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

연산자와 연산식

❖ 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

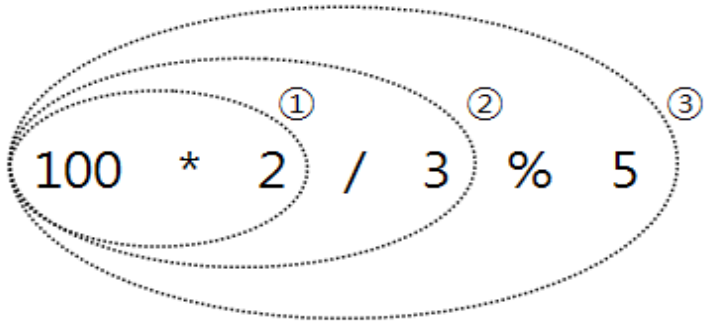
❖ 연산의 방향과 우선 순위

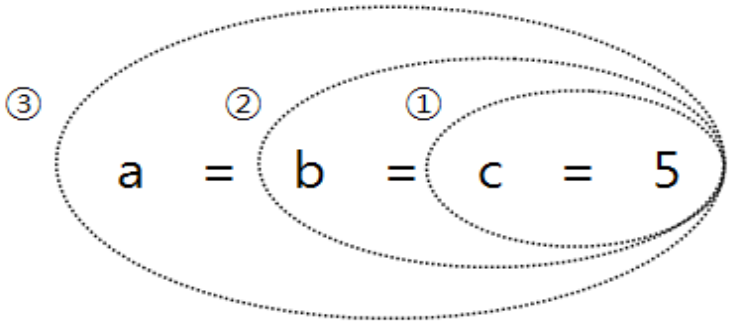
- 연산자의 우선 순위에 따라 연산된다.

```
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.

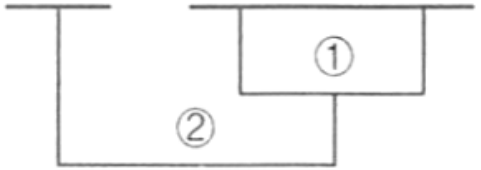
연산의 방향과 우선 순위

❖ 연산의 방향과 우선 순위

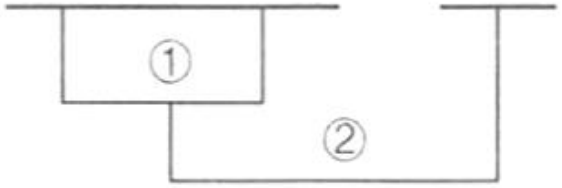
연산자	연산 방향	우선 순위
증감(++, --), 부호(+, -), 비트(~), 논리(!)	←	높음
산술(*, /, %)	→	
산술(+, -)	→	
쉬프트(<<, >>, >>>)	→	
비교(<, >, <=, >=, instanceof)	→	
비교(==, !=)	→	
논리(&)	→	
논리(^)	→	
논리(|)	→	
논리(&&)	→	
논리(||)	→	
조건(?:)	→	
대입(=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=)	←	낮음

❖ 우선 순위 조정

```
int var1 = 1;
int var2 = 3;
int var3 = 2;
int result = var1 + var2 * var3;
```




```
int result = (var1 + var2) * var3;
```



❖ 연산 방향과 우선순위

1. 단항, 이항, 삼항 연산자 순으로 우선순위를 가진다.
2. 산술, 비교, 논리, 대입 연산자 순으로 우선순위를 가진다.
3. 단항과 대입 연산자를 제외한 모든 연산의 방향은 왼쪽에서 오른쪽이다(→).
4. 복잡한 연산식에는 괄호( )를 사용해서 우선순위를 정해준다.

단항 연산자

❖ 단항연산자란?

- 피연산자가 1개인 연산자

- 단항 연산자의 종류
  - 부호 연산자: +, -
  - boolean 타입과 char 타입을 제외한 기본 타입에 사용 가능

  - 부호 연산자의 산출 타입은 int

```
short s = 100;
short result = -s;  //컴파일 에러
```

```
short s = 100;
int result3 = -s;
```

단항 연산자

❖ 부호연산자 : SignOperatorExample.java

```
public class SignOperatorExample {
   public static void main(String[] args) {
      int x = -100;
      int result1 = +x;
      int result2 = -x;

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

      short s = 100;
      // short result3 = -s; //컴파일 에러
      int result3 = -s;
      System.out.println("result3=" + result3);

   }
}
```

단항 연산자

❖ 단항 연산자의 종류

- ○ 증감 연산자: ++, --
  - ▪ 변수의 값을 1증가 시키거나 (++) 1 감소 (--) 시키는 연산자
  - ▪ 증감 연산자가 변수 뒤에 있으면 다른 연산자 먼저 처리 후 증감 연산자 처리

- ○ ++a; --a; a++; a--;

연산식		설명
++	피연산자	다른 연산을 수행하기 전에 피연산자의 값을 1 증가시킴
--	피연산자	다른 연산을 수행하기 전에 피연산자의 값을 1 감소시킴
피연산자	++	다른 연산을 수행한 후에 피연산자의 값을 1 증가시킴
피연산자	--	다른 연산을 수행한 후에 피연산자의 값을 1 감소시킴

단항 연산자

❖ 단항 연산자의 종류

- 증감 연산자: ++, --

++i;
i++; } 모두 i = i + 1; 로 동일

--i;
i--; } 모두 i = i - 1; 로 동일

```
int x = 1;
int y = 1;
int reuslt1 = ++x + 10;
int result2 = y++ + 10;
```

단항 연산자

❖ 증감연산자 : IncreaseDecreaseOperatorExample.java

```
public class IncreaseDecreaseOperatorExample {
   public static void main(String[] args) {
      int x = 10;
      int y = 10;
      int z;

      System.out.println("----------------------");
      x++;
      ++x;
      System.out.println("x=" + x);

      System.out.println("----------------------");
      y--;
      --y;
      System.out.println("y=" + y);

      System.out.println("----------------------");
      z = x++;
      System.out.println("z=" + z);
      System.out.println("x=" + x);
```

❖ 증감연산자 : IncreaseDecreaseOperatorExample.java

```
        System.out.println("-----------------------");
        z = ++x;
        System.out.println("z=" + z);
        System.out.println("x=" + x);

        System.out.println("-----------------------");
        z = ++x + y++;
        System.out.println("z=" + z);
        System.out.println("x=" + x);
        System.out.println("y=" + y);
    }
}
```

단항 연산자

❖ 단항 연산자의 종류

- ㅇ 논리 부정 연산자: !
  - ▪ boolean type 에만 사용가능

연산식		설명
!	피연산자	피연산자가 true 이면 false 값을 산출 피연산자가 false 이면 true 값을 산출

❖ 논리 부정 연산자 : DenyLogicOperatorExample.java

```
public class DenyLogicOperatorExample {
   public static void main(String[] args) {
      boolean play = true;
      System.out.println(play);

      play = !play;
      System.out.println(play);

      play = !play;
      System.out.println(play);
   }
}
```

단항 연산자

❖ 단항 연산자의 종류

- 비트 반전 연산자: ~
  - byte, short, int, long 타입만 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)

- 연산결과를 int 타입을 가짐

```
byte v1 = 10;
byte v2 = ~v1; //컴파일 에러
```

```
byte v1 = 10;
int v2 = ~v1;
```

단항 연산자

❖ 비트 반전 연산자: BitReverseOperatorExample.java

```
public class BitReverseOperatorExample {

   public static String toBinaryString(int value) {
      String str = Integer.toBinaryString(value);
      while (str.length() < 32) {
         str = "0" + str;
      }
      return str;
   }
```

단항 연산자

❖ 비트 반전 연산자: BitReverseOperatorExample.java

```
   public static void main(String[] args) {
      int v1 = 10;
      int v2 = ~v1;
      int v3 = ~v1 + 1;
      System.out.println(toBinaryString(v1) + " (십진수: " + v1 + ")");
      System.out.println(toBinaryString(v2) + " (십진수: " + v2 + ")");
      System.out.println(toBinaryString(v3) + " (십진수: " + v3 + ")");
      System.out.println();

      int v4 = -10;
      int v5 = ~v4;
      int v6 = ~v4 + 1;
      System.out.println(toBinaryString(v4) + " (십진수: " + v4 + ")");
      System.out.println(toBinaryString(v5) + " (십진수: " + v5 + ")");
      System.out.println(toBinaryString(v6) + " (십진수: " + v6 + ")");
   }

}
```