3. 데이터 타입과 변수 선언

Java 프로그래밍에서 사용되는 데이터의 종류는 사용 방식에 따라 변수, 상수 그리고 리터럴로 나뉘며 처리되는 데이터 값의 종류에 따라서 타입이라는 것이 정해져 사용된다. Java 에서 사용하는 데이터의 타입은 기본형과 참조형으로 나뉘어 각각의 특성에 맞게 처리된다.

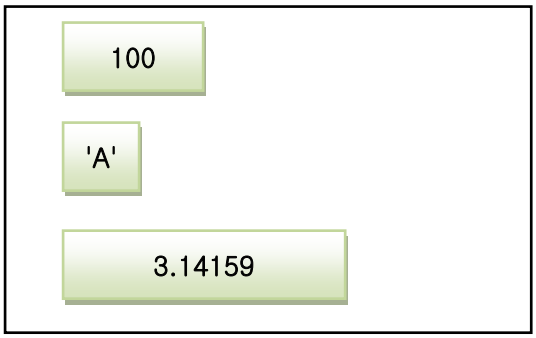
3.1 데이터의 타입

■ 기본형과 참조형

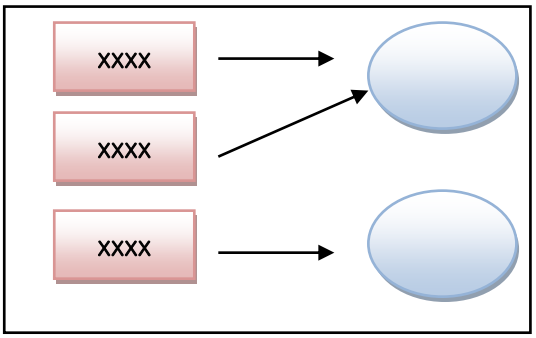
Java 의 데이터 타입은 기본형(primitive)과 참조형(reference)으로 나뉜다. 기본형은 실제 계산할 데이터의 값을 저장하여 처리하는 타입이고 참조형은 다른 곳에 저장되어 있는 데이터 값을 참조하는 방식으로 처리하는 타입이다.

기본형은 논리형/문자형/정수형/실수형이 있으며 어떠한 타입으로 변수를 선언하는가에 따라서 메모리상에 할당되는 영역의 크기가 다르다.

참조형은 처리하고자 하는 데이터가 객체일 경우에 사용되는 유형이므로 객체형이라고도 한다. Java 에서 만들어지는 객체들 즉, 인스턴스들을 참조하여 사용할 수 있도록 하는 유형으로서 사용하려는 객체에 따라서 참조형이 사용되므로 'Java에서 지원되는 참조형 타입은 몇 개다'라고 그 개수를 예측할 수는 없다. 사용하려는 객체가 String 형 객체이면 String 형, Date 형 객체이면 Date 형을 참조형으로 사용하게 되는 것이다. 참조형으로 변수를 선언하면 참조하는 객체의 참조값 즉, 논리적인 주소값(0x00000000~0xffffffff)을 가지게 되므로 메모리상에 할당되는 영역의 크기는 모든 참조형 변수가 동일하게 4바이트 할당되어 사용된다.



[그림 3-1 기본형]



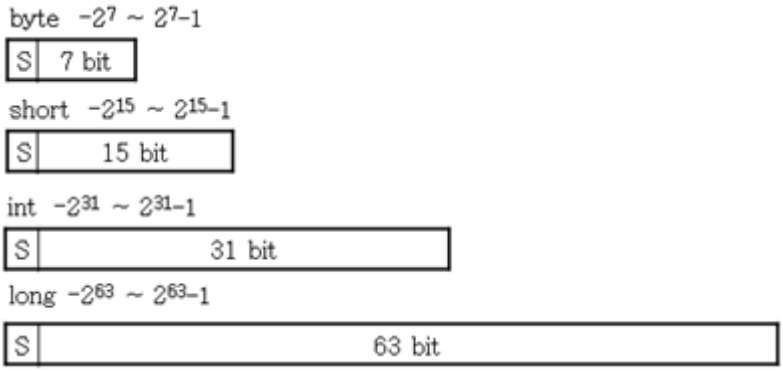
[그림 3-2 참조형]

■ 정수형

정수 데이터를 저장하고 처리하는 타입으로서 byte, short, int 그리고 long 형, 네 가지 타입이 있으며 타입에 따라서 저장하여 처리할 수 있는 값의 크기(표현 범위)가 다르다.

데이터형	크기	표현 범위
byte	1바이트	-2^7 ~ 2^7-1 (-128 ~ 127)
short	2바이트	-2^15 ~ 2^15-1 (-32768 ~ 32767)
int	**4바이트**	**-2^31 ~ 2^31-1 (-2147483648 ~ 2147483647)**
long	8바이트	-2^63 ~ 2^63-1 (-9223372036854775808 ~ 9223372036854775807)

int 형이 디폴트 타입이며 int 타입으로 처리할 수 없을 정도로 큰 값이거나 작은 값인 경우에는 long 형을 사용한다. 다음 그림과 같이 부호가 있는 정수 데이터만을 처리할 수 있다.

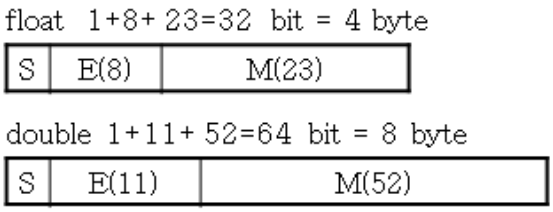


■ **실수형**

부동 소수점 형식의 실수 데이터를 저장하고 처리하는 타입으로서 float 와 double 형이 사용되며 double 형이 디폴트 타입이다. 소수점 이하의 값을 유효 값으로 처리하려면 반드시 실수형으로 처리해야 한다.

데이터형	크기	표현 범위
float	4바이트	1.4E-45 ~ 3.4028235E38
double	8바이트	4.9E-324 ~ 1.7976931348623157E308

숫자를 저장할 때 다음 그림과 같이 지수부와 가수부로 나누어 부동 소수점 형식으로 저장하므로 4바이트 크기의 float 형이 8 바이트 크기의 long 형보다 더 큰 범위의 값을 저장하고 처리할 수 있다.

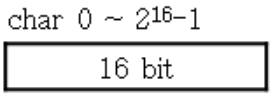


■ **문자형**

하나의 문자 데이터 값을 저장하고 처리하는데 사용되는 데이터 타입이다. Java 언어에서 처리되는 문자 데이터 값은 Unicode 로서 Unicode 의 경우에는 데이터의 코드값을 2바이트로 처리하기 때문에 Java 에서는 하나의 문자 데이터 값을 처리하는 char 형의 크기를 2바이트로 지원한다.

데이터형	크기	표현 범위
char	2바이트	'₩u0000 ' ~ '₩uffff'

char 형은 2 바이트 크기의 영역에 정수형 값으로 정해져 있는 영문 알파벳 이나 한글과 같은 문자 데이터의 값을 저장하여 처리하는 타입으로서 다음 그림과 같이 2바이트 공간에 양의 값(표준화 되어 있는 문자의 코드 값)만을 저장하여 수 있는 정수형이라고 할 수 있다.



■ **논리형**

true 와 false 중 하나를 값으로 갖으며, 조건식과 논리적 계산에 사용되는 데이터 타입이다. 메모리상에는 1 바이트의 영역이 할당되어 사용되는 유형이다.

데이터형	크기	표현 범위
boolean	1바이트만 사용	true 또는 false

조건식이나 논리식을 사용해야 하는 제어문이나 연산자에서는 반드시 boolean 형을 사용해야 한다.

3.2 데이터의 종류

프로그램에서의 데이터는 변수(variable), 상수(constant) 그리고 리터럴(literal) 로 사용될 수 있다. 변수란 하나의 데이터 값을 저장하기 위한 메모리상의 공간으로서 저장되는 데이터 값을 얼마든지 변경할 수 있다. 상수란 데이터 값을 한번만 저장할 수 있는 메모리상의 공간으로서 변수의 변형이라고 할 수 있다. 리터럴은 프로그램에서 사용되는 고정된 데이터 값을 의미한다.

■ 리터럴

리터럴은 프로그램에서 사용되는 고정된 데이터 값으로서 Java 에서는 정수형 리터럴, 실수형 리터럴, 문자형 리터럴, 논리형 리터럴 그리고 문자열형 리터럴이 사용될 수 있다.

▶ 정수형 리터럴

정수값을 나타내는 리터럴로서 int 형 리터럴과 long 형 리터럴로 나뉜다. 정수형 숫자는 디폴트로 int 형 로 인식되며 long 형 리터럴은 숫자값 뒤에 L(소문자도 가능)을 붙여서 표현한다. 10진수 형식, 8진수 형식 그리고 16진수 형식으로 나타낼 수 있다.

int 형 리터럴	10진수	1, 10, 100, 1000
	8진수	01, 010, 0100, 01000
	16진수	0x1, 0x10, 0x100, 0x1000
long 형 리터럴	10진수	1L, 10L, 100L, 1000L
	8진수	01L, 010L, 0100L, 01000L
	16진수	0x1L, 0x10L, 0x100L, 0x1000L

일반적으로 표현하려는 값이 -2147483648 ~ 2147483647 사이이면 int 형 리터럴을 사용하며 이 외의 값인 경우에는 long 형 리터럴을 사용한다.

▶ 실수형 리터럴

실수값을 나타내는 리터럴로서 float 형 리터럴과 double 형 리터럴로 나뉜다. 실수형 숫자는 디폴트로 double 형 으로 인식되며 float 형 리터럴은 숫자값 뒤에 f(대문자도 가능)를 붙여서 표현한다.

float 형 리터럴	고정소수점 방식	3.14f, 12.345f, 3140.0f
	지수 방식	3.14e3f, 1e1f
double 형 리터럴	고정소수점	3.14, 12.345, 3140.0
	지수 방식	3.14e3, 1e1

▶ 논리형 리터럴

Java 에서 논리형 리터럴은 true 와 false 두 가지 값이 사용된다. true 와 false 는 Java 키워드이지만 리터럴로 사용되며 true 는 참의 값을 false 는 거짓의 값을 나타낸다. boolean 타입의 변수를 초기화 하거나 조건식이나 논리식을 필요로 하는 곳에 사용 가능하다.

▶ 문자형 리터럴

하나의 문자를 데이터 값으로 나타내는 리터럴이다. 앞과 뒤에 단일 인용부호(')를 사용하여 표현하며 안에는 반드시 하나의 문자가 존재해야 한다. 단일 인용부호(') 안에 하나의 문자 기호를 사용해도 되고 Unicode 코드 값으로 나타낼 수도 있는데 이런 경우에는 코드값에 ₩u 을 붙여서 Unicode 임을 나타내야 한다. ₩n, ₩t, ₩₩, ₩", ₩' 와 같은 이스케이프 문자도 올 수 있다.

```
문자형 리터럴 : 'A', '1', '가', ' ', '₩u0041', '₩uac00'
```

▶ 문자열형 리터럴

0 개 이상으로 구성되는 문자들을 데이터 값으로 나타내는 문자열이다. 이중 인용부호(")를 사용하여 표현하며 안에는 0 개 이상의 문자들이 올 수 있다. 문자가 하나도 없는 문자열 리터럴("")을 널문자열이라고 한다. 문자열의 길이에는 제한이 없으며 어떠한 문자든 구성될 수 있으며 ₩n, ₩t, ₩₩, ₩", ₩' 와 같은 이스케이프 문자도 올 수 있다.

```
문자형 리터럴 : "A", "1", "가", " ", "",  "ABC", "123", "가나다"
```

문자열 리터럴은 String 형 인스턴스로 메모리 할당되고 사용되어 다른 리터럴과는 다르게 참조형 리터럴로 사용되어 문자열 리터럴에 멤버 연산자(.)를 사용하여 String 클래스에서 제공되는 문자열 처리 메서드를 호출할 수 있다.

```
"ABC".length()          // 문자열 리터럴의 길이를 리턴한다.
"12345".substring(2)    // 세 번째 위치부터 끝까지의 부분문자열을  리턴한다.
"가나다".charAt(0)      // 첫 번째 위치의 문자 하나만 char 형으로 리턴한다.
```

▶ 참조형 리터럴

null 은 참조형 변수에만 할당할 수 있는 리터럴로서 **참조되는 대상이 없음**을 의미하는 참조형 리터럴이다.

```
StringBuffer buffer = null; // buffer 변수는 StringBuffer 인스턴스를 참조할 수 있는
                            // 참조형 변수이지만 아직은 참조되는 대상이 없음을
                            // 의미하는 null 로 초기화 하고 있다.
int number = null;          // number 는 기본형 변수이므로 참조형 리터럴인 null
로
```

■ **변수**

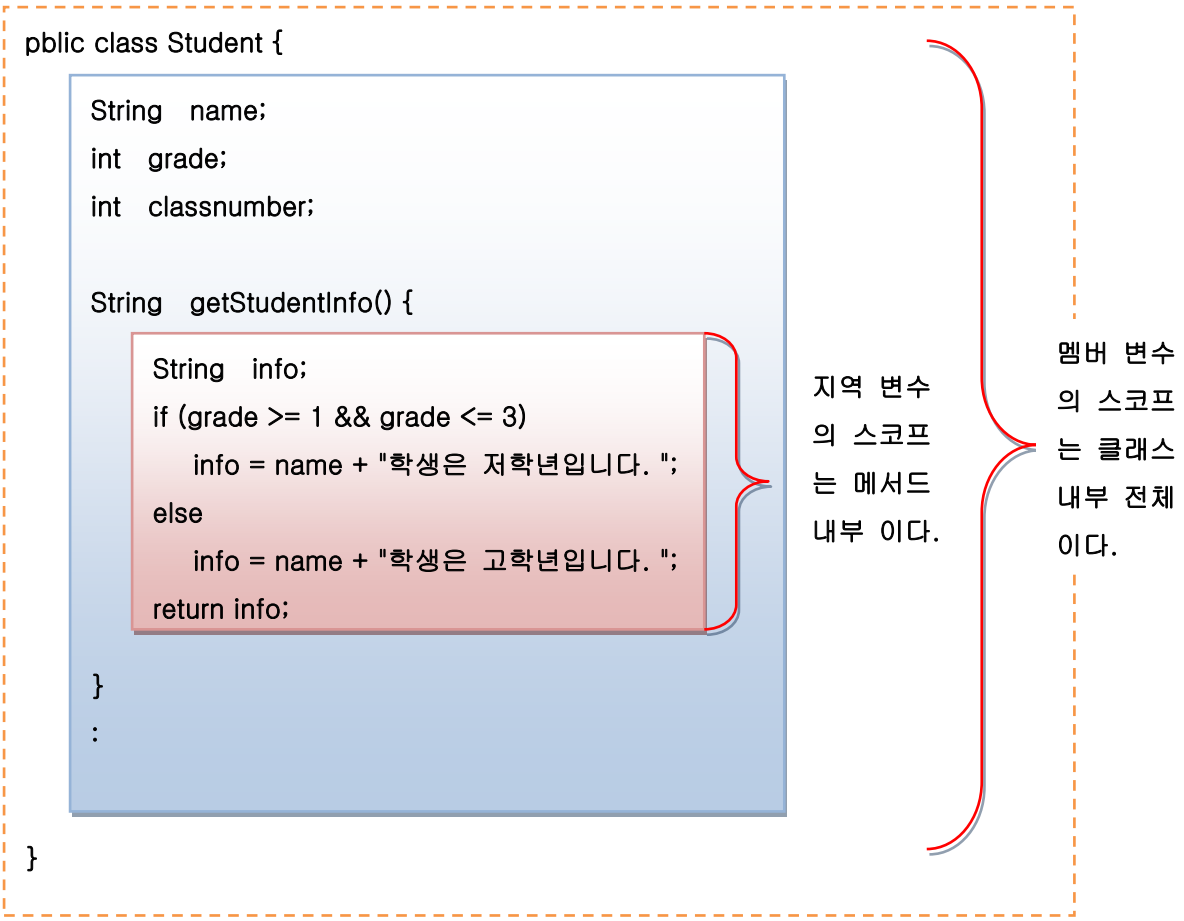
변수란 하나의 데이터 값을 저장하기 위한 메모리상의 공간으로서 연산자를 사용하여 저장된 데이터 값을 변경하는 것이 가능하다. 변수를 사용하기 위해서는 선언한 후에 사용해야 하는데 변수 선언이라는 것은 변수의 명칭과 타입을 지정하여 메모리상에 데이터를 저장하기 위한 공간을 만드는 과정을 의미한다.

```
데이터타입  변수명;
데이터타입  변수명 = XX;
```

변수를 선언하기 위해서는 다음의 네 가지를 고려해야 한다.

1. 변수에 저장하고자 하는 값의 종류와 크기를 예측한다.
2. 변수의 용도에 따라서 Java 의 식별자 규칙을 적용하여 적당한 명칭을 정한다.
3. 변수를 사용하고자 하는 위치와 변수의 적용 범위(scope)에 따라서 지역변수, 매개변수 그리고 멤버변수 중 하나를 결정한다.
4. 초기화를 언제 어떻게 할 것인지를 결정한다. 초기화란 변수를 만들고 최초로 값을 저장하는 것을 말한다.

```
int number1;          // 정수형 값을 저장하기 위한 변수 선언.(4바이트)
int number2 = 10;     // 정수형 값을 저장하기 위한 변수 선언. 10으로 초기화.(4바이트)
char ch = '가';       // 문자형 값을 저장하기 위한 변수 선언. '가'로 초기화.(2바이트)
double bonus = 10.5;  // 실수형 값을 저장하기 위한 변수 선언. 10.5로 초기화.(8바이트)
String s = "AB";      // 문자열 값을 저장하기 위한 변수 선언. "AB"로 초기화.(8바이트)
float pi = 3.14f;     // 실수형 값을 저장하기 위한 변수 선언. 3.14f로 초기화.(4바이트)
```



[멤버 변수, 매개 변수 그리고 지역 변수]

멤버 변수란 클래스의 멤버로 선언되는 변수로서 클래스에 존재하는 메서드들이 공유하는 변수이다.

매개 변수란 처리하려는 데이터 값을 외부에서 전달받아서 수행하는 메서드의 경우 메서드 호출시 전달되는 데이터를 받는 역할의 변수이다.

지역 변수란 메서드 안에 선언되는 변수로서 메서드 내에서만 사용되는 변수이다. 메서드 호출시 메모리 영역을 할당하고 수행이 끝나면 자동으로 메모리 영역이 해제되는 변수이다. 메서드 내에서도 변수가 선언된 위치부터 변수 선언을 포함하고 있는 블록이 끝날 때까지만 유효한 변수이므로 다음과 같이 블록 안에 선언된 변수는 해당 블록을 벗어나면 사용할 수 없게 된다.

```
public int   getBigNumber(int num1, num2) {
    if (num1 > num2) {
        int result = num1;
        System.out.println("TRUE");
 } else   {
        int result = num2;
        System.out.println("FALSE");
}
    return result;
}
```

result 는 이 블록안에서만 사용되는 스코프가 적용된다.

result 는 이 블록안에서만 사용되는 스코프가 적용된다.

■ 상수

상수란 데이터 값을 한번만 저장할 수 있는 메모리상의 공간으로서 변수를 선언하면서 또는 선언한 후에 한 번 초기화하면 더 이상 값을 바꿀 수 없는 변수를 의미한다.

상수를 만드는 방법은 변수를 선언할 때 데이터 타입 앞에 final 이라는 제어자를 사용한다.

```
final 데이터타입   변수명;
final 데이터타입   변수명 = XX;
```

상수라고 해서 선언하면서 초기화해야 하는 것은 아니다. 선언한 후에 필요한 시점에 뎅터 값을 토기화해도 되며 초기화를 한 후에는 값을 바꿀 수 없다.

```
final int number1;          // final 을 지정하여 정수형 데이터를 저장하기 위한 변수 선언
number1 = 100;              // number1 을 100 이라는 값으로 초기화
number1 = 1000;             // 오류 발생!!
                             // 한 번 초기화되면 더 이상 값을 바꿀 수 없다.
final int number2 = 10;     // final 을 지정하여 정수형 데이터를 저장하기 위한 변수 선언
                            // 하면서 10 이라는 값으로 초기화
number2++;                  // 오류 발생!!
                            // 한 번 초기화되면 더 이상 값을 바꿀 수 없다.
```

■ Java 의 식별자 규칙

식별자 규칙이란 명칭을 정하는 규칙으로 변수명, 메서드명, 클래스명 등을 정할 때 적용되는 규칙이다. 다음과 같은 식별자 규칙이 적용된다.

1. 대소문자가 구분되며 길이에 제한이 없다.
2. 예약어를 사용해서는 안된다.
3. 숫자로 시작해서는 안된다.
4. 특수문자는 _ 와 $ 만을 허용한다.

[Java 의 예약어]

다음은 Java 에서 기능이 정해져 있는 예약어 리스트이다. Java 에서는 대소문자를 구별하고 예약어는 모두 소문자이므로 어떠한 예약어든 대문자로 사용하면 예약어로서 인식되지 않는다. 예약어는 식별자로 사용할 수 없다. const 와 goto 예약어는 실제 사용이 불가능한 예약어이며 식별자로도 사용될 수 없다.

abstract	assert	boolean	break	byte	cast	catch
char	class	const	continue	default	do	double
else	extends	false	final	finally	float	for
goto	if	implements	import	instanceof	int	interface
long	native	new	null	package	private	protected
public	return	short	static	super	switch	synchronized
this	throw	throws	transient	true	try	void
volatile	while	enum	volatile	strictfp		

■ 포매팅 출력 : System.out.printf()

System.out.printf() 는 Java SE 5.0 부터 지원하는 API 로서 정수형, 문자형, 실수형, 논리형 그리고 문자열형에 대하여 타입별 데이터 값을 각 타입에 알맞게 포맷팅 출력을 할 수 있는 기능을 지원한다.

소수점 이하의 자리수 지정, 정수형 숫자의 8 진수 및 16 진수 출력, 출력 공간의 너비 지정 등

을 원하는 포맷을 적용한 출력을 간단하게 할 수 있다.

Java SE 5.0 의 이전 버전에서는 변수들의 내용과 문자열 형식의 타이틀을 결합하여 출력하려면 다음과 같이 문자열 결합연산자를 사용해야 했지만

```
String   name = "자바";
int   age = 15;
System.out.println("이름은 "+ name +"이고 나이는 " + age + "입니다.");
```

System.out.printf() 를 사용하면 다음과 같이 구현하는 것이 가능하다.

```
String name = "자바";
int age = 15;
System.out.println("이름은 %s이고 나이는 %d입니다.", name, age);
```

▶ System.out.println() 메서드의 호출 스펙

java.io.PrintStream 에서 제공되는 printf() 메서드의 헤더는 다음과 같다. 첫 번째 인수로는 출력하고자 하는 포맷 내용을 포함한 문자열을 지정한다. 두 번째 이후의 인수로는 0 개 이상을 지정할 수 있으며 인수의 데이터 타입에는 제한이 없으나 첫 번째 인수에 지정된 포맷 문자열에 포함되어 있는 포맷 지정자의 개수와 종류에 알맞게 정의하여야 한다.

```
public PrintStream printf(String format, Object... args)
```

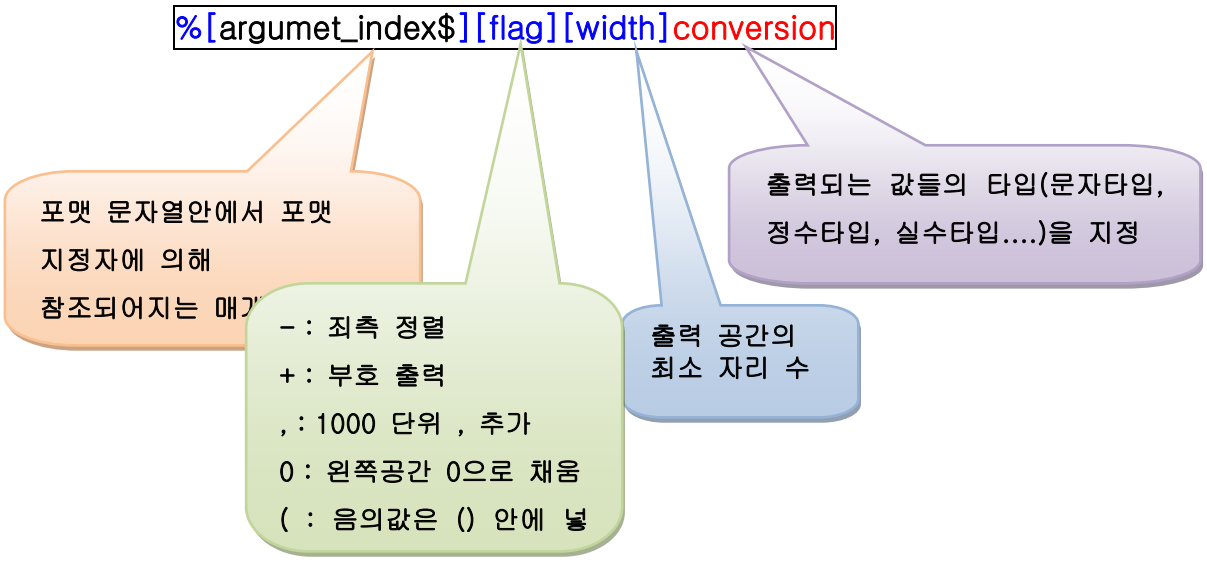
[가변인수(variable argument) : Object...]

가변인수라는 것은 메서드 호출시 전달되는 인수의 개수를 가변적으로 할 수 있다는 것을 뜻하는 구문으로 Java SE 5.0 부터 추가된 구문이다. 메서드 정의 시 매개변수 선언 시에만 사용 가능하며 마지막 매개변수의 타입에만 지정할 수 있다.

```
void pr(int... su)          :  int 형 데이터를 0 개 이상 전달할 수 있다.
void pr(String... s)        :  문자열 데이터를 0 개 이상 전달할 수 있다.
void pr(String s, int... su):  문자열 데이터 한 개와 int 형 데이터를 0 개 이상 전달할 수 있
```

가변인수의 타입으로 어떠한 타입이든 가능하며 실제 수행될 때는 배열로 변환되어 처리된다.

포맷 지정자는 다음과 같은 구조로 구성되며 출력하고자 하는 데이터 값의 종류와 출력 방식에 알맞게 문자형, 정수형, 실수형 등의 값을 나타내는 다양한 변환 문자(conversion)를 사용할 수 있다.



[주요 변환 문자]

%형식		입력	출력	설명
문자	%c	'A'	A(char)	문자형식 출력
정수	%c	65	A(char)	문자형식 출력
	%d	65	65(10진수)	10진수 숫자형식 출력
	%h(H), %x(X)	65	41(16진수)	16진수 숫자형식 출력
	%o	65	101(8진수)	8진수 숫자형식 출력
실수	%f	65.65	65.650000	소수점 이하 6자리까지의 실수 출력
	%e	65.65	6.565000e+01	지수형식 출력
	%g	65.65	65.6500	정수부와 소수부의 길이가 모두 6개 되는 실수 출력

```
System.out.printf("1  : %d  %d  %f  %f  %f  %c  %b %n", a, b, c, d, e, f, g);
System.out.printf("2  : %20d %n", a);
System.out.printf("3  : %-20d %n", a);
System.out.printf("4  : %020d %n", b);
System.out.printf("5  : %+20d  %+20d %n", b, -b);
System.out.printf("6  : %,20d %n", b);
System.out.printf("7  : %(20d %(20d %n", b, -b);
System.out.printf("8  :  10진수-%1$d  8진수-%1$o  16진수-%1$x,%1$X  %n",
a);
System.out.printf("9  : %f  %1$e  %1$g %n", e);
System.out.printf("10 : %.2f  %1$g %n", e);
```