

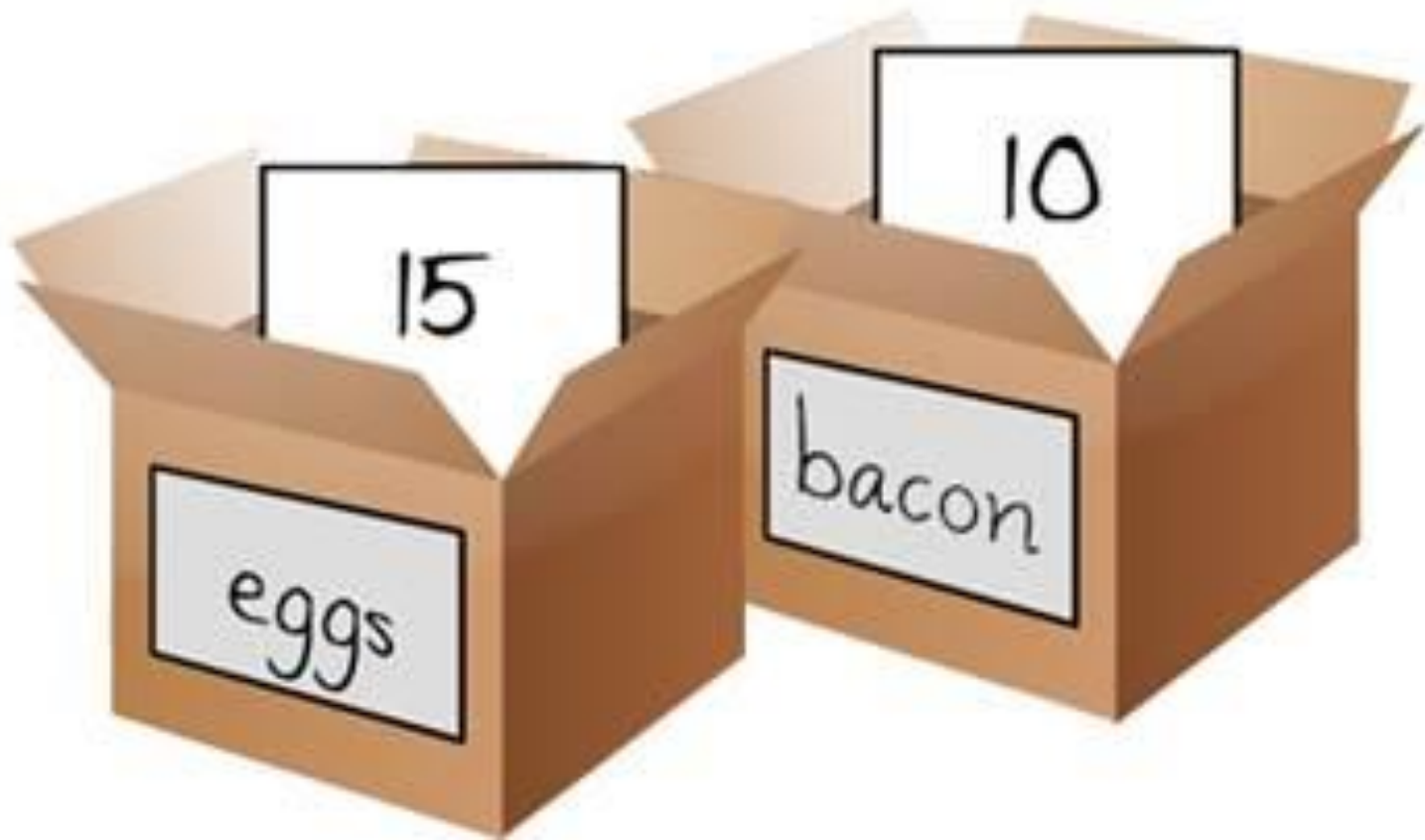
```
for object to mirror_mod.mirror_object
operation == "MIRROR_X":
mirror_mod.use_x = True
mirror_mod.use_y = False
mirror_mod.use_z = False
operation == "MIRROR_Y":
mirror_mod.use_x = False
mirror_mod.use_y = True
mirror_mod.use_z = False
operation == "MIRROR_Z":
mirror_mod.use_x = False
mirror_mod.use_y = False
mirror_mod.use_z = True
```

```
@selection at the end -add
mirror_ob.select= 1
modifier_ob.select=1
context.scene.objects.active
("Selected" + str(modifier_ob.name))
mirror_ob.select = 0
= bpy.context.selected_object
data.objects[one.name].select
print("please select exactly one mirror")
```

```
-- OPERATOR CLASSES -----
bpy.types.Operator):
X mirror to the selected
object.mirror_mirror_x"
mirror X"
```

Java 기초

변수



변수(Variable)

- 변수란?

변수(Variable)

- 변수란?
 - 단 하나의 값 만을 지정할 수 있는 공간
 - 변수는 단 하나의 값 만을 저장할 수 있으며 변수의 값은 바뀔 수 있다.
 - 하나의 변수에 여러 번 다른 값을 저장할 시 맨 마지막에 저장한 값이 변수에 저장이 된다.
 - 이때 기존에 있던 변수의 값은 소멸된다.
 - 변수는 각각의 타입을 가지며 해당 타입에 맞는 자료형의 값 만을 가질 수 있다.

변수(Variable)

- 변수의 선언



- 처음 선언 시 반드시 변수 타입이 붙고 변수 명이 들어간다.
- 변수 값 앞에는 (=)를 선언해주는데 이 뜻은 변수 number라는 곳에 변수 값을 넣겠다 라는 뜻이다.
- 이 (=)연산자를 대입 연산자라고 하는데 해당 변수에 해당 값을 집어넣는다 라는 뜻이 된다.
- 기본적으로는 변수는 이런 식으로 쓰이고 여러가지 변형을 통해 변수를 다르게 선언이 가능하다.

변수(Variable)

- 변수의 사용 방법
 - 기존에 사용했던 값 대신에 변수 명을 그대로 넣어주면 된다.
 - 가령 예를 들어 `System.out.println(3);` 이라고 되어있는 로직에 3 대신 변수 `int a = 3;` 을 선언하고 기존의 로직에 `System.out.println(a);` 라고 선언이 가능하다.
 - 이런 식으로 선언함으로써 3이라는 숫자를 재사용이 가능하며 값을 바꿀 경우 다른 곳에서 바꾼 값으로 사용이 가능하다.
 - 즉 두 번 해야 될 일을 한번에 선언이 가능하다.

변수(Variable)

변수 선언/초기화/사용 예제

```
public class VariableEx1 {  
  
    Run | Debug  
    public static void main(String[] args) {  
        // int라는 타입은 숫자를 받을 때 사용하는 변수 타입이다.  
        // int는 상당히 많이 쓰이는 타입 중 하나이다.  
        // 기존의 로직이라면 sysout에 3을 넣어 사용이 가능하다.  
        //System.out.println(3);  
        // 하지만 위와 같은 로직은 3이라고 하는 숫자를 재사용 할 수 없으며  
        // 값에 의한 동기화가 일어나지 않는다.  
        int a = 3;  
        // 변수를 쓸 때는 해당 값이 들어갈 위치에 변수 명만을 적어줌으로써  
        // 이 값을 사용하겠다 라고 선언할 수 있다.  
        System.out.println(a);  
        // 만약 3이라는 값이 여러군데 쓰인다면 굳이 우리가 그 값을 여러군데에서  
        // 사용하지 않고 단순히 a란 값을 다시 선언함으로써 값을 재사용이 가능하다.  
        System.out.println(a);  
    }  
}
```

변수(Variable)

변수 선언/초기화/사용 예제

```
public class VariableEx2 {  
  
    Run | Debug  
    public static void main(String[] args) {  
        // 변수를 선언하는데 맨 처음 초기화를 시키지 않을 경우  
        // 아래와 같이 나중에 초기화가 가능하다.  
        int a;  
        // 단 초기화전에 변수를 사용할 경우 에러가 나므로 주의할 것  
        // System.out.println(a); // 에러  
        a = 3;  
        System.out.println(a); // 3  
    }  
}
```

변수(Variable)

변수 선언/초기화/사용 예제

```
public class VariableEx3 {  
  
    Run | Debug  
    public static void main(String[] args) {  
        // 변수의 값이 중간에서 갱신될 경우 변수 = 값 을 통해  
        // 변수의 값을 중간에 바꿀 수 있다.  
        // 이때 앞에 변수 타입은 들어가지 않으며 이를 작성할 경우  
        // 에러가 발생한다.  
        int a = 3;  
        System.out.println(a);  
        // int a = 4; // 에러  
        a = 4;  
        System.out.println(a);  
        // 변수는 복수개를 선언할 수가 있으며  
        // 다른 변수를 선언하기 위해서는 반드시 변수 이름은 달라야 한다.  
        int b = 5;  
        System.out.println(b);  
    }  
}
```


변수(Variable)

변수 선언/초기화/사용 예제

```
public class VariableEx4 {  
  
    Run | Debug  
    public static void main(String[] args) {  
        // 변수는 한줄에 하나씩 선언할 수 있지만, 한줄에  
        // 다수개의 변수를 선언할 수 도 있다.  
        // 복수에 변수를 선언하기 위해서는 변수타입 뒤에 각기 다른 변수 명을  
        // 열거하여 사용하는데 이때 두 변수의 변수 타입은 동일하게 선언되며  
        // 다르게 타입을 선언이 불가능하다.  
        // int a = 3, long b = 4; // 에러  
        int a = 3, b = 4;  
        System.out.println(a);  
        System.out.println(b);  
        // 또한 저 방식에서 변수명을 먼저 선언하고 해당 값을 나중에 선언할 수도 있다.  
        int c, d;  
        c = 5;  
        d = 6;  
        System.out.println(c);  
        System.out.println(d);  
        // 한 줄에 다수 변수 선언 시 선언되는 변수의 갯수에는 제한이 없다.  
        // int e, f, g, h;  
    }  
}
```

변수(Variable)

- 변수 명명 규칙

- 변수의 이름, 메서드의 이름, 클래스의 이름 등 모든 이름을 지을 때 반드시 지켜야 할 규칙이 있으며 다음과 같다.

1. 대소문자가 구분되며 길이에 제한이 없다.

-AA와 aa는 서로 다른것으로 간주된다.

2. 예약어를 사용해서는 안된다.

-true는 예약어라 사용이 안되지만 True는 가능하다.

3. 숫자로 시작해서는 안된다.

-7up(x), top10 (o)

4. 특수문자는 '_' 또는 '\$'만을 허용한다

- ex) up_down, letter#

변수(Variable)

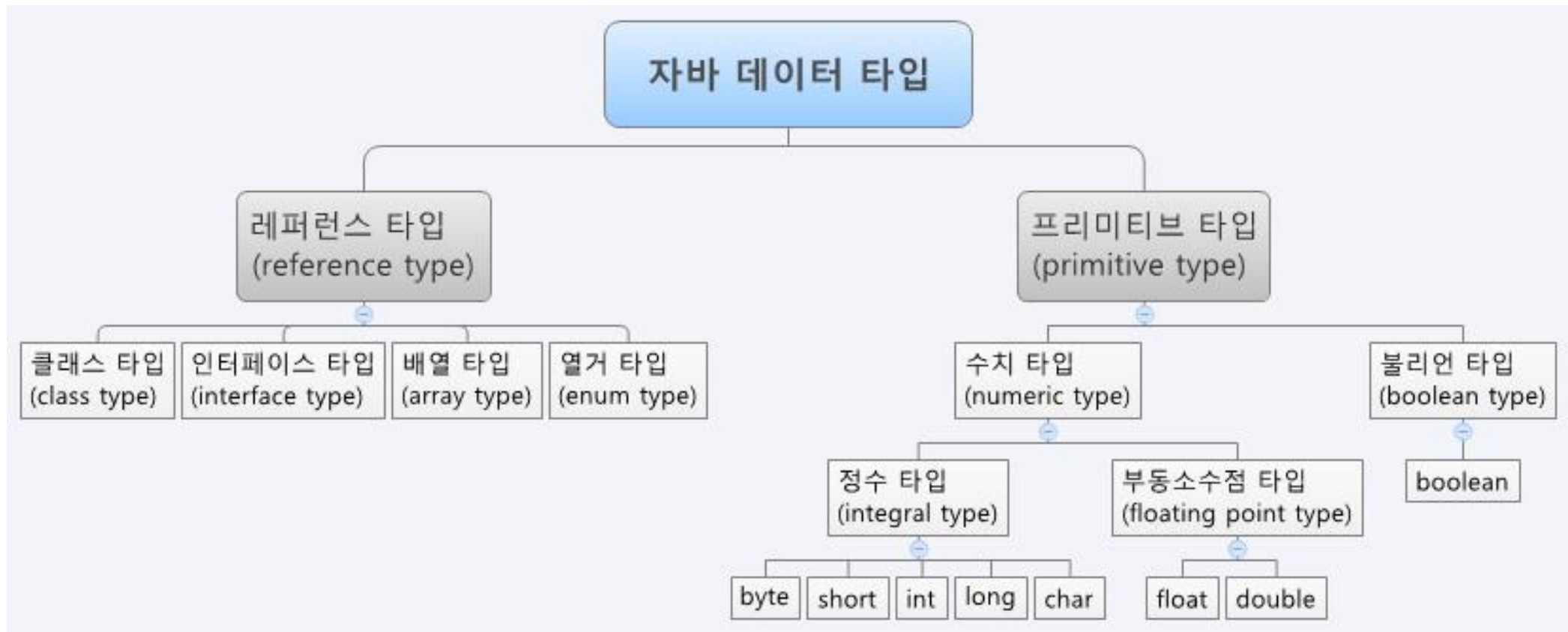
- 변수 및 클래스 명명 권장 규칙
 - 필수 규칙은 아니지만 일종의 권장하는 규칙들이 존재한다.
 - 업무에서 쉽게 메서드, 변수, 클래스를 구분하기 위한 용도로 사용한다.

클래스 명명 규칙 : Pascal Casing

- 소문자를 기본으로 사용하되, 구분되는 단어를 대문자로 연결. 첫 단어의 첫 글자는 대문자를 사용
- ex) MainClass, InsertServiceImpl

변수 명명 규칙 : Camel Casing

- 소문자를 기본으로 사용하되, 구분되는 단어를 대문자로 연결. 첫 단어의 첫 글자는 소문자를 사용
- ex) appleTree, maxCount

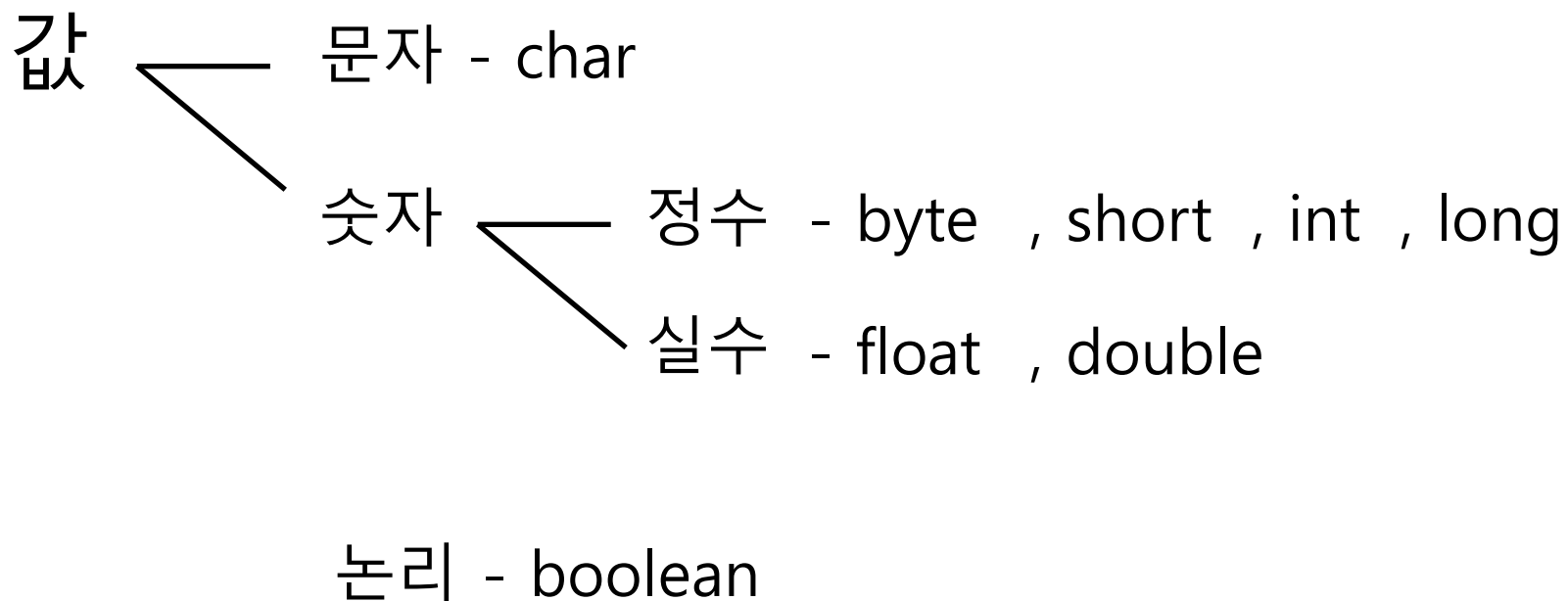


변수의 타입

변수의 타입

변수의 타입

기본형 변수의 타입과 종류



변수의 타입

기본형 변수 타입 크기

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

변수(Variable)

- 정수형

```
byte b = 1; short s = 2;  
int finger = 3; long big = 100L
```

- 정수형은 실제 정수를 받는 변수형이다.
- 정수형에는 모두 4개의 자료형이 있다.(byte, short, int, long)
- 각각의 크기는 byte가 가장 작고 long이 가장 크다.
Ex) byte(1byte) < short(2byte) < int(4byte) < long(8byte)
- 10진수 외에도 16진수 또는 8진수로도 변수에 저장 가능하다.
- 정수형 변수들은 자신이 저장할 수 있는 범위를 넘어서면 의도되지 않은 수가 나온다.
- Long형은 원래 인자 뒤에 L값을 붙여줘야 되지만 안 붙여줘도 상관이 없다.
- 정수형의 기본형은 **int** 형이다

변수(Variable)

정수형 예제

```
// byte 선언
byte a = 1;
System.out.println(a); // 1
// short 선언
short b = 2;
System.out.println(b); // 2
// int 선언
int c = 3;
System.out.println(c); // 3
// long 선언
long d = 4;
System.out.println(d); // 4
```


변수(Variable)

정수형 예제

```
// 8진수 입력
int e1 = 010;
System.out.println(e1); // 8
// 16진수 입력
int e2 = 0x10;
System.out.println(e2); // 16

// byte 최대 크기 : 2^7-1
byte f = (byte)128;
System.out.println(f); // -128
// short 최대 크기 : 2^15-1
short g = (short)32768;
System.out.println(g); // -32768
// int 최대 크기 : 2^31-1
int h = (int)2147483648L;
System.out.println(h); // -2147483648
```

변수(Variable)

정수형 예제

```
// printf 에서의 변수 사용법
int i = 7;
int j = 8;
int k = 9;
System.out.printf("%d\n", i); // 7
System.out.printf("%d , %d \n", j, k); // 8 , 9
System.out.printf("%d + %d = %d ", i, k, i + k); // 8 + 9 = 17

int l = 12345;
int m = 2101010101;
System.out.printf("%d*\n", l); // 12345*
System.out.printf("%10d*\n", l); // 12345*
System.out.printf("%-10d*\n", l); // 12345 *
System.out.printf("잔액 : %,d원\n", m); // 잔액 : 2,101,010,101원
```

변수(Variable)

- 실수형

float pi = 3.14f, double = 1.234567d

- 실수형은 실수를 저장하는데 사용된다.
- 실수형에는 2개의 자료형이 있다.(float, double)
- float형이 4바이트고 double이 8바이트다
- float형에 값을 할당 시 뒤에 f를 붙여준다. 만약 안 붙일 시에 할당 값은 double형으로 인식하며 에러가 난다.
- double형도 값을 할당 시 뒤에 d를 붙여줘야 하지만 안 붙여도 크게 상관은 없다.
- 모든 소수의 기본형은 double 형이다.
- 값에 소수점, 혹은 10의 제곱을 나타내는 e(E), 접미사 f(F), d(D)를 포함하고 있으면 실수형 변수로 간주한다.
- 지수형 데이터도 실수형으로 넣을 수 있다.

변수(Variable)

정수형 예제

```
// float 선언
// float 선언 시 값의 뒤에 항상 f를 붙여야 한다.
float a = 1.23f;
System.out.println(a);

// double 선언
// double 선언 시 값의 뒤에 d를 붙여야 하지만
// 보통 생략이 가능하다.
double b = 1.23d;
System.out.println(b);
double c = 1.23;
System.out.println(c);

// 지수형 데이터도 사용이 가능하다.
double d = 12345.0e-03;
System.out.println(d);

// 실수형 변수와 printf 사용 예제
double e = 123.236;
System.out.printf("%f\n", e); // 123.236000
System.out.printf("%8.2f\n", e); // 123.24
System.out.printf("%.2f\n", e); // 123.24
double f = 123236.184;
System.out.printf("%,4.2f\n", f); // 123,236.18
double g = 12345.0e-03;
System.out.printf("%f\n", g); // 12.345000
```

변수(Variable)

- 문자형

```
char i = 'a';
```

- 문자형은 char형 말고는 없다.
- 변수는 한 글자만 가능
- 숫자를 넣었을 시 Unicode로 인식 문자로 변환
- 크기는 2byte이다.
- 특수문자도 하나의 문자로 저장 가능하다.
- 유니코드 숫자를 넣기 위해선 숫자 앞에 0x를 넣는다.(16진수)
- Ex) 0x0041 = 'A'

변수(Variable)

- 아스키코드(ASCII 코드)
 - 컴퓨터는 0과 1 숫자 밖에 모르기 때문에 문자도 숫자로 기억한다.
 - 이때, 어떤 숫자와 어떤 문자를 대응시키는가에 따라 여러 가지 인코딩 방식이 있는데 통상 아스키 코드 방식을 많이 사용한다.
 - 아스키코드는 1바이트 안에 실제 표현하고자 하는 글자를 매치시켜 만든 표를 의미한다.
 - 1 바이트에서 맨 끝에 존재하는 1비트를 제외한 나머지 비트를 통해 0 부터 127까지의 숫자를 표현할 수 있으며 매치되는 글자는 대략 128개의 글자를 매치시켜 표현할 수 있다.

변수(Variable)

아스키 코드표

제어 문자			공백 문자			구두점			숫자			알파벳		
10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자
0	0x00	NUL	32	0x20	SP	64	0x40	␣	96	0x60				
1	0x01	SOH	33	0x21	!	65	0x41	A	97	0x61	a			
2	0x02	STX	34	0x22	"	66	0x42	B	98	0x62	b			
3	0x03	ETX	35	0x23	#	67	0x43	C	99	0x63	c			
4	0x04	EOT	36	0x24	\$	68	0x44	D	100	0x64	d			
5	0x05	ENQ	37	0x25	%	69	0x45	E	101	0x65	e			
6	0x06	ACK	38	0x26	&	70	0x46	F	102	0x66	f			
7	0x07	BEL	39	0x27	'	71	0x47	G	103	0x67	g			
8	0x08	BS	40	0x28	(72	0x48	H	104	0x68	h			
9	0x09	HT	41	0x29)	73	0x49	I	105	0x69	i			
10	0x0A	LF	42	0x2A	*	74	0x4A	J	106	0x6A	j			
11	0x0B	VT	43	0x2B	+	75	0x4B	K	107	0x6B	k			
12	0x0C	FF	44	0x2C	,	76	0x4C	L	108	0x6C	l			
13	0x0D	CR	45	0x2D	-	77	0x4D	M	109	0x6D	m			
14	0x0E	SO	46	0x2E	.	78	0x4E	N	110	0x6E	n			
15	0x0F	SI	47	0x2F	/	79	0x4F	O	111	0x6F	o			
16	0x10	DLE	48	0x30	0	80	0x50	P	112	0x70	p			
17	0x11	DC1	49	0x31	1	81	0x51	Q	113	0x71	q			
18	0x12	DC2	50	0x32	2	82	0x52	R	114	0x72	r			
19	0x13	DC3	51	0x33	3	83	0x53	S	115	0x73	s			
20	0x14	DC4	52	0x34	4	84	0x54	T	116	0x74	t			
21	0x15	NAK	53	0x35	5	85	0x55	U	117	0x75	u			
22	0x16	SYN	54	0x36	6	86	0x56	V	118	0x76	v			
23	0x17	ETB	55	0x37	7	87	0x57	W	119	0x77	w			
24	0x18	CAN	56	0x38	8	88	0x58	X	120	0x78	x			
25	0x19	EM	57	0x39	9	89	0x59	Y	121	0x79	y			
26	0x1A	SUB	58	0x3A	:	90	0x5A	Z	122	0x7A	z			
27	0x1B	ESC	59	0x3B	;	91	0x5B	[123	0x7B	{			
28	0x1C	FS	60	0x3C	<	92	0x5C	\	124	0x7C				
29	0x1D	GS	61	0x3D	=	93	0x5D]	125	0x7D	}			
30	0x1E	RS	62	0x3E	>	94	0x5E	^	126	0x7E	~			
31	0x1F	US	63	0x3F	?	95	0x5F	_	127	0x7F	DEL			

변수(Variable)

- 유니코드(Unicode)
 - 아스키 코드는 모든 전세계의 문자를 대응하기에는 한계가 있었음
 - 그래서 기존 아스키코드에서 확장해서 2byte의 문자 체계인 유니코드를 채택함
 - 유니코드는 전 세계의 모든 문자를 다루도록 설계된 표준 문자 전산 처리 방식.
 - 유니코드를 사용하면 한글과 한자, 아랍 문자 등을 통일된 환경에서 깨뜨리지 않고 사용할 수 있다.

변수(Variable)

유니코드표

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

- 로마자, 로마자권 기호
- 기타 유럽 문자
- 아프리카 문자
- 중동·서남아시아 문자
- 남부와 중앙 아시아 문자
- 동남아시아 문자
- 동아시아 문자
- CJK 문자
- 인도네시아, 오세아니아 문자
- 북미 및 남미 문자
- Notational systems
- 기호
- 사용자 정의 영역
- UTF-16 상·하위 대체 영역
- 쓰이지 않음

유니 코드 버전 12.0

변수(Variable)

문자형 예제

```
Run | Debug
public static void main(String[] args) {
    // char 사용 예제
    char a = 'a';
    System.out.println(a); // a

    // 아스키코드 번호를 char에 삽입
    char b = (char)97;
    System.out.println(b); // a

    // 16진수 형태로 char에 데이터 삽입
    char c = (char)0x41;
    System.out.println(c); // A

    // 유니코드 삽입
    char d = '\u0041';
    System.out.println(d); // A

    // 유니코드로 한글 '가' 입력하기
    char e = '\uAC00';
    System.out.println(e); // 가
}
```

변수(Variable)

- 논리형

`boolean i = true(false);`

- 논리형은 boolean 한가지 밖에 없다
- 변수는 true 혹은 false만 가능.
- 논리 구현에 주로 사용됨
- 크기가 가장 작다(1 byte)

```
Run | Debug
public static void main(String[] args) {
    // 논리형 사용 예제
    boolean a = true;
    System.out.println(a);

    boolean b = false;
    System.out.println(b);
}
```

변수(Variable)

- 문자열(String)

```
String s = "문자열 값";
```

- 문자열은 기본형 변수가 아닌 참조형 변수이다.
- 단 Java에서 많이 쓰이는 변수형이며 다양한 곳에서 이 변수를 쓰고 있다.
- 문자열 변수는 ""를 쓰는 값을 기본으로 가져가며 문자열을 담을 수 있는 변수이다.
- 참고로 문자열과 문자열간의 합을 통해 문자열을 합칠 수 있는데 이것을 결합 연산자라고 하며 '+'기호를 사용하여 붙일 수 있다.

변수(Variable)

문자열 사용 예제

```
Run | Debug
public static void main(String[] args) {
    // String 변수 예제
    String a = "안녕하세요";
    System.out.println(a); // 안녕하세요

    // String 변수의 결합 연산
    String b = "반갑습니다";
    System.out.println(a+b); // 반갑습니다


    // String과 다른 타입의 연산이 발생한 경우
    // 실제 다른 타입 또한 String 타입으로 강제 변환한다.
    String c = "abcde";
    int d = 1;
    int e = 2;
    System.out.println(c+d+e); // abcde12
}
```

형 변환

- 형 변환

- 형 변환이란, 변수 또는 리터럴의 타입을 다른 타입으로 변환하는 것
- 연산을 하거나 변수에 넣을 값의 타입을 맞추지 않으면 에러가 날 수 있다.
- 형 변환하고자 하는 변수나 리터럴 앞에 변환하고자 하는 타입을 괄호 안에 붙여주면 형 변환이 된다.
- 형 변환 시 사용된 괄호를 캐스팅연산자, 혹은 형 변환 연산자라고 한다.

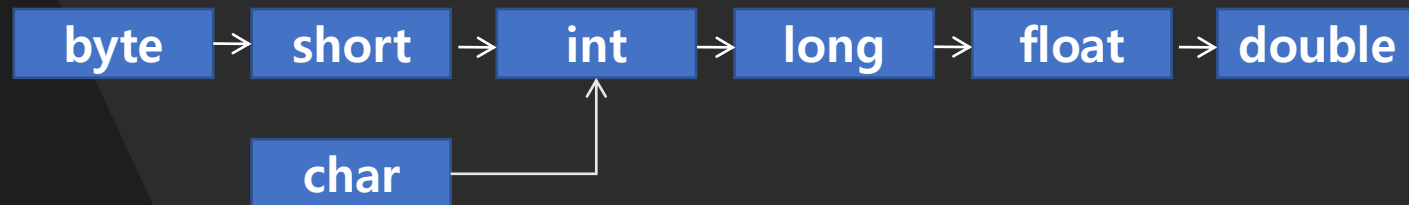
형 변환 연산자



```
int number = (int)85.4;
```

형 변환

- 형 변환
 - 8개의 기본형 중에서 boolean을 제외한 나머지 7개의 기본형 간에는 서로 형변환이 가능하다
 - 범위가 큰 자료형에서 작은 자료형으로 변환은 값 손실이 발생할 수 있다
ex) (int)85.4 => 85
 - 반대로 작은 자료형에서 큰 자료형으로 변환하는 경우에는 값 손실이 발생하지 않으므로 변환에 아무런 문제가 없다.
 - 작은 자료형에서 큰 자료형으로 변환하는 경우 캐스팅 연산자를 생략하는 것을 허용하며 순서는 아래와 같다.



※ 기본형의 자동형변환이 가능한 방향

형 변환

형 변환 예제

Run | Debug

```
public static void main(String[] args) {  
    // 작은곳에서 큰 곳으로의 형변환은  
    // 캐스팅 연산자를 사용하지 않고 깔끔하게 일어난다  
    byte a = 1;  
    short b = a;  
    System.out.println(b); // 1  
  
    // 하지만 큰 곳에서 작은곳의 형변환은  
    // 반드시 캐스팅 연산자를 사용해야 하며  
    // 형변환을 하는 중에 값이 소실될 수 있다.  
    double c = 12.34;  
    int d = (int)c;  
    System.out.println(d); // 12  
}
```


변수의 생존범위

- 변수의 생존범위
 - java 변수는 위치에 따라 생존할 수 있는 범위가 있다.
 - {}(브레이스)를 기준으로 변수의 생존 범위가 결정된다.
 - 변수의 생존 범위는 변수가 생성된 곳 부터 해당 영역이 끝날 때 까지({가 끝날때 까지)가 된다.
 - 변수가 생존하고 있는 동안에만 접근이 가능하며 변수가 생존하지 않는 다른 영역에서는 생존이 불가능하다.
 - 밖의 {}에서 선언된 변수는 안의 {}에서 접근이 가능하지만, 안의 {}선언된 변수는 밖에서 접근이 되지 않는다.

변수의 생존범위

변수의 생존범위 예제

Run | Debug

```
public static void main(String[] args) {  
    int a = 1;  
    {  
        int b = 2;  
        {  
            int c = 3;  
            System.out.println(a);  
            System.out.println(b);  
            System.out.println(c);  
        }  
        System.out.println(a);  
        System.out.println(b);  
        // System.out.println(c); // 에러  
    }  
    System.out.println(a);  
    // System.out.println(b); // 에러  
    // System.out.println(c); // 에러  
}
```