Chapter 2. 변수와 타입

2.1 변수

2.1.1) 변수란?

: 하나의 값을 저장할 수 있는 메모리 공간

2.1.2) 변수 선언

- **변수 선언**

```
타입     변수 이름
int     age;        // 정수 값을 저장할 age 변수
double  value;      // 실수 값을 저장할 value 변수

int x, y, z;        // 여러개 선언 가능
```

- **작성 규칙**

작성 규칙	예
첫 번째 글자는 문자이거나 '$', '_' 이어야 한다. **(숫자로 시작X)**	가능 : price,$price_company 안됨 : 1v, @speed
영어 대소문자가 구분된다.	Age != age
첫 문자는 영어 소문자로, 다른단어가 붙을 경우 첫 문자를 대문자로 한다. (관례)	maxSpeed, firstName
문자 수 제한 없다.	
자바 예약어는 사용할 수 없다.	(뒤쪽에서 배우도록 한다)

2.1.3) 변수의 사용

변수값 저장

변수 초기화 : 변수에 초기값을 주는 행위

```
int score;      // 변수 선언
score = 90;     // 값 저장

int score = 90; // 선언과 동시에 값을 저장할 수 있다.
```

- **리터럴(literal)** : 소스 코드 내에서 직접 입력된 값
  - **종류** : 정수 리터럴, 실수 리터럴, 문자 리터럴, 논리 리터럴
  - **상수와 리터럴 차이** : 상수는 값을 한 번 저장하면 변경할 수 없는 변수이다. 리터럴은 값이 변경될 수도 있다.

정수 리터럴

```
0, 75, -100     // 소수점이 없는 정수 리터럴은 10진수
02, -04         // 0으로 시작되는 리터럴은 8진수
0x5, 0xA, 0xB3  // 0x 또는 0X 로 시작하고 0~9, A~F(a~f) 리터럴은 16진수
```

> 이외에 byte, char, short, int, long 도 있지만 조금 후에 설명

실수 리터럴

```
0.25, -3.14     // 소수점이 있는 리터럴은 10진수 실수

// E(e) 가 있는 리터럴은 10진수 지수와 가수로 간주
5E7         // 5 x 10^7
0.12E-5     // 0.12 x 10^-5
```

> 이외에 float, double도 있다.

문자 리터럴

```
'A', '한', '\t', '\n'    // 작은 따옴표(')로 묶은 텍스트는 하나의 문자 리터럴
```

- **역슬래쉬가 붙은 문자 리터럴** : 이스케이프 문자 로써 특수한 용도로 사용

이스케이프 문자	용도	유니코드
'\t'	수평 탭	0x0009
'\n'	줄 바꿈	0x000a
'\r'	리턴	0x000d
'\"' (큰 따옴표)	"	0x0022
'\'' (작은 따옴표)	'	0x0027
'\\'	\	0x005c
'\u16진수'	16진수에 대항하는 유니코드	0x0000 ~ 0xffff

> 문자 리터럴을 저장할 수 있는 타입은 **char** 하나 이다.

문자열 리터럴

: 큰 따옴표로 묶은 텍스트

```
"대한민국"
"탭 만큼 이동 \t 합니다"    // 문자열 내부에서도 이스케이프 문자를 사용 가능
```

> 문자열 리터럴을 저장할 수 있는 타입은 **String** 하나 이다.

논리 리터럴

```
true, false
```

> 논리 리터럴을 저장할 수 있는 타입은 **boolean** 하나 이다.

변수값 읽기

: 변수는 **초기화가 되어야만** 읽을 수 있다.

- **잘못된 예시**

```
int value;                  // 초기화를 하지 않음
int result = value + 10;    // value 값을 읽음

// 위의 코드는 value가 초기화되어 있지 않아 에러가 발생한다.

int value = 30;
int result = value + 10;

// 이처럼 value를 초기화해야만 변수를 읽어올때 에러가 발생하지 않는다.
```

- **예제 코드**

```
public class VariableExample {
    public static void main(String[] args){
        // 10을 변수 value의 초기값으로 저장
        int value = 10;

        // 변수 value 값을 읽고 10을 더하고
        // 연산의 결과값을 변수 result의 초기값으로 저장
        int result = value + 10;

        // 변수 result 값을 읽고 콘솔에 출력
        System.out.println(result);
    }
}
```

실행 결과

```
20
```

2.1.4) 변수의 사용 범위

: 변수는 선언된 **중괄호{} 블록 내에서만** 사용이 가능하다. (로컬 변수 : 메소드 블록 내에서 선언된 변수) **로컬 변수**는 메소드 실행이 끝나면 메모리에서 자동으로 없어진다.

```
public class VariableExample {
    // 클래스 블록
    public static void main(String[] args){
        // 메소드 블록
        int value = 10;
        int result = value + 10;
        System.out.println(result);
    }
}
```

실행 결과

```
20
```

- **제어문 블록에서 선언된 변수**

  : 제어문 블록에서 선언된 변수는 해당 제어문 블록 내에서만 사용이 가능하다

```
public static void main(String[] args){
    int var1;       // 해당 메소드 블록에서 사용가능

    if(...){
        int var2;   // 해당 if 블록에서만 사용가능
    }

    for(...){
        int var3;   // 해당 for 블록에서만 사용가능
    }
}
```

- **예제 코드**

```
public class VariableScopeExample {
    public static void main(String[] args){
        int v1 = 15;    // 메소드 블록에 선언
        if(v1 > 10){
            int v2 = v1 - 10;   // if 블록에 선언
        }
        // v2 변수 사용 불가하여 컴파일 에러
        int v3 = v1 + v2 + 5;
    }
}
```

2.2 데이터 타입

: 타입에 따라 저장할 수 있는 **값의 종류와 범위**가 달라진다.

2.2.1) 기본(원시: primitive) 타입

값의 종류	기본 타입	메모리 사용 크기
정수	**byte**	1 byte
	char	2 byte
	short	2 byte
	int	4 byte
	long	8 byte
실수	**float**	4 byte
	double	8 byte
논리	**boolean**	1 byte

> 메모리에는 0과 1을 저장하는 비트가 있다. (1 byte = 8 bit)

2.2.2) 정수 타입(byte, char, short, int ,long)

- **메모리 크기 순으로 나열**

  byte < **char** == **short** < **int** < **long**

byte 타입

- 색상 정보 및 파일 또는 이미지 등의 이진 데이터를 처리할 때 주로 사용.
- **값의 범위** : -128 ~ 127
- **예제 코드**

```
public class ByteExample {
    public static void main(String[] args){
        byte var1 = -128;
        byte var2 = -30;
        byte var3 = 0;
        byte var4 = 30;
        byte var5 = 127;
        // byte var6 = 128;  컴파일 에러!!

        System.out.println(var1);
        System.out.println(var2);
        System.out.println(var3);
        System.out.println(var4);
        System.out.println(var5);
    }
}
```

실행 결과

```
-128
-30
0
30
127
```

- **오버플로우** : 저장할 수 있는 값의 범위를 초과해서 값이 저장될 경우 엉터리 값이 변수에 저장된다.
  - **예제 코드**

```
public class GarbageValueExample {
    public static void main(String[] args){
        byte var1 = 125;
        int var2 = 125;
        for(int i=0; i<5; i++){
            var1++;
            var2++;
            System.out.println("var1: " + var1 + "\t" + "var2 : " + var2);
        }
    }
}
```

실행 결과

```
var1: 126   var2 : 126
var1: 127   var2 : 127
var1: -128  var2 : 128
var1: -127  var2 : 129
var1: -126  var2 : 130
```

> 127에서 증가시킬때 var1은 byte 값의 범위를 초과해서 오버플로우가 발생하여 **-128** 값이 나온다. var2는 int 타입이므로 오버플로우가 발생하지 않는다.

char 타입

: 자바는 모든 문자를 **유니코드**로 처리한다. 유니코드 음수가 없기 때문에 char 타입의 변수에는 음수 값을 저장할 수 없다. 작은 따옴표로 감싼 문자를 대입하면 해당 문자의 유니코드가 저장된다.

```
char var1 = 'A';        // 유니코드 : 0x0041
char var2 = 'B';        // 유니코드 : 0x0042
char var3 = '가';
char var4 = '나';
```

- **char 변수에 저장된 유니코드를 알고 싶을 때**

```
char c = 'A';       // 문자를 c에 저장
int uniCode = c;    // c에 저장된 문자를 유니코드로 저장
```

- **char 타입 변수 예제 코드**

```
public class CharExample {
    public static void main(String[] args){
        char c1 = 'A';      // 문자를 직접 저장
        char c2 = 65;       // 10 진수로 저장
        char c3 = '\u0041'; // 16 진수로 저장

        char c4 = '가';      // 문자를 직접 저장
        char c5 = 44032;    // 10 진수로 저장
        char c6 = '\uac00'; // 16 진수로 저장

        int uniCode = c1;   // 유니코드 얻기

        System.out.println(c1);
        System.out.println(c2);
        System.out.println(c3);
        System.out.println(c4);
        System.out.println(c5);
        System.out.println(c6);
        System.out.println(uniCode);
    }
}
```

실행 결과

```
A
A
A
가
가
가
65
```

- **빈 문자 대입 에러**

```
char c = '';    // 컴파일 에러
```

> 공백 하나라도 포함해서 초기화해야 에러가 발생하지 않는다.

문자열

: 문자열을 저장하고 싶다면 **String 타입** 변수를 선언하고, 큰 따옴표로 감싼 문자열을 리터럴에 대입하면 된다.

```
String name = "홍길동";
```

- **빈 문자 대입 에러**

```
String str = "";        // 정상 작동
```

> String 변수는 char와 다르게 큰 따옴표 두 개를 연달아 붙인 빈 문자를 대입해도 된다.

short 타입

: 2 byte 정수값을 저장할 수 있는 데이터 타입이다. 비교적 자바에서는 잘 사용되지 않는다.

int 타입

: 4 byte **정수값**을 저장하는 데이터 타입이다. 자바에서는 정수 연산을 **4 byte로 처리**하기 때문에 byte 타입 또는 short 타입의 변수를 **+ 연산**하면 int 타입이 된다.

- **10을 여러 진수로 저장하는 코드**

```
int number = 10;        // 10 진수
int octNumber = 012;    // 8 진수
int hexNumber = 0xA;    // 16 진수
```

- **int 타입 변수 초기화 예제 코드**

```
public class IntExample {
    public static void main(String[] args){
        int var1 = 10;      // 10 진수로 저장
        int var2 = 012;     // 8  진수로 저장
        int var3 = 0xA;     // 16 진수로 저장

        System.out.println(var1);
        System.out.println(var2);
        System.out.println(var3);
    }
}
```

실행 결과

```
10
10
10
```

long 타입

: 8 byte 로 표현되는 정수값을 저장할 수 있는 데이터 타입.

- long 타입의 변수를 초기화할 때에는 정수값 뒤에 **대문자 L이나 소문자 l** 을 붙일 수 있다. : L을 붙이는 이유는 4 byte 정수 데이터가 아니라 **8 byte 정수 데이터임**을 컴파일러에게 알려주기 위함.(int 타입의 저장 범위를 넘어서는 **큰 정수**는 반드시 **'L'** 을 붙여야 한다.)

- **long 타입 변수 초기화 예제 코드**

```
public class LongExample {
    public static void main(String[] args){
        long var1 = 10;
        long var2 = 20L;
        // long var 3 = 1000000000000;  컴파일 에러
        long var4 = 1000000000000L;

        System.out.println(var1);
        System.out.println(var2);
        System.out.println(var4);
    }
}
```

실행 결과

```
10
20
1000000000000
```

> var3 은 L을 붙이지 않았기 때문에 컴파일 에러가 된다.

2.2.3 실수 타입(float, double)

실수 타입	float	double
바이트 수	4	8

- **float과 double**

  : 높은 정밀도가 요구하는 계산에서는 double을 사용해야 한다.

  - **변수 초기화 방법**

```
double var1 = 3.14;
float var2 = 3.14;  // 컴파일 에러, float는 실수 뒤에 f를 붙어야 한다.
float var3 = 3.14F;
```

- **10의 지수를 나타내는 e를 포함하여 변수에 저장**

```
int var6 = 3000000;     // 3000000
double var7 = 3e6;      // 3000000
float var8 = 3e6f;      // 3000000
double var9 = 2e-3;     // 0.002
```

- **float 과 double 초기화 예제 코드**

```
public class FloatDoubleExample {
    public static void main(String[] args){
        // 실수값 저장
        double var1 = 3.14;
        // float var2 = 3.14;   컴파일 에러
        float var3 = 3.14F;

        // 정밀도 테스트
        double var4 = 0.123456789123456789;
        float var5 = 0.1234567890123456789F;

        System.out.println("var1 : " + var1);
        System.out.println("var3 : " + var3);
        System.out.println("var4 : " + var4);
        System.out.println("var5 : " + var5);

        // e 사용하기
        int var6 = 3000000;
        double var7 = 3e6;
        float var8 = 3e6F;
        double var9 = 2e-3;

        System.out.println("var6 : " + var6);
        System.out.println("var7 : " + var7);
        System.out.println("var8 : " + var8);
        System.out.println("var9 : " + var9);
    }
}
```

실행 결과

```
var1 : 3.14
var3 : 3.14
var4 : 0.12345678912345678
var5 : 0.12345679
var6 : 3000000
var7 : 3000000.0
var8 : 3000000.0
var9 : 0.002
```

2.2.4 논리 타입(boolean)

: 1 byte 로 표현되는 **논리값(true/false)**을 저장할 수 있는 데이터 타입이다. **두 가지 상태값**을 지정할 필요성이 있을 경우에 사용된다.

- **boolean 타입 예제 코드**

```
public class BooleanExample {
    public static void main(String[] args){
        boolean stop = true;
        // 조건문에 stop이 true이면 "중지합니다." 를
        // false면 "시작합니다." 를 출력시킨다.
        if(stop){
            System.out.println("중지합니다.");
        }
        else{
            System.out.println("시작합니다.");
        }
    }
}
```

실행 결과

```
중지합니다.
```

2.3 타입 변환

: 데이터 타입을 다른 데이터 타입을 변환하는 것. **ex) byte 타입 <--> int 타입**

- **타입 변환 종류**
  - **자동(묵시적) 타입 변환**
  - **강제(명시적) 타입 변환**

2.3.1) 자동 타입 변환