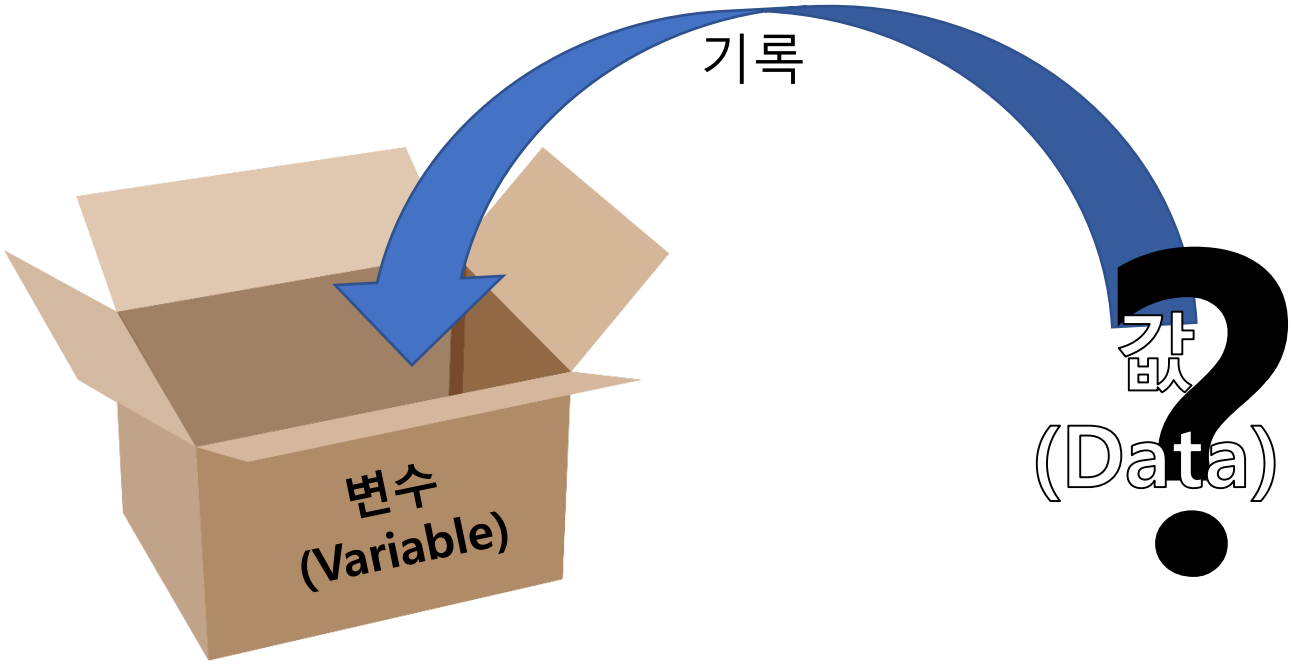
# 변수
# (Variable)

# 변수(Variable)

메모리(RAM)에 값을 기록하기 위한 공간



# ▶ 변수 사용 목적

## ✓ 변수를 사용하지 않으면

```
System.out.println(2 * 3.141592653589793 * 10);
System.out.println(3.141592653589793 * 10 * 10);
System.out.println(3.141592653589793 * 10 * 10 * 20);
System.out.println(4 * 3.141592653589793 * 10 * 10);
```

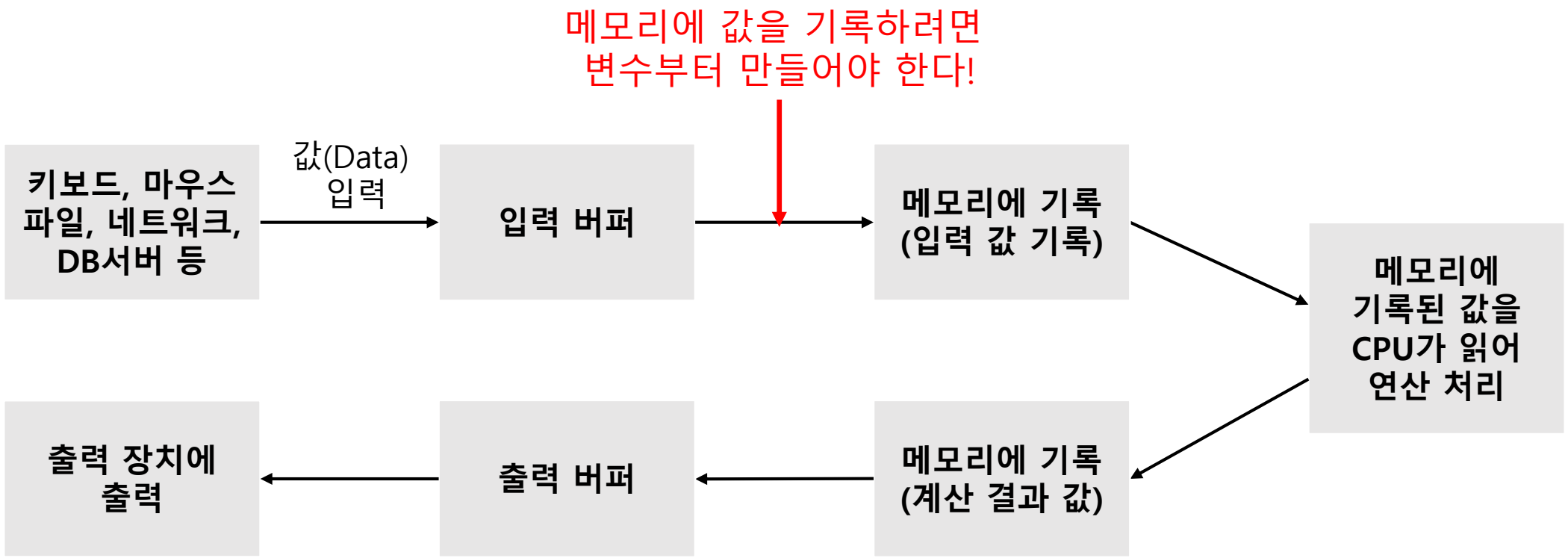
## ✓ 변수를 사용하면

```
double pi = 3.141592653589793;
int r = 10;
int h = 20;

System.out.println(2 * pi * r);
System.out.println(pi * r * r);
System.out.println(pi * r * r * h);
System.out.println(4 * pi * r * r);
```

➡ 가독성이 좋아짐
재사용성 증가로 인한 코드량 감소
유지보수 용이

# ▶ 변수에 값 기록 이유

프로그램 실행 시 사용할 값(Data)이 있다면 그 값은 먼저 메모리에 기록 되어야 함




**프로그램 자동 원리**

# ▶ 변수의 선언

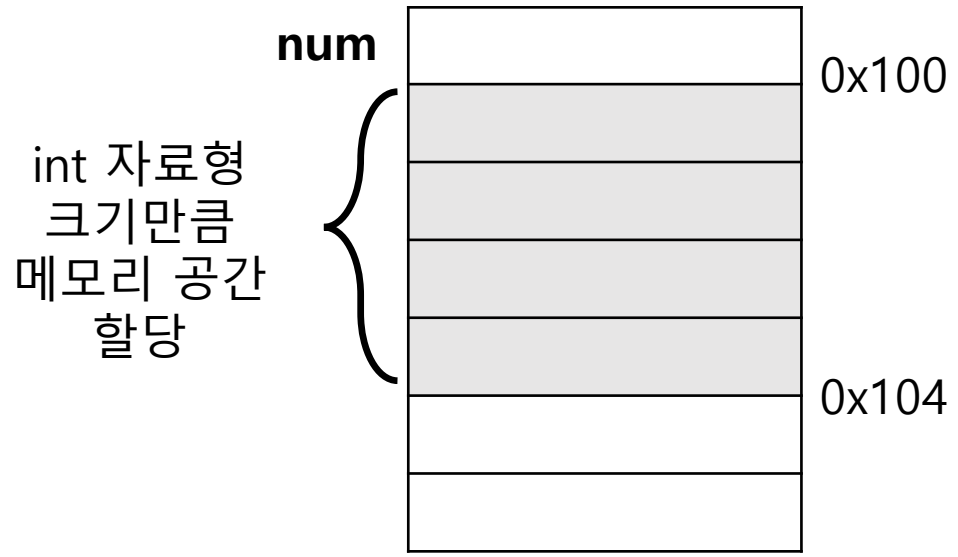
메모리 공간에 데이터를 저장할 수 있는 공간을 할당하는 것

**자료형** **변수명** **;** 마침

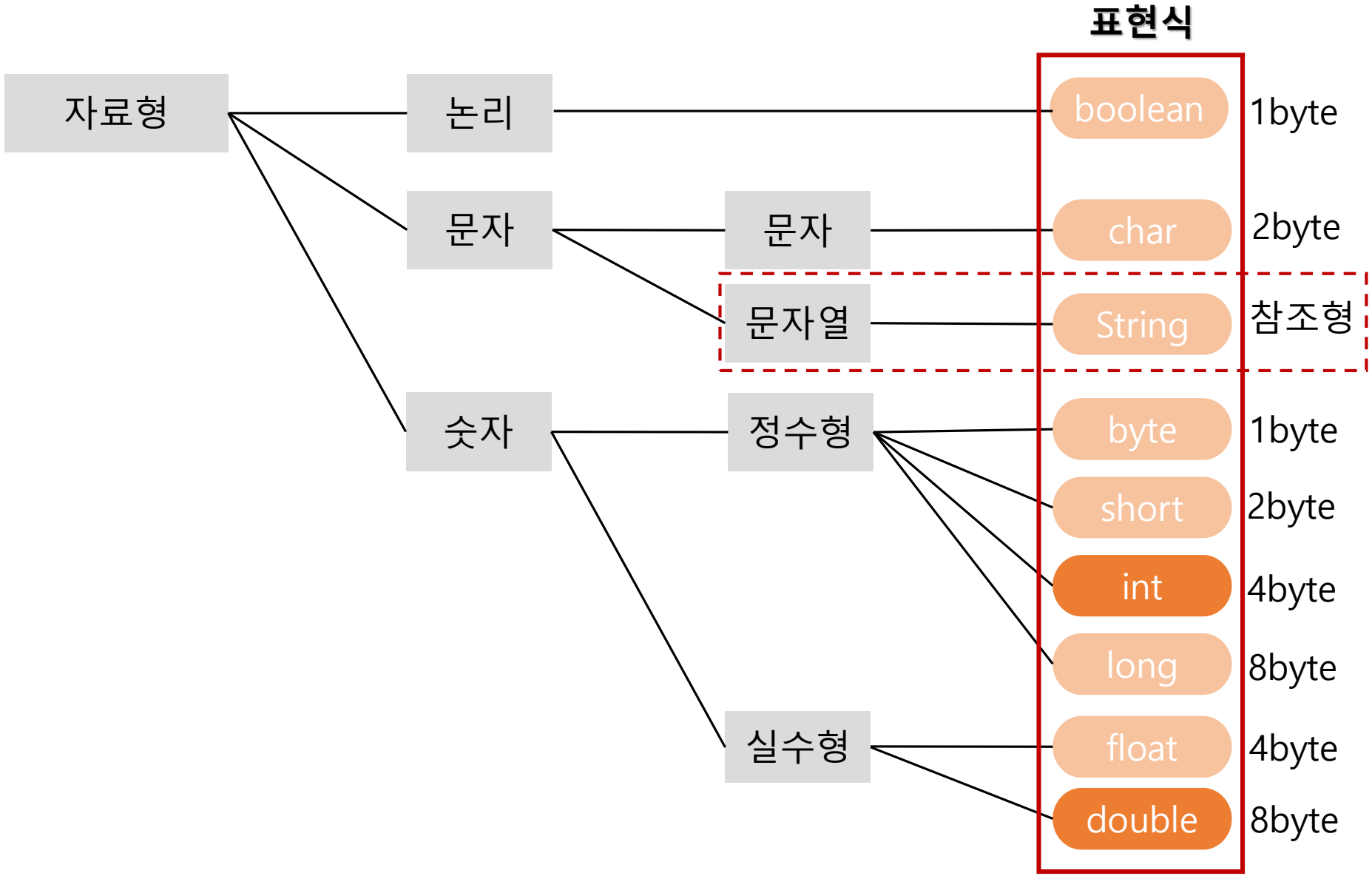
변수타입지정 변수명지정

## ✓ 선언 예시

```
// 논리형 변수 선언
boolean isTrue;

// 문자형 변수 선언
char ch;

//문자열 변수 선언
String str;

// 정수형 변수 선언
byte bnum;
short snum;
int inum;
long lnum;

// 실수형 변수 선언
float fnum;
double dnum;
```



# ▶ 자료형(Type)


표현식
자료형
논리
문자
숫자
문자
문자열
정수형
실수형
boolean
1byte
char
2byte
String
참조형
byte
1byte
short
2byte
int
4byte
long
8byte
float
4byte
double
8byte

# ▶ 데이터 저장 단위

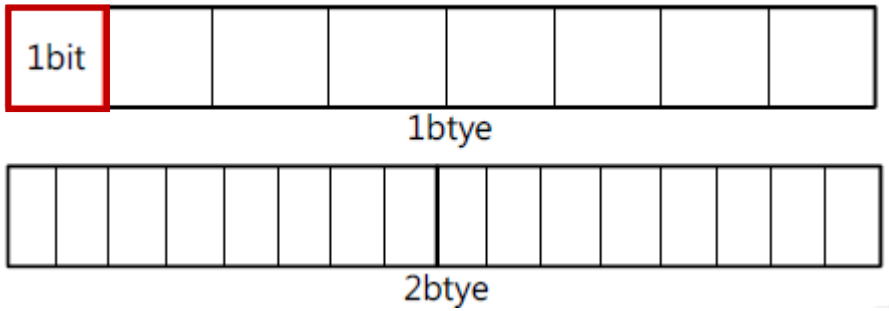
저장 공간이 제한적이기 때문에 저장 크기에 대한 기준과 CPU가 데이터를 처리할 때 일정한 기준 필요

## ✓ 비트(bit)

컴퓨터가 나타내는 데이터의 저장 최소 단위로서 2진수 값 하나를 저장할 수 있는 메모리공간을 의미

## ✓ 바이트(byte)

데이터 처리 또는 문자의 최소 단위로서 8개의 비트가 모여 하나의 바이트가 구성됨



# ▶ 변수 저장 가능 범위

| 자료형 | 범 위 | 크기(bit) | 기본 값 |
|---|---|---|---|
| boolean | true, false | 8 | false |
| char | 0~65,535(유니코드문자) | 16 | '₩u0000' |
| byte | -128 ~ 127 | 8 | 0 |
| short | -32,768 ~ 32,767 | 16 | 0 |
| int | -2,147,483,648 ~ 2,147,483,647 | 32 | 0 |
| long | -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807 | 64 | 0L |
| float | ±1.4E-45 ~ 3.4E38 | 32 | 0.0f |
| double | ±4.9E-324 ~ 1.8E308 | 64 | 0.0 또는 0.0d |

컴퓨터는 2진수로 인지하기 때문에 $2^n$(n = 비트 크기)로 범위 할당

# ▶ 변수의 명명 규칙

**1. 대소문자가 구분되며 길이 제한이 없다.**

**2. 예약어를 사용하면 안 된다.**
ex) true, final, String 등

**3. 숫자로 시작하면 안 된다.**
ex) age1은 가능하지만 1age는 불가능

**4. 특수문자는 '_'와 '$'만을 허용한다.**
- '$'는 내부 클래스에서 사용
- '_' 사용 시 컴파일 에러는 없지만 관례상 사용하지 않는 것이 좋음

ex) sh@rp는 불가능하지만 $harp는 가능

**5. 여러 단어 이름은 단어의 첫 글자를 대문자로 한다.**
**단, 첫 시작 글자는 소문자로 하는 것이 관례이다.**
ex) ageOfVampire, userName

# ▶ 주요 예약어

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

# ▶ 값 대입과 리터럴

## ✓ 값 대입

생성한 변수(저장 공간)에 값을 대입하는 것

```
int age;
age = 10;
age = 20;
```

* 변수는 한 개의 데이터만 보관, 마지막에 대입한 값만 보관

## ✓ 리터럴

변수에 대입되는 값 자체

```
short s = 32767;
int i = 100;
long l = 10000L;
float f = 0.123f;
double d = 3.14;
```

```
char c = 'A';
String str = "ABC";
```

# ▶ 변수의 초기화

변수를 사용하기 전에 처음으로 값을 저장하는 것

→ 지역변수는 반드시 초기화 해야 된다.

✓ **선언 후 초기화**

```
int age;
age = 100;
```

✓ **선언과 동시에 초기화**

```
int age = 100;
```

# ▶ 상수란?

수학에서는 변하지 않는 값 의미

컴퓨터(Java)에서는 한 번만 저장(기록)할 수 있는 메모리 의미

## ✓ 상수 선언 방법

final int AGE;

## ✓ 상수 초기화 방법

**1) 선언과 동시에 초기화**

```
final int NUM = 100;
```

**2) 선언 후 초기화**

```
final int NUM;
NUM = 100;
```

* 초기화 이후 다른 데이터(값)을 대입할 수 없다.

# ▶ 문자열

## ✓ 문자열 표현

컴퓨터에서 "기차", "출력하세요"등과 같이 단어나 문장을 문자열이라고 표현

""로 묶여 있으면 문자열로 인식하며 Java에서는 String 객체를 이용하여 저장

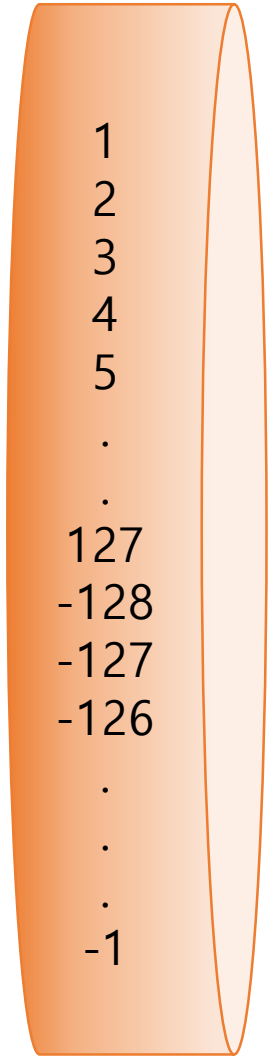
## ✓ 문자열 초기화

```
String str = "기차";
String str = new String("기차");
String str = "기차" + "칙칙폭폭";
String str = new String("기차" + "칙칙폭폭");
String str = "기차" + 123 + 45 + "출발";
String str = 123 + 45 + "기차" + "출발";
```

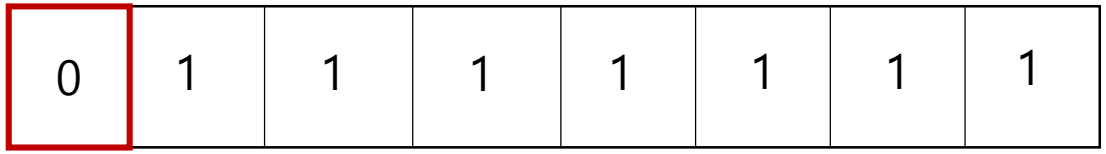
?

**다른 자료형 + "문자열" → 문자열**
**"문자열" + 다른 자료형 → 문자열**

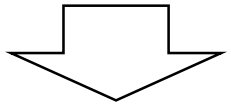
# 데이터 오버플로우

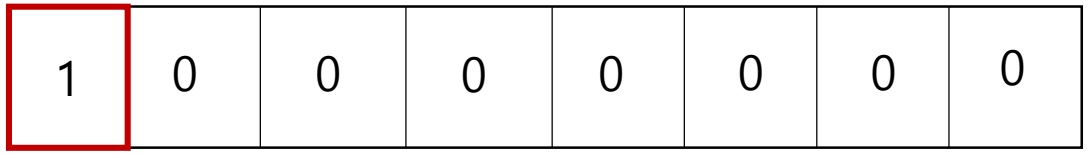



byte형




127+1을 하면 범위를 초과한 128이 되고 허용된 범위 이상의 비트를 침범하게 되는데 이를 **오버플로우**라고 한다.






byte형 허용범위 최소값인 -128이 되는 것이다.

# ▶ 형변환(casting)

값(Data)의 자료형을 바꾸는 것 (boolean 제외)

## ✓ 컴퓨터의 값 처리 원칙

같은 종류 자료형만 대입 가능
같은 종류 자료형만 계산 가능
계산의 결과도 같은 종류의 값이 나와야 함

→ 이러한 원칙이 지켜지지 않은 경우에 형 변환이 필요함

## ✓ 형변환 예시

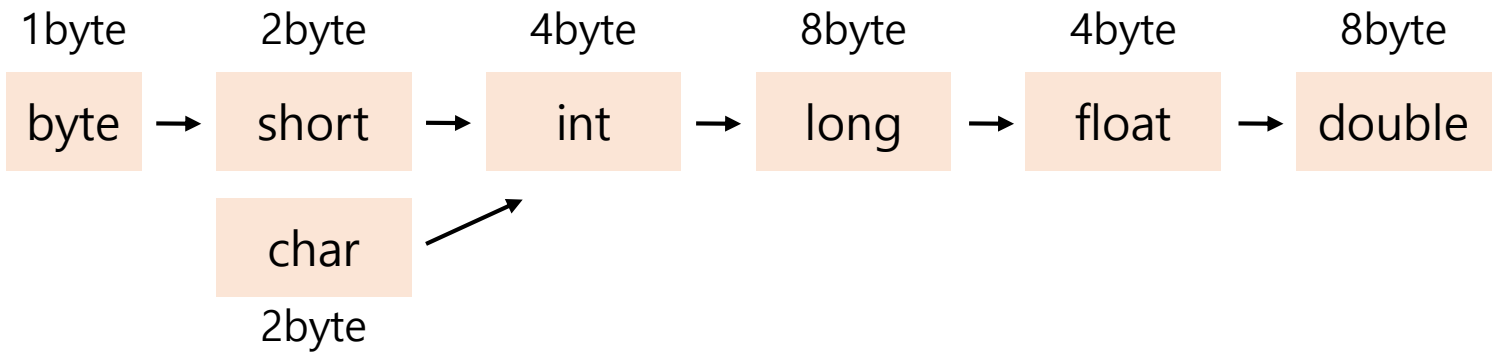
| | | |
|---|---|---|
| 123456789 | → | 123456789.0 |
| (int) | | (double) |
| 'A' | → | 65 |
| (char) | | (int) |
| 3.14f | → | 3 |
| (float) | | (int) |

→ 형변환 하고자 하는 값과
자료형의 표현 범위 차이에 따라
형변환 방법이 나뉨
(자동 형변환, 강제 형변환)

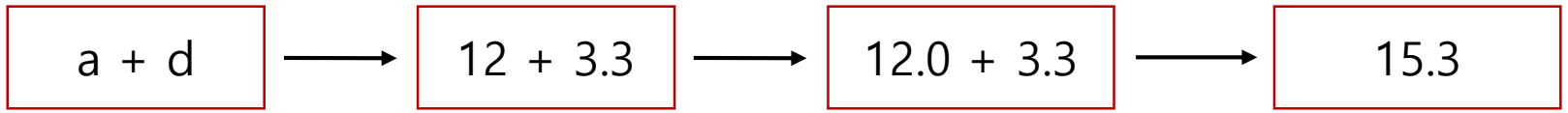
# ▶ 형변환(casting)

## ✓ 자동 형변환

컴파일러가 자동으로 값의 범위가 작은 자료형을 값의 범위가 큰 자료형으로 변환




예시 ) int a = 12;
double d = 3.3;
double result = a + d;




*** 단, byte와 short 자료형 값의 계산 결과는 무조건 int로 처리한다.**

# ▶ 형변환(casting)

## ✓ 강제 형변환

값의 범위가 큰 자료형을 값의 범위가 작은 자료형으로 변환

강제 형변환 시 데이터 손실이 발생할 수 있음 → 데이터의 변형, 손실을 감수하고 강제 변환

```
double temp;
int name = (int)temp;
```

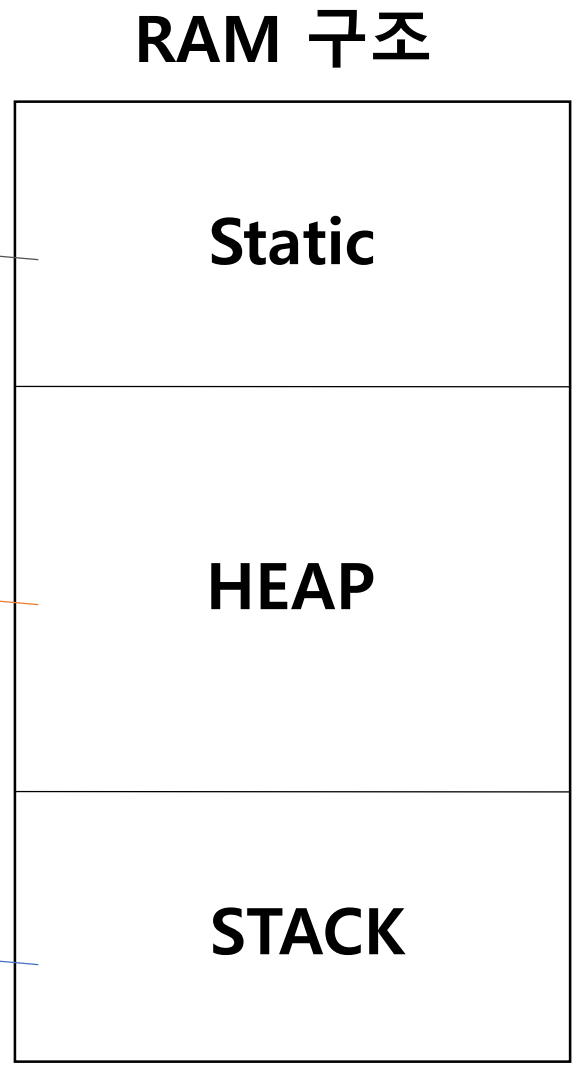
## ✓ 데이터 손실

# 변수와 메모리 구조

static예약어로 선정된 필드,
메소드가 저장되는 공간
클래스 변수 등

new연산자에 의해
동적으로 할당하고 저장되는 공간,
객체 , 배열 등

메소드를 호출하면 자동생성
메소드가 끝나면  자동소멸
지역변수, 매개변수, 메소드 호출 스택 등



# 출력메소드

## ✓ System.out.print( )

() 안의 변수, 문자, 숫자, 논리 값을 모니터에 출력해주는 메소드

## ✓ System.out.println( )

print문과 동일하게 출력은 해주지만 출력 후 자동으로 출력창에 줄바꿈을 해주는 메소드

예)
```
System.out.print("안녕하세요");
System.out.print(123);
System.out.print(변수명);
```

```
System.out.println("안녕하세요");
System.out.println(123);
System.out.println(변수명);
```

# ▶ 출력메소드

## ✓ System.out.printf(“%형식”, 변수 등)

정해져 있는 형식에 맞춰서 그 형식에 맞는 값(변수)을 줄바꿈 하지 않고 출력

**%d : 정수형, %o : 8진수, %x : 16진수**
**%c : 문자, %s : 문자열**
**%f : 실수(소수점 아래 6자리), %e : 지수형태표현, %g : 대입 값 그대로**
**%A : 16진수 실수**
**%b : 논리형**

**정렬방법**

- %5d : 5칸을 확보하고 오른쪽 정렬
- %-5d : 5칸을 확보하고 왼쪽 정렬
- %.2f : 소수점 아래 2자리까지만 표시

# ▶ escape 문자

| 특수문자 | 문자 리터럴 | 비 고 |
|---|---|---|
| tab | ₩t | 정해진 공간만큼 띄어쓰기 |
| new line | ₩n | 출력하고 다음라인으로 옮김 |
| 역슬래쉬 | ₩₩ | 특수문자 사용시 백슬러시(₩)를 넣고 특수문자를 넣어야 함 |
| 작은 따옴표 | ₩′ | |
| 큰 따옴표 | ₩″ | |
| 유니코드 | ₩u | 유니코드 표시할 때 사용 |

# ▶ Scanner

## ✓ Scanner Class

사용자로부터 입력되는 정수, 실수, 문자열을 처리하는 클래스

## ✓ import 작성

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

## ✓ Scanner 생성

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

## ✓ 키보드 입력값 받기

1. **정수 : sc.nextInt();**
2. **실수 : sc.nextFloat(); 또는 sc.nextDouble();**
3. **문자열 : sc.next(); 또는 sc.nextLine();**

next()는 띄어쓰기 입력불가, 띄어쓰기를 구분인자로 생각하여 각각 저장, 줄 구분까지 저장하지 않음
nextLine()은 문자열에 띄어쓰기 가능, 줄 구분까지 저장