OPERATORS, CONDITIONALS, SWITCH IN JAVA

Date 17/10/2022

PART-3

⇒ Operators:-

Operators are special symbols that perform specific operations on one, two or three operands, and then return a result.

→ Types of operators:—

(i) Arithmetic operators: (+, -, /, *, %)

(ii) Increment & Decrement operators: (++, --)

(iii) Logical operators: [redacted] (&&, ||, !)

(iv) Assignment operators: (=) → compound Assignment operator (+=, -=, *=, /=, %=)

(v) conditional: (if, else)

↓

Ternary operators:-

(vi) Relational / comparison operator:- (>, <, >=, <=, ==, !=)

(vii) Bitwise operators:- (&, |, ~)

(i) Arithmetic operators:-

+, -, *, /, % (addition, Substraction, multiplication, [redacted] Divition, modulo operator)

% ⇒ (modulo) It gives the reminder.

int a = 10 % 2 ⇒ 0 (3 | 10 | 5)

a = 0 % 10 ⇒ 10) 0 (0

0

(ii) Increment & Decrement operators:- ++, --

(iii) Logical operators:-

AND	OR	NOT
&&	||	!

⇒ It gives Boolean value.

AND ⇒ &&

exp1	exp2	exp3	Result	
T	T	T	⇒ T	if all true ⇒ T
T	T	F	⇒ F	if 2 true 1 false ⇒ F
F	T	T	⇒ F	
T	F	T	⇒ F	
F	F	F	⇒ F	if all false ⇒ F

→ when entire expression are true then it will true other wise it will false if, even only one will false the total result will false.

⇒ T && T && T && T ⇒ T

⇒ T && F && T && T ⇒ F

OR ⇒ ||

exp1	exp2	exp3	Result	
T	T	T	⇒ T	
T	F	F	⇒ T	
T	T	F	⇒ T	
T	F	T	⇒ T	
F	F	F	⇒ F	

→ when entire expression are false, then it will false. otherwise, it will give true only if even one statement is true.

⇒ F || F || F || F ⇒ F

⇒ T || F || F || F ⇒ T

⇒ T || T || T || F ⇒ T

NOT ⇒ !

!T ⇒ F

!F ⇒ T

[redacted]

- S.O.P(!true);
  // Prints False
- S.O.P(!false);
  // Prints true.

int count = 2;

S.O.P(!(count > 2)); // true (false)

S.O.P(!(count <= 2)); // false (true)

⇒ Relational operator:- / comparision operator:-

(act like OR operator) >, <, >=, <=, ==, !=

It gives boolean value only.

ex:- int a = 10; → (a>b) => false
int b = 20; (a<b) = true.
S.O.P(a==b); // false (a<=b) = true.

→ == (equal to), != (not equal to), = (assignment)

⇒ Assignment operator:-

→ (i) Single Assignment operator:- (=)

int a; → So, whatever present in right side
a = 10; assign that value to the left side.
Whenever we use = (assignment operator)
(Single)

→ Chained Assignment operator:-

int a; ⇒ int a,b,c,d;
int b;
int c;
int d;

a = b = c = d = 10; }- chained A.op

a [10]
b [10]
c [10]
d [10]
d=10

→ Compound Assignment operator:- (a = a+20)

int a=10; (+=, -=, *=, /=, %=)

a += 20 // ⇒ 10+20 = 30 So, Doing operation during assignment here.
a -= 20 // 30-20 = 10
a *= 20 // 10*20 = 200
a /= 20 // 200/20 = 10
a %= 20 // 10 %. 20 = 0

(*) Unary operators:- (=, ++, --, and compound Assign. op)

only one operand is sufficient to perform any expression.

int a = 10; // a is a unary operand
(operand)

or, a++ or ++a

(*) Binary operators:- (+, -, ==, >, &&, etc)

more than one operand is required to perform a operation/expression

(a) + (b), (a) - (b). So, and logical, Arithmetic

(a) == (b) (a) < (b) operators are Binary operator.

Conditional

⇒ Whe you are going to perform any task / operation / activity Based on condition. like if-else

⇒ **if-else :-**

```
int a = 10;
int b = 5;
if (a>b) {
    int res = a - b;
    S.O.P(res);
}
else
{
    int res = a+b;
    S.O.P (res);
}
```

(if block: `if (a>b) { int res = a - b; S.O.P(res); }`; else block: `else { int res = a+b; S.O.P (res); }`; false → else block; true → if block)

⇒ `if (a>b) { }` — true → if the condition is true then it will run whatever present in the {} (a-b)

- if false then don't run the {}. directly exit. (a+b)
- you can make a else statement if you want to exect your condition false and run a certain things.

in if or if-else block only one block will exeented. it does not matter how many condition/block you created.

⇒ **else - if :-**

you can check multiple condition by using else-if in ifelse statement.

ex:-
```
int a=10;
int b=2;
if (a>b) {                    ——— (cond 1)
    S.O.P (a-b);
}
else if (a==b)  → (cond 2)
{
    S.O.P (a+b)
}
else if (a<b) {  → (cond 3)
    S.O.P("a is lesser");
}
```

⇒ Nested if-else:-

we can write multiple if, else, else-if in bether a another if, else, if-else block.

P) Find the least, in between 3 number?

⇒
```
int a = 10;
int b = 20;
int c = 5;
⇒ if (a<b){
        if (a<c){
            S.O.P("a is Least number");
        }else{
            S.O.P("c is least");
        }
    }
⇒   else if (b<c){
        S.O.P("b is least");
    }
    else{
        S.O.P("c is least");
    }
```

⇒ output:
c is least

→ you can also use &&, || operator in condition of if-else

(*) ⇒ Ternary operator:- (-?-:-)

```
int a = 10;
int b = 20;
if (a>b){
    S.O.P(a);
}
else{
    S.O.P(b);
}
```

⇒

```
int a = 10;
int b = 20;
int result = (a>b)? a:b;
S.O.P(result);
```

(a>b)? a:b;
condition if true if false

output ⇒ 20

⇒ Nested Ternary operator:-

same least finding example

```
int a = 10;
int b = 20;
int c = 30;
int d = (a<b)? (a<c? a:c):(b<c? b:c);
S.O.P(d);
```

⇒ output = 30

NOTE:=
(10<20)? 30:40; ⇒ compile ✓
(a<b)? c:d; ⇒ compile ✓
(10<20)? a:b; ⇒ ✗ compile ✓
(a<b)? 10:20; ⇒ ✗ compile ✓
So, no problem

the result type and storing variable type should be same.

=> Switch case:-

Instead of writting so many if.. else statements you can use switch statement.

→ syntax:-

```
Switch (expression){
case x:
    // code block
    break;
case y:
    // code block
    break;
default:
    //code block
}
```

=> How it works?

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- The break and Default keywords are optional.

(fallthrough)

- NOTE: (i) When the condition is true and matches a case jvm print the code inside the block as well as next case's code if present. So, these why we use break command to exit from the matched case.

(ii) If no cases matching you can use default: to print/do some expression/commands.

code (Snippet)

Q5> What will be the result of compiling and executing Test class?

```
Public class Test{
    Public S v m (String[] args){
        byte b1 = (byte) (127+21); // byte b1=(byte)(148)
        S.O.P (b1); }}
```

Options => a>148, b>Compilation Error, c> -108, d> -128.

Variable overflow formula:- ////////

Ans JVM: minrange + (result - maxrange - 1) ⭐

= -128 + (148 - 127 - 1)

= -128 + 20

= -108 (Ans) ©

Formula ⭐

Q) For the given code what is the output?

```
int x = 100;
int a = x++;
int b = ++x;
int c = x++;
int d = (a<b)? (a<c)? a : (b<c)? b : c : x;
S.o.p(d);
```

- options :- a) 100, b) 101, c) 102, D) 103, E) compilation fail

Ans:-

x = 100

a = 100 // x = 101 exp = (a < b)? ⇒ (100 < 102) yes

b = 102 // x = 102 (a < c)? = (100 < 102) yes

c = 102 // x = 103 ∴ (100 < 102)? (100 < 102)? (100) Print

∴ d = 100 (Ans (a))

Q)

```
class Test {
    P.S.V.m(S[] args){
        int a=100; // a=101
        S.o.P(-a++); }} // S.o.P(-100); now a = 101
```

ans = (-100)

⇒ Fallthrough in Switch :-

```
int a=97;
switch (a) {
    case 97: S.o.P("hello");
    case 98: S.o.P("hiee");
}
```

(true)

→ Output :- hello
hiee } fallthrough

- Siree there is no break automatically control execute the next case also, this condition in java inter switch we call as "fallthrough".

Q)

```
javac Test.java
java Test ind.txt
java Test ans.txt
```

Why we need command line? (Parameterized)

ind.txt: rahul, rohit, rohit (five)

ans.txt: Smith, Warner, Finch (five)

```
Public class Test {
    P.S.V.m(String[] args){
    // file name will be supplied from the command line arguments
    // code written to open the file and read the contents from the file.
}}
```