JAVA OPERATORS

GENERAL

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:

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

ONLINE COMPILER

As there were certain problems with propper settings of NetBeams, I decided to demonstrate all examples using on-line compiler and runner

http://www.browxy.com/

Please consider using this tool.

THE ARITHMETIC OPERATORS

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators. Assume integer variable A holds 10 and variable B holds 20, then:

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

Example

The following simple example program demonstrates the arithmetic operators. Copy and paste the following Java program into Test.java file and compile and run this program:

```
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++) );
     System.out.println("b--   = " +  (a--) );
     // Check the difference in d++ and ++d
     System.out.println("d++   = " +  (d++) );
     System.out.println("++d   = " +  (++d) );
  }
}
```

This would produce the following result:

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

THE RELATIONAL OPERATORS

There are following relational operators supported by Java language.

Assume variable A holds 10 and variable B holds 20, then:

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	(A != B) is true.

	then condition becomes 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 left operand is greater than or equal to 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 or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Example

The following simple example program demonstrates the relational operators. Copy and paste the following Java program in **Test.java** file and compile and run this program. :

```
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) );
  }
}
```

This would produce the following result:

```
a == b = false
a != b = true
a > b = false
a < b = true
b >= a = true
b <= a = false
```

THE BITWISE OPERATORS:

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60; and b = 13; now in binary format they will be `a = 0011 1100` and `b = 0000 1101`.

```
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a  = 1100 0011
```

The following table lists the bitwise operators. Assume integer variable A holds 60 and variable B holds 13 then:

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 -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	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

Example

The following simple example program demonstrates the bitwise operators. Copy and paste the following Java program in Test.java file and compile and run this program:

```
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 );
   }
}
```

This would produce the following result:

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

THE LOGICAL OPERATORS

The following table lists the logical operators. Assume Boolean variables A holds true and variable B holds false, then:

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.

Example

The following simple example program demonstrates the logical operators. Copy and paste the following Java program in Test.java file and compile and run this program:

```
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));
   }
}
```

This would produce the following result:

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

THE ASSIGNMENT OPERATORS

There are following assignment operators supported by Java language:

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

Example

The following simple example program demonstrates the assignment operators. Copy and paste the following Java program in **Test.java** file and compile and run this program:

```
public class Test {

  public static void main(String args[]) {
     int a = 10;
     int b = 20;
     int c = 0;

     c = a + b;
```

```
        System.out.println("c = a + b = " + c );

        c += a ;
        System.out.println("c += a  = " + c );

        c -= a ;
        System.out.println("c -= a = " + c );

        c *= a ;
        System.out.println("c *= a = " + c );

        a = 10;
        c = 15;
        c /= a ;
        System.out.println("c /= a = " + c );

        a = 10;
        c = 15;
        c %= a ;
        System.out.println("c %= a  = " + c );

        c <<= 2 ;
        System.out.println("c <<= 2 = " + c );

        c >>= 2 ;
        System.out.println("c >>= 2 = " + c );

        c >>= 2 ;
        System.out.println("c >>= a = " + c );

        c &= a ;
        System.out.println("c &= 2  = " + c );

        c ^= a ;
        System.out.println("c ^= a   = " + c );

        c |= a ;
        System.out.println("c |= a   = " + c );
    }
}
```

This would produce the following result:

```
c = a + b = 30
c += a  = 40
c -= a = 30
c *= a = 300
c /= a = 1
c %= a  = 5
c <<= 2 = 20
c >>= 2 = 5
c >>= 2 = 1
c &= a  = 0
c ^= a   = 10
```

```
c |= a   = 10
```

MISCELANEOUS OPERATORS

There are few other operators supported by Java Language:

Conditional Operator (? :)

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide which value should be assigned to the variable. The operator is written as:

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

EXAMPLE

```
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 );
   }
}
```

This would produce the following result:

```
Value of b is : 30
Value of b is : 20
```

instanceOf Operator

NOTE

This is only for documentation purposes.

This operator is used only for object reference variables. The operator checks whether the object is of a particular type(class type or interface type). The **instanceOf** operator is wriiten as:

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

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true. Following is the example:

EXAMPLE

```
String name = = 'James';
boolean result = name instanceOf String;
// This will return true since name is type of String
```

This operator will still return true if the object being compared is the assignment compatible with the type on the right. Following is one more example:

EXAMPLE

```
class Vehicle {}

public class Car extends Vehicle {
   public static void main(String args[]){
      Vehicle a = new Car();
      boolean result =  a instanceof Car;
      System.out.println( result);
   }
}
```

This would produce the following result:

```
true
```

PRECEDENCE OF JAVA OPERATORS

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.

Example

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.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right

Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>> = <<= &= ^= |=	Right to left
Comma	,	Left to right

CONTROL STRUCRURES

LOOPS

There may be a situation when we need to execute a block of code several number of times, and is often referred to as a loop.

Java has very flexible three looping mechanisms. You can use one of the following three loops:

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

As of Java 5, the enhanced for loop was introduced. This is mainly used for Arrays and we will not explain it.

while Loop

A while loop is a control structure that allows you to repeat a task a certain number of times.

SYNTAX:

The syntax of a while loop is:

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

When executing, if the **boolean_expression** result is true, then the actions inside the loop will be executed. This will continue as long as the expression result is true.

Here, key point of the while loop is that the loop might not ever run. When the expression is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

EXAMPLE

```
public class Test {

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

      while( x < 20 ) {
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }
```

```
   }
}
```

This would produce the following result:

```
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
```

do … while Loop

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

SYNTAX:

The syntax of a do...while loop is:

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

Notice that the **Boolean_ expression** appears at the end of the loop, so the statements in the loop executes once before the Boolean is tested.

If the Boolean expression is true, the flow of control jumps back up to do, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

EXAMPLE:

```
public class Test {

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

      do{
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }while( x < 20 );
   }
}
```

This would produce the following result:

```
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 for Loop:

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

A for loop is useful when you know how many times a task is to be repeated.

SYNTAX:

The syntax of a for loop is:

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

As you can see, for consist of three parts, or steps.

The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.

Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement past the for loop.

After the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the Boolean expression.

The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates.

EXAMPLE:

```
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");
         }
      }
   }
```

This would produce the following result:

```
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 break Keyword:

The break keyword is used to stop **the entire loop**. The break keyword must be used inside any loop or a switch statement.

The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

SYNTAX:

The syntax of a break is a single statement inside any loop:

```
break;
```

EXAMPLE:

```
public class Test {

   public static void main(String args[]) {
      int x;
      for(x=0; x<=50; x++ ) {
         if( x == 3 ) {
            break;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```

This would produce the following result:

```
0
```

```
1
2
```

The continue Keyword:

The **continue** keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.

In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.

SYNTAX:

The syntax of a continue is a single statement inside any loop:

```
continue;
```

EXAMPLE:

```
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");
      }
   }
}
```

This would produce the following result:

```
10
20
40
50
```

DECISION MAKING

There are two types of decision making statements in Java. They are:

- if statements

- switch statements

The if Statement:

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

SYNTAX:

The syntax of an if statement is:

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

If the Boolean expression evaluates to true then the block of code inside the if statement will be executed. If not the first set of code after the end of the if statement (after the closing curly brace) will be executed.

Example:

```
public class Test {

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

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

This would produce the following result:

```
This is if statement
```

The if...else Statement:

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

SYNTAX:

The syntax of an if...else is:

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

Example:

```
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");
         }
      }
   }
```

This would produce the following result:

```
   This is else statement
```

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

An if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.

- When using if , else if , else statements there are few points to keep in mind:
- An if can have zero or one else's and it must come after any else if's.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of the remaining else if's or else's will be tested.

SYNTAX:

The syntax of an if...else is:

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

EXAMPLE:

```
   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");
         }
      }
   }
```

This would produce the following result:

```
Value of X is 30
```

Nested if...else Statement:

It is always legal to nest if-else statements which means you can use one if or else if statement inside another if or else if statement.

SYNTAX:

The syntax for a nested if...else is as follows:

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

You can nest else if...else in the similar way as we have nested if statement.

EXAMPLE:

```
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");
          }
       }
    }
}
```

This would produce the following result:

```
X = 30 and Y = 10
```

The switch Statement:

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

SYNTAX:

The syntax is:

```
switch(expression){
    case value1 :
       //Statements
       break; //optional
    case value2 :
       //Statements
       break; //optional
    //You can have any number of case statements.
    default : //Optional
       //Statements
}
```

The following rules apply to a switch statement:

- The variable used in a switch statement can only be a byte, short, int, or char.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.

Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.

A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

EXAMPLE:

```
public class Test {

   public static void main(String args[]){
      char grade = 'C';
```

```
         switch(grade)
         {
            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 " + grade);
      }
   }
```

This would produce the following

```
Well done
Your grade is a C
```