Programming Java

STEP IT ACADEMY

Lesson 3

Operators

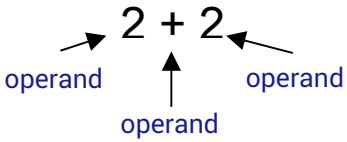
Contents

1. Operations

In Java, operations are special characters that indicate what action should be performed for the specified operands.

Operand can be a literal, variable, or expression, on which the operation is performed.

Operations can be applied to one (unary operator), two (binary operation), or three (ternary operation) operands.



Operations can be combined in expressions.

Unlike C++, Java has no operator overloading since the creators of this language decided that operator overloading significantly confuses the code and makes it harder to understand it.

There are several operations, which are overloaded by default:

- + is used for adding numbers and string concatenation;
- & is used for bitwise operations on numbers and logical AND;
- | is used for bitwise operations on numbers and logical OR;
- ^ is used for bitwise operations on numbers and logical XOR;
- == compares any types;
- != compares any types.

Arithmetical operations

Operator	Description	Example

+	Unary plus; does not change the value of operand.	`int i = -1;`
		`i = +i;`
		`System.out.println(i);`

	Binary plus; adds the operands.	`int i = -1;`
		`i = i + 3;`
		`System.out.println(i);`

–	Unary minus; reverses the sign of an operand.	`int i = 1;`
		`i = -i;`
		`System.out.println(i);`

	Binary minus; subtracts the first operand from the second.	`int i = 3;`
		`i = i - 2;`
		`System.out.println(i);`

*	Operand multiplication.	`int i = 2;`
		`i = i * 2;`
		`System.out.println(i);`

/	Division of the left operand by the right.	`int i = 4;`
		`i = i / 2;`
		`System.out.println(i);`

%	Calculates the remainder of division of the left operand by the right.	`int i = 3;`
		`i = i % 2;`
		`System.out.println(i);`

++	Binary operator that increases the value of a variable by 1. Increment.	i++; replaces the expression i = i + 1;
––	Binary operator that decreases the value of variable by 1. Decrement.	i--; replaces the expression i = i - 1;

Bitwise Operations

Bitwise operations perform calculations using bitwise representation of a number (in binary system) as operands.

NOTE!!! *Only integer primitive types or their wrapper classes can be used in bitwise operations as operands.*

Operation	Description	Example	Outcome
&	AND	If a significant bit is in the left AND right operand, we obtain a significant bit in each digit of the outcome. `int i = 123 & 456;` Bit representation 00000000000000000000000001111011 & 00000000000000000000000111001000 = 00000000000000000000000001001000	72

|	OR	If a significant bit is in the left OR right operand, we obtain a significant bit in each digit of the outcome.
		`final int one = 123;`
		`final int two = 456;`
		`int i = one | two;`
		Bit representation
		00000000000000000000000001111011 |
		00000000000000000000000111001000 =
		00000000000000000000000111111011
			507

^	XOR (excluding OR)	If a significant bit is in the left OR right operand, we obtain a significant bit in each digit of the outcome. If a bit is in a digit of both operands, the result will be 0. `int i = 123 ^ 456;` Bit representation 00000000000000000000000001111011 ^ 00000000000000000000000111001000 = 00000000000000000000000110110011	435
~	NOT (bitwise negation)	Inverts all bits of a number to the opposite (including a significant bit) `int i = ~123;` Bit representation: ~ 00000000000000000000000001111011= 11111111111111111111111110000100	-124
<<	left shift	Shifts the bits of the left operand to the left by the number of bits specified in the right operand, filling the right digits with zeros.	
>>	right shift	Shifts the bits of the left operand to the right by the number of bits specified in the right operand, filling the right digits with zeros.	

Operation	Description	Example	Outcome
>>>	right shift unsigned	Shifts the bits of the left operand to the right by the number of bits specified in the right operand, filling the left digits with zeros ignoring the bit shift.	

Integer.toBinaryString(value) converts an integer value to a string in the bitwise representation.

Example of a method that outputs a number in the bitwise representation with the addition of insignificant zeros:

```
private static void printToBinary(int value)
{
    System.out.println(String.format("%32s",
                Integer.toBinaryString(value))
                .replace(' ', '0'));
}
```

Comparison Operators

Outcome of the comparison operation is always of **boolean** type.

Operator	Condition	Example
>	The left operand is greater than the right operand.	int a = 7; boolean r = a > 5;
<	The left operand is less than the right operand.	int a = 7; boolean r = a < 5;
>=	The left operand is greater or equal to the right operand.	int a = 7; boolean r = a >= 5;
<=	The left operand is less or equal to the right operand.	int a = 7; boolean r = a <= 5;
==	The left operand is equal to the right operand.	int a = 7; boolean r = a == 5;

Operator	Condition	Example
!=	The left operand is not equal to the right operand.	int a = 7; boolean r = a != 5;

Logical Operations

Logical operations are applied only to the **boolean** type operands. Outcome of logical operations is of the **boolean** type.

Operator operations	Description
&	Logical AND
|	Logical OR
^	Logical XOR
||	Short circuit OR*
&&	Short circuit AND*
!	Negation

** Short circuit operators do not calculate the result of expression in the right operand if the result of operation can be determined by the value of the first operand.*

Example for AND:

```
boolean t = (5 == 3) || 5 != 3
```

In this example, the result of calculating the left operand expression will be false, so there is no need to calculate the right operand in order to calculate the result of operation.

Example of OR:

```
boolean t = (2 == 2) || 3 != 2
```

In this example, the result of calculating the left operand expression will be true, so there is no need to calculate the right

operand in order to calculate the result of the AND operation.

Truth table of logical operations

Operand 1	Operand 2	Operation			
		|	&	^	!Operand1
false	false	false	false	false	true
true	false	true	false	true	false
false	true	true	false	true	true
true	true	true	true	false	false

Assignment Operation

Assignment operation is used for storing a literal value, variable, or expression value in a variable. Assignment operator has the lowest priority, so all the operations in the expression are executed first, and only then the calculated value will be passed to a variable.

Syntax:

```
variable = expression;
```

For example:

```
int x = 3; // variable will be initialized
           // with the value of 3

x = 6; // variable value will be overwritten,
       // and the old value will be lost
```

Java allows assigning values to multiple variables at a time.

For example:

```
int x;
int y;
int z;
```

```
x = y = z = 3;

System.out.println(x + y + z);
```

Outcome: 9

Ternary Operator

Ternary operator contains three operands. The second or third operand can be an outcome of operation. The first operand should be of the boolean type.

Syntax:

```
expression1 ? expression2 : expression3;
```

Operation will return the value of "expression2" if the result of calculating "expression1" is true, otherwise it returns the value of "expression3".

For example:

```
int x = 1;
int y = 2;
int a = (x != y) ? x : y; // for better code readability,
// it is recommended to put condition in brackets.
System.out.println(a);
```

Outcome: 1

Operator Precedence

Operations	Description
() []	Round braces and square brackets
++ -- + - ~ !	Decrement, increment, unary plus, unary minus, bitwise negation, logical negation
* / %	Multiplication, division, remainder of division
-	Addition and subtraction

Operations	Description
>> >>> <<	Bitwise shifts: right, right unsigned, bitwise left shift
> >= < <=	Comparison: greater than or equal to, less than, less than or equal
== !=	Comparison of equality, comparison of inequality
& ^ | && ||	Bitwise and logical: AND, XOR, OR, short circuit AND, OR
? :	Ternary conditional operation
= operation	Combined operators

Operations are arranged in the table by priority descending from top to bottom and from left to right.

2. Control Structures

Java has two conditional operators for implementing the branching algorithm in the code: **if ... else** and **switch**. There are three operators for implementing looping algorithms: **for**, **while**, and **do..while**. The syntax and operation principle is similar to that in C++ and C#.

Conditional Operators

Syntax:

```
if (expression) operation;
```

Parentheses may contain any specified expression, the result of which should be of **boolean** type. If the expression in parentheses takes the value of true, the *operation* following the parentheses will be executed, otherwise program control will be passed to the next line of code.

For example:

```
int a = 3;
if (a == 3) a = a + 2;
System.out.println(a);
```

Outcome: 5

The block separator (curly braces) should be used for indication of the block of operations that will be executed if multiple operations should be executed in the **if** operator as a result of condition execution.

For example:

```
int a = 3;
if (a == 3) { // beginning of the block
    a = a + 2;
    a--;
    // end of the block }
System.out.println(a);
```

Outcome: 4

The **else** keyword can be used along with the **if** operator. Syntax:

```
if (expression) operation1;
else operation2;
```

If the *expression* in parentheses takes the value of false, then the "operation 2" will be executed, while the "operation 1" will not be executed.

For example:

```
int x = 3;
if (x > 3) System.out.println(++x);
else       System.out.println(--x);
```

Outcome: 2

Block separator can also be used for **else** in case there is a need to execute multiple commands.

For example:

```
int a = 3;
if (a != 3) {
    System.out.println(a - 3);
}
else
{
    System.out.println(a + 3);
}
```

The **if** operator can be used in conjunction with the **else** operator for creating logical chains.

For example:

```
int age = 18;
if (age < 1) System.out.println("Baby");
else if (age >= 1 && age < 16)
        System.out.println("Child");
else if (age >= 16 && age < 19)
        System.out.println("Teenager");
else if (age >= 19) System.out.println("Adult");
```

The **switch** operator was added in java to simplify the implementation of multiple choice algorithm. This operator

analyzes the expression in parentheses and passes control to one of the **cases**. Then, the program will execute all the code in the case to the end of the **switch** operator. The values of options in the **case** should not be repeated.

Syntax:

```
switch (expression)
{
    case option1:
         operation1;
    case option2:
         operation2;
    case option3:
         operation3;
    default:
         operation;
}
```

The values of the `byte`, `short`, `char`, `int` primitive types are allowed to be specified in the braces.

For example:

```
int x = 2;
switch(x)
{
    case 1:
            System.out.println(1);
    case 2: // this case will be selected,
            // and all the //commands
            // to the end of the switch block will
            // be executed
             System.out.println(2);
    case 3:
             System.out.println(3);
}
```

Outcome: 2, 3

The break keyword can be used in the switch block for terminating the case.

For example:

```
int x = 2;
switch(x)
{
     case 1:
          System.out.println(1);
           break; // termination of case execution;
                  // control is passed to the end
                  // beyond the switch block
     case 2:
           System.out.println(2);
           break;
      case 3:
           System.out.println(3);
           break;
}
```

Outcome: 1

The `default` keyword can be used in the **switch** operator if none of the cases were executed. Only one `default` keyword can be in the switch block. The order of the **case** and **default** blocks does not matter, but it is commonly accepted to arrange the case blocks by the values in ascending order, while default is located at the very end.

For example:

```
int x = 5;

switch(x)
{
     case 3:
```

```
            System.out.println(3);
            break;
      case 1:
            System.out.println(1);
            break;
            default:
            System.out.println("default");
      case 2:
             System.out.println(2);
            break;
}
```

Outcome: default

Lesson 3
Operators