**Arithmetic operators**

| Operator (العملية) | Operands (الحدود) | Example |
|---|---|---|
| + Additive | 2 | 5 + 2 |
| - Subtraction | 2 | 5 – 2 |
| * Multiplication | 2 | 5 * 2 |
| / Division | 2 | 5 / 2 |
| % Remainder | 2 | 10 % 2 |
| = Assignment | 2 | x = 10 |
| + Unary plus | 1 | +x |
| - Unary minus | 1 | -x |
| ++ Increment | 1 | x++ or ++x |
| – Decrement | 1 | x-- or –x |
| += Addition assignment | 2 | x += 2 |
| -= Subtractions assignment | 2 | x -= 2 |
| *= Multiplication assignment | 2 | x *= 2 |
| /= Division assignment | 2 | x /= 2 |
| %= Reminder assignment | 2 | x %= 2 |

**Operator Precedence**

| Level | Operators | Associativity |
|---|---|---|
| 1 | ()<br>+ (unary plus)<br>- (unary minus) | Left to right |
| 2 | *<br>/<br>% | Left to right |
| 3 | +<br>- | Left to right |
| 4 | =<br>+= -= *= /= %= | Right o left |

**Arithmetic Expression evaluations**

When java evaluates the arithmetic expression, it will divide the expression into operations and evaluate every operation alone, it will start with the height precedence.

**Examples:**

**Example 1**

x = 4 + 3 * 6 – 9 / 3
we have * and / same level, you go left to right

1)- 3 * 6 = 18
x = 4 + 18 – 9 / 3

2)- 9 / 3 = 3
x = 4 + 18 – 3
we have + and - same level, you go left to right

3)- 4 + 18 = 22
x = 22 – 3

4)- x = 19

**Example 2**

x = (4 + 3) * 6 – 9 / 3
we have () , you must start with it.

1)- (4 + 3) = 7
x = 7 * 6 – 9 / 3

2)- 7 * 6 = 42
we have * and / same level, you go left to right
x = 42 – 9 / 3

3)- 9/3 = 3
x = 42 – 3

4- x = 39

**Example 3**

```
x = y = z = 3
```

we have = repeated 3 times, you go right to left

```
1)- z = 3
    x = y = 3

2)- y = 3
    x = 3
```

at the end we have x = 3, y = 3, z = 3

**Note about the assignment operator (=)**

The assignment operator has 2 operands, the left operand must be identifier not a constant or equation.

Example:

```
x = 10; ✔
x = y * 3; ✔
10 = x; ✘
4 = x * 3; ✘
y + 4 = x * 3; ✘
x = y = z; ✔
x = y + 4 = z; ✘
```

**Working with ++ and --**

What happens when you have x++;

This post increment and it is the only operator in the expression so it will run the same as ++x;

**But what will happen when you have**

```
y = x++;     (post increment)

suppose x = 4;

then it will do the following:
     1)- y = x;
         y becomes 4

     2)- x++
         x becomes 5

finally x = 5 and y = 4
```

**What will happen when you have**

```
y = ++x;     (pre increment)

suppose x = 4;

then it will do the following:
     1)- ++x
          x becomes 5

     1)- y = x;
          y becomes 5


finally x = 5 and y = 5
```

Example for ++ and --

Suppose t = 4, f = 2, d = 10, y = 3

x = y * t++ - 4 * ++f / 2 + --d

1)- ++f

f = f + 1

f = 2 + 1 = 3

x = y * t++ - 4 * 3 / 2 + --d

2)- --d

d = d - 1

d = 10 - 1 = 9

x = y * t++ - 4 * 3 / 2 + 9

3)- y * t = 3 * 4 = 12 it will do ++ with t at the end

x = 12 - 4 * 3 / 2 + 9

4)- 4 * 3 = 12

x = 12 - 12 / 2 + 9

5)- 12 / 2 = 6

x = 12 – 6 + 9

6)- 12 – 6 = 6

x = 6 + 9

7)- x = 15

8)- Now you add 1 to t (t++), t will be 5

9)- Final values : x = 15 , y = 3, t = 5, d = 9, f = 3

**Example about casting:**

Suppose we have the following
int x = 7;
int y = 2;
int z;

z = x/y;

(**int / int** = int, it will truncate the decimal value, the actual value is 3.5, but it will make it 3 with out rounding.

z = 7/2 = 3

So z will have the value of 3.

suppose we have:
double r =7.0;
double s = 2.0;
double t;

t = x / y;
x / y = 3;
It will store the value as double 3.0 because t is double.
t == 3.0

t = r/s;
double / double = double
7.0 / 2.0 = 3.5;
t = 3.5

z = r/s;
This is will give **an error**. You can not store double value into int variable.

If we have two different types, then the result will be double
t = x/s

7/2.0 = 3.5
**be careful** if you do the following it will give compile error:
z = x/s; because you can not store double into int variable.

**Explicit casting: تحويل صريح**

Suppose you want to have double as a result when you do x/y (7/2) as int/int, you can do casting as following:
t = (double) x / y;
or
t = x / (double) y;

if you do t = (double) (x/y);
You got 3.0, because it will calculate first x/y (int/int) you got 3, then it will convert it to 3.0

**You can cast from double to int, but you have to be careful.**