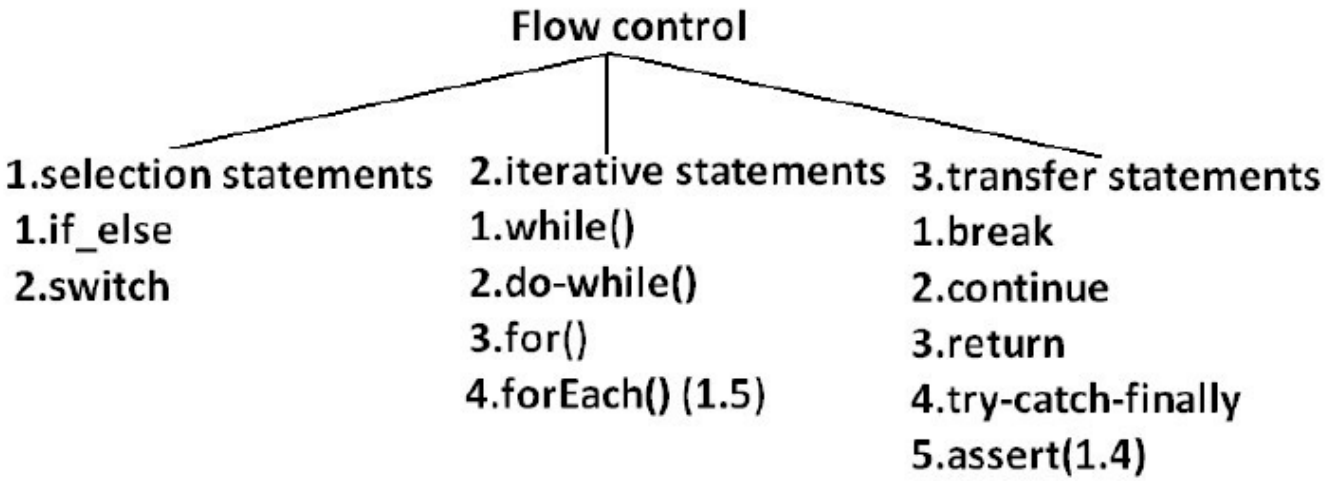
Flow Control

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



1.. Selection Statements

Selection Statements Selects the Blocks of Code to be executed based on the Condition Satisfied. There are two types of Selection Statements in Java:

- If-Else
- Switch

1.1 If-Else

Just Consider a Case where We login to Gmail Account, we are asked to provide Email ID and Password in order to Login. So, it is One of the best examples to Understand If-Else Selection Statement. **If** User provides Correct Email ID and Password then User Should be able to Login **else** User Should get Invalid Credentials Message on the Screen.

```
1  class IfElseDemo {
2      public static void main(String[] args)
3      {
4          if(condition) {
5              statement 1;
6              statement 2;
7              ...
8              statement N;
9          }
10     }
11 }
```

```
class IfElseDemo {
    public static void main(String[] args)
    {
        if(condition) {
            statement 1;
            statement 2;
        }
        else {
            statement 3;
            statement 4;
        }
    }
}
```

```
class IfElseDemo {
    public static void main(String[] args)
    {
        if(condition 1) {
            statement 1;
        }
        else if(condition 2) {
            statement 2;
        }
        else {
            statement 3;
            statement 4;
        }
    }
}
```

Above is General Syntax of if, if-else and if-else if -else. Inside if and else if we are required to provide a Condition which will evaluate to either of the boolean values (true or false).

Rules Related to If-Else

Rule - 1

The argument to the if statement should be Boolean if by mistake, we are providing any other type we will get "compile time error".

```
class IfElseDemo {
    public static void main(String[] args)
    {
        boolean x=true;
        if(x) {
            System.out.println("Hello");
        } else {
            System.out.println("Hi");
        }
    }
}
```

```
java -cp /tmp/DvRwUCdgbk IfElseDemo
Hello
```

```
class IfElseDemo {
    public static void main(String[] args)
    {
        int x=0;
        if(x) {
            System.out.println("Hello");
        } else {
            System.out.println("Hi");
        }
    }
}
```

```
ERROR!
javac /tmp/DvRwUCdgbk/IfElseDemo.java
/tmp/DvRwUCdgbk/IfElseDemo.java:5: error: incompatible types: int cannot be converted to boolean
        if(x) {
           ^
1 error
```

```
class IfElseDemo {
    public static void main(String[] args)
    {
        int x=10;
        if(x=20) {
            System.out.println("Hello");
        } else {
            System.out.println("Hi");
        }
    }
}
```

```
ERROR!
javac /tmp/DvRwUCdgbk/IfElseDemo.java
/tmp/DvRwUCdgbk/IfElseDemo.java:5: error: incompatible types: int cannot be converted to boolean
        if(x=20) {
^
1 error
```

```
class IfElseDemo {
    public static void main(String[] args)
    {
        int x=10;
        if(x==20) {
            System.out.println("Hello");
        } else {
            System.out.println("Hi");
        }
    }
}
```

```
java -cp /tmp/DvRwUCdgbk IfElseDemo
Hi
```

Rule – 2

Both else part and curly braces are optional. Without curly braces we can take only one statement under if, but it should not be declarative statement.

```
class IfElseDemo {
    public static void main(String[] args)
    {
        int x=10;
        if(x==10)
            System.out.println("Hello");
    }
}
```

```
java -cp /tmp/DvRwUCdgbk IfElseDemo
Hello
```

If without Curly braces and containing only Declarative Statement give us compile time error.

```
class HelloWorld {
    public static void main(String[] args) {
        if(true)
            int x=10;
    }
}
```

```
ERROR!
javac /tmp/DFrMTNq2fO/HelloWorld.java
/tmp/DFrMTNq2fO/HelloWorld.java:4: error: variable declaration not allowed here
            int x=10;
^
1 error
```

Rule - 3

Semicolon(;) is a valid java statement which is call empty statement and it won't produce any output.

```
class IfElseDemo {
    public static void main(String[] args)
    {
        if(true);
    }
}
```

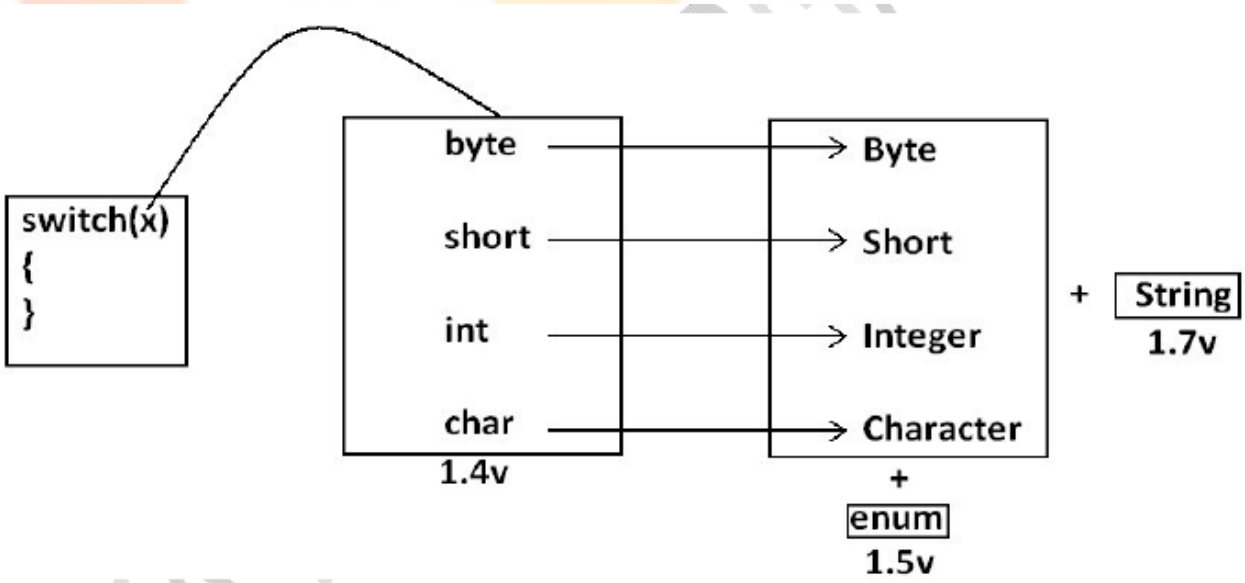
```
java -cp /tmp/DvRwUCdgbk IfElseDemo
```

1.2 Switch

If several options are available and out of it we have to Select one options then it is not recommended to use if-else but we should go for switch statement because it improves readability of the code.

```java
class SwitchSyntax {
    public static void main(String[] args)
    {
        switch(X)
        {
            case 1: {
                // Case 1 Block
                break;
            } case 2: {
                // Case 2 Block
                break;
            } case 3: {
                // Case 3 Block
                break;
            } default: {
                //default block
            }
        }
    }
}
```

Until 1.4 version the allow types for the switch argument are byte, short, char, int but from 1.5 version on wards the corresponding wrapper classes (Byte, Short, Character, Integer) and "enum" types also allowed.



Rules Related to Switch Case

Rule - 1

Curly braces are mandatory (except switch case in all remaining cases curly braces are optional).

Rule - 2

Both case and default are optional.

```java
class SwitchSyntax {
    public static void main(String[] args)
    {
        int x = 10;
        switch(x)
        {
        }
    }
}
```

```
java -cp /tmp/DvRwUCdgbk SwitchSyntax
```

Rule - 3

Every statement inside switch must be under some case (or) default. Independent statements are not allowed.

```
class SwitchSyntax {
    public static void main(String[] args)
    {
        int x = 10;
        switch(x)
        {
            System.out.println("Hello");
        }
    }
}
```

```
ERROR!
javac /tmp/DvRwUCdgbk/SwitchSyntax.java
/tmp/DvRwUCdgbk/SwitchSyntax.java:7: error: case, default, or '}' expected
            System.out.println("Hello");
            ^
/tmp/DvRwUCdgbk/SwitchSyntax.java:7: error: case, default, or '}' expected
            System.out.println("Hello");
                ^
/tmp/DvRwUCdgbk/SwitchSyntax.java:7: error: case, default, or '}' expected
            System.out.println("Hello");
                 ^
/tmp/DvRwUCdgbk/SwitchSyntax.java:7: error: case, default, or '}' expected
            System.out.println("Hello");
                  ^
/tmp/DvRwUCdgbk/SwitchSyntax.java:7: error: case, default, or '}' expected
            System.out.println("Hello");
                   ^
/tmp/DvRwUCdgbk/SwitchSyntax.java:7: error: case, default, or '}' expected
            System.out.println("Hello");
                    ^
/tmp/DvRwUCdgbk/SwitchSyntax.java:7: error: case, default, or '}' expected
            System.out.println("Hello");
                     ^
/tmp/DvRwUCdgbk/SwitchSyntax.java:7: error: case, default, or '}' expected
            System.out.println("Hello");
                      ^
/tmp/DvRwUCdgbk/SwitchSyntax.java:7: error: case, default, or '}' expected
            System.out.println("Hello");
                       ^
9 errors
```

Rule - 4

Every case label should be "compile time constant" otherwise we will get compile time error.

```
class SwitchSyntax {
    public static void main(String[] args)
    {
        int x = 10; int y =1;
        switch(x)
        {
            case y:
                System.out.println("Hi");
                break;
        }
    }
}
```

```
ERROR!
javac /tmp/DvRwUCdgbk/SwitchSyntax.java
/tmp/DvRwUCdgbk/SwitchSyntax.java:7: error: constant expression required
case y:
     ^
1 error
```

If we make y as final then we won't get any compile time error.

```
class SwitchSyntax {
    public static void main(String[] args)
    {
        int x = 10;
        final int y =1;
        switch(x)
        {
            case y:
                System.out.println("Hi");
                break;
        }
    }
}
```

```
java -cp /tmp/DvRwUCdgbk SwitchSyntax
```

Rule – 5

Both switch argument and case label can be expressions, but case label should be constant expression.

```
class SwitchSyntax {
    public static void main(String[] args)
    {
        int x = 10;
        final int y =1;
        switch(x + 1)
        {
            case 10+20:
                System.out.println("Hi");
                break;
        }
    }
}
```

```
java -cp /tmp/DvRwUCdgbk SwitchSyntax
```

Rule – 6

Every case label should be within the range of switch argument type.

```
class SwitchSyntax {
    public static void main(String[] args)
    {
        byte x = 10;
        switch(x)
        {
            case 10:
                System.out.println("Hi");
                break;
            case 100:
                System.out.println("Hi Hi");
                break;
            case 1000:
                System.out.println("Hi Hello");
                break;
        }
    }
}
```

```
ERROR!
javac /tmp/DvRwUCdgbk/SwitchSyntax.java
/tmp/DvRwUCdgbk/SwitchSyntax.java:13: error: incompatible types: possible lossy conversion from int to byte
            case 1000:
                 ^
1 error
```

```
class SwitchSyntax {
    public static void main(String[] args)
    {
        byte x = 10;
        switch(x)
        {
            case 10:
                System.out.println("Hi");
                break;
            case 100:
                System.out.println("Hi Hi");
                break;
        }
    }
}
```

```
java -cp /tmp/DvRwUCdgbk SwitchSyntax
Hi
```

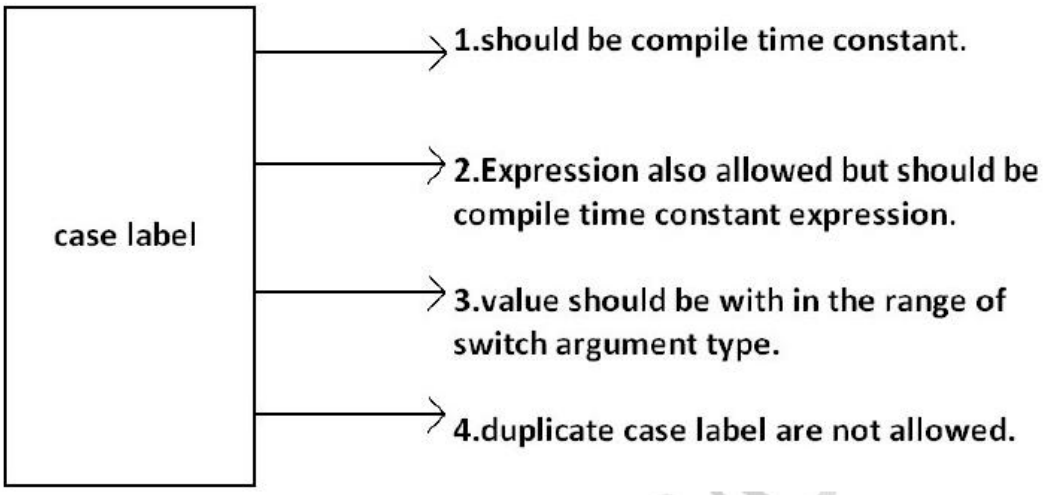
Rule – 7

Duplicate case labels are not allowed.

```
class SwitchSyntax {
    public static void main(String[] args)
    {
        int x = 10;
        switch(x)
        {
            case 97:
                System.out.println("Hi");
                break;
            case 'a':
                System.out.println("Hello");
                break;
        }
    }
}
```

```
ERROR!
javac /tmp/DvRwUCdgbk/SwitchSyntax.java
/tmp/DvRwUCdgbk/SwitchSyntax.java:10: error: duplicate case label
            case 'a':
                 ^
1 error
```

Rules Summary



Fall Through Inside the Switch

With in the switch statement if any case is matched from that case onwards all statements will be executed until end of the switch (or) break. This is call "**fall-through**" inside the switch. The main advantage of fall-through inside a switch is we can define common action for multiple cases.

```
class SwitchSyntax {
    public static void main(String[] args)
    {
        int x = 0;
        switch(x)
        {
            case 0:
                System.out.println("0");
            case 1:
                System.out.println("1");
            case 2:
                System.out.println("2");
                break;
            case 3:
                System.out.println("3");
                break;
            default:
               System.out.println("-1");
        }
    }
}
```

```
java -cp /tmp/DvRwUCdgbk SwitchSyntax
0
1
2
```

Default Case

- With in the switch we can take the default only once.
- If no other case matched then only default case will be executed
- With in the switch we can take the default anywhere, but it is convention to take default as last case.

```
class SwitchSyntax {
    public static void main(String[] args)
    {
        int x = 10;
        switch(x)
        {
            case 0:
                System.out.println("0");
            case 1:
                System.out.println("1");
            case 2:
                System.out.println("2");
                break;
            case 3:
                System.out.println("3");
                break;
            default:
               System.out.println("-1");
        }
    }
}
```

```
java -cp /tmp/DvRwUCdgbk SwitchSyntax
-1
```

2.. Iterative Statements

There are various Scenarios where the statements repeat or iterates itself and such statements are called **Iterative Statements**. There are Four types of Iterative Statements or Loops we have in Java:

- While Loop
- Do While Loop
- For Loop
- Enhanced For Loop

2.1 While Loop

If we don't know the number of Iterations in Advance then the best loop is While Loop. While Loop will gets executed till the given condition is true and once the Condition becomes false is Satisfied Control will Come out of the Loop. One best Example of While Loop is Suppose we are running any Job whose Final State is "Complete" and to reach that "Complete" State it can takes any number of Times so in that case we can Use While Loop with a condition for "Complete".

```
class HelloWorld
{
    public static void main(String[] args)
    {
        while(processStatus != "Complete")
        {
            Thread.sleep(2000);
        }
    }
}
```

Another example is File Reading or Collections Traversing, we don't know in advance how much contents are present in a File or the size of Collection, in that case too we can go for while loop.

Rules Related to While Loop

Rule – 1

The argument to the while statement should be Boolean type. If we are using any other type we will get compile time error.

```
class HelloWorld
{
    public static void main(String[] args)
    {
        while(1)
        {
            Thread.sleep(2000);
        }
    }
}
```

```
ERROR!
javac /tmp/gL9U1NPOY8/HelloWorld.java
/tmp/gL9U1NPOY8/HelloWorld.java:5: error: incompatible types: int cannot be converted to boolean
        while(1)
              ^
1 error
```

Rule – 2

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

```
class HelloWorld
{
    public static void main(String[] args)
    {
        int i=0;
        while(i < 3)
        {
            System.out.println(i);
            i++;
        }
    }
}
```

```
java -cp /tmp/gL9U1NPOY8 HelloWorld
0
1
2
3
```

Above program without Curly Braces will leads to Infinite Loop.

```
class HelloWorld
{
    public static void main(String[] args)
    {
        int i=0;
        while(i <= 3)
            System.out.println(i);
            i++;
    }
}
```

```
java -cp /tmp/gL9U1NPOY8 HelloWorld
0
00
0
0
0
0
0
0
0
0
00
0
0
0
0
0
0
0
00
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

If we have While Loop without any Curly Braces and containing Declarative Statement then we will get compile time error.

```
class HelloWorld {
    public static void main(String[] args) {
        while(true)
            int x=10;
    }
}
```

```
ERROR!
javac /tmp/DFrMTNq2fO/HelloWorld.java
/tmp/DFrMTNq2fO/HelloWorld.java:4: error: variable declaration not allowed here
            int x=10;
^
1 error
```

If above declarative statement is written within {} then there won't be any compile time error.

```
class HelloWorld {
    public static void main(String[] args) {
        while(true){
            int x = 10;
        }
    }
}
```

```
java -cp /tmp/DFrMTNq2fO HelloWorld
```

While loop with just a Semicolon.

```
class HelloWorld {
    public static void main(String[] args) {
        while(true);
    }
}
```

```
java -cp /tmp/DFrMTNq2fO HelloWorld
```

Rule – 3

If we have any While Loop which is running Infinite Times and after that loop we have any Statement that statement is never going to execute i.e., it is an Unreachable Statement in our Program and hence we will get Compile Time Error.

```
class HelloWorld {
    public static void main(String[] args) {
        while(true)
        {
            System.out.println("Hello");
        }
        System.out.println("Hi");
    }
}
```

```
ERROR!
javac /tmp/DFrMTNq2fO/HelloWorld.java
/tmp/DFrMTNq2fO/HelloWorld.java:7: error: unreachable statement
        System.out.println("Hi");
^
1 error
```

If we replace boolean true with any expression then there will be infinite loop in the output.

```
class HelloWorld {
    public static void main(String[] args) {
        int a=10;
        int b=20;
        while(a < b);
        {
            System.out.println("Hello");
        }
        System.out.println("Hi");
    }
}
```

Similarly, if we have any While loop which is never going to execute in the program then also we will get compile time error.

```
class HelloWorld {
    public static void main(String[] args) {
        while(false)
        {
            System.out.println("Hello");
        }
        System.out.println("Hi");
    }
}
```

```
ERROR!
javac /tmp/DFrMTNq2fO/HelloWorld.java
/tmp/DFrMTNq2fO/HelloWorld.java:4: error: unreachable statement
        {
        ^
1 error
```

But if instead of passing directly boolean false if we pass any expression then it won't be any compile time error.

```
class HelloWorld {
    public static void main(String[] args) {
        int a=10;
        int b=20;
        while(a > b);
        {
            System.out.println("Hello");
        }
        System.out.println("Hi");
    }
}
```

```
java -cp /tmp/DFrMTNq2fO HelloWorld
Hi
```

If we declare both and b variable as final then also we will get compile time error for unreachable statement.

```
class HelloWorld {
    public static void main(String[] args) {
        final int a=10, b=20;
        while(a<b){
            System.out.println("Hello, World!");
        }
        System.out.println("Hiiii");
    }
}
```

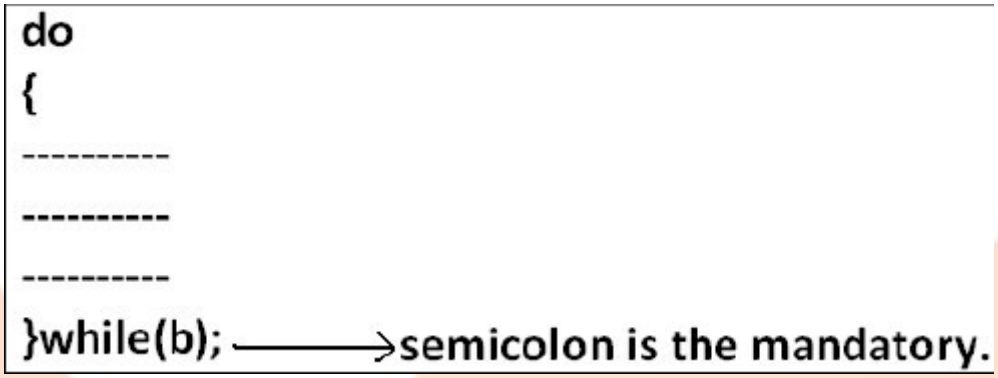
```
ERROR!
javac /tmp/gl4tThDAUX/HelloWorld.java
/tmp/gl4tThDAUX/HelloWorld.java:7: error: unreachable statement
        System.out.println("Hiiii");
        ^
1 error
```

Notes:

- Every final variable will be replaced with the corresponding value by compiler.
- If any operation involves only constants or final variables, then compiler is responsible to perform that operation.
- If any operation involves at least one variable compiler won't perform that operation. At runtime JVM is responsible to perform that operation.

2.2 Do While Loop

If we want to execute loop body at least once then we should go for do-while. The basic Syntax of Do While Loop Is:



Curly braces are Optional. Without curly braces we can take only one statement between do and while and it should not be declarative statement.

Below Program will lead to Infinite Loop.

```
class HelloWorld {
    public static void main(String[] args) {
        do{
            System.out.println("Hello, World!");
        }while(true);
    }
}
```

```
java -cp /tmp/gl4tThDAUX HelloWorld
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

```
class HelloWorld {
    public static void main(String[] args) {
        do
            System.out.println("Hello, World!");
        while(true);
    }
}
```

```
java -cp /tmp/gl4tThDAUX HelloWorld
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

```
class HelloWorld {
    public static void main(String[] args) {
        do;
        while(true);
    }
}
```

```
java -cp /tmp/gl4tThDAUX HelloWorld
```

```
class HelloWorld {
    public static void main(String[] args) {
        do
            int x = 1;
        while(true);
    }
}
```

```
ERROR!
javac /tmp/gl4tThDAUX/HelloWorld.java
/tmp/gl4tThDAUX/HelloWorld.java:4: error: variable declaration not allowed here
            int x = 1;
^
1 error
```

```
class HelloWorld {
    public static void main(String[] args) {
        do {
            int x = 1;
        }
        while(true);
    }
}
```

```
java -cp /tmp/gl4tThDAUX HelloWorld
```

```
class HelloWorld {
    public static void main(String[] args) {
        do
        while(true);
    }
}
```

```
ERROR!
javac /tmp/gl4tThDAUX/HelloWorld.java
/tmp/gl4tThDAUX/HelloWorld.java:4: error: while expected
while(true);
          ^
/tmp/gl4tThDAUX/HelloWorld.java:5: error: illegal start of expression
    }
    ^
2 errors
```

There can be Unreachable Code in Do While Loop too. Consider below Examples:

```
class HelloWorld {
    public static void main(String[] args) {
        do {
            System.out.println("Hello");
        }
        while(true);
        System.out.println("Hii");
    }
}
```

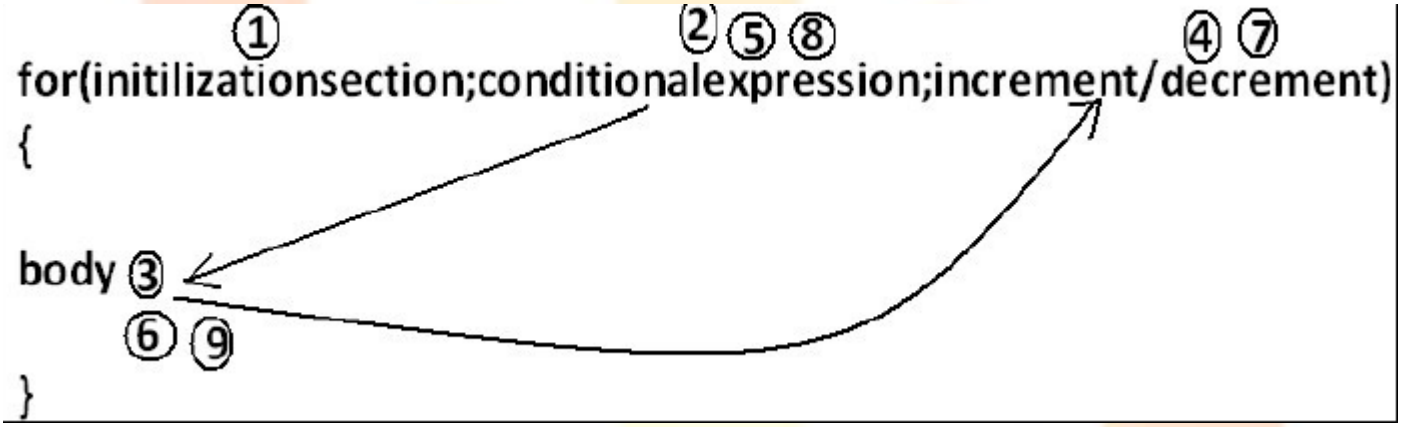
```
ERROR!
javac /tmp/gl4tThDAUX/HelloWorld.java
/tmp/gl4tThDAUX/HelloWorld.java:7: error: unreachable statement
        System.out.println("Hii");
^
1 error
```

```
class HelloWorld {
    public static void main(String[] args) {
        do {
            System.out.println("Hello");
        }
        while(false);
        System.out.println("Hii");
    }
}
```

```
java -cp /tmp/gl4tThDAUX HelloWorld
Hello
Hii
```

2.3 For Loop

This is the most commonly used loop and best suitable if we know the no of iterations in advance. The basic Syntax of For Loop is:



Here also Curly braces are optional and without curly braces we can take only one statement which should not be declarative statement.

```
class HelloWorld {
    public static void main(String[] args) {
        for(int i=0; i < 10; i++)
            System.out.println("Hii");
    }
}
```

```
Hii
Hii
Hii
Hii
Hii
Hii
Hii
Hii
Hii
```

```
class HelloWorld {
    public static void main(String[] args) {
        for(int i=0; i < 10; i++)
            int x=3;
    }
}
```

```
ERROR!
javac /tmp/gl4tThDAUX/HelloWorld.java
/tmp/gl4tThDAUX/HelloWorld.java:4: error: variable declaration not allowed here
            int x=3;
^
1 error
```

```
class HelloWorld {
    public static void main(String[] args) {
        for(int i=0; i < 10; i++);
    }
}
```

```
java -cp /tmp/gl4tThDAUX HelloWorld
```

Initialization Section

This section will be executed only once. Here usually we can declare loop variables and we will perform initialization. We can declare multiple variables but should be of the same type and we can't declare different type of variables.

Example:

```
Int i=0,j=0; valid
Int i=0,Boolean b=true; invalid
Int i=0,int j=0; invalid
```

In initialization section we can take any valid java statement including "System.out.println" also.

```
class HelloWorld {
    public static void main(String[] args) {
        int i=0;
        for(System.out.println("Hello"); i < 10; i++)
        {
            System.out.println("Hii");
        }
    }
}
```

```
java -cp /tmp/gl4tThDAUX HelloWorld
Hello
Hii
Hii
Hii
Hii
Hii
Hii
Hii
Hii
Hii
```

Conditional Check

We can take any java expression but should be of the type Boolean. Conditional expression is optional and if we are not taking any expression compiler will place true. Below Program will run infinite times.

```
class HelloWorld {
    public static void main(String[] args) {
        for(int i=0; ; i++)
        {
            System.out.println("Hi " + i);
        }
    }
}
```

```
java -cp /tmp/gl4tThDAUX HelloWorld
Hi 0
Hi 1
Hi 2
Hi 3
Hi 4
Hi 5
Hi 6
```

Increment/Decrement Section

Here we can take any java statement including "System.out.println" also.

```
class HelloWorld {
    public static void main(String[] args) {
        for(int i=0; i < 3; System.out.println("Changing"))
        {
            i++;
        }
    }
}
```

```
java -cp /tmp/gl4tThDAUX HelloWorld
Changing
Changing
Changing
```

Note:

All 3 parts of for loop are independent of each other and all optional. Below Program will produce Infinite Output.

```
class HelloWorld {
    public static void main(String[] args) {
        for(; ;)
        {
            System.out.println("hi");
        }
    }
}
```

```
java -cp /tmp/gl4tThDAUX HelloWorld
hi
hi
hi
hi
hi
hi
hi
```

Note:

Just Like While Loop and Do While Loop we can have Unreachable Statement in For Loop too in the same manner.

2.4 Enhanced for (for-each) Loop (1.5V)

Best suitable to retrieve the elements of arrays and collections.

Example Code to print the elements of One-dimensional array by normal for loop and enhanced for loop.

```
class OneDArray {
    public static void main(String[] args) {
        int[] x = {10,20,30,40};
        for(int i=0; i < x.length; i++)
        {
            System.out.println(x[i]);
        }
    }
}
```

```
java -cp /tmp/gl4tThDAUX OneDArray
10
20
30
40
```

```
class OneDArray {
    public static void main(String[] args) {
        int[] x = {10,20,30,40};
        for(int x1 : x)
        {
            System.out.println(x1);
        }
    }
}
```

```
java -cp /tmp/gl4tThDAUX OneDArray
10
20
30
40
```

Example Code to print the elements of Two-dimensional array by normal for loop and enhanced for loop.

```
class OneDArray {
    public static void main(String[] args) {
        int[][] x = {{10,20,30,40}, {50,60,70,80},{90,100,110,120}};
        for(int x1[] : x)
        {
            for(int x2 : x1)
            {
                System.out.println(x2);
            }
        }
    }
}
```

```
java -cp /tmp/gl4tThDAUX OneDArray
10
20
30
40
50
60
70
80
90
100
110
120
```

- We can't write equivalent for each loop.
- For each loop is the more convenient loop to retrieve the elements of arrays and collections, but its main limitation is it is not a general-purpose loop.
- By using normal for loop, we can print elements either from left to right or from right to left. But using for-each loop we can always print array elements only from left to right.

Iterator vs Iterable (1.5 V)

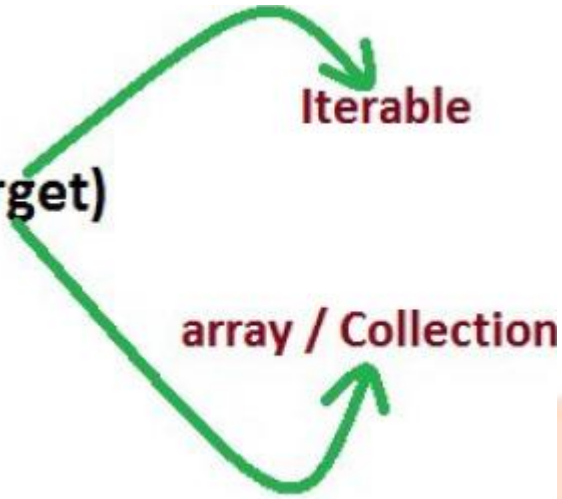
- The target element in for-each loop should be Iterable object.
- An object is set to be Iterable if corresponding class implements java.lang.Iterable interface.
- Iterable interface introduced in 1.5 version and it's contains only one method iterator().

Syntax : public Iterator iterator();

Every array class and Collection interface already implements Iterable interface.

Syntax :

```
for(each item : target)
{
 ---------
 ---------
}
```



Iterable	Iterator
It is related to forEach loop	It is related to Collection
The target element in forEach loop should be Iterable	We can use Iterator to get objects one by one from the collection
Iterator present in java.lang package	Iterator present in java.util package
contains only one method iterator()	contains 3 methods hasNext(), next(), remove()
Introduced in 1.5 version	Introduced in 1.2 version

3.. Transfer Statements

There are certain statements which is used to transfer the control from One block to another and such statements are called **Transfer** Statements.

Examples of Transfer Statements in Java are:

- Break
- Continue
- Return
- Assert
- Try-Catch-Finally

3.1 Break Statement

We can use break statement in the following cases.

- Inside switch to stop fall-through.

```
class BreakDemo {
    public static void main(String[] args) {
        int x = 0;
        switch(x)
        {
            case 0:
                System.out.println("Hello");
                break;
            case 1:
                System.out.println("Hi");
        }
    }
}
```

```
java -cp /tmp/gl4tThDAUX BreakDemo
Hello
```

- Inside loops to break the loop based on some condition.

```
class BreakDemo {
    public static void main(String[] args) {
        int i = 0;
        while(i <= 10)
        {
            if(i==5)
                break;
            System.out.println(i);
            i++;
        }
    }
}
```

```
java -cp /tmp/gl4tThDAUX BreakDemo
0
1
2
3
4
```

- Inside label blocks to break block execution based on some condition.

```
class BreakDemo {
    public static void main(String[] args) {
        int x = 10;
        l1 : {
            System.out.println("Begin");
            if(x == 10)
                break l1;
             System.out.println("End");
        }
         System.out.println("Hii");
    }
}
```

```
java -cp /tmp/gl4tThDAUX BreakDemo
Begin
Hii
```

Note: These are the only places where we can use break statement. If we are using anywhere else we will get compile time error.

```java
class BreakDemo {
    public static void main(String[] args) {
        int x = 10;
        System.out.println("Begin");
        if(x == 10)
            break;
        System.out.println("End");
    }
}
```

```
ERROR!
javac /tmp/gl4tThDAUX/BreakDemo.java
/tmp/gl4tThDAUX/BreakDemo.java:6: error: break outside switch or loop
            break;
            ^
1 error
```

3.2 Continue Statement

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

```java
class ContinueDemo {
    public static void main(String[] args) {
        int i = 0;
        while(i <= 10)
        {
            i++;
            if(i==5)
                continue;
            System.out.println(i);
        }
    }
}
```

```
java -cp /tmp/gl4tThDAUX ContinueDemo
1
2
3
4
6
7
8
9
10
11
```

We can use continue only inside loops if we are using anywhere else, we will get compile time error saying "**continue outside of loop**".

```java
class ContinueDemo {
    public static void main(String[] args) {
        int i = 0;
        if(i==5)
            continue;
        while(i <= 10)
        {
            i++;
            if(i==5)
                continue;
            System.out.println(i);
        }
    }
}
```

```
ERROR!
javac /tmp/gl4tThDAUX/ContinueDemo.java
/tmp/gl4tThDAUX/ContinueDemo.java:5: error: continue outside of loop
            continue;
            ^
1 error
```

Labeled break and continue statements

In the nested loops to break (or) continue a particular loop we should go for labeled break and continue statements.

Note:

Compiler won't check unreachability in the case of if-else it will check only in loops.