecti object, stream, database, etc.

* Syntax — finally
{
// crutical statements which needs to get executed.
}

Ex. —

1) class Demo
{
p.s.v.m.(s[] a){
s.o.p ln.("Db can open");
try{
s.o.p ln.(10/0);
}
catch (ArithmaticException ae)
{
s.o.p ln.("Arith. Exce. handled");
}
finally
{
s.o.p ln.("DB conn. closed");
}
} }

O/p. → Db conn. open.
Arith. Excep. Handled.
Db conn closed.

2)
```
Class Demo.
{
    p.s.v.m.(s[]a)
    {
        s.o.pln.("Db conn. open");
        try{
            s.o.pln.(10/0);
        }
        finally
        {
            s.o.pln.("DB conn. closed");
        }
}
```

O/p. → Db conn open
DB conn. closed
Exception in thread "main".

* Characteristics of finally. —
- A finally block is associated with try block, we can not use a finally block without a try block
- finally block is optional because try & catch block is sufficient to handle the exception.
- However if we have a specified a finally block it will always get's ~~exec~~ excicuted after the execution of try & catch block.

- In a normal case when there is no exception in the block then the finally block gets exicuted directly by skipping the catch ~~bot~~ block.
- However, the exception occurs then the catch block gets exicuted before finally block.
- If an exception occurs in a finally block it behave exactly like any other exception ~~or~~ which occurs in any of the block.

Imp.- The Statement present in finally block get's exicuted even if the try or catch block contains control transfered statement like return, break or continue.

* In what all circumstances the finally block doesn't get exicute.

1) Death of a thread — when we close program forcefully
2) when we use sys.exit()

* final	finally	finalized
1) final is a keyword & a non-access modifier	finally is a block used in exception handling.	finalized is a method present in Object class.
2) It is used to restrict the Inheritance of a class, Overiding of a method & reinitiation of a variable	It is used in exception handling to excicute some critical statements either an exception occurs or not.	It is used to do the clean up procedure Immidiatly whenever an object is eligible for garbage collection.
3) It is only applicable to class, method & variable	finally block is associate with try & catch block.	finalized method is applied to objects.
4)	After try catch block finally block get excicuted.	Just before the object distroyed the finalized method is called.

10)

```
class FinallyBlockExample
{
    p.s.v.m.(s[]a)
    {
        try {
            s.o.pln.("1010");
        }
        catch(ArithmeticException ae)
        {
            s.o.pln.("ArithmeticException handle");
        }
        finally
        {
            s.o.pln.("hello from Finally");
        }
}  }
```

O/p. → 1010
hello from Finally.

11)

```
class ~~Ex~~ FinallyBlockExample
{
    p.s.v.m.(s[]a)
    {
        s.o.pln.("main starts");
        m1();
        s.o.pln.("main ends");
    }

    p.s.v.m1()
    {
        try {
```

```
        s.o.pln.(10/0);
    } catch (ArithmeticException ae)
    {
        s.o.pln.("ArithmeticException handle");  2
        return;
    }
    finally
    {
        s.o.pln.("hello from finally");  3
    }
} }
```

o/p. → main starts
ArithmeticException handled
hello from finally.
main ends

12) class FinallyBlockExample

```
{
    p.s.v.m.(s[]a)
    {
        s.o.pln.("main starts");  1
        m1();
        s.o.pln.("main ends");
    }
    p.s.v.m1() {
    try
    {
        s.o.pln.(10/0);
    }
    finally
    {
```

```
        s.o.pln.("hello from Finally");
    } }
}
```

O/P → main starts
hello from finally
Exception in thread "main".

13)

```
class FinallyBlocksExample
{
    p.s.v.m.(S[]a)
    {
        s.o.pln.("main starts");   1
        m1();
        s.o.pln.("main ends");
    }
    p.s.v.m1()
    {
        Mansi;
        for (int i=1; i<=10; i++)
        {
            try {
                s.o.pln.("try"+i);   2
                s.o.pln.(1010);
            }
            Catch (ArithmeticException ae)
            {
                s.o.pln.("Arithmetic Exception ae");   3
                if(i==3)
                {
                    break mansi;
                }
```

```
}
finally
{
    s.o.pln.("hello from Finally "+ i);
} }
    }
}
```

o/p → main starts

try 1
ArithmeticException
hell from finally 1
try 2
ArithmeticException
hello from finally 2
try 3
ArithmeticException
hello from finally 3.
main ends.

14) class Demo

```
{
    p.s.v.m.(S[]a)
    {
        s.o.pln.("main Starts");
        m1();
        s.o.pln.("main ends");
    }
    p. s. void m1()
    {
        try
        {
            s.o.pln.("try");
            s.o.pln.(10/0);
        }
```

```
catch (Arithmetic Exception ae");
System.Exit(0);
}
Finally
{
    s.o.pln. ("hello from Finally");
}
}}
```

o/p. ⟶ main start
try.

```
19) class Demo
{
    p.s.v. m(S[]a)
    {
        try {
            s.o.pln. ("outertry");
            try {
                s.o.pln. ("Inner Try");
                s.o.pln. (1010);
            }
        } finally {
            s.o.pln. ("Innerfinally");
            }
        } catch (Arithmetic Exception a)
        {
            s.o.pln. ("Arith. Exce. Handled");
            s.o.pln. ("outer catch");
        } finally {
            s.o.pln. ("Outer finally");
    }
```

25/12/23.

```
    }
}
```

O/p. → Outer try Arith. Exce. Handled
Inner try outer catch
Inner finally outer finally.

2)

```
Class Demo
{
    p.s.v.m(S[]a)
    {
        try {
            s.o.pln.("Outer try");
            try {
                s.o.pln.("Inner try");
                s.o.pln.(1010);
                return;
            }
            finally {
                s.o.pln.("Inner finally");
            } catch (ArithmeticException a)
        {
            s.o.pln.("ArithmeticException Handled");
            s.o.pln.("Outer catch");
        }
        finally {
            s.o.pln.("outer finally");
    }  }  }
```

O/p. → Outer try ArithmeticException a Handled
Inner try outer catch.
Inner finally

3)
```
class Demo
{
    p.s.v.m.(SCJa)
    {
        try{
            s.o.pln.("outer try");
            try{
                s.o.pln.("Inner Try");
                s.o.pln.(1010);
            }
        catch (ArithmeticException a)
        {
            s.o.pln.("Inner catch");
            s.o.pln.(1010);
        }
        finally{
            s.o.pln.("Inner finally");
        }
    } catch (ArithmeticException a)
    {
        s.o.pln.("ArithmeticException Handled");
        s.o.pln.("outer catch");
    }
    finally{
        s.o.pln.("outer finally");
    }  }
}
```

O/p → outer try
inner try
~~ArithmeticException~~
inner catch
inner finally
ArithmeticException Handled
outer catch
outer finally

4)
```
class Demo
{
    p.s.v.m(S[]a)
    {
      try {
            s.o.pln("Try");
            s.o.pln(10/0);
      } catch (ArithmeticException a)
      {
          s.o.pln("ArithmeticException Handled");
          s.o.pln(10/0);
      }
      finally {
            s.o.pln("finally");
      }
}  }
```

O/p. → Try
ArithmeticException Handled.
Finally
Exception in thread main.

5)
```
class Demo
{
    p.s.v.m(S[]a)
    {
      try {
            s.o.pln("Try");
            s.o.pln(10/0);
      } catch (ArithmeticException a)
      {
          s.o.pln("ArithmeticException Handled");
```

exception that we write in a catch block is get ignore / cannot hand by the another catch block.

```
        s.o.pln.(10/0);  ⟶ these exception is not handle
}                          by the Exception a cause here we are
finally{                   not using nesting try & catch block
        s.o.pln.("finally");
        try{
            s.o.pln.("Inner try");
        } catch(Exception a)
        {
            s.o.pln.("Inner catch");
        }
}}
```

O/p. ⟶ try
Arithmetic Exception Handled
finally

Inner try
Exception in main thro

```
class Demo
{
    p.s.v m(S [Ja)
    {
        try{
            S.o.pln.("Try");
            s.o.pln.(10/0);
        } catch(Arithmetic Exception a)
        {
            S.o.pln.("Arithmetic Exception Handled");
            s.o.pln.(10/0);
        }
        catch(Exception a)
        {
            s.o.pln.("outer catch");
```

```
}
finally {
        S.o.pln. ("finally");
        try {
            s .o. pln. ("Inner try");
            } catch (Exception a)
            {
            s. o. pln. ("Inner catch");
            }
        }
} }
```

O/p ⟶ try.
Arithmetic Exception handled.
inner try.
Exception in thread "main".

4) class Demo

```
{
    p. s. v. m (S[]a)
    { try {
            S.o.pln.("Try");
            Thread.sleep (2000);
        } catch (Arithmetic Exception ae)
        { s.o.pln. ("Arith. Handled");
        }
        catch( InterruptedException e) {
            s.o.pln. ("Inte. Exce. Handle");
        }
        finally {
            s.o. pln. ("inner try");
            s.o.pln. (10/0); } catch (Exception e) {
} } }                                s.o.pln.("Inner catch"); }
```

* Throwable. —

- Throwable is root class for exceptions in Java. all
- exception classes are derived from these throwable class.
- all the exception objects ~~are~~ which are ~~executed~~ thrown will be of throwable type
- There are two main subclass of Throwable
  1) Exception.
  2) Error.

methods of throwable

1) getMessage()
2) printStackTrace()
3) toString() (overridding method).

* toString() . —

to string method is a overridding method in throwable class which is used to display the exception class name & Exception discripti

Ex. —

```
class ThrowableExample
{
    P.S.V.M. (String)
    {
        try{
            S.O.println(10/0);
        }
        catch (ArithmeticException ae)
```

```
{
        s.o.pln.(ae);
        s.o.pln.(ae.toString());
}
}
```

O/p. ⟶

* getMessage() —

The getMessage() method is used to get an discription of an exception.

Ex. —

```
class Demo
{
    p.s.v.m.(s[]a)
    {
        try {
            s.o.pln.(1010);
        }
        catch(ArithmeticException ae)
        {
            s.o.pln.(ae.getMessage());
        }
        s.o.pln.("********");
        try {
            String a = null;
            s.o.pln.(a.isEmpty());
        }
        catch(NullPointerException npe)
        {
            s.o.pln.(npe.getMessage());
        }
```

```
    }
}
```

o/p. ⟶ ! byzero
cannot invoke "String.isEmpty()" because "str" is null.

* printStackTrace() —

The printStackTrace method writtens the exception class name, exception discription & stack tracin

Ex. —

```
class Demo
{
    p.s.v.m(S[]a)
    {
        m1();
    }
    p.s.v.m1(S[]a)
    {
        m2();
    }
    p.s.v.m2()
    {
        s.o.pln.(1/0);
    }
}
```

o/p. ⟶ Arithmeetic Exceeption.

Stack trace seeing ⟵

```
java.lang.ArithmeticException: / by zero
    at ThrowableExample.m2(ThrowableExample.java:
    at ThrowableExample.m1(ThrowableExample.java:
    at ThrowableExample.main(ThrowableExample.java:
```

* Explicit Exception. —

* ~~Through~~ — throw. —

- Through keyword is used to through an exception explicitely.
- through keyword is used only to through a throwable type of object.
- Once the exception object is through an, the JVM checks wethether any catch block is there to handle these exception if not it will be handled by default exception Handling.

* Syntax. — throw new ClassName();

throw → used to throw exception explicitely

new → create obj

ClassName() → constructor call

* Ex. —

```
class Throwable
{
    p. s. v. m (s[]a)
    {
        throw new IndexOutOfBoundsException(); // Runtime
        throw new ArithmeticException(); // Runtime ~~compile time~~
        throw new Exception(); // compiletime
        throw new InterruptedException(); // compiletime
        throw new RuntimeException(); // Runtime
        throw new IOException(); // compiletime
        throw new ClassCastException(); // Runtime
    }
}
```

* Ex. —

```
class Demo
{
    p.s.v.m(S[]a)
    {
        throw new String(); // compiletime error
        throw new object(); // compiletime error
    }
}
```

incompatible types: obj. cannot be converted to Throwable

```
throw new obj();
```

Note. —

- throw keyword throw only throwable type of object.
- It means other than exception Hirarchi if we try to through any other class we will get the compiletime error. (Incompatible types).
- To overcome these problem we need to extend that class with exception hirarchi, which is nothing but customise exception handling.

* Customised Exception handling in Java. —

- To create a customised error message we use user defined exception.
- We can create user defined exception as checked exception or uncheked exception.
- To create user defined exception the class must extend with any of exception class in the hirarchi

* Note. —

- It is always recomended to keep our Customised ~~et~~ exception class as uncheked exception.
- That is we need to extend it with runtime exception

1) Ex. —

```
import java.util.scanner;
class Demo extends RuntimeException
{
    ~~Excep~~ Demo(string desc)
    {
        super (desc);
    }
    p.s.v.m (s[]a)
    {
        scanner sc = new scanner (system.in);
        s.o.pln ("Enter ur age :");
        int age = sc.nextInt();
        if (age >=18)
        {
```

```
s.o.pln.("u r eligible");
}
else
{
    throw new Demo("u r age is not eligible kid");
}
}
```

o/p. → Enter u r age:
15
Exception in thread "main" Demo: u r age is not eligible kid at ~~at~~ Demo.main(Demo.java

2)
```
class InsufficientFundsException extends RuntimeException
{
    InsufficientFundsException(String desc)
    {
        super(desc);
    }
}
class BankFeature
{
    private double bal = 10000;
    static BankFeature obj = new BankFeature();
    Public double getBal()
    {
        return this.bal;
    }
    public void setBal(double nbal)
    {
        this.bal = nbal;
    }
```

```
p. v. deposit (double dmat)
{
        this. setBal (this.getBal()+dmat);
        S.o. pln.("amt deposit");
        s.o. pln. ("new bal is :"+ this.getBal());
}
p. v. withdraw(double wamt)
{
        if( this. getBal() >= wamt)
        {
            obj. setBal(this.getBal()+wamt);
            s.o.pln. ("amt debited");
            s.o.pln. ("new bal is:"+ this.getBal());
        }
        else {
            try {
                throw new InsufficientFunds Exception ("no paisa");
            }
            catch(InsufficientFunds Exception j)
            {
                s.o.pln.(j);
            }
        }  }
        p.s.v.m. (S[]a)
        {
            obj. withdraw(200);
        }
}
```

o/p →

26/12/23

* Throws. —

- Throws keyword is used at the end of method declaration to Indicated that exception of given type may thrown by the method.
- The main purpose of throws keyword is to delica the responsibility ~~to~~ of handling the exception to the caller (method or JVM).
- In case of unchecked exception it is not required to use throws keyword.
- we can use throws keyword Only for Throwable ty otherwis we will get a compile time error (Incompartible Types).

1) Ex. —

```
class Demo
{
    p. s. v. m. (s[] a)
    {
        string a = "Mansi";
        for (int i=0; i<a.length(); i++)
        {
            s.o.p(a.charAt(i));
            Thread.sleep(2000);
        }
    }
}
```

O/p → Mansi (print each letter after 2 sec.)

2) class Demo
{

```
p.s.v.m.(S[]a)
{
        mansi();
}
p.s.v. Anushka()
{
        int num1 = 10;
        int num2 = 0;
        try {
                amit (num1, num2)
        } catch (ArithmeticException a)
        {
                s.o.pln. (" a has been handled");
        }
}
p.s.v.m (int num1, int num2) throws ArithmeticException
{
        int op = num1/num2;
        s.o.pln. (op);
}
}
```

O/p. ⟶

Ex. —

```
class WrongException extends Exception
{
        WrongException( String desc)
        {
                super( desc);
        }
}
class Mangi
{
        p. s. v. evenOdd ( int num) throws WrongException
        {
                if(num % 2 = 0)
                {
                        s.o.pln. (" even");
                } else
                {
                        s.o.pln. ("odd");
        }       }
                else {
                        throw new WrongException ("zero");
        }       }
}
class Patil
{
        p. s. v. checkNumbers () throws WrongException
        {
                Mangi. evenOdd(10);
        }
}
```

```
class Demo
{
    P s.v.m(S[]a)
    {
        s.o.pln("execution starts");
        try {
            Patil.checkNumbers();
        }
        catch(WrongException n)
        {
            s.o.pln("Wrong Exception handled");
        }
        s.o.pln("execution ends");
    }
}
```

o/p. ⟶ execution starts
even
execution ends.

*	throw	throws
1)	It is used to throw an exception Explicitely. (single Exception)	1) It is used to throw Exception Implicitely. (multiple Exceptoion).
2)	by using throw only uncheked exception propogate	2) By using throws be checked & uncheked exception can be propogate
3)	It is used within the method body.	3) It is used in method declaration statemen
4)	multiple exception cnnot be throwon using throw key word.	4) multiple Exceptions can throw using throws keyword.
5)	throw new Exception class name () syntax →	5) method declaration thro Exception class name 1, Exception class name 2 syntax →

1) can we ~~override~~ inherit static method?

→ NO, cause method overriding is based on dynamic binding at runtime & static methods are binded using static binding at compile time.

2) can we make Interface final?

→ NO, cause Interface in java is similar to a class but it contains only abstract methods & fields, which are final & static. since all the methods are abstract ∴ we cannot Instantiate interface.

3) can we make constructor final?

→ NO, final keyword is used to prevent them to be overidden by subclass consis also like a methods, but as we know that cons. can not be inherited in subclass hence there is no use of final keyword with cons.

4) can we make cons ~~final~~ static?

→ No, we will get CTE. cons is accessible to all methods so, we don't need to make a copy of cons making it static.

5) can we perform overriding in constructor?

→ No, cause if we try to write a super class's const in the ~~compiler~~ + sub class compiler treats it as a method & expects a return type & generate CTE.

6) can we perform overloading? in cons?

→ yes, we can perform overloading in const. like methods. A class having more than 1 cons. is known as cons. overloading.

7) can we overload static method?

→ Yes.

E) can we overide a const?

→ No, cons. cannot be overridden in the same way that methods can be overriden.
Cons. do not have a return type & are not inherited by sub class.

6) Why we can't inherit static methods?

→ ~~Static~~ they are associated with the class itself rather than with any particular instance of the class
OR
static methods do not use any instance variables of any object of the class they are defined in.

7) why main method is static?

→ There is no object of the class existing when the java runtime starts. This is why the main must be static for the JVM to load the class into memory & call the main fun^n.

8) why string [] args is compulsary in java?

→ It allows the method to accept zero or multiple arguments.

9) Why do we need use relationship in java:
→ It is used for code reusability. Basically it means that an instance of the one class has a reference to the instance of another class or the other instance of the same class. This relationship helps to minimize the duplicate ion of code as well as the bugs.

10) Why do we need has a relationship?
→ has a relationship in java is known to be as composition. it is used for code reusability. Basically it means that an instance of another class or the other instance of the same class. This relationship helps to minimize the duplication of code as well as the bugs.

11) Why we use public class in java.
→ public is a ~~g~~ java keyword which declares a members access ~~at~~ as public, public member's are visible to all other classes. This means that any other class can access a public field or ~~or~~ method.

12) Can we overide main method?
→ No, we can not override main method of java because a static method cannot be overridden.

13) Which methods we cannot override?
→ static methods can not be overridden since they are bounded at compile time & method overriding

relies on dynamic binding at runtime.

14) can we declare main method as final.

→ yes, we can declare the main() method as final in java. The compile does not throw any error if we declare any method as final by placing the final keyword then that method becomes the final method. The main use of final method in java is they are not overridden.

15) can we inherit const. in java?

→ No, const. can not be inherited in java in inheritance such class inherits the members of a super class except const. constructors are the initializers we can not inherit it.

16) Adv. of class in java.

→ 1) Abstraction.
2) Reusability.

17) How many cons. can we create in java?

→ As many (Infinite).

18) main method can be overloaded?

→ yes, we can. but JVM always calls the original method. It does not call the overloaded main method

19) What is null pointer Exception?

→ Null pointer exception is runtime exception in java that occurs when a variable is accessed which is not pointing to any object & refers to nothing or null.

20) why does java have null pointer Exception.

→ Reasons for null pointer Exception in java. Invoking a method on an object instance but at runtime the object is null.

21) why does java have null pointer Exception.

→ Reasons for null pointer Exception in Java Invoking a method on an object instance but at runtime the object is null.

[illegible]

[illegible]

[illegible]

[illegible] (Install) check on linked in

4) [illegible] .com

5) [illegible]

* search on [illegible]

1) multiple ways to print the 2D array
(for while, do while, for each looping, [illegible]
2) int type array string double [illegible] char

3) multiple ways to print 2D array
(for while, do while, for each looping, [illegible]
byte short long string int char

3) multiple ways to print 3D array
(for while, do while, for each looping, [illegible]
string char double string char string

4) WAP to create an array using [illegible] class
(user specified size, data)

5) WAP to [illegible] input from user and the [illegible]

6) WAP to find sum of elements from an array

7) WAP to find even & odd elements

8) WAP to find prime elements from an array

9) WAP to [illegible]

10) WAP to find palindrome from an array

* Software testing :-

- It is a process of finding a bugs in the s/w
- verification of appl^n against req. specification

Q: Why we need software testing?

→ If the software

1) - every s/w is built for supporting Buisness.

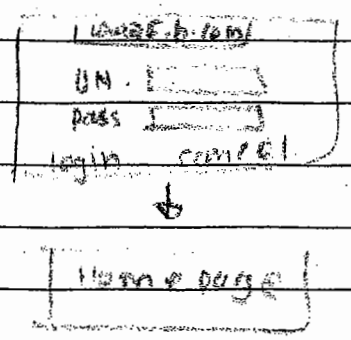
2) - If there is a problem in the s/w it is going to to affect the Buisness, so, we do s/w Testing to find all possible bugs.

3) To improve the quality of the prod

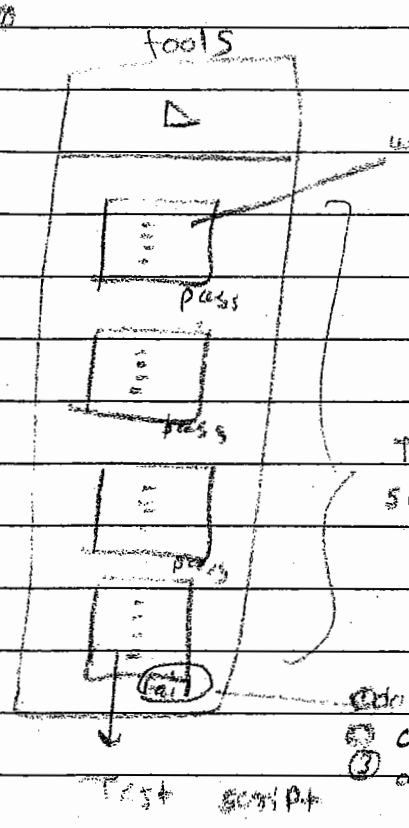
* Types of s/w testing :-

1) WBT 2) BBT

2) BBT - (we do it by manually or by Automation)



Automation Testing



- verification against of an appl^n against req. specification is called as BET

BET

Types of ~~Stg~~ Testing

1) Functionality Testing
2) Integration Testing
3) System testing
4) Acceptance "
5) Regration "
6) compatibility "
7) performance "
8) smoke "
9) ~~Fai~~ recovery "
10) Adhoc Testing, etc.

1) functionality testing / component T. / field level T.

= Testing each & every component ~~is~~ with all possible scenarios as thoroughly against req. specification is called as functionality testing

Components

- Name [ ]	read fields
- [send]	Button
- Gender male / female	Radio button
- country [▼]	drop down
- [ ]	check box
- [✓] agree	"
- link	www.flipkart.com
- [≡] [I] [⊞]	menue
- [ ]	mice
- - minimize, x maximize	

does java support pointers? → yes

pointer in java only supports internally. it is not [illegible] at a programming level

30% | dia. | summ | 50%

// [illegible]

- +ve summ which matches the req.
- -ve summ which reduces the [illegible] summ
- +ve testing
- -ve [illegible]

SBI _ SRS

1. Create A/C

2.1 Profile

3.0 Amt Transfer

3.1 FAN [illegible]

30.3 Amount textfield

30.1 Source account

only +ve digits

+ve int

11 digit int.

30.3 Source account

only +ve int.

A/C bal	Amt Deposit
Amt	FAN
login	FAN
logout	Amt
	Send / Cancel

100 ✓

-100 ✗

1,100 ✗

0 → not serious, [illegible]

-9124 ✗

2.00 ✗

10 digit ✗

[illegible] ✗

[illegible] ✓

$100 ✗

10.00 ✗

2217 ✗

112.3 ✗

[illegible] ✗

9214[illegible]90 +ve summ

[illegible]

[illegible] +ve summ

[illegible]

- 1 @ 1[illegible] +ve summ

- 1902145678 12 [illegible]

lesson 7 -

1) always start dealing with the scenarios
2) If the scenario fails stop testing
3) If the scenario fails report to the developers & continue testing
4) If there is any confusion or conflict in the req / raise assume a req.
4) spend more time in finding more scenarios & optimizing them.
5) avoid ~~the~~ overtesting & undertesting

- write scenarios for following cases.

1) pincode text feild (6 digit no.) 2) scenario
2) ~~zos~~ User name text feild (3 to 7 chars, upper & lower case).
3) write scenarios for debit card text feilds which accept master card, discover, Rupay, visa, more

	CN	Scenario	E.R
1) PIN			
2)			

3) Debit card [ ]

— Visa

Rupay discover,

const. chainning

```
class Demo
{
    p.s.v.m.(S[]a)
    {
        Demo();
        }
            ~~pH~~ s.o.pln.("Hii const 1");
            Demo();

        }
        Demo();
        {
            s.o.pln.("Hii const 2");
            Demo();
        }
}   }
```

class shapes

p v display shapes()

pln("shapes")

```
class Demo
{
    p.s.v.m.(S[]a)
    {
        m1();
    }
    p.v.m1()
    {
        ———— m1(int a)
    }
    p.v.m2(int a)
    {
        ————
    }
}
```

even odd

```
class Demo
{       p.s.v.m.(S[] a)
        {
        int num = 2;
        if (
```

```
if(num/2=0)
{   pln.("even");
}
else{
    pln.("odd");
}
```

2 + 2

$\frac{5\frac{1}{2}}{6}$ $\frac{5\frac{1}{3}}{3}$

```
class parent
{
        p.s.v.m(S[] a)
        {
                int a = 10;
                ~~sto~~ int b = 20;
        }
}
class ~~parent~~ child extends parent
{
        ~~in~~ pln.("Hello");
}
```

AT P

Can we do override main → NO
we can call main in anywhere in program just pass ~~arguments~~ ~~param~~

method Example –

```
main (s[]a)
{
    int a = add1();
    int b = add();
    s.o.pln. ("a" a);
    s.o.pln. (b);
    p.s v = add();
    {
        s.o.p. (v+i);
        return n;
    }
    p.s. ~~v~~ char add1();
    {
        s.o.pln. ("Hii");
        return 'a';
    }
}
```

error cause void cannot contain in int.

Output

Box Box Box

AT P

does object class constructor has super call statement? Justifying it.

1) can we declare an interface without any method? ye marker interface
2) can use private methods in an interface? ✓ after 1.9
3) can we overload private methods? ~~No~~ Yes
4) can we declare interface as final? No, cause it is constant & we can not inherit the o.t. interface ~~the~~ another class.
5) can we define a constructor inside an interface? No cause interface is abstract.
6) can we have non-static variables inside an interface? No cause they are static & final by default
7) can we have an protected method in an interface? NO

destructor is used to destroy the java | we don't have destructor in java, we have garbage collector
but we have ——— which is similar like a destructor

- destroy an unwanted obj (which does not have any ref. it ~~a~~ ~~static~~ memory) we destroy the obj
we use finalized method is present in obj. class
- it is done by the garbage collector

Rules of Identification I

- [illegible]

1) WAJP to check wether the no is +ve, -ve, 0.
2) ——— " ——— the student is pass / fail.
3) ——— " ——— vowel / consonant.
4) ——— " ——— leap year or not.
5) ——— " ——— driving linces eligibility.
6) photo.
7) photo.

1) WAJP ~~to~~ using switch to check wether letter is consonant or vowel.

2)

[illegible]

- [illegible]
- [illegible] find factors of a no. using recursion
- find factors of a no. using recursion
- Palindrome no. using recursion
- [illegible]
- even no's from 1 to 100 using [illegible]

>>> >>

- Can we create static count without static main method?
  Yes we can create but it is not a feasible standard
  - e.g. - multiness
  - e.g. - Hierarchical