

# 자바 프로그래밍

## 제2장 자바 프로그래밍 기초

# 학습 내용

## 학습목차

- 01 "Hello World!" 예제 분석
- 02 기초 개념들
  - LAB 순차적인 프로그램 작성하기
- 03 변수와 자료형
  - LAB 지구에서 가장 가까운 별까지의 거리 계산하기
- 04 수식과 연산자
- 05 형변환
- 06 우선순위와 결합 규칙
  - LAB 2차 방정식의 근을 계산하여 보자
- 07 문자열
- 08 입력과 출력
  - LAB 원의 면적 계산하기
  - LAB 직사각형의 둘레와 면적 계산하기

이제 본격적으로  
프로그래밍이 시작되나요?

네, 이번 장에서는 변수나  
함수와 같은 프로그래밍 기초  
개념들을 학습합니다. 이것들은  
프로그래밍의 토대가 됩니다.



# 예제 소스 코드

*Hello.java*

```
01 public class Hello
02 {
03     public static void main(String[] args)
04         System.out.println("Hello World!");
05     }
06 }
```

클래스를 정의하는  
문장이다.

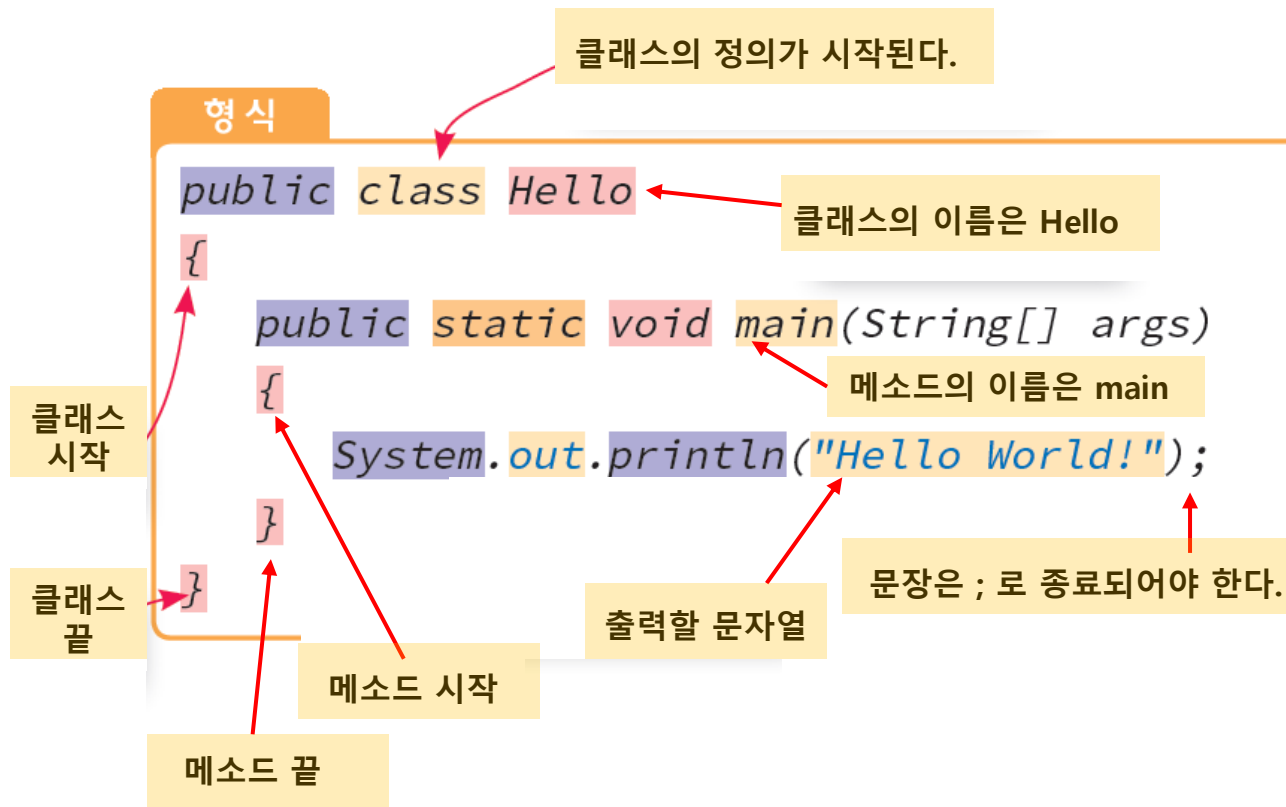
메소드를 정의하는  
문장이다.

실행결과



Hello World!

# 예제 소스 코드 분석



생소한 개념과 용어가  
너무 많죠? 이제부터 자세  
히 설명될 것입니다.



# 클래스

- **클래스(class)**는 자바와 같은 객체 지향 언어의 기본적인 빌딩 블록이다.
- 필요한 클래스를 하나씩 만들어 감으로써 전체 프로그램이 완성된다.



클래스는 자바 프로그램을  
이루는 기본적인  
빌딩블록 입니다.



# 클래스의 정의

이 클래스는 누구든지  
사용 가능

클래스를 선언하는 키워드

클래스 이름

클래스 시작

전체적인 구조



형식

```
public class Hello {
```

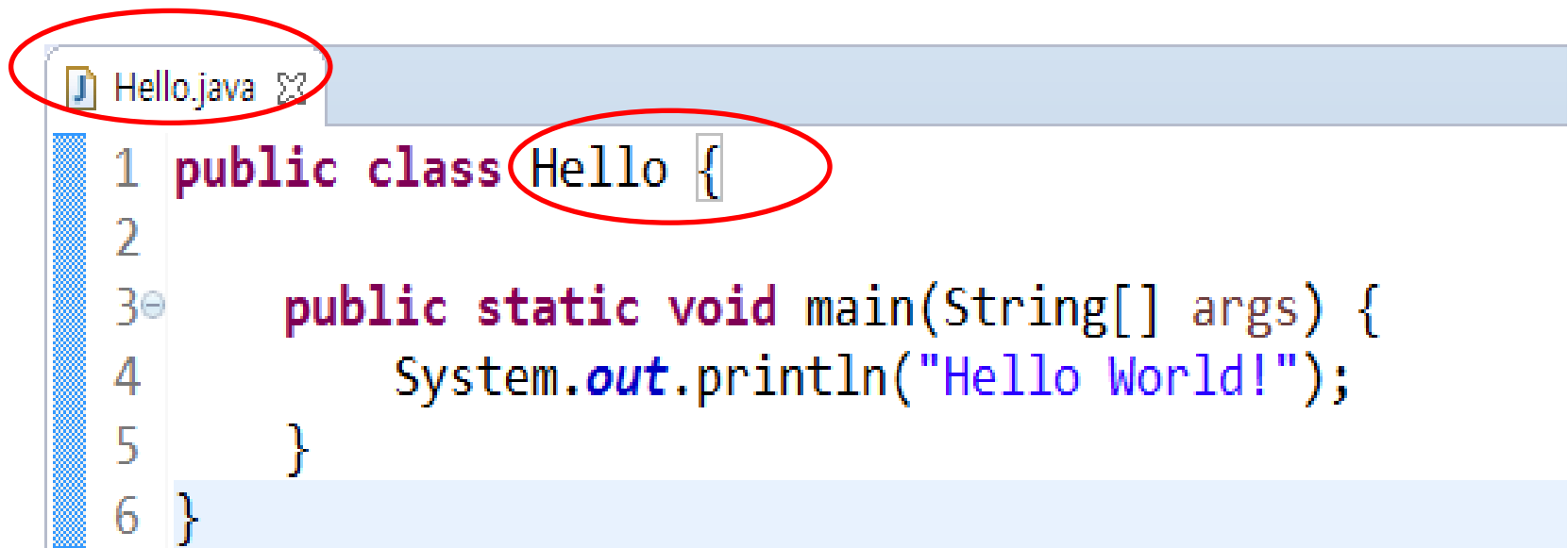
여기에 우리가 원하는  
문장을 추가한다.

```
}
```

클래스 종료

# 소스 파일과 클래스 이름

- public 클래스가 있는 경우, 소스 파일의 이름은 public 클래스의 이름과 일치하여야 한다.

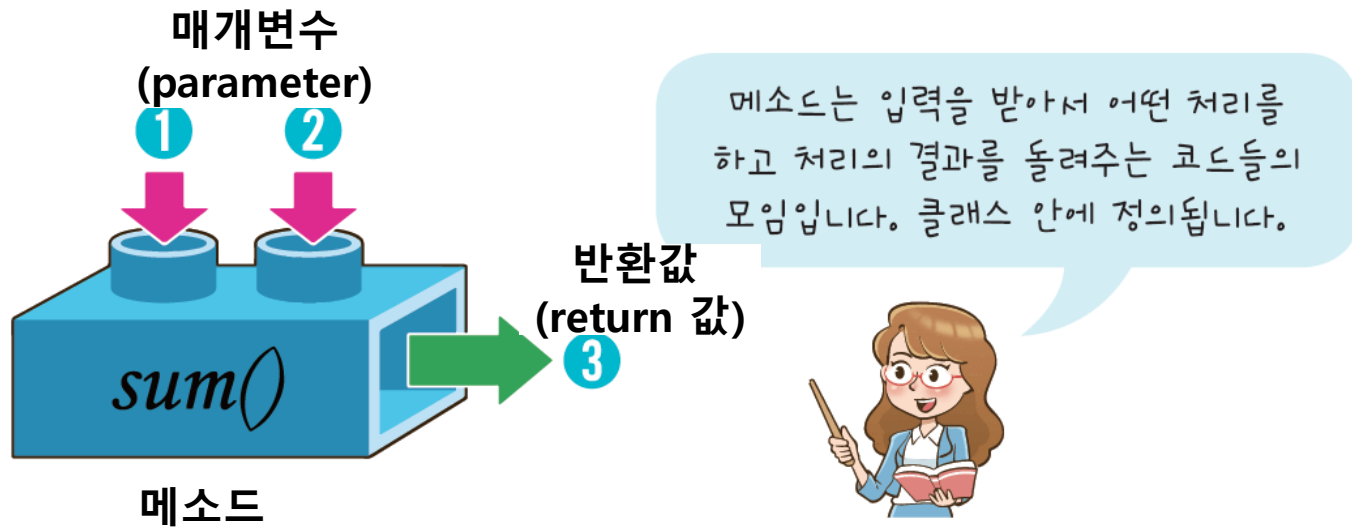


```
1 public class Hello {  
2  
3     public static void main(String[] args) {  
4         System.out.println("Hello World!");  
5     }  
6 }
```

- 하나의 소스 파일 안에 public 클래스가 2개 이상 있으면 컴파일 오류가 발생한다.

# 메소드

- 메소드(method)는 특정한 작업을 수행하는 코드의 묶음
- 메소드는 외부로부터 입력을 받아서 특정한 작업을 수행하고 작업의 결과를 반환하는 블랙 박스





# 메소드의 정의

- 반드시 클래스 안에서 정의됨
  - 아래 예에서는 Hello 클래스 안에 main 메소드를 정의함

```
1 public class Hello {  
2  
3     public static void main(String[] args) {  
4         System.out.println("Hello World!");  
5     }  
6 }
```

누구나 호출  
가능

정적 메소드이다.

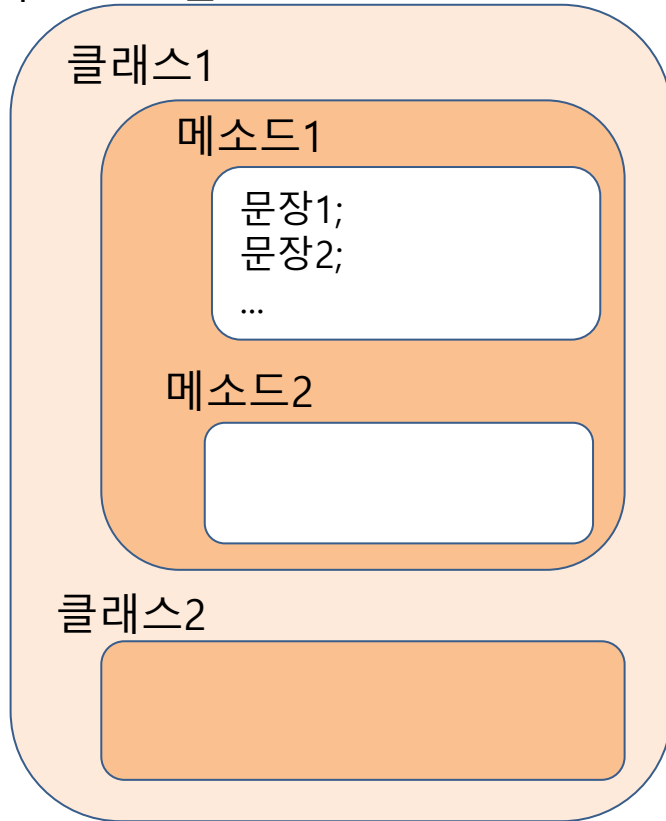
결과값을  
반환하지 않음

메소드 이름

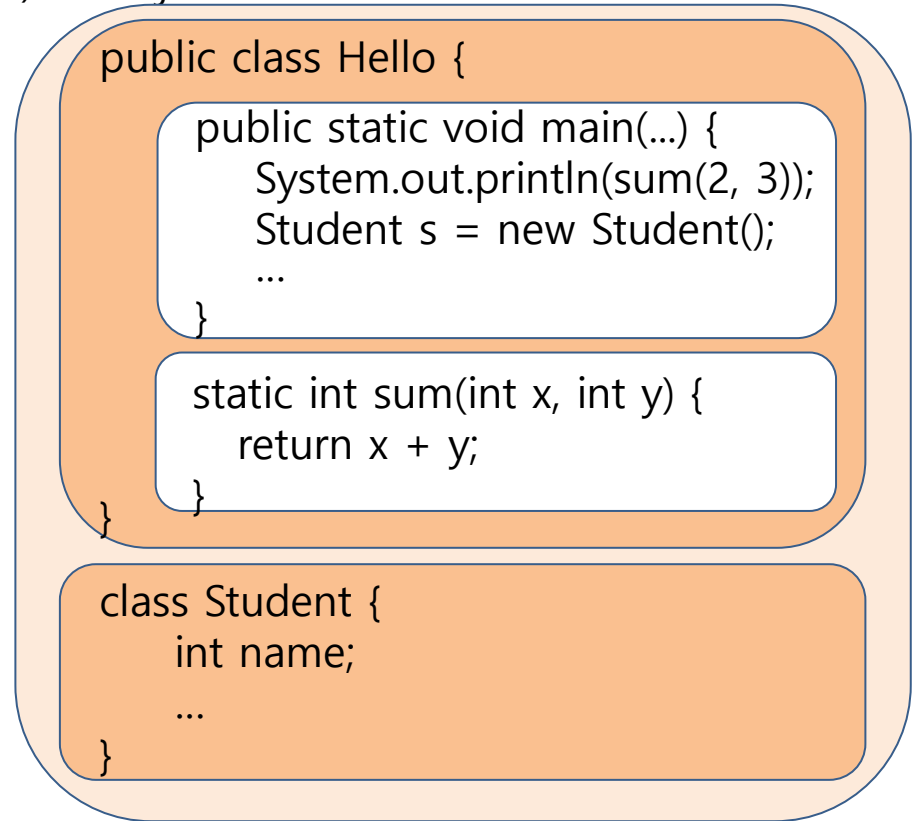
외부에서 주어지는 데이터를  
받는 매개 변수

# 자바 프로그램의 일반적인 구조

자바 프로그램



예) Hello.java



- 하나의 프로그램은 여러 개의 클래스를 포함할 수 있다.
- 하나의 클래스는 여러 개의 메소드를 포함할 수 있다.
- 하나의 메소드는 여러 개의 문장을 포함할 수 있다.

# main() 메소드

- main() 메소드에서 자바 프로그램의 실행이 시작된다.
  - main()의 첫번째 문장부터 시작하여 실행되다가 main()의 마지막 문장을 실행한 후 종료
- 모든 클래스가 main() 메소드를 가지고 있는 것은 아님
- 프로그램에 main() 메소드를 포함한 클래스가 있어야 함



JVM

나는 main() 메소드를  
제일 먼저 실행합니다.

# 문장

- **문장(statement)**은 사용자가 컴퓨터에게 작업을 지시하는 단위

*Hello.java*

```
01 public class Hello {  
02  
03     public static void main(String[] args) {  
04         System.out.println("Hello World!");  
05     }  
06 }
```

이것이 작업의 내용을  
기술하는 문장이다.

# 주석

- 주석(**comment**)은 소스 코드가 하는 일을 설명하는 글로서, 프로그램 실행 결과에 영향을 끼치지 않음
- 세가지 주석 표시 방법

**`/* TEXT */`**

주석의 시작과 끝을 `/*`와 `*/`로 표시

**`// TEXT`**

`//`에서 줄의 끝까지가 주석이다. 한 줄짜리 주석만 가능

**`/** DOCUMENTATION */`**

`/**`에서 `*/` 까지가 주석이다.

javadoc 프로그램이 이 부분의 주석을 추출하여 소스를 설명하는 HTML 문서를 생성

# 주석을 붙인 소스 코드

## Hello.java

```
01 /**
02  * 첫 번째 예제 프로그램
03  * @version 1.0 2015-08-01
04  * @author 홍길동
05  */
06 public class Hello {
07
08     public static void main(String[] args) {
09         System.out.println("Hello World!"); // "Hello World!"를 출력한다.
10     }
11 }
```

주석을 붙인다.

# 변수와 자료형

- 변수(variable)는 데이터를 담아 두는 상자



- 자료형(data type)은 변수에 저장되는 데이터의 타입

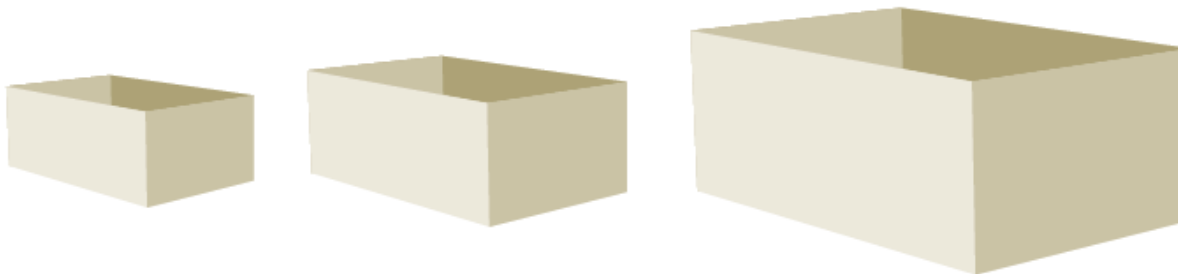
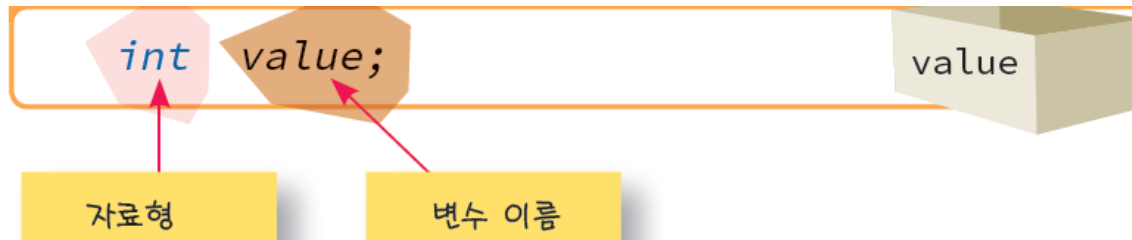


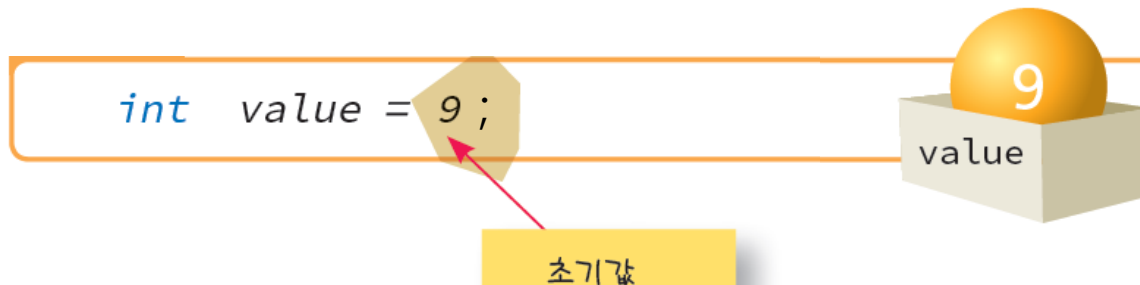
그림 2-7 • 자료형에는 여러 가지 종류가 있다.

# 변수 선언

- 변수는 사용하기 전에 반드시 미리 선언해야 한다.
- 변수 선언 형식



- 변수 선언과 동시에 초기값을 지정할 수 있다.





# 자료형의 종류



정수형: byte, short, int, long

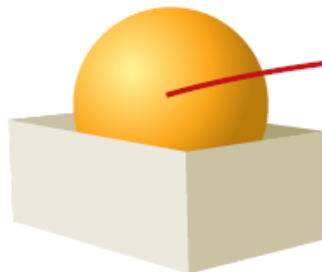
실수형: float, double

논리형: boolean

문자형: char

## 기초형 (primitive type)

기초형 변수에는 실제 값이 저장됨

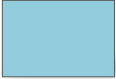
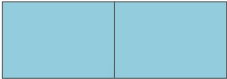
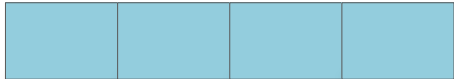
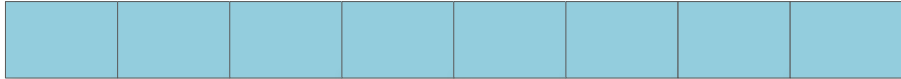
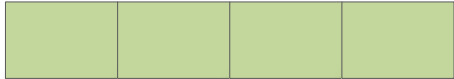
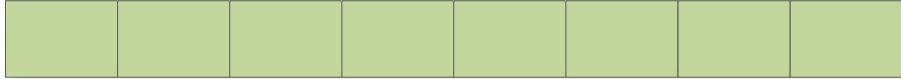




클래스, 인터페이스, 배열

## 참조형 (reference type)

참조형 변수에는 실제 객체를 가리키는 주소가 저장됨

# 기초 자료형

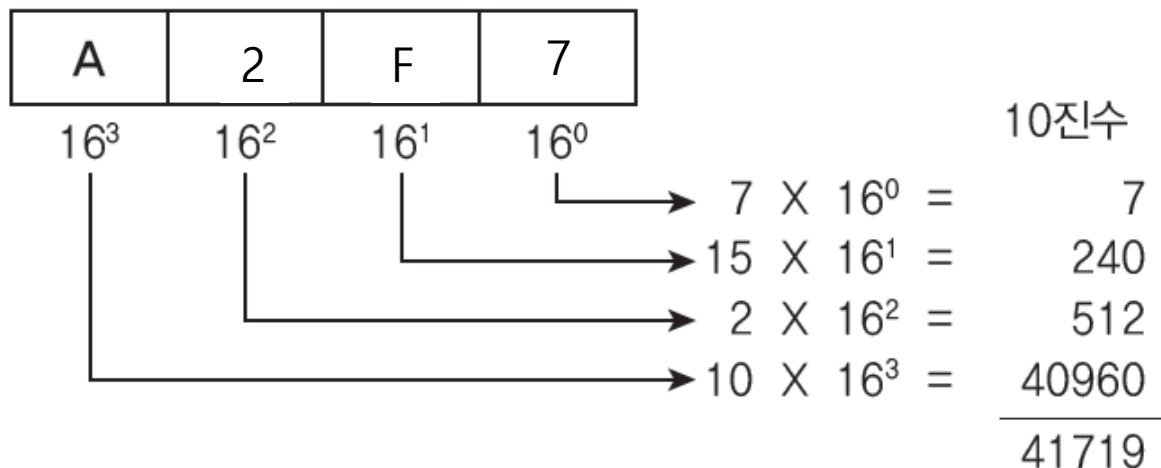
byte		-128 ~ 127
short		-32768 ~ 32767
int		약 -21억 ~ 21억
long		
float		
double		
<b>boolean</b>		true, false
char		유니코드

# 정수형

- 네 가지 부호 있는 정수형
  - byte : 1바이트 크기, -128 ~ 127
  - short : 2바이트 크기, -32768 ~ 32767
  - int : 4바이트 크기, -2147483648 ~ 2147483647
  - long : 8바이트 크기
- 이 중 int형이 가장 많이 사용됨
- 정수형 리터럴(literal)은 int형이 기본. 예를 들어
  - 76 : int형    ← 기본
  - 76L : long형    ← L을 붙임
- byte, short는 메모리가 부족한 상황에서만 사용
- 참고: long 범위 밖의 큰 정수는 BigInteger 클래스 사용

# 정수형 리터럴

- 10진수(decimal): 14, 16, 789
- 2진수(binary): **0b**1100
- 8진수(octal): **016, 071, 0234**
- 16진수(hexadecimal): **0xe, 0xA2F7**



# 예: 빛이 1년 동안 가는 거리

*Light.java*

```
01 public class Light {
02     public static void main(String args[]) {
03         long lightspeed;
04         long distance;
05
06         lightspeed = 300000;
07         distance = lightspeed * 365L * 24 * 60 * 60;
08
09         System.out.println("빛이 1년 동안 가는 거리 : " + distance +
10                             " km.");
11     }
12 }
```

빛이 1년 동안 가는 거리 : 946080000000000 km.

# 실수형

- 두가지 부동소수점형(floating point)
  - float : 32비트 크기, 약  $\pm 3.40282347 \times 10^{+38}$  (유효숫자 6~7개)
  - double : 64비트 크기, 약  $\pm 1.7976931 \times 10^{+308}$  (유효숫자 약15개)
- 부동소수점형 리터럴
  - double형이 기본
    - float t1 = 12.3; // 12.3은 double형이므로 오류!
    - float t2 = 12.3F; // OK
    - double n = 123\_456\_789.0; // JDK 7부터 밑줄기호 사용가능
  - 지수 표기법
    - 1.4691E+2 // 146.91
    - 8.1E-4 // 0.00081

# 예: 반지름이 5.0인 원의 면적을 계산하는 프로그램 작성

## *AreaTest.java*

```
01 public class AreaTest {  
02     public static void main(String args[]) {  
03         double radius, area;  
04  
05         radius = 5.0;  
06         area = 3.141592 * radius * radius;  
07         System.out.println("원의 면적은 " + area);  
08     }  
09 }
```

원의 면적은 78.5398

# 문자형

- 문자형 char
  - 하나의 문자를 저장
  - 자바는 유니코드(Unicode)를 지원하기 위해 문자 하나를 16비트로 표현
  - 유니코드 규격 중에서 UTF-16 인코딩 방식 사용
- 한글 표현
  - `char ch1 = '가';`
  - `char ch2 = '\uac00';`     `// 유니코드로 '가'를 나타낸다.`



# 문자형 리터럴 - 특수문자

특수문자 표기	의미
\\	역슬래시
\b	백스페이스
\r	캐리지 리턴
\f	폼피드
\t	수평 탭
\n	새 라인
\'	단일 따옴표
\"	이중 따옴표
\udddd	16진수 dddd에 해당하는 유니코드 문자

# 논리형

- 논리형 boolean
  - true, false 값만 가질 수 있음

## *BooleanTest.java*

```
01 public class BooleanTest {  
02     public static void main(String args[]) {  
03         boolean b;  
04  
05         b = true;  
06         System.out.println("b : " + b);  
07         b = ( 1 > 2 );  
08         System.out.println("b : " + b);  
09  
10     }  
11 }
```

boolean형은 true  
또는 false만을 가질  
수 있습니다.



b : true  
b : false

# 변수 초기화

- 변수를 초기화하지 않고 사용하면 오류 발생

*VarInitTest.java*

```
01 public class VarInitTest {  
02     public static void main(String args[]) {  
03         int index;  
04  
05         index = index + 1;  
06         System.out.println("index : " + index);  
07     }  
08 }
```

초기화 하지 않고  
변수를 사용하였다.

컴파일 오류: The local variable index may not have been initialized

# 변수의 이름

- 식별자(identifier)
  - 변수 이름, 메소드 이름, 클래스 이름 등
- 식별자 이름을 정하는 규칙
  - 문자와 숫자의 조합. 특수문자는 `_` \$만 허용한다.
  - 첫 문자로 숫자 사용 불가능하다.
  - 첫 문자로 `_` \$도 가능하나 특별한 경우로 제한할 것
  - 두 번째 문자부터는 문자, 숫자, `_` \$ 가능하다.
  - 대문자와 소문자는 구별된다.
  - 식별자의 이름으로 자바 키워드(keyword)를 사용해서는 안됨
- 잘못된 변수 선언 예

```
int    1stPrizeMoney; // 첫글자가 숫자
double super;         // 키워드
int    #ofComputer;   // 첫글자가 허용되지 않는 기호
```

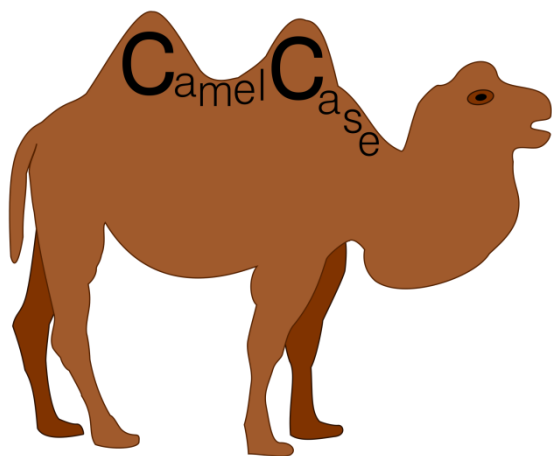
# 자바 키워드

abstract	continue	for	new	switch
assert	default	goto*	package	synchronized
boolean	do	if	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

- 키워드 const와 goto는 사용하지는 않지만 예약되어 있으므로 (reserved) 식별자 이름으로 사용할 수 없음
- 리터럴 true, false, null은 키워드는 아니지만 식별자 이름으로 사용할 수 없음

# 자바 식별자 관례

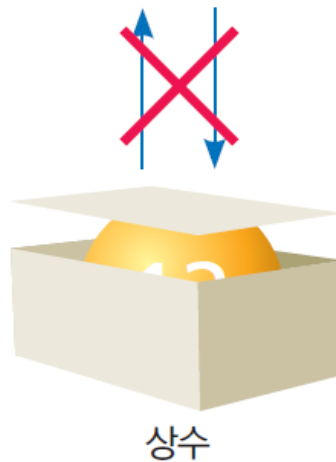
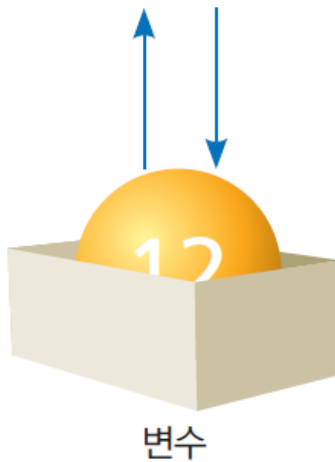
종 류	사용 방법	예
클래스명	각 단어의 첫글자는 대문자로 한다.	StaffMember, ItemProducer
변수명, 메소드명	첫번째 단어는 소문자로 시작되어 2번째 단어부터 첫글자는 대문자로 한다.	width, payRate, acctNumber, getMonthDays(), fillRect(),
상수	상수는 모든 글자를 대문자로 한다.	MAX_NUMBER



# 상수(constant)

- 상수는 프로그램을 실행하는 동안 값이 변하지 않는 수를 말한다.
  - 리터럴(literal) : 100, 3.14 등
  - 기호 상수 : 변수 선언시 final 키워드를 붙임.

**final double** PI = 3.141592;    // PI는 기호 상수



변수는 실행도중에  
값을 변경할수 있으나  
상수는 한번 값이  
정해지면 변경이  
불가능 합니다.



# 상수를 만드는 방법

## Constant.java

```
01 public class Constant {  
02     public static void main(String[] args) {  
03         final double KM_PER_MILE = 1.609344;  
04         double km;  
05         double mile = 60.0;  
06         km = KM_PER_MILE * mile;  
07  
08         System.out.println("60마일은 " + km + "킬로미터입니다.");  
09     }  
10 }
```

final을 붙여서 부동소수점형  
기호상수를 정의하고 있다.

60마일은 96.56064킬로미터입니다.



# LAB: 지구에서 가장 가까운 별까지의 거리 계산하기

*CalTime.java*

```
01 public class CalTime {
02
03     public static void main(String[] args) {
04         final double light_speed = 30e4;
05         double distance = 40e12;
06
07         double secs;
08
09         secs = distance/light_speed;
10
11         double light_year = secs/(60.0*60.0*24.0*365.0);
12         System.out.println("걸리는 시간은 " + light_year + "광년입니다.");
13     }
14 }
```

final을 붙여서 빛의 속도를  
부동소수점형 기호상수로  
정의하고 있다.

걸리는 시간은 4.227972264501945광년입니다.

# 수식

- 수식(expression)은 피연산자와 연산자의 조합
  - 연산자(operator) : +, -, \*, / 등
  - 피연산자(operand) : 변수, 상수 등

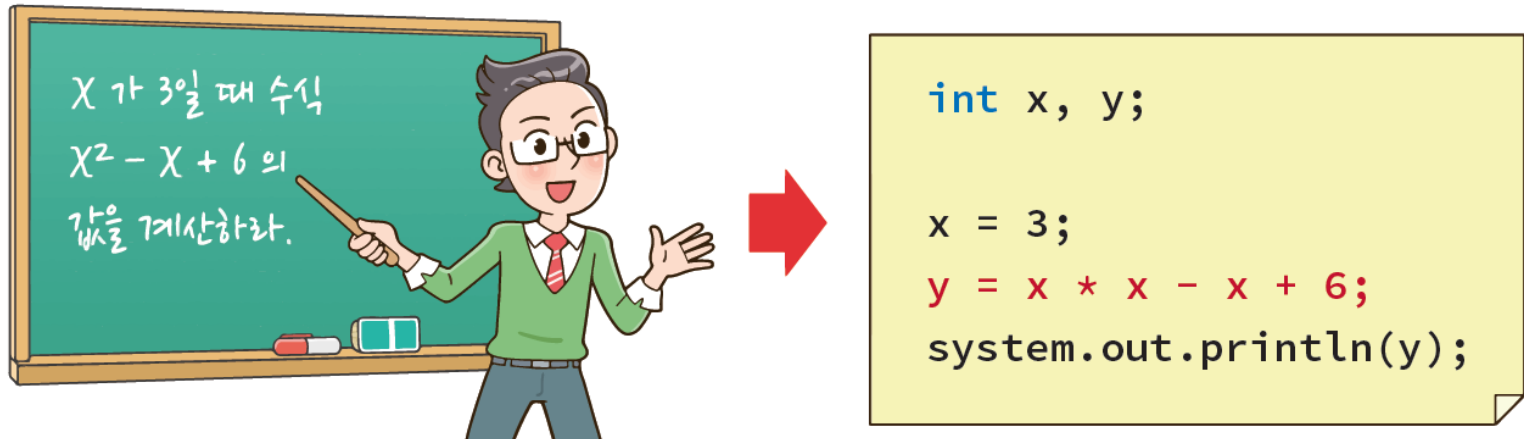


그림 2-2 • 수식의 예

# 대입 연산자

- 대입 연산자(=)는 왼쪽 변수에 오른쪽 수식의 값을 계산하여 저장
- 예:

$x = 2 + 3;$  // 5라는 정수 값을 변수 x에 대입한다.

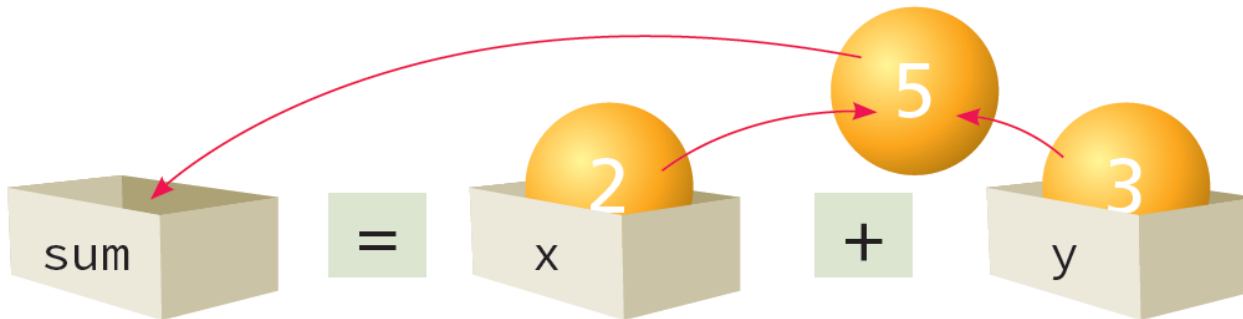


그림 2-4 • 산술 연산의 과정: 먼저 x와 y에서 값을 가져와서 덧셈연산이 수행되고 그 결과값이 sum에 저장된다.

# 산술 연산자

연산자	기호	의미	예
덧셈	+	x와 y를 더한다	$x+y$
뺄셈	-	x에서 y를 뺀다.	$x-y$
곱셈	*	x와 y를 곱한다.	$x*y$
나눗셈	/	x를 y로 나눈다.	$x/y$
나머지	%	x를 y로 나눌 때의 나머지값	$x\%y$

# 증감 연산자

연산자	의미
<code>++x</code>	x값을 먼저 증가한 후에 다른 연산에 사용한다. 이 수식의 값은 증가된 x값이다.
<code>x++</code>	x값을 먼저 사용한 후에, 증가한다. 이 수식의 값은 증가되지 않은 원래의 x값이다.
<code>--x</code>	x값을 먼저 감소한 후에 다른 연산에 사용한다. 이 수식의 값은 감소된 x값이다.
<code>x--</code>	x값을 먼저 사용한 후에, 감소한다. 이 수식의 값은 감소되지 않은 원래의 x값이다.

예)

```
int a = 3;
int b = 10;
int c = ++a * b--;
System.out.println(a);           // 4
System.out.println(b);           // 9
System.out.println(c);           // 40
```

# 관계 연산자

연산자 기호	의미	사용예
==	x와 y가 같은가?	$x == y$
!=	x와 y가 다른가?	$x != y$
>	x가 y보다 큰가?	$x > y$
<	x가 y보다 작은가?	$x < y$
>=	x가 y보다 크거나 같은가?	$x >= y$
<=	x가 y보다 작거나 같은가?	$x <= y$

# 논리 연산자

연산자 기호	사용예	의미
&&	x && y	AND 연산, x와 y가 모두 참이면 참, 그렇지 않으면 거짓
	x    y	OR 연산, x나 y중에서 하나만 참이면 참, 모두 거짓이면 거짓
!	!x	NOT 연산, x가 참이면 거짓, x가 거짓이면 참

# 비트 연산자

연산자	의미	예
~	비트별 NOT	~(0x0FFF)은 0xfffff000이 된다.
&	비트별 AND	(0x0FFF & 0xFFF0)은 0x0FF0이 된다.
^	비트별 XOR	(0x0FFF ^ 0xFFF0)은 0xF00F이 된다.
	비트별 OR	(0x0FFF   0xFFF0)은 0xFFFF이 된다.
<<	비트 왼쪽 이동	0xFFF << 4는 0xFFF0이 된다.
>>	비트 오른쪽 이동	0xFFF0 >> 4는 0xFFF이 된다.
>>>	비트 오른쪽 이동(unsigned)	왼쪽 비트가 부호비트로 채워지지 않고 0으로 채워진다

예)

```
int a = 3;
int b = a << 2;
int c = a | 5;
System.out.println(a);           // 3
System.out.println(b);           // 12
System.out.println(c);           // 7
```



# 형 변환(type conversion)

- 자동적인 형 변환

- 산술 연산에서의 자동 형 변환(확대 변환)

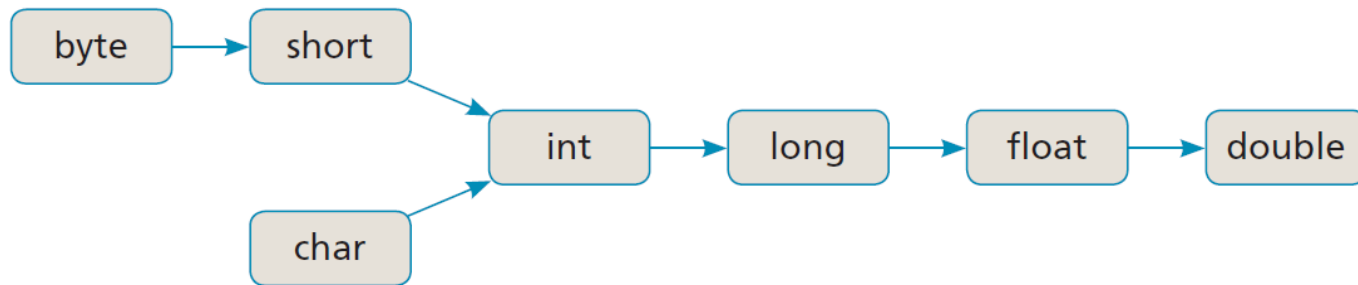
- if 한 피연산자가 double형, 다른 피연산자도 double형으로 변환

- else if 한 피연산자가 float형, 다른 피연산자도 float형으로 변환

- else if 한 피연산자가 long형, 다른 피연산자도 long형으로 변환

- else 모든 피연산자는 int형으로 변환

- 확대 변환 : 다음 그림의 화살표 방향



- 단, 정수를 float 형으로 변환시 정밀도를 잃을 수 있음

- `float x = 11112345678L; // float형이 표현할 수 있는 유효숫자를 초과`
    - `System.out.print(x); // 출력: 1.11123456E10`

# 형 변환

- 강제적인 형 변환

- 변환하려는 값의 앞에 원하는 자료형을 괄호와 함께 적어줌(type casting)

형식

(새로운 자료형) 수식;

- 예:

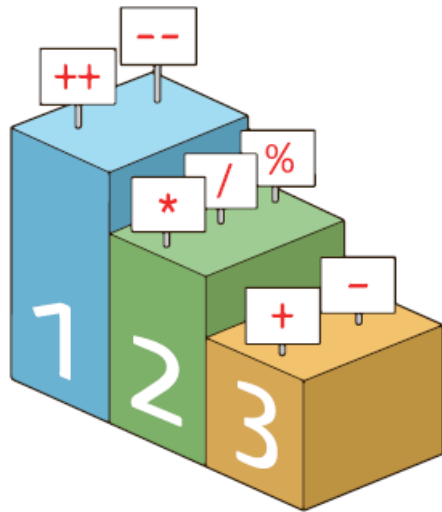
```
double x = 10.3;
```

```
int a = (int) x;
```

```
System.out.println(a);           // 10
```

```
System.out.println(x);           // 10.3
```

# 연산자의 우선순위(precedence)



우선 순위는 어떤 연산자를  
먼저 계산할지를 나타내는  
순위입니다.



그림 2-5 • 산술 연산자의 우선 순위

$$x + y * z$$

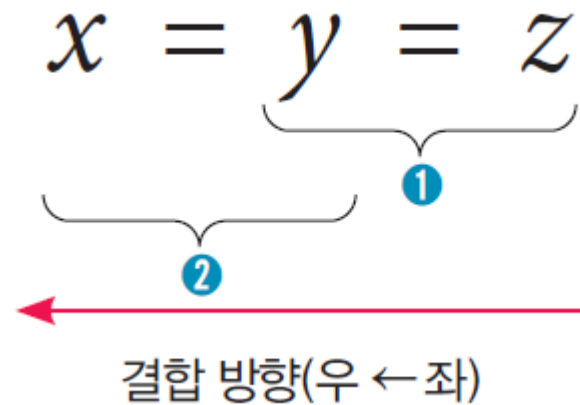
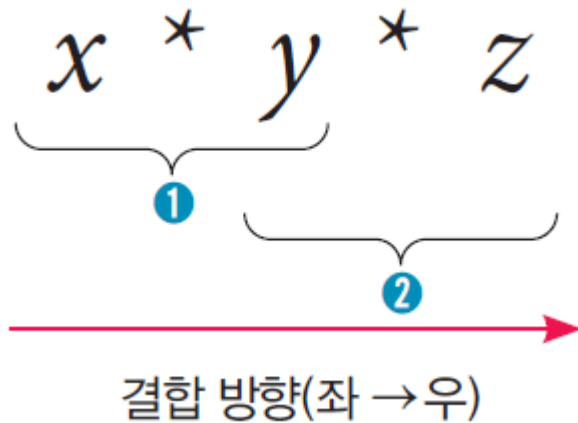
The diagram shows the expression  $x + y * z$  with two curly braces indicating the order of operations. The first brace, labeled with a blue circle containing the number 1, is positioned under the  $y * z$  part of the expression. The second brace, labeled with a blue circle containing the number 2, is positioned under the entire expression  $x + y * z$ .

# 연산자의 우선순위(precedence)

Operators	Precedence
postfix	<i>expr++ expr--</i>
unary	<i>++expr --expr +expr -expr ~ !</i>
multiplicative	<i>* / %</i>
additive	<i>+ -</i>
shift	<i>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</i>
relational	<i>&lt; &gt; &lt;= &gt;= instanceof</i>
equality	<i>== !=</i>
bitwise AND	<i>&amp;</i>
bitwise exclusive OR	<i>^</i>
bitwise inclusive OR	<i> </i>
logical AND	<i>&amp;&amp;</i>
logical OR	<i>  </i>
ternary	<i>? :</i>
assignment	<i>= += -= *= /= %= &amp;= ^=  = &lt;&lt;= &gt;&gt;= &gt;&gt;&gt;=</i>

# 결합 규칙

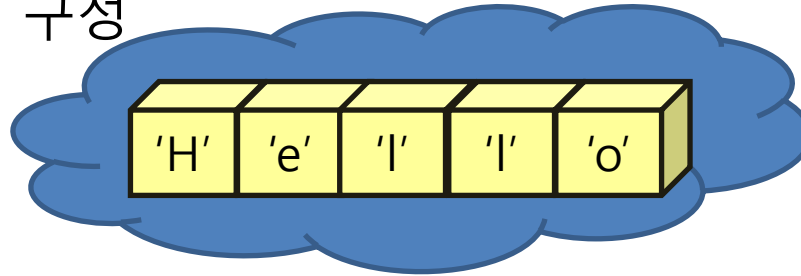
- 동일한 우선 순위의 연산이 있는 경우에 무엇을 먼저 수행하느냐에 대한 규칙



# 문자열

- 자바에서 문자열(string)은 문자들의 모임이다.
- 자바 언어에 내장된 문자열 자료형은 없고, 대신 문자열 표현을 위해 String 클래스가 제공된다.

문자열 "Hello"의 구성



- 문자열 결합에 +를 이용할 수 있다.

```
int age = 20;
```

```
String s = "I am " + age + "years old.";
```

## 예: 문자열 프로그램

### *StringTest.java*

```
01 public class StringTest {  
02     public static void main(String args[]) {  
03         String s1 = "Hello World!";  
04         String s2 = "I'm a new Java programmer!";  
05  
06         System.out.println(s1 + "\n" + s2);  
07     }  
08 }
```

Hello World!

I'm a new Java programmer!

# 사용자로부터 입력 받으려면?

- Scanner 클래스를 사용한다.

```
import java.util.Scanner; // (1) Scanner 클래스 포함

public class ScannerTest {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in); // (2) Scanner 객체 생성
        System.out.print("문장을 입력하시오: "); // 입력 안내문 출력
        String line = input.nextLine();           // 사용자로부터 한 줄 입력 받음
        System.out.println(line);                 // 입력 받은 문장을 출력
    }
}
```



# 사용자 입력

- Scanner는 자바 클래스 라이브러리의 일종으로서 입력을 받을 때 사용한다.

## (1) import 문장

- Scanner 클래스를 사용하려면 import 문장을 이용하여 Scanner 클래스를 포함시켜야 함

**import** java.util.Scanner; // java.util 패키지의 Scanner 포함

## (2) Scanner 객체를 생성하는 문장

- 표준입력(키보드입력)을 위해서는 System.in을 매개변수로 하여 Scanner 객체를 생성해야 함

Scanner input = **new** Scanner(System.in);

// 키보드 입력을 담당하는 객체를 생성하여 input이라고 명명

# 사용자로부터 정수값을 입력 받으려면?

- Scanner 클래스의 nextInt() 메소드 사용

```
import java.util.Scanner; // Scanner 클래스 포함

public class ScannerTest {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in); // Scanner 객체 생성
        System.out.println("정수값을 두 개 입력하십시오: ");
        int a = input.nextInt(); // 사용자로부터 int형 값 하나를 입력 받음
        int b = input.nextInt(); // 사용자로부터 int형 값 하나를 입력 받음
        System.out.println(a + b);
    }
}
```

# 예: 사용자로부터 받은 2개의 정수 더하기



첫번째 숫자를 입력하시오: 10

두번째 숫자를 입력하시오: 20

30

## Add2.java

```
01 // 사용자가 입력한 두 개의 숫자를 더해서 출력한다.
02 import java.util.Scanner; // Scanner 클래스 포함
03
04 public class Add2 {
05     // 메인 메소드에서부터 실행이 시작된다.
06     public static void main(String args[]) {
07         Scanner input = new Scanner(System.in);
08         int x; // 첫 번째 숫자 저장
09         int y; // 두 번째 숫자 저장
10         int sum; // 합을 저장
11
12
13         System.out.print("첫번째 숫자를 입력하십시오: "); // 입력 안내 출력
14         x = input.nextInt(); // 사용자로부터 첫 번째 숫자를 읽는다.
15
16         System.out.print("두번째 숫자를 입력하십시오: "); // 입력 안내 출력
17         y = input.nextInt(); // 사용자로부터 두 번째 숫자를 읽는다.
18
19         sum = x + y; // 두 개의 숫자를 더한다.
20
21         System.out.println(sum); // 합을 출력한다.
22
23     }
24 }
```

println()은 출력 후 줄을 바꾸지만,  
print()는 출력 후 줄을 바꾸지 않음  
예) System.out.print("a");  
System.out.println("b");  
System.out.println("c");

nextInt()는 int형 자료  
하나를 읽어들이

# 예: 원의 면적 계산하기

## CircleArea.java

```
01 import java.util.*;    // Scanner 클래스를 포함한다.
02
03 public class CircleArea {
04     public static void main(String args[]) {
05
06         double radius; // 원의 반지름
07         double area;   // 원의 면적
08         Scanner input = new Scanner(System.in);
09         System.out.print("반지름을 입력하시오: "); // 입력 안내 출력
10         radius = input.nextDouble();
11         area = 3.14 * radius * radius;
12
13         System.out.println(area);
14     }
15 }
```

nextDouble()은 double형 자료  
하나를 읽어들이

```
반지름을 입력하시오: 5
78.5
```

# 예: 직사각형의 둘레와 면적 계산하기

```
01 import java.util.*;
02
03 public class Box {
04     public static void main(String[] args) {
05         double w;
06         double h;
07         double area;
08         double perimeter;
09
10         Scanner input = new Scanner(System.in);
11
12         System.out.print("사각형의 가로를 입력하시오: ");
13         w = input.nextInt();
14         System.out.print("사각형의 세로를 입력하시오: ");
15         h = input.nextInt();
16         area = w * h;
17         perimeter = 2.0 * (w + h);
18
19         System.out.println("사각형의 넓이는 " + area);
20         System.out.println("사각형의 둘레는 " + perimeter);
21     }
22 }
```

사각형의 가로를 입력하시오: 10  
사각형의 세로를 입력하시오: 5  
사각형의 넓이는 50.0  
사각형의 둘레는 30.0