***-: Flow control :-***

Flow control describes the order in which the statements will be executed at tun time.

There are 3 types of statements.

|  |  |  |
| --- | --- | --- |
| **Selection** | **Iterative** | **Transfer** |
| 1. If-else 2. switch() | 1. while() 2. do-while() 3. for( ) 4. for-each loop (1.5 V) | 1. break 2. continue 3. return 4. try-catch-finally 5. assert(1.4 V) |

Selection statements:

**If-else:**

Syntax:

If(b){ // b argument should be Boolean type only

Action if b is true;

}else{

Action if b is false;

}

* The argument to the if statement should be Boolean type. By mistake if we are trying to provide then we will get compile time error.

Ex: int x = 0; int x = 10;

if(x){ if(x = 20){

Sop(“hello”); Sop(“hello”);

} else{ } else{

sop(“hii”); Sop(“hii”);

} }

CE: incompatible type, found : int required : Boolean

int x = 10;

if(x == 20){

Sop(“hello”);

} else{

Sop(“hii”);

}

O/P: hii

* else part and curly braces( { , } ) are optional without curly braces only one statement is allowed under if which should not be declarative statement.

Ex: if( true ) if( true ) ; if( true ) if( true ){

Sop(“Hello”); ;;;;;;; int x =10; int x =10; }

// O/p : Hello // O/p : Compilation Error // O/p:

Note: semicolon ( **;** ) is a valid java statement which is also known as empty statement.

* There is no dangling else problem in java every else is mapped to the nearest if statement.

**switch statement:**

If several options are available then it is not recommended to use nested if else because it reduces readability. To handel this requirement 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;

}

* The allowed argument types for the switch statement are byte, short, char, int till 1.4 version. But from 1.5v onwards corresponding wrapper classes and enum type allowed. From 1.7v string type allowed.

|  |  |  |
| --- | --- | --- |
| 1.4v | 1.5v | 1.7v |
| byte  short  char  int | Byte  Short  Character  Integer  +  enum | String |

* Curley braces are mandatory. Except switch curly braces are optional.
* Both case and default are optional i.e. an empty switch statement is a valid java statement.

Ex: int x =10;

switch( x ){

}

* Inside a switch every statement should be under some case or default i.e. independent statements are not allowed inside switch. Other wise we will get compile time error.

Ex:

int x =10;

switch( x ){

System.out.println(“Hello”); // CE: case default } excepted (9 errors)

}

* Every case level should be compile constant(constant expression).

Ex:

Int x= 10, y= 20;

switch(x) {

case 10:

Sop(10);

break;

case y:

Sop(20); // CE: Constant expression required

break;

}

**Note:** If we declare y as final then we wont get any compile time error.

* Both switch argument and case level can be expression but case level should be constant expression.

Ex:

int x = 10;

switch( x+1 ){

case 10:

Sop(10); break;

case 10+20+30:

Sop(60);

}

// perfectly valid

* Every case level should be in the range of switch otherwise we will get compilation error.

Ex:

byte b = 10;

switch(b){

case 10:

sop(10);

break;

case 100:

sop(100);

break;

case 1000: //CE: P L P found : int ,required : byte

Sop(1000);

}

Ex:

byte b = 10;

switch(b +1){

case 10:

sop(10);

break;

case 100:

sop(100);

break;

case 1000:

Sop(1000);

}

* Duplicate case level are not allowed in a switch block otherwise we will get compile time error.

Ex: int x =10;

switch(x){

case 97:

sop(97);

break;

case 98:

sop(98);

break;

case ‘a’; // CE : Duplicate case level

sop(‘a’); }

**Fall through inside switch:**

With the switch if any case is matched from that case all statements will be executed until break or end of the switch this is called fall through inside a switch.

The main advantage of fall through inside a switch is we can define common action for multiple cases (code reusability).

Ex:

int x = 2;  
switch(x){  
 case 1:  
 case 2:  
 case 3:  
 System.*out*.println("First section");  
 break;  
 case 4:  
 case 5:  
 case 6:  
 System.*out*.println("Second section");  
 break;  
 default:  
 System.*out*.println("Third section");  
}

O/P: First section

* Within the switch we can use default statement at most once.
* Default case will be executed there is no case matched.
* Within the switch we can write default statement any where but it is recommended to write as last case.

Ex:

switch(x){  
 default:  
 System.*out*.println("default");  
 case 1:  
 System.*out*.println(1);  
 break;  
 case 2:  
 System.*out*.println(2);  
 case 3:  
 System.*out*.println(3);  
 break;  
}

**Iterative statement:**

**while loop:**

if we don’t know number of iteration in advance then we should go for while loop.

Ex:

while(rs.nest()) while(e.hasmoreElement()) while(its.hasNext())

{ { {

…… ……. ……

} } }

Syntax: while(b) // b compulsory should be Boolean type

{

Action;

}

* The argument should Boolean type. If we are trying to provide any other we will get compile time error.

Ex: while( 1 ){ // CE: incompatible type found : int required boolean

}

* Curly braces are optional and without curly braces we can take only one statement under while which shouldn’t be declarative statement.

Ex:

While(true) While(true); While(true) While(true){

Sop(“Hello”); int x = 10; int x = 10; }

// Valid //Valid // Invalid // Valid

Ex:

1. (2) (3) (4)

while(true){ while(false){ int a = 10,b=20; int a = 10,b=20;

Sop(“Hello”); Sop(“Hello”); while(a<b){ while(a>b){

}Sop(“Hi”); }Sop(“Hi”); sop(“Hello”); sop(“Hello”);

} sop(“Hi”); } sop(“Hi”);

(5) (6)

final int a = 10,b=20; int a = 10,b=20;

while(a>b){ while(a<b){

sop(“Hello”); sop(“Hello”);

} sop(“Hi”); } sop(“Hi”);

1: CE: unreachable statement Sop(“Hi”);

2: CE: unreachable statement { }

3: O/P: Hello Hello Hello……

4: O/P: Hi

5: CE: Unreachable statement at a<b always true

6: CE: Unreachable statement ‘{‘

**Note:** Every final variable will variable will be replaced by the value at compile time only.

Ex: final int a = 10;

int b = 20;

Sop(a);

Sop(b);

After compilation -> Sop(10); Sop(b);

**Note2**: If every argument is a final variable (compile time constant) then that operation should be performed at compile time only.

final int a = 10, b = 20;

int c =20;

Sop(a + b);

Sop(a + c);

Sop(a < b);

Sop( a < c);

After compilation :

Sop(30);

Sop(10 + c);

Sop(true);

Sop(10 < c);

**do-while():**

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

Syntax:

**do{**

**body;**

**}while(b);**

* Curley braces are optional and without curly braces we can take only one statement which shouldn’t be declarative statement.

Ex:

do do ; do do{ do

Sop(“Hello”); while(true); int x=10; int x =10; while(true);

While(true); // valid while(true); }while(true); // Invalid without

// valid // Invalid // Valid // Body

Ex-2

do while(true) do

Sop(“Hello”); <Equivalent code> while(true)

while(false); sop(“Hello”);

// valid while(false);

O/P: hello printed infinite number of time.

Ex-3

do{ do{ int a =10, b = 20; int a =10, b = 20;

Sop(“Hello”); Sop(“Hello”); do{ do{

} while(true); } while(false); Sop(“Hello”); Sop(“Hello”);

Sop(“Hi”); Sop(“Hi”); }while(a<b); }while(a>b);

// CE: unreachable //O/P: Hello Sop(“Hi”); Sop(“Hi”);

//statement Hi Hi O/p: Hello…Inf o/p: Hello

Hi

final int a =10, b = 20; final int a =10, b = 20;

do{ do{

Sop(“Hello”); Sop(“Hello”);

}while(a<b); }while(a>b);

Sop(“Hi”); Sop(“Hi”);

O/P: Unreachable statement O/p: Hello Hi

**for loop:**

for loop is the most commonly used loop in java. If we know number of iteration in advance then for loop is the best choice.

Syntax:

for(initialization-section(1) ; conditional-section(2)(5).. ; increment-decrement section(4)(7)..) {

loop body;(3)(6)..

}

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

Ex: for(int i = 0 ; i<10; i++)

int x =10;

// Compile time error

**Initialization Section:**

This part will be executed only once in the loop life circle.

Here we can declare and initialize local variable of for loop.

Here we can declare any number of variable of same type. By mistake if we are trying to declare different data type variables then we will get compile time error.

Ex:

int i =0, j=0; // Valid

int j = 0;String s = “Sonu”; // Invalid

int i = 0, int j = 0; // In-valid

In the initialization section we can take any valid java statement including Sop statement.

Ex: int i =0;

for(Sop(“Hello Boss”); i< 2; i++){

Sop(“Good Morning”);

}

O/P:-

Hello Boss

Good Morning

Good Morning

**Condition Section:**

Here we can take any java expression but should of the type Boolean. This part is optional and if are not taking any thing then compiler will always place true.

**Increment-decrement section:**

In the increment or decrement section we can take any valid java statement including Sop().

Int i =0;

for(Sop(“Hello”); i<3;Sop(“Hi”)){

i++;

}

O/p: Hello

Hi

Hi

..

All three parts of for loop are independent and optional.

for( ; ; ){

Sop(“Hello”); for( ; ; );

}

// The above two loop are infinite loops.

* In this loop unreachability problem is there also.

**for-each loop (Enhanced for loop):**

introduced in 1.5 version. It is the specially designed loop to retrieve the elements of Arrays and Collections.

Ex: To print elements of 1-d array.

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

for( int x1 : x){

Sop(x1);

}

Ex-2: To print elements of 2-d array.

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

for(int[ ] x1 : x){

for(int x2 : x1){

Sop(x2);

} }

Ex: To print element of 3-d array.

for(int[ ][ ] x1: x){

for(int[] x2 : x1){  
for(int x3 : x2){

Sop(x3);

} } }

**Note:1** for-each loop is the best choice to retrieve the elements of array and collections but its limitation is only applicable for arrays and collection and it is not a general purpose loop.

**Note:2** By using normal for loop we can print array elements either in original order or reverse order. But by using for-each we can print array elements only in original order not in reverse order.

**Iterable Interface:**

for( eachitem x : target){

…… // target should be a iterable(I) object

}

The target element in foreach loop should be iterable object .

An object is said to be iterable corresponding class implements java.lang.Iterable interface. It is introduced in 1.5 V. and it contains only one method : *public Iterator iterator();*

All array related classes and collection implement already implemented Iterable interface. Being a programmer we are not required to anything just a point to know.

Difference between Iterator Iterable:

|  |  |
| --- | --- |
| **Iterator(I)** | **Iterable(I)** |
| 1. It is related to collections. 2. We can use to retrieve the elements of the collection one by one. 3. It is defined in java.util package. 4. It contains 3 methods 5. hasNext() 6. next() 7. remove() | 1. It is related to for-each loop 2. The target element in for-each loop should be Iterable. 3. Present in java.lang package 4. It contains only one method.   iterator() |

**Transfer statement:**

**break:**

we can use break statement in the following places:

* Inside switch to stop fall through:

Ex: int x =10;

switch(x) {

case 0:

Sop(0);

case 1:

Sop(1);

break;

case 2:

Sop(2);

break;

default:

Sop(“def”);

break;

}  
O/P:

0

1

* Inside loops:

Inside loop to break loop execution based on some condition.

for(int i =0;i < 10; i++){  
if(I == 5)

break;

Sop(i);

}

O/P:

0

1

2

3

4

1. Inside labeled block:

Inside labeled block to break block execution based on some condition.

Ex: class Test{

p s v m(){

int x = 10;

l1{

Sop(“Begin”);

if(x == 10)

break l1;

Sop(“End”);

}

Sop(“Hello”);

} }

O/p:-

Begin

Hello

These are the only places where we can use break statement. If we are using any where else we will get compile time error saying : break outside switch or loop

Ex: int x =10;

if(x == 10)

break; // break outside switch or loop

System.out.println(“Hello”);

**continue:**

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

Ex:

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

if(i%2 == 0)

continue;

Sop(i);

}

O/P:

1 3 5 7 9

We can use continue only inside loop. Is we are using any where else CE saying : continue outside of loop.

Ex: int x =10;

if(x == 10)

continue; //CE: continue outside of loop.

System.out.print(“Hello”);

**Labeled break and continue:**

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

l1:

for(-----------){

l2:

for(---------------------){

for(------------){

break l1; // break the first for()

break l2; // break 2nd inner for()

break; // break innermost for()

}

}

}

**do-while vs continue(Dangerous combination):**

Ex: int x = 0;

do{

x++;

Sop(x);

if(++x < 5)

continue; // here control not goes to first statement after do if will goes to

x++; // condition under while after if it is true then it will go to first statement.

Sop(x);

}while(++x < 10 );

O/p: -

1 4 6 8 10