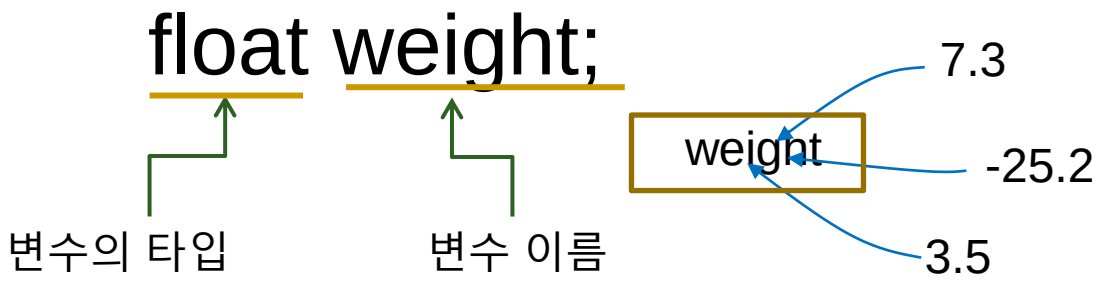
제 3 장
변수와 기본데이터 타입

변수와 타입

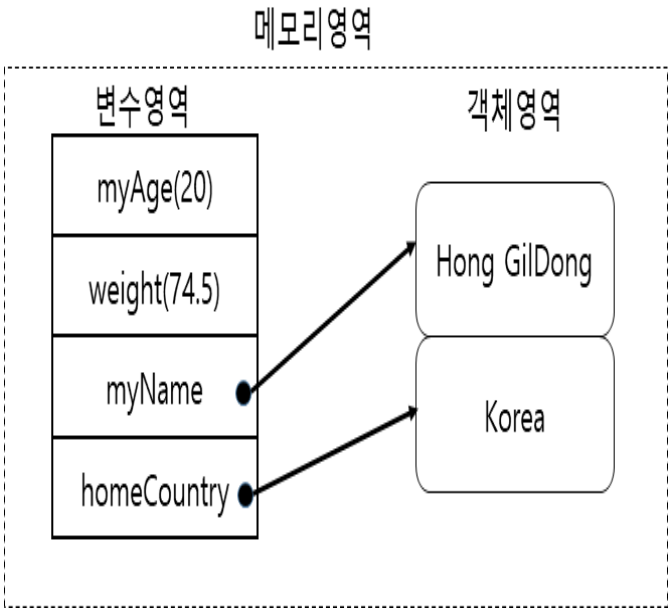
- 변수
  - 프로그램이 실행 중에 값을 임시 저장하기 위한 공간
    - 변수 값은 프로그램 수행 중 변경될 수 있다.
  - 변수의 타입 : 저장되는 데이터의 유형
  - 데이터 타입에 맞는 크기의 메모리 할당
  - 반드시 변수 선언 후 사용
  - 변수 선언
    - 변수의 타입 다음에 변수 이름을 적어 변수를 선언



데이터 타입의 분류

- 기본 데이터 타입(primitive data type)
  - 선언된 타입의 데이터가 변수에 직접 저장되는 유형
  - 선언된 변수 자체가 저장된 데이터 값을 의미
  - C 언어의 변수 타입과 유사
  - 자바에서는 8가지 종류의 타입이 존재
- 참조 타입 혹은 객체 타입(reference type 혹은 object type)
  - 데이터가 저장된 위치를 가리키는 주소가 저장되는 데이터 타입
  - 보통 객체를 가리키다 – 배열, 스트링, 클래스, 인터페이스 등
  - 타입의 수가 정해져 있지 않다 – 존재하는 객체 타입의 수만큼 존재

```
Int myAge = 20;
Double weight = 74.5;
String myName = “Hong GilDong”;
String homeCountry = “Korea”;
```



- 변수 선언 사례

```
int radius;
char c1, c2, c3; // 3 개의 변수를 한 번에 선언한다.
double weight;
```

- 변수 선언과 초기화
  - 선언과 동시에 초기값 지정

```
int radius = 10;
char c1 = ‘a', c2 = 'b', c3 = 'c';
double weight = 75.56;
```

- 변수에 값 대입
  - 대입 연산자인 ′=′ 사용

```
radius = 10 * 5;
c1 = ’r’;
weight = weight + 5.0;
```

- 기본 타입 : 8 개
  - boolean
  - char
  - byte
  - short
  - int
  - long
  - float
  - double
- 참조 타입 : 3 개
  - 클래스(class)에 대한 레퍼런스
  - 인터페이스(interface)에 대한 레퍼런스
  - 배열(array)에 대한 레퍼런스

- 변수의 이름 혹은 식별자(identifier)
  - 클래스, 변수, 상수, 메소드 등에 붙이는 이름
  - 식별자의 원칙
    - '@', '#', '!'와 같은 특수 문자, 공백 또는 탭은 식별자로 사용할 수 없으나 '_', '$'는 사용 가능
    - 유니코드 문자 사용 가능. 한글 사용 가능
    - 자바 언어의 **키워드는 식별자로 사용할 수 없음**
    - 식별자의 첫 번째 문자로 숫자는 사용할 수 없음
    - '_' 또는 '$'를 식별자 첫 번째 문자로 사용할 수 있으나 일반적으로 잘 사용하지 않는다.
    - Boolean 리터럴 (**true, false**)와 NULL 리터럴(**null**)은 식별자로 사용 불허
    - 길이 제한 없음
  - 대소문자 구별
    - Test와 test는 별개의 식별자

자바 키워드(예약어)

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

- 사용 가능한 예

```
int name;
charstudent_ID;                          // '_' 사용 가능
void$func() { }                          // '$' 사용 가능
classMonster3 { }                        // 숫자 사용 가능
int whatsyournamemynameiskitae;          // 길이 제한 없음
int barChart; int barchart;              // 대소문자 구분
int 가격;                                // 한글 이름 사용 가능
```

- 잘못사용된 예

```
int 3Chapter;          // 숫자로 사용하였기 때문
class if { }           // if는 자바의 예약어임
char false;            // false는 사용 불가
void null() { }        // null 사용 불가
class %calc { }        // '%'는 특수문자
```

식별자 이름 붙이는 관습

- 헝가리안 표기법(Hungarian Notation) 관습 – camelCase, PascalCase
  - 클래스 이름

```
public class HelloWorld {}
class Vehicle {}
class AutoVendingMachine {}
```

    - 첫 번째 문자는 대문자로 시작
    - 여러 단어가 복합되어 있을 때는 각 단어의 첫 번째 문자만 대문자로 표시
  - 변수, 메소드 이름

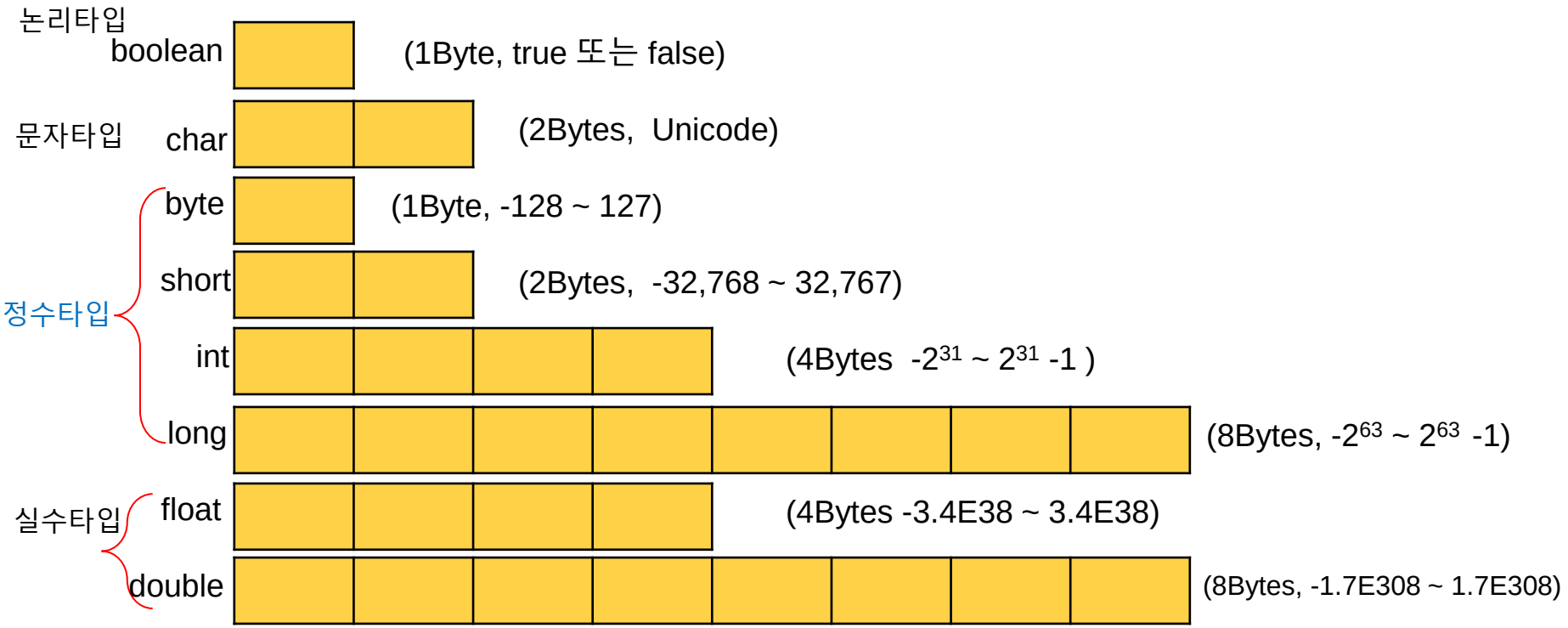
```
int iAge;                   // iAge의 i는 int의 i를 표시
boolean bIsSingle;          // bIsSingle의 처음 b는 boolean의 b를 표시
String strName;             //strName의 str은 String의 str을 표시
public int iGetAge() {}     //iGetAge의 i는 int의 i를 표시
```

    - 첫 단어 이후 각 단어의 첫 번째 문자는 대문자로 시작
  - 상수 이름
    - 모든 문자를 대문자로 표시

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

기본 데이터 타입

- 특징
  - 기본 데이터 타입의 크기가 정해져 있음(상세설명은 p.48 참조)
  - 기본 데이터 타입의 크기는 CPU나 운영체제에 따라 변하지 않음



기본 데이터 타입		특징
정수 타입	byte	● 2의 보수로써 표현하는 8-비트(1-byte) 길이의 정수 타입 ● 표현 가능한 정수 범위 : –128(-2^7) ~ 127(2^7-1) ● 표현 범위가 제한되는 작은 크기의 정수로 메모리의 절약을 요구하는 다량의 배열을 구성하는 경우 유용
	short	● 2의 보수로써 표현하는 16-비트(2-byte) 길이의 정수 타입 ● 표현 가능한 정수 범위 : –32768(-2^{15}) ~ 32767(2^{15}-1) ● 용도는 byte 타입과 유사
	int	● 2의 보수로써 표현하는 32-비트(4-byte) 길이의 정수 타입 ● 표현 가능한 정수 범위 : -2^{31} ~ 2^{31}-1 ● 자바 8.0부터 0부터 2^{32}-1까지의 음이 아닌 정수 표현 가능 - C처럼 unsigned라는 타입을 사용하지는 않으며, - int의 참조 타입인 Integer 클래스를 사용
	long	● 2의 보수로써 표현하는 64-비트(8-byte) 길이의 정수 타입 ● 표현 가능한 범위 : -2^{63} ~ 2^{63}-1 ● 자바 8.0부터 0부터 2^{64}-1까지의 음이 아닌 정수 표현 가능 - unsigned라는 타입을 사용하지는 않으며, - long의 참조 타입인 Long 클래스를 사용
실수 타입	float	● 단정밀도(single-precision) 32-비트(4-byte) 길이의 부동소수점 형식(IEEE 754) 사용 ● 실수 표현의 범위 : 부동소수점 형식(IEEE 754)에 의해 결정되며, 보통 long 타입보다 넓다
	double	● 배정밀도(double-precision) 64-비트(8-byte) 길이 부동소수점 형식(IEEE 754) 사용 ● 실수 표현의 범위 : IEEE 754 형식에 의해 결정되며, 보통 float 타입보다 넓다 ● float보다 더 정밀한 수의 표현과 연산에서 사용
논리 타입	boolean	● 1-byte 길이를 가지며 2개의 값, true 혹은 false만 가진다 - C처럼 1과 0이 아니라 true, false 자체가 값이다 ● 보통 참과 거짓을 나타내는 간단한 플래그 용도로 사용
문자 타입	char	● 하나의 16-비트 Unicode 문자를 나타낸다 (2-byte) ● 최소값은 '₩u0000'(혹은 0) 이며 최대값은 '₩uffff'(혹은 65535)

타입의 변환(type casting)

- 어떤 데이터 타입을 다른 타입으로 변환하는 과정
- 왜 필요한가?
  - 대입연산자 좌변과 우변의 타입은 일치하여야 한다
    - 일치하지 않는 경우 타입 변환이 일어난다
  - 우측 값의 크기는 좌측 변수 타입이 표현할 수 있는 범위내에 있어야 한다
- boolean 타입
  - true 혹은 false의 2가지 값만 저장
  - 다른 타입으로 변환이 불가능
- 자동 타입변환(implicit type casting)
- 강제 타입변환(explicit type casting)

자동 타입 변환

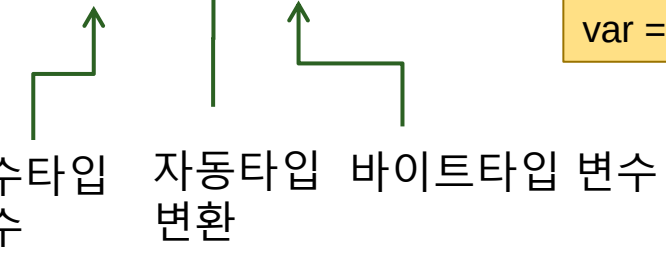
- 표현범위가 좁은 타입으로부터 넓은 타입으로의 변환
  - 대입연산자 좌측 변수 타입의 표현 범위가 우측 보다 넓은 경우
  - 컴파일러에 의해 자동으로 변환

byte → short → int → long → float → double

  - 원본 데이터 그대로 보존

- 자동 타입 변환 사례

```
byte   a;
int    price;
price  =  a;
```

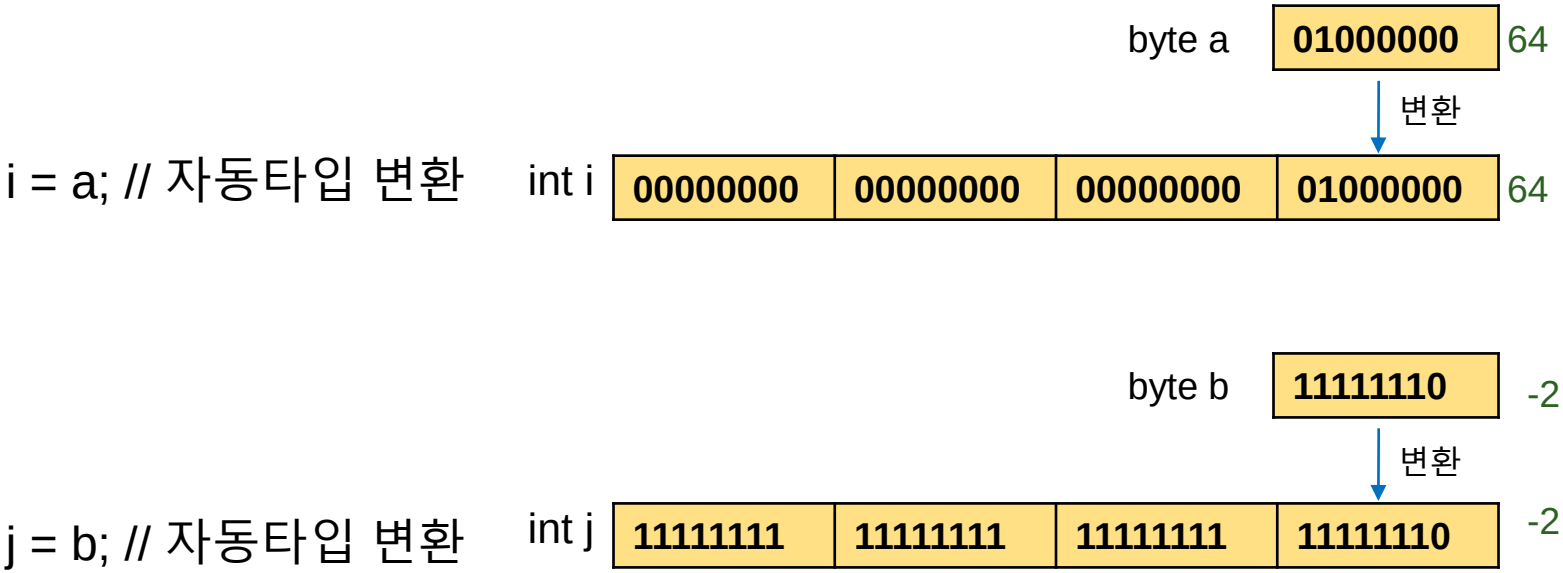



```
long  var;
int   n = 32555;
byte  b = 25;
var = n;     // int 타입에서 long 타입으로 자동 변환. var 값은 32555
var = b;     // byte 타입에서 long 타입으로 자동 변환. var 값은 25
```

byte 타입으로부터 int 타입으로의 자동 변환 예

```
int i, j;
byte a = 64;
byte b = -2;

i = a;      // 자동타입 변환
j = b;      // 자동타입 변환
```



강제 타입 변환

- 표현범위가 넓은 타입으로부터 좁은 타입으로의 변환
  - 대입연산자 좌측 변수 타입의 표현 범위가 우측 보다 좁은 경우
  - 컴파일러에 의해 자동으로 변환되지 않으며, 명시적 지정이 필요
  - 정보 손실이 발생하므로 유의하여야 한다

- 강제 타입 변환 방법

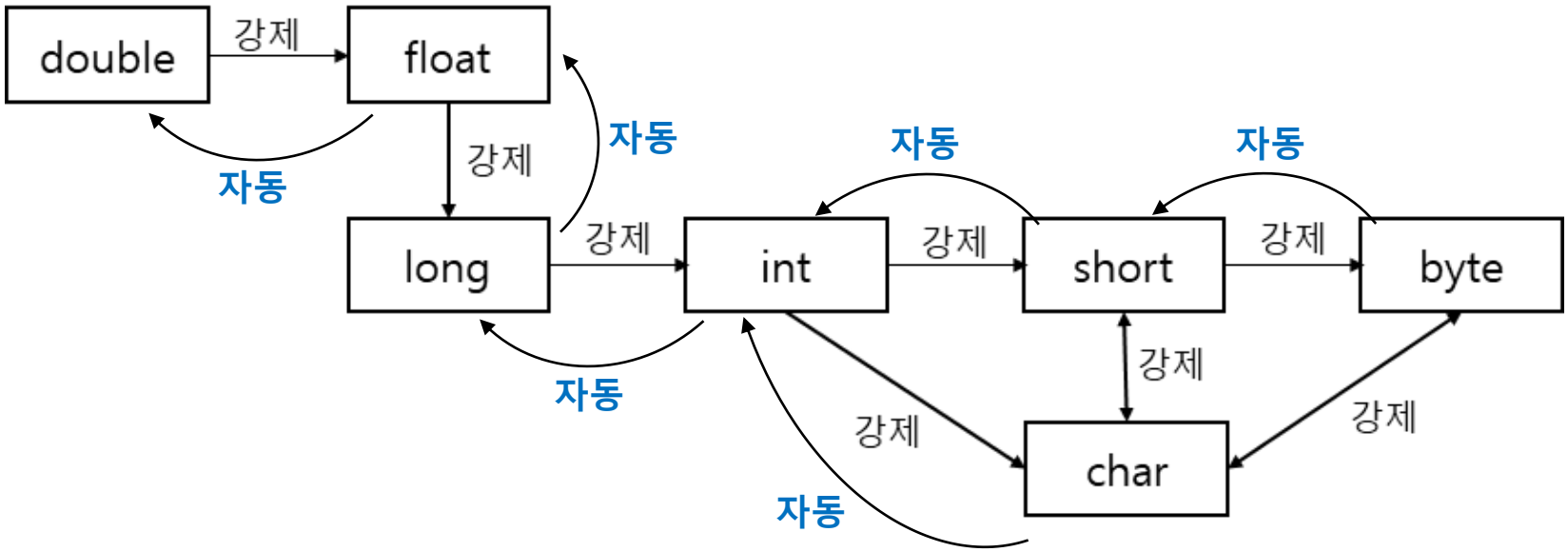
```
byte  a;
int   price;
a  =  (byte)  price;
```

바이트타입 변수 — price 정수 값을 byte 타입으로 강제 타입 변환 — 정수타입 변수

  - 강제 타입 변환 사례
    - 실수타입이 정수타입으로 강제 변환 시 소수점 아래가 버려짐(데이터의손실)

```
short  var;
int  n = 855638017;      // n의 16진수 값은 0x33000001
var = (short) n;         // int 타입에서 short 타입으로 강제 변환. Var 값은 1
double d = 1.9;
int n = (int) d;         // n은 1이 된다 - 정보손실
```

각 타입 간의 변환 모형



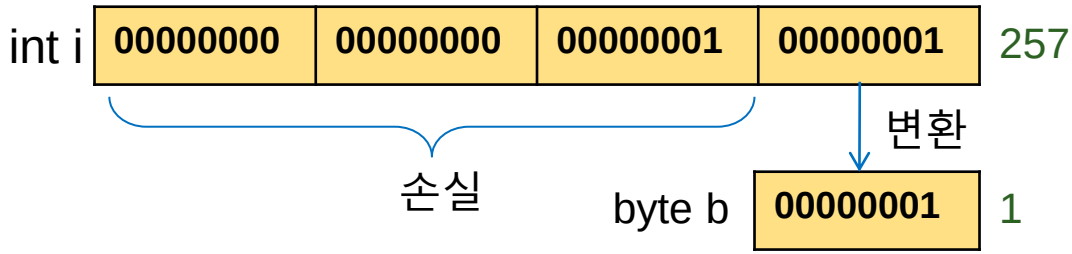
[QUIZ] byte 타입과 char 타입, 그리고 short 타입과 char 타입 간에는 왜 강제 변환만 가능한가?

int → byte, double→ int 로 강제 변환

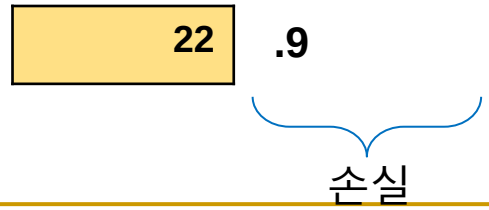
```
int i = 257;
byte b;
b = (byte) i; // 강제타입 변환
i = (int) 22.9; // 강제타입 변환
```

double 타입을 강제로 int 타입으로 변환
22.9에서 .9가 손실되어 22만 i 에 저장된다.

b = (byte)i; // 강제타입변환



i = (int)22.9; // 강제타입 변환



리터럴(literal)

- 고정된 값 자체를 소스코드에 표현한 것
- 정수형, 실수형

리터럴 타입	비고
정수형 리터럴	· 리터럴이 문자 'L' 이나 'l' 로 끝나면 long 타입의 리터럴이며, 그렇지 않으면 int 타입으로 간주한다. · 정수 타입의 리터럴은 다음과 같은 형식으로 표현될 수 있다. - 10진수 : 언급 없이 사용하는 모든 정수는 10진수로 간주 예) int decVal = 26; - 16진수(Hexadecimal) : 수 앞에 "0x" 접두어를 붙이면 16진수로, 사용 가능한 숫자는 0 – 9 사이의 숫자와 A – F 까지 영문자 예) int hexVal = 0x1a; - 2진수(자바 7.0부터 사용) : 수 앞에 "0b" 접두어를 붙인다. 사용 가능한 숫자는 0과 1 예) int binVal = 0b11010;
(부동소수점)실수형 리터럴	· 마지막 문자가 'F' 혹은 'f' 로 끝나면 float 타입의 부동소숫점 리터럴 (32-비트 부동소숫점 타입) 예) float f1 = 123.4f; · 언급이 없으면 double 타입으로 간주 (64-비트 부동소숫점 타입) - 리터럴이 문자 'D' 혹은 'd' 로 끝나도 double 타입이다 예) double d1 = 123.4; double d2 = 123.4d; · 부동소숫점 타입은 'E' 혹은 'e' 문자를 사용하여 표현하기도 한다. 예) double d2 = 1.234e2;

- 논리형
  - true 혹은 false 2가지만 존재
  - 논리타입은 어떤 타입과도 변환 허용 안 됨

```
int i;
if ((boolean)i) {}        // 컴파일 에러
```

- 문자형
  - 단일 인용부호('')로 문자 하나 표현
    - 'a', 'W', '가', '*', '3', '7'
  - \ 가 붙은 문자 리터럴 : escape 문자라고 하며 특수 용도로 사용
    - '\b' (backspace), '\t' (tab), '\n' (line feed), '\f' (form feed), '\r' (carriage return),
    - '\"' (double quote), '\'' (single quote), '\\' (backslash)
  - \u다음에 4자리 16진수 → 2 바이트의 유니코드(Unicode)
    - ₩u0041 -> 문자 'A'의 유니코드(0041)
    - ₩uae00 -> 한글문자 '글'의 유니코드(ae00)

- 문자열 리터럴
  - 이중 인용부호로 묶어서 표현
    - "Good", "Morning", "자바", "3.19", "26", "a"
  - 자바에서 **문자열은 객체이므로 기본 타입이 아님**
  - 문자열 리터럴은 String 타입의 객체로 생성됨
- null 리터럴
  - 어떠한 참조 타입의 값으로도 사용 가능. 그러나 기본 타입에는 사용 불가
    - ~~int n = null~~; // 기본 데이터 타입에는 사용 불가
    - String str = null;
- underscore('_')를 사용하는 정수형 혹은 실수형 리터럴
  - 여러 개의 '_' 문자를 정수 혹은 실수형 리터럴의 어느 위치에나 삽입
  - 수를 나타내는 리터럴의 가독성 개선을 위해 숫자들을 그룹으로 분리하는 용도로 사용할 수 있다.

올바른 사용예

```
long cardNumber = 1234_5678_9012_3456L;
long studentId = 999_99_9999L;
float pi = 3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
int bytes = 0b11010010_01101001_10010100_10010010
```

잘못된 사용예

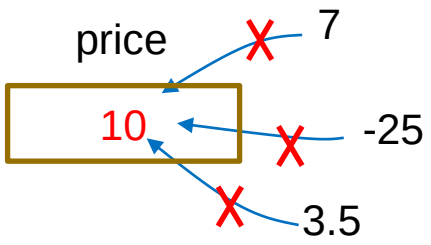
```
float pi1 = 3_.1415F;
float pi2 = 3._1415F;
long socialSecurityNumber1 = 999_99_9999_L;
int x2 = 52_;
int x4 = 0_x52;
int x5 = 0x_52;
```

상수

- 상수 선언
  - **final** 키워드 사용
  - 값 변경 불가
  - 선언 시 반드시 초기값 지정

```
final  int  PRICE = 10;
```

상수 선언 데이터 타입 상수 이름 초기화



- 상수 선언 사례

```
final double PI = 3.141592;
final int LENGTH = 20;
```

자바 프로그램 구조 맛보기

```
/*
*  맛보기 예제.
*  소스 파일 : Hello2.java
*/
public class Hello2 {

   public static int sum(int n, int m) {
      return n + m;
   }

   // main() 메소드에서 실행 시작
   public static void main(String[] args) {
      int i = 20;
      int s;
      char a;

      s = sum(i, 10); // sum() 메소드 호출
      a = '?';
      System.out.println(a); // 문자 '?' 화면 출력
      System.out.println("Hello2"); // "Hello2" 문자열 화면 출력
      System.out.println(s); // 정수 s 값 화면 출력
   }
}
```

```
?
Hello2
30
```

클래스

메소드

메소드

- 클래스 만들기
  - Hello2 라는 이름의 클래스 선언

```
public class Hello2 {
 }
```

  - class 라는 키워드로 클래스 정의
  - public으로 선언하면 다른 클래스에서도 접근 가능
  - 클래스 본문은 '{'으로 시작하여 '}'으로 끝남
- main() 메소드
  - **public static void**으로 선언되어야 함

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

  - 자바 프로그램은 main() 메소드부터 실행 시작
  - String[] args로 실행 인자를 전달 받음(3장 참고)
- 멤버 메소드
  - 메소드 sum() 정의

```
public static int sum(int n, int m)
{
...
}
```

  - 클래스 에 속한 함수, 클래스 내에서만 선언
  - 인자들의 타입과 변수 명을 ','로 분리하여 나열
  - 메소드 코드는 '{'과 '}' 사이에 작성
- 변수 선언
  - 개발자가 변수 이름을 붙이고 같이 선언

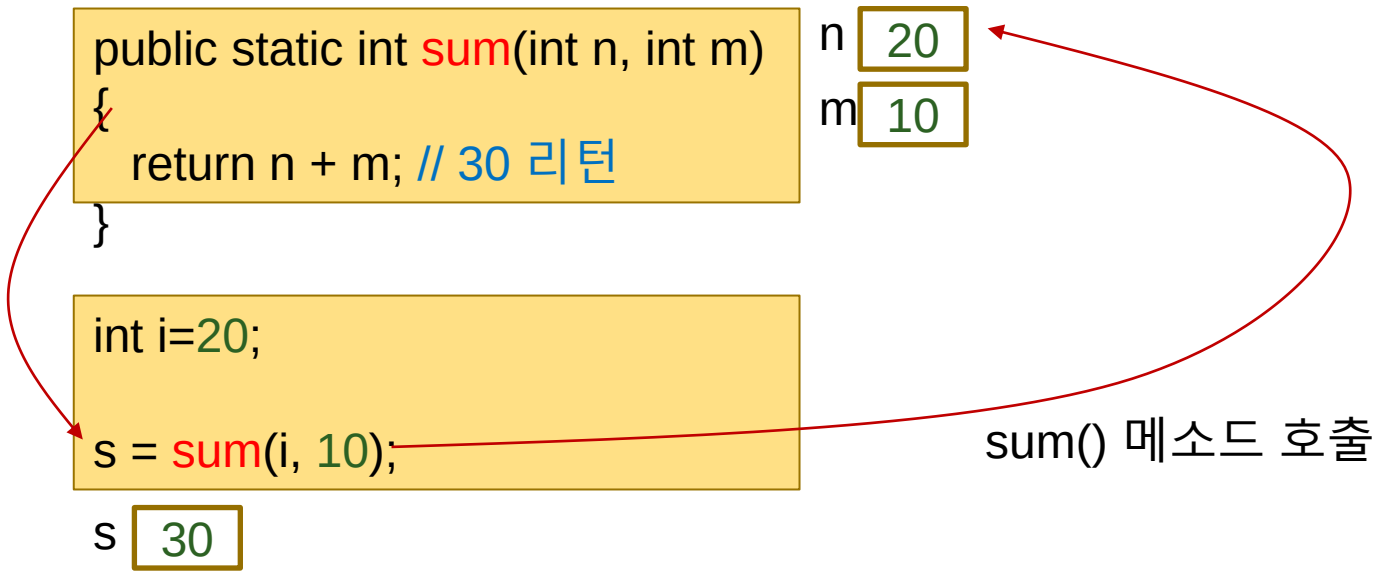
```
int i=20;
int s;
char a;
```

  - 메소드 내에서 선언된 변수가 지역 변수
  - 지역 변수는 메소드 실행이 끝나면 저장 공간 반환
- 메소드 호출
  - sum() 메소드 호출

```
s = sum(1,10); // 메소드 sum() 호출
```

  - sum() 메소드의 호출 시 변수 i의 값과 정수 10을 전달
  - sum() 메소드의 인자인 n, m은 각각 20, 10의 값을 전달 받음
  - sum() 메소드는 n과 m 값을 더한 30을 리턴
  - 호출한 부분에서 변수 s는 정수 30을 전달받아 저장

sum() 메소드 호출과 리턴



- 주석문
  - 실행에 영향을 주지 않음.
  - “//”을 만나면 행 끝날 때가지 한 라인을 주석문 처리

```
// main() 메소드에서 실행 시작
s=sum(1,10); // 메소드 호출
```

  - “/*”을 만나면 “*/”을 만날 때까지 여러 행을 주석문 처리

```
/*
* 맛보기 예제
* 소스 파일 : Hello2.java
*/
```

- 화면 출력
  - 표준 출력 스트림에 메시지 출력

```
System.out.println(a); // 문자 ? 화면 출력
System.out.println("Hello2"); // "Hello2" 문자열 화면 출력
System.out.println(s); // 정수 s 값 화면 출력
```

  - **표준 출력 스트림 System.out**의 println 메소드 호출
  - println은 여러가지 데이터 타입 출력
  - println은 주어진 인자를 출력 후 다음 행으로 커서 이동

- 문장
  - ;로 한 문장의 끝을 인식

```
int i=20;
b = ’?’;
s = sum(i, 20);
```

  - 한 문장을 여러 줄에 작성해도 무방

```
b
= ’?’;
```

  - 주석문 끝에는 ‘;’를 붙이지 않음
- 블록
  - 블록은 {으로 시작하여 }으로 끝남

```
public class Hello2 {
....
} // Hello2 클래스 선언문 끝

public static void main(String[] args) {
...
} // 메소드 main() 선언문 끝
```

  - 클래스 선언과 메소드 선언 등은 블록으로 구성

변수, 리터럴, 상수 사용하기

원의 면적을 구하는 프로그램을 작성해보자.

```
public class CircleArea {
   public static void main(String[] args) {
      final double PI = 3.14; // 원주율을 상수로 선언
      double radius = 10; // 원의 반지름
      double circleArea = 0; // 원의 면적
      circleArea = radius*radius*PI; // 원의 면적 계산
      // 원의 면적을 화면에 출력한다.
      System.out.print("원의 면적 = ");
      System.out.println(circleArea);
   }
}
```

```
원의 면적 = 314.0
```

자동 타입 변환, 강제 타입 변환

자동 타입 변환과 강제 타입 변환의 이해를 위한 예제이다.
다음 소스의 실행 결과는 무엇인가?

```
public class TypeConversion {
   public static void main(String[] args) {
      byte b = 127;
      int i = 100;
      System.out.println(b+i);
      System.out.println(10/4);
      System.out.println(10.0/4);
      System.out.println((char)0x12340041);
      System.out.println((byte)(b+i));
      System.out.println((int)2.9 + 1.8);
      System.out.println((int)(2.9 + 1.8));
      System.out.println((int)2.9 + (int)1.8);
   }
}
```

```
227
2
2.5
A
-29
3.8
4
3
```

자바에서 키 입력, System.in

- **System.in**
  - 자바의 표준 입력 스트림
  - java.io의 InputStream 클래스 타입
  - InputStream이 바이트 스트림이므로 문자 스트림으로 변환하려면 InputStreamReader 클래스를 이용
  - 입력 동안 IOException이 발생가능, 예외 처리 필요

표준 입력 스트림 기반의 키입력

다음 소스의 실행 결과는 무엇인가?
System.in을 InputStreamReader에 연결하여 사용자로부터 키를 입력받는다. 입력받은 문자를 화면에 출력하고 사용자가 ctrl-z를 누르면 읽기가 종료된다.

```
import java.io.*;
public class InputExample {
   public static void main (String args[]) {
      InputStreamReader rd = new InputStreamReader(System.in);
      try {
         while (true) {
            int a = rd.read();
            if (a == -1)  // ctrl-z가 입력되면 read()는 -1을 리턴
               break;
            System.out.println((char)a);// 입력된 문자 출력
         }
      }
      catch (IOException e) {
         System.out.println("입력 에러 발생");
      }
   }
}
```

```
자바 실습
자
바

실
습
```

키 입력부분

Scanner를 이용한 키 입력

- Scanner 클래스
  - java.util.Scanner 클래스
  - Scanner 객체 생성

```
Scanner a = new Scanner(System.in);
```

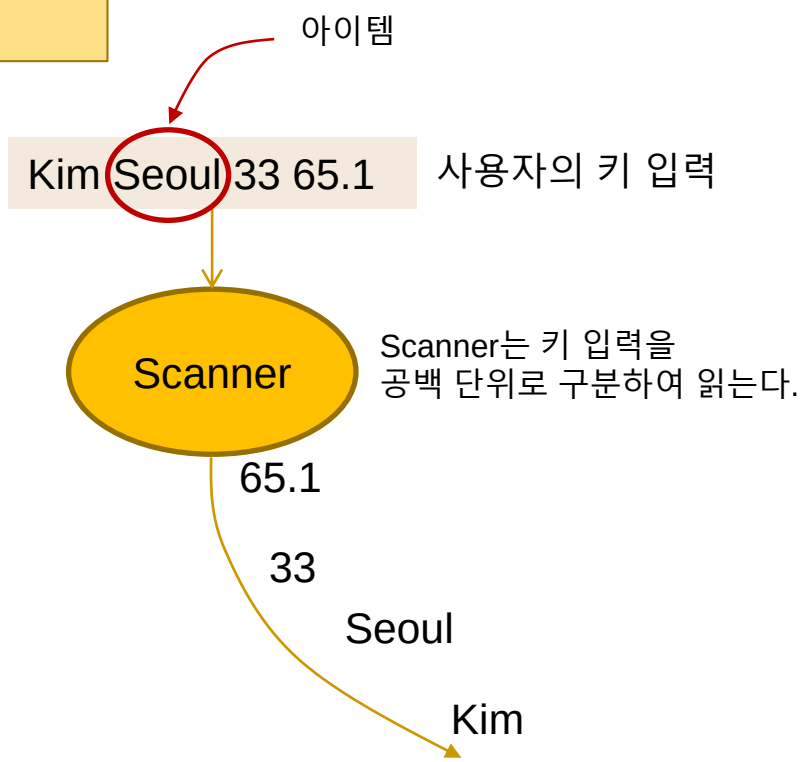
  - import문 필요
    - 소스 맨 앞줄에 사용

```
import java.util.Scanner;
```

  - Scanner에서 키 입력 받기
    - Scanner는 입력되는 키 값을 공백 (‘\t’,‘ \f’,‘ \r’,‘ ’,‘ \n’)으로 구분되는 아이템 단위로 읽음

```
Scanner scanner = new Scanner(System.in);

String name = scanner.next();                // "Kim"
String addr = scanner.next();                // "Seoul"
int age = scanner.nextInt();                 // 23
double weight = scanner.nextDouble();        // 65.1
```



Scanner 주요 메소드

생성자/메소드	설명
String next()	다음 아이템을 찾아 문자열로 반환
boolean nextBoolean()	다음 아이템을 찾아 boolean으로 변환하여 반환
byte nextByte()	다음 아이템을 찾아 byte로 변환하여 반환
double nextDouble()	다음 아이템을 찾아 double로 변환하여 반환
float nextFloat()	다음 아이템을 찾아 float로 변환하여 반환
int nextInt()	다음 아이템을 찾아 int로 변환하여 반환
long nextLong()	다음 아이템을 찾아 long으로 변환하여 반환
short nextShort()	다음 아이템을 찾아 short로 변환하여 반환
String nextLine()	한 라인 전체('\n' 포함)를 문자열 타입으로 반환

Scanner를 이용한 키 입력 연습

Scanner를 이용하여 나이, 체중, 신장 데이터를 키보드에서 입력 받아 다시 출력하는 프로그램을 작성해보자.

```
import java.util.Scanner;
public class ScannerExam {
   public static void main (String args[]) {
      Scanner a = new Scanner(System.in);
      System.out.println("나이, 체중, 신장을 빈칸으로 분리하여 순서대로 입력하세요");
      System.out.println("당신의 나이는 " + a.nextInt() + "살입니다.");
      System.out.println("당신의 체중은 " + a.nextDouble() +"kg입니다.");
      System.out.println("당신의 신장은 " + a.nextDouble()+ "cm입니다.");
   }
}
```

키 입력부분

```
나이, 체중, 신장을 빈칸으로 분리하여 순서대로 입력하세요
35 75 175
당신의 나이는 35살입니다.
당신의 체중은 75.0kg입니다.
당신의 신장은 175.0cm입니다.
```