03장 연산자(Operator)

3.1 연산자와 연산식

■ 연산이란?
- 데이터를 처리하여 결과를 산출하는 것
- 연산자(Operations)
  - 연산에 사용되는 표시나 기호(+, -, *, /, %, =, …)
- 피연산자(Operand): 연산 대상이 되는 데이터(리터럴, 변수)
- 연산식(Expressions)
  - 연산자와 피연산자를 이용하여 연산의 과정을 기술한 것
  - 반드시 하나의 값을 산출한다. 두개 이상의 값을 산출할 수 없다.

■ 연산자의 종류

연산자 종류	연산자	피연산자 수	산출값 타입	기능 설명
산술	+, -, *, /, %	이항	숫자	사칙연산 및 나머지 계산
부호	+, -	단항	숫자	음수와 양수의 부호
문자열	+	이항	문자열	두 문자열을 연결
대입	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=	이항	다양	우변의 값을 좌변의 변수에 대입
증감	++, --	단항	숫자	1 만큼 증가/감소
비교	==, !=, >, <, >=, <=, instanceof	이항	boolean	값의 비교
논리	!, &, |, &&, ||	단항 이항	boolean	논리적 NOT, AND, OR 연산
조건	(조건식) ? A : B	삼항	다양	조건식에 따라 A 또는 B 중 하나를 선택
비트	~, &, |, ^	단항 이항	숫자 blooean	비트 NOT, AND, OR, XOR 연산
쉬프트	>>, <<, >>>	이항	숫자	비트를 좌측/우측으로 밀어서 이동

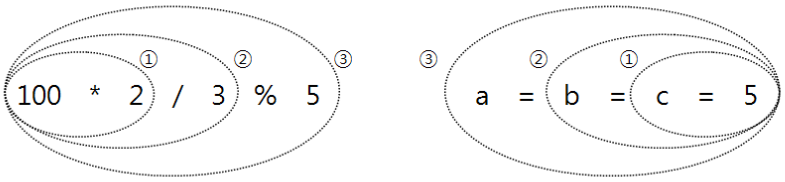
```
단항 연산자: ++y;
이항 연산자: x + y;
삼항 연산자: (sum>90) ? "A" : "B";

int result = x + y;
boolean result = (x+y) < 5;
```

3.2 연산의 방향과 우선순위

■ 연산자의 우선 순위에 따라 연산된다.
예) x > 0 && y < 0

■ 동일한 우선 순위의 연산자는 연산의 방향 존재

*, /, %는 같은 우선 순위를 갖고 있다. 이들 연산자는 연산 방향이 왼쪽에서 오른쪽으로 수행된다. 100 * 2 가 제일 먼저 연산되어 200 이 산출되고, 그 다음 200 / 3 이 연산되어 66 이 산출된다. 그 다음으로 66 % 5 가 연산되어 1 이 나온다.



하지만 단항 연산자(++, --, ~, !), 부호 연산자(+, -), 대입 연산자(=, +=, -=, ...)는 오른쪽에서 왼쪽(←)으로 연산된다. 예를 들어 다음 연산식을 보자.

예) a = b = c = 5;

■ 연산의 방향과 우선 순위

연산자	연산 방향	우선 순위
증감(++, --), 부호(+, -), 비트(~), 논리(!)	←	높음
산술(*, /, %)	→	
산술(+, -)	→	
쉬프트(<<, >>, >>>)	→	
비교(<, >, <=, >=, instanceof)	→	
비교(==, !=)	→	
논리(&)	→	
논리(^)	→	
논리(|)	→	
논리(&&)	→	
논리(||)	→	
조건(?:)	→	
대입(=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=)	←	낮음

3.3 단항 연산자

■ 피연산자가 단 하나뿐인 연산자를 말한다.

3.3.1 부호 연산자(+,-)

■ boolean 타입과 char 타입을 제외한 기본 타입에 사용 가능
■ 부호 연산자의 산출 타입은 int

3.3.2 증감 연산자(++,--)

■ 변수의 값을 1증가(++) 시키거나 1감소(--) 시키는 연산자
■ 증감 연산자가 변수 뒤에 있으면 다른 연산자 먼저 처리 후 증감 연산자 처리
■ 예) int x = 1;
```
int y = 1;
int result1 = ++x + 10; //result1=12
```

```
int result2 = y++ + 10; //result2=11
```

3.3.3 논리 부정 연산자(!)

■ Boolean type 에만 사용가능
예) boolean play = true;
play = !play; //play=false

3.3.4 비트 반전 연산자(~)

■ 정수 타입(byte, short, int, long)만 피연산자가 될 수 있다.
■ 비트값을 반전(0->1, 1->0)시킨다.

연산식		설명
~	10 (0 0 … 0 1 0 1 0)	산출결과: -11 (1 1 … 1 0 1 0 1)

■ 산출 타입은 int 타입이 된다.
■ 예) byte v1 = 10;
byte v2 = ~v1; //컴파일 에러
int v3 = ~v1;

3.4 이항 연산자

■ 피연산자가 두 개인 연산자를 말한다.

3.4.1 산출 연산자(+,-,*,/,%)

■ boolean 타입을 제외한 모든 기본 타입에 사용 가능
예) int result = num % 3; //num을 3으로 나눈 나머지
■ 산술 연산자의 특징
- byte + byte -> int + int = int //int 타입으로 변환 후, int 타입으로 산출
- int + long -> long + long = long
- int + double -> double + double = double
- int + char -> int + int = int

예)
```
byte byte1 = 1;
byte byte2 = 1;
byte byte3 = byte1 + byte2; //컴파일 에러
int result1 = byte1 + byte2;
byte byte4 = (byte)(byte1 + byte2);
char c2 = 'A';
char c3 = (char) (c2 + 1);  // 연산결과가 int 타입으로 강제타입변환을 해야 함.
```

(1) 오버플로우 탐지

■ 산출 타입으로 표현할 수 없는 값이 산출되었을 경우, 오버플로우가 발생하고 쓰레기값(엉뚱한 값)을 얻을 수 있다.

■ 예) int x = 1000000; //10^6
int y = 1000000; //10^6
int z = x * y; // z = -727379968
long z = x * y; // z = 10^6 * 10^6 = 10^{12}

(2) 정확한 계산은 정수 사용

■ 부동소수점 타입(float, double)은 0.1을 정확히 표현할 수 없어 근사치로 처리한다.

예) int apple = 1;
double pieceUnit = 0.1;
int number = 7;
double result = apple - number * pieceUnit; //result=0.299999999999993

(3) NaN과 Infinity 연산

■ 0으로 나누면 컴파일은 정상적으로 되지만, 실행 시 ArithmeticException(예외)이 발생한다.
■ 실수 타입인 0.0 또는 0.0f로 나누면 예외가 발생하지 않고, / 연산의 결과는 Infinity(무한대) 값을 가지며, % 연산의 결과는 NaN(Not a Number)을 가진다.
■ /와 % 연산의 결과가 Infinity 또는 NaN인지 확인하려면 Double.isInfinite()와 Double.isNaN() 메소드를 이용하면 된다.
■ 예) double x = 5 / 0 // 예외 발생
double y = 5 / 0.0 //Infinity
double z = 5 % 0.0 //NaN
Double.isInfinite(y); //true
Double.isNaN(z); //true

(4) 입력값의 NaN 검사

■ 예) String userInput = "NaN";
double val = Double.valueOf(userInput);
if(Double.isNaN(val)) {
...
}

3.4.2 문자열 연결 연산자(+)

■ 피연산자 중 문자열이 있으면 문자열로 결합
■ 문자열과 숫자가 혼합된 + 연산식은 왼쪽에서부터 오른쪽으로 연산이 진행된다.

[StringConcatExample.java] 문자열 연결 연산자

```
package sec04.exam02_string_concat;

public class StringConcatExample {
          public static void main(String[] args) {
                    String str1 = "JDK" + 6.0;
                    String str2 = str1 + " 특징";
                    System.out.println(str2); //JDK6.0 특징

                    String str3 = "JDK" + 3 + 3.0;
                    String str4 = 3 + 3.0 + "JDK";
```

```
                System.out.println(str3); //JDK33.0
                System.out.println(str4); //6.0JDK
        }
}
```

3.4.3 비교(관계) 연산자(<,<=,>,>=,==,!=)

- 대소(<, <=, >, >=) 또는 동등(==, !=) 비교해 boolean 타입인 true/false 산출

구분	연산식			설명
동등 비교	피연산자	==	피연산자	두 피 연산자의 값이 같은지를 검사
	피연산자	!=	피연산자	두 피 연산자의 값이 다른지를 검사
크기 비교	피연산자	>	피연산자	피 연산자 1 이 큰지를 검사
	피연산자	>=	피연산자	피 연산자 1 이 크거나 같은지를 검사
	피연산자	<	피연산자	피 연산자 1 이 작은지를 검사
	피연산자	<=	피연산자	피 연산자 1 이 작거나 같은지를 검사

- 동등 비교 연산자는 모든 타입에 사용, 크기 비교 연산자는 boolean 타입 제외한 모든 기본 타입에 사용
- 흐름 제어문인 조건문(if), 반복문(for, while)에서 주로 이용 -> 실행 흐름을 제어할 때 사용
- String 타입인 문자열의 값을 비교할 때에는 == 연산자(주소를 비교함) 대신에 equals() 메소드를 사용해야 한다.

```
'A' < 'B'  //(65 < 66)
'A' == 65  //true
3 == 3.0  //true, int타입(3)이 double타입(3.0)으로 변환한 다음 비교한다.
0.1 == 0.1f  //false, 0.1f는 0.1의 근사값(0.10000000149011612)으로 표현되어 0.1보다 큰값이 되어 버린다.
(float)(0.1) == 0.1f  //true, double형(0.1)을 float형으로 강제형변환하여 비교한다.

if(a == b){...}  //a와 b가 같으면
 if(a != b){...}  //a와 b가 같지 않으면

 String strVal1 = "신용권";
 String strVal2 = "신용권";  //문자열 리터럴이 동일하다면 동일한 String 객체를 참조한다.
 String strVal3 = new String("신용권");  //객체 생성 연산자인 new로 생성한 새로운 String 객체의 번지값을 가진다.
 strVal1 == strVal2  //true
 strVal2 == strVal3  //false
 strVal1.equals(strVal2)  //true
 strVal2.equals(strVal3)  //true
```

[StringEqualsExample.java]

```
package sec04.exam03_compare;

public class StringEqualsExample {
        public static void main(String[] args) {
                String strVar1 = "신민철";
                String strVar2 = "신민철";
                String strVar3 = new String("신민철");

                System.out.println(strVar1 == strVar2);  //true
                System.out.println(strVar1 == strVar3);  //false
                System.out.println();
                System.out.println(strVar1.equals(strVar2));  //true
                System.out.println(strVar1.equals(strVar3));  //true
```

```
        }
}
```

3.4.4 논리 연산자(&&,||,&,|,^,!)

- 논리곱(&&), 논리합(||), 배타적 논리합(^) ,논리 부정(!) 연산 수행
- 피연산자는 boolean 타입만 사용 가능

구분	연산식			결과	설명
AND (논리곱)	true	&& 또는 &	true	true	피 연사자 모두가 true 일 경우에만 연산 결과는 true
	true		false	false	
	false		true	false	
	false		false	false	
OR (논리합)	true	|| 또는 |	true	true	피 연산자 중 하나만 true 이면 연산 결과는 true
	true		false	true	
	false		true	ture	
	false		false	false	
XOR (배타적 논리합)	true	^	true	false	피 연산자가 하나는 ture 이고 다른 하나가 false 일 경우에만 연산 결과는 true
	true		false	true	
	false		true	ture	
	false		false	false	
NOT (논리부정)		!	true	false	피 연산자의 논리값을 바꿈
			false	true	

3.4.5 비트 연산자(&,|,^,~,<<,>>,>>>)

- 비트(bit) 단위로 연산 하므로 0과 1이 피연산자
  - 0과 1로 표현이 가능한 정수 타입만 비트 연산 가능
  - 실수 타입인 float과 double은 비트 연산 불가
- 종류
  - 비트 논리 연산자(&, |, ^, ~)
  - 비트 이동 연산자(<<, >>, >>>)

(1) 비트 논리 연산자(&,|,^)

- 피 연산자가 boolean타입일 경우 일반 논리 연산자
- 피연산자가 정수 타입일 경우 비트 논리 연산자로 사용
- 비트 연산자는 피연산자를 int타입으로 자동 타입 변환 후 연산 수행

구분	연산식			결과	설명
AND (논리곱)	1	&	1	1	두 비트가 모두가 1 일 경우에만 연산 결과는 1
	1		0	0	
	0		1	0	
	0		0	0	
OR (논리합)	1	|	1	1	두 비트 중 하나만 1 이면 연산 결과는 1
	1		0	1	
	0		1	1	
	0		0	0	
XOR (베타적 논리합)	1	^	1	0	두 비트 중 하나는 1 이고 다른 하나가 0 일 경우 연산 결과는 1
	1		0	1	
	0		1	1	
	0		0	0	
NOT (논리부정)		~	1	0	보수
			0	1	

■ 예) byte num1 = 10; //1010
byte num2 = 6; //0110
byte result = num1 & num2; //컴파일 에러, int result = num1 & num2;, result=2

(2) 비트 이동 연산자(<<,>>,>>>)

■ 정수 데이터의 비트를 좌측 또는 우측으로 밀어 이동시키는 연산 수행

구분	연산식			설명
이동 (쉬프트)	a	<<	b	정수 a 의 각 비트를 b 만큼 왼쪽으로 이동 (빈자리는 0 으로 채워진다.)
	a	>>	b	정수 a 의 각 비트를 b 만큼 오른쪽으로 이동 (빈자리는 정수 a 의 최상위 부호 비트(MSB)와 같은 값으로 채워진다.)
	a	>>>	b	정수 a 의 각 비트를 오른쪽으로 이동 (빈자리는 0 으로 채워진다.)

3.4.6 대입 연산자(=,+=,-=,*=,/=,%=,&=,^=,|=,<<=,>>=,>>>=)

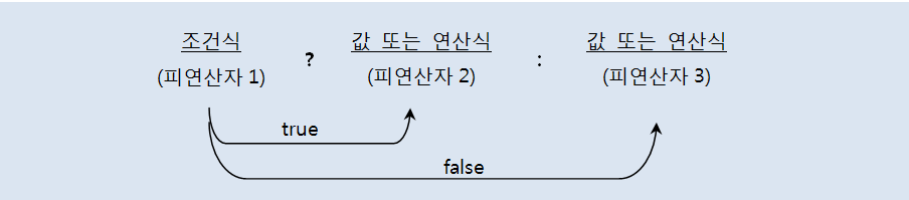
■ 오른쪽 피연산자의 값을 좌측 피연산자인 변수에 저장
■ 모든 연산자들 중 가장 낮은 연산 순위 -> 제일 마지막에 수행
■ 종류: 단순 대입 연산자, 복합 대입 연산자(정해진 연산을 수행한 후 결과를 변수에 저장)

구분	연산식			설명
단순 대입 연산자	변수	=	피연산자	우측의 피연산자의 값을 변수에 저장
복합 대입 연산자	변수	+=	피연산자	우측의 피연산자의 값을 변수의 값과 더한 후에 다시 변수에 저장 (변수=변수+피연산자 와 동일)
	변수	-=	피연산자	우측의 피연산자의 값을 변수의 값에서 뺀 후에 다시 변수에 저장 (변수=변수-피연산자 와 동일)
	변수	*=	피연산자	우측의 피연산자의 값을 변수의 값과 곱한 후에 다시 변수에 저장 (변수=변수*피연산자 와 동일)
	변수	/=	피연산자	우측의 피연산자의 값으로 변수의 값을 나눈 후에 다시 변수에 저장 (변수=변수/피연산자 와 동일)
	변수	%=	피연산자	우측의 피연산자의 값으로 변수의 값을 나눈 후에 나머지를 변수에 저장 (변수=변수%피연산자 와 동일)
	변수	&=	피연산자	우측의 피연산자의 값과 변수의 값을 & 연산 후 결과를 변수에 저장 (변수=변수&피연산자 와 동일)
	변수	|=	피연산자	우측의 피연산자의 값과 변수의 값을 | 연산 후 결과를 변수에 저장 (변수=변수|피연산자 와 동일)
	변수	^=	피연산자	우측의 피연산자의 값과 변수의 값을 ^ 연산 후 결과를 변수에 저장 (변수=변수^피연산자 와 동일)
	변수	<<=	피연산자	우측의 피연산자의 값과 변수의 값을 << 연산 후 결과를 변수에 저장 (변수=변수<<피연산자 와 동일)
	변수	>>=	피연산자	우측의 피연산자의 값과 변수의 값을 >> 연산 후 결과를 변수에 저장 (변수=변수>>피연산자 와 동일)
	변수	>>>=	피연산자	우측의 피연산자의 값과 변수의 값을 >>> 연산 후 결과를 변수에 저장 (변수=변수>>>피연산자 와 동일)

■ 예) a+=b; // a = a + b;
a-=b; // a = a - b;
a*=b; // a = a * b;
a/=b; // a = a / b;
a%=b; // a = a % b;

3.5 삼항(조건) 연산자

■ 세 개의 피연산자를 필요로 하는 연산자
■ 앞의 조건식 결과에 따라 콜론 앞 뒤의 피연산자 선택 -> 조건 연산식



■ 예) int score = 95;
char grade = (score>90) ? 'A':'B'; //grade='A'

[ConditionalOperationExample.java] 삼항 연산자

```
01  package sec04.exam07_conditional;
02
03  public class ConditionalOperationExample {
04          public static void main(String[] args) {
05                  int score = 85;
06                  char grade = (score > 90) ? 'A' : ( (score > 80) ? 'B' : 'C' );
07                  System.out.println(score + "점은 " + grade + "등급입니다.");
08          }
```

```
09    }
```

[과제] 최대값 및 최소값 구하기

키보드로 3개의 정수를 입력 받았을때 최대값과 최소값을 구하는 프로그램을 조건 연산자를 활용해서 작성하라.

[MinMax.java]

```
package verify;

import java.util.Scanner;
 |
public class MinMax {

          public static void main(String[] args) {
                    // TODO Auto-generated method stub

                    System.out.print("3개의 정수를 입력 하세요?");
                    int n1, n2, n3, max, min;

                    Scanner sc = new Scanner(System.in);
                    n1 = sc.nextInt(); // 첫번째 입력값
                    n2 = sc.nextInt(); // 둘번째 입력값
                    n3 = sc.nextInt(); // 세번째 입력값

                    // 아래에 코드를 작성하세요.
                    // ?

                    System.out.println("max="+max);
                    System.out.println("min="+min);
          }
}
```

[과제] 확인문제

1. 연산자와 연산식에 대한 설명 중 틀린 것은 무엇입니까?

(1) 연산자는 피연산자의 수에 따라 단항, 이항, 삼항 연산자로 구분된다.
(2) 비교 연산자와 논리 연산자의 산출 타입은 boolean(true/false)이다.
(3) 연산식은 하나 이상의 값을 산출할 수도 있다.
(4) 하나의 값이 올 수 있는 자리라면 연산식도 올 수 있다.

2. 다음 코드를 실행했을 때 출력 결과는 무엇입니까?

[Exercise02.java]

```
package verify;
public class Exercise02 {
          public static void main(String[] args) {
                    int x = 10;
                    int y = 20;
                    int z = (++x) + (y--);
                    System.out.println(z);
```

```
        }
}
```

3. 다음 코드를 실행했을 때 출력 결과는 무엇입니까?

[Exercise03.java]

```
package verify;
public class Exercise03 {
        public static void main(String[] args) {
                int score = 85;
                String result = (!(score>90))? "가":"나";
                System.out.println(result);
        }
}
```

4. 534자루의 연필을 30명의 학생들에게 똑같은 개수로 나누어 줄 때 학생당 몇 개를 가질 수 있고, 최종적으로 몇 개가 남는지를 구하는 코드입니다. (#1)과 (#2)에 들어갈 알맞는 코드를 작성하세요.

[Exercise04.java]

```
package verify;

public class Exercise04 {
        public static void main(String[] args) {
                int pencils = 534;
                int students = 30;

                // 학생 한 명이 가지는 연필 수
                int pencilsPerStudent = (       #1       );
                System.out.println(pencilsPerStudent);  // 17

                // 남은 연필 수
                int pencilsLeft = (       #2       );
                System.out.println(pencilsLeft);  // 24
        }
}
```

5. 다음은 십의 자리 이하를 버리는 코드입니다. 변수 value의 값이 356이라면 300이 나올 수 있도록 (#1)에 알맞은 코드를 작성하세요. (산출 연산자만 사용하세요)

[Exercise05.java]

```
package verify;

public class Exercise05 {
        public static void main(String[] args) {
                int value = 356;
                System.out.println(          #1          );  // 300
        }
}
```

6. 다음 코드는 사다리꼴의 넓이를 구하는 코드입니다. 정확히 소수자릿수가 나올 수 있도록 (#1)에 알맞는 코드를 작성하세요.

[Exercise06.java]

```
package verify;

public class Exercise06 {
	public static void main(String[] args) {
		int lengthTop = 5;
		int lengthBottom = 10;
		int height = 7;
		double area = (		#1		); // 52.5
		System.out.println(area);
	}
}
```

7. 다음 코드는 비교 연산자의 논리 연산자의 복합 연산식입니다. 연산식의 출력 결과를 괄호() 속에 넣으세요.

[Exercise07.java]

```
package verify;

public class Exercise07 {
	public static void main(String[] args) {
		int x = 10;
		int y = 5;

		System.out.println((x > 7) && (y <= 5));  // (   #1   )
		System.out.println((x % 3 == 2) || (y % 2 != 1));  // (   #2   )
	}
}
```

8. 다음은 % 연산을 수행한 결과값에 10을 더하는 코드입니다. NaN 값을 검사해서 올바른 결과가 출력될 수 있도록 (#1)에 들어갈 NaN을 검사하는 코드를 작성하세요.

[Exercise08.java]

```
package verify;

public class Exercise08 {
	public static void main(String[] args) {
		double x = 5.0;
		double y = 0.0;

		double z = 5 % y;

		if (		#1		) {
			System.out.println("0.0으로 나눌 수 없습니다.");
		} else {
			double result = z + 10;
			System.out.println("결과: " + result);
		}
	}
}
```