19/02/2024

# Var {added in JDK-10}

↳ Context sensitive Keyword

→ Can be used only for local variables.

eg → double db = 2.5;
OR
Var db = 2.5; ✓

class A{
psvmc_){
Keyword ← Var ab = 2.5; ✓
identifier ← int Var = 10; ✓

{ even
Var Var = 2.5; ✓ }

To create array using Var

① Var ma = new Int[10]; ✓

② Var ma[] = new int[10]; ✗

③ Var [] ma = new int[10]; ✗

④ var ma = {1, 2, 3}; ✗

## Limitation

① This can be used only with local variables.

② Can't be used to define the name of class, interface, enumeration & annotation.

③ Can't be used as exception type in catch statement.

# Control Statements

① Selection statements → If, Switch.

② Iterative statements → while, do-while, For → Same as C, forEach

③ Jump Statements → Break → Same as C, Labeled breaks; Continue → Same as C, Labeled Continue; Return → Simple Return, Return with expression

## Selection Statements

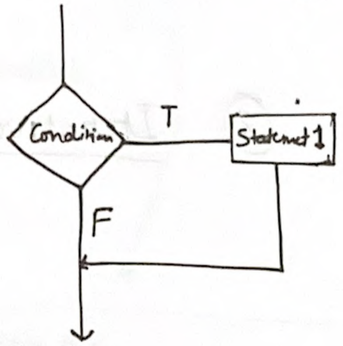
① If → if, if-else, if-else-if ladder, nested-if.

20/2/24

① Simple if ⇒ Execution of statements or set of statements will be dependend upon the condition given in if statement.

Syntax ⇒

```
if (condition){
multiple statements
}
```




Or

```
if (condition) statement;
```

Single statement.

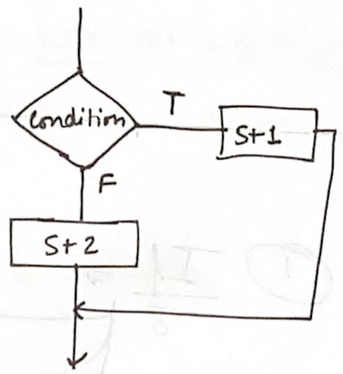
{ Result of condition has to be Boolean }

---

② If-else ⇒

Syntax ⇒

```
if (condition)
{
}
else{
}
```




Q Find out larger out of two no.

```
class A {
    PSVM(_){
        int a, b;
        a=10, b=15
        if (a>b)
            S.o.P("a is larger");
        else
            S.o.P" a is not larger");
    }
}
```

OR

③ If-else-if-ladder ⇒

```
int a, b;
a=10, b=15;
if (a>b)
    S.aP ("a is larger");
else if (b>a)
    S.o.P(" b is larger");
else
    S.aP(" equal");
```

```
if (a>b)
    St1;
    St2;
else if (b>a)
    St3;
else
    St4;
```

→ without brackets will give us an error of misplaced else

```
int a=5;

if (a>0)
    S.o.P("+ive no.");
    S.o.P("Not -ive");

if (a<0)
    s.o.P("-ive no");

else
    s.o.P("0");
```

Output
⇒
→ +ive no.
→ 0

Lets say a=-5
Output
→ Not -ive
→ -ive no.

## (4) Nested if-else →

Q Largest among three no. :->

```
int a, b, c;

if (a > b)
    if (a > c)
        sOP("a");
    else
        s.o.P("c");
else
    if (b > c)
        S.O.P("b");
    else
        s.o.P("c");
```

```
{ if (a > 0); }
```

Empty if statement.

# Switch Statement

→ Multiway selection statements.

(1) From JDK-8 we can use strings also in switch expression, before that only byte, char, int, short & enumeration were allowed in switch statement.

(2) from jdk-14, four new features were added in switch

- Switch expression
- Yield
- Case arrow
- List of case elements.

eg→ with out switch

```
int day no = 3;

if (day no == 1)
    SOP("Monday");
else if (day no == 2)
    SOP("Tuesday");
else if (day no == 3)
    SOP("wednesday");
:
else if (day no == 7)
    SOP("sunday");
else
    SOP("Not Valid");
```

## With Switch

① No duplicate

{We cannot have cases with same value}

```
int dayno = 3;

switch (dayno) {
    Case 1 :
        SOP("Monday");
        break;
    Case 2 :  SOP("Tuesday");
        break;
    :
    Case 7 :  SOP("Sunday");
        breaks;
    default : SOP("Not Valid");
}
```

Multiple Cases

1 Optional default

② Case Label will always be constant expression (can't be variable)

```
int i = 2;
switch (i){
    case 1 :  SOP("Monday");
              break;
    Case 2:   SOP("Tuesday");
    case 3:   SOP("wednesday");
    case 4:   SOP("Thursday");
              break;
}
}
```

Output :→
Tuesday
wednesday
Thursday

{Because of fall through rule}

21/2/24

## Switch Expression & yield (JDK-14)

```
int days = Switch()
{
    ____
    ____
}
```

```
int month, days;
month = 4;
days = Switch(month)
{
Case 1,3,5,7,8,10,12: Yield 31
Case 4,6,9,11: Yield 30;
Case 2 : Yield 28;
default: Yield 0;
}
```

### Arrow Case:→

```
int days;
int month = Sc.nextInt();
days = switch (month){
    Case 1,3,5,7,8,10,12 → 31;
    Case 4,6,9,11 → 30;
    Case 2 → 28;
}
```

## Cases of ~~Arrow Case~~ Switch Constant ⇒

① Case Constant → expression

② Case Constant → {
Block of statement
}

③ Case Constant → Throw exception

## Traditional Switch & Arrow ⇒

- Shorthand for switch expression
- No fall through.
- Efficient way of using Yield.

```
int days;
int month = Sc.nextInt();
days= Switch (month)
    {
        case 1,3,5,7,8,10,12 → {
                S.o.P ("These are 7 month");
                Yield 31;
        }
        case 4,6,9,11 → {
                S.o.P("4 Months");
                Yield 30;
        }
        case 2 → {
                S.o.P("Only Month");
                Yield 28;
        }
    }
```

*** We cannot mix (→) Cases & traditional Cases

22/2/24

# Loops

① Initialisation { i = 17; }

② Loop Body { print; i = i+1; }

③ Termination { i <= 170; }

## While ⇒

```
int i = 17;
while (i<=170){
    S.o.P(i);
    i+=17;
}
```

```
int i=17;
while (i<170){
    S.o.P(i)
    i+=17;
}
```

Output :=>

17
34
51
68
.
.
153

9 iteration

10 condition check

```
int i=17
while (i<=17)
    S.o.P(i);
    i=i+17;
```

17
17
.
.

Infinite loop.

```
int i=17;
while (i<=17);
    S.o.P(i);
    i= i+17;
```

infinite loop.

```
int i=0;
while (i++<10){
    SOP(i);
}
```

1
2
3
4
5
6
7
8
9
10

Condition check = 11

```
int i=0;
while( i++ <10);
{
    SOP(i);
}
```

Output = 10

## ② do-while ⇒

```
do {
   ____
   ____
} while (condition)
```

In normal while, condition gets checked at entry point & in do-while condition is checked at exit point

If condition is false in the start while loop will not iterate once but in case of do-while we will have atleast one iteration.

```
n = 123;
sum = 10;
do{
    r = n % 10;
    n = n/10;
    sum = sum + r;
} while (n > 0);

SoP(sum); // Output = 6
```

$$a \% b = a - (a/b) * b$$

Numerator = -ive
Result = -ive

$-10 \% 3 = -10 - (-10/3) * 3$

$-10 + 9 = \boxed{-1}$

$-10 \% -3 = -10 - (10/-3) * -3$

$-10 + 9 = \boxed{-1}$

$10 \% -3 = 10 - (10/-3) * -3$

$\Rightarrow 1$

## ③ for

```
for ( initialisation ; condition ; increment / decrement)
{
   ____
}
```

{ All three statements are Optional. }

eg →

```
for (int i = 17 ; i <= 170 ; i = i + 17)
    S.o.P (i);
```

OR

```
int i = 17;
for ( ; i <= 170 ; i = i + 17)
    S.o.P(i);
```

OR

```
int i=17;
for( ; i<=170; ){
    SOP(i)
    i=i+17;
}
```

OR

```
int i=17;
for( ; ; ){
    SOP(i)
    if(i==170) break;
    i=i+17;
}
```

```
int i,j;
for(i=5,j=10; - ; -)
for (i=5, j=10; — ; i++,j++)
```

(4) For-Each => To iterate through each element of array or collection.

23/2/24

```
int ar[]={5,10,15,20};
int sum=0;
for (int a: ar)
    sum = sum+a;
SOP(sum);
```

(1) a=5
sum= 0+5

(2) a=10
sum= 10+5

(3) a=15
sum= 15+15

(4) a=20
sum= 30+20 => 50

=> Java A 15 Aug 2015

```
Class A {
    PSVM(String[] args){
        for (String s:args){
            SOP(s);
        }
    }
```

Output => 15
AUG
2015

```
int arr[][]= {{1,2,3},{4,5,6}};
for(int[] a: arr){
    for(int i: a){
        SOP(i);
    }
}
```

=> 1
2
3
4
5
6

# Jump Statement

→ Break

→ Continue

→ Return.

① **Break** → For Terminating Loop or Switch.

```
while (condition) {
    if (—) {
        break;
    }
}
```

---

```
while ( _){
    while (_) {
        while (_) {
            if (_) {
                break;
            }
        }
    }
}
```

We can use label {Identifier} with break to have more option ~~of~~ on name jump.

```
One: while (—) {
    Two : while (—) {
        Three : while (_) {
            if (_)
                break two;
        }
    }
}
```

---

```
Class A {
    PSVM( —) {
        int i=5;
        if (i>=5)
            SOP ("B");
        break;
        SOP ("A");
    }
}
```

Error

** {break outside loop or switch}

```
int i=5
int k= i* 6/3 - 4%5 + 3*2;
SOP (i);
```

Output = 5

② **Continue** ⇒ Skips the current iteration of the loop.

Q → ~~Disp~~ Display 1 to 100 & skip those which are divisible by 7?

⇒

```
for (int i=0; i<=100; i++){
    if(i % 7 == 0){
        continue;
    }
    SOP(i);
}
```

⇒

```
One : while (_) {
    Two : while (_) {
        Three : while {
            if (_) {
                continue One;
            }
        }
    }
}
```

**Return** → To get out of any ~~meth~~ method/function.

① ✓
```
int m1()
{
    return 0;
}
```

② ✗
```
int m1()
{
    return;
}
```

③ ✓
```
void m1()
{
    return;
}
```

④ ✗
```
void m1()
{
    return 0;
}
```