Module – II

Operators and Statements


Java
JavaScript
Python
C++
C
Objective-C
Perl
Haskell
Ruby
PHP
Pascal
Assembly
Scratch
Scheme
R
Q
PostScript
Logo
Prolog
ColdFusion
Forth
PL/SQL
Ada
Lua
Yacc
TCL
PL/I
Delphi
MATLAB
ActionScript
Fortran
Go
Lisp
Swift
Mathematica
Groovy
Tcl
Haskell
SAS
cT
Max/MSP
Scala
Dart
COBOL
Erlang
C#
D
Scala
F#
ML
OpenEdge ABL
ABAP
Visual Basic
Transact-SQL
Clarion
FoxPro

What you will Learn

- Operators
- Introduction to Control Structure
- Decision Making statements
- Loops

Operators

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

There are many types of operators in Java

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Arithmetic Operator

ICTACADEMY

Operator	Name	Description	Example
+	**Addition**	Adds together two values	x **+** y
-	**Subtraction**	Subtracts one value from another	x **-** y
*	**Multiplication**	Multiplies two values	x ***** y
/	**Division**	Divides one value by another	x **/** y
%	**Modulus**	Returns the division remainder	x **%** y
++	**Increment**	Increases the value of a variable by 1	**++**x or **++**x
--	**Decrement**	Decreases the value of a variable by 1	**--**x or x**--**

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 0;
   int y = 0;
   int z = 0;
   x = (++x + y--) * z++;
   System.out.println(x);
 }
}
```

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   if(1 + 1 + 1 + 1 == 5) {
    System.out.println(“TRUE”);
   } else {
    System.out.println(“FALSE”);
   }
 }
}
```

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 5;
   System.out.println(++x * x);
 }
}
```

Assignment Operator

ICTACADEMY

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 5;
   x += 3;
   System.out.println(x);
 }
}
```

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 11 & 9;
   int y = x ^ 3;
   System.out.println( y | 12 );
 }
}
```

Bitwise Operator

ICTACADEMY

Operator	Description	Example	Same as	Result	Decimal
&	AND - Sets each bit to 1 if both bits are 1	5 & 1	0101 & 0001	1	1
|	OR - Sets each bit to 1 if any of the two bits is 1	5 | 1	0101 | 0001	101	5
~	NOT - Inverts all the bits	~ 5	~0101	1010	10
^	XOR - Sets each bit to 1 if only one of the two bits is 1	5 ^ 1	0101 ^ 0001	100	4
<<	Zero-fill left shift - Shift left by pushing zeroes in from the right and letting the leftmost bits fall off	9 << 1	1001 << 1	10	2
>>	Signed right shift - Shift right by pushing copies of the leftmost bit in from the left and letting the rightmost bits fall off	9 >> 1	1001 >> 1	1100	12

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 5;
   x &= 3;
   System.out.println(x);
 }
}
```

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 5;
   x |= 3;
   System.out.println(x);
 }
}
```

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 5;
   x ^= 3;
   System.out.println(x);
 }
}
```

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 5;
   x >> = 3;
   System.out.println(x);
 }
}
```

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 5;
   x << = 3;
   System.out.println(x);
 }
}
```

Comparison Operator

ICTACADEMY

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 5;
   int y = 6;
   System.out.println(x > y);
 }
}
```

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 5;
   int y = 3;
   System.out.println(x == y);
 }
}
```

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 5;
   int y = 3;
   System.out.println(x != y);
 }
}
```

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 5;
   int y = 3;
   System.out.println(x >= y);
 }
}
```

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 5;
   int y = 3;
   System.out.println(x <= y);
 }
}
```

Logical Operator

ICTACADEMY

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 5;
   System.out.println(x > 3 && x < 10);
 }
}
```

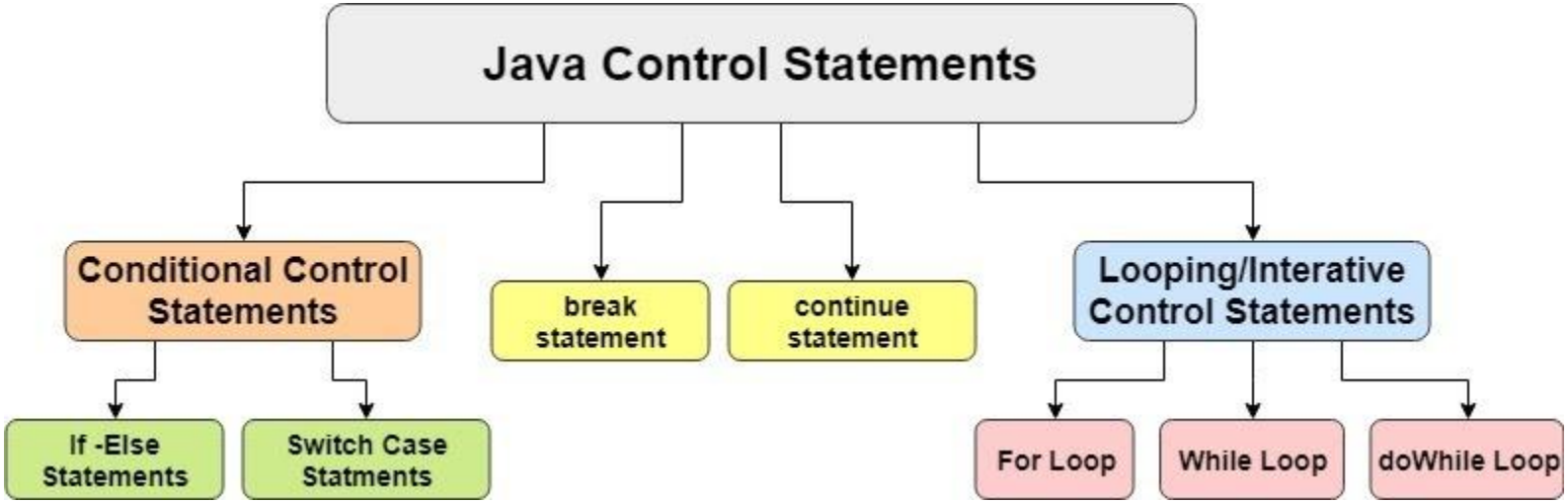
Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 5;
   System.out.println(x > 3 || x < 4);
 }
}
```

Predict the Output


```
public class Main {
 public static void main(String[] args) {
   int x = 5;
   System.out.println(!(x > 3 || x < 4));
 }
}
```

DECISION MAKING


Java Control Statements
Conditional Control Statements
break statement
continue statement
Looping/Interative Control Statements
If -Else Statements
Switch Case Statments
For Loop
While Loop
doWhile Loop

Definition

- Java decision-making statements allow you to make a decision, based upon the result of a condition.

- Either the condition can be true or condition can be false.

- Decision-making statements : if statement,if-else statement,if-else-if ladder, nested if statement, switch-case

How Decision Making Works?

- Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed.

- if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false

If statement

If statement consists a condition, followed by statement or a set of statements as shown below

```
if (condition) {
    Statement(s);
}
```

If statement

- The statements gets executed only when the given condition is true.
- If the condition is false then the statements inside if statement body are completely ignored

Example


```
public class Ifdemo {
        public static void main(String[] args) {
        //defining an 'age' variable
    int age=20;
    //checking the age
    if(age>18){
        System.out.print("Age is greater than 18");
    }
}}
```

If – else statement

- This is how an if-else statement looks

```
if (condition) {
    Statement(s);
} else {
    Statement(s);
}
```

- The statements inside “if” would execute if the condition is true, and the statements inside “else” would execute if the condition is false

- In other words,It executes the *if block* if condition is true otherwise *else block* is executed.

Example


```
public class IfElsedemo{
public static void main(String[] args) {
    //defining a variable
    int number=13;
 //Check if the number is divisible by 2 or not
    if(number%2==0){
        System.out.println("even number");
    }
else{
        System.out.println("odd number");
    }
}
}
```

If – else – if ladder statement

ICTACADEMY

- The if-else-if ladder statement executes one condition from multiple statements.

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are
false
}
```

Example


```
public class PositiveNegativeExample
{
public static void main(String[] args)
{
    int number=-13;
    if(number>0){
    System.out.println("POSITIVE");
    }
else if(number<0){
    System.out.println("NEGATIVE");
    }else{
    System.out.println("ZERO");
   }
}
}
```

NESTED IF STATEMENT

- The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.

```
if (condition_1) {
    Statement1(s);

    if (condition_2) {
        Statement2(s);
    }
}
```

Example


```
public class JavaNestedIfExample {
public static void main(String[] args)
{
   //Creating two variables for age and
weight
   int age=20;
   int weight=80;
   //applying condition on age and
weight
   if(age>=18){
      if(weight>50){
         System.out.println("You are
eligible to donate blood");
      }
   }
}}
```

SWITCH CASE STATEMENT

- Switch case statement is used when we have number of options (or choices/conditions) and we may need to perform a different task for each choice.

In other words,

- The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement.

SWITCH CASE STATEMENT – Rules

- There can be *one or N number of case values* for a switch expression.
- The case value must be of switch expression type only. The case value must be *literal or constant*. It doesn't allow variables.
- The case values must be *unique*. In case of duplicate value, it renders compile-time error.
- The Java switch expression must be of *byte, short, int, long (with its Wrapper type), enums and string*.
- Each case statement can have a *break statement* which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
- The case value can have a *default label* which is optional.

SWITCH CASE STATEMENT

The syntax of Switch case statement looks like this

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
.....
 default:
 code to be executed if all cases are not matched;
}
```

Example


```
public class SwitchExample {
public static void main(String[] args) {
    //Declaring a variable for switch
expression
    int number=20;
    //Switch expression
    switch(number){
    //Case statements
    case 10: System.out.println("10");
    break;
    case 20: System.out.println("20");
    break;
    case 30: System.out.println("30");
    break;
    //Default case statement
    default:System.out.println("Not in 10,
20 or 30");
    }
}
}
```

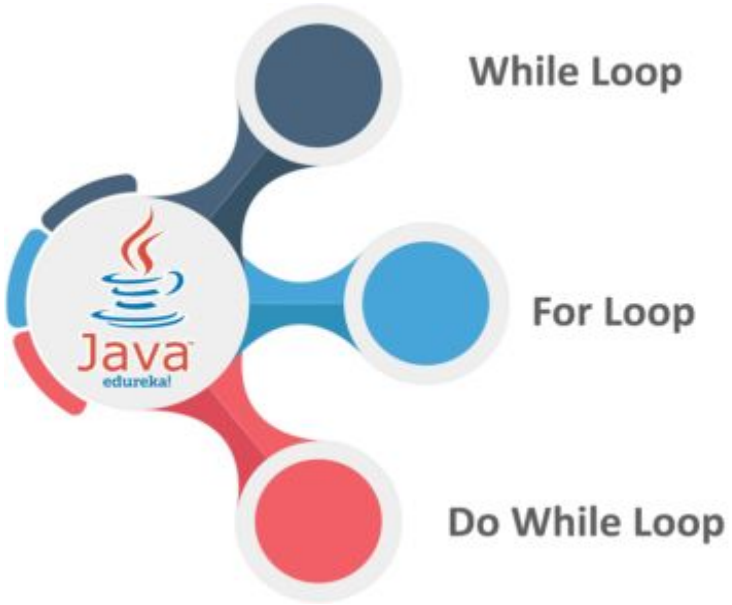
Working with Loops

What you will Learn

- for loop
- while loop
- do-while loop

Loops in Java

- Executes a set of instructions/functions repeatedly when some conditions become true. There are three types of loops in Java.
- for loop
- while loop
- do-while loop



Loops in Java

- Iterates a part of the program several times if the number of iteration is fixed.

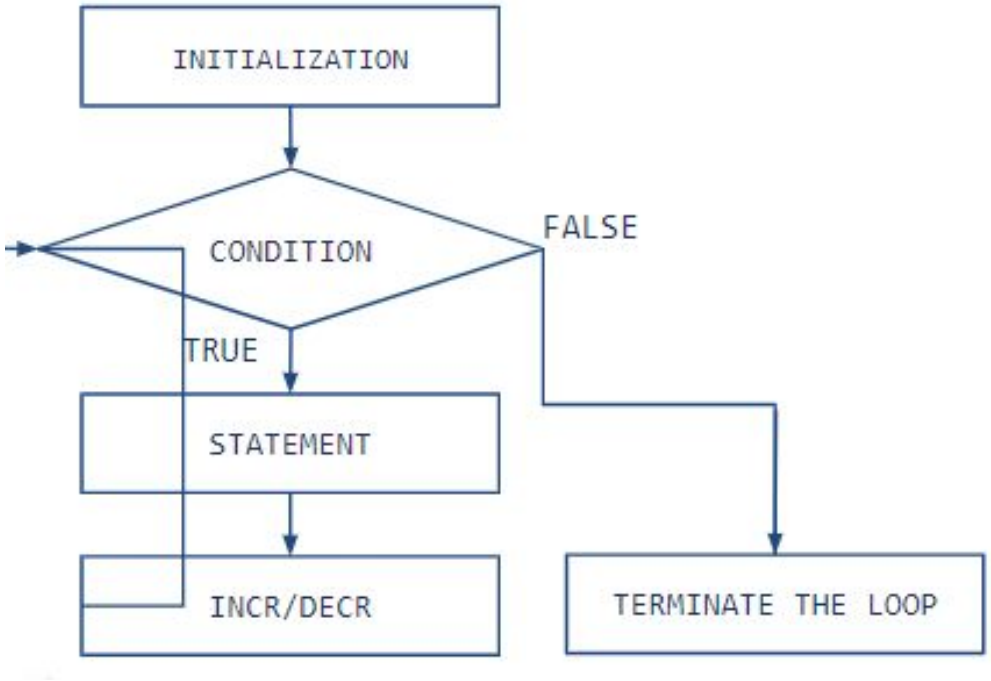
- Simple For Loop
- Nested for loop
- For-each or Enhanced For Loop
- Labeled For Loop

Simple For Loop

Syntax:

```
for(Initialization; Condition; Increment/Decrement){
  Statements;
}
```

For loops


INITIALIZATION
CONDITION
FALSE
TRUE
STATEMENT
INCR/DECR
TERMINATE THE LOOP

For Loop


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

i = 0	I < 3	o/p	i++
0	true	0	1
1	true	1	2
2	true	2	3
3	False	-	-

Example


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

NOT WORKING, WHY ?

Example


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

NOT WORKING, WHY ?

Nested For Loop


```
public class Main {
  public static void main(String[] args) {
    for(int i = 1;i <= 3;i++) {
      for(int j = 1;j <= 3;j++) {
        System.out.println(i+" "+j);
      }
    }
  }
}
```

Predict the output


```
public class Main {
  public static void main(String[] args) {
    for(int i = 1;i <= 5;i++) {
      for(int j = 1;j <= i;j++) {
        System.out.print("* ");
      }
      System.out.println();
    }
  }
}
```

Predict the output


```
public class Main {
  public static void main(String[] args) {
    int term = 6;
    for(int i = 1;i <= term;i++) {
      for(int j = term;j >= i;j--) {
        System.out.print("* ");
      }
      System.out.println();
    }
  }
}
```

For Each Loop


```
public class Main {
  public static void main(String[] args) {
    int arr[] = {12,23,44,56,78};
    for(int i:arr) {
      System.out.println(i);
    }
  }
}
```

Java Labeled For Loop


```
public class Main {
  public static void main(String[] args) {
    aa:
    for(int i = 1;i <= 3;i++) {
      bb:
      for(int j = 1;j <= 3;j++) {
        if(i == 2 && j == 2) {
          break aa;
        }
        System.out.println(i+" "+j);
      }
    }
  }
}
```

Infinite For Loop

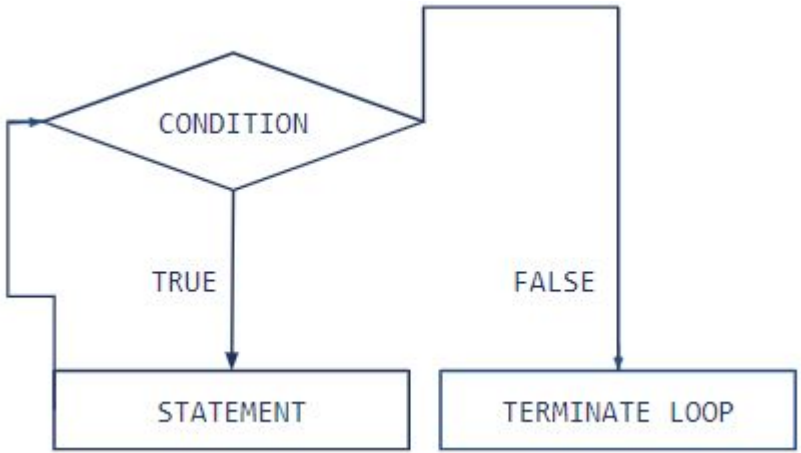

```
public class Main {
  public static void main(String[] args) {
    for(;;) {
      System.out.println("infinitive loop");
    }
  }
}
```

while Loop

- Iterates a part of the program several times.
- The number of iteration is not fixed
  Syntax:

```
 while(condition) {
//code to be executed
}
```

while Loop



Example


```
public class Main {
  public static void main(String[] args) {
    int i = 1;
    while(i <= 10) {
      System.out.println(i);
      i++;
    }
  }
}
```

Infinite while Loop

- If you pass **true** in the while loop, it will be infinitive while loop

```
while(true) {
//code to be executed
 }
```

Infinite while Loop


```
public class Main {
  public static void main(String[] args) {
    while(true) {
      System.out.println("infinitive while loop");
    }
  }
}
```

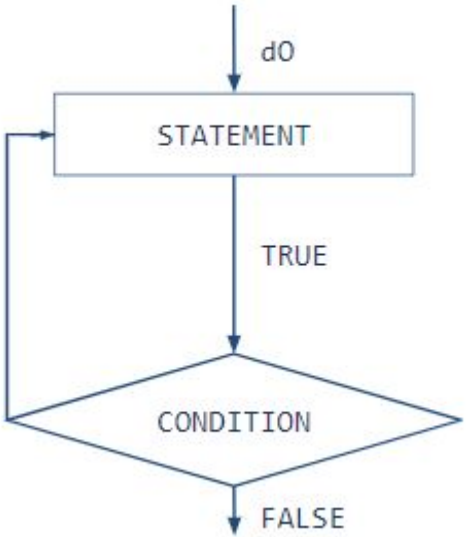
do while Loop

- Iterates a part of the program several times.
- The number of iteration is not fixed and you must have to execute the loop at least once.

Syntax:

```
do{
//code to be executed
}while(condition);
```

do while Loop



Example


```
public class Main {
  public static void main(String[] args) {
    int i = 1;
    do{
      System.out.println(i);
      i++;
    }while(i <= 10);
  }
}
```

Infinite do While - Syntax

- If you pass **true** in the do-while loop, it will be infinitive do-while loop

Syntax :

```
do{
//code to be executed
}while(true);
```

Infinite do while


```
public class Main {
  public static void main(String[] args) {
    do{
      System.out.println("infinitive do while loop");
    }while(true);
  }
}
```

Predict the output


```
public class Main {
  public static void main(String[] args) {
   for(int i = 0; i <= 5; i++ )
        {
            System.out.println("i = " + i );
        }
  }
}
```

Java break statement

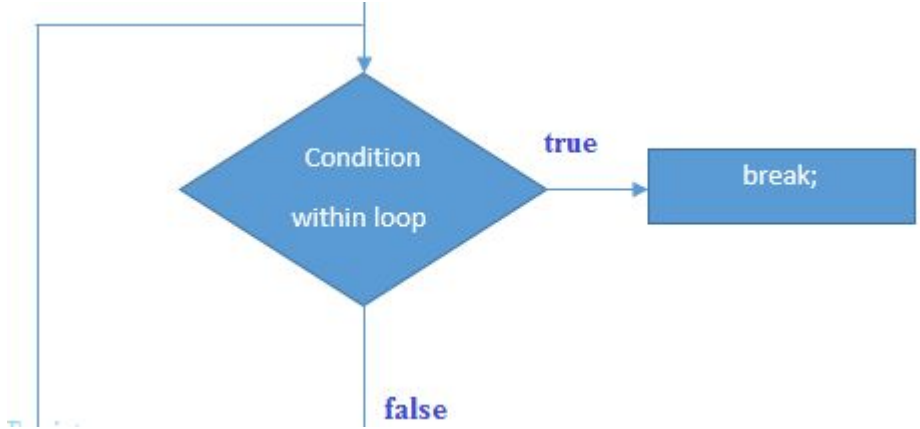
- When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

- The Java *break* statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

- We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:

jump-statement;
break;



Example


```
//Java Program to demonstrate the use of break statement
//inside the for loop.
public class BreakExample {
public static void main(String[] args) {
    //using for loop
    for(int i=1;i<=10;i++){
        if(i==5){
            //breaking the loop
            break;
        }
        System.out.println(i);
    }
}
}
```

break statement with inner loop


```
public class BreakExample2 {
public static void main(String[] a
rgs) {
        //outer loop
        for(int i=1;i<=3;i++){
            //inner loop
            for(int j=1;j<=3;j++){
                if(i==2&&j==2){
                    //using brea
k statement inside the inner l
oop
                    break;
                }
                System.out.pri
ntln(i+" "+j);
            }
        }
}
}
```

Java continue statement

The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.

The Java *continue statement* is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.

We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:

jump-statement;
continue;

Example


```
//Java Program to demonstrate the use of continue statement
//inside the for loop.
public class ContinueExample {
public static void main(String[] args) {
    //for loop
    for(int i=1;i<=10;i++){
        if(i==5){
            //using continue statement
            continue;//it will skip the rest statement
        }
        System.out.println(i);
    }
}
}
```

continue statement with inner loop


```
//Java Program to illustrate the u
se of continue statement
//inside an inner loop
public class ContinueExample2 {

public static void main(String[] a
rgs) {
        //outer loop
        for(int i=1;i<=3;i++){
            //inner loop
            for(int j=1;j<=3;j++){
 if(i==2&&j==2){
                    //using conti
nue statement inside inner l
oop
                        continue;
                    }
 System.out.println(i+" "+j);

                }
        }
}
}
```

Assignment

Write a program to calculate the sum of first 10 natural number.

Write a program that prompts the user to input a positive integer. It should then print the multiplication table of that number.

Write a program to find the factorial value of any number entered through the keyboard.

Write a program to print Fibonacci series.