JAVA 입문 : 이론과 실습

제 3장 연산자

차 례

연산자

- 식의 의미를 결정
  - 식은 연산자(operator)와 피연자(operand)로 구성
  - 식의 값에 따라
    - 산술식, 관계식, 논리식으로 구분
- 표준 C 언어와 비슷
  - 언어시스템에서 제공하는 연산자들의 의미와 사용법 습득
  - 특히, 수학적인 의미와 구별

연산자의 종류

자바 언어의 연산자

- 산술 연산자 : + - * / % 단항+ 단항-
- 관계 연산자 : > >= < <= == !=
- 논리 연산자 : && || !
- 증감 연산자 : ++ --
- 비트 연산자 : & | ^ ~ << >> >>>
- 조건 연산자 : ? :
- 배정 연산자 : = += -= *= /= %= &= ^= |= >>= <<= >>>=
- 배열 연산자 : []
- 메소드 연산자 : () .
- 캐스트 연산자 : (자료형)
- instanceof 연산자 : instanceof

산술 연산자

- 수치 연산을 나타내는 연산자
  - 단항 산술 연산자 : +, -
  - 이항 산술 연산자 : +, -, *, /, %

```
x = -5;
x = -(-5);
x = -(3-5);
```

  - % : 나머지 연산자(remainder operator)
    - x % y = x - (x / y) * y

산술 연산자

- 실수형 연산
  - 부동 소수점 표현방법과 연산방법 : **IEEE 754 표준**
  - underflow, overflow
- 무한연산(infinite arithmetic)
  - java.lang.**Float**, java.lang.**Double 클래스**에서 기호상수 제공
    - **POSITIVE_INFINITY** (+∞)
    - **NEGATIVE_INFINITY**(-∞)
    - **NaN**(Not a Number)

산술 연산자 – 무한 연산

- 덧셈, 뺄셈 연산

x	y	x+y	x-y
+ ∞	+ ∞	+ ∞	NaN
+ ∞	- ∞	NaN	+ ∞
- ∞	+ ∞	NaN	- ∞
- ∞	- ∞	- ∞	NaN
NaN	+ ∞	NaN	NaN

- 곱셈, 나눗셈 연산

x	y	x/y	x%y
유한수	± 0.0	± ∞	NaN
유한수	± ∞	± 0.0	x
± 0.0	± 0.0	NaN	NaN
± ∞	유한수	± ∞	NaN
± ∞	± ∞	NaN	NaN

산술 연산자

- **ArithmeticOperators**.java

```
public class ArithmeticOperators {
    public static void main(String[] args) {
        int x=10, y=3;
        int add, sub, mul, div, rem;
        add = x + y;
        sub = x - y;
        mul = x * y;
        div = x / y;
        rem = x % y;
        System.out.println(x + " + " + y + " = " + add);
        System.out.println(x + " - " + y + " = " + sub);
        System.out.println(x + " * " + y + " = " + mul);
        System.out.println(x + " / " + y + " = " + div);
        System.out.println(x + " % " + y + " = " + rem);
    }
}
```

실행결과

```
x+y=13
x-y=7
x*y=30
x/y=3
x%y=1
```

산술 연산자 – 무한 연산

[예제 3.3 - InfinityArithmetic.java]

```
public class InfinityArithmetic {
    public static void main(String[] args) {
        double x=Double.POSITIVE_INFINITY, y=Double.NEGATIVE_INFINITY;
        double z=Double.MAX_VALUE;

        System.out.println(" POSITIVE_INFINITY  +  POSITIVE_INFINITY   =  " + (x+x));
        System.out.println(" POSITIVE_INFINITY  -  POSITIVE_INFINITY   =  " + (x-x));
        System.out.println(" POSITIVE_INFINITY  + NEGATIVE_INFINITY   = " + (x+y));
        System.out.println(" POSITIVE_INFINITY * 0.0                   = " + (x*0.0));
        System.out.println(" Double.MAX_VALUE  /  POSITIVE_INFINITY  = " + (z / x));
        System.out.println(" 0.0 / 0.0                                 = " + (0.0/0.0));
    }
}
```

실행 결과 :

```
POSITIVE_INFINITY + POSITIVE_INFINITY     = Infinity     (Infinity)
POSITIVE_INFINITY - POSITIVE_INFINITY     = NaN          (NaN)
POSITIVE_INFINITY + NEGATIVE_INFINITY     =NaN           (-Infinity)
POSITIVE_INFINITY * 0.0                   = NaN
Double.MAX_VALUE  /  POSITIVE_INFINITY    = 0.0
0.0 / 0.0                                 = NaN
```

관계 연산자

- 두 개의 값을 비교하는 연산자
  - 연산결과 : true or false
  - 관계 연산자가 포함된 식 : 관계식
    - for, while, do-while의 조건식
- 연산자
  - <, <=, >, >=, ==, !=

b == x < y ===> b == (x < y)

↑ 우선순위 ↓

  - 산술 연산자보다 낮음

a > b + c ===> a > (b + c)

관계 연산자

- **RelationalOperators**.java

```
public class RelationalOperators {
    public static void main(String[] args) {
        int x=3, y=5, z=7;
        boolean f, t;

        f = x > y;
        t = x < y  &&  y < z;
        System.out.println("f = " + f + ", t = " + t);

        f = x <= y;
        t = x >= y == y >= x;
        System.out.println("f = " + f + ", t = " + t);
    }
}
```

실행결과

f=false, t=true

f=true, t=false

논리 연산자

- 두 피연산자의 논리 관계를 나타내는 연산자
- 연산자의 종류
  - ! , && , ||

(높음) ⟷ (낮음)

```
a < b  &&  b < c
└──┘       └──┘
 1          2
└──────────┘
     3
```

[예제 3.6] 테스트

논리 연산자

- **LogicalOperators.java**

```
public class LogicalOperators {
    public static void main(String[] args) {
        int x=3, y=5, z=7;
        boolean b;

        b = x < y && y < z;
        System.out.println("Result = " + b);

        b = x == y || x < y && y > z;
        System.out.println("Result = " + b);
    }
}
```

실행결과

Result=true

Result=false

증가 및 감소 연산자

- 연산자 기호
  - ++, --
  - 변수가 아닌 식에는 사용 못함
  - 실수형 적용 안됨 : (a+b)++ // error
- 전위 연산자(prefix operator)

```
n = 1;
x = ++n;          // x=2, n=2
```

- 후위 연산자(postfix operator)

```
n = 1;
x = n++;          // x=1, n=2
```

```
x = x++ - --x;    // x = ?
```

증가 및 감소 연산자

- **IncDecOperators**.java

```
public class IncDecOperators {
    public static void main(String[] args) {
        int x=3, y=5;
        int a, b;

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

        a = (++x) + 1;
        System.out.println("x = " + x + " , a = " + a);

        b = (y++) + 1;
        System.out.println("y = " + y + " , b = " + b);
    }
}
```

실행결과

x=4, y=4

x=5, a=6

y=5, b=5

비트 연산자 [1/3]

- 비트 연산자
  - 비트 단위로 연산 --- 기억장소 절약
  - 피연산자는 반드시 정수형
  - 종류(7가지) --- &, |, ^, ~, <<, >>, >>>
  - 우선순위

연산자	우선순위
~	(높음) ↕ (낮음)
<< >> >>>	
&	
^	
|	

비트 연산자 [2/3]

- 비트 논리곱(Bitwise AND)
  - 1001_2 & 0011_2 = 0001_2
  - 변수의 일정 부분을 매스킹(masking)하여 특정 부분만을 추출하기 위해 사용
- 비트 논리합(Bitwise OR)
  - 1001_2 | 0011_2 = 1011_2
- 배타적 논리합(Exclusive OR)
  - 1001_2 ^ 0011_2 = 1010_2
- 1의 보수(One's Complement)
  - ~ 00001010_2 = 11110101_2

비트 연산자 [3/3]

- 비트 이동 연산자(shift operator)
  - 왼쪽 이동(<<)

$$x << y = x * 2^y$$

  - 오른쪽 이동(>>)

$$x >> y = x / 2^y$$

  - 부호없는 오른쪽 이동(>>>)
    - 부호없는 정수(unsigned integer)를 지원하지 않기 때문에 제공
    - (-1) >>> 1 ?

비트 연산자 [3/3]

- **BitOperators**.java

```
public class BitOperators {
    public static void main(String[] args) {
        int x=9, y=3;

        System.out.println(x + " & " + y + " = " + (x&y));
        System.out.println(x + " | " + y + " = " + (x|y));
        System.out.println(x + " ^ " + y + " = " + (x^y));
        System.out.println("~10 = " + (~10));
    }
}
```

실행결과

9&3=1

9|3=11

9^3=10

~10=-11

비트 연산자 [3/3]

- **ShiftOperators**.java

```
public class ShiftOperators {
    public static void main(String[] args) {

        System.out.println(" 10 <<  2 = " + (10<<2));
        System.out.println(" 10 >>  2 = " + (10>>2));
        System.out.println(" 10 >>> 2 = " + (10>>>2));
        System.out.println("-10 >>> 2 = " + (-10>>>2));
    }
}
```

실행결과

10<< 2=40

10>> 2=2

10>>>2=2

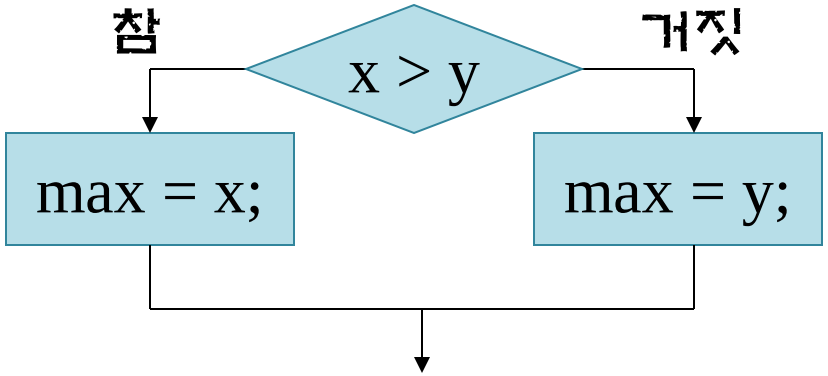
-10>>>2=1073741821

조건 연산자

- 조건 연산자
  - 형태 : **식1 ? 식2 : 식3** (3항 연산자)

```
max = x > y ? x : y ;
```

```
if (x > y)  max = x;
else        max = y;
```



 [예제 3.10] 테스트 --- 교과서 80쪽

조건 연산자

- **ConditionalOperators**.java

```
public class ConditionalOperator {
    public static void main(String[] args) throws java.io.IOException {
        int a, b, c;
        int m=0;

        System.out.print("Enter three numbers : ");
        a = System.in.read() - '0';
        b = System.in.read() - '0';
        c = System.in.read() - '0';
        m = (a > b) ? a : b;
        m = (m > c) ? m : c;
        System.out.println("The largest number = " + m);
    }
}
```

입력자료: Enter three numbers : 526

실행결과: The largest number = 6

배정 연산자

- 배정 연산자의 형태

식1 = 식1 op 식2 ⟷ 식1 op= 식2

- 결합 연산자
  - 산술 연산자 : + - * / %
  - 비트 연산자 : & | ^ << >> >>>
- 의미 :

sum += i ; ⟷ sum = sum + i ;

x *= y + 1 ; ✗ x = x * y + 1 ;

x = x * (y+1)

배정 연산자

- **AssignmentOperator**.java

```
public class AssignmentOperators {
    public static void main(String[] args) {
        int x, y=2;

        x = 10;   x += y;     System.out.print("1. x = " + x + ", ");
        x = 10;   x -= y;     System.out.print("2. x = " + x + ", ");
        x = 10;   x *= y;     System.out.print("3. x = " + x + ", ");
        x = 10;   x /= y;     System.out.print("4. x = " + x + ", ");
        x = 10;   x %= y;     System.out.println("5. x = " + x);

        x = 10;   x &= y;     System.out.print("6. x = " + x + ", ");
        x = 10;   x |= y;     System.out.print("7. x = " + x + ", ");
        x = 10;   x ^= y;     System.out.println("8. x = " + x);
        x = 10;   x <<= y;    System.out.print("9. x = " + x + ", ");
        x = 10;   x >>= y;    System.out.print("10. x = " + x + ", ");
        x = 10;   x >>>= y;   System.out.println("11. x = " + x);
    }
}
```

실행결과 : 1. x=12, 2. x=8, 3. x=20, 4. x=5, 5. x=0
6. x=2, 7. x=10, 8. x=8
9. x=40, 10. x=2, 11. x=2

캐스트 연산자

- 캐스트 연산자 --- 자료형 변환 연산자
  - 형태 :

```
(자료형) 식
```

  - 캐스트 연산자 사용 예 :

```
(int) 3.75        ===>   3
(float) 3         ===>   3.0
(float) (1 / 2)   ===>   0.0
(float) 1 / 2     ===>   0.5
```

    - 정수사이의 연산 결과는 정수

캐스트 연산자

- **CastOperator**.java

```
public class CastOperator {
    public static void main(String[] args) {
        int i = 0Xffff;
        short s;

        s = (short) i;
        System.out.println("i = " + i);
        System.out.println("s = " + s);
    }
}
```

실행결과

i=65535

s=-1

연산자 우선순위 [1/2]

연산자	결합법칙	우선순위
() [] .	좌측결합	(높음)
! ~ ++ -- 단항+ 단항- (자료형)	우측결합	
* / %	좌측결합	
+ -	좌측결합	
<< >> >>>	좌측결합	
< <= > >= instanceof	좌측결합	
== !=	좌측결합	
&	좌측결합	
^	좌측결합	
|	좌측결합	
&&	좌측결합	
||	좌측결합	
? :	우측결합	(낮음)
= += -= *= /= %= &= ^= |= <<= >>= >>>=	우측결합	

연산자 우선순위 [2/2]

- y = x + y - z ; // 좌측 결합
- y = -x ; // 우측 결합
- y = -x++ ; // x의 값에 단항 – 연산을 적용한 후 y에 배정하고 x를 증가
- y = -++x ; // x를 증가한 후 x의 값에 단항 - 연산을 적용한 후 y에 배정
- y = -x + z ; // x의 값에 단항 – 연산한 후 z를 더하고 그 결과를 y에 배정

형 변환

- 자료형의 크기 방향
  - 광역화 형 변환
    - 작은 자료형의 값을 큰 자료형의 값으로 변환
  - 협소화 형 변환
    - 큰 자료형의 값을 작은 자료형의 값으로 변환
- 형 변환의 주체
  - 묵시적 형 변환
    - 컴파일러에 의해 자동수행
  - 명시적 형 변환
    - 캐스팅을 이용하여 프로그래머가 형 변환을 명시

자료형의 크기 방향 [1/2]

- 광역화 형 변환
  - 컴파일러에 의해 자동으로 수행되는 묵시적 변환
  - 예 :

byte	→	short, int, long, float, double
short	→	int, long, float, double
char	→	int, long, float, double
int	→	long, float, double
long	→	float, double
float	→	double

  - 정밀도 상실 : int → float
    long → float
    long → double

자료형의 크기 방향 [1/2]

- **WideningTypeConversion** .java

```
public class WideningTypeConversion {
   public static void main(String[] args) {
        short s=1; int i; long l;
        float f; double d;

        i = s; l = i;
        System.out.println("s = " + s + " i = " + i + ”l = " +l);
        f = l; d = f;
        System.out.println("f = " + f + " d = " + d);
   }
}
```

실행결과: s=1, i=1, l=1
f=1.0, d=1.0

자료형의 크기 방향 [2/2]

- 협소화 형 변환
  - 프로그래머가 반드시 캐스트 연산자를 사용하여 변환될 자료형을 표시하여 변환
  - 예 :

byte	→	char
short	→	byte, char
char	→	byte, short
int	→	byte, short, char
long	→	byte, short, char, int
float	→	byte, short, char, int, long
double	→	byte, short, char, int, long, float

형 변환의 주체 [1/5]

- 묵시적 형 변환
  - 컴파일러에 의해 자동적으로 수행

```
char c = 'A';
short s=1; int i=2; long l=3;
float f=2.1f; double d=3.2;

①      i    =    (   c    +    s   );  // i = 66
      (int)        (int)     (short)
                              (int)
                   (short)
```

[예제 3.16] 테스트

형 변환의 주체 [2/5]

② l = s + i ; // l = 3

(long) (short) (int)

(int)

(long)

③ d = f + d; // d = 5.3

(double) (float) (double)

(double)

(double)

형 변환의 주체 [2/5]

- **LowerToUpperConversion** .java

```
public class LowerToUpperConversion {
    public static void main(String[] args) throws java.io.IOException {
        char  c;
        int x;

        System.out.print("Enter the lower char = ");
        c = System.in.read();
        if (c >= 'a' && c <= 'z')
            x = c + ('A' - 'a');          /* 소문자를 대문자로 변환 */
        else x = c;
        System.out.println("Upper char of " + c + "= " + (char)x);
    }
}
```

입력자료: Enter a lower char=q

실행결과: Upper char of q=Q

형 변환의 주체 [3/5]

- 명시적 형 변환
  - 프로그래머가 캐스트 연산자를 사용하여 변환

```
char c = 'A';
short s=1; int i=2; long l=3;
float f=2.1f; double d=3.2;

①     s   =   (short) (  c    +    i  );  // s = 67
   (short)                (char)      (int)
                              (int)
                  (short)
```

형 변환의 주체 [4/5]

② **f = (float) (f + d);** // f = 5.3

(float) (float) (double)

(double)

(float)

형 변환의 주체 [4/5]

- **ExplicitTypeConversion** .java

```
public class ExplicitTypeConversion {
   public static void main(String[] args) {
         int x;
         float y, z;
         char c='A';

         x = 7 / 2;
         y = (float) 7 / 2;
         z = 7 / 2;
         System.out.println("x = " + x + " y = " + y + " z = " + z);
         c++;
         System.out.println("c = " + c);
   }
}
```

실행결과

x=3 y=3.5 z=3.0

c=B

형 변환의 주체 [5/5]

[예제 3.18 - LosePrecision.java]

```
public class LosePrecision {
    public static void main(String[] args) {
        int big = 1234567890;
        float approx;

        approx = (float) big;
        System.out.println("difference = " + (big - (int)approx));
    }
}
```

실행 결과 :

```
difference = -46
```

형 변환 금지

- 같은 자료형 이외에 다른 자료형으로의 변환이 금지된 자료형
- boolean 형

[예제 3.19] 테스트 --- 교과서 93쪽

형 변환 금지

- **Forbidden**.java

```
public class Forbidden {
    public static void main(String[] args) {
        boolean f = false, b;
        int i;

        b = f;
        System.out.println("f = " + f + " b = " + b );
        // i = f;              // 에러 : 묵시적, 광역화 형변환 금지
        // i = (short)f;    // 에러 : 명시적 형변환 금지
    }
}
```

실행결과

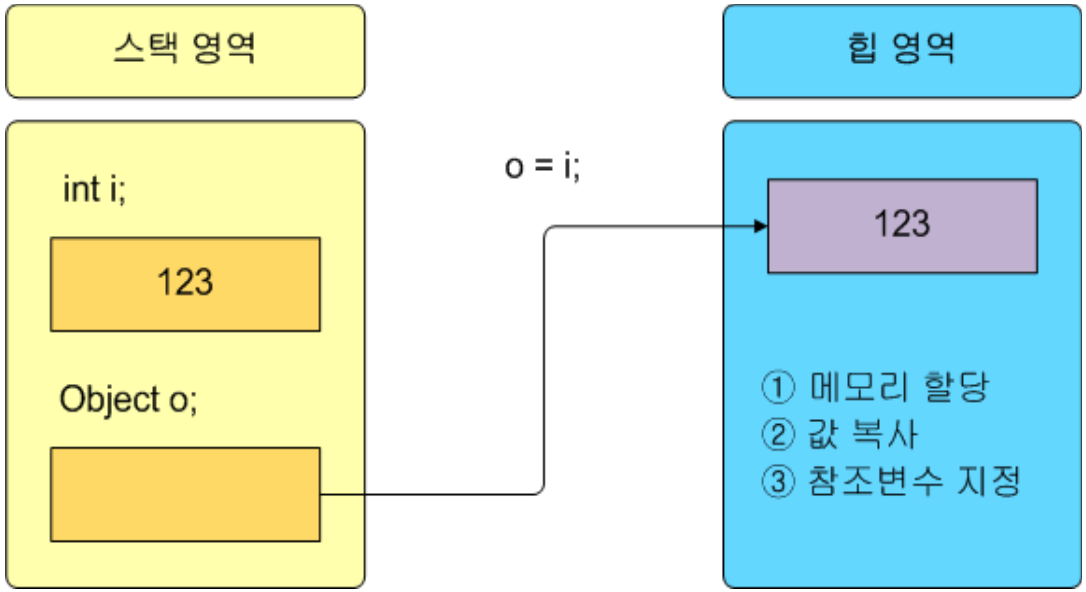
f=false b= false

박싱과 언박싱 (Boxing/Unboxing)

- 박싱
  - 기본형의 데이터를 참조형으로 변환
- 언박싱
  - 참조형의 데이터를 기본형으로 변환
- 자동 박싱/자동 언박싱 (auto boxing/auto unboxing)
  - 자바 컴파일러가 프로그램 문맥에 따라 자동으로 박싱/언박싱

박싱 [1/2]

- 스택에 저장된 기본형 데이터를 힙 영역의 참조형으로 변환



박싱 [2/2]

- 다음과 같은 기본형과 참조형 간에 지원

boolean	→	Boolean
byte	→	Byte
char	→	Char
short	→	Short
int	→	Integer
long	→	Long
float	→	Float
double	→	Double

언박싱 [1/2]

- 다음과 같은 참조형과 기본형 간에 지원

Boolean	→	boolean
Byte	→	byte
Char	→	char
Short	→	short
Integer	→	int
Long	→	long
Float	→	float
Double	→	double

- 잘못된 형식의 형 변환 시 Exception 발생

언박싱 [2/2]

[예제 3.20 - BoxingUnBoxing.java]

```
public class BoxingUnboxing {
    public static void main(String[] args) {
        int foo = 526;
        Object bar = foo; // foo is boxed to bar.
        System.out.println(bar);
        try {
            double d = (Double)bar;
            System.out.println(d);
        } catch (ClassCastException e) {
            System.err.println(e.toString());
        }
    }
}
```

실행 결과 :

```
526
java.lang.ClassCastException: java.lang.Integer cannot be cast to java.lang.Double
```

단원 요약

- 자바의 연산자
  - 연산자는 식의 의미를 결정하며, 식의 값에 따라 산술식, 관계식, 논리식으로 구분
  - 표준 C(ANSI C) 언어와 유사

  ⚠ >>> 연산자 : unsigned형을 지원하지 않기 때문에 필요

- 형변환
  - 광역화 형 변환
    - 작은 자료형의 값을 큰 자료형의 값으로 변환
  - 협소화 형 변환
    - 큰 자료형의 값을 작은 자료형의 값으로 변환, 반드시 명시적 형 변환(캐스트 연산자사용)을 해야함

- 박싱과 언박싱
  - 박싱 : 기본형의 데이터를 참조형으로 변환
  - 언박싱 : 참조형의 데이터를 기본형으로 변환