연산자
(Operator)

연산자

- 연산자(Operator)
  - +, -, *, /, <, > 등등…
- 피연산자(Operand)
  - 연산자의 기능(동작) 적용 대상(변수, 상수…)

산술연산자 / 대입연산자

연산자	연산자의 기능	결합방향
=	연산자 오른쪽에 있는 값을 연산자 왼쪽에 있는 변수에 대입한다. 예) val = 20;	⬅
+	두 피연산자의 값을 더한다. 예) val = 4 + 3;	➡
-	왼쪽의 피연산자 값에서 오른쪽의 피연산자 값을 뺀다. 예) val = 4 - 3;	➡
*	두 피연산자의 값을 곱한다. 예) val = 4 * 3;	➡
/	왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눈다. 예) val = 7 / 3;	➡
%	왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눴을 때 얻게 되는 나머지를 반환한다. 예) val = 7 % 3	➡

복합대입연산자

&=, ^=, |=, <<=, >>=, >>>=

비교(관계)연산자

연산자	연산자의 기능	결합방향
〈	예) n1 〈 n2 n1이 n2보다 작은가?	➡
〉	예) n1 〉 n2 n1이 n2보다 큰가?	➡
〈=	예) n1 〈= n2 n1이 n2보다 같거나 작은가?	➡
〉=	예) n1 〉= n2 n1이 n2보다 같거나 큰가?	➡
==	예) n1 == n2 n1과 n2가 같은가?	➡
!=	예) n1 != n2 n1과 n2가 다른가?	➡

논리연산자

연산자	연산자의 기능	결합방향
&&	예) A && B A와 B 모두 true이면 연산결과는 true (논리 AND)	➡
||	예) A || B A와 B 둘 중 하나라도 true이면 연산결과는 true (논리 OR)	➡
!	예) !A 연산결과는 A가 true이면 false, A가 false이면 true (논리 NOT)	⬅

연산자 우선 순위

연산기호	결합방향	우선순위
[ ], .	➡	1(높음)
expr++, expr--	⬅	2
++expr, -- expr, +expr, -expr, ~, !, (type)	⬅	3
*, /, %	➡	4
+, -	➡	5
〈〈, 〉〉, 〉〉〉	➡	6
〈, 〉, 〈=, 〉=, instanceof	➡	7
==, !=	➡	8
&	➡	9
^	➡	10
|	➡	11
&&	➡	12
||	➡	13
? expr : expr	⬅	14
=, +=, -=, *=, /=, %=, &=, ^=, |=, 〈〈=, 〉〉=, 〉〉〉=	⬅	15(낮음)

Short-Circuit Evaluation

```
1.  public class SCE {
2.      public static void main(String[] args) {
3.          int num1 = 0, num2 = 0;
4.          boolean result;
5.
6.          result = (num1 += 10) < 0 && (num2 += 10) > 0;
7.          System.out.println("result = " + result);
8.          System.out.println("num1 = " + num1 + ", num2 = " + num2);
9.
10.         result = (num1 += 10) > 0 || (num2 += 10) > 0;
11.         System.out.println("result = " + result);
12.         System.out.println("num1 = " + num1 + ", num2 = " + num2);
13.     }
14. }
```

증감연산자

연산자	연산자의 기능	결합방향
++ (prefix)	피연산자에 저장된 값을 1 증가 예) val = ++n;	←
-- (prefix)	피연산자에 저장된 값을 1 감소 예) val = --n;	←

연산자	연산자의 기능	결합방향
++ (postfix)	피연산자에 저장된 값을 1 증가 예) val = n++;	←
-- (postfix)	피연산자에 저장된 값을 1 감소 예) val = n--;	←

Exam.

```
1.  public class Prefix {
2.      public static void main(String[] args) {
3.          int num1 = 7;
4.          int num2, num3;
5.
6.          num2 = ++num1; // num1++;
7.          num3 = --num1; // num1--;
8.
9.          System.out.println(num1);
10.         System.out.println(num2);
11.         System.out.println(num3);
12.     }
13. }
```

비트연산자

연산자	연산자의 기능	결합방향
&	비트단위로 AND 연산을 한다. 예) n1 & n2;	➡
|	비트단위로 OR 연산을 한다. 예) n1 | n2;	➡
^	비트단위로 XOR 연산을 한다. 예) n1 ^ n2;	➡
~	피연산자의 모든 비트를 반전시켜서 얻은 결과를 반환 예) ~n;	⬅

비트시프트연산자

연산자	연산자의 기능	결합방향
〈〈	• 피연산자의 비트 열을 왼쪽으로 이동 • 이동에 따른 빈 공간은 0으로 채움 • 예) n 〈〈 2; → n의 비트 열을 두 칸 왼쪽으로 이동 시킨 결과 반환	➡
〉〉	• 피연산자의 비트 열을 오른쪽으로 이동 • 이동에 따른 빈 공간은 음수의 경우 1, 양수의 경우 0으로 채움 • 예) n 〉〉 2; → n의 비트 열을 두 칸 오른쪽으로 이동 시킨 결과 반환	➡
〉〉〉	• 피연산자의 비트 열을 오른쪽으로 이동 • 이동에 따른 빈 공간은 0으로 채움 • 예) n 〉〉〉 2; → n의 비트 열을 두 칸 오른쪽으로 이동 시킨 결과 반환	➡

Exam.

```
1.  public class BitShift {
2.      public static void main(String[] args) {
3.          System.out.println(2 << 1);
4.          System.out.println(2 << 2);
5.
6.          System.out.println(8 >> 1);
7.          System.out.println(8 >> 2);
8.
9.          System.out.println(-8 >> 1);
10.         System.out.println(-8 >> 2);
11.
12.         System.out.println(-8 >>> 1);
13.     }
14. }
```