Flow Control

16/05/2011

Flow Control :-

→ Flow Control describes the order in which the statements will be executed at runtime.

Flow-Control

Selection Statements
① if-else
② Switch

Iterative Statements
① while
② do-while
③ for()
④ for-each loop

Transfer Statements
① break
② Continue
③ return
④ try
⑤ catch
⑥ final

(a) Selection Statements:-

(i) if-else :-

Syn :-
```
if(b)
{
    Action if b is true
}
else
{
    Action if b is false
}
```

1. The arguement to the if Statement should be boolean type. if we are providing any other type we will get Compiletime Error.

Ex:-

①
```
int x=0
if(x)
{
S.o.pln("Hello");
}
else
{
S.o.pln("Hi");
}
}
```
C.E:- incompatable types
found : int
required : boolean

②
```
int x=10
if(x=20)
{
S.o.pln("Hello");
}
else
{
S.o.pln("Hi");
}
}
```

③
```
int x=10;
if(x = =20)
{
S.o.pln("Hello");
}
else
{
S.o.pln("Hi");
}
}
```
o/p:- Hi

④
```
boolean b= false;
if(b =true)
{
S.o.pln("Hello");
}
else
{
S.o.pln("Hi");
}
}
```
o/p:- Hello

⑤
```
boolean b= false;
if (b = = true)
{
S.o.pln("Hello");
}
else
{
S.o.pln("Hi");
}
}
```
o/p:- Hi

(2) Curlly braces ({ . }) are optional and without Curllybraces we can take only one statement & which should not be declarative statement

Ex:-

```
if(true)                  | if(true)        | if(true)        | if(true);
   S.o.pln("Hello");      |   int x=10;     | {               |
                          |                 |   int x=10;     |    ✓
   ✓                      |   ✗             | }               |
                          |   C.E!          |   ✓             |
```

Switch Statement :-

→ If Several options are possible then it is never recommended to use if-else, we should go for switch statement.

Syn:-

```
Switch(x)
{
  Case1 :
          Action1;
  Case 2:
          Action 2;
     :
     :
  default :
          default Action;
}
```

↦ Curlly braces are mandatory.

→ both Case & default are optional inside a Switch

Ex:-

```
int x=10;
Switch(x)
{
}          ✓
```

→ with in the Switch, every Statement should be under some case or default. Independent Statements are not allowed.

Ex:-

```
int x=10;
Switch (x)
{
  S.o.p("Hello");
}
```

C.E:-
Case, default or '}' expected

→ until 1.4v the allowed datatypes for switch arguement are

byte
Short
int
char

→ But from 1.5v onwards in addition these the Corresponding wrapper classes (Byte, Short, Character, Integer) & enum types are allowed.

1.4 V		1.5V	1.7V
byte	⊕	Byte	
Short		Short	⊕ String
char		Character	
int		Integer	
		+	
		enum	

→ if we are passing any other type we will get Compiletime Error.

Ex:-

```
byte b=10;        | char ch='a';     | long l=10l;       | boolean b=true;
switch(b)         |  switch(ch)      | switch(l)         | switch(b)
{                 |  {               | {                 | {
}                 |  }               | }   X             | }
  ✓               |     ✓            |                   |
                  |                  | C.E:-             | C.E:-
                  |                  | possible loss of precision | incompatible types
                  |                  | found: long       | found: boolean
                  |                  | required: int     | required: int
```

→ every case label should be within the range of switch arguement type otherwise we will get Compiletime Error.

ex:-

```
byte b=10;                        byte b=10;
switch(b)                         switch(b+1)  ⟶  int type
   ↳ byte                         {
{                                  case 10:
 case 10:                               S.o.pln("10");
     S.o.pln("10");                case 100:
 case 100:                              S.o.pln("100");
     S.o.pln("100");               case 1000:
 case 1000:          -128 to            S.o.pln("1000");
     S.o.pln("1000");  127         }
}                                                  ✓
```

C.E:- possible Loss of precision
  found: byte int
  required: byte.

→ every Case label Should be a valid Compile-time Constant, if we are taking as Variable as Case label we will get Compiletime Error.

Ex:-
```
int x=10;
int y=20;
Switch(x)
{
 Case 10:
        S.o.pln("10");
 Case y:
        S.o.pln("20");   X
}
X
```
Suppose
```
final int y=20;
Case y:
    S.o.pln("20");
```

C.E!. Constant Expression required.

→ If we declare y as final then we wont to get any Compiletime Error

→ Expressions are allowed for both Switch arguement & Case label but Case label should be Constant Expression

ex!-
```
int x=10;
Switch (x+1)
{
 Case 10:
        S.o.pln("10");
 Case 10+20:
        S.o.pln("10+20");
}
```
✓

→ duplicate Case labels are not allowed.

ex:-
```
int x=10;
Switch(x)
{
  Case 97:
        S.o.pln("97");
  Case 98:
        S.o.pln("98");
  Case 99: S.o.pln("99");
  Case 'a':
        S.o.pln("a");     X
}
```
C.E:- duplicate Case label

Summary:-

Case label

1. It should be Compile time Constant.
2. Expressions also allowed but should be Constant Expression
3. Value should be with the range of Switch argument type
4. Duplicates are not allowed.

fall-through inside switch :-

→ within the switch statement if any case is matched from that case onwards all statements will be executed until break statement or end of the switch. this is called fall-through in inside switch.

Ex:-

```
Switch (x)
{
  Case 0:
        S.o.pln ("0");

  Case 1:
        S.o.pln ("1");
        break;
  Case 2 :
        S.o.pln ("2");

  default:
        S.o.pln ("def");
}
```

o/p:-

```
if x=0 :-      if x=1 :-      if x=2      if x=3
  0               1             2           def
  1                             def
```

→ fall-through inside switch is useful to define some common action for several cases.

Ex:-
```
Switch (x)
{
 Case 3:
 Case 4:
 Case 5:
         S.o.pln ("Summer");
         break;
 Case 6:
 Case 7:
 Case 8:
 Case 9:
         S.o.pln (" Rainy");
         break;
 Case10:
 Case11:
 Case12:
 Case 1:
 Case 2:
         S.o.pln ("winter");
         break;
```

<u>default case</u> :-

→ We Can use default Case to define default action.

→ This case will be executed iff no other Case is matched

→ we can take default case any where within the Switch but it is Convension to take as Last case.

Ex:-
```
Switch (x)
{
 default: S.o.pln ("def");
  Case0: S.o.pln ("0");
         break;
  Case1: S.o.pln ("1");
}  Case2: S.o.pln ("2");
```

$x=0$	$x=1$
0	1
	2

$x=2$	$x=3$
2	def
	0

(b) Iterative Statements :-

(1) while :-

→ if we don't know the no. of iterations in advance then the best suitable loop is while loop.

Ex:- ①
```
while (rs.next())        ↳ ResultSet
{
   --
}
```

②
```
while (itr.hasNext())    ↳ Iterator
{
   -
   -
}
```

③
```
while (e.hasMoreElements())   → enumeration
{
   --
   --
}
```

Syntax :-
```
while (b)   → boolean type
{
  Action
}
```

→ The arguement to the while loop should be boolean type. if we are using any other type we will get Compiletime Error.

Ex:-
```
while (1)
{
 S.o.pln("Hello");
}
```

C.E :- incompatible types
found : int
required : boolean

→ Curlly brases are optional and without curlly brases we can take only one statement which should not be declarative statement.

Ex:①

```
while (true)
    S.o.pln("Hello");
```
✓

```
while(true);
```
✓

```
while (true)
int x=10;
```
✗

```
while (true)
{
 int x=10;
}
```
✓

Ex②:-

①
```
while (true)
{
S.o.pln("Hello");
}
S.o.pln("Hi");
```
✗

C.E:- unreachable statement

②
```
while (false)
{
S.o.pln("Hello");
}
S.o.pln("Hi");
```
✗

C.E:- unreachable statement

③
```
int a=10, b=20;
while(a<b)
{
S.o.pln("Hello");
}
S.o.pln("Hi");
```
✓

o/p:- Hello
Hello
Hello
⋮

④
```
final int a=10, b=20;
while (a<b)  → True
{
S.o.pln("Hello");
}
S.o.pln("Hi");
```
✗

unreachable statement

⑤ do-while :-

→ if we want to execute loop body atleast once then we should go for do-while loop.

Syn:-

```
do
{
  Action
} while (b);
```

b → should be boolean type

; → mandatory

→ Curly brases are optional & without having Curly brases we can take only one Statement b/w do & while which should not be declarative Statement.

Ex:- ①

```
do
  S.o.pln("Hello");
while(true);
```
✓

②

```
do ;
while(true);
```
✓

; is a valid java statement

③

```
do
  int x=10;
while(true);
```
✗

④

```
do
{
  int x=10;
}while(true);
```
✓

⑤

```
do
while(true);
```
✗ C.E:-

→ Compulsary one statement declare (or) take ";"

⑥

```
do while(true)
  S.o.pln("Hello");
while(false);
```
✓

(or)

```
do
  while (true)
    S.o.pln("Hello");
while(false);
```

O/p:-
```
Hello
Hello
;
;
```

note:- ";" is a valid java statement

Eg:- ①

```
do            ✗
{
  S.o.pln("Hello");
}
while (true);
✗ S.o.pln("Hi");
```

C.E! unreachable Statement

②

```
do            ✓
{
  S.o.pln(" Hello");
}
while (false);
S.o.pln(" Hi");
```

o/p!- Hello
Hi

③

```
int a=10, b=20;
do
{
  S.o.pln("Hello");
}
while (a<b);
S.o.pln("Hi");
```

o/p!- Hello
Hello
⋮

④

```
int a=10, b=20;
do
{
  S.o.pln(" Hello");
}
while(a>b);
  S.o.pln ("Hi");
```

✓

o/p Hello
Hi

⑤

```
final int a=10, b=20;
do
{
  S.o.pln("Hello");
}
while(a<b);
✗ S.o.pln("Hi");
```

C.E!- unreachable Statement

⑥

```
final int a=10, b=20;
do
{
  S.o.pln("Hello");
}
while (a>b);
S.o.pln ("Hi");
```

o/p!- Hello
Hi ✓

for() :-

→ This is the most Commonly used loop

Syntax:-

```
                 only once
            ①  - - - - - - - - ->  ② ⑤ ⑧            ④ ⑦
for ( initialisation - Section ; Conditional-expression ; increment/decrement)
{
        Body ③ ⑥ ⑨
}
```

→ Curly braces are optional & without Curly braces we Can take only one statement which should not be declarative Statement.

(a) initialization-Section :-

→ This will be executed only once.

→ usually we are ~~perf~~ declaring and performing initialization for the variables in this Section.

→ Here we Can declare multiple variables of the Same type but different datatype variables we Can't declare.

Ex:- ① int i=0, j=0; ✓

② int i=0, byte b=0; ✗

③ int i=0, int j=0; ✗

→ in the initialization Section we Can take any Valid java Statement including S.o.p() also

Ex!- int i=0;

```
for( System.out.print("Hello U R Sleeping"); i<3 ; i++)
 {
   S.o.pln(" No Boss U only sleeping");
 }
```

O/P!-
```
Hello UR Sleeping
 No Boss U only Sleeping
 No Boss U only Sleeping
 No Boss U only Sleeping
```

Conditional Expression:-

→ Here, we can take any java Expression but the result should be boolean type.

→ It is optional and if we are not specifying then Compiler will always places "True".

Increment & decrement Section :-

→ We can take any valid java Statement including S.o.p() also.

Ex!- int i=0;

```
for( S.o.pln("Hello"); i<3 ; S.o.pln("Hi"))
 {
   ~~S.o.pln(~~ i++;
 }
```

O/P!-
```
Hello
Hi
Hi
```

→ All 3 parts of for loop are independent of each other.

→ All 3 parts of for loop are optional

ex!- for (; $\overset{T}{}$;) ; → Statement So, it is True.

⇒ represent infinite loop

Note:-

; is a Valid Java Statement

ex!-

```
for(int i=0; true; i++)           ✗
{
  S.o.pln("Hello");
}
S.o.pln("Hi");   ✗
C.E!- unreachable
```

```
for(int i=0; false; i++)          ✗
{
  S.o.pln("Hello");
}
S.o.pln("Hi");
C.E!- unreachable
```

```
for(int i=0; ; i++)               ✗
{
  S.o.pln("Hello");
}
S.o.pln("Hi");   ✗
C.E: unreachable
```

```
int a=10, b=20;
for(int i=0; a<b; i++)
{
  S.o.pln("Hello");
}
S.o.pln("Hi");
```

O/P!- Hello ✓
Hello
⋮

```
final int a=10; b=20;
for(int i=0; a<b; i++)
                 True
{
  S.o.pln("Hello");
}
S.o.pln("Hi");   ✗
```

O/P:- C.E!- unreachable Statement.

for-each() Loop :- (Enhanced for loop) :.

→ Introduced in 1.5v. this

→ this is the most convenient loop to retrieve the elements of Arrays & Collections

Ex:- ① print elements of single dimensional Array by using General & enchanced for loops

```
int[] a = {10, 20, 30, 40, 50};
```

for-loop

```
for(int i=0; i<a.length; i++)
{
  S.o.pln(a[i]);
}
```

```
10
20
30
40
50
```

for-each

```
for (int x: a)
{
  S.o.pln(x);
}
```

```
10
20
30
40
50
```

② print the elements of 2D-int Array by using General & for-each loop

```
int[][] a = {{10,20,30},{40,50}};
```

for-loop

```
for(int i=0; i<a.length; i++)
{
  for (int j=0; j<a[i].length; j++)
  {
    S.o.pln(a[i][j]);
  }
}
```

```
10
20
```

for-each

```
for (int[] x: a)
{
  for (int y: x)
  {
    S.o.pln(y);
```

```
10
20
30
40
50
```

→ Even though for-each loop is more Conveient to Use, but it has the following limitations.

(i) It is not a General purpose loop –

(ii) It is applicable only for Arrays & Collections

(iii) By using for-each loop we should retrive all values of Arrays & Collections and Can't be used to retrieved a particular Set of Values.

(C) Transfer statements :-

(1) break :-

→ We Can use break Statement in the following Cases

(1) within the Switch to stop fall through

(2) inside loops to break the loop execution based on Some Condition

(3) inside labeled blocks to break that block execution based on some Condition.

Ex:-

```
Switch (b)
{
   :
   break;
   :
}
```

```
for(int i=0 ; i<10; i++)
{
 if (i==5)
 {
   break;
 S.o.pln (i);
 :
```

```
Class Test
{
  p.s.v.m ( —— )
  {
    int i=10;
    l1:
    {
     S.o.pln(" Hello");
     if(i==10)
      break l1;
     S.o.pln(" Hi");
    }
  }} S.o.pln(" End");
```

Op:
Hello
End

→ if we are useing break Statement Any where else we will get Compiletime Error

Ex:-

```
Class Test
{
  p.s.v.m( ——)
  {
    int x=10;
    if(x==10)
       break;
    S.o.pln("Hello");
  }
}
```

C.E break outside Switch or loop.

Continue Statement:-

→ we Can use Continue Statement to Skip Current iteration and Continue for the next iteration inside loops

Ex:-

```
for(int i=0 ; i<=10 ; i++)
{
   if( i%2 == 0)
      Continue;
   S.o.pln(i);
}
```

```
1
3
5
7
9
```

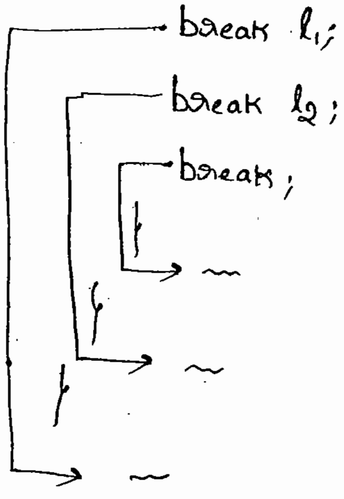
→ If we are using Continue outside of loops we will get Compiletime Error.

Ex:-
```
int x=10;
if(x==10)
    continue;  ——> ✗
S.o.pln("Hello");
```
C.E:- Continue outside of loop

labeled break & Continue Statements:-

→ In the case of nested loops to break and continue a particular loop we should go for labeled break & continue statements.

Ex:-
```
l1 :
for(----)
{
  l2 :
   for(----)
   {
     for(----)
     {
       break l1;
       break l2;
       break;
     }
     ~
   }
   ~
}
~
```



ex 2:-
```
l1:
for(int i=0; i<3; i++)
{
  for(int j=0; j<=3; j++)
  {
    if(i==j)
       break;
    S.o.pln(i+"-----"+j);
  }
}
```

break:-
```
1 ...... 0
2 ...... 0
2 ------ 1
```

break l1:-

No output

Continue :-
```
0----1      2---0
0---2       2---1
1----0
1---- 2
```

Continue l1 :-
```
1....0
2--0
2--1
```

do-while Vs Continue :- (Very hot Combination)

Ex:-
```
int x=0;
do
{
  x++;
  S.o.pln(x);
  if(++x<5)
   continue;
   x++;
  S.o.pln(x);
} while(++x<10);
```

```
1
4
6
8
10
```

x = ~~0~~ 1

Imp Note!:-

→ Compiler will check for unreachable statements only in the case of loops but not in 'if-else'.

Ex!. ①
```
if(true)
{
S.o.pln("Hello");
}
else
{
S.o.pln("Hi");
}
```
o/p!:- Hello

②
```
while(true)
{
  S.o.pln("Hello");
}
S.o.pln("Hi");
```
o/p!:- C.E:-
unreachable
Statement