

Basic Operators

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

The Arithmetic Operators:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0
++	Increment - Increases the value of operand by 1	B++ gives 21
--	Decrement - Decreases the value of operand by 1	B-- gives 19

```

public class Test{

public static void main(String args[]){
int a =10;
int b =20;
int c =25;
int d =25;

System.out.println("a + b = "+(a + b));
System.out.println("a - b = "+(a - b));
System.out.println("a * b = "+(a * b));
System.out.println("b / a = "+(b / a));
System.out.println("b % a = "+(b % a));
System.out.println("c % a = "+(c % a));
System.out.println("a++= "+(a++)); // 1 0 + +
System.out.println("a-- = "+(a--));//11 --
// Check the difference in d++ and ++d
System.out.println("d++= "+(d++)); // p o s t i n c r e m e n t
System.out.println("++d= "+(++d));//pre increment
}

```

Output:

Output:

```

a + b =30
a - b =-10
a * b =200
b / a =2
b % a =0
c % a =5
a++=10
a--=11
d++=25
++d =27

```

Relational Operators:

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of right operand is greater than or equal to the value	(A >= B) is not true.
<=	Checks if the value of left operand is greater than or equal to the value	(A <= B) is true.

```
public class Test{  
  
    public static void main(String args[]){  
        int a =10;  
        int b =20;  
        System.out.println("a == b = "+(a == b));  
        System.out.println("a != b = "+(a != b));  
        System.out.println("a > b = "+(a > b));  
        System.out.println("a < b = "+(a < b));  
        System.out.println("b >= a = "+(b >= a));  
        System.out.println("b <= a = "+(b <= a));  
    }  
}
```

Output:

a == b =false

a != b =true

a > b =false

a < b =true

b >= a =true

b <= a =false

The Bitwise Operators:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
 	Binary OR Operator copies a bit if it exists in either operand.	(A B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>>	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

```

public class Test{

public static void main(String args[]){
int a =60;  /* 60 = 0011 1100 */
int b =13;  /* 13 = 0000 1101 */
int c =0;


    c = a & b; /* 12 = 0000 1100 */
    System.out.println("a & b = "+ c );

    c = a | b; /* 61 = 0011 1101 */
    System.out.println("a | b = "+ c );

    c = a ^ b; /* 49 = 0011 0001 */
    System.out.println("a ^ b = "+ c );

    c = ~a; /* -61 = 1100 0011 */
    System.out.println("~a = "+ c );

    c = a <<2; /* 240 = 1111 0000 */
    System.out.println("a << 2 = "+ c );

    c = a >>2; /* 215 = 1111 */
    System.out.println("a >> 2 = "+ c );

    c = a >>>2; /* 215 = 0000 1111 */
    System.out.println("a >>> 2 = "+ c );
}

```

Output:

```

a & b =12
a | b =61
a ^ b =49
~a =-61
a <<2=240
a >>15
a >>>15

```

Logical Operators:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

```
public class Test{  
  
    public static void main(String args[]){  
        boolean a =true;  
        boolean b =false;  
        System.out.println("a && b = "+(a&&b));  
        System.out.println("a || b = "+(a||b));  
        System.out.println("!(a && b) = "+!(a && b));  
    }  
}
```

Output:

```
a && b =false  
a || b =true  
!(a && b)=true
```

Assignment Operators:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C=A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
 =	bitwise inclusive OR and assignment operator	C = 2 is same as C = C 2

Increment and Decrement Operators

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

```
        // declare variables  
        int a = 12, b = 12;  
        int result1, result2;
```

```
        // original value  
        System.out.println("Value of a: " + a);
```

```
        // increment operator  
        result1 = ++a;  
        System.out.println("After increment: " + result1);
```

```
        System.out.println("Value of b: " + b);
```

```
        // decrement operator  
        result2 = --b;  
        System.out.println("After decrement: " + result2);  
    }  
}
```

Output:

Value of a: 12
After increment: 13
Value of b: 12
After decrement: 11

Misc Operators

Conditional Operator (?:):

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions.

variable x = (expression) ? value if true : value if false

```
public class Test{
public static void main(String args[]){
int a , b;
a =10;
b =(a ==1) ?20:30;
System.out.println("Value of b is : "+ b );
b =(a ==10) ?20:30;
System.out.println("Value of b is : "+ b );
}
}
```

Output:

Value of b is:30

Value of b is:20

instanceof Operator:

This operator is used only for object reference variables.

```
(Object reference variable ) instanceof (class/interface type)
```

```
class Simple1{  
    public static void main(String args[]){  
        Simple1 s=new Simple1();  
        System.out.println(s instanceof Simple1);//true  
    }  
}
```

Output:

true

Precedence of Java Operators:

Operator precedence determines the grouping of terms in an expression.

For example, $x = 7 + 3 * 2$; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with $3*2$ and then adds into 7

BODMAS - Bracket Order Division Multiplication Addition Subtraction

Bracket = ()

Order = Square root , Power of

Division = /

Multiplication = *

Addition = +

Subtraction = -

Loop Control

- while Loop
- do...while Loop
- for Loop

while Loop:

```
public class Test{  
    public static void main(String args[]) {  
        int x =10;  
        while( x <20) {  
            System.out.println("value of x : "+ x );  
            x++;  
            //System.out.print("\n");  
        }  
    }  
}
```

Output:

```
value of x :10  
value of x :11  
value of x :12  
value of x :13  
value of x :14  
value of x :15  
value of x :16  
value of x :17  
value of x :18  
Value of x : 19
```

The do...while Loop:

Syntax:

```
do
{
//Statements
}while(Boolean_expression);
```

```
public class Test{
public static void main(String args[]) {
int x =20;
do{
System.out.print("value of x : "+ x );
x++;
System.out.print("\n");
}while( x <20);
}
}
```

Output:

```
value of x :10
value of x :11
value of x :12
```

The for Loop:

Syntax:

```
for (initialization; Boolean_expression; update)
{
//Statements
}
```

```
public class Test{
```

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

```
for (int x =10; x <20; x = x+1) {
```

```
System.out.print ("value of x : "+ x );
```

```
System.out.print ("\n");
```

```
}
```

```
}
```

```
}
```

Output:

```
value of x :10
```

```
value of x :11
```

```
value of x :12
```

Enhanced for loop in Java:

Syntax:

```
for (declaration : expression)
{
//Statements
}
```

- **Declaration:** The newly declared block variable, which is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.
- **Expression:** This evaluates to the array you need to loop through. The expression can be an array variable or method call that returns an array.

```
public class Test{

public static void main(String args[]){
int[] numbers ={10,20,30,40,50};

for(int x : numbers ){
System.out.print(x);
System.out.print(",");
}
System.out.print("\n");
String[] names={"James","Larry","Tom","Lacy"};
for(String name : names ){
System.out.print( name );
System.out.print(",");
}
}
}
```

Output:

```
10,20,30,40,50,
James,Larry,Tom,Lacy,
```


The break Keyword:

```
public class Test{  
  
    public static void main (String  
    args[]) {  
        int[] numbers = {10, 20, 30, 40, 50};  
        for (int x : numbers) {  
            if (x == 30) {  
                break;  
            }  
            System.out.print ( x );  
            System.out.print ("\n");  
        }  
    }  
}
```

Output:

10
20

The continue Keyword:

```
public class Test{

public static void main(String args[]) {
int[] numbers ={10,20,30,40,50};

for(int x : numbers){
if( x ==30){
    continue;
}
System.out.print( x );
System.out.print("\n");
}
}
}
```

Output:

10
20
40
50

Decision Making

The if Statement:

An if statement consists of a Boolean expression followed by one or more statements.

Syntax:

```
if (Boolean_expression)
{
//Statements will execute if the Boolean expression is true
}
```

```
public class Test{
```

```
public static void main(String args[]) {
int x =10;
```

```
if( x <20) {
System.out.print("This is if statement");
}
}
}
```

Output:

This is if statement

The if...else Statement:

An if statement can be followed by an optional e/se statement, which executes when the Boolean expression is false.

Syntax:

```
if(Boolean_expression){  
    //Executes when the Boolean expression is true  
}  
else{  
    //Executes when the Boolean expression is false  
}
```

```
public class Test{  
  
    public static void main(String args[]){  
        int x =30;  
  
        if(x <20){  
            System.out.print("This is if statement");  
        }else{  
            System.out.print("This is else statement");  
        }  
    }  
}
```

Output:

This is else statement

The if...else if...else Statement:

Syntax:

```
if(Boolean_expression1){  
    //Executes when the Boolean expression 1 is true  
}  
else if(Boolean_expression2){  
    //Executes when the Boolean expression 2 is true  
}  
else if(Boolean_expression3){  
    //Executes when the Boolean expression 3 is true  
}  
else{  
    //Executes when the none of the above condition is true.  
}
```

```
public class Test{
```

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

```
        int x =30;
```

```
        if( x ==10){
```

```
            System.out.print("Value of X is 10");
```

```
        }else if( x ==20){
```

```
            System.out.print("Value of X is 20");
```

```
        }else if( x ==30){
```

```
            System.out.print("Value of X is 30");
```

```
        }else{
```

```
            System.out.print("This is else statement");
```

```
        }
```

```
    }
```

```
}
```

Output:

Value of X is30

Nested if...else Statement:

Syntax:

```
if(Boolean_expression1){  
    //Executes when the Boolean expression 1 is true  
    if(Boolean_expression2){  
        //Executes when the Boolean expression 2 is true  
    }  
}
```

Output:

X =30and Y =10

```
public class Test{  
  
    public static void main(String args[]){ int x =30;  
    int y =10;  
  
    if( x ==30){  
        if( y ==10){  
            System.out.print("X = 30 and Y = 10");  
        }  
    }  
}
```

The switch Statement:

```
switch(expression){  
    case value :  
        //Statements  
        break;//optional  
    case value :  
        //Statements  
        break;//optional  
    //You can have any number of case statements.  
    default://Optional  
        //Statements  
}
```

```

import java.util.*;
public class Test{
public static void main(String args[]){
Scanner scan=new Scanner(System.in);
System.out.println("Enter the grade:");
char ch=scan.next().charAt(0);
switch(ch)
{
case 'A': System.out.println("Excellent!");
break;
case 'B':
case 'C': System.out.println("Well done");
break;
case 'D':
System.out.println("You passed");
case 'F':
System.out.println("Better try again");
break;

default:
System.out.println("Invalid grade");
}
System.out.println("Your grade is "+ ch);
}
}

```

Output:

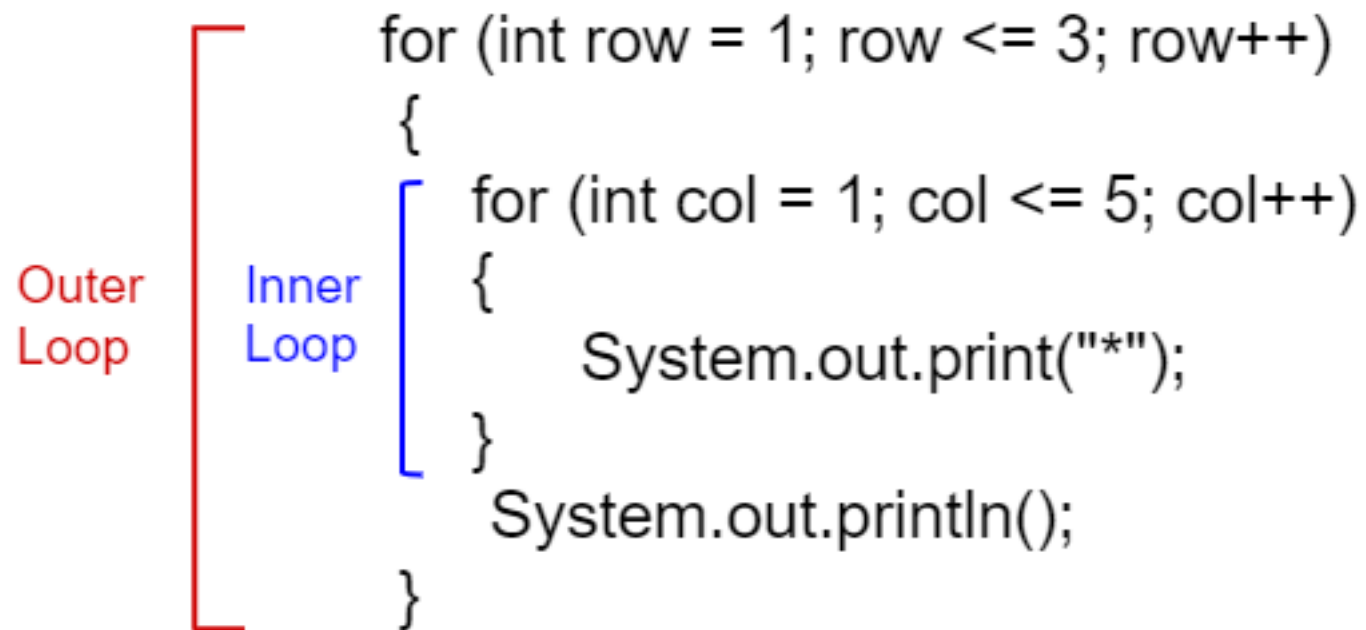
```

$ java Test a Invalid grade Your grade is a
$ java Test A Excellent!
Your grade is a A
$ java Test C Well done
Your grade is a C
$

```

Nested Loops

A *nested loop* is a (inner) loop that appears in the loop body of another (outer) loop. The inner or outer loop can be any type: **while**, **do while**, or **for**. For example, the inner loop can be a **while** loop while an outer loop can be a **for** loop.



The diagram illustrates nested loops using a code example. A red bracket on the left, labeled "Outer Loop", spans the entire code block. A blue bracket on the left, labeled "Inner Loop", spans the inner loop's body. The code is as follows:

```
for (int row = 1; row <= 3; row++)  
{  
    for (int col = 1; col <= 5; col++)  
    {  
        System.out.print("*");  
    }  
    System.out.println();  
}
```

Labelled and Unlabelled Statements:

The *continue* Statement, Without a Label