

자료타입과 연산식

강사 : 강병준

시작하기 전에

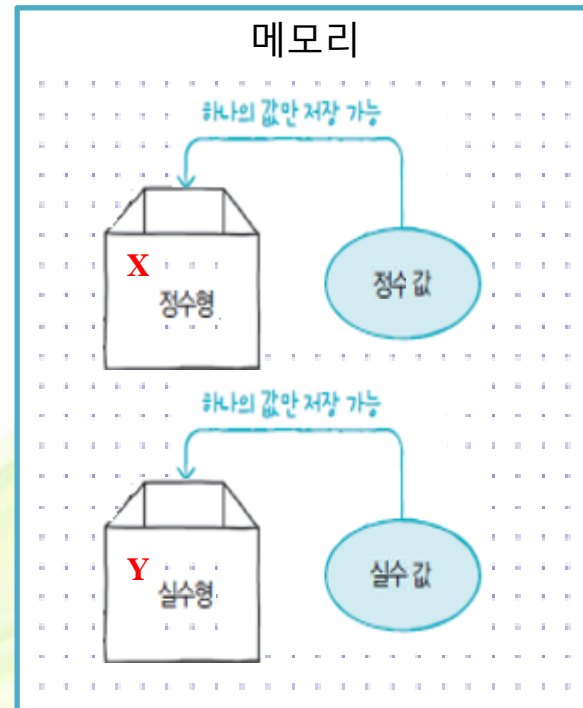
[핵심 키워드]: 변수, 변수 선언, 변수 사용, 변수 사용 범위

[핵심 포인트]

- 컴퓨터 메모리(RAM)는 값을 저장할 수 있는 수많은 번지(주소)들로 구성되어 있다.
- 메모리의 어디에, 어떤 방식으로 저장할지 정해놓지 않으면 프로그램 개발이 무척 어렵게 된다.
- 프로그래밍 언어는 이 문제를 해결하기 위해 변수라는 개념을 사용한다.
- 변수의 역할 및 사용 방법에 대해 알아본다.

❖ 변수 (Variable)

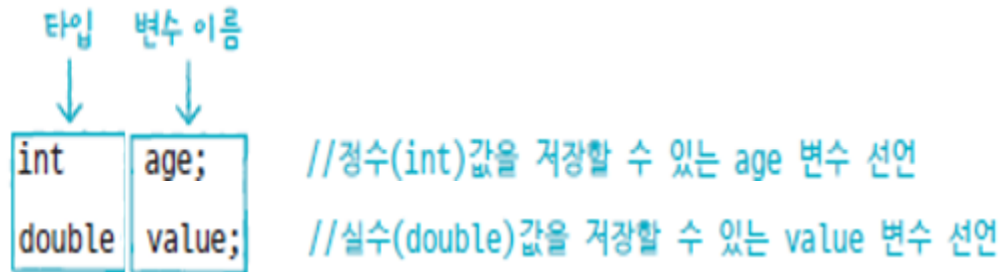
- 값을 저장할 수 있는 메모리의 특정 번지에 붙여진 이름
- 변수 통해 해당 메모리 번지에 하나의 값 저장하고 읽을 수 있음
- 변수는 정수, 실수 등 다양한 타입의 값을 저장할 수 있음



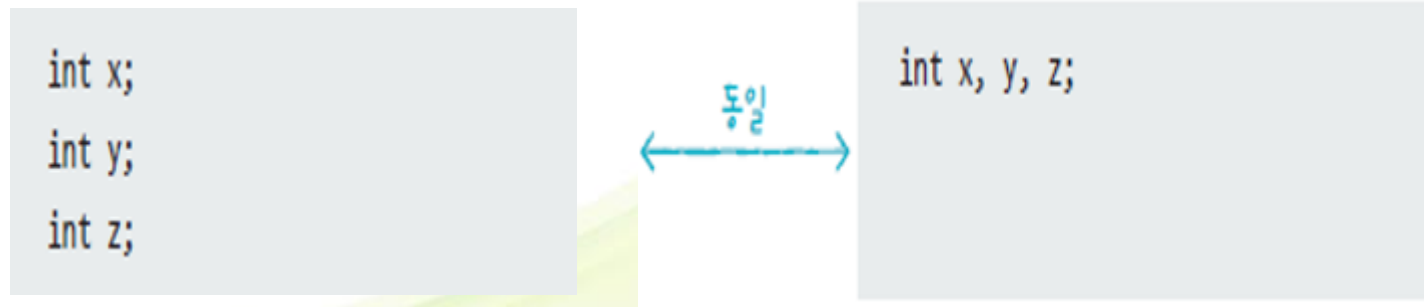
변수 선언

❖ 변수 사용 위해서 **변수 선언** 필요

- 변수에 어떤 타입의 데이터 저장할 것인지, 변수 이름 무엇인지 결정



- 같은 타입의 변수는 콤마 이용해 한꺼번에 선언할 수 있음



식별자와 예약어

• 식별자

프로그램을 작성하다 보면 직접 이름을 주어야 하는 클래스 이름, 메소드 이름, 변수 등과 같은 이름을 식별자라고 한다.

• 예약어

자바 프로그래밍을 하는데 있어 특정한 의미가 미리 부여되어 이미 만들어진 식별자를 말한다. 예약어에 등록되어 있는 것을 프로그래밍에서 식별자로 사용할 수 없다.

(const와 goto는 예약어로 등록만 되어 있을 뿐 사용되지 않는 예약어다)

• 식별자 명명 규칙

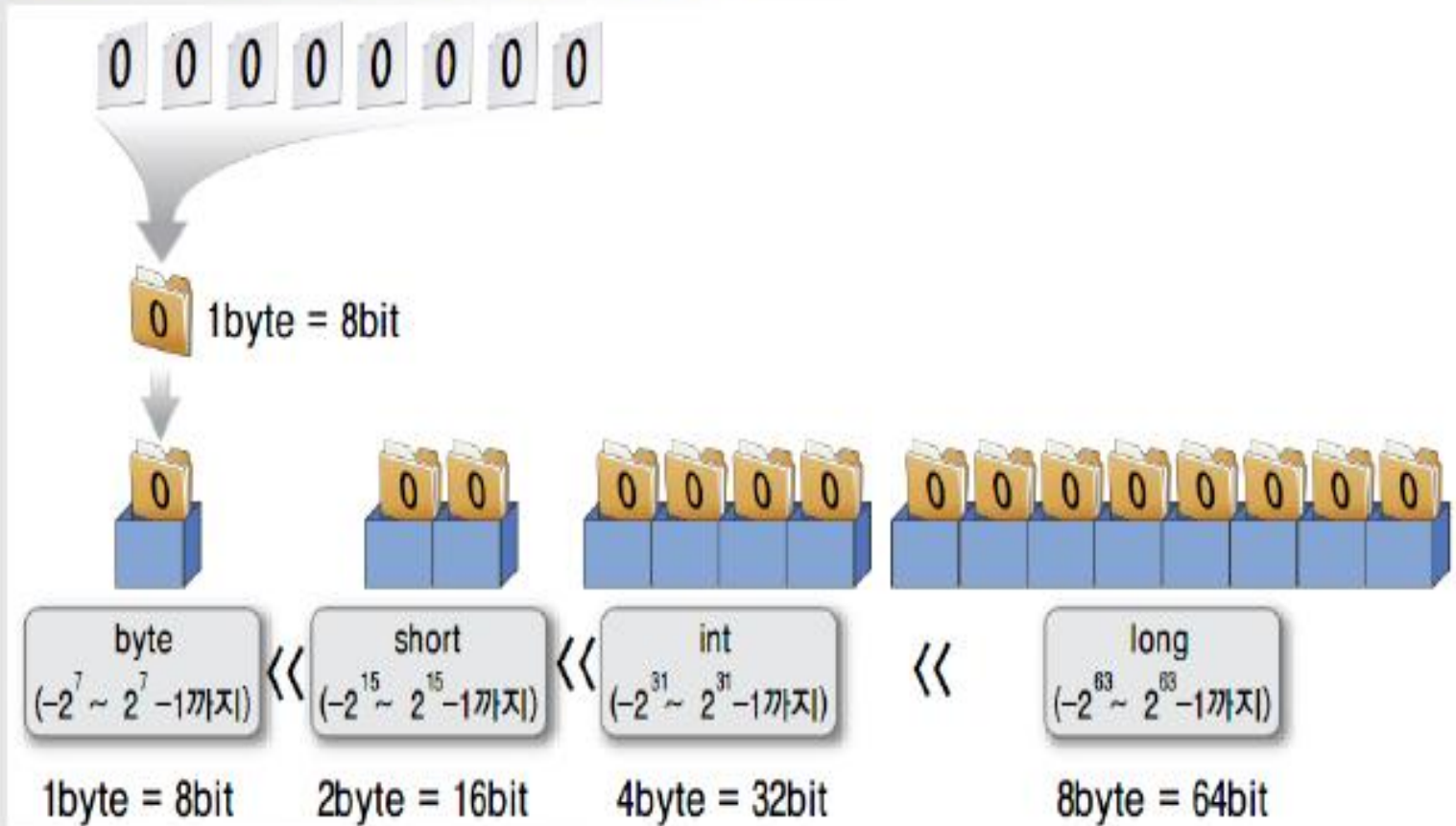
- 사용자 정의 명칭 (클래스, Method, Field)
 - 첫 글자는 ‘_’, ‘\$’, 영문 대,소문자 (한글 가능)
 - 글자수에 제한 없다.
 - 공백문자 및 특수 문자 사용 불가
 - 숫자는 첫 글자가 아닐 때 사용 가능
 - 예약어 사용 불가
 - 기타 단순 약속 (대, 소문자의 규칙)
- 첫 문자 구별
 - 클래스, 생성자 대문자 시작
 - 메세드, 멤버변수 소문자
 - 두 개 이상 단어를 붙여 쓸 때 단어시작은 시작은 대문자
예) Hello, HelloWorld, HelloWorldJava

자바 기본 유형

- 기본 자료형 : 자바에서 제공되는 8개의 기본 자료형은 다음과 같은 형태



- 정수형 : 4가지의 자료형이 제공



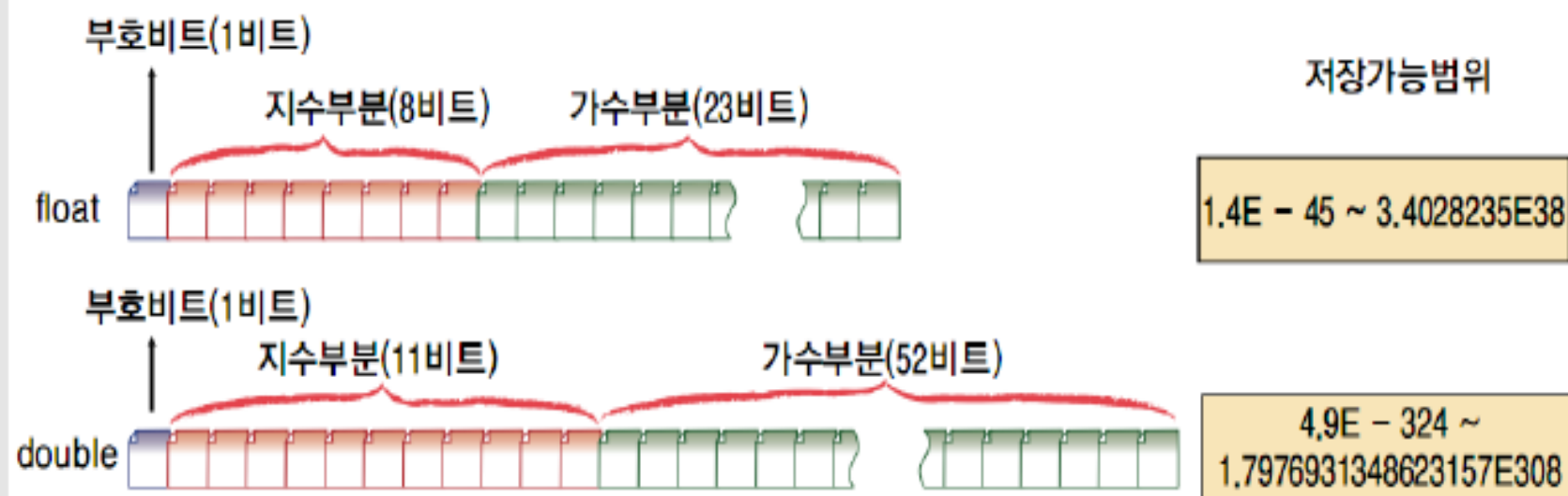
기본형(Primitive type)

- ▶ 논리형 - true와 false중 하나를 값으로 갖으며, 조건식과 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

● 실수형

- 부호와 지수(exponential)부분, 가수(mantissa)부분으로 구성
- 저장할 수 있는 크기에 따라 float형과 double형으로 구분
- 묵시적(default) 데이터형은 double형



변수(variable)

▶ 변수란?

1. 변수의 사전적 의미 : 변화하는 것
2. 프로그래밍 언어에서 사용되는 변수
값(value)이 저장된 메모리의 위치에 주어진 이름
변수에 값을 배정(assignment)할때, '=' 기호를 사용
3. 프로그램에 전달되는 정보나 그 밖의 상황에 따라 바뀔 수 있는
값을 의미한다 즉 상수를 기억시킬 수 있는 기억공간

☞ 변수선언이란?

자바가상머신에게 데이터를 저장하기 위한, 메모리를 할당해 달라고
부탁하는 것이다.

변수선언 데이터형 변수명;

-변수 선언시 주의사항

- 1, 숫자로 시작하면 안 된다. (하지만 뒤에는 올 수 있다.)
- 2, 특수문자는 들어갈 수 없다. (예외 _ \$는 들어올수 있다.)
- 3, 예약어는 들어 갈수 없다.

자료 선언과 할당

“이제부터 int 형태의 기억공간을 할당 받아 그 곳을 a란 이름으로 사용한다.”

int a;
↓ ↓
자료타입 변수명



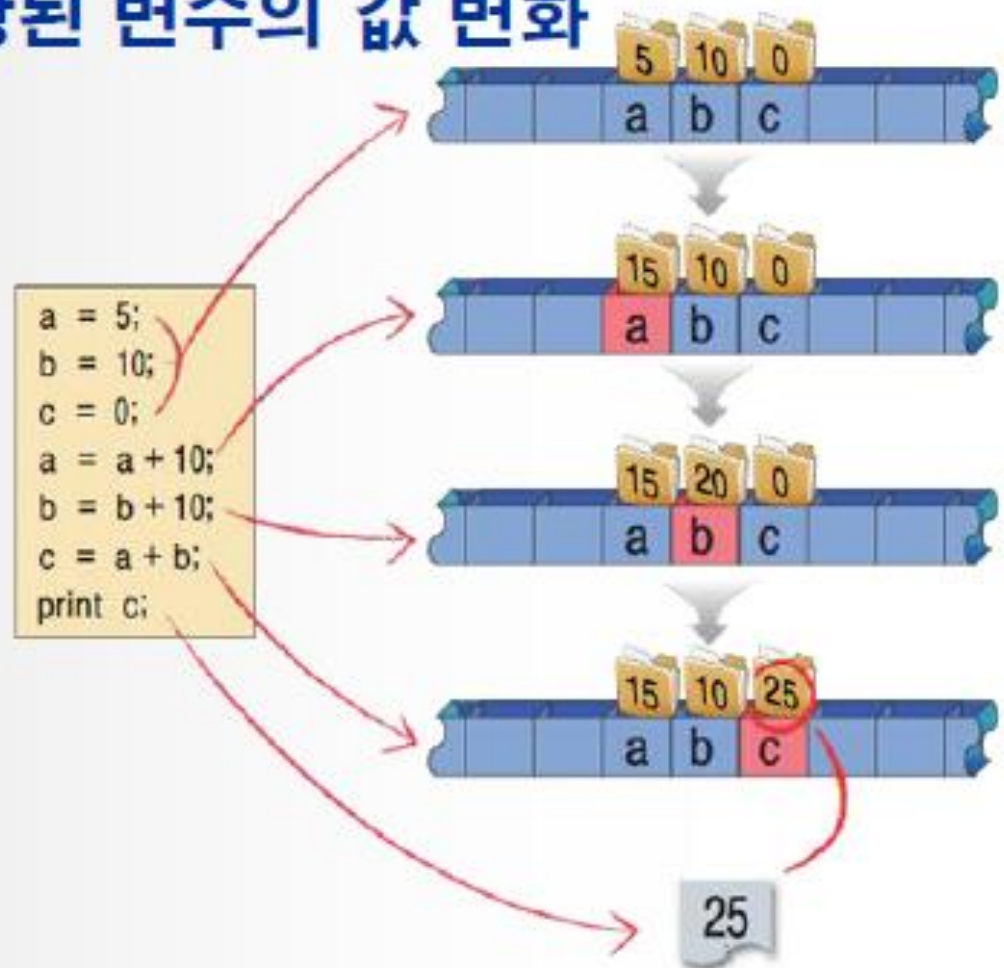
“a란 이름으로 할당된 기억공간에 5란 값을 저장하겠다.”

a란 이름으로 할당된 기억공간에 5란 값을 저장하겠다.

a = 5;
↓ ↓
변수명 저장할 값



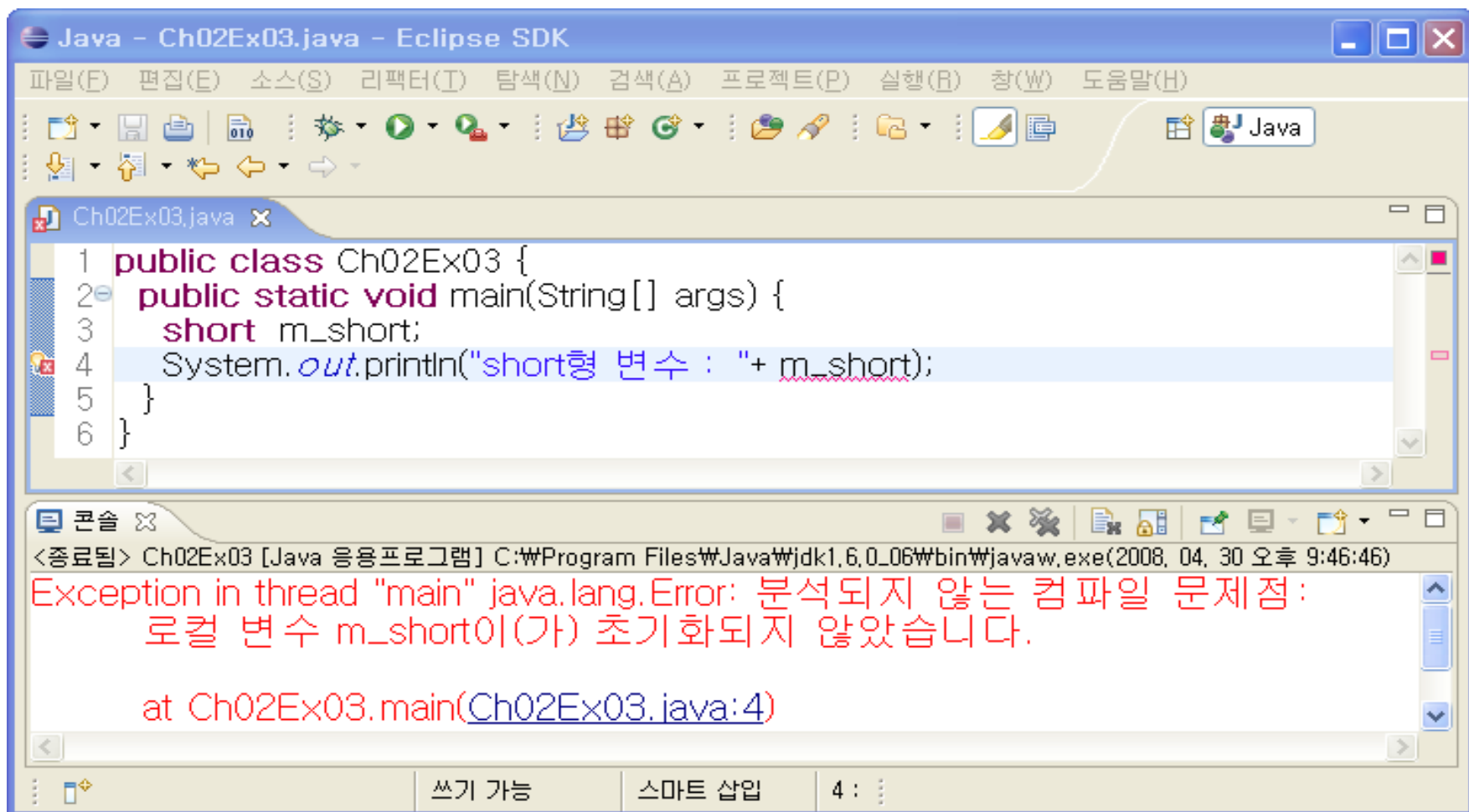
● 메모리에 저장된 변수의 값 변화



메모리에 저장된 변수값은 프로그램의 진행에 따라 변화됩니다.

변수 사용시 주의 점

- 값을 할당하지 않은 상태



The screenshot shows the Eclipse IDE with a Java file named Ch02Ex03.java. The code defines a public class Ch02Ex03 with a main method. Inside the main method, a short variable m_short is declared but not assigned a value before being used in a println statement. This results in a runtime exception.

```
1 public class Ch02Ex03 {  
2     public static void main(String[] args) {  
3         short m_short;  
4         System.out.println("short형 변수 : "+ m_short);  
5     }  
6 }
```

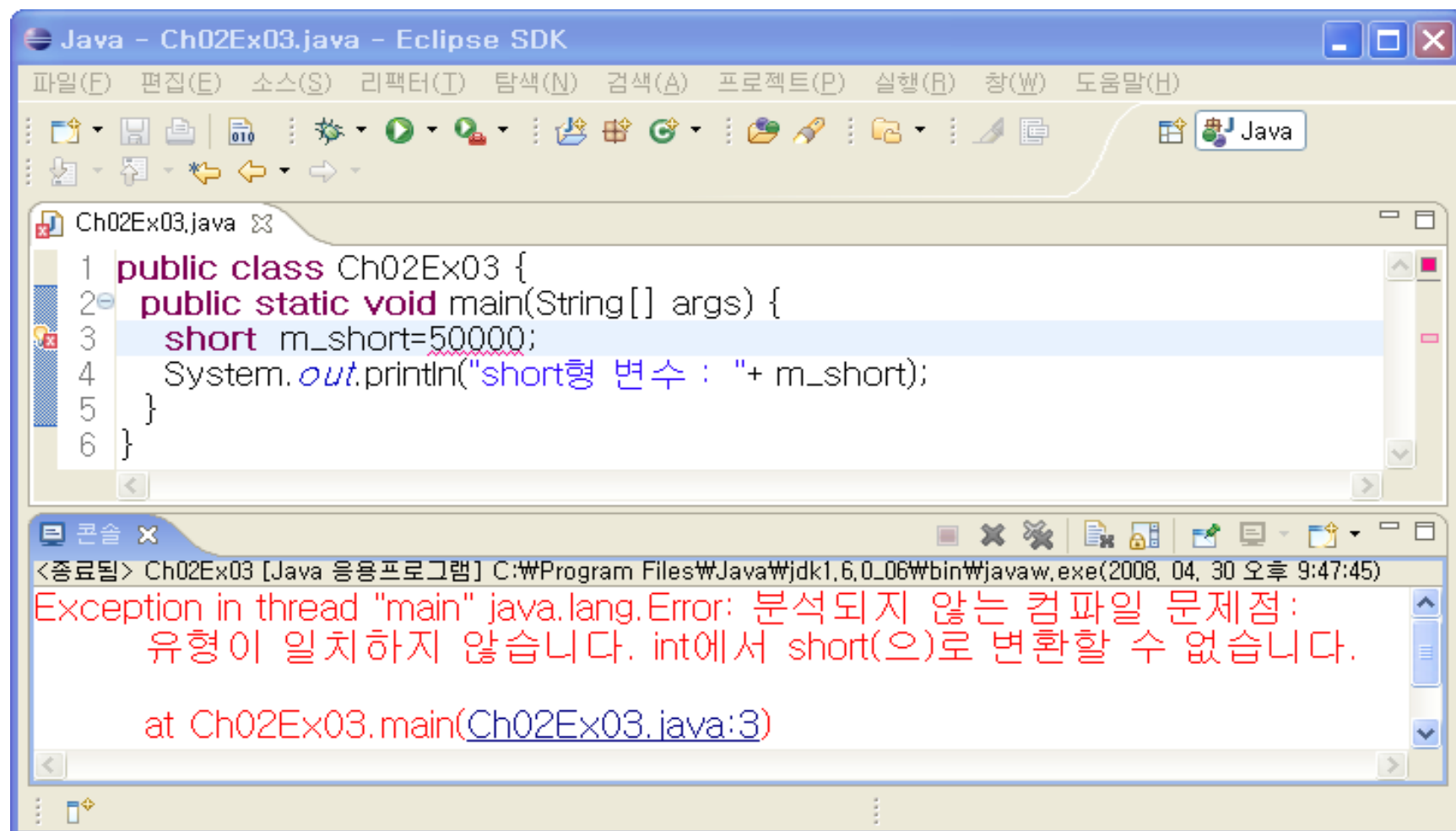
The console output shows the following error message:

```
<종료됨> Ch02Ex03 [Java 응용프로그램] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe(2008. 04. 30 오후 9:46:46)  
Exception in thread "main" java.lang.Error: 분석되지 않는 컴파일 문제점:  
로컬 변수 m_short이(가) 초기화되지 않았습니다.  
  
at Ch02Ex03.main(Ch02Ex03.java:4)
```

The status bar at the bottom indicates "쓰기 가능" (Writeable), "스마트 삽입" (Smart Insert), and "4 : ...".

변수 사용시 주의 점

- 값의 범위를 초과



Java - Ch02Ex03.java - Eclipse SDK

파일(E) 편집(E) 소스(S) 리팩터(T) 탐색(N) 검색(A) 프로젝트(P) 실행(R) 창(W) 도움말(H)

Ch02Ex03.java

```
1 public class Ch02Ex03 {
2     public static void main(String[] args) {
3         short m_short=50000;
4         System.out.println("short형 변수 : "+ m_short);
5     }
6 }
```

콘솔

<종료됨> Ch02Ex03 [Java 응용프로그램] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe(2008. 04. 30 오후 9:47:45)

Exception in thread "main" java.lang.Error: 분석되지 않는 컴파일 문제점:
유형이 일치하지 않습니다. int에서 short(으)로 변환할 수 없습니다.

at Ch02Ex03.main(Ch02Ex03.java:3)

정수형 - byte, short, int, long

정수형에는 byte, short, int, long 4개의 자료 타입이 있다. 이들을 크기순으로 나열하면 다음과 같다. 단위:byte

byte < short < int < long
1 2 4 8

실수형 - float, double

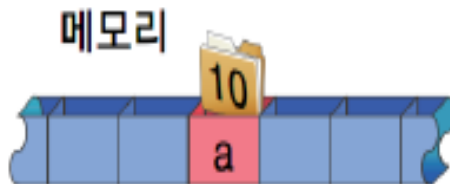
종 류	예	설 명
소수형	1234.5, 0.0000987	가장 일반적으로 사용하는 실수형 데이터
지수형	1.2345E3, 0.987E-5	영문자 E를 기준으로 앞에는 가수부 뒤에는 지수 부를 기술함.

자바 기본 유형

● 기본 자료형과 참조 자료형의 차이

```
int a = 10;
```

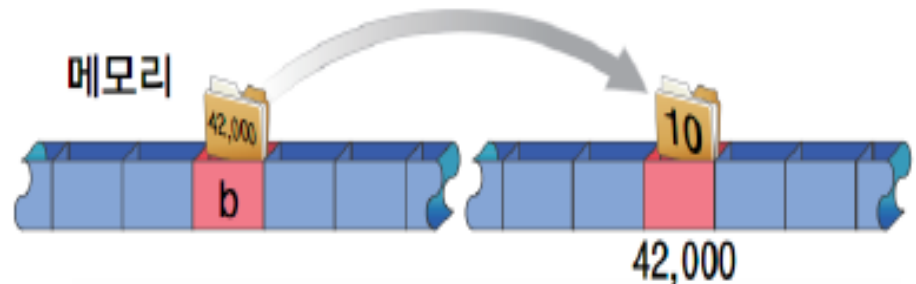
기본 자료형으로서 변수로 지정된 위치에 값이 저장되어 있습니다.



한 번의 접근으로 값을 가져올 수 있습니다.

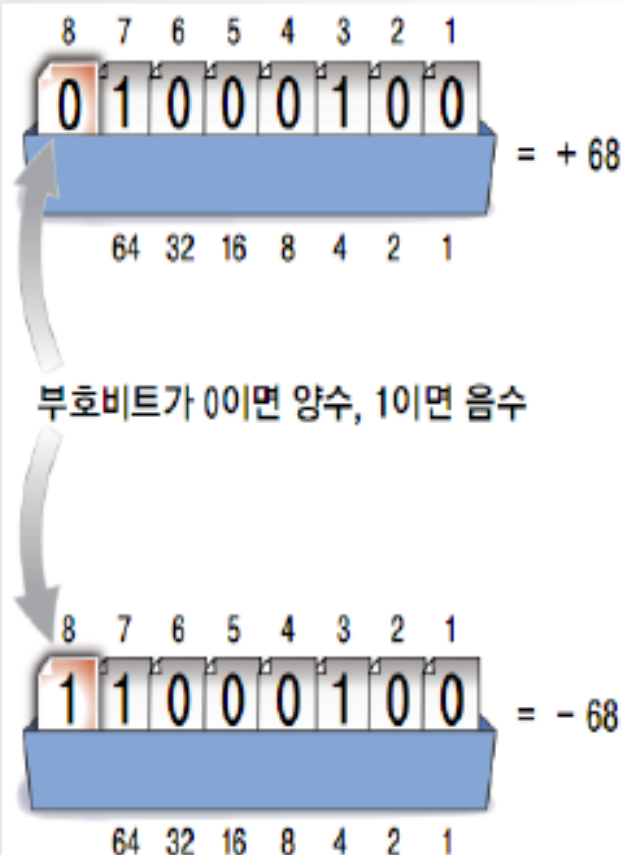
```
Integer b = new Integer(10);
```

참조 자료형으로서 변수로 지정된 위치에는 실제 값이 있는 곳의 주소가 저장됩니다.

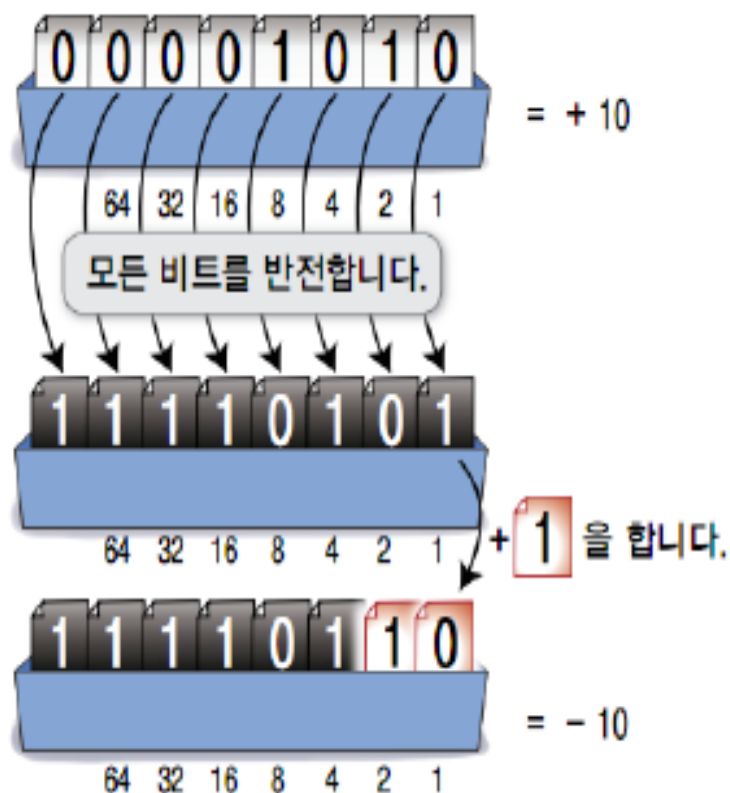


두 번의 접근으로 값을 가져올 수 있으므로 기본 자료형에 비해 효율성이 떨어질 수 있습니다.

● 정수의 음수 표현법(2의 보수법)



(a) 부호비트에 의한 음수표기법



(b) 2의 보수를 사용한 음수 표기법

상수

▶ 상수란?

변하지 않는 값, 특정한 값이나 의미가 있는 값, 저장공간

상수선언 변수명 = 상수 값;

-상수의 종류

논리형 상수, 문자형 상수, 정수형 상수, 실수형 상수...

☞ 논리형 상수 = true와 false만 가진다. `boolean a=true;`

☞ 문자형 상수 = 할당되는 값은 ' ' 에 들어간다. `char a= 'A' ;`

☞ 정수형 상수 = 일반 숫자를 의미한다. `int a=12;`

☞ 실수형 상수 `float a=3.122222f;`

`double a=123.1234567;`

소수점이 없는 정수형/ 정수형 상수

- '연필이 필통에 5자루 있다.'
- '연필 한 타스는 2400원인데, 한 타스에는 연필이 12자루가 있다.'
- 5, 2400, 12
 - 와 같이 소수점이 없는 데이터가 정수형이다.
 - 그 자체에 의미가 있고
 - 프로그램 실행 중에 그 값을 절대 변경할 수 없기에 이를 상수 (Constant)라 한다.

소수점이 있는 실수형

```
int pi=3.141592; //컴파일 에러  
double pi=3.142592;
```

```
float pi=3.142592; //컴파일 에러  
float pi=3.142592f;
```

```
public class VariableUseExample {  
    public static void main(String[] args) {  
        int hour = 3;  
        int minute = 5;  
        System.out.println(hour + "시간 " + minute + "분");  
  
        int totalMinute = (hour*60) + minute;  
        System.out.println("총" + totalMinute + "분");  
    }  
}
```

변수 사용

```
public class VariableExchangeExample {  
    public static void main(String[] args) {  
        int x = 3;                int y = 5;  
        System.out.println("x:" + x + ", y:" + y);  
        int temp = x;    x = y;    y = temp;  
        System.out.println("x:" + x + ", y:" + y);  
    }  
}  
  
public class VariableScopeExample {  
    public static void main(String[] args) {  
        int v1 = 15;  
        if(v1 > 10) {  
            int v2;                v2 = v1 - 10;  
        }  
        // v2 변수를 사용할 수 없기 때문에 컴파일 에러가 생김  
        //int v3 = v1 + v2 + 5;  
    }  
}
```

원의 면적 구하는 예

```
public class Ex02 {  
    public static void main(String[] args) {  
        double pi=3.141592;  
        int r=5;  
        double area;  
        area=r*r*pi;  
        System.out.println( "반지름이 " + r + "인 원의 면적은 " + area + "이다." );  
    }  
}  
  
public class LongExample {  
    public static void main(String[] args) {  
        long var1 = 10;  
        long var2 = 20L;  
        //long var3 = 10000000000000;    //컴파일 에러  
        long var4 = 10000000000000L;  
        System.out.println(var1);  
        System.out.println(var2);  
        System.out.println(var4);  
    }  
}
```

문자형 - char

10진수	ASCII	10진수	ASCII	10진수	ASCII	10진수	ASCII
0	NULL	32	SP	64	@	96	.
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	D
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(72	H	104	h
9	HT	41)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	SC2	50	2	82	R	114	r
19	SC3	51	3	83	S	115	s
20	SC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

문자형

[예] 'A', 'a', '3', '&', '#', '\n', '\t'

대문자 유니코드 값

'A'	'B'	'C'	'D'	'E'	'F'	...	'X'	'Y'	'Z'
65	66	67	68	69	70		88	89	90

소문자 유니코드 값

'a'	'b'	'c'	'd'	'e'	'f'	...	'x'	'y'	'z'
97	98	99	100	101	102		120	121	122

숫자형 유니코드 값

'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
48	49	50	51	52	53	54	55	56	57

● 문자형

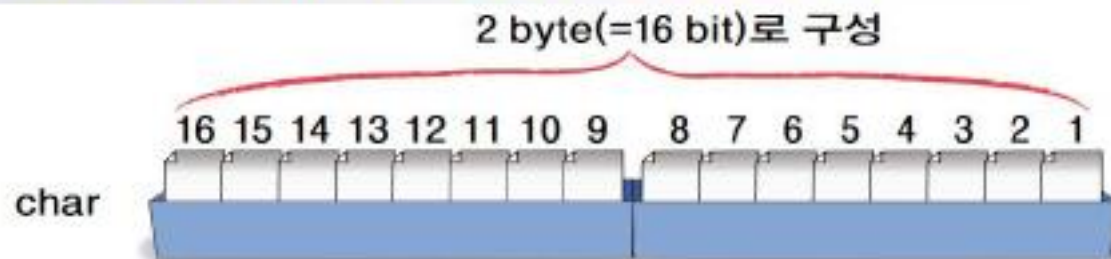
- 하나의 문자를 나타낼 수 있는 char형을 기본 자료형으로 제공
- 자바에서는 아스키코드(ASCII:8비트) 체계가 아닌 유니코드(Unicode:16비트) 사용
- 16비트의 유니코드를 사용함으로써 세계 다양한 나라들의 모든 언어를 나타낼 수 있음
- 유니코드는 $65,536(2^{16})$ 개의 문자를 나타낼 수 있음

● 문자 자료형

- 하나의 문자를 나타내기 위해 사용
- 문자 자료형 데이터를 나타내기 위해서는 하나의 따옴표(single quotation)

```
char grade1 = 'A';  
char grade2 = '\u0041'; // 'A'와 같은 의미의 유니코드  
char years = '2';
```

● 문자형과 유니코드(unicode)



총 2^{16} 개 (65, 536)개의
문자표현 가능

	AC0	AC1	AC2	AC3	AC4	AC5
0	가	감	갸	갯	갈	각
	AC06	AC08	AC09	AC0B	AC0C	AC0D
1	각	갑	갸	갯	갯	갈
	AC11	AC13	AC14	AC16	AC17	AC18
2	갸	갯	갯	갯	갯	갯
	AC20	AC22	AC23	AC25	AC26	AC27
3	갯	갯	갯	갯	갯	갯
	AC30	AC32	AC33	AC35	AC36	AC37

가 : AC00
각 : AC01

(a) 한글 유니코드

	304	305	306	307	308
0		ぐ	だ	ば	む
		3044	3045	3046	3047
1	あ	け	ち	ぼ	め
	3048	3049	3050	3051	3052
2	あ	げ	ち	ひ	も
	3053	3054	3055	3056	3057
3	い	こ	っ	び	や
	3058	3059	3060	3061	3062

ぐ : 3050
あ : 3041

(b) 일본어 유니코드

	003	004	005	006	007
0	0	@	P	`	p
	0030	0040	0050	0060	0070
1	1	A	Q	a	q
	0031	0041	0051	0061	0071
2	2	B	R	b	r
	0032	0042	0052	0062	0072
3	3	C	S	c	s
	0033	0043	0053	0063	0073

A : 0041
B : 0042

(c) 영문자 유니코드

```

public class CharExample {
    public static void main(String[] args) {
        char c1 = 'A'; char c2 = 65;      //문자를 직접 저장 //십진수로 저장
        char c3 = '\u0041'; char c4 = '가'; //16진수로 저장 //문자를 직접 저장
        char c5 = 44032; char c6 = '\uac00'; //십진수로 저장 //16진수로 저장
        System.out.println(c1);      System.out.println(c2); System.out.println(c3);
        System.out.println(c4);      System.out.println(c5); System.out.println(c6);
    }
}

public class Ex06 {
    public static void main(String[] args) {
        System.out.printf( " 문자 %c 의 유니코드는 %d \n" , 'A', (int)'A');
        System.out.printf( " 문자 %c 의 유니코드는 %d \n" , 'a', (int)'a');
        System.out.printf( " 문자 %c 의 유니코드는 %d \n" , '0', (int)'0');
        System.out.printf( "----- \n");
        System.out.printf( " 문자 %c 의 유니코드는 %d \n" , 'A'+1, (int)('A'+1));
    }
}

```

확장 특수 출력 문자

확장문자	의미
\n	엔터의 기능을 갖으며 줄을 바꾼다.(New Line)
\t	탭으로 일정한 간격을 띄운다(horizontal tab)
\b	백스페이스 기능으로 뒤로 한 칸 후진한다.(backspace)
\r	동일한 줄의 맨 앞으로 커서만 옮긴다. (carriage return)
\f	출력 용지를 한 페이지 넘긴다.(form feed)
\\	\문자를 출력한다.(back slash)
\'	'문자를 출력한다.(single quote)
\"	"문자를 출력한다.(double quote)
\0	null 문자

확장 특수 출력 문자 출력하기

```
public class Ex07 {  
    public static void main(String[] args) {  
        System.out.println( " \\n => \n 엔터키 다음문자");  
        System.out.println( " \\t => t 탭키 다음문자");  
        System.out.println( " \\r\\r => \r\r");  
        System.out.println( " \\r\\r' => \r'");  
        System.out.println( " \\r\\r" => \r"");  
    }  
}  
  
public class EscapeExample {  
    public static void main(String[] args) {  
        System.out.println("번호\\t이름\\t직업 ");  
        System.out.print("행 단위 출력\\n");  
        System.out.print("행 단위 출력\\n");  
        System.out.println("우리는 \\\"개발자\\\" 입니다.");  
        System.out.print("봄\\\"여름\\\"가을\\\"겨울");  
    }  
}
```

문자형의 종류

종 류	유 형	크 기	허 용 값
문자형	char	2 Byte (16 bit) 16비트 유니 코드	0 ~ 65535

char ch1 = 'A', ch2 = 'B';

ch1

ch2

65

66

```
char var1 = 'A';    //유니코드: 65
char var2 = 'B';    //유니코드: 66
char var3 = '가';    //유니코드: 44032
char var4 = '각';    //유니코드: 44033
```

하나의 문자를 저장할 수 있는 타입

‘A’

‘한’

작은 따옴표로 감싼 문자 리터럴은 유니코드로 변환되어 저장
=> char 타입은 정수 타입

문자열 (String)

▶ 문자열은 쉽게 문자의 연속이다. 자바에서는 “ ” 로 묶어서 표현

☞ 문자열의 예

```
String name = “kim min hee” ;
```

```
String hi = “Hi !!” ;
```

```
String a = name + hi;
```

```
public class StringExample {
```

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

```
        String name = "홍길동";
```

```
        String job = "프로그래머";
```

```
        System.out.println(name);
```

```
        System.out.println(job);
```

```
    }
```

```
}
```

문자열형

- 문자열(string) 상수는 일련의 문자들의 집합으로 구성되고 반드시 이중따옴표로 둘러싸야만 한다.
- ' Good' //잘못된 표현
- "Good"
- char ch = 'G';
- char ch2 = "Good";
//컴파일 에러:char형 변수에 문자열을 저장 못함

기본 입출력

- 출력을 위한 메소드
- `println()`, `print()`, `printf()`
- `printf()` 메소드는 “형식 지정자”에 %와 영문자를 조합해서 다양한 자료 타입을 출력할 수 있다.

정수형 상수의 출력 형식 지정자

형식	적용	출력 상태						설명
%자리수d	<code>printf("\n%5d\n", 16);</code>				1	6		빈 공간이 왼쪽에 생김
%-자리수d	<code>printf("\n%-5d\n", 16);</code>	1	6					빈 공간이 오른쪽에 생김
%0자리수d	<code>printf("\n%05d\n", 16);</code>	0	0	0	1	6		왼쪽에 생긴 빈 공간을 0으로 채움

이스케이프 문자 (escape)

문자열 내부에 \는 이스케이프 문자를 뜻함

이스케이프 문자를 사용하면 특정 문자를 포함시키거나,
문자열의 출력을 제어할 수 있음

```

public class PrintTest {
    public static void main(String args[]) {
        System.out.println("kim\nlee\npark");
        // \n 줄바꿈 (line skip)
        System.out.println("kim\rlee\rpark");
        // \r 줄바꿈 (carrage return)
        System.out.println("kim\tlee\tpark");
        // \t 일정간격 띄움
    }
}

```

- 캐리지 리턴(Carriage Return)과 라인 피드(Line Feed)는 타자기 따온 것.
- 캐리지 리턴(Carriage Return) 은 현재 위치를 나타내는 커서를 맨 앞으로 이동시킨다는 뜻
- 라인피드 (Line Feed) 는 커서의 위치를 아랫줄로 이동시킨다는 뜻.
- 윈도우에서는 이 두 동작을 합쳐 Enter 동작을 하는것
"CR+LF" 로 커서를 앞으로 보낸후 줄을 한줄 바꾼다

```
class StringTest {
    public static void main(String args[])    {
        String str1="안녕 ";
        String str2="자바 ";
        String str3="반갑다 ";

        int age=2023-1990-1;

        System.out.print(str1+str2+str3);
        System.out.println("나이 : "+age + "살");
        System.out.printf("%5d", 16);
        System.out.printf("%-5d", 16);
        System.out.printf("%05d\n", 16);
    }
}
```

```
class DataTest {  
    public static void main(String[] arg) {  
        String str = "데이터형과 변수예제" ;  
        byte b=5;  
        short s=10;  
        int i=100;  
        long l=100L;  
        float f=12.34f;  
        double d=123.345;  
        char c='A';  
        boolean bo=true;  
  
        System.out.println(str);  
        System.out.println(b);  
        System.out.println(s);  
        System.out.println(i);  
        System.out.println(l);  
        System.out.println(f);  
        System.out.println(d);  
        System.out.println(c);  
        System.out.println(bo);  
    }  
}
```

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

```
        byte a1=100;  
        char a2='A';  
        short a3=2000;  
        int a4=100;  
        long a5=125l;  
        float a6=12.5f;  
        double a7=12.5;  
        boolean a8=true;  
        String str1="kim";
```

```
        System.out.println("byte 형 : "+a1);  
        System.out.println("char 형 : "+a2);  
        System.out.println("short 형 : "+a3);  
        System.out.println("int 형 : "+a4);  
        System.out.println("long 형 : "+a5);  
        System.out.println("float 형 : "+a6);  
        System.out.println("double 형 : "+a7);  
        System.out.println("boolean 형 : "+a8);  
        System.out.println("String 형 : "+str1);
```

```
    }  
}
```


변수, 상수, 리터럴

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

소스 코드에서 프로그래머에 의해 직접 입력된 값

```
int score = 100;
```

```
    score = 200;
```

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

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

변수의 기본값과 초기화

변수의 초기화 : 변수에 처음으로 값을 저장하는 것

* 지역변수는 사용되기 전에 반드시 초기화해주어야 한다.

자료형	기본값
boolean	false
char	'\u0000'
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d 또는 0.0
참조형 변수	null

```
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
```

```
double d = 0; // 0.0로 자동변환
```

```
String s1 = null;
```

```
String s2 = ""; // 빈 문자열
```

형 변환(Casting)

형 변환이란?

- boolean을 제외한 7개의 기본형은 서로 형변환이 가능하다.
- 값의 타입을 다른 타입으로 변환하는 것이다.

```
float f = 1.6f;
```

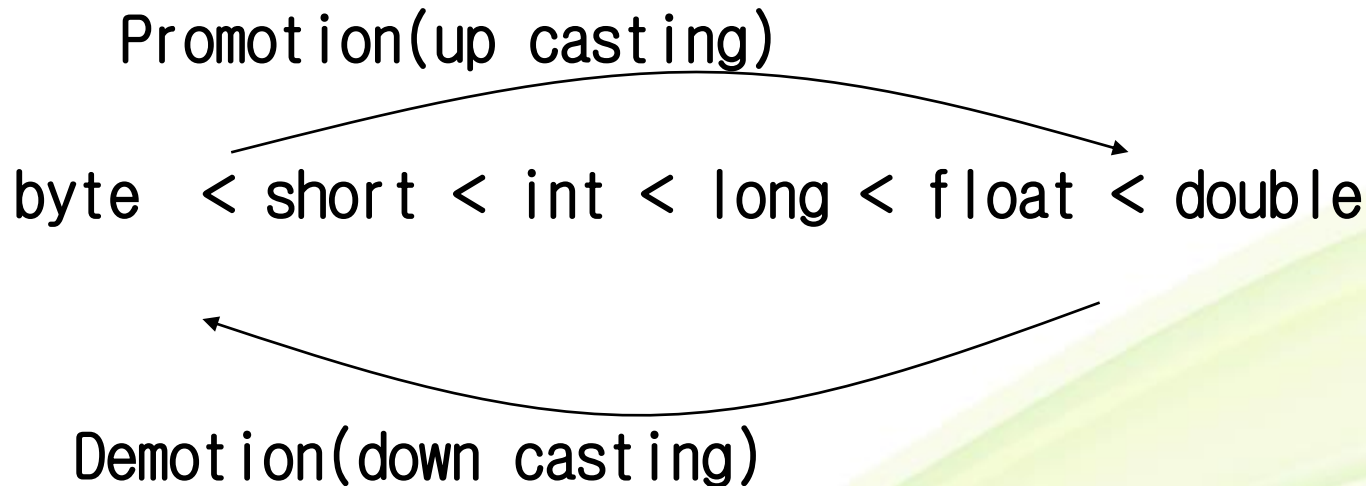
```
int i = (int)f;
```

변 환	수 식	결 과
int → char	(char) 65	'A'
char → int	(int) 'A'	65
float → int	(int) 1.6f	1
int → float	(float) 10	10.0f

형 변환 (type conversion, casting)

▶ 자바는 독자적인 형변환 변수형을 제공하지 않는다. 따라서 형 변환을 하려면 명시적으로 이를 지정해야 한다. Cast 연산자를 사용한다.

▶ Data 사이즈의 순위



☞ promotion : 형변환시 데이터타입이 큰 사이즈로 이동, 자동적으로 가능

☞ demotion : 작은 사이즈로 이동, 강제로 지정해야 한다.

형 변환 예제

예제 1)

```
long no = 991;  
int num = (int)no;  
long no = 5;  
int num = 991;
```

예제 2)

```
long no = 6;  
int num = 991;  
double z = 12.414;  
float z1 = 12.414f;
```

- ▶ 데이터 타입에서 short와 char는 둘다 16비트이긴 하지만 그 데이터 범위가 다르기 때문에 반드시 형변환을 해야 한다.

```
class DataTypeTest2{
    public static void main(String args[]) {
        /*
            byte a1=100;
            double a2;
            a2=a1; // 큰자료형=작은자료형 할당 가능
            System.out.println("byte 형 :"+a1);
            System.out.println("deouble 형 :"+a2);
        */
        double b1=100.5;
        byte b2;
        //b2=b1; // 작은자료형=큰자료형 불가능
        b2=(byte)b1; // 형변환 (casting)
        System.out.println(" double 형 :"+ b1);
        System.out.println(" byte 형 :"+ b2);
    }
}
```

```
//byte c1=300;
//System.out.println(" byte
형: "+c1)
char ch=(char)65;
char ch2=(char)66;
char ch3=(char)67;
char ch4=(char)97;
char ch5=(char)98;

System.out.println(""+ch);
System.out.println(""+ch2);
System.out.println(""+ch3);
System.out.println(""+ch4);
System.out.println(""+ch5)
    }
}
```

실습예제

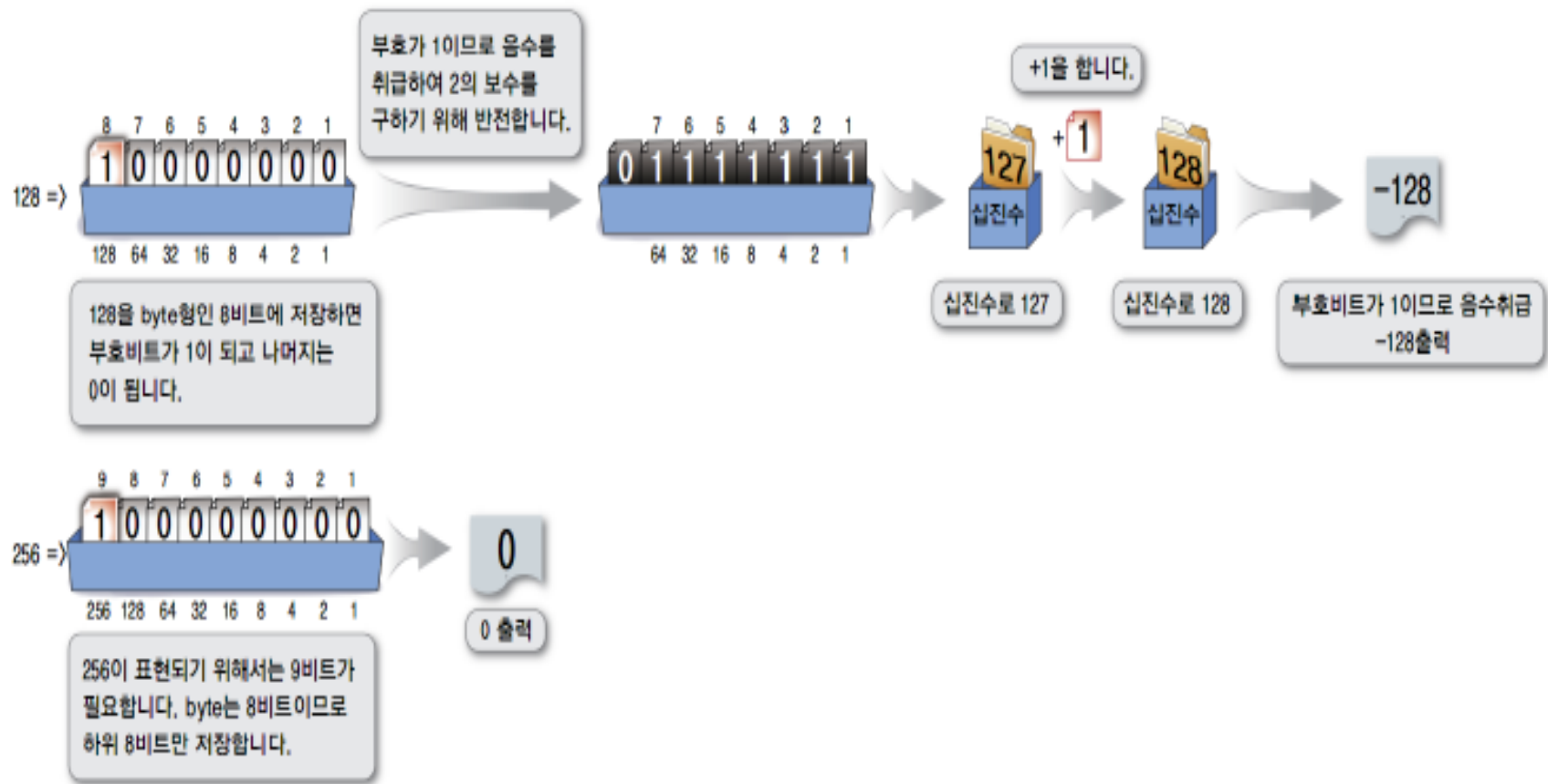
지정된형이나타낼수있는값의범위를벗어난값을배정하려면,
다음프로그램과같이강제로형변환을시켜주면된다.
강제로형변환을하는경우값의손실이발생하여예상하지못한결과가출력된다

```
public class ByteTypeTest{  
    public static void main(String args[]) {  
        bytea = (byte)128;  
        System.out.println("128을저장한byte 값은: " + a);  
        byteb = (byte)256;  
        System.out.println("256을저장한byte 값은: " + b);  
    }  
}
```

실행 결과

```
128을 저장한 byte 값은: -128  
256을 저장한 byte 값은: 0
```

- 전 페이지와 같은 결과가 출력된 이유
 - byte 변수에 128과 256을 저장한 결과



자바의 예약어(keyword)

❖ 예약어

- 자바 언어에서 의미를 가지고 사용되는 단어
- 변수 이름으로 사용할 경우 컴파일 에러 발생

분류	예약어
기본 타입	boolean, byte, char, short, int, long, float, double
접근 제한자	private, protected, public
클래스와 관련된 것	class, abstract, interface, extends, implements, enum
객체와 관련된 것	new, instanceof, this, super, null
메소드와 관련된 것	void, return
제어문과 관련된 것	if, else, switch, case, default, for, do, while, break, continue
논리값	true, false
예외 처리와 관련된 것	try, catch, finally, throw, throws
기타	package, import, synchronized, final, static

연산자(Operator)란?

- ▶ 연산자(Operator)
 - 어떠한 기능을 수행하는 기호(+,-,*,/ 등)
- ▶ 피연산자(Operand)
 - 연산자의 작업 대상(변수,상수,리터럴,수식)

$$a + b$$

연산자의 종류

▶ 단항 연산자 : + - (타입) ++ -- ~ !

▶ 이항 연산자 $\left\{ \begin{array}{l} \text{산술} : + - * / \% \ll \gg \ggg \\ \text{비교} : > < >= <= == != \\ \text{논리} : \&\& \parallel \& \wedge | \end{array} \right.$

▶ 삼항 연산자 : ? :

▶ 대입 연산자 : =

연산자(operator) 와 수식

연산의 주의사항

- 1, 자바의 연산자는 연산대상이 될 수 있는 데이터 형이 정해져 있다.
- 2, 범위를 넘는 연산에 대해서는 에러가 일어나지 않고 단지 연산결과에 이상한 쓰레기 값이 들어가게 된다.(자바에서의 연산에러는 오로지 0으로 나누거나 나머지를 구할 때 일어난다.)

연산자의 유형

- | | |
|---------|------------------------------------------------------|
| -단항 연산자 | <code>operator op</code>
<code>op operator</code> |
| -이항 연산자 | <code>op1 operator op2</code> |
| -삼항 연산자 | <code>expr ? op1 : op2</code> |

산술 연산자

● 산술 연산자

연산자	사용법	설명	비고
+	op1+op2	op1과 op2를 더한다.	단항 및 이항
-	op-op2	op1에서 op2를 뺀다.	단항 및 이항
*	op1*op2	op1과 op2를 곱한다.	이항
/	op1/op2	op1을 op2로 나눈다.	이항
%	op1%op2	op1을 op2로 나눈 나머지를 구한다.	이항
++	var++ ++var	var 값 1 증가. var 값을 증가시키기 전에 평가 var 값 1 증가. var 값을 증가시킨 다음 평가	단항 단항
--	var-- --var	var 값 1 감소. var 값을 감소시키기 전에 평가 var 값 1 감소. var 값을 감소시킨 다음 평가	단항 단항

```
public class IncreaseDecreaseOperatorExample {  
    public static void main(String[] args) {  
        int x = 10;  int y = 10;      int z;  
        System.out.println("-----");  
        x++;      ++x;      System.out.println("x=" + x);  
        System.out.println("-----");  
        y--;      --y;      System.out.println("y=" + y);  
        System.out.println("-----");  
        z = x++;      System.out.println("z=" + z);  
        System.out.println("x=" + x);  
        System.out.println("-----");  
        z = ++x;      System.out.println("z=" + z);  
        System.out.println("x=" + x);  
        System.out.println("-----");  
        z = ++x + y++;  
        System.out.println("z=" + z);  
        System.out.println("x=" + x);  
        System.out.println("y=" + y);  
    }  
}
```

산술 연산자의 우선순위

1. 괄호 안에 있는 연산자가 가장 먼저 계산된다.
여러 개의 괄호가 중첩되어 있을 경우 가장 안쪽의 괄호가 우선적으로 계산된다.
2. $*$ (곱셈) $/$ (몫 구하는 나눗셈) $%$ (나머지구하는 나눗셈) 연산자가 다음으로 계산된다.
수식안의 여러 개의 $*$ $/$ $%$ 가 있을 경우 왼쪽에서 오른쪽으로 연산이 실행된다. 이러한 3가지 연산자는 같은 우선순위를 갖는다.
3. $+$ (덧셈) $-$ (뺄셈) 연산자가 그 다음으로 계산된다.
4. 수식안의 여러 개의 덧셈 뺄셈이 있을 경우 왼쪽에서 오른쪽으로 연산이 실행된다. 이러한 2 가지 연산자는 같은 우선순위를 갖는다.

```
class OperTest {  
    public static void main(String[] args) {  
        System.out.println("증감연산자");  
        int a=2;  
        int b=a;  
        a++;  
        System.out.println("a="+a);  
        b=a++; //a-- ->증감연산자->변수뒤에 위치(postfix)  
        System.out.println("a="+a);/  
        System.out.println("b="+b);// /  
        int c=++a;//변수앞에 위치(prefix)  
        System.out.println("a="+a);  
        System.out.println("c="+c);//4  
    }  
}
```



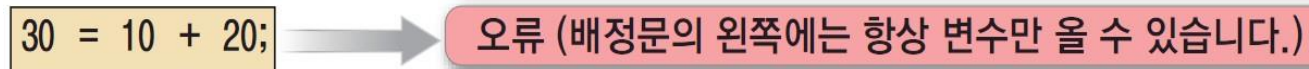
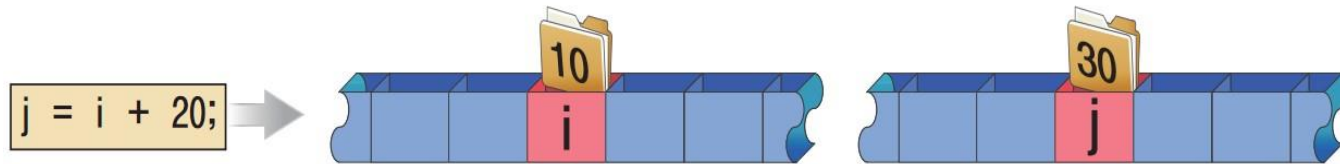
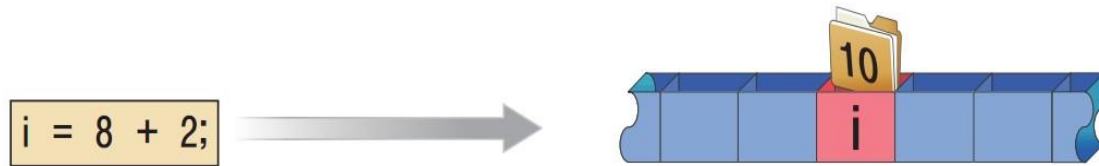
```

class VarTest {
    public static void main(String[] args) {
        byte k=21;          short k2;  k2=3000; //초기값 부여가능
        int a,b,c; //같은 형의 변수는 ,로 나열이 가능
        int d=23,e=500,f=2300; int $test;//_,$만 가능          //int 3test;
        byte k3=12;          short s2=23; //byte su=k3+s2;
        //자바의 연산결과->기본적인 숫자->int를 사용
        int su=k3+s2;        System.out.println("변수의 종류");
        System.out.println("k="+k+",su="+su);//출력할 문자형태+변수명
        float f2=3.5f;//소수점은 무조건 double형임
        System.out.println("f2="+f2);  boolean b1=true,b2=false;
        boolean b3=3<5;//수식을 통해서 결과값을 반환
        System.out.println("==논리형==");System.out.println("b1="+b1+",b2="+b2+",b3="+b3);
        System.out.println("==문자형==");    char munja='a'; char munja2='A'+1;//66
        char munja3=65; System.out.println("munja="+munja);//아스키코드값 , a
        System.out.println("munja2="+munja2); //B
        System.out.println("munja3="+munja3);//A
        System.out.println("형변환 munja3="+((int)munja3)); //추가=>형변환 munja3=65
        //문자열형=>기본형처럼 사용 //String a=new String("abc");
        String str1="abc"; String str2="자료형";
        String str3=str1+" "+str2;//문자열결합
        System.out.println("str1="+str1); System.out.println("str2="+str2); //str3=abc 자료형
        System.out.println("str3="+str3);
    }
}

```

배정(대입) 연산자

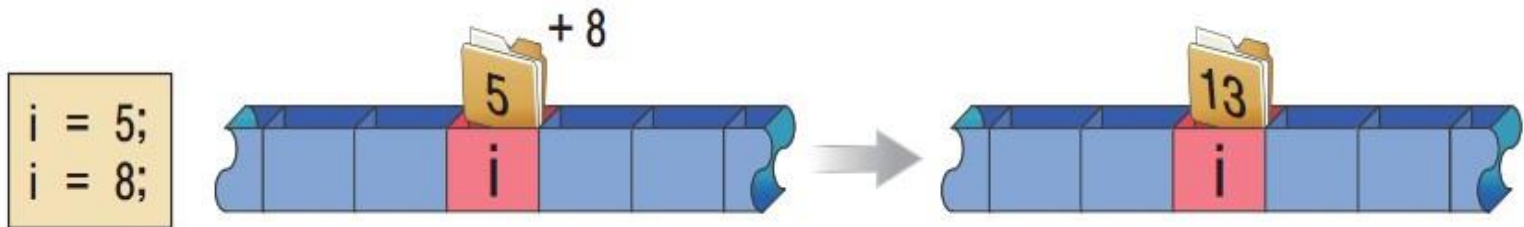
- 배정 연산자 “ = ”



배정 연산자

- 단축 배정 연산자

$i = i + 8 \quad \Rightarrow \quad i += 8$



배정 연산자

- 단축 배정 연산자

연산자	사용법	의미
<code>+=</code>	<code>op1 += op2</code>	<code>op1 = op1 + op2</code>
<code>-=</code>	<code>op1 -= op2</code>	<code>op1 = op1 - op2</code>
<code>*=</code>	<code>op1 *= op2</code>	<code>op1 = op1 * op2</code>
<code>/=</code>	<code>op1 /= op2</code>	<code>op1 = op1 / op2</code>
<code>%=</code>	<code>op1 %= op2</code>	<code>op1 = op1 % op2</code>
<code>&=</code>	<code>op1 &= op2</code>	<code>op1 = op1 & op2</code>
<code> =</code>	<code>op1 = op2</code>	<code>op1 = op1 op2</code>
<code>^=</code>	<code>op1 ^= op2</code>	<code>op1 = op1 ^ op2</code>
<code><<=</code>	<code>op1 <<= op2</code>	<code>op1 = op1 << op2</code>
<code>>>=</code>	<code>op1 >>= op2</code>	<code>op1 = op1 >> op2</code>
<code>>>>=</code>	<code>op1 >>>= —p2</code>	<code>op1 = op1 >>> op2</code>

배정 연산자

- 실습예제

➤ 다음 프로그램은 배정 연산자와 단축 배정 연산자를 사용하는 예이다.

```
public class AssignOPTest {  
    public static void main(String args[]) {  
        int a = 10;  
        System.out.println("a = "+ a);  
        a += 4;  
        System.out.println("a += 4의 결과 " + a);  
        a -= 4;  
        System.out.println("a -= 4의 결과 " + a);  
        a *= 4;  
        System.out.println("a *= 4의 결과 " + a);  
        a /= 4;  
        System.out.println("a /= 4의 결과 " + a);  
        a %= 4;  
        System.out.println("a %= 4의 결과 " + a);  
    }  
}
```

단축 배정 연산자 사용

단축 배정 연산자 사용

단축 배정 연산자 사용

단축 배정 연산자 사용

단축 배정 연산자 사용

배정 (대입) 연산자 예제

```
class OperTest2 {  
    public static void main(String[] arg){  
        String str= "배정연산자 예제 입니다." ;  
        int x=8,y=4;  
        System.out.println(str);  
        System.out.println("x += y =" + (x+=y));  
        System.out.println("x -= y =" + (x-=y));  
        System.out.println("x *= y =" + (x*=y));  
        System.out.println("x /= y =" + (x/=y));  
        System.out.println("x %= y =" + (x%=y));  
    }  
}
```

관계(비교) 연산자

구분	연산자	의미	수학적 기호	자바 표현	자바 결과
항등 연산자	==	좌측이 우측과 같다	=	2==4	false
	!=	좌측이 우측과 같지 않다	≠	2!=4	true
비교 연산자	>	좌측이 우측보다 크다	>	3>2	true
	>=	좌측이 우측보다 크거나 같다	≥	2>=3	false
	<	좌측이 우측보다 작다	<	4<5	true
	<=	좌측이 우측보다 작거나 같다	≤	4<=4	true

관계 연산자

● 실습예제

- ▶ 다음 프로그램은 다양한 관계 연산자를 사용하고 있다. 관계 연산자는 두 개의 오퍼랜드를 비교하여 결과로 true 또는 false를 반환한다.

```
public class RelationalOPTest {  
    public static void main(String args[]) {  
        byte a = 20; double d = 3.14;  
        boolean fag;  
        fag = a > d;          // 비교 연산자 사용  
        System.out.println("a가 d보다 큰가? " + fag);  
        fag = a == 20.0f;     // 비교 연산자 사용  
        System.out.println("a가 20.0f와 같은가? " + fag);  
        fag = 10 != 10;       // 비교 연산자 사용  
        System.out.println("10이 10과 같지 않은가? " + fag);  
        fag = 10 <= 20;       // 비교 연산자 사용  
        System.out.println("10이 20보다 작거나 같은가? " + fag);  
        System.out.println("10이 20보다 작은가? " + (10 < 20));  
        System.out.println("10이 20보다 크거나 같은가? " + (10 >= 20));  
    }  
}
```


논리연산자 - && ||

-피연산자가 반드시 boolean이어야 하며 연산결과도 boolean이다.

&&가 || 보다 우선순위가 높다. 같이 사용되는 경우 괄호를 사용하자

▶ OR연산자(||) : 피연산자 중 어느 한 쪽이 true이면 true이다.

▶ AND연산자(&&) : 피연산자 양 쪽 모두 true이면 true이다.

x	y	x y	x && y
true	true	true	true
true	false	true	false
false	true	true	false
false	false	false	false

논리연산자 - && ||

```
int i = 7;
```

```
i > 3 && i < 5
```

```
i > 3 || i < 0
```

x	y	x y	x && y
true	true	true	true
true	false	true	false
false	true	true	false
false	false	false	false

```
char x = 'j';
```

```
x >= 'a' && x <= 'z'
```

```
(x >= 'a' && x <= 'z') || (x >= 'A' && x <= 'Z')
```

```

class LogicalTest {
    public static void main(String[] args) {
        System.out.println("관계,논리연산자");
        boolean b1,b2,b3; boolean b4=true;
        int i1=3, i2=5; int i3=7; int i4;
        System.out.println("(i1>i2)==>" + (i1>i2)); //3>5->f
        System.out.println("(i2<i3)==>" + (i2<i3)); //5<7->t //3>5
        b1=(i1>i2) && (i2<i3); System.out.println("(i1>i2) && (i2<i3)=" + b1); //false
        b2=(i1>i2) || (i2<i3);
        System.out.println("(i1>i2) || (i2<i3)=" + b2); //true
        b3=!b4;
        System.out.println("b3=" + !b4); //false
        b1=(i1>i2) & (i2<i3);
        System.out.println("(i1>i2) & (i2<i3)=" + b1);
        b2=(i1>i2) | (i2<i3);
        System.out.println("(i1>i2) | (i2<i3)=" + b2);

        b3=(i1>i2) ^ (i2<i3); //f ^ t==>true
        System.out.println("(i1>i2) ^ (i2<i3)=" + b3);
    }
}

```

3항 연산자

- 자바는 3개의 오퍼랜드를 가진 3항 연산자 “?:” 를 제공

형식

수식1 ? 수식2 : 수식3

- 3항 연산자는 수식1을 평가하여 **true**인지 **false**인지를 판별하여 **true**이면 수식2가 평가되고, **false**이면 수식3이 평가되어 그 결과가 **flag**로 반환

`flag = count > 0 ? 0 : 1;`

3항 연산자

- 실습예제

- 숫자가 짝수인지 홀수인지를 구분하는 프로그램

```
public class TernaryOPTest {  
    public static void main(String args[]) {  
        int i=7;  
        boolean j;  
        System.out.println(i + "이 짝수입니까?");  
        j = (i % 2 == 0) ? true : false; 3항 연산자의 사용  
        System.out.println(j);  
    }  
}
```

비트 연산자

연산자	명칭	설명
&	비트 논리곱(AND)	두 피연산자의 대응비트 두 비트가 모두 1이면 결과 비트는 1이 된다.
	비트 논리합(OR)	두 피연산자의 대응비트 두 비트 중 적어도 한 비트가 1이면 결과 비트는 1이 된다.
^	비트 배타적논리합(XOR)	두 피연산자의 대응비트 두 비트 중 1비트가 1이면 (두 비트가 서로 다른 비트일 때) 결과 비트는 1이 된다.
<<	왼쪽 이동(left shift)	두 번째 피 연산자의 지정개수만큼 첫번째 피연산자의 비트들을 왼쪽으로 이동(시프트)한다. 오른쪽은 0으로 채운다.
>>	오른쪽 이동(right shift)	두 번째 피 연산자의 지정개수만큼 첫번째 피연산자의 비트들을 오른쪽으로 이동한다. 왼쪽에 값 채우는 방법은 시스템에 따라 다르다.
~	1의 보수(one's complement)	모든 0비트는 1이 되고 모든 1비트는 0이 된다.

비트 연산자

대응 피연산자 비트		비트 AND	비트 OR	비트 XOR
X	Y	$X \& Y$	$X Y$	$X \wedge Y$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

피연산자 비트 X	비트 NOT($\sim X$)
0	1
1	0

10

6[illegible]

[illegible]

10 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0

6 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0

[illegible]

비트 OR 연산

10	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0

비트 XOR 연산

10

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

6

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

^

0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

비트 NOT 연산

10

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

~

1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

```

class BitOperator {
    public static void main(String[] args)      {
        int num1=5;      /* 00000000 00000000 00000000 00000101 */
        int num2=3;      /* 00000000 00000000 00000000 00000011 */
        int num3=-1;     /* 11111111 11111111 11111111 11111111 */
        System.out.println(num1 & num2);
        System.out.println(num1 | num2);
        System.out.println(num1 ^ num2);
        System.out.println(~num3);
    }
}

public class Ex07 {
    public static void main(String[] args) {
        short x = 10, y = 6 ;
        System.out.println( " x & y  : " + (x & y) );
        System.out.println( " x | y  : " + (x | y) );
        System.out.println( " x ^ y  : " + (x ^ y) );
        System.out.println( " ~x    : " + (~x) );
    }
}

```

시프트 연산자

연산자	예	의미
\ll (왼쪽 시프트 연산자)	$a \ll 1$	a의 값을 왼쪽으로 1비트만큼 이동한다 $X \ll n$ 은 $x * 2^n$
\gg (오른쪽 시프트 연산자)	$a \gg 1$	a의 값을 오른쪽으로 1비트만큼 이동한다. $X \gg n$ 은 $x / 2^n$

\ll 연산자는 부호에 상관없이 이동시키지만

\gg 연산자는 음수의 경우 부호를 유지시키기 위해
빈자리를 1로 채운다

\ggg 연산자는 부호에 상관없이 빈자리를 0으로 채운다

왼쪽 시프트 연산자

15

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

60

0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

오른쪽 시프트 연산자

15 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1

3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1

쉬프트연산자 - $<<$, $>>$, $>>>$

- 2^n 으로 곱하거나 나눈 결과를 반환한다.
- 곱셈, 나눗셈보다 빠르다.

$x << n$ 은 $x * 2^n$ 과 같다.

$x >> n$ 은 $x / 2^n$ 과 같다.

$8 << 2$ 는 $8 * 2^2$ 과 같다.

$8 >> 2$ 는 $8 / 2^2$ 과 같다.

부호비트와 시프트 연산

부호비트와 왼쪽 시프트 연산(Left - Shift)

왼쪽으로 시프트 연산을 수행하는 경우 값은 2배씩 증가하게 됩니다. 이때 아래의 그림과 같이 최상위 비트와 그 다음 비트가 서로 다른 값인 경우 플로우(Overflow)가 발생합니다.



위의 그림이 왼쪽 시프트 연산을 하기 전 값은 79 이지만 "79 << 1" 을 연산한 결과는 "10011110"이 되며 10진수로 변환한 결과 값은 -98이 됩니다. 최상위 비트가 부호비트가 아니라면 연산 결과는 79 의 2 배인 158 이지만 최상위비트가 변경되어 오버플로우가 발생하였기 때문에 부호가 바뀌고 결과 값이 원하는 값과 다르게 나온 것입니다.

오버플로우는 부호비트가 1 에서 0 으로 변경된 경우에도 발생하며 이 때에는 음수가 양수로 바뀌게 됩니다. 그러므로 부호가 있는 데이터의 왼쪽 시프트 연산 시에는 오버플로우가 발생할 수 있는 것을 고려해주어야 합니다.

부호비트와 오른쪽 시프트 연산(Right - Shift)

오른쪽으로 시프트 연산을 하는 경우는 2의 제곱으로 나누기 연산을 하는 것입니다. 이때 비트들이 오른쪽으로 이동하면서 생긴 공간은 부호비트의 값으로 채워집니다. 아래의 그림에서는 부호비트가 1이기 때문에 1로 채워지지만 만약 부호비트값이 0이라면 0으로 채워집니다.



예를 들어 위의 그림과 같이 "1011 1111" 값을 오른쪽으로 한번 이동하면 왼쪽에 생긴 빈 공간은 최상위 비트인 1 값으로 채워지는 것입니다. 만약 "-160 >> 4" 를 연산하면 -160의 이진수 값인 "1010 0000"가 오른쪽으로 4비트만큼 이동하고, 왼쪽에 생긴 네개의 빈 공간은 최상위 비트인 1로 채워지므로 연산 결과는 "1111 1010"가 됩니다.

시프트 연산시 주의 사항

시프트 연산자의 우선순위 산술 연산자보다 낮습니다. 그렇기 때문에 다음과 같은 문제가 발생할 수 있으므로 시프트 연산시 괄호를 사용하여 연산하는 습관을 들이는 것이 좋습니다.

```
int data = 10 << 3 - 2 >> 1;  
          = 10 << 1 >> 1  
          = 20 >> 1  
          = 10      // 원하지 않는 결과 값 발생
```

위의 예제는 "10 << 3" 을 연산한 결과에서 "2 >> 1" 을 연산한 결과를 뺄 목적으로 수식을 작성한 것이지만 연산자 우선순위에 의해 "3 - 2" 가 먼저 연산되어 원하지 않는 연산 결과가 발생하였습니다.

올바른 연산 결과를 얻기 위해서는 아래와 같이 시프트 연산 부분을 괄호로 묶어주면 됩니다.

```
int data = (10 << 3) - (2 >> 1);  
          = 80 - (2 >> 1)  
          = 80 - 1  
          = 79      // 올바른 결과 값 발생
```

시프트 연산자

```
class BitShiftOp {  
    public static void main(String[] args)    {  
        System.out.println(2 << 1);  
        System.out.println(2 << 2);  
        System.out.println(2 << 3);  
        System.out.println(8 >> 1);  
        System.out.println(8 >> 2);  
        System.out.println(8 >> 3);  
  
        System.out.println(-8 >> 1);  
        System.out.println(-8 >> 2);  
        System.out.println(-8 >> 3);  
  
        System.out.println(-8 >>> 1);  
    }  
}
```

```
public class ShiftTest {  
    public static void main(String[] ar) {  
        int x = 1 << 3;  
        int y = 8 >> 3;  
        System.out.println("x = " + x);  
        System.out.println("y = " + y);  
    }  
}
```

```
public class BitTest {  
    public static void main(String[] ar) {  
        int a = 4;  
        int b = 7;  
        System.out.println("& = " + (a & b)); // 모두 참일 때 참  
        System.out.println("| = " + (a | b)); // 모두 거짓일 때 거짓  
        System.out.println("^ = " + (a ^ b)); // 양쪽이 틀리면 참  
    }  
}
```

연산자의 우선순위

종 류	연산방향	연 산 자	우선순위
단항 연산자	←	++ -- + - ~ ! (타입)	높음
산술 연산자	→	* / %	
	→	+ -	
	→	<< >> >>>	
비교 연산자	→	< > <= >= instanceof	
	→	== !=	
논리 연산자	→	&	
	→	^	
	→		
	→	&&	
	→		
삼항 연산자	→	?:	
대입 연산자	←	= *= /= %= += -= <<= >>= >>>= &= ^= =	낮음

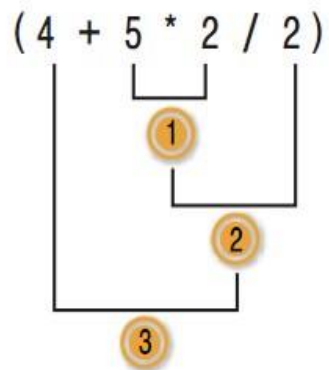
연산자의 종류와 우선순위

연산자 우선순위

- 연산자의 우선순위

$(4 + 5 * 2 / 2)$ 의 우선순위

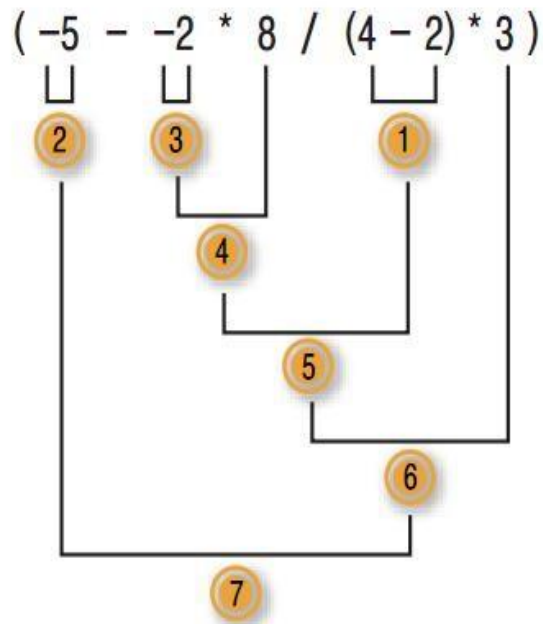
예 : $(4 + 5 * 2 / 2)$ 의 우선순위



연산자 우선순위

- 연산자의 우선순위

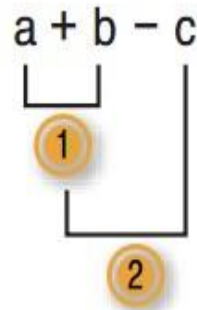
$(-5 - -2 * 8 \% (4 - 2) * 3)$



연산자 우선순위

- 연산자의 우선순위

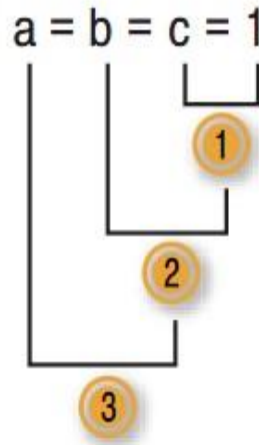
$a + b - c$ (우선순위가 같을 때 사칙연산의 경우는 왼쪽부터 계산)



연산자 우선순위

- 연산자의 우선순위

$a = b = c = 1$ (우선순위가 같을 때 배정 연산은 오른쪽부터 계산)



연산자의 우선순위

- 괄호의 우선순위가 제일 높다.
- 산술 > 비교 > 논리 > 대입
- 단항 > 이항 > 삼항
- 연산자의 연산 진행방향은 왼쪽에서 오른쪽(\rightarrow)이다.
단, 단항, 대입 연산자만 오른쪽에서 왼쪽(\leftarrow)이다.

$$3 * 4 * 5$$

$$x = y = 3$$

연산자의 우선순위

- 상식적으로 생각하라. 우리는 이미 다 알고 있다.

ex1) $-x + 3$ 단항 > 이항

ex2) $x + 3 * y$ 곱셈, 나눗셈 > 덧셈, 뺄셈

ex3) $x + 3 > y - 2$ 산술 > 비교

ex4) $x > 3 \&\& x < 5$ 비교 > 논리

ex5) `int result = x + y * 3;` 항상 대입은 맨 끝에

연산자의 우선순위

- 그러나 몇 가지 주의해야 할 것이 있다.

1. $<<$, $>>$, $>>>$ 는 덧셈연산자보다 우선순위가 낮다.

ex5) $x << 2 + 1$ $x << (2 + 1)$ 과 같다.

2. \parallel , $|$ (OR)는 $\&\&$, $\&$ (AND)보다 우선순위가 낮다.

ex6) $x < -1 \parallel x > 3 \&\& x < 5$

$x < -1 \parallel (x > 3 \&\& x < 5)$ 와 같다.

산술(수치), 증감, 감소 연산자 예제

```
class OperTest1 {  
    public static void main(String[] arg){  
        String str= "산술, 증감, 감소, 연산자 예제" ;  
        int x,y;  
        x = 10; y = 5;  
        System.out.println(str);  
        System.out.println("x + y = "+(x + y));  
        System.out.println("x - y = "+(x - y));  
        System.out.println("x * y = "+(x * y));  
        System.out.println("x / y = "+(x / y));  
        System.out.println("x % y = "+(x % y));  
        System.out.println("x++ = "+(x++));  
        System.out.println("y++ = "+(y++));  
        x=10; y=5;  
        System.out.println("++x = "+(++x));  
        System.out.println("++y = "+(++y));  
    }  
}
```

이항연산자의 특징(1/7)

이항연산자는 연산을 수행하기 전에 피연산자의 타입을 일치시킨다.

- int보다 크기가 작은 타입은 int로 변환한다.

(byte, char, short \rightarrow int)

- 피연산자 중 표현범위가 큰 타입으로 형변환 한다.

byte + short \rightarrow int + int \rightarrow int

char + int \rightarrow int + int \rightarrow int

float + int \rightarrow float + float \rightarrow float

long + float \rightarrow float + float \rightarrow float

float + double \rightarrow double + double \rightarrow double

이항연산자의 특징(2/7)

```
byte a = 10;
```

```
byte b = 20;
```

```
byte c = a + b;
```

byte + byte → int + int → int

```
byte c = (byte)a + b;    // 에러
```

```
byte c = (byte)(a + b);  // OK
```

이항연산자의 특징(3/7)

```
int a = 1000000; // 1,000,000
```

```
int b = 2000000; // 2,000,000
```

```
long c = a * b; // c는 2,000,000,000,000 ?  
// c는 -1454759936 !!!
```

$\text{int} * \text{int} \rightarrow \text{int}$

```
long c = (long)a * b; // c는 2,000,000,000,000 !
```

$\text{long} * \text{int} \rightarrow \text{long} * \text{long} \rightarrow \text{long}$

이항연산자의 특징(4/7)

`long a = 1000000 * 1000000; // a는 -727,379,968`

`long b = 1000000 * 1000000L; // b는 1,000,000,000,000`

`int c = 1000000 * 1000000 / 1000000; // c는 -727`

`int d = 1000000 / 1000000 * 1000000; // d는 1,000,000`

이항연산자의 특징(5/7)

```
char c1 = 'a';
```

```
char c2 = c1 + 1; // 에러
```

```
char c2 = (char)(c1 + 1); // OK
```

```
char c2 = ++c1; // OK
```

```
int i = 'B' - 'A';
```

```
int i = '2' - '0';
```

문자	코드
...	...
0	48
1	49
2	50
...	...
A	65
B	66
C	67
...	...
a	97
b	98
c	99
...	...

이항연산자의 특징(6/7)

```
float pi = 3.141592f;
```

```
float shortPi = (int)(pi * 1000) / 1000f;
```

```
(int)(3.141592f * 1000) / 1000f;
```

```
(int)(3141.592f) / 1000f;
```

```
3141 / 1000f;
```

```
3141.0f / 1000f
```

```
3.141f
```

이항연산자의 특징(7/7)

* Math.round() : 소수점 첫째자리에서 반올림한 값을 반환

```
float pi = 3.141592f;
```

```
float shortPi = Math.round(pi * 1000) / 1000f;
```

```
Math.round(3.141592f * 1000) / 1000f;
```

```
Math.round(3141.592f) / 1000f;
```

```
3142 / 1000f;
```

```
3142.0f / 1000f
```

```
3.142f
```

관계 (비교) 연산자 예제

```
class OperTest3 {  
    public static void main(String[] arg){  
        String str= "관계연산자 예제 입니다." ;  
        int x=10;  
        int y=5;  
        System.out.println(str);  
        System.out.println("x = "+x+" y = "+y);  
        System.out.println("[x>=y] = "+(x>=y));  
        System.out.println("[x<y]   = "+(x<y));  
        System.out.println("[x<=y] = "+(x<=y));  
        System.out.println("[x==y] = "+(x==y));  
        System.out.println("[x!=y] = "+(x!=y));  
    }  
}
```

논리연산자 예제

```
class OperTest4 {  
    public static void main(String[] arg){  
        String str= “논리연산자 예제 입니다.” ;  
        boolean x,y;  
        x=true;  
        y=false;  
        System.out.println(str);  
        System.out.println("x & y =" +(x&y));  
        System.out.println("x && y =" +(x&&y));  
        System.out.println("x | y =" +(x|y));  
        System.out.println("x || y =" +(x||y));  
        System.out.println("!x =" +(!x));  
        System.out.println("!y =" +(!y));  
    }  
}
```


3항 연산자 예제

```
class Opertest4 {  
    public static void main(String[] arg){  
        String str= "3항 연산자 예제입니다." ;  
        int x=10;  
        int y;  
        y=x<9?5:20;  
        System.out.println(str);  
        System.out.println("y의 결과는 "+y);  
    }  
}
```

연습문제

1. 바구니 갯수를 구하는 식

```
int numOfApples = 123; // 사과의 갯수
```

```
int sizeOfBucket = 10;
```

// 바구니의 크기(바구니에 담을 수 있는 사과의 갯수)

```
int numOfBucket =
```

2. 다음 계산결과는 ? 왜 ?

```
int num = 456;
```

```
System.out.println(num/100*100);
```

3. 다음 계산결과는 ? 왜 ?

```
int num = 24;
```

```
System.out.println(10 - num%10);
```