INDEX

NAME : _____ SUB : _____

STD : _____ DIV : _____ ROLL No: _____

SCHOOL / COLLEGE : _____

Sl. No.	DATE	TITLE	PAGE No.	TEACHER'S SIGNATURE
		Basic Java		
		i) Evolution of programming langs		
		ii) History of high level programming langs		
		iii) Structured, unstructured & Object oriented		
		iv) History & versions of Java		
		v) Platform dependent & independent		
		vi) JDK, JRE, JVM		
		vii) Overview of Java architecture		
		viii) Features of JAVA.		
		Core Java		
		i) 1st pgm in JAVA		
		ii) Interview question		
		iii) Tokens(keywords, identifiers, literals operators & special Symbols)		
		iv) Data Types & Type casting		
		v) Flow control statements		
		vii) object orientation (class, Objects, variables, arrays, Anonymous array for each loop)		

Sl. No.	DATE	TITLE	PAGE No.	TEACHER'S SIGNATURE
		Methods ✓		
		Vars		
		Strings (Mutable, immutable strings)		
		Encapsulation		
		Constructors		
		(Type of constructors)		
		(Constructor overloading)		
		Polymorphism		
		Inheritance		
		(Method overriding)		
		(Method hiding)		
		Aggregation, Composition, Association.		
		Final keyword		
		Abstraction		
		Interfaces (marker interfaces)		
		Exception handling (try, catch, finally, throw, throws)		
		• Custom exceptions		
		checked & unchecked exceptions		
X		Multithreading.		
		file i/o streams		
		buffered streams & i/o streams		
		Serialization & deserialization		
		transient keyword & externalizable interface		
X		Socket programming		
		One way & multiway communication		
		—||— End of JSE.		

Java full Stack

Day-1 13/02/24

Basic Java

Diff b/w RAM & ROM.

RAM	ROM.
* Volatile	* Non-volatile
* Random access memory	* Read only memory
* Stores temporary data	* Stores permanent data
* We can read and write memory	* Read-Only Memory
* Writing data is faster	* Writing data is slower.
* Store data in MBs	* Store data in GBs
* large size with higher capacity	* Small size with less capacity.
* costlier	* cheaper than RAM
* used in CPU cache, primary memory	* used in firmware, micro-controllers
* made up of transistors & capacitors	* Made up of logic gates.
* can be upgraded	* cannot be upgraded.
* Normal operations	* Starting processes.
* Static, Dynamic	* PROM, EPROM, EEPROM

Loading & saving

* The process of storing the data from RAM to harddisk is called as saving.
* The process of fetching the data from hard disk to RAM is called loading.

Date / / /

J2EE (Java enterprise edition)

To connect to DB

JDBC (Java database connectivity)

- Types of drivers
- Transaction management

Servelets (To connect to web pages)

- Servelet life cycle
- Servelet config, context
- servelet dispatcher, HTTP request & HTTP response.

JSP (Java server pages)

Frameworks

* ORM framework - Hibernet
* Spring
  Springboot (Springboot JPA & Springboot MVC)
* Microservices (REST)

Date / / /

Evolution of programming languages

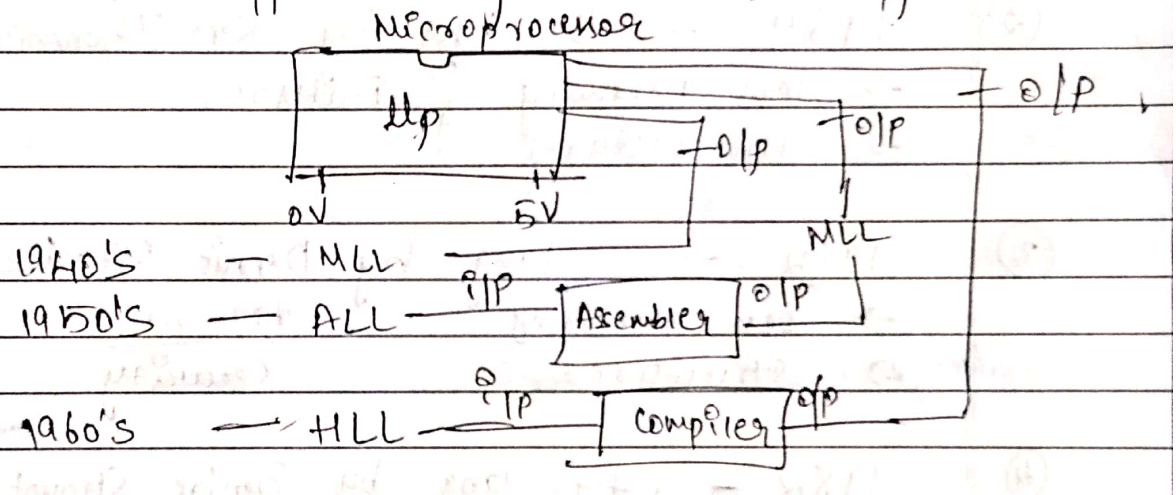
(MLL) Machine level language — 1940's.

(ALL) Assembly level lang — 1950's.

Assembler is a software used to convert ALL to MLL, takes ALL as i/p & gives MLL as o/p.

(HLL) High level lang — 1960's.

Compiler is a software used to convert HLL to MLL, takes HLL i/p & gives MLL o/p.



llp — process o/p.

	MLL	ALL	HLL
Addition	01101110	add	+
Subtraction	11011100	sub	−
Multiplication	00011001	mull	*
Division	01111011	div	/

Drawback 'remember' 'memory' 3+4
code add (3,4)

MLL → Machine level lang or Binary code (0's & 1's)

ALL → Assembly level lang (Pneumonics)

Date / / /

HLL - High level lang (Symbols)

History of high level programming languages

① First progamming lang - BCPL (Basic combined programming lang)
1962 by Martin Richards
→ more memory } Features &
→ Unstructured } drawbacks.

② 1969 - B lang by Ken Thomson
→ less memory } Features
→ Unstructured }

③ 1972 - C lang by Dennis Ritchie
→ less memory drawback
→ structured. Security.

④ 1982 - C++ lang by Banjre Stroustrup
→ less memory drawback
→ Structured platform dependent
→ Security.

⑤ 1992 (unofficial) β version (Beta)
Java - James Gosling.
1996 (official)
→ less memory
→ OOP's
→ Security
→ Platform independent.

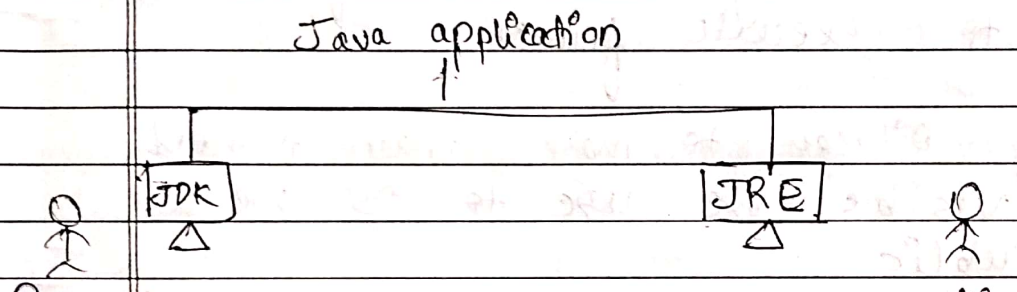
Date / / /

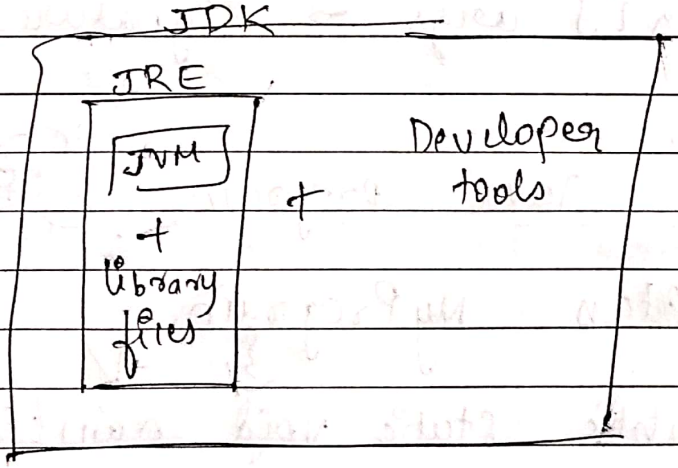
Interview questions

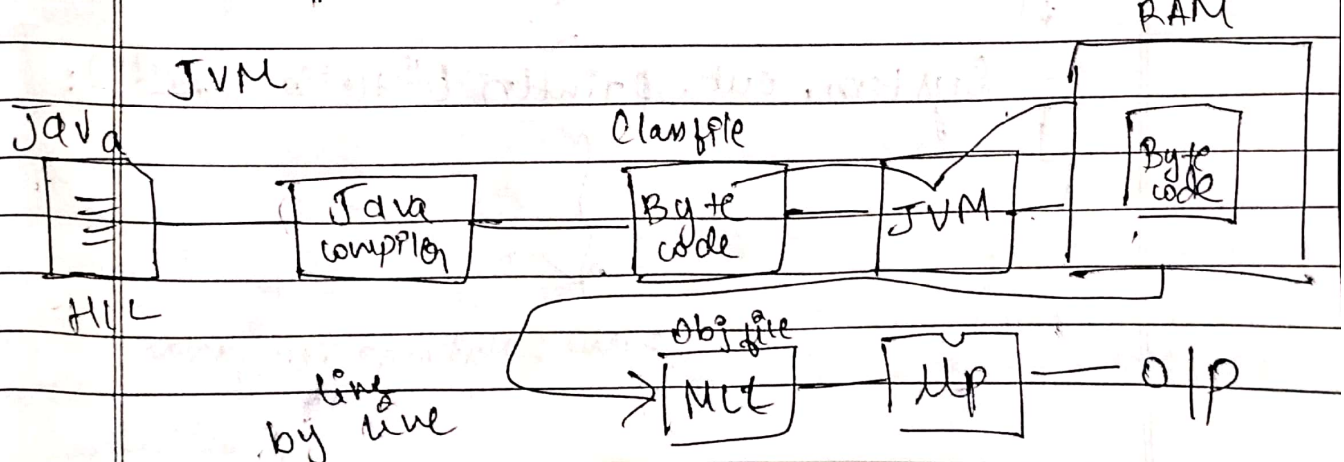
1. Tell me about yourself.
   Greet, name, Place, Highest degree (college), %, Intermediate & Schooling (place), Skills, Certifications, Internship, Interest or passion.
   Thank

Day - 3
15/2/24

What is JDK, JRE and JVM







Date / / /

* Java Virtual Machine
* loads bytecode into RAM
* converts bytecode into MLL (line by line)
* Interpreter

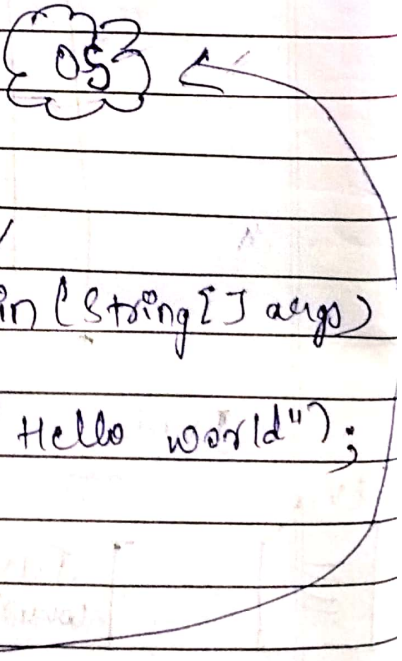
why should you have class
rule in Java → everything enclosed in class

→ OS will give control for JRE to execute func

In order to make main method visible ~~use~~ to OS we use public
static is used to access main method by OS.
String [] args → signature. (1D array)

~~Q.~~

1st Java program



Date / / /

* Java Virtual Machine
* loads bytecode into RAM
* converts bytecode into MLL (line by line)
* Interpreter

Why should you have class
rule in Java → everything enclosed in class

→ OS will give control for JRE to execute func

In order to make main method visible to OS we use public
static is used to access main method by OS.

String[] args → signature (1D array)

1st Java program


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

Save: MyProgram.java

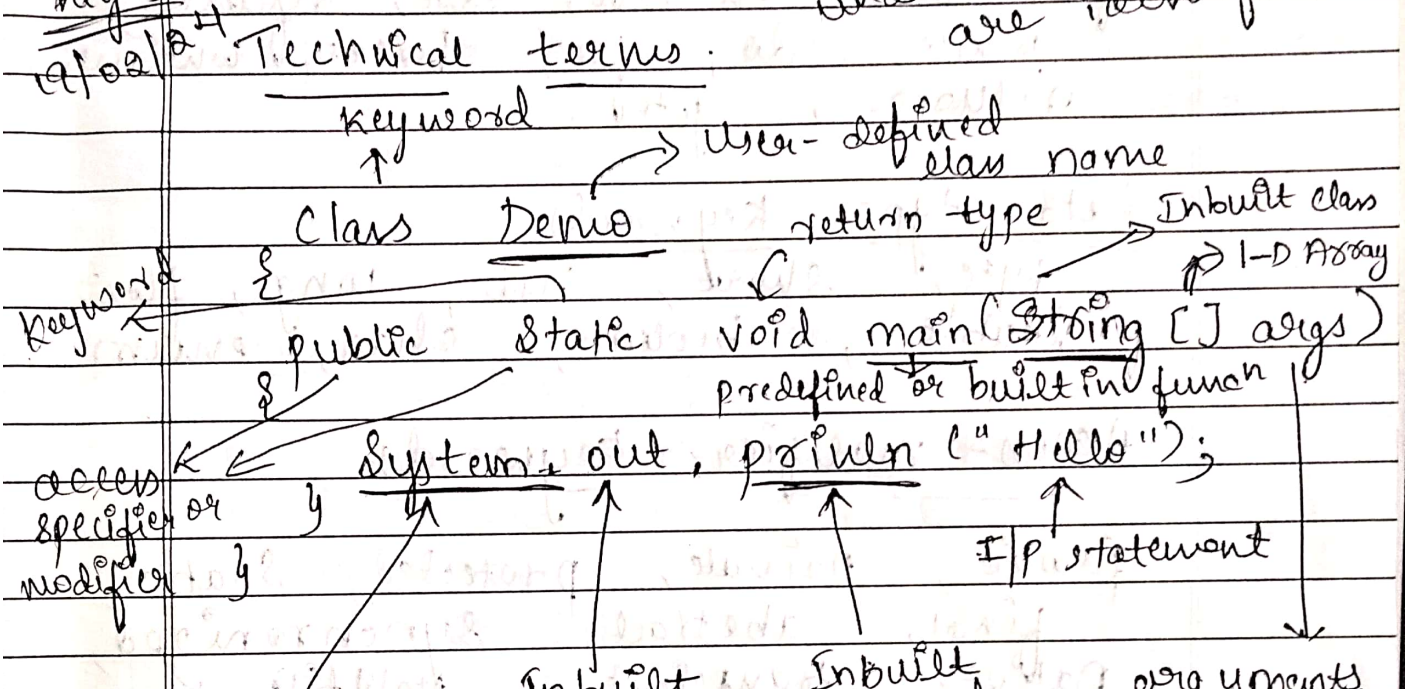
Date / / /

Save : MyProgram.java

1. Compile → javac MyProgram.java
2. Execute → java MyProgram
   (It asks for JVM)

Day 5
4/02/24

Technical terms.



Tokens

Tokens are the smallest unit or element in the program.
Tokens are divided into 5 categories

Keywords -
Identifiers
literals
operators
Special symbols.

Keywords — predefined words which has predefined meaning also called as reserved words.

Date / / /

meaning of word is predefined to the java compiler. Totally there are 15- Keywords.

Flow control keywords
if, else, switch, case, default, while, do, for, break, continue, return, goto.

Data-type keywords
byte, short, int, long, float, double, boolean, char, enum

Access-modifier keywords
public, private, protected, static, final, abstract, synchronized, native, transient, volatile, strictfp

Exception keywords
try, catch, throw, throws, finally & assert.

Class related keywords
class, interface, extends, implements, package, import

Object related keywords
new, instanceof, super, this.

Written type keywords
void

Date / / /

Literal Keywords
roust

* Identifier
It is the name given in java for identification.
Ex: classname, variable name, method name or funcs name, labelname, interface name etc
switch case1, case 2.

Rules for identifier
* cannot be keywords
* cannot start with numbers/digits (0-9)
* An identifier can be created using alphabets (uppercase & lowercase) ex: int Sal = 32000, int sal = 3000, int SAL = 6000
* only special characters such as (dollar) $ & underscore (_) are allowed while creating an identifier.
ex: int $sal = 30000; int emp_sal = 40000;
ex: int emp1_sal = 30; ✓
int 1emp_sal = 40; ✗
* Identifiers are case sensitive.
int age = 30; int Age = 32 ✗
age = 32; ✓
* keywords cannot be used as identifiers
int assert = 40; ✗

* Literals
literal is a value associated with a variable or it is a value given to an identifiers.

Date / / /

Ex:- float PI = 3.147f;
↑
Literal (const)

int a = 10;
datatype variable name (identifier) = value or literal

Types of Literals
1) integer literal (int a = 10;)
2) floating literal (float PI = 3.147;
3) character literal (char name = 'M/F
4) Boolean —11—
5) double —11—
6) short —11—
7) long —11—
8) String —11—

* Number system
There are 4 types
1.) Binary
2) Octal
3) Decimal
4) Hexa decimal

*** Note :- By default java makes use of decimal number system

zero b (0b) 0 → not zero its 0b | zero x (0x)

Date / /

Binary	Octal	Decimal	Hexa-decimal
number - 0 & 1 (base 2)	number - (0-7) (base 8)	number - 0-9	number - 0-9, a-f
prefix - 0b	prefix - 0	prefix	prefix - 0x

```
Ex: class Demo
{
psvm (....)
{
int num
= 0b101;
sop (num);
}}
```

```
Ex: class Demo
{
psvm (....)
{
int num =
016;
sop (num);
}}
```

```
int num =
243
```

```
int num = 28b
```

Calculation:

(0)

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

$4 + 0 + 1 = 5$

o/p: 5

Calculation:

16

$1\times8^1 + 6\times8^0$

$= 8 + 6 = 14$

o/p: 14

o/p: 243

Calculation:

28b

$2\times16^2 + 8\times16^1 + 11\times16^0$

o/p: 640

Operators

operators are the special symbol used to perform some operation on the operands

a + b

+ operator (sum)

Types of operators

- Arithmetic
- Unary
- Assignment
- Relational
- logical bitwise
- shift

* Comparison

* Binary or conditional
* Short circuit or logical

Arithmetic - +, -, *, /, %

```
class Arop
{
    psvm c....)
    {
        int a=10; int b=4;
        SOPln (a+b); // 14
               (a-b); // 6
               (a*b); // 40
               (a/b); // 2
               (a%b); // 2
               (10%10); // 0
               (25%100); // 25.
```

if it is not divisible it returns the smaller number

Unary - ++ , --

pre	post	pre	post
++a	a++	--a	a--

pre (inc/dec)
1. inc/dec
2. display

post (inc/dec)
1. display
2. inc/dec

a = 7, 8, 9

1) class Demo
{
psvm(...) {
int a = 7
SOP (a++); // 7
SOP (++a); // 9
SOP (a); // 9
}
}

a = 9

a = 7, 8, 9

2) class Demo
{ psvm (...)
{ int a = 7;
SOP (a--); // 7
SOP (--a); // 5
SOP (a); // 5
}
}

a = 5

3) int a = 7 8 9 8
SOP (++a); // 8
SOP (a++); // 8
SOP (--a); // 8
SOP (a);

a = 8

4) int a = 7 8 9 8 7
SOP (a++); // 7
(++a); // 9
(--a); // 8
(a--); // 8
SOP (a);

a = 7

5) int a = 7 8 9, 10 9 10 9 8
SOP (++a); // 8
(++a); // 9
(a++); // 9
(--a); // 9
(++a); // 10
(--a); // 9
(a--); // 9
SOP (a);

a = 8

6) int a = 7;
SOP (a++ - a--);
SOP (a); 7 - 8 = -1

a = 7

7) int a = 7; 8 + 7 - 8 = 7
SOP (++a + --a - ++a)
SOP (a); a = 8

8 + 7 - 8 = 7

8) int a = 7 6 + 5 + 5
SOP (--a + --a + a++); // 16.
SOP (a); // 6

INFINITY

Date / / /

9) int a = 7
sop (++a + --a - ++a + --a - ++a
+ ++a - ++a - a++); // -5
sop (a); //

8 + 7 - 8 + 7 - 8 + 9 - 10 - 10
- 20

10) int a = 4;
sop (a++ + ++a + --a + a-- + a++
+ a++ --a - a--); // 19
sop (a);

4 + 6 + 5 + 5 + 4 + 5 - 5
- 5

a = 4

11 + 64 - 7
11 + 8 * 8 - 7
75 - 7 = 68

4, 6 a = 3

11) int a = 7(6 + 5) + (5 + 3) *
sop (a-- + --a) + (a-- + --a) *
(++a + a++) - (--a + --a));
sop (a); (4 + 4) - (4 + 3)
a = 3

* Assignment operators (write pgm)
=, +=, -=, %=, *=

* Relational <=, >=, ==

* Shift operators

left shift | Right shift

Conditional Operators

Syntax: Condition ? Exp 1 : Exp 2;
Ex: (num % 2 == 0) ? even : odd;

* WAP in Java to find the no is even or odd using conditional operator
* WAP to find largest among 3 no in Java using conditional operators

Flow control Statements (HW)

```
public class EvenOdd {
    public static void main (String[] args) {
        Scanner reader = new Scanner (System.in);
        System.out.print ("Enter a no");
        int num = reader.nextInt ();
        if (num % 2 == 0)
            System.out.println (num + " is even");
        else
            SOpln (num + " is odd ");
    }
}
```

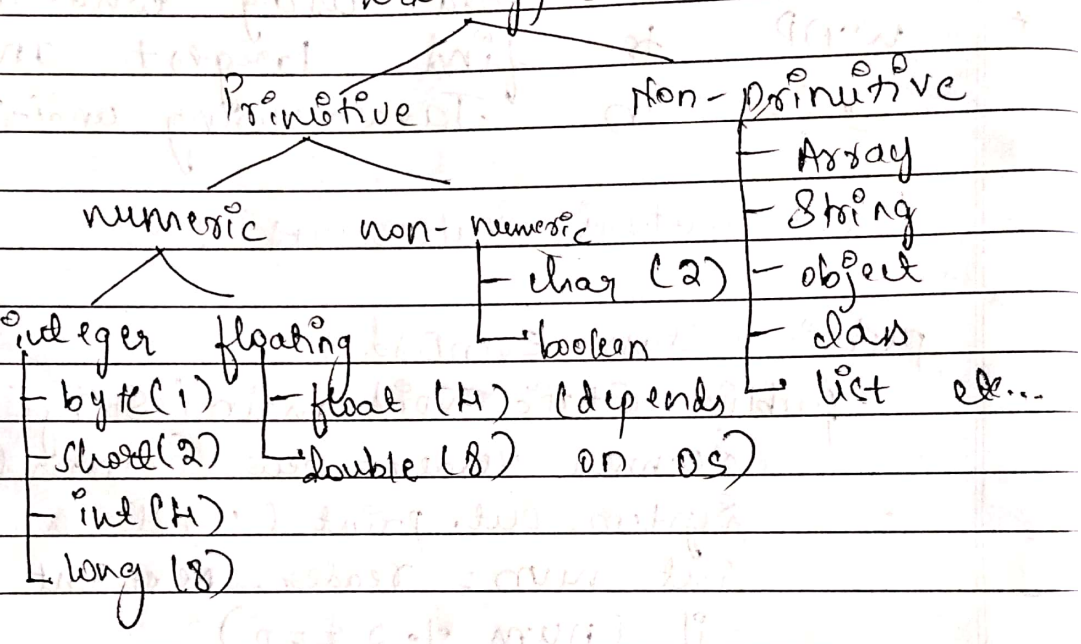
```
if (num1 > num2)
    greatest = num1
if (greatest > num
```

6th day
20/2/24

Date / / /

Data types in Java.

Datatypes are used to specify the values into the variables. It helps to convert data into binary format.



Interview question: Why the float value is suffixed with f.

If float value is not suffixed with f then it is treated as double in Java.

```
float a = 3.481f;
          4 (float)

float a = 3.481;
          8 (double)
```

Data overflow happens.

Definition	Byte	Short	int	long	float
Declaration	byte a= 28;	Short a=128;	int a=32;	long int a= 256;	float a = 3.481f;
Size	1byte = 8 bits	2 byte= 16bits	4bytes = 32 bits	8 bytes = 64bit	4 bytes = 32 bits
Range	-2^{n-1} to $2^{n-1}-1$	-2^{n-1} to $2^{n-1}-1$	$n=32$	$n=64$	$-3.4e^{38}$ to $3.4e^{38}$
	$n=8$ -2^{7} to $2^{7}-1$	$n=16$ -2^{15} to $2^{15}-1$	-2^{31} to $2^{31}-1$	-2^{63} to $2^{63}-1$	
	-128 to 127	-32768 to 32767	int Sal= 1,00,000	long int a= 245678	
	byte age= 22;	Short int range= 30,000			
	byte marks= 88;	Short int Sal= 15,000			

double

8 bytes = 64 bits

$-1.7e^{308}$ to $1.7e^{308}$

char

char a [redacted];

2byte = 16bits

boolean

bool

no range

depends on OS

value will be T|F

ex: logical gate

e.g. distance is variable b/w sun & moon

Why

Interview question

Why char size in java is 2 bytes and C, C++ has 1 byte

Because Java uses unicode format (UTF) & C uses ASCII format. Java contains 18 langs whereas C contains only 256 characters. 256 is represented in 1 byte but Java contains 65,535 & can't be repre-sented in 1 byte. So java character size is 2 byte.

Examples on datatypes

1)
```
int a = 5;
int b = ++a;
SOP (a); // 6
SOP (b); // 6
```

2)
```
int a = 5;
int b = a++;
SOP (a); // 6
SOP (b); // 5
```

3)
```
int a = 5;
int b = ++10;
SOP (a);
SOP (b); // ERROR
```
Unary cannot be applied on 10 → 11

4)
```
int a = 5;
int b = --(++a);
SOP (a); //
SOP (b); // ERROR
```
→ 11

5)
```
char c = 'a';
c++;
SOP (c); // b
```

6)
```
double d = 12.4;
d++;
SOP (d); // 13.4
```

7)
```
boolean b = false;
b++;
SOP (b); // ERROR
```

8)
```
byte b = 5;
b++;
SOP (b); // 6
```

9)
```
byte b = 5;
b = b + 1;
SOP (b); // ERROR
```

10)
```
Short s = 6;
S = S + 1;
SOP (s); // ERROR
```

max (int, type of op1, type of op2)

```
b = b + 1
    op1 op2
```

max (int, byte, int)
4 1 4

max = 4

```
b = b + 1
1     4
  < 4
```

So error

max (int, short, int)
4, 2, 4

Date / / /

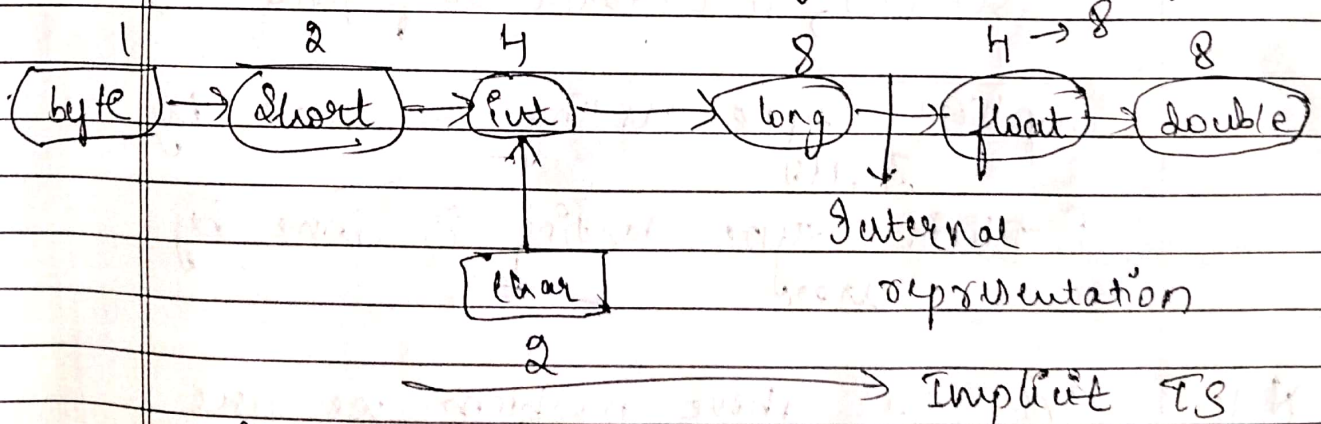
Note :- op1 op2 → max (int, type of op1, type of op2)

byte + byte → max (int, byte, byte)	→ int
byte + short	int
short + int	int
byte + long	long
short + long	long
long + long	long
long + int	long
int + int	int
→ float + long (4 + 8)	float [Type casting]
float + double	double

Type Casting in Java

Converting one type of data to another type is called type casting.

implicit type casting : It is converting lower data type to higher data type. Type checking is done by compiler. Type casting is done by JVM



Ex :- byte b = 12
int i;
i = b
Sop (i); // 12

Examples on datatypes

1)
```
int a = 5;
int b = ++a;
SOP (a); // 6
SOP (b); // 6
```

2)
```
int a = 5;
int b = a++;
SOP (a); // 6
SOP (b); // 5
```

3)
```
int a = 5;
int b = ++10;
SOP (a);
SOP (b); // ERROR
```
Unary cannot be applied on 10 ⟶ // ⟶

4)
```
int a = 5;
int b = --(++a);
SOP (a); //
SOP (b); // ERROR
```

5)
```
char c = 'a';
c++;
SOP (c); // b
```

6)
```
double d = 12.4;
d++;
SOP (d); // 13.4
```

7)
```
boolean b = false;
b++;
SOP (b); // ERROR
```

8)
```
byte b = 5;
b++;
SOP (b); // 6
```

9)
```
byte b = 5;
b = b + 1;
SOP (b); // ERROR
```

10)
```
Short s = 6;
S = S + 1;
SOP (s); // ERROR
```

max (int, type of op1, type of op2)

```
b = b + 1
    op1 op2
```

max (int, byte, int)
4 1 4

max = 4

```
b = b + 1
1     4
  \ x /
```

So error

max (int, short, int)
4, 2, 4

Date / / /

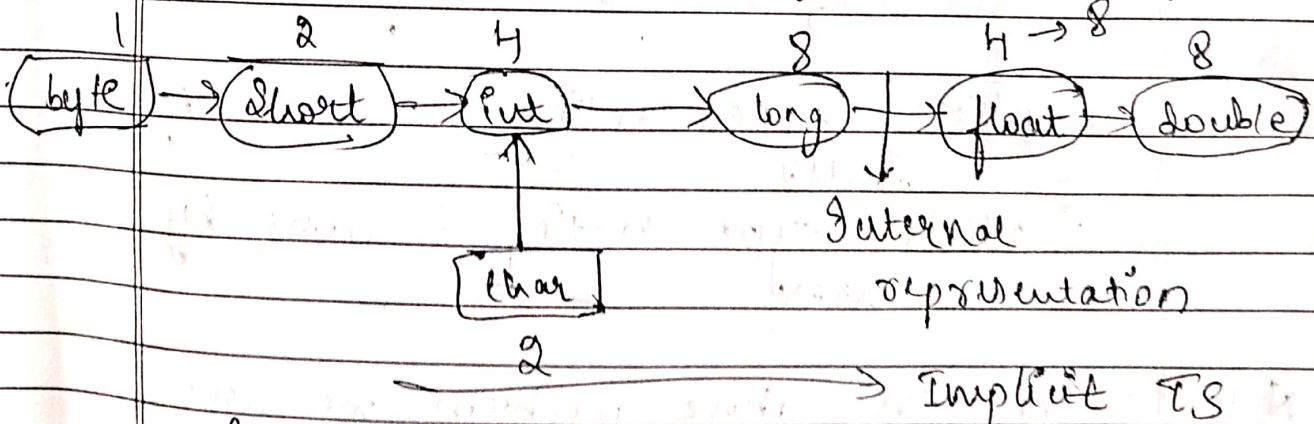
Note :- op1 op2 → max (int, type of op1, type of op2)

byte + byte → max (int, byte, byte)	→ int
byte + short	int
short + int	int
byte + long	long
short + long	long
long + long	long
long + int	long
int + int	int
→ float + long (4 + 8)	float [Type casting]
float + double	double

Type Casting in Java

Converting one type of data to another type is called type casting.

Implicit type casting : It is converting lower data type to higher data type. Type checking is done by compiler. Type casting is done by JVM



Ex :- byte b = 12
int i;
i = b
Sop (i); // 12

Date / / /

→ long can be converted to float by internal representation extension of memory.

2)
```
long l = 85;
float f;
f = l;
SOP (f); //85
```

3)
```
int i = 30;
float f;
f = i;
SOP (f); //30.0
```

4)
```
char c = 'A';
byte b;
b = c;
SOP (b); // ERROR
```

5)
```
char c = 'A';
short s;
s = c;
S.O.P (s); //ERROR
```

6)
```
char c = "A";
int i;
i = c;
SOP (i); //65
```

7)
```
char c = 'A';
float f;
f = c;
S.O.P (f); //65.0
```

8)
```
double d = 35.5;
byte b;
b = d;
SOP (b); // Error (loss of precision)
```

implicit type casting is done by JVM
Explicit type casting is done by user

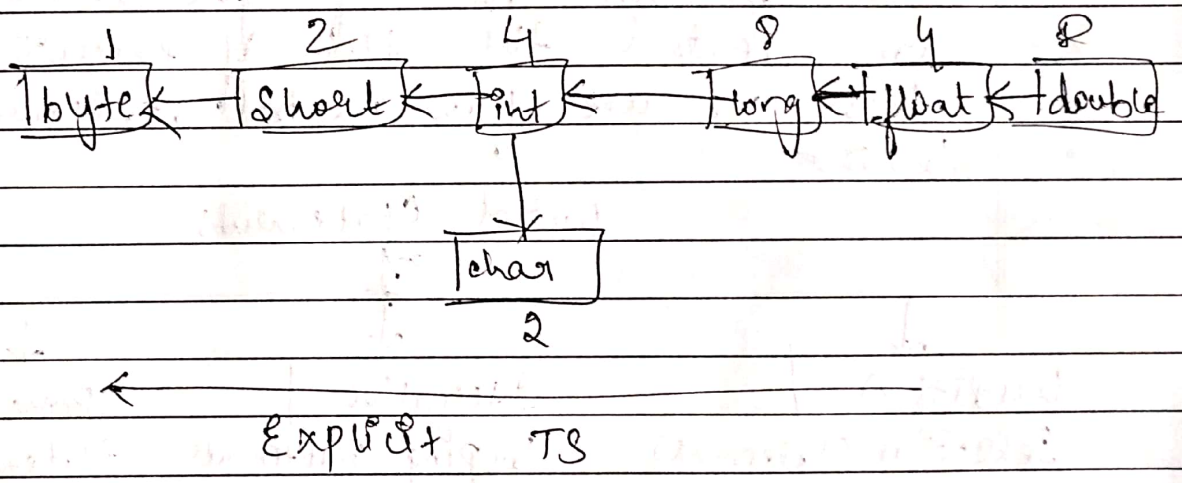
Note :- In the above problem we are trying to convert double to byte hence we get compilation error loss of precision. This problem can be overcome using

Date / / /

explicit type casting -

Explicit type casting : Converting the higher data type to lower datatype is called as explicit type casting. It should be performed by the programmer.

* Loss of data will not occur in implicit type casting as it is done by JVM. Whereas loss of data will occur in explicit type casting because it is performed by the programmer.



Ex 1: `int i= 120`
`byte b;`
`b= (byte) i;`
`sop (b); // 120`

2) `int i=129;`
`byte b;`
`b= (byte) i;`
`sop (b); // -127`

sign
1 0 0 0 0 0 0 1
-ve
0 1 1 1 1 1 1 0 1's
+1
-ve 1 1 1 1 1 1 1

2 | 129
2 | 64 - 1
2 | 32 - 0
2 | 16 - 0
2 | 8 - 0
2 | 4 - 0
2 | 2 - 0
1 - 0

Default execution of Java program is sequential execution

Date / / /

base 2

$1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$= 64 + 32 + 16 + 8 + 4 + 2 + 1 = 127$

-127

7th day
21/2/24

Flow control | Control Statements

Java program executes from top to bottom and if you want to control the flow of execution then we use flow control statements

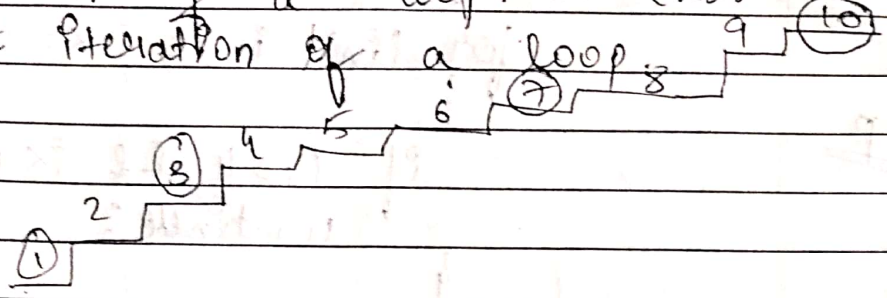
Control Statements

- Condition / Selection Statements
  - if
  - else - if
  - nested if
  - else if ladder
  - switch
- Iterative / looping statements
  - for
  - while
  - do while
  - for - each
    - Arrays
- Jumping statement
  - break
  - continue
  - return

Date / / /

Continue

Continue statement skips the current iteration of a loop. or ends the current iteration of a loop.



the action to miss out execution which are supposed to be executed, is interpreted as continue statement

- It is often used inside the loop & control structures
- when a continue statement is encountered the control directly jumps to the beginning of the loop for the next iteration instead of executing the statements of the current iteration

Syntax: continue;

```
while (test Expression) {
    if (test Expression) {
        continue;
    }
    SOPln ();
}
```

```
do {
    if (testExp) {
        continue;
    }
    SOPln ();
} while (testExp);
```

```
for (init ; testExp ; inc/dec or update) {
    if (testExp) {
        continue;
    }
    SOPln ();
}
```

Date / / /

Ex:
```
class Main {
  psvm (String [] args) {
    for (int i = 0; i <= 10; i++)
    {
      if (i > 4 && i < 9) {
        continue;
      }
      System.out.println(i);
    }
  }
}
```

o/p
1
2
3
4
9
10

Java continue with nested loop

```
while (testExpression) {
  while (test Expression) {
    if (test Exp) {
      continue;
    }
    // codes
  }
  // codes
}
```

```
int i = 1, j = 1;
while (i <= 3) {
  Sop ("outer loop:");
  while (j <= 3) {
    if (j == 2)
      j++;
    continue;
  }
  Sop ("Inner loop:");
  j++;
}
i++;
}}}
```

o/p: OL: 1
IL: 1
IL: 3
OL: 2
OL: 3

Labeled continue statement

```
continue label;

label:
while (testExp) {
  while (--i--) {
    if (--ii--) {
      continue label;
    } // codes } // codes }
```

first : for (int i=1; i<=3; i++) {
second : for (int j=1; j<=3; j++) {
if (i==2 && j==2) {
continue first;
}
sopln (i+" "+j);
}}}

o/p :
11
12
13
21
31
32
33

labeled continue discouraged to use

Underaged ~~or~~ overaged to get license.

class Demo
{
psvm (String [] args)
{
int age=35;
if (age<18) { underaged
sopln ("~~not eligible~~");
}
}
}

8th day

22/2/24

Date / / /

Object orientation

Object orientation is an approach used to solve the problem using different programming lang's (C++, Java)

The features of object orientation are

1. Object
2. Class
3. Encapsulation
4. Polymorphism
5. Inheritance
6. Abstraction

Note: The first object oriented programming lang is Simula in the year 1967.

Class - Class is a group of objects

* Class is a type
* Class is a blueprint or a prototype or a template.
* Class doesn't exist in realworld.
* Class doesn't occupy memory.

Syntax of class

```
access modifier class classname
{
    Variables
    methods
    objects
    constructors
    Inner class
    Interfaces
    Abstract class
    etc....
}
```

Date / / /

object → Cat, Dog, Tiger

↓

Real world entity

class → Animal

↓

Prototype

object - Object is a real world entity

* object is an instance of class
* object occupy memory.
* Every object consists of
  a) has part (properties)
  b) does part (behaviour)

(properties) Dog	(behaviour)
has	does
name	~~breed~~
breed	bark
color	eat
	sleep

Ex:

```
class Dog
{
    String name;
    String color;
    double cost;

    void bark()
    {
        Sopln(" Bow bow");
    }
    void eat()
    {
        Sopln(" Yummy");
    }
    Void Sleep() {
        Sopln(" Dog sleeps"); } }
```

Date / / /

Principles of object orientation

Rule 1: World is a collection of objects
Rule 2: No object is useless. Every object is useful.
Rule 3: Every object is under constant interaction. There exist no object that is isolated.
Rule 4: Every object belongs to a type and that type in object orientation we call it as class. (Class doesn't exist in real world. It is just the blue print).
Rule 5: Every object has something and every object does something

Camel case conventions:

1. Whenever we write a variable name or declare a variable it should be in small letters (lower case)

Eg: int age = 10;
int emp_salary = 25000;

2. Whenever we declare a method the name should be written in following way. The first word first letter should be small & the subsequent ~~letters~~ word 1st letter should be in capital

Ex: void bark
{
}

or

void dogBark()
{
}

3. Whenever we write a classname we follow 1st words 1st letter should be capital subsequent words first letter should also be capital

Ex: class Dog
{
≡
}

class DogLaunch
{
≡
}

4. object can be created in any one of the following ways
* new keyword
* instanceof method (newInstance())
* clone methods
* Factory method
* Deserialization.

⇒ Creation of object using new keyword
Steps to create object

① Declaration.
Declarate a variable to the type as shown below.

Dog d;
↑
ref varible.

② Instantiation means allocating the memory using new keyword.

Dod d = new Dog();
Instantiation.

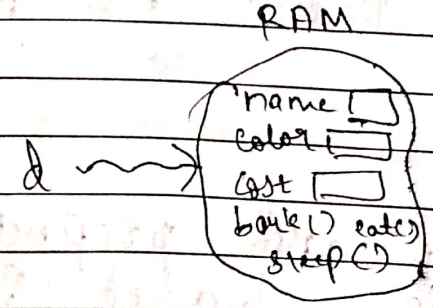
③ Variable and properties of the object type is loaded into memory

by JVM

Ex:-
```
class Dog Student
{
String name;
String color;        college
int double cost;     ROLLNO
void bark()          study
{
SOPln("Dog bow bow");
}
void eat()           sleep
{
SOPln("Yummy");
}
void sleep()         project
{
SOPln("Dog sleeps");
}}
class DogLaunch
{
psvm(...)
{
Dog d = new Dog();   (2) (1) (3)
d.bark;
d.eat;
d.sleep;
}
}
```

O/P

Dog bow bow
Yummy
Dog sleeps



Date / / /

Initializing the value of an object.

Objects values can be initialized 3 ways
* Using reference variables
* Using methods
* Using constructors.

Ex:
```
class Dog
{
    String name;
    String color;
    double cost;
}
Dog d = new Dog();
d.name = "Snoopy";
d.color = "white";
d.cost = 25000.0;
Sop(d.name);
Sop(d.color);
Sop(d.cost);
d.bark();
d.eat();
d.sleep();
```

```
void bark
void eat
void sleep
}
```

* In a class can we create multiple objects? Yes, in a class we can create n no of objects.

WAP to create 3 employee objects in an employee class with atleast 3 properties & behaviour & display the o/p.

Date / / /

Notes:- Command line arguments (CLA)

Arguments provided in the form of command line is called as CLA.

```
class Demo
{   psvm (String [] args.) {
    SOP (args[0]);
    SOP (args[1]);
    SOP (args[2]);
    SOP (args[3]);
}
}
```

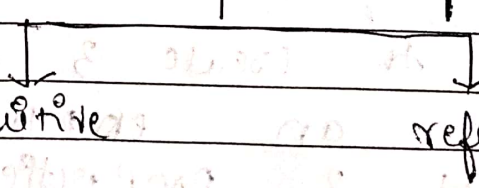
Reverse a no	Converting num to word
Palindrome no	Fibonacci
Armstrong no	Fascinating numbers
Prime no	
Random no	
Factorial no	

23/2/24

Day 9 Variables

Primitive variables → Stores the value.

Variables has been classified into 2 types



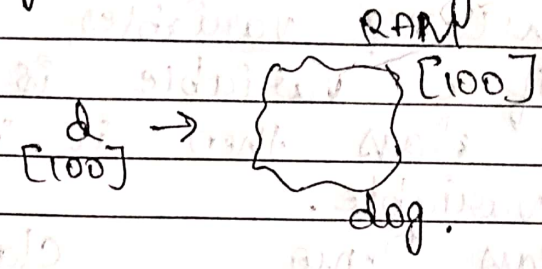
Primitive → If a variable holds the primitive data (value) then it is called as primitive variable.

Date / / /

int a = 10;

Reference → If a variable holds the address (memory location) then such variables are reference variable.

Dog d = new Dog()



Segments on RAM

without new keyword ↑ constant pool	Code segment	→ program
	Stack segment	→ activation records
	heap segment	→ object
non constant pool ↓ with new keyword	Static segment	→ Static members

Constant pool stores the variable that are declared without new keyword. It does not allow duplicates Non-constant pool stores the variables/objects that are declared with new keyword. It allows duplicate.

Based on the position & declaration of a variable again it is classified into

- instance variable → declared inside class
- local variable → declared inside method/block

Date / / /

ex :-

Instance

```
class Demo
{ int a;
  float b;
}
```

local

```
class Demo
{ psvm ()
  { int a;
    float b;
  }
}
```

→ block

Instance variables

If a variable is declared within a class then it is called as instance variable.

ex

```
class Demo
{ byte a;
  long b;
  Short c;
  int d;
  double f;
  boolean g;
  char h;
}
```

```
class Launch
{ psvm (...)
  { Demo d = new Demo()
    SOP (d.a); // 0
    (d.b); // 0
    (d.c); // 0
    (d.d); // 0
    (d.e); // 0.0
    (d.f); // 0.0
    (d.g); // false
    (d.h); // 0
  }
}
```

default values

* Explain Scope of instance variable

Instance variables are present inside heap segment. (Inside the object)
For instance variable memory is allocated during the object creation & it is deallocated during the object destruction. Instance variables are initialized with the default values provided by JVM and the memory is deallocated by the garbage collector.

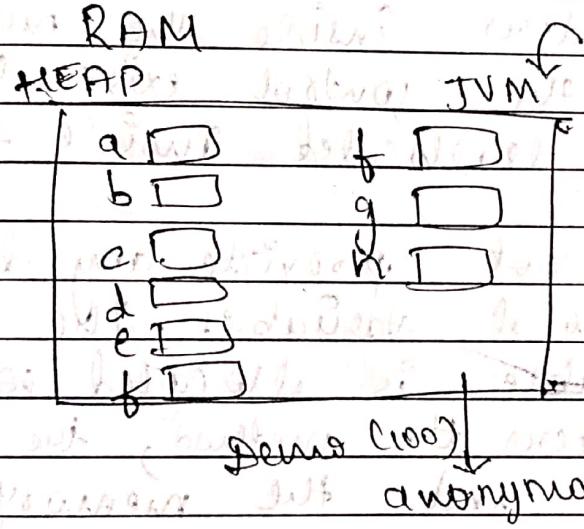
Date / / /

The scope of the instance variable is same as object.

Scope

memory allocated [during object creation]

memory deallocated [object is destroyed]



ex:

```
Class Demo
{ int a=20;
  PSVM (...)
  { SOP(a);
  }
}
```

o/p ⇒ compilation error [object not declared]

```
Class Demo
{ int a=20;
  PSVM (...)
  { Demo d = new Demo();
    SOP(d.a);
  }}
```

o/p ⇒ 20

Local variables

Local variables are such variables which are present inside a method as block.

Date / / /

ex:-

```
class Demo
{ psvm()
  { int x;
    sop(x);
  }
}
```

o/p ⇒ CE ⇒ x is not initialized

⇒

```
class Demo
{ psvm(...)
  { int x=20;
    sop(x);
  }
}
```

o/p ⇒ 20

Note: Memory is allocated as soon as control enters inside the method, as soon as the control exits the memory is deallocated inside the stack

Note: JVM does not provide any default value for local variables. The scope of local variable is allocated when the control enters the method, the memory is allocated and the memory is deallocated when the control leaves or exits the method.

Ex

```
class Demo
{ psvm(...)
  { int n=20;
    for(int i=1; i<=3; i++)
    { sop(i);
    }
    sop(n);
    sop(i)
  }
}
```

o/p

Compilation error because 'i' is a local variable only for the for loop, but it is called outside the loop.

Date / / /

Diff b/w instance & local variables.

Instance	local
* Can have default values	Cannot have default values
* ~~Memory is allocated~~ Variable that is declared in a class but outside a method	Variable declared within a method or a constructor
* Possible to use access modifier	Not possible to use access modifiers
* Destroy when destroying the object	Destroy when exiting the method or constructor.
* A variable that is bounded to object itself	* variable is used in a function or constructor

* Value type assignment & reference type assignment

Value type → assigning value of one variables to other variable.
whenever a value of one variable is assigned to another variable then it is called as value type assignment

ex:- int a=10; o/p : a=10
int b; b=10.
b=a;

Reference type → whenever the address of one variable is assigned to

another variable then it is called as reference type assignments. After the assignment all the reference variable would be pointing to the same object, modification done by any one of these reference variable could affect the same object.

Note One object can be pointed to multiple reference variables as shown below:

```
class Dog
{ String name;
  int cost;
}
class Launch
{ PSVM(...)
  { Dog d1 = new Dog();
    d1.name = "Leo";
    d1.cost = 1000;
    Dog d2;
    d2 = d1;
    d2.name = "Sniffer";
    d2.cost = 2000;
    Dog d3;
    d3 = d2;
    d3.name = "Sunny";
    d3.cost = 3000;
    SOP(d3.name);
    SOP(d1.name);
  }
}
```

o/p
Sunny
Sunny

Date / / /

Arrays

Explain length of 2D & 3D array.

Arrays

why do we need? → To enable multiple elements and values to be stored under single name.

int [1000] = [][][]...
0 1 2 3 ....

* To store small data we use variable
* To store large no of data we use array

what is array → collection of homogeneous elements stored in continuous location.

int a[] = new int[5]

Advantages

* Large amount of data can be stored
* Accessing of data is very simple.

Disadvantages

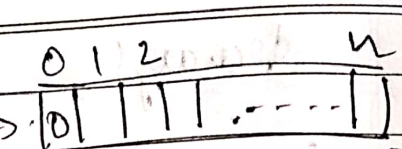
* It cannot store hetegeneous types of data
* Memory cannot be changed during runtime
* Array cannot increase or decrease its size once its initialized.

Note:- * In order to overcome this disadvantages we use collection.

* Array require contigeous block of memory.

Date / / /

1D Array



Syntax :- int a[];
int a[] = new int [20]; (2)
a[0] = 10; (3)
(1)

In this type the data can be stored either in rows or columns

3 steps :- Declaration of array (1)
Creating an array object (2)
Initialization (3)

Note :- Array is an object since it is created using new keyword.
Superclass of array is object.
When the array is created the default value will be added by the JVM based on the datatypes.

int a[] = new int [3];
a → [0|0|0]

float a[] = new float [3];
a → [0.0 | 0.0 | 0.0]

The declaration of the array can be done in any one of the following ways.

int a[]
int[] a;
int[] a;

→ Can we specified size of an array during declaration ? → At the time of array declaration size should not be specified size should be specified only during creation of array.

int a[5]; ✗
int a[];
int[] a = new int[3];

→ Can we specify array size or it is valid to create an array with size 0.
int a[] = new int[0];

→ Can we specify negative array size ?
No, an array size cannot be specified in negative, if we do so then during execution we get a message i.e -ve array size exc

ex:
```
class Demo
{
  PSVM
  {
    int a[] = new int[5];
    a[0]=10; a[1]=20; a[2]=30;
    SOP(a[0]); SOP(a[1]);
    SOP(a[2]); SOP(a[3]); SOP(a[4]);
  }
}
```
o/p ⇒ 10, 20, 30, 0, 0

WAP to store & print marks of 5 students
WAP to store 5 students marks using while.

2-D Array.

1-D Array
2-D —11—
3-D —11—
Jagged —11—

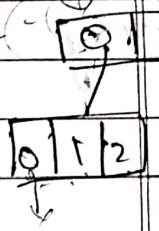
length of 2D & 3D array

Day 10
24/2/24

Date / / /

Program for length of 2-D Array

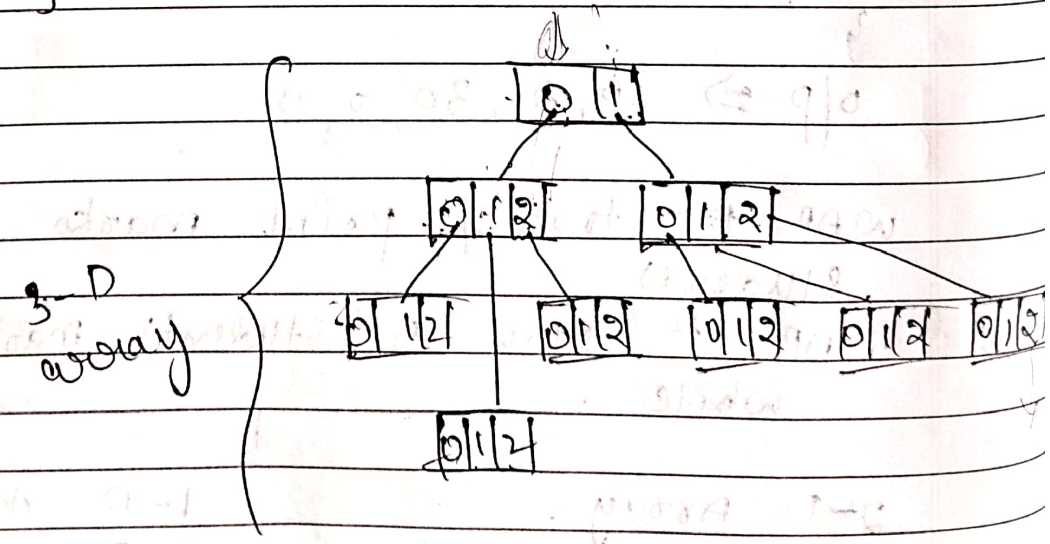
```
class twoDimensionalArray
{   psvm (....)
{
    int [][]a = new int [2][3];
    SOPln(a.length); // 2
    SOPln(a[0].length); // 3
    SOPln(a[1].length); // 3
    SOPln(a[0].[0]length); // C.E
}}
```



→ Row

Program for length of 3-D Array

```
class ThreeDimensional Array
{   psvm (....)
{
    int [][][]a = new int [2][3][3];
    SOPln (a.length); // 2
    SOPln (a[0].length); // 3
    SOPln (a[0][].length); // 3
    SOPln (a[1][2][2].length); // C.E
}}
```



Date / / /

For - each loop

This loop will only be used in arrays.

The syntax is

```
for (datatype var : array)
{
    body | statements ;
}
```

ex: x =

0	1	2	3
10	20	30	40

o/p : 10 20 30 40

for loop

```
class Demo
{   psvm (...)
{
  int x[] = {10, 20, 30, 40};
for (int i=0; i<3; i++)
  {
    sop (x[i]);
  }
} }
```

for - each loop

```
class Demo
{  perm(...)
{
 int x[] = {10, 20, 30, 40};
 for (int i : x)
 {
    sop (i);
 }
} }
```

for - each loop

i) index 0

ii) Automatically increment.

* WAP to print 1-10 by skipping 6 using for - each loop.

```
public class Main {
    ps vm (...)
    int[] numbers = {1, 2, 3, 4, 5, 6, 7,
                     8, 9, 10};
        for (int num : numbers) {
            if (num == 5) {
                continue;
            }
            SOPln (num);
        }
    }
}
```

EMP
or

Anonymous array

Anonymous array are such arrays which doesn't have name. It can be declared & initialized in a single line

Anonymous array can be single or multidimensional. It can be passed as an argument to the method.

If the array usage is only once, then we can make use of anonymous array.

```
int [] x = new int [3];
int [] x = new int [] {10, 20, 30};
int [] x = {10, 20, 30};
```

1D ⇒ new int [] {10, 20, 30};
2D ⇒ new int [] [] {{10, 20, 30}, {40, 50}}

Ex for anonymous array (1D)

```
class Demo
{   psvm (...)
{
    Addition a = new Addition();
    a.add(new int[]{10,20});
}
}
```

```
class Addition
{
    void add(int[] x)
    {   int sum=0;
        for(int i=0; i<x.length; i++)
        {
            sum = sum + x[i]; }
        sop(sum);
    }
}
```

x → 10 20

i=0 0<2 T
i=1 1<2 T
i=2 2<2 F

o/p
30

→ Ex for anonymous array (2D)

* WAP to calculate sum of {10, 20, 30} and {40, 50}

```
class Addition
{
    void add(int[] x)
    {   int sum=0;
        for(int i=0; i<x.length; i++)
        {
            for(int j=0; j<x[i].length; j++)
            sum = sum + x[i][j];
```

i=0 i<3 T

```
        SOP (sum);
    }
}
```

o/p
150

```
class Demo
{
    psvm(...)
    {
        Addition a = new Addition();
        a.add(new int[][]{{10,20,30},{40,50}});
    }
}
```

Methods / Functions

Methods are the set of statements used to perform some specific task.

Syntax :

```
access modifier   return type   method name(parameter 0...n)
{
    body / statements;
}
```

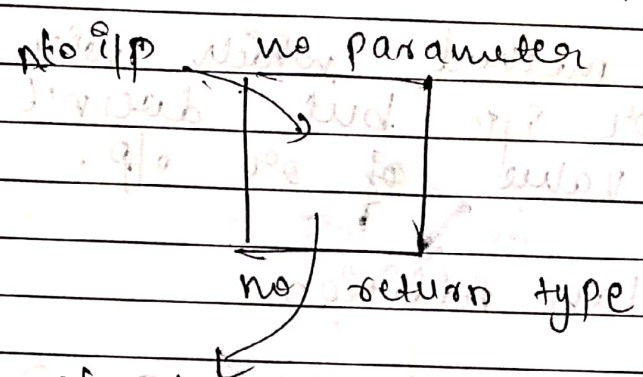
Ex:
```
public void add()
{
    int a = 10;
    int b = 20;
    int c = a+b;
    SOP(c);
}
```

All local variables will be stored inside a stack segment.

Date / / /

There are 4 types of methods.

1) Methods which will not accept any parameter or i/p & it will not return any value or o/p



Ex:
```
class Addition
{
    void add()
    {
        int a = 10;
        int b = 20;
        int c = a+b;
        Sop(c);
    }
}
class Demo
{
    perm(...)
    {
        Addition a = new Addition();
        a.add();
    }
}
```

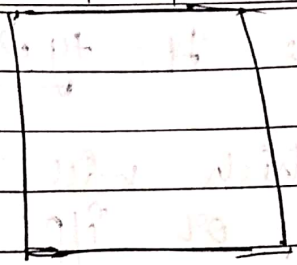
o/p
30

II

Accept parameter

Date / / /

2)

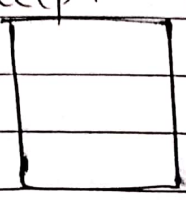


No return type

A method which will accept parameter or i/p but doesn't return any value or o/p.

Ex
```
class Addition
{
  void add (int a, int b)
  {
    int c=a+b;
    SOP (c);
  }
}
class Demo
{ psvm (...)
  {
    Addition a = new Addition ();
    a.add (10,20);
  }
}
```

3) A method which accepts parameter and also returns. output



Ex
```
class Addition
{
    int add (int a, int b)
    {
        int c = a+b;
        return c;
    }
}
class Demo
{
    psvm (...)
    {
        Addition a = new Addition ();
        int sum = a.add (10, 20);
        sop (sum); }
}
```

called method / formal parameters

calling method / actual parameter

H) Method which doesn't accept parameter or i/p but return a value.

Ex
```
class Addition
{
    int add ()
    {
        int a = 10;
        int b = 20;
        int c = a+b;
        return c;
    }
}
```

```
class Demo
{
    psvm (...)
    {
        Addition a = new
                Addition ();
        int sum = a.add ();
    }
}
```

Method
|
return
|

primitive Datatypes	Non-primitive
int	array
double	String
byte	object
char	constructor
short	etc
float etc	

→ A method returning an array.

```
class Test
{
  int[] fun()
  {
    int[] a = {10, 20, 30};
      return a;
  }
}
class Demo
{
   psvm(....)
   {
     Test t = new Test();
     int[] x = t.fun();
for (int i=0; i<x.length; i++)
   {
     sop(x[i]);
   }
}
}
```

→ A method returning a string

```
class Test
{
  String fun()
  {
    String str = "Hello";
    return str;
  }
}
class Demo
{
  psvm(...)
  {
    Test t = new Test();
    String s = t.fun();
    SOP(s);
  }
}
```

o/p
Hello

Mutators and Accessors

Based on the object state modifications the methods are classified into

1. Mutators or Setters
2. Accessors or getters.

1. Mutators are used to set or modify the value in objects.
2. Accessors are used to get the value from the object.

This is also an example of encapsulation

Date / / /

Ex:

```
class cred
{
private String username;
private int password;
   void setun (String un)
   {
      username = un;
   }
   void setpwd (int pwd)
    {
      password = pwd;
   }
   String getun ()
    {
      return username;
    }
   int getpwd ()
    {
      return password;
    }
}
class Demo
{
   psvm (....)
  {
  cred c = new cred ();
      c. setun ("Prashanth");
      c.. setpwd (123);
       sop (c.getun ());
       sop (c. getpwd ());
    }
}
```

o/p
Prashanth
123

Date / / /

Explain Pass by Value in Java.

```
class Test
{
  void swap (int x, int y)
  {
    int temp;
    temp = x;
    x = y;
    y = temp;
  }
}
class Demo
{
  psvm (....)
  {
    int a = 10; int b = 20;
    Test t = new Test ();
    SOP ("Before swapping");
    SOP ("The value of a is " +a);
    SOP ("The value of b is " +b);
    t.swap (a, b);
    SOP ("After swapping");
    SOP ("The value of a is " +a);
    SOP ("      "  "  b is " +b);
  }
}
```

x=10 y=20.
temp=x
temp=10
x=y.
x=20
y=temp
y=10.

O/P

Before swapping
a = 10
b = 20

After swapping
a = 10
b = 20.

Assignment

After swapping
a = 20
b = 10

Date / / /

Note:- In C language, we can write any number of functions, but the function name should be unique (2 func^n's should not have same name)

In the below pgm even though we are performing addition operation but still we should remember multiple func^n name which is very complex coz in C func^n names shud be unique. this problem can be overcome using method overloading in Java.

```
class Test
{
int add (int x, int y)
{
    return x+y;
}
float add (float x, float y)
{
    return x+y;
}
double add (double x, double y)
{
    return x+y;
}
int add (int x, int y)
{
    return x+y;
}
double add (int x, double y)
{
```

```
        return x+y;
    }
    double add (float x, double y)
    {
        return x+y;
    }
    float add ( float x, int y)
    {
        return x+y;
    }
    double add (double x, float y)
    {
        return (x+y);
    }
    double add (int x, float y, double z)
    {
        return x+y+z;
    }
}
class Demo
{
    psvm (....)
    {
        int x=10; int y=20; int z=30;
        float a = 10.1f; float b=20.1f;
                         float c = 30.1f;
        double p=30.2; double q=40.2,
                       double r= 50.2
        Test t= new Test ();
        SOP (t. add (x, a));
        SOP (t. add (x,a, p) // 50.3;
    }
}
```

Date / / /

Method overloading in Java is defined as 2 or more methods within a class having same name but diff no of parameters, or in simple words; multiple methods with same name is called as method overloading.

* whenever a call is made for overloaded method the compiler will resolve the method calls in the following way.
  - → Name of the method
  - → no of parameters
  - → datatype of parameters
  - → order of parameters

—u—

Ex:-

```
class Demo
{
    void add (int x, float y)
    {
        int z;
        z = x+y;
        SOP (z);
    }
    float add (int x, float y)
    {
        return (x+y);
    }
}
class Test
{
    psvm(...)
    {
        Demo d1 = new Demo();
```

```
int a = 10;
float b = 10.1f;
d1.add(a,b);
}
}
```

o/p

CE - add() is ambigous.

Note:- ① Method overloading will never consider the return type

② when a call is made for the overloaded method to resolve the method call only if factors are considered.

Interview question

In method overloading what is overloaded ?

Ans In method overloading, nothing is overloaded, it is virtual polymorphism, the user will be under illusion that one method is overloaded with multiple activities but in fact there are diff methods doing these respective tasks.

⇒ Method overloading using numeric type promotion

Date / / /

Ex 1:-

```
class Demo
{
add (int x, long y)
{
  return x+y;
}
add (double x, double y)
{
  return x+y;
}
}
```

```
class Test
{ psvm (---)
{
Demo d = new Demo();
  int a=10;
  int b=20;
float c = 1.1f;
  float d= 2.2f;
sop (d.add (a,b));
sop (d.add (c,d));
}
}
```

1) (int, int) → add (int, long)
2) (float, float) → add (double, double)

Note:- During the method overloading if the datatypes of the parameters doesn't matter then before displaying the error compiler will try to promote the data type to next available type & try to resolve the method call.

* In the above pgm (Ex1) even though the class doesn't contain a method add (int, int) & add (float, float) still no error is displayed coz the compiler has performed numeric type promotion

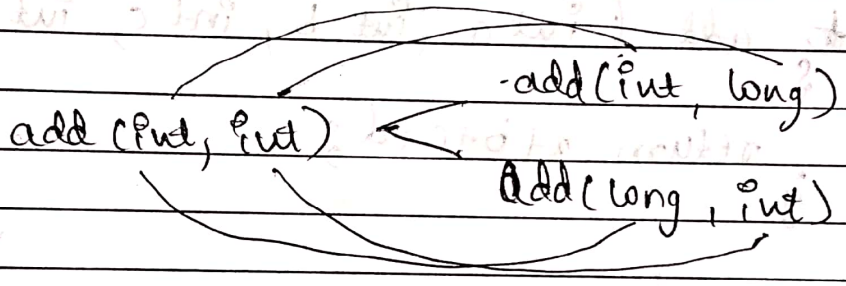
Ex 2:-

```
class Demo
{
long add (int x, long y)
{
   int z;
   z = x+y;
   SOP (z);
}
long add (long x, int y)
{
   return x+y;
}
}
```

```
class Test
{ psvm (....)
{
Demo d1 = new Demo();
   int a = 10;
   int b = 20;
d1. add (a, b);
}
}
```

o/p

CE: Ambigous.



Note:- (1) Problems with Numeric type promotion if in the above prgm, during the type promotion of a method call is matched with multiple methods then the compile time error would be displayed as ambiguous.

(2) Using return keyword we can only return one value where multiple values cannot be written.

Ex: 1) return x+y; ✓ Valid

2) return x+y, x*y, x/y; ✗ Invalid

Day 41
26/2/24

Date / / /

Var - args

```
Class Test
{
 int add (int a, int b)
  {
    return a+b;
  }
int add (int a, int b, int c)
   {
     return a+b+c;
   }
int add (int a, int b, int c, int d)
   {
     return a+b+c+d;
   }
}
  class launch
{
  psvm (String [] args)
  {
    Test t= new Test;
    SOPln (t.add (10, 30));
    SOPln ( t: add (10,20,30,40));
    SOPln ( t.add (10,20,30));
  } }
```

The problem in the above pgm is identified by sun microsystem where single add method consists of n no. of arguments when the soln of this problem was introduced in Java 1.5 where overloaded methods can be simplified using var-args

```
class Test
{
  int sum = 0;
int add (int ... a)
{
  for (int i : a)
  {
    sum = sum + i;
  }
  return sum;
}
}
```

```
class Launch
{ psvm (String[] args)
{
  Test t = new Test();
  SOPln(t.add(10,30));
  SOPln(t.add(10,20,
        30,40));
  SOPln(t.add(10,20,
        30));
  SOPln(t.add());
}
}
```

Note:- (1) Var-args allow the methods to accept 0 or multiple arguments. If we do not know how many parameters we need to pass to the methods. Then we can go for Var-args. Advantage is we do not have to provide overloaded methods.

```
      0...n
add (int ... a)
      ↓
    [] - 1D
```

(2) Var-args can be any primitive data-type.

a) 1) byte ... a ⟷ byte [] a
   2) short ... a ⟷ short [] a
   3) int ... a ⟷ int [] a
   4) long ... a ⟷ long [] a
   5) float ... a ⟷ float [] a
   6) double ... a ⟷ double [] a
   7) char ... a ⟷ char [] a
   8) Boolean ... a ⟷ Boolean [] a

(3) Valid syntax for var args is

1) int ... a — int [] a
2) int ... a — int [] a
3) int ... a — int [] a

(4) Var-arg parameter is internally converted into 1D array.
int ... a → int [] a → 1D array

(5) add (int ... a) it means add method can accept 0 or more integer parameters.

(6) Type promotion in var args is possible from the below example.

```
class Test
{
  float sum = 0.0f;
  float add (float ... a)
  {
    for (float i : a)
    {
      sum = sum + i;
    } return sum;
  }
}
class Demo.
{ psvm (String [] args)
  {
    Test t = new Test ();
    SOP (t.add ());
    SOP (t.add (1.1f, 2.1f));
```

```
        SOP (t.add(1.1f));
        SOP(t.add(10,20));
        SOP(t.add(10,20.1f));
        SOP(t.add(10,20.1));
    }
}
```

(7) In a method along with var arg parameter, we can also have other parameters.

```
class Test
{
  void display(int num, String... s)
  {
    SOP("number value is" +num);
    for(String i:s)
    {
      SOP("String value is" +i);
    }
  }

  class Demo
  {  psvm(...)
  {  Test t= new Test();
       t.display(10);          // 10
       t.display(10,"Hi");     // Hi
       t.display(10,"Hi","Bye"); // 10, Hi, Bye
       t.display();            // Error
       t.display(10,10);       // Error
       t.display(10,"10");     // 10, 10
    }
  }
```

t.display("Hi",10); ← // Error