

Chapter

# 02

## 변수와 타입

### 02-1. 변수

- 0. 시작하기 전에
- 1. 변수 선언
- 2. 값 저장
- 3. 변수 사용
- 4. 변수 사용 범위
- 5. 키워드로 끝내는 핵심 포인트

# 0. 시작하기 전에

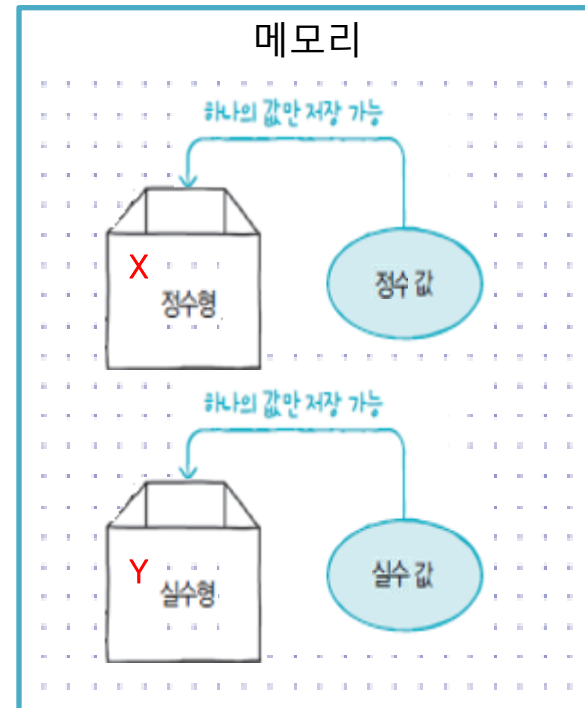
[핵심 키워드] : 변수, 변수 선언, 변수 사용, 변수 사용 범위

## [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)

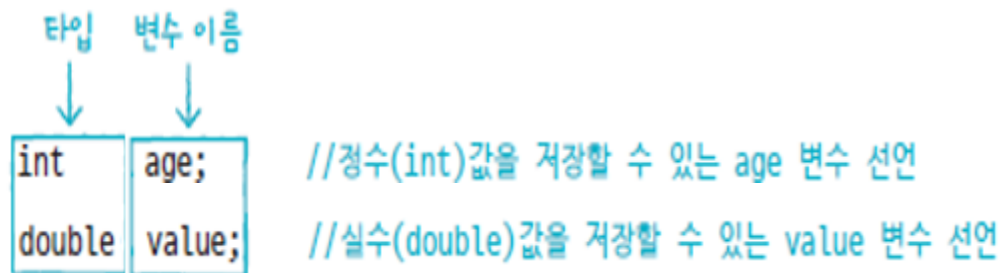
- 값을 저장할 수 있는 메모리의 특정 번지에 붙여진 이름
- 변수 통해 해당 메모리 번지에 하나의 값 저장하고 읽을 수 있음
- 변수는 정수, 실수 등 다양한 타입의 값을 저장할 수 있음



# 1. 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



# 1. 변수 선언

## ❖ 변수 이름

- 자바 언어에서 정한 명명 규칙 따라서 작성

작성 규칙	예
첫 번째 글자는 문자이거나 '\$', '_' 이어야 하고 숫자로 시작할 수 없습니다(필수).	가능: price, \$price, _companyName 불가능: 1v, @speed, \$#value
영어 대소문자를 구분합니다(필수).	firstname과 firstName은 다른 변수
첫 문자는 영어 소문자로 시작하되, 다른 단어가 붙을 경우 첫 문자를 대문자로 합니다(관례).	maxSpeed, firstName, carBodyColor
문자 수(길이)의 제한은 없습니다.	
자바 예약어는 사용할 수 없습니다(필수).	

# 1. 변수 선언

## ❖ 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

## 2. 값 저장

### ❖ 값을 저장할 경우 대입 연산자 (=) 사용

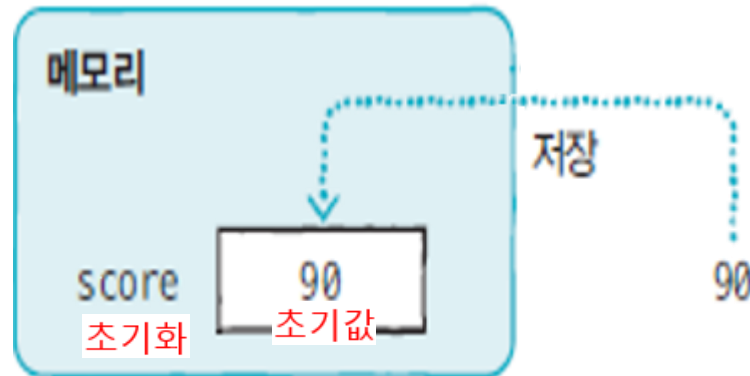
- 변수를 선언하고
- 대입 연산자를 사용해서 오른쪽의 값을 왼쪽의 변수에 저장

```
int score;           //변수 선언
score = 90;          //값 저장
```



### ❖ 변수 초기화

- 변수에 최초로 값이 저장될 때
- 메모리에 변수가 생성
- 이것을 변수 초기화라 하고
- 이 때의 값을 초기값이라고 함



## 2. 값 저장

- 초기화를 하지 않은 변수는 메모리에서 값을 읽을 수 없음

```
int value;           //변수 value 선언  
int result = value + 10; //변수 value 값을 읽고 10을 더해서 변수 result에 저장
```



```
int value = 30;      //변수 value가 30으로 초기화됨  
int result = value + 10; //변수 value 값(30)을 읽고 10을 더해서 변수 result에 저장
```

실습: VariableInitializationExample.java



## 2. 값 저장

실습: VariableInitializationExample.java

```
1 package sec01.exam01;
2
3 public class VariableInitializationExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         int value;
8         int result = value + 10;
9
10        System.out.println(result);
11    }
```

### 3. 변수 사용

❖ 변수 사용: 변수의 값을 이용해서, 출력문이나 연산식을 수행하는 것

- 예시

```
int hour = 3;  
int minute = 5;
```

- 출력문에서 변수 값 사용

```
System.out.println(hour + "시간 " + minute + "분"); //변수 hour와 minute 값을 출력: 3시간 5분
```

- 연산식 내부에서 변수 값 사용

```
int totalMinute = (hour*60) + minute;
```

실습: VariableUseExample.java

### 3. 변수 사용

실습: VariableUseExample.java

```
1 package sec01.exam02;
2
3 public class VariableUseExample {
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6         int hour = 3;
7         int minute = 5;
8         System.out.println(hour + "시간" + minute + "분");
9
10        int totalMinute = (hour*60) + minute;
11        System.out.println("총"+totalMinute + "분");
12
13    }
```

### 3. 변수 사용

실습: VariableExchangeExample.java

❖ 변수 값 복사: 변수의 값을 다른 변수에 저장



```
int x = 10;    //변수 x에 10을 저장
int y = x;     //x에 저장된 값을 변수 y에 복사(저장)
```

```
1 package sec01.exam03;
2
3 public class VariableExchangeExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         int x=3;
8         int y=5;
9         System.out.println("x: " + x + ", y: " + y );
10        int temp = x;
11        x=y;
12        y=temp;
13        System.out.println("x: " + x + ", y: " + y);
14    }
```

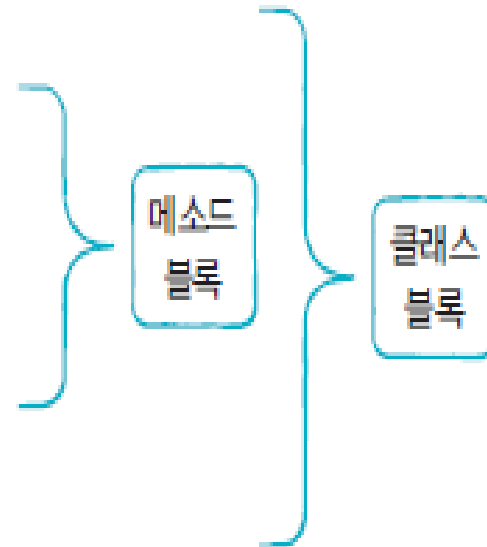
## 4. 변수 사용 범위

### ❖ 로컬 변수 (Local Variable)

- 메소드 블록 내에서 선언된 변수를 로컬 변수라고 함

```
public class VariableExample {  
    public static void main(String[] args) {  
        int value = 10;           //로컬 변수 value  
        int sum = value + 20;      //로컬 변수 sum  
        System.out.println(sum);  
    }  
}
```

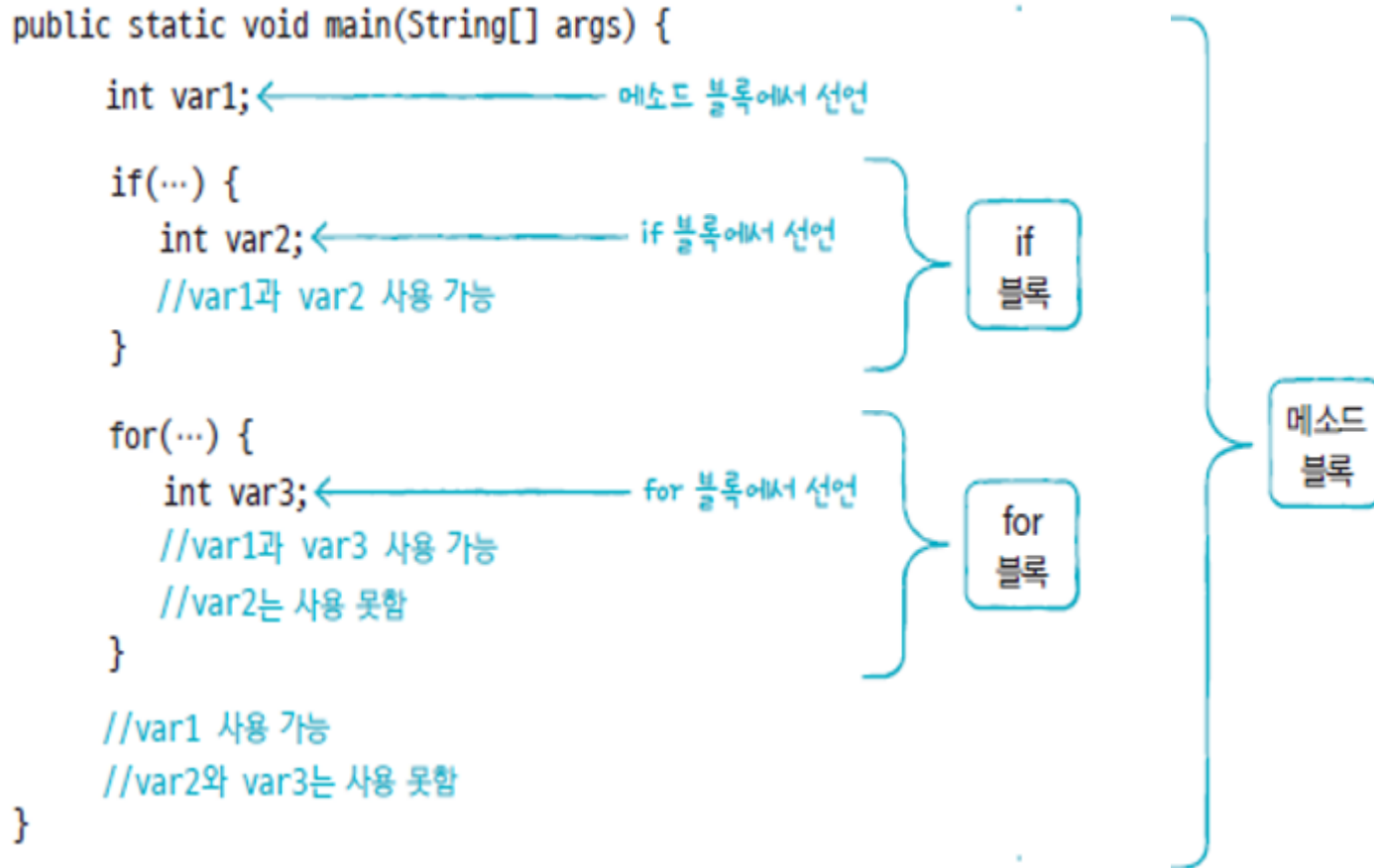
이 위치에서는 value와 sum 변수는 사용 못함



- 로컬 변수는 메소드 블록 내에서만 사용되고 메소드 실행이 끝나면 자동 삭제됨

## 4. 변수 사용 범위

- 로컬 변수는 해당 중괄호 블록 내에서만 사용 가능



## 4. 변수 사용 범위

### ■ 다음 사항에 주의

- 변수가 어떤 범위에서 사용될 것인지 고려하여 선언 위치 결정할 것
- 메소드 블록 전체에서 사용하려는 경우 메소드 블록 첫머리에 선언
- 특정 블록 내부에서만 사용하려는 경우 해당 블록 내에 선언

실습: VariableScopeExample.java

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    int v1=15;  
    if(v1>10){  
        int v2;  
        v2=v1-10;  
    }  
    int v3=v1+v2+5;  
    //v2변수를 사용할 수 없음,에러  
}
```

## 5. 키워드로 끝내는 핵심 포인트

- **변수**: 값을 저장할 수 있는 메모리 번지에 붙인 이름. 변수를 통해 프로그램은 메모리 번지에 값을 저장하고 읽을 수 있다.
- **변수 선언**: 변수에 어떤 타입의 데이터를 저장할지 그리고 변수 이름이 무엇인지를 결정하는 것
- **변수 사용**: 변수의 값을 읽거나 변경하는 것. 변수는 출력문이나 연산식 내부에서 사용되어 변수에 저장된 값을 출력하거나 연산에 사용한다.
- **변수 사용 범위** : 변수는 자신이 선언된 위치에서 자신이 속한 블록 내부까지만 사용이 가능하고 밖에서는 사용할 수 없다.



Chapter

# 02

## 변수와 타입

### 02-2. 기본 타입

- 0. 시작하기 전에
- 1. 정수 타입
- 2. 실수 타입
- 3. 논리 타입
- 4. 키워드로 끝내는 핵심 포인트

## 0. 시작하기 전에

[핵심 키워드] :정수 타입, char 타입, string 타입, 실수 타입, boolean 타입

[핵심 포인트]

- 변수 타입에 따라 변수에 저장할 수 있는 값의 종류와 허용 범위가 달라진다.
- 그래서 타입의 종류와 허용 범위에 대해서 학습한다.

❖ 기본 타입 (Primitive Type)

- 자바 언어는 정수, 실수, 논리값 저장하는 총 8 개의 기본 타입을 제공

구분	저장되는 값에 따른 분류	타입의 종류
기본 타입	정수 타입	byte, char, short, int, long
	실수 타입	float, double
	논리 타입	boolean

## 1. 정수 타입

정수 타입

- 메모리 사용 크기와 저장되는 값의 허용 범위 각기 다름

타입	메모리 사용 크기		저장되는 값의 허용 범위	
byte	1byte	8bit	$-2^7 \sim (2^7-1)$	$-128 \sim 127$
short	2byte	16bit	$-2^{15} \sim (2^{15}-1)$	$-32,768 \sim 32,767$
char	2byte	16bit	$0 \sim (2^{16}-1)$	$0 \sim 65535$ (유니코드)
int	4byte	32bit	$-2^{31} \sim (2^{31}-1)$	$-2,147,483,648 \sim 2,147,483,647$
long	8byte	64bit	$-2^{63} \sim (2^{63}-1)$	$-9,223,372,036,854,775,808 \sim 9,223,372,036,854,775,807$

메모리

[illegible]

# 1. 정수 타입

## ❖ 리터럴 (literal)

- 소스 코드에서 프로그래머에 의해 직접 입력된 값
- 다음 경우를 자바에서 정수로 인식

2진수: 0b 또는 0B로 시작하고 0과 1로 구성됩니다.

0b1011	$\rightarrow 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$	$\rightarrow 11$
0b10100	$\rightarrow 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$	$\rightarrow 20$

8진수: 0으로 시작하고 0~7 숫자로 구성됩니다.

013	$\rightarrow 1 \times 8^1 + 3 \times 8^0$	$\rightarrow 11$
0206	$\rightarrow 2 \times 8^2 + 0 \times 8^1 + 6 \times 8^0$	$\rightarrow 134$

10진수: 소수점이 없는 0~9 숫자로 구성됩니다.

12  
365

16진수: 0x 또는 0X로 시작하고 0~9 숫자와 A, B, C, D, E, F 또는 a, b, c, d, e, f로 구성됩니다.

0xB3	$\rightarrow 11 \times 16^1 + 3 \times 16^0$	$\rightarrow 179$
0x2A0F	$\rightarrow 2 \times 16^3 + 10 \times 16^2 + 0 \times 16^1 + 15 \times 16^0$	$\rightarrow 10767$

실습: IntegerLiteralExample.java, ByteExample.java,  
LongExample.java

# 1. 정수 타입

실습: IntegerLiteralExample.java, ByteExample.java, LongExample.java

```
1 package sec02.exam01;
2
3 public class IntegerLiteralExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         int var1 = 0b1011;           //2진수
8         int var2 = 0206;             //8진수
9         int var3 = 365;              //10진수
10        int var4 = 0xB3;             //16진수
11
12        System.out.println("var1: " + var1);
13        System.out.println("var2: " + var2);
14        System.out.println("var3: " + var3);
15        System.out.println("var4: " + var4);
```

# 1. 정수 타입

실습: IntegerLiteralExample.java, ByteExample.java, LongExample.java

```
package sec02.exam02;

public class ByteExample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        byte var1 = -128;
        byte var2 = -30;
        byte var3 = 0;
        byte var4 = 30;
        byte var5 = 127;
        //byte var6 = 128;
        //컴파일 에러 (Type mismatch: cannot covert from int to byte)

        System.out.println(var1);
        System.out.println(var2);
        System.out.println(var3);
        System.out.println(var4);
        System.out.println(var5);
    }
}
```

# 1. 정수 타입

실습: IntegerLiteralExample.java, ByteExample.java, LongExample.java

```
1 package sec02.exam03;
2
3 public class LongExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         long var1 = 10;
8         long var2 = 20L;
9         //long var3 = 10000000000000; //컴파일 에러
10        long var4 = 100000000000000L;
11
12        System.out.println(var1);
13        System.out.println(var2);
14        System.out.println(var4);
```



# 1. 정수 타입

## ❖ char 타입

- 하나의 문자를 저장할 수 있는 타입 'A' '한'
- 작은 따옴표로 감싼 문자 리터럴은 유니코드로 변환되어 저장 => char 타입은 정수 타입



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

- char는 정수 타입이므로 10 진수 또는 16진수 형태의 유니코드 저장



```
char c = 65;        //10진수
char c = 0x0041;    //16진수
```

실습: CharExample.java

# 1. 정수 타입

실습: CharExample.java

```
1 package sec02exam04;
2 public class CharExample {
3     public static void main(String[] args) {
4         // TODO Auto-generated method stub
5         char c1 = 'A';           //문자를 직접 저장
6         char c2 = 65;           //십진수로 저장
7         char c3 = '\u0041';     //16진수로 저장
8
9         char c4='가';           //문자를 직접 저장
10        char c5 = 44032;         //십진수로 저장
11        char c6 = '\uac00';     //16진수로 저장
12
13        System.out.println(c1);
14        System.out.println(c2);
15        System.out.println(c3);
16        System.out.println(c4);
17        System.out.println(c5);
18        System.out.println(c6);
```

# 1. 정수 타입

## ❖ 문자열

- 큰따옴표로 감싼 문자들을 문자열이라고 함.
- 문자열은 char 타입에 저장할 수 없음

```
char var1 = "A";  
char var2 = "홍길동";
```

- String 타입
  - 문자열을 String 타입 변수에 저장

```
String var1 = "A";  
String var2 = "홍길동";
```

실습: StringExample.java

# 1. 정수 타입

실습: StringExample.java

```
1 package sec02exam05;
2
3 public class StringExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         String name = "홍길동";
8         String job = "프로그래머";
9         System.out.println(name);
10        System.out.println(job);
11    }
```

# 1. 정수 타입

## ❖ 이스케이프 문자 (escape)

- 문자열 내부에 \는 이스케이프 문자를 뜻함
- 이스케이프 문자를 사용하면 특정 문자를 포함시키거나, 문자열의 출력을 제어할 수 있음
- 예) 문자열 내부에 " 문자 포함

```
String str = "나는 \"자바\"를 좋아합니다.";
System.out.println(str);
```

→ 나는 "자바"를 좋아합니다.

- 예) 문자열 출력 제어

```
String str = "번호\t이름\t나이";
System.out.println(str);
```

→ 번호      이름      나이  
    tab 공간    tab 공간

```
String str = "홍길동\n감자바";
System.out.println(str);
```

→ 홍길동  
    감자바

# 1. 정수 타입

실습: EscapeExample.java

```
1 package sec02.exam06;
2
3 public class EscapeExample {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         System.out.println("번호\t이름\t직업 ");
8         System.out.print("행 단위 출력\n");
9         System.out.print("행 단위 출력\n");
10        System.out.println("우리는 \"개발자\" 입니다.");
11        System.out.print("봄\\여름\\가을\\겨울");
```

## 1. 정수 타입

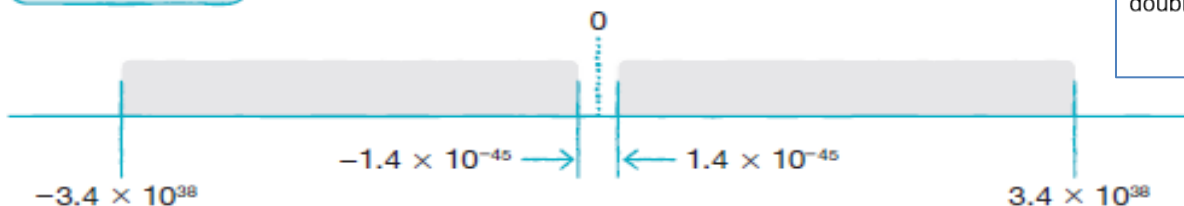
이스케이프 문자	출력 용도
\t	탭만큼 띄움
\n	줄 바꿈(라인 피드)
\r	캐리지리턴
\"	" 출력
\'	' 출력
\\	\ 출력
\u16진수	16진수 유니코드에 해당하는 문자 출력

## 2. 실수 타입

### ❖ 실수 타입

타입	메모리 사용 크기		저장되는 값의 허용 범위(양수 기준)	정밀도(소수점 이하 자리)
float	4byte	32bit	$(1.4 \times 10^{-45}) \sim (3.4 \times 10^{38})$	7자리
double	8byte	64bit	$(4.9 \times 10^{-324}) \sim (1.8 \times 10^{308})$	15자리

float 타입



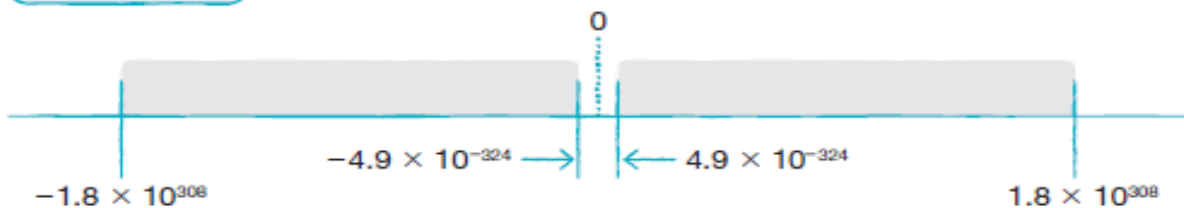
float: 소수점 이하자리 약 7 (6~9)

**0.12345679**

double: 소수점 이하자리 약 15 (15~18)

**0.1234567890123457**

double 타입





## 2. 실수 타입

### ❖ 실수 리터럴

- 소스 코드에서 소수점 있는 리터럴은 10진수 실수로 인식

```
0.25, -3.14
```

- 알파벳 e 또는 E 포함된 숫자 리터럴은 지수 및 가수로 표현된, 소수점 있는 10진수 실수로 인식

```
5e2    →  $5.0 \times 10^2 = 500.0$ 
```

```
0.12E-2 →  $0.12 \times 10^{-2} = 0.0012$ 
```

## 2. 실수 타입

- **double 타입** 변수에 저장: 자바는 실수 리터럴을 기본적으로 double 타입으로 해석

```
float var = 3.14; ← 컴파일 에러(Type mismatch: cannot convert from double to float)
```

```
double var = 3.14;  
double var = 314e-2;
```

- **float 타입**으로 저장하려는 경우: 리터럴 뒤 f 혹은 F 붙여 float 타입 표시

```
float var = 3.14f;  
float var = 3E6F;
```

- double 타입이 float 타입보다 2배 가량 정밀도 높아 정확한 데이터 저장 가능

- float: 소수점 이하자리 약 7 ( 6~9 )

**0.12345679**

- double: 소수점 이하자리 약 15 (15~18)

**0.1234567890123457**

```

1 package sec02.exam07;
2
3 public class FloatDoubleExample {
4     public static void main(String[] args) {
5         //실수값 저장
6         //float var1 = 3.14; //컴파일 에러 (Type mismatch)
7         float var2 = 3.14f;
8         double var3 = 3.14;
9
10        //정밀도 테스트
11        float var4 = 0.1234567890123456789f;
12        double var5 = 0.1234567890123456789;
13
14        System.out.println("var2: " + var2);
15        System.out.println("var3: " + var3);
16        System.out.println("var4: " + var4);
17        System.out.println("var5: " + var5);
18
19        //e 사용하기
20        double var6 = 3e6;
21        float var7 = 3e6F;
22        double var8 = 2e-3;
23        System.out.println("var6: " + var6);
24        System.out.println("var7: " + var7);
25        System.out.println("var8: " + var8);
26    }
27 }

```

### 3. 논리 타입

#### ❖ 논리 타입

- 참과 거짓에 해당하는 true와 false 리터럴을 저장하는 타입

```
boolean stop = true;  
boolean state = false;
```

- 두 가지 상태값에 따라 제어문의 실행 흐름을 변경하는데 사용

#### 실습: BooleanExample.java

```
package sec02.exam08;  
  
public class BooleanExample {  
    public static void main(String[] args) {  
        boolean stop = true;  
        if(stop) {  
            System.out.println("중지합니다.");  
        } else {  
            System.out.println("시작합니다.");  
        }  
    }  
}
```

## 4. 키워드로 끝내는 핵심 포인트

- **정수 타입**: 정수를 저장할 수 있는 타입으로 byte, short, int, long 타입을 말함.
- **char 타입**: 작은따옴표(')로 감싼 하나의 문자 리터럴을 저장할 수 있는 타입.
- **String 타입**: 큰따옴표(")로 감싼 문자열을 저장할 수 있는 타입
- **실수 타입** : 실수를 저장할 수 있는 타입으로 float, double 타입을 말함.
- **boolean 타입** : 참과 거짓을 의미하는 true와 false를 저장할 수 있는 타입

Chapter

# 02

## 변수와 타입

### 02-3. 타입 변환

- 0. 시작하기 전에
- 1. 타입 변환
- 2. 정수 연산에서의 자동 타입 변환
- 3. 실수 연산에서의 자동 타입 변환
- 4. + 연산에서의 문자열 자동 타입 변환
- 5. 문자열을 기본 타입으로 강제 타입 변환
- 6. 키워드로 끝내는 핵심 포인트

## 0. 시작하기 전에

[핵심 키워드] : 자동 타입 변환, 강제 타입 변환, 문자열 결합 연산,  
Integer.parseInt( ), Double.parseDouble( )

### [핵심 포인트]

- 타입 변환이란 데이터 타입을 다른 데이터 타입으로 변환하는 것을 말한다.
  - byte 타입 -> int 타입,
  - int 타입 -> byte 타입,
  - double 타입 -> int 타입
  - String 타입 -> int 타입

### ❖ 타입 변환

- 변수 값을 다른 타입의 변수에 저장할 때 타입 변환이 발생할 수 있다.

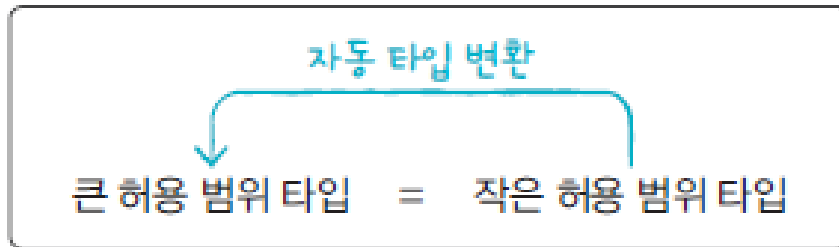
```
byte a = 10;    //byte 타입 변수 a에 10을 저장
int b = a;      //byte 타입 변수 a에 저장된 10을 int 타입 변수 b에 복사해서 저장
```



# 1. 타입 변환

## ❖ 자동 타입 변환 (promotion)

- 값의 허용 범위가 작은 타입이 큰 타입으로 저장될 경우



- 기본 타입의 허용 범위 순

```
byte < short < int < long < float < double
```

```
byte byteValue = 10;  
int intValue = byteValue;    //자동 타입 변환됨
```

```
long longValue = 5000000000L;  
float floatValue = longValue;    //5.0E9f로 저장됨  
double doubleValue = longValue;  //5.0E9로 저장됨
```

# 1. 타입 변환

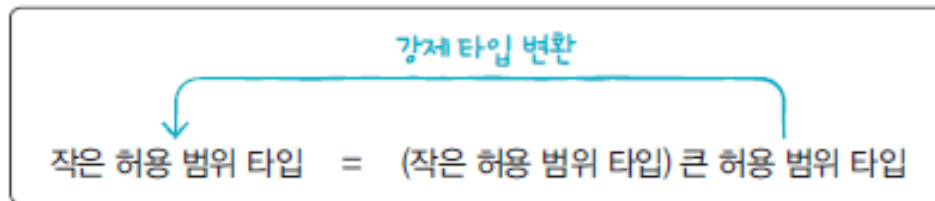
- char 타입의 경우 int 타입으로 자동변환되면 유니코드 값이 int 타입에 저장

```
char charValue = 'A';  
int intValue = charValue;    //65가 저장됨
```

```
byte byteValue = 65;  
char charValue = byteValue; ← 컴파일 에러
```

## ❖ 강제 타입 변환 (casting)

- 큰 허용 범위 타입을 작은 허용 범위 타입으로 강제로 나누어 한 조각만 저장



- 캐스팅 연산자 괄호 () 사용: 괄호 안이 나누는 단위

```
int intValue = 10;  
byte byteValue = (byte) intValue;    //강제 타입 변환
```

# 1. 타입 변환

- ex) int 타입을 char 타입으로 강제 변환

- 문자 출력 위함

```
int intValue = 65;  
char charValue = (char) intValue;  
System.out.println(charValue);    //"A"가 출력
```

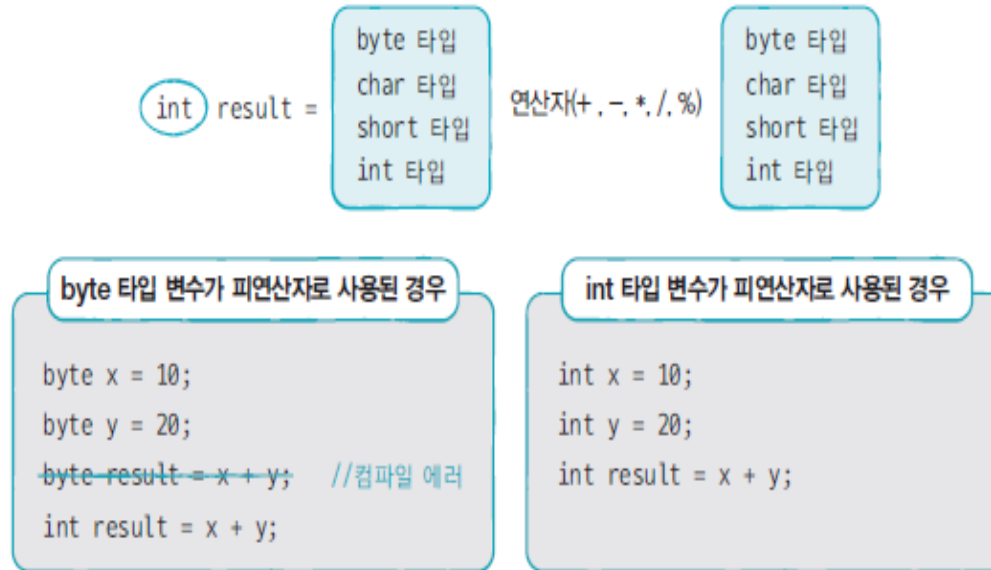
- ex) 실수 타입을 정수 타입으로 강제 변환

- 소수점 이하 부분 버려지고 정수 부분만 저장

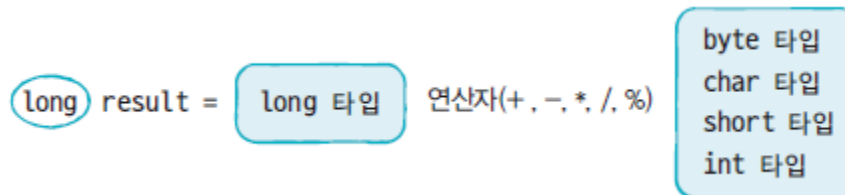
```
double doubleValue = 3.14;  
int intValue = (int) doubleValue;    //intValue는 정수 부분인 3만 저장
```

## 2. 정수 연산에서의 자동 타입 변환

- ❖ 정수 타입 변수가 산술 연산식에서 피연산자로 사용되는 경우
  - byte, char, short 타입 변수는 int 타입으로 자동 변환



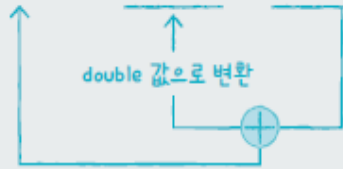
- 특별한 경우 아니라면 정수 연산에 사용하는 변수는 int 타입으로 선언하는 것이 효과적
- 피 연산자 중 하나가 long 타입이면 다른 피연산자는 long 타입으로 자동 변환



### 3. 실수 타입의 자동 타입 변환

- ❖ 피연산자 중 하나라도 double 타입일 경우 다른 피연산자도 double 타입으로 자동 변환

```
int intValue = 10;  
double doubleValue = 5.5;  
double result = intValue + doubleValue;    //result에 15.5가 저장됨
```



```
int intValue = 10;  
double doubleValue = 5.5;  
int result = intValue + (int) doubleValue;    //result에 15가 저장됨
```

2.3;

3f;

### 3. 실수 연산에서의 자동 타입 변환

#### ❖ 정수 연산의 결과를 실수로 저장할 때 주의할 점

- 정수 연산의 결과는 정수

```
int x = 1;
int y = 2;
double result = x / y;
System.out.println(result);
```

- 실수 결과를 얻으려면 실수 연산으로의 변환 필요

방법 1	<pre>int x = 1; int y = 2; double result = (double) x / y; System.out.println(result);</pre>
방법 2	<pre>int x = 1; int y = 2; double result = x / (double) y; System.out.println(result);</pre>
방법 3	<pre>int x = 1; int y = 2; double result = (double) x / (double) y; System.out.println(result);</pre>

## 4. + 연산에서의 문자열 자동 타입 변환

### ❖ + 연산

- 피연산자가 모두 숫자일 경우 덧셈 연산
- 피연산자 중 하나가 문자열일 경우 나머지 피연산자도 문자열로 자동 변환되고 문자열 결합 연산

```
int value = 3 + 7;    → int value = 10;  
String str = "3" + 7; → String str = "3" + "7"; → String str = "37";  
String str = 3 + "7"; → String str = "3" + "7"; → String str = "37";
```

- + 연산은 앞에서부터 순차적으로 수행
  - 먼저 수행된 연산이 곱한 연산이 경우 이후 모든 연산이 곱한 연산이 됨

```
int value = 1 + 2 + 3;    → int value = 3 + 3;    → int value = 6;  
String str = 1 + 2 + "3"; → String str = 3 + "3"; → String str = "33";  
String str = 1 + "2" + 3; → String str = "12" + 3; → String str = "123";  
String str = "1" + 2 + 3; → String str = "12" + 3; → String str = "123";
```

## 5. 문자열을 기본 타입으로 강제 타입 변환

### ❖ 문자열을 기본 타입으로 강제 변환

변환 타입	사용 예
String → byte	<pre>String str = "10"; byte value = Byte.parseByte(str);</pre>
String → short	<pre>String str = "200"; short value = Short.parseShort(str);</pre>
String → int	<pre>String str = "300000"; int value = Integer.parseInt(str);</pre>
String → long	<pre>String str = "40000000000"; long value = Long.parseLong(str);</pre>
String → float	<pre>String str = "12.345"; float value = Float.parseFloat(str);</pre>
String → double	<pre>String str = "12.345"; double value = Double.parseDouble(str);</pre>
String → boolean	<pre>String str = "true"; boolean value = Boolean.parseBoolean(str);</pre>



## 5. 문자열을 기본 타입으로 강제 타입 변환

- 문자열이 숫자 외 요소를 포함할 경우 숫자 타입 변환 시도할 경우 숫자 형식 예외 발생

```
String str = "1a";  
int value = Integer.parseInt(str);    //NumberFormatException 발생
```

- **String.valueOf()** 메소드 사용하여 기본 타입을 문자열로 변환
  - String str = String.valueOf(3);

## 6. 키워드로 끝내는 핵심 포인트

- **자동 타입 변환**: 자동으로 타입이 변환되는 것. 값의 허용 범위가 작은 타입이 허용 범위가 큰 타입으로 저장될 때 발생함.
- **강제 타입 변환**: 강제로 타입을 변환하는 것. 값의 허용 범위가 큰 타입을 허용 범위가 작은 타입으로 쪼개어서 저장하는 것을 말함.
- **문자열 결합 연산**: 문자열과 + 연산을 하면 다른 피연산자도 문자열로 변환되어 문자열 결합이 발생함.
- **Integer.parseInt()** : 문자열을 정수 int 타입으로 변환
- **Double.parseDouble()** : 문자열을 실수 double 타입으로 변환

Chapter

# 02

## 변수와 타입

### 02-4. 변수와 시스템 입출력

- 0. 시작하기 전에
- 1. 모니터로 변수값 출력하기
- 2. 키보드에서 입력된 내용을 변수에 저장하기
- 3. 키워드로 끝내는 핵심 포인트

# 0. 시작하기 전에

[핵심 키워드] : `System.out.println()`, `System.out.print()`, `System.out.printf()`,  
`System.in.read()`, `Scanner`

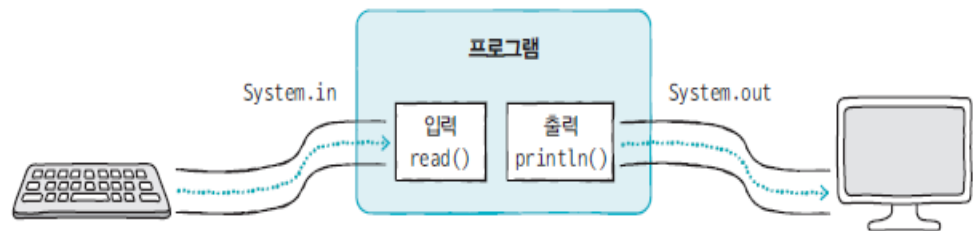
[핵심 포인트]

- 프로그래밍 언어에는 시스템 표준 출력 장치와 표준 입력 장치 이해  
표준 출력 장치 -> 모니터  
표준 입력 장치 -> 키보드
- 변수에 저장된 값을 모니터로 출력하는 방법과  
키보드로부터 데이터를 읽고 변수에 저장하는 방법을 학습

## ❖ System.out

- 시스템의 표준 출력 장치로 출력

```
System.out.println("출력 내용");
```



## ❖ System.in

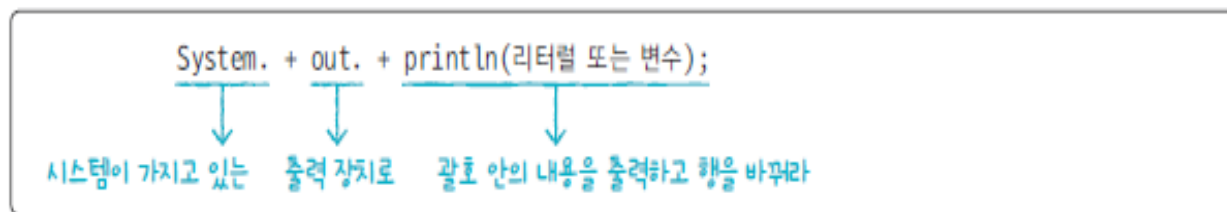
- 시스템의 표준 입력 장치에서 읽음

```
System.in.read();
```

# 1. 모니터로 변수값 출력하기

## ❖ println() 메소드

- 괄호 안에 리터럴 넣으면 그대로 출력 / 변수 넣으면 저장된 값 출력



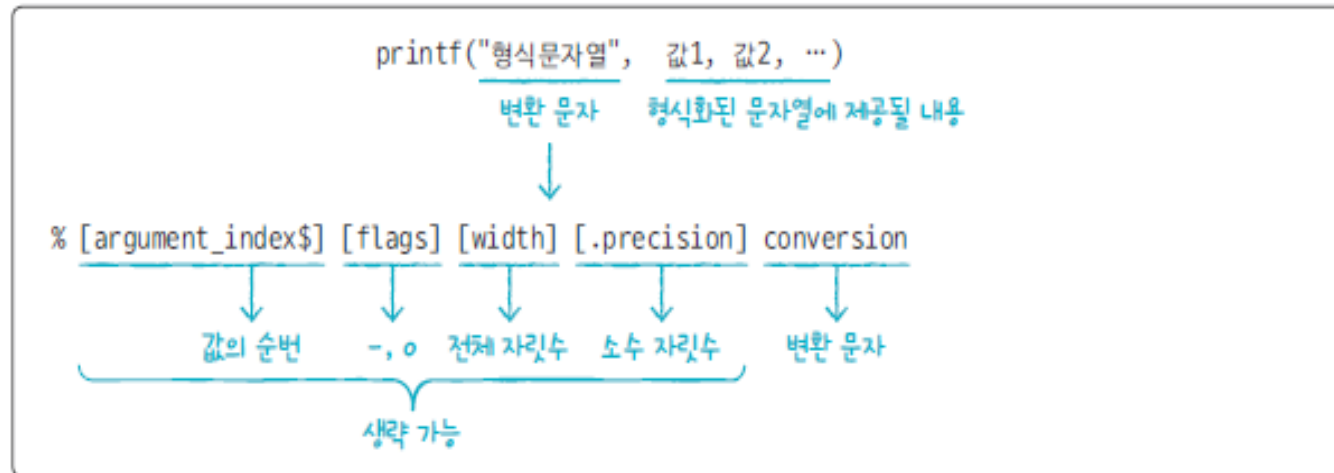
## ❖ 다양한 출력 메소드

메소드	의미
<code>println(내용);</code>	괄호 안의 내용을 출력하고 행을 바꿔라
<code>print(내용);</code>	괄호 안의 내용을 출력만 해라
<code>printf("형식문자열", 값1, 값2, ...);</code>	괄호 안의 첫 번째 문자열 형식대로 내용을 출력해라

# 1. 모니터로 변수값 출력하기

## ❖ printf() 메소드

- 개발자가 원하는 형식화된 문자열 (format string) 출력 (전체 출력 자리수 및 소수 자리수 제한)



- 형식 문자열에서 %와 conversion 외에는 모두 생략 가능
- **conversion**에는 제공되는 값의 타입에 따라 d(정수), f(실수), s(문자열) 입력

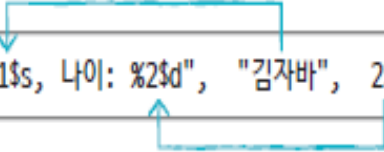
The diagram shows two lines of code with arrows indicating the values being passed to the `printf` function's arguments:

```
System.out.printf("이름: %s", "감자바"); → 이름: 감자바  
System.out.printf("나이: %d", 25 );    → 나이: 25
```

# 1. 모니터로 변수값 출력하기

- 형식 문자열에 포함될 값 2개 이상인 경우 값의 **순번(argument\_index\$)** 표시해야

`System.out.printf("이름: %1$s, 나이: %2$d", "김자바", 25);` → 이름: 김자바, 나이: 25



- 다양한 형식 문자열

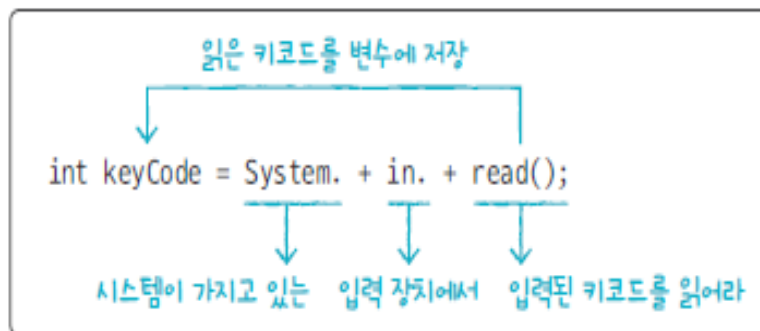
형식화된 문자열		설명	출력 형태
정수	%d	정수	123
	%6d	6자리 정수, 왼쪽 빈 자리 공백	__123
	%-6d	6자리 정수, 오른쪽 빈 자리 공백	123__
	%06d	6자리 정수, 왼쪽 빈 자리 0 채움	000123
실수	%10.2f	소수점 이상 7자리, 소수점 이하 2자리, 왼쪽 빈 자리 공백	__123.45
	%-10.2f	소수점 이상 7자리, 소수점 이하 2자리, 오른쪽 빈 자리 공백	123.45__
	%010.2f	소수점 이상 7자리, 소수점 이하 2자리, 왼쪽 빈 자리 0 채움	0000123.45
문자열	%s	문자열	abc
	%6s	6자리 문자열, 왼쪽 빈 자리 공백	__abc
	%-6s	6자리 문자열, 오른쪽 빈 자리 공백	abc__
특수 문자	\t	탭(tab)	
	\n	줄 바꿈	
	%%	%	%



## 2. 키보드에서 입력된 내용을 변수에 저장하기

### ❖ 키코드

- 키보드에서 키를 입력할 때 프로그램에서 숫자로 된 키코드를 읽음
- System.in의 `read()` 사용
- 얻은 키코드는 대입 연산자 사용하여 int 변수에 저장



숫자	알파벳				기능키	방향키
0 = 48	A = 65	N = 78	a = 97	n = 110	BACK SPACE = 8	← = 37
1 = 49	B = 66	O = 79	b = 98	o = 111	TAB = 9	↑ = 38
2 = 50	C = 67	P = 80	c = 99	p = 112	ENTER = [CR=13, LF=10]	→ = 39
3 = 51	D = 68	Q = 81	d = 100	q = 113	SHIFT = 16	↓ = 40
4 = 52	E = 69	R = 82	e = 101	r = 114	CONTROL = 17	
5 = 53	F = 70	S = 83	f = 102	s = 115	ALT = 18	
6 = 54	G = 71	T = 84	g = 103	t = 116	ESC = 27	
7 = 55	H = 72	U = 85	h = 104	u = 117	SPACE = 32	
8 = 56	I = 73	V = 86	i = 105	v = 118	PAGEUP = 33	
9 = 57	J = 74	W = 87	j = 106	w = 119	PAGEDN = 34	
	K = 75	X = 88	k = 107	x = 120		
	L = 76	Y = 89	l = 108	y = 121		
	M = 77	Z = 90	m = 109	z = 122		

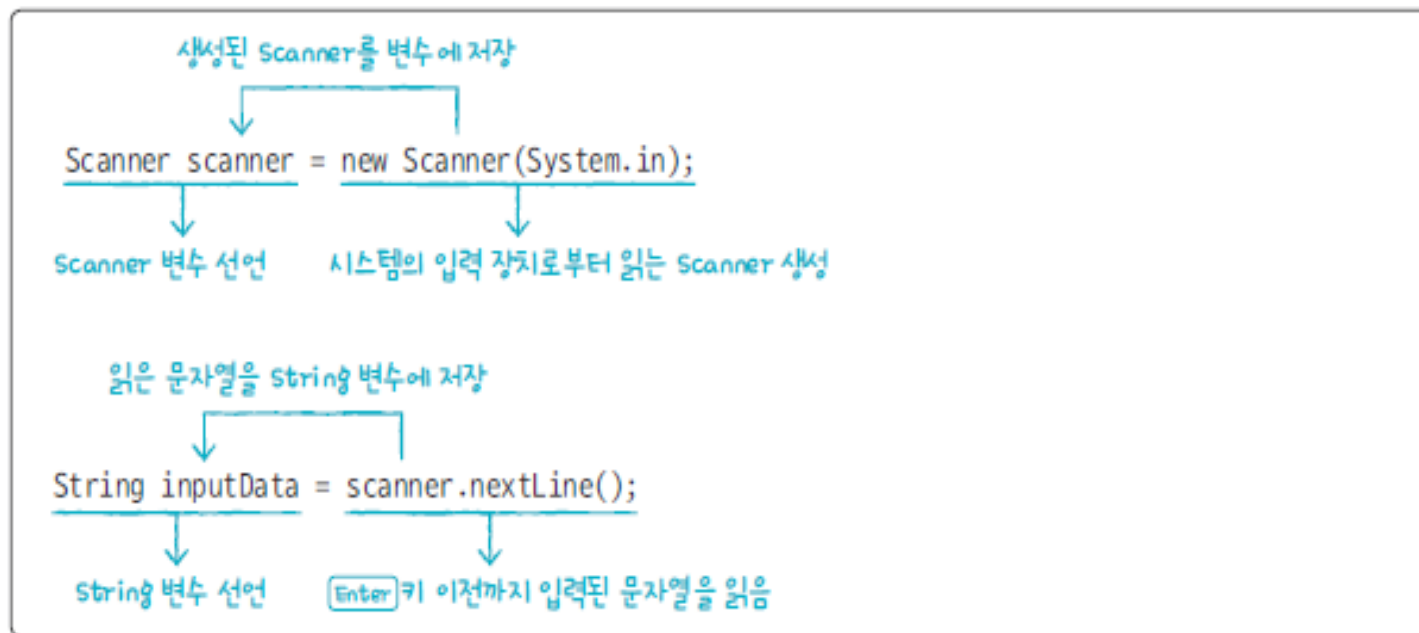
## 2. 키보드에서 입력된 내용을 변수에 저장하기

### ❖ System.in.read()의 단점

- 2개 이상 키가 조합된 한글 읽을 수 없음
- 키보드로 입력된 내용을 통문자열로 읽을 수 없음

### ❖ Scanner 로 해결

- 자바가 제공하는 Scanner 클래스를 이용하면 입력된 통문자열을 읽을 수 있음



## 2. 키보드에서 입력된 내용을 변수에 저장하기

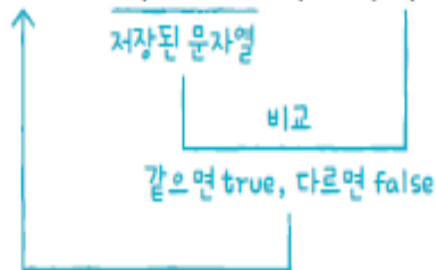
### ❖ 기본 타입의 값 비교와 문자열 비교

#### ■ 기본 타입의 값 비교는 `==` 를 사용

- `int x = 5;`
- `boolean result = (x == 5); //true`

#### ■ 문자열의 비교는 `equals()` 메소드 사용

```
boolean result = inputData.equals("비교문자열");
```



- `String str1 = "java";`
- `boolean result1 = str.equals("java"); //true`
- `boolean result2 = str.equals("Java"); //false`

### 3. 키워드로 끝내는 핵심 포인트

- `System.out.println()`: 괄호에 주어진 매개값을 모니터로 출력하고 개행을 한다.
- `System.out.print()`: 괄호에 주어진 매개값을 모니터로 출력만 하고 개행을 하지 않는다.
- `System.out.printf()`: 괄호에 주어진 형식대로 출력한다.
- `System.in.read()` : 키보드에서 입력된 키코드를 읽는다.
- `Scanner`: 키보드로부터 입력된 내용을 통 문자열로 쉽게 읽기 위해서 `Scanner`를 사용한다.



Thank You!