3. 연산자

#1.인강/0.자바/1.자바-입문

산술 연산자

연산자 시작

`+`, `-`, `*`, `/`와 같이 계산을 수행하는 기호를 연산자라 한다. 자바에는 다음과 같은 다양한 연산자가 있다. 참고로 더 많은 연산자가 있지만, 여기서는 실무에서 주로 다루는 연산자 위주로 설명하겠다.

연산자 종류

- 산술 연산자: `+`, `-`, `*`, `/`, `%`(나머지 연산자)
- 증감(증가 및 감소) 연산자: `++`, `--`
- 비교 연산자: `==`, `!=`, `>`, `<`, `>=`, `<=`
- 논리 연산자: `&&` (AND), `||` (OR), `!` (NOT)
- 대입 연산자: `=`, `+=`, `-=`, `*=`, `/=`, `%=`
- 삼항 연산자: `? :`

연산자와 피연산자

```
3 + 4
a + b
```

- 연산자(operator): 연산 기호 - 예) `+`, `-`
- 피연산자(operand): 연산 대상 - 예) `3`, `4`, `a`, `b`

산술 연산자

산술 연산자는 주로 숫자를 계산하는 데 사용된다. 우리가 이미 잘 알고 있는 수학 연산을 수행한다.

- `+` (더하기)
- `-` (빼기)
- `*` (곱하기)
- `/` (나누기)
- `%` (나머지)

다음 코드를 작성해보자.

Operator1

```
package operator;

public class Operator1 {

    public static void main(String[] args) {
        // 변수 초기화
        int a = 5;
        int b = 2;

        // 덧셈
        int sum = a + b;
        System.out.println("a + b = " + sum);  // 출력: a + b = 7

        // 뺄셈
        int diff = a - b;
        System.out.println("a - b = " + diff);  // 출력: a - b = 3

        // 곱셈
        int multi = a * b;
        System.out.println("a * b = " + multi);  // 출력: a * b = 10

        // 나눗셈
        int div = a / b;
        System.out.println("a / b = " + div);  // 출력: a / b = 2

        // 나머지
        int mod = a % b;
        System.out.println("a % b = " + mod);  // 출력: a % b = 1
    }
```

```
}
```

int sum = a + b 계산 과정

```
int sum = a + b //1. 변수 값 읽기
int sum = 5 + 2 //2. 변수 값 계산
int sum = 7 //3. 계산 결과를 sum에 대입
sum = 7 //최종 결과
```

실행 결과

```
a + b = 7
a - b = 3
a * b = 10
a / b = 2
a % b = 1
```

- `5 / 2`의 결과는 `2.5`가 되어야 하지만 결과는 소수점이 제거된 `2`가 나왔다.
  - 자바에서 같은 `int`형끼리 계산하면 계산 결과도 같은 `int`형을 사용한다. `int`형은 정수이기 때문에 소수점 이하를 포함할 수 없다.
  - 이 부분에 대한 자세한 내용과 해결 방안은 뒤의 형변환에서 다룬다.
- 나머지 연산자(`%`)
  - 이름 그대로 나머지를 구하는 연산자이다. `5 / 2`는 몫이 2 나머지가 1이다. 따라서 나머지 연산자 `5 % 2`의 결과는 `1`이 된다.
  - 나머지 연산자는 실무와 알고리즘 모두 종종 사용되므로 잘 기억해두자

주의! 0으로 나누기

`10 / 0`과 같이 숫자는 0으로 나눌 수 없다. (수학에서 허용하지 않음)
방금 예제에서 변수 `b = 0`을 대입하면 `5 / 0`이 된다. 이 경우 프로그램에 오류가 발생한다.

실행하면 다음과 같은 예외를 확인할 수 있다.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

예외가 발생하면 해당 시점 이후의 결과가 출력되지 않고 프로그램이 종료된다. 예외에 대한 자세한 부분은 예외 처리에서 다룬다.

문자열 더하기

자바는 특별하게 문자열에도 `+` 연산자를 사용할 수 있다. 문자열에 `+` 연산자를 사용하면 두 문자를 연결할 수 있다.

다음 코드를 작성해보자.

```
package operator;

public class Operator2 {

    public static void main(String[] args) {

        //문자열과 문자열 더하기1
        String result1 = "hello " + "world";
        System.out.println(result1);

        //문자열과 문자열 더하기2
        String s1 = "string1";
        String s2 = "string2";
        String result2 = s1 + s2;
        System.out.println(result2);

        //문자열과 숫자 더하기1
        String result3 = "a + b = " + 10;
        System.out.println(result3);

        //문자열과 숫자 더하기2
        int num = 20;
        String str = "a + b = ";
        String result4 = str + num;
        System.out.println(result4);
    }

}
```

실행 결과

```
hello world
string1string2
a + b = 10
a + b = 20
```

문자열과 문자열 더하기1

`String result1 = "hello " + "world"`

- `"hello "` 문자열과 `"world"` 문자열을 더해서 `"hello world"` 문자열을 만든다.

- 결과를 `result1`에 저장한다.

문자열과 문자열 더하기2

`String result2 = s1 + s2`

- `s1`과 `s2` 변수에 있는 문자열을 읽는다.
- `"string1"` + `"string2"` 연산을 수행해서 `"string1string2"` 문자열을 만든다.
- 결과를 `result2`에 저장한다.

문자열과 숫자 더하기1

다음 식은 문자열과 숫자를 더한다. 자바에서 **문자와 숫자를 더하면 숫자를 문자열로 변경한 다음에 서로 더한다.**

`"a + b = " + 10`

- 문자: `"a + b = "`
- 숫자: `10`

계산 과정

```
"a + b = "(String) + 10(int) //문자열과 숫자 더하기
"a + b = "(String) + "10"(int -> String) //숫자를 문자열로 변경
"a + b = " + "10" //문자열과 문자열 더하기
"a + b = 10" //결과
```

문자열과 숫자 더하기2

변수에 담겨 있어도 문자와 숫자를 더하면 문자가 된다. 계산 과정을 확인해보자.

계산 과정

```
str(String) + num(int)
"a + b = "(String) + num(int) //str 변수에서 값 조회
"a + b = "(String) + 20(int) //num 변수에서 값 조회
"a + b = "(String) + "20"(int -> String) //숫자를 문자열로 변경
"a + b = " + "20" //문자열과 문자열 더하기
"a + b = 20" //결과
```

자바는 문자열인 `String` 타입에 다른 타입을 더하는 경우 대상 타입을 문자열로 변경한다. 쉽게 이야기해서 문자열에 더하는 것은 다 문자열이 된다.

연산자 우선순위

수학에서 `1 + 2 * 3`의 연산 결과는 무엇일까? 덧셈보다 곱셈이 우선순위가 더 높다. 따라서 다음과 같이 계산한다.

```
1 + (2 * 3) //곱셈(*)이 연산자 우선순위가 높다. 따라서 먼저 계산한다.
1 + 6
7 //결과
```

자바도 마찬가지이다. 코드로 연산자 우선순위를 확인해보자.

```
package operator;

public class Operator3 {

    public static void main(String[] args) {
        int sum1 = 1 + 2 * 3; //1 + (2 * 3)과 같다.
        int sum2 = (1 + 2) * 3;
        System.out.println("sum1 = " + sum1); //sum1 = 7
        System.out.println("sum2 = " + sum2); //sum2 = 9
    }

}
```

실행 결과

```
sum1 = 7
sum2 = 9
```

- 출력 결과를 보면 `sum1 = 7`이 나왔다. 연산자 우선순위에 의해 곱셈이 먼저 계산된 것이다.
- 연산자 우선순위를 변경하려면 수학과 마찬가지로 괄호 `()`를 사용하면 된다. `()`를 사용한 곳이 먼저 계산된다.
- `sum2`는 괄호를 사용해서 덧셈이 먼저 처리되도록 했다.

sum2 계산 순서

```
(1 + 2) * 3
3 * 3
9
```

이번에는 조금 더 복잡한 예제를 만들어보자.

```
package operator;

public class Operator4 {

    public static void main(String[] args) {
        int sum3 = 2 * 2 + 3 * 3; //(2 * 2) + (3 * 3)
```

```
        int sum4 = (2 * 2) + (3 * 3); //sum3과 같다.
        System.out.println("sum3 = " + sum3); //sum3 = 13
        System.out.println("sum4 = " + sum4); //sum4 = 13
    }

}
```

실행 결과

```
sum3 = 13
sum4 = 13
```

`sum3`, `sum4`에 저장하는 두 연산은 같은 연산이다. 그런데 괄호가 없는 `2 * 2 + 3 * 3` 연산은 평소 수학을 잘 하는 분들은 금방 풀겠지만 보통은 이 연산을 보고 잠깐 연산자 우선순위를 생각을 해야한다.

```
2 * 2 + 3 * 3
(2 * 2) + (3 * 3) //곱셈이 우선순위가 높다
4 + 9
13
```

이렇게 복잡한 경우 `sum4`의 `(2 * 2) + (3 * 3)`와 같이 괄호를 명시적으로 사용하는 것이 더 명확하고 이해하기 쉽다.
코드를 몇자 줄여서 모호하거나 복잡해 지는 것 보다는 코드가 더 많더라도 명확하고 단순한 것이 더 유지보수 하기 좋다.
연산자 우선순위가 애매하거나 조금이라도 복잡하다면 언제나 괄호를 고려하자!

연산자 우선순위 암기법

자바는 다음과 같은 연산자 우선순위가 있다. 높은 것에서 낮은 순으로 적었다. 처음에 나오는 괄호`()`가 우선순위가 가장 높고, 마지막의 대입 연산자(`=`)가 우선순위가 가장 낮다.

1. **괄호 ()**
2. **단항 연산자** (예: `++`, `--`, `!`, `~`, `new`, `(type)`)
3. **산술 연산자** (`*`, `/`, `%` 우선, 그 다음에 `+`, `-`)
4. **Shift 연산자** (`<<`, `>>`, `>>>`)
5. **비교 연산자** (`<`, `<=`, `>`, `>=`, `instanceof`)
6. **등식 연산자** (`==`, `!=`)
7. **비트 연산자** (`&`, `^`, `|`)

8. **논리 연산자** (`&&`, `||`)
9. **삼항 연산자** (`? :`)
10. **대입 연산자** (`=`, `+=`, `-=`, `*=`, `/=`, `%=` 등등)

그러면 이 많은 우선순위를 어떻게 외워야 할까? 사실 대부분의 실무 개발자들은 연산자 우선순위를 외우지 않는다.
연산자 우선순위는 딱 2가지만 기억하면 된다.

1. 상식선에서 우선순위를 사용하자
우선순위는 상식선에서 생각하면 대부분 문제가 없다.

다음 예를 보자
`int sum = 1 + 2 * 3`
당연히 `+` 보다 `*`이 우선순위가 높다.

다음으로 산술 연산자(`+`)와 대입연산자(`=`)를 비교하는 예를 보자.
`int sum = 1 + 2`

```
int sum = 1 + 2
int sum = 3 //산술 연산자가 먼저 처리된다.
sum = 3 //대입 연산자가 마지막에 처리된다.
```

- `1 + 2`를 먼저 처리한 다음에 그 결과 값을 변수 `sum`에 넣어야 한다. 대입 연산자인 `=`이 먼저 수행된다고 생각하기가 사실 더 어렵다.

2. 애매하면 괄호()를 사용하자
코드를 딱 보았을 때 연산자 우선순위를 고민해야 할 것 같으면, 그러니까 뭔가 복잡해보이면 나 뿐만 아니라 모든 사람이 그렇게 느낀다. 이때는 다음과 같이 괄호를 사용해서 우선순위를 명시적으로 지정하면 된다.
`((2 * 2) + (3 * 3)) / (3 + 2)`

정리

- 연산자 우선순위는 상식선에서 생각하고, 애매하면 괄호를 사용하자
- 누구나 코드를 보고 쉽고 명확하게 이해할 수 있어야 한다. 개발자들이 연산자 우선순위를 외우고 개발하는 것이 아니다! 복잡하면 명확하게 괄호를 넣어라!
- 개발에서 가장 중요한 것은 단순함과 명확함이다! 애매하거나 복잡하면 안된다.

증감 연산자

증가 및 감소 연산자를 줄여서 증감 연산자라 한다.

증감 연산자는 `++`와 `--`로 표현되며, 이들은 변수의 값을 1만큼 증가시키거나 감소시킨다.

프로그래밍에서는 값을 1씩 증가하거나 1씩 감소할 때가 아주 많기 때문에 이런 편의 기능을 제공한다.

OperatorAdd1

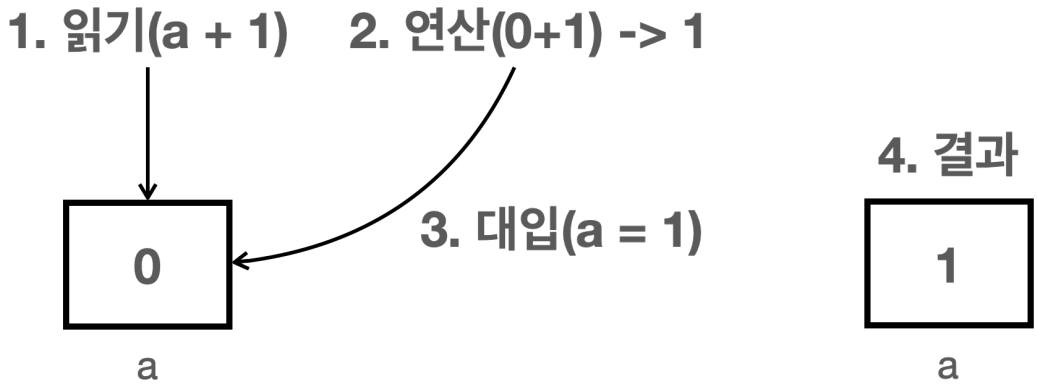
```
package operator;

public class OperatorAdd1 {

    public static void main(String[] args) {
        int a = 0;

        a = a + 1;
        System.out.println("a = " + a); //1
        a = a + 1;
        System.out.println("a = " + a); //2

        //증감 연산자
        ++a; //a = a + 1
        System.out.println("a = " + a); //3
        ++a; //a = a + 1
        System.out.println("a = " + a); //4
    }

}
```

실행 결과

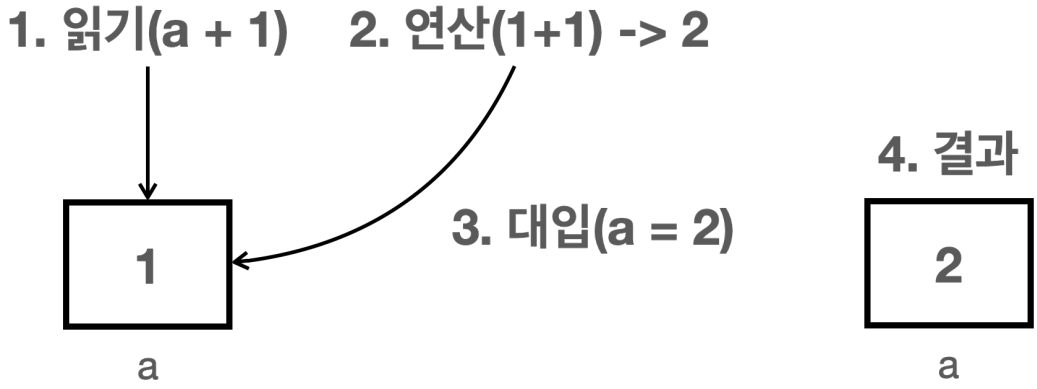
```
a = 1
a = 2
a = 3
a = 4
```

변수 `a`의 값을 하나 증가하려면 `a = a + 1` 연산을 수행해야 한다. 자기 자신에 `1`을 더하고 그 결과를 자신에게 다시 저장해야 한다.

코드는 다음과 같이 수행된다.




```
//a = 0
a = a + 1
a = 0 + 1 //변수 a의 값 확인(0)
a = 1
```




```
//a = 1
a = a + 1
a = 1 + 1 //변수 a의 값 확인(1)
a = 2
```

`a = a + 1`을 `++a`로 간단히 표현할 수 있는 것이 바로 증감 연산자이다.
정리하면 해당 변수에 들어있는 숫자 값을 하나 증가하는 것이다.

`++` (증가), `--` (감소)
값을 하나 감소할 때는 `--a`와 같이 표현하면 된다. 이것은 `a = a - 1`이 된다.

전위, 후위 증감연산자

증감 연산자는 피연산자 앞에 두거나 뒤에 둘 수 있으며, 연산자의 위치에 따라 연산이 수행되는 시점이 달라진다.

- `++a`: 증감 연산자를 피연산자 앞에 둘 수 있다. 이것을 앞에 있다고 해서 전위(Prefix) 증감 연산자라 한다.
- `a++`: 증감 연산자를 피연산자 뒤에 둘 수 있다. 이것을 뒤에 있다고 해서 후위(Postfix) 증감 연산자라 한다.

코드로 확인해보자.

OperatorAdd2

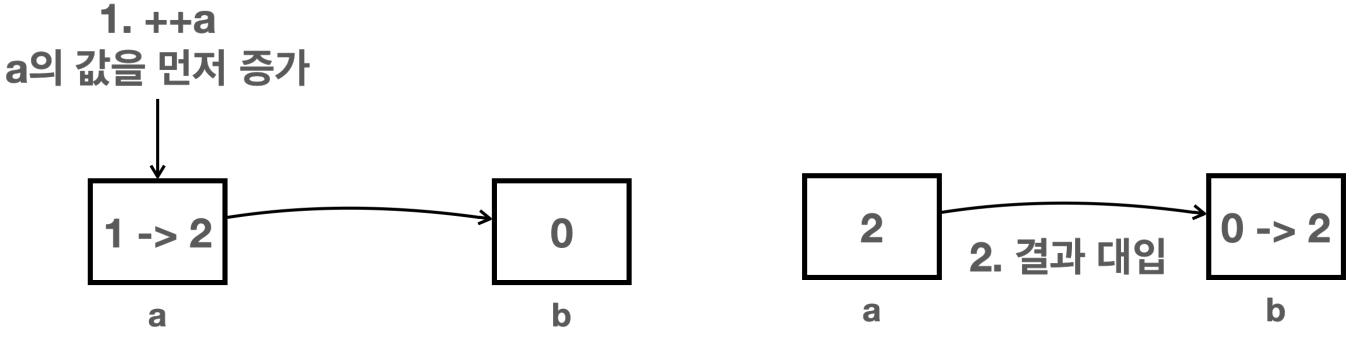
```
package operator;

public class OperatorAdd2 {

    public static void main(String[] args) {
        // 전위 증감 연산자 사용 예
        int a = 1;
        int b = 0;
        b = ++a; // a의 값을 먼저 증가시키고, 그 결과를 b에 대입
        System.out.println("a = " + a + ", b = " + b); // 결과: a = 2, b = 2

        // 후위 증감 연산자 사용 예
        a = 1; // a 값을 다시 1로 지정
        b = 0; // b 값을 다시 0으로 지정
        b = a++; // a의 현재 값을 b에 먼저 대입하고, 그 후 a 값을 증가시킴
        System.out.println("a = " + a + ", b = " + b); // 결과: a = 2, b = 1
    }

}
```

실행 결과

```
a = 2, b = 2
a = 2, b = 1
```

증감 연산자가 변수 앞에 오는 경우를 **전위 증감 연산자**라고 하며, 이 경우에는 증감 연산이 먼저 수행된 후 나머지 연산이 수행된다.

예) ++a 전위 증감 연산자

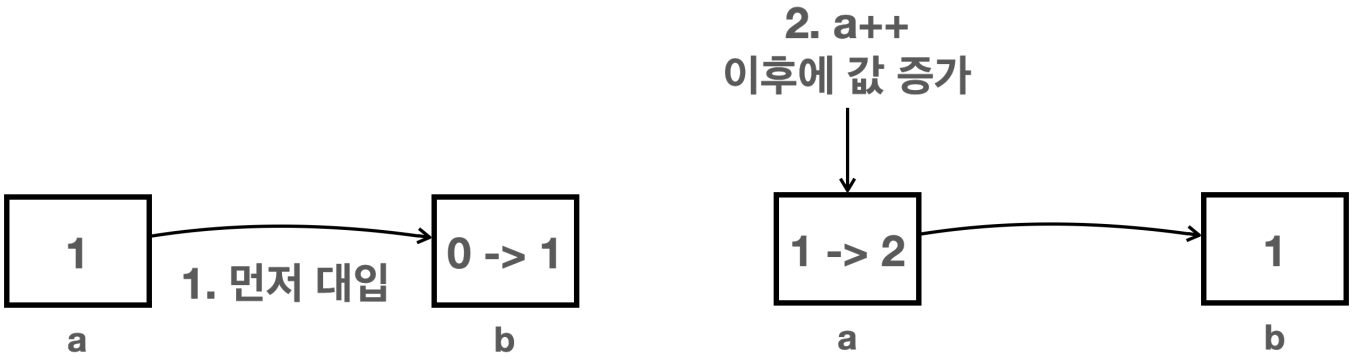



```
a = 1, b = 0
b = ++a //전위 증감 연산자
```

```
a = a + 1 //a의 증감 연산이 먼저 진행, a = 2
b = a //이후에 a를 대입 b = 2

결과: a = 2, b = 2
```

증감 연산자가 변수 뒤에 오는 경우를 **후위 증감 연산자**라고 하며, 이 경우에는 다른 연산이 먼저 수행된 후 증감 연산이 수행된다.

예) a++ 후위 증감 연산자




```
a = 1, b = 0
b = a++ //후위 증감 연산자

b = a; //a의 값을 먼저 b에 대입 b = 1
a = a + 1; //이후에 a의 값을 증가 a = 2

결과: a = 2, b = 1
```

참고로 다음과 같이 증감 연산자를 단독으로 사용하는 경우에는 다른 연산이 없기 때문에, 본인의 값만 증가한다. 따라서 전위이든 후위이든 둘다 결과가 같다.

```
++a;
a++;
```

비교 연산자

비교 연산자는 두 값을 비교하는 데 사용한다. 비교 연산자는 주로 뒤에서 설명하는 조건문과 함께 사용한다.

비교 연산자

- `==` : 동등성 (equal to)

- `!=` : 불일치 (not equal to)
- `>` : 크다 (greater than)
- `<` : 작다 (less than)
- `>=` : 크거나 같다 (greater than or equal to)
- `<=` : 작거나 같다 (less than or equal to)

비교 연산자를 사용하면 참(`true`) 또는 거짓(`false`)이라는 결과가 나온다. 참 거짓은 `boolean`형을 사용한다.

여기서 주의할 점은 `=`와 `==`(`=`x2)이 다르다는 점이다.

- `=` : 대입 연산자, 변수에 값을 대입한다.
- `==` : 동등한지 확인하는 비교 연산자

불일치 연산자는 `!=`를 사용한다. `!`는 반대라는 뜻이다.

비교 연산자를 예제 코드로 확인해보자.

Comp1

```
package operator;

public class Comp1 {
    public static void main(String[] args) {
        int a = 2;
        int b = 3;

        System.out.println(a == b);  // false, a와 b는 같지 않다
        System.out.println(a != b);  // true, a와 b는 다르다
        System.out.println(a > b);   // false, a는 b보다 크지 않다
        System.out.println(a < b);   // true, a는 b보다 작다
        System.out.println(a >= b);  // false, a는 b보다 크거나 같지 않다
        System.out.println(a <= b);  // true, a는 b보다 작거나 같다

        //결과를 boolean 변수에 담기
        boolean result = a == b; //a == b: false
        System.out.println(result); //false
    }
}
```

문자열 비교

문자열이 같은지 비교할 때는 `==`이 아니라 `.equals()` 메서드를 사용해야 한다.

`==`를 사용하면 성공할 때도 있지만 실패할 때도 있다. 지금은 이 부분을 이해하기 어려우므로 지금은 단순히 문자열의 비교는 `.equals()` 메서드를 사용해야 한다 정도로 알고 있자, 자세한 내용은 별도로 다룬다.

Comp2 - 문자열 비교 예시

```
package operator;

public class Comp2 {
    public static void main(String[] args) {
        String str1 = "문자열1";
        String str2 = "문자열2";

        boolean result1 = "hello".equals("hello"); //리터럴 비교
        boolean result2 = str1.equals("문자열1");//문자열 변수, 리터럴 비교
        boolean result3 = str1.equals(str2);//문자열 변수 비교

        System.out.println("result1 = " + result1);
        System.out.println("result2 = " + result2);
        System.out.println("result3 = " + result3);

    }
}
```

실행 결과

```
result1 = true
result2 = true
result3 = false
```

논리 연산자

논리 연산자는 `boolean`형인 `true`, `false`를 비교하는데 사용한다.

논리 연산자

- `&&` (그리고) : 두 피연산자가 모두 참이면 참을 반환, 둘중 하나라도 거짓이면 거짓을 반환
- `||` (또는) : 두 피연산자 중 하나라도 참이면 참을 반환, 둘다 거짓이면 거짓을 반환
- `!` (부정) : 피연산자의 논리적 부정을 반환. 즉, 참이면 거짓을, 거짓이면 참을 반환

코드로 간단히 확인해보자

Logical1

```
package operator;

public class Logical1 {

    public static void main(String[] args) {
        System.out.println("&&: AND 연산");
        System.out.println(true && true); //true
        System.out.println(true && false);//false
        System.out.println(false && false);//false

        System.out.println("||: OR 연산");
        System.out.println(true || true); //true
        System.out.println(true || false);//true
        System.out.println(false || false);//false

        System.out.println("! 연산");
        System.out.println(!true); //false
        System.out.println(!false); //true

        System.out.println("변수 활용");
        boolean a = true;
        boolean b = false;
        System.out.println(a && b); // false
        System.out.println(a || b); // true
        System.out.println(!a);     // false
        System.out.println(!b);     // true
    }
}
```

- `&&` : 두 피연산자가 모두 참이어야 `true` 를 반환한다. 둘중 하나라도 거짓이면 `false` 를 반환한다.
- `||` : 두 피연산자 중 하나라도 참이면 `true` 를 반환한다. 둘다 모두 거짓이면 `false` 를 반환한다.
- `!` : 피연산자의 논리적 부정을 반환한다. 참이면 거짓을, 거짓이면 참을 반환한다.

- `a && b` 는 `false` 를 반환한다. 왜냐하면 둘 중 하나인 `b` 가 거짓이기 때문이다.
- `a || b` 는 `true` 를 반환한다. 왜냐하면 둘 중 하나인 `a` 가 참이기 때문이다.
- `!a` 와 `!b` 는 각각의 논리적 부정을 반환한다

논리 연산자 활용

논리 연산자를 활용하는 다음 코드를 만들어보자.

변수 `a`가 `10`보다 크고 `20`보다 작은지 논리 연산자를 사용해서 확인해보자.

Logical2

```
package operator;

public class Logical2 {

    public static void main(String[] args) {
        int a = 15;
        //a는 10보다 크고 20보다 작다
        boolean result = a > 10 && a < 20; //(a > 10) && (a < 20)
        System.out.println("result = " + result);
    }
}
```

실행 결과

```
result = true
```

참고로 다음과 같이 변수의 위치를 변경해도 결과는 같다.

범위를 나타내는 경우 이렇게 작성하면 코드를 조금 더 읽기 좋다.

```
boolean result = 10 < a && a < 20;
```

대입 연산자

대입 연산자

대입 연산자(`=`)는 값을 변수에 할당하는 연산자다. 이 연산자를 사용하면 변수에 값을 할당할 수 있다.

예를 들어, `int a = 1`는 `a`라는 변수에 `1`이라는 값을 할당한다.

축약(복합) 대입 연산자

산술 연산자와 대입 연산자를 한번에 축약해서 사용할 수 있는데, 이것을 축약(복합) 대입 연산자라 한다.

연산자 종류: `+=`, `-=`, `*=`, `/=`, `%=`

이 연산자는 연산과 대입을 한번에 축약해서 처리한다. 다음 왼쪽과 오른쪽의 결과는 같다.

`i = i + 3` ➜ `i += 3`

`i = i * 4` ➜ `i *= 4`

예제 코드를 통해 확인해보자

Assign1

```
package operator;

public class Assign1 {

    public static void main(String[] args) {
        int a = 5;  // 5
        a += 3;     // 8 (5 + 3):  a = a + 3
        a -= 2;     // 6 (8 - 2):  a = a - 2
        a *= 4;     // 24 (6 * 4): a = a * 4
        a /= 3;     // 8 (24 / 3): a = a / 3
        a %= 5;     // 3 (8 % 5) : a = a % 5
        System.out.println(a);
    }
}
```

실행 결과

```
3
```

문제와 풀이

> **코딩이 처음이라면 필독!**
>
> 프로그래밍이 처음이라면 아직 코딩 자체가 익숙하지 않기 때문에 문제와 풀이에 상당히 많은 시간을 쓰게 될 수 있다. 강의를 들을 때는 다 이해가 되는 것 같았는데, 막상 혼자 생각해서 코딩을 하려니 잘 안되는 것이다. 이것은 아직 코딩이 익숙하지 않기 때문인데, 처음 코딩을 하는 사람이라면 누구나 겪는 자연스러운 현상이다.
>
> 문제를 스스로 풀기 어려운 경우, 너무 고민하기 보다는 먼저 **강의 영상의 문제 풀이 과정을 코드로 따라하면서 이해하자. 반드시 코드로 따라해야 한다.** 그래야 코딩하는 것에 조금씩 익숙해질 수 있다. 그런 다음에 정답을 지우고 스스로 문제를 풀어보면 된다. 참고로 강의를 듣는 시간만큼 문제와 풀이에도 많은 시간을 들어야 제대로 성장할 수 있다!

문제1 - int와 평균

다음과 같은 작업을 수행하는 프로그램을 작성하세요

클래스 이름은 `OperationEx1` 라고 적어주세요.

1. `num1` , `num2` , `num3` 라는 이름의 세 개의 `int` 변수를 선언하고, 각각 10, 20, 30으로 초기화하세요.
2. 세 변수의 합을 계산하고, 그 결과를 `sum` 이라는 이름의 `int` 변수에 저장하세요.
3. 세 변수의 평균을 계산하고, 그 결과를 `average` 라는 이름의 `int` 변수에 저장하세요. 평균 계산 시 소수점 이하의 결과는 버림하세요.
4. `sum` 과 `average` 변수의 값을 출력하세요.

참고

자바에서 `int` 끼리의 나눗셈은 자동으로 소수점 이하를 버린다.

문제1 - 정답

OperationEx1

```
package operator.ex;

public class OperationEx1 {
    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 20;
        int num3 = 30;

        int sum = num1 + num2 + num3;
        int average = sum / 3; //int 끼리의 나눗셈은 자동으로 소수점 이하를 버림

        System.out.println(sum);
        System.out.println(average);
    }
}
```

실행 결과

```
60
20
```

문제2 - double과 평균

클래스 이름: `OperationEx2`

```
// 다음 double 변수들을 선언하고 그 합과 평균을 출력하는 프로그램을 작성하세요.
double val1 = 1.5;
double val2 = 2.5;
double val3 = 3.5;
```

정답 - double 데이터 타입

```
package operator.ex;

public class OperationEx2 {
    public static void main(String[] args) {
        double val1 = 1.5;
        double val2 = 2.5;
        double val3 = 3.5;

        double sum = val1 + val2 + val3;
        double avg = sum / 3;

        System.out.println(sum);
        System.out.println(avg);
    }
}
```

실행 결과

```
7.5
2.5
```

문제3 - 합격 범위

클래스 이름: `OperationEx3`

- `int` 형 변수 `score` 를 선언하세요.
- `score` 가 80점 이상이고, 100점 이하이면 `true` 를 출력하고, 아니면 `false` 를 출력하세요.

정답 - OperationEx3

```
package operator.ex;

public class OperationEx3 {
    public static void main(String[] args) {
        int score = 80;
```

```
        boolean result = score >= 80 && score <= 100;
        System.out.println(result);
    }
}
```

실행 결과

```
true
```

정리

자주 사용하는 연산자

- 산술 연산자: `+`, `-`, `*`, `/`, `%` (나머지)
- 증가 및 감소 연산자: `++`, `--`
- 비교 연산자: `==`, `!=`, `>`, `<`, `>=`, `<=`
- 논리 연산자: `&&` (AND), `||` (OR), `!` (NOT)
- 대입 연산자: `=`, `+=`, `-=`, `*=`, `/=`, `%=`

다음 연산자들도 자주 사용하는데, 뒷 부분에서 학습한다

- 삼항 연산자: `? :`
- `instanceof` 연산자: 객체 타입을 확인한다.
- 그외: `new`, `[]` (배열 인덱스), `.` (객체 멤버 접근), `()` (메소드 호출)

비트 연산자는 실무에서 거의 사용할 일이 없다. 필요할 때 찾아보자.

- 비트 연산자: `&`, `|`, `^`, `~`, `<<`, `>>`, `>>>`