# Operators in Java

Operator in Java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

- ➢ Unary Operator,
- ➢ Arithmetic Operator,
- ➢ Shift Operator,
- ➢ Relational Operator,
- ➢ Bitwise Operator,
- ➢ Logical Operator,
- ➢ Ternary Operator and
- ➢ Assignment Operator.

# Java Operator Precedence

| Operator Type | Category | Precedence |
|---|---|---|
| Unary | postfix | *expr++ expr--* |
| | prefix | *++expr --expr +expr -expr ~* ! |
| Arithmetic | multiplicative | * / % |
| | additive | + - |
| Shift | shift | << >> >>> |
| Relational | comparison | < > <= >= instanceof |
| | equality | == != |
| Bitwise | bitwise AND | & |
| | bitwise exclusive OR | ^ |
| | bitwise inclusive OR | \| |
| Logical | logical AND | && |
| | logical OR | \|\| |
| Ternary | ternary | ? : |
| Assignment | assignment | = += -= *= /= %= &= ^= \|= <<= >>= >>>= |

## Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

➢ incrementing/decrementing a value by one
➢ negating an expression
➢ inverting the value of a boolean

**Java Unary Operator Example: ++ and --**

```
public class OperatorExample{

public static void main(String args[]){

int x=10;

System.out.println(x++);//10 (11)

System.out.println(++x);//12

System.out.println(x--);//12 (11)

System.out.println(--x);//10

}}
```

**Output:**

10

12

12

10

**Java Unary Operator Example: ~ and !**

```
public class OperatorExample{

public static void main(String args[]){

int a=10;

int b=-10;

boolean c=true;

boolean d=false;

System.out.println(~a);//-11 (minus of total positive
value which starts from 0)
```

```
System.out.println(~b);//9 (positive of total minus,
positive starts from 0)
```

```
System.out.println(!c);//false (opposite of boolean
value)
```

```
System.out.println(!d);//true
```

```
}}
```

**Output:**

-11

9

false

true


**Java Arithmetic Operators**

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.


**Java Arithmetic Operator Example**

```
public class OperatorExample{
```

```
public static void main(String args[]){
```

```
int a=10;
```

```
int b=5;
```

```
System.out.println(a+b);//15
```

```
System.out.println(a-b);//5
```

```
System.out.println(a*b);//50
```

```
System.out.println(a/b);//2
```

```
System.out.println(a%b);//0
```

```
}}
```

**Output:**

15

5

50

2

0

## Java Left Shift Operator

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

## Java Left Shift Operator Example

```
public class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240
}}
```

**Output:**

40

80

80

240

## Java Right Shift Operator

The Java right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

## Java Right Shift Operator Example

```java
public OperatorExample{

public static void main(String args[]){

System.out.println(10>>2);//10/2^2=10/4=2

System.out.println(20>>2);//20/2^2=20/4=5

System.out.println(20>>3);//20/2^3=20/8=2

}}
```

## Output:

2

5

2


## Java AND Operator Example: Logical && and Bitwise &

The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.


The bitwise & operator always checks both conditions whether first condition is true or false.

```java
public class OperatorExample{

public static void main(String args[]){

int a=10;

int b=5;

int c=20;

System.out.println(a<b&&a<c);//false && true = false

System.out.println(a<b&a<c);//false & true = false

}}
```

**Output:**

false

false

### Java AND Operator Example: Logical && vs Bitwise &

```
public class OperatorExample{

public static void main(String args[]){

int a=10;

int b=5;

int c=20;

System.out.println(a<b&&a++<c);//false && true =
false

System.out.println(a);//10 because second condition
is not checked

System.out.println(a<b&a++<c);//false && true = false

System.out.println(a);//11 because second condition
is checked

}}
```

**Output:**

false

10

false

11

### Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

```
public class OperatorExample{

public static void main(String args[]){

int a=10;

int b=5;

int c=20;

System.out.println(a>b||a<c);//true || true = true

System.out.println(a>b|a<c);//true | true = true

//|| vs |

System.out.println(a>b||a++<c);//true || true = true

System.out.println(a);//10 because second condition
is not checked

System.out.println(a>b|a++<c);//true | true = true

System.out.println(a);//11 because second condition
is checked

}}
```

**Output:**

true

true

true

10

true

11

**Java Ternary Operator**

Java Ternary operator is used as one line replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

**Java Ternary Operator Example**

```
public class OperatorExample{

public static void main(String args[]){

int a=2;

int b=5;

int min=(a<b)?a:b;

System.out.println(min);

}}
```

**Output:**

2

**Java Assignment Operator**

Java assignment operator is one of the most common operators. It is used to assign the value on its right to the operand on its left.

**Java Assignment Operator Example**

```
public class OperatorExample{

public static void main(String args[]){

int a=10;

int b=20;

a+=4;//a=a+4 (a=10+4)

b-=4;//b=b-4 (b=20-4)
```

```
System.out.println(a);

System.out.println(b);

}}
```

**Output:**

14

16

# Java Keywords

Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name.

**List of Java Keywords**

A list of Java keywords or reserved words are given below:

1. **abstract:** Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.
2. **boolean:** Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
3. **break:** Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.
4. **byte:** Java byte keyword is used to declare a variable that can hold 8-bit data values.
5. **case:** Java case keyword is used with the switch statements to mark blocks of text.
6. **catch:** Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
7. **char:** Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
8. **class:** Java class keyword is used to declare a class.
9. **continue:** Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
10. **default:** Java default keyword is used to specify the default block of code in a switch statement.
11. **do:** Java do keyword is used in the control statement to declare a loop. It can iterate a part of the program several times.
12. **double:** Java double keyword is used to declare a variable that can hold 64-bit floating-point number.

13. **else:** Java else keyword is used to indicate the alternative branches in an if statement.

14. **enum:** Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.

15. **extends:** Java extends keyword is used to indicate that a class is derived from another class or interface.

16. **final:** Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.

17. **finally:** Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.

18. **float:** Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.

19. **for:** Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true. If the number of iteration is fixed, it is recommended to use for loop.

20. **if:** Java if keyword tests the condition. It executes the if block if the condition is true.

21. **implements:** Java implements keyword is used to implement an interface.

22. **import:** Java import keyword makes classes and interfaces available and accessible to the current source code.

23. **instanceof:** Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.

24. **int:** Java int keyword is used to declare a variable that can hold a 32-bit signed integer.

25. **interface:** Java interface keyword is used to declare an interface. It can have only abstract methods.

26. **long:** Java long keyword is used to declare a variable that can hold a 64-bit integer.

27. **native:** Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).

28. **new:** Java new keyword is used to create new objects.

29. **null:** Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.

30. **package:** Java package keyword is used to declare a Java package that includes the classes.

31. **private:** Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.

32. **protected:** Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.

33. **public:** Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.

34. **return:** Java return keyword is used to return from a method when its execution is complete.

35. **short:** Java short keyword is used to declare a variable that can hold a 16-bit integer.

36. **static:** Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.

37. **strictfp:** Java strictfp is used to restrict the floating-point calculations to ensure portability.

38. **super:** Java super keyword is a reference variable that is used to refer to parent class objects. It can be used to invoke the immediate parent class method.

39. **switch:** The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.

40. **synchronized:** Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.

41. **this:** Java this keyword can be used to refer the current object in a method or constructor.

42. **throw:** The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.

43. **throws:** The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.

44. **transient:** Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.

45. **try:** Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.

46. **void:** Java void keyword is used to specify that a method does not have a return value.

47. **volatile:** Java volatile keyword is used to indicate that a variable may change asynchronously.

48. **while:** Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use the while loop.

# Java Control Statements | Control Flow in Java

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

1. Decision Making statements
   ➢ if statements
   ➢ switch statement
2. Loop statements
   ➢ do while loop
   ➢ while loop
   ➢ for loop
   ➢ for-each loop
3. Jump statements
   ➢ break statement
   ➢ continue statement

**Decision-Making statements:**

As the name suggests, decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided. There are two types of decision-making statements in Java, i.e., If statement and switch statement.

**1) If Statement:**

In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.

- ➤ Simple if statement
- ➤ if-else statement
- ➤ if-else-if ladder
- ➤ Nested if-statement

Let's understand the if-statements one by one.

## 1) Simple if statement:

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax of if statement is given below.

```
if(condition) {
statement 1; //executes when condition is true
}
```

Consider the following example in which we have used the if statement in the java code.

```
Student.java
public class Student {
public static void main(String[] args) {
int x = 10;
int y = 12;
if(x+y > 20) {
System.out.println("x + y is greater than 20");
}
}
}
```

**Output:**

x + y is greater than 20

*2) if-else statement*

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

*Syntax:*

```
if(condition) {

statement 1; //executes when condition is true

}

else{

statement 2; //executes when condition is false

}
```

Consider the following example.

```
public class Student {

public static void main(String[] args) {

int x = 10;

int y = 12;

if(x+y < 10) {

System.out.println("x + y is less than      10");

}   else {

System.out.println("x + y is greater than 20");

}

}

}
```

**Output:**

x + y is greater than 20

*3) if-else-if ladder:*

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.

Syntax of if-else-if statement is given below.

```
if(condition 1) {

statement 1; //executes when condition 1 is true

}
else if(condition 2) {

statement 2; //executes when condition 2 is true

}
else {

statement 2; //executes when all the conditions are
false

}
```

Consider the following example.

```
public class Student {

public static void main(String[] args) {

String city = "Delhi";

if(city == "Meerut") {

System.out.println("city is meerut");

}else if (city == "Noida") {

System.out.println("city is noida");

}else if(city == "Agra") {

System.out.println("city is agra");

}else {

System.out.println(city);

}

}

}
```

**Output:**

Delhi


*4. Nested if-statement*

In nested if-statements, the if statement can contain a if or if-else statement inside another if or else-if statement.


Syntax of Nested if-statement is given below.


```
if(condition 1) {

statement 1; //executes when condition 1 is true

if(condition 2) {

statement 2; //executes when condition 2 is true
```

```
}
else{
statement 2; //executes when condition 2 is false
}
}
```

Consider the following example.

```
public class Student {
public static void main(String[] args) {
String address = "Delhi, India";

if(address.endsWith("India")) {
if(address.contains("Meerut")) {
System.out.println("Your city is Meerut");
}else if(address.contains("Noida")) {
System.out.println("Your city is Noida");
}else {
System.out.println(address.split(",")[0]);
}
}else {
System.out.println("You are not living in India");
}
}
}
```

**Output:**

Delhi

*Switch Statement:*

In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

Points to be noted about switch statement:

- ➢ The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java
- ➢ Cases cannot be duplicate
- ➢ Default statement is executed when any of the case doesn't match the value of expression. It is optional.
- ➢ Break statement terminates the switch block when the condition is satisfied.
- ➢ It is optional, if not used, next case is executed.
- ➢ While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

The syntax to use the switch statement is given below.

```
switch (expression){
    case value1:
     statement1;
     break;
     .
     .
     .
    case valueN:
     statementN;
     break;
    default:
```

```
    default statement;
}
```

Consider the following example to understand the flow of the switch statement.

```java
public class Student implements Cloneable {
public static void main(String[] args) {
int num = 2;
switch (num){
case 0:
System.out.println("number is 0");
break;
case 1:
System.out.println("number is 1");
break;
default:
System.out.println(num);
}
}
}
```

Output:

2

While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value. The switch permits only int, string, and Enum type variables to be used.

**Loop Statements**

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.

In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.
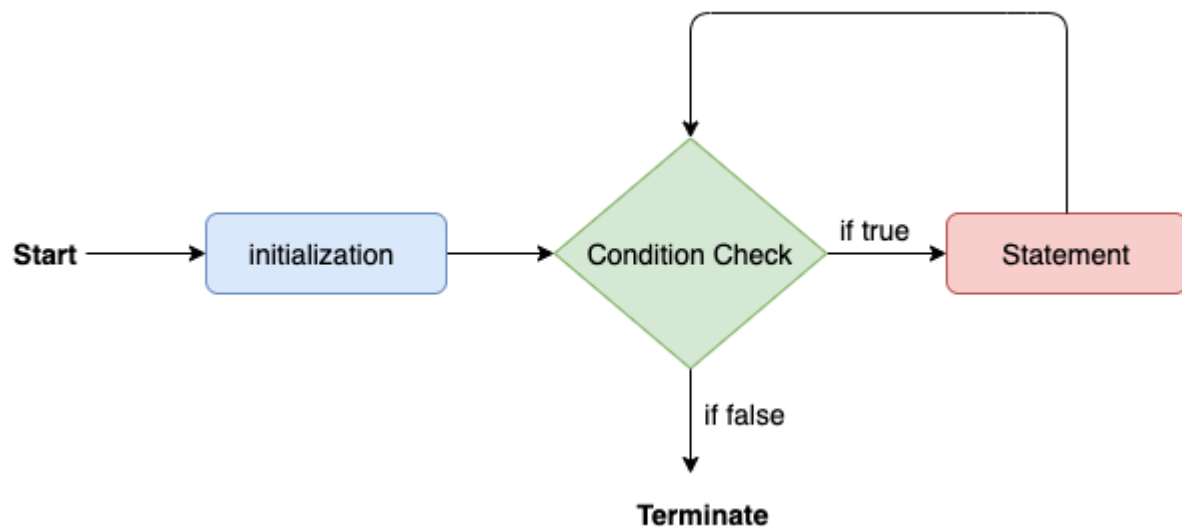
- ➢ for loop
- ➢ while loop
- ➢ do-while loop

Let's understand the loop statements one by one.

**Java for loop**

In Java, for loop is similar to C and C++. It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.

```
for(initialization, condition, increment/decrement) {

//block of statements

}
```

The flow chart for the for-loop is given below.

Consider the following example to understand the proper functioning of the for loop in java.

```java
public class Calculattion {
public static void main(String[] args) {
// TODO Auto-generated method stub
int sum = 0;
for(int j = 1; j<=10; j++) {
sum = sum + j;
}
System.out.println("The sum of first 10 natural numbers is " + sum);
}
}
```

**Output:**

The sum of first 10 natural numbers is 55

Java for-each loop

Java provides an enhanced for loop to traverse the data structures like array or collection. In the for-each loop, we don't need to update the loop variable. The syntax to use the for-each loop in java is given below.

```
for(data_type var : array_name/collection_name){
//statements
}
```

Consider the following example to understand the functioning of the for-each loop in Java.

```
public class Calculation {
public static void main(String[] args) {
// TODO Auto-generated method stub
String[]                        names                        =
{"Java","C","C++","Python","JavaScript"};
System.out.println("Printing the content of the array
names:\n");
for(String name:names) {
System.out.println(name);
}
}
}
```

**Output:**

Printing the content of the array names:

Java

C

C++

Python

JavaScript

## Java while loop

The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.

It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

The syntax of the while loop is given below.

```
while(condition){
//looping statements
}
```

The flow chart for the while loop is given in the following image.
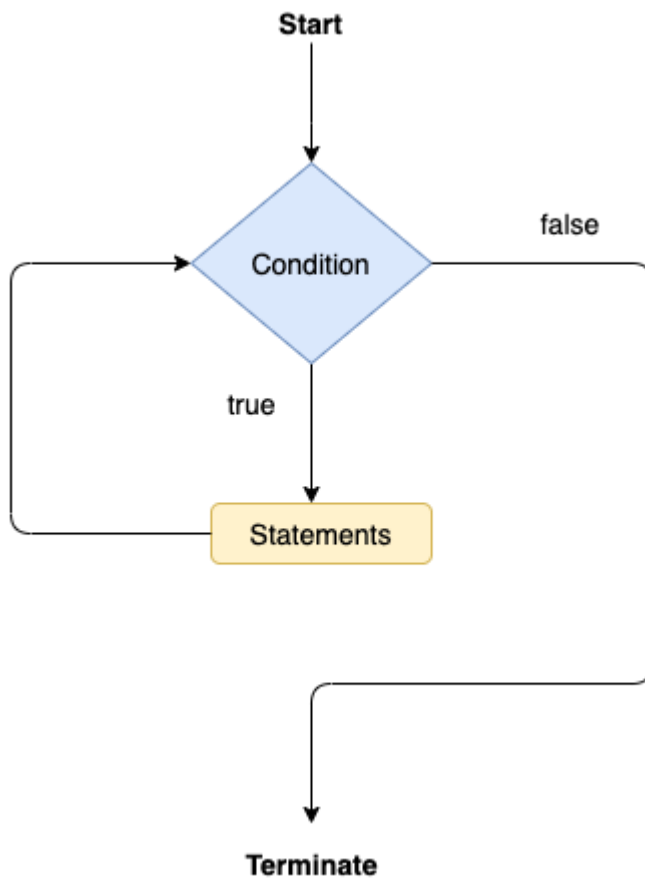
Consider the following example.

```
public class Calculation {

public static void main(String[] args) {

// TODO Auto-generated method stub

int i = 0;

System.out.println("Printing the list of first 10 even
numbers \n");

while(i<=10) {

System.out.println(i);

i = i + 2;

}

}

}
```

**Output:**

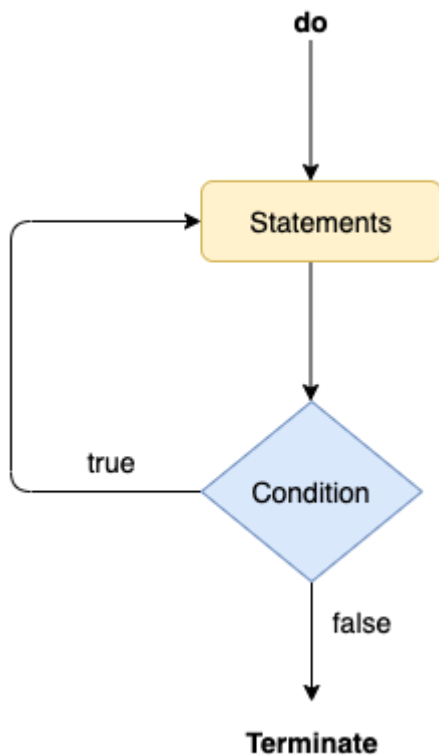Printing the list of first 10 even numbers

0

2

4

6

8

10

**Java do-while loop**

The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

```
do
{
//statements
} while (condition);
```

The flow chart of the do-while loop is given in the following image.



Consider the following example to understand the functioning of the do-while loop in Java.

```
public class Calculation {
public static void main(String[] args) {
// TODO Auto-generated method stub
int i = 0;
System.out.println("Printing the list of first 10 even
numbers \n");
do {
System.out.println(i);
i = i + 2;
}while(i<=10);
```

```
    }
}
```

**Output:**

Printing the list of first 10 even numbers

0

2

4

6

8

10

## Jump Statements

Jump statements are used to transfer the control of the program to the specific statements. In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e., break and continue.

## Java break statement

As the name suggests, the break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

The break statement example with for loop

Consider the following example in which we have used the break statement with the for loop.

BreakExample.java

```java
public class BreakExample {

public static void main(String[] args) {
// TODO Auto-generated method stub
for(int i = 0; i<= 10; i++) {
System.out.println(i);
if(i==6) {
break;
}
}
}
}
```

Output:

0

1

2

3

4

5

6

**Java continue statement**

Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

Consider the following example to understand the functioning of the continue statement in Java.

public class ContinueExample {

```
public static void main(String[] args) {
// TODO Auto-generated method stub


for(int i = 0; i<= 2; i++) {


for (int j = i; j<=5; j++) {


if(j == 4) {
continue;
}
System.out.println(j);
}
}
}


}
```

**Output:**

0
1
2
3

5

1

2

3

5

2

3

5