연산자

연산자(operator)

- 자바에서는 여러 종류의 연산을 수행하기 위한 다양한 기호.

연산자 종류

- 1. 산술 연산자(arithmetic operator)
- 2. 대입 연산자(assignment operator)
- 3. 증감 연산자(increment and decrement operators)
- 4. 비교 연산자(comparison operator)
- 5. 논리 연산자(logical operator)
- 6. 비트 연산자(bitwise operator)
- 7. 삼항 연산자(ternary operator)

산술 연산자(arithmetic operator)

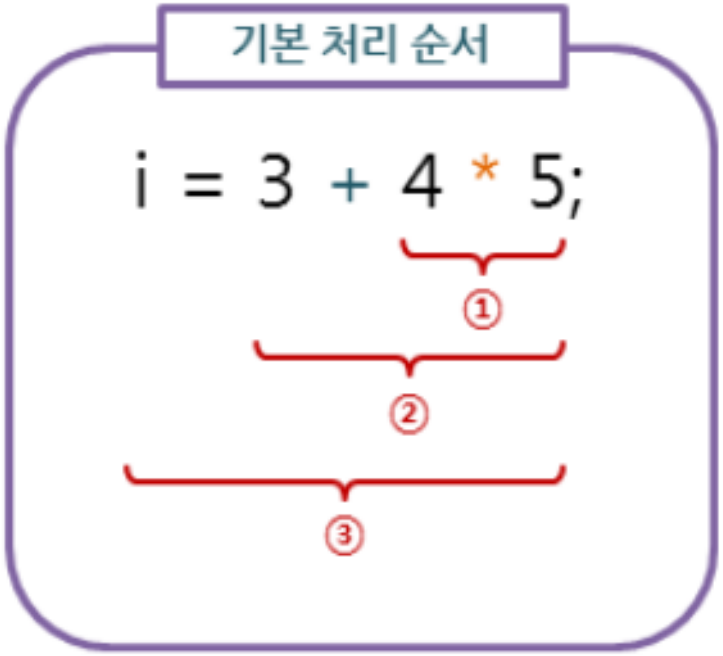
- 사칙연산을 다루는 연산자

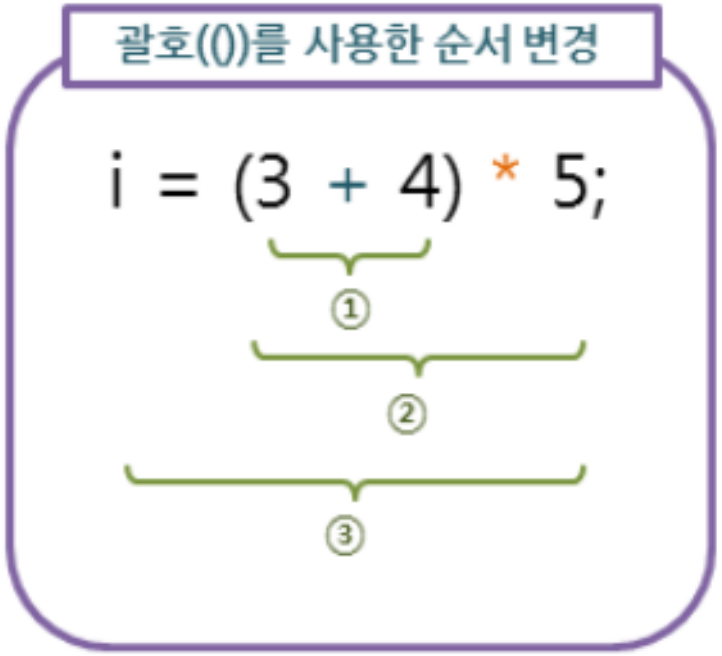
산술 연산자	설명
+	왼쪽의 피연산자에 오른쪽의 피연산자를 더함.
-	왼쪽의 피연산자에서 오른쪽의 피연산자를 뺌.
*	왼쪽의 피연산자에 오른쪽의 피연산자를 곱함.
/	왼쪽의 피연산자를 오른쪽의 피연산자로 나눔.
%	왼쪽의 피연산자를 오른쪽의 피연산자로 나눈 후, 그 나머지를 반환함.

연산자의 우선순위와 결합 방향

연산자의 우선순위는 수식 내에 여러 연산자가 함께 등장할 때, 어느 연산자가 먼저 처리될 것인가를 결정합니다.

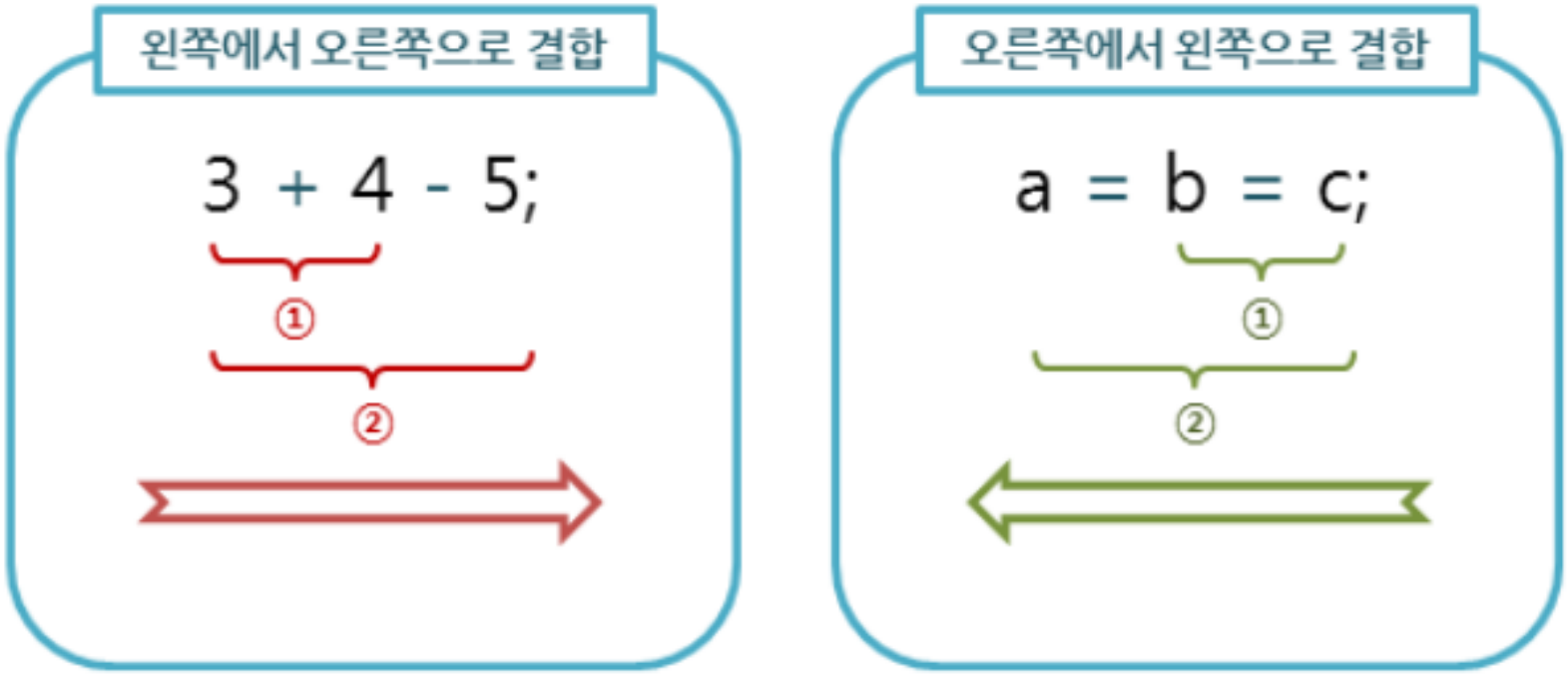
다음 그림은 가장 높은 우선순위를 가지고 있는 괄호(()) 연산자를 사용하여 연산자의 처리 순서를 변경하는 것을 보여줍니다.





연산자의 우선순위와 결합 방향

연산자의 결합 방향은 수식 내에 우선순위가 같은 연산자가 둘 이상 있을 때, 먼저 어느 연산을 수행할 것인가를 결정합니다.



대입 연산자

- 대입 연산자는 변수에 값을 대입할 때 사용하는 이항 연산자이며, 피연산자들의 결합 방향은 오른쪽에서 왼쪽입니다.

복합 대입 연산자

대입 연산자	설명
=	왼쪽의 피연산자에 오른쪽의 피연산자를 대입함.
+=	왼쪽의 피연산자에 오른쪽의 피연산자를 더한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
-=	왼쪽의 피연산자에서 오른쪽의 피연산자를 뺀 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
*=	왼쪽의 피연산자에 오른쪽의 피연산자를 곱한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
/=	왼쪽의 피연산자를 오른쪽의 피연산자로 나눈 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
%=	왼쪽의 피연산자를 오른쪽의 피연산자로 나눈 후, 그 나머지를 왼쪽의 피연산자에 대입함.
&=	왼쪽의 피연산자를 오른쪽의 피연산자와 비트 AND 연산한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
|=	왼쪽의 피연산자를 오른쪽의 피연산자와 비트 OR 연산한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
^=	왼쪽의 피연산자를 오른쪽의 피연산자와 비트 XOR 연산한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
<<=	왼쪽의 피연산자를 오른쪽의 피연산자만큼 왼쪽 시프트한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
>>=	왼쪽의 피연산자를 오른쪽의 피연산자만큼 부호를 유지하며 오른쪽 시프트한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
>>>=	왼쪽의 피연산자를 오른쪽의 피연산자만큼 부호에 상관없이 오른쪽 시프트한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.

증감 연산자

- 증감 연산자는 피연산자를 1씩 증가 혹은 감소시킬 때 사용하는 연산자입니다.
- 이 연산자는 피연산자가 단 하나뿐인 단항 연산자입니다.

증감 연산자 종류

증감 연산자	설명
++x	먼저 피연산자의 값을 1 증가시킨 후에 해당 연산을 진행함.
x++	먼저 해당 연산을 수행하고 나서, 피연산자의 값을 1 증가시킴.
--x	먼저 피연산자의 값을 1 감소시킨 후에 해당 연산을 진행함.
x--	먼저 해당 연산을 수행하고 나서, 피연산자의 값을 1 감소시킴.

증감 연산자의 연산 순서

예제

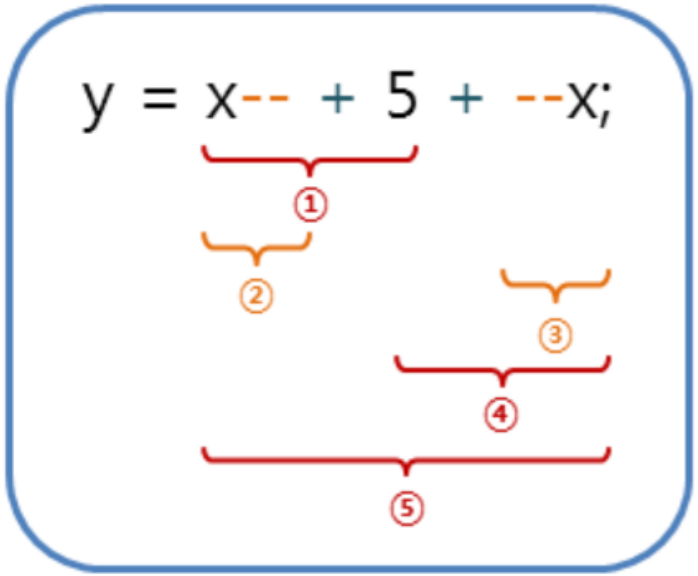
```
int x = 10;
int y = x-- + 5 + --x;


System.out.println("x : "+ x + ", y : " + y);
```

실행 결과

```
x : 8, y : 23
```

증감 연산자의 연산 순서

다음 그림은 위의 예제에서 수행되는 연산의 순서를 보여줍니다.



① : 첫 번째 감소 연산자(decrement operator)는 피연산자의 뒤쪽에 위치하므로, 덧셈 연산이 먼저 수행됩니다.

② : 덧셈 연산이 수행된 후에 감소 연산이 수행됩니다. (x의 값 : 9)

③ : 두 번째 감소 연산자는 피연산자의 앞쪽에 위치하므로, 덧셈 연산보다 먼저 수행됩니다. (x의 값 : 8)

④ : 감소 연산이 수행된 후에 덧셈 연산이 수행됩니다.

⑤ : 마지막으로 변수 y에 결괏값의 대입 연산이 수행됩니다. (y의 값 : 23)

비교 연산자

- 비교 연산자는 피연산자 사이의 상대적인 크기를 판단하는 연산자
- 비교 연산자는 왼쪽의 피연산자와 오른쪽의 피연산자를 비교하여, 어느 쪽이 더 큰지, 작은지, 또는 서로 같은지를 판단합니다.
- 비교 연산자는 모두 두 개의 피연산자를 가지는 이항 연산자이며, 피연산자들의 결합 방향은 왼쪽에서 오른쪽입니다.

비교 연산자

비교 연산자	설명
==	왼쪽의 피연산자와 오른쪽의 피연산자가 같으면 참을 반환함.
!=	왼쪽의 피연산자와 오른쪽의 피연산자가 같지 않으면 참을 반환함.
>	왼쪽의 피연산자가 오른쪽의 피연산자보다 크면 참을 반환함.
>=	왼쪽의 피연산자가 오른쪽의 피연산자보다 크거나 같으면 참을 반환함.
<	왼쪽의 피연산자가 오른쪽의 피연산자보다 작으면 참을 반환함.
<=	왼쪽의 피연산자가 오른쪽의 피연산자보다 작거나 같으면 참을 반환함.

논리 연산자

논리 연산자는 주어진 논리식을 판단하여, 참(true)과 거짓(false)을 결정하는 연산자입니다.
AND 연산과 OR 연산은 두 개의 피연산자를 가지는 이항 연산자이며, 피연산자들의 결합 방향은 왼쪽에서 오른쪽입니다.
NOT 연산자는 피연산자가 단 하나뿐인 단항 연산자이며, 피연산자의 결합 방향은 오른쪽에서 왼쪽입니다.

논리 연산자	설명
&&	논리식이 모두 참이면 참을 반환함. (논리 AND 연산)
||	논리식 중에서 하나라도 참이면 참을 반환함. (논리 OR 연산)
!	논리식의 결과가 참이면 거짓을, 거짓이면 참을 반환함. (논리 NOT 연산)

논리 연산자

다음은 논리 연산자의 모든 동작의 결과를 보여주는 진리표(truth table)입니다.

A	B	A && B	A || B	!A
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

비트 연산자

- 비트 연산자는 논리 연산자와 비슷하지만, 비트(bit) 단위로 논리 연산을 할 때 사용하는 연산자입니다.
- 또한, 비트 단위로 왼쪽이나 오른쪽으로 전체 비트를 이동하거나, 1의 보수를 만들 때도 사용됩니다.

비트 연산자

비트 연산자	설명
&	대응되는 비트가 모두 1이면 1을 반환함. (비트 AND 연산)
|	대응되는 비트 중에서 하나라도 1이면 1을 반환함. (비트 OR 연산)
^	대응되는 비트가 서로 다르면 1을 반환함. (비트 XOR 연산)
~	비트를 1이면 0으로, 0이면 1로 반전시킴. (비트 NOT 연산, 1의 보수)
<<	명시된 수만큼 비트들을 전부 왼쪽으로 이동시킴. (left shift 연산)
>>	부호를 유지하면서 지정한 수만큼 비트를 전부 오른쪽으로 이동시킴. (right shift 연산)
>>>	지정한 수만큼 비트를 전부 오른쪽으로 이동시키며, 새로운 비트는 전부 0이 됨.

비트 연산자

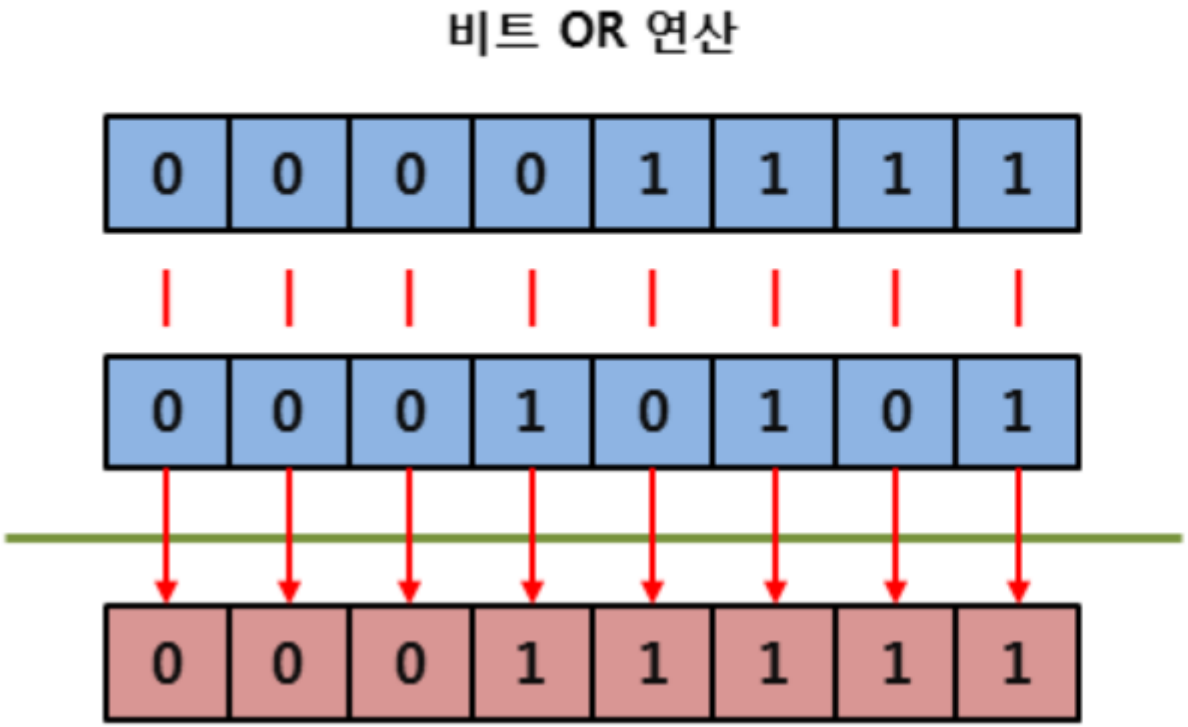
다음 그림은 비트 AND 연산자(&)의 동작을 나타냅니다.

이처럼 비트 AND 연산자는 대응되는 두 비트가 모두 1일 때만 1을 반환하며, 다른 경우는 모두 0을 반환합니다.

비트 연산자

다음 그림은 비트 OR 연산자(|)의 동작을 나타냅니다.

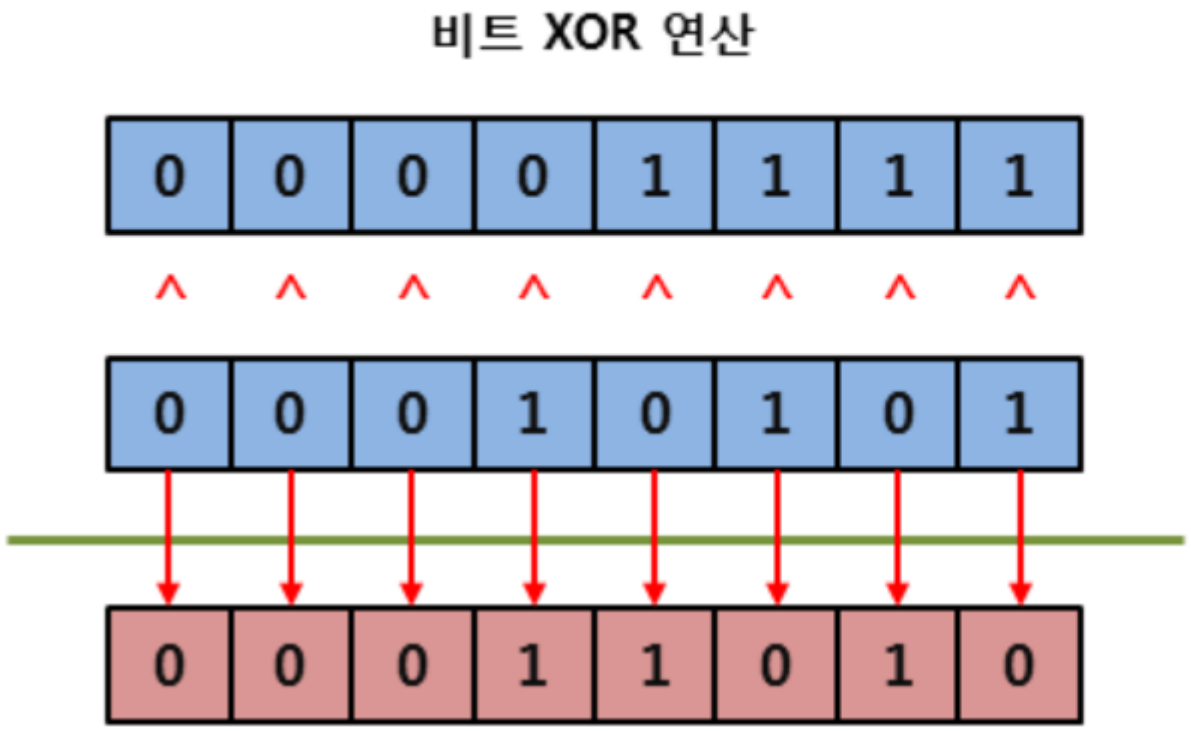
이처럼 비트 OR 연산자는 대응되는 두 비트 중 하나라도 1이면 1을 반환하며, 두 비트가 모두 0일 때만 0을 반환합니다.



비트 연산자

다음 그림은 비트 XOR 연산자(^)의 동작을 나타냅니다.

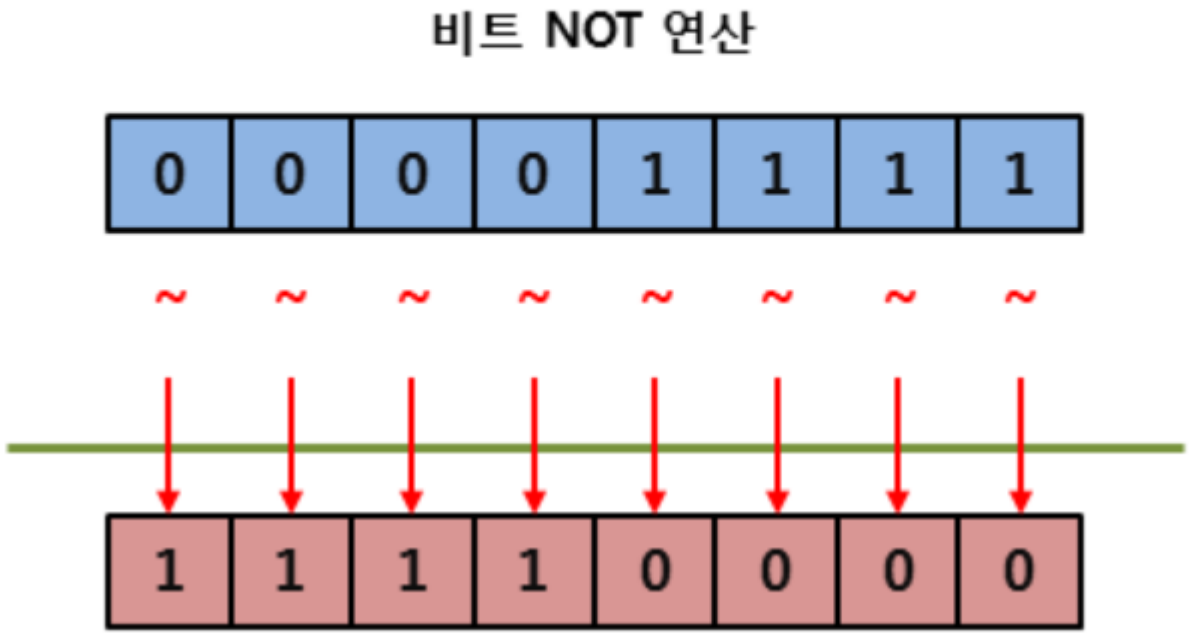
이처럼 비트 XOR 연산자는 대응되는 두 비트가 서로 다르면 1을 반환하고, 서로 같으면 0을 반환합니다.



비트 연산자

다음 그림은 비트 NOT 연산자(~)의 동작을 나타냅니다.

이처럼 비트 NOT 연산자는 해당 비트가 1이면 0을 반환하고, 0이면 1을 반환합니다.



비트 연산자 예제

예제

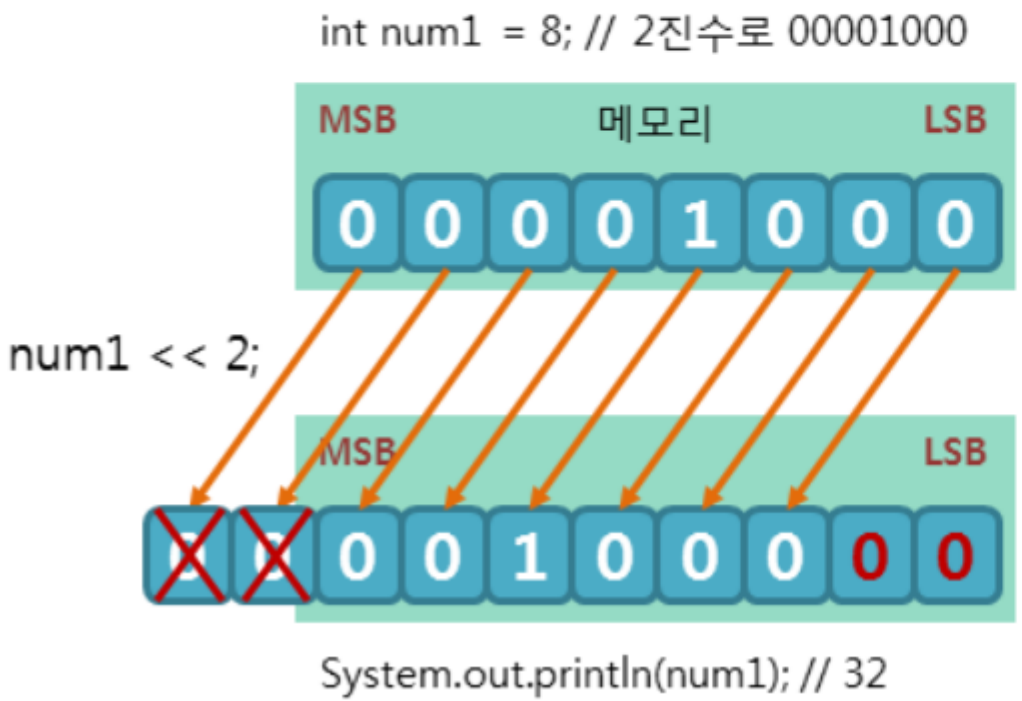
```
int num1 = 8, num2 = -8;


① System.out.println("~ 연산자에 의한 결과 : "+ ~num1);
② System.out.println("<< 연산자에 의한 결과 : "+ (num1 << 2));
③ System.out.println(">> 연산자에 의한 결과 : "+ (num2 >> 2));
④ System.out.println(">>> 연산자에 의한 결과 : "+ (num1 >>> 2));
⑤ System.out.println(">>> 연산자에 의한 결과 : "+ (num2 >>> 2));
```

실행 결과

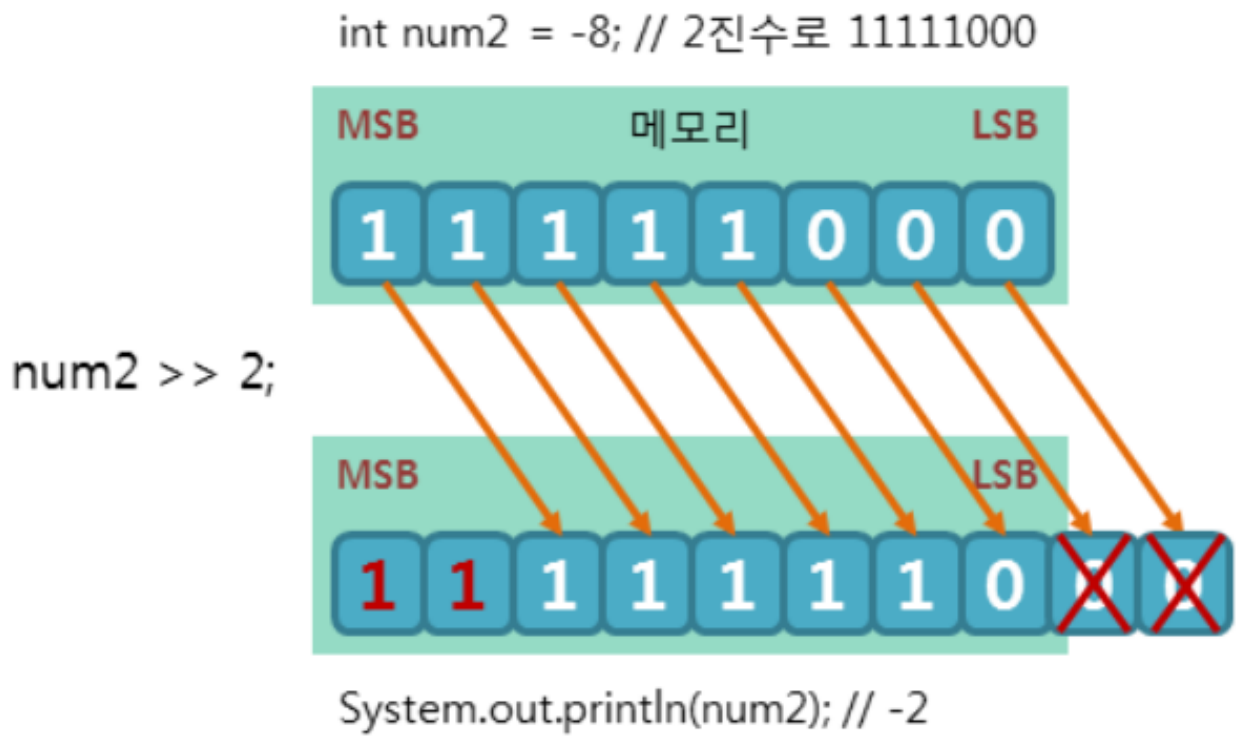
```
~   연산자에 의한 결과 : -9
<<  연산자에 의한 결과 : 32
>>  연산자에 의한 결과 : -2
>>> 연산자에 의한 결과 : 2
>>> 연산자에 의한 결과 : 1073741822
```

위 예제의 ①번 라인에서 비트 반전 연산자(~)는 피연산자의 1의 보수를 반환하므로, 피연산자의 부호만 반대로 변경됩니다.

②번 라인의 왼쪽 시프트 연산자(<<)는 지정한 수만큼 피연산자의 모든 비트를 전부 왼쪽으로 이동시킵니다.
이때 비트의 이동으로 새로 생기는 오른쪽 비트들은 언제나 0으로 채워집니다.
실행 결과를 살펴보면, 모든 비트가 한 비트씩 왼쪽으로 이동할 때마다 그 값은 2배씩 증가한다는 사실을 알 수 있습니다.

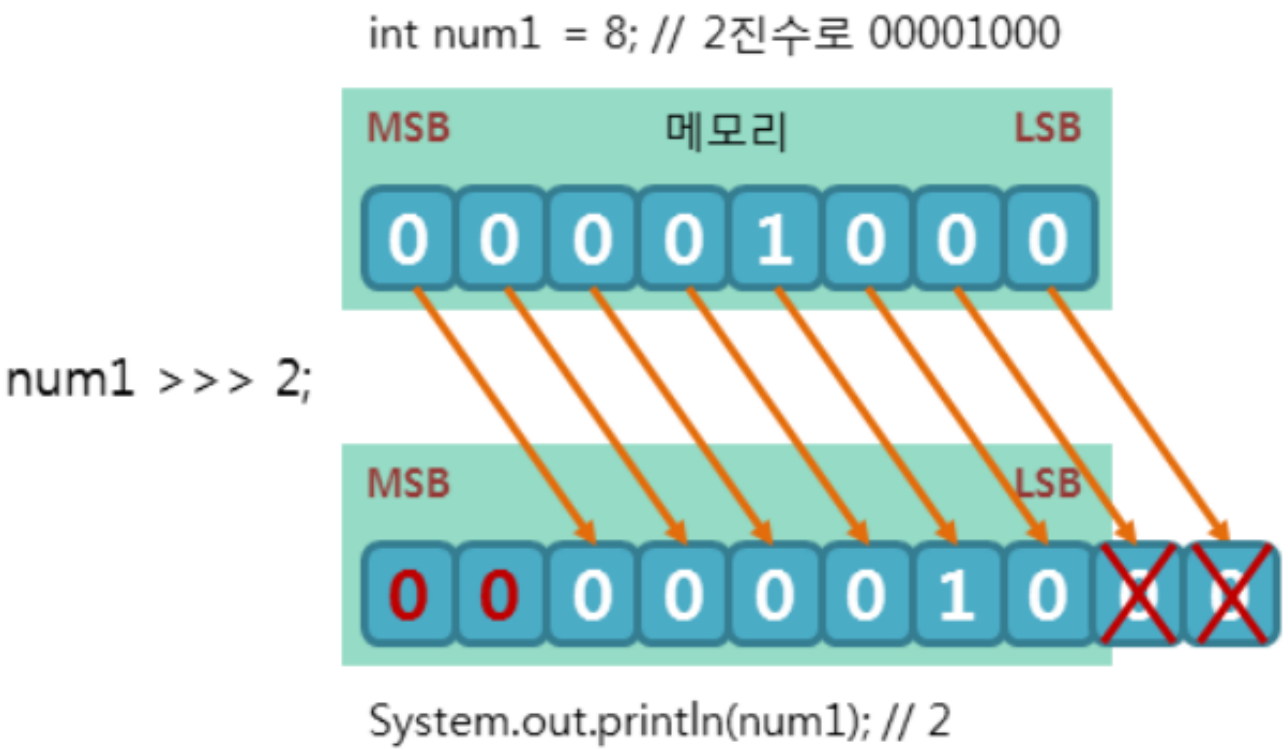


또한, ③번 라인의 오른쪽 시프트 연산자(>>)는 지정한 수만큼 피연산자의 모든 비트를 전부 오른쪽으로 이동시킵니다.
이때 비트의 이동으로 새로 생기는 왼쪽 비트들은 양수일 경우에는 모두 0으로 채워지며, 음수일 경우에는 모두 1로 채워집니다.
따라서 부호는 변하지 않습니다.
실행 결과를 살펴보면, 모든 비트가 한 비트씩 오른쪽으로 이동할 때마다 그 값은 2배씩 감소한다는 사실을 알 수 있습니다.

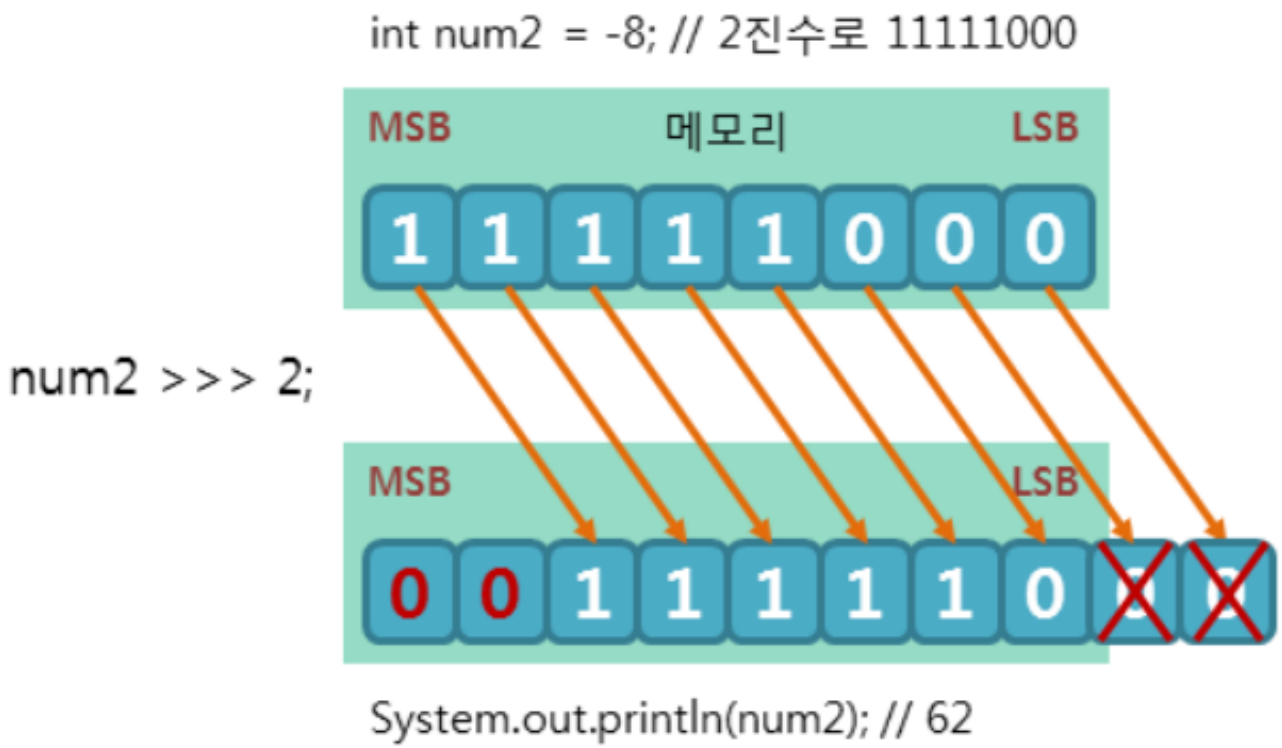


④번 라인의 오른쪽 시프트 연산자(>>>)는 부호 비트까지 포함하여 모든 비트를 전부 오른쪽으로 이동시킵니다.
이때 비트의 이동으로 새로 생기는 왼쪽 비트들은 언제나 0으로 채워집니다.

따라서 피연산자가 양수인 경우에는 부호 비트를 이동하지 않는 오른쪽 시프트 연산자(>>)와 같은 결과를 반환합니다.



하지만 피연산자가 음수인 경우에는 부호 비트까지도 이동하므로, 전혀 다른 결과가 반환됩니다.
다음 그림은 1바이트의 경우일 때 연산 결과를 나타나며, 위의 예제에서는 총 4바이트일 경우의 연산 결과를 보여줍니다.



따라서 이 시프트 연산자는 10진수의 연산보다는 2진수의 연산에서만 주로 사용됩니다.

삼항 연산자

- 삼항 연산자는 자바에서 유일하게 피연산자를 세 개나 가지는 조건 연산자입니다.

삼항 연산자의 문법은 다음과 같습니다.

문법

```
조건식 ? 반환값1 : 반환값2
```

물음표(?) 앞의 조건식에 따라 결괏값이 참(true)이면 반환값1을 반환하고, 결괏값이 거짓(false)이면 반환값2를 반환합니다.

삼항 연산자

예제

```
int num1 = 5, num2 = 7;
int result;


result = (num1 - num2 > 0) ? num1 : num2;
System.out.println("두 정수 중 더 큰 수는 " + result + "입니다.");
```

실행 결과

```
두 정수 중 더 큰 수는 7입니다.
```