

제02장

# 변수와 자료형

구디아카데미 ▷ 민경태 강사

# 학습목표

1. 변수와 상수에 대해서 알 수 있다.
2. 자바의 기본 자료형에 대해서 알 수 있다.
3. 자바의 기본 자료형 변환 방식에 대해서 알 수 있다.
4. 자바의 참조 자료형에 대해서 알 수 있다.

# 목차

1. 변수와 상수
2. 자바의 기본 자료형
3. 기본 자료형 변환
4. 자바의 참조 자료형

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
    ),
    inArray: function(e, t, n) {
        var r;
        if (t) {
            if (m) return m.call(t, e, n);
            for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n; r in t && t[r] === e) return r;
        }
    }
}

```

# 1. 변수와 상수

# 식별자

- Identifier
- 클래스, 메소드, 변수 등에 붙이는 이름을 의미함
- 한글, 숫자, 영문, 특수문자(underscore, dollar)를 사용할 수 있음
- 불가능한 식별자
  1. 숫자로 시작하는 이름
  2. 이미 사용 중인 키워드와 동일한 이름
  3. 한글 사용은 가능하지만 가급적 사용 자제

# 자바 키워드

- 미리 정의된 키워드들은 식별자로 사용할 수 없음
- 미리 정의된 키워드 목록

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

# Naming Convension

- 패키지, 클래스, 메소드, 변수 등의 이름을 결정할 때는 각각 고유한 이름 지정 방법이 있음
- 각각의 이름을 만들 때는 항상 이름 규칙을 지킬 수 있도록 노력해야 함
- 4가지 이름 규칙이 있음
  1. Pascal Case
  2. Camel Case
  3. Snake Case
  4. Dash Case

# Pascal Case

- 모든 단어의 첫 글자는 대문자 나머지는 소문자로 표시하는 방식

클래스, 인터페이스의 이름 규칙

Pascal Case

MyHomeAddress



# Camel Case

- Pascal Case와 동일하나 첫 단어의 경우 소문자로 시작하는 방식

변수와 메소드의 이름 규칙

Camel Case

myHomeAddress

# Snake Case

- 밑줄(\_)을 이용해서 각 단어를 연결하는 방식

상수의 이름 규칙

Snake Case

`my_home_address`

# Dash Case

- 대쉬(-)를 이용해서 각 단어를 연결하는 방식

자바에서는 사용할 수 없는 방식 (대쉬를 식별자로 사용할 수 없다.)

Dash Case

my-home-address

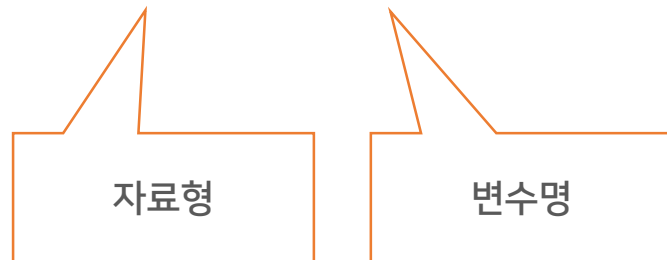
# 변수

- 자바 프로그램이 사용하는 값을 저장하기 위한 메모리 공간
- 저장하려는 값의 종류에 따라 자료형(Data Type)을 결정해야 함
- 변수 선언 후 사용할 수 있음
- 변수에 저장된 값은 언제든지 다른 값으로 바꿀 수 있음
- 변수 이름은 Camel Case 규칙을 따름

# 변수 선언

- 변수는 사용하기 전에 어떤 변수를 사용할 지 미리 선언(declare) 해야 함
- 선언된 변수에 값을 저장하기 전에는 Garbage 값(쓰레기 값)을 가짐
- 형식  
자료형 변수명;

`int price;`



int는 정수 자료형 중 하나이다.

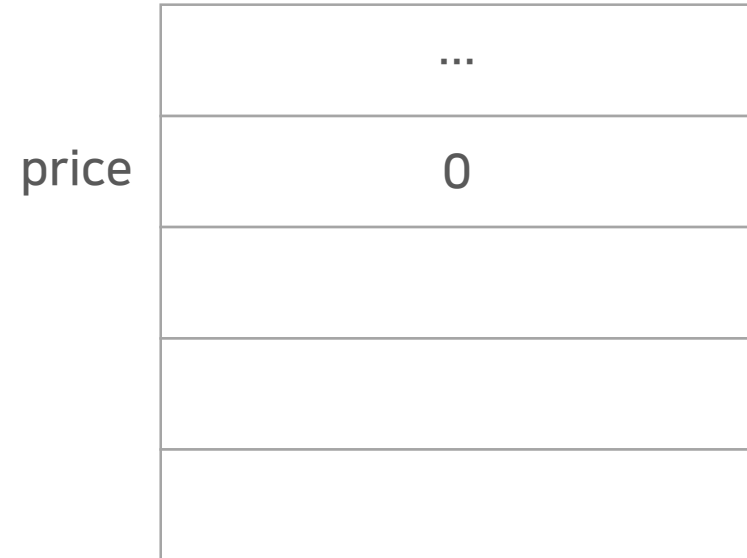
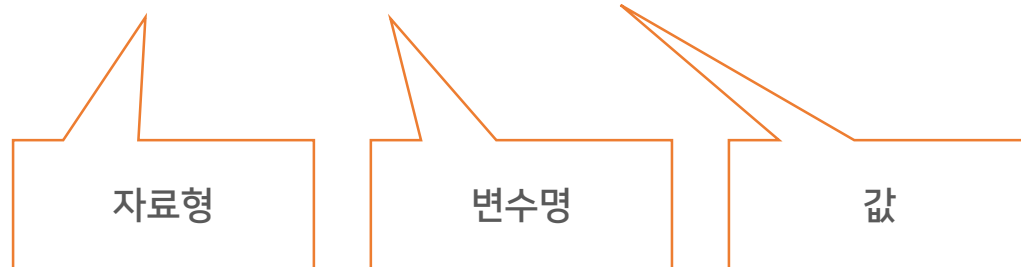


메모리

# 변수 선언과 초기화

- 변수 선언과 동시에 변수에 값을 저장하는 것을 초기화라고 함
- Garbage 값은 사용할 수 없기 때문에 초기화 하는 것을 권장함
- 형식  
자료형 변수명 = 값;

`int price = 0;`



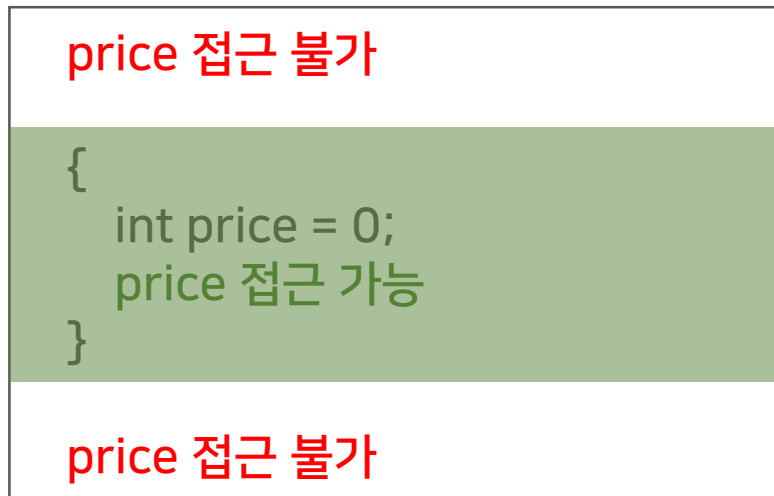
# 변수 선언 위치

- 모든 변수는 코드 블록 내부에서 선언함
- 모든 변수는 자신이 선언된 코드 블록 내부에서만 사용할 수 있음
- 코드 블록 : 중괄호 {}로 묶은 코드 구역
- 코드 블록 예시
  - 클래스 ▶ `class 클래스명 { }`
  - 메소드 ▶ `반환타입 메소드명() { }`
  - 분기문 ▶ `if(조건식) { }`   `switch(표현식) { }`
  - 반복문 ▶ `while(조건식) { }`   `for( ; ; ) { }`

# 지역 변수

- 어떤 변수를 사용할 수 있는 유효 범위를 변수의 스코프(Scope)라고 함
- 어떤 코드 블록에서 선언한 변수는 해당 코드 블록에서만 사용할 수 있는 지역 변수(Local Variable)가 됨

- 개념

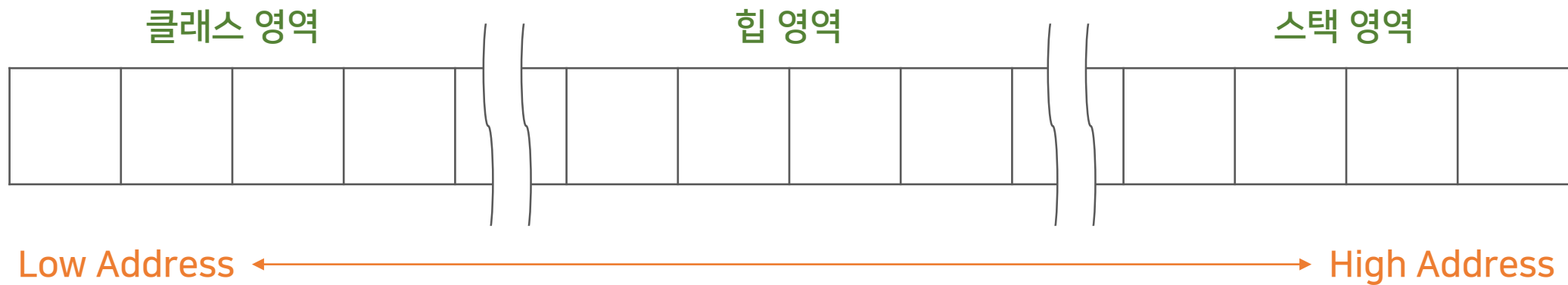


price 변수를 선언한 코드 블록



# 변수와 메모리

- 자바의 메모리는 클래스 영역, 힙 영역, 스택 영역으로 나뉜다
- 종류나 역할 등 다양한 요인에 의해 변수가 저장되는 영역이 구분됨



# 메모리 영역에 따른 구분

## ■ 메모리 영역별 저장되는 구성 요소와 생명 주기

구분	클래스 영역	힙 영역	스택 영역
저장 정보	클래스의 정보 변수의 정보 메소드의 정보 정적 멤버(static)	new 키워드로 생성한 것	지역 변수 값 메소드 반환값
정보 구분	프로그램에 필요한 정보	동적으로 생성한 정보	임시로 필요한 정보
생성 시점	클래스 로딩	new 키워드 호출	변수 선언 메소드 호출
소멸 시점	JVM 종료	Garbage Collector* 동작	변수가 선언된 코드 블록 종료 메소드 실행 종료

\*Garbage Collector

▶ 힙 영역의 저장된 요소 중 아무도 참조하지 않는 요소를 찾아서 자동으로 소멸시켜 주는 자바 구성요소

# 상수

- 저장된 값을 변경할 수 없는 변수를 상수라고 함
- 변수 선언 시 final 키워드를 추가하면 상수가 됨
- 상수는 선언 시 반드시 초기화를 진행해야 함
- 상수 이름은 Snake Case 규칙을 따르며 관례상 모두 대문자로 작성함

# 상수 선언과 초기화

- 변수 선언 시 final 키워드를 추가하면 상수가 만들어 짐
- 상수는 반드시 선언과 동시에 초기화를 진행해야 함
- 형식

final 자료형 변수명 = 값;

**final** **int** PRICE = 30000;

상수 키워드

자료형

변수명

앞으로 수정이  
불가능한 값

PRICE

...
30000

메모리

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return m.call(t, e, n);
        for (r = t.length, r = r ? 0 > r ? Math.max(0, r + n) : r; r--;)
            if (n in t && t[n] === e) return n;
    }
}



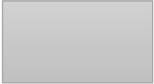
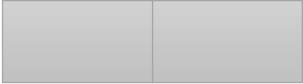


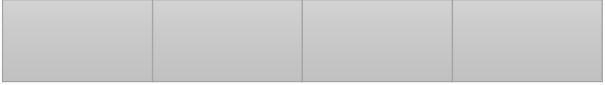

```

## 2. 자바의 기본 자료형

# 자바의 기본 자료형

- Primitive Type
- 어떤 값 자체를 저장할 수 있는 Raw Data 타입의 자료형을 의미함
- 8가지 종류가 있음
  - boolean                  char
  - byte                      short
  - int                        long
  - float                     double

# 기본 자료형 메모리 할당 및 값의 범위

논리 타입	boolean	 (1비트*, true 또는 false)
문자 타입	char	 (2바이트, Unicode)
	byte	 (1바이트, $-2^7 \sim 2^7 - 1$ : -128~127)
정수 타입	short	 (2바이트, $-2^{15} \sim 2^{15} - 1$ : -32,768~32,767)
	int	 (4바이트, $-2^{31} \sim 2^{31} - 1$ : -2,147,483,648~2,147,483,647)
	long	 (8바이트, $-2^{63} \sim 2^{63} - 1$ )
	float	 (4바이트, $-3.4E38 \sim 3.4E38$ )
실수 타입	double	 (8바이트, $-1.7E308 \sim 1.7E308$ )

\*메모리의 기본 단위는 1바이트(8비트) 단위이므로 실제 JVM이 처리하는 boolean의 크기는 1비트가 아닐 수 있다.

# boolean

- 이론 상 1비트의 저장 공간 필요
- 실제로는 더 큰 저장 공간을 사용할 수 있음 (JVM이 결정함)
- true 또는 false 값을 저장할 수 있음

```
boolean isHappy = true;
```



# char

- 2바이트 저장 공간 사용
- 어떤 하나의 문자(영문, 한글 모두 가능)를 저장할 수 있음
- 문자는 작은 따옴표(")로 묶어서 표현해야 함

```
char grade = 'A';
```

# int

- 4바이트 저장 공간 사용
- 정수 값을 저장할 수 있음
- -2,147,483,648 ~ 2,147,483,647 범위의 값을 가질 수 있음

```
int point = 10000;
```

# long

- 8바이트 저장 공간 사용
- int 범위를 벗어나는 정수 값을 저장할 수 있음
- -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807 범위의 값을 가질 수 있음
- int 범위를 벗어난 값은 반드시 알파벳 L을 추가해서 표시해야 함

`long balance = 9876543210L;`

# double

- 8바이트 저장 공간 사용
- 실수 값을 저장할 수 있음
- $-1.7 \times 10^{308} \sim 1.7 \times 10^{308}$  범위의 값을 가질 수 있음

`double score = 3.5;`

# 리터럴

## ■ 코드 상에서 어떤 값을 표현하는 방식

리터럴	의미	예시
25	10진수 25	<code>int n = 25;</code>
025	8진수 25 (10진수 21)	<code>int n = 025;</code>
0x25	16진수 25 (10진수 37)	<code>int n = 0x25;</code>
0b1010	2진수 1010 (10진수 10)	<code>int n = 0b1010;</code>
25L	long 타입의 25	<code>long n = 25L;</code>
3.14	double 타입의 3.14	<code>double n = 3.14;</code>
3.14F	float 타입의 3.14	<code>float n = 3.14F;</code>
true, false	논리 리터럴	<code>boolean isNum = true;</code>
'a'	문자 리터럴	<code>char ch = 'a';</code>
"apple"	문자열 리터럴	<code>String fruit = "apple";</code>
null	널(없음을 의미)	<code>String message = null;</code>

# 이스케이프 시퀀스

- 역슬래시(\, \)로 시작하는 문자를 의미함
- 보통 출력이 어려운 특수 문자를 나타낼 때 사용함
- 주요 이스케이프 시퀀스

이스케이프 시퀀스	의미
\n	라인 피드(line feed)
\r	캐리지 리턴(carriage return)
\t	탭(tab)
\'	작은 따옴표(single quote)
\"	큰 따옴표(double quote)
\\	역슬래시(backslash)

```

each: function(e, t, n) {
  var r, i = 0,
      o = e.length,
      a = M(e);
  if (n) {
    if (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break;
    } else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break;
  } else if (a) {
    for (; o > i; i++)
      if (r = t.call(e[i], e[i]), r === !1) break;
  } else
    for (i in e)
      if (r = t.call(e[i], e[i]), r === !1) break;
  return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
  return null == e ? "" : b.call(e);
} : function(e) {
  return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
  var n = t || [];
  return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
  var r;
  if (t) {
    if (n) return m.call(t, e, n);
    for (r = t.length, r = r ? 0 > r ? Math.max(0, r + n) : r; r--;)
      if (n in t && t[n] === e) return n;
  }
}

```

### 3. 기본 자료형 변환

# 자동 형 변환

- Promotion
- 어떤 자료형을 가진 변수가 다른 자료형으로 자동으로 변환되는 것
- 자동 변환이기 때문에 별도의 코드 작성은 필요하지 않음
- 형 변환을 위한 코드가 없기 때문에 묵시적 형 변환이라고도 함



# 자동 형 변환이 발생하는 경우

- 변환을 해도 원본 데이터의 손실이 없는 경우
  - 원본 데이터의 손실이 발생하는 경우에는 컴파일 오류가 발생함
  - 이런 경우는 강제 형 변환을 해야 함
- 표현할 수 있는 값의 범위\*(도메인, Domain)가 작은 자료형을 큰 자료형으로 변환하는 경우
  - byte ► short ► int ► long ► float ► double
  - char ► int

\* 정수의 경우  $n$  비트로 표현할 수 있는 값의 범위는  $-2^{(n-1)} \sim 2^{(n-1)} - 1$  사이의 범위를 가진다. (23페이지 참고)

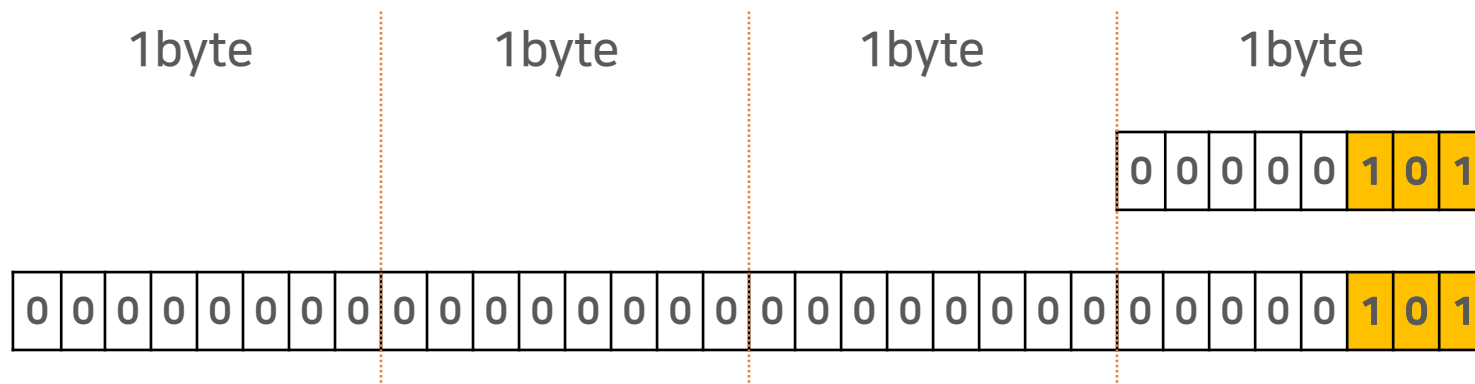
# 자동 형 변환 예시

- 값의 표현 범위가 작은 자료형의 변수를 큰 자료형으로 변환하는 경우

byte → int

```
byte a = 5;
```

```
int b = a;
```



3Byte가 추가로 생기지만 원본 데이터의 변화는 없다.

# 강제 형 변환

- Casting
- 어떤 자료형을 가진 변수를 잠시 다른 자료형으로 강제로 변환하는 것을 의미함
- 자료형을 변경할 변수 앞에 괄호()를 추가하고 괄호 안에 변경할 자료형을 작성함
- 변경할 자료형을 코드로 작성하기 때문에 명시적 형 변환이라고도 함

# 강제 형 변환 예시1

- 실수 자료형을 정수 자료형으로 강제 형 변환하면 소수점 이하 부분은 모두 손실됨

```
double num1 = 5.9;
```

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

5.9를 강제로 정수형(int)으로 변환하면  
소수점 아래 부분이 손실되어 5가 반환된다.

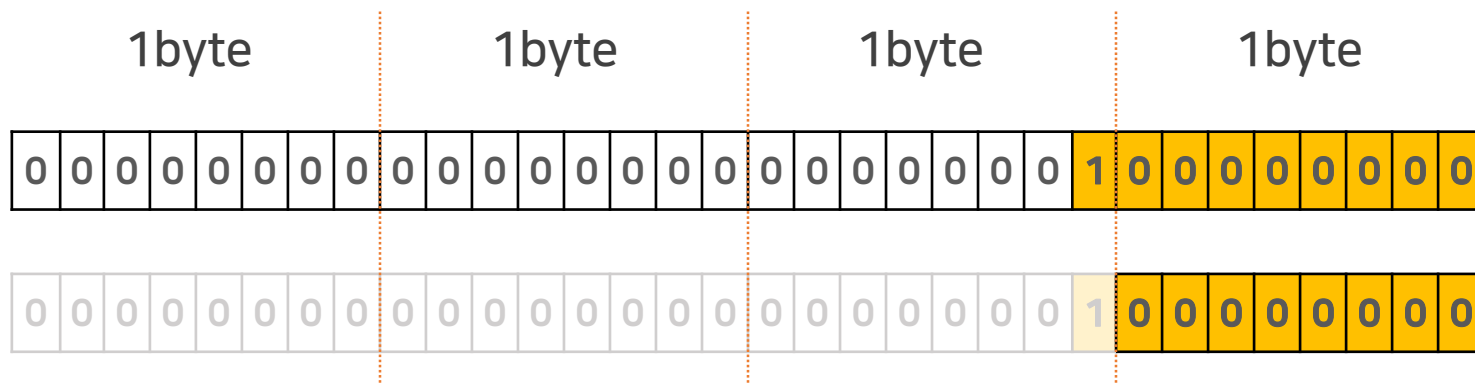
# 강제 형 변환 예시2

- 값의 표현 범위가 큰 자료형의 변수를 작은 자료형으로 변환하는 경우

int → byte

int a = 256;

byte b = (byte)a;



3Byte는 손실된다.

1Byte만 남는다.

강제 형 변환 이후 256이 0으로 변환된다.

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return m.call(t, e, n);
        for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n : 0; r in t && t[r] === e) return r;
    }
}

```

## 4. 자바의 참조 자료형

# 자바의 참조 자료형

- Reference Type
- 메모리는 1바이트마다 고유의 위치 정보를 정수 값으로 가지고 있는데 이를 주소값 또는 참조값이라고 함
- 메모리의 특정 위치를 참조(Reference)해서 해당 위치에 저장된 값을 사용하는 방식의 자료형을 의미함
- 자바는 기본 자료형(Primitive Type) 8가지를 제외한 나머지 모든 데이터를 참조 자료형으로 관리함

# String 클래스

- 문자열을 저장하거나 처리하기 위한 자바 클래스
- String은 널리 사용하지만 기본 자료형이 아니고 참조 자료형임

String name = "tom";

참조 자료형

변수명

값

