



3. 자바의 연산자

1. 기본 연산
2. 비트 연산자

항과 연산자

항(**operand**) : 연산에 사용되는 값

연산자(**operator**) : 항을 이용하여 연산하는 기호

항의 개수에 따른 연산자 구분

연산자	설명	연산 예
단항 연산자	항이 한 개인 연산자	<code>++num</code>
이항 연산자	항이 두 개인 연산자	<code>num1 + num2;</code>
삼항 연산자	항이 세 개인 연산자	<code>(5 > 3) ? 1 : 0;</code>

대입 연산자

변수에 값을 대입 하는 연산자

연산의 결과를 변수에 대입


우선 순위가 가장 낮은 연산자

왼쪽 변수(lvalue)에 오른쪽 변수(값) (rvalue)를 대입

```
int age = 24; //나이를 의미하는 age 변수에 값 24를 대입함
```



```
totalScore = mathScore + engScore; //① mathScore 값과 영어 점수 engScore 값을 더하여  
                                     //② 총점을 의미하는 totalScore 변수에 수학 점수에 대입함
```



부호 연산자

단항 연산자

양수/음수의 표현, 값의 부호를 변경

연산자	기능	연산 예
+	변수나 상수 값을 양수로 만듭니다.	+3
-	변수나 상수 값을 음수로 만듭니다.	-3

변수에 +, - 를 사용한다고 해서 변수의 값이 변하는 것은 아님

변수의 값을 변경하려면 대입연산자를 사용해야 함

```
int num = 10;

System.out.println(+num); //값 10이 그대로 출력됨
System.out.println(-num); //값 10에 -가 붙어서 -10이 출력되지만 num 값이 실제로 바뀌지는 않음
System.out.println(num)   //값 10이 그대로 출력됨

num = -num;                //num 값을 음수로 바꿔서 다시 num에 대입함
System.out.println(num);   //값 -10이 출력됨
```

산술 연산자

사칙연산에 사용되는 연산자

연산자	기능	연산 예
+	두 항을 더합니다.	$5 + 3$
-	앞에 있는 항에서 뒤에 있는 항을 뺍니다.	$5 - 3$
*	두 항을 곱합니다.	$5 * 3$
/	앞에 있는 항에서 뒤에 있는 항을 나누어 몫을 구합니다.	$5 / 3$
%	앞에 있는 항에서 뒤에 있는 항을 나누어 나머지를 구합니다.	$5 \% 3$

%는 나머지를 구하는 연산자

⇒ 숫자 n 의 나머지는 $0 \sim n-1$ 범위의 수

⇒ 특정 범위 안의 수를 구할 때 종종 사용

증가 감소 연산자

단항 연산자

1만큼 더하거나 1만큼 뺄 때 사용하는 연산자

항의 앞/뒤 위치에 따라 연산의 결과가 달라짐에 유의

연산자	기능	연산 예
++	항의 값에 1을 더합니다.	<code>val = ++num;</code> // 먼저 num 값이 1 증가한 후 val 변수에 대입 <code>val = num++;</code> // val 변수에 기존 num 값을 먼저 대입한 후 num 값 1 증가
--	항의 값에서 1을 뺍니다.	<code>val = --num;</code> // 먼저 num 값이 1 감소한 후 val 변수에 대입 <code>val = num--;</code> // val 변수에 기존 num 값을 먼저 대입한 후 num 값 1 감소

관계 연산자

이항 연산자

연산의 결과가 **true(참)**, **false(거짓)**으로 반환 됨

연산자	기능	연산 예
>	왼쪽 항이 크면 참을, 아니면 거짓을 반환합니다.	num > 3;
<	왼쪽 항이 작으면 참, 아니면 거짓을 반환합니다.	num < 3;
>=	왼쪽 항이 오른쪽 항보다 크거나 같으면 참, 아니면 거짓을 반환합니다.	num >= 3;
<=	왼쪽 항이 오른쪽 항보다 작거나 같으면 참, 아니면 거짓을 반환합니다.	num <= 3;
==	두 개 항의 값이 같으면 참, 아니면 거짓을 반환합니다.	num == 3;
!=	두 개 항이 다르면 참, 아니면 거짓을 반환합니다.	num != 3;

논리 연산자

관계 연산자와 혼합하여 많이 사용 됨

연산의 결과가 **true(참)**, **false(거짓)**으로 반환 됨

연산자	기능	연산 예
&& (논리 곱)	두 항이 모두 참인 경우에만 결과 값이 참입니다. 그렇지 않은 경우는 거짓입니다.	booleanval = (5 > 3) && (5 > 2);
 (논리 합)	두 항 중 하나의 항만 참이면 결과 값은 참입니다. 두 항이 모두 거짓이면 결과 값은 거짓입니다.	booleanval = (5 > 3) (5 < 2);
! (부정)	단항 연산자입니다. 참인 경우는 거짓으로 바꾸고, 거짓인 경우는 참으로 바꿉니다.	booleanval = !(5 > 3);

단락 회로 평가 (short circuit evaluation)

논리 곱(&&)은 두 항이 모두 **true** 일 때만 결과가 **true**

=> 앞의 항이 **false** 이면 뒤 항의 결과를 평가하지 않아도 **false** 임

논리 합(||)은 두 항이 모두 **false** 일 때만 결과가 **false**

=> 앞의 항의 **true** 이면 뒤 항의 결과를 평가하지 않아도 **true** 임

단락 회로 평가 실습

```
package operator;
```

```
public class OperationEx3 {
```

```
    public static void main(String[ ] args) {
```

```
        int num1 = 10;
```

```
        int i = 2;
```

```
        boolean value = ((num1 = num1 + 10) < 10) && ((i = i + 2) < 10);
```

```
        System.out.println(value);
```

```
        System.out.println(num1);
```

```
        System.out.println(i);
```

논리 곱에서 앞 항의 결과 값이 거짓이므로
이 문장은 실행되지 않음

```
        value = ((num1 = num1 + 10) > 10) || ((i = i + 2) < 10);
```

```
        System.out.println(value);
```

```
        System.out.println(num1);
```

```
        System.out.println(i);
```

논리 합에서 앞 항의 결과 값이 참이므로
이 문장은 실행되지 않음

```
    }
```

```
}
```

복합 대입 연산자

대입 연산자와 다른 연산자를 함께 사용함
프로그램에서 자주 사용하는 연산자

연산자	기능	연산 예
<code>+=</code>	두 항의 값을 더해서 왼쪽 항에 대입합니다.	<code>num1 += 2;</code> <code>num1 = num1 + 2;</code> 와 같음
<code>-=</code>	왼쪽 항에서 오른쪽 항을 빼서 그 값을 왼쪽 항에 대입합니다.	<code>num1 -= 2;</code> <code>num1 = num1 - 2;</code> 와 같음
<code>*=</code>	두 항의 값을 곱해서 왼쪽 항에 대입합니다.	<code>num1 *= 2;</code> <code>num1 = num1 * 2;</code> 와 같음
<code>/=</code>	왼쪽 항을 오른쪽 항으로 나누어 그 몫을 왼쪽 항에 대입합니다.	<code>num1 /= 2;</code> <code>num1 = num1 / 2;</code> 와 같음
<code>%=</code>	왼쪽 항을 오른쪽 항으로 나누어 그 나머지를 왼쪽 항에 대입합니다.	<code>num1 %= 2;</code> <code>num1 = num1 % 2;</code> 와 같음

복합 대입 연산자

<code><<=</code>	비트를 왼쪽으로 이동하고 그 값을 왼쪽 항에 대입합니다.	<code>num1 <<= 2;</code> <code>num1 = num1 << 2;</code> 와 같음
<code>>>=</code>	비트를 오른쪽으로 이동하고 그 값을 왼쪽 항에 대입합니다(왼쪽에 채워지는 비트 값은 부호 비트와 동일합니다).	<code>num1 >>= 2;</code> <code>num1 = num1 >> 2;</code> 와 같음
<code>>>>=</code>	비트를 오른쪽으로 이동하고 그 값을 왼쪽 항에 대입합니다(왼쪽에 채워지는 비트 값은 0입니다).	<code>num1 >>>= 2;</code> <code>num1 = num1 >>> 2;</code> 와 같음
<code>&=</code>	두 항의 <code>&</code> 비트 연산 후 그 값을 왼쪽 항에 대입합니다.	<code>num1 &= num2;</code> <code>num1 = num1 & num2;</code> 와 같음
<code> =</code>	두 항의 <code> </code> 비트 연산 후 그 값을 왼쪽 항에 대입합니다.	<code>num1 = num2;</code> <code>num1 = num1 num2;</code> 와 같음
<code>^=</code>	두 항의 <code>^</code> 비트 연산 후 그 값을 왼쪽 항에 대입합니다.	<code>num1 ^= num2;</code> <code>num1 = num1 ^ num2;</code> 와 같음

조건 연산자

삼항 연산자

조건 식의 결과가 **true(참)** 인 경우와 **false(거짓)** 인 경우에 따라
다른 식이나 결과가 수행됨

제어문 중 조건문을 간단히 표현할 때 사용할 수 있음

연산자	기능	연산 예
조건식 ? 결과1 : 결과2;	조건식이 참이면 결과1, 조건식이 거짓이면 결과2가 선택됩니다.	int num = (5 > 3) ? 10 : 20;

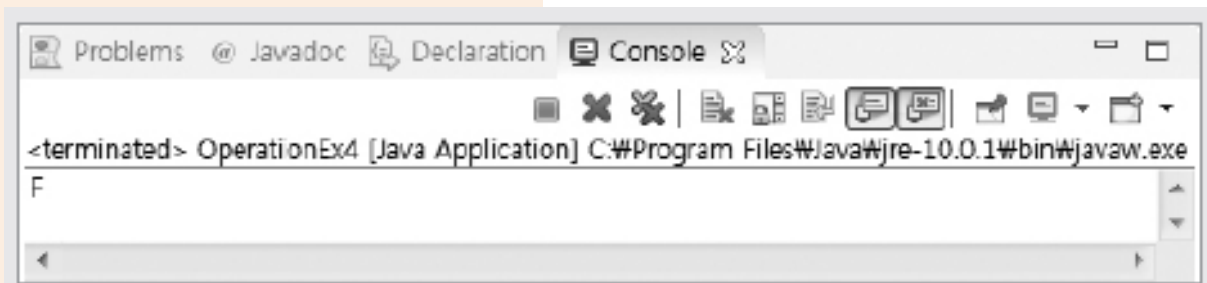
조건 연산자

```
package operator;

public class OperationEx4 {
    public static void main(String[ ] args) {
        int fatherAge = 45;
        int motherAge = 47;

        char ch;
        ch = (fatherAge > motherAge) ? 'T' : 'F';

        System.out.println(ch);
    }
}
```



비트 연산자

연산자	설명	예
~	비트의 반전 (1의 보수)	$a = \sim a;$
&	비트 단위 AND	1 & 1 1반환 그 외는 0
	비트 단위 OR	0 0 0반환 그 외는 1
^	비트 단위XOR	두개의 비트가 서로 다른 경우 에 1을 반환
<<	왼쪽 shift	$a \ll 2$ 변수 a를 2비트 만큼 왼 쪽으로 이동
>>	오른쪽 shift	$a \gg 2$ 변수 a를 2비트만큼 오른 쪽으로 이동
>>>	오른쪽 shift	>> 동일한 연산 채워지는 비트가 부호와 상관 없이 0 임

비트 연산자는 정수에만 사용할 수 있다

비트 연산자

&(AND) 연산자 : 두 비트가 모두 1인 경우만 1 아니면 0

```
int num1 = 5;  
int num2 = 10;  
int result = num1 & num2;
```



```
num1 : 0 0 0 0 0 1 0 1  
& num2 : 0 0 0 0 1 0 1 0  
result : 0 0 0 0 0 0 0 0
```

| (OR) 연산자 : 두 비트가 모두 0 인 경우만 0 아니면 1

```
int num1 = 5;  
int num2 = 10;  
int result = num1 | num2;
```



```
num1 : 0 0 0 0 0 1 0 1  
| num2 : 0 0 0 0 1 0 1 0  
result : 0 0 0 0 1 1 1 1
```


비트 연산자

^(XOR) 연산자: 두 비트가 다른 값이면 1, 같은 값이면 0

```
int num1 = 5;  
int num2 = 10;  
int result = num1 ^ num2;
```



num1 :	0	0	0	0	0	1	0	1
^ num2 :	0	0	0	0	1	0	1	0
<hr/>								
result :	0	0	0	0	1	1	1	1

~(반전) 연산자: 비트 값을 0은 1로 1은 0으로 바꾸는 연산자

```
int num = 10;  
int result = ~num;
```



num :	0	0	0	0	1	0	1	0
<hr/>								
~ num :	1	1	1	1	0	1	0	1

비트 연산자

<< (왼쪽 shift) : 비트를 왼쪽으로 이동하는 연산자

>> (오른쪽 shift) : 비트를 오른쪽으로 이동하는 연산자

<<<, >>> : shift 로 비트이동은 동일한데, 남은 공간을 무조건
부호비트가 아닌 0으로 채움

정수 15의 왼쪽 2자리 이동는 경우 ($15 \ll 2$)는 경우

00000000 00000000 00000000 00001111

00000000 00000000 00000000 0000111100

→ 결과 : 60

비트 연산자의 활용

마스크 : 특정 비트들은 가리고 몇 개의 비트들의 값만 사용할 때

비트켜기 : 특정 비트들만을 1로 설정해서 사용하고 싶을 때

예) $\& 00001111$ (하위 4비트 중 1인 비트만 꺼내기)

비트끄기 : 특정 비트들만을 0으로 설정해서 사용하고 싶을 때

예) $| 11110000$ (하위 4비트 중 0인 비트만 0으로 만들기)

비트 토글 : 모든 비트들을 0은 1로, 1은 0으로 바꾸고 싶을 때

연산자 우선 순위

우선순위	형	연산자	연산 방향
1	일차식	() [] .	→
2	단항	! ++ -- + -	←
3	산술	% /	→
4	산술	+ -	→
5	비트 이동	<< >>	→
6	관계	< > <= >=	→
7	관계	== !=	→
8	비트 곱	&	→
9	비트 차	^	→
10	비트 합		→
11	논리 곱	&&	→
12	논리 합		→
13	조건	? :	→
14	대입	= += -= *= %= /=	←