

JAVA

- 변수

기본 자료형

System.out의 기본(1)

- 화면에 내용을 출력
 - `System.out.println()` : 괄호 안의 내용을 출력한 후 한 행을 띄움
 - `System.out.print()` : 괄호 안의 내용을 출력한 후 한 행을 띄지 않고 유지

System.out의 기본(2)

- System.out.printf() 메소드의 기본적인 사용법

```
System.out.println("안녕하세요?");  
System.out.println("Java입니다.");
```

실행 결과 ▶

안녕하세요?
Java입니다.

```
System.out.print("안녕하세요?");  
System.out.print("Java입니다.");
```

실행 결과 ▶

안녕하세요?Java입니다.

서식 문자	설명	비고
Wn	새로운 줄로 이동	<input type="button" value="Enter"/> 키를 누른 효과
Wt	다음 탭으로 이동	<input type="button" value="Tab"/> 키를 누른 효과
Wb	뒤로 한 칸 이동	<input type="button" value="Back Space"/> 키를 누른 효과
Wr	줄의 맨 앞으로 이동	<input type="button" value="Home"/> 키를 누른 효과
WWW	\ 출력	
W'	' 출력	
W"	" 출력	

->

예제

```
package printtest;

public class printTest {

    public static void main(String[] args) {

        System.out.print("\n줄바꿈\n연습\n");
        System.out.print("\t탭키\t연습\n");
        System.out.print("이것은\r 앞으로 이동합니다.\n");
        System.out.print("글자가 \"강조\" 됩니다.\n");
        System.out.print("\\\\\\\\역슬래시 세개 출력");

    }

}
```

비트(1)

- 비트

- 0(OFF)과 1(ON)만 존재

전기 스위치		의미	2진수	10진수
		꺼짐, 꺼짐	00	0
		꺼짐, 켜짐	01	1
		켜짐, 꺼짐	10	2
		켜짐, 켜짐	11	3

- n개의 전기 스위치로 표현할 수 있는 가짓 수 = 2^n
- 3비트로 표현할 수 있는 가짓수는 $2^3=8$ 개, 4비트로 표현할 수 있는 가짓수는 $2^4=16$ 개

비트(2)

- 진수

10진수(0~9)	2진수(0, 1)	16진수(0~F)
00	0000	0
01	0001	1
02	0010	2
03	0011	3
04	0100	4
05	0101	5
06	0110	6
07	0111	7
08	1000	8
09	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

비트(3)

- **바이트**

- JAVA에서 가장 많이 사용되는 단위. 바이트는 8개의 비트가 합쳐진 것
0 or 1

비트 수	바이트 수	표현 개수	2진수	10진수	16진수
1		$2^1=2$	0~1	0~1	0~1
2		$2^2=4$	0~11	0~3	0~3
4		$2^4=16$	0~1111	0~15	0~F
8	1	$2^8=256$	0~11111111	0~255	0~FF
16	2	$2^{16}=65536$	0~11111111 11111111	0~65535	0~FFFF
32	4 int	$2^{32}=\text{약 } 42\text{억}$	0~...	0~약 42억	0~FFFF FFFF
64	8 long	$2^{64}=\text{약 } 1800\text{경}$	0~...	0~약 1800경	0~...

-21 ~ + 21

비트(4)

– 2진수를 10진수로 변환하는 방법


2진수	1	0	0	1		0	0	1	1
	×	×	×	×		×	×	×	×
	2^7	2^6	2^5	2^4		2^3	2^2	2^1	2^0
	128	0	0	16		0	0	2	1
	<div style="border-top: 1px solid black; border-left: 1px solid black; border-right: 1px solid black; height: 20px; position: relative; margin-top: 10px;"> + </div>								
10진수	147								

비트(5)


– 2진수를 16진수로 변환한 후 10진수로 변환하는 방법

2진수

1	0	0	1			
×	×	×	×			
2^3	2^2	2^1	2^0			
8	+	0	+	0	+	1



0	0	1	1			
×	×	×	×			
2^3	2^2	2^1	2^0			
0	+	0	+	2	+	1



16진수

9
×
16^1
144

3
×
16^0
3

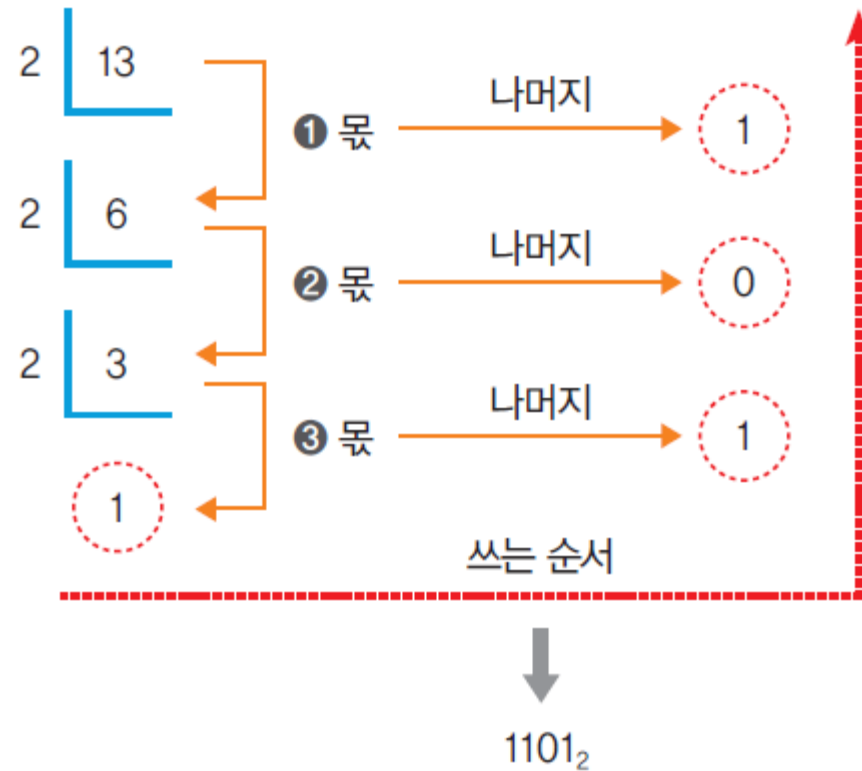
+

10진수

147

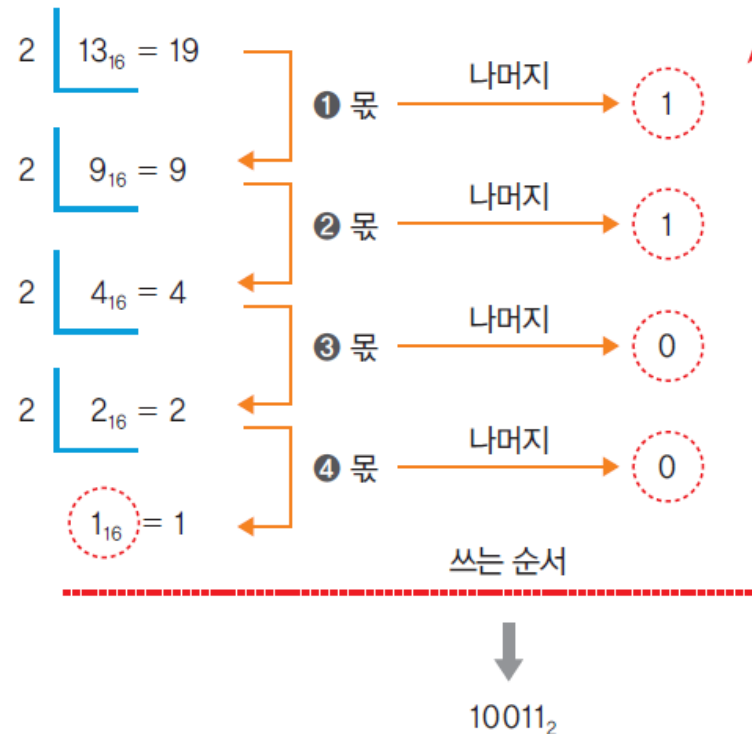
비트(6)

- 2진수 변환 연습
 - 10진수를 2진수로 변환



비트(7)

- 16진수를 2진수로 변환. 16진수라는 것을 나타내기 위해 13_{16} 과 같이 표현함



- JAVA에서 16진수를 표현할 때는 숫자 앞에 '0x' 또는 '0X'를 붙이면 된다. 예를 들어 'a=10'은 a에 10진수 10을 대입하라는 것이지만, 'a=0x10'은 16진수 10(일영이라 읽으며 10진수로는 16이다)을 대입하라는 의미

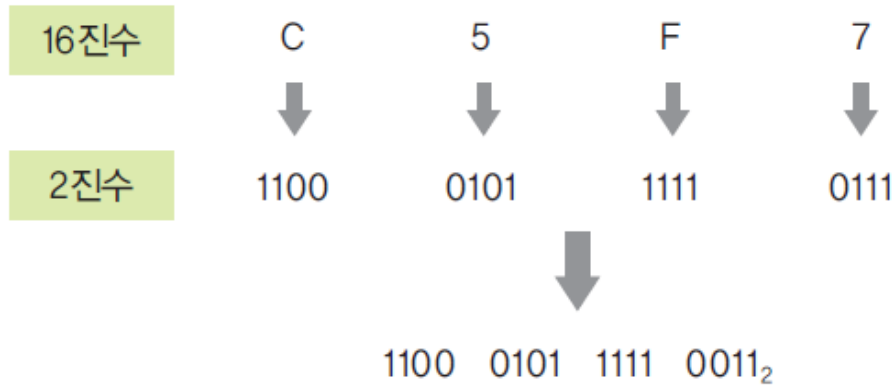
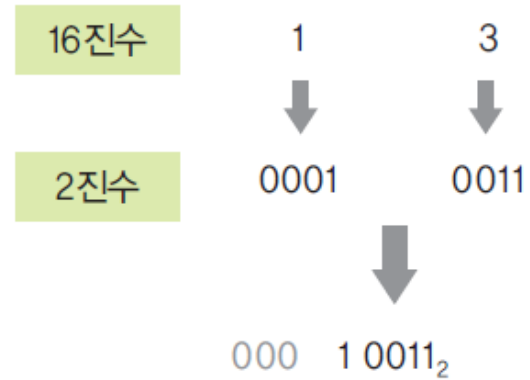
비트(8)

- 16진수와 2진수 변환표

16진수	2진수	16진수	2진수
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

비트(9)

- 16진수를 2진수로 변환하는 간편한 방법



변수

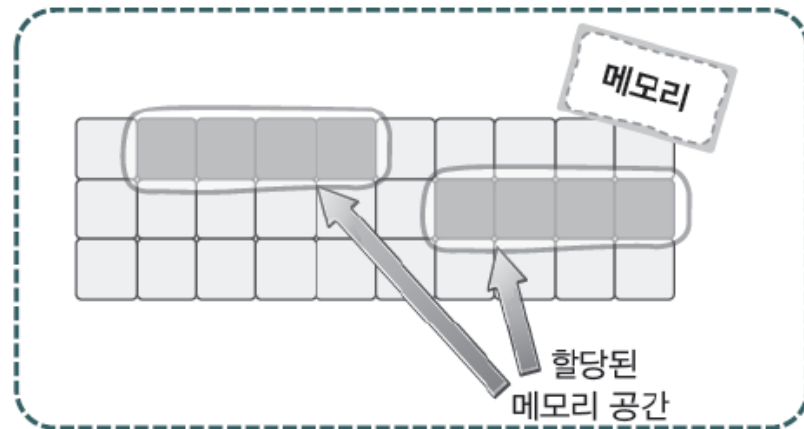
메모리 공간의 활용과 변수와의 관계

- 메모리 공간의 활용에 필요한 두 가지 요소

1. 데이터 저장을 위한 메모리 공간의 할당 방법 =>
2. 할당된 메모리 공간의 접근(저장 및 참조) 방법 => 가 ?



'변수(Variable)'라는 것을 통해 이 두 가지 방법을 모두 제공



변수(Variable)에 대한 간단한 이해

“난 정수의 숫자(데이터) 저장을 위한
메모리 공간을 할당 하겠다”

“그리고 그 메모리 공간의
이름을 num이라 하겠다.”

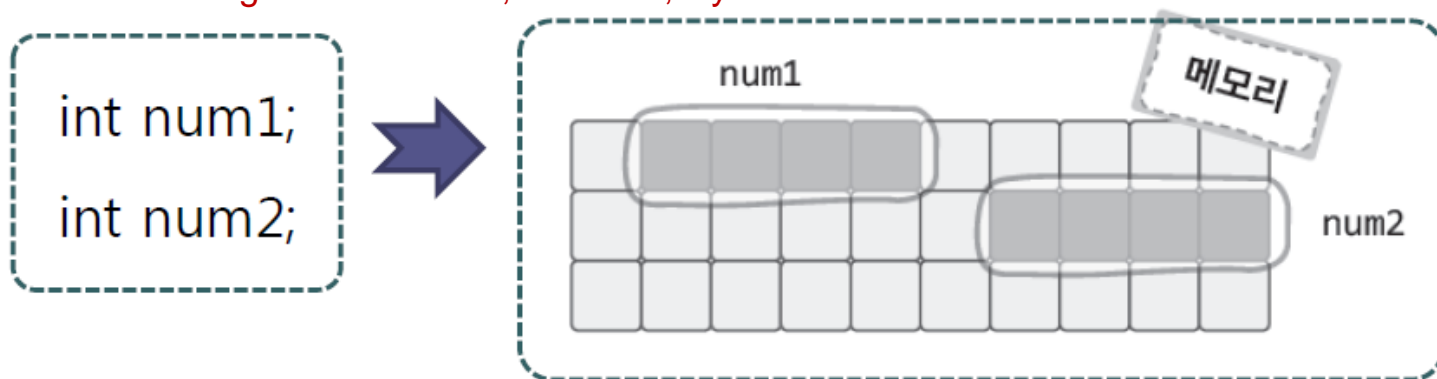


```
int num;
```

변수 선언

int와 같이 변수의 특성을 결정짓는 키워드를 가리켜 **‘자료형’**이라 한다.

Data? Data ?
e.g. -> , 0 ~ 150, byte -> int



변수(Variable)선언 방법

타입 변수 명;

```
int score ;
```

// int 타입의 자료형으로 이름을 score로 변수선언

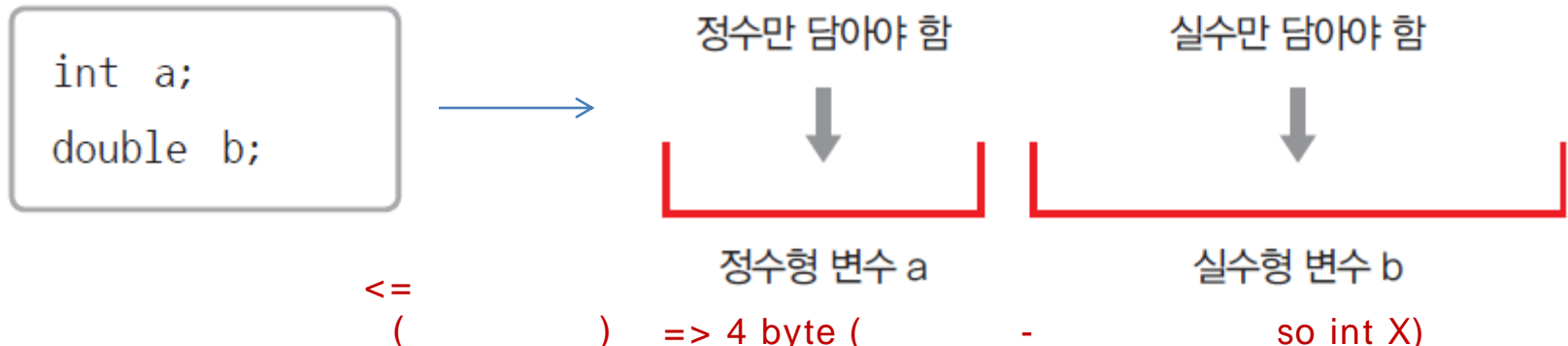
```
score = 100;
```

// 변수 score에 (저장장소에) 100(데이터)를 저장

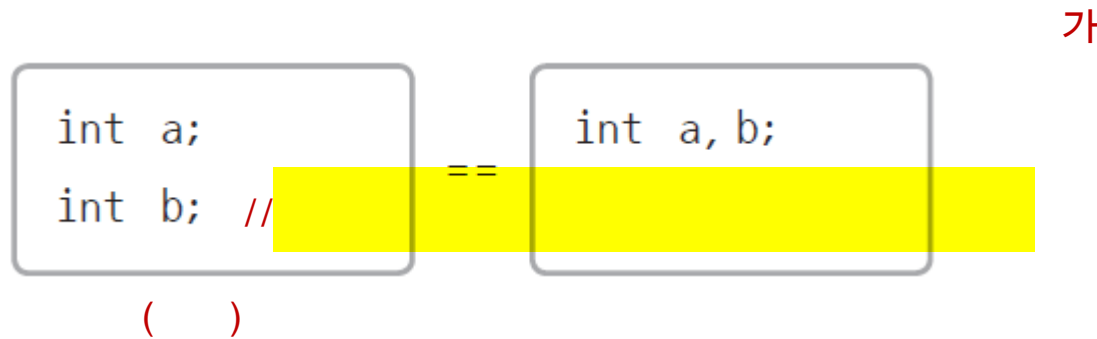
```
int score = 100; &
```

```
String str = new String("abc");
```

```
-> , str = null;
```



변수(Variable)선언 방법



① 가능

```
int a;  
float b;  
int c;  
float d;
```

② 가능

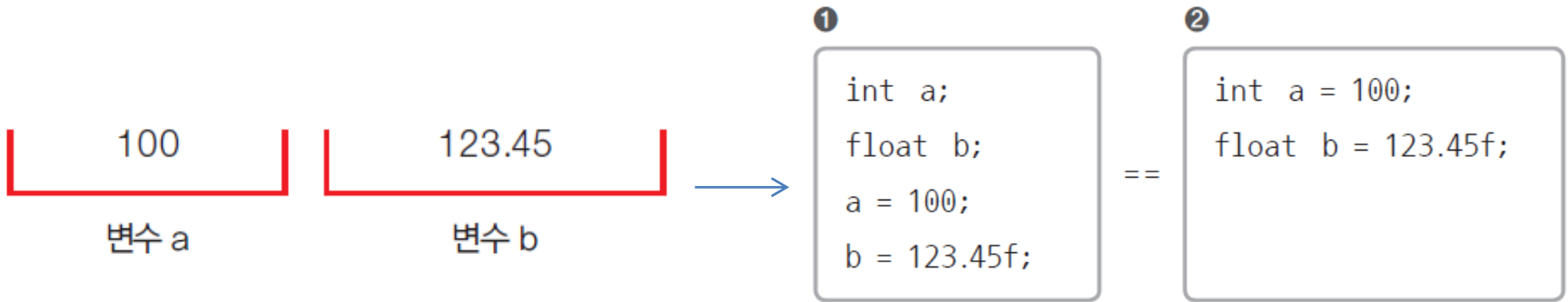
```
int a, c;  
float b, d;
```

③ 불가능

```
int a, float b;  
int c, float d;
```

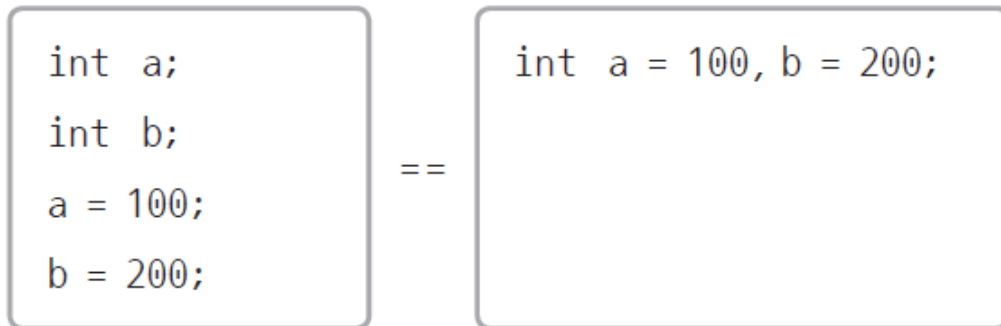
변수(Variable)선언 방법

- 변수에 값을 대입하는 방법



실수형인 float에 값을 대입할 때는 숫자의 맨 뒤에 float를 의미하는 'f'를 붙여야 함

a, b가 모두 정수형 변수일 때



예제

```
class UseVariable {  
    public static void main(String[] args) {  
        // 1. 정수형 타입의 변수 num1 변수를 선언하자  
        // 2. 변수 num1에 숫자 데이터 10을 저장하자.  
        // 3. 정수형 타입의 변수 num2 변수를 선언하고  
        //   숫자 20을 대입하자.  
        // 4. 정수형 타입의 변수 num3을 선언하고  
        //   변수 num1과 num2를 합하는 연산한 결과를  
        //   변수 num3에 대입한다.  
        // 5. 연산의 결과를 출력하자.  
    }  
}
```

$$10 + 20 = 30$$

예제

```
class UseVariable {  
    public static void main(String[] args) {  
        int num1;  
        num1=10;  
        int num2=20;  
        int num3=num1+num2;  
        System.out.println(num1+" "+num2+"="+num3);  
    }  
}
```

10+20=30

자료형의 종류와 구분

▶ 기본형(Primitive type)

- 8개 (boolean, char, byte, short, int, long, float, double)
- 실제 값을 저장

▶ 참조형(Reference type)

- 기본형을 제외한 나머지(String, System 등)
- 객체의 주소를 저장(4 byte, 0x00000000~0xffffffff)

자료형의 종류와 구분

자료형	데이터	메모리 크기	표현 가능 범위
boolean	참과 거짓	1 바이트	true, false
char	문자	2 바이트	모든 유니코드 문자
byte	정수	1 바이트	-128 ~ 127
short		2 바이트	-32768 ~ 32767
int		4 바이트	<u>-2147483648 ~ 2147483647</u>
long		8 바이트	-9223372036854775808 ~ 9223372036854775807
float	실수	4 바이트	$\pm(1.40 \times 10^{-45} \sim 3.40 \times 10^{38})$ 6
double		8 바이트	$\pm(4.94 \times 10^{-324} \sim 1.79 \times 10^{308})$ 15

- 정수 표현 byte, short, int, long
- 실수 표현 float, double
- 문자 표현 char
- 참과 거짓의 표현 boolean

자료형의 종류와 구분

- ▶ **논리형** – true와 false중 하나를 값을 가지고, 조건 식과 논리적 계산에 사용된다.
- ▶ **문자형** – 문자를 저장하는데 사용되며, 변수 당 하나의 문자만을 저장할 수 있다.
- ▶ **정수형** – 정수 값을 저장하는데 사용된다. 주로 사용하는 것은 int와 long이며, byte는 이진데이터를 다루는데 사용되며, short은 c언어와의 호환을 위해 추가되었다.
- ▶ **실수형** – 실수 값을 저장하는데 사용된다. float와 double이 있다.

크기 종류	1	2	4	8
논리형	boolean			
문자형		char		
정수형	byte	short	int	long
실수형			float	double

char + short 가 <-> char + int 가 ('a' + 1)
b/c short 가 char 가 _____

변수의 이름을 짓는 방법

- 변수 이름의 제약사항

- 숫자로 시작 불가
- \$와 _ 이외의 다른 특수문자 사용 불가
- 키워드는 변수의 이름으로 사용 불가

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

변수의 이름을 짓는 방법

1. 대소문자가 구분되며 길이에 제한이 없다.

- True와 true는 서로 다른 것으로 간주된다.

2. 예약어(Reserved word)를 사용해서는 안 된다.

- true는 예약어라 사용할 수 없지만, True는 가능하다.

3. 숫자로 시작해서는 안 된다.

- top10은 허용하지만, 7key는 허용되지 않는다.

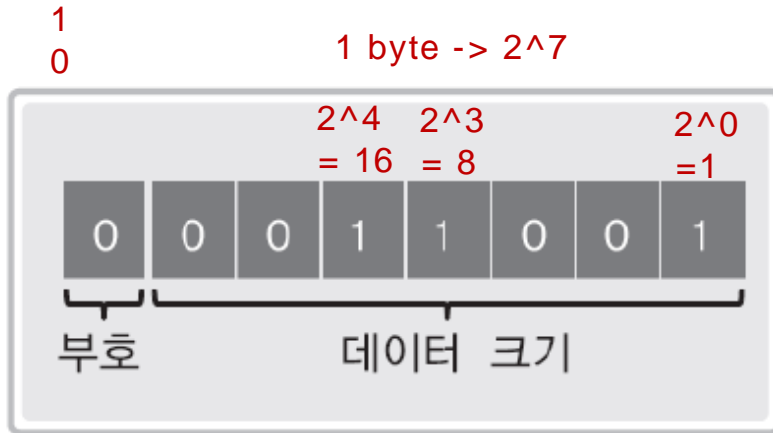
4. 특수문자는 '_'와 '\$'만을 허용한다.

- \$harp은 허용되지만 S#arp는 허용되지 않는다.

정수를 표현하는 방식

• 정수의 표현

- 가장 왼쪽 비트인 MSB(Most Signification Bit)는 부호를 나타낸다.
- MSB를 제외한 나머지는 크기를 나타낸다.
- 바이트 크기의 차이는 표현범위의 차이로 이어진다.



=> 25



- MSB 0 양 수
- 0011001 25

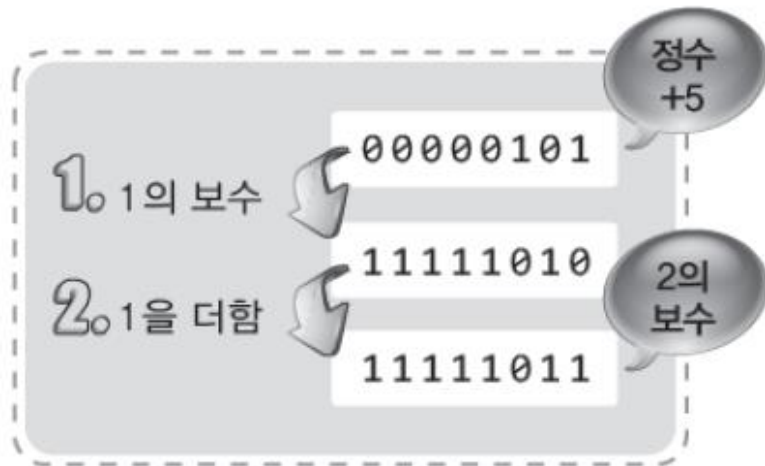
양의 정수 25

음의 정수 표현방식

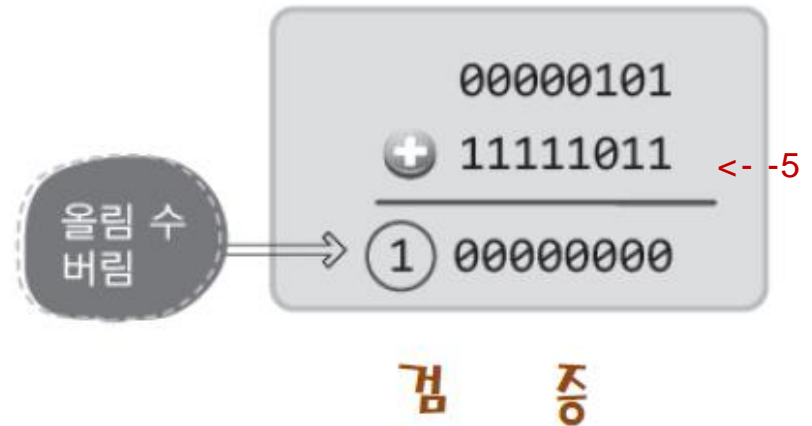
- 음의 정수 표현

* CPU
 $1 + (-1) = 0$

- 양의 정수 표현방식과 다르다.
- 양의 정수와의 합이 0이 되는 구조로 정의
- 2의 보수가 음의 정수 표현방식



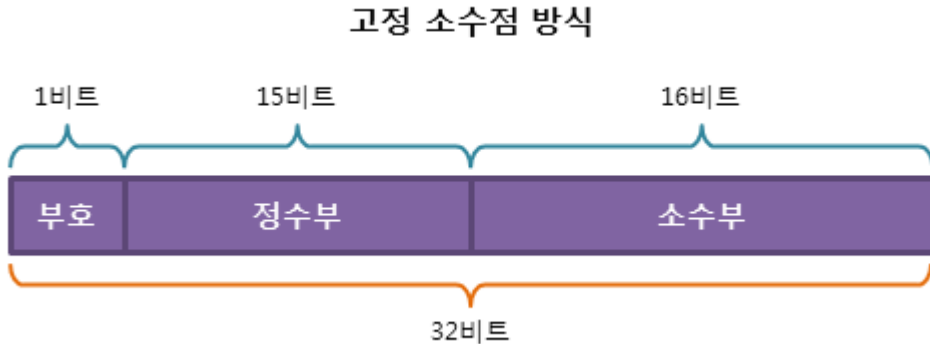
-5의 2진수 표현



실수 표현

- 실수표현의 문제점

- 0과 1사이의 실수만 해도 그 수가 무한대
- 단순히 몇 바이트 정도로 모든 실수의 표현은 불가능하다.



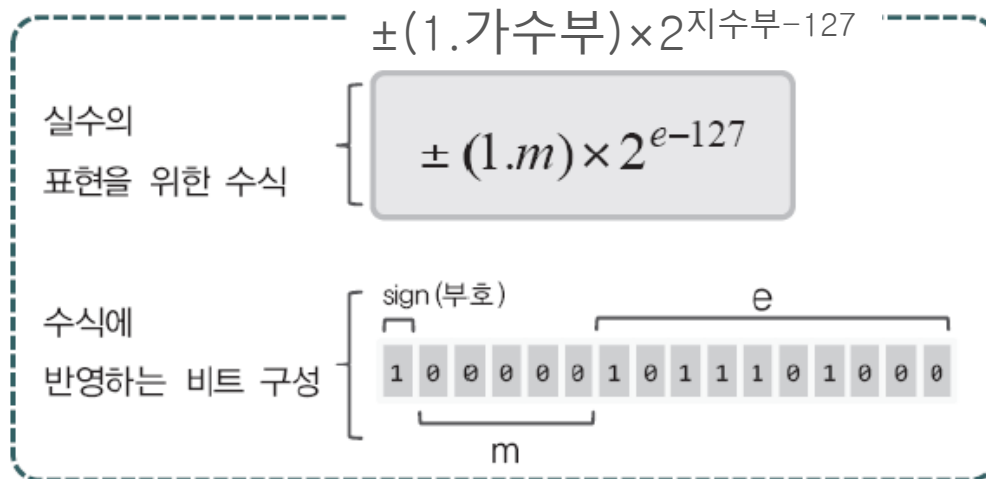
정수부와 소수부의 자릿수가 크지 않아 표현할 수 있는 범위가 매우 적다는 단점이 있다.

실수 표현

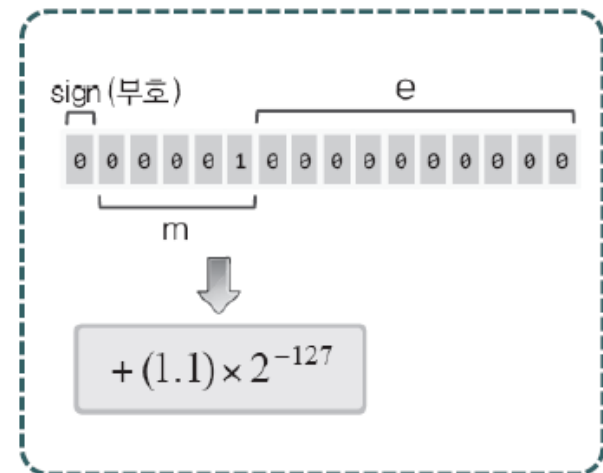
- 문제점에 대한 해결책

- 부동 소수점 표현

- 정밀도를 포기하고, 대신에 표현할 수 있는 값의 범위를 넓히자.
- 하지만 부동소수점의 표현은 항상 오차가 발생



실수 표현을 위한 수식의 도입



표현의 예

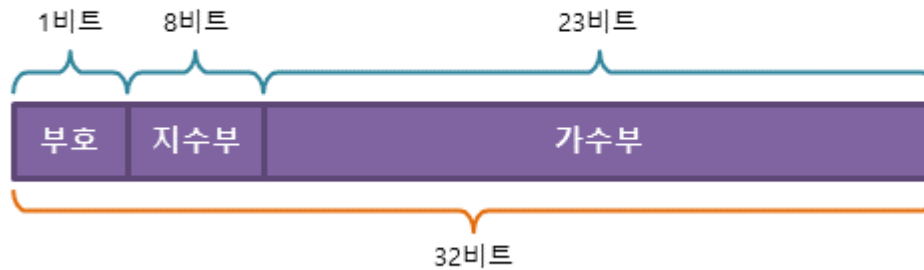
실수 표현

- 문제점에 대한 해결책
 - 부동 소수점 표현

가

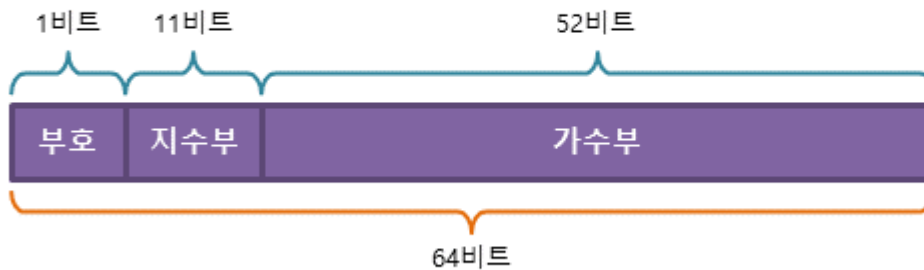
!

IEEE float형 부동 소수점 방식



float형 타입은
소수 6자리까지는 표현

IEEE double형 부동 소수점 방식



double형 타입은
소수 부분 15자리까지 표현

예제

```
class VariableDecl {  
    public static void main(String[] args) {  
        double num1 = 1.00000001; -> . X  
        double num2 = 2.00000001; =>  
        result = num1 + num2; ((double)n1);  
        System.out.println(result); () ->  
  
        double num3 = num1 * 100000000;  
        double num4 = num2 * 100000000;  
        double result1 = num3 + num4;  
        System.out.println(result1/100000000);  
    }  
}
```

```
3.000000019999999997  
3.00000002
```

정수 자료형에

- 자바의 4가지 정수 자료형

- Byte, short, int, long
- 정수를 표현하는데 사용되는 바이트 크기에 따라서 구분이 됨

- 작은 크기의 정수 저장에는 short? 아니면 int?

- CPU는 int형 데이터의 크기만 연산 가능
- 때문에 연산직전에 short형 데이터는 int형 데이터로 자동 변환
- 변환 과정을 생략할 수 있도록 int를 선택

- 그럼 short 와 byte는 왜 필요한가?

- 연산보다 데이터의 양이 중시되는 상황도 존재!!
- MP3 파일, 동영상 파일
- 데이터의 성격이 강하다면 short와 byte를 사용!!

실수 자료형

- 자바의 2가지 실수 자료형

- float, double
- Float는 소수점 이하 6자리 double은 ¹⁵~~12~~자리 정밀도

- 실수 자료형의 선택 기준

- float, double 모두 매우 충분한 표현 범위를 자랑한다.
- 이 둘의 가장 큰 차이점은 정밀도 이다.
- 따라서 필요한 정밀도를 바탕으로 자료형을 결정한다.
- 일반적으로 double의 선택이 선호된다.

실수의 e 표기법과 16진수 8진수 표현

```
class ENotation
{
    public static void main(String[] args)
    {
        double e1=1.2e-3;
        double e2=1.2e+3;

        int num1=0xA0E;
        int num2=0752;

        System.out.println(e1);
        System.out.println(e2);
        System.out.println(num1);
        System.out.println(num2);
    }
}
```

1.2×10^{-3}

$1.2 \times 10^{+3}$

16진수 A0E

8진수 752

0x -> 16

0 -> 8

실수의 e 표기법과 16진수 8진수 표현

```
class ENotation
{
    public static void main(String[] args)
    {
        double e1=1.2e-3;
        double e2=1.2e+3;

        int num1=0xA0E;
        int num2=0752;

        System.out.println(e1);
        System.out.println(e2);
        System.out.println(num1);
        System.out.println(num2);
    }
}
```

0.0012
1200.0
2574
490

문자 자료성 char

- 자바의 문자 표현 char 2 byte

- 문자 하나를 2바이트로 표현하는 유니코드 기반으로 표현
- 유니코드는 전 세계의 문자를 표현할 수 있는 코드의 집합
- 문자는 작은 따옴표로 표현한다.
- 문자는 char형 변수에 저장한다. 저장 시 실제로는 유니코드 값 저장

문자 자료형 char

- 자바의 문자 표현

오른쪽의 두 코드는
사실상 동일한 코드

```
char ch1='A';  
char ch2='한';
```



변환 발생

```
char ch1=65;           // 65는 16진수로 0x41  
char ch2=54620;        // 54620은 16진수로 0xD55C
```

예제

```
class UnicodeChar {  
  
    public static void main(String[] args) {  
  
        char ch1='A';  
        char ch2='한';  
        System.out.println(ch1);  
        System.out.println(ch2);  
  
    }  
}
```

A
한

'참'과 '거짓'을 표현하기 위한 자료형 : boolean

- 논리적인 표현을 위한 두 가지 약속

- true '참'을 의미하는 키워드
- false '거짓'을 의미하는 키워드

t/f
chk = false;
if(num > 10)
 boolean
while()

- 키워드 true와 false에 대한 좋은 이해

- 숫자의 관점에서 이해하려 들지 말자.
- 자바에서의 true와 false는 그 자체로 저장 가능한 데이터이다.
- true와 false의 저장을 위한 자료형 boolean!

예제

```
class BooleanVar
{
    public static void main(String[] args)
    {
        boolean b1=true;
        boolean b2=false;

        System.out.println(b1);
        System.out.println(b2);
        System.out.println(3<4);
        System.out.println(3>4);
    }
}
```

true
false
true
false

변수, 상수, 리터럴

- ▶ 변수(variable) – 하나의 값을 저장하기 위한 공간
- ▶ 상수(constant) – 한 번만 값을 저장할 수 있는 공간
- ▶ 리터럴(literal) – 그 자체로 값을 의미하는 것

```
int score = 100;
```

```
    score = 200;
```

```
char ch = 'A';
```

```
String str = "abc";
```

```
<- final int MAX = 100;
```

e.g. final double PI = 3.141592 ->

X)

```
MAX = 200; // 에러
```

|

리터럴과 접미사

```
boolean power = true;
```

```
char ch = 'A';
```

```
char ch = '\u0041';
```

```
char tab = '\t';
```

```
byte b = 127;
```

```
short s = 32767;
```

```
int i = 100;
```

```
int oct = 0100;
```

```
int hex = 0x100;
```

```
long l = 1000000000000L;
```

```
float f = 3.14f
```

```
double d = 3.14d
```

```
float f = 100f;
```

```
10.    →    10.0
```

```
.10    →    0.10
```

```
10f    →    10.0f
```

```
3.14e3f → 3140.0f
```

```
1e1    →    10.0
```

변수의 기본값과 초기화

- 변수의 초기화 : 변수에 처음으로 값을 저장하는 것
 - * 지역변수는 사용되기 전에 반드시 초기화해줘야 한다.

자료형	기본값
boolean	false
char	'\u0000'
byte	0
short	0
int	0
long	0L <small>L</small> <small>ㄱ</small>
float	0.0f
double	0.0d 또는 0.0 <small>d</small> <small>ㄱ</small>
참조형 변수	null <small>=</small>

```
boolean isGood = false;
```

```
char grade = ' '; // 공백
```

```
byte b = 0;
```

```
short s = 0;
```

```
int i = 0;
```

```
long l = 0; // 0L로 자동변환
```

```
float f = 0; // 0.0f로 자동변환
```

```
ㄱ double d = 0; // 0.0로 자동변환
```

```
String s1 = null;
```

```
String s2 = ""; // 빈 문자열
```

문자와 문자열

```
char ch = 'A';
```

```
char ch = 'AB'; // 에러
```

```
String s1 = "AB";
```

```
char ch = "; // 에러
```

```
String s1 = "";
```

-> null !

```
String s1 = "A" + "B"; // "AB"
```

"" + 7 → "" + "7" → "7"

"" + 7 + 7 → "7" + 7 → "7" + "7" → "77"

7 + 7 + "" → 14 + "" → "14" + "" → "14"

문자열 + any type → 문자열

any type + 문자열 → 문자열

상수와 형변환

자료형을 기반으로 표현이 되는
리터럴

리터럴 언제 사용했었지?

표현되는 데이터는 리터럴 아니면 변수

```
int num = 1 + 5;  
System.out.println(2.4 + 7.5);
```

리터럴 과 변수의 비교

- 변수와 마찬가지로 상수도 메모리 공간에 저장이 된다.
- 다만 이름이 존재하지 않으니 값의 변경이 불가능하다.
- 상수는 존재 의미가 없어지면 바로 소멸된다.

리터럴도 자료형을 기반으로 저장됩니다.

리터럴 의 저장방식의 근거는 자료형

int, double과 같은 자료형은 데이터 표현의 기준이다.

따라서 변수뿐만 아니라 상수의 데이터 저장 및 참조의 기준이다.

정수형 리터럴과 실수형 리터럴의 표현 자료형

정수형 리터럴

int형으로 표현

실수형 리터럴

double형으로 표현

접미사

다음 세 문장에서 컴파일 오류가 발생하는 이유는?

```
int num1=100000000000;           // num에 저장 불가!  
long num2=100000000000;          // 상수의 표현이 먼저이므로!  
float num3=12.45;                 // 12.45는 double형상수
```

접미사를 이용한 상수 표현방식의 변경

```
long num1=100000000000L;          // 접미사L은 long형 상수표현을 의미  
float num2=12.45F;                 // 접미사F는 float형 상수표현을 의미
```

예제

```
class SuffixConst {  
    public static void main(String[] args) {  
  
        double e1=7.125;  
        float e2=7.125;  
  
        long n1=10000000000;  
        long n2=150;  
  
        System.out.println(e1);  
        System.out.println(n1);  
    }  
}
```

예제

```
class SuffixConst {  
    public static void main(String[] args) {  
  
        double e1=7.125;  
        float e2=7.125F;  
  
        long n1=1000000000000L;  
        long n2=150;  
  
        System.out.println(e1);  
        System.out.println(n1);  
    }  
}
```

7.125
1000000000000

자료형의 변환

자료형 변환변환의 의미

- 자료형의 변환은 표현 방법의 변환

byte, short => int

int + long => long + long :

```
int main(String[] args)
{
    short num1=10;
    short num2=20;
    short result = num1 + num2;
    . . . .
}
```

num1(10) → 00000000 00001010

num2(20) → 00000000 00010100

short to int

int형 정수 10 → 00000000 00000000 00000000 00001010

int형 정수 20 → 00000000 00000000 00000000 00010100

int/long + float => float + float
가 가 !

int형 정수 1 → 00000000 00000000 00000000 00000001 -> 1

int to float

float형 실수 1.0 → 00111111 10000000 00000000 00000000 -> 1.0

자료형을 일치시켜야 하는 이유?

int	00000000	00000000	00000000	00000001
float	00000000	00000000	00000000	00000010
<hr/>				
	00000000	00000000	00000000	00000011

[그림 3-1 : 1 더하기 2는 3]

byte + int -> int + int -> int

.

자료형을 일치시켜야 하는 이유?



00000000 00000000 00000000 00000001

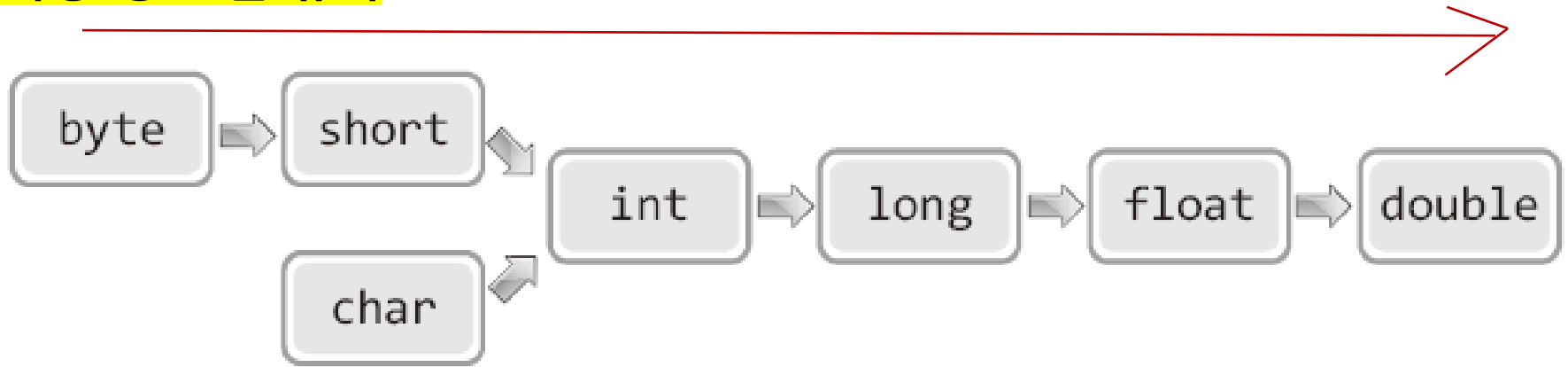
00111111 10000000 00000000 00000000

???????? ???? ?????? ???? ??????

[그림 3-2 : 1 더하기 1.0은?]

자동 형 변환 (Implicit Conversion)

자동 형 변환 규칙



```
double num2=3.5f+12;
```

2. 1.

12가 12f로 자동 형 변환

float + int -> float + float => float

double num2 = float
double

**

명시적 형 변환

명시적 형 변환을 하는 이유

→ 자동형 변환 발생지점의 표시를 위해서 case 1

→ 자동형 변환의 규칙에 위배되지만 변환이 필요한 상황 case 2

case 1

```
long num1 = 2147483648L;
```

```
int num2 = (int)num1;
```

case 2

```
int num3 = 100;
```

```
long num4 = (long)num3;
```

예제

```
class CastingOperation {  
    public static void main(String[] args) {  
        char ch1='A';  
        char ch2='Z';  
  
        int num1=ch1;  
        int num2=(int)ch2;  
  
        System.out.println("문자 A의 유니코드 값: "+num1);  
        System.out.println("문자 Z의 유니코드 값: "+num2);  
  
    }  
}
```

문자 A의 유니코드 값:65
문자 Z의 유니코드 값:90

예제

```
class CharToCode {  
    public static void main(String[] args) {  
        char ch = 'A'; // char ch = '₩u0041';로 바꿔 써도 같다.  
        int code = (int)ch; // ch에 저장된 값을 int형으로 변환  
        하여 저장한다.  
  
        System.out.println(ch);  
        System.out.println(code);  
    }  
}
```

A
65

예제

```
class CodeToChar {  
    public static void main(String[] args) {  
        int code = 65; // 또는 int code = 0x0041;  
        char ch = (char)code;  
  
        System.out.println(code);  
        System.out.println(ch);  
    }  
}
```

65
A

예제

```
class SpecialChar {  
    public static void main(String[] args) {  
        char single = '₩';    // single = "";와 같이 할 수 없다.  
        String dblQuote = "₩"Hello₩"";  
                                // 겹따옴표를 출력하려면 이렇게 한다.  
        String root = "c:₩₩";  
        System.out.println(single);  
        System.out.println(dblQuote);  
        System.out.println(root);  
    }  
}
```

```
'  
"Hello"  
C: \
```

예제

```
class StringTest {  
    public static void main(String[] args) {  
        double dd = 1D;  
        String a = 7 + " ";  
        String b = " " + 7;  
        String c = 7 + "";  
        String d = "" + 7;  
        String e = "" + "";  
        String f = 7 + 7 + "";  
        String g = "" + 7 + 7;  
        System.out.println(a);  
        System.out.println(b);  
        System.out.println(c);  
        System.out.println(d);  
        System.out.println(e);  
        System.out.println(f);  
        System.out.println(g);  
    }  
}
```

7
7
7
7
14
77


```
class ByteOverflow
{
    public static void main(String[] args)
    {
        byte b = 0;        // byte형 변수 b를 선언하고 0으로 초기화.
        int i = 0;

        // 반복문을 이용해서 b의 값을 1씩, 0부터 270까지 증가시킨다.

        for(int x=0; x <= 270; x++) {
            System.out.print(b++);
            // print는 출력 후 줄 바꿈을 하지 않는다.
            System.out.print('\t'); // tab을 출력한다.
            System.out.println(i++);
        }
    }
}
```

예제

```
class PrecisionTest
{
    public static void main(String[] args)
    {
        float f = 1.2345678901234567890f;
        double d = 1.2345678901234567890;
        float f2 = 0.100000001f;           // 0.1
        double d2 = 0.100000001;           // 접미사 생략되었음.
        double d3 = 0.100000000000000001;

        System.out.println(f);
        System.out.println(d);
        System.out.println(f2);
        System.out.println(d2);
        System.out.println(d3);
    }
}
```

1.23456789 ← 끝자리에서 반올림 됨
1.2345678901234567
0.1
0.100000001
0.100000000000000001

예제

```
class CastingEx1
{
    public static void main(String[] args)
    {
        double d = 100.0;
        int i = 100;
        int result = i + (int)d;

        System.out.println("d=" + d);
        System.out.println("i=" + i);
        System.out.println("result=" + result);
    }
}
```

d=100.0 ← 형변환 후에도 피연산자에는 아무런 변화가 없다.
i=100
Result=200

예제

```
class CastingEx2
{
    public static void main(String[] args)
    {
        byte b = 10;
        int i = (int)b;
        System.out.println("i=" + i);
        System.out.println("b=" + b);

        int i2 = 300;
        byte b2 = (byte)i2;
        System.out.println("i2=" + i2);
        System.out.println("b2=" + b2);
    }
}
```

```
i=10
B=10
i2=300
b2=44
```

정리합시다.

- 변수의 사용 목적

- 변수 선언 방법

to () <> : 가

- 기본 자료형

- 변수 와 상수 ^{final} 그리고 리터럴

int -> ->

- 자료형의 변환 (형 변환)

- 다루어야 하는 데이터에 맞는 변수 선언을 할 수 있다.

- 데이터들의 연산에 맞는 변수 선언을 할 수 있다.