## 변수와 연산자

변수의 저장

```
int score;      //변수 선언
score = 90;     //값저장
```

초기값은 변수를 선언함과 동시에 줄 수도 있다.

```
int score = 90;
```

변수는 초기화 되어야 읽기 가능함.

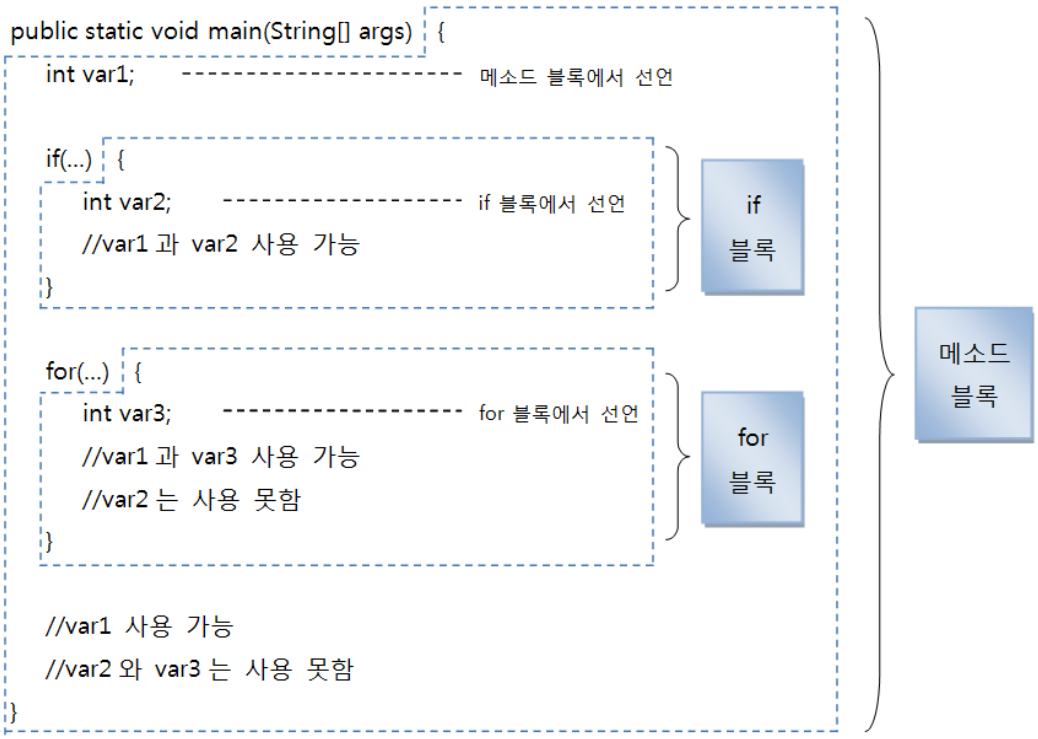
잘못된 코딩의 예

```
int value;                  //변수 value 선언 (초기화 안됨)
int result = value + 10;    //변수 value 값을 읽고 10 을 더한 결과값을 변수 result 에 저장
```

맞게 고친 후의 코드

```
int value = 30;             //변수 value 가 30 으로 초기화 됨
int result = value + 10;    //변수 value 값을 읽고 10 을 더한 결과값(40)을 변수 result 에 저장
```

### 변수의 사용 범위



## 데이터 타입

메모리의 최소 기억 단위인 bit가 모여 byte 형성

| 값의 종류 | 기본 타입 | 메모리 사용 크기 | | 저장되는 값의 범위 |
|---|---|---|---|---|
| 정수 | byte | 1 byte | 8 bit | $2^7$~$2^7$-1 (-128~127) |
| | char | 2 byte | 16 bit | 0~$2^{16}$-1 (유니코드: ₩u0000~₩uFFFF, 0~65535) |
| | short | 2 byte | 16 bit | -$2^{15}$~$2^{15}$-1 (-32,768~32,767) |
| | int | 4 byte | 32 bit | -$2^{31}$~$2^{31}$-1 (-2,147,483,648~2,147,483,647) |
| | long | 8 byte | 64 bit | -$2^{63}$~$2^{63}$-1 |
| 실수 | float | 4 byte | 32 bit | (+/-)1.4E-45 ~ (+/-)3.4E38 |
| | double | 8 byte | 64 bit | (+/-)4.9E-324 ~ (+/-)1.7E308 |
| 논리 | boolean | 1 byte | 8 bit | true, false |

byte(1) < short(2) < int(4) < long(8) < float(4) < double(8)

## 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 |

아래 코드 실습하세요

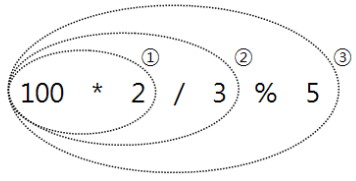
```
class AsarOperators
{
	public static void main(String[] args)
	{
		int num1;
		int num2;

		num1 = 714;
		num2 = 500;

		System.out.println(num1 + " + " + num2 + " = " + (num1 + num2));
		System.out.println(num1 + " - " + num2 + " = " + (num1 - num2));
		System.out.println(num1 + " * " + num2 + " = " + num1 * num2);
		System.out.println(num1 + " / " + num2 + " = " + num1 / num2);
		System.out.println(num1 + " % " + num2 + " = " + num1 % num2);
	}
}
```

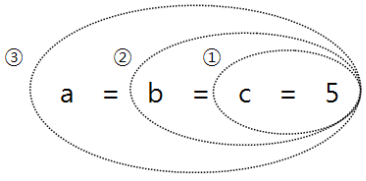
결과는 아래와 같습니다.

```
714 + 500 = 1214
714 - 500 = 214
714 * 500 = 357000
714 / 500 = 1
714 % 500 = 214
```

## 연산자의 우선 순위

*, /, %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.

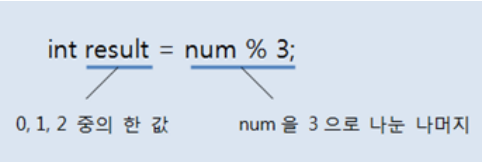
## 논리부정 연산자

Boolean type에만 사용 가능

| 연산식 | | 설명 |
|---|---|---|
| ! | 피연산자 | 피연산자가 true 이면 false 값을 산출<br>피연산자가 false 이면 true 값을 산출 |

### 산술연산자

| 연산식 | | | 설명 |
|---|---|---|---|
| 피연산자 | + | 피연산자 | 덧셈 연산 |
| 피연산자 | - | 피연산자 | 뺄셈 연산 |
| 피연산자 | * | 피연산자 | 곱셈 연산 |
| 피연산자 | / | 피연산자 | 좌측 피연산자를 우측 피연산자로 나눗셈 연산 |
| 피연산자 | % | 피연산자 | 좌측 피연산자를 우측 피연산자로 나눈 나머지를 구하는 연산 |




### 실습예제1

정수 하나를 입력 받고 입력 받은 정수가 초라고 할 때, 몇 시간, 몇 분, 몇 초인가를 구하는 프로그램을 작성하시오.

```
import java.util.Scanner;

public class ArithmeticOperator {
	public static void main (String[] args) {
		int time;
		int second;
		int minute;
		int hour;
		Scanner sc = new Scanner(System.in);
		System.out.print("정수를 입력하세요:"); // 시,분,초로 변환될 정수 입력
		time = sc.nextInt();
		second = time % 60;		// 60으로 나눈 나머지는 초를 의미
		minute = (time / 60) % 60;	// 60으로 나눈 몫을 다시 60으로 나눈 나머지는 분을 의미
		hour = (time / 60) / 60;	// 60으로 나눈 몫을 다시 60으로 나눈 몫은 시간을 의미
		System.out.print(time + "초는 ");
		System.out.print(hour + "시간, ");
		System.out.print(minute + "분, ");
		System.out.println(second + "초입니다.");
	}
}
```

```
정수를 입력하세요:500
500초는 0시간, 8분, 20초입니다.
```

### 증감연산자

변수의 값을 1 증가(++) 혹은 1 감소시키는 연산 수행( --)

### 실습예제2

대입연산자와 증감연산자를 사용해 보는 코드를 코딩해 보세요. 시험에 잘 나오는 부분입니다.

```
public class UnaryOperator {
```

```
        public static void main(String[] args) {
                int opr = 0;
                opr += 3;                       // opr = opr + 3
                System.out.println(opr++);      // opr 출력 후 증가
                System.out.println(opr);
                System.out.println(++opr);      // opr 증가 후 출력
                System.out.println(opr);
                System.out.println(opr--);      // opr 출력 후 감소
                System.out.println(opr);
                System.out.println(--opr);      // opr 감소 후 출력
                System.out.println(opr);
        }
}
```

```
3
4
5
5
5
4
3
3
```

```
class IncrementOpers
{
        public static void main(String[] args)
        {
                int num = 20;

                System.out.println(num++);
                System.out.println(num--);
                System.out.println(--num);
                System.out.println(++num);
        }
}
```

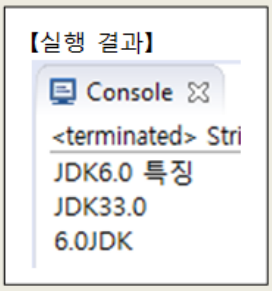
```
20
21
19
20
```

**문자열 연산자**

피연산자에 문자열이 있으면 문자열로 결합

```
String str1 = "JDK" + 6.0;
String str2 = str1 + " 특징";
System.out.println(str2);

String str3 = "JDK" + 3 + 3.0;
String str4 = 3 + 3.0 + "JDK";
System.out.println(str3);
System.out.println(str4);
```

【실행 결과】




### 비교 연산자

| 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 |

### 논리 연산자

| 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 | |

### 실습예제3

비교연산자와 논리연산자를 이해하기 위해 아래 코드를 코딩해 보세요. 코딩할때 기본적으로 사용되는 연산자이므로 완벽하게 이해해야 합니다.

```
public class LogicalOperator {
	public static void main (String[] args) {
		System.out.println('a' > 'b');
		System.out.println(3 >= 2);
		System.out.println(-1 < 0);
		System.out.println(3.45 <= 2);
		System.out.println(3 == 2);
```

```
                System.out.println(3 != 2);
                System.out.println(!(3 != 2));
                System.out.println((3 > 2) && (3 > 4));
                System.out.println((3 != 2) || (-1 > 0));
                System.out.println((3 != 2) ^ (-1 > 0));
        }
}
```

```
false
true
true
false
false
true
false
false
true
true
```

## 비트 연산자

| 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 (배타적 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 | |

## 비트 이동 연산자

| 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.) |

```
class ShiftOpers
{
	public static void main(String[] args)
	{
		System.out.println(2 << 1);
		System.out.println(2 << 2);

		System.out.println(16 >> 1);
		System.out.println(16 >> 2);

		System.out.println(-32  >>> 1);
	}
}
```

```
4
8
8
4
2147483632
```

## 대입 연산자

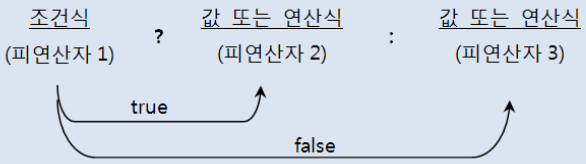
| 구분 | 연산식 | | | 설명 |
|---|---|---|---|---|
| 단순 대입 연산자 | 변수 | = | 피연산자 | 우측의 피연산자의 값을 변수에 저장 |
| 복합 대입 연산자 | 변수 | += | 피연산자 | 우측의 피연산자의 값을 변수의 값과 더한 후에 다시 변수에 저장 (변수=변수+피연산자 와 동일) |
| | 변수 | -= | 피연산자 | 우측의 피연산자의 값을 변수의 값에서 뺀 후에 다시 변수에 저장 (변수=변수-피연산자 와 동일) |
| | 변수 | *= | 피연산자 | 우측의 피연산자의 값을 변수의 값과 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) |

아래코드를 실습하세요

```
class CompoundOpers
{
	public static void main(String[] args)
	{
		int num1=5;
		int num2=4;

		num1 += num2; // num1 = num1 + num2
		System.out.println("num1의 값: " + num1);

		num2 *= num1; // num2 = num2 * num1
		System.out.println("num2의 값: " + num2);
	}
}
```

```
num1의 값: 9
num2의 값: 36
```

## 삼항 연산자




| | | |
|---|---|---|
| int score = 95;<br>char grade = (score>90) ? 'A' : 'B' | = | int score = 95;<br>char grade;<br>if(score>90) {<br>  grade = 'A';<br>} else {<br>  grade = 'B';<br>} |

### 실습예제4

삼항 연산자를 사용하는 예제를 코딩해 보세요.

```
public class TernaryOperator {
	public static void main (String[] args) {
		int a = 3, b = 5;
		System.out.println("두 수의 차는 " + ((a>b)?(a-b):(b-a)));
	}
}
```

```
두 수의 차는 2
```

수고했습니다. 위의 문법은 C언어의 기본 문법이기 때문에 반드시 알아야 합니다.

이에 아래 문제를 해결해 보세요.

**공통: 1114, 1116, 1117, 1118, 1119, 1120, 1121, 1122, 1038, 1046, 1047, 1049, 1063, 1064, 1151, 1153, 1155, 1156, 1157, 1160, 1164, 1230**