

```
for object to mirror_mod.mirror_object
operation == "MIRROR_X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
operation == "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True
```

```
@selection at the end -add
mirror_ob.select= 1
modifier_ob.select=1
context.scene.objects.active
("Selected" + str(modifier_ob.name))
mirror_ob.select = 0
= bpy.context.selected_objects[0]
data.objects[one.name].select
```

```
print("please select exactly one object")
-- OPERATOR CLASSES --
```

```
types.Operator):
    X mirror to the selected
    object.mirror_mirror_x"
    mirror X"
```

# Java 기초

연산자

# 연산자(Operator)

---

- 연산자란?
  - 변수와 값의 연산을 위해 쓰이는 기호들을 이야기 한다.
  - Java에서는 연산을 수행하기 위한 여러가지 기호 타입을 제공하고 있다.
  - 각각의 연산자들은 연산에 대한 우선순위를 가진다.
  - 각 연산자들은 연산방향이 다르다.

# 산술연산자

---

- 산술연산자

+ (더하기), - (빼기), \* (곱하기), / (나누기), % (나머지)

- 산술연산자는 더하기, 빼기, 곱하기, 나누기, 나머지 다섯개의 연산을 의미한다.
- 곱셈, 나눗셈, 나머지 연산이 덧셈, 뺄셈 연산보다 연산 우선순위가 높다.

# 산술연산자

산술연산자 예제

```
Run | Debug
public static void main(String[] args) {
    // 덧셈 연산
    int a1 = 3;
    int a2 = 4;
    System.out.println(a1+a2); // 7

    // 뺄셈 연산
    int b1 = 8;
    int b2 = 2;
    System.out.println(b1-b2); // 6

    // 곱셈 연산
    int c1 = 2;
    int c2 = 4;
    System.out.println(c1*c2); // 8

    // 나눗셈 연산
    int d1 = 10;
    int d2 = 2;
    System.out.println(d1/d2); // 5

    // 나머지 연산
    int e1 = 9;
    int e2 = 4;
    System.out.println(e1%e2); // 1
}
```

# 산술연산자

---

- 산술연산 시 타입 강제 변환
  - 산술 연산 시 int형보다 크기가 작은 자료형은 int형으로 형변환 후에 연산을 한다
  - int형 보다 클 경우 두 개의 피연산자 중 자료형의 범위가 큰 쪽에 맞춰서 형변환 된 후 연산을 수행한다.

`byte + byte → int + int → int`  
`byte + short → int + int → int`  
`char + char → int + int → int`

`float + int → float + float → float`  
`long + float → float + float → float`  
`float + double → double + double → double`

# 산술연산자

## 산술연산 시 타입 강제 변환 예제

```
// byte + byte 연산
byte a1 = 2;
byte a2 = 3;
//byte a3 = a1+a2; //에러
byte a3 = (byte)(a1+a2);
System.out.println(a3); // 5

// byte + short 연산
byte b1 = 2;
short b2 = 3;
//short b3 = b1+b2; //에러
short b3 = (short)(b1+b2);
System.out.println(b3); // 5

// char + char 연산
char c1 = 'A';
char c2 = 'A';
// char c3 = c1 + c2 // 에러
char c3 = (char)(c1+c2);
System.out.println(c3);
```

```
// float + int 연산
float d1 = 1.02f;
int d2 = 3;
//int d3 = d1 + d2; // 에러
float d3 = d1 + d2;
System.out.println(d3); // 4.02

// float + long 연산
float e1 = 1.34f;
long e2 = 4;
//long e3 = e1 + e2; // 에러
float e3 = e1 + e2;
System.out.println(e3); // 5.34

// float + double 연산
float f1 = 1.23f;
double f2 = 2.34;
//double f3 = f1 + f2; // 에러
double f3 = f1 + f2;
System.out.println(f3); // 3.570000019073486
```

## 산술연산자

```
Run | Debug
public static void main(String[] args) {
    // 연산을 통해 타입을 벗어난 값의 예제
    int a = 1000000;
    int b = 2000000;
    System.out.println(a*b); // -1454759936
}
```

- 초과범위 연산
  - 해당 연산자의 범위를 초과한 숫자를 넣을 경우 예상한 결과에 다르게 나올 수 있다.

# 산술연산자

---

- 증감연산자

증감 연산자(++): 피연산자(operand)의 값을 1 증가시킨다.

감소 연산자(--): 피연산자(operand)의 값을 1 감소시킨다.

- 변수의 값을 1 증가시키거나 감소시킬 때 쓰임
- 연산자의 위치에 따라 결과가 달라질 수 있다.

```
// 증감연산자 예제
// 증감연산자는 변수의 전에 오느냐 후에 오느냐에 따라
// 연산의 결과가 달라지며 전과 후에 올 경우 상황은 다음과 같다.
// 전 : 로직이 실행되기 전 연산이 먼저 실행
// 후 : 로직이 실행되고 연산이 그 다음에 실행
int a = 3;
System.out.println(++a); // 4
System.out.println(a++); // 4
System.out.println(a);   // 5
System.out.println(--a); // 4
System.out.println(a--); // 4
System.out.println(a);   // 3
```



# 비교연산자

---

- 비교연산자

>(크다), <(작다), >=(크거나같다), <=(작거나같다) ==(같다), !=(같지 않다)

- 두 개의 리터럴을 비교하는데 사용되는 연산자.
- 주로 반복문이나 조건문의 조건식에 사용된다.
- 연산 결과는 무조건 논리형(true/false)이다.
- 비교하는 피연산자의 자료형이 서로 다를 경우에는 자료형이 큰 쪽으로 형변환하여 피연산자의 타입을 일치시킨 후에 비교한다.
- 등가 비교 연산자(==, !=)는 어떤 자료형이든 사용이 가능하지만 대소 비교 연산자(>, <, >=, <=)는 논리형과 참조형에서 사용이 불가능하다.

# 비교연산자

---

비교연산자 예제

```
// 비교연산자 예제
int a1 = 3;
int a2 = 5;
System.out.println(a1<a2);

int b1 = 11;
int b2 = 7;
System.out.println(b1>b2);

int c1 = 6;
int c2 = 8;
System.out.println(c1<=c2);

int d1 = 7;
int d2 = 3;
System.out.println(d1>=d2);

int e1 = 5;
int e2 = 5;
System.out.println(e1 == e2);

int f1 = 6;
int f2 = 3;
System.out.println(f1 != f2);
```

# 비교연산자

비교연산자 예제

Run | Debug

```
public static void main(String[] args) {  
    // 비교연산자 특징  
    String s1 = "hi";  
    String s2 = "hi";  
  
    System.out.println(s1 == s2);  
    // System.out.println(s1 > s2); // 에러  
  
    Boolean b1 = true;  
    Boolean b2 = true;  
    System.out.println(b1 != b2);  
    // System.out.println(b1 > b2); // 에러  
}
```

# 논리연산자

- 논리연산자
  - 논리연산자는 논리형 값 과의 연산 결과를 출력하는 연산자이다.
  - 연산의 결과값은 논리형으로 나온다
  - 논리 연산자의 종류는 아래와 같다.
  - &&(and) : 두 값이 모두 true면 true, 아니면 false.
  - ||(or) : 두 값 중 하나라도 true면 true, 아니면 false.

	true true	true false	false false
&&(and)	true	false	false
(or)	true	true	false

# 논리연산자

- 논리 부정 연산자(!)
  - 논리연산자에는 논리값 자체를 뒤집는 연산자가 있는데 바로 부정연산자이다.
  - (!)를 앞에 붙여 사용하며 true를 false로 false를 true로 바꾼다.

	TRUE	FALSE
!(not)	false	true

# 논리연산자

논리연산자 예제

Run | Debug

```
public static void main(String[] args) {  
    boolean a1 = true;  
    boolean a2 = true;  
    boolean b1 = true;  
    boolean b2 = false;  
    boolean c1 = false;  
    boolean c2 = false;  
  
    System.out.println(a1 && a2);    // true  
    System.out.println(b1 && b2);    // false  
    System.out.println(c1 && c2);    // false  
  
    System.out.println(a1 || a2);    // true  
    System.out.println(b1 || b2);    // true  
    System.out.println(c1 || c2);    // false  
}
```

# 논리연산자

- 논리연산자의 맹점
  - && 연산에서 좌변에 false일 경우 우변의 연산을 하지 않는다.
  - || 연산에서 좌변에 true일 경우 우변의 연산을 하지 않는다.

Run | Debug

```
public static void main(String[] args) {  
    int a1 = 3;  
    int a2 = 4;  
    System.out.println((3 > 5)&&((++a1) > 1));  
    System.out.println(a1);    // 3  
  
    System.out.println((5 > 3)||((++a2) > 2));  
    System.out.println(a2);    // 4  
}
```

## 비트연산자

연산자	설명
&(논리곱, and)	양쪽 모두 1 이면 1, 아니면 0 을 반환
(논리합, or)	어느 한쪽이 1 이면 1, 그렇지 않으면 0 을 반환
^(배타적 논리합, xor)	한쪽이 1 이고 다른 한쪽이 0 이면 1을, 아니면 0 을 반환
~(부정, not)	각 비트를 반전시킨 값을 반환
<< (왼쪽 시프트)	이진코드를 왼쪽으로 시프트 한다
>> (오른쪽 시프트)	이진코드를 오른쪽으로 시프트 한다
>>> (논리 오른쪽 시프트)	오른쪽으로 시프트한다. 오른쪽으로 밀면서 비게되는 앞쪽 비트를 무조건 0 으로 채워넣는다

- 비트연산자
  - 비트 연산자는 데이터를 비트화 하여 연산할 때 쓴다.
  - 빈도가 많지 않으나 고속 연산을 할 경우 사용할 수 있다.



# 비트 연산자

## 비트 연산자 예제

Run | Debug

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

```
    byte a = 10; // 00001010
```

```
    byte b = 9;  // 00001001
```

```
    byte c = 1; //시프트할 칸수
```

```
    System.out.println(a + " & " + b + " = " + (a&b)); //논리합
```

```
    System.out.println(a + " | " + b + " = " + (a|b)); //논리곱
```

```
    System.out.println(a + " ^ " + b + " = " + (a^b)); //배타적 논리합(xor)
```

```
    System.out.println("~10 = " + (~a)); //a 의 보수(반전)
```

```
    System.out.println(a + " << " + c + " = " + (a<<c)); //왼쪽 1비트 시프트(뒤를 0 으로 채움)
```

```
    System.out.println(a + " >> " + c + " = " + (a>>c)); //오른쪽 1비트 시프트(앞을 밀리기전 첫째자리와 동일한 비트로 채움)
```

```
    System.out.println(a + " >>> " + c + " = " + (a>>>c)); //오른쪽 1비트 논리 시프트(앞을 0 으로 채움)
```

```
    System.out.println(-1 * a + " >> " + c + " = " + (-1*a >> c));
```

```
    System.out.println(-1 * a + " >>> " + c + " = " + (-1*a >>> c));
```

```
}
```

10 & 9 = 8

10 | 9 = 11

10 ^ 9 = 3

~10 = -11

10 << 1 = 20

10 >> 1 = 5

10 >>> 1 = 5

-10 >> 1 = -5

-10 >>> 1 = 2147483643

# 삼항연산자

---

- 삼항연산자

(조건식) ? 참일때의 값 : 거짓일때의 값

- 유일하게 논리값을 통해 선택적으로 값을 가져올 수 있는 연산자.
- 삼항연산자는 조건식, 참일때의 값, 거짓일때의 값 세가지가 필요하다.
- 조건식이 참일때는 참일때의 값이, 거짓일때는 거짓일때의 값이 필요하다.
- 주의할 점은 참과 거짓의 타입이 동일하거나 다르더라도 자동 형변환이 가능해야 한다.
- 삼항 연산자 안에 삼항 연산자를 넣어 사용할 수 있으며 이렇게 해서 다수의 조건 선택을 만족시키는 로직을 만들 수 있다.

# 삼항연산자

삼항연산자 예제

Run | Debug

```
public static void main(String[] args) {  
    // 삼항연산자 예제  
    int a1 = 4;  
    // int a2 = (a1 > 2) ? 7 : 2.4; // 형 변환 에러  
    int a2 = (a1 > 2) ? 7 : 2;  
    System.out.println(a2); // 7  
  
    // 삼항연산자를 두개 사용해서 다중 조건이 가능해진다.  
    int b1 = 5;  
    int b2 = (b1 > 6) ? 7 : (b1 > 4) ? 5 : 3;  
    System.out.println(b2); // 5  
}
```

# 대입연산자

---

- 대입연산자

```
int i = 0; i = i + 3;
```

- 대입 연산자는 우측의 변수에 값을 대입할 때 사용하며 보통 (=)를 사용하는 것을 의미한다.
- 대입연산자는 왼쪽에서 오른쪽으로 연산하는 연산자이다.
- 대입 연산자는 모든 연산자 중에서 가장 나중에 연산되는 연산자이다.

# 대입연산자

## 대입연산자 예제

Run | Debug

```
public static void main(String[] args) {  
    // 대입 연산자 예제  
    int i = 3;  
    // 보통 자주 쓰이는 패턴 중 하나이며  
    // 변수 i 의 이전값에 1을 더해서 i에 다시 대입하겠다는 뜻이다.  
    // 이렇게 되면 기존의 i 값은 소멸되며  
    // 기존 값에 1이 더해진 i값이 i에 들어있게 된다.  
    i = i + 1;  
}
```

# 대입연산자

---

- 대입 산술연산자

```
i += 1; i -= 1; i *= 2; i /= 3; i %= 3;
```

- 대입 연산자를 쓰다보면 간혹 연산중에 '  $i = i + 1$  ' 같은 연산을 쓰게 된다.
- 이 의미는 해당 변수에 들어있는 값에 +1을 하여 다시 대입한다란 뜻이 된다.
- 하지만 불필요한 연산자를 없애기 위해 Java에서는 이를 간략하게 하여 대입산술연산자 라는 것이 존재한다.
- 예를들면 '  $i = i + 1$  ' 같은 연산을 '  $i += 1$  ' 로 단순화 시킬 수 있다.

# 대입연산자

대입산술연산자 예제

```
Run | Debug
public static void main(String[] args) {
    // 대입 산술 연산자 예제
    int a = 1;
    a += 1; // a = a + 1;
    System.out.println(a); // 2

    int b = 2;
    b -= 1;
    System.out.println(b); // 1

    int c = 3;
    c *= 3;
    System.out.println(c); // 9

    int d = 8;
    d /= 2;
    System.out.println(d); // 4

    int e = 16;
    e %= 7;
    System.out.println(e); // 2
}
```

# 연산자의 종류 및 우선순위

우선순위	연산자	내용
1	() , []	괄호 / 대괄호
2	!, ~, ++, --	부정 / 증감 연산자
3	*, /, %	곱셈 / 나눗셈 연산자
4	+, -	덧셈 / 뺄셈 연산자
5	<<, >>, >>>	비트단위의 쉬프트 연산자
6	<, <=, >, >=	관계 연산자
7	==, !=	
8	&	비트단위의 논리연산자
9	^	
10		
11	&&	논리곱 연산자
12		논리합 연산자
13	?:	조건 연산자
14	=, +=, -=, *=, /=, %= &=, ^=, ~=	대입 / 할당 연산자



# 연산자의 종류 및 우선순위

---

- 연산자의 종류 및 우선순위
  - 연산자는 각각의 종류마다 우선순위를 가지고 있다.
  - 하지만 연산자 우선순위를 외우기보단 ()로 먼저 연산할 부분을 감싸서 연산의 우선순위를 지정하는 것이 더 효과적이다.

```
Run | Debug
public static void main(String[] args) {
    int a = 6 + 3 * 8 / 2;
    System.out.println(a); // 18
    int b = (6 + 3) * (8 / 2);
    System.out.println(b); // 36
}
```