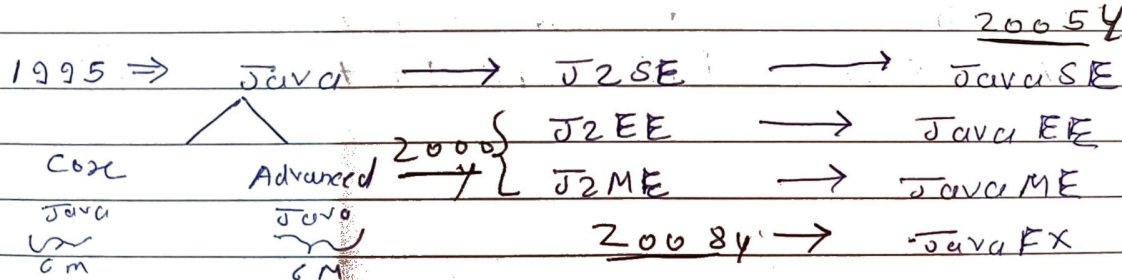
26/10/24

Lec-3

* Full Stack Java :

1) Core Java
2] Advanced Java
3] Spring, Spring Boot & micro services
4] DevSecOps (15+ Tools)
5] HTML, CSS & Java script
6] Bootstrap & React JS
7] Oracle (SQL & PL/SQL)
8] AWS ~~ser~~ Basics

1972 => C
↓
1990 => internet
↓
1995 => Java
(more secure)



JavaSE : Java platform : standard Edition

JavaEE : Java platform : Enterprise Edition

JavaME : Java platform : Micro Edition

JavaFX : Java platform : Fx (Effects)

→ Core Java is a part of JavaSE.
→ Advanced Java is a part of JavaSE & JavaEE.
→ J2EE is a JavaEE

* Core Java :-

1] Java Fundamentals
2] object oriented programming (OOP)
3] Inner class & packages
4] String Handling & wrapper classes.
5] Exception Handling.
6] Java Streams & serilization
7] Network programing
8] collection framework & Generics
9] Multithreading & sychronization
10] New Features [from JDK 1.1 to JDK 21]

27/10/24

Lec-4

* Java Fundamentals :->

→ Java is a programing language, Platform & technology.

↳ Java is called as programing language b.coz by using Java, we can write programs.

Platform :

It can be a software or hardware environment in which program runs.

C code

demo.c (uni code)

↓

Compiler

↓

demo.exe (bit code)

↓

Windows

↓

output

Java code

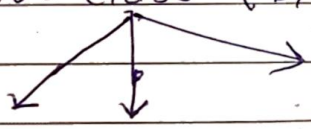
Demo.Java (uni code)

↓

Compiler

↓

Demo.class (byte code)



Windows	Linux	Solaries
JVM	JVM	JVM
↓	↓	↓
Windows	Linux	Solaries
↓	↓	↓
output	output	output

→ "C" compiler converts uni code to bit code whereas Java compiler converts uni code to byte code.

JVM :-

→ JVM stand for Java virtual Machine.

→ It contain an interpeter & it convert byte code into bit code.

→ Both compiler & interpeter are called translati Softwares.

* Defferences beth compiler & interpeter

Compiler	Interpeter
1) It convert the whole program at a time.	1) It convert line by line
2) It produce a file.	2) It does not produce the file.
3) It is fast.	3) It is slow.

→ "C" program is a platform depedant where as Java program is a platform Indepedant.

→ Java is called as platform Independant. because program written in Java language can be executed on any platform.

→ JVM is called as platform depedant. Bcoz window JVM for window only, linux JVM for linux etc.

→

10/30/24

→ JVM is a part of JRE (Java Runtime Envisoment).
→ Java Runtime Envisoment is called as Java platform because Java program runs under JRE.

→ JRE is a part of JDK.
→ JDK stand for Java Devolpment kit & it is called as Java software.

Lec ④⑤

→ In 1995 · Java was developed by James Gosling, patrick Naughton, Ed Frank, chris Warth & Mike sheridan at sun microsystems.
(Now owned by Oracle corporation).

* Identifiers :-> An identifier is a word & it is used to identfy ~~word~~ method, variable, class, interface etc.
→ It can be a varible name; method name, interface name etc.

* Rules to declare an identifier :->

→ 1) It can be formed by using alphabets (A to Z & a to z), digits (0 to 9), underscore symbol (_) and dollar symbol ($).
→ 2] It must begins with alphabet, underscore symbol or dollar symbol.
→ 3] The length of the identifier is not limited.
→ 4] It should not contain special symbol other than underscore & dollar symbols.
→ 5] It should not contain white spuce characters.

Examples

1) demo ✓
2) Demo ✓
3) DEMO ✓
4) 5Demo ✗
5) Demo5 ✓
6) version id ✗
7) version_id ✓
8) version-id ✗
9) version#id ✗
10) version$id ✓
11) _args ✓
12) _5_ ✓
13) $_$ ✓
14) _$args ✓
15) args_$ ✓

* Keywords :> A set of words reserved by language itself & those words are called keywords.

Example :> public, private, protected, int, char, float, double, if, else, while, for, static, void, final, class, interface etc....

→ All keyword must be written in lower case letters only.
→ There are 50+ keyword in Java.
→

Note 1 : const & goto keywords presently not in use.
Note 2 : Keyword can not be used as an identifier.

* Literals :>

→ A literal is a source code representation of a fixed value.

→ In Java, literals are divided into following categories

1) Integer Literals :>

15, 23, 134, 98, -9, -34, 0 --- etc

2] Floating point ~~Int~~ Literals :->
5.23 , 0.008 , -2.343 , 5.1345 , ... etc

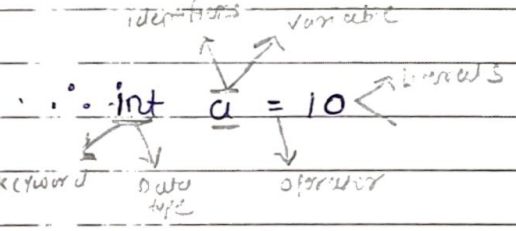
3] Charcter Literals :->
'a' , 'x' , 'A' , 'Z' , '0' , '] ' , '%' etc.

4] String Literals :->
"a" , "be" , "welcome" , "Hello" , "123" , "5.3" etc

5] Boolean Literals :-> true , false

6] object Literals :-> null

* ↳ Note 1 → true , false & null are not keywords.
↳ Note 2 → true , false & null are also cannot be used as an identifier.



Lec ⑥

* Data Types :-> A data type that determines what value variable can hold & what are the operations can be performed on variables.

↳ In Java , data types are divided into two categories
1) primtive Data Types
2] Reference Data Types.

Lec 6

1) Primitive Data Types :->

All primitive data types are predefined data types & those are named by keyword.

↳ There are 8 primitive data types & those are divided into the following sub categories :

1) Integers
2) Floating point Numbers
3) Characters
4) Boolean

↳ 1) Integers :->

	Data Type	memory size	Range
↳ 1)	byte	→ 8 bits (1 Byte)	→ -2^7 to 2^7-1
↳ 2)	short	→ 16 bits (2 Bytes)	→ -2^{15} to $2^{15}-1$
↳ 3)	int	→ 32 bits (4 Bytes)	→ -2^{31} to $2^{31}-1$
↳ 4)	long	→ 64 bits (8 Bytes)	→ -2^{63} to -2^{63}

↳ 2] Floating point Numbers :->

	Data Type	Memory size	Range
↳ 5)	float	32 bits (4 Bytes)	-3.4×10^{-38} to 3.4×10^{3}
↳ 6)	double	64 bits (8 Bytes)	-1.7×10^{-308} to 1.7×10^{5}

↳ 3] Characters :->

	Data Type	Memory size	Range
↳ 7)	char	16 bits (2 Bytes)	0 to 65535

* ASCII (American standards code for Information Interchange) :->

A → 65
B → 66
Z → 90

a → 97
b → 98
z → 122

0 → 48
1 → 49
9 → 57

0 to 47
58 to 64
91 to 96
123 to 255
} Special Symbols

4] Boolean :->

Data Type	Memory Size	Range
8) boolean	1 bit	true / false 1/0

2] Reference Data Types :-> Reference Data types.
interface, class, Array, string etc

Variables :->

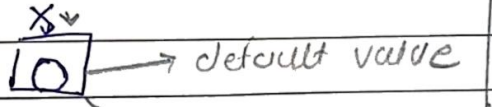
A variable is a container that contains data.

Syantax :

DataType variable ;

Example :

int x ;



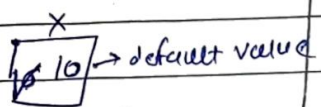
→ Assignment :

syantax :

variable = Literals ;

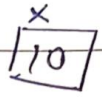
Ex

x = 10 ;



Initialization :

~~int x=10~~ int x = 10 ;



Lec (7)

int

* byte a = 5 ✓

byte b = 130 ×

short c = 130 ✓

short d = 40000 ; ×

int e = 40000 ; ✓

int f = ; ×

↓

out of range

then,

long g = ; ×

↓ out of range and within long range

long h = 1 (or) L

Note : long literal must be suffixed with l (or) L.

float

* float a = 5.24 ×

float b = 5.24 f (or) F ; ✓

double c = 5.24 ; ✓

Note : float literal must be suffixed with f (or) F

* char

```
char a = 'A';
char b = 65;

boolean a = true;
boolean b = false;
```

8 Bit type
7 another dT (concept sdf)
56
14 → conversion not possible
42
-23 → not possible but done by programmer
19 → automatically done

* Type conversion

↳ convert one type to another type is called as type conversion.

↳ boolean type can not be converted to other types & other types cannot be converted to boolean types. (How 0 & 1 operate as true & false?)

↳ The following 19 conversion are done by system implicitly :->

byte to short, int, long, float, double	=> 5	→ byte
Short to int, long, float, double	=> 4	→ short
int to long, float, double	=> 3	→ int
long to float, double	=> 2	→ long
float to double	=> 1,	→ float
char to int, long, float, double	=> 4	→ double
	Total → 19	→ char

These convertion are called widening conversion.

↳ The following 23 conversion must be done by programmer explicitly otherwise compile time error occours :=>

byte to char ⇒ 1
Short to byte, char ⇒ 2
int to byte, short, char ⇒ 3
long to byte, short, int, char ⇒ 4
float to byte, short, int, long, float, char ⇒ 5
~~char to byte, short~~ ⇒ 6
double to byte, short, int, long, float, char ⇒ 6
char to byte, short ⇒ 2
Total ⇒ 23

These conversion are called narrowing conversion.

Example :

1) byte a = 5 ;
int b = a ; → byte to int ✓
System.out.println(b)
output = 5

2] int a = 5 ;
byte b = a ; → int to byte X Error
sout(b);

3] int a = 5 ; ✓
byte b = (byte) a ; → Explicite Type conversion (or) Type casting
sout(b);

4] char a = 'A';
int b = a;
sout (b);

→ char to int ✓

output :→ 65

5] int a = 65;
char b = a;
sout (b);

→ int to char ✗ Error

Then How we solve this ? Here

int a = 65;
char b = (char) a; ✓
Sout (b);

7]

int a = 130;
byte b = (byte) a;
Sout (b);

→ It is valid output is out of Range so, it rotate -128 to 127

output = -126

128 → -128
129 → -127
130 → -126
131 → -125

8] boolean a = true;
int b = a;
Sout (b); ✗ Error

9] boolean a = true;
int b = int (a); ✗ Error
Sout (b);

31/10/24

Lec (8)

10] int a = 10;
float b = a;
sout (b);

INT to float

output- 10.0

11] float a = 93.456F;
int b = a;
sout (b);

X Error
F to I

12] float a = 94.456f;
int b = int (a)
sout (b);
∴ output = 94;

13] short a = 129;
byte b = (byte) a;
sout (b);
output : -127

-128 to
128 = -12
129 = -12
130 = -12

14] float a = 98.99 f
Char b = (char) a;
sout (b);

output = b { 98}

* operations on data :->

1) (byte, short, int, char) ± (byte, short, int, char) ⇒ int

2] (byte, short, int, long, char) ± long ⇒ long

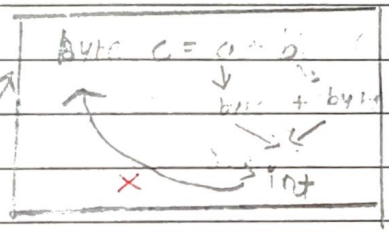
3] (byte, short, int, long, float, char) ± float ⇒ float

4] (byte, short, int, long, float, double, char) ± double ⇒ double

5] (byte, short, int, long, float, double, char, boolean) + boolean = Error

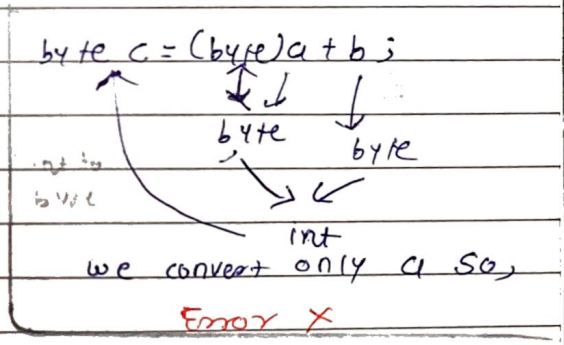
Example :->

1} byte a = 5;
byte b = 10;
byte c = a+b;
sout (c);

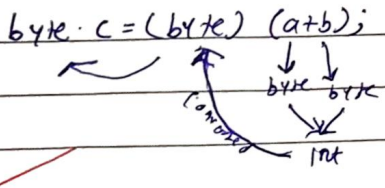


✗ Error

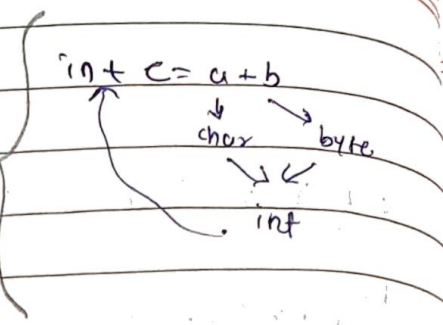
2] byte a = 5;
byte b = 10;
byte c = (byte) a+b;
sout (c);



3) byte a = 5;
byte b = 10;
byte c = (byte) (a+b);
sout c;

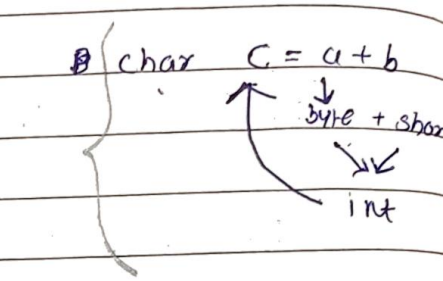


4] char a = 'A';
byte b = 5;
int c = a + b;
Sout (c);

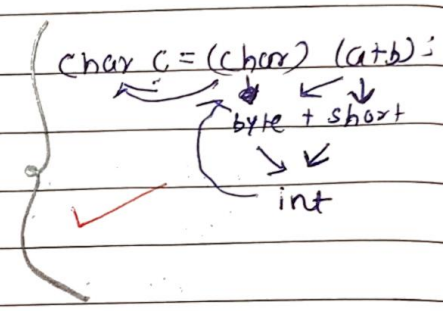


output : 70

5] byte a = 97;
short b = 3;
char c = a + b;
Sout (c);

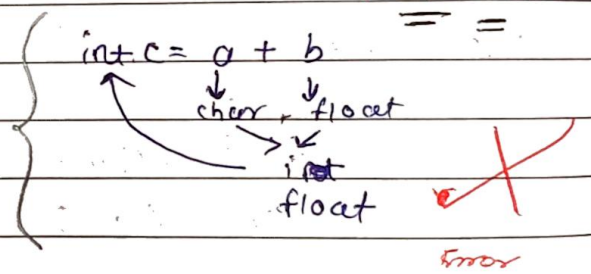


6] byte a = 97;
short b = 3;
char c = (char)(a+b);
Sout (c);
output : a

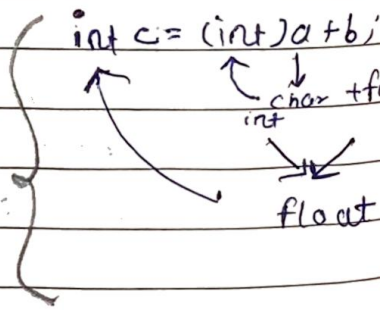


→ Lec (7)

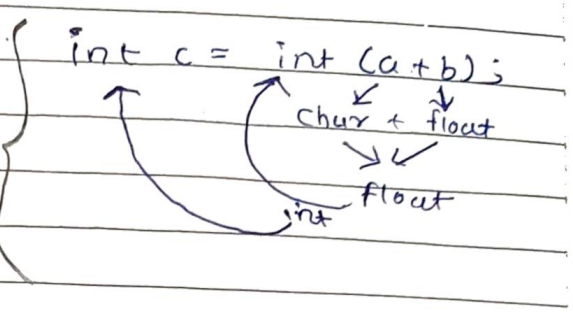
(7) char a = 'A';
float b = 2.93f;
int c = a + b
Sout (c);



8] char a = 'A';
float b = 2.93f;
int c = int (a) + b;
Sout (c);



3]
```
char a = 'A';
float b = 2.23f;
int c = int (a+b);
sout(c);
```
output : 67

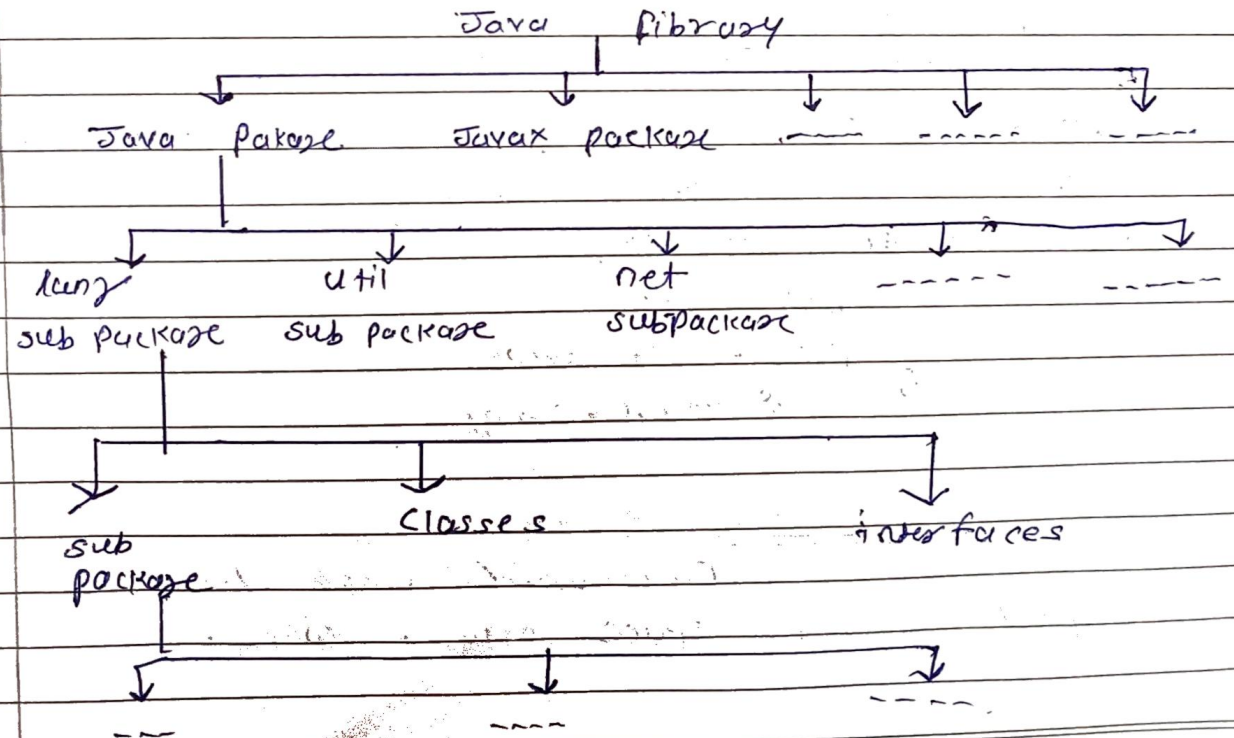


* Java Library :->

→ Java library is called as Java API. (Application programming Interface) because it is an interface b/w application & programming language.

→ Java library organized in the form of packages.

→ A package is collection of sub packages, classes & interfaces.



~~import java~~

↳ import java.lang.*;

This statement import all classes & interfaces of java.lang package into our java program.

↳ import java.lang.System;

This system import only system class of java.lang package into our java program.

* Note :-> java.lang is a default package & it is implicitly imported in every java program.

* Java Notations :->

1) package & sub package :->

↳ All small letters. sub package are separated with dot (.) symbol.

Ex. 1) java.lang
2) java.net
3) java.awt.event
4) javax.swing

2] class & interface :->

Each word first letter is capital & no space between word.

Ex. 1) System
2) StringBuffer
3] NullPointerException
} classes

4] Runnable
5] AutoCloseable
} interfaces

3] Variable and Method

Second word onwards first letter is capital & no space between words.

Ex.

1) name
2) empNo
} variables

3) main()
4) compareTo()
5) getEnclosingWriter()
} methods

* Lec 10 *

* How we write a program ? :->

1) * Declaration of rules to a source file (.java file).

IMP Rules

1) A source file can have only one public class.
2) A source file can have any number of non public class.
3) If the source file contain public class the file name must match with public class name.
4) If the source file does not contain public class then no naming restriction to a file name

Ex

B.java
```
public class A
{
}
```

A.java
```
public class A
{
}
public class B
{
}
```

B.java
```
class A
{
}
```

A.java
```
public class A
{
}
class B
{
}
```

✗ ✗ ✓ ✓

Demo.java

```
class Demo
{
    public static void main (String args[])
    {
        System.out.println("Welcome to Java");
    }
}
```

↳ In above example, String & System are predefined classes in java.lang package.

↳ java.lang package is a default package and it is implicitly imported in every java program.

Demo.java
```
class Demo
{
}
```

In command prompt

To compile ↓

C:\> javac Demo.java

Demo.class

To run ↓

C:\> java Demo

output

↳ Both javac & java are called jdk tools.
↳ All jdk tools are the part of bin folder in JDK.
↳ JDK stand for Java Development Kit & it is called as Java software.
↳ latest version of JDK is 21.

Demo.java

```
class Demo
{
}
```

C:\> javac Demo.java
↓
Demo.class

C:\> java Demo
↓
output

B.java

```
class A
{
}
```

C:\> javac B.java
↓
A.class

**C:\> java A
↓
output

D.java

```
ClassA
{
}
class B
{
}
class C
{
Public static void main
(String args[])
{
}
}
```

C:\> javac D.java
↓
A.class
B.class
C.class
} same

**C:\> java C
↓
whose contain main method

↳ The Java compiler generates .class file for every class in source file.

↳ A class that contains main() method can only be used to execute the program.

lec - 11

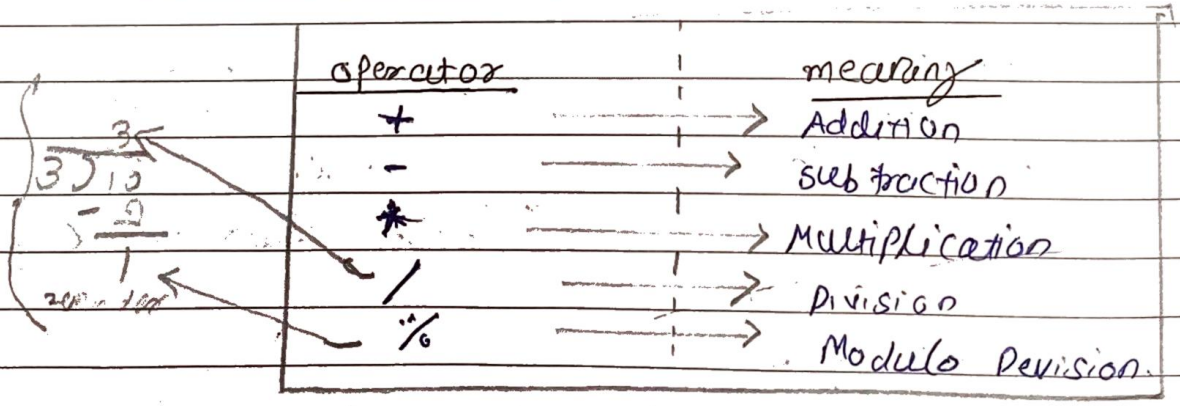
* operators :->

↳ An operator is a special symbol that operates on data.

↳ In Java, operators are divided into following categories :->

1] Arithmetic operators (5 Binary)
2] Relational operators. (6 Binary)
3] Logical operators (1 unary & 2 Binary)
4] Increment and Decrement operators (2 unary)
5] Bitwise operators (1 unary & 5 Binary)
6] Assignment operators (11 Binary)
7] conditional operators (1 Ternary)
8] other operators
8] unary plus & unary Minus operators [2 un

1] Arithmetic operators :->



* operator precedence :-

↳ In Java, BodMas (बोदमास) does not work so,

1]	()	Left to Right (Inner most to outer most)
2]	*, /, %	Left to Right
3]	+, -	Left to Right

+ Priority -

Example :

```
class Demo {
    public static void main (String args[]) {
        int a=7, b=5, c=3;
        int d= a* b/c;
        int e= a/b* c;
        sout (d);
        sout (e);
    }
}
```

output :- 11
3

a* b /c
7* 5 /c
35/3 = 11
same 2nd

Ex

```
int d= a* b + c;   // 38
int e= a+b * c;    // 22
```

Lec - 12 *

Ex. `int a=7, b=3, c=5`

```
int d= a+b* c %a;
sout (d);
```

output = 8

→ Lec-12

Ex int a=8, b=7, c=5, d=3, e=2

int f = ((a+b%c) - (d*e%c)) + (b+c%d-e)

10 1 2

Explanation

Solve:

((a+b%c) - (d*e%c)) + (b+c%d-e)

(2+8=10) - (6%5=1) + (7+2-2) = 7

9 + 7 = 16 ← output

* 2] Relational operators :->

Operator	Meaning
<	Less than
>	Greator than
<=	less than or equals to
>=	Greather than or equal to
==	Equals to
!=	Not equals to

↳ The above all operators return boolean values (true or false)

Ex.

```
int a = 5, b=3;
boolean c = a>b;
boolean d = a==b;
sout(c); // True
```

↳ If we write int instead of boolean like

```
int a = 5 , b = 3
int c = a>b  ⇒ Error
```

3] Logical operator :→

operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

↳ These operator also return boolean values. These operatores operate on boolean values only.

&&	T	f
T	T	f
f	f	f

||	T	f
T	T	T
f	T	F

	!
True →	false
false →	True

Ex. int a = 5 , b = 3, c = 2;

~~boolean a = true , boolean b = false ;~~

```
boolean d = (a>b) && (a>c);
sout (d);
```

(a>b) → T, (a>c) → T

Output True

```
boolean e = !((a != b) || (a == c));
sout (e);
```

∴ false

* Types of operators :⇒

There are 5 type of operator

1) unary operator
2) Binary operator
3) Ternary operator

1) Unary operator

An operator that operates on only one operand is called as unary operators. Here operand can be a varible, literals or expression.

ex. ~~Arithmetic operator which~~ ! is work on ~~two~~ one operand

2] Binary operator :⇒

A operator that operates on two oprands is called as binary operators.

ex, +, -, * work on two operand

3] Ternary operator :- A operator that operates on three operands is called as ternary operator.

4] Increment & Decrement operator

operator	Meaning
++	increment by 1
--	Decrement by 1

++a → pre increment
a++ → post increment
--a → pre decrement
a-- → post decrement

1/11/24

pre increment Excample

1) int a = 5
++a; 5 6
sout (a); ⇒ 6

2) int a = 5;
int b = ++a; 6
sout(a); → 6
sout(b); → 6

3) int a = 5;
sout (++a); → 6

post increment Excample

1) int a = 5
a++ 5 6
sout (a); ⇒ 6

* 2) int a = 5;
int b = a++;
sout (a); → 5 6
sout (b); → 5

3) int a = 5
sout (a++); → 5

→ Lec - 13

4) int a = 5, b = 3;
int c = a++ + b;
sout (a); // 6
sout (b); // 3
sout (c); // 8

6) int a = 5, b = 3
int c = a++ + ++b;
sout (a); // 6
sout (b); // 4
sout (c); // 9

5) int a = 5;
++a; // (6)
a++; // (7)
sout (a++); → 7 (8)
sout (++a); → 9

7)

* Pre-increment → same line ✗ Same position
post-increment → next line ✗ Next position

Test - 1

1) byte a = 10
short b = 3
int c = a+b;
sout (c);

2) int a = 3
float b = 4.24f
int c = a+b
sout (c);

4) boolean a = true;
int b = a+1;
sout (b);

3) ~~char a = 3;~~
~~float b = 4.24f~~
~~int c = a+b~~
~~sout (c);~~

3) char a = 'a'
int b = 3;
float c = a+b;
int d = (int) c;
char e = (int) d;
sout (e)

5) float a = 3.24f
float b = 7.26f;
cloeble c = a+b;
sout (c);

5) ~~char a = 'a';~~
~~int b = 3~~
~~float c = a+b;~~

float a = 3.24f
float b = 7.26f

6) char Char = 'A';
short Short = 1;
int Int = Char + Short;
sout (Int);

7) char a = "A"
byte b = 3;
int c = a+b
sout (c)

8) int _$_ = 5;
byte $_$ = 3;
int $ = _$_ + $_$;
sout ($);

9) byte a-b = 10;
shor c-d = 20;
int x = a-b+c-d;
sout (x);

10) boolean True = false;
sout (True);

output :-
1) 13 ✓
2) Error ✓
3) Error ✓
4) Error ✓
5) Error X. 10.50

6) 66 ✓
7) 68 X 1st line error
8) 5 ✓
9) 30 ✓
10) false ✓

8/10

Lec - 14

5] Bitwise operator :->

operator	meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Left shift
>>	Right shift
~	complement (Tilde)

↳ All bitwise operaters operate on Binary Data.

&	1	0
1	1	0
0	0	0

|	1	0
1	1	1
0	1	0

^	1	0
1	0	1
0	1	0

Ex,

```
int a=9,b=12;
int c= a&b;
int d= a|b;
int e = a^b;
sout (c); // 8
sout(d); // 13
sout(e); // 5
```

9 → 1001
12 → 1100

1001 & 1100 = 1000 → $2^3 2^2 2^1 2^0$ → 8

1001 | 1100 = 1101 → 8 + 4 + 1 => 13

1001 ^ 1100 = 0101 → 4 + 1

Ex: 2

```
int a = 9
int b = a << 2;
int c = a >> 3;
Sout (b); → 36
Sout (c); → 1
```

Convert number to Bit

9 = 1001

1 byte = 4 bit so two int = 32 bit

int a = 9

0000 0000 0000 0000 0000 0000 0000 1001

Shift 2 ←← b = 0000 0000 0000 0000 0000 0000 0010 0100

= 36

c = 0000 0000 0000 0000 0000 0000 0000 0001

= 1

Ex 3

```
int a = 9
int b = ~a;
Sout (a); // -10
```

~a

0000 0000 0000 0000 0000 0000 0000 1001

↳ 1111 1111 1111 1111 1111 1111 1111 0110

→ −10

```
if · int a = 34;
   int b = ~a;
   Sout (b) // -35
```

```
int a = -24;
int b = ~a;
Sout b; // 23
```

6) * Assignment operator:

operator	Meaning	
=	Normal Assignment	
a+ = b ;	a = a + b ;	Assignment operator
a- = b ;	a = a - b ;	
a* = b	a = a * b ;	
a / = b	a = a / b ;	
a % = b	a = a % b ;	
a & = b	a = a & b ;	Bitwise Assignment
a | = b ;	a = a | b ;	
a ^ = b ;	a = a ^ b ;	
a << = b ;	a = a << b ;	
a >> = b	a = a >> ;	

o

Ex :->

```
int a=5, b=3;
a* =b;
sout (a); // 15
sout(b); // 3
```

2/11/24

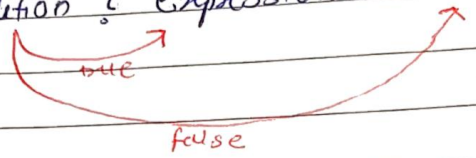
Lec - 15

7] Conditional operator :=>

It is a ternary operator & it operates on three operands.

- Syantax

condition ? expression 1 : expression 2

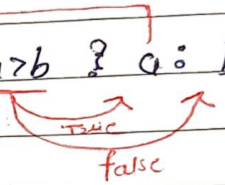


↳ If the condition return true then expression 1 evaluated. otherwise expression 2 evaluated.

Example :=>

1)
```
int a = 5, b = 3
int c = a>b ? a : 5
Sout (c); // 5
```

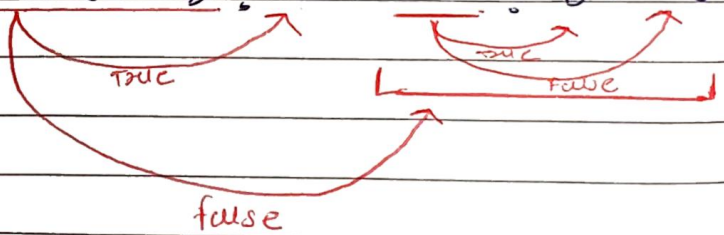
```
int c = a>b ? a : b
```



2)
```
int a = 5, b = 3, c = 8
int d = ((a>b) && (a>c)) ? a : b>c ? b : c;
sout (d);
```

Explanation

```
int d = ((a>b) && (a>c)) ? a : b>c ? b : c;
```



output = 8

8] Unary plus & unary Minus operators :-

operator	Meaning
+	positive
-	Negative

Example :->

```
int a = -5 ;
int b = -a;
sout (b); // 5
```

1)
```
int a=5, b=3;
a += b
Sout (a); // 8
Sout (b); // 3
```

1)
```
int a=5, b=3
a = +b;
sout (a); // 3
sout (b); // 3
```

2)
```
int a=5, b=3;
a -= b;
Sout (a); // 2
Sout (b); // 3
```

2)
```
int a=5, b=3
a = -b ;
sout(a) ; // -3
sout(b); // 3
```

3)
```
int a=5, b=3;
a *= b;
Sout (a); // 15
Sout (b); // 3
```

3)
```
int a=5, b=3;
a = *b;
sout(a);
sout(b);
```
Error

↳ Bitwise AND operator work as a logical AND operator also in Java.

Example :->

```
int a = 5, b = 3, c = 2;
boolean d = (a>b) && (a>c);
boolean e = (a>b) & (a>c);
sout(d);        // T
sout(e);        // T
```

↳ Logical / ~~Bitwise~~ AND operator checks the second condition if and only if the first condition is true where as Bitwise AND operator ~~also~~ always checks the second condition Also.

Logical AND

```
int a = 5 ; b = 8, c = 2;
boolean d = (a>b) && (a++>c);
sout(d);
sout(a);
```

output : false
5

Bitwise AND

```
int a = 5 ; b = 8, c = 2;
boolean d = (a>b) & (a++>c);
sout(d);
sout(a)
```

output : false
6

↳ Bitwise OR operator works as a logical OR operator also in java

Example :-> int a = 5, b = 3, c = 2

boolean d = (a>b) || (a != c);

boolean e = (a>b) | (a != c);

sout (d); // T

sout (e); // T

↳

↳ Logical OR operator checks the second condition if and only if the first condition is ~~true~~ false where as Bitwise OR operator always checks the second condition Also.

Logical OR	Bitwise OR
int a = 5, b = 3, c = 2;	int a = 5, b = 3, c = 2;
boolean d = (a>b) || (a++ != c);	boolean d = (a>b) | (a++ != c)
sout (d);	sout (d);
sout (a);	sout (a);
Output = true	ouput = true
5	6

↳ Bitwise XOR operator operates on boolean values also as follow:

^	T	F
T	False	true
F	true	false

^	1	0
1	0	1
0	1	0

Excample:

```
int a=5, b=3, c=2;
boolean d= (a!=b) ^ (b == c);
Sout (d);
```

output : True

Test

1)
```
int a = 1
boolean b = (int) a;
Sout (b);
```

2]
```
Char a='A';
byte b = a;
Sout (b);
```

3]
```
Short a=10;
double b=a;
Sout (c);
```

4]
```
float a= 50.999f
Short b = (Short) a;
Sout (b);
```

5]
```
float a=3.14;
float b= 8.86;
float c= 8.86 a+b;
flo Sout (c);
```

6]
```
Char a = 99;
byte b = 1;
Char C = (Char) (a+b);
Sout (C);
```

7)
```
byte a=9;
float b=3.29f
int c= a+(int) b
Sout (c);
```

8]
```
byte true = 1;
Short false = 2
int x = true + false;
Sout (x)
```

9]
```
char a = 'A';
Char b = 'a';
int c = a + b
Sout (c);
```

10]
```
byte a = 3;
Short b = 2;
int c = a + b;
float float d = c;
sout (d);
```

output / answer :->

1) Error ✓
2) 65 ✗ Error
3) Error ✓
4) 50 ; ✓
5) Error ✓
6) Error ✗ (d)
7) 12 ✓
8) Error ✓
9) 162 ✓
10) Error ✗ 5.0

Explantaion of pb NO. 6 :-

(6) ↳ In Java we can store @ integer in char, like 99 in char it ~~going~~ going to be c.

(10) ↳ ~~fto~~ int value we can store in float also.

~~3~~ (2)
↳ Char to byte it is not possible. it is done by programmer explicitly.

7/10

* Control Flow statement :->

1) Selection statements :->
   1) if statements
   2) if else statement
   3] if else if ..... else statement
   4) Nested statement
   5) switch statement

2] Iteration statement (Loops) :->
   1) while loop
   2) do while loop
   3) for loop
   4) Enhanced for loop
   5) Nested loop

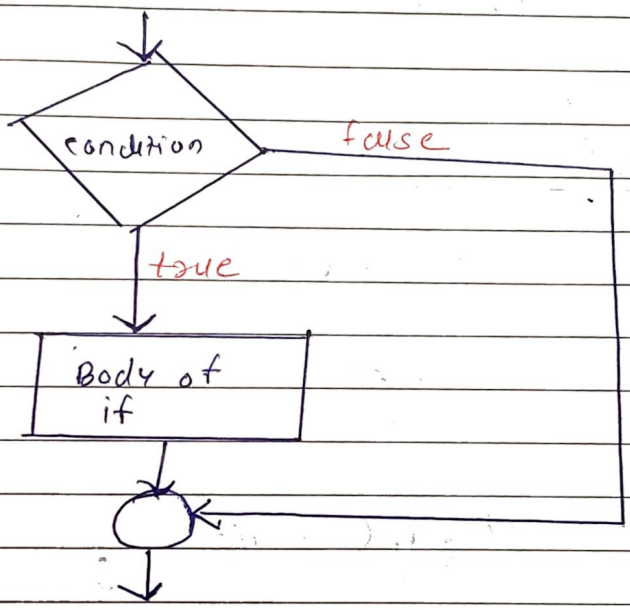
3] Jump statement :->
   1) break
   2) break LABEL statement
   3) continue statement
   4) continue LABEL statement
   5) return statement.

1] if statement :->

Syantaxes=> i) if (condition)
;

ii) if (condition)
{
;
;
}

Control flow :->



out of if statement

Example :-

int a=10;
if (a>0)
Socet ("positive");

output = positive

:- if condition true the execute.

Example 2 :-

```
{
int a = 5;
if (a++ > 5)
    sout (++a);
    sout (a++);   // 6
}
```

↳ If the statement is true all lines are executed.
↳ If statement false except 1st line all other are executed.

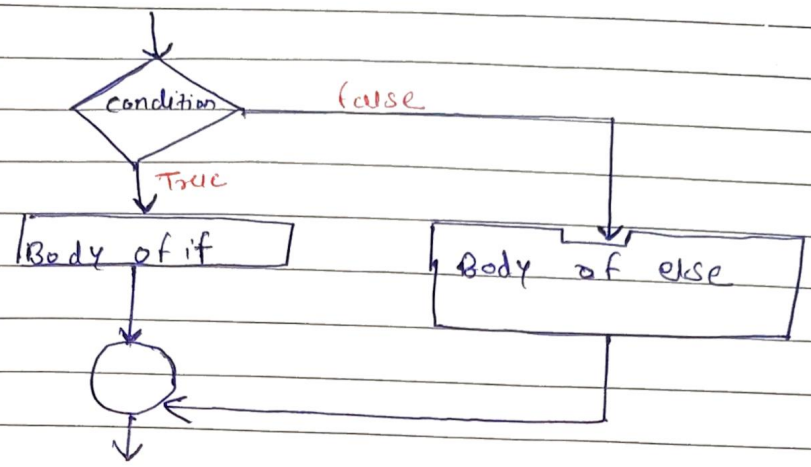
Java - 18 - lec

* if else statement :-

Syntaxes :- 1) if (condition)
;
else
;

2) { if (condition)
;
;
}
{ else
{
;
;
}

Control flow



Out of if else statement

ex :-

```
int a=0;
If (a>0)
   Sout ("Positive");
else
   Sout("Negative");
}
```

output :- Negative

Ex 2 :-

```
int a=0
if (a>0)
   Sout("Positive");
   Sout("End 1");
else
   Sout ("Negative number");
   Sout ("End 2");
```

← This is out of if

output : error

Ex 3 :

```
int a=0
if(a>0)
   Sout ("Positive");
else
   Sout ("negative")
   Sout ("End");
```

output = Negative
End

It is out of if else block

Ex. 4

```
{
int a=5;
if (a++ >= 5)
  sout(--a);
else
  sout(a--);
}
```

output :-> 5 (if blocks)

Comments :- comments are used to describe program, classes, method --- etc

↳ comments are ignored by the compiler.

There are two types of comments :-

1) Single line comments :-

syntax :-> //.....

2] Multiple lines comments :->

syantax :-> /*.....

.....*/

Ex program to check whether odd or even

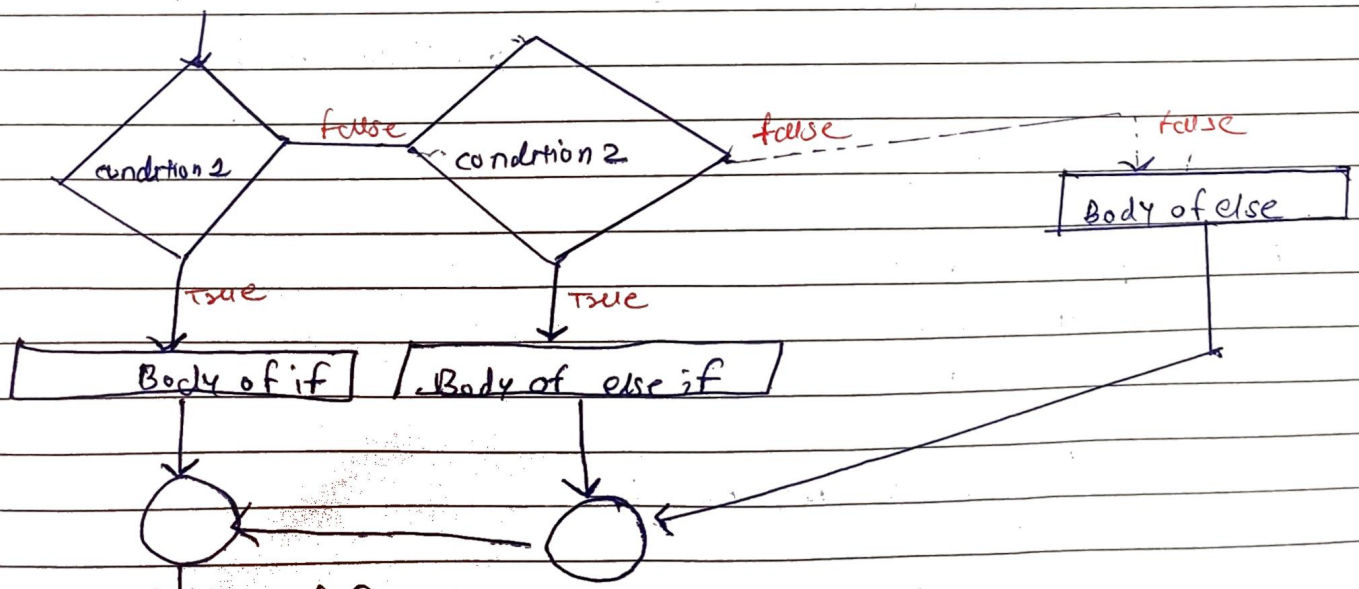
```
import java.util.*;
class Demo
{
  public psvm ( )
  Scanner S = new Scanner (System.in);
  sout (" Enter number ");
  int x = S.nextInt ();
```

```
if (x%2 == 0)
    Sout ("Even");
else
    Sout("odd");
}
```

3] if & else if ------ else statement :->

- Syantax

```
if (condition 1)
.....;
else if (condition 2) {
.....;
.....
.....
else
.....;
```



Ex:

```
import java.util.*;
psvm
{
    Scanner sc = new Scanner(System.in);
    Sout("Enter number");
    int a = sc.nextInt();
    if(a>0)
        Sout("positive");
    else if(a<0)
        Sout("Negative");
    else
        Sout("zero");
}}
```

Java - 11 - lec

Ex.

```
int a = -10;
if(a>0);
Sout("positive");
}
```

→ it is out of if

equivalent to

```
if(a>0)
;   → it is an empty statement
Sout("positive");
```

output:

positive

Ex.

```
int a = 10
if(a>0);
Sout("Positive");
else
Sout("Negative");
}
```

output :- error (else without if)

Nested if statement :->

↳ if statement that is defined in another if statement is called as nested if statement.

Ex

```
int a = s.nextInt();
if (a>0)
{
    if (a%2 == 0)
    {
        Sout ("Even");
    }
    else
    {
        Sout ("odd");
    }
}
else {
    if (a<0)
    {
        Sout ("Negative");
    }
    else {
        Sout ("zero");
    }
}
}
}
```

* Switch statement :->

It is equivalent to if else if .... else statement.

Syantax:

```
switch (variable)        → int/char/ string type
{
case literal 1 : ---------- ;
                 break ;
case literal 2 : ---------- ;
                 break ;
.....
.....
default : ---------- ;
}
```

Ex:->

```
Scanner s = new Scanner (System.in);
Sout ("Enter number");
int a = s.nextInt ();
switch (a)
{
case 1 : sout ("one");
         break ;
case 2 : sout ("Two");
         break;
case 3 : sout ("Three");
         break ;
default : sout ("Invalid");
}
```

↳ Here break statement terminates the switch statement.

2] Iteration statement (Loops) :->

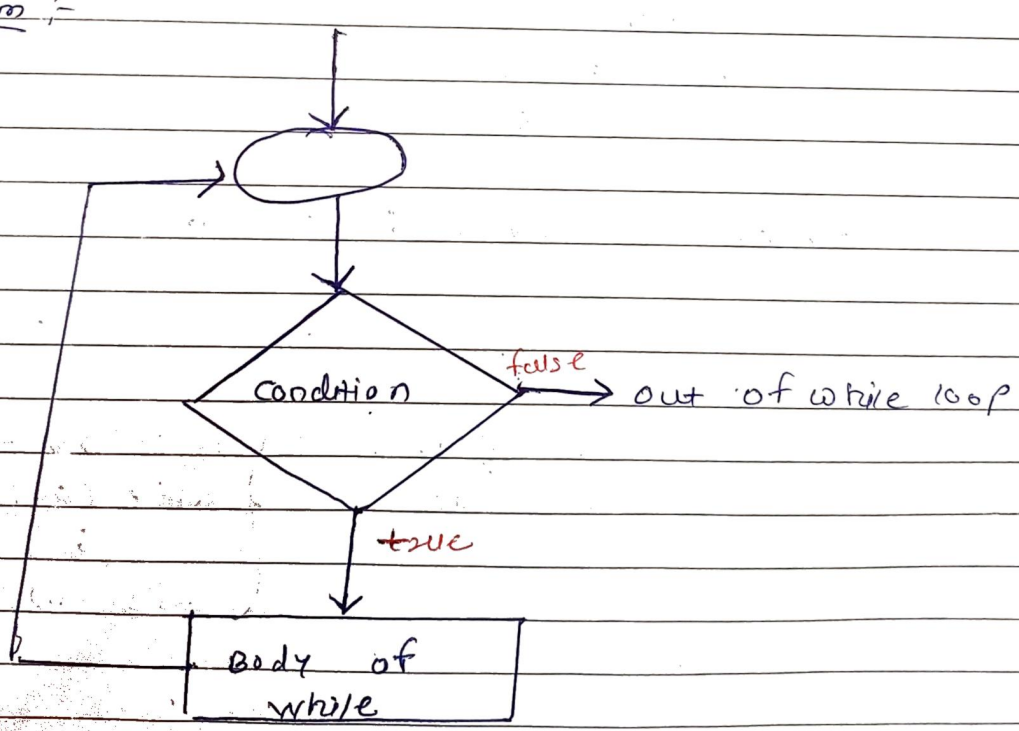
1) while loop :->

Syantax :->

(i) while (condition)
.....;

(ii) while (condition)
{
.....;
.....;
}

Diagram :-



↳ Here the body of while loop is executed repeatedly will the condition become false

Example :-

```
int a = 1;
while (a <= 10)
{
  sout (a);
  a++;
}
```

output :- 1 2 3 4 5 6 7 8 9 10

Example : 3)

```
int a=1;
while (++a <= 10)
    sout (a);
```

output :- 2, 3, 4, 5 6 7 8 9 10

Ex.

```
int a=1;
while (a++ <= 10);
    sout (a);
```

→ It is equivalent to

```
while (a++<=10)
;
sout (a);
```

Sec. 20

output :- 12

9/11/24

Lec - 20

Ex>

```
int a=6, b=1;
while (a>b)
{
  if (a%b==0)
    Sout(b);
    b++;
}
```

output :-> ~~1 2 3~~ 1 2 3

Ex

```
int a = 7398; b;
while (a>0)
{
  b = a%10;
  Sout(b);
  a = a/10;
}
```

output : ~~8 3 9 7~~ 8 3 9 7

Program to display factor of a given numbers

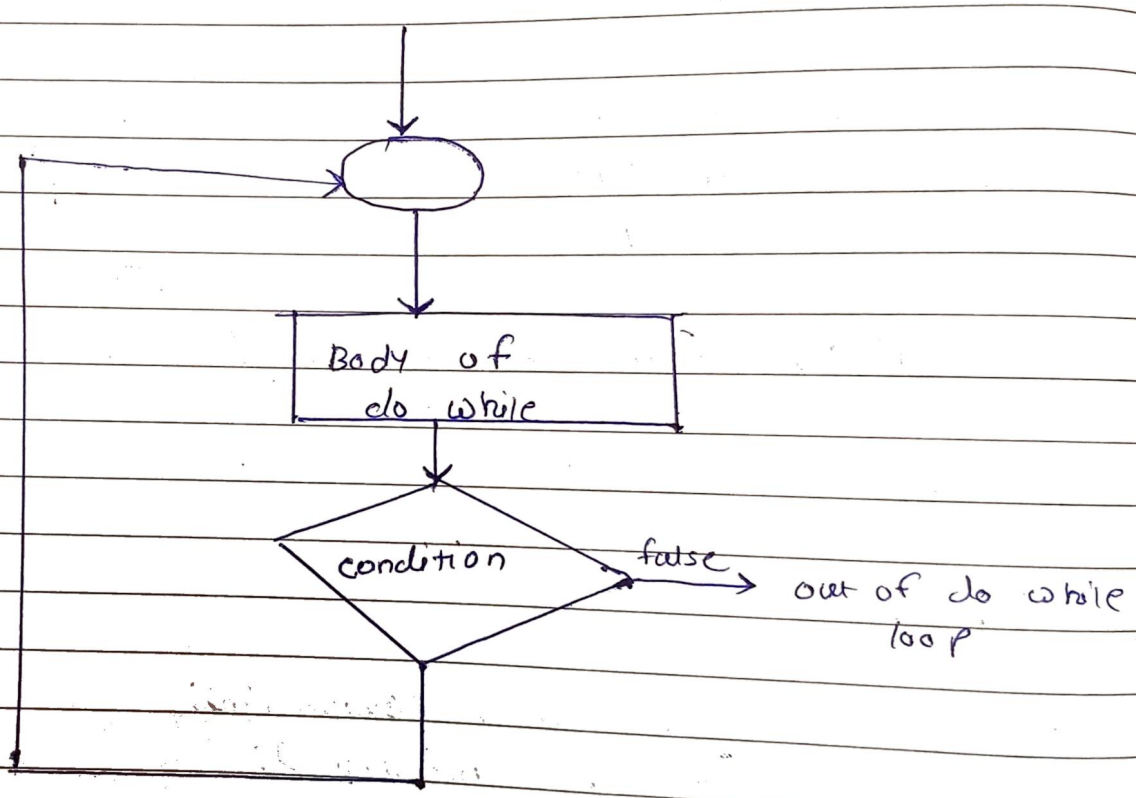
```
class demo
{
  PSVM
  {
    Scanner s = new Scanner(System.in);
    Sout("Enter any number");
    int a = s.nextInt();
    int b = 1;
    while (b<=a)
    {
      if (a%b==0)
        Sout(b);
      b++;
    }
```

* do while loop :->

syntaxes :->

(1)
```
do
-----------;
while (condition);
```

(2)
```
do
{
-----------
-----------
}
while (condition);
```



↳ Here the body of do while loop executed at least once.

Ex.
```
int a = 1;
do
{
    cout (a) ;
    a++;
} while (a<=10);
}
```

Lec - 21

while loop	do while loop

i) `int i = 1;`
`while (i<=10)`
`{`
`cout(i);`
`}`
output : 1 to 10
	i) `int i = 1;`
	`do {`
	`cout (i);`
	`i++;`
	`}`
	`while (i<=10);`
	output : 1 to 10

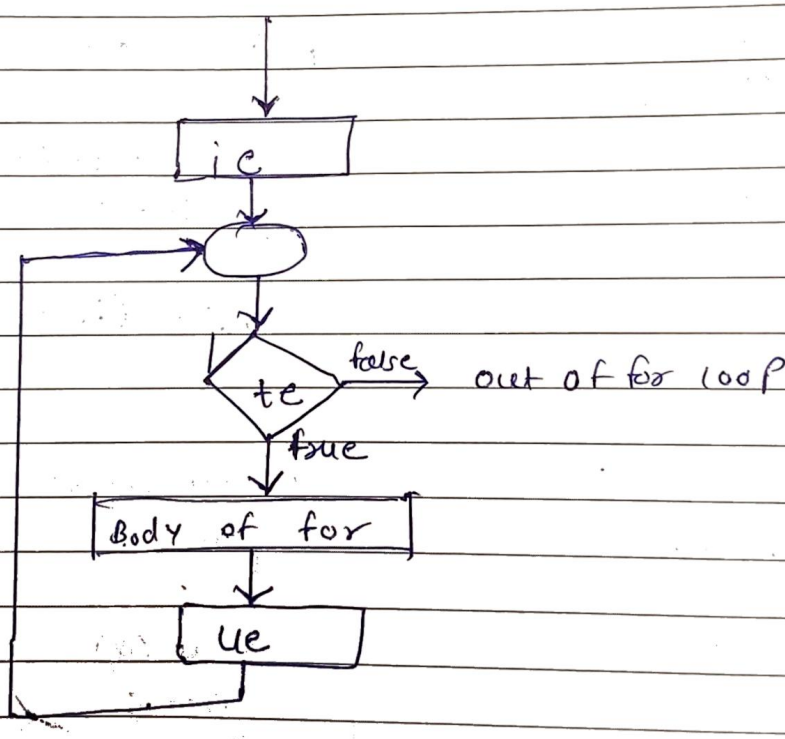
ii) `int a = 5;`
`while (a<5)`
`{`
`cout (a);`
`a++;`
`}`
No output
	2) `int a = 5;`
	`do`
	`{`
	`cout (a) ;`
	`a++;`
	`}`
	`while (a<5);`
	output : 5

* for loop :->

Syantax :->

(i) for (initilization - expression ; test expression ; updation - expression)

(2) for (ie ; te ; ue)
{
;
;
}



Ex ①
```
for (int i = 1; i <= 10; i++)
{
    sout (i);
}
```

output : 1 to 10

In onther way ;

②
```
int i = 1;
for (i <= 10)
for ( ; i <= 10 ; )
{
    sout (i);
    i++;
}
```
output is same

③
```
for (int i = 12 ; a > 5 ; a--)
    sout (a);
}
```

output :-> 12 to 6

④
```
for (int a = 2 ; a <= 10 ; a *= 3)
    sout (a);
```

output :-> 2 6

* Nested loop

↳ A loop that is defined in another loop is called as nested loop.

Ex:
```
for (int i = 1; i<=3; i++)
{
    for (int j = 1; j<=10; j++)
    {
        cout (j);
    }
```

output :- 1 to 10
1 to 10
1 to 10
} 3 times 1 to 10

Ex (2) -
```
for(int i=1; i<=9; i++)
{
    for(int j = 1; j<= i; j++)
    {
        cout (i);
    }
    if (i<9)
        cout (" ");
    else
        cout (" ");
}
```

output :-

1, 22, 333, 4444, 55555, 666666, 7777777,
88888888, 999999999.

③
```
for (int i = 1 ; i<=5 ; i++)
{
    for (int j = 1 ; j<= i ; j++)
    {
        sout (j);
    }
    sout ();
```

output:
1
12
123
1234
12345

④
```
int K=1;
for (int i = 1 ; i<=5 ; i++)
{
    for (int j = 1 ; j<= i ; j++)
    {
        sout (K + " ");
        K++;
    }
    sout ();
```

output
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15

⑤
```
for (int i = 1 ; i<= 5 ; i++) {
    for(int j = 1 ; j<= i ; j++) {
        Sout (" ");
    }
    for (int K=5 ; K>= i ; K--)
    {
        Sout (i);
    }
    Sout ();
```

22
lec

output:
1 1 1 1 1
2 2 2 2
3 3 3
4 4
5

10/11/24

Lec - 22

3] Jump statements :->

1) break statement :->

It terminates the nearest enclosing loop or switch statement.

Ex.

①
```
for(int i=1; i<=10; i++){
    if(i==5)
        break;
    cout(i);
}
```
output: 1 to 4

②
```
for(int i=1; i<=10; i++)
{
    if(i>5)
        break;
    cout(i);
}
```
output :-> 1 to 5

③
```
for(int i=1; i<=10; i++){
    if(i<5)
        break;
    cout(i);
}
```
output = No

④
```
for(int i=1; i<=3; i++)
{
    for(int j=1; j<=10; j++){
        if(j==5)
            break;
        cout(j);
    }
```
output: 1 to 4 {3 times}

② break LABEL statement :->

It terminates the specific LABEL loop.
Here break is a keyword & LABEL is an identifier.

Ex (1)

```
FIRST : for (int i = 1 ; i<=3; i++) {
        SECOND : for (int j = 1 ; j<=10 ; j++) {
                    if (j == 5)
                        break FIRST;
                    sout (j);
                }
        }
```

out of FIRST loop

output 1 to 4 :

③ Continue statement :->

It passes the control to the next iteration of a loop.

Ex.

①

```
for (int i = 1 ; i<=10 ; i++)
{
    if (i == 5)
        continue ;
    Sout (i) ;
```

②

```
for (int i=1; i<=10 ;i++)
{
    if (i>=5)
        continue ;
    Sout (i);
```

output :- 1 2 3 4 6 7 8 9 10

output :-> 1 to 4

```
if (i<5);
  continue;
  sout(i);
}
```

output : 5 to 10

Lec-23

④ Continue LABEL statement :->

↳ It passes the control to the next iteration of specified LABEL loop.

↳ Here, continue is a keyword & LABEL is an identifier.

Ex ①

```
ONE : for(int i=1; i<=3; i++) {
  TWO : for (int j=1; j<=10; j++) {
    if (j==5)
      continue ONE;
    Sout(j);
  }}
```

output : 1 to 4
3 times

Object Oriented Programming (OOP)

↳ Java is an object oriented programming language

OOPL → A language that support all the priciples of an object oriented programming is know as an object oriented programming language

↳ pricipales :→

1) ~~Encr~~ Encapsulation
2) Abstraction
3] Inheritance
4] polymorphism

↳ object oriented = object based + Inhesitence + Runtime polymorphism

object Based Languages

Java script, VB script, Visual Basic :- etc

object oriented Languages :-

C++, Java, C# .Net, python, small talk, Ada, Simula --- etc