DAY-1 13/2/24

Basic Java:

(1) Difference b/w RAM and ROM

RAM	ROM
(i) RAM stands for Random Access memory	(i) ROM stands Read only memory
(ii) It's an a violatile	(ii) It is non violatile
(iii) Speed of RAM is higher	(iii) Speed of ROM is slower
(iv) Data in RAM can be modified, erased or read	(iv) ROM data can only be read only, it cannot be modified or erased
(v) Data Stored on RAM can be accessed by CPU	(v) Data should be loaded to RAM first, then accessed by CPU.
(vi) Large size with higher capacity.	(vi) Small size with less capacity
(vii): CPU cache, primary mem.	(vii) firmware, micro-controller
(viii) Types static, dynamic	(viii) Programable ROM, Erasable " ", Electrical " ", Mask ROM

medaulin

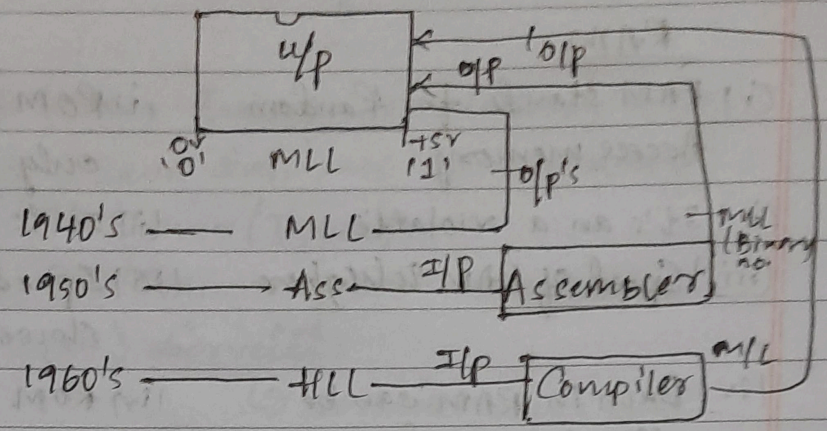
Point's to Remember:

Saving: The process of storing the data from ram to harddisk is called saving.

Loading: The process of fetching data from hard disk to ram is called loading.

Programming:

Evolution Of Programming Language:



	MLL	Ass	HLL
Addition	10111100	add	+
Subtraction	00111100	sub	-
Multiplication	11001000	mul	*
Division	11000111	div	%

add(3,4)

numonics

limits 'memory'

'KUSHAL' - 75 85 83 72 65 76

7 - 0111 01110101 10000101 10000011

8 - 1000 01110010 01100101 01110110

5 - 0101

3 - 0011

2 - 0010

6 - 0110

points to Remember:

MLL (Binary code) - '0's and '1's
ALL (Assembly level Lang) (Mnemonics)
HLL (High level lang. (Symbols).
Assembler: It is software with take assembly level lang. as i/p & produces MLL or Binary code as o/p.

Compiler: It is an software which HLL as i/p & produces MLL as o/p.

* History of High level Programming lang.

1962
BCPL (Basic combined Prgm lang.)
Martin Richards
↳ More Memory
↳ Unstructured

1969
B
Ken Thomson
↳ less memory
↳ Unstructured

1972
C
Dennis Ritchie
↳ less mem.
↳ structured
Disadv:
↳ Security

1982
C++
Banjre Stroustroop
↳ less mem.
↳ structured.
↳ security
Disadv:
↳ Platform dependent

1992 (unoffical) (B revision)
Java
(1996) official
↳ less mem.
↳ OOPL
↳ Security
↳ platform independent

DAY-2 14/2/24

* Structured, Unstructured and Object Oriented

⇒ Unstructured:

↳ It doesn't have any structure

* lines of code often repeated.

```
add()          add()
{              {
 ---            ---
}              }
sub()
{
 ---
```

Disadvantage: (1) More mem. (2) Time consuming (3) Debugging

⇒ Structured:

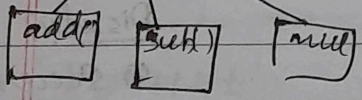
↳ Using a function / modules are seperated.

↳ Their is a seperate main function.

* All modules are connected to main module.

test.java

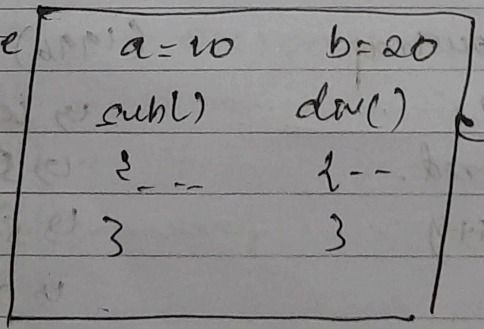
```
add();
sub();
add();
mul();
```



* Disadvantage:

Security.

⇒ Object Oriented:



Tirn Examples

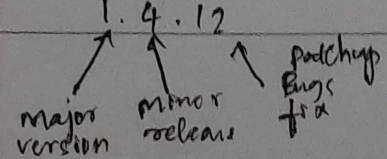
Versions Of Java:

Java β - 1995	Java 9 - Sep 2014
Java 1.0 - Jan 1996	Java 10 - Mar 2018
Java 1.1 - Feb 1997	Java 11 - Sep 2018
Java 1.2 - Dec 1998	Java 12 - Mar 2019
Java 1.3 - May 2000	Java 13 - Sep 2019
Java 1.4 - Feb 2002	Java 14 - Mar 2020
Java 5.0 - Sep 2004	Java 15 - Sep 2020
Java 6.0 - Dec 2006	Java 16 - Mar 2021
Java 7.0 - Jul 2011	Java 17 - Sep 2021
Java 8.0 - Mar 2014	Java 18 - Mar 2022
LTS (long term support)	Java 19 - Sep 2022
	Java 20 - Mar - 2023
latest version ←	Java 21 - Sep 2023
	Java 22 - Mar 2024
	Java 23 - Sep 2024

History of Java:

* In 1991 Sun Microsystems formed a group of 30 members, called as Green Team headed by James Gosling (father of java), patrick naughton, mike sheridel. However, it was suitable for internet in the year 1996 first version of Java was realized (java 1.0). Initially for technology develop by Sun microsystem they gave the name as Green then renamed to Oak & finally to Java.

Points: * Now, java is not product of Sun microsystem it is acquired by Oracle in 2010.



Interview qu.

* What is meant by platform dependent & independent. Is java is platform independent or not?

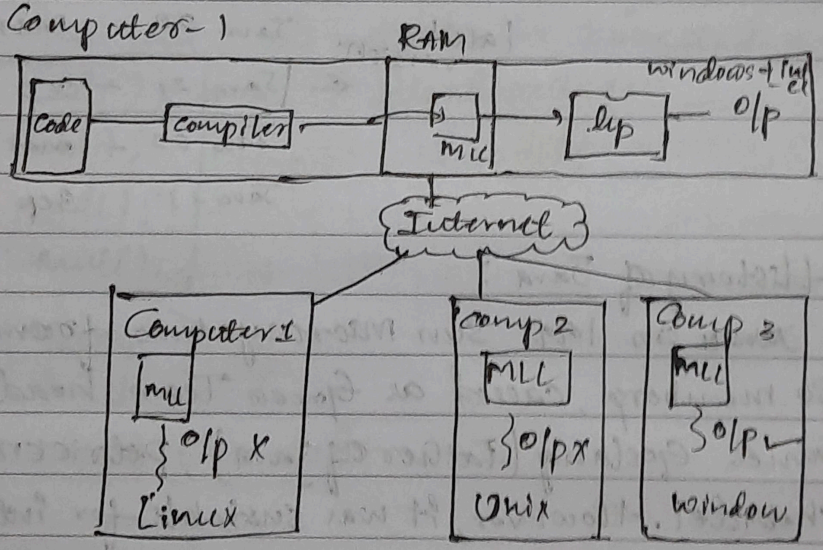
* Platform dependent?

* The program is compiled in one operating system & if it is executed only in the same OS, then it is platform depedent or Machine dependent.

Ex: C, C++

Note:

Platform is combination of software (OS) & Hardware (Processor) Environment.



* Platform independent:

If the prgm is compiled in OS can be executed in any other Operating system so it is called as platform independent.

Ex: java.

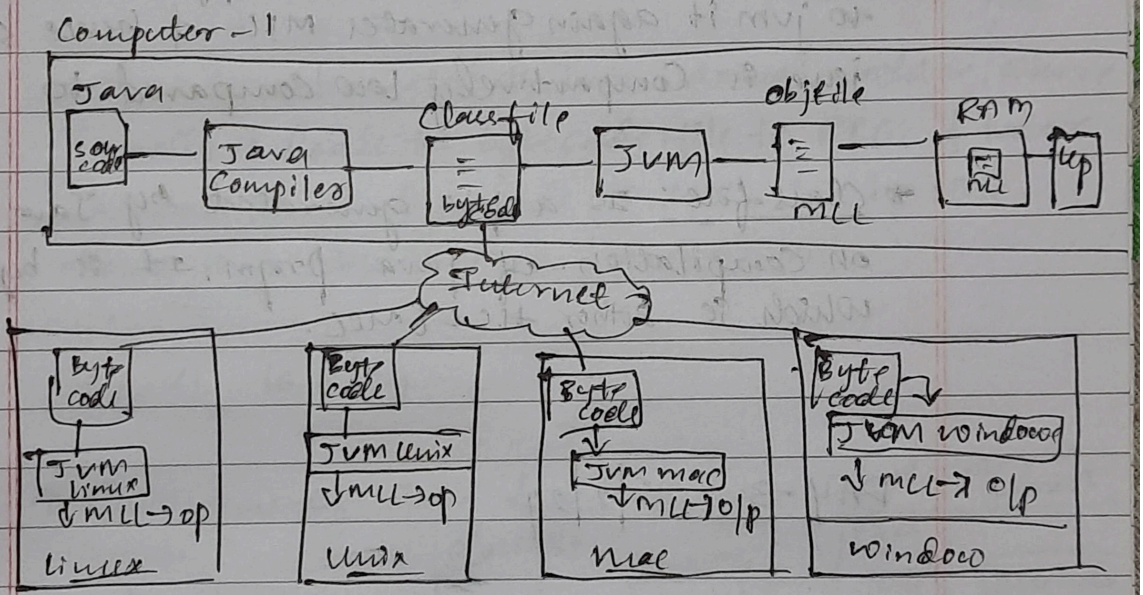
Note:

(i) What is WORA (Write Once Read-Anywhere.)

A:

If a java prgm is writtened & compiled in one OS, it can be executed in any other OS. Hence it is called as WORA.

(ii) Java is platform independent, but jvm platform dependent. Since JVM is coded in C language.



- Mention the below terms as platform dependent or independent.

(i) Java prgm - platform independent

(ii) JVM - platform dependent

(iii) MLL - " "

(iv) ByteCode - platform independent

- Interview question:

Why java is realitively slow, when compared to C language.

(C) ① (0 & 1's)
HLL → MLL
compiler

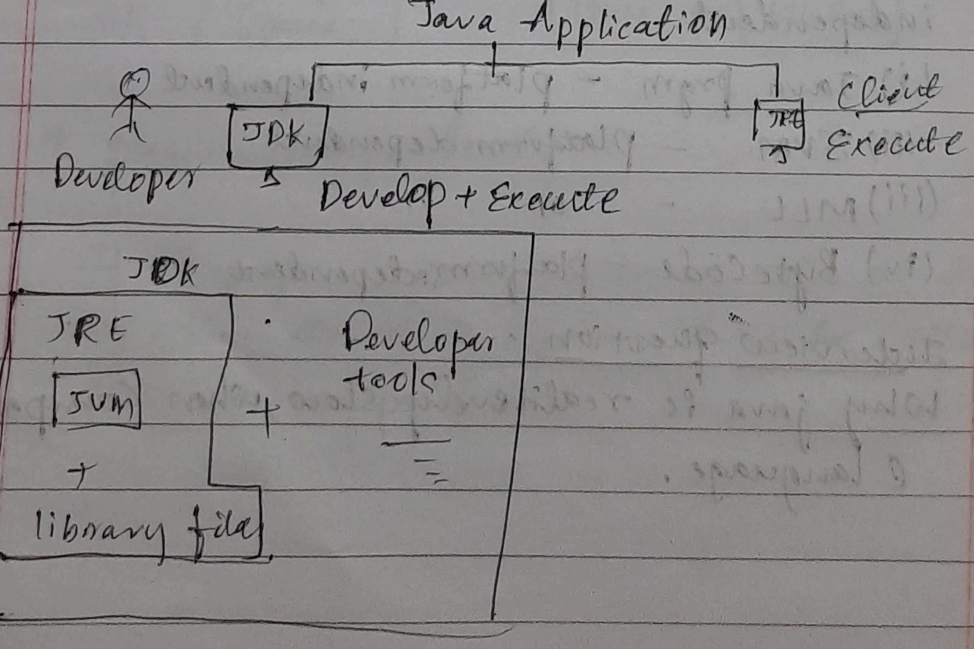
(Java)
HLL ① → HLL/MLL → ② → 0's & 1's MLL
compiler Bytecode JVM
Interpreting

* In C lang, if the prgm is given to the compiler it generates binary code, which can be executed. In case of java prgm, when it is given to compiler, it generates byte code, these byte code again given to jvm it again generates MLL & hence speed of java is Compartively less compared to C.

* Class file: Is a file generated by Java compiler on compilation of java prgm. It is byte code which is either HLL/MLL.

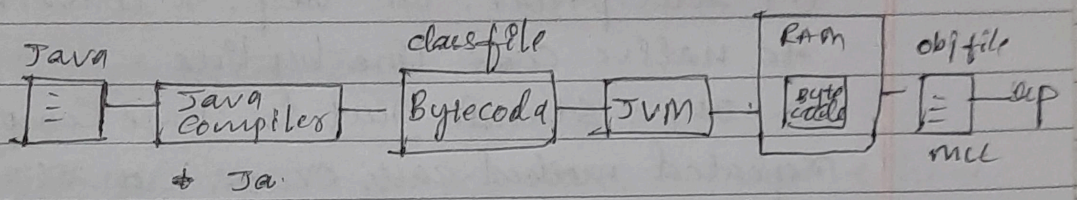
DAY-3 15/2/24

* What is JDK, JRE and JVM?



9 - 11
11 - 11:30 - Tea
11:30 - 1:00 pm ... ds.

JDK is abbreviated as Java development kit, used by the developer to write code & execute it. It contains both JRE and development tools, where development tools consists of some tools like javac & where JRE provides a Environment, where the JVM & some library files are present.

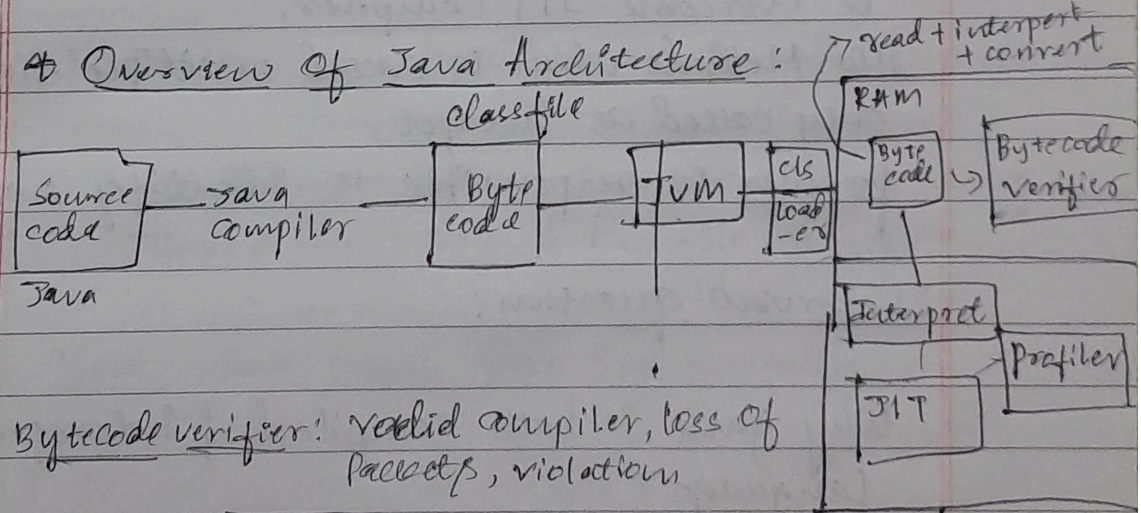


* Ja

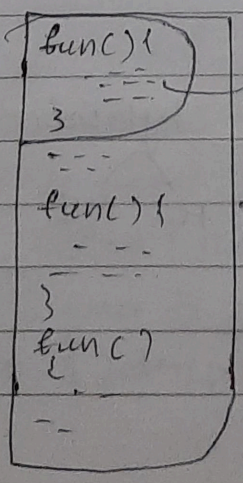
* JVM is abbreviated as Java virtual machine, where initially it loads the bytecode file to RAM & later it converts the bytecode to mcl line by line. Hence it is called Java Interpreter

DAY - 4 16/2/24

* Overview of Java Architecture:



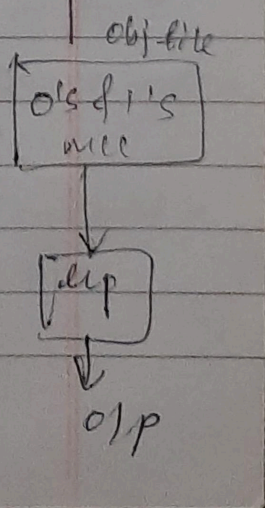
Bytecode verifier: valid compiler, loss of packets, violation



→ hotspot

count = 0, 1, 2

Profiler



JVM consists of Classloader, which helps to load class file on to the RAM. Byte code verifier helps to check whether bytecode present in class file is generated by valid compiler or not.

Execute Engine:

(1) Interpreter: This helps to convert byte code to native code line by line.

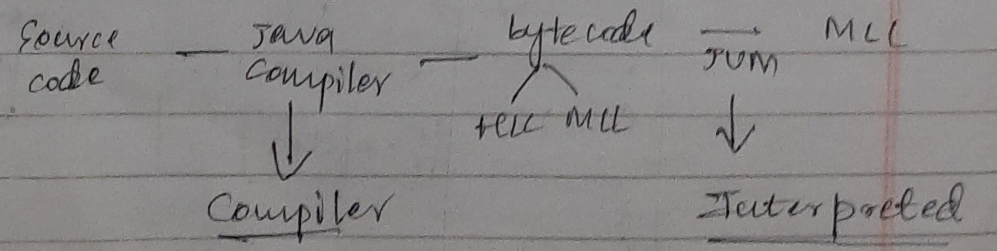
(2) JIT: stands Just In Time Compiler, whenever repeated method calls occur, then JIT is activated & helps to convert Bytecode to Native code. The native code will be used directly for repeated method calls, the problem in Interpreter is whenever method is called multiple time, then it will execute line by line & convert bytecode to native code, again & again. These problem can be overcome JIT Compiler.

The Methods which is called multiple times is only called as Hotspot.

Profiler is responsible to identify hotspot.

Interview question:

Why java is both compiled & interepreted prgm language?



* Initially, java prgm is compiled by java compiler which in-turn generates a bytes which is not executable by processor, hence, these byte code is given to JVM which interpret the code line by line & converts it into m/c. Hence now processor can execute it.

Features Of Java:

(1) Simple: It simple bcz, unlike C it doesn't have pointers & multiple inheritance.

(2) Platform Independent

(3) Architecture Neutral: Just in C lang, the data types takes up two diff memory, but in java it takes up the same memory.

(4) Portable: Java bytecodes are portable provides easy transfer

(5) Multi Thread: Parallel Execution of Prgm using Thread inbuilt class & Runnable Interface

Note: Java Prgms Executed Sequentially

(6) Security: Bytecode verifier.

(7) Object-Oriented: ↳ Everything in java is object.

(8) Robust: Strong bcz, strong typed, exception handling.

(9) High performance: Achieved by JIT.

(10) Distributed: The Application logic is distributed in multiple computer to provides better performance.

(11) Compiled & Interepreted:

My First Java Program:

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

* Main method should be made as public in order to make it visible to Operating System
* Main method is made as static becz:
  (i) To Make method accessable for OS.
  (ii) To Access method without creating an instance of a cls.
* System.out.println is the pipe that is used to display the messages that has to be displayed on o/p screen.
* Javac is cmd used to invoke the compiler, which converts java prgm into cls file.
* Java cmd is used to invoke JVM, which converts byte code into native code & help for execution.

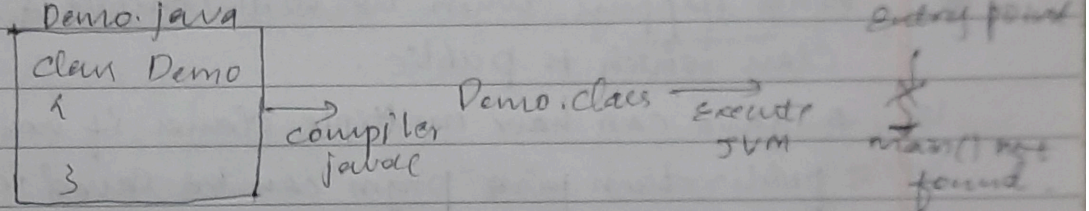
Note: It is good practice to save the java prgm name as a same name as Class name.

```
sop("Hi")
sopln(",")
sopln("welcome")
sop("to")
sopln("JFS Course")
sopln("!")
```

Very Good morning.
Greetings from
Destination IT
, welcome to
*
JFS Course
! Byee

* What happens when you Compile & Execute Empty class file in java?

↳ It Compiles & generates the class file but execution does not takes place bcz, their is no main method.



↳ It is not mandatory for java prgm the file name to same as cls name but their are some restriction.

(1) An Empty cls file can be compiled & cls file generated in the same name given to class.

If we try to execute Empty cls.file then we get error.

* psvm can be defined in one of following ways:

var args...

psvm (String [] args)	psvm (String... args)
psvm (String[] args)	psvm (String ... args)
spvm (String[] args)	spvm (String... args)
psvm (String [] args)	
psvm (String args[])	

* Can we have multiple Empty Classes in Java file?

↳ In a java prgm, we can have multiple classes. The java file name need not to be class name. When we try to compile java prgm separate class file will be generated with same as that of " name & when we Execute main method not found Error got displayed.

Democlass

```
class Demo1{ ... }
class Demo2{ ... }
class Demo3{ ... }
```

Java compile → Democlass, Demo2.class, Demo3.class

JVM → Exe → main() not found

* What happens, when we will compile & Execute a class which is public.

↳ If java can have multiple classes, if non of the class is public, then java prgm can be saved with any file name, if class is made is as public then given prgm name should be same as class name which is made as public. There is restriction in order to name the java prgm.

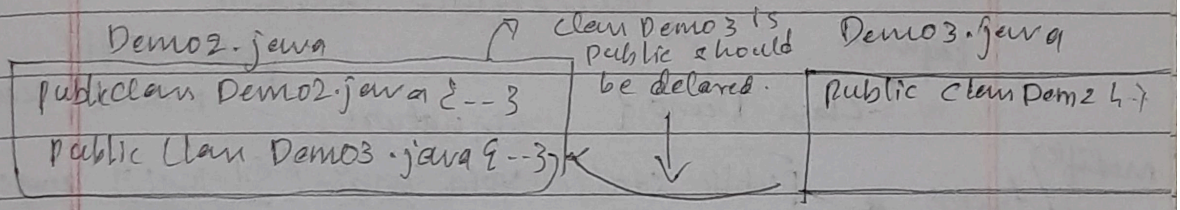
Demo1.java

```
class Demo1
{ ... }
public class Demo2
{ ... }
class Demo3
{ ... }
```

compile → Javac → Demo1.java

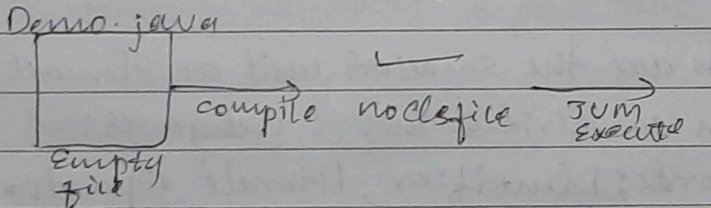
* What happen when two Class are public ?

↳ Multiple Class can't made as public in java.



* What | Is it possible to Compile & Execute Empty java file.

↳ An Empty java file can be Compiled. The compiler will not throw an Error, no class files will generated. Since, no class file will be avail|ble we can't execute any Empty java prgm.

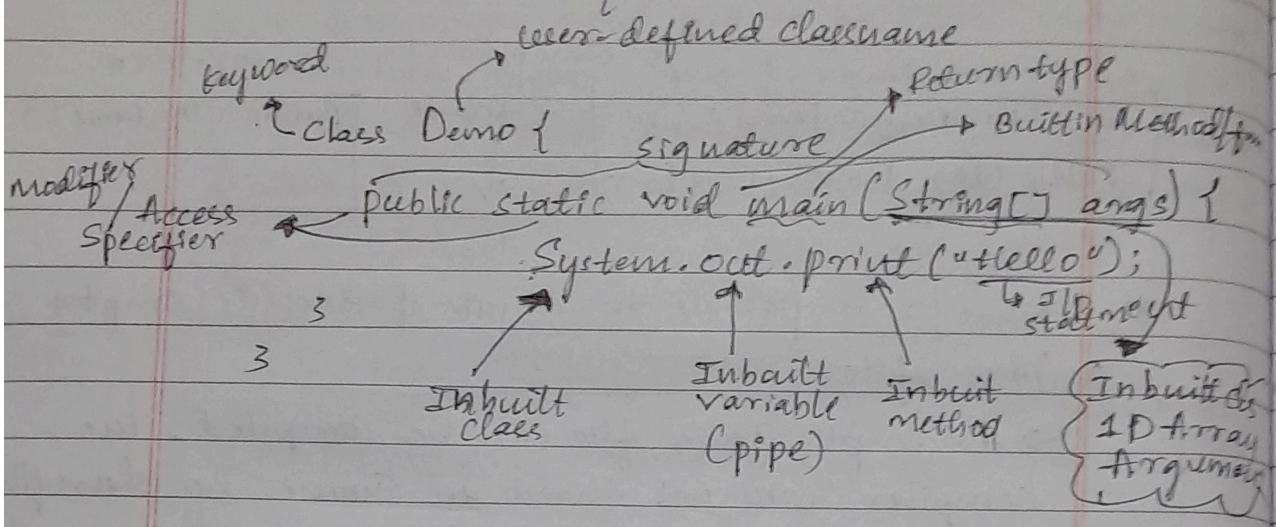


* Can, I compile & execute java prgm without providing the Class name just given .java file extention.

↳ It compiles & Execute the java using .java

DAY-~~7~~ 19/2/24

* Technical Terms:



* Tokens:

Tokens are the smallest unit or Element in the program
Tokens are divided into 5 categories:
Keywords, Identifier, literals, Operators & special symbols.

(i) Keywords:

↳ They are also called as reserve words, where the meaning of word is predefined to Java Compiler
Totally there are 50 keywords present:

* Flow Control keywords:
if, else, switch, case, default, while, do, for, break, continue, return & goto

* Datatype keywords:
byte, short, int, long, float, double, boolean, char & enum

* Access modifier keywords:
public, private, protected, static, final, abstract, synchronised, 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

* Return-type keywords:
void

* Literal keyword: (value or constant)
const.

(ii) Identifier:
It is the name given in java, for identification. It can either class name, variable name, method name, label name, interface name etc.,

Rules for Identifiers:
* An identifier can be created, using alphabets. (uppercase (A-Z) or lowercase (a-z)).

(*) Only special character such as $ _ are allowed while creating an identifier.
Ex: int $sal, int emp_sal;

(*) Identifier should not start with digits (0-9).
(*) Identifier are case sensitive
Ex: int age = 30 ; int Age = 32
age = 32
(*) Keywords can't be used as Identifiers.
(*)

(iii) Literals:
Literal is a value associated with a variable or it is a value given to an identifier.

Ex: int a = 100;
↑ Identifier (variable name)
↑ value or literal

↳ Types of literals:

(i) Integer literal: Ex: int a = 69;
(ii) Floating literal: ex: float PI = 3.141f;
(iii) Character literal: Ex: char a = 'K';
(iv) Boolean literal: Ex: boolean True/false;
(v) Short literal: Ex: short a = 220;
vi String literal: Ex: String name = "bushi";
(vii) Double literal: Ex: double a = 100.0;
(viii) Byte literal: Ex: byte b = -128;

* There are 4 types of number systems:
(i) Binary (ii) Optal (iii) Decimal (iv) Hexa decimal

Note:

By default, java makes use of decimal number system.

Binary:

number: 0 & 1 (base 2)

Prefix: 0b

Ex:
```
class Demo { psvm(--)
{
int num = 0b 101;
sop(num);
} }
```

cal: 101

$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$

Optal:

number: 0-7 (base 8)

Prefix: 0

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

cal: 16

$1 \times 8^1 + 6 \times 8^0$

$= 8 + 6 = 14$ /:

Decimal:

number: (0-9)

Prefix: x

Ex:
```
class Demo { psvm(--)
{ int num = 243;
sop(num);
}
}
```

Hexadecimal:

base 16

Number: (0-9), a-f

prefix: 0x

Ex:

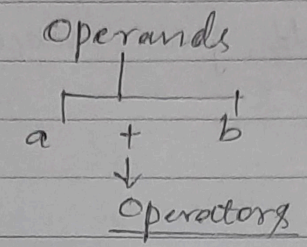
cal: 28b

$2 \times 16^1 + 8 \times 16^0 + 11 = 32 + 8 + 11$

$= 51$

* Operators:

↳ They are the special symbol used to perform, some Operation on the Operands.



↳ Types of Operators:

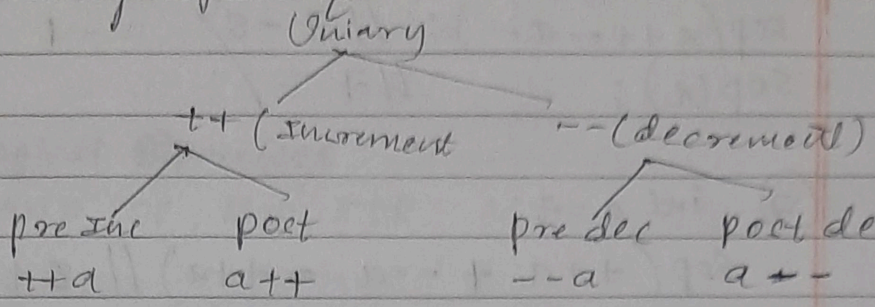
(i) Arthematic Operators
(ii) Unary "
(iii) Assignment "
(iv) Relational "
(v) Logical Bitwise "
(vi) Shift "
(vii) Binary or Conditional "
(viii) Short Circuit or logical ".

(1) Arthematic Operators:

+, -, *, %, /

Ex:
```
class Arop { psvm(--){
    int a=10; int b=4;
        Sopln(a+b) // 14
        " (a-b) // 6
        " (a*b) // 40
        " (a/b) // 2
        " (a%b) // 2
        " (205%100) // 25
    }
}
```

(2) Uniary Operators:



↳ pre (Inc/dec) : (i) Inc/dec (ii) Display.

↳ post (Inc/dec) : (i) Display (ii) Inc/dec.

Examples:

①
```
class Demo {
 psvm(---){
  int a = 7
  sop(a++);//7
  sop(++a);//9
  sop(a);//9
 }
}
```

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

③
```
int a=7;
sop(++a);//8
sop(a++);//8
sop(--a);//8
sop(a)//8
```

④ int a = 7,8,9
```
sop(a++);//7
sop(++a);//9
sop(--a);//8
sop(a--);//8
sop(a);//7
```

⑤ int a = ~~7~~ 8 a
```
sop(++a);//8
"(++a);//9
"(a++);//9
"(--a);//9
(++a);//10
(--a);//9
(a--);//9
(a)//8
```

⑥ int a = 7; // 7, 8, 9
sop(a++ - a--); // 7 - 8 = -1
sop(a); // 7

⑦ int a = 7;
sop(++a + --a - ++a) // 8 + 7 - 8 = 7
sop(a) // 7

⑧ int a = 7; 6
sop(--a + --a + a++) // 6 + 5 + 5
sop(a) // 6

⑨ int a = 7;
sop(++a + --a - ++a + --a - ++a
+ ++a - ++a - a++)
o/p: 8 + 7 - 8 + 7 - 8 + 9 - 10 - 10 = 14 - 8 + 9 - 20
= 14 + 1 - 20
= -5
sop(a) // 11

⑩ int a = 4;
sop(a++ + ++a + --a + a-- + a++ +
a++ + --a - a--);
sop(a) // 4
4 + 6 + 5 + 5 + 4 + 5 - 5 - 5
= 29 - 10 = 19

⑪
int a = 7; 4 5
sop((--a + --a) + (a-- + --a) +
(++a + a++) - (--a + --a));
sop(a); // 3
(6 + 5) + (5 + 3) + (4 + 4) - (4 + 3)
= 11 + 8 + 8 - 7 = 27 - 7 = 20

* Relational Operator:
  >, <, <=, >=, ==

* Logical Operators:
  AND, OR, NOR, XOR

* Shift Operator: >>, <<

* Conditional Operator or Ternary Operator:

  Condition ? Exp1 : Exp2
  (T) (F)

Ex: (num % 2 == 0) ? Even : Odd

Question 1: WAP to find the number is Even or odd using conditional operator.

Question 2: Prgm to find largest among 3 number using conditional Operator.

* Assignment Operators:
  =, +=, -=, *=, /=, %=, &=, |=, ^=, >>=, <<=

Ex:

```
Public class Aso{
    public static void main(String[] args){
        int x=5; x+=5
        System.out.println(x+=5, "x"); // 10
        x-=5
        Sopln(x) // 5
        x*=5
        Sopln(x) = 25
        x^=3
        Sopln(x) // 6
    }
```

101
011
110

* Logical Operators:
&&, ||, !

Ex: class LOO{
    psvm(String[] args){
        int x=5;
        System.out.println(x>3 && x<10); //true
        System.out.println(x<5 || x<4); // false
        " " " !(x<5 && x<10); true
}}

* Shift Operator: >>, <<, >>>

Ex: Right shift, Left shift
public class SO{
    public static void main(String[] args){
        int x=5;   101
        x>>=3;   000
        System.out.println(x); //0
        int y=5;
        y<<=2;   101
        Sopln(y);// 4   100
}}

Ex: Unsigned Right shift:
class URS{
    psvm(String[] args){
        byte num1=8;   1000
        byte num2=-8;   0010
        Sop(num1>>>2); //2
        Sop(num2>>>2); //1073741822
}}

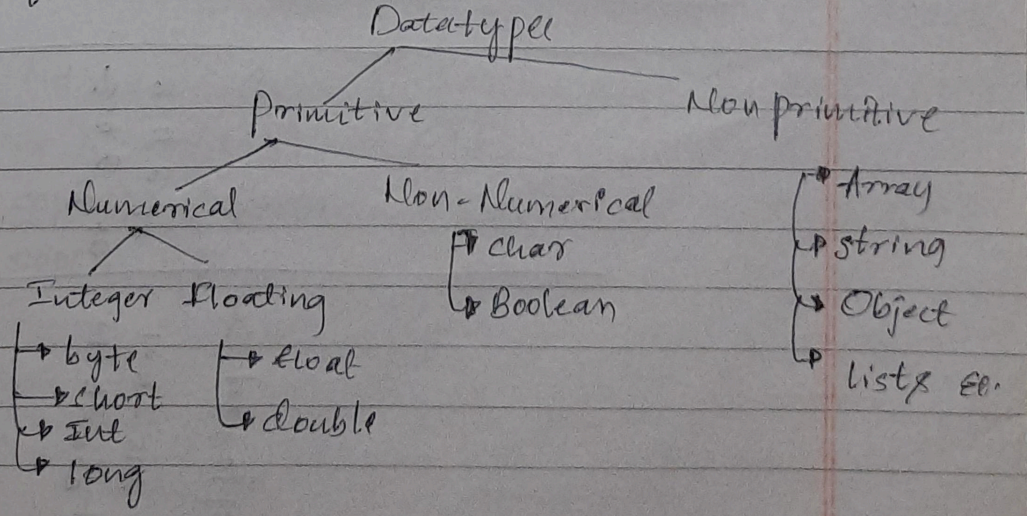
Q1:
```
int temp = (num%2 == 0)? Even : odd
import java.util.Scanner;
public class EvenOdd{
    public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        int number = s.nextInt();
        (number % 2 == 0)? sop(even) : sop(odd);
    }
}
```

Q2:
```
public class Max{
    public static void main(String[] args){
        int a = 10, b = 20, c = 30, max = 0;
        max = (a>b)? (a>c? a : c) : (b>c? b : c);
        System.out.println(max);
    }
}
```

DAY-8 20/2/2024

* Data types in Java:

↳ Data types are used to specify the values to the variables. It helps to convert data into binary format.



Definitation	byte	short	int	long	char	boolean	float	double
Declaration:	byte a = 28;	short	int	long =	char c = 'k';	boolean true;	float f = 1.4f	double d =
Size :	1byte = 8bits;	2byte	4byte	8byte	2byte	Dependent on OS		
Range :	-128 to 127	-2^{n-1} to $2^{n-1}-1$	n = 32bit	64bit	n = 16bit		$-3.4e^{38}$ to $3.4e^{38}$	$-1.7e^{308}$ to $1.7e^{308}$
Example:	byte age = 128 byte age = 127	-32,768 to 32,767 Ex: short s = 10000	Ex: salary of freshers	Ex: salary of experienced employee	unicode	No range values: true false Ex: logical operators.	Ex:	Ex: Speed of light,

Interview Question: Java

(1) Why the float value is suffix with f ?

↳ If the float value is not suffix with f, then it is treated as double in java.

float a = 3.481f ← (4) (4)

float a = 3.481 ; (4) ↳ double (8) overflow

(2) Why char size in java, is of two bytes where as C & C++ have 1 byte ?

A: Bcz, java uses unicode-format (utf) and C uses ASCII format. Java contains 18 lang. where C contains only 256 characters. 256 is represented in 1 byte, but java contains 65,535 & can't be represented in 1 byte so java char size is 2 byte.

Examples on Data types:

(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 = ++6;
sop(a); //
sop(b); //
```
Error

(4)
```
int a = 5;
int b = --(++a);
sop(a); // Error
sop(b); //
```

(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

Note:

op1 + op2 → max(int, typeofop1, typeofop2)

byte + byte → max(int, byte, byte) - int
byte + short → max(int, byte, short) - int
short + int → max(int, short, int) - int
byte + long → max(int, byte, long) - long
short + long → max(int, short, long) - long
long + long → max(int, long, long) - long
long + int → max(int, long, int) - long
int + int → max(int, int, int) - int
float + long → max(int, float, long) - float (Type casting)
float + double - double

* Typecasting in Java:

Converting one type of data to another data type is called typecasting

→ As been classified into 2 categories:

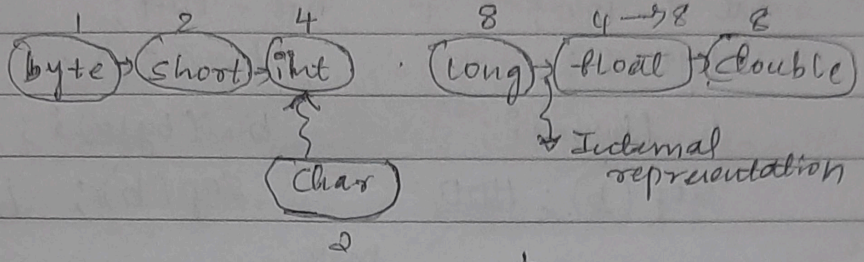
(i) Implicit.
(ii) Explicit.

↳ Implicit Casting:

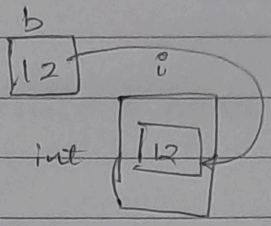
↳ Converting lower datatype into higher datatype

↳ Type checking is done by compiler.

↳ Type casting is done by javac



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



↳

In the above we trying to convert double into byte

&

There, problem can overcome using Explicit typecasting

↳ Explicit Typecasting:

↳ converting higher datatype to lower datatype is called as Explicit typecasting. It should be performed by programmer.

↳ Loss of data not occur in implicit typecasting as it done by JVM. Whereas, Loss of data occur in Explict type casting becz it is performed by programmer.

March - 30

Explicit TC:

byte → short → int ← char
int ← long ← float ← double
(float → internal representation)

Ex:

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

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

MSB is 1

0 0 0 0 0 0 0 1
0 1 1 1 1 1 1 0 1's
1 2's
-ve 1 1 1 1 1 1 1 (-127)

* Flow Control Statements:

Java program executes from top to bottom & if we want to control, the follow of execution, then we use flow control statements.

Control Statements

Conditional / ~~Selection~~ statements	Iterative / looping	Jumping Statements
* if	* for	* break
* else-if	* while	* continue
* nested if	* do-while	* return
* else if ladder	* for-Each	
* switch		

DAY-9

* If and Else-If Statements:

Syntax:

```
if (condition) // checks for condition
{
   ...        } Block of code executed if condition become true
}
```

↳ If the if condition fails, it Enter into Else-if

```
else-if (condition) // checks for the cond
{
   ...        } Execute a block of code.
}
else {
   ...        } Execute a block of code.
}
```

Ex:

```
if (Auditoriom 1 != full) {
        sop ("Students can Enter into class");
}
else-if (Auditoriom 2 != full) {
        sop ("students can Enter into class");
}
else { sop ("Can join to next Batch");
}
```

* Nested-If:

Syntax:

```
if (condition) // true
{
    if (condition) {
        ... } block of code
}
```

Scanner scanner = new Scanner(System.in);

Ex:

```
public class hisence {
    public static void main(String[] args) {
        byte under_age, Over_age;

        if(under_age < 18 && Over_age > 60)
        {
            System.out.println("Not Eligible");
        }
        Else {
            System.out.println("Eligible");
        }
    }
}
```

* while & do-while;

```
int n
while (n <= 10) {
        System.out.print(n);
        n += 1;
}
```

```
int mul = 1;
while (mul <= 10) {
        System.out.print(mul * 3);
        mul++;
}
```

```
int mul = 10;
for(int i = 1; i <= mul; i++) {
        if(i * 2 == 12)
            continue;
        else
        sop(i * 2);
}
```