Power Java

제5장 변수, 연산자, 수식

이번 장에서 학습할 내용

- 변수 선언
- 기초 자료형과 참조 자료형
- 각종 연산자
- 수식의 계산

변수

- **변수(variable)** : 데이터 값들이 저장되는 메모리 공간

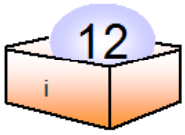


그림 5.2 변수는 데이터를 담아두는 상자와 같다.

자료형

- 자료형(data type)은 자료의 타입
- 기초형과 참조형으로 나누어진다.

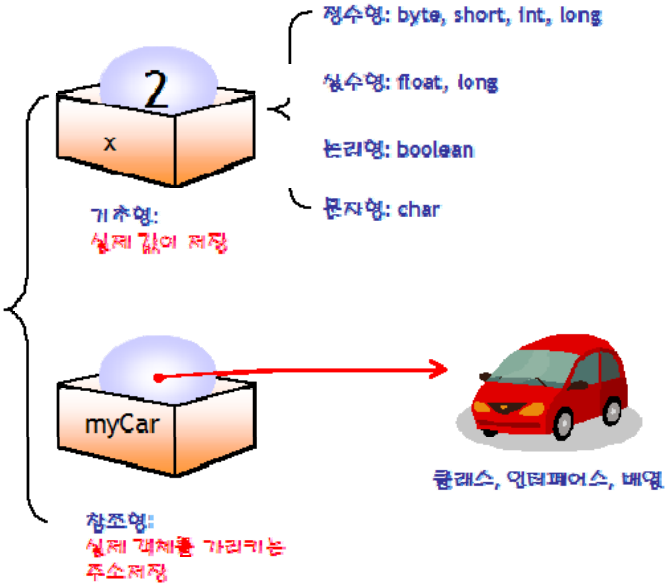
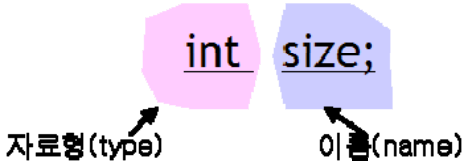


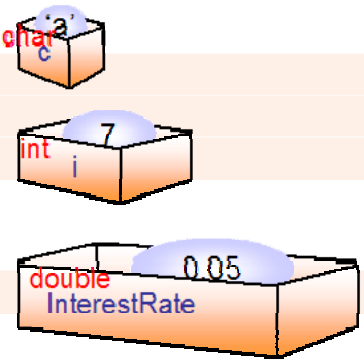
그림 5.4 기초형과 참조형

변수의 선언과 초기화




```
char   c;
int    i;
double interestRate;
```

```
char c = 'a';
int i = 7;
double interestRate = 0.05;
```



하나의 문장에서 변수를 여러 개 선언할 수도 있다.

```
int   index, total = 0;
```


변수의 이름

- 변수의 이름은 식별자(identifier)의 일종
- 변수 이름의 규칙
  - 식별자는 유니코드 문자와 숫자의 조합
  - 식별자의 첫 문자는 일반적으로 유니코드 문자
  - 두 번째 문자부터는 문자, 숫자, _, $ 등이 가능하다.
  - 대문자와 소문자는 구별된다.
  - 식별자의 이름으로 키워드(keyword)를 사용해서는 안 된다.

변수 이름의 예

적법한 변수 선언의 예를 들어보면 다음과 같다.

```
int    speed;
long   earthPopulation;
int    _count;          // _로 시작할 수 있다.
long   $value;          // $로 시작할 수 있다.
int    반복횟수;        // 유니코드를 지원하므로 한글 변수 이름도 가능
int    Henry8;          // 맨 처음이 아니라면 숫자도 넣을 수 있다.
```

잘못된 변수 선언의 예는 다음과 같다.

```
int    1stPrizeMoney;   // 첫글자가 숫자
double super;           // 키워드
int    #ofComputer;     // 첫글자가 허용되지 않는 기호
```


식별자 관례

종 류	사용 방법	예
클래스명	각 단어의 첫글자는 대문자로 한다.	Employee StaffMember, ItemProducer
변수명, 메소드명	소문자로 시작되어 2번째 단어의 첫글자는 대문자로 한다.	balance, width, height payRate, acctNumber, currentImage getMonthDays(), fillRect(), setColor()
static final 변수	변하지 않는 숫자를 나타내는 변수, 모든 글자를 대문자로 한다.	MAX_NUMBER

예제

IsHangulOK.java

```
public class IsHangulOK {
        public static void main(String args[]) {
                int 인덱스 = 0;
                인덱스++;
                System.out.println(인덱스);
        }
}
```

실행결과

```
1
```


중간 점검 문제

1. 변수에 새로운 값이 대입되면 기존의 값은 어떻게 되는가?

2. days와 Days는 동일한 변수인가 아닌가?

3. 다음 중에서 올바르지 않은 변수이름은?
   x, 8items, march09, sales_report, theProfit2009, #ofPlayer

기초형

데이터형	설 명	크기 (비트)	기본값	최소값	최대값
byte	부호있는 정수	8비트	0	-128	127
short	부호있는 정수	16 비트	0	-32768	32767
int	부호있는 정수	32비트	0	-2147483648	2147483647
long	부호있는 정수	64비트	0L	-9223372036854775808	9223372036854775807
float	실수	32	0.0f	약 $\pm 3.4 \times 10^{-38}$ (유효숫자 7개)	약 $\pm 3.4 \times 10^{+38}$ (유효숫자 7개)
double	실수	64	0.0d	약 $\pm 1.7 \times 10^{-308}$ (유효숫자 15개)	약 $\pm 1.7 \times 10^{+308}$ (유효숫자 15개)
char	문자(유니코드)	16	null	'\u0000'(0)	'\uFFFF'(65535)
boolean	true 또는 false	8	false	해당없음	해당없음

정수형

- int는 32비트를 이용하여 약 -21억에서 21억 정도의 정수를 표현
- long은 64비트를 이용
- short는 16비트를 이용하여 -32,768에서 +32767사이의 정수를 표현
- byte는 8비트 정수로서 -128에서 +127까지의 정수를 표현

(Q) 만약 다음과 같이 정수형의 변수에 범위를 벗어나는 값을 대입하면 어떻게 될까?

```
byte number = 300;// 오류!!
```

(A) 컴파일 오류가 발생한다.

정수형 상수

- 상수 또는 리터럴(literal)이란, x = 100;에서 100과 같이 소스 코드에 쓰여 있는 값

- 여러 진법 사용 가능
  - 10진수(Decimal): 14, 16, 17
  - 8진수(Octal): 016, 018, 019
  - 16진수(hexadecimal):0xe, 0x10, 0x11

예제

```
public class IntegerLiterals {
        public static void main(String args[]) {
                int i = 26;                 // 26을 10진수로 표현
                int oi = 032;               // 26을 8진수로 표현
                int xi = 0x1a;              // 26을 16진수로 표현

                System.out.println(i);
                System.out.println(oi);
                System.out.println(xi);
        } // end method main

} // end class IntegerLiterals
```

실행결과

```
26
26
26
```

기호 상수

- 상수에 이름을 주어서 변수처럼 사용

```
final double PI = 3.141592;
```

- 숫자보다 이해하기 쉽고, 값의 변경이 용이하다.

논리형

- 논리형(boolean type)은 true 아니면 false만을 가질 수 있다.

```
boolean condition = true;
```

오류주의

C나 C++ 언어에서는 정수값이 논리형으로 사용된다. 0은 false에 해당되고 나머지값은 true에 해당된다. 그러나 자바에서는 그렇지 않다. 따라서 정수값을 논리형으로 형변환할 수 없다.

실수형

- float는 32비트를 이용하여 실수를 표현
- double은 64비트를 이용하여 실수를 표현

- float는 약 7개 정도의 유효 숫자
- double은 약 15개 정도의 유효 숫자
- 대부분의 경우에는 double을 사용하는 것이 바람직

특수한 실수값

- 양의 무한대(positive infinity): 오버플로우
- 음의 무한대(negative infinity): 언더플로우
- NaN(Not a Number): 유효하지 않은 연산

예제

```
public class FloatPoint {
        public static void main(String args[]) {
                double r1 = 123.5;
                double r2 = 1.2E-300;
                double r3 = 1.2E+300;
                System.out.println(0.0/0.0);          // NaN
                System.out.println(r1/0.0);           // 오버플로우
                System.out.println(r2/r3);            // 언더플로우
                System.out.println(r3/r2);            // 오버플로우
        } // end method main
} // end class FloatPoint
```

실행결과

```
NaN
Infinity
0.0
Infinity
```


실수형 상수

FloatLiterals.java

```
public class FloatLiterals {
        public static void main(String args[]) {
                double r1 = 123.5;
                float r2 = 123.5F;
                double r3 = 1.235e2;
                double r4 = 1.235e-2;

                System.out.println(r1);
                System.out.println(r2);
                System.out.println(r3);
                System.out.println(r4);
        } // end method main
} // end class FloatLiterals
```

double형 상수(64비트)

float형 상수(32비트)

지수표기법

실행결과

```
123.5
123.5
123.5
0.01235
```

문자형

- 아스크 코드가 아니라 유니 코드(unicode)를 사용

```
char ch1 = '가';
char ch2 = '\uac00';        // '가'를 나타낸다.
```

참고: 유니코드

유니코드(unicode)는 전세계의 모든 문자를 컴퓨터에서 일관되게 표현하고 다룰 수 있도록 설계된 산업 표준이다. 유니코드 협회(unicode consortium)가 제정하며, 현재 최신판은 유니코드 5.0이다. 이 표준에는 문자 집합, 문자 인코딩, 문자 정보 데이터베이스, 문자들을 다루기 위한 알고리즘 등이 포함된다.

예제

CharLiterals.java

```
public class CharLiterals {
        public static void main(String args[]) {
                char a = '가';
                char b = 'a';
                char c = '\u0041';
                boolean d = true;

                System.out.println("실행 결과: "+a+" "+b+" "+c+" "+d);
        } // end method main
} // end class CharLiterals
```

실행결과

```
실행 결과: 가 a A true
```

중간 점검 문제

1. 변수가 36에서 5000정도의 값을 저장하여야 하다면 어떤 자료형이 최적인가?

2. 변수가 -3000에서 +3000까지의 값을 저장하여야 하다면 어떤 자료형이 최적인가?

3. 0.025를 지수 표기법으로 표기하여 보라.

4. 어떤 리터럴(상수)이 더 많은 메모리 공간을 차지하는가?
   28.9 28.9F

5. booelan 자료형이 가질 수 있는 값을 전부 쓰시오.

연산자와 피연산자

- 연산자(operator)는 특정한 연산을 나타내는 기호
- 피연산자(operand)는 연산의 대상

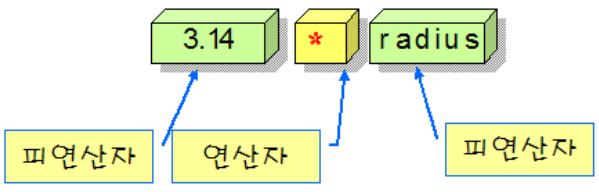


그림 5.11 연산자와 피연산자

자바에서 지원하는 연산자

표 5.4 산술 연산자의 종류

연산자	우선 순위
후위증감	expr++ expr--
단항	++expr --expr +expr -expr ~ !
곱셈	* / %
덧셈	+ -
이동	<< >>>>
관계	< ><= >= instanceof
동등	== !=
비트별 AND	&
비트별 XOR	^
비트별 OR	|
논리적 AND	&&
논리적 OR	||
조건	? :
대입	= += -= *= /= %= &= ^= |= <<= >>= >>>=

수식

- 수식이란 상수나 변수, 함수와 같은 피연산자들과 연산자의 조합

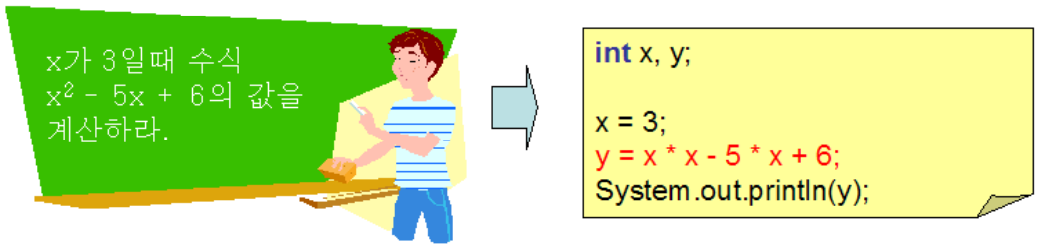


그림 5.12 수식의 예

대입 연산자

- 대입 연산자(=)는 왼쪽 변수에 오른쪽 수식의 값을 계산하여 저장
- 대입 연산자 == 할당 연산자 == 배정 연산자라고도 한다.

x = 10; // 상수 10을 변수 x에 대입한다.

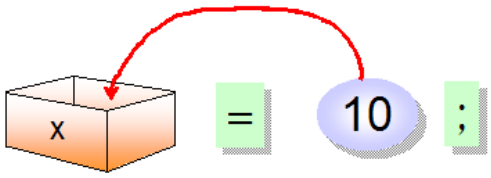


그림 5.13 대입 연산

산술 연산자

표 5.5 산술 연산자의 종류

연산자	기호	의미	예
덧셈	+	x와 y를 더한다	x+y
뺄셈	-	x에서 y를 뺀다.	x-y
곱셈	*	x와 y를 곱한다.	x*y
나눗셈	/	x를 y로 나눈다.	x/y
나머지	%	x를 y로 나눌 때의 나머지값	x%y

ArithmeticOperator.java

```
public class ArithmeticOperator {

    public static void main (String[] args){

        int result;
        double resultDouble;

            result = 3 + 2;
            System.out.println(result);
            result = 3 - 2;
            System.out.println(result);
            result = 3 * 2;
            System.out.println(result);
            result = 3 / 2;        // 정수 계산의 경우, 소수점 이하부분은 없어진다.
            System.out.println(result);
            resultDouble = 3.0 / 2.0;
            System.out.println(resultDouble);
            result = 3 % 2;
            System.out.println(result);
    }
}
```

실행결과

```
5
1
6
1
1.5
1
```


+ 연산자는 문자열을 결합

StringOperator.java

```
public class StringOperator {
    public static void main(String[] args){
        String s1 = "Hello";
        String s2 = " World";
        String s3 = s1+s2;
        System.out.println(s3);
    }
}
```

실행결과

```
Hello World
```

단항 연산자

연산자	의미
+x	x를 양수로 만든다.
-x	x를 음수로 만든다.
++x	x값을 먼저 증가한 후에 다른 연산에 사용한다. 이 수식의 값은 증가된 x값이다.
x++	x값을 먼저 사용한 후에, 증가한다. 이 수식의 값은 증가되지 않은 원래의 x값이다.
--x	x값을 먼저 감소한 후에 다른 연산에 사용한다. 이 수식의 값은 감소된 x값이다.
x--	x값을 먼저 사용한 후에, 감소한다. 이 수식의 값은 감소되지 않은 원래의 x값이다.

예제

UnaryOperator

```
public class UnaryOperator {
      public static void main(String[] args){
        int x = 1;
        int y = 1;

        int nextx = ++x; // x의 값이 사용되기 전에 증가된다. nextx는 2가 된다.
        int nexty = y++; // y의 값이 사용된 후에 증가된다. nexty는 1이 된다.
      }
}
```

복합 대입 연산자

복합 대입 연산자	의미
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

표 5.8 복합 대입 연산자

관계 연산자

표 5.9 관계 연산자

연산자 기호	의미	사용예
==	x와 y가 같은가?	x == y
!=	x와 y가 다른가?	x != y
>	x가 y보다 큰가?	x > y
<	x가 y보다 작은가?	x < y
>=	x가 y보다 크거나 같은가?	x >= y
<=	x가 y보다 작거나 같은가?	x <= y

ComparisonOperator.java

```
public class ComparisonOperator {

    public static void main(String[] args){
        int x = 3;
        int y = 4;
        System.out.println(x == y);
        System.out.println(x != y);
        System.out.println(x > y);
        System.out.println(x < y);
        System.out.println(x <= y);
    }
}
```

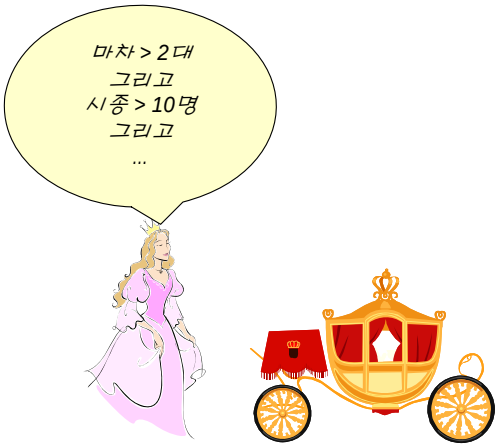
실행결과

```
false
true
false
true
true
```


논리 연산자

연산자 기호	사용예	의미
&&	x && y	AND 연산, x와 y가 모두 참이면 참, 그렇지 않으면 거짓
||	x || y	OR 연산, x나 y중에서 하나만 참이면 참, 모두 거짓이면 거짓
!	!x	NOT 연산, x가 참이면 거짓, x가 거짓이면 참

표 10 논리 연산자



예제

LogicalOperator.java

```
public class LogicalOperator {

        public static void main(String[] args){
            int x = 3;
            int y = 4;
            System.out.println((x == 3) && (y == 7)) ;
            System.out.println((x == 3 || y == 4)) ;

        }
}
```

실행결과

```
false
true
```


삼항 연산자

condition ? exp1 : exp2

- condition이 참이면 exp1이 선택되고 그렇지 않으면 exp2가 선택된다.
- (예) max_value = (x > y) x : y; / / 최대값 계산

중간 점검 문제

1. 다음의 각 변수의 값을 적어보라.

```
int x = 1;
int y = 1;
int a = ++x * 2;// a의 값은 _____
int b = y++ * 2; // b의 값은 _____
```

2. 다음 수식의 값을 쓰시오.

```
12/5 - 3
5 + 19%3
```


연산자의 우선 순위

- 어떤 연산자를 먼저 계산할 것인가?

```
result = x * y % z - a / b
         ⑤ ①  ② ④ ③
```

- 괄호 안은 먼저 계산된다.

```
m = ( x + y + z ) / 3
    ④  ①  ②  ③
```

연산자의 결합 규칙

- 만약 같은 우선 순위를 가지는 연산자들이 여러 개가 있으면 어떤 것을 먼저 수행하여야 하는가?

```
result = x % y * z;
         ③ ①  ②
```

대부분의 연산자가 왼쪽에서 오른쪽으로 수행되지만 몇개의 연산자는 오른쪽에서 왼쪽으로 연산이 진행된다. 대입 연산자가 대표적이다.

```
x = y = w = z ;
  ③  ② ①
```


중간 점검 문제

1. 다음의 수식에서 연산의 순서를 적으시오.

(1) x = y = 3 / 5 * 2 % 6;

(2) y = a * x * x + b * x + c;

비트 연산자

표 5.4 비트 연산자

연산자	의미	예
~	비트별 NOT	~(0x0FFF)은 0xfffff000이 된다.
&	비트별 AND	(0x0FFF & 0xFFF0)은 0x0FF0이 된다.
^	비트별 XOR	(0x0FFF ^ 0xFFF0)은 0xF00F이 된다.
|	비트별 OR	(0x0FFF | 0xFFF0)은 0xFFFF이 된다.
<<	비트 왼쪽 이동	0xFFF << 4은 0xFFF0이 된다.
>>	비트 오른쪽 이동	0xFFF0 >> 4은 0xFFF이 된다.

예제

BitOperator.java

```
public class BitOperator {

        public static void main(String[] args) {
                int x = 0x0fff;
                int y = 0xfff0;
                System.out.printf("%x\n", (x & y));
                System.out.printf("%x\n", (x | y));
                System.out.printf("%x\n", (x ^ y));
                System.out.printf("%x\n", ~x);
                System.out.printf("%x\n", (x << 4));
                System.out.printf("%x\n", (x >> 4));
        }
}
```

실행결과

```
ff0
ffff
f00f
fffff000
fff0
fff
```

중간 점검 문제

1. 변수 y, z, a, b의 값은?

```
int x = 0xff0f;

int y = x << 4;

int z = x >> 4;

int a = x & 0xf0ff;

int b = x | 0xf0ff;
```


형변환

- 형변환(cast)는 어떤 자료형의 값을 다른 자료형의 값으로 바꾸어 주는 연산

```
(새로운 자료형) 수식;
```

```
y = (double) x;
```

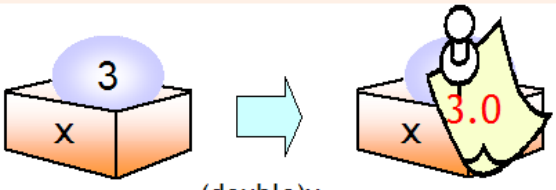


그림 5.21 형변환

축소 변환

- i = (**int**) 12.5;// i에는 12만 저장
- (주의) 위의 예에서는 소수점 이하는 사라진다.

확대 변환

- 더 큰 크기의 변수로 값을 이동하는 변환
- **double** d = (**double**) 100;// 정수 100이 변수 d에 100.0으로 형변환 되어서 저장

예제

TypeConversion.java

```
public class TypeConversion {
    public static void main(String args[]) {
        int i;
        double f;

        f = 5 / 4;                          // (a) f는 1.0
        System.out.println(f);
        f = (double) 5 / 4;                 // (b) f는 1.25
        System.out.println(f);
        f = 5 / (double) 4;                 // (c) f는 1.25
        System.out.println(f);
        f = (double) 5 / (double) 4;        // (d) f는 1.25
        System.out.println(f);
        i = (int) 1.3 + (int) 1.8;          // (e) i는 2
        System.out.println(i);
    } // end method main
} // end class TypeConversion
```

실행결과

```
1.0
1.25
1.25
1.25
2
```


Q & A