

# 객체지향 프로그래밍 강의노트 #02

---

## 자료형과 연산자, 키보드 입력

조용주  
ycho@smu.ac.kr

# 변수(variable)과 대입(assignment) 연산

## □ 기억 장소의 필요성

- 프로그래밍은 주어진 문제를 컴퓨터를 이용해서 해결하는 과정

## □ 원의 반지름과 사각형의 너비, 높이가 주어졌다고 가정하고, 원과 사각형의 면적의 합을 구하는 문제를 생각해볼 것

## □ 문제 해결 과정

- 원의 면적 계산(반지름  $\times$  반지름  $\times$  3.1415)
- 사각형 면적 계산 (너비  $\times$  높이)
- 계산된 원과 사각형의 면적 합계 계산
- 합계를 화면에 출력

사각형의 면적을 계산하기 전에  
원의 면적을 기억하고 있어야 함

# 변수(variable)과 대입(assignment) 연산

- 문제를 풀다 보면, 무엇인가 기억해야 하는 경우들이 발생함
  - 숫자 값 일수도 있고, 집 주소 또는 사람의 이름이 될 수도 있음
- 사람의 뇌가 계산과 기억에 사용되는 것처럼, 컴퓨터에서는 CPU와 메모리가 각각 계산과 기억을 담당
- **메모리는 바이트(byte)라고 불리는 일정한 크기의 공간(상자)들이 붙어 있는 형태라고 생각하면 됨**
  - 값을 기억하기 위해 상자에 값을 넣는다면 어디에 있는지 알아야 함
  - 메모리는 연속적인 숫자로 상자마다 번호(address)를 붙임

# 변수

---

- 자바에서는 변수를 이용해서 기억함
- 프로그래밍 언어의 변수는 메모리의 일부 영역이고, 그곳에 값을 저장하거나 저장된 값을 확인할 수 있음
  - 변수는 메모리 영역 즉 공간(상자)
  - 메모리에는 주소가 붙음
  - 주소가 29183982번지 같은 숫자로 구성되어, 기억하기 어려움
  - 원의 면적을 주소 29183982번지에 저장했다기 보단 AreaOfCircle이라는 이름이 붙은 공간에 저장

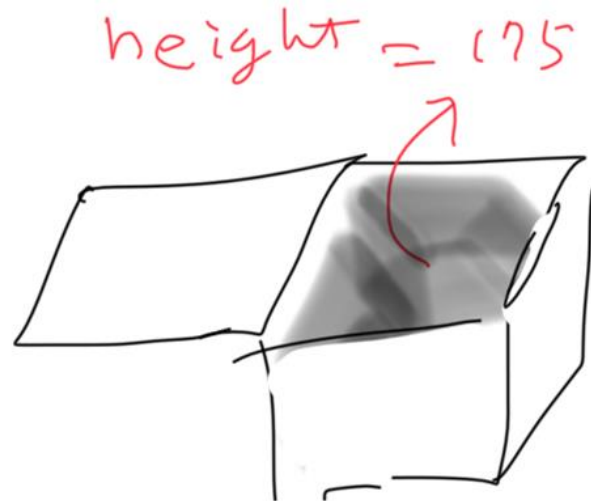
# 변수 이름과 값

## □ 변수 이름

- 메모리 공간 또는 해당 공간에 있는 값을 나타냄

## □ "변수 상자"에 값을 넣을 때에는 변수 이름을 이용해서 "값을 저장"하라고 요청할 수 있음

- 값을 저장: height에 175를 넣어줘 (공간)
- 값을 사용: height를 사용 (값)



# 변수 이름과 값

---

## □ 공간 또는 값

```
int x;  
x = 1;  
int y;  
y = 2;  
y = x;
```

## □ 변수에서 값을 사용한다고 해서 사라지는 것은 아님

# 변수 생성 방법과 초기화

- 변수는 사용하기 전에 반드시 생성해야 함
- 자바의 변수는 만들어질 때 자료형(data type)을 지정해야 함
  - 자료형은 변수에 저장할 수 있는 값의 종류
- 자바는 **자료형을 엄격하게 지키는 언어**
  - 따라서 생성할 때 **자료형을 명확하게 지정하고 사용**
- 변수를 생성하는 과정을 변수를 정의(define)한다고 함
- 변수 생성 방법
  - 자료형 변수\_이름 [ = 초기값];
  - [] 안의 내용은 생략 가능

# 변수 생성 방법과 초기화

요소	설명
자료형	기본형 또는 클래스
변수_이름	이름 짓기 규칙에 따름
초기값	정의할 때 초기화할 수 있음 또는 생략 가능

- 예 `int num;`
- 변수는 사용 전에 값을 저장해야 의미가 있음

```
int num;  
num = 3;
```

- 변수를 생성하면서 초기화(초기값을 지정)

```
int num = 3;
```

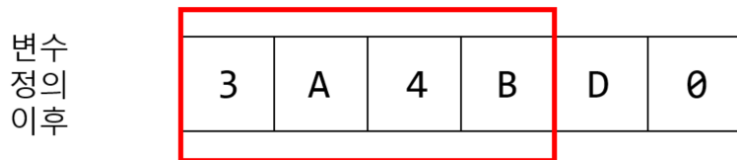
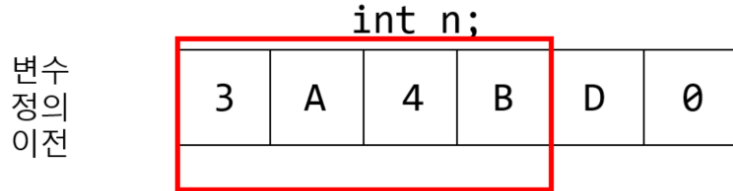


# 변수 생성 방법과 초기화

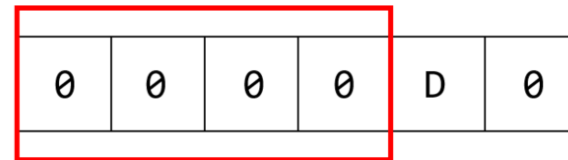
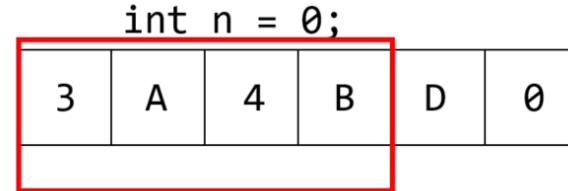
## □ 변수 생성 예

```
int n = 3;  
double i;  
char c = '가';  
boolean b = false;
```

- 변수의 초기값을 지정하지 않으면, 해당 변수에 어떤 값이 들어가 있는지 알 수 없음
  - 초기값이 지정되지 않는 변수는 메모리 공간에 원래 있던 값이 남게 됨 (쓰레기 값)



변수 n  
16진수로 3A4B(십진수: 14923)라는  
값이 있음



변수 n  
0이 있음

# 변수 생성 방법과 초기화

- 자바에서는 변수를 정의하면서 초기값을 지정하지 않으면 스스로 초기화 시키거나 오류 발생
- JShell이나 클래스의 멤버 변수는 자동으로 0에 준하는 값으로 초기화

```
jshell> int a;  
a ==> 0
```

```
jshell> System.out.println(a);  
0
```

# 변수 생성 방법과 초기화

- ❑ 컴파일할 때 클래스 멤버 변수는 자동으로 초기화
- ❑ 초기화시키지 않은 변수를 사용할 때 오류 발생
- ❑ 다음 코드는 컴파일 오류 발생

```
// UninitializedVariable.java
public class UninitializedVariable {
    public static void main(String[] args) {
        int num;
        System.out.println("num = " + num);
    }
}
```

출력 결과

```
UninitializedVariable.java:5: error: variable num might
not have been initialized
        System.out.println("num = " + num);
                                   ^
```

# 변수 생성 방법과 초기화

## ▣ 오류없는 코드

```
// InitializedVariable.java
public class InitializedVariable {
    public static void main(String[] args) {
        int num;
        num = 3; // 변수 num에 값을 저장함
        // num 값을 화면에 출력
        System.out.println("num = " + num);
    }
}
```

출력 결과

```
num = 3
```

# 식별자 이름 짓기(naming) 규칙

---

- 식별자(identifier)는 변수, 상수, 함수, 클래스, 인터페이스, 패키지 등의 이름을 나타냄
- 프로그래밍에서 변수를 사용하는 것은 나중에 그 값을 다시 사용하기 위함
  - 어떤 변수에 1을 저장하고, 다른 변수에 2를 저장
  - 이 변수들을 어떻게 부를 것인가?
- 자바 변수는 적절한 이름을 붙여야만 사용 가능

# 식별자 이름 짓기 규칙

## □ 식별자 이름을 짓는 규칙

- 알파벳 영문자, 밑줄 문자('\_',) 또는 달러 문자('\$')로 시작해야 함 (**관례적으로 알파벳 영문자로 시작**)
- 두 번째 글자부터는 알파벳 영문자, 밑줄 문자, 달러문자, 또는 숫자를 쓸 수 있음
- 중간에 공백문자가 들어갈 수 없음
- 달러 문자가 아닌 다른 특수 기호 사용 못함
- 자바의 키워드(keyword)를 사용할 수 없음
- 영문 대문자와 소문자를 구별함('a'와 'A'는 다른 글자)
- 일반적으로 변수 이름을 지정할 때 카멜 표기법을 사용
- 이름의 길이는 무제한, 하지만 계속 입력해야 하므로, 너무 길지 않으면서도 의미가 뚜렷하게 할 것
  - 학생 숫자 변수를 n보다는 numberOfStudents

# 식별자 이름 짓기 규칙

## ▣ 키워드

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

# 식별자 이름 짓기 규칙

---

## □ 카멜 표기법

- 단어를 여러 개 붙여서 이름을 만들 때 각 단어의 시작 글자만 대문자로 표기하고 나머지 철자는 소문자로 지정하는 방법
- 예: NumberOfStudents
- 관례적으로 자바에서는 클래스 이름은 대문자로 시작하고, 함수와 변수 이름은 소문자로 시작함

## □ 변수 이름이 잘 지정된 예

- box1, Box1, coffee, hello, \$hello, \_hello, new\_instance, newinstance, NewInstance, new1Instance, New1Instance, new1



# 식별자 이름 짓기 규칙

## ▣ 변수 이름이 잘못 지정된 예

변수 이름	이유
box 1	변수 이름 중간에 공백 문자가 있음
*box1	특수 기호로 시작
Box*	특수 기호가 있음
new.instance	특수 기호가 있음
1box	숫자로 시작
new	키워드

# 대입 연산자

## □ 대입 연산자(Assignment operator)

- 변수에 초기값을 지정하거나 또는 값을 저장할 때 사용되는 '=' 기호
- '=' 기호 왼쪽에 있는 변수는 메모리 공간을 나타냄
- 기호 오른쪽에 있는 변수는 값을 나타냄
  - 오른쪽에 변수가 여러 개 있어도 역시 값

```
int x = 3;  
int y = 5;  
int z = x;
```

- 왼쪽의 x, y, z는 변수 공간, 오른쪽 x는 값을 나타냄

```
x = x + 3;  
y = x + y;
```

# 변수와 자료형

---

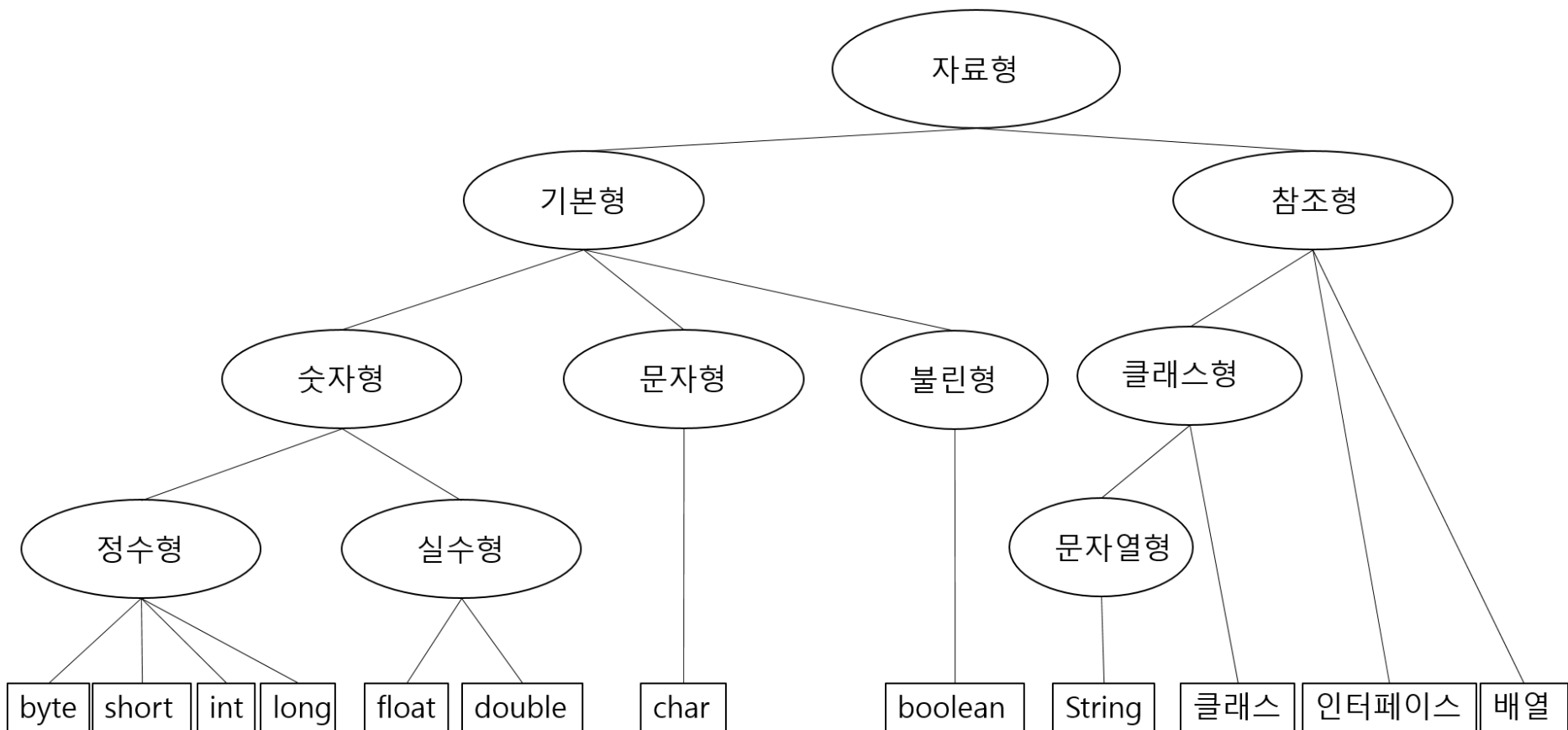
- 자바에는 다양한 자료형이 존재함
  - 자바 언어에서 이미 정의된 것들도 있고, 프로그래머가 만들어서 사용하는 것들도 있음
- 자바 언어에서 기본적으로 제공하는 자료형을 **기본 자료형(primitive type)** 또는 **기본형**
  - 사람들이 컴퓨터에서 많이 사용하는 값을 표현 (정수, 실수, 참/거짓, 문자 등)
  - **byte, short, int, long, float, double, char, boolean** 등의 이름을 붙임
  - 각 자료형이 표현할 수 있는 값의 종류와 범위가 명확하게 정의됨
    - 메모리 공간의 크기를 정해져 있음

# 변수와 자료형

---

- 자바에는 복잡한 형태의 자료(complex data type)들을 담을 수 있도록 만들어진 참조형이 있음
- 참조형
  - 기본형이 아닌 모든 자료형
  - 참조값(주소값)을 가지는 자료형
- 참조형 종류
  - 문자열
    - 한 개 이상의 문자들로 구성된 단어나 문장
    - String에 저장
  - 배열
    - 같은 종류 자료형을 여러 개 묶어 놓은 것
  - 클래스(class)와 인터페이스(interface) 등

# 변수와 자료형



# 변수와 자료형

---

## □ 숫자

- 정수와 소수(실수) 등이 있는데, 서로 특성이 다르고 값의 범위가 다름

## □ 문자

- 한 글자의 알파벳, 한글, 아라비아 숫자, 외국어 등을 표시하는데 사용

## □ 불린

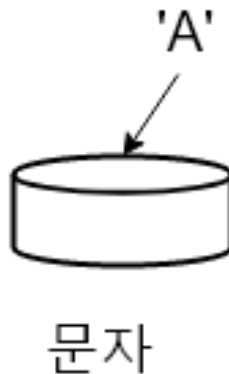
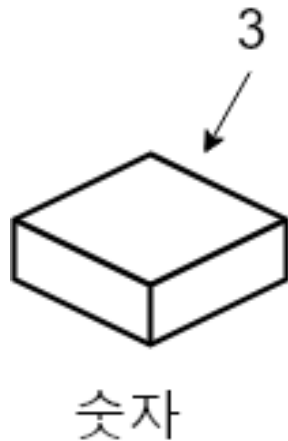
- 참(true) 또는 거짓(false)를 나타냄

## □ String

- 문자열을 표현

# 변수와 자료형

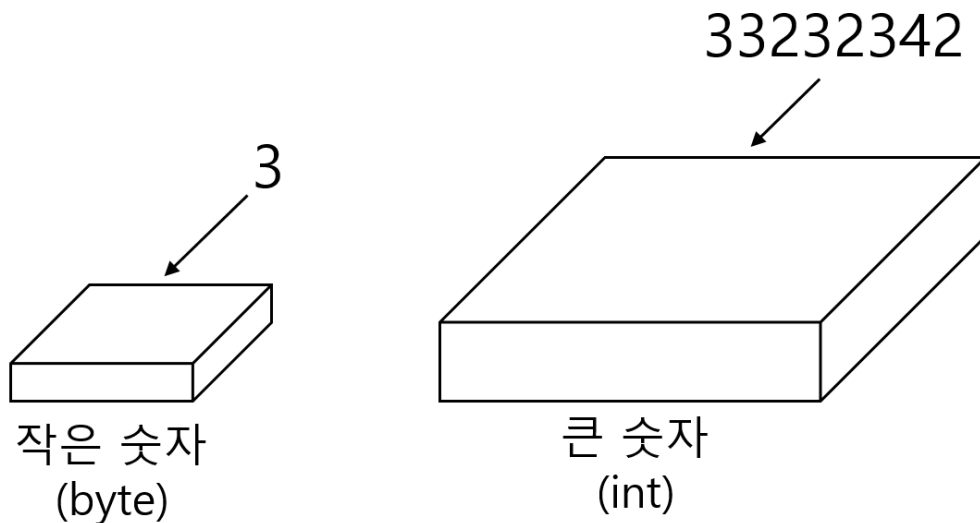
- 같은 자료형으로 생성된 변수들은 다른 변수값을 저장할 수 있음
- 가끔 다른 자료형 변수들 사이에서도 변수값을 저장할 수 있는 경우가 있음
- 변수를 상자에 비유해서 설명



- 모양이 다르면 내용물을 옮겨서 담는 것이 쉽지 않듯이, 일반적으로 다른 종류의 변수에 값을 저장할 수 없음

# 변수와 자료형

- 자료형이 달라도 값을 저장할 수 있는 경우가 있음
- 자료형 중에는 같은 종류인데, 크기에 따라 다른 것으로 분류되는 것들이 있음
- 숫자형 – 정수형과 실수형
  - 정수형은 byte, short, int, long으로 분류
  - 실수형은 float과 double로 분류





# 기본형(primitive type)

---

- 주로 컴퓨터에서 자주 사용되는 **단순한 값**
- 숫자형에 속하는 자료형은 메모리 공간에서 차지하는 크기, 언어적으로는 표현할 수 있는 숫자의 범위에 따라 다양하게 제공됨
- 이러한 숫자값들을 객체로 표현해야 할 때가 있고, 이를 위해 각 자료형에 대응하는 클래스 자료형이 있음 → 포장 클래스(wrapper class)

# 기본형(primitive type)

기본형	메모리 크기(bits)	범위 (최소—최대)	포장 클래스
byte	8	-128~127	Byte
short	16	-32768~32767	Short
int	32	-2147483648 ~2147483647 (약 21.4억)	Integer
long	64	-9223372036854775808 ~9223372036854775807	Long
float	32	1.4E-45~3.4028235E38 (E-45는 $10^{-45}$ 승)	Float
double	64	4.9E-324 ~1.7876931348623157E308	Double

# 기본형(primitive type)

기본형	메모리 크기(bits)	범위(최소—최대)	포장 클래스
char	16	0~65536	Character
boolean	8	true, false	Boolean

# 정수형(byte, short, int, long형)

---

- 자바에는 작은 범위의 정수를 표현하는 byte부터 큰 범위의 정수를 표현하는 long까지 다양한 정수 자료형이 있음
- 같은 정수를 표현하는데 여러 가지 자료형을 제공하는 것은 메모리를 효율적으로 활용하고 실행 속도를 빠르게 처리할 수 있게 하기 위함
- 예를 들어 프로그램에서 다루는 내용이 서울의 행정 구역 중 한 가지인 구라고 가정
  - 만약 구를 컴퓨터로 처리하기 위해 표에서 보인 것처럼 숫자로 표현한다고 가정
  - 메모리를 효율적으로 사용하려면 byte형 사용

# 정수형(byte, short, int, long형)

구 이름	숫자
강남구	1
강동구	2
강북구	3
...	...
중구	24
중랑구	25

- 국내에는 일반 대학, 대학원 대학, 전문대학, 기능 대학 등 약 400여개의 고등교육기관이 있음
  - 이러한 자료를 컴퓨터에서 다룬다면 최소한 short가 필요
- 국내 인구를 다룬다면 int형
- 세계 인구를 다룬다면 long형이 필요

# 정수형(byte, short, int, long형)

---

- 일반적으로 int형 사용
- int형으로 처리할 수 없는 큰 범위의 숫자들을 다뤄야 하면 long 자료형
- 실행속도가 떨어지더라도 메모리 사용량을 최소화 시키겠다면 byte나 short 자료형을 사용
- 대충 범위를 기억할 것
  - byte: -128~127
  - short: -3만 2천~3만 2천
  - int: -21억~21억
  - long: -92경~92경
- 자세한 내용은 검색해서 사용

# 정수형(byte, short, int, long형)

- ▣ 변수의 자료형이 표현할 수 있는 범위 밖의 값을 저장하면 오류 발생

```
jshell> byte b1 = -128;  
b1 ==> -128
```

```
jshell> byte b2 = 127;  
b2 ==> 127
```

```
jshell> byte b3 = -129;  
| Error:  
| incompatible types: possible lossy conversion  
from int to byte  
| byte b3 = -129;  
|           ^_ _^
```

## 정수형(byte, short, int, long형)

---

```
jshell> byte b3 = 128;  
|   Error:  
|   incompatible types: possible lossy conversion  
from int to byte  
|   byte b3 = 128;  
|               ^_^
```



## 정수형(byte, short, int, long형)

- 큰 범위의 자료형 변수에서 작은 범위의 자료형 변수로 저장할 때에는 값을 유실된다고 봄
  - 실제 값이 작은 범위의 자료형에 저장될 수 있는지 확인하지 않음

```
jshell> byte b1 = 10;  
b1 ==> 10
```

```
jshell> int i1 = 10;  
i1 ==> 10
```

```
jshell> i1 = b1;  
i1 ==> 10
```

# 정수형(byte, short, int, long형)

```
jshell> b1 = i1;  
|   Error:  
|   incompatible types: possible lossy conversion  
from int to byte  
|   b1 = i1;  
|           ^^
```

- ❑ 정수형 변수인 i1에서 바이트형 변수인 b1에 저장할 때 오류 발생
  - i1의 값이 바이트형 변수의 범위 내인지 확인하지 않음
- ❑ 정수 뒤에 영문 소문자 l 또는 L을 붙이면 int형이 아니라 long형으로 취급함

```
long x = 3L;  
long y = 3l;
```

# 실수형(float, double형)

- float과 double형은 실수를 표현하는데 사용됨
  - 실수란 컴퓨터에서 소수점이 붙은 숫자들을 의미함
  - float은 4바이트,  $10^{-45} \sim 10^{38}$  범위의 숫자 표현
  - double은 8바이트,  $10^{-324} \sim 10^{308}$ 정도 표현 가능
- 자바에서 숫자 뒤에 'f' 또는 'F'를 붙이면 해당 숫자는 float형임을 나타냄 (붙지 않으면 double형)

```
jshell> float f1 = 0.1f;  
f1 ==> 0.1
```

```
jshell> float f2 = 0.1F;  
f2 ==> 0.1
```

```
jshell> double d1 = 0.1;  
d1 ==> 0.1
```

```
jshell> float f3 = 0.1;
|   Error:
|   incompatible types: possible lossy conversion
from double to float
|   float f3 = 0.1;
|               ^_^
```

- 큰 범위의 double형 숫자 값을 float 변수에 저장할 때 오류 발생
  - 큰 범위 → 작은 범위는 허용 안 함

## 실수형(float, double형)

- float형 변수값을 double 변수에 저장할 때 허용은 되지만, 주의해야 함

```
jshell> d1 = f1;  
d1 ==> 0.10000000149011612  
  
jshell> System.out.println(d1);  
0.10000000149011612  
  
jshell> System.out.println(f1);  
0.1
```

- d1이 0.1이 아닌 다른 값이 출력됨
  - 실숫값을 저장하는 방식에서 나오는 오차

```
jshell> float f2 = 0.1f;  
f2 ==> 0.1
```

```
jshell> f2 = f2 * 0.001f;  
f2 ==> 1.00000005E-4
```

```
jshell> double d2 = 0.1;  
d2 ==> 0.1
```

```
jshell> d2 = d2 * 0.001;  
d2 ==> 1.0E-4
```

```
jshell> System.out.println(f2);  
1.00000005E-4
```

```
jshell> System.out.println(d2);  
1.0E-4
```

- float에서는 오류 발생 (약간의 계산 오차 발생)
- double에서는 제대로 계산됨
- 실숫값을 다룰 때에는 가능하면 double을 사용할 것

# 문자형(char형)

---

- 문자 한 개를 표현
- 자바에서 문자는 영문 알파벳, 아라비아 숫자, 마침표나 콤마 같은 특수 기호, 한글 자모 등을 포함해서 거의 세계 모든 나라의 언어를 표현할 수 있음
  - 알파벳이나 특수 기호는 한 글자씩이 char로 표현됨
  - 한글은 '가', '나' 같은 글자뿐만 아니라 'ㄱ' 또는 'ㄴ' 같은 자음과 모음이 모두 char형으로 표현됨
  - 자바 코드에서 문자를 표현할 때에는 작은 따옴표(') 두개 사이에 문자를 넣음

# 문자형(char형)

```
jshell> char ch1 = 'A';  
ch1 ==> 'A'
```

```
jshell> char ch2 = '가';  
ch2 ==> '가'
```

```
jshell> char ch3 = ch2;  
ch3 ==> '가'
```

```
jshell> ch3 = 'ㄴ';  
ch3 ==> 'ㄴ';
```

```
jshell> System.out.println(ch1);  
jshell> System.out.println(ch2);  
jshell> System.out.println(ch3);
```



# 이스케이프 시퀀스(escape sequence)

---

- 화면에 안 보이는 글자들과 따옴표같이 특별한 용도로 사용되는 글자들을 코드에서 문자 또는 문자열로 표현할 때 사용됨
  - 입력하면서 줄을 바꿀 때 엔터(Enter)키를 누르는데, 이 글자는 보이지 않지만 줄바꿈 문자가 존재함
  - 이런 문자를 코드에서 문자 또는 문자열로 표현할 때 '\n' 같은 형태로 표현할 수 있음
- 항상 백슬래시('\')로 시작함)

# 이스케이프 시퀀스(escape sequence)

이스케이프 시퀀스	설명
\\	백슬래시, \
'\''	작은 따옴표, '
\"	큰 따옴표, "
\n	줄바꿈 문자
\t	탭 문자

"hello\n" // 줄바꿈 문자를 포함하는 문자열

'\n' // 줄바꿈 문자

'\"' // 큰 따옴표 문자

"연희가 \"안녕\"이라고 말함" // 연희가 "안녕"이라고 말함

# 불린형(boolean형)

- 참과 거짓을 의미하는 true/false값
  - 참/거짓을 표현
  - **true**와 **false**라는 키워드를 제공함

```
jshell> boolean b11 = true;  
b11 ==> true
```

```
jshell> boolean b12 = false;  
b12 ==> false
```

```
jshell> System.out.println(b11);  
jshell> System.out.println(b12);
```

# 숫자 상수(numeric literals)

---

- 이렇게 코드에서 숫자 값을 직접 표기하는 것을 숫자 상수라고 부름
  - 앞에서 본 코드에서 x와 y 변수에 1과 2라는 숫자 값을 저장함 (1과 2가 숫자 상수)
  - 숫자 상수에 소수점이 있으면 소수, 부동 소수점 수, 또는 실수라고 표현 → 여기서는 실수라고 주로 사용할 예정
  - 숫자 상수에 소수점이 없다면 정수
- 자바에서는 숫자 상수에 소수점이 있으면 double형, 없으면 int형으로 가정
  - 숫자의 범위와 상관없음 (7, -10이 byte가 아니라 int)

# 숫자 상수(numeric literals)

수	자료형	설명
7	int	소수점이 없음
-10	int	음수도 정수
0	int	당연히 0도 정수에 포함됨
0.7	double	소수점이 포함되므로 실수
.7	double	소수점 앞에 0이 생략되어도 됨. 0이 있는 것으로 가정
7.	double	소수점 뒤에 숫자가 없으면 0이 생략된 것으로 봄
7E5	double	$7 \times 10^5$ 표현
7E-5	double	$7 \times 10^{-5}$ 표현
7.2E-3	double	$7.2 \times 10^{-3}$ 표현

# 대입 연산자와 자료형

- 기본적으로 대입 연산자의 왼쪽에 있는 변수와 오른쪽에 있는 값의 자료형은 같아야 함
- 다음 코드는 JShell에서 변수와 값의 자료형이 같을 때와 다를 때 어떻게 결과가 나타나는지 보임

```
jshell> int x = 1; // x와 1은 모두 정수형  
x ==> 1
```

```
jshell> int y;           // 변수 정의  
y ==> 0
```

```
jshell> y = 2;           // y와 2는 모두 정수형(int)  
y ==> 2
```

# 대입 연산자와 자료형

```
jshell> x = y;           // x와 y는 같은 정수형  
x ==> 2
```

```
jshell> double d = 2.3;  
d ==> 2.3
```

```
jshell> x = d;  
|   Error:  
|   incompatible types: possible lossy conversion  
|   from double to int  
|   x = d;  
|       ^
```

```
jshell> d = x;  
d ==> 2.0
```

# 대입 연산자와 자료형

```
jshell> float f;
```

```
f ==> 0.0
```

```
jshell> f = d;
```

```
| Error:  
| incompatible types: possible lossy conversion  
from double to float  
| f = d;  
|      ^
```

- double형 값을 int형 변수나 float형 변수에 저장할 때 오류가 발생함
  - 서로 호환되지 않는 타입(incompatible type)이라고 나타남
    - 자료형이 다르기때문에 발생



# 대입 연산자와 자료형

- "possible lossy conversion from double to int" 또는 "possible lossy conversion from double to float"
  - 큰 범위의 숫자 값을 작은 범위의 자료형 변수에 저장할 때 숫자 값이 왜곡될 수 있음을 나타냄
- 자바에서는 큰 범위의 숫자 값을 작은 범위의 자료형 변수에 저장할 때 오류 발생 시킴
- 작은 범위의 자료형 값을 큰 범위의 자료형에 넣을 때에는?

```
jsshell> f = x;  
f ==> 2.0  
jsshell> byte b = 3;  
jsshell> x = b;  
jsshell> d = f;  
d ==> 2.0
```

작은 범위를 큰 범위로  
저장하는 것은  
괜찮음

# 자료형 변환

- 대입 연산자의 왼쪽과 오른쪽 자료형이 다른 경우 **자료형 불일치(type mismatch)**가 발생
  - 이러한 경우에 앞에서 본 것처럼 오류가 발생하거나 자동으로 **자료형 변환(type conversion)**이 일어남
- 다음 표는 오류 또는 자동 형 변환이 일어나는 경우를 설명

구분	자료형 변환
큰 범위의 숫자를 표현할 수 있는 자료형 → 작은 범위의 자료형	데이터 손실이 발생할 수 있으므로 오류 발생
작은 범위 자료형 → 큰 범위 자료형	자동 자료형 변환 발생

# 자료형 변환

## □ 오류 또는 자동 형 변환 사례

```
int a = 1;    // 자료형 일치
double b = 1.1; // 자료형 일치함
long b = 1;   // 자료형 불일치. 자동으로 1을 1L로 변환
float c = 1;  // 자료형 불일치. 1.0f로 자동 변환
double d = 1.1f; // 자료형 불일치. 1.1로 자동 변환
float e = 1.1; // 오류 발생. double형 → float형
int f = 1.1;  // 오류 발생. double형 → int형
int g = 1L;   // 오류 발생. long형 → int형
```

- e에 저장하는 1.1이나 g에 저장하는 1L은 자료형이 다르지만, 실질적으로 float이나 int 범위 안에 들어가는 값이지만, 오류 발생

# 자료형 변환

- 강제 형 변환(**type casting**)이란 한 가지 자료형을 다른 자료형으로 강제로 변환시키는 것
- 큰 범위를 나타내는 자료형의 숫자 값을 작은 범위에 해당되는 자료형 변수에 저장할 때 오류 발생
- 이를 해결하려면 강제 형변환을 사용해야 함
- 값이 저장될 변수의 범위를 넘치지 않을 것 같으면 강제 형 변환을 통해 변환 후 사용할 수 있음
- 강제 형 변환 방법
  - 변수 = (변환 자료형) 값;
  - 변환 자료형은 왼쪽 변수와 같은 자료형 또는 왼쪽 변수 자료형으로 자동 형 변환 가능한 자료형

# 자료형 변환

```
float e = (float) 1.1; // 1.1을 1.1f로 강제 변환
```

- 변환해서 저장할 값이 왼쪽 변수 범위 내에 있으면 문제 없음

```
int num = (int) 1000L;  
byte b = (byte) 100;
```

- 왼쪽 변수의 범위에 포함되지 않는 강제 형 변환이 발생한다면?

```
jshell> int f = (int) 1.1; // 어떻게 되나?
```

# 자료형 변환

- 강제 형 변환은 값이 왼쪽 변수 범위 안에 속할 때만 사용하는 것은 아님
  - 범위를 넘치는 경우 논리적 오류
  - 만능이 아님. 프로그래머가 예측할 수 있을 때 사용할 것

```
long b = (long) 1; // 불필요함
float c = (float) 1; // 불필요함
double d = (double) 1.1f; // 불필요함
float e = (float) 1.1;
int f = (int) 1.1;
int g = (int) 1L;
```

# 강제 형 변환의 문제점

- byte 형으로 변수를 정의하고 범위 내(1)/외(128) 값을 저장해볼 것

```
jshell> byte b = 1;  
jshell> b = 128;
```

- 비슷하게 short형으로 변수를 정의하고 범위 내(128)/외(32768) 값을 저장해볼 것

```
jshell> short s = 128;  
jshell> s = 32768;
```

# 강제 형 변환의 문제점

## ▣ 강제 형 변환을 시켜본다면?

```
jshell> b = (byte) 128;  
jshell> b = (byte) 129;  
jshell> b = (byte) -129;  
jshell> b = (byte) -130;  
jshell> s = (short) 32768;  
jshell> s = (short) 32769;
```

## ▣ float형도 강제 형 변환을 이용해서 범위 밖 값을 저장해보기

```
// float의 최대 값은 약 3.4E38;  
jshell> float f = (float) 3.5E38;  
jshell> f = 1.4E-46;  
jshell> f = (float) 1.4E-46;
```



# 자료형 변환

형 변환 방법	설명
강제 형 변환	<ul style="list-style-type: none"><li>- 사용자가 직접 해야 함</li><li>- 변환 대상 자료형을 괄호 안에 적음</li><li>- 예) 실수를 정수로 강제 변환 (int) 1.1 → 1</li><li>- 큰 범위의 값을 작은 범위의 자료형으로 변환하려 할 때에는 오류가 발생하는데 이럴 때 강제 형 변환 기법을 사용해서 오류 제거 가능</li><li>- 자동 형 변환이 되는 경우에도 형 변환이 발생하는 것을 명확하게 인지시키기 위해 강제 형 변환 기법을 사용하는 경우도 있음</li></ul>
자동 형 변환	<ul style="list-style-type: none"><li>- 사용자가 직접 변환하지 않아도 됨</li><li>- 일반적으로 작은 범위의 값을 큰 범위의 자료형으로 승격시킬 때 발생</li><li>- 예) float x = 1; 같은 코드에서는 자동으로 승격해서 1.0f를 저장</li></ul>

# 숫자형 정리

---

- 정수형: byte, short, int, long
- 실수형: float, double
- long형 숫자값을 다른 정수형 int, short, byte형 변수에 넣는 것은 오류 발생 (강제 형 변환 필요)
- int형 숫자값을 int, short, byte 형 변수에 넣는 것은 허용하는 범위에 있다면 자동으로 저장됨
  - 허용 범위를 넘치는 값이 저장되면 오류 발생
- 범위를 넘치는 숫자 값을 강제 형 변환을 통해 저장하면 예측하지 못한 결과가 나타남
  - 정수, 실수 모두 예측 못한 결과가 나타남

# 산술(arithmetic) 연산

- 산술 연산
  - 수학의 사칙 연산
  - 똑같은 우선 순위를 가짐(곱셈/나눗셈 > 덧셈/뺄셈)
  - **작은 범위의 자료형과 큰 범위의 자료형의 연산사이의 결과 값은 큰 범위의 자료형임**
- 곱하기 기호는 '\*', 나누기 기호는 '/'
- 정수를 정수로 나누는 연산에서는 몫에 해당되는 정숫값을 반환
- 나눗셈에서 피연산자 중 한 개가 실수이면 결과는 실수
- 나머지 연산자(%)
  - 정수와 정수의 나눗셈에서 나머지만 반환

# 산술(arithmetic) 연산

- 다음 나눗셈과 나머지 연산 실행 해보기

```
jshell> 3 / 2;  
jshell> 2 / 3;  
jshell> 3 / 2.;  
jshell> 3. / 2;  
jshell> 3 % 2;  
jshell> 2 % 3;
```

- System.out.println()함수는 기본형을 출력할 수 있음

```
jshell> System.out.println((double)1 / 2);  
jshell> System.out.println((int)1.2 / 2);
```

# 최소값, 최대값 확인 및 System.out.printf()

- 때로는 자료형의 최소값과 최대값을 프로그램에서 확인하고 써야 할 경우가 있음
- 최소값과 최대값을 찾는 방법
  - 문서 파일을 찾아보고 직접 사용하는 것
  - 프로그래밍을 통해서 알아보는 방법
- byte의 최대/최소값 알아보기
  - 포장 클래스인 Byte를 활용
  - Byte 클래스에는 MIN\_VALUE와 MAX\_VALUE 속성 제공

```
System.out.printf("byte: min: %s, max: %s\n",  
                  Byte.MIN_VALUE, Byte.MAX_VALUE);
```

# 최소값, 최대값 확인 및 System.out.printf()

- System.out.printf() 함수는 서식에 맞춰 값을 화면에 출력하는 함수
  - 첫 번째 인자인 문자열이 서식
    - 서식 문자열은 문자열을 그대로 출력하지만, '%'로 시작하는 서식 지시어만 따로 처리
    - 서식 지시어는 특별한 문자를 출력하거나 인자의 값으로 대치해서 출력하는 용도
- 서식 지시어 처리 방법

The diagram illustrates the mapping of format specifiers to values in the printf statement. A box at the top contains the text "byte: min: %s, max: %s\n". Two arrows originate from this box: one points down to the "%s" following "min:" in the printf statement below, and the other points down to the "%s" following "max:". The printf statement is: "byte: min: %s, max: %s\n", Byte.MIN\_VALUE, Byte.MAX\_VALUE. An arrow also points from the Byte.MIN\_VALUE argument up to the "%s" following "max:".

```
"byte: min: %s, max: %s\n", Byte.MIN_VALUE, Byte.MAX_VALUE
```

# 최소값, 최대값 확인 및 System.out.printf()

서식 지시어	설명
%%	% 글자를 화면에 출력
%n 또는 \n	줄바꿈 문자 출력
%d	인자로 전달되는 정숫값을 출력 (byte, short, int, long)
%f	인자로 전달되는 실숫값을 출력 (float, double)
%c	인자로 전달되는 문자를 출력(char)
%s	인자로 전달되는 내용을 문자열 형태로 변환해서 출력
%h	인자의 해시코드(hashcode)를 출력. 주로 참조값을 출력할 때 사용

# 최소값, 최대값 확인 및 System.out.printf()

## □ 문자열로 바꿔서 출력해볼 것

```
jshell> System.out.printf("byte: min: %s, max: %s\n", Byte.MIN_VALUE, Byte.MAX_VALUE);
```

## □ float의 최소값과 최대값을 출력해볼 것. 실숫값을 출력할 것이므로 "%f"와 "%s"를 이용해서 출력해보고 차이를 확인할 것

```
jshell> System.out.printf("float: min: %f, max: %f\n", Float.MIN_VALUE, Float.MAX_VALUE);
```

```
jshell> System.out.printf("float: min: %s, max: %s\n", Float.MIN_VALUE, Float.MAX_VALUE);
```



# 사용자로부터 키보드 입력 받기

---

## □ 외부 패키지의 클래스 사용하기

- 자바 9는 약 6000여 개 클래스를 제공
- 이름 충돌이 발생할 수 있는데, 이를 피하기 위해 패키지를 사용
- 패키지는 관련 있는 클래스들을 묶어서 관리하는 단위
- 패키지에 대해서는 나중에 학습. 여기서는 사용하는 방법만 설명

# 외부 패키지 클래스 사용

- 사용자로부터 입력을 받으려면 java.util 패키지의 Scanner 클래스를 사용해야 함
  - 이렇게 특정 패키지에 있는 클래스를 사용하려면 자바 컴파일러나 JShell에 알려줘야 함
- 클래스 명시 방법 두 가지
  - 패키지 이름과 클래스 이름을 명시  
패키지\_이름.클래스\_이름;  
  
패키지\_이름.클래스\_이름 변수\_이름;
  - 예  

```
java.util.Scanner sc = new java.util.Scanner();
```
  - 매번 긴 이름을 입력해야 하는 것이 불편함

# 외부 패키지 클래스 사용

- import 키워드를 사용해서 패키지의 클래스를 명시

```
import 패키지이름.클래스이름;
```

```
클래스이름 변수이름;
```

□ 예

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

```
Scanner sc = new Scanner();
```

# 외부 패키지 클래스 사용

- 한 개 패키지에는 여러 개 클래스가 있음
- 프로그램에서 사용하는 클래스들이 같은 패키지에 있다면, 여러 번 import 해야 함
  - 예: java.util 패키지의 여러 클래스를 이용함

```
import java.util.Vector;  
import java.util.Random;  
import java.util.Scanner;  
import java.util.Date;
```

- 간단하게 줄여서 패키지의 모든 클래스를 import 함

```
import java.util.*;
```

# 외부 패키지 클래스 사용

- 컴파일러를 사용할 때에는 java.lang 패키지에 있는 클래스를 제외하고, 필요한 패키지의 클래스들을 모두 import해야 함
- JShell에서는 시작할 때 자주 사용하는 패키지들이 미리 import 되어 있음
  - JShell에서 /imports 명령을 사용하면 미리 import 된 패키지나 클래스 확인 가능
- 다른 패키지의 클래스를 사용하려면 import 할 것

```
jshell> /imports
|   import java.io.*
|   import java.math.*
|   import java.net.*
|   import java.nio.file.*
|   import java.util.*
|   import java.util.concurrent.*
|   import java.util.function.*
|   import java.util.prefs.*
|   import java.util.regex.*
|   import java.util.stream.*
```

# 사용자로부터 입력 받기

- java.util.Scanner를 이용하면 키보드, 문자열, 파일 등 다양한 입력 소스(input source)로부터 정해진 형식에 맞게 입력 받을 수 있음
  - 기본적으로 공백 문자(white space)로 분리되는 단어
  - 다양한 함수를 이용해서 정수, 실수 등을 입력 받음
- Scanner 클래스는 객체를 생성하면서 어떤 입력 소스로부터 받을 것인지 지정 가능
  - 키보드를 통해 입력 받는다면 생성자에 System.in을 전달

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

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

# 사용자로부터 입력 받기

## ▣ Scanner 클래스가 제공하는 입력용 함수

함수	설명
String next()	다음 입력을 문자열 형태로 받음. 공백 문자를 만날 때까지 입력된 모든 문자들을 문자열 형태로 반환
String nextLine()	줄바꿈 문자를 만날 때까지 입력된 모든 문자들을 있는 그대로 문자열 형태로 반환 (줄바꿈 문자 제거)
byte nextByte()	다음 입력을 byte형 정수로 받음
short nextShort()	다음 입력을 short형 정수로 받음
int nextInt()	다음 입력을 int형 정수로 받음
long nextLong()	다음 입력을 long형 정수로 받음

# 사용자로부터 입력 받기

- Scanner 클래스가 제공하는 입력용 함수

함수	설명
float nextFloat()	다음 입력을 float형 실수로 받음
double nextDouble()	다음 입력을 double형 실수로 받음

- 정숫값 한 개와 한 줄의 문자열을 입력 받는 코드

```
int num = sc.nextInt();  
String line = sc.nextLine();
```

- 사용자로부터 키보드 입력을 받을 때에는 사용자 입력이 언제 끝나는지 확인할 수 있어야 함
  - 일반적으로 엔터키를 누르면 입력이 종료되었음을 파악하고 입력 처리



# 사용자로부터 입력 받기

---

- `nextLine()`을 제외한 나머지 `next~()` 함수들은 공백 문자가 아닌 첫 글자부터 입력 받기 시작하며, 그 다음에 나오는 첫 번째 공백 문자에서 입력을 멈춤
- `nextLine()` 함수는 공백 문자를 포함해서 모든 문자들을 입력 받음
  - 다음 줄바꿈 문자를 입력 받으면 멈추며, 해당 줄바꿈 문자는 제거한 후에 문자열을 반환함

# 실습문제 1: 단어와 숫자 입력 받기

---

## □ 문제

- 사용자로부터 문자열과 정수를 순차적으로 입력 받고, 그 내용을 바로 화면에 출력하는 프로그램을 작성

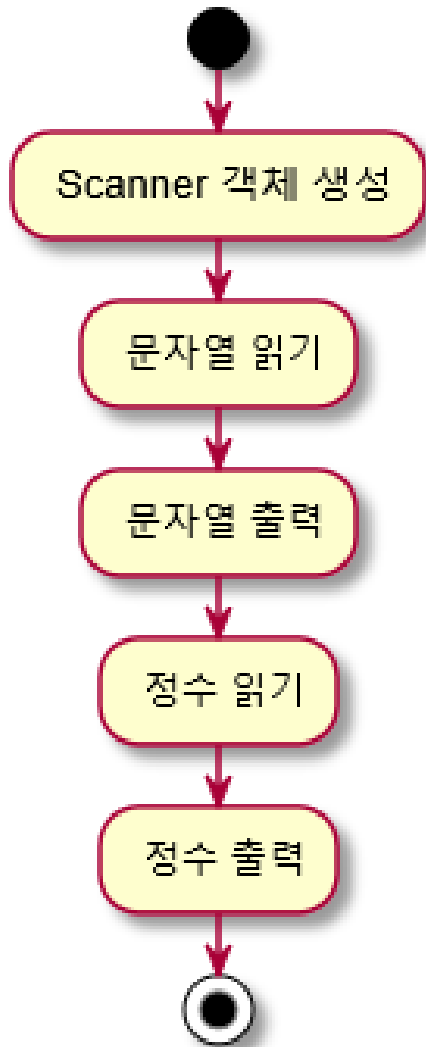
## □ 요구사항

- 문자열을 입력 받고 화면에 출력. 그리고 다시 정수를 입력 받고 출력
- 문자열은 1개 단어로 구성되고, 정수는 int형 범위 내의 값

## □ 해결

- 문자열과 정수를 한 개씩 입력 받고 화면에 출력
- 입력은 Scanner 클래스 사용
- 문자열은 next(), 정수는 nextInt() 함수로 입력 받음

# 실습문제 1: 단어와 숫자 입력 받기



□ Scanner 클래스의 객체를 생성

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

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

□ 다음에는 문자열을 읽음

- 단어를 입력

- next() 또는 nextLine()을 사용

```
String s = sc.next();
```

□ 단어 입력 후 엔터 키

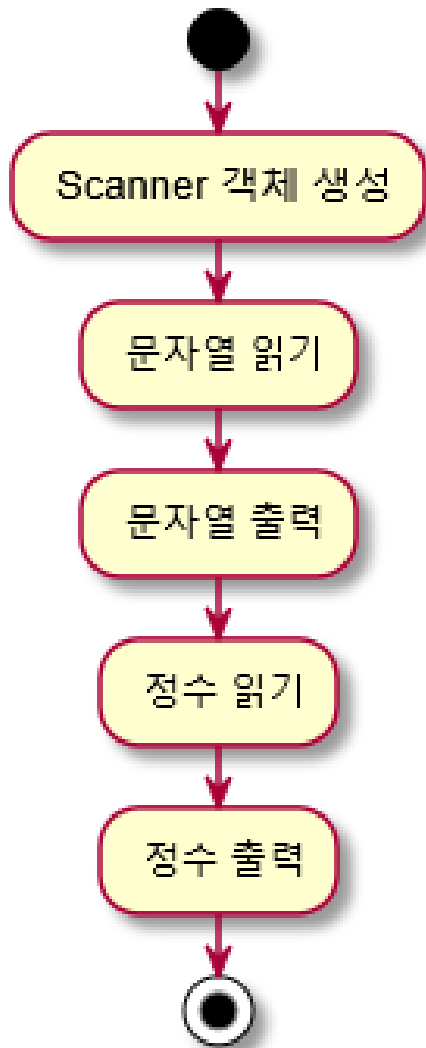
□ 다음에 출력

```
System.out.println("출력 문자열: " + s);
```

# 실습문제 1: 단어와 숫자 입력 받기

## □ 정수 입력 받고 화면에 출력

```
int num = sc.nextInt();  
System.out.println("출력 정수: " + num);
```



# 실습문제 1: 단어와 숫자 입력 받기

## ▣ 최종 코드

```
// Scanner1.java
import java.util.Scanner;

class Scanner1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        System.out.println("출력 문자열: " + s);
        int num = sc.nextInt();
        System.out.println("출력 정수: " + num);
    }
}
```

# 실습문제 1: 단어와 숫자 입력 받기

---

## □ 실행 결과

- 실행 후 "hello"와 "1"을 입력하고 제대로 출력되는지 확인
- " hello"와 " 1"을 입력해보고 확인

# 실습문제 2: `nextLine()` 함수로 입력 받기

---

## □ 문제

- 문자열을 문장 단위로 입력 받고 화면에 출력하는 프로그램을 작성
- 두 번 실행시키면서 "hello 1"과 " hello 1"을 입력해 보고 어떻게 출력되는지 확인할 것

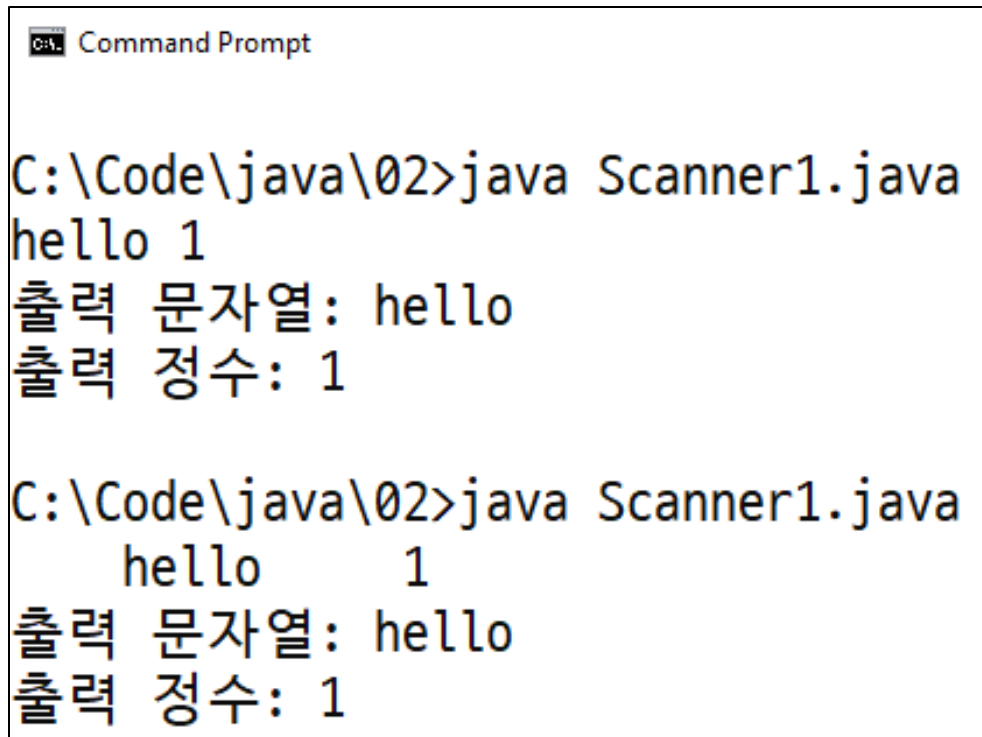
## □ 요구사항

- `Scanner` 클래스의 `nextLine()`을 이용
- 프로그램의 이름은 `ReadLine`으로 할 것

# 입력 버퍼(Input Buffer)

## □ Scanner1 프로그램을 다시 실행

- "hello 1" 또는 " hello 1" 형태로 입력할 것



```
Command Prompt

C:\Code\java\02>java Scanner1.java
hello 1
출력 문자열: hello
출력 정수: 1

C:\Code\java\02>java Scanner1.java
 hello 1
출력 문자열: hello
출력 정수: 1
```

- next() 함수는 사용자 입력을 기다리지만, nextInt()함수는 바로 1을 반환. 왜?



# 입력 버퍼(Input Buffer)

---

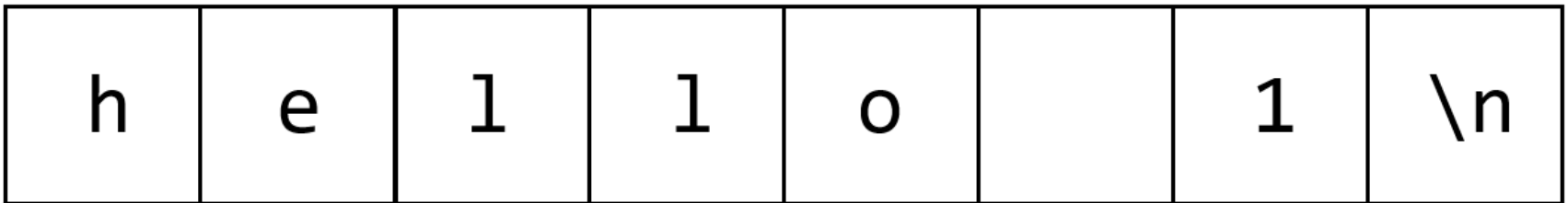
- 키보드로 입력할 때에는 입력 버퍼가 사용됨
- 입력 버퍼는 사용자가 입력한 내용을 임시로 저장해 두는 메모리 공간
- next~()함수들이 실행되고 입력을 기다릴 때 사용자가 입력한 내용은 모두 입력 버퍼에 저장됨
- 입력 버퍼에서는 어떤 글자까지 처리했는지 파악하기 위해 입력 포인터(input pointer)를 활용
- 입력 포인터는 입력 버퍼에서 다음에 사용하게 될 글자의 시작 위치를 나타내는 주소(위치)값

# 입력 버퍼(Input Buffer)

```
String s = sc.next();  
int n = sc.nextInt();
```

- 사용자가 "hello 1"이라고 입력하고 엔터 키를 눌렀다면?

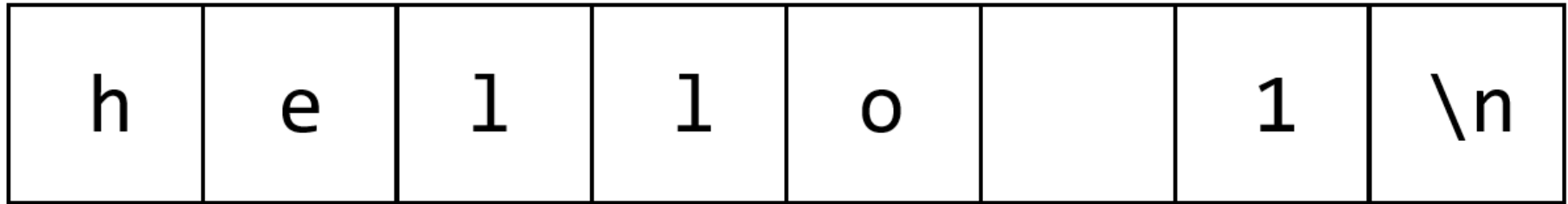
입력 버퍼



입력 포인터

# 입력 버퍼(Input Buffer)

- next() 함수가 실행된 후  
입력 버퍼



입력 포인터

# 입력 버퍼(Input Buffer)

- nextInt() 함수가 실행될 때, 실제 입력을 받는 것

입력 버퍼

h	e	l	l	o		1	\n
---	---	---	---	---	--	---	----



입력 포인터

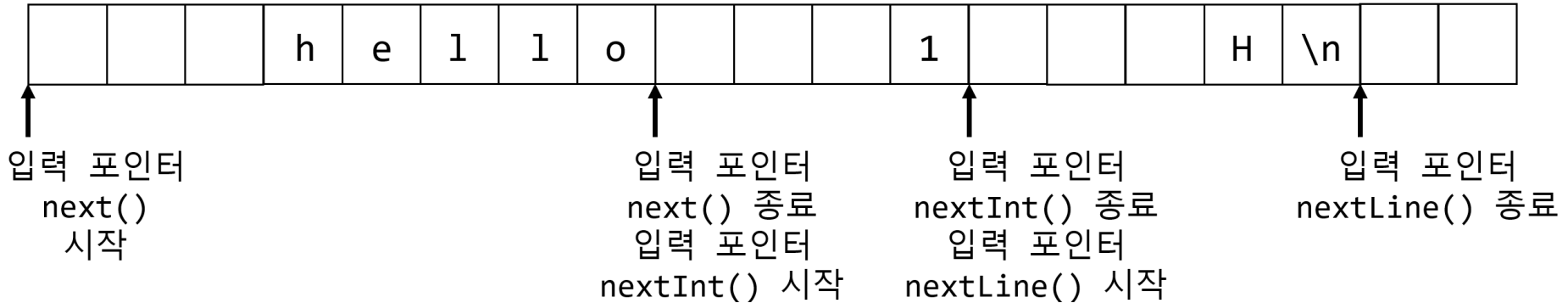
# 입력 버퍼(Input Buffer)

```
// InputBuffer.java
import java.util.Scanner;

class InputBuffer {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        System.out.println("첫 번째 문자열: " + s);
        int n = sc.nextInt();
        System.out.println("출력 정수: " + n);
        String s1 = sc.nextLine();
        System.out.println("나머지 문자열: " + s1);
    }
}
```

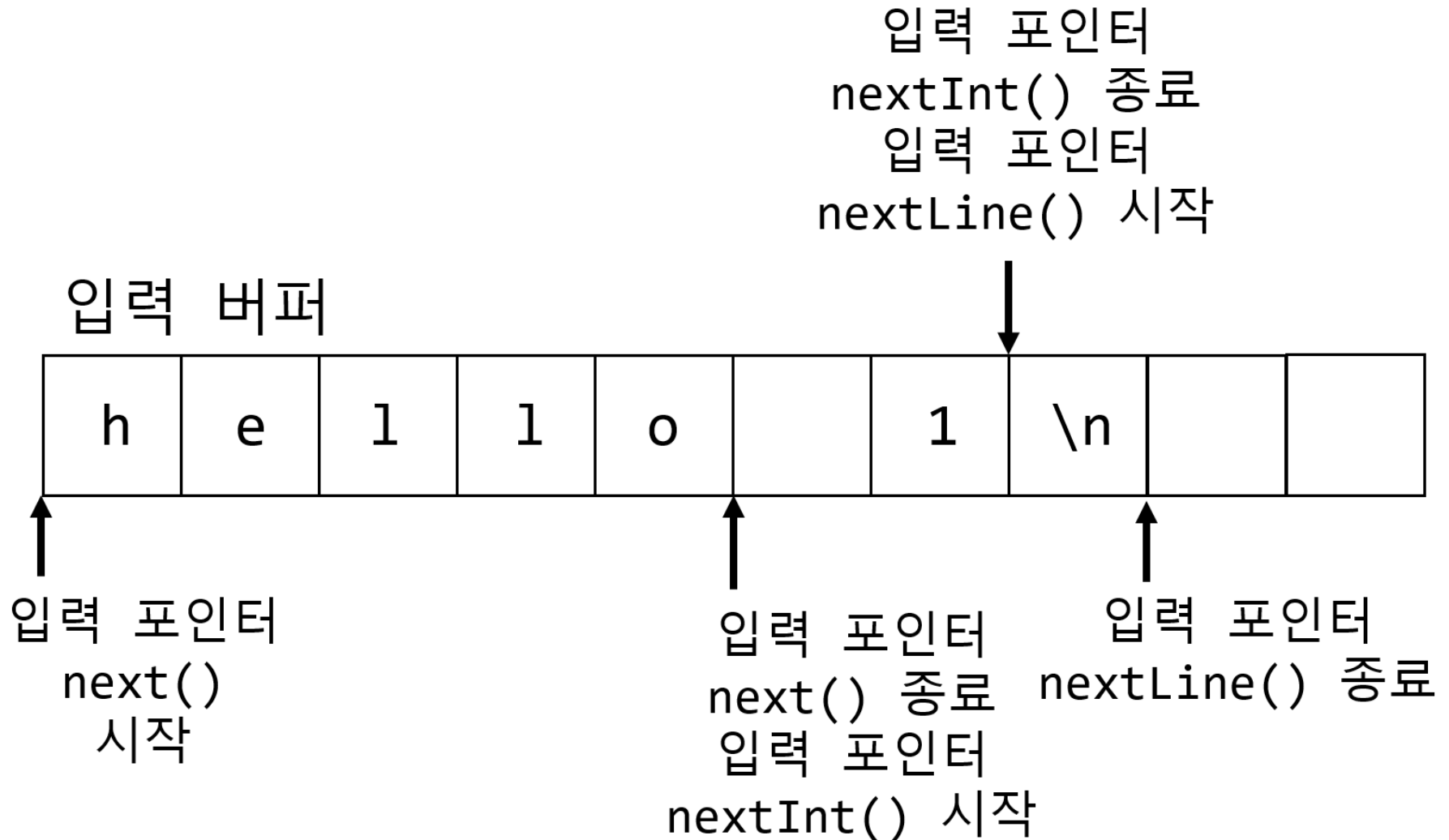
# 입력 버퍼(Input Buffer)

□ "hello 1 H"를 입력



# 입력 버퍼(Input Buffer)

□ "hello 1"을 입력



# 실습문제 3

---

## □ 문제

- 화씨 23도를 섭씨 온도로 변환하는 프로그램을 작성

## □ 요구사항

- 결과 값은 소수 둘 째 자리까지 출력



# 실습문제 4

---

## □ 문제

- 두 개의 숫자를 입력 받아서 합을 구하는 프로그램을 작성
- "1 2"를 입력해볼 것

## □ 요구사항

- 사용자는 정확하게 두 개의 숫자를 입력
- 두 개의 숫자라고 했으므로, 정수 또는 실수가 입력 될 지 정해지지 않는
- 문자열로 입력 받고 Integer클래스의 parseInt() 또는 Float클래스의 parseFloat() 함수를 사용해서 문자열을 숫자 값으로 변환해서 사용할 것

# Integer클래스의 parseInt() 함수 사용법

- 기본형 중에서 숫자형을 포장하는 클래스들과 Boolean 클래스는 문자열을 값으로 변환해주는 함수들을 포함
  - 이러한 함수들은 parseType() 형태로 구성
    - Type은 자료형
    - 예를 들어 정수로 구성된 문자열을 정숫값으로 변환해주는 함수는 Integer클래스의 parseInt()
    - Short.parseShort(), Double.parseDouble(), Boolean.parseBoolean() 등이 있음
- 객체를 생성하지 않고  
클래스이름.parseType() 형태로 사용함

# Integer클래스의 parseInt() 함수 사용법

```
// 숫자로_구성된_문자열을 정수로 바꾸어서 반환함  
int value = Integer.parseInt(숫자로_구성된_문자열);
```

- parseInt()함수에 전달되는 문자열은 정수로만 구성되어 있어야 함
  - 잘못된 입력이 전달되면 제대로 함수가 동작 안함
  - 입력이 정확하게 주어진다고 가정할 것

```
jshell> int value = Integer.parseInt("13");  
value ==> 13  
jshell> System.out.println(value);  
13
```

# Integer클래스의 parseInt() 함수 사용법

- Short.parseShort(), Double.parseDouble(), Boolean.parseBoolean() 함수들도 Integer.parseInt()와 비슷하게 사용됨

```
jshell> double pi = Double.parseDouble("3.1415");  
pi ==> 3.1415  
jshell> System.out.println(pi);  
3.1415
```

# 실습문제 5

---

## □ 문제

- 사용자로부터 화씨 온도를 입력 받고, 섭씨 온도로 변환한 후에 화면에 출력하는 프로그램을 작성

## □ 요구사항

- 사용자가 입력하는 화씨 온도는 실수 값
- 변환된 섭씨 온도는 소수 첫 째 자리까지만 출력
- 사용자는 화씨 온도를 제대로 입력할 것이라고 가정

# 연산자

---

- 자바에는 약 30 여개 연산자가 있음
  - 사칙 연산자, 나머지 연산자
  - 비교 연산자(같다, 다르다, 크다, 작다 등)
  - 논리 연산자
  - 자바에 특화된 연산자
  - 우선 여기서는 산술 연산 관련된 것들 위주로 살펴봄
- 부호 연산자
  - 양수와 음수를 표현하는 부호 연산자 (+, -)
  - 대표적인 단항 연산자
  - 변수 앞에도 연산자 사용 가능
    - 변수값을 바꾸는 것이 아니라 결과 값만 부호 변경

# 부호 연산자

```
jshell> +12;  
$81 ==> 12
```

```
jshell> -12;  
$82 ==> -12
```

```
jshell> int a = 3;  
a ==> 3
```

```
jshell> int b = +a;  
b ==> 3
```

```
jshell> int c = -a;  
c ==> -3
```

```
jshell> a;  
3
```

# 증감 연산자

- 변숫값을 1 증가시키거나 감소시키는 연산자
- ++와 --를 사용
- 변수와 함께 사용해야 하고 변수 자신에 1을 더해서 다시 저장하거나 1을 빼고 다시 저장하는 것과 같음

```
a++; // a = a + 1  
a--; // a = a - 1
```

- 증감 연산자는 변수 앞 또는 뒤에 위치할 수 있고, 장소에 따라 다른 결과가 나타남
- 앞에 위치할 때
  - 변숫값이 사용(evaluation)되기 전에 1을 증가 또는 감소
- 뒤에 위치할 때
  - 변숫값 사용 후 1을 증가 또는 감소



# 증감 연산자

---

```
jshell> int a = 4;  
a ==> 4
```

```
jshell> int b = 4;  
b ==> 4
```

```
jshell> int c = ++a + 2;  
c ==> 7
```

```
jshell> int d = b++ + 2;  
d ==> 6
```

# 증감 연산자

- 증감 연산자는 가능하면 독립적으로만 사용할 것

- 나쁜 예

```
int a = 4;  
int b = ++a + a++ - --a;
```

- 좋은 예

```
a++;  
++a;  
a++;  
--a;  
for (int i = 0; i < 10; i++) {  
    // 코드  
}
```

# 복합 연산자

- ▣ 대입 연산자와 산술 연산자를 함께 묶어서 줄여 쓸 수 있는 방법
- ▣  $+=$ ,  $-=$ ,  $*=$ ,  $/=$ ,  $\%=$

사용 예	설명
$a += b$	$a = a + b$ 를 축약시킨 코드
$a -= b$	$a = a - b$ 를 축약시킨 코드
$a *= b$	$a = a * b$ 를 축약시킨 코드
$a /= b$	$a = a / b$ 를 축약시킨 코드
$a \% = b$	$a = a \% b$ 를 축약시킨 코드

# 복합 연산자

```
jshell> int a = 1;  
jshell> a += 1;    // a = a + 1;  
jshell> System.out.println(a);  
2
```

```
jshell> float b = 3.2f;  
jshell> b -= a;    // b = b - a;  
jshell> System.out.println(b);  
1.2
```

```
jshell> b *= a;    // b = b * a;  
jshell> System.out.println(b);  
2.4
```

# 복합 연산자

```
jshell> b /= a;    // b = b / a;
```

```
jshell> System.out.println(b);  
1.2
```

```
jshell> a *= 5;    // a = a * 5; --> 10
```

```
jshell> a %= 3;    // a = a % 3; --> 10 % 3
```

```
jshell> System.out.println(a);  
1
```

# 연산자 우선 순위

순위	연산자	설명
1	(), [], .	괄호, 배열 인덱스, 종속 연산자
2	!, ~, ++, --, +, -, (type)	부정(논리), 부정(비트), 증가, 감소, 부호, 형변환
3	*, /, %	곱셈, 나눗셈, 나머지
4	+, -	덧셈, 뺄셈
5	<<, >>, >>>	비트 단위의 쉬프트(shift)
6	<, <=, >, >=	작다, 작거나 같다, 크다, 크거나 같다
7	==, !=	같다, 다르다
8	&	비트 단위의 논리 연산 (and)
9	^	비트 단위의 논리 연산 (xor)
10		비트 단위의 논리 연산 (or)
11	&&	논리곱
12		논리합
13	?:	조건
14	=, +=, -=, *=, /=, %= <<=, >>=, &=, ^=, ~=	대입, 복합 연산자

# 연산자 우선 순위

우선 순위	종류	대표 연산자
1	단항 연산자	부호, 증감 연산자
2	산술 연산자	덧셈, 뺄셈, 곱셈, 나눗셈, 나머지 등
3	비교 연산자	크기 비교, 같다, 틀리다
4	논리 연산자	논리곱, 논리합
5	조건 연산자	?:
6	대입 연산자	=, 복합 연산자

# 배열(Array)

## 배열의 필요성

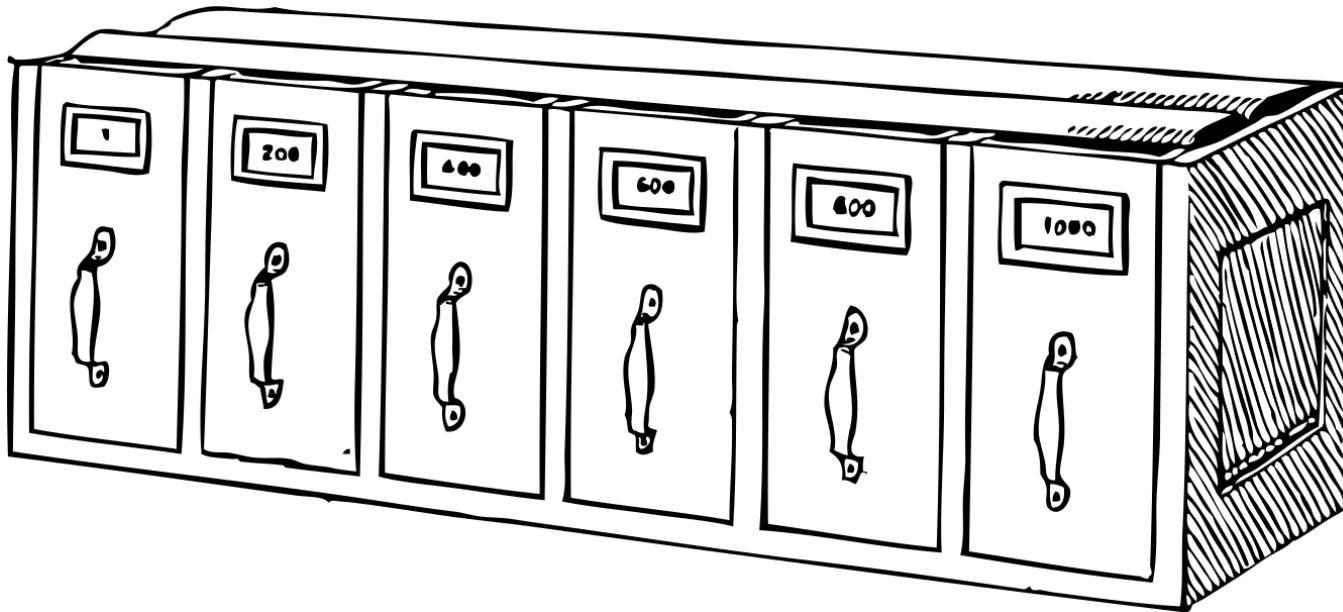
- 학생 100명의 성적을 저장한 후에 총점과 평균을 구하는 프로그램을 작성한다고 가정
  - 변수를 student0, student1, ..., student98, student99로 정의
  - 총점을 구하려면 100개 변수의 합을 구해야 함
  - 평균은 총점을 다시 100으로 나눔
- 만약 학생이 150명으로 늘어난다면?
  - student100~student149까지의 변수를 추가
  - 덧셈을 하는 코드에서도 50개 변수를 더 써야 함
- 비효율적
  - 변수를 사람 수 만큼 만들어야 함
  - 여러 개 변수에 작업하는 코드가 계속 수정됨



# 배열(Array)

## 배열

- 연관된 자료들을 여러 개 사용할 때 변수 이름 한 개로 여러 값을 저장하고 사용할 수 있음
  - 많은 변수들을 사용해야 하는 문제를 단순하게 처리할 수 있도록 해줌
- 배열은 사물함과 비슷함

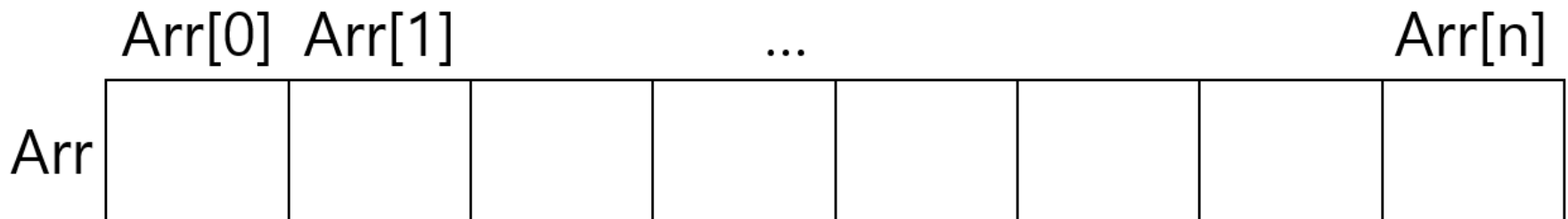


# 배열(Array)

- 배열은 사물함처럼 여러 개의 변수를 붙여놓은 것
  - 배열도 이름을 한 개로 붙여서 사용
  - 배열을 구성하는 각 변수가 요소(element)
  - 배열의 요소는 0, 1처럼 번호를 붙여서 사용(인덱스)
  - 자바에서 인덱스는 0부터 시작

배열이름[인덱스]

- 배열의 메모리 공간



# 배열(Array)

배열 사용하지 않는 경우

student0	90
student1	80
⋮	⋮
student99	85

student1	80
student0	90
⋮	⋮
student98	85

배열 사용하는 경우

90	student[0]
80	student[1]
⋮	⋮
85	student[99]

# 배열(Array)

## □ 합을 구하는 코드

### ■ 배열 사용하지 않을 때

```
int sum = 0;

sum += student0;
sum += student1;

sum += student99;
```

### ■ 배열 사용하지 않을 때

```
int sum = 0;

sum += student[0];
sum += student[1];

sum += student[99];
```

### ■ 반복문 사용할 때

```
int sum = 0;

for (int num = 0; num <= 99; num++) {
    sum += student[num];
}
```

# 배열 생성 방법

## □ 배열 선언

- 코드에서 특정 이름을 가진 배열을 소개하는 것
- 공간 확보는 아님 (기본형과 다름)

```
자료형[] 변수명;  
자료형 변수명[];
```

- Jshell에서 선언해봄

```
jshell> int[] arr;  
arr ==> null
```

```
jshell> int arr2[];  
arr2 ==> null
```

# 배열 생성 방법

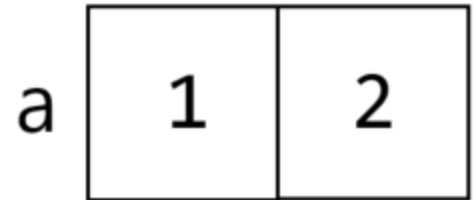
## ▣ 배열 메모리 공간 확보 (두 가지)

### ■ 초기값 지정

```
자료형[] 변수명 = { 초기값1, 초기값2, ..., 초기값n };  
자료형 [] 변수명 = { 초기값1, 초기값2, ..., 초기값n };  
자료형 변수명[] = { 초기값1, 초기값2, ..., 초기값n };
```

### ■ 예

```
jshell> int[] arr = { 1, 2 };
```



### ■ 이미 선언된 변수에 초기값 지정 못함

```
jshell> int[] arr;  
jshell> arr = { 1, 2 }; // 오류 발생
```

# 배열 생성 방법

- new 사용

```
자료형[] 변수명; // 배열 선언
```

```
// 배열 선언 후 공간 확보
```

```
변수명 = new 자료형[배열_크기];
```

- 예

```
jshell> int[] arr;  
arr ==> null
```

```
jshell> arr = new int[2];  
arr ==> int[2] { 0, 0 }
```

# 배열 생성 방법

## □ 선언과 공간 확보를 함께

```
자료형[] 변수명 = new 자료형[배열 크기];  
자료형 [] 변수명 = new 자료형[배열 크기];  
자료형 변수명[] = new 자료형[배열 크기];
```

## ■ 예

```
jshell> int[] arr = new int[2];  
arr ==> int[2] { 0, 0 }
```



# 배열 사용

## □ 배열 사용 방법

- 배열 요소는 인덱스를 이용해서 접근

## □ 예

```
jshell> arr[0] = 1;  
jshell> arr[1] = 2;  
jshell> System.out.println(arr[0]);  
1  
jshell> System.out.println(arr[1]);  
2
```

## □ 배열의 크기 확인

- 배열이름.length를 이용

## □ 예

```
jshell> System.out.println(arr.length)  
2
```

# 배열 사용

---

- 배열에 데이터를 넣거나 값을 사용하기 위해 접근할 때 정확한 자료형이 사용되는지 확인
- 배열의 인덱스 번호가 범위를 벗어나면 **ArrayIndexOutOfBoundsException** 오류 발생

# 실습 문제 6

---

## □ 문제

- 배열을 1.1, 2.3, 4.7, 7.5로 초기화시켜 생성하고, 배열의 크기 및 요소들을 화면에 출력

## □ 요구사항

- 초기화를 시키라고 했으므로, 배열을 생성하면서 값을 지정할 것

# 실습 문제 7

---

## □ 문제

- 4개짜리 배열을 먼저 생성하고, 각 요소에 1.1, 2.2, 3.3, 4.4를 저장한다. 그리고 배열의 크기와 각 요소를 화면에 출력

## □ 요구사항

- 값을 보면 double형으로 지정