JAVA 연산자

프로그램은 사람이 이해하는 코드를 작성.

느려도 꾸준하면 경기에서 이긴다.

작성자 : 홍효상
이메일 : hyomee@naver.com
소스 : https://github.com/hyomee/JAVA_EDU

Content

3. 연산자

1. 연산자

"식을 구성하는 기본 단위"

연산자 (Operator)

- 연산 결과 : 산술, 증감, 비트,시프트 연산자
- 연산 결과가 참/거짓 : 비교, 논리 연산자
- 연산 결과가 아닌 대입 : 대입, 삼항 연산자

연산자 (Operator)	연산기호	기능	결과
산술 연산자(arithmetic operator)	+, -, *, /, %	사칙연산, 나머지 연산 모두 두 개의 피연산자를 가지는 이항 연산자 결합 방향은 왼쪽에서 오른쪽	값
증감 연산자(increment and decrement operators)	++, --	값이 1 씩 증가 및 감소	값
비트 연산자(bitwise operator)	&. |, ~, ^	비트 AND, OR, NOT, XOR	값
시프트 연산자(shift operator)	>>, <<. >>>	비트 단위로 이동	값
비교 연산자(comparison operator)	<, >, <=, >=, ==, !=	값의 비교	참/거짓
논리 연산자(logical operator)	&&, ||, !, ^	논리 AND, OR, NOT, XOR	참/거짓
대입 연산자(assignment operator)	=, +=, -=, *=, /=, &=, |=, >>=, <<=, >>>=	산술연산 결과를 대입 (축약)	실행
삼항 연산자(ternary operator)	(조건) ? 참실행 : 거짓실행	조건이 참이면 참실행, 거짓이면 거짓 실행	실행
instanceof 연산자 (instanceof operator)	instanceof	객체가 어떤 클래스인지, 어떤 클래스를 상속받았는지 확인하는데 사용하는 연산자	참/거짓

2. 연산자 우선 순위

연산자 우선 순위

- 연산자의 우선순위는 수식 내에 여러 연산자가 함께 등장할 때, 어느 연산자가 먼저 처리될 것인가를 결정

위	연산자	설명	결합 방향
1	[]	첨자 연산자	왼쪽에서 오른쪽으로
	.	멤버 연산자	왼쪽에서 오른쪽으로
2	++	후위 증가 연산자	왼쪽에서 오른쪽으로
	--	후위 감소 연산자	왼쪽에서 오른쪽으로
3	!	논리 NOT 연산자	오른쪽에서 왼쪽으로
	~	비트 NOT 연산자	오른쪽에서 왼쪽으로
	+	양의 부호 (단항 연산자)	오른쪽에서 왼쪽으로
	-	음의 부호 (단항 연산자)	오른쪽에서 왼쪽으로
	++	전위 증가 연산자	오른쪽에서 왼쪽으로
	--	전위 감소 연산자	오른쪽에서 왼쪽으로
	(타입)	타입 캐스트 연산자	오른쪽에서 왼쪽으로
4	*	곱셈 연산자	왼쪽에서 오른쪽으로
	/	나눗셈 연산자	왼쪽에서 오른쪽으로
	%	나머지 연산자	왼쪽에서 오른쪽으로
5	+	덧셈 연산자 (이항 연산자)	왼쪽에서 오른쪽으로
	-	뺄셈 연산자 (이항 연산자)	왼쪽에서 오른쪽으로
6	<<	비트 왼쪽 시프트 연산자	왼쪽에서 오른쪽으로

위	연산자	설명	결합 방향
6	>>	부호 비트를 확장하면서 비트 오른쪽 시프트	왼쪽에서 오른쪽으로
	>>>	부호 비트까지 모두 비트 오른쪽 시프트	왼쪽에서 오른쪽으로
7	<	관계 연산자(보다 작은)	왼쪽에서 오른쪽으로
	<=	관계 연산자(보다 작거나 같은)	왼쪽에서 오른쪽으로
	>	관계 연산자(보다 큰)	왼쪽에서 오른쪽으로
	>=	관계 연산자(보다 크거나 같은)	왼쪽에서 오른쪽으로
	instanceof	인스턴스의 실제 타입 반환	왼쪽에서 오른쪽으로
8	==	관계 연산자(와 같은)	왼쪽에서 오른쪽으로
	!=	관계 연산자(와 같지 않은)	왼쪽에서 오른쪽으로
9	&	비트 AND 연산자	왼쪽에서 오른쪽으로
10	^	비트 XOR 연산자	왼쪽에서 오른쪽으로
11	|	비트 OR 연산자	왼쪽에서 오른쪽으로
12	&&	논리 AND 연산자	왼쪽에서 오른쪽으로
13	||	논리 OR 연산자	왼쪽에서 오른쪽으로
14	? :	삼항 조건 연산자	오른쪽에서 왼쪽으로
15	=	대입 연산자 및 복합 대입 연산자 (=, +=, -=, *=, /=, %=, <<=, >>=, >>>=, &=, ^=, |=)	오른쪽에서 왼쪽으로

3. 대입 연산자

대입 연산자

- 오른쪽 피연산자의 연산 결과를 왼쪽 변수에 대입
- 산술 연산자, 비트연산자, 쉬프트 연산자와 합쳐서 축약으로 표현 할 수 있음


```java
public class AssignmentOperator {
    public static void main(String[] args) {
        int numA = 10;
        int numB = 3;

        numA += numB;
        System.out.println(String.format("numA += numB   : %s", numA));
        numA %= numB;
        System.out.println(String.format("numA %%= numB   : %s", numA));
    }
}
```

```
numA += numB   : 13
numA %= numB   : 1
```

일반 표현	축약 표현
a = a + b	a += b
a = a - b	a -= b
a = a * b	a *= b
a = a / b	a /= b
a = a % b	a %= b
a = a & b	a &= b
a = a | b	a |= b
a = a >> b	a >>= b
a = a << b	a <<= b
a = a >>> b	a >>>= b
a = a <<< b	a <<<= b

4. 산술연산자

산술 연산자

- 사칙 연산 (+, - ,* , /) 와 나머지 연산 (%)
- 나누기 연산(/)은 몫에 대한 결과, 나머지를 구하기 위해서는 나머지 연산 (%)를 사용 함 (모듈로 연산)

```java
public class ArithmeticOperator {
    public static void main(String[] args) {
        int numA = 10;
        int numB = 3;

        System.out.println("numA + numB = " + (numA + numB) );
        System.out.println("numA - numB = " + (numA - numB) );
        System.out.println("numA * numB = " + (numA * numB) );
        System.out.println("numA / numB = " + (numA / numB) );
        System.out.println("numA % numB = " + (numA % numB) );

        double doubleA = 10.27;
        double doubleB = 3.5;

        System.out.println("doubleA + doubleB = " + (doubleA + doubleB) );
        System.out.println("doubleA - doubleB = " + (doubleA - doubleB) );
        System.out.println("doubleA * doubleB = " + (doubleA * doubleB) );
        System.out.println("doubleA / doubleB = " + (doubleA / doubleB) );
        System.out.println("doubleA % doubleB = " + (doubleA % doubleB) );
    }
}
```

```
numA + numB = 13
numA - numB = 7
numA * numB = 30
numA / numB = 3
numA % numB = 1
doubleA + doubleB = 13.77
doubleA - doubleB = 6.77
doubleA * doubleB = 35.945
doubleA / doubleB = 2.934285714285714
doubleA % doubleB = 3.2699999999999996
```

5. 증감연산자

증감 연산자

- 변수 값을 1씩 증가, 1씩 감소
- 전위형 (++ 변수) : 변수에 1증가 후 다른 변수에 대입, 후위형(변수 ++) 다른 변수에 대입 이후 변수에 1증가

```java
public class IncrementOperators {
    public static void main(String[] args) {
        int num = 10;
        int postfixIncrementNum = num ++;
        System.out.println("num = " + num);
        System.out.println("postfixIncrementNum = " + postfixIncrementNum);

        int num01 = 10;
        int prefixIncrementNum = ++num01;
        System.out.println("num01 = " + num01);
        System.out.println("prefixIncrementNum = " + prefixIncrementNum);
    }
}
```

```
num = 11
postfixIncrementNum = 10
num01 = 11
prefixIncrementNum = 11
```

➢ **전위형 (Postfix)**

변수A | = | 변수B ++

변수 A = 변수 B;
변수 B = 변수 B + 1;

➢ **후위형 (Prefix)**

변수A | = | ++ 변수B

변수 B = 변수 B + 1;
변수 B = 변수 B;

6. 비트연산자

비트 연산자

- 비트 단위로 연산 처리 1 or 0

A	B	AND (&)	OR (|)	XOR (^)
0	0	0	1	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	0

A	NOT(~)
0	1
1	0

```
int numA = 3;
int numB = 2;

int numC = numA & numB;
System.out.println(String.format("%s & %s = %s",
      Integer.toBinaryString(numA),
      Integer.toBinaryString(numB),
      Integer.toBinaryString(numC)));

System.out.println(String.format("%s & %s = %s", numA, numB,  numC));
```

```
11 & 10 = 10
3 & 2 = 2
```

➢ 10진수 -> 진법 변환

```
Integer.toBinaryString(10);
Integer.toOctalString(10);
Integer.toHexString(10);
```

➢ 진법 -> 10진수 변환

```
Integer.parseInt(Integer.toBinaryString(10),2);
Integer.parseInt(Integer.toOctalString(10),8);
Integer.parseInt(Integer.toHexString(10),16);
```

7. 시프트 연산자

시프트 연산자

- 비트의 위치를 좌우로 이동 하는 연산
- 산술 시프트 (>>) : 부호비트는 유지 하면서 이동 : 2의 배수
- 1Byte 기준으로 시프트 할 때 넘어 가면 삭제, 들어 오면 0으로 채워 짐

2의 배수

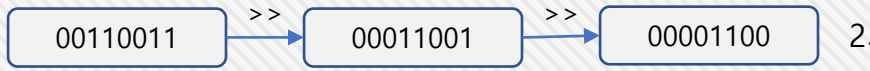
```
int flags = 0b00110011;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flags), Integer.toBinaryString(flags)));
flags = flags << 1;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flags), Integer.toBinaryString(flags)));
flags = flags << 1;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flags), Integer.toBinaryString(flags)));

int flagsNegative = -0b00110011;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flagsNegative), Integer.toBinaryString(flagsNegative)));
flagsNegative = flagsNegative << 1;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flagsNegative), Integer.toBinaryString(flagsNegative)));
flagsNegative = flagsNegative << 1;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flagsNegative), Integer.toBinaryString(flagsNegative)));
```

```
십진수 : 51, 이진수 : 110011
십진수 : 102, 이진수 : 1100110
십진수 : 204, 이진수 : 11001100
십진수 : -51, 이진수 : 11111111111111111111111111001101
십진수 : -102, 이진수 : 11111111111111111111111110011010
십진수 : -204, 이진수 : 11111111111111111111111100110100
```

7. 시프트 연산자

시프트 연산자

- 비트의 위치를 좌우로 이동 하는 연산
- 산술 시프트 (<<) : 부호비트는 유지 하면서 이동 : 2로 나눔
- 1Byte 기준으로 시프트 할 때 넘어 가면 삭제, 들어 오면 0으로 채워 짐



2의 나눔

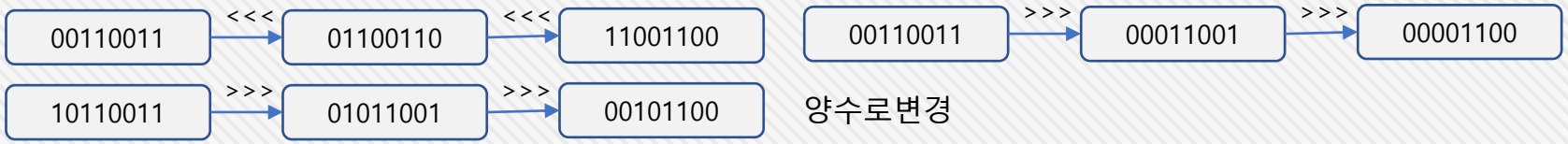
```
int flagsRight = 0b00110011;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flagsRight), Integer.toBinaryString(flagsRight)));
flagsRight = flagsRight >> 1;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flagsRight), Integer.toBinaryString(flagsRight)));
flagsRight = flagsRight >> 1;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flagsRight), Integer.toBinaryString(flagsRight)));

int flagsRightNegative = -0b00110011;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flagsRightNegative), Integer.toBinaryString(flagsRightNegative)));
flagsRightNegative = flagsRightNegative << 1;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flagsRightNegative), Integer.toBinaryString(flagsRightNegative)));
flagsRightNegative = flagsRightNegative << 1;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flagsRightNegative), Integer.toBinaryString(flagsRightNegative)));
```

```
십진수 : 51, 이진수 : 110011
십진수 : 25, 이진수 : 11001
십진수 : 12, 이진수 : 1100
십진수 : -51, 이진수 : 11111111111111111111111111001101
십진수 : -102, 이진수 : 11111111111111111111111110011010
십진수 : -204, 이진수 : 11111111111111111111111100110100
```

7. 시프트 연산자

시프트 연산자

- 비트의 위치를 좌우로 이동 하는 연산
- 논리 시프트 (>>>) : 부호비트를 포함 이동 : 음수인 경우 논리 시프트 이후 양수
- 1Byte 기준으로 시프트 할 때 넘어 가면 삭제, 들어 오면 0으로 채워 짐




```
int flagsLogic = 0b00110011;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flagsLogic), Integer.toBinaryString(flagsLogic)));
flagsLogic = flagsLogic >>> 1;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flagsLogic), Integer.toBinaryString(flagsLogic)));
flagsLogic = flagsLogic >>> 1;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flagsLogic), Integer.toBinaryString(flagsLogic)));

int flagsLogicNegative = -0b00110011;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flagsLogicNegative), Integer.toBinaryString(flagsLogicNegative)));
flagsLogicNegative = flagsLogicNegative >>> 1;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flagsLogicNegative), Integer.toBinaryString(flagsLogicNegative)));
flagsLogicNegative = flagsLogicNegative >>> 1;
System.out.println(String.format("십진수 : %s, 이진수 : %s",
      Integer.valueOf(flagsLogicNegative), Integer.toBinaryString(flagsLogicNegative)));
```

```
십진수 : 51, 이진수 : 110011
십진수 : 25, 이진수 : 11001
십진수 : 12, 이진수 : 1100
십진수 : -51, 이진수 : 11111111111111111111111111001101
십진수 : 2147483622, 이진수 : 11111111111111111111111100110
십진수 : 1073741811, 이진수 : 11111111111111111111111110011
```

8. 비교 연산자

비교 연산자

- 크기 비교 (>, < , <=, >=)와 등가 비교 (==, !=)
- Stack에 있는 값을 비교 함

```
public class ComparisonOperators {
   public static void main(String[] args) {
      int numA = 10;
      int numB = 20;

      System.out.println(String.format("%s > %s : %s", numA, numB, (numA > numB)));
      System.out.println(String.format("%s < %s : %s", numA, numB, (numA < numB)));
      System.out.println(String.format("%s >= %s : %s", numA, numB, (numA < numB)));
      System.out.println(String.format("%s <= %s : %s", numA, numB, (numA < numB)));

      System.out.println(String.format("%s == %s : %s", numA, numB, (numA == numB)));
      System.out.println(String.format("%s != %s : %s", numA, numB, (numA != numB)));

      String str01 = new String("안녕");
      String str02 = new String("안녕");

      System.out.println(String.format("%s == %s : %s", str01, str02, (str01 == str02)));
      System.out.println(String.format("%s equals %s : %s", str01, str02, (str01.equals(str02))));

      System.out.println(String.format("%s = %s : %s", numA, numB, (numA = numB)));
   }
}
```

```
10 > 20 : false
10 < 20 : true
10 >= 20 : true
10 <= 20 : true
10 == 20 : false
10 != 20 : true
안녕 == 안녕 : false
안녕 equals 안녕 : true
10 = 20 : 20
```

Stack Memory 값 비교

대입 연산자 -> 실행

9. 논리 연산자

논리 연산자

- 피연산자로 boolean (true/false) 만 올 수 있고 결과는 boolean type 이다.
- 쇼트 서킷 (short circuit) : 연산 결과가 확정이 되면 나머지 연산을 하지 않는 것 (논리 연산자 적용, 비트 연산자 미적용)

A	B	AND (&&)	OR (||)	XOR (^)
false	false	false	true	false
false	true	false	true	true
true	false	false	true	true
true	true	true	false	false

A	NOT(!)
0	true
1	false

short circuit

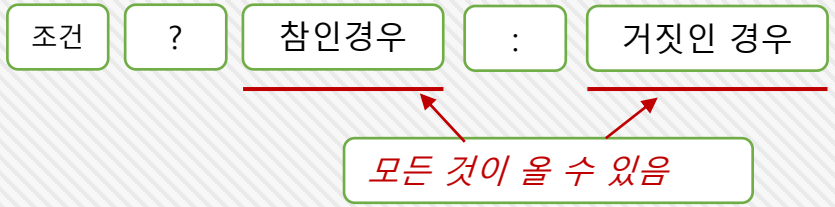
```
public class LogicalOperator {
   public static void main(String[] args) {
      int numA = 5;  // 7
      int numB = 6;

      System.out.println(String.format("(%s > %s) && (%s < %s) : %s",
            numA, numB, numA, numB, (numA > numB) && (numA < numB)));

      System.out.println(String.format("(%s > %s) || (%s < %s) : %s",
            numA, numB, numA, numB, (numA > numB) || (numA < numB)));

      System.out.println(String.format("(%s > %s) ^ (%s < %s) : %s",
            numA, numB, numA, numB, (numA > numB) ^ (numA < numB)));

      System.out.println(String.format("!(%s > %s) : %s",
            numA, numB, !(numA > numB) ));

      System.out.println(String.format("(%s > %s) && ++numA(%s) <= %s : %s",
            numA, numB, ++numA, numB, (numA > numB) && (++numA >= numB)));
   }
}
```

```
(5 > 6) && (5 < 6) : false
(5 > 6) || (5 < 6) : true
(5 > 6) ^ (5 < 6) : true
!(5 > 6) : true
(5 > 6) && ++numA(6) <= 6 : false
```

10. 삼항 연산자

삼항 연산자

- 3개의 피연산자로 되어 있으며 조건에 결과 처리




```java
public class TernaryOperator {
   public static void main(String[] args) {
      int value = ( 5 > 4) ? 3 : 4;
      System.out.println(String.format("( 5 > 4) ? 3 : 4  =>  %s", value));

      int numA = 10;
      int numB = 3;
      boolean bl = numA > numB ;
      System.out.println(String.format("%s > %s :  =>  %s", numA, numB, bl));

      value = numA > numB ? ++numB : numB;
      System.out.println(String.format("%s > %s ? ++numB(%s) : %s  =>  %s",
            numA, numB, ++numB, numB, value));

      value = numA > numB ? addNumB(numB) : numB;
      System.out.println(String.format("%s > %s ? ++numB(%s) : %s  =>  %s",
            numA, numB, ++numB, numB, value));
   }

   public static int addNumB(int numB) {
      return ++numB;
   }
}
```

```
( 5 > 4) ? 3 : 4  =>  3
10 > 3 :  =>  true
10 > 4 ? ++numB(5) : 5  =>  4
10 > 5 ? ++numB(6) : 6  =>  6
```

11. instanceof 연산자

instanceof 연산자

- 3연산자는 객체가 어떤 클래스인지, 어떤 클래스를 상속받았는지 확인하는데 사용하는 연산자

```
public class InstanceofOperator {
   public static void main(String[] args) {
      ArrayList list = new ArrayList();

      System.out.println(list instanceof ArrayList);
      System.out.println(list instanceof List);
      System.out.println(list instanceof Set);
   }
}
```

```
true
true
false
```