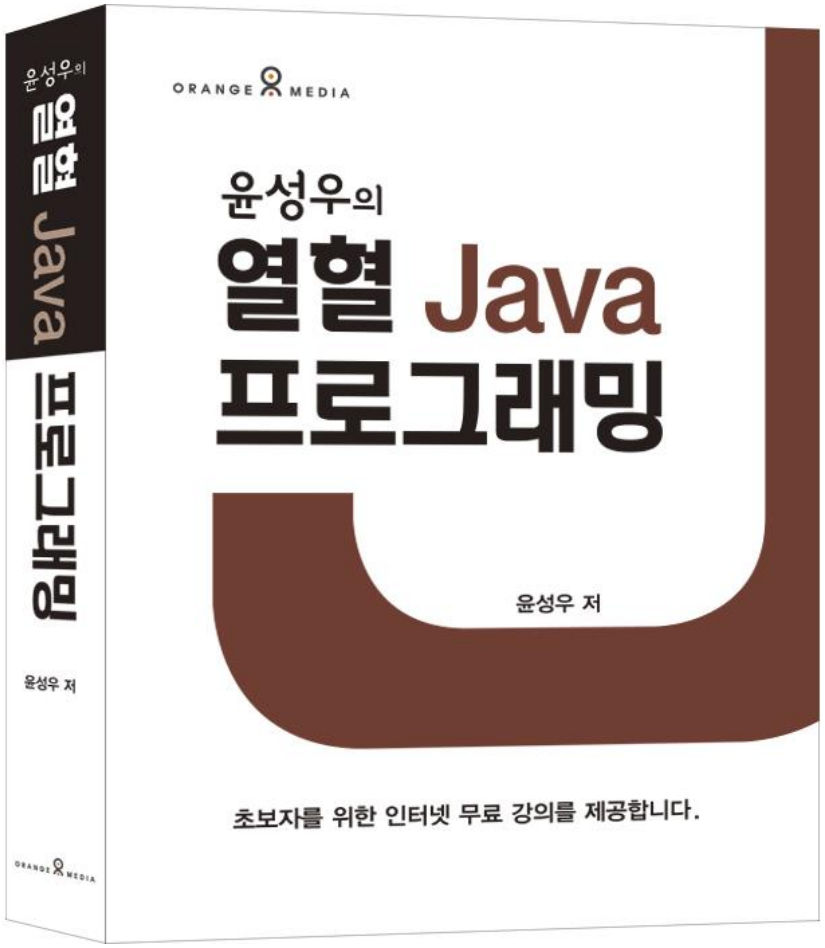
2022년 봄학기

JAVA

나사렛대학교
IT인공지능학부
김광기



열혈 Java 프로그래밍

Chapter 04. 연산자

04-1.

자바에서 제공하는 이항 연산자들

연산기호	결합 방향	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)

우선순위 적용

$2 - 1 - 3 \times 2$

결합 방향 적용

$= 2 - 1 - 6$

$= 1 - 6$

결합 방향은 우선순위가 같을 때 적용하는 기준.

연산자의 우선순위와 결합 방향

연산자	연산자의 기능	결합 방향
=	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;	➡

대입 연산자와 산술 연산자

◈ ArithOp.java

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

대입 연산자와 산술 연산자의 예

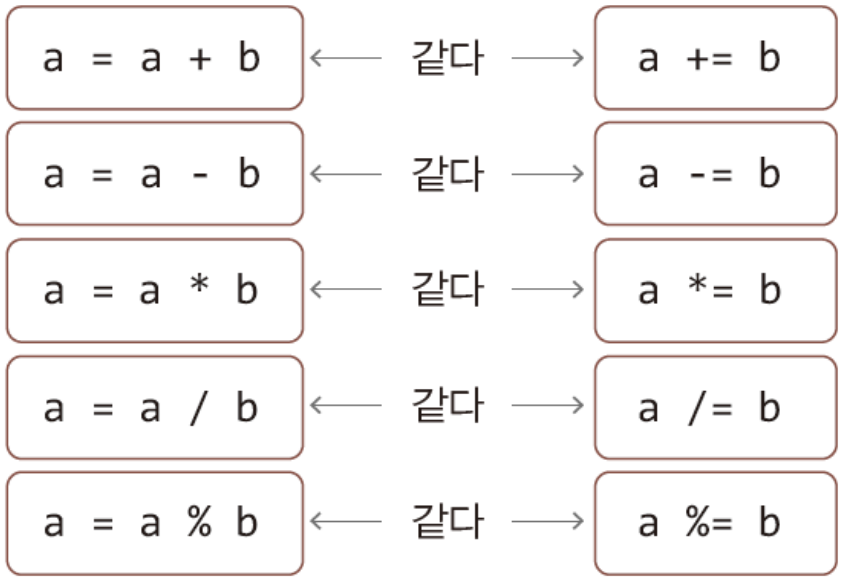
정수형 나눗셈과 실수형 나눗셈

```
int num1 = 7;
int num2 = 3;
System.out.println("num1 / num2 = " + (num1 / num2));
```

정수형 나눗셈 진행

```
System.out.println("num1 / num2 = " + (7.0 / 3.0));
```

실수형 나눗셈 진행



ex1)

num = num + 5;

→ num += 5;

ex2)

num = num * 3;

→ num *= 3;

복합 대입 연산자

A &= B	↔	A = A & B
A ^= B	↔	A = A ^ B
A <<= B	↔	A = A << B
A >>>= B	↔	A = A >>> B

복합 대입 연산자 추가

◈ CompAssignOp.java

```
1.  class CompAssignOp {
2.      public static void main(String[] args) {
3.          short num = 10;
4.          num = (short)(num + 77L);     // 형 변환 안하면 컴파일 오류 발생
5.          int rate = 3;
6.          rate = (int)(rate * 3.5);     // 형 변환 안하면 컴파일 오류 발생
7.          System.out.println(num);
8.          System.out.println(rate);
9.
10.         num = 10;
11.         num += 77L;     // 형 변환 필요하지 않다.
12.         rate = 3;
13.         rate *= 3.5;    // 형 변환 필요하지 않다.
14.         System.out.println(num);
15.         System.out.println(rate);
16.     }
17. }
```

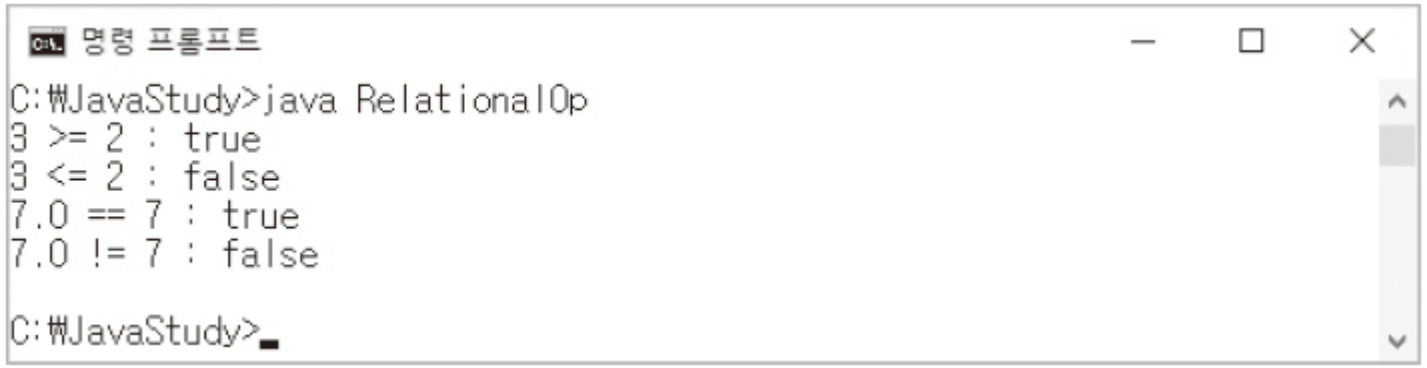
복합 대입 연산자 예제

연산자	연산자의 기능	결합 방향
〈	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?	➡

관계 연산자

◈ RelationalOp.java

```
1.  class RelationalOp {
2.      public static void main(String[] args) {
3.          System.out.println("3 >= 2 : " + (3 >= 2));
4.          System.out.println("3 <= 2 : " + (3 <= 2));
5.          System.out.println("7.0 == 7 : " + (7.0 == 7));
6.          System.out.println("7.0 != 7 : " + (7.0 != 7));
7.      }
8.  }
```



관계 연산자 예제

논리 연산자

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

피연산자 1(OP1)	피연산자 2(OP2)	연산 결과(OP1 && OP2)
true	true	true
true	false	false
false	true	false
false	false	false

피연산자 1(OP1)	피연산자 2(OP2)	연산 결과(OP1 || OP2)
true	true	true
true	false	true
false	true	true
false	false	false

피연산자(OP)	연산 결과(!OP)
true	false
false	true

◈ LogicalOp.java

```
1.  class LogicalOp {
2.      public static void main(String[] args) {
3.          int num1 = 11;
4.          int num2 = 22;
5.          boolean result;
6.
7.          // 변수 num1에 저장된 값이 1과 100 사이의 수인가?
8.          result = (1 < num1) && (num1 < 100);
9.          System.out.println("1 초과 100 미만인가? " + result);
10.
11.         // 변수 num2에 저장된 값이 2 또는 3의 배수인가?
12.         result = ((num2 % 2) == 0) || ((num2 % 3) == 0);
13.         System.out.println("2 또는 3의 배수인가? " + result);
14.
15.         // 변수 num1이 0 인가?
16.         result = !(num1 != 0);
17.         System.out.println("0 인가? " + result);
18.     }
19. }
```

논리 연산자 예제

논리 연산자 사용시 주의점 : SCE

```
result = ((num1 += 10) < 0) && ((num2 += 10) > 0);

result = ((num1 += 10) > 0) || ((num2 += 10) > 0);
```

num1과 num2의 값이 모두 증가할 수 있을까?

false 이면...

(num1 += 10) < 0 && (num2 += 10) > 0

이 부분 생략

true 이면...

(num1 += 10) > 0 || (num2 += 10) > 0

이 부분 생략

◈ SCE.java

```
1.  class SCE {
2.      public static void main(String[] args) {
3.          int num1 = 0;
4.          int num2 = 0;
5.          boolean result;
6.
7.          result = ((num1 += 10) < 0) && ((num2 += 10) > 0);
8.          System.out.println("result = " + result);
9.          System.out.println("num1 = " + num1);
10.         System.out.println("num2 = " + num2 + '\n');   // '\n'은 개 행
11.
12.         result = ((num1 += 10) > 0) || ((num2 += 10) > 0);
13.         System.out.println("result = " + result);
14.         System.out.println("num1 = " + num1);
15.         System.out.println("num2 = " + num2);
16.     }
17. }
```

SCE 동작을 확인하는 예제

04-2.

자바에서 제공하는 단항 연산자들

부호 연산자

```
double e1 = 3.5;

double e2 = -e1;        // e1에 저장되는 값은 -3.5
```

부호 연산자 -는 변수에 저장된 값의 부호를 바꾸어 반환한다.

◈ UnaryAddMin.java

```
1.  class UnaryAddMin {
2.      public static void main(String[] args) {
3.          short num1 = 5;
4.          System.out.println(+num1);    // 결과적으로 불필요한 + 연산
5.          System.out.println(-num1);    // 부호를 바꿔서 얻은 결과를 출력
6.
7.          short num2 = 7;
8.          short num3 = (short)(+num2);    // 형 변환 하지 않으면 오류 발생
9.          short num4 = (short)(-num2);    // 형 변환 하지 않으면 오류 발생
10.         System.out.println(num3);
11.         System.out.println(num4);
12.     }
13. }
```

부호 연산자 예제

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

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

증가 감소 연산자

◈ PrefixOp.java

```
1.  class PrefixOp {
2.      public static void main(String[] args) {
3.          int num = 7;
4.          System.out.println(++num);    // num의 값 하나 증가 후 출력
5.          System.out.println(++num);    // num의 값 하나 증가 후 출력
6.          System.out.println(num);
7.      }
8.  }
```

Prefix 증가 감소 연산자 예제

◈ PostfixOp.java

```
1.  class PostfixOp {
2.      public static void main(String[] args) {
3.          int num = 5;
4.          System.out.print((num++) + " ");   // 출력 후에 값이 증가
5.          System.out.print((num++) + " ");   // 출력 후에 값이 증가
6.          System.out.print(num + "\n");
7.
8.          System.out.print((num--) + " ");   // 출력 후에 값이 감소
9.          System.out.print((num--) + " ");   // 출력 후에 값이 감소
10.         System.out.print(num);
11.     }
12. }
```

Postfix 증가 감소 연산자 예제

04-3.

비트를 대상으로 하는 연산자들

비트 연산자의 이해

```
public static void main(String[] args) {
    byte n1 = 13;
    byte n2 = 7;
    byte n3 = (byte)(n1 & n2);
    System.out.println(n3);
}
```

연산자	연산자의 기능
&	비트 단위로 AND 연산을 한다. 예) n1 & n2;

각각의 비트를 대상으로 연산을 진행, 그리고 각 비트를 대상으로

진행된 연산 결과를 묶어서 하나의 연산 결과 반환

비트 연산자

연산자	연산자의 기능	결합 방향
&	비트 단위로 AND 연산을 한다. 예) n1 & n2;	
|	비트 단위로 OR 연산을 한다. 예) n1 | n2;	
^	비트 단위로 XOR 연산을 한다. 예) n1 ^ n2;	
~	피연산자의 모든 비트를 반전시켜서 얻은 결과를 반환 예) ~n;	

비트	~비트
1	0
0	1

비트 A	비트 B	비트 A & 비트 B
1	1	1
1	0	0
0	1	0
0	0	0

비트 A	비트 B	비트 A | 비트 B
1	1	1
1	0	1
0	1	1
0	0	0

비트 A	비트 B	비트 A ^ 비트 B
1	1	0
1	0	1
0	1	1
0	0	0

◈ BitOperator.java

```
1.  class BitOperator {
2.      public static void main(String[] args) {
3.          byte n1 = 5;     // 00000101
4.          byte n2 = 3;     // 00000011
5.          byte n3 = -1;    // 11111111
6.
7.          byte r1 = (byte)(n1 & n2);     // 00000001
8.          byte r2 = (byte)(n1 | n2);     // 00000111
9.          byte r3 = (byte)(n1 ^ n2);     // 00000110
10.         byte r4 = (byte)(~n3);         // 00000000
11.
12.         System.out.println(r1);    // 00000001은 1
13.         System.out.println(r2);    // 00000111은 7
14.         System.out.println(r3);    // 00000110은 6
15.         System.out.println(r4);    // 00000000은 0
16.     }
17. }
```

비트 연산자 예제

연산자	연산자의 기능	결합 방향
〈〈	• 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	➡

비트 쉬프트 연산자

◈ BitShiftOp.java

```
1.  class BitShiftOp {
2.      public static void main(String[] args) {
3.          byte num;
4.
5.          num = 2;     // 00000010
6.          System.out.print((byte)(num << 1) + " ");         // 00000100
7.          System.out.print((byte)(num << 2) + " ");         // 00001000
8.          System.out.print((byte)(num << 3) + " " + '\n');  // 00010000
9.
10.         num = 8;     // 00001000
11.         System.out.print((byte)(num >> 1) + " ");         // 00000100
12.         System.out.print((byte)(num >> 2) + " ");         // 00000010
13.         System.out.print((byte)(num >> 3) + " " + '\n');  // 00000001
14.
15.         num = -8;   // 11111000
16.         System.out.print((byte)(num >> 1) + " ");         // 11111100
17.         System.out.print((byte)(num >> 2) + " ");         // 11111110
18.         System.out.print((byte)(num >> 3) + " " + '\n');  // 11111111
19.     }
20. }
```

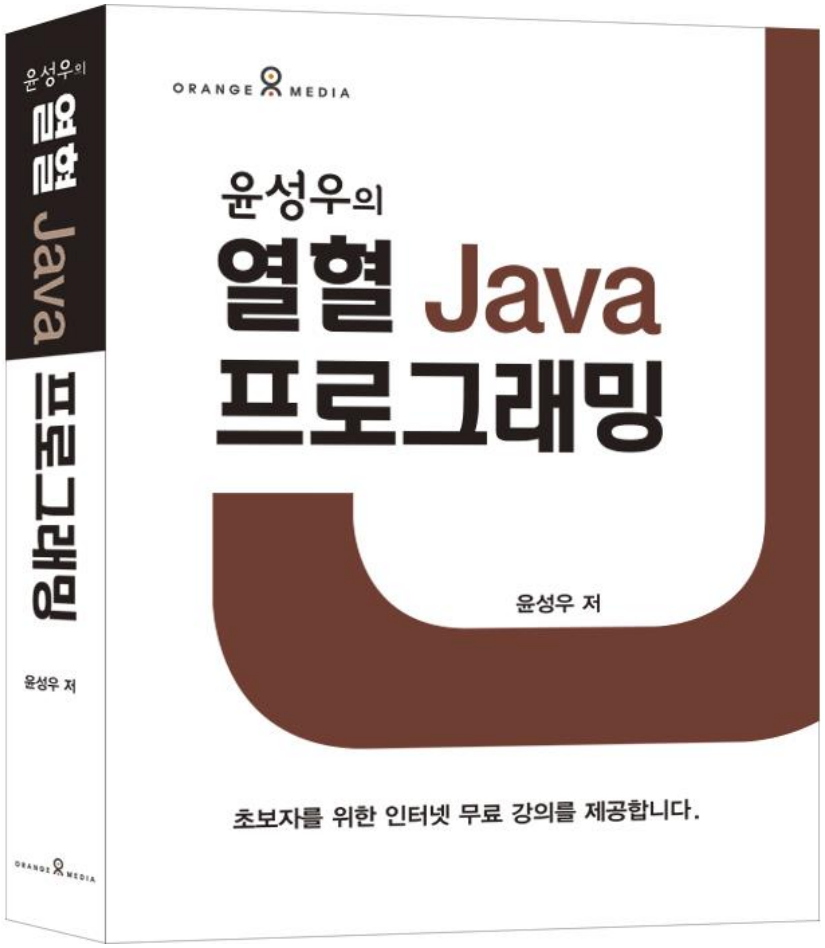
왼쪽으로의 쉬프트는 값의 2배 증가,

오른쪽으로의 쉬프트는 값을 2로 나눈 결과로 이어진다.

명령 프롬프트

```
C:₩JavaStudy>java BitShiftOp
4  8  16
4  2  1
-4  -2  -1

C:₩JavaStudy>
```

비트 쉬프트 연산자 예제



열혈 Java 프로그래밍

Chapter 05. 실행 흐름의 컨트롤

05-1. if 그리고 else

if문

```
if(true or false) {
    조건 true 시 실행되는 영역
}
```

ex1)

```
if(n1 < n2) {
    System.out.println("n1 > n2 is true");
}
```

ex2) if문에 속한 문장이 하나일 경우 중괄호 생략 가능

```
if(n1 < n2)
    System.out.println("n1 > n2 is true");
```

if ~ else문

```
if(true or false) {
    조건 true 시 실행되는 영역
} else {
    조건 false 시 실행되는 영역
}
```

ex)

```
if(n1 == n2) {
    System.out.println("n1 == n2 is true");
}
else {
    System.out.println("n1 == n2 is false");
}
```

if문과 마찬가지로 if절 또는 else 절에 속한 문장이 하나일 경우 중괄호 생략 가능

```java
public static void main(String[] args) {
    int n1 = 5;
    int n2 = 7;

    // if문
    if(n1 < n2) {
        System.out.println("n1 > n2 is true");
    }

    // if ~ else 문
    if(n1 == n2) {
        System.out.println("n1 == n2 is true");
    }
    else {
        System.out.println("n1 == n2 is false");
    }
}
```

if문, if ~ else문의 예

if ~ else if ~ else 문

```
if(...)
   System.out.println("...");
```

```
else if(...)
   System.out.println("...");
```

```
else if(...)
   System.out.println("...");
```

```
else
   System.out.println("...");
```

else if 절, 중간에 얼마든지 추가 가능

```
public static void main(String[] args) {
    int num = 120;

    if(num < 0)
        System.out.println("0 미만");
    else if(num < 100)
        System.out.println("0 이상 100 미만");
    else
        System.out.println("100 이상");
}
```

if ~ else if ~ else 문의 예

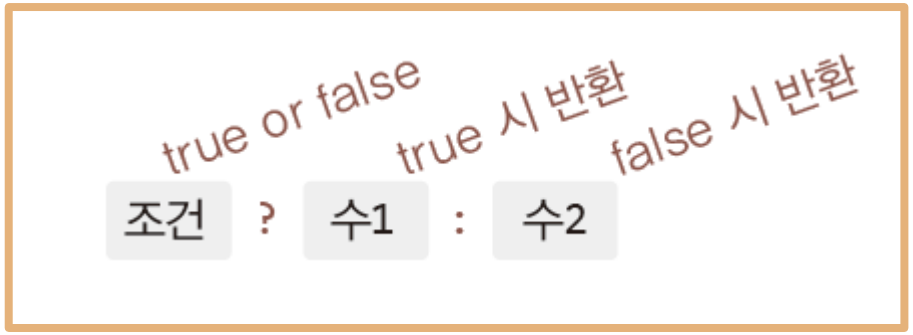
if ~ else if ~ else문과 if ~ else문의 관계

```
if(num < 0) {
   System.out.println("...");
}
else {
   if(num < 100)
      System.out.println("...");
   else
      System.out.println("...");
}
```

```
if(num < 0)
   System.out.println("...");
else
   if(num < 100)
      System.out.println("...");
   else
      System.out.println("...");
```

```
 if(num < 0)
    System.out.println("...");
 else if(num < 100)
    System.out.println("...");
 else
    System.out.println("...");
```

if ~ else문과 유사한 성격의 조건 연산자




```
ex1)
big = (num1 > num2) ? num1 : num2;


ex2)
diff = (num1 > num2) ? (num1 - num2) : (num2 - num1);
```

05-2. switch와 break

switch문의 기본 구성

```
switch(n) {
case 1:     n이 1이면 여기서부터 실행
     . . .
case 2:     n이 2이면 여기서부터 실행
     . . .
case 3:     n이 3이면 여기서부터 실행
     . . .
default:    해당하는 case 없으면 여기서부터 실행
     . . .
}
```

case와 default는 레이블!

따라서 실행 위치를 표시하는 용도로 사용될 뿐!

```
public static void main(String[] args) {
    int n = 3;

    switch(n) {
    case 1:
        System.out.println("Simple Java");
    case 2:
        System.out.println("Funny Java");
    case 3:
        System.out.println("Fantastic Java");
    default:
        System.out.println("The best programming language");
    }

    System.out.println("Do you like Java?");
}
```

switch문의 예

switch문 + break문

```
switch(n) {
case 1:
    . . . .
    break;
case 2:
    . . . .
    break;
case 3:
    . . . .
    break;
default:
    . . . .
}
```

case 1 영역

case 2 영역

case 3 영역

default 영역

break문이 실행되면 switch문을 빠져나간다.

```
public static void main(String[] args) {
    int n = 3;

    switch(n) {
    case 1:
        System.out.println("Simple Java");
        break;
    case 2:
        System.out.println("Funny Java");
        break;
    case 3:
        System.out.println("Fantastic Java");
        break;
    default:
        System.out.println("The best programming language");
    }

    System.out.println("Do you like Java?");
}
```


```
switch(n) {
case 1:
case 2:
case 3:
    System.out.println("case 1, 2, 3");
    break;
default:
    System.out.println("default");
}
```

switch + break 구성의 다른 예

switch문 + break문의 예

05-3.

for, while 그리고 do ~ while

while문

반복 조건

```
while(num < 5) {
    System.out.println("I like Java"+ num);
    num++;
}
```

반복 영역

먼저! 조건 검사

그리고 결과가 true이면 중괄호 영역 실행

```
public static void main(String[] args) {
    int num = 0;
    while(num < 5) {
        System.out.println("I like Java " + num);
        num++;
    }
}
```

do ~ while문

```
do {                                    반복 영역
    System.out.println("I like Java " + num);
    num++;
} while(num < 5);
        반복 조건
```

먼저! 중괄호 영역 실행

그리고 조건 검사 후 결과가 true이면 반복 결정

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

    do {
        System.out.println("I like Java " + num);
        num++;
    } while(num < 5);
}
```

for문 (while문과의 비교)

```
①
int num = 0;
②
while( num < 5 ) {
  System.out.println("...");
  num++;
}
```

```
①      ②      ③
for( int num = 0 ; num < 5 ; num++ ) {
  System.out.println("...");
}
```

① → 반복의 횟수를 세기 위한 변수

② → 반복의 조건

③ → 반복의 조건을 무너뜨리기 위한 연산

for문

```
          ①      ②    ④
for(int i = 0; i < 3; i++) {
    System.out.println(. . .);   ③
}
```

첫 번째 루프의 흐름 [i=0]
① → ② → ③ → ④

두 번째 루프의 흐름 [i=1]
② → ③ → ④

세 번째 루프의 흐름 [i=2]
② → ③ → ④

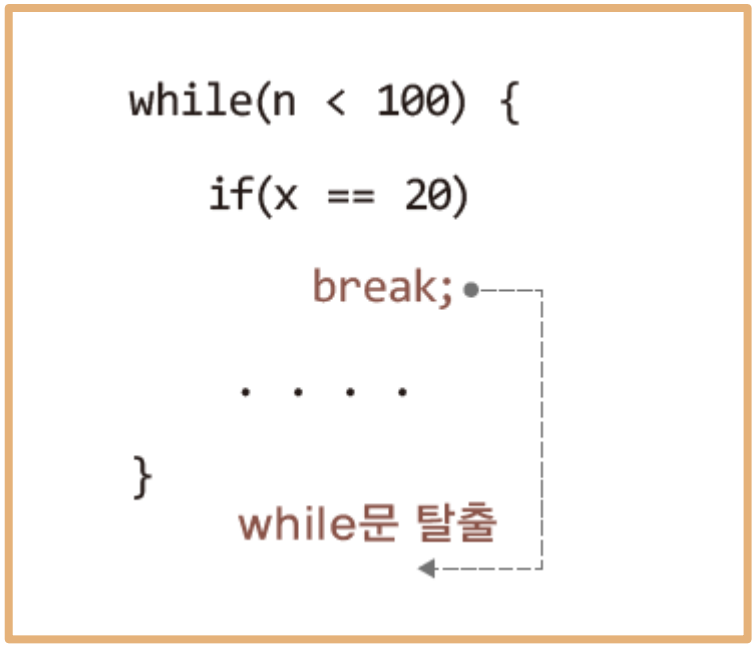
네 번째 루프의 흐름 [i=3]
② i=3이므로 탈출!

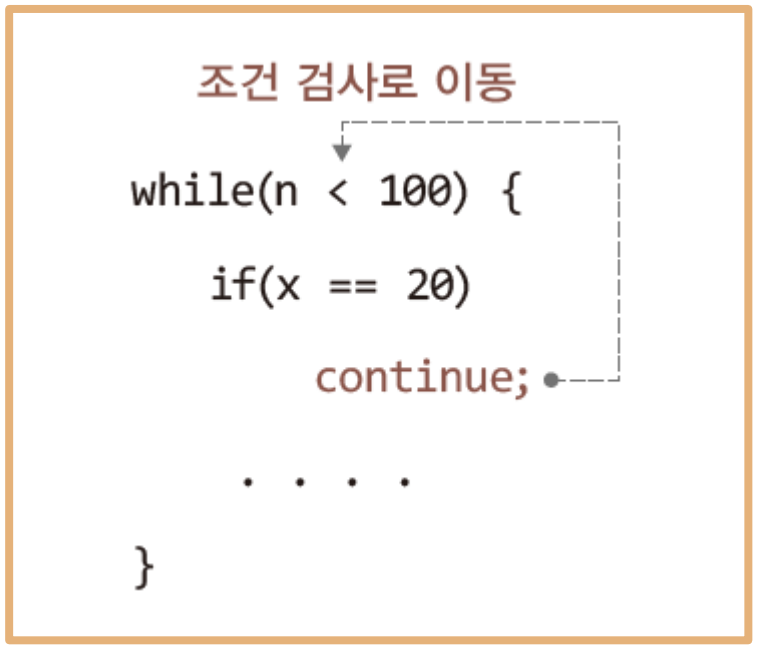
```
public static void main(String[] args) {
    for(int i = 0; i < 5; i++)
        System.out.println("I love Java " + i);
}
```

for문의 예

05-4. break & continue

break와 continue

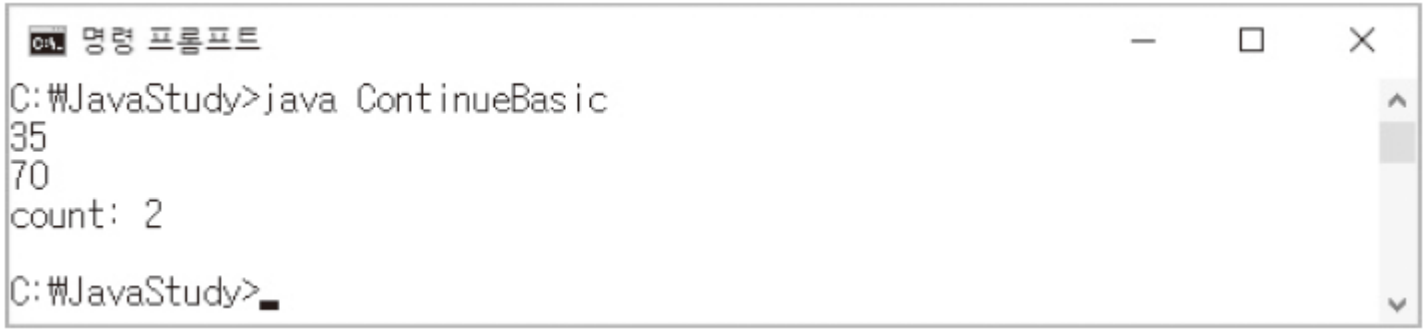




```
public static void main(String[] args) {
    int num = 1;
    boolean search = false;

    // 처음 만나는 5의 배수이자 7의 배수인 수를 찾는 반복문
    while(num < 100) {
        if(((num % 5) == 0) && ((num % 7) == 0)) {
            search = true;
            break;    // while문을 탈출
        }
        num++;
    }

    if(search)
        System.out.println("찾는 정수 : " + num);
    else
        System.out.println("5의 배수이자 7의 배수인 수를 찾지 못했습니다.");
}
```

break문의 예

```
public static void main(String[] args) {
    int num = 0;
    int count = 0;

    while((num++) < 100) {
        if(((num % 5) != 0) || ((num % 7) != 0))
            continue;    // 5와 7의 배수 아니라면 나머지 건너뛰고 위로 이동
        count++;    // 5와 7의 배수인 경우만 실행
        System.out.println(num);   // 5와 7의 배수인 경우만 실행
    }
    System.out.println("count: " + count);
}
```



continue문의 예

```
for( ; ; ) {
    ....
}
```

```
while(true) {
    ....
}
```

```
do {
    ....
} while(true)
```

무한루프

```
int num = 1;

while(true) {
    if(((num % 6) == 0) && ((num % 14) == 0))
        break;
    num++;
}
```

'6의 배수이면서 14의 배수인 가장 작은 자연수'를 찾는 반복문

무한루프와 break문의 예

05-5. 반복문의 중첩

```
for(…;…;…) {
    for(…;…;…) {
        . . .
    }
}
```

```
while(…) {
    for(…;…;…) {
        . . .
    }
}
```

```
do {
    for(…;…;…) {
        . . .
    }
} while(…);
```

```
for(…;…;…) {
    while(…) {
        . . .
    }
}
```

```
while(…) {
    while(…) {
        . . .
    }
}
```

```
do {
    while(…) {
        . . .
    }
} while(…);
```

```
for(…;…;…) {
    do {
        . . .
    } while(…);
}
```

```
while(…) {
    do {
        . . .
    } while(…);
}
```

```
do {
    do {
        . . .
    } while(…);
} while(…);
```

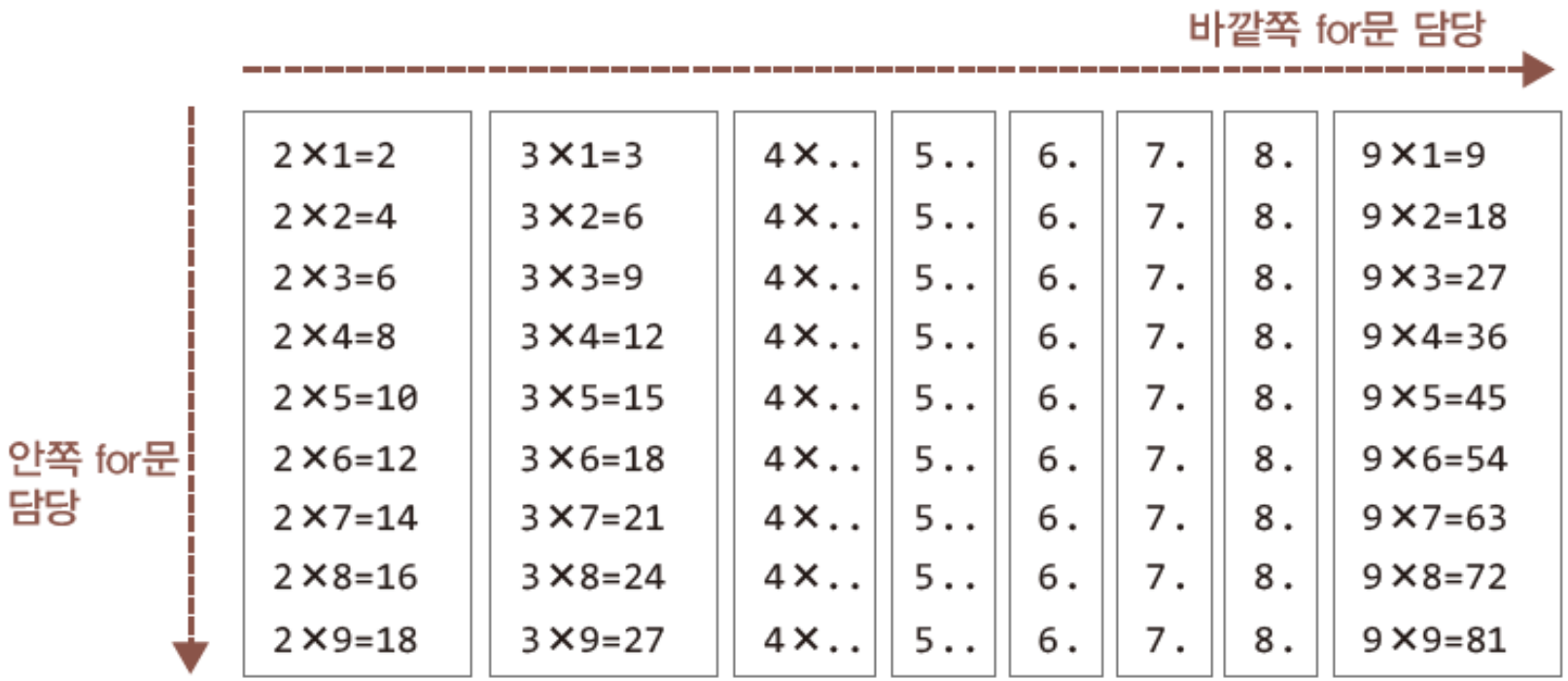
생각해 볼 수 있는 반복문의 중첩의 형태

◈ ForInFor.java

```
1.  class ForInFor {
2.      public static void main(String[] args) {
3.          for(int i = 0; i < 3; i++) {     // 바깥쪽 for문
4.              System.out.println("--------------------");
5.              for(int j = 0; j < 3; j++) {     // 안쪽 for문
6.                  System.out.print("[" + i + ", " + j + "] ");
7.              }
8.              System.out.print('\n');
9.          }
10.     }
11. }
```

```
명령 프롬프트
C:₩JavaStudy>java ForInFor
--------------------
[0, 0] [0, 1] [0, 2]
--------------------
[1, 0] [1, 1] [1, 2]
--------------------
[2, 0] [2, 1] [2, 2]

C:₩JavaStudy>
```

for문 중첩의 예



구구단 전체 출력을 위한 관찰

```
for(int i = 2; i < 10; i++) {    // 2단부터 9단까지 진행 위한 바깥쪽 for문
    for(int j = 1; j < 10; j++)    // 1부터 9까지의 곱을 위한 안쪽 for문
        System.out.println(i + " x " + j + " = " + (i * j));
}
```

구구단 출력 예제