LOOPS & PATTERN Programming

18/10/2022

PART - 1

Agenda: Loops ⇒ conditionals (if-else) ⇒ pattern programming
Loops + conditionals + operators ⇒ start to programming.

⇒ Basic structure to print 5 times "*" ?

```
System.out.println("*");
S.O.P("*");
S.O.P("*");
S.O.P("*");
S.O.P("*");
```

output:
*
*
*
*
*

we can do 5 times or 10 times etc. But when comes to print so many times the loop concept required.

or S.O.print("*"); ⇒ ***** (on same line)

(*) There have 4 types of loops :- 1) For, 2) while, 3) do-while
4) For-each (enhanced for loop)

⇒ 1) For Loop :-

The java for loop is used to iterate a part of the program several times. If the number of iteration is fixed or known, it is recommended to use for loop.

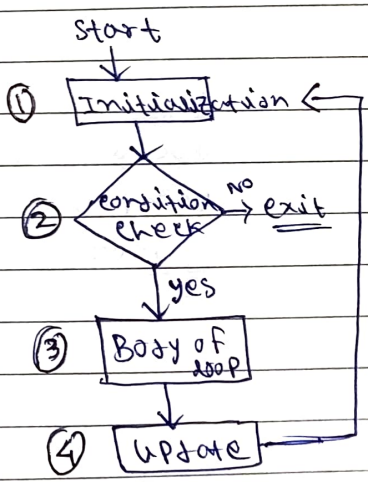
Syntax :- for (initialization ④; condition ②; incre/Decrement ④) {
③ // code here
}

→ Ex :- Print 5 times star "*".

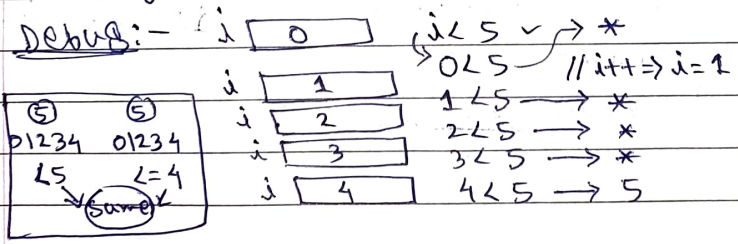
(initialization)(condition check)(update)

```
for(int i=0; i<5; i++){
    S.O.P("*");
}
```

① ② ④ ③



Debug :-



We can also do :-

```
int n=5;
for(int i=0; i<n; i++){
    S.O.P("*");
}
```

→ we can update any time.

output :⇒ * * * * * (same line)

⇒ for loop : → (i) simple for loop, (ii) For-each or Enhanced for loop, (iii) Labeled for loop.
(3 type)

2) ⇒ While loop :- (as long as)

The java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed or unknown, it is recommended.

Syntax :-

```
initialization ;   // optional      ①
while (condition) {                 ②
    // code Body                    ③
    incre / Decrementation ;        ④  → optional
}
```

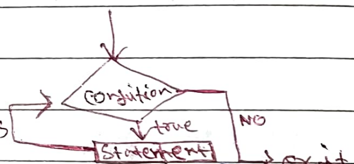
⊛ Where initialization & inc/Decrementation are optional. You can use only as long as (while (condition)) part perticularly.

ex :-

```
int i = 0 ;          ①
while ( i < n ) {    ②
   S.O.P ("*") ;     ③
   i++ ;             ④
}
```

output :- * * * * *

⊛ for loop & while loop is same only syntax change.



3) ⇒ do - while loop :- (exit control loop)

use it if the number of iteration is not fixed & unknown and you must have to execute the loop at least once.

Syntax :-

```
initialization ;  // int i = 0      ①
do {
   S.O.P ("*") ;                     ②
   i++ ;                             ③
} while ( i < 5 ) ;                  ④
```

int i = 0 1 (update) ⇒ *
1 < 5 ⇒ *   2

⇒ Difference between while( ) & for( ) loop ?

for( )	while
① Here, 3 things are mandatory - initialization, condition and incre / Decrement (update)	② only condition is mandatory initialization & update in optional

⇒ Java Infinitive While Loop :-

If you pass true in the while loop, it will be infinite while loop. (ctrl+c ⇒ to exit)

Syntax :

```
while (true) {
    // code to be executed
}
```

output :- Till Infinity.

ANY PATTERN BY FOR LOOPS

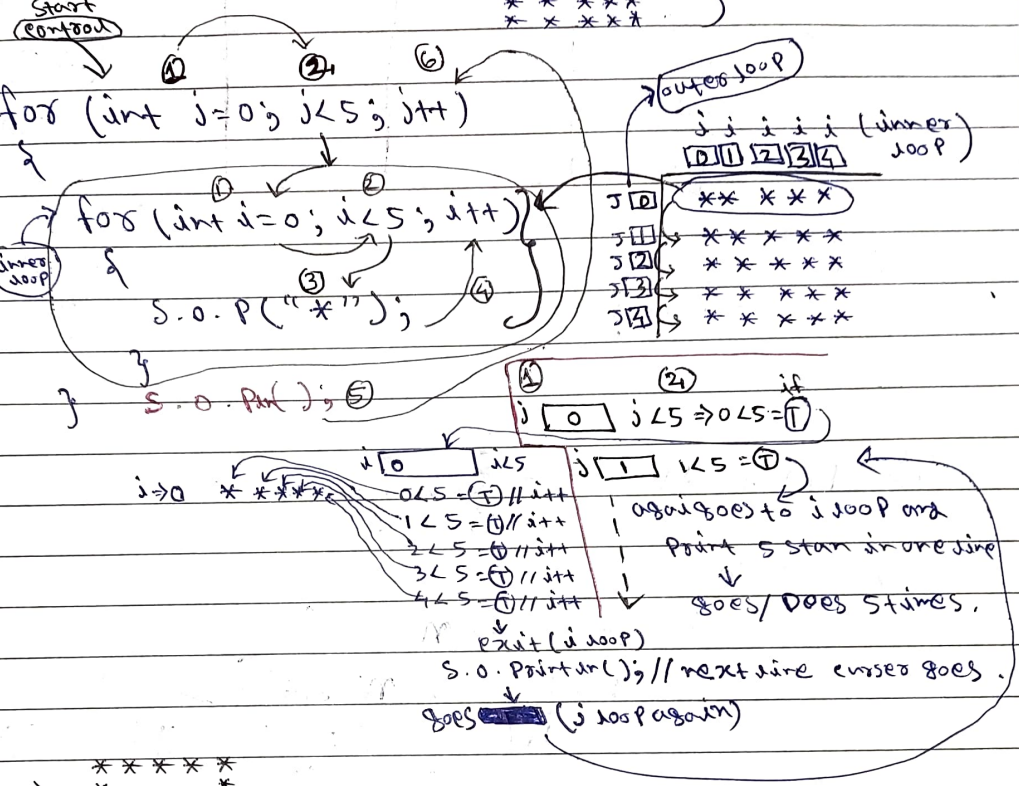
⭐ Nested Loop :-

When we want loops under a loop that is nested loop.

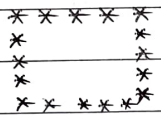
a) ex:- we want 5 "*" on same line and 5 "*" of row on the next line. →

Solution:-

```
for (int j=0; j<5; j++)
{
    for (int i=0; i<5; i++)
    {
        S.O.P("*");
    }
    S.O.Println();
}
```



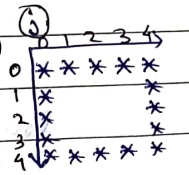
b) Print ⇒



Ans To create this we use nested loop, inside it use if-else (conditional statement) and logical operators.

Solution:-

```
int n=5;
for (int i=0; i<5; i++){
    for (int j=0; j<5; j++){
        if (i==0 || j==4 || i==4 || j==0){
            S.O.P("*");
        }
        else{
            S.O.P(" ");
        }
    }
    S.O.Println();
}
```



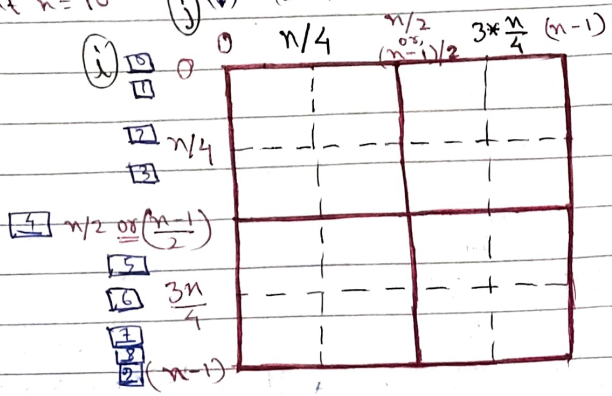
n = 5
(n-1) = 4
we consider 4 = (n-1) the last

i = 0 i<n ⇒ 0<4 = T
j = 0 0<5 = T (if) check
j = 1 1<5 = T [T F F F F]T
j = 2 2<5 = T [T F F F]T
j = 3 3<5 = T [T F F F]T
j = 4 4<5 = T [T T F F]T

18/10/2022
motivation
2:00hr

All Pattern's intersec ⇒

int n = 10



∵ i = rows
j = columns

n=5 suppose

∴ 0, 1, 2, 3, 4 = 5

i < 5 / i ≤ 4

End { ∴ (n-1) ⇒ (5-1) = 4 ← the last index

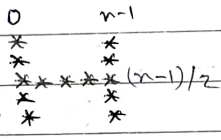
middle { ∴ 5/2 = 2.5 or ⇒ (n-1)/2 ⇒ 5-1/2 = 4/2 = 2 (n/2)

Both are used for almost middle

First middle's middle (quarter) { ⇒ n/4 ⇒ 5/4 ⇒ 1.25 approx

last middle's middle (3× quarter) { 3 × n/4 ⇒ 3×5/4 = 3.75

Q) Print this ⟶



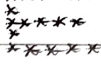
⇒
```
int n = 5;
for (i=0; i<n; i++){
    for (j=0, j<n; j++){ → (n-1)/2
        if ( j==0 || j==(n-1) || i=(n-1)/2){
            S.O.P ("*");
        }
        else {
            S.O.P (" ");
        }
    }
    S.O.println();
}
```

Q) Print this ⟶

"

```
if (i==(n-1)/2 || j==(n-1)/2) {
    S.O.P ("*");}
```

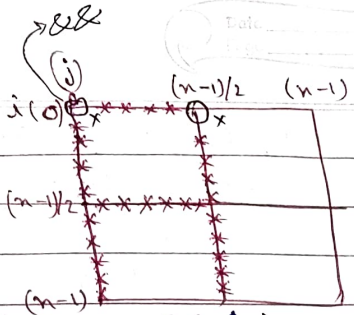
Q) Print E ⟶



```
if (i==0 || j==0 || i==(n-1)/2 || i==(n-1)) {
    S.O.P ("*");}
```

Qs) Pattern A →

```
int n=10;
for (int i=0; i<n; i++){
    for(int j=0; j<n; j++) {
        if ( i==0 && (j!=0/j>0) || j==0 && j>0 && j<(n-1)
            || j==(n-1) && i>0 || i==(n-1)/2) {
            S.O.P("*");
        }else{
            S.O.P(" ");
        }}
    S.O.Pln();
}
```



(*) By using this above logic you can print all letters except - X, Y, M, W, V, Z, N, R, Q it requires diff logic.

Snippets:- 18/10/2022

Qs) (1)

```
Public class Test{ P.S.V.m(SI args) {
    char c='z';
    long l=100_00l; // from JDK 1.7 compiler will
    int i=9_2;      automatically remove all
    float f=2.02f;  "_" when creating .class file
    double d=10_0.35d;

    l=c+i; //char+int => int   int ---> long (implicit)
    f=c*l*i*f; // char * long * int * float = float   long ---> float (implicit)
    f=l+i+c; // long + int + char = long   long ---> float (implicit)
    i=(int)d; //double ---> int X (explicit ☑)   int ---> long (implicit)
    f=(long)d; // double ---> long (explicit), long ---> float (implicit)
}}
```

→ Does above code compile sucessfully? (yes)

Qs) (2)

```
int a=20 //a=18//
int var=--a*a++ + a-- ---a; // 19*19+20-18=363
S.O.Pln("a="+a);
S.O.Pln("var="+var);
```

output? - a=18, var=363 (Ans)

Qs) (3)

```
int i=5;
if(i++<6){ S.O.Pln(i++); }
```

i=5 → 5<6 (true) → a=6 → print ⇒ a=6 (Ans) → a=7

Qs) (4)

```
int x=4; // Line 1 OK
int y=4++;
int z=++(++x);
```

(Compile time Error) whether it is post or pre-increment it can only be done on variables not on direct literals.

8) byte c=(10>20)? 30:40; // Literals are involved so compiler performs operation ⇒ byte c=40;
byte d=(10<20)? 30:40; // d=30.
S.o.Pln(c); // c=40
S.o.Pln(d); // d=30 } Ans

(*) Q) int a=10; b=20; // type checking is valid no problem.
byte c=(a>b)? 30:40; // literals are not involved in operation
(operation)
So, compiler would just check type checking of result so, compiler will see 30,40 type it knows is int, so the result should be of int type only. If compiler only perform the operation it will try to map with casting chart otherwise it wants the exact type.
byte d=(a<b)? 30:40; // compiler Error (same ↑)
S.o.Pln(c); // C:E
S.o.Pln(d); // C:E } Ans

Q) int x=5____0; // valid (compiler wise remove)
int y=____50; // invalid bez start with _ } (X)
int z=50___; // invalid bez end with _ } (X)
float f=123.76-86f; // valid
double d=1_2_3_4; // valid

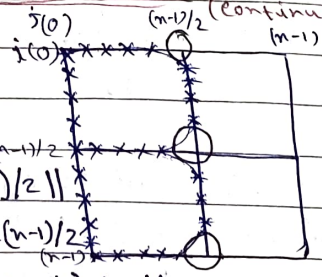
(*) NOTE: compile ⇒ it checks only the syntax and makes the jvm execution smoothfull.
Jvm ⇒ creats the memory for the variables and perform type casting and generates the result.

I.8) Why float is larger than long when its size is smaller than long?
Ans:- To tally depends on internal architecture.
long → 8 bytes (size) } ex:- 1 class room --> 8 benches are there..
float → 4 byte } 1 person can sit --> 8 students can be accomodated
1 class room --> 4 benches are there -->
1 person can sit --> 8 students can be accomodated

19/10/2022

(PART-2)

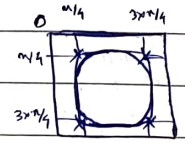
PATTERN PROGRAMMING



(continued)

⊗ Pattern - B --->

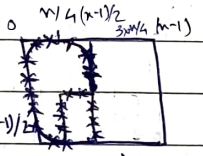
```
if ( j==0 || i==0 && j<(n-1)/2 ||
   j==(n-1)/2 && i>0 && i<(n-1)/2 ||
   j==(n-1)/2 && i>(n-1)/2 && i<(n-1)/2
   || i==(n-1)/2 && < j==(n-1)/2 ||
   i==(n-1) && i<(n-1)/2) {
   S.o.P("*");
   }
   else {
      S.o.P(" ");
   }
```



⊗ Pattern - □ ---->

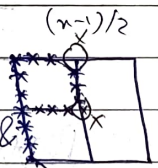
```
if (i==n/4 && j>n/4 && i<3*n/4 || j==n/4 && j>n/4 && i<3*n/4
   || i==3*n/4 && j>n/4 && j<3*n/4 || j==3*n/4 && i>n/4 && i<3*n/4)
```

⊗ Pattern - G ---->




```
if (i==0 && i>0 && i<(n-1) || j==0 && j>0 && j<(n-1)/2
   || j==(n-1)/2 && i>0 && i<=n/4 || i==(n-1) && j>0 && i<=n/4
   || j==n/4 && i>=(n-1)/2 || i==(n-1)/2 && j>=n/4 && i<=(n-1)/2
   || j==(n-1)/2 && i>=(n-1)/2) {
   S.o.P("*"); }
```

⊗ Pattern - P ---->




```
If( i==0 || i==0 && j<(n-1)/2 || j==(n-1)/2 &&
i>0 && i<(n-1)/2 || i==(n-1)/2 && j<(n-1)/2) { S.o.P("*");}
```

⊗ Print A BC Together ---->

```
int n=10;
(outer loop) for(int i=0; i<n; i++){
(inner loop)    for(int j=0; j<n; j++){
for = A and exit)    if( ) {S.o.P("*")} else {S.o.P(" ")}
                }
```

=> // Don't end outer loop and end ore patter inner loop and you can also add space between two pattern, for space create a inner j (loop) and match condition by how much space you want (j<n/2, j<n/4, j<(n-1)/2, j<(n-1))

for space between (A & B) ⇒ // For Space

```
for (int j = 0; j < n/2 ; j++){
    System.out.print (" ");
}
```

n = 10

j [0] 0 < n/2 ⇒ 10/2 = 5

0 < 5 = T

for (B) ⇒

```
for (int j = 0; j < n; j++) {
if() { S.o.P ("*"); } else { S.o.P (" ") }
} // end B's inner loop
```

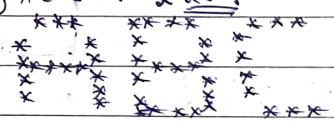
for (C) ⇒

```
for (int j = 0; j < n; j++){
    if(){ S.o.P("*"); } else {S.o.P(" ")}
} // end C's inner loop
```

⇒ `S.o.Pln( );` // print a next line for every row (i)

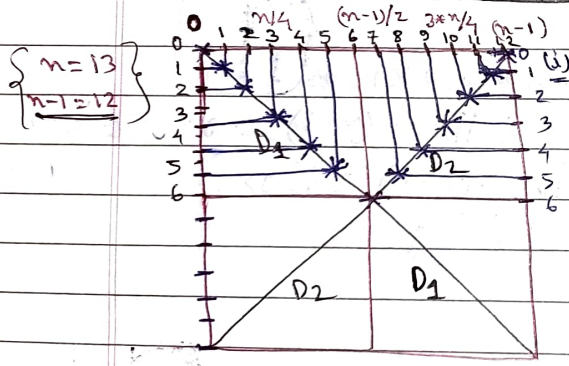
} // end of i loop

output:



2nd logic

⇒ How to write (I X Y Z N M Q W V) ? —



⇒ D1 ⇒ i == j

0	0
1	1
2	2
3	3
4	4
5	5

⇒ D2 ⇒ i + j == n-1 (n = 13)

0 + 12 ⇒ 12 ⇒ (n-1) ⇒ (13-1) ⇒ 12

1 + 11 ⇒ 12

2 + 10 ⇒ 12

3 + 9 ⇒ 12

4 + 8 ⇒ 12

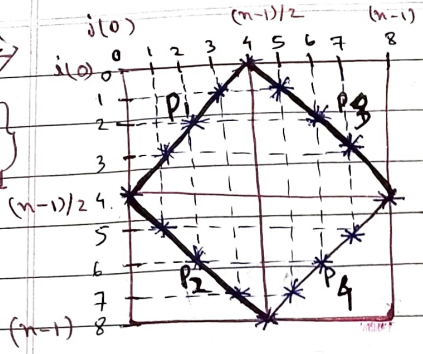
Q) Pattern X ---->

```
{ if ( i == j || i + j == (n-1) || j == 0 || i == (n-1) || j == 0 || j == (n-1))
    { S.o.P ("*"); } else { S.o.P (" "); }
}
```

Q) Pattern N ---->

```
{ if (j == 0 || i == j || j == (n-1)) { S.o.P ("*"); } else {S.o.P(" ");
}
```

3rd Logic



if n = 9, n-1 = 8, (n-1)/2 = 4

⇒ **P1 ⇒ i + j == (n-1)/2**

0 + 4 = 4 ⇒ (n-1)/2
1 + 3 = 4 ⟶ ,,
2 + 2 = 4 ⟶ ,,
3 + 1 = 4 ⟶ ,,

⇒ **P2 ⇒ i - j == (n-1)/2** (P.T.O)

4 - 0 ⇒ 4 ⟶ (n-1)/2
5 - 1 = 4 ⟶ ,,
6 - 2 = 4 ⟶ ,,
7 - 3 = 4 ⟶ ,,

⇒ **P3 ⇒ i + j == (n-1) + (n-1)/2**

∴ n-1+(n-1)/2 ⇐ 9-1+4 ⇒ 12

i + j	(n = 9) (n-1) = 8
4 + 8	= 12
5 + 7	= 12
6 + 6	= 12
7 + 5	= 12
8 + 4	= 12

⇒ **P3 ⇒ j - i == (n-1)/2**

j - i = 4	i	j
4 - 0 = 4	0	4
5 - 1 = 4	1	5
7 - 3 = 4	2	6
8 - 4 = 4	3	7
	4	8

Qs) Print K ⟶ 

```
for(i){
    if(j==0 || i+j==(n-1) || i-j==(n-1)/2){
        S.o.p("*");
    }else{
        S.o.p(" ");
    }
}
```


i = j
i + j = n-1


i + j = (n-1)/2
i - j = (n-1)/2

P4 = j - i = (n+1)/2

P3 i + j = (n-1) + (n-1)/2

Qs) Print Q ⟶

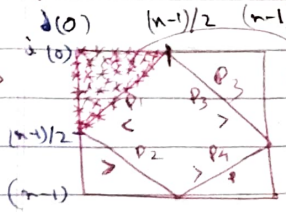
```
if(i==n/4 && j>n/4 && j<(3*n)/4 || i==(3*n)/4 && j>n/4
   && j<(3*n)/4 || j==n/4 && i>n/4 && i<(3*n)/4
   || j==(3*n)/4 && i>n/4 && i<(3*n)/4
   || i==j && j>(n-1)/2){ S.o.p("*");
}}
```

Qs) Print Z ⟶

```
if(i==0 || i+j==(n-1) || i==(n-1)){S.o.p("*")}else{S.o.p(" ")}
```

i + j = n-1

⇒ Triangle Pattern/Shape logic



⇒ we know P_1. and we also just fill that P_1 area. thats it.

Q) Print ---> P1 (use formula for P_1 > cover it by other side > make those P_1 formula from == to >= or <=)

```
if ( i+j <= (n-1)/2 || i==0 && j<=(n-1)/2
     || j==0 && i<=(n-1)/2 ) ---> you get this
{ S.o.p("*") } else { S.o.p(" "); } }
```

to find inner area just change P_1 formula == to <=

Q) Print ---> P3 (change P3 formula as above)

```
if (j-i >= (n-1)/2 || i==0 && j>=(n-1)/2
     || j==(n-1) && i<=(n-1)/2){
S.o.p("*") } else {S.o.p(" ");} }
```

Q) Print ---> P2 (change P_2 formula as above)

>= ✓ <= ✗

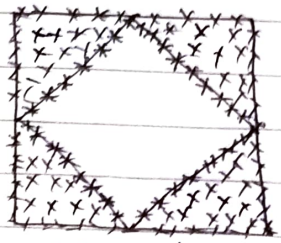
```
if (i-j >= (n-1)/2 || j==0 && i>=(n-1)/2
     || i==(n-1) && j<=(n-1)/2){....
```

Q) Print ---> P4 change P_4 formula as above

```
if( i+j >= (n-1)+(n-1)/2 || j==(n-1) && i>=(n-1)/2
     || i==(n-1) && j>=(n-1)/2){
```

Q) Print ---> P1 P3

```
if ( i+j <= (n-1)/2 || j-i >= (n-1)/2 || i==0
     || j==0 && i<=(n-1)/2 || j==(n-1)&& i<=(n-1)/2){...
```

Q) Print ---> 

```
if ( i+j <= (n-1)/2 || j-i >= (n-1)/2 || i-j >= (n-1)/2
     || i+j >= (n-1)+(n-1)/2 || i==0 || j==0 || i==(n-1)
     || j==(n-1) )
```

8) Print this ---> 

```
if ( i+j >= (n-1)/2 && i<=(n-1)/2 && j<=(n-1)/2
||j-i<=(n-1)/2 && i<=(n-1)/2 && j>=(n-1)/2
||i-j <=(n-1)/2 && i >=(n-1)/2 j <= (n-1)/2
|| i+j <= n-1+(n-1)/2 && i>(n-1)/2 && j>(n-1)/2)
}
```

=> So, From now you can draw/print any pattern. (only one approach left)

8) Print this ---> 

```
if( i+j >=(n-1)/2 && j<=(n-1)/2 && j<=(n-1)/2
   || j-i<=(n-1)/2 && i<=(n-1)/2 && j>=(n-1)/2)
```

8) Print this ---> 

```
if ( i-j <=(n-1)/2 && i>=(n-1)/2 && j<=(n-1)/2
   || i+j <=(n-1)+(n-1)/2 && i>(n-1)/2 && j>(n-1)/2
   || i==(n-1)/2){S.o.P("*");}esse{So.P(" ");}}
```

Assignment

(1) 8) Print this ---> 1 1 1 1 1
2 2 2 2
3 3 3
4 4

```
int n=4;
for(j=0;j<n;j++){S.o.P(1+" ");} //for 1111
    System.out.println();
for(j=0;j<n;j++){S.o.P(2+" ");} //for 2222
    S.o.Pln();
for(j=0;j<n;j++){S.o.P(3+" ");} //for 3333
    S.o.Pln();
for(j=0;j<n;j++){S.o.P(4+" ");} //for 4444
```

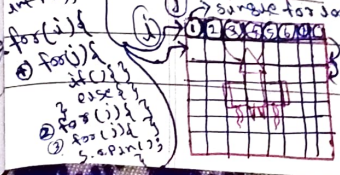
Ass:(2)

8)

```
int n=19;
if(i==0 || j==0 || i==(n-1) || j==(n-1)
   i+j <=(n-1)/2 || j-i<=(n-1)/2)   // i==(3*n/26)
```

Output 

(*) Complex pattern logic :-



=> If you want to print any pattern -> then consider one outer loop(j) and each and every small block consider as inner small for loop P(j) and apply our previous learned logic and after our for loop P(j) S.o.Pln(); apply before j loop end.

PTO

↳ for the next row again creat a (i) loop and same nested (j) loop to get desire output

Code snippets

⑥ Q>
```
int x=10, y=15
if(++x<10 || ++y>15){        // 11<10 (false) || 16>15 (true)
    x++;
}else{
    y++;
}S.o.p(x+"----"+y);
```

⇒ Predict x and y value by replacing the operators with:-

→ || ⇒ x and y value?
  x=?
  y=?

→ && ⇒ x and y value?

→ | ⇒ x and y value?

→ & ⇒ x and y value?

⑦ Ans

|| ⇒ it is called as "Short circuit OR operator"
- means, second operand evaluation will happen only if the first operand result is false. if 1st operand is true JVM will never go to the second operand and execute.

true (1st operand) → 2nd operand
false (1st)

x = 11, y = 16

&& ⇒ it is called as "Short circuit AND operator"
means ⇒ if the first operand evaluation result false then JVM never goes to second, bez the result will false always. if true then go.

true (1st) → 2nd
false (1st) ✗

x = 11, y = 16 ⇒ Bez JVM don't evaluated 2nd operand.

| ⇒ it is called as "Logical OR operator"
means → it will evaluate all the operand and create boolean result (true/false). Both ⇒ JVM will execute first and second operand. (x=11, y=16)

& ⇒ it is called as "Logical AND operator"
means → JVM will execute both 1st and 2nd also

1st (true/false) → 2nd
(true/false) 1st ope. → 2nd ✓ there

⑧ Q>
```
int x=10;
if(++x<10 && (x/0)>10){
    S.o.Pln("Hello world");
}
else{
    S.o.Pln("Hi");
}
```

x = 10
⇒ 11 < 10 && .!0/0 > 10.)
(False) ✗ don't go

⇒ output ⇒ Hi

Q) int x = 10;
if(x){ S.o.p("hello");
} else { S.o.p("hiee");
}
=> So, if(x) (integer) // x = 10 => Compile Time error: CE: Unexpected type required: boolean, found: int.

if syntax:
=> if (boolean) {
Stmt-1;
Stmt-2; }
else {
Stmt-3;
Stmt-4; }

Q) boolean b = false; // b = false (bez, in boolean we can give true/false)
if (b = true) { // assignment operator is evaluted on boolean
S.o.p("hello"); data type, jvm if (true) print. the true value.
} else {
S.o.p("hiee");
}

Q) if you give if(b == true) => false == true => false => Print hiee

Q) if (boolean)
Stmt-1;

NOTE:- if there is only one statement which needs to be a part of if, then {} is option.

=> if (true)
System.out.println("hello");

Ans hello

Q) Public class Test {
P S V M (String[] args){
if (true){
int x = 10; //CE: declaration not allowed here only statement allowed here because you are not giving {}.

Q) if(true){
int x=10; // valid for compiler bez of {}.
}

Q) if (true)
S.o.p("hello"); // No CE error But, // Dependent of if statement
S.o.p("hiee"); // Independent of if statement it will not print bez you don't use {}

output => hello

Number Pattern Logic

```
int n=4;
for(int i=0; i<n; i++){
    for(int j=1; j<=n; j++)
    {
        System.out.print(j);
    }
    System.out.println();
}
```

Output:
1234
1234
1234
1234

```
int n=4;
for(int i=1; i<=n; i++){
    for(int j=1; j<=n; j++){
        S.o.p(i+" ");}
    S.o.Pln();}
```

Output:
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4

```
int n=5;
for(int i=1; i<n; i++){
    for(int j=1; j<n-i; j++){
        S.o.p(j);}
    S.o.Pln();}
```

Output:
1 2 3 4
1 2 3
1 2
1

Same as above

```
S.o.p(i);
}
S.o.P();}
```

Output:
1 1 1 1
2 2 2
3 3
4