

2025년 상반기 K-디지털 트레이닝

# 연산자

[KB] IT's Your Life



### ♡ 부호 연산자

○ 부호 연산자는 변수의 부호를 유지하거나 변경

연산식		설명	
+ 피연산자		피연산자의 부호 유지	
_	피연산자	피연산자의 부호 변경	

## ☑ 증감 연산자

○ 증감 연산자는 변수의 값을 1 증가시키거나 1 감소시킴

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

# ch03.sec01.SignOperatorExample.java

```
package ch03.sec01;
public class SignOperatorExample {
  public static void main(String[] args) {
    int x = -100;
   X = -X;
    System.out.println("x: " + x);
    byte b = 100;
    int y = -b;
    System.out.println("y: " + y);
```

```
x: 100
y: -100
```

## 부호/증감 연산자

# ch03.sec01.IncreaseDecreaseOperatorExample.java

```
package ch03.sec01;
                                                           Z = ++X;
public class IncreaseDecreaseOperatorExample {
                                                           System.out.println("z=" + z);
                                                           System.out.println("x=" + x);
 public static void main(String[] args) {
                                                           System.out.println("----");
   int x = 10;
   int y = 10;
   int z;
                                                           Z = ++X + V++;
                                                           System.out.println("z=" + z);
                                                           System.out.println("x=" + x);
   X++;
                                                           System.out.println("y=" + y);
   ++X;
   System.out.println("x=" + x);
   System.out.println("----");
                                                                x = 12
   y--;
                                                                y=8
   --y;
   System.out.println("y=" + y);
                                                                z = 12
   System.out.println("----");
                                                                x = 13
   Z = X++;
                                                                z = 14
   System.out.println("z=" + z);
                                                                x = 14
   System.out.println("x=" + x);
   System.out.println("----");
                                                                z = 23
                                                                x = 15
                                                                y=9
```

## ☑ 산술 연산자

○ 더하기(+), 빼기(-), 곱하기(\*), 나누기(/), 나머지(%)로 총 5개

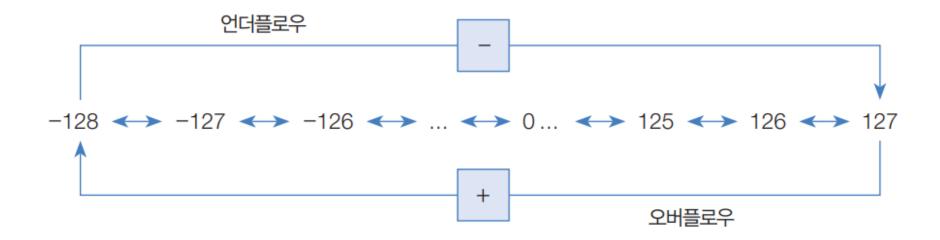
연산식			설명	
피연산자	+	피연산자	덧셈 연산	
피연산자	_	피연산자	뺄셈 연산	
피연산자	*	피연산자	곱셈 연산	
피연산자	/	피연산자	나눗셈 연산	
피연산자	%	피연산자	나눗셈의 나머지를 산출하는 연산	

# ch03.sec02.ArithmeticOperatorExample.java

```
package ch03.sec02;
public class ArithmeticOperatorExample {
 public static void main(String[] args) {
   byte v1 = 10;
   byte v2 = 4;
   int v3 = 5;
   long v4 = 10L;
   int result1 = v1 + v2; //모든 피연산자는 int 타입으로 자동 변환 후 연산
   System.out.println("result1: " + result1);
   long result2 = v1 + v2 - v4; //모든 피연산자는 long 타입으로 자동 변환 후 연산
   System.out.println("result2: " + result2);
   double result3 = (double) v1 / v2; //double 타입으로 강제 변환 후 연산
   System.out.println("result3: " + result3);
   int result4 = v1 % v2;
   System.out.println("result4: " + result4);
                                               result1: 14
                                               result2: 4
                                               result3: 2.5
                                               result4: 2
```

## 오버플로우와 언더플로우

- 오버플로우 overflow
  - 타입이 허용하는 최대값을 벗어나는 것
- ♥ 언더플로우 underflow
  - 타입이 허용하는 최소값을 벗어나는 것



# 오버플로우와 언더플로우

# ch03.sec03.OverflowUnderflowExample.java

```
package ch03.sec03;
public class OverflowUnderflowExample {
 public static void main(String[] args) {
   byte var1 = 125;
   for(int i=0; i<5; i++) { //{ }를 5번 반복 실행
     var1++; //++ 연산은 var1의 값을 1 증가시킨다.
     System.out.println("var1: " + var1);
   System.out.println("----");
   byte var2 = -125;
   for(int i=0; i<5; i++) { //{ }를 5번 반복 실행
     var2--; //-- 연산은 var2의 값을 1 감소시킨다.
     System.out.println("var2: " + var2);
```

```
var1: 126
var1: 127
var1: -128
var1: -127
var1: -126
------
var2: -126
var2: -127
var2: -127
var2: -128
var2: 127
var2: 126
```

### ☑ 정수 연산

- 산술 연산을 정확하게 계산하려면 실수 타입을 사용하지 않는 것이 좋음
- 정확한 계산이 필요하면 정수 연산으로 변경

```
## AccuracyExample1.java

package ch03.sec04;

public class AccuracyExample1 {

public static void main(String[] args) {

int apple = 1;

double pieceUnit = 0.1;

int number = 7;

double result = apple - number*pieceUnit;

System.out.println("사과 1개에서 남은 양: " + result);

}

**The continuation of the cont
```

실행 결과

사과 1개에서 남은 양: 0.29999999999993

# ch03.sec04.AccuracyExample1.java

```
package ch03.sec04;
public class AccuracyExample1 {
 public static void main(String[] args) {
   int apple = 1;
   double pieceUnit = 0.1;
   int number = 7;
   double result = apple - number*pieceUnit;
   System.out.println("사과 1개에서 남은 양: " + result);
```

사과 1개에서 남은 양: 0.299999999999993

# ch03.sec04.AccuracyExample2.java

```
package ch03.sec04;
public class AccuracyExample2 {
 public static void main(String[] args) {
   int apple = 1;
   int totalPieces = apple * 10;
   int number = 7;
   int result = totalPieces - number;
   System.out.println("10조각에서 남은 조각: " + result);
   System.out.println("사과 1개에서 남은 양: " + result/10.0);
```

```
10조각에서 남은 조각: 3
사과 1개에서 남은 양: 0.3
```

## ○ 나눗셈 연산에서 예외 방지하기

- 나눗셈(/) 또는 나머지(%) 연산에서 좌측 피연산자가 정수이고 우측 피연산자가 0일 경우 ArithmeticException 발생
- 좌측 피연산자가 실수이거나 우측 피연산자가 0.0 또는 0.0f이면 예외가 발생하지 않고 연산의 결과는 Infinity(무한대) 또는 NaN(Not a Number)이 됨

```
5 / 0.0 → Infinity
5 % 0.0 → NaN
```

- Infinity 또는 NaN 상태에서 계속해서 연산을 수행하면 안 됨
- o Double.isInfinite()와 Double.isNaN()를 사용해 /와 % 연산의 결과가 Infinity 또는 NaN인지 먼저 확인 하고 다음 연산을 수행하는 것이 좋음

# ch03.sec05.InfinityAndNaNCheckExample.java

```
package ch03.sec05;
public class InfinityAndNaNCheckExample {
 public static void main(String[] args) {
   int x = 5;
   double y = 0.0;
   double z = x / y;
   //double z = x % y;
   //잘못된 코드
   System.out.println(z + 2);
   //알맞은 코드
   if(Double.isInfinite(z) || Double.isNaN(z)) {
     System.out.println("값 산출 불가");
   } else {
     System.out.println(z + 2);
```

```
Infinity
값 산출 불가
```

# ☑ 비교 연산자

- 비교 연산자는 동등(==, !=) 또는 크기(<, <=, >, >=)를 평가해서 boolean 타입인 true/false를 산출
- 흐름 제어문인 조건문(if), 반복문(for, while)에서 실행 흐름을 제어할 때 주로 사용

구분	연산식			설명	
동등	피연산자1	==	피연산자2	두 피연산자의 값이 같은지를 검사	
비교	피연산자1	!= 피연산자2		두 피연산자의 값이 다른지를 검사	
<i>크</i> 기 비교	피연산자1	>	피연산자2	피연산자1이 큰지를 검사	
	피연산자1	>=	피연산자2	피연산자1이 크거나 같은지를 검사	
	피연산자1	<	피연산자2	피연산자1이 작은지를 검사	
	피연산자1	<b>&lt;=</b>	피연산자2	피연산자1이 작거나 같은지를 검사	

○ 문자열을 비교할 때는 동등(==, !=) 연산자 대신 equals()와 !equals()를 사용

# ch03.sec06.CompareOperatorExample.java

```
package ch03.sec06;
                                                                   int num3 = 1;
                                                                   double num4 = 1.0;
                                                                   boolean result5 = (num3 == num4);
public class CompareOperatorExample {
  public static void main(String[] args) {
                                                                   System.out.println("result5: " + result5);
    int num1 = 10;
    int num2 = 10;
                                                                   float num5 = 0.1f;
    boolean result1 = (num1 == num2);
                                                                   double num6 = 0.1;
    boolean result2 = (num1 != num2);
                                                                   boolean result6 = (num5 == num6);
    boolean result3 = (num1 <= num2);</pre>
                                                                   boolean result7 = (num5 == (float)num6);
                                                                   System.out.println("result6: " + result6);
    System.out.println("result1: " + result1);
    System.out.println("result2: " + result2);
                                                                   System.out.println("result7: " + result7);
    System.out.println("result3: " + result3);
                                                                   String str1 = "자바";
                                                                   String str2 = "Java";
    char char1 = 'A';
                                                                                                                 result1: true
    char char2 = 'B';
                                                                   boolean result8 = (str1.equals(str2));
                                                                                                                 result2: false
    boolean result4 = (char1 < char2); //65 < 66
                                                                   boolean result9 = (! str1.equals(str2));
                                                                                                                 result3: true
    System.out.println("result4: " + result4);
                                                                   System.out.println("result8: " + result8);
                                                                                                                 result4: true
                                                                   System.out.println("result9: " + result9);
                                                                                                                 result5: true
                                                                                                                 result6: false
                                                                                                                 result7: true
                                                                                                                 result8: false
                                                                                                                 result9: true
```

## ☑ 논리 연산자

- 논리곱(&&), 논리합(||), 배타적 논리합(^) 그리고 논리 부정(!) 연산을 수행
- 흐름 제어문인 조건문(if), 반복문(for, while) 등에서 주로 이용

구분	연산식			결과	설명
	true	&&	true	true	피연산자 모두가 true일 경우에만
AND	true		false	false	
(논리곱)	false	또는 &	true	false	연산 결과가 true
	false		false	false	
	true		true	true	
OR	true	 또는 	false	true	피연산자 중 하나만 true이면 연산 결과는 true
(논리합)	false		true	true	
	false		false	false	
	true	^	true	false	
XOR	true		false	true	피연산자가 하나는 true이고
(배타적 논리합)	false		true	true	다른 하나가 false일 경우에만 연산 결과가 true
	false		false	false	
NOT (논리 부정)			true	false	피연산자의 논리값을 바꿈 
		!	false	true	- 백단인시의 <b>근디없</b> 글 미곱

# ch03.sec07.LogicalOperatorExample.java

```
package ch03.sec07;
public class LogicalOperatorExample {
 public static void main(String[] args) {
                                                             int value = 6;
   int charCode = 'A';
                                                             //int value = 7;
   //int charCode = 'a';
   //int charCode = '5';
                                                             if( (value%2==0) | (value%3==0) ) {
                                                               System.out.println("2 또는 3의 배수이군요.");
   if( (65<=charCode) & (charCode<=90) ) {
     System.out.println("대문자이군요.");
                                                             boolean result = (value%2==0) || (value%3==0);
                                                             if(!result) {
                                                               System.out.println("2 또는 3의 배수가 아니군요.");
   if( (97<=charCode) && (charCode<=122) ) {
     System.out.println("소문자이군요.");
   if( (48<=charCode) && (charCode<=57) ) {
     System.out.println("0~9 숫자이군요.");
```

```
대문자이군요.
2 또는 3의 배수이군요.
```

### 비트 논리 연산자

- o bit 단위로 논리 연산을 수행. 0과 1이 피연산자가 됨
- o byte, short, int, long만 피연산자가 될 수 있고, float, double은 피연산자가 될 수 없음

구분	연산식			결과	설명
	1	. &	1	1	두 비트 모두 1일 경우에만
AND	1		0	0	
(논리곱)	0	α	1	0	연산 결과가 1
	0		0	0	
	1		1	1	
OR	1		0	1	두 비트 중 하나만 1이면 연산 결과는 1
(논리합)	0		1	1	
	0		0	0	
	1	_	1	0	
XOR	1		0	1	두 비트 중 하나는 10고
(배타적 논리합)	0		1	1	다른 하나가 0일 경우 연산 결과는 1
	0		0	0	
NOT		~	1	0	보수
(논리 부정)			0	1	

# ch03.sec08.BitLogicExample.java

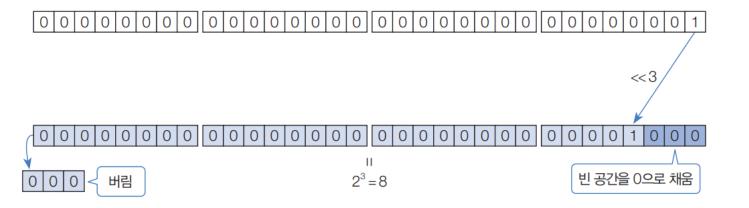
```
package ch03.sec08;
public class BitLogicExample {
 public static void main(String[] args) {
   System.out.println("45 & 25 = " + (45 \& 25));
   System.out.println("45 | 25 = " + (45 | 25));
   System.out.println("45 ^2 25 = " + (45 ^2 25));
   System.out.println("\sim45 = " + (\sim45));
   System.out.println("----");
   byte receiveData = -120;
   //방법1: 비트 논리곱 연산으로 Unsigned 정수 얻기
   int unsignedInt1 = receiveData & 255;
   System.out.println(unsignedInt1);
                                                        45 \& 25 = 9
                                                        45 | 25 = 61
   //방법2: 자바 API를 이용해서 Unsigned 정수 얻기
                                                        45 ^ 25 = 52
   int unsignedInt2 = Byte.toUnsignedInt(receiveData);
                                                         \sim 45 = -46
   System.out.println(unsignedInt2);
                                                         136
   int test = 136;
                                                         136
   byte btest = (byte) test;
                                                         -120
   System.out.println(btest);
```

# 비트 이동 연산자

### 🧿 비트 이동 연산자

○ 비트를 좌측 또는 우측으로 밀어서 이동시키는 연산을 수행

구분	연산식			설명
	а	~	b	정수 a의 각 비트를 b만큼 왼쪽으로 이동 오른쪽 빈자리는 0으로 채움 a x 2 <sup>b</sup> 와 동일한 결과가 됨
이동 (shift)	а	<b>&gt;&gt;</b>	b	정수 a의 각 비트를 b만큼 오른쪽으로 이동 왼쪽 빈자리는 최상위 부호 비트와 같은 값으로 채움 a / 2 <sup>b</sup> 와 동일한 결과가 됨
	а	<b>&gt;&gt;&gt;</b>	b	정수 a의 각 비트를 b만큼 오른쪽으로 이동 왼쪽 빈자리는 0으로 채움



# ch03.sec09.BitShiftExample1.java

```
package ch03.sec09;
public class BitShiftExample1 {
  public static void main(String[] args) {
    int num1 = 1;
    int result1 = num1 << 3;</pre>
    int result2 = num1 * (int) Math.pow(2, 3);
    System.out.println("result1: " + result1);
    System.out.println("result2: " + result2);
    int num2 = -8;
    int result3 = num2 >> 3;
    int result4 = num2 / (int) Math.pow(2, 3);
    System.out.println("result3: " + result3);
    System.out.println("result4: " + result4);
```

```
result1: 8
result2: 8
result3: -1
result4: -1
```

# ch03.sec09.BitShiftExample2.java

```
package ch03.sec09;
public class BitShiftExample2 {
 public static void main(String[] args) {
   int value = 772; //[00000000] [00000000] [000000011] [00000100]
   //우측으로 3byte(24bit) 이동하고 끝 1바이트만 읽음: [00000000]
   byte byte1 = (byte) (value >>> 24);
   int int1 = byte1 & 255;
   System.out.println("첫 번째 바이트 부호 없는 값: " + int1);
   //우측으로 2byte(16bit) 이동하고 끝 1바이트만 읽음: [00000000]
   byte byte2 = (byte) (value >>> 16);
   int int2 = Byte.toUnsignedInt(byte2);
   System.out.println("두 번째 바이트 부호 없는 값: " + int2);
   //우측으로 1byte(8bit) 이동하고 끝 1바이트만 읽음: [00000011]
   byte byte3 = (byte) (value >>> 8);
   int int3 = byte3 & 255;
   System.out.println("세 번째 바이트 부호 없는 값: " + int3);
   //끝 1바이트만 읽음: [00000100]
   byte byte4 = (byte) value;
   int int4 = Byte.toUnsignedInt(byte4);
   System.out.println("네 번째 바이트 부호 없는 값: " + int4);
```

```
첫 번째 바이트 부호 없는 값: 0
두 번째 바이트 부호 없는 값: 0
세 번째 바이트 부호 없는 값: 3
네 번째 바이트 부호 없는 값: 4
```

### 🗸 대입 연산자

- 우측 피연산자의 값을 좌측 피연산자인 변수에 대입.
- 우측 피연산자에는 리터럴 및 변수, 다른 연산식
   이 올 수 있음
- 단순히 값을 대입하는 단순 대입 연산자와 정해 진 연산을 수행한 후 결과를 대입하는 복합 대입 연산자가 있음

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

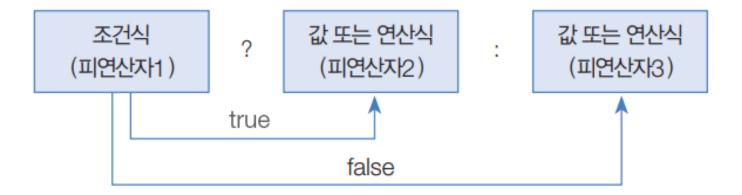
# ch03.sec10.AssignmentOperatorExample.java

```
package ch03.sec10;
public class AssignmentOperatorExample {
 public static void main(String[] args) {
    int result = 0;
    result += 10;
    System.out.println("result=" + result);
    result -= 5;
    System.out.println("result=" + result);
    result *= 3;
    System.out.println("result=" + result);
    result /= 5;
    System.out.println("result=" + result);
    result %= 3;
    System.out.println("result=" + result);
```

```
result=10
result=5
result=15
result=3
result=0
```

### 🗸 삼항 연산자

- 총 3개의 피연산자를 가짐
- ? 앞의 피연산자는 boolean 변수 또는 조건식.
  - → 이 값이 true이면 콜론(:) 앞의 피연산자가 선택되고, false이면 콜론 뒤의 피연산자가 선택됨



# ch03.sec11.ConditionalOperationExample.java

```
public class ConditionalOperationExample {
  public static void main(String[] args) {
    int score = 85;
    char grade = (score > 90) ? 'A' : ((score > 80) ? 'B' : 'C');
    System.out.println(score + "점은 " + grade + "등급입니다.");
  }
}
```

85점은 B등급입니다.

# 11 연산의 방향과 우선순위

### 💟 연산이 수행되는 순서

- 덧셈(+), 뺄셈(-) 연산자보다는 곱셈(\*), 나눗셈(/) 연산자가 우선. &&보다는 >, < 가 우선순위가 높음</li>
- 우선순위가 같은 연산자의 경우 대부분 왼쪽에서부터 오른쪽으로(→) 연산을 수행

연산자	연산 방향	우선순위
증감(++,), 부호(+, -), 비트(~), 논리(!)	<b>~</b>	LO
산술(*, /, %)	<b>→</b>	높음
산술(+, -)	<b>→</b>	<b>^</b>
쉬프트(〈〈, 〉〉, 〉〉〉)	<b></b>	
비교(<, >, <=, >=, instanceof)		
비교(==, !=)	<b></b>	
논리(&)	<b>→</b>	
논리(^)	<b></b>	
논리( )	<b></b>	
논리(&&)	<b></b>	
논리(  )	<b></b>	
조건(?:)	<b></b>	낮음
대입(=, +=, -=, *=, /=, %=, &=, ^=,  =, 〈〈=, 〉〉=, 〉〉〉=)	<u> </u>	