Flow Control

Topics

1. **Introduction**
2. **Selection Statement**
3. **Iterative Statement**
4. **Increment/ decrement section**
5. **For-each loop (enhanced for loop)**
6. **Iterable (I)**
7. **Transfer Statement**
8. **Labeled break & continue**

Introduction

Flow control describes the order in which the statement will be executed in run-time.

1. **Flow Control**
    a. **Selection Statement**
        i. **if-else**
        ii. **switch ()**
    b. **Iterative Statement**
        i. **while ()**
        ii. **do-while ()**
        iii. **for**
        iv. **for-each loop (1.5V)**
    c. **Transfer Statement**
        i. **break**
        ii. **continue**
        iii. **return**
        iv. **try-catch-finally**
        v. **assert (1.4V)**

Selection Statement

If-else

Syntax:

```
if (b) // b should be boolean type {
        //Action if b is true
}
else {
        //Action if b is false
}
```

The Argument of the if statement should be boolean type. By mistake if we are trying to provide any other type then we will get CE.

Example:

int x =0; if (x) { System.out.println (“Hello”); } else { System.out.println (“Hi”); } **//CE: incompatible types** **Found: int** **Required: boolean**	int x =10; if (x = 20) { System.out.println (“Hello”); } else { System.out.println (“Hi”); } **//CE: incompatible types** **Found: int** **Required: boolean**	int x =10; if (x == 20) { System.out.println (“Hello”); } else { System.out.println (“Hi”); } **//Hi**	boolean b = true; if (b = false) { System.out.println (“Hello”); } else { System.out.println (“Hi”); } **//Hi**	boolean b = false; if (b == false) { System.out.println (“Hello”); } else { System.out.println (“Hi”); } **//Hello**

if (true) System.out.println ("Hello");	if (true);	if (true) int x =10;	if (true) { int x =10; }

else part and curly braces are optional.

Without curly braces only 1 statement is allowed under if which should not be declarative statement.

Note:

1. Semicolon is a valid java statement which is also known as empty statement.
2. There is no dangling-else problem in Java. Every else is mapped to the nearest if statement.

switch

If several options are available then it is not recommended to use nested if-else because it reduces readability. To handle this requirement, we should go for switch statement.

Syntax:

```
switch (x) {
        Case 1: Action-1;
                break;
        Case 2: Action-2;
                break;
        Case n: Action-n;
                break;
        default: default-Action
}
```

Conclusions

1. The allowed argument types for the switch statement are byte, short, char, int until 1.4 version but from 1.5 version onwards corresponding wrapper classes and enum type also allowed. From 1.7 version onwards String type also allowed.

1.4V	1.5 V	1.7 V
byte short Char int	Byte Short Character Integer enum	String

2. Curly braces are mandatory, except switch everywhere curly braces are optional.
3. Both case and default are optional. i.e. an empty switch statement is valid java statement.

   Example:
   ```
   int x =10;
   switch (x) {
   }
   ```

4. Inside switch, every statement should be under some case or default. i.e. independent statements are not allowed inside switch otherwise we will get CE.
5. **Example:**
```
int x =10;
switch (x) {
        System.out.println ("Hello");
}
//CE: case, default or } expected
```
6. Every case label should be compile-time constant i.e. constant expression.
```
int x =10;
int y = 20;
switch (x) {
        case 10:
        case y: //CE: constant expression required
}
```
If we declare y as final then we won't get any CE.

7.
```
int x =10;
switch (x+1) {
        Case 10:
        Case 10+20+30:
}
```
Both switch argument and case –label can be expressions. But case label should be constant expression.

8. **Example:**

byte b =10; switch (b) { case 10: case 100: case 1000: //CE: Possible Loss of Precision, Required : int, Found: byte }	byte b =10; switch (b+1) { case 10: case 100: case 1000: }

Every case label should be in the range of switch argument type otherwise we will get CTE.

9. Duplicate case labels are not allowed otherwise we will get CTE.
```
int x =10;
switch (x) {
        Case 97:    break;
        Case 98:    break;
        Case 99:    break;
        Case 'a': //CE: duplicate case label
}
```

Summary:

Case label

- Should be constant expression.
- Value should be in the range of switch-argument type.
- Duplicate case-1 label are not allowed.

Fall-through inside switch

Within the switch if any-case is matched from that case onwards all statements will be executed until break or end of the switch. This is called fall-through inside switch.

The main advantage of fall-through inside switch is we can define common action for multiple cases (Code Reusability).

Code Reusability

```
Switch (x) {
        Case 1:
        Case 2:
        Case 3:
                System.out.println ("Q-4");
                Break;
        Case 4:
        Case 5:
        Case 6:
                System.out.println ("Q-1");
}
```

Example:

```
Switch (x) {
        Case 0: System.out.println (0);
        Case 1: System.out.println (1); break;
        Case 2: System.out.println (2);
        default: System.out.println ("default");
}
```

Output: x=0 → 0 1 x=1 → 1 x=2 → 2 def x=3 → def

Within the switch we can take default case almost once. Default case will be executed if and only if there is no case matched. Within the switch we can write default case anywhere but it is recommended to write as last case.

```
Switch (x) {
        Default: System.out.println ("default");
        Case 0: System.out.println (0);        break;
        Case 1: System.out.println (1);
        Case 2: System.out.println (2);
}
```

Output: x=0 → 0 x=1 → 1 2 x=2 → 2 x=3 → def 0

Iterative Statement

while ()

If we don't know no. of iteration in advance then we should go for while loop.

Example:

```
while (rs.next ()) {
}
```

```
while (e.hasMoreElement ()) {
}

while (itr.hasNext ()) {
}
```

Syntax:

```
while (b) { //B should be of boolean types
        Action
}
```

The argument should be of boolean type. If we are trying any other type then we will get CTE: incompatible types.

Example:

```
while (1) {
        System.out.println ("Hello");
}
//CE: incompatible types
        Found: int
        Required: boolean
```

Curly braces are optional and without curly braces we can take only one statement under while, which should not be declarative statement.

while (true) System.out.println ("Hello");	while (true) ;	while (true) int x =10;	while (true) { int x =10; }

while (true) { System.out.println ("Hello"); } System.out.println ("Hi"); //CE: unreachable statement	while (false) { System.out.println ("Hello"); } System.out.println ("Hi"); //CE: unreachable statement

int a =10, b=20; while (a<b) { System.out.println ("Hello"); } System.out.println ("Hi");	int a =10, b=20; while (a>b) { System.out.println ("Hello"); } System.out.println ("Hi");

final int a =10, b=20; while (a<b) { System.out.println ("Hello"); } System.out.println ("Hi"); //CE: unreachable statement	final int a =10, b=20; while (a>b) { System.out.println ("Hello"); } System.out.println ("Hi"); //CE: unreachable statement

Note:

1. Every final variable will be replaced by the value at compile-time only.
   Example: final int a =10;
   int b = 20;
   System.out.println (a); →System.out.println (10); //After Compilation
   System.out.println (b); → System.out.println (b); //After Compilation
2. If every argument is a final variable (compile-time constant) then that operation should performed at compile-time only.
   Example: final int a =10, b =20;
   int c = 30;
   System.out.println (a+b); → System.out.println (30); //After Compilation
   System.out.println (a+c); → System.out.println (10+c); //After Compilation
   System.out.println (a<b); → System.out.println (true); //After Compilation
   System.out.println (a<c); → System.out.println (10<c); //After Compilation

do-while ()

If we want to execute loop body at least once then we should go for do-while.

Syntax:
```
do {
    //body
} while (b); //mandatory and should be boolean type
```

Curley braces are optional and without curly braces we can take only one statement between do and while which should not be declarative.

do System.out.println (“Hello”); while (true);	do; while (true);	do int x =10; while (true);
do { int x =10; } while (true);	do while (true);	do while (true) System.out.println(“Hello”); while (false);

do { System.out.println (“Hello”); } while (true); System.out.println (“Hi”); **//CE: unreachable statement**	do { System.out.println (“Hello”); } while (false); System.out.println (“Hi”); **Output: Hello Hi**
int a =10, b=20; do { System.out.println (“Hello”); } while (a<b); System.out.println (“Hi”); **Output : Hello Hi**	int a =10, b=20; do { System.out.println (“Hello”); } while (a>b); System.out.println (“Hi”); **Output: Hello Hi**

final Int a =10, b=20; do { System.out.println (“Hello”); } while (a<b); System.out.println (“Hi”); **CE: unreachable statement**	final Int a =10, b=20; do { System.out.println (“Hello”); } while (a>b); System.out.println (“Hi”); **Output : Hello Hi**

For loop

If you know no. of iterations in advance then for loop is best choice.

Syntax: **For (initialization, conditional-check, increment/ decrement operator) {**
//Loop body
}

Curley braces are optional and without curly braces we can take only one statement under for loop, which should not be declarative statement.

for (int i=0;true;i++) System.out.println (“Hello”);	for (int i=0; i<10 ;i++)	for (int i=0; i<10; i++) int x = 10;

initialization

i. This part will be executed only in loop life cycle.
ii. Here we can declare and initialize local variables of for loop.
iii. Here we can declare any no. of variables but should be of the same type. By mistake if we are trying to declare different data types of variables then we will get CE.

```
int i=0, j=0;
int i=0, String s = “durga”;
int i=0, int j=0;
```

iv. In initialization section, we can take any valid Java statement including System.out.println statement.

```
int i =0;
for (System.out.println (“Hello”) ; i<3;i++) {
        System.out.println (“Hi”);
}
```

Output: Hello Hi Hi Hi

Conditional check

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

Here we can take any valid java expression but should be of the type boolean.

This part is optional and if we are not taking anything then compiler will always place true.

Increment/ decrement operator

In the Increment/decrement section we can take any valid Java statement including System.out.println.

```
int i=0;
for (System.out.println("Hello") ; i<3 ; System.out.println("Hi")) {
        i++;
}
```

Output: **Hello Hi Hi Hi**

All three parts of for loops are independent of each other and optional.

Infinite loops	Infinite loops
for (; ;) { System.out.println("Hello") }	for (; ;);

for(int i=0; true; i++) { System.out.println ("Hello"); } System.out.println ("Hi"); **//CE: unreachable statement**	for(int i=0; false; i++) { System.out.println ("Hello"); } System.out.println ("Hi"); **//CE: unreachable statement**	for(int i=0; ; i++) { System.out.println ("Hello"); } System.out.println ("Hi"); **//CE: unreachable statement**

int a =10, b=20; for(int i=0; a<b ; i++) { System.out.println ("Hello"); } System.out.println ("Hi"); **Output: Hello …**	int a =10, b=20; for(int i=0; a>b ; i++) { System.out.println ("Hello"); } System.out.println ("Hi"); **Output: Hi …**

final int a =10, b=20; for(int i=0; a<b ; i++) { System.out.println ("Hello"); } System.out.println ("Hi"); **//CE: unreachable statement**	final int a =10, b=20; for(int i=0; a>b ; i++) { System.out.println ("Hello"); } System.out.println ("Hi"); **//CE: unreachable statement**

For-each loop (enhanced for loop)

1. Introduces in 1.5 version
2. Especially designed loop to retrieve elements of Arrays and collections

Case 1: to print elements of 1-d array

int [] x = {10, 20, 30, 40, 50}

Normal for loop	For-each loop
for (int i=0;i< x .length ;i++) System.out.println(x[i])	for (int x1 : x) System.out.println(x1)

Case 2: to print elements of 2-d array

int [][] x = {{10,20,30},{40,50}}

Normal for loop	For-each loop
for (int i=0;i< x .length ;i++) { for (int j=0; j< x[i] .length ;j++) { System.out.println(x[i][j]) } }	for (int[] x1 : x) { for (int x2 : x1) { System.out.println(x2) } }

Case 3: to print elements of 3-d array

For-each loop

```
For (int [][] x1: x)
        For (int [] x2: x1)
                For (int x3: x1)
                        System.out.println(x3)
```

Limitations

1. For each loop is the best choice to retrieve elements of arrays and collections but its limitation it is applicable. Only for arrays and collections and it is not general purpose loop.

   ```
   for (int i=0; i< 10; i++ )
   System.out.println ("Hello"); //We can't write an equivalent for each loop directly.
   ```

2. By using normal for loop we can print array element either in original order or in reverse order. But by using for each loop . We can print array in original order but not in reverse order.

   ```
   int [] x = {10, 20, 30, 40, 50}
   for (int i= x .length-1; i>=0; i-- )
   System.out.println (x[i]); //We can't write an equivalent for each loop directly.
   ```

Iterable (I)

```
for (each item x: target) {
…
}
```

1. The target element in for each loop should be Iterable object. An object is said to be Iterable if and only if corresponding class implements Java.lang.Iterable (I).

2. Iterable interface introduced in 1.5 v and it contains only one method: iterator ().

Public Iterator iterator ()

So, All Arrays related classes and collection implemented classes already implement Iterable interface. Being a programmer we are not required to do anything just we should aware the point.

Difference between Iterator and Iterable

Iterator (I)	Iterable (I)
It is related to collections.	It is related to for-each loop.
We can use to retrieve the elements of collections one by one.	The target element in for each loop should be Iterable.
Java.util package	Java.lang package
Contains 3 method: hasNext () next () remove ()	Contain 1 method: iterator ()

Transfer Statement

Break

1. **Inside switch**
   We can use break statement to stop fall through.
2. **Inside loop**
   To break loop execution based on some condition.
3. **Inside labeled block**
   To break block execution based on some condition.

Continue

We can use continue statement inside loops to skip current iteration and continue for the next iteration.

```
for (int i =0; i<10; i++) {
        if (i%2 ==0)
                Continue;
        System.out.println (i);
}
```

Output: 1 3 5 7 9

We can use continue in loops. If we are using anywhere else then CE: continue outside of loop

```
int x =10;
if (x==10)
        Continue; //CE: outside of loop
System.out.println ("Hello");
```

Labeled break and continue

We can use labeled break and continue to break or continue a particular loop in nested loops.

```
l1:
for (……) {
        l2:
        for (……) {
                for (……) {
                        break l1;
                        break l2;
                        break;
                }
        }
}
```

```
l1:
for (int i=0; i<j; i++) {
        for (int j=0; j<3; j++) {
                if (i == j)
                        break;
                System.out.println (i+…….+j);
        }
}
```

break	break l1	continue	continue l1
1—0 2—0 3---0	No O/p	0-----1 0-----2 1-----0 1-----2 2-----1 2-----1	1-----0 2-----0 2-----1

Example:

```
int x=10;
do {
        x++;
        System.out.println (x);
        if (++x<5)
                continue;
        x++
        System.out.println (x);
} while (++x<10);
```

Output: **1 4 6 8 10**