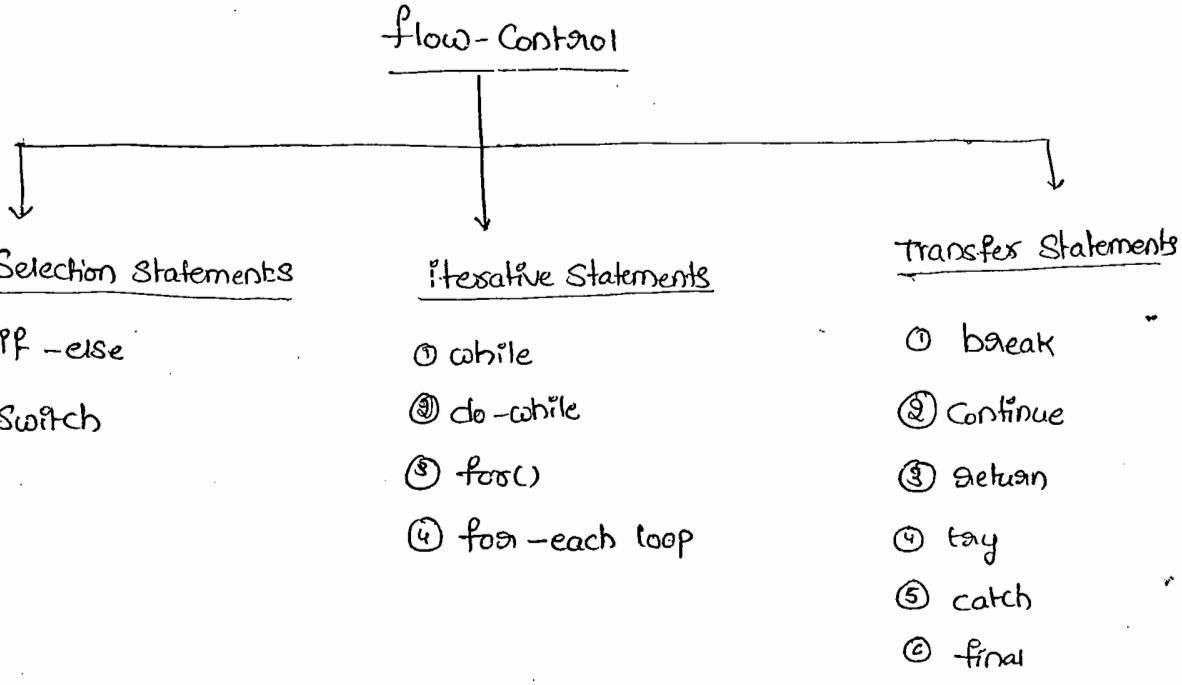
Flow Control

16/05/2011 52

Flow Control :-

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



(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 Curlybraces we can take only one statement which should not be declarative statement

Eg:-

```
if(true)
  S.o.p("Hello");
```
✓

```
if(true)
  int x=10;
```
✗ C.E:

```
if(true)
{
  int x=10;
}
```
✓

```
if(true);
```
✓

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;
  Case2 :
        Action2;
    :
  default :
        default Action;
}
```

↦ Curlly braces are mandatory.

→ both Case & default are optional inside a Switch

Eg:-

```
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 argument 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.

1.4V	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;
switch(b)
{
}
```
✓

```
char ch='a';
switch(ch)
{
}
```
✓

```
long l=10l;
switch(l)
{
}
```
✗

C.E:- possible loss of precision
found : long
required : int

```
boolean b=true;
switch(b)
{
}
```

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

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

Ex:-

```
byte b=10;
switch(b)        → byte
{
case 10:
    S.o.pln("10");
case 100:
    S.o.pln("100");
case 1000:
    S.o.pln("1000");
}
```
(-128 to 127)

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

```
byte b=10;
switch(b+1)      → int type
{
case 10:
    S.o.pln("10");
case 100:
    S.o.pln("100");
case 1000:
    S.o.pln("1000");
}
```
✓

→ Every Case label should be a valid Compile-time Constant, if we are taking as variable as Case label we will get Compiletime Error.

Eg:-

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

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 Argument & Case label but Case label should be Constant Expression

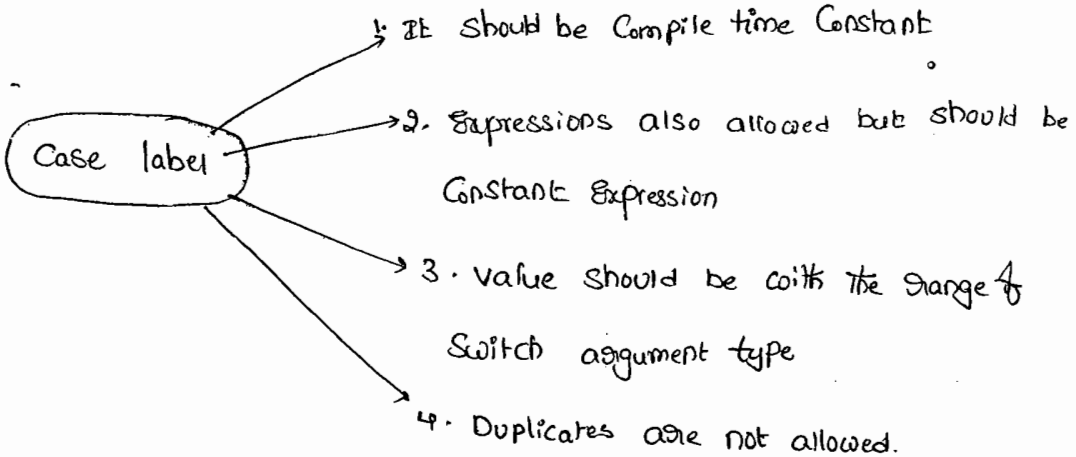
Ex:-

```
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");
}
```
C.E: duplicate Case label

Summary:-



fall-through inside switch :-

→ with in 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.

Eg:-
```
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;
  Case 10:
  Case 11:
  Case 12:
  Case 1:
  Case 2:
        S.o.pln ("winter");
        break;
```

default Case :-

→ 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.

Eg:-
```
Switch (x)
{
  default: S.o.pln ("def");
  Case 0:
        S.o.pln ("0");
        break;
  Case 1: S.o.pln ("1");
  Case 2: S.o.pln ("2");
```

x=0	x=1
0	1 2

x=2	x=3
2	def

(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())` — `rs` : Result Set

② `while (itr.hasNext())` — `itr` : Iterator

③ `while (e.hasMoreElements())` — `e` : enumeration

Syntax :-

```
while(b)        // b : boolean type
{
  Action
}
```

→ The argument 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 braces are optional and without Curlly braces we can take only one statement which should not be declarative statement.

Ex①

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

```
while(true);
```
✓

```
while (true)
    int x=10;
```
✗

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

Ex③ :-

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)
{
    S.o.pln("Hello");
}
S.o.pln("Hi");
```
(while(a<b) → True)

✗ 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 braces are optional & without having curly braces 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 ;` is a valid java statement

```
do ;
while(true);
```

✓

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

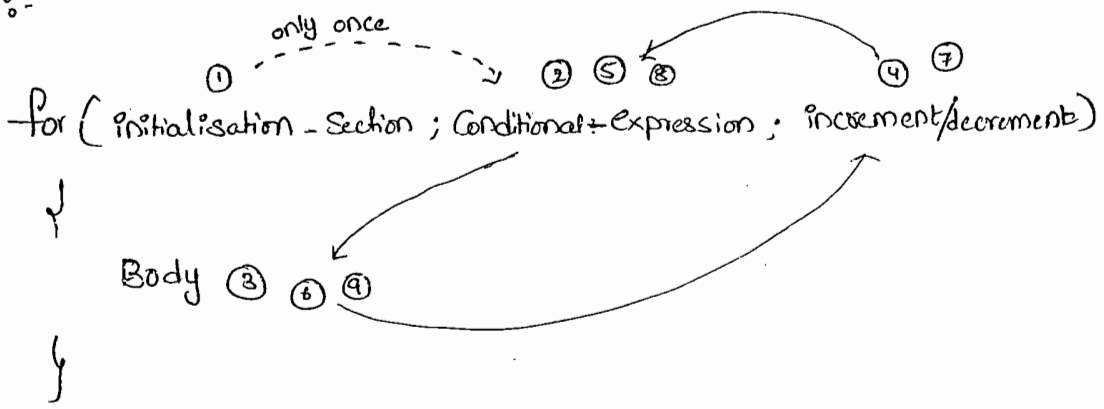
```
final int a=10, b=20;
do
{
S.o.pln("Hello");
}
while (a>b);
S.o.pln("Hi");
```

o/p:- Hello
Hi

for() :-

→ This is the most Commonly used loop

Syntax :-



→ 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 ~~prof~~ 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(;T;);` → Statement &, 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

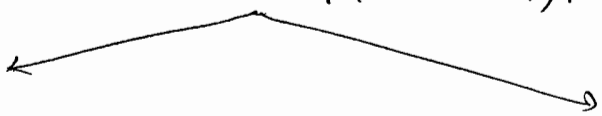
for-each

```
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 convient to use, but it has the following limitations.

(i) It is not a General purpose loop -

(ii) It is applicable only for Arrays & Collections

(ii) 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 (i+1);
    }
  S.o.pln (" End");
```

Op:
Hello
End

→ If we are using 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

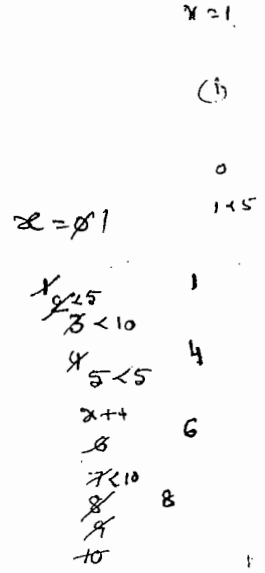
Continue:-
```
0---1
0---2
1---0
1---2
```
```
2---0
2---1
```

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

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