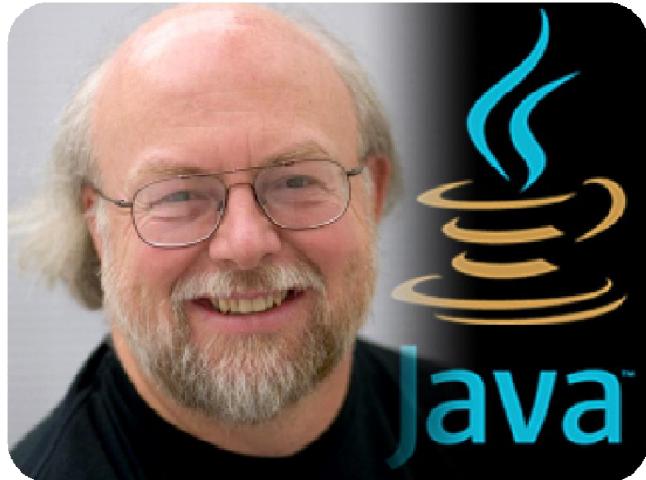
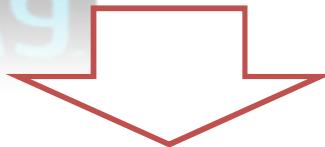


**JAVA ?**



- 1991년 썬마이크로시스템즈의 제임스고슬링과 켄아놀드
- **오디오, TV 세탁기 등 각각의 가전제 품을 제어하는 통합언어**를 만들기 위해서 시작된 그린프로젝트의 'Oak'라는 언어를 만들면서 시작



각각의 하드웨어와 운영체제에 종속받지 않고  
이식성이 높은 플랫폼을 만들기 위해서 시작

**Write once, Run anywhere**

## 자바의 특징

---

- ◆ 단순하다.
- ◆ 객체지향(Object-oriented)적이다.
- ◆ 분산(Distributed)환경에 적합하다.
- ◆ 인터프리터에(Interpreter)에 의해 실행된다.
- ◆ 견고(Robust)한 기능을 제공한다.
- ◆ 안전(Secure)하다.
- ◆ 구조중립(Architecture-neutral)적이고 이식성(Potable)이 높다.
- ◆ 높은 성능(High-performance)을 제공한다.
- ◆ 다중 쓰레드(Multithreaded)를 제공한다.
- ◆ 동적(Dynamic)이다.

# Dynamic Binding / Linking

---

## ◆ Binding

- Static Binding
  - 컴파일 시에 어떤 클래스의 어떤 메소드가 호출되는지 정한다.
- Dynamic Binding
  - 실행 중에 어떤 클래스의 어떤 메소드가 호출되는지 정한다.

# **Dynamic Binding / Linking**

---

## ◆ **Linking**

- **Static Linking**

- 라이브러리나 다른 메소드 호출 시 실행파일에 그 부분을 합침.
  - 실행파일 크기가 커짐 / 호출 성능 향상.

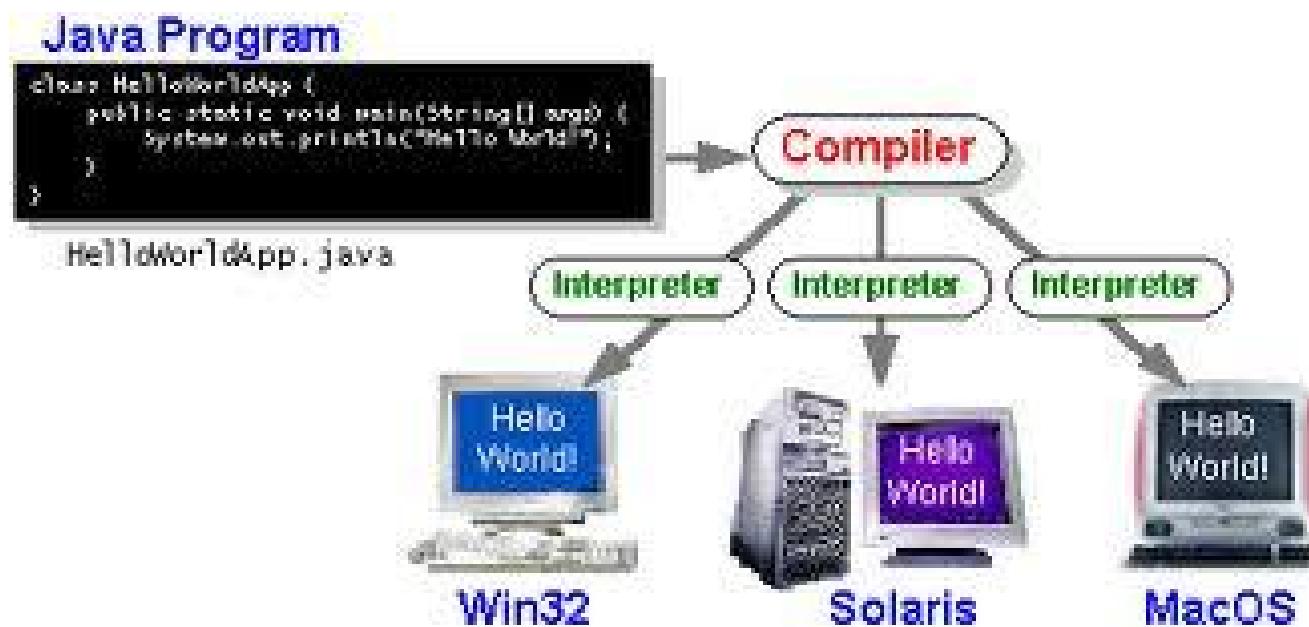
- **Dynamic Linking**

- 호출하는 부분에 메소드를 호출한다는 표시만 한다. 실행되는 부분은 다른 파일에 저장.
  - 실행파일 크기가 작아짐 / 실행 중 동적으로 찾아야 함으로 시간이 걸린다.
  - 실행파일 내부에 호출할 부분의 주소가 아닌 호출한 부분의 정확한 이름을 기재해야 이후에 찾을 수 있다.(주소는 무의미)

## ◆ **자바는 완벽한 Dynamic Binding / Dynamic Linking 지원**

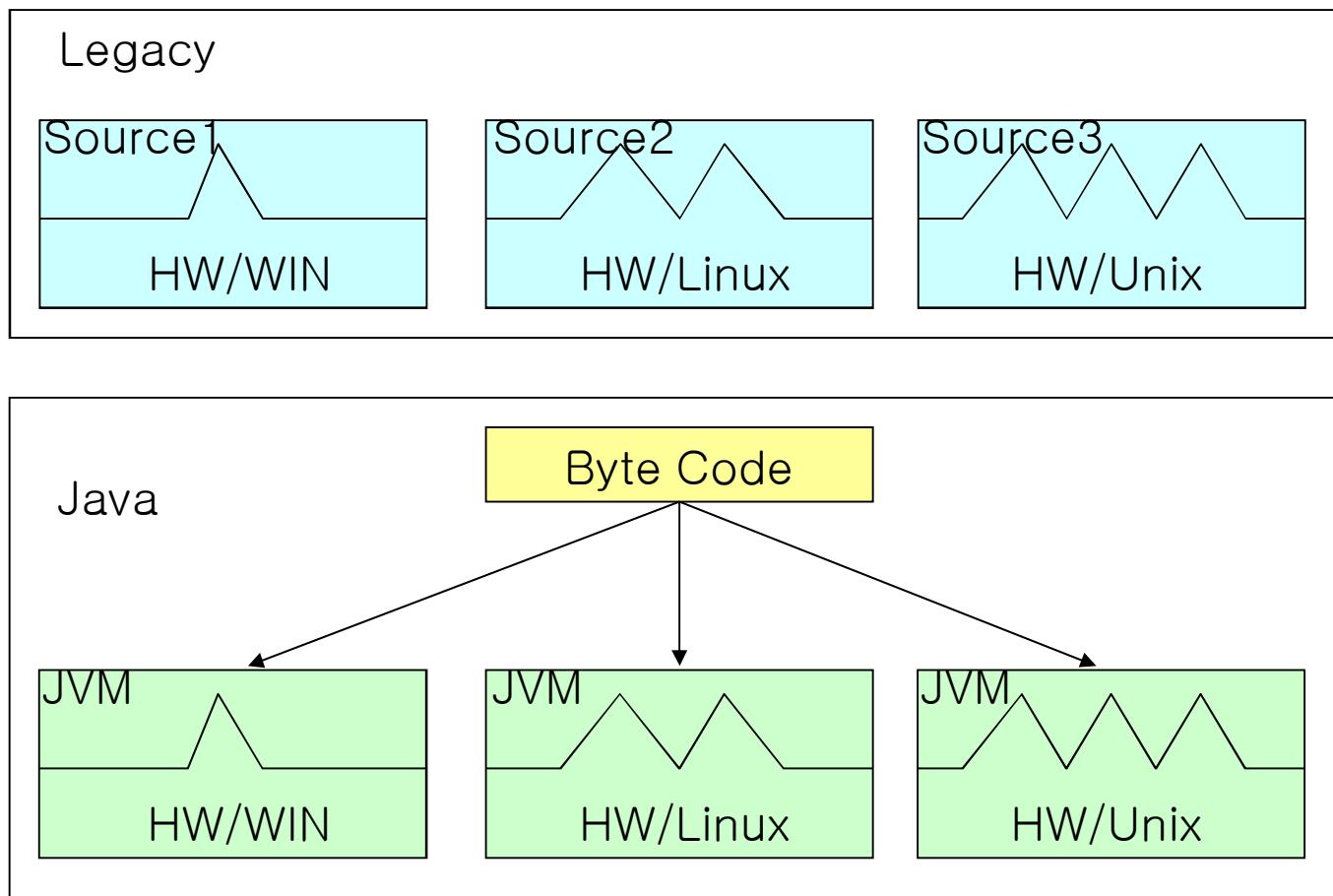
# 자바의 특징

- 컴파일 환경이 아니라, 인터프리터 환경

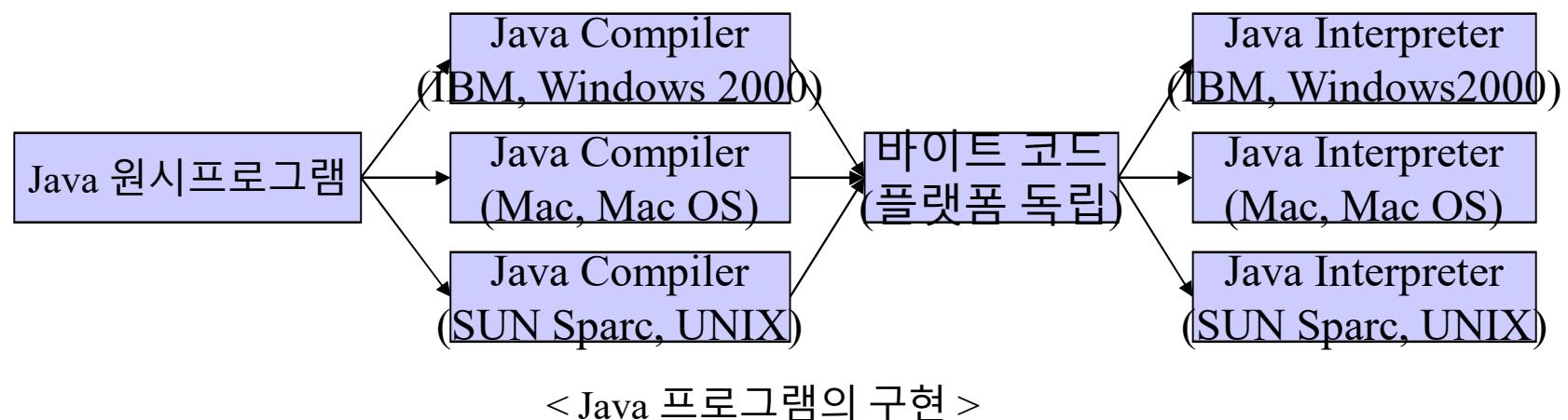
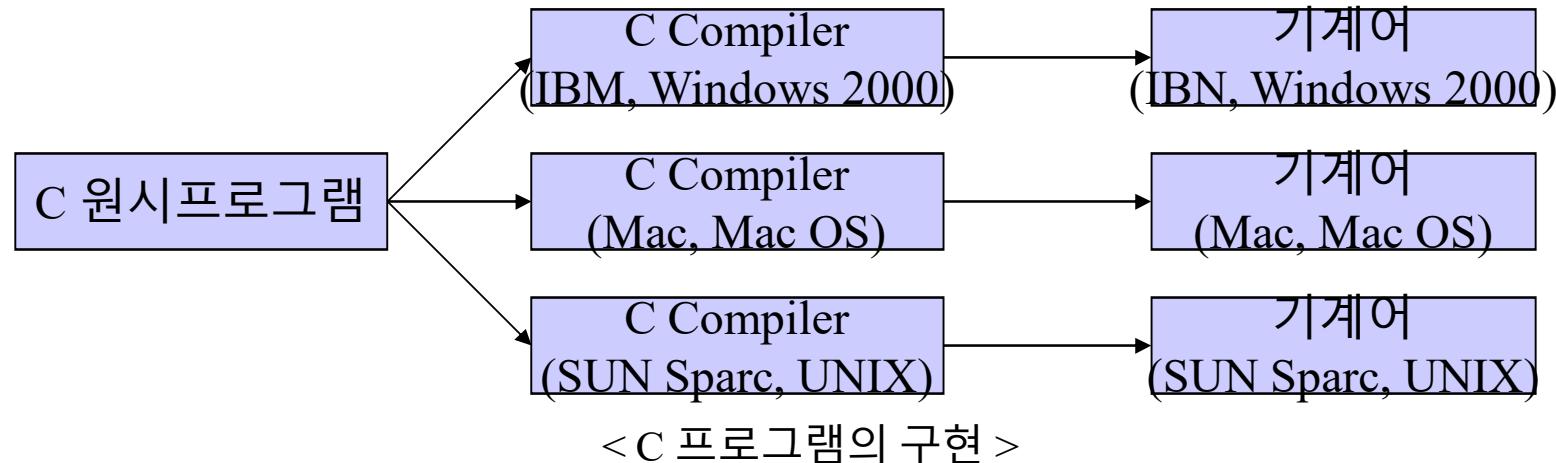


# 플랫폼 독립

- ◆ 자바는 VM을 통해서 플랫폼 독립을 제공한다. (자바의 가장 큰 장점)

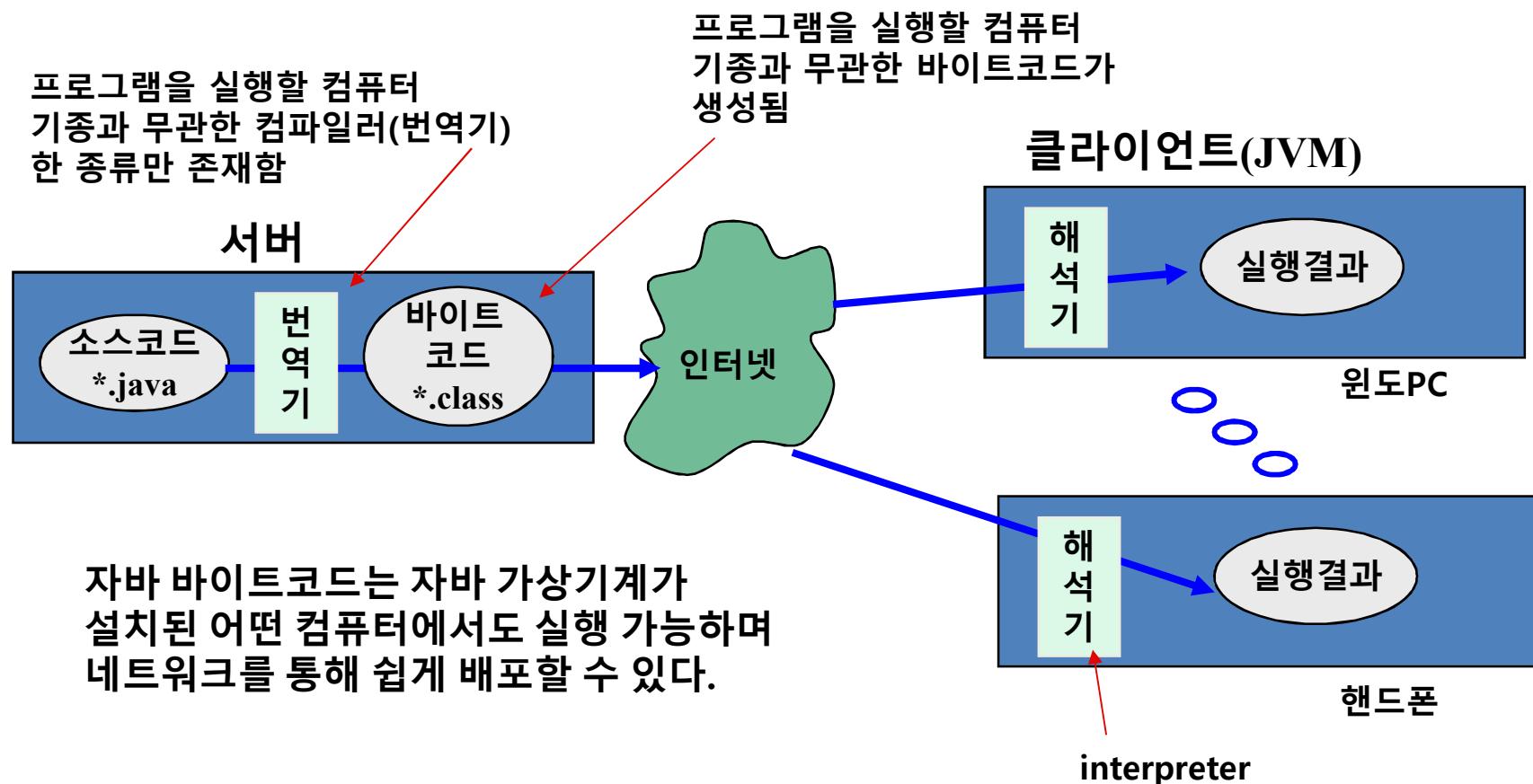


# 자바의 컴파일 및 로드 방식



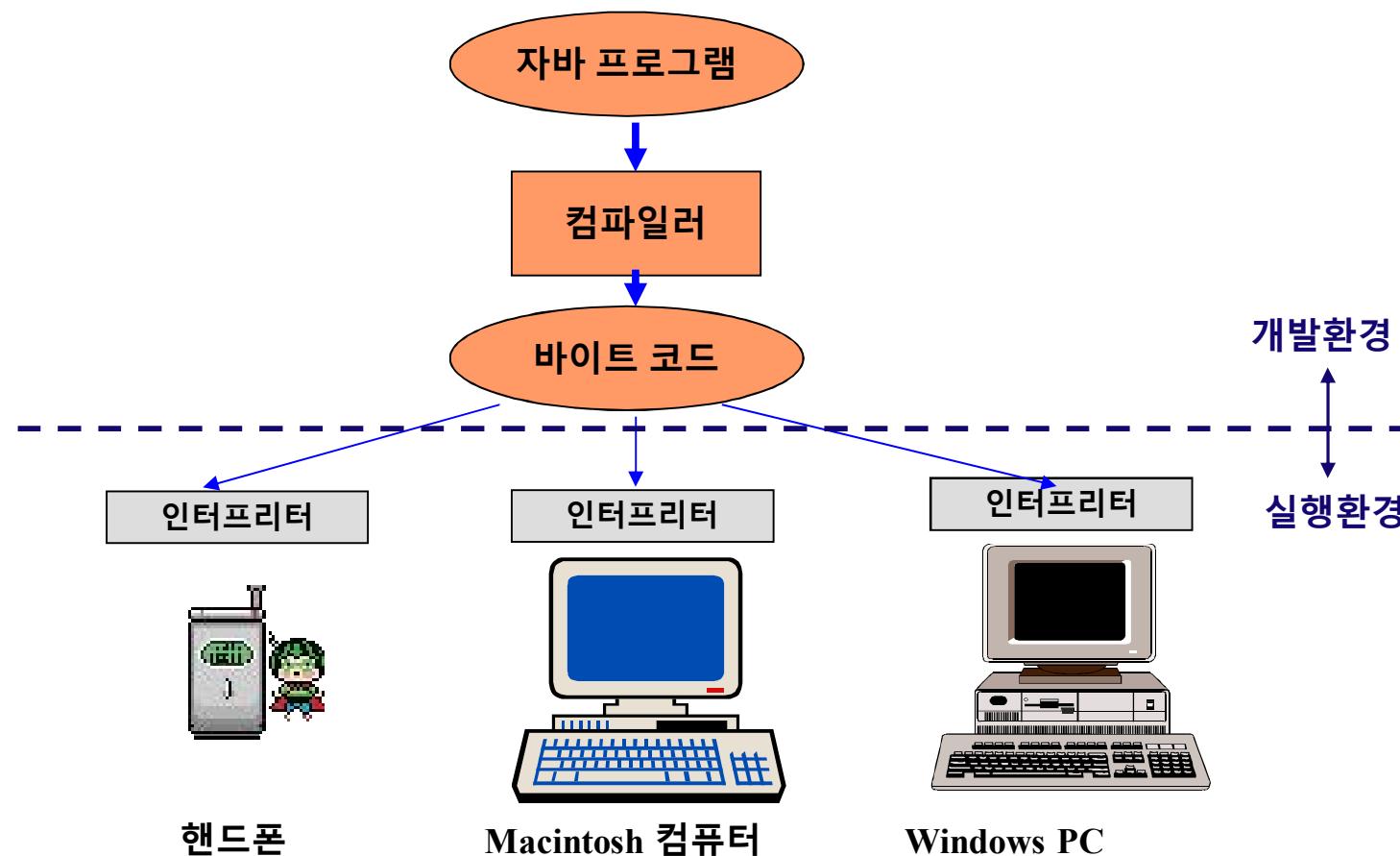
# 자바의 특징

## ◆ Java Virtual Machine



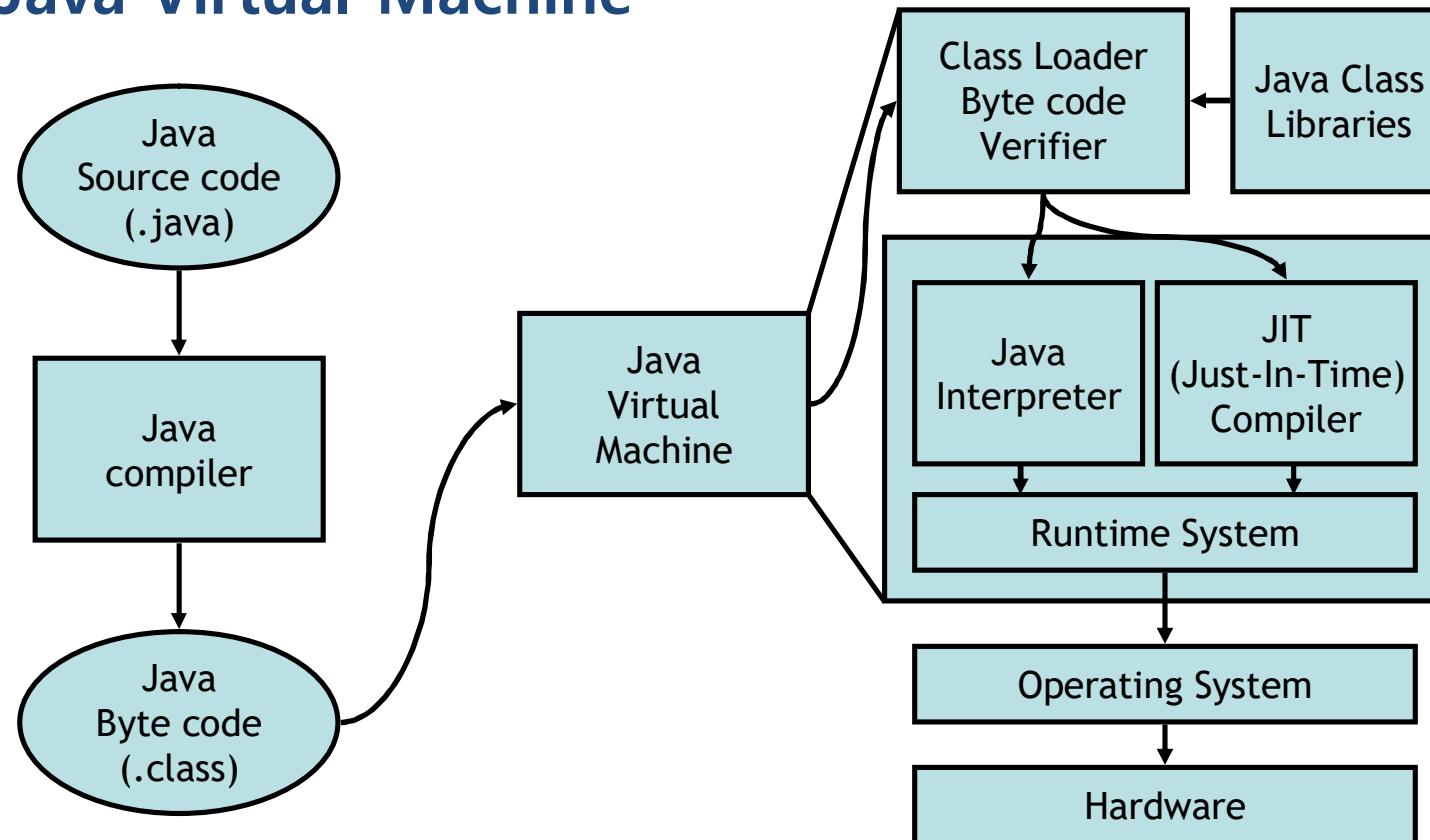
# 자바의 특징

## ◆ Java Virtual Machine



# 자바의 특징

## Java Virtual Machine



Java ?

# 자바의 주요 문법

# 자바 프로그램 작성의 기초

## ◆ 주석

- 주석(comment) : 프로그램의 실행에 영향을 미치지 않게 만들어진 텍스트

```
1  /* 프로그램 이름 : HelloJava10
2      프로그램 설명 : Hello, Java를 10번 출력하는 프로그램
3      작성일 : 2006/2/11
4      작성자 : 흥길동 */
5  class HelloJava10 {
6      public static void main(String args[]) {
7          int num = 0;           // num: 반복 회수를 카운트하는 변수
8          while (num < 10) {
9              System.out.println("Hello, Java");
10             num = num + 1;
11         }
12     }
13 }
```



# 변수

## ◆ 변수

- 데이터를 담는 일종의 그릇
- 변수를 사용하기 위해서는 먼저 선언을 해야 함
- 변수의 선언문(declaration statement)

`int num;`

정수(int) 타입의 값을 num이라는 이름의 변수를  
담을 수 있는 선언하는 선언문

세미콜론(;)은 명령문의 끝을  
표시하는 문자

- 변수 선언시 키워드는 사용하면 안됨

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

# 지역변수

---

## 지역변수 (local variable) : 메소드 안에 선언한 변수

- 타입 식별자;

```
int num;  
float f2;  
long t2;
```

- 타입 식별자=초기화;

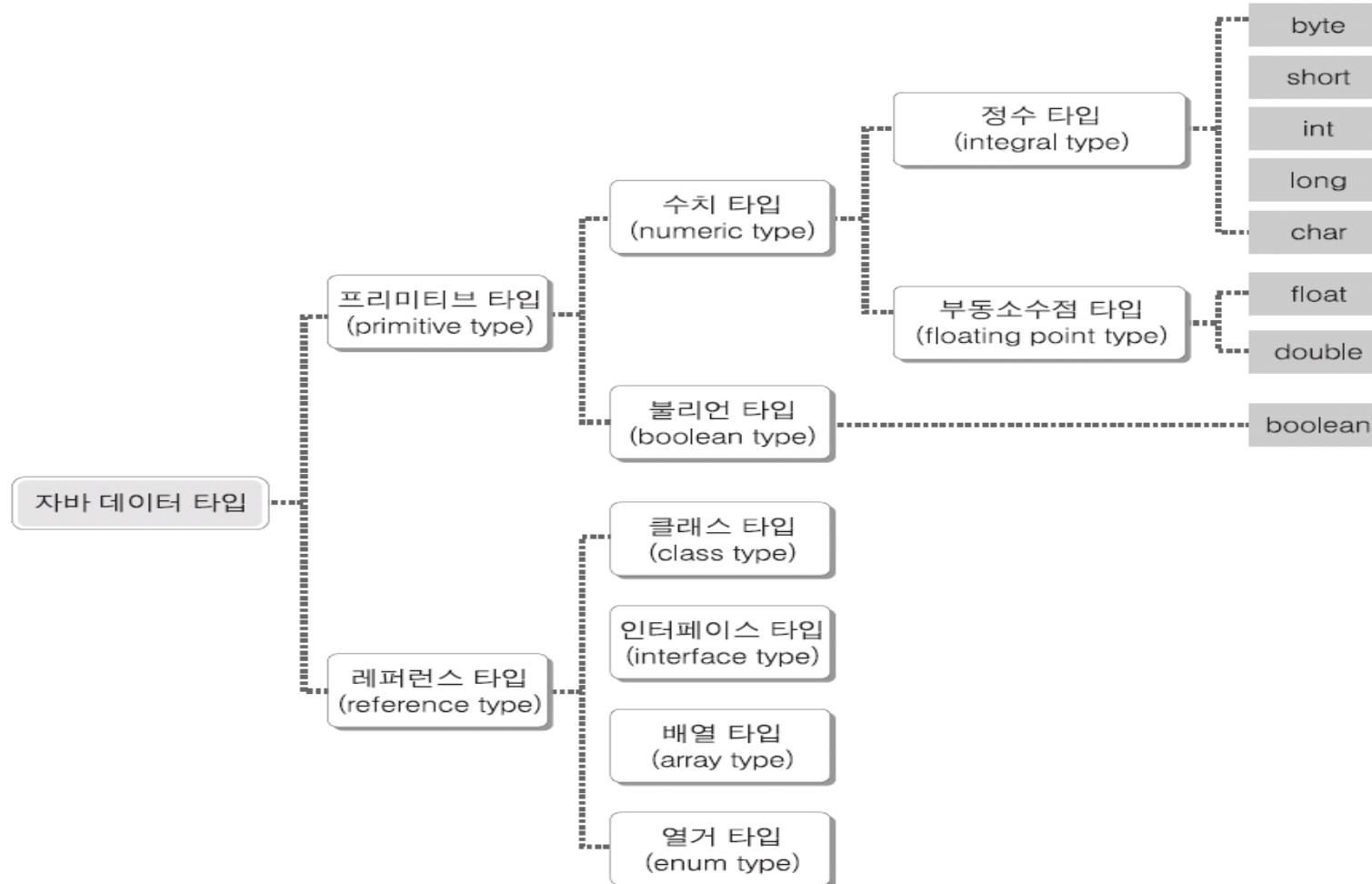
```
int a=10;  
long age=30  
String name="hello java";
```

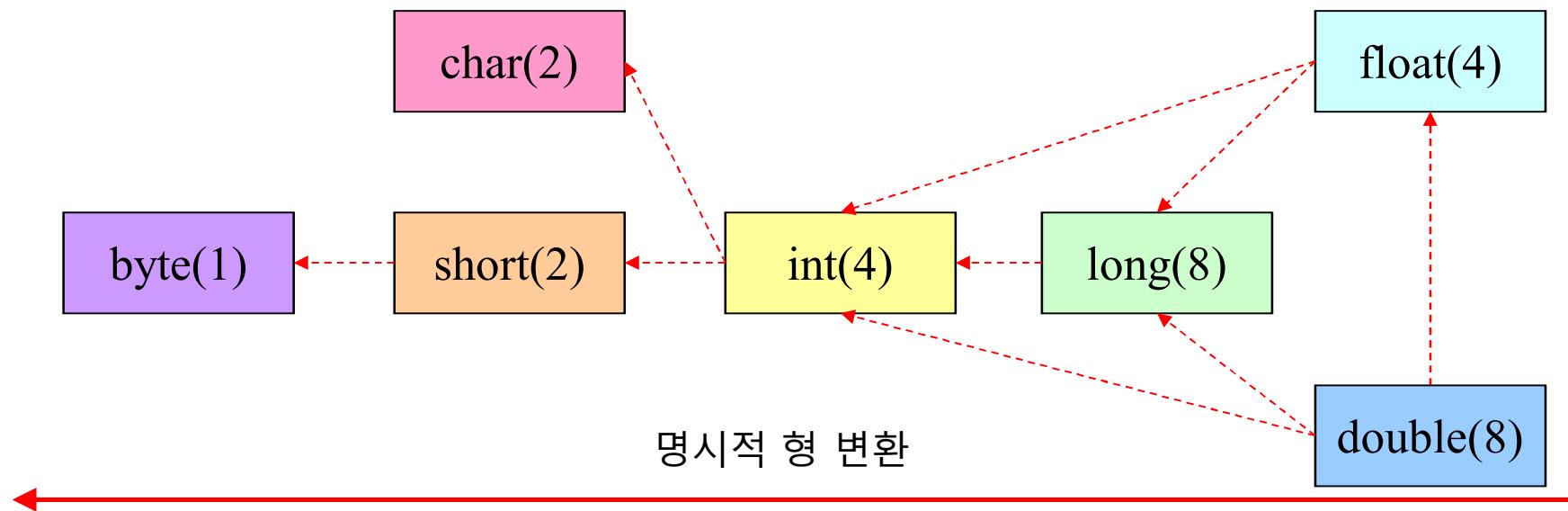
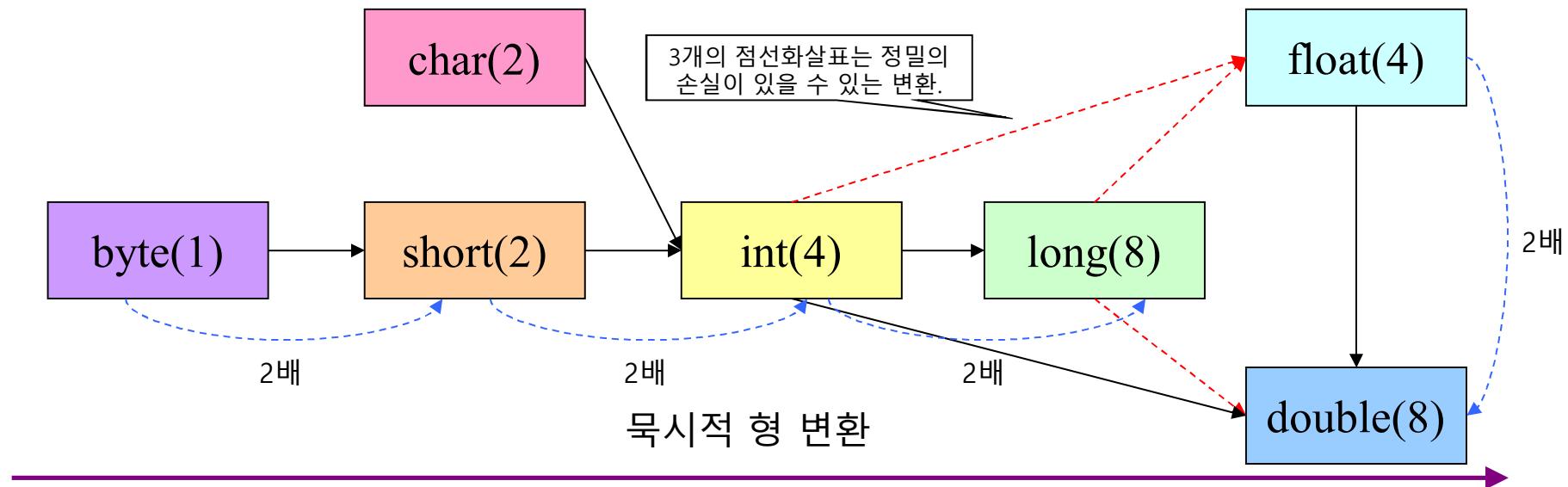
- 타입 식별자1, 식별자2; //한꺼번에 선언

```
int w,h;  
double pi=3.24, radius;
```

# 데이터 타입

## ◆ 자바 데이터 타입의 분류 체계



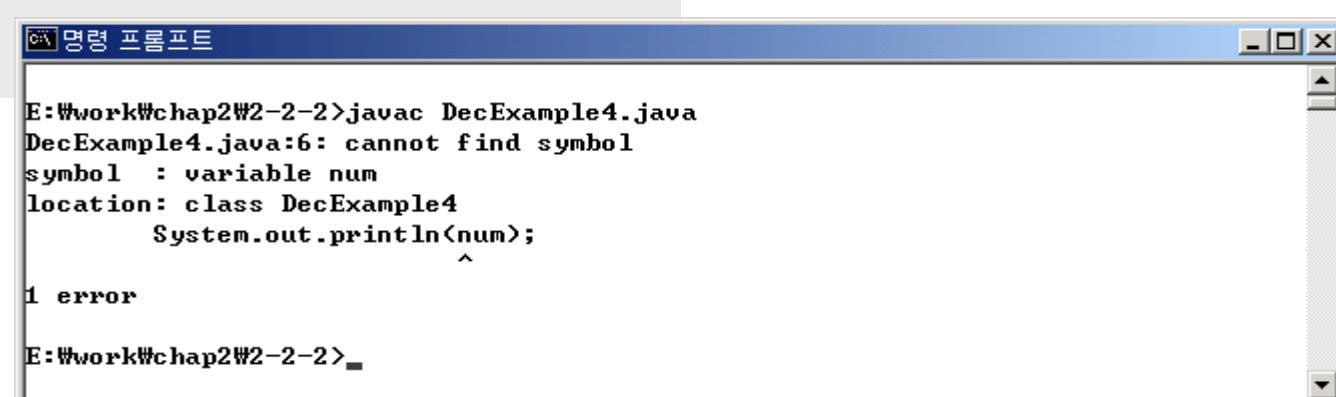


# 자바의 기초 문법

## ◆ 로컬 변수의 사용 범위

-블록 안에서 선언된 변수를 잘못 사용한 예

```
1  class DecExample4 {  
2      public static void main(String args[]) {  
3          {  
4              int num = 10; ----- 블록 안에 선언된 변수를 했습니다.  
5          }  
6          System.out.println(num); ----- 블록 안에 선언된 변수를 블록 밖에서  
7      }  
8  }
```



```
E:\work\chap2\2-2>javac DecExample4.java  
DecExample4.java:6: cannot find symbol  
symbol  : variable num  
location: class DecExample4  
        System.out.println(num);  
                           ^  
1 error  
E:\work\chap2\2-2>
```

# 자바의 기초 문법

## ◆ final 변수

-final 변수 : 변수에 값을 딱 한번만 대입할 수 있는 변수

-final 변수의 사용 예

```
1  class FinalExample1 {  
2      public static void main(String args[]) {  
3          final double pi = 3.14;           ----- final 변수를 선언합니다  
4          double radius = 2.0, circum;  
5          circum = 2 * pi * radius;       ----- final 변수를 사용합니다  
6          System.out.println(circum);  
7      }  
8  }
```



# 연산자

## ◆ 자바의 연산자들

구분	연산자	기능 설명
사칙 연산자	+ - * / %	사칙연산 및 나눗셈의 나머지 계산
부호 연산자	+	음수와 양수의 부호
문자열 연결 연산자	+	두 문자열을 연결
단순 대입 연산자	=	우변의 값을 좌변의 변수에 대입
증가/감소 연산자	++ --	변수 값을 1만큼 증가/감소
수치 비교 연산자	< > <= >=	수치의 크기 비교
동등 연산자	== !=	데이터의 동일 비교
논리 연산자	&   ^ !	논리적 AND, OR, XOR, NOT 연산
조건 AND/OR 연산자	&&	최적화된 논리적 AND, OR 연산
조건 연산자	?:	조건에 따라 두 값 중 하나를 택일
비트 연산자	&   ^ ~	비트 단위의 AND, OR, XOR, NOT 연산
쉬프트 연산자	<< >> >>>	비트를 좌측/우측으로 밀어서 이동
복합 대입 연산자	  += -= *= /= %= &=  = ^= <<= >>= >>>=	+ - * / % &   ^ << >> >>> 연산자와 =의 기능을 함께 수행
캐스트 연산자	(타입 이름)	타입의 강제 변환

# 연산자

---

## ◆ 단항연산자

부호연산자 + , -

증감연산자 ++ --

ex ++x or x++ , --x, x--

## ◆ 이항 연산자

대입연산자 =

산술 연산자 + - \* / % (나머지)

복합대입연산자 += -= \*= /= %=

연결연산자 +

비교연산자 > < >= <= == (동등비교) !=

논리연산자 && || !(not) ^ (xor )

비트연산자 & | ~ ^

쉬프트 연산 >> <<

## ◆ 삼항연산자 – 조건연산자

조건? 참: 거짓:

# 연산자

---

## ◆ 부호 연산자

-부호 연산자와 덧셈 연산자의 차이

```
num2 = - num1;  
sum = + a + b;           // a앞의 +는 부호 연산자, a와 b사이의의 +는 덧셈 연산자
```

- 오류 구문

```
1  class SignExample1 {  
2      public static void main(String args[]) {  
3          short num1 = 100;  
4          short num2 = - num1;  
5          System.out.println(num2);  
6      }  
7  }
```

# 연산자

---

## ◆ 증가 연산자와 감소 연산자

-증가 연산자 ++ : 변수의 기존 값에 1을 더하는 연산자

```
int x=10;
```

```
int y==++x; // 전위 증가 연산
```

```
int x=10;
```

```
int y=x++; // 후위 증가 연산
```

-감소 연산자 -- : 변수의 기존 값에서 1을 빼는 연산자

```
int x=10;
```

```
int y=--x; //전위 감소연산
```

```
int x=10;
```

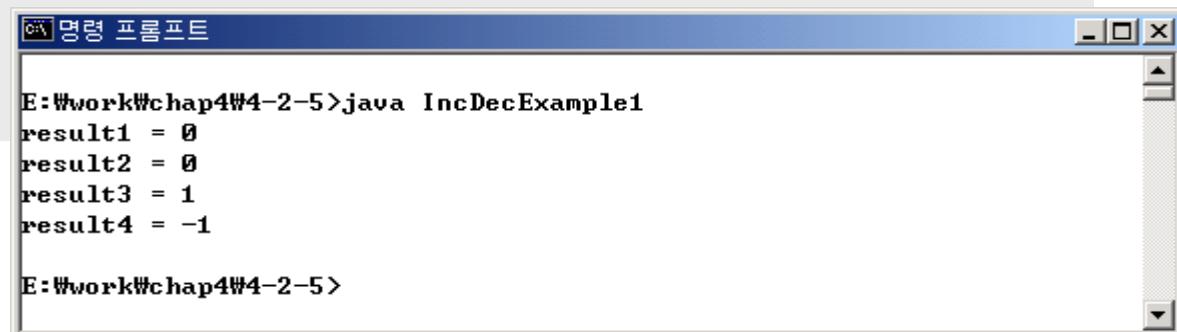
```
int y=x--; //후위 감소 연산
```

# 연산자

## ◆ 증가 연산자와 감소 연산자

- 증가 연산자와 감소 연산자가 산출하는 값을 출력하는 프로그램

```
1  class IncDecExample1 {  
2      public static void main(String args[]) {  
3          int num1 = 0, num2 = 0, num3 = 0, num4 = 0;  
4          int result1 = num1++;           // ++ 연산의 결과는 num1의 기존 값  
5          int result2 = num2--;           // -- 연산의 결과는 num2의 기존 값  
6          int result3 = ++num3;           // ++ 연산의 결과는 num3의 새로운 값  
7          int result4 = --num4;           // -- 연산의 결과는 num4의 새로운 값  
8          System.out.println("result1 = " + result1);  
9          System.out.println("result2 = " + result2);  
10         System.out.println("result3 = " + result3);  
11         System.out.println("result4 = " + result4);  
12     }  
13 }
```



# 연산자

---

## ◆ 대입문(assignment statement)

- 변수에 데이터를 담는 명령문
- 기호 `=`을 이용해서 만들 수 있음
- `data=10+20; // 10과 20을 더한 값을 data에 저장`

# 연산자

## ◆ 사칙 연산자

-사칙연산자 : 덧셈, 뺄셈, 곱셈, 나눗셈을 하는 연산자

$\underline{\text{피연산자1}} + \underline{\text{피연산자2}}$

↑  
이 값과 이 값을 더합니다.

$\underline{\text{피연산자1}} - \underline{\text{피연산자2}}$

↑  
이 값에서 이 값을 뺍니다.

$\underline{\text{피연산자1}} * \underline{\text{피연산자2}}$

↑  
이 값과 이 값을 곱합니다.

$\underline{\text{피연산자1}} / \underline{\text{피연산자2}}$

↑  
이 값을 이 값으로  
나눈 몫을 계산합니다.

$\underline{\text{피연산자1}} \% \underline{\text{피연산자2}}$

↑  
이 값을 이 값으로  
나눈 나머지를 계산합니다.

# 연산자

---

## ◆ 사칙 연산자

-사칙 연산자의 자동 타입 변환 (1)

```
100 + 200L          // 결과는 long 타입의 300  
3.0 - 2            // 결과는 double 타입의 1.0  
10.0f / 2L          // 결과는 float 타입의 5.0
```

-사칙 연산자의 자동 타입 변환 (2)

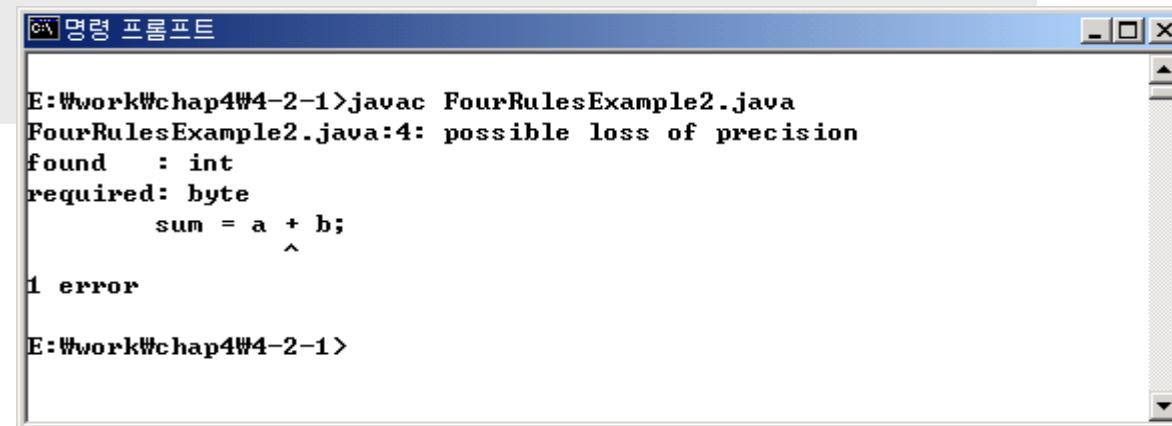
```
byte a = 2, b = 3;  
short c = 4;  
sum = a + b;        // a + b의 결과는 byte 타입이 아니라 int 타입이 됨  
diff = c - b;       // c - b의 결과는 short 타입이 아니라 int 타입이 됨
```

# 연산자

## ◆ 사칙 연산자

-사칙 연산자의 자동 타입 변환으로 문제가 발생하는 프로그램

```
1  class FourRulesExample2 {  
2      public static void main(String args[]) {  
3          byte a = 2, b = 3, sum;  
4          sum = a + b;  
5          System.out.println(sum);  
6      }  
7  }
```



The screenshot shows a Windows command prompt window titled "명령 프롬프트". The command entered is "E:\work\chap4\4-2-1>javac FourRulesExample2.java". The output shows a single error message: "FourRulesExample2.java:4: possible loss of precision found : int required: byte sum = a + b; ^ 1 error". The window has a standard Windows title bar and a scroll bar on the right side.

# 연산자

## ◆ 복합 대입 연산자

-복합 대입 연산자의 사용 예

```
1  class AssignmentExample1 {  
2      public static void main(String args[]) {  
3          int num = 17;  
4          num += 1;        // num = num + 1; 과 동일  
5          num -= 2;        // num = num - 2; 과 동일  
6          num *= 3;        // num = num * 3; 과 동일  
7          num /= 4;        // num = num / 4; 과 동일  
8          num %= 5;        // num = num % 5; 과 동일  
9          System.out.println(num);  
10         }  
11     }
```



# 연산자

---

`num += 3;`



`num = num + 3;` 과 똑같은 일을 하는 명령문

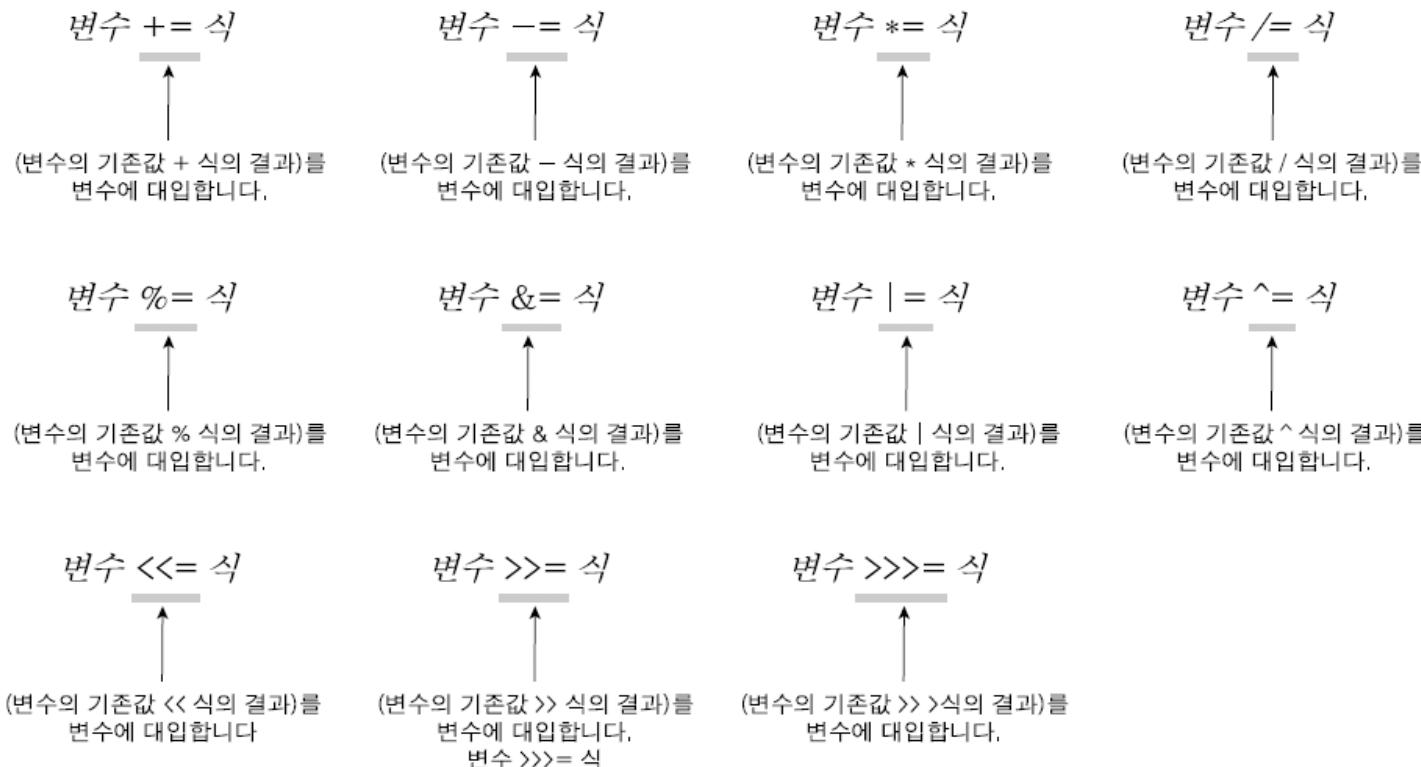
-사칙연산과 대입을 함께 수행하는 복합대입 연산자

연산자	설명
<code>+=</code>	<code>+와 = 기능의 복합</code>
<code>--</code>	<code>-와 = 기능의 복합</code>
<code>*=</code>	<code>*와 = 기능의 복합</code>
<code>/=</code>	<code>/와 = 기능의 복합</code>
<code>%=</code>	<code>%와 = 기능의 복합</code>

# 연산자

## ◆ 복합 대입 연산자

-복합 대입 연산자 : +, -, \*, /, %, &, |, ^, <<, >>, >>> 와 = 연산자의 기능을 함께 수행하는 연산자

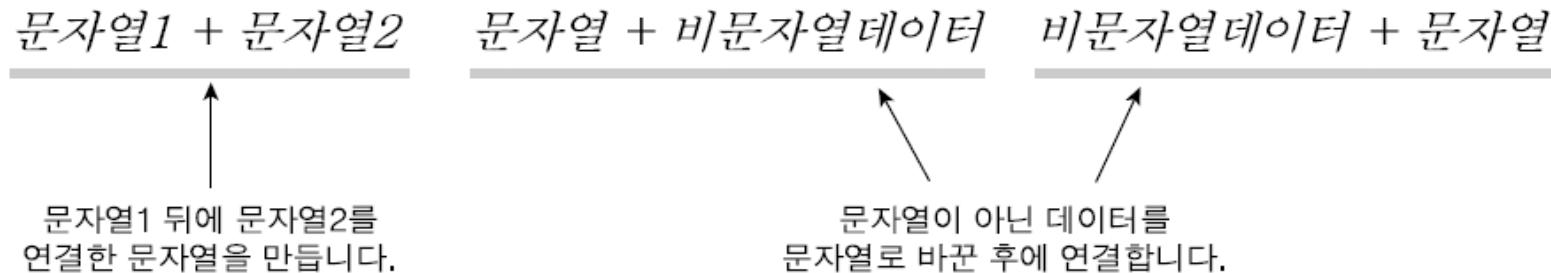


# 연산자

---

## ◆ 문자열 연결 연산자

-문자열 연결 연산자 : 두 개의 문자열을 연결해서 새로운 문자열을 만드는 연산자



**System.out.println("결과는 "+3+4);**

**vs**

**System.out.println(3+4+"결과는 ");**

# 연산자

## ◆ 비교 연산자

-비교 연산자 : 두 수의 크기를 비교하는 연산자

피연산자1 < 피연산자2

이 값이 이 값보다 작으면 true,  
그렇지 않으면 false입니다.

피연산자1 > 피연산자2

이 값이 이 값보다 크면 true,  
그렇지 않으면 false입니다.

피연산자1 <= 피연산자2

이 값이 이 값보다 작거나 같으면 true,  
그렇지 않으면 false입니다.

피연산자1 >= 피연산자2

이 값이 이 값보다 크거나 같으면 true,  
그렇지 않으면 false입니다.

▶▶ 피연산자는 모두 수치 타입이어야 합니다.

# 연산자

## ◆ 동등 연산자

-동등 연산자 : 두 데이터의 값이 같은지 다른지 판단하는 연산자

피연산자1 == 피연산자2

↑                   ↑  
이 값과 이 값이 같으면 true,  
다르면 false입니다.

피연산자1 != 피연산자2

↑                   ↑  
이 값과 이 값이 다르면 true,  
같으면 false입니다.

# 연산자

## ◆ 논리 연산자

- 논리 연산자 : boolean 값들을 가지고 논리식을 만드는 연산자

피연산자1 & 피연산자2



두 값이 모두 true면 true,  
그렇지 않으면 false입니다.

피연산자1 | 피연산자2



두 값이 모두 false면 false,  
그렇지 않으면 true입니다.

피연산자1 ^ 피연산자2



하나가 true, 하나가 false면 true,  
그렇지 않으면 false입니다.

!피연산자1



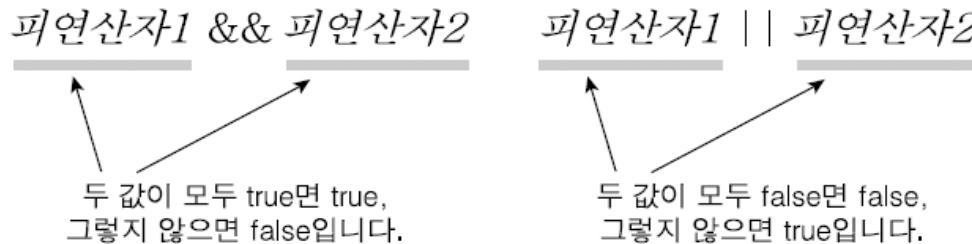
이 값이 true이면 false,  
false이면 true입니다.

▶▶ 피연산자는 모두 불리언 타입이어야 합니다. 피연산자가 정수 타입인 & | ^는 논리 연산자가 아니라 비트 연산자입니다.

# 연산자

## ◆ 조건 AND/OR 연산자

-조건 AND/OR 연산자 : 최적화된 AND/OR 연산자



▶▶ 피연산자는 모두 불리언 타입이어야 합니다.

-사용 예

true && false

false || true

# 연산자

## ◆ 비트 연산자

-비트 연산자 : 데이터 구성 비트를 가지고 AND, OR, XOR, NOT 연산을 수행하는 연산자

피연산자1 & 피연산자2



두 데이터의 같은 위치 비트들을  
AND 연산합니다.

피연산자1 | 피연산자2



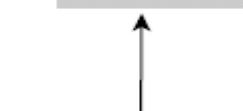
두 데이터의 같은 위치 비트들을  
OR 연산합니다.

피연산자1 ^ 피연산자2



두 데이터의 같은 위치 비트들을  
XOR 연산합니다.

~피연산자



데이터의 각 비트들을  
NOT 연산합니다.

# 연산자

## ◆ 비트 연산자

-비트 연산의 규칙

[비트 AND 연산]

피연산자2 \\	1	0
피연산자1		
1	1	0
0	0	0

[비트 OR 연산]

피연산자2 \\	1	0
피연산자1		
1	1	1
0	1	0

[비트 XOR 연산]

피연산자2 \\	1	0
피연산자1		
1	1	1
0	1	0

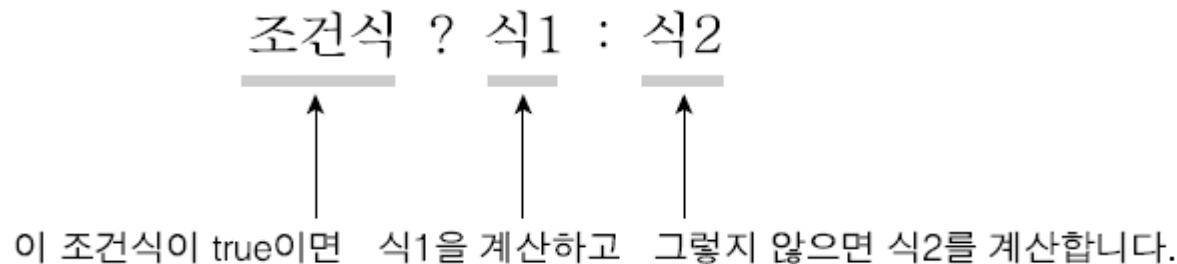
[비트 NOT 연산]

피연산자	결과
1	0
0	1

# 연산자

## ◆ 조건 연산자

-조건 연산자 : 조건식의 결과에 따라 두 피연산자 중 하나를 취하는 연산자



-사용 예

```
a < b ? a + 1 : b * 2
```

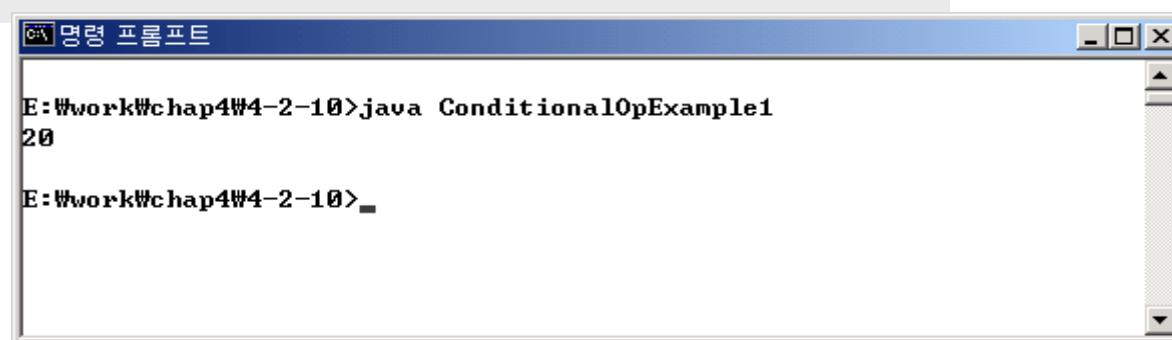
# 연산자

## ◆ 조건 연산자

-조건 연산자는 다음과 같은 대입문 형태로 많이 사용됩니다.

max = a < b ? a : b; // a < b 이면 a 값, 그렇지 않으면 b 값이 max에 대입됨

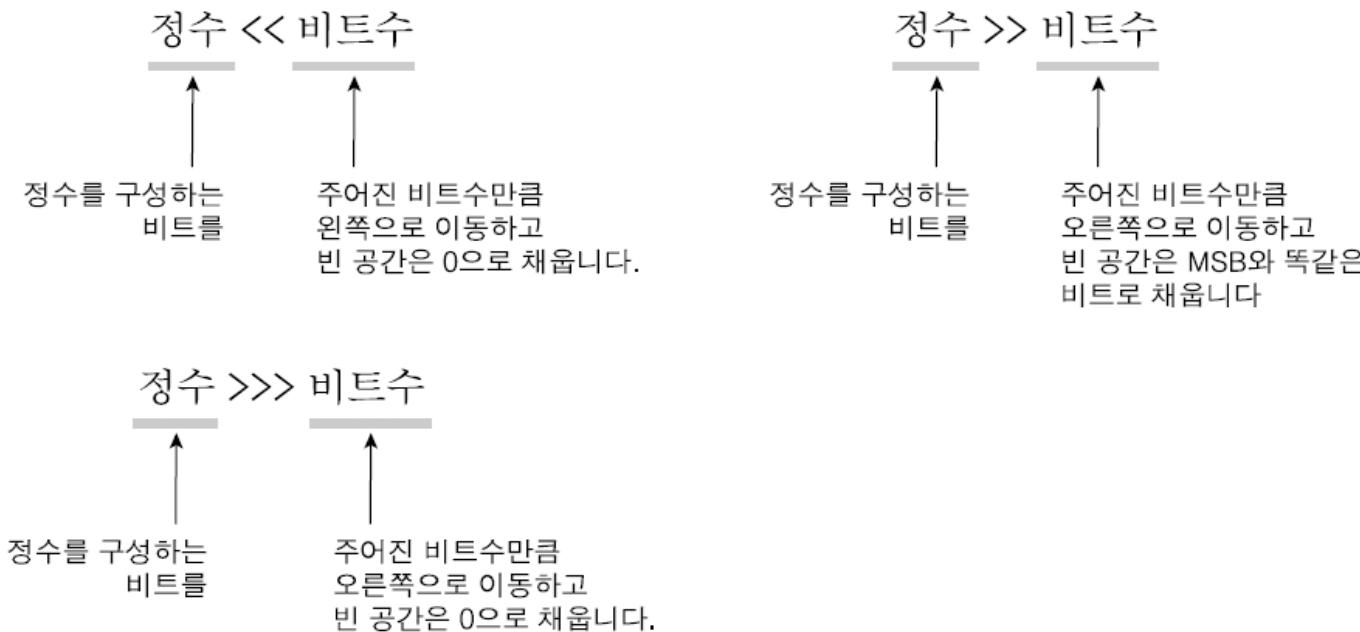
```
1  class ConditionalOpExample1 {  
2      public static void main(String args[]) {  
3          int a = 20, b = 30, max;  
4          max = a < b ? a : b;  
5          System.out.println(max);  
6      }  
7  }
```



# 연산자

## ◆ 쉬프트 연산자

- 쉬프트 연산자 : 데이터 구성 비트를 오른쪽/왼쪽으로 밀어서 이동시키는 연산자

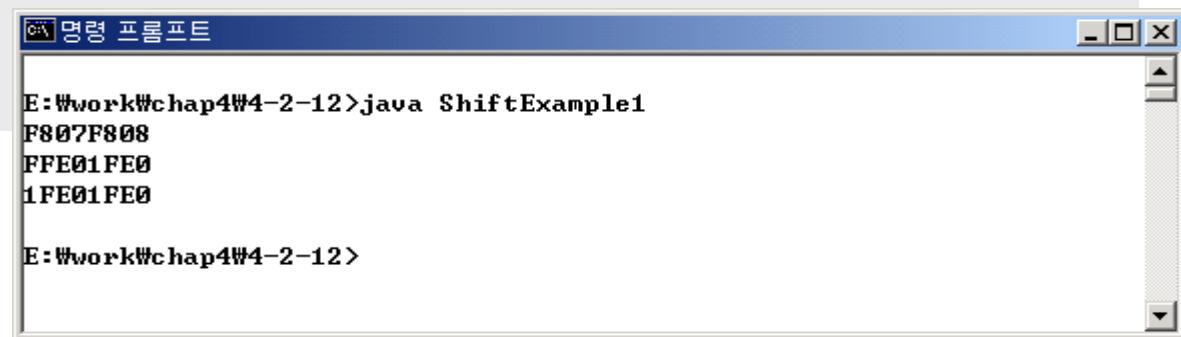


# 연산자

## ◆ 쉬프트 연산자

- 쉬프트 연산자의 사용 예

```
1  class ShiftExample1 {  
2      public static void main(String args[]) {  
3          int num = 0xFF00FF01;           // 11111111 00000000 11111111 00000001  
4          int result1 = num << 3;       // 11111000 00000111 11111000 00001000  
5          int result2 = num >> 3;       // 11111111 11100000 00011111 11100000  
6          int result3 = num >>> 3;      // 00011111 11100000 00011111 11100000  
7          System.out.printf("%08X %n", result1);  
8          System.out.printf("%08X %n", result2);  
9          System.out.printf("%08X %n", result3);  
10     }  
11 }
```

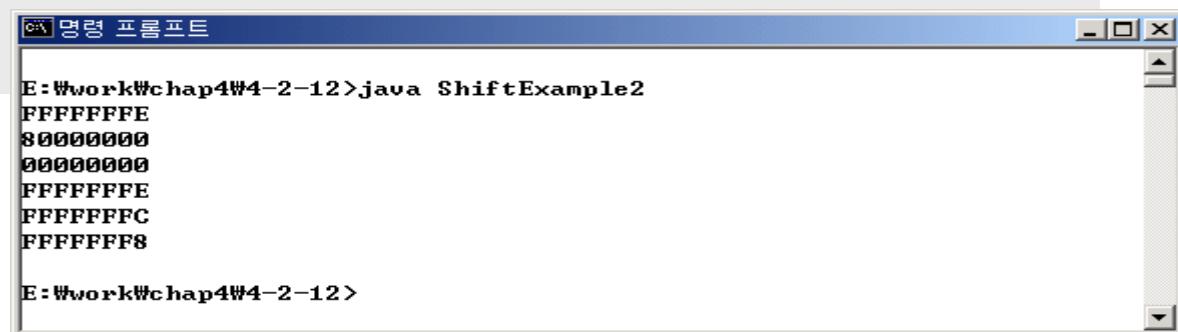


# 연산자

## ◆ 쉬프트 연산자

- 쉬프트 연산자의 사용 예

```
1  class ShiftExample2 {  
2      public static void main(String args[]) {  
3          int num1 = 0xFFFFFFF;    // 11111111 11111111 11111111 11111110  
4          int num2 = num1 << 30;  
5          int num3 = num1 << 31;  
6          int num4 = num1 << 32;  
7          int num5 = num1 << 33;  
8          int num6 = num1 << 34;  
9          System.out.printf("%08X %n", num1);  
10         System.out.printf("%08X %n", num2);  
11         System.out.printf("%08X %n", num3);  
12         System.out.printf("%08X %n", num4);  
13         System.out.printf("%08X %n", num5);  
14         System.out.printf("%08X %n", num6);  
15     }  
16 }
```



The screenshot shows a Windows Command Prompt window titled "명령 프롬프트". The command entered is "E:\work\chap4\#4-2-12>java ShiftExample2". The output displayed is:  
FFFFFFF  
00000000  
00000000  
FFFFFFF  
FFFFFFFC  
FFFFFFFS

# 연산자

---

## ◆ 쉬프트 연산자

- 쉬프트 연산자는 피연산자가 int보다 좁은 타입이면 int 타입으로 자동 변환을 수행합니다.

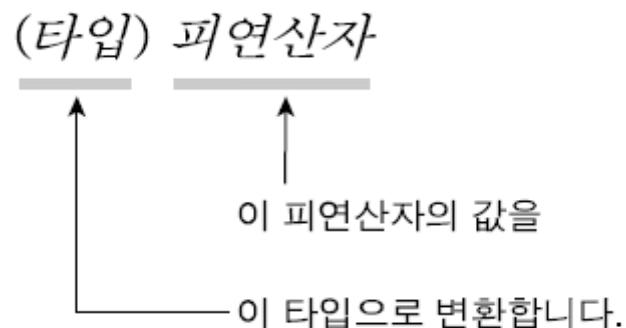
```
byte num1 = 1;
short num2 = 3;
char ch = 'A';
byte result1 = num1 << 3;           // 결과는 int 타입이므로 잘못된 대입문
short result2 = num2 >> 2L;          // 결과는 int 타입이므로 잘못된 대입문
char result3 = ch >>> 1;           // 결과는 int 타입이므로 잘못된 대입문
```

# 연산자

## ◆ 캐스트 연산자

-캐스트 연산자 : 타입의 변환을 수행하는 연산자

-사용 예



(long) 15    (int) ch    (byte) 12.5    (float) 0.000725

# 연산자

---

## ◆ 캐스트 연산자

- 캐스트 연산자는 피연산자와 똑같은 값을 갖는 새로운 타입의 값을 만들어 냅니다.

```
int num1 = 3;  
double num2 = (double) num1; // int 타입을 double 타입으로 변환
```

- 캐스트 연산자는 넓은 범위 수치 타입을 좁은 범위로 변환할 수도 있습니다.

num2에는 새로 만들어진 double 타입의  
3.0이라는 값이 대입됩니다.

- 캐스트 연산자가 모든 종류의 타입 변환을 다 수행할 수 있는 것은 아닙니다.

```
int num4 = (int) 12.9; // double 타입을 int 타입으로 변환
```

```
int num5 = (int) true; // 불가능한 캐스트 연산  
boolean truth = (boolean) 8; // 불가능한 캐스트 연산
```

# 조건문 -if

---

## ◆ if 조건문

-if 조건문의 기본 형식 (1)

if ( 조건식 )

명령문

```
if (num1 > num2)  
    System.out.println("num1 값이 더 큽니다.");
```

-if 조건문의 기본 형식 (2)

if ( 조건식 )

블록

```
if (num1 > num2) {  
    System.out.println("num1 값이 더 큽니다.");  
    System.out.println(num1);  
}
```

# 조건문 -if

## ◆ if 조건문

-if 문의 사용 예

```
1  class IfExample1 {  
2      public static void main(String args[]) {  
3          int num1 = 52;  
4          int num2 = 24;  
5          if (num1 > num2) {  
6              System.out.println("num1 값이 더 큽니다.");  
7              System.out.println("num1 = " + num1);  
8          }  
9          System.out.println("Done.");  
10     }  
11 }
```



# 조건문 -if

---

## ◆ if 조건문

-if-else 조건문의 기본 형식

- if(조건식)

    명령문1 // 조건이 true일때

    else

    명령문2 // 조건이 false일때

## ◆ if ..else if ..else

- if-else if... else 조건문의 기본 형식

- if(조건식1)

    명령문1 // 조건식1이 true일때

    else if(조건식2)

        명령문2 // 조건식2이 true일때

    else if(조건식3)

        명령문3 //조건식3이 true일때

    else

        명령문4 //나머지 경우

# 조건문 -if

---

## ◆ if 조건문

- dangling else : 어느 if 키워드와 짹을 이루는지 모호한 else 키워드
- 자바의 dangling else 규칙
- "dangling else는 가장 가까이 있는 if 키워드와 짹을 이룬다."

```
if (num1 > num2)
    if (num1 > num3)
        System.out.println("num1 = " + num1);
else
    System.out.println("num2 = " + num2);
```

# 조건문-switch

## ◆ switch 조건문

-switch 조건문의 전형적인 형식

```
switch (식) {  
    case 값1:  
        명령문들  
        break;  
    case 값2:  
        명령문들  
        break;  
    case 값3:  
        명령문들  
        break;  
    default :  
        명령문들  
        break;  
}
```

정수나 char 타입의 값을 산출할 수 있는 식

1회 이상 여러 번 반복 가능한 부분

생략 가능한 부분

# 반복문-while

---

## ◆ while 반복문

-while 문의 기본 형식

```
while ( 조건식 )  
    실행부분
```

true 또는 false 값을 산출할 수 있는 식  
조건식이 true일 동안 반복 실행되는 부분

# 반복문

---

## ◆ do-while 반복문

-do-while 문의 기본 형식

do

**실행부분** ————— 조건식이 true일 동안 반복 실행되는 부분

    while ( 조건식 );

                        ————— true 또는 false 값을 산출할 수 있는 식

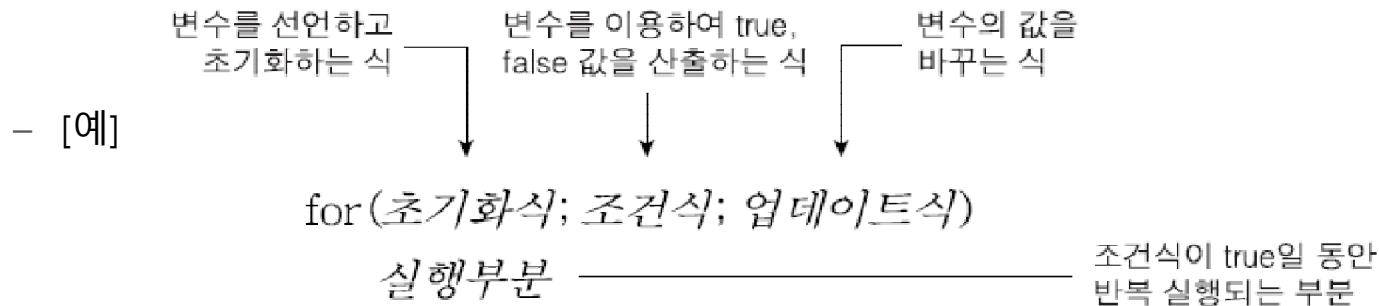
-while 문과의 차이점

- ✓ 조건식을 검사하기 전에 무조건 실행 부분을 한 번 실행
- ✓ 마지막에 세미콜론(;)을 반드시 써야 함

## 반복문-for

## ◆ for 반복문

## -for 문의 기본 형식



```
for (int cnt = 0; cnt < 10; cnt++)
    System.out.println(cnt);
```

# 반복문-for

---

## ◆ for 반복문

-향상된 for 문의 형식

for(변수타입 변수이름 : 배열이름)

    실행부분 —————— 반복 실행되는 부분

- ✓ 변수 타입 : 배열 항목과 동일한 타입
- ✓ 변수 이름 : 프로그래머가 나름대로 정할 수 있음

-[예]

```
for (int num : arr)  
    System.out.println(num);
```

# break

---

## ◆ break 문

- while, do, for 문 안에서 사용되면 반복문을 빠져나가는 기능
- switch 문 안에서 사용되면 switch 문을 빠져나가는 기능
- break 문의 기본 형식

break;

## ◆ 중첩된 반복문과 break 문

- 중첩된 반복문을 한꺼번에 빠져나가는 방법
  - ✓ 1) 반복문에 라벨을 붙인다
  - ✓ 2) break 문에 라벨을 지정한다

loop:

```
for (int cnt = 0; cnt < 100; cnt++) {  
    System.out.println(cnt);  
    if (cnt > 10)  
        break loop;  
}
```

for 문에 붙여진 라벨

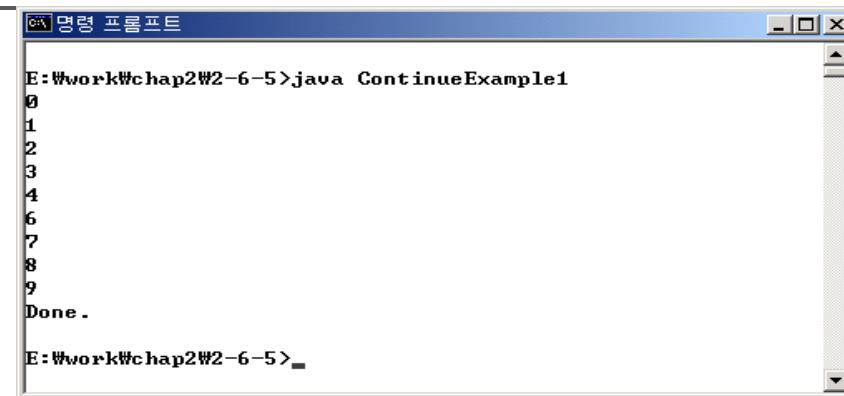
라벨을 지정한 break 문

# continue

## ◆ continue 문

- 반복문 안에서만 사용 가능
- 반복문의 다음 번 반복을 계속하는 기능
- continue 문의 기본 형식

continue;



```
1 class ContinueExample1 {
2     public static void main(String args[]) {
3         for (int cnt = 0; cnt < 10; cnt++) {
4             if (cnt == 5)
5                 continue;
6             System.out.println(cnt);
7         }
8         System.out.println("Done.");
9     }
10 }
```

cnt가 5이면 for 문의 다음번  
반복 과정을 계속합니다.

# continue

## ◆ continue 문

-continue 문의 잘못된 사용 예

```
1  class ContinueExample2 {  
2      public static void main(String args[]) {  
3          int cnt = 0;  
4          while (cnt < 10) {      <-----  
5              if (cnt == 5)           ----->  
6                  continue;  
7              System.out.println(cnt);  
8              cnt++;  
9          }  
10         System.out.println("Done.");  
11     }  
12 }
```

cnt가 5이면 while 문의 다음번 반복 과정을 계속합니다.

```
E:\work\chap2\2-6-5>java ContinueExample2  
0  
1  
2  
3  
4
```

# 배열

## ◆ 배열의 생성 (1차원 배열)

- 배열은 선언뿐만 아니라 생성을 해야만 사용할 수 있음
- 배열 생성식의 형식

- [예]      new 타입 [크기];

↑                 ↑  
배열 항목의 타입    배열 항목의 수

```
arr = new int[10];
num = new float[5];
strArr = new String[3];
```

# 배열

## ◆ 배열의 선언 (1차원 배열)

- 배열 변수 선언문의 형식 (1)

- [예]

타입 식별자[];

배열 항목의 타입 배열 변수의 이름

- 배열 변수 선언문의 형식 (2)

[예]

타입[] 식별자;

배열 항목의 타입 배열 변수의 이름

```
int      arr[];
float    num[];
String   strArr[];
```

```
int[]      arr;
float[]    num;
String[]   strArr;
```

백  
열

## ◆ 배열의 이용 (1차원 배열)

- 배열 이름과 인덱스를 이용하면 배열 항목을 단일 변수처럼 사용 가능
  - 배열 항목을 가리키는 식

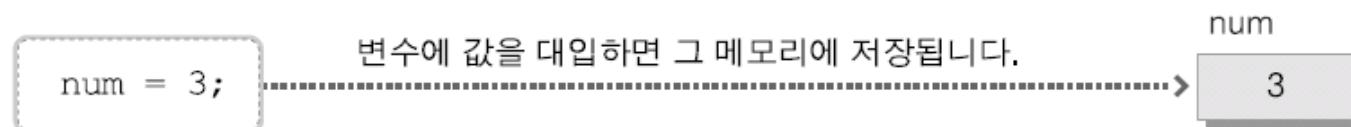
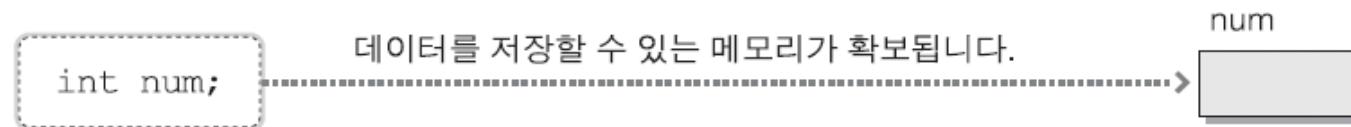
- [예] 배열이름[인덱스];

```
arr[0] = 12;  
num[3] = num[1] + num[2];  
System.out.println(strArr[2]);
```

# 배열

## ◆ 배열을 생성해야 하는 이유

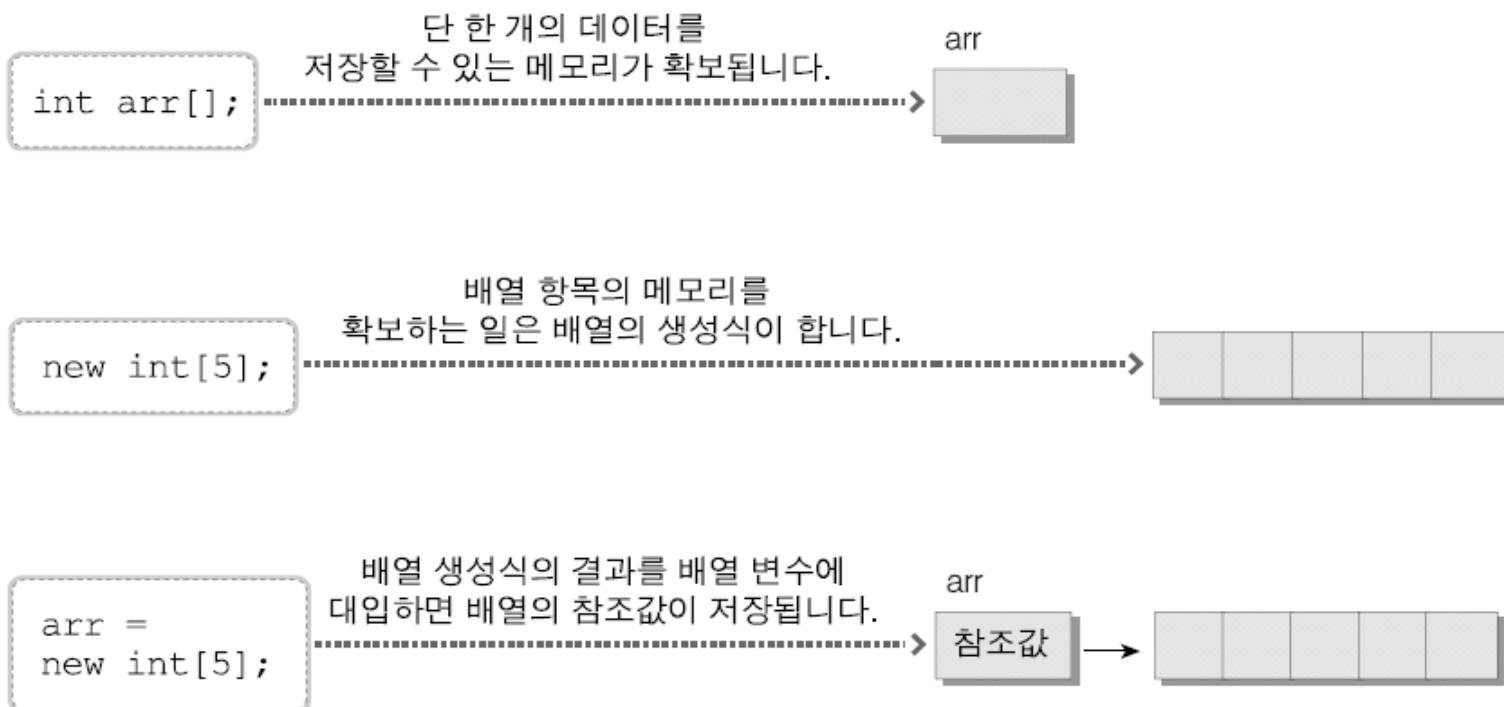
- 단일 변수의 메모리



# 배열

## ◆ 배열을 생성해야 하는 이유

- 배열의 메모리



# 배열

## ◆ 배열의 선언 (2차원 배열)

- 배열 변수 선언문의 형식 (1)

- [예] 타입 식별자[  ] [  ];  
    ↑                       ↑  
    배열 항목의 타입    배열 변수의 이름

- 배열 변수 선언문의 형식 (2)

[예] 타입[  ] [  ] 식별자;  
    ↑                       ↑  
    배열 항목의 타입    배열 변수의 이름

```
int      arr[][];
float    num[][];
String   strArr[][];
```

```
int[][]
float[][]
String[][] arr;
num;
strArr;
```

백  
열

## ◆ 배열의 생성 (2차원 배열)

### -배열 생성식의 형식

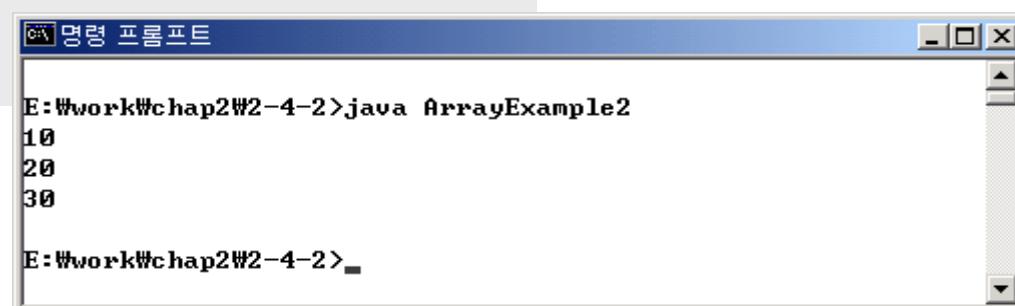
```
arr = new int[10][10];
num = new float[5][2];
strArr = new String[3][15];
```

# 배열

## ◆ 배열의 선언, 생성, 이용

-2차원 배열의 사용 예

```
1  class ArrayExample2 {  
2      public static void main(String args[]) {  
3          int table[][] = new int[3][4];  
4          table[0][0] = 10;  
5          table[1][1] = 20;  
6          table[2][3] = table[0][0] + table[1][1];  
7          System.out.println(table[0][0]);  
8          System.out.println(table[1][1]);  
9          System.out.println(table[2][3]);  
10     }  
11 }
```



# 열거형 타입

---

- ◆ 몇 개의 값으로만 한정적으로 이루어져 가지고 있는 타입
- ◆ **public enum Week{ Sun ,Mon, Tue, Wed, Thur, Fri, Sat}**

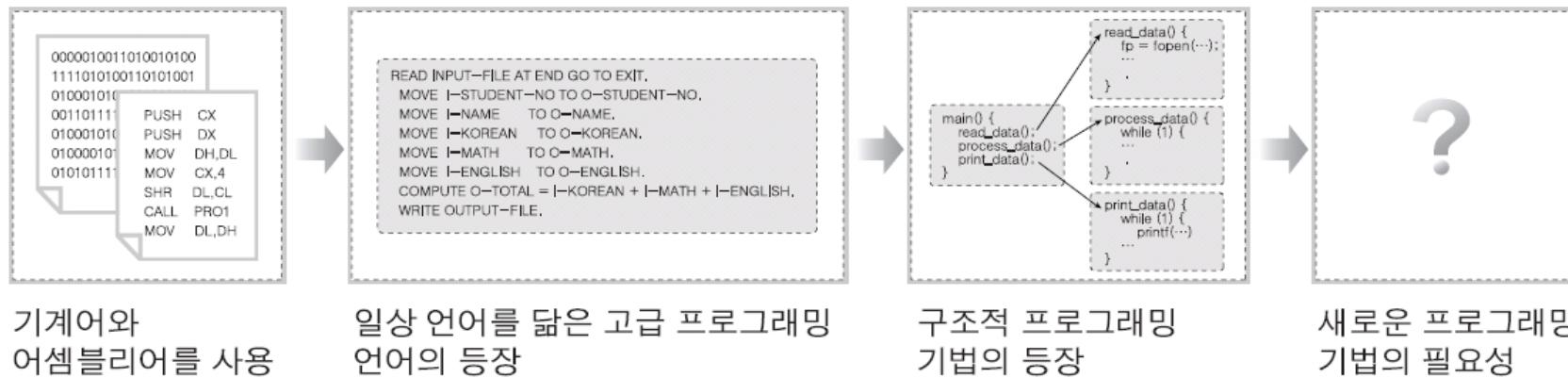
```
1  enum Week
2  {
3      Sun, mon, Tue, Wed, Thur, Fri, Sat
4  }
5  public class WeekTest {
6      public static void main(String[] args) {
7          Week monday=Week.mon;
8          System.out.println(monday);
9          System.out.println(Week.mon);
10     }
11 }
```

**CLASS 기초**

# 객체와 클래스

## ◆ 프로그래밍 기술의 변천

- 프로그래밍 기술의 변천 과정

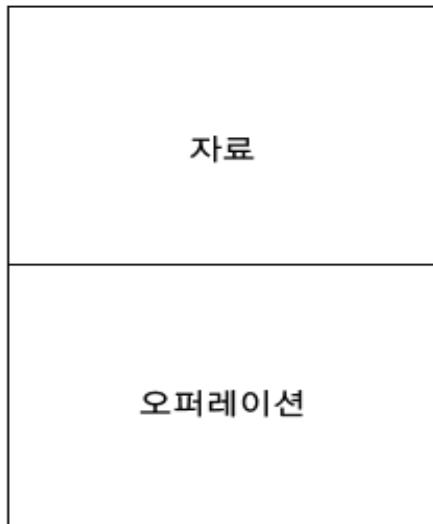


# 객체지향 기본 개념 - 객체

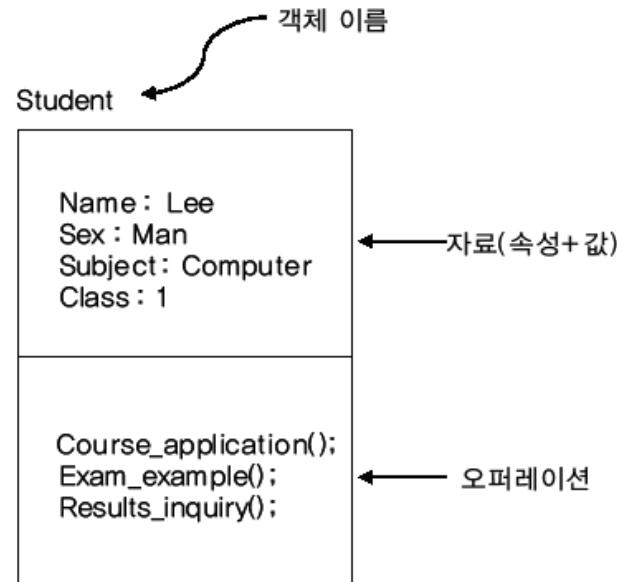
## ◆ 객체

- 객체란 우리가 살아가는 세계에 존재하거나 생각할 수 있는 것을 말함
- 현실세계에 존재하는 개념들 중 소프트웨어 개발 대상이 되는 것들은 모두 객체라고 할 수 있다.
- 같은 종류 객체들의 공통된 데이터 구조와 기능을 정의하는 클래스

[그림 1-2] 객체의 구조



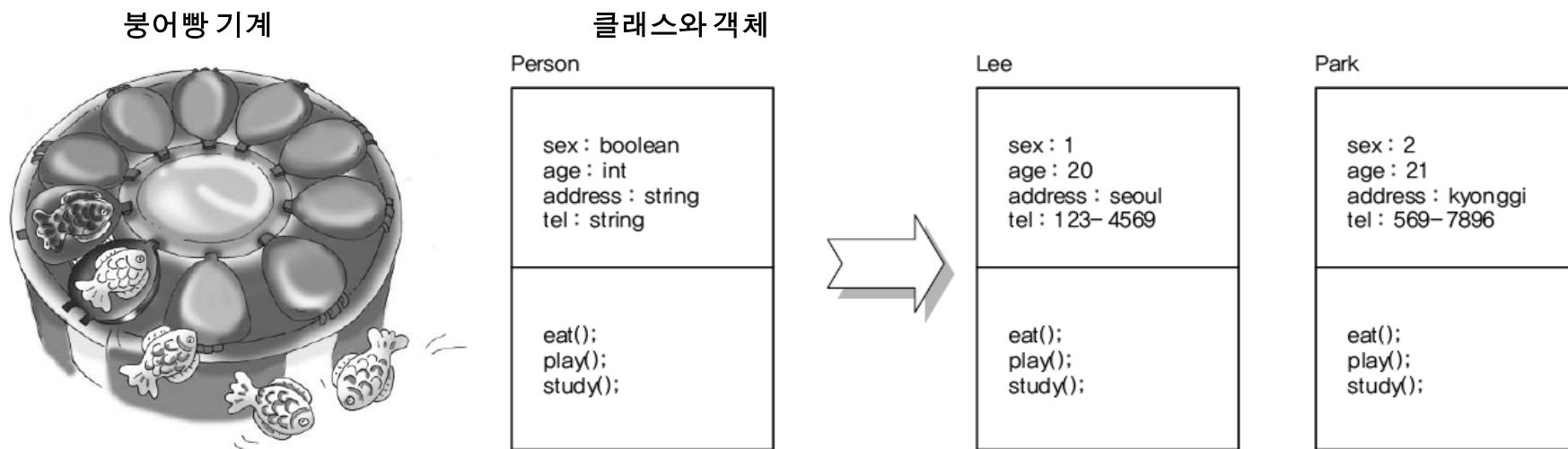
[그림 1-3] 학생에 대한 객체의 예



# 객체지향 기본 개념 - 클래스

## ◆ 클래스

- 클래스는 객체를 생성할 수 있는 구조와 정보를 가지는 템플릿(template)이라고 정의할 수 있다.
- 봉어빵 기계를 클래스라고 하면 이 기계에서 나오는 봉어빵은 객체
- 클래스는 개념적인 의미이며, 객체는 구체적인 의미
- 클래스에서 생성된 객체들은 같은 속성과 같은 오퍼레이션에 대한 정의를 갖음



---

## ◆ 클래스

- 객체 지향에서 가장 중요한 개념
- 새로운 데이터 타입을 만드는 **데이터 타입 생성기**
  - 많은 변수들을 모아서 새로운 데이터 타입을 만들어 냄
- 사용자 정의 데이터 타입의 집합체
- 클래스는 데이터 형을 의미
- 구조체에서 발전된 형태
- 특징
  - **new** 연산자를 사용하는 순간 힙메모리에 생성
  - 메서드 포함 가능
  - 접근 지정의 개념 적용(public, private, protected, default)
  - 상속의 개념 포함됨

---

## ◆ 인스턴스

- 인스턴스는 그 데이터 형의 실체를 의미
- 실체라는 것은 메모리에 생성됐다는 것을 의미.
- Object is an instance of a class

### 메소드

- 연산과 구분해서 사용될 때에는 연산의 구현을 뜻함
- 동작에 관한 내용

## ◆ 봉어빵과 봉어빵 틀

- 봉어빵 틀
  - 봉어빵 틀은 하나 밖에 없다.
  - 봉어빵 틀을 봉어빵에 대한 명세 또는 정의라 볼 수 있다.
  - 봉어빵 틀은 클래스 이다.
- 봉어빵
  - 틀에 봉어빵 재료를 넣어 구우면 봉어빵을 무제한적으로(재료가 남아있는 한) 만들어 낼 수 있다.
  - 봉어빵이 인스턴스 이다.

---

붕어빵 틀(클래스)



객체지향 프로그램이란  
데이터형(클래스)의 집합체

붕어빵의 실체(인스턴스)



한 마리가  
하나하나의  
인스턴스(=객체)

인스턴스는 필요에 따라  
얼마든지 만들어 낼 수 있다.  
이런 인스턴스들을 통틀어서 객체라  
부른다.

# 객체지향 기본 개념 - 객체 생성하기

- ◆ 봉어빵 틀에 재료를 주입하면 봉어빵이 나옴.



- ◆ 클래스에 메모리를 주입하면 객체가 생성.

```
class Person {  
    int age;  
    long height;  
    float weight;  
}
```

사용자 정의  
데이터 타입

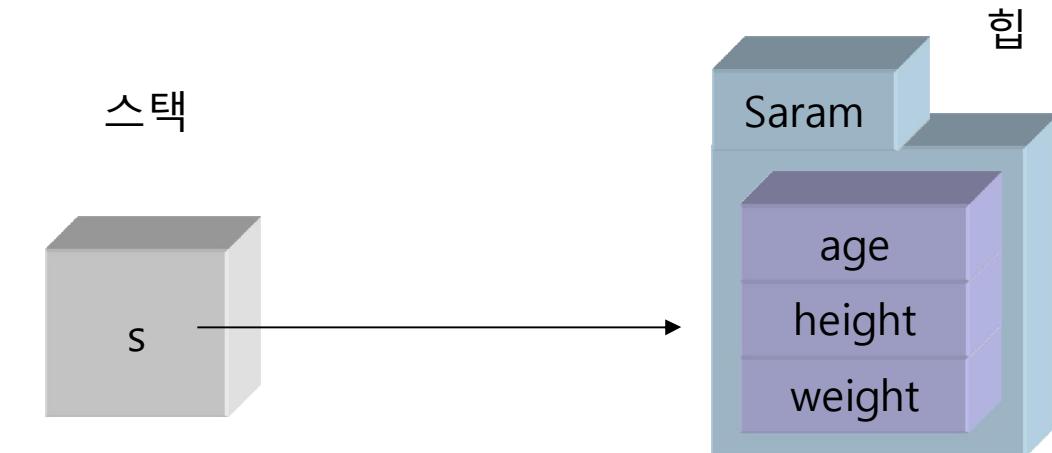
메모리할당

Person brother = new Person();

객체

생성자메서드

```
class Saram{  
    int age;  
    int height;  
    int weight;  
}  
  
Saram s;  
s= new Saram();
```



◆ **Saram는 int형 값 3개를 내장할 수 있는 형식**

◆ **Saram s**

- 객체를 “가리키기 위한 값”을 넣을 수 있는 “상자”
- 객체 참조 변수, 참조값, 인스턴스 변수

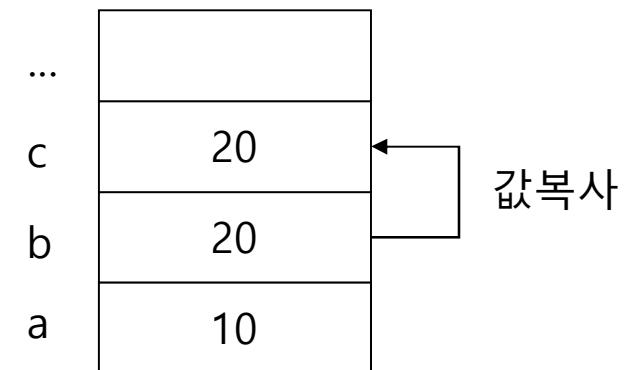
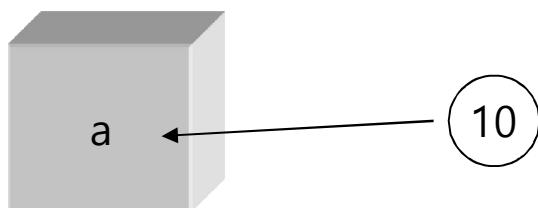
◆ **`s = new Saram()`**

- `new`라는 키워드를 통해 VM이 `Saram` 객체를 Heap영역에 새로 생성하고 그 참조값(주소)를 리턴 한다.

## ◆ 기본 데이터 타입의 변수

- 변수 안에 데이터를 직접 복사(copy of value)

```
int a = 10;  
int b = 20;  
int c = b;
```



## ◆ 클래스 타입의 변수

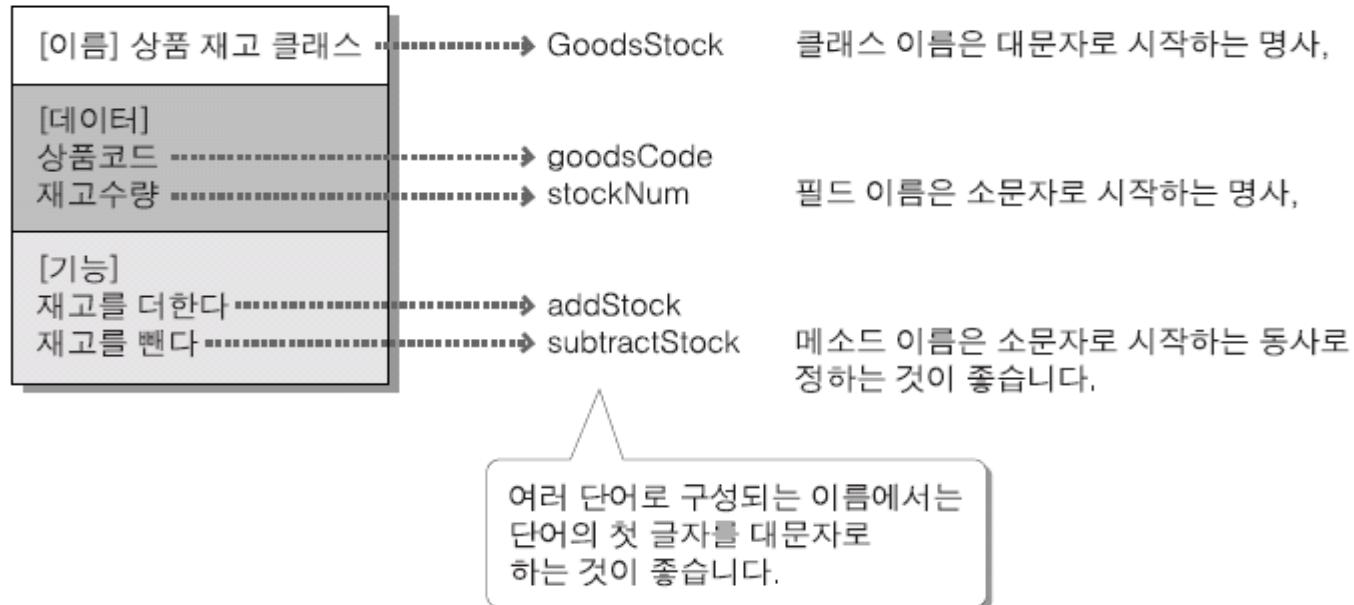
- 객체변수는 메모리를 복사하지 않음
- 할당을 하므로 참조를 함
- 모든 객체변수(인스턴스)는 참조 값

# 객체지향 기본 개념 - 클래스(class) 선언

## ◆ 클래스를 선언하는 방법

### - 클래스의 선언

- ✓ 제일 먼저 해야할 일은 정의된 클래스와 필드, 메소드 이름을 정하는 것



---

## ◆ 필드

-필드의 선언 예

```
1  class Circle {  
2      double radius; // 필드  
3      Circle(double radius) { // 생성자  
4          this.radius = radius;  
5      }  
6      double getArea() { // 메소드  
7          double area;  
8          area = radius * radius * 3.14;  
9          return area;  
10     }  
11 }
```

## ◆ 필드

-private 필드를 갖는 클래스의 예

```
1  class Circle {  
2      private double radius;  
3      Circle(double radius) {  
4          this.radius = radius; <  
5      }  
6      double getArea() {  
7          double area;<  
8          area = radius * radius * 3.14;  
9          return area;  
10     }  
11 }
```

클래스 외부 접근을 금지시키는 키워드

하지만 클래스 내부에서의 접근은  
가능합니다.

---

## ◆ 필드

-여러 가지 형태의 필드 선언문을 포함하는 클래스

```
1  class SomeClass1 {  
2      int a, b;          // 여러 필드들을 한꺼번에 선언하는 선언문  
3      double c = 1.2;    // 초기값을 지정하는 선언문  
4      double d = c * 2;  // 다른 필드의 값을 초기값으로 사용하는 선언문  
5  }
```

---

## ◆ 필드

-final 필드를 포함하는 클래스 (1) – 올바른 예

```
1  class Square {  
2      final int sideLength = 10;      // 올바른 선언문  
3  }
```

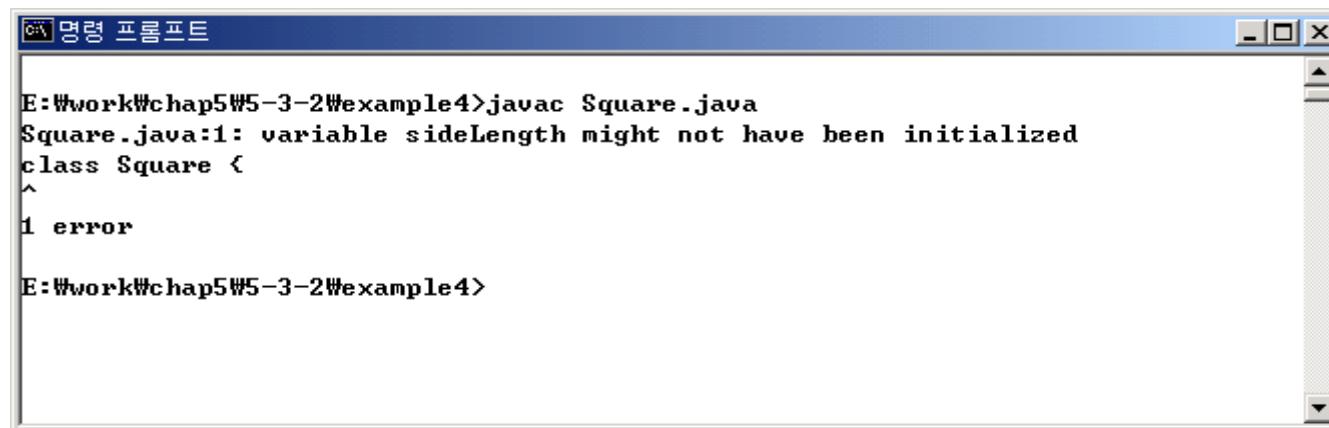
-final 필드를 포함하는 클래스 (2) – 올바른 예

```
1  class Square {  
2      final int sideLength;          // 선언문에서는 초기화 하지 않았지만  
3      Square(int sideLength) {  
4          this.sideLength = sideLength; // 생성자 안에서 초기화했음  
5      }  
6  }
```

## ◆ 필드

-final 필드를 포함하는 클래스 (3) – 잘못된 예

```
1  class Square {  
2      final int sideLength;          // 잘못된 선언문  
3  }
```



The screenshot shows a Windows Command Prompt window titled "명령 프롬프트". The command entered is "javac Square.java". The output shows a single error: "Square.java:1: variable sideLength might not have been initialized". The cursor is positioned at the start of the "sideLength" identifier in the code above. The window has standard Windows-style scroll bars on the right.

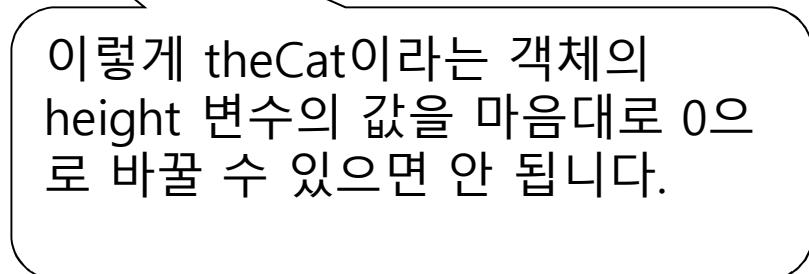
```
E:\work\chap5\5-3-2\example4>javac Square.java  
Square.java:1: variable sideLength might not have been initialized  
class Square {  
^  
1 error  
E:\work\chap5\5-3-2\example4>
```

# 객체지향 기본 개념 - 캡슐화(encapsulation)

---

## ◆ 데이터 노출!!!

- 지금까지 우리가 만든 프로그램에는 데이터가 완전히 노출되어 있다는 심각한 문제가 있습니다. 즉 아무나 인스턴스 변수를 마음대로 보고 건드릴 수 있었습니다.
- `theCat.height = 27;`
- `theCat.height = 0;`



이렇게 `theCat`이라는 객체의 `height` 변수의 값을 마음대로 0으로 바꿀 수 있으면 안 됩니다.

---

## ◆ 이런 문제를 어떻게 해결할 수 있을까요?

- 세터 메소드를 쓰면 됩니다.

~~theCat height = 0;~~

```
public void setHeight(int ht) {  
    if (height > 9) {  
        height = ht;  
    }  
}
```

세터 메소드를 사용하면  
이렇게 인스턴스 변수의  
값이 합당한지 검사할  
수도 있습니다.

---

◆ 그렇다면 데이터를 직접 건드릴 수 없도록 하려면 어떻게 해야 할까요?

◆ 액세스 변경자(access modifier)

- 인스턴스 변수: private으로 선언
- 게터 및 세터 메소드: public으로 선언

# 객체지향 기본 개념 - 게터(getter)와 세터(setter)

---

## ◆ 게터(getter)

- 인스턴스 변수의 값을 알아내기 위한 메소드
- 일반적으로 인스턴스 변수의 값을 리턴함
- getBrand(), getNumOfPickups()...

## ◆ 세터(setter)

- 인스턴스 변수의 값을 설정하기 위한 메소드
- 전달된 값을 확인하고 인스턴스 변수의 값을 설정함
- setBrand(), setNumOfPickups()...

---

```
class ElectricGuitar {  
    private String brand;  
    private int num;  
    private boolean yseOrNo;  
  
    public String getBrand() {  
        return brand;  
    }  
  
    public void setBrand(String brand) {  
        this. brand = brand;  
    }  
  
    public int getNumOfPickups() {  
        return num;  
    }  
  
    public void setNumOfPickups(int num) {  
        this.num= num;  
    }  
  
    public boolean getRockStarUsesIt() {  
        return rockStarUsesIt;  
    }  
  
    public void setRockStarUsesIt  
        (boolean yesOrNo) {  
        this. yseOrNo = yseOrNo;  
    }  
}
```

# 객체지향 기본 개념 - 생성자(constructor)

---

## ◆ 생성자

- 객체가 생성될 때 호출되어 실행
- 리턴형이 없음
- 생성자 이름은 클래스의 이름과 동일
- new 연산자가 호출된 직후에 호출
  - new 연산자가 메모리를 생성하면 멤버변수들이 메모리를 할당 받음
  - 따라서 변수들에 대한 초기화 작업이 가능해짐
- 기본생성자( default constructor)
  - 클래스에 생성자가 없을 경우 컴파일러가 자동으로 생성해줌
  - 컴파일러가 생성해주는 기본 생성자는 매개변수가 없고 블록이 비어 있음
- 생성자 사용이유
  - 할당 받은 멤버변수를 초기화 할 때
  - 객체가 생성되기 전의 미리 해야 할 작업이 있을 때

## ◆ 생성자

-생성자를 추가한 Student클래스

```
1  class Student {  
2      private String stucode;  
3      private String name;  
4      private int age;  
5      Student(String code, String n, int a) {  
6          stucode = code;  
7          name = n;  
8          age=a;  
9      }  
10 }
```

}      생성자(constructor)

객체 생성 - 1

Student obj;

obj = new Student( "a1" , " hong" , 10);

객체 생성 - 2

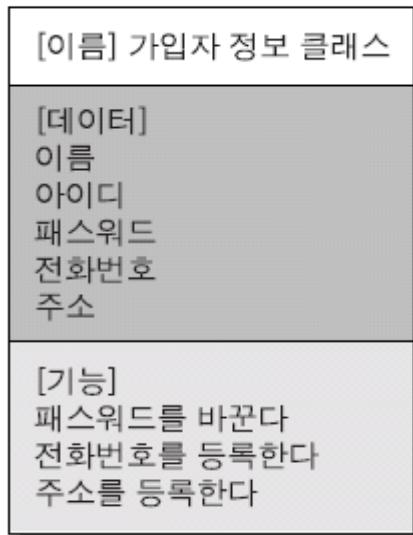
Student obj= new Student( "a1" , "hong" , 10);

# 클래스 선언의 기초 문법 – 생성자 오버로딩

---

## ◆ 둘 이상의 생성자

-둘 이상의 생성자가 필요한 클래스



- ✓ 어떤 가입자는 이름, 아이디, 패스워드만 입력하고
- ✓ 어떤 가입자는 이름, 아이디, 패스워드, 전화번호, 주소까지 입력한다면?

## ◆ 둘 이상의 생성자

### - 두 개의 생성자를 갖는 클래스 – 가입자 정보 클래스

```
1  class SubscriberInfo {  
2      String name, id, password;  
3      String phoneNo, address;  
4      SubscriberInfo(String name, String id, String password) {  
5          this.name = name;  
6          this.id = id;  
7          this.password = password;  
8      }  
9      SubscriberInfo(String name, String id, String password, String phoneNo, String address) {  
10         this.name = name;  
11         this.id = id;  
12         this.password = password;  
13         this.phoneNo = phoneNo;  
14         this.address = address;  
15     }  
16     void changePassword(String password) {  
17         this.password = password;  
18     }  
19     void setPhoneNo(String phoneNo) {  
20         this.phoneNo = phoneNo;  
21     }  
22     void setAddress(String address) {  
23         this.address = address;  
24     }  
25 }
```

3개의 파라미터를  
받는 생성자

5개의 파라미터를  
받는 생성자

파라미터 변수와 필드의 이름이 같을 때는  
필드 이름 앞에 this.을 붙여서 구분해야 합니다.

## ◆ 둘 이상의 생성자

-두 개의 생성자를 갖는 클래스의 객체를 생성하는 프로그램

```
1  class ConstrExample2 {  
2      public static void main(String args[]) {  
3          SubscriberInfo obj1, obj2;  
4          obj1 = new SubscriberInfo("연홍부", "poorman", "zebi");  
5          obj2 = new SubscriberInfo("연놀부", "richman", "money",  
6                                      "02-000-0000", "타워팰리스");  
7          printSubscriberInfo(obj1);  
8          printSubscriberInfo(obj2);  
9      }  
10     static void printSubscriberInfo(SubscriberInfo obj) {  
11         System.out.println("이름:" + obj.name);  
12         System.out.println("아이디:" + obj.id);  
13         System.out.println("패스워드:" + obj.password);  
14         System.out.println("전화번호:" + obj.phoneNo);  
15         System.out.println("주소:" + obj.address);  
16     }  
17 }
```

3개의 파라미터를 받는  
생성자를 호출

5개의 파라미터를 받는  
생성자를 호출

SubscriberInfo 객체의  
필드 값을 모두 출력하는  
메소드

## ◆ 둘 이상의 생성자

-필드의 디폴트 값

	데이터 타입	디폴트 값
수치 타입	byte	0
	short	
	int	
	long	
	char	
	float	
	double	
불리언 타입	boolean	false
레퍼런스 타입	그 밖의 모든 타입	null

## ◆ 둘 이상의 생성자

- 두 개의 생성자를 갖는 클래스 – 잘못된 예

```
1  class SubscriberInfo {  
2      String name, id, password;  
3      String phoneNo, address;  
4      SubscriberInfo(String name, String id, String password, String phoneNo) {  
5          this.name = name;  
6          this.id = id;  
7          this.phoneNo = phoneNo;  
8          this.password = password;  
9      }  
10     SubscriberInfo(String name, String id, String password, String address) {  
11         this.name = name;  
12         this.id = id;  
13         this.password = password;  
14         this.address = address;  
15     }  
16     void changePassword(String password) {  
17         this.password = password;  
18     }  
19     void setPhoneNo(String phoneNo) {  
20         this.phoneNo = phoneNo;  
21     }  
22     void setAddress(String address) {  
23         this.address = address;  
24     }  
25 }
```

파라미터 변수의 이름만  
다를 뿐 타입, 수, 순서가  
똑같습니다.

## ◆ 생성자의 호출

- 생성자 안에서 다른 생성자를 호출하는 방법  
✓ : this 키워드를 사용

## ◆ 생성자의 호출

-생성자에서 생성자를 호출하는 클래스의 예

```
1  class SubscriberInfo {  
2      String name, id, password;  
3      String phoneNo, address;  
4      SubscriberInfo() {  
5          }  
6      SubscriberInfo(String name, String id, String password) {  
7          this.name = name;  
8          this.id = id;                                     <-----  
9          this.password = password;  
10         }  
11         SubscriberInfo(String name, String id, String password, String phoneNo, String  
12 address) {  
13             this(name, id, password);-----  
14             this.phoneNo = phoneNo;  
15             this.address = address;  
16         }  
17         void changePassword(String password) {  
18             this.password = password;  
19         }  
20         void setPhoneNo(String phoneNo) {  
21             this.phoneNo = phoneNo;  
22         }  
23         void setAddress(String address) {  
24             this.address = address;  
25         }  
}
```

String 타입의 파라미터 3개를 받는  
생성을 호출합니다.

## ◆ 생성자의 호출

- 생성자 안에서 다른 생성자를 호출할 때 주의할 점
  - ✓ 생성자 호출문은 생성자 안에서 첫번째 명령문이어야 함

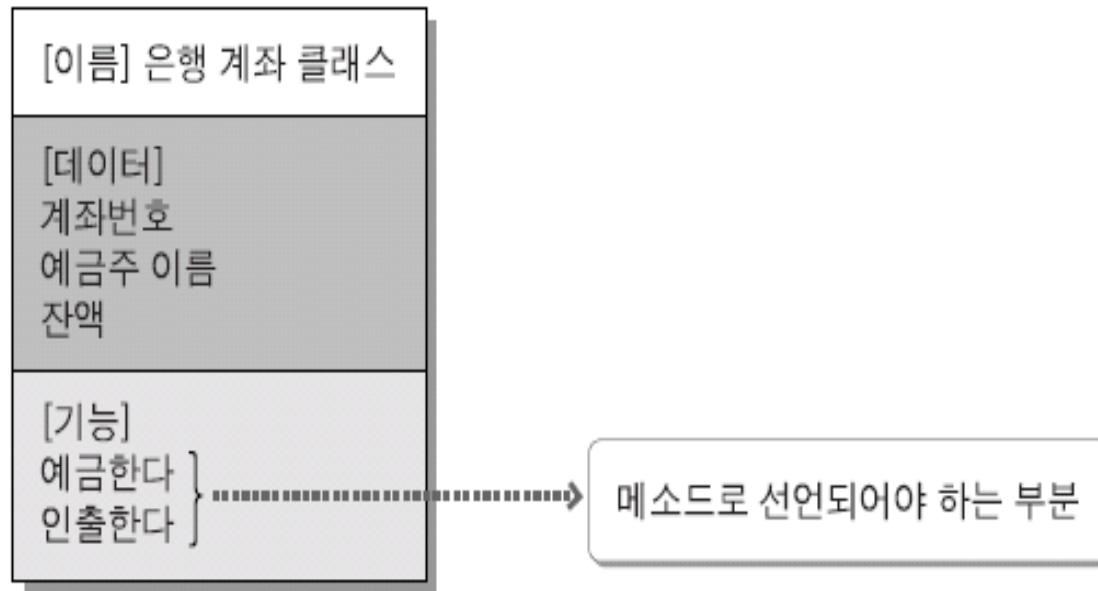
```
class SubscriberInfo {  
    . . .  
    SubscriberInfo(String name, String id, String password,  
                   String phoneNo, String address) {  
        this.phoneNo = phoneNo;  
        this.address = address;  
        this(name, id, password); }  
    . . .  
}
```

생성자 호출문이 이렇게 다른 명령문보다  
뒤에 오는 것은 잘못입니다.

# 클래스 선언의 기초 문법 – 메소드(method)

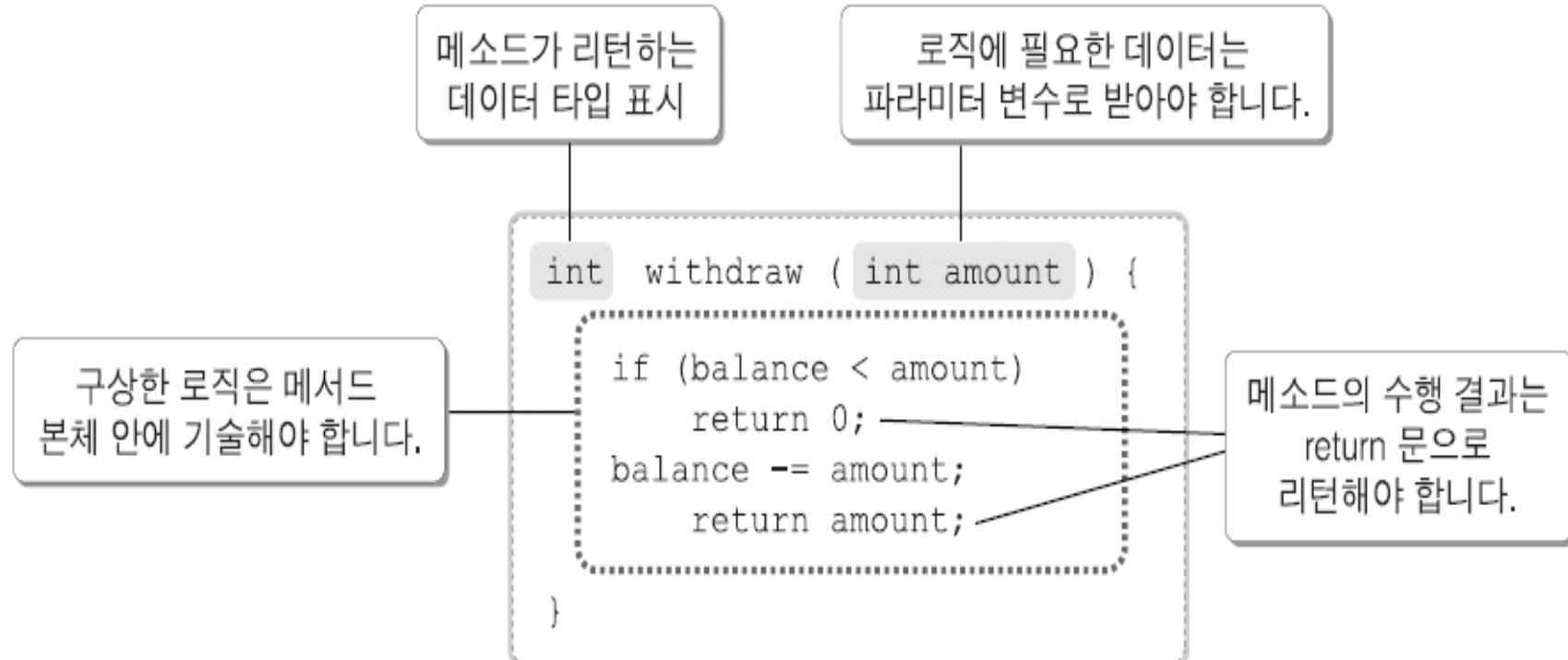
## ◆ 메소드

-예제로 사용할 클래스



## ◆ 메소드

-메소드를 선언하는 방법



# 클래스 선언의 기초 문법 - 구현

## ◆ 메소드

-클래스의 기능을 구현하는 메소드 선언

```
1  class Account {  
2      String accountNo;    // 계좌번호  
3      String ownerName;   // 예금주 이름  
4      int balance;        // 잔액  
5      Account(String accountNo, String ownerName, int balance) {    // 생성자  
6          this.accountNo = accountNo;  
7          this.ownerName = ownerName;  
8          this.balance = balance;  
9      }  
10     void deposit(int amount) {  
11         balance += amount;  
12     }  
13     int withdraw(int amount) {  
14         if (balance < amount)  
15             return 0;  
16         balance -= amount;  
17         return amount;  
18     }  
19 }  
20 }
```

리턴 값이 없음을 표시하는 자바 키워드

“예금한다” 기능을 구현하는 메소드 선언

“인출한다” 기능을 구현하는 메소드 선언

---

## ◆ 메소드

-특정 객체에 속하는 메소드의 호출 방법

obj1.deposit(1000000);  
          ↑                       ↑  
  객체 변수 이름                메소드 이름

## ◆ 클래스 내부에서의 메소드 호출

-정수 배열의 통계를 내는 클래스 – 평균 계산 기능 추가

```
1  class Numbers {  
2      int num[];  
3      Numbers(int num[]) {  
4          this.num = num;  
5      }  
6      int getTotal() { // 합계를 구하는 메소드  
7          int total = 0;  
8          for (int cnt = 0; cnt < num.length; cnt++)  
9              total += num[cnt];  
10         return total;  
11     }  
12     int getAverage() { // 평균을 구하는 메소드  
13         int total;  
14         total = getTotal(); ----- 같은 클래스 내의 메소드 호출  
15         int average = total / num.length;  
16         return average;  
17     }  
18 }
```

# 클래스 선언의 기초 문법 -메서드 오버로딩

---

## ◆ 메소드 오버로딩

-자바에서의 메소드 호출 조건

- 메소드와 메소드 호출문의 파라미터 수, 타입, 순서가 맞아야 함

→ 이런 특성을 이용하면 한 클래스 안에 똑같은 이름의 메소드 여러 개를 선언할 수 있음



메소드 오버로딩 (method overloading)

## ◆ **overloading**을 사용 할 때의 규칙

- 클래스내에 이름이 같은 메서드를 여러 개 선언
- 매개변수의 개수 다를 것
- 매개변수의 형이 다를 것
- 매개변수의 선언 순서가 다를 것
- 위의 조건 중 하나만 달라도 overloading은 성립
- 메서드의 리턴 형은 overloading을 구분할 때 사용하지 않음

## ◆ 메소드 오버로딩

-오버로드된 메소드를 포함하는 클래스

```
1  class PhysicalInfo {  
2      String name;  
3      int age;  
4      float height, weight;  
5      PhysicalInfo(String name, int age, float height, float weight) {  
6          this.name = name;  
7          this.age = age;  
8          this.height = height;  
9          this.weight = weight;  
10     }  
11     void update(int age) {  
12         this.age = age;  
13     }  
14     void update(int age, float height) {  
15         this.age = age;  
16         this.height = height;  
17     }  
18     void update(int age, float height, float weight) {  
19         this.age = age;  
20         this.height = height;  
21         this.weight = weight;  
22     }  
23 }
```

메소드의 이름은 같지만 파라미터 변수의 수와 타입이 다릅니다.

메소드 이름과 파라미터 변수의 수, 타입, 순서를  
묶어서 **메소드 시그니처(method signature)**라고 부릅니다

예) update(int)  
update(int, float)  
update(int, float, float)

## ◆ 메소드 오버로딩

-오버로드된 메소드를 호출하는 프로그램

```
1  class MethodExample6 {  
2      public static void main(String args[]) {  
3          PhysicalInfo obj;  
4          obj = new PhysicalInfo("해리", 10, 132.0f, 35.0f);  
5          printPhysicalInfo(obj);  
6          obj.update(11, 145.0f, 45.0f);  
7          printPhysicalInfo(obj);  
8          obj.update(12, 157.0f);  
9          printPhysicalInfo(obj);  
10         obj.update(13);  
11         printPhysicalInfo(obj);  
12     }  
13     static void printPhysicalInfo(PhysicalInfo o)  
14     {  
15         System.out.println("이름:" + obj.name)  
16         System.out.println("나이:" + obj.age);  
17         System.out.println("키:" + obj.height)  
18         System.out.println("몸무게:" + obj.weight);  
19         System.out.println();  
20     }  
}
```

update(int, float, float) 메소드를 호출합니다.

```
E:\work\chap5\5-3\example5>java MethodExample6  
이름:해리  
나이:10  
키:132.0  
몸무게:35.0  
이름:해리  
나이:11  
키:145.0  
몸무게:45.0  
이름:해리  
나이:12  
키:157.0  
몸무게:45.0  
이름:해리  
나이:13  
키:157.0  
몸무게:45.0  
E:\work\chap5\5-3\example5>
```

# 클래스 선언의 기초 문법 - this

---

- ◆ **this**는 자기 자신을 포함시키는 객체를 말함

```
public class ThisTest1{
    private int i= 4;

    public void mth1(){
        int i= 10;
        System.out.println("local i=" + i);
        System.out.println("Member i=" + this.i);
    }

    public static void main(String args[]){
        ThisTest1 obj1= new ThisTest1();
        obj1.mth1();
        // mth1() 를 호출할 때 obj1 객체를 통하여 호출했다.
    }
}
```

---

## ◆ **this**

- 언젠가 생성될 자신 객체의 메모리의 주소

## ◆ **this.멤버필드**

- 클래스내의 자신의 멤버를 직접 이용
- 메서드의 매개변수와 클래스의 멤버필드가 동일할 때 이를 구분하기 위해 사용
- this 키워드를 사용하지 않아도 무방

## ◆ **this.멤버 메서드**

- this.멤버필드와 마찬가지로 자신의 멤버 메서드를 직접 이용

## ◆ **this()**

- 클래스 자신의 생성자 메서드를 호출할 때 사용
- 자신의 생성자 메서드를 재이용
  - 생성자 메서드에서 다른 생성자 메서드를 호출할 때  
생성자 메서드의 호출은 제일 윗부분에서 사용해야 함

# 클래스 선언의 기초 문법 – 정적(static) 구성 요소

---

static 변수- 클래스 변수, 정적변수

- static으로 정의된 멤버변수는 전역(공유)의 뜻한다
- 모든 객체에서 공통으로 사용하는 메모리
- 메서드 내에는 static변수를 선언할 수 없음
- 클래스 로딩시 생성
- 일반적으로 public static final로 선언을 한다.
- 클래스이름.변수

static 메서드 - 클래스 메서드, 정적메서드

- 자동으로 final 메서드가 됨(overriding 불가능)
- 클래스 메서드에 지역변수, 클래스 변수를 선언 가능
- 인스턴스 변수는 선언 불가능( 생성시점이 객체 생성시)
- 클래스이름.메서드

static 초기화 블록

- 객체가 생성되기 전에 static영역의 메모리를 제어하기 위한 블록

---

## ◆ 정적 필드( 클래스 변수)

- 정적 필드(static field) : static 키워드가 붙은 필드
- 정적 필드가 있는 클래스

```
1  class Accumulator {  
2      int total = 0;  
3      static int grandTotal = 0;          // 정적 필드를 선언하는 선언문  
4      void accumulate(int amount) {  
5          total += amount;  
6          grandTotal += amount;          // 정적 필드를 사용하는 명령문  
7      }  
8  }
```

# 클래스의 정적 구성 요소

---

## ◆ 정적 필드

-Accumulator 클래스를 사용하는 프로그램

```
1  class StaticFieldExample1 {  
2      public static void main(String args[]) {  
3          Accumulator obj1 = new Accumulator();  
4          Accumulator obj2 = new Accumulator();  
5          obj1.accumulate(10);  
6          obj2.accumulate(20);  
7          System.out.println("obj1.total = " + obj1.total);  
8          System.out.println("obj1.grandTotal = " + obj1.grandTotal);  
9          System.out.println("obj2.total = " + obj2.total);  
10         System.out.println("obj2.grandTotal = " + obj2.grandTotal);  
11     }  
12 }
```

---

## ◆ 정적 필드

-정적 필드를 사용하는 또 다른 방법

```
System.out.println("총계 = " + Accumulator.grandTotal);
```

클래스 이름      정적 필드이름

## ◆ 상수 필드

- 상수 필드(또는 상수 변수) : static과 final 키워드가 모두 붙은 필드
- 상수 필드가 있는 클래스
  - 초기값을 바꿀 수 없으므로 상수로 사용하기에 적합
- static final로 선언된 필드는 직접 초기화하거나 static 초기화 블록을 사용한다.

```
1  class LimitedValue {  
2      final static int UPPER_LIMIT = 100000; ----- 상수 필드의 선언  
3      int value;  
4      void setValue(int value) {  
5          if (value < UPPER_LIMIT)  
6              this.value = value;  
7          else  
8              this.value = UPPER_LIMIT; ----- 상수 필드의 사용  
9      }  
10 }
```

## ◆ 상수 필드

-상수 필드를 사용하는 프로그램

```
1  class StaticFieldExample2 {  
2      public static void main(String args[]) {  
3          LimitedValue v = new LimitedValue();  
4          v.setValue(200000);  
5          System.out.println("v.value = " + v.value);  
6          System.out.println("상한값 = " + LimitedValue.UPPER_LIMIT);  
7      }  
8  }
```

The screenshot shows a Windows command prompt window titled "명령 프롬프트". The command line shows the following sequence of commands and their outputs:

```
E:\work\chap5\5-4-1\example2>javac LimitedValue.java  
E:\work\chap5\5-4-1\example2>javac StaticFieldExample2.java  
E:\work\chap5\5-4-1\example2>java StaticFieldExample2  
v.value = 100000  
상한값 = 100000  
E:\work\chap5\5-4-1\example2>
```

---

## ◆ 정적 메소드

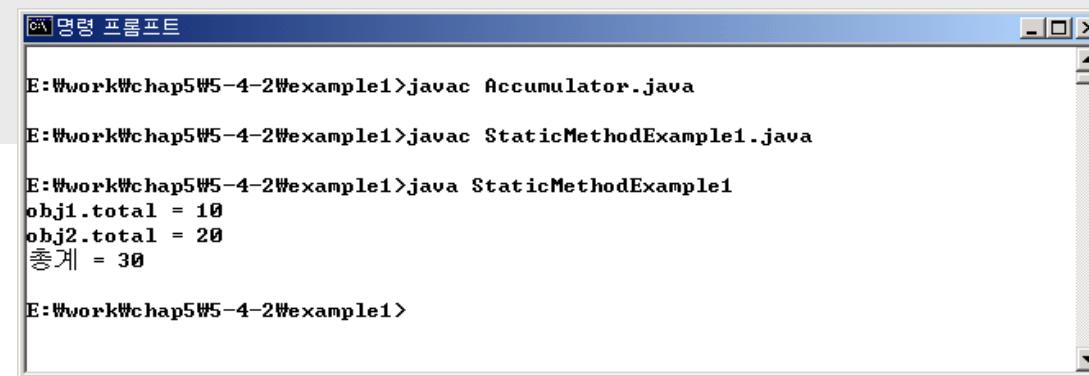
- 정적 메소드(static method) : static 키워드가 붙은 메소드
- 정적 메소드를 포함하는 클래스

```
1  class Accumulator {  
2      int total = 0;  
3      static int grandTotal = 0;  
4      void accumulate(int amount) {  
5          total += amount;  
6          grandTotal += amount;  
7      }  
8      static int getGrandTotal() {           // 정적 메소드 선언  
9          return grandTotal;  
10     }  
11 }
```

## ◆ 정적 메소드

-정적 메소드를 호출하는 프로그램

```
1  class StaticMethodExample1 {  
2      public static void main(String args[]) {  
3          Accumulator obj1 = new Accumulator();  
4          Accumulator obj2 = new Accumulator();  
5          obj1.accumulate(10);  
6          obj2.accumulate(20);  
7          int grandTotal = Accumulator.getGrandTotal();  
8          System.out.println("obj1.total = " + obj1.total);----- 정적 메소드 호출  
9          System.out.println("obj2.total = " + obj2.total);  
10         System.out.println("총계 = " + grandTotal);  
11     }  
12 }
```



The screenshot shows a Windows Command Prompt window titled '명령 프롬프트' (Command Prompt). The command line shows the user navigating to the directory 'E:\work\chap5\5-4-2\example1' and then running two Java compilation commands: 'javac Accumulator.java' and 'javac StaticMethodExample1.java'. Finally, the user runs the static method example with the command 'java StaticMethodExample1'. The output of the program is displayed below the commands, showing the values of 'obj1.total' (10), 'obj2.total' (20), and the total sum ('총계') (30).

```
E:\work\chap5\5-4-2\example1>javac Accumulator.java  
E:\work\chap5\5-4-2\example1>javac StaticMethodExample1.java  
E:\work\chap5\5-4-2\example1>java StaticMethodExample1  
obj1.total = 10  
obj2.total = 20  
총계 = 30  
E:\work\chap5\5-4-2\example1>
```

## ◆ 정적 메소드

-정적 메소드들로만 구성된 기능적 클래스

```
1  class IntBytes {  
2      static byte firstByte(int num) { } } int 타입 데이터의 1번째 바이트를 리턴하는 메소드  
3          num = (num >> 24) & 0xFF;  
4          return (byte) num;  
5      }  
6      static byte secondByte(int num) { } } int 타입 데이터의 2번째 바이트를 리턴하는 메소드  
7          num = (num >> 16) & 0xFF;  
8          return (byte) num;  
9      }  
10     static byte thirdByte(int num) { } } int 타입 데이터의 3번째 바이트를 리턴하는 메소드  
11         num = (num >> 8) & 0xFF;  
12         return (byte) num;  
13     }  
14     static byte forthByte(int num) { } } int 타입 데이터의 4번째 바이트를 리턴하는 메소드  
15         num = num & 0xFF;  
16         return (byte) num;  
17     }  
18 }
```

## ◆ 정적 초기화 블록

- 정적 초기화 블록 : static 키워드가 붙은 블록
  - ✓ 클래스가 사용되기 전에 자바 가상 기계에 의해 단 한 번 호출됨
  - ✓ 정적 필드의 초기값 설정에 주로 사용됨
- 정적 초기화 블록을 포함하는 클래스의 예

```
1 class HundredNumbers {  
2     final static int arr[];  
3     static {  
4         arr = new int[100];  
5         for (int cnt = 0; cnt < 100; cnt++)  
6             arr[cnt] = cnt;  
7     }  
8 }
```

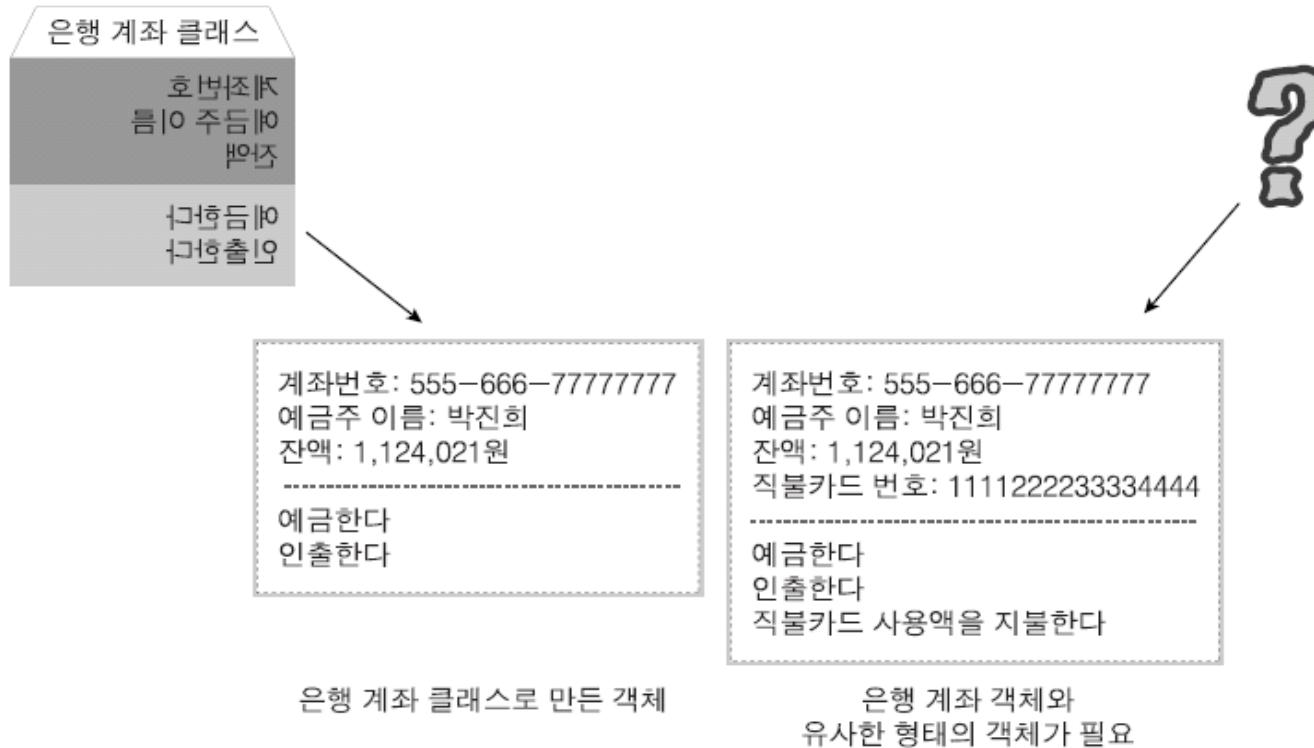
정적 초기화 블록

# 클래스 상속

# 클래스의 상속

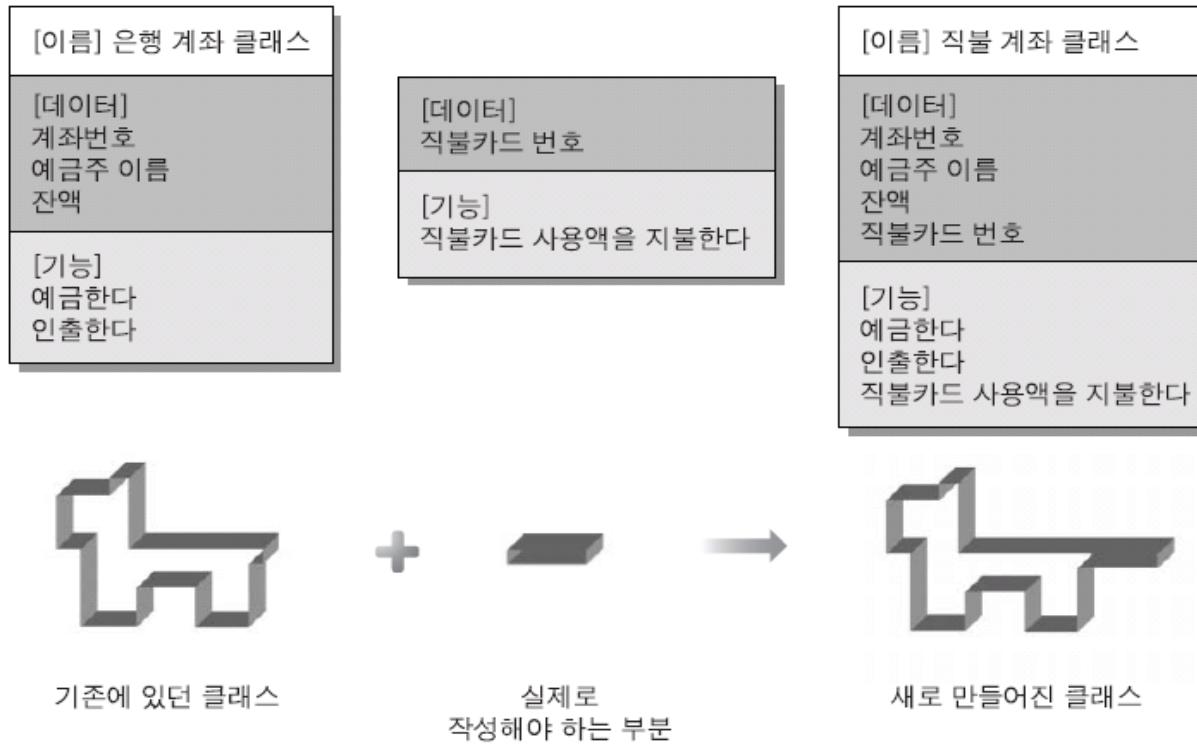
## ◆ 언제 상속이 필요한가?

-기존 클래스와 유사한 클래스를 만들어야 할 경우



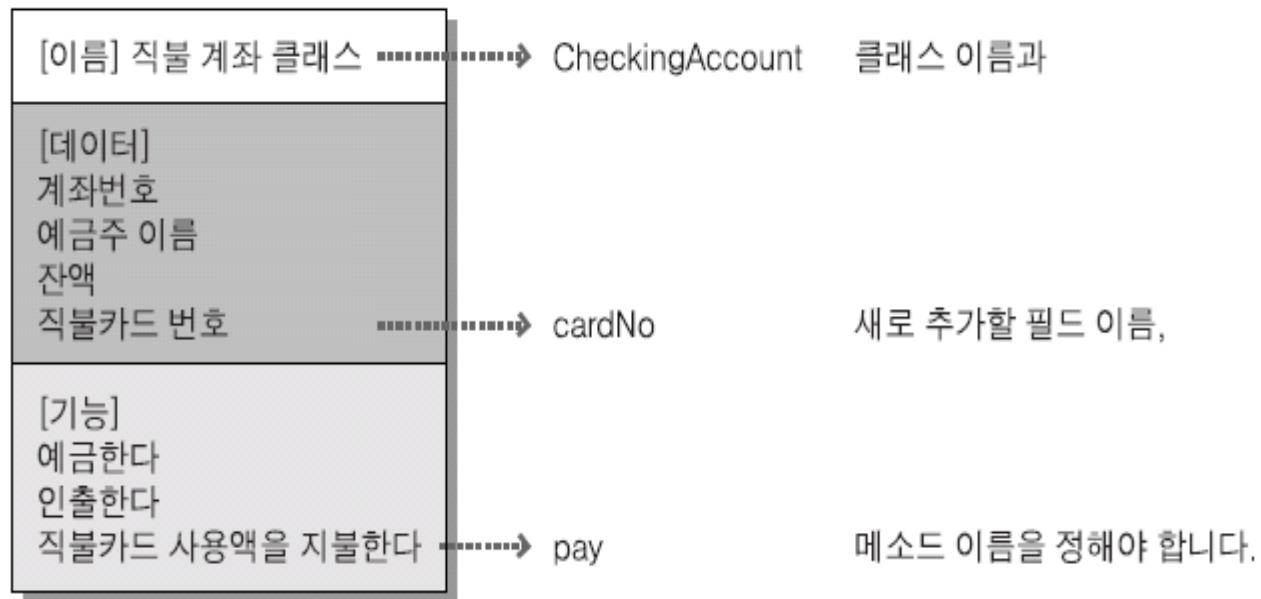
## ◆ 상속이란?

-상속(inheritance) : 기존 클래스를 확장해서 새로운 클래스를 만드는 기술



## ◆ 클래스 상속의 기초 문법

- 다른 클래스를 상속하는 클래스의 선언
- ✓ 제일 먼저 해야 할 일은 정의된 클래스와 추가되는 필드, 메소드 이름을 정하는 것

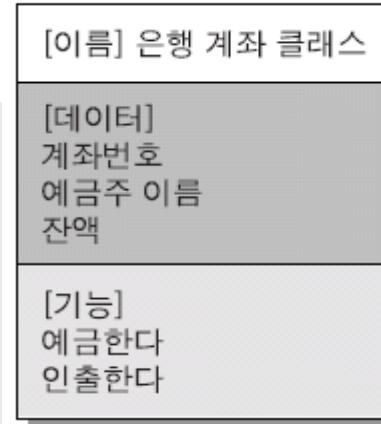


# 클래스의 상속 – 기초 문법

## ◆ 클래스 상속의 기초 문법

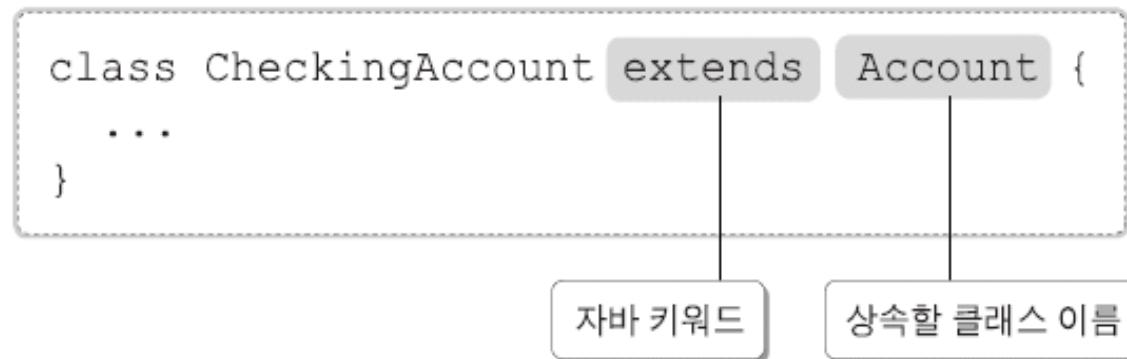
-은행 계좌 클래스

```
1  class Account {  
2      String accountNo;  
3      String ownerName;  
4      int balance;  
5      void deposit(int amount) {  
6          balance += amount;  
7      }  
8      int withdraw(int amount) throws Exception {  
9          if (balance < amount)  
10              throw new Exception("잔액이 부족합니다.");  
11          balance -= amount;  
12          return amount;  
13      }  
14  }
```



## ◆ 클래스 상속의 기초 문법

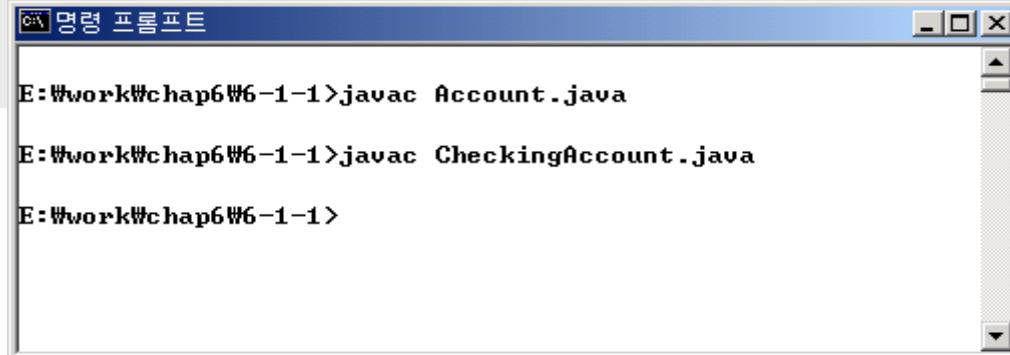
- 다른 클래스를 상속하는 클래스의 선언
  - ✓ - extends 키워드를 이용하여 상속할 클래스 이름을 명시해야 함



## ◆ 클래스 상속의 기초 문법

-은행 계좌 클래스를 상속하는 직불 계좌 클래스

```
1  class CheckingAccount extends Account {  
2      String cardNo; ----- extends 절  
3      int pay(String cardNo, int amount) throws Exception {  
4          if (!cardNo.equals(this.cardNo) || (balance < amount)) } ----- 직불카드 번호에 해당하는 필드  
5              throw new Exception("직불이 불가능합니다.");  
6          return withdraw(amount);  
7      }  
8  }
```



```
E:\work\chap6\6-1-1>javac Account.java  
E:\work\chap6\6-1-1>javac CheckingAccount.java  
E:\work\chap6\6-1-1>
```

## ◆ 클래스 상속의 기초 문법

-직불 계좌 클래스를 사용하는 프로그램

```
1  class InheritanceExample1 {  
2      public static void main(String args[]) {  
3          CheckingAccount obj = new CheckingAccount();  
4          obj.accountNo = "111-22-33333333";  
5          obj.ownerName = "홍길동";  
6          obj.cardNo = "5555-6666-7777-8888";  
7          obj.deposit(100000);  
8          try {  
9              int paidAmount = obj.pay("5555-6666-7777-8888", 47000);  
10             System.out.println("지불액:" + paidAmount);  
11             System.out.println("잔액:" + obj.balance);  
12         }  
13         catch (Exception e) {  
14             String msg = e.getMessage();  
15             System.out.println(msg);  
16         }  
17     }  
18 }
```

Account 클래스로부터 상속받은 필드 사용

-----

Account 클래스로부터 상속받은 메소드 호출

## 클래스의 상속 - 생성자

## ◆ 상속과 생성자

-생성자가 추가된 직불 계좌 클래스

```
1 class CheckingAccount extends Account {  
2     String cardNo;  
3     CheckingAccount(String accountNo, String ownerName,  
4                         int balance, String cardNo) { // 생성자  
5         this.accountNo = accountNo; }  
6         this.ownerName = ownerName; }  
7         this.balance = balance;  
8         this.cardNo = cardNo; ----- 클래스 안에 선언된 필드를 초기화합니다.  
9     }  
10    int pay(String cardNo, int amount) throws Exception {  
11        if (!cardNo.equals(this.cardNo) || (balance < amount))  
12            throw new Exception("지불이 불가능합니다.");  
13        return withdraw(amount);  
14    }
```

## ◆ 상속과 생성자

-직불 계좌 클래스의 생성자를 사용하는 프로그램

```
1  class InheritanceExample2 {  
2      public static void main(String args[]) {  
3          CheckingAccount obj = new CheckingAccount("111-22-33333333",  
4                                              "홍길동", 0, "5555-6666-7777-8888"); ----- CheckingAccount  
5          obj.deposit(100000);                                         클래스의 생성자 호출  
6          try {  
7              int paidAmount = obj.pay("5555-6666-7777-8888", 47000);  
8              System.out.println("지불액:" + paidAmount);  
9              System.out.println("잔액:" + obj.balance);  
10         }  
11         catch (Exception e) {  
12             String msg = e.getMessage();  
13             System.out.println(msg);  
14         }  
15     }
```

---

## ◆ 생성자가 있는 클래스의 상속

-생성자가 있는 Account 클래스

```
1  class Account {  
2      String accountNo;  
3      String ownerName;  
4      int balance;  
5      Account(String accountNo, String ownerName, int balance) {  
6          this.accountNo = accountNo;  
7          this.ownerName = ownerName;  
8          this.balance = balance;  
9      }  
10     void deposit(int amount) {  
11         balance += amount;  
12     }  
13     int withdraw(int amount) throws Exception {  
14         if (balance < amount)  
15             throw new Exception("잔액이 부족합니다.");  
16         balance -= amount;  
17         return amount;  
18     }  
19 }
```

#### ◆ 생성자가 있는 클래스의 상속

- 자바 컴파일러는 생성자 안의 첫번째 명령문이 슈퍼클래스의 생성자 호출문이 아니면 자동으로 슈퍼 클래스의 파라미터 없는 생성자(no-arg constructor) 호출문을 추가하기 때문
  - 슈퍼클래스의 생성자 호출문 작성 방법

```
super(accountNo, ownerName, balance);
```

↑ ↑

## 슈퍼클래스의 생성자를 호출할 때 사용하는 자바 키워드

## 생성자에 넘겨지는 파라미터들

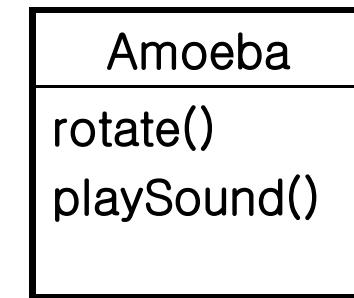
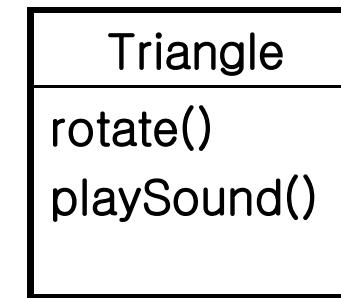
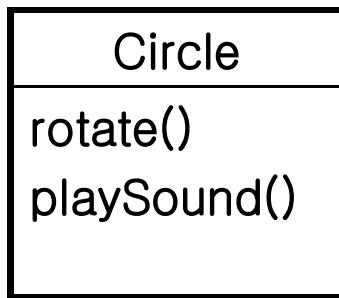
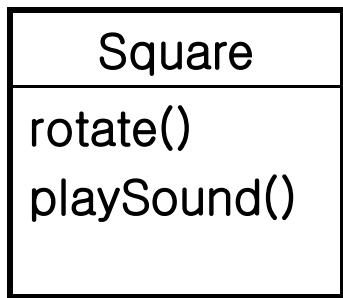
## ◆ 생성자가 있는 클래스의 상속

-슈퍼클래스의 생성자를 호출하는 직불 계좌 클래스

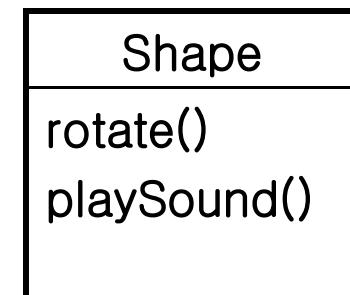
```
1  class CheckingAccount extends Account {  
2      String cardNo;  
3      CheckingAccount(String accountNo, String ownerName,  
4                          int balance, String cardNo) {  
5          super(accountNo, ownerName, balance); ----- 슈퍼클래스의 생성자 호출  
6          this.cardNo = cardNo;  
7      }  
8      int pay(String cardNo, int amount) throws Exception {  
9          if (!cardNo.equals(this.cardNo) || (balance < amount))  
10             throw new Exception("지불이 불가능합니다.");  
11          return withdraw(amount);  
12      }  
13  }
```

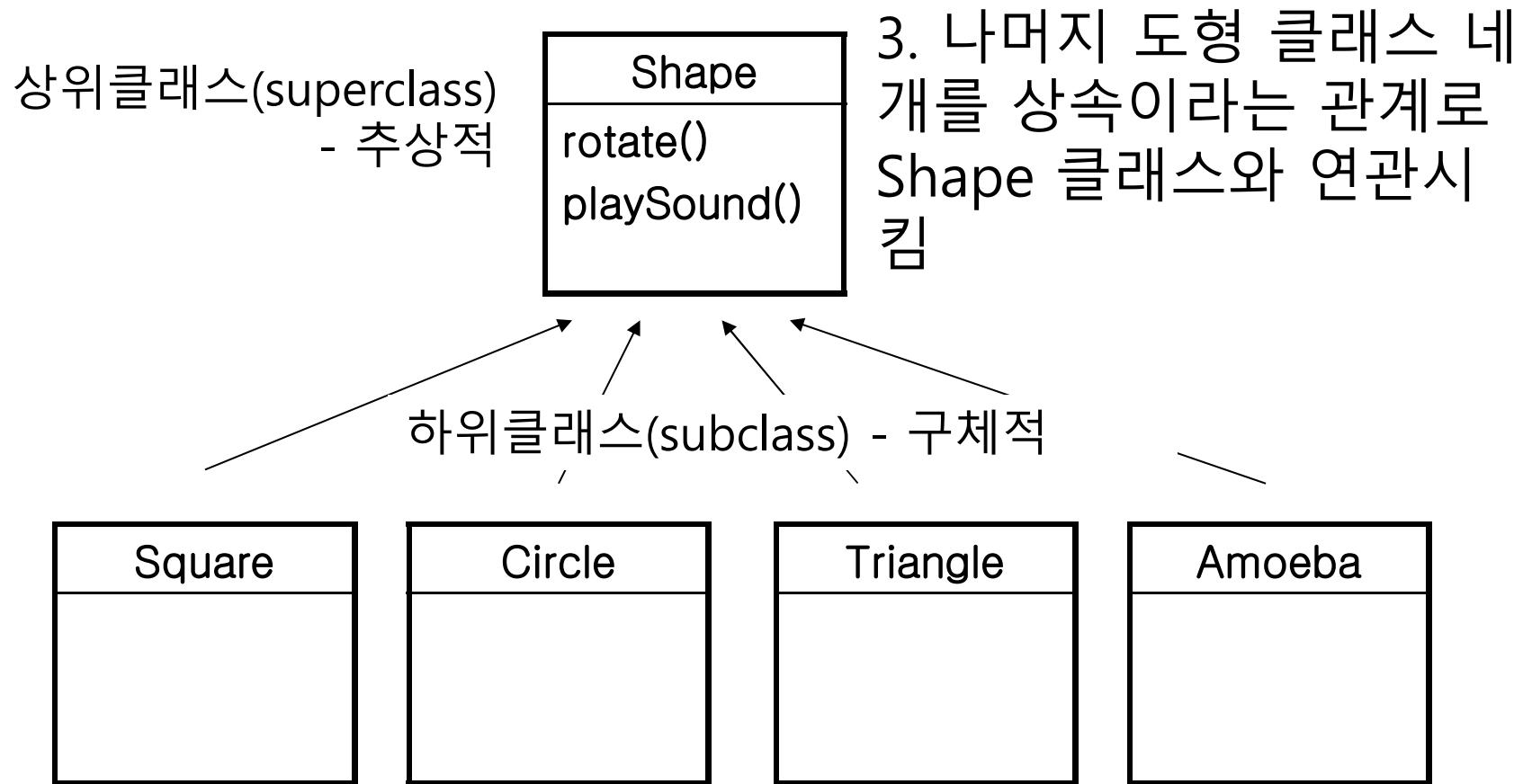
---

1. 네 클래스에 공통적으로 들어있는 것을 찾아낸다.

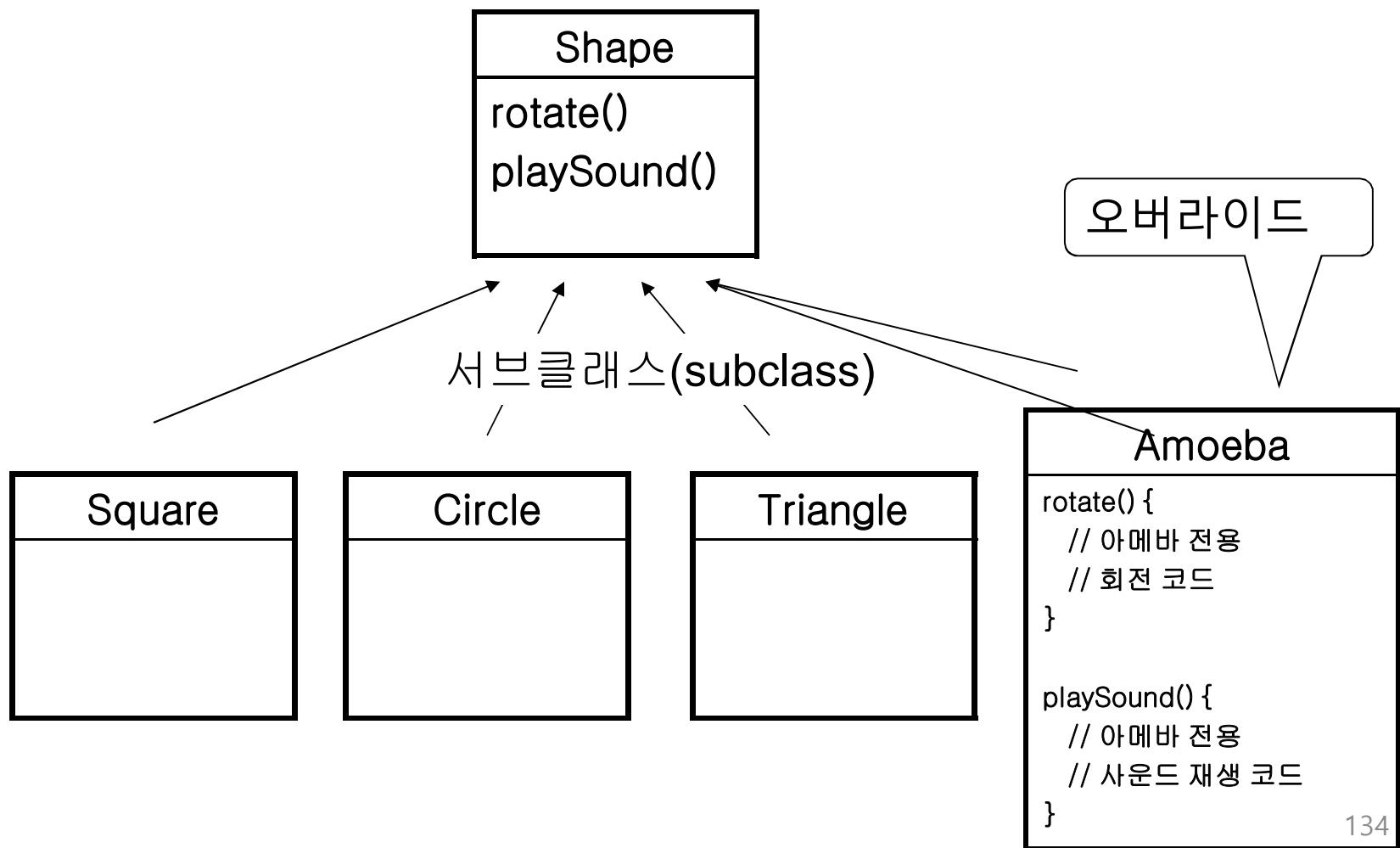


2. 모두 도형이고 회전(rotate()) 및 사운드 재생(playSound()) 기능이 있으므로 공통적인 기능을 모두 뽑아서 Shape이라는 클래스를 만든다.



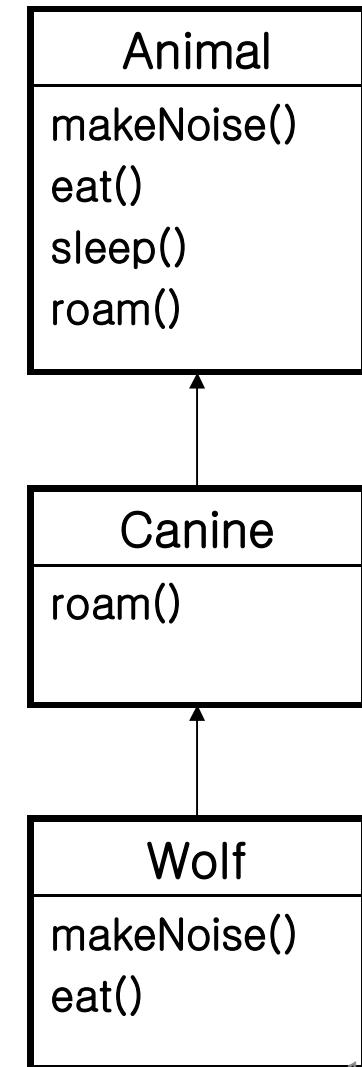


아메바의 메소드는 어떻게 처리할 수 있을까요?



## 클래스의 상속 - 'A는 B다' 테스트

- ◆ 상속 트리를 제대로 설계했다면 어떤 하위클래스에 대해서도 '하위클래스는 상위클래스이다'라는 관계가 성립합니다.
- ◆ B라는 클래스가 A라는 클래스를 확장하면 B 클래스는 A 클래스입니다.
- ◆ C라는 클래스가 B라는 클래스를 확장하면 C 클래스는 B 클래스이며, 또한 A 클래스이기도 합니다.



## 클래스의 상속 - 상속 트리 설계시 주의할 점

---

- ◆ 어떤 클래스가 다른 클래스(상위클래스)를 더 구체화한 유형이라면 상속을 활용합니다.
  - 버드나무 – 나무
  - 자동차 – 승용차
- ◆ 같은 일반적인 유형에 속하는 여러 클래스에서 공유해야 하는 어떤 행동이 있다면 상속을 활용합니다.
  - Square, Circle, Triangle – Shape

# 클래스의 상속 - 일반화 vs 특수화

## ◆ 상속은

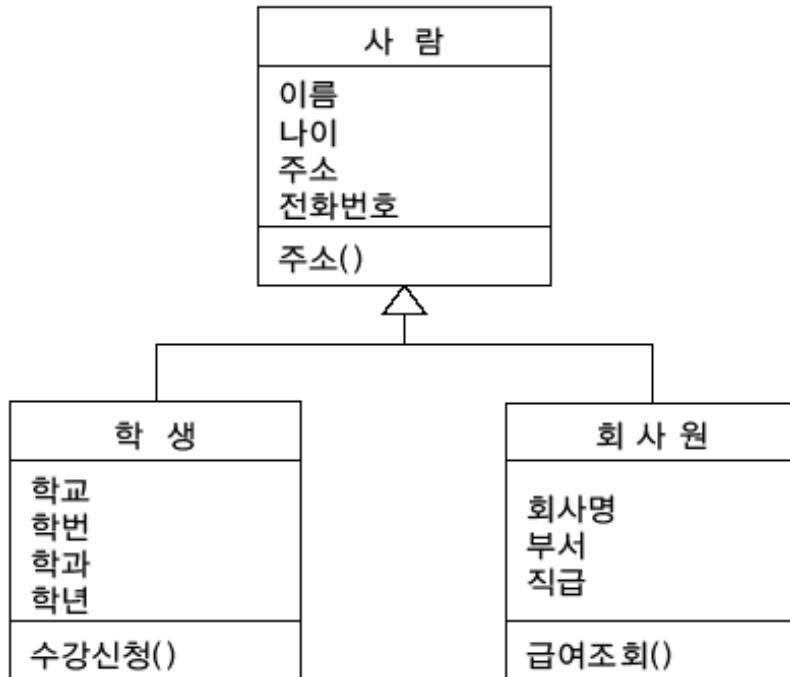
- 객체지향의 기본 개념으로, 프로그램을 쉽게 확장할 수 있도록 도와주는 수단
- 상위 클래스의 모든 특성을 하위 클래스가 이어받음으로써 이미 정의한 클래스를 재사용하고 확장할 수 있도록 지원하는 개념



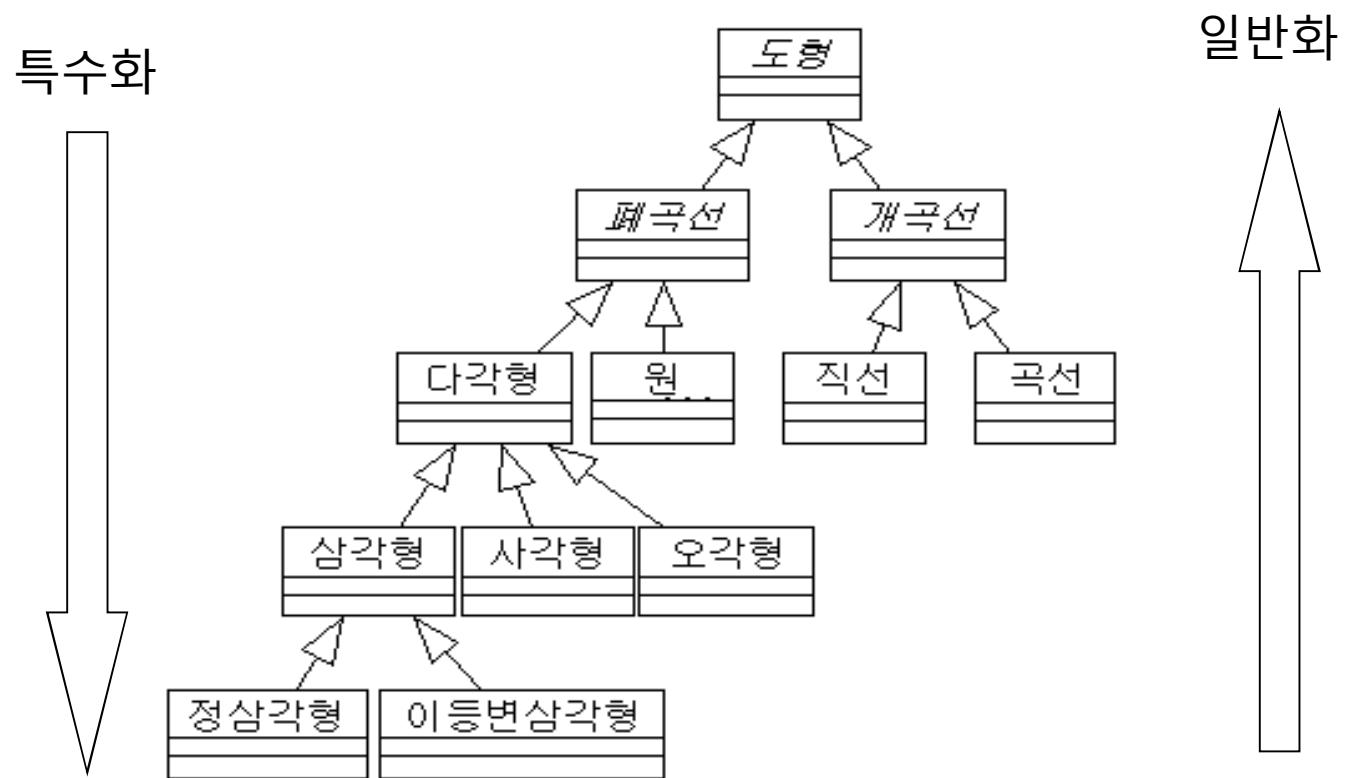
[그림 1-9] 상속

## ◆ 일반화

- 하위 클래스의 공통적인 특성을 추상화하여 상위 클래스로 정의하는 것을 일반화

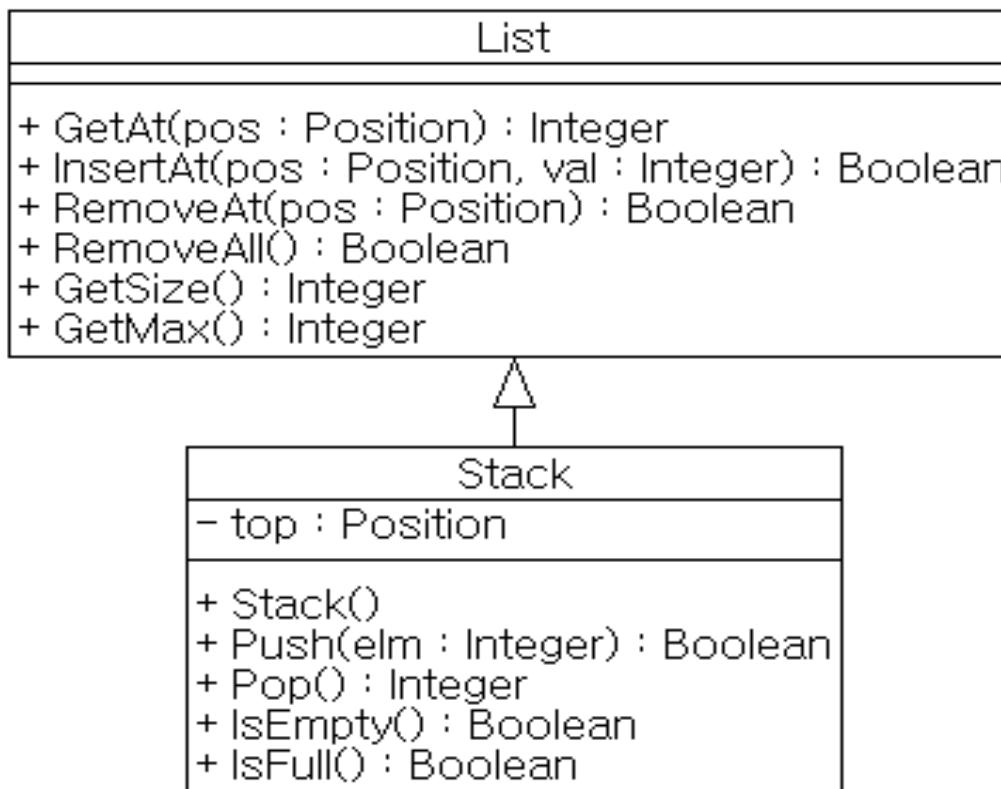


[그림 1-10] 상속을 표현한 클래스 다이어그램



# 클래스의 상속 - 올바른 상속의 사용

- ◆ IS-A 또는 IS-KIND-OF 관계가 적용될 때만 상속을 사용
- ◆ Implementation Inheritance는 지양
- ◆ HAS-A 와 구분한다(포함관계)



# 클래스의 상속 - 다형성(polymorphism)

---

## ◆ 다양한 형태의 성질

### ◆ 다형성의 개념이 적용되는 곳

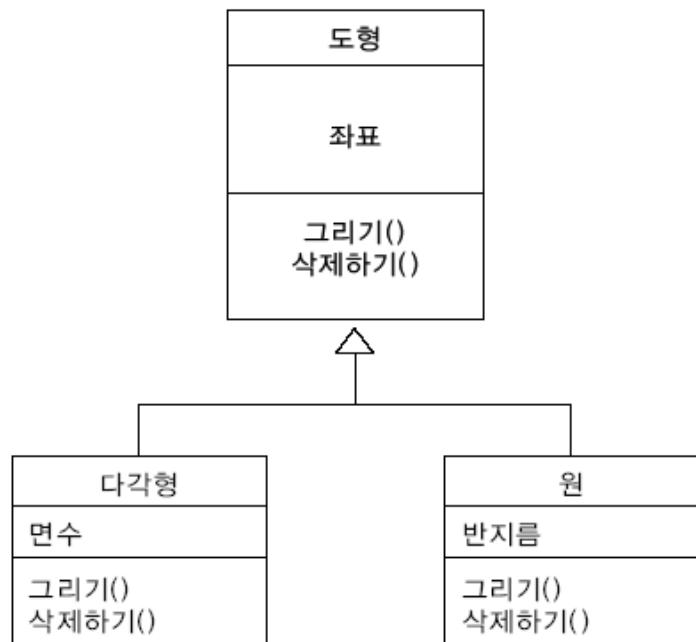
- 상속(Inheritance)
- 중복정의(Overloading)
- 재정의(Overriding)
- Upcasting
- Abstract의 상속과 Interface의 구현

### ◆ 동적 메소드 바인딩에 기반

- 동적 메소드 바인딩에는 어떤 메소드를 호출할 지 컴파일 시 지정하지 않고 실행 시에 동적으로 결정된다.
- 코드에는 호출할 주소가 아닌, 어떤 메소드를 호출해야 하는지 전체 이름이 적혀있다.
- JVM은 이걸 보고 객체 계층을 뒤져 적절한 메소드를 호출한다.
- 그러므로 어느 객체의 어느 메소드가 호출될지 컴파일 시에는 알 수 없다.
- 컴파일러는 컴파일 시 단지 타입 정보에만 의존하여 에러 체킹을 한다.

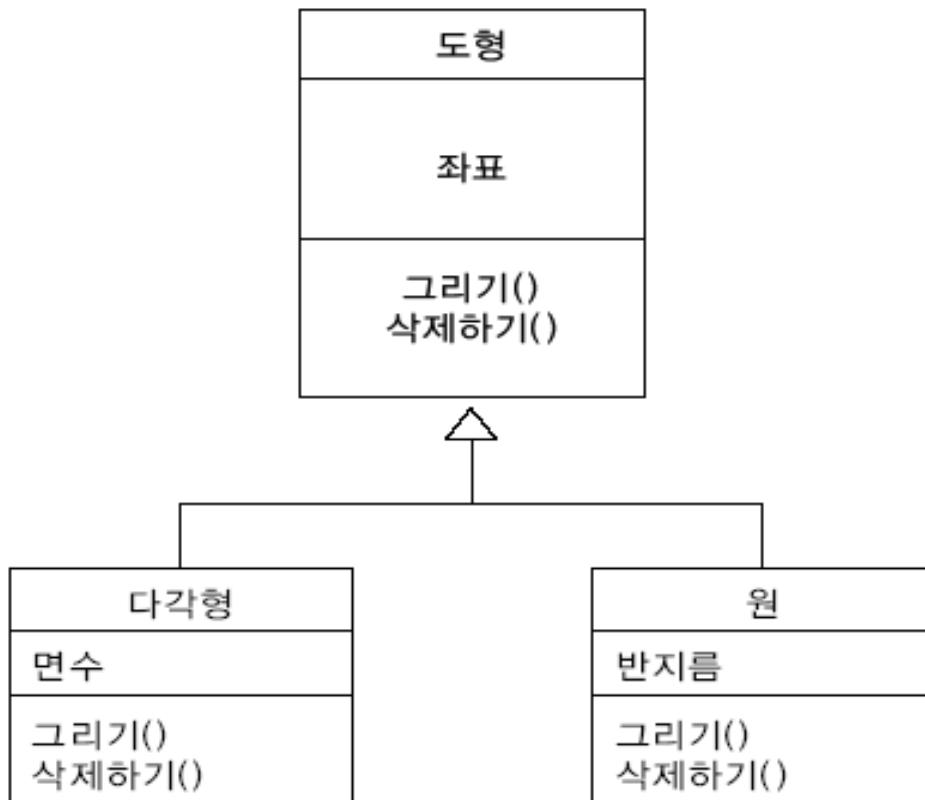
## ◆ 다형성은

- 여러 클래스에 같은 이름의 함수가 존재하지만 동작은 다르게 수행함을 의미
- 객체지향 언어에서 메서드 오버라이딩(Method Overriding) 방식으로 구현



## ◆ 다형성은

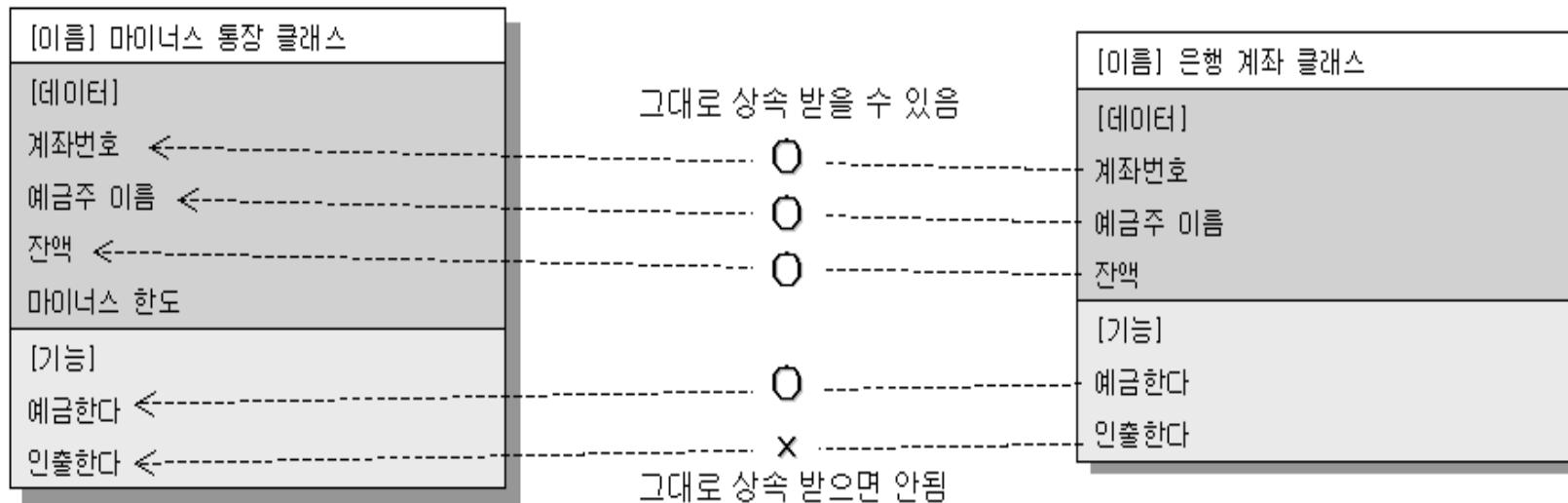
- 상위 클래스에 메서드(오퍼레이션)가 정의되고 그 메서드를 하위 클래스에서 상속받아 그 대로 사용할 수 있지만, 하위 클래스의 객체가 다른 방법으로 동작을 해야 하는 경우에는 상속받은 메서드를 같은 이름으로 재정의함으로써 생성된 객체가 상위 클래스의 정의와는 다른 동작을 할 수 있도록 구현



# 클래스의 상속 - 메서드 오버라이딩(overriding)

## ◆ 메소드 오버라이딩

- 메소드 오버라이딩이 필요한 경우

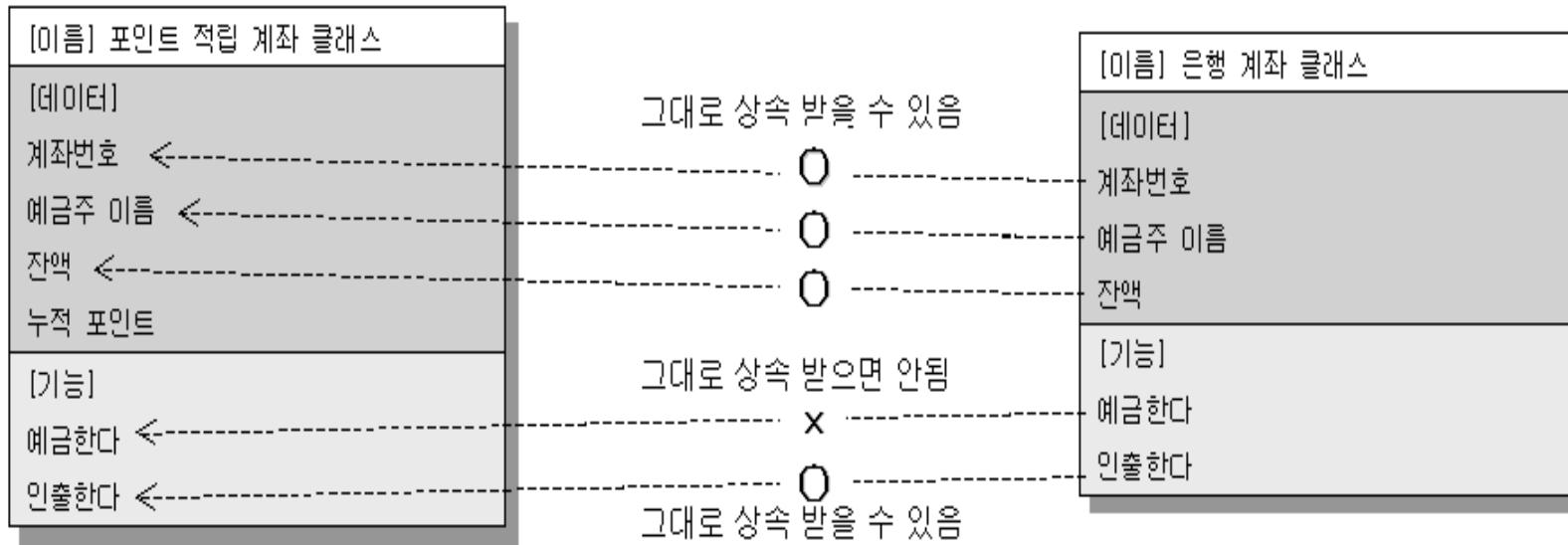


- 메소드 오버라이딩(method overriding)의 방법
  - 슈퍼클래스와 똑같은 시그니쳐를 갖는 메소드를 서브클래스에 선언

# 클래스의 상속 - 메서드 오버라이딩(overriding)

## ◆ 메소드 오버라이딩

-메소드 오버라이딩이 필요한 또 다른 경우의 예



# 클래스의 상속 - 메서드 오버라이딩(overriding)

## ◆ 메소드 오버라이딩

- 슈퍼클래스에 있는 오버라이드된 메소드를 호출하는 방법

```
super.deposit(amount);
```

호출하는 메소드가  
슈퍼클래스의 메서드임을  
가리키는 자바 키워드

메소드 이름

파라미터

# 클래스의 상속 - 메서드 오버라이딩(overriding)

---

## ◆ 상위 클래스의 일부 메서드가 하위 클래스에 적합하지 않을 경우

- 하위 클래스에서 해당 메서드만 재정의
  - 나머지 부분도 재사용될 수 있도록 해줌

## ◆ 꼭 필요한 메서드에 대해 그 프로토타입 만을 추상 메서드로 정의

- 이를 상속하는 클래스에서 메서드 재정의

## ◆ 재사용 가능한 강력한 인터페이스 구축

- 메서드 오버라이딩되면 Super 클래스의 메서드가 가려짐
- 이럴 경우 super를 사용하면 Super클래스의 메서드를 사용

# 클래스의 상속 - 메서드 오버라이딩(overriding)

---

## ◆ overriding(메서드 재정의)시 규약

- 인스턴스 메서드일 것
  - 즉, static으로 선언되어있지 않아야 함
- 메서드의 이름이 일치 할 것
- 매개변수의 개수가 일치할 것
- 매개변수 각각의 자료 형이 일치할 것
- 메서드의 리턴 형이 일치할 것
- 접근 제어자는 부모 클래스의 메서드 보다 좁은 범위로 변경할 수 없음
- 부모 클래스의 메서드 보다 많은 수의 예외를 선언할 수 없다.

# 클래스의 상속 - overloading vs overriding

---

## ◆ overloading

- 동일한 클래스 내에서 같은 이름의 메소드를 중복 정의하여 다형성을 지원
- 메서드 이름은 동일하나 매개변수의 형이나 매개변수의 수가 다를 것

## ◆ overriding

- 상속 관계에 있는 클래스간에 메소드를 다시 정의하여 다형성을 지원
- 즉, 메소드 오버라이딩을 이용하면 하위 클래스에서  
동일 이름의 메소드를 새롭게 정의 가능
- 매개변수의 형이나 매개변수의 수 모두 동일해야 함

# 클래스의 상속 - super

---

## ◆ 상위 클래스의 멤버변수나 메서드를 명시할 때

- `super.variable_name`
- `super.method_name(arguments)`

## ◆ 상위 클래스의 생성자를 호출할 때

- `super( arguments )`

# 클래스의 상속 -final

---

- ◆ final 변수 : 상수

- => 지역변수 final 을 불일시 선언후 변수 사용시 초기화가 가능하나 그 이후에 변경을 할 수 없다.

- => final 멤버 변수 : 선언시 초기화를 하거나 생성자를 이용하여 초기화

- => final 클래스 변수 : 선언시 초기화하거나 static 초기화 블록을 이용한다.

- ◆ final 메서드 : 오버라이딩 불가

- ◆ final 클래스 : 상속 불가 ( 종단클래스)

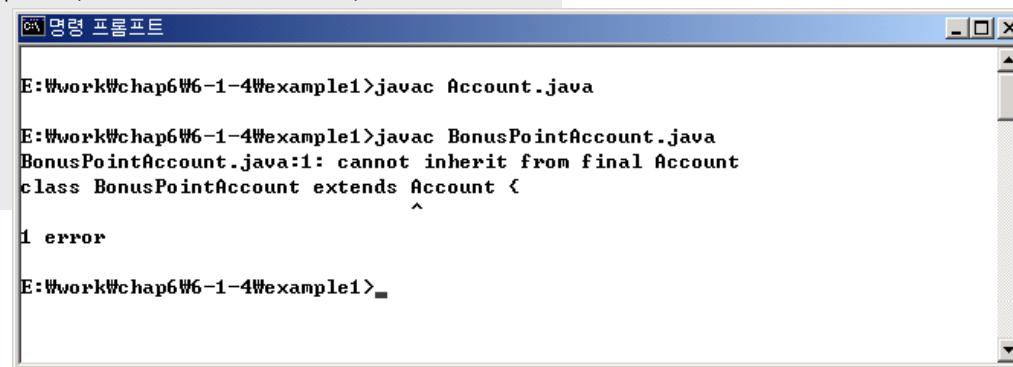
# 클래스의 상속 -final

## ◆ 상속을 금지하는 final 키워드

-final 키워드를 추가한 Account 클래스

final 키워드 추가

```
1  [final] class Account {  
2      String accountNo;  
3      String ownerName;  
4      int balance;  
5      Account(String accountNo, String ownerName, int balance) {  
6          this.accountNo = accountNo;  
7          this.ownerName = ownerName;  
8          this.balance = balance;  
9      }  
10     void deposit(int amount) {  
11         balance += amount;  
12     }  
13     int withdraw(int amount) throws Exception {  
14         if (balance < amount)  
15             throw new Exception("잔액이 부족합니다.");  
16         balance -= amount;  
17         return amount;  
18     }  
19 }
```

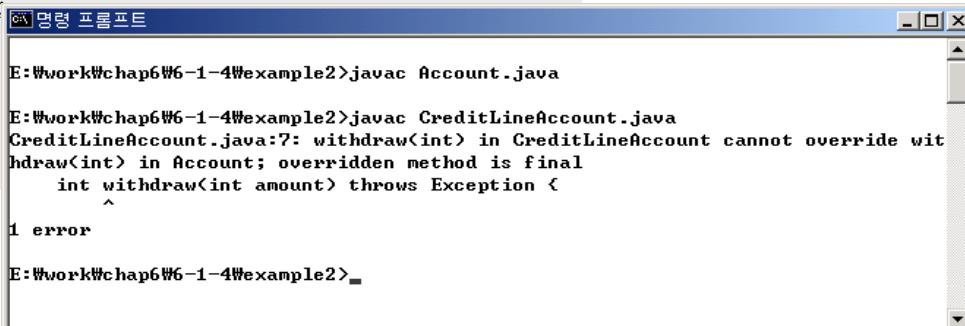


# 클래스의 상속 -final

## ◆ 메소드 오버라이딩을 금지하는 final 키워드

-withdraw 메소드에 final 키워드를 추가한 Account 클래스

```
1  class Account {  
2      String accountNo;  
3      String ownerName;  
4      int balance;  
5      Account(String accountNo, String ownerName, int balance) {  
6          this.accountNo = accountNo;  
7          this.ownerName = ownerName;  
8          this.balance = balance;  
9      }  
10     void deposit(int amount) {  
11         balance += amount;  
12     } [----- final 키워드 추가 -----]  
13     final int withdraw(int amount) throws Exception {  
14         if (balance < amount)  
15             throw new Exception("잔액이 부족합니다.");  
16         balance -= amount;  
17         return amount;  
18     }  
19 }
```



The terminal window shows the following output:

```
E:\work\chap6\6-1-4\example2>javac Account.java  
  
E:\work\chap6\6-1-4\example2>javac CreditLineAccount.java  
CreditLineAccount.java:7: withdraw(int) in CreditLineAccount cannot override withdraw(int) in Account; overridden method is final  
    int withdraw(int amount) throws Exception {  
                           ^  
1 error  
  
E:\work\chap6\6-1-4\example2>
```

# 클래스의 상속 - 추상 클래스

---

## ◆ 추상 클래스(abstract class)란?

- 인스턴스를 만들 수 없는 클래스
- 반드시 확장해야 함
- 추상 유형을 레퍼런스로 사용할 수는 있음
- 다형적인 인자, 리턴 유형, 배열 등에 활용

```
abstract class Canine extends Animal {  
    public void roam() {}  
}
```

## 클래스의 상속 - 추상 클래스

---

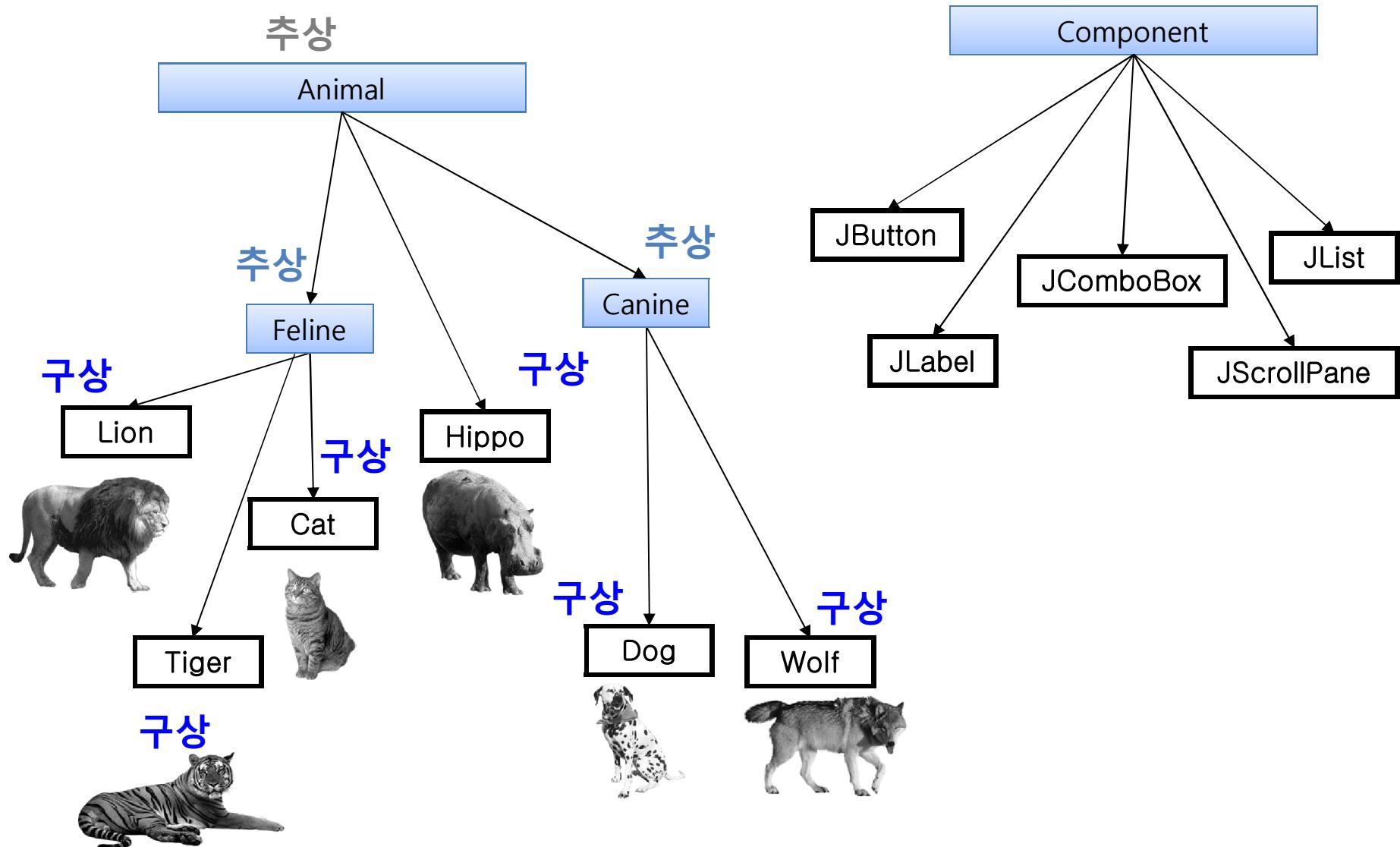
```
public abstract class Canine extends Animal
{
    public void roam() { }
}
```

---

```
public class MakeCanine {
    public void go() {
        Canine c;
        c = new Dog();
        c = new Canine();
        c.roam();
    }
}
```

```
% javac MakeCanine.java
MakeCanine.java:5: Canine is abstract;
Cannot be instantiated
        c = new Canine();
                           ^
1 error
```

# 클래스의 상속 - 추상 vs. 구상



# 클래스의 상속 - 추상 클래스

## ◆ 인스턴스화를 금지하는 **abstract** 키워드

-**abstract** 키워드를 추가한 Account 클래스

```
1  [-----] class Account {  
2      String accountNo;  
3      String ownerName;  
4      int balance;  
5      Account(String accountNo, String ownerName, int balance) {  
6          this.accountNo = accountNo;  
7          this.ownerName = ownerName;  
8          this.balance = balance;  
9      }  
10     void deposit(int amount) {  
11         balance += amount;  
12     }  
13     int withdraw(int amount) throws Exception {  
14         if (balance < amount)  
15             throw new Exception("잔액이 부족합니다.");  
16         balance -= amount;  
17         return amount;  
18     }  
19 }
```

**abstract** 키워드 추가

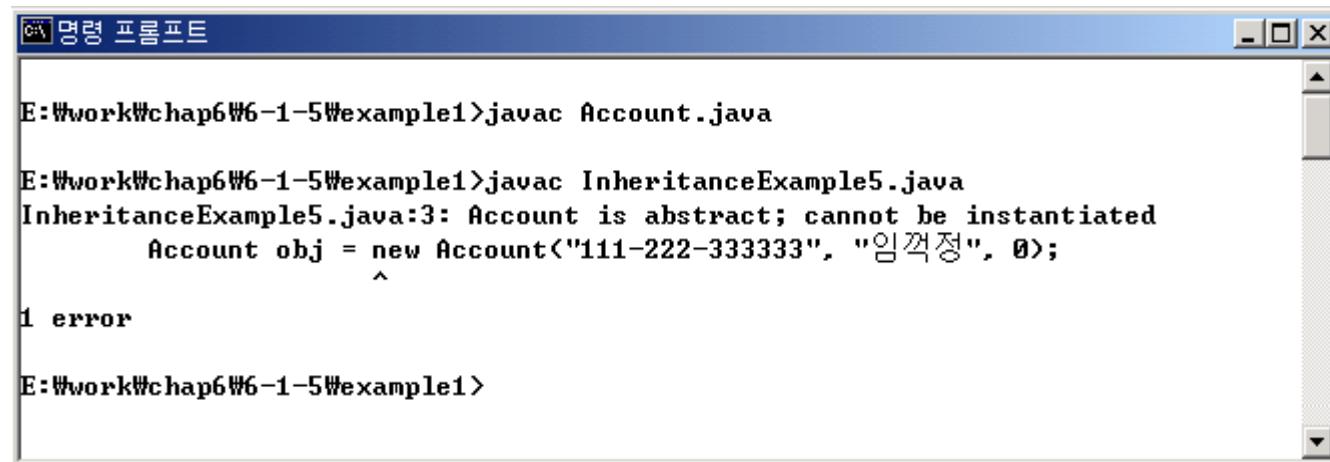
**abstract** 키워드가 붙은 클래스를  
**추상 클래스(Abstract Class)**라고 함

# 클래스의 상속 - 추상 클래스

## ◆ 인스턴스화를 금지하는 **abstract** 키워드

-추상 클래스를 인스턴스화하는 잘못된 프로그램

```
1 class InheritanceExample5 {  
2     public static void main(String args[]) {  
3         Account obj = new Account("111-222-33333", "임꺽정", 0);  
4     }  
5 }
```



The screenshot shows a Windows Command Prompt window titled '명령 프롬프트' (Command Prompt). The command line shows two compilation steps: 'javac Account.java' and 'javac InheritanceExample5.java'. The second compilation results in an error message: 'InheritanceExample5.java:3: Account is abstract; cannot be instantiated' followed by the line of code 'Account obj = new Account("111-222-33333", "임꺽정", 0);' with a cursor arrow pointing to the first character of 'Account'. Below the error message, it says '1 error'. The command prompt ends with 'E:\work\chap6\6-1-5\example1>'.

# 클래스의 상속 - 추상 메소드

---

## ◆ 추상 메소드 (abstract method)

- 몸통이 없는 메소드
- 반드시 오버라이드해야 합니다.

```
public abstract void eat();
```

- 추상 메소드를 만들 때는 클래스도 반드시 추상 클래스로 만들어야 합니다.
- 추상 클래스가 아닌 클래스에는 추상 메소드를 집어넣을 수 없습니다.



# 클래스의 상속 - 추상메서드

---

## ◆ 추상 메소드 구현 == 메소드 오버라이드

- 상속 트리에서 처음으로 등장하는 구상 클래스에서 모든 추상 메소드를 구현해야만 합니다.
- 추상 메소드에는 추상 메소드와 구상 메소드가 모두 들어갈 수 있지만 구상 메소드에는 추상 메소드가 들어갈 수 없습니다.
- 메소드 서명과 리턴형이 똑같은 추상 메소드가 아닌 메소드를 만들기만 하면 됩니다.  
(자바에서는 메소드의 코드가 어떻게 되는지는 신경 쓰지 않습니다.)

# 클래스의 상속 - 추상메서드

## ◆ 추상 메소드

- 추상 메소드(abstract method) : 메소드 본체가 없는 메소드
- 추상 메소드가 필요한 경우

[이름] 메시지 발송 클래스
[데이터] 제목 발송자 이름
[기능] 메시지를 송신한다



[이름] 이메일 송신 클래스
[데이터] 제목 발송자 이름 발송자 이메일 주소 이메일 내용
[기능] 메시지를 송신한다

[이름] 문자메시지 송신 클래스
[데이터] 제목 발송자 이름 회신 전화번호 메시지 본문
[기능] 메시지를 송신한다

# 클래스의 상속 - 추상메서드

## ◆ 추상 메소드

-추상 메소드의 선언 방법

```
abstract void sendMessage(String recipient);
```

본체가 없는 메소드를 선언할 때  
반드시 써야 하는 자바 키워드

메소드의 리턴 타입, 이름,  
파라미터 변수 선언

메소드 본체 대신  
세미콜론을 써야 함

# 클래스의 상속 - 추상메서드

## ◆ 추상 메소드

-추상 클래스를 포함하는 클래스 – 메시지 발송 클래스

```
1 abstract class MessageSender {  
2     String title;  
3     String senderName;  
4     MessageSender(String title, String senderName) {  
5         this.title = title;  
6         this.senderName = senderName;  
7     }  
8     abstract void sendMessage(String recipient);  
9 }
```

----- 클래스 자체도 추상 클래스로 선언

----- 추상 메소드 선언

# 클래스의 상속 - 추상메서드

## ◆ 추상 메소드

-메시지 발송 클래스를 상속하는 이메일 송신 클래스

```
1  class EMailSender extends MessageSender {  
2      String senderAddr;  
3      String emailBody;  
4      EMailSender(String title, String senderName,  
           String senderAddr, String emailBody) {  
5          super(title, senderName);  
6          this.senderAddr = senderAddr;  
7          this.emailBody = emailBody;  
8      }  
9      void sendMessage(String recipient) {  
10         System.out.println("-----");  
11         System.out.println("제목: " + title);  
12         System.out.println("보내는 사람: " + senderName +  
           " " + senderAddr);  
13         System.out.println("받는 사람: " + recipient);  
14         System.out.println("내용: " + emailBody);  
15     }  
16 }
```

슈퍼클래스의 메소드를  
오버라이드하는 메소드

# 클래스의 상속 - 추상메서드

## ◆ 추상 메소드

-메시지 발송 클래스를 상속하는 문자 메시지 송신 클래스

```
1  class SMSSender extends MessageSender {  
2      String returnPhoneNo;  
3      String message;  
4      SMSSender(String title, String senderName,  
5                  String returnPhoneNo, String message) {  
6          super(title, senderName);  
7          this.returnPhoneNo = returnPhoneNo;  
8          this.message = message;  
9      }  
10     void sendMessage(String recipient) {  
11         System.out.println("-----");  
12         System.out.println("제목: " + title);  
13         System.out.println("보내는 사람: " + senderName);  
14         System.out.println("전화번호: " + recipient);  
15         System.out.println("회신 전화번호: " + returnPhoneNo);  
16         System.out.println("메시지 내용: " + message);  
17     }  
18 }
```

슈퍼클래스의  
메소드를  
오버라이드하는  
메소드

# 클래스의 상속 - 추상메서드

## ◆ 추상 메소드

- 이메일 송신 클래스와 문자 메시지 송신 클래스를 사용하는 프로그램

```
1  class InheritanceExample6 {  
2      public static void main(String args[]) {  
3          EMailSender obj1 = new EMailSender("생일을 축하합니다", "고객센터",  
4                                         "admin@dukeeshop.co.kr", "10% 할인쿠폰이 발행되었습니다.");  
5          SMSender obj2 = new SMSender("생일을 축하합니다", "고객센터",  
6                                         "02-000-0000", "10% 할인쿠폰이 발행되었습니다.");  
7          obj1.sendMessage("hatman@eyeye.com");  
8          obj1.sendMessage("stickman@hahaha.com");  
9          obj2.sendMessage("010-000-0000");  
11     }  
12 }
```

}

추상 메소드를 구현하는 메소드를 호출합니다.

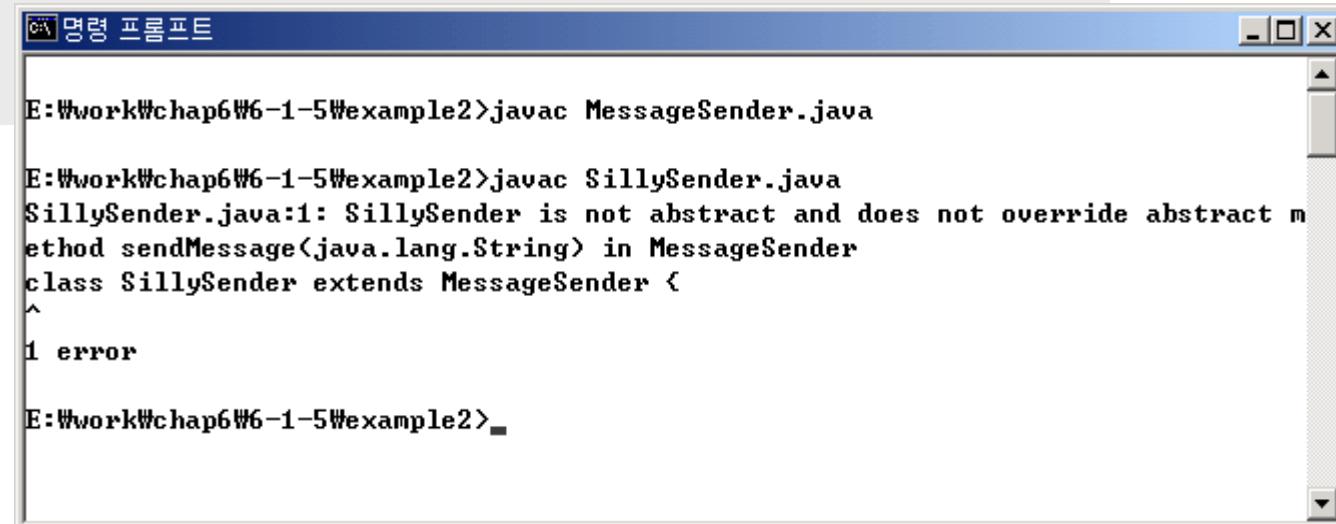
```
E:\work\chap6\6-1-5\example2>java InheritanceExample6  
제목: 생일을 축하합니다  
보내는 사람: 고객센터 admin@dukeeshop.co.kr  
받는 사람: hatman@eyeye.com  
내용: 10% 할인쿠폰이 발행되었습니다.  
-----  
제목: 생일을 축하합니다  
보내는 사람: 고객센터 admin@dukeeshop.co.kr  
받는 사람: stickman@hahaha.com  
내용: 10% 할인쿠폰이 발행되었습니다.  
-----  
제목: 생일을 축하합니다  
보내는 사람: 고객센터  
전화번호: 010-000-0000  
회신 전화번호: 02-000-0000  
메시지 내용: 10% 할인쿠폰이 발행되었습니다.  
E:\work\chap6\6-1-5\example2>
```

# 클래스의 상속 - 추상메서드

## ◆ 추상 메소드

-메시지 발송 클래스를 상속하는 클래스 – 잘못된 예

```
1 class SillySender extends MessageSender {  
2     SillySender(String title, String senderName) { // 생성자  
3         super(title, senderName);  
4     }  
5 }
```



The screenshot shows a Windows Command Prompt window titled "명령 프롬프트". The command E:\work\chap6\6-1-5\example2>javac MessageSender.java is run, followed by E:\work\chap6\6-1-5\example2>javac SillySender.java. The output shows an error message: "SillySender.java:1: SillySender is not abstract and does not override abstract method sendMessage(java.lang.String) in MessageSender class SillySender extends MessageSender < ^ 1 error".

```
E:\work\chap6\6-1-5\example2>javac MessageSender.java  
  
E:\work\chap6\6-1-5\example2>javac SillySender.java  
SillySender.java:1: SillySender is not abstract and does not override abstract m  
ethod sendMessage(java.lang.String) in MessageSender  
class SillySender extends MessageSender <  
^  
1 error  
  
E:\work\chap6\6-1-5\example2>
```

# 클래스의 상속 - 추상메서드

---

- ◆ interface는 인터페이스
- ◆ 추상 메서드만을 포함하는 추상클래스의 일종
- ◆ 일반 메서드 사용시 default를 앞에서 선언(JDK 8)
- ◆ 메서드는 묵시적으로 public abstract
- ◆ 변수는 public static final만 사용가능
- ◆ 구현을 위해 implements를 사용  
ex) implements Cloneable, Serializable

- ◆ 단일 상속의 한계를 극복
  - 자바에서는 다중 상속을 지원하지 않았다.
  - 그러나 다중 서브타이핑이 필요한 경우도 있다.
- ◆ interface A {

```
default void prt() {  
    System.out.println("ttt");  
}  
void tt();  
}
```

# 인터페이스 - 구조

---

- ◆ 모든 메서드는 묵시적으로 **public abstract**  
메서드에 **default**를 사용하여 **non-abstract method** 작성
- ◆ 변수는 **public static final**

## static final의 예

Math클래스의 멤버필드 – public static final double PI  
Color클래스의 멤버필드 – public static final Color black

사용할 때

Math.PI

Color.black

- ◆ 자체적으로 객체 생성이 불가능
- ◆ **Implements**를 사용하여 하위클래스에서 구현

# 인터페이스

---

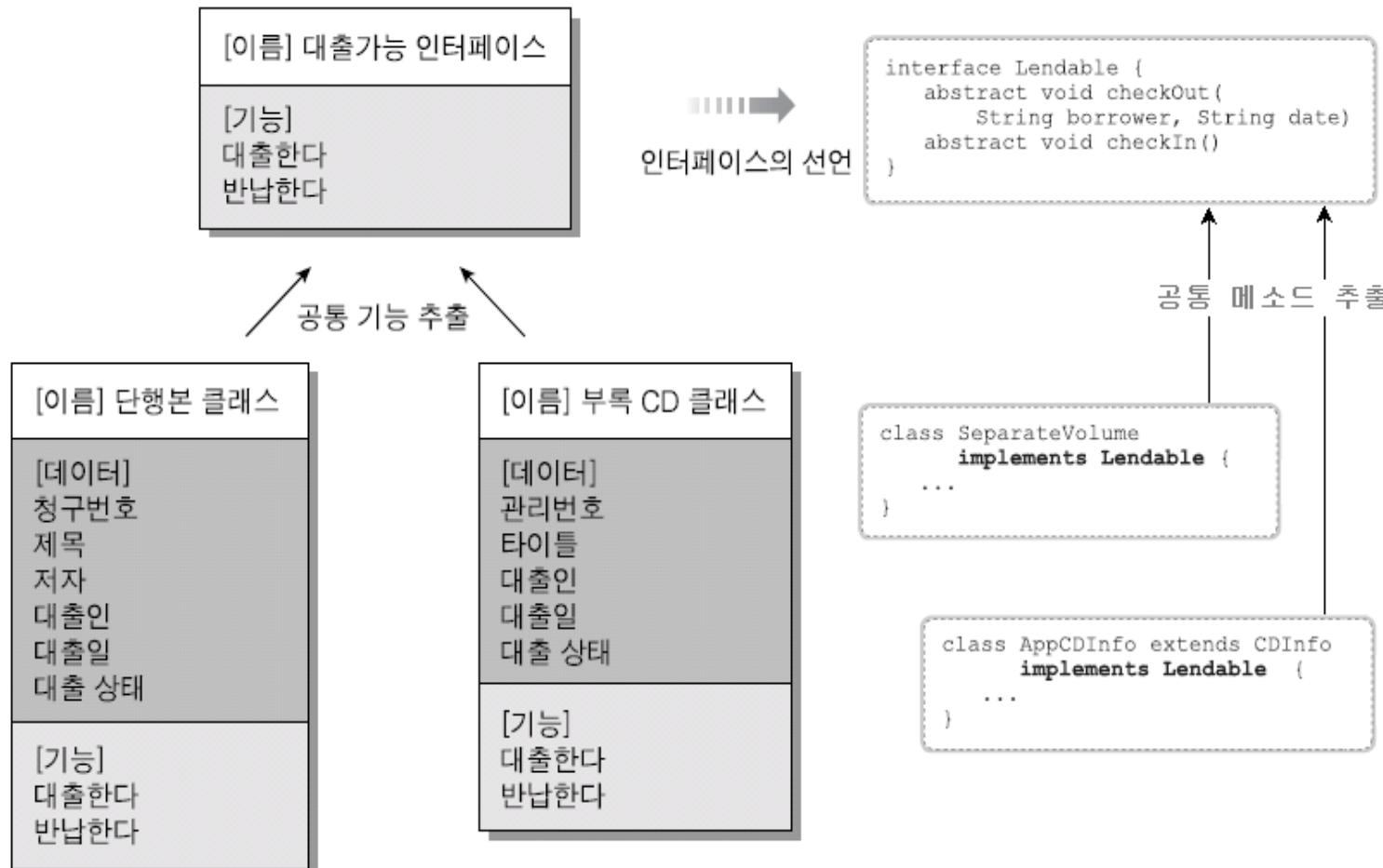
- ◆ 서로 다른 상속 트리에서 같은 인터페이스를 구현할 수 있습니다.
- ◆ 한 클래스에서 여러 개의 인터페이스를 구현할 수도 있습니다.

```
public class Dog extends Animal implements Pet, Saveable, Paintable {...}
```

# 인터페이스

## ◆ 인터페이스란?

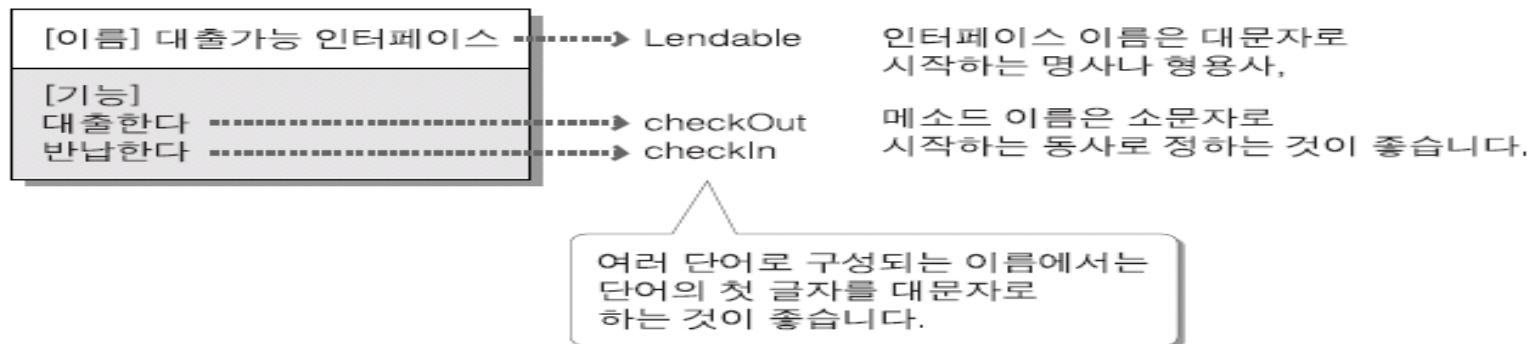
- 자바에서는 클래스의 다중 상속을 허용하지 않음



# 인터페이스

## ◆ 인터페이스의 선언

- 인터페이스의 선언 방법



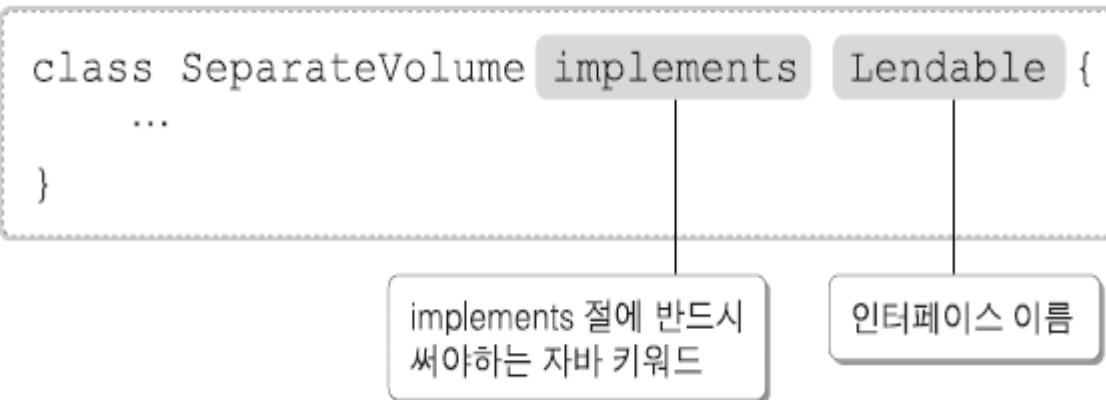
```
interface Lendable {  
    void checkOut(String borrower, String date);  
    void checkIn();  
}
```

```
interface Lendable {  
    public abstract void checkOut(String borrower, String date);  
    public abstract void checkIn();  
}
```

# 인터페이스

## ◆ 인터페이스를 구현하는 클래스의 선언

- 인터페이스를 구현하는 클래스의 선언 방법



# 인터페이스

## ◆ 인터페이스를 구현하는 클래스의 선언

-extends 절과 implements 절이 모두 있는 클래스의 선언 방법

```
class AppCDInfo extends CDInfo implements Lendable {  
    ...  
}
```

extends 절이 먼저 오고

그 다음에 implements 절이 와야 합니다.

# 인터페이스

---

## ◆ 인터페이스의 사용 방법

- 인터페이스를 가지고 할 수 없는 일

✓ 객체 생성에 사용하는 것은 불가능

- 클래스에 메소드 로직을 상속해줄 수 없음

```
obj = new Lendable(); // 잘못된 예
```

- 인터페이스를 가지고 할 수 있는 일

- 클래스의 선언 방법을 제한할 수 있음

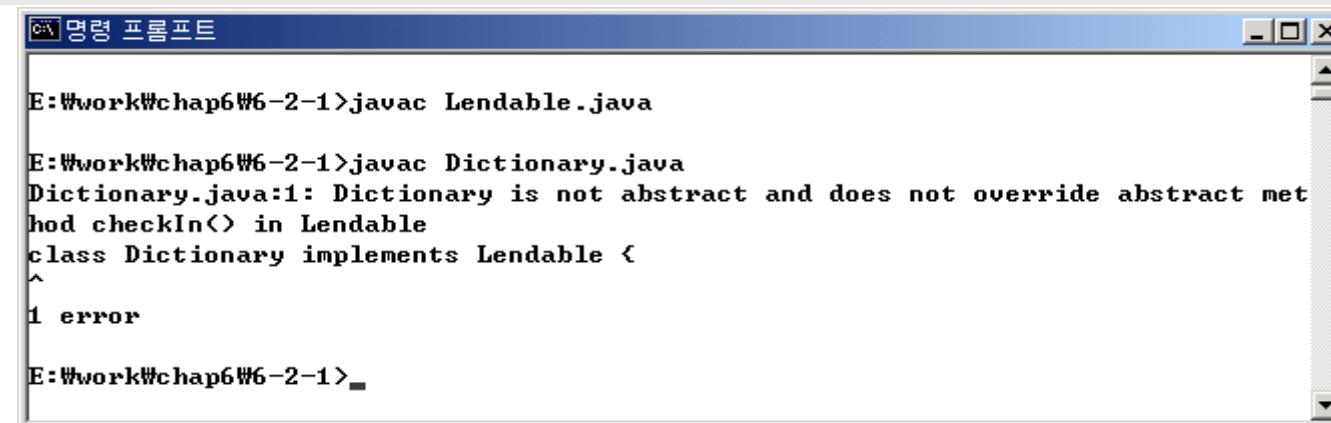
- 인터페이스 변수 선언에 사용됨

# 인터페이스

## ◆ 클래스의 선언 방법을 제한하는 인터페이스

-Lendable 인터페이스를 구현하는 클래스의 예 – 잘못된 예

```
1  class Dictionary implements Lendable {  
2      String title;  
3      Dictionary(String title) {  
4          this.title = title;  
5      }  
6  }
```



The screenshot shows a Windows command prompt window titled "명령 프롬프트". The command line shows two compilation attempts:

```
E:\work\chap6\6-2-1>javac Lendable.java  
E:\work\chap6\6-2-1>javac Dictionary.java
```

The second compilation attempt results in an error message:

```
Dictionary.java:1: Dictionary is not abstract and does not override abstract method checkIn() in Lendable  
class Dictionary implements Lendable {  
^  
1 error
```

The command line ends with:

```
E:\work\chap6\6-2-1>
```

# 인터페이스- 다형성

---

## ◆ 인터페이스 변수의 다형성

- 인터페이스 변수의 선언

```
Lendable obj;
```

- 인터페이스 변수의 사용

- ✓ - 인터페이스 변수에는 그 인터페이스를 구현하는 클래스의 객체를 대입할 수 있음

```
obj = new SeparateVolume("863MB774개", "개미", "베르베르");  
obj = new AppCDInfo("2006-7001", "Redhat Fedora");
```

## ◆ 인터페이스 변수의 다형성

- Lendable 인터페이스 변수의 다형성을 이용하는 프로그램

```
1  class InterfaceExample2 {  
2      public static void main(String args[]) {  
3          Lendable arr[] = new Lendable[3];  
4          arr[0] = new SeparateVolume("883◦", "푸코의 진자", "에코");  
5          arr[1] = new SeparateVolume("609.2", "서양미술사", "곰브리치");  
6          arr[2] = new AppCDInfo("02-17", "XML을 위한 자바 프로그래밍");  
7          checkOutAll(arr, "윤지혜", "20060315");  
8      }  
9      static void checkOutAll(Lendable arr[], String borrower, String date) {  
10         for (int cnt = 0; cnt < arr.length; cnt++)  
11             arr[cnt].checkOut(borrower, date);  
12     }  
13 }
```

인터페이스 타입의 배열

배열에 여러 타입의 객체 저장

배열을 파라미터로 넘김

배열의 모든 항목에 대해  
checkOut 메소드 호출

# 인터페이스

---

## ◆ 인터페이스의 상수 필드

-인터페이스의 상수 필드 선언 방법 (1)

```
final static int MAXIMUM = 100;
```

-인터페이스의 상수 필드 선언 방법 (2)

```
static int MAXIMUM = 100;  
final int MAXIMUM = 100;  
int MAXIMUM = 100;
```

- 인터페이스 상수 필드 선언 방법 (3)

```
public final static int MAXIMUM=100;
```

# 인터페이스 – 인터페이스 상속

---

## ◆ 인터페이스의 상속

– 인터페이스의 상속이 필요한 경우의 예

[이름] 위치이동 인터페이스

[기능]  
절대위치로 이동한다  
상대위치만큼 이동한다

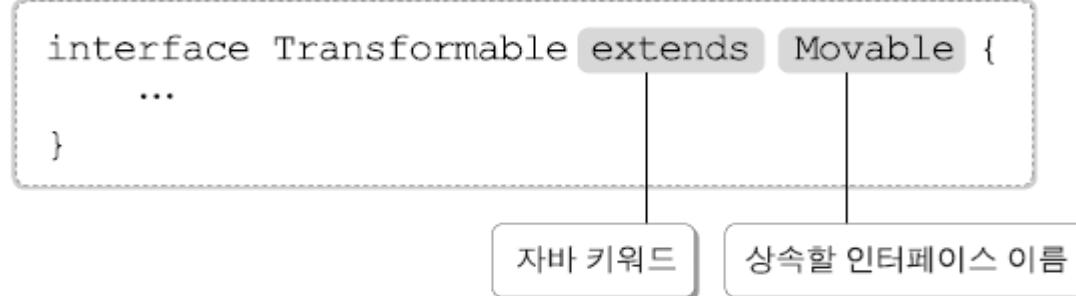
[이름] 변환 인터페이스

[기능]  
절대위치로 이동한다  
상대위치만큼 이동한다  
크기를 변경한다

# 인터페이스

## ◆ 인터페이스의 상속

- 다른 인터페이스를 상속 받는 인터페이스의 선언 방법



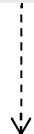
# 인터페이스

## ◆ 인터페이스의 상속

- 위치이동 인터페이스를 상속하는 변환 인터페이스

위치이동 인터페이스

```
1 interface Movable {  
2     void moveTo(int x, int y);           // 절대 위치로 이동한다  
3     void moveBy(int xoffset, int yoffset); // 상대 위치만큼 이동한다  
4 }
```



변환 인터페이스

```
1 interface Transformable extends Movable {  
2     void resize(int width, int height); // 크기를 변경한다  
3 }
```

# 인터페이스

## ◆ 인터페이스의 상속

-변환 인터페이스를 구현하는 사각형 클래스

```
1  class Rectangle implements Transformable {  
2      int x, y, width, height;  
3      Rectangle(int x, int y, int width, int height) {  
4          this.x = x;  
5          this.y = y;  
6          this.width = width;  
7          this.height = height;  
8      }  
9      public void resize(int width, int height) {  
10         this.width = width;  
11         this.height = height;  
12     }  
13     public void moveTo(int x, int y) {  
14         this.x = x;  
15         this.y = y;  
16     }  
17     public void moveBy(int x0ffset, int y0ffset) {  
18         this.x += x0ffset;  
19         this.y += y0ffset;  
20     }  
21 }
```

implements 절

Transformable 인터페이스의 메소드를 구현합니다.

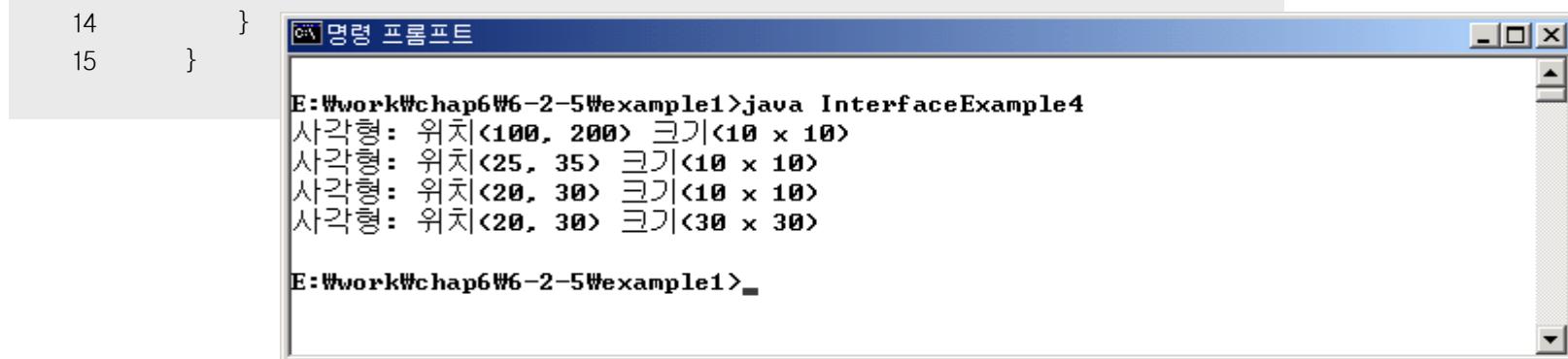
Movable 인터페이스의 메소드를 구현합니다.

# 인터페이스

## ◆ 인터페이스의 상속

-Rectangle 클래스를 사용하는 프로그램

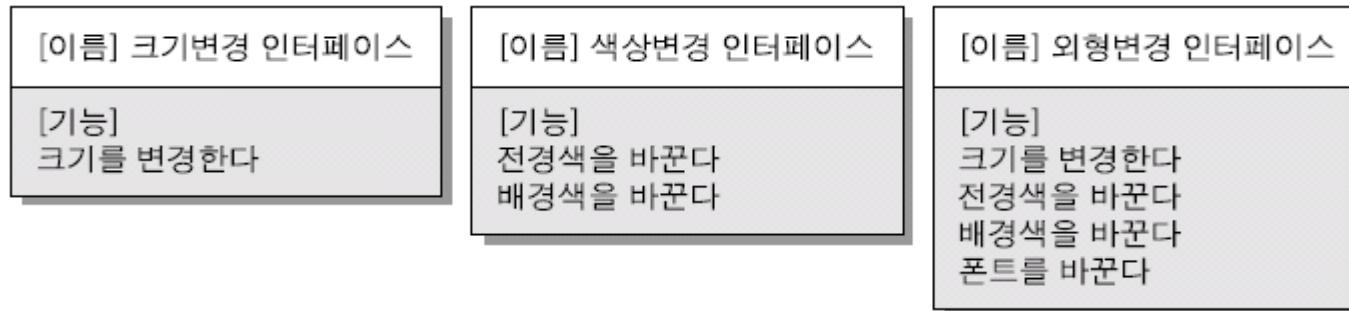
```
1  class InterfaceExample4 {
2      public static void main(String args[]) {
3          Rectangle obj = new Rectangle(100, 200, 10, 10);
4          printRectangle(obj);
5          obj.moveTo(25, 35);
6          printRectangle(obj);
7          obj.moveBy(-5, -5);
8          printRectangle(obj);
9          obj.resize(30, 30);
10         printRectangle(obj);
11     }
12     static void printRectangle(Rectangle obj) {
13         System.out.printf("사각형: 위치(%d, %d) 크기(%d x %d) %n",
14             obj.x, obj.y, obj.width, obj.height);
15     }
}
```



# 인터페이스 – 여러 개의 인터페이스 상속

## ◆ 인터페이스의 다중 상속

– 인터페이스의 다중 상속이 필요한 경우의 예



# 인터페이스

## ◆ 인터페이스의 다중 상속

-다중 상속을 하는 인터페이스의 선언 방법

```
interface Changeable extends Resizable, Colorable {  
    ...  
}
```

자바 키워드

상속할 인터페이스 이름

# 인터페이스

## ◆ 인터페이스의 다중 상속

- 두 개의 인터페이스를 동시에 상속받는 외형변경 인터페이스

크기변경 인터페이스 선언

```
1 interface Resizable {  
2     void resize(int width, int height);  
3 }
```

색상변경 인터페이스 선언

```
1 interface Colorable {  
2     void setForeground(String color);  
3     void setBackground(String color);  
4 }
```

외형변경 인터페이스 선언

```
1 interface Changeable extends Resizable, Colorable {  
2     void setFont(String font);  
3 }
```

# 인터페이스

## ◆ 인터페이스의 다중 상속

- 외형변경 인터페이스를 구현하는 라벨 클래스

```
1  class Label implements Changeable {
2      String text;
3      int width, height;
4      String foreground, background;
5      String font;
6      Label(String text, int width, int height, String foreground, String background, String font) {
7          this.text = text;
8          this.width = width;
9          this.height = height;
10         this.foreground = foreground;
11         this.background = background;
12         this.font = font;
13     }
14     public void resize(int width, int height) {
15         this.width = width;
16         this.height = height;
17     }
18     public void setForeground(String color) {
19         this.foreground = color;
20     }
21     public void setBackground(String color) {
22         this.background = color;
23     }
24     public void setFont(String font) {
25         this.font = font;
26     }
27 }
```

Resizable 인터페이스의 메소드를 구현

Colorable 인터페이스의 메소드를 구현

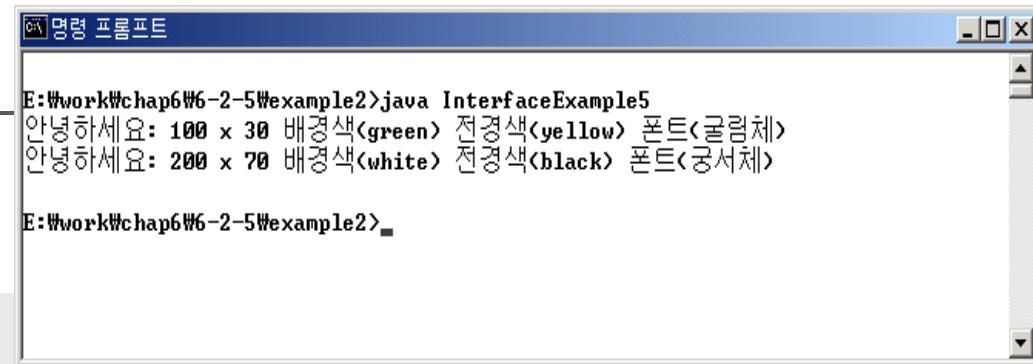
Changeable 인터페이스의 메소드를 구현

# 인터페이스

## ◆ 인터페이스의 다중 상속

-라벨 클래스를 사용하는 프로그램

```
1  class InterfaceExample5 {  
2      public static void main(String args[]) {  
3          Label obj = new Label("안녕하세요", 100, 30, "yellow", "green", "굴림체");  
4          printLabel(obj);  
5          obj.resize(200, 70);  
6          obj.setForeground("black");  
7          obj.setBackground("white");  
8          obj.setFont("궁서체");  
9          printLabel(obj);  
10     }  
11     static void printLabel(Label obj) {  
12         System.out.printf(  
13             "%s: %d x %d 배경색(%s) 전경색(%s) 폰트(%s) %n",  
14             obj.text, obj.width, obj.height,  
15             obj.background, obj.foreground, obj.font);  
16     }  
17 }
```



# 하위클래스? 추상 클래스? 인터페이스?

---

- ◆ 어떤 클래스가 다른 어떤 클래스에 대해서도  
‘A는 B다’ 테스트를 통과할 수 없다면 그냥 클래스를 만듭니다.
- ◆ “더 구체적인 클래스”를 만들고 싶다면 하위클래스를 만듭니다.
- ◆ 하위클래스에서 사용할 템플릿(template)을 정의하고 싶다면,  
그리고 구현 코드가 조금이라도 있으면 추상 클래스를 사용합니다.
- ◆ 상속 트리에서의 위치에 상관없이 어떤 클래스의 역할을 정의하고 싶다면  
인터페이스를 사용하면 됩니다.

# 하위클래스? 추상 클래스? 인터페이스?

---

- 객체지향의 4가지 특성을 실제로 구현함에 있어서 반드시 알아야 할 것이 추상 클래스(Abstract Class)와 인터페이스(Interface)

## ◆ 추상클래스

- 추상 클래스는 클래스의 명칭과 메서드는 있으나 메서드의 처리 내용은 없다.
- 상속을 통해서 메서드가 구체화(Implementation)
- 추상 클래스를 상속받은 하위 클래스에서는 추상적인 기능 구현

## ◆ 인터페이스

- 인터페이스는 상수와 추상 메서드만을 가진다.
- 클래스는 하나의 상위 클래스로서만 상속받을 수 있지만, 인터페이스는 여러 개의 인터페이스로부터 상속받을 수 있기 때문에 다중 상속의 기능 제공

# 형변환 - Casting

---

## ◆ 객체들의 형 변환

- 상속관계의 객체들에 한함

## ◆ 큰 메모리(**하위클래스**) ->작은 메모리(**상위클래스**)

- 묵시적으로 가능
- Upcasting

## ◆ 작은 메모리(**상위클래스**) -> 큰 메모리(**하위클래스**)

- 묵시적으로 불가능
- Casting 연산자인 ()를 사용해서 형을 명시해야 함
- Downcasting

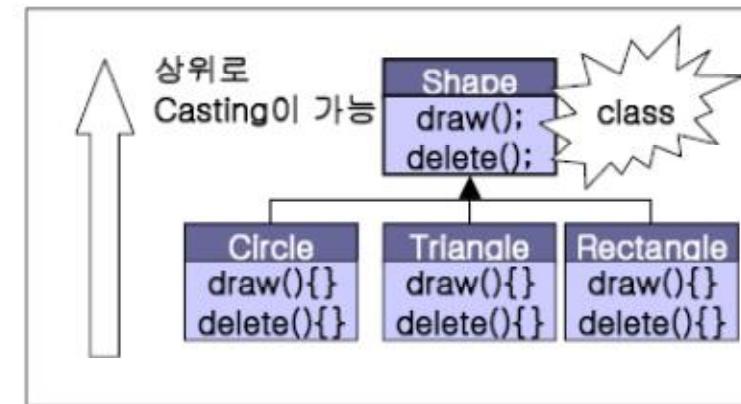
## ◆ 기본데이터 타입의 형 변환은 별개

# Casting - Upcasting

- ◆ 상위클래스로의 형 변환
- ◆ 하위클래스(Super Class) -> 상위클래스(Sub Class)
- ◆ 컴파일러에 의해 자동변환
- ◆ 상속계층의 측면고려
- ◆ 하위클래스는 상위 클래스의 서브타입이다.

Upcasting의 예

```
class Shape {  
}  
  
class Circle extends Shape {  
}  
  
Public class InheritanceTest {  
    public static void main(String[] args) {  
        //Upcasting  
        Shape shape = new Circle();  
    }  
}
```

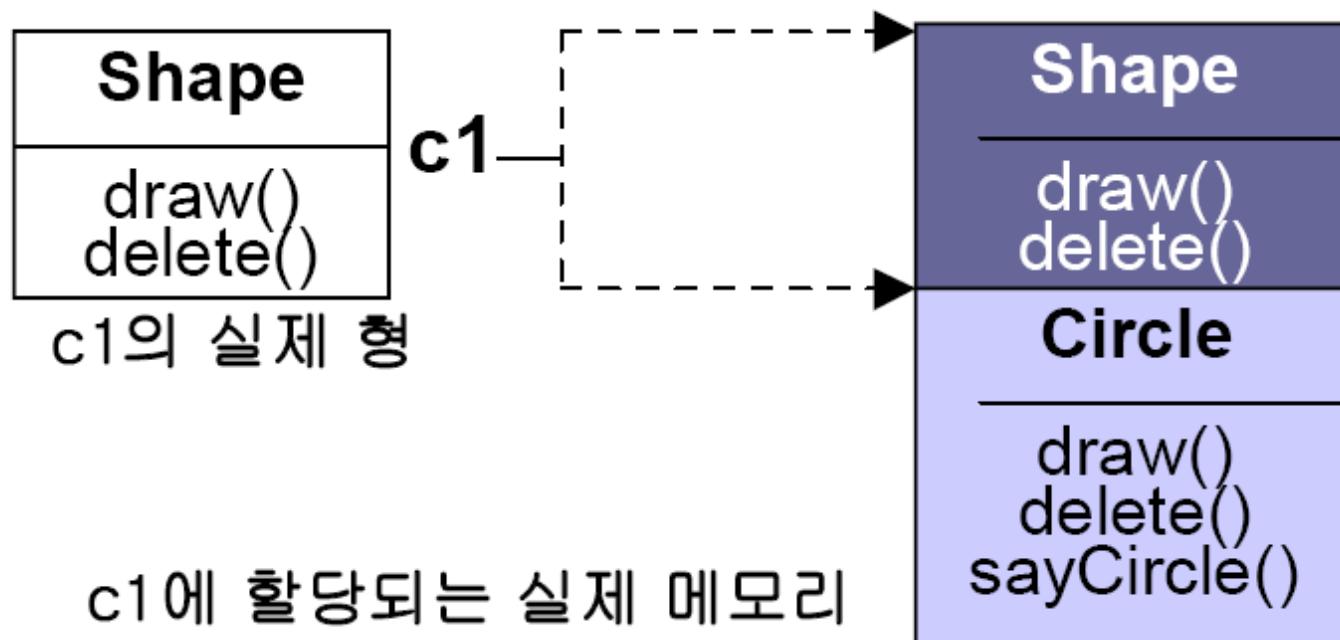


# Casting - Upcasting

## ◆ Upcasting 시 메모리 구조

- Shape c1 = new Circle();

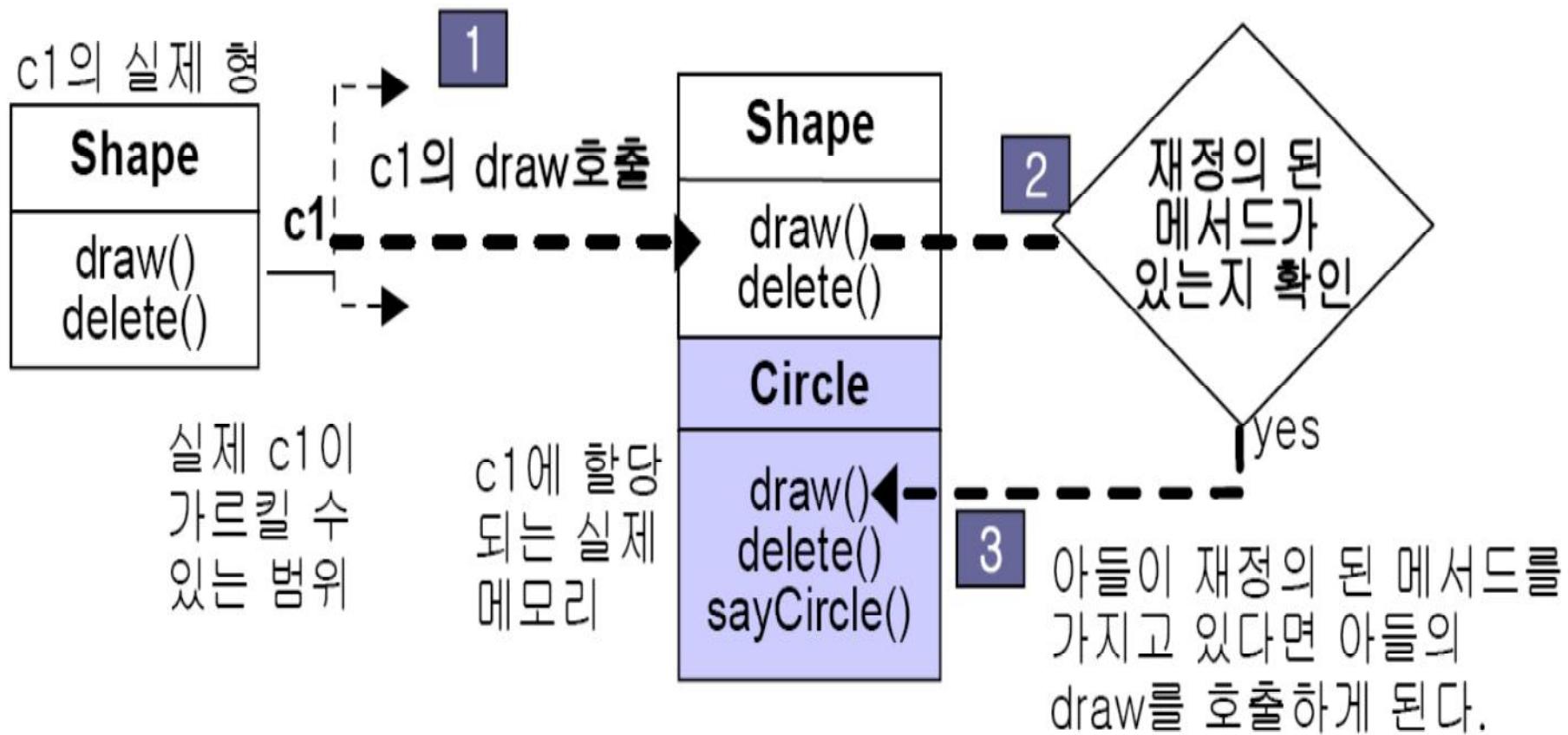
실제 c1이 가르킬 수 있는 범위



# Casting - Upcasting

## ◆ Upcasting 시의 메서드 사용법

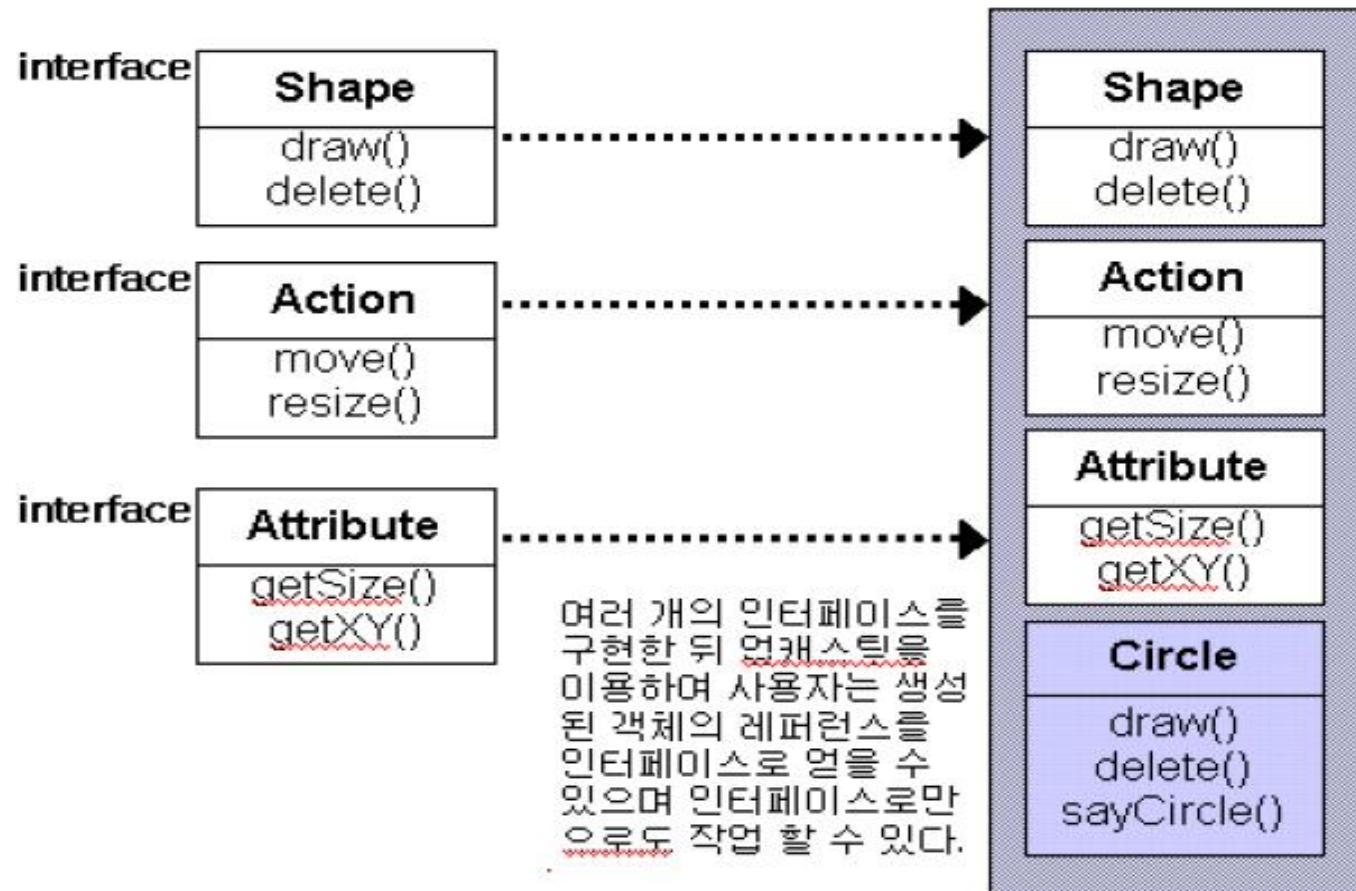
- 상위클래스에 선언된 메서드만 사용가능
- 하위클래스에서 재정의된 메서드가 호출



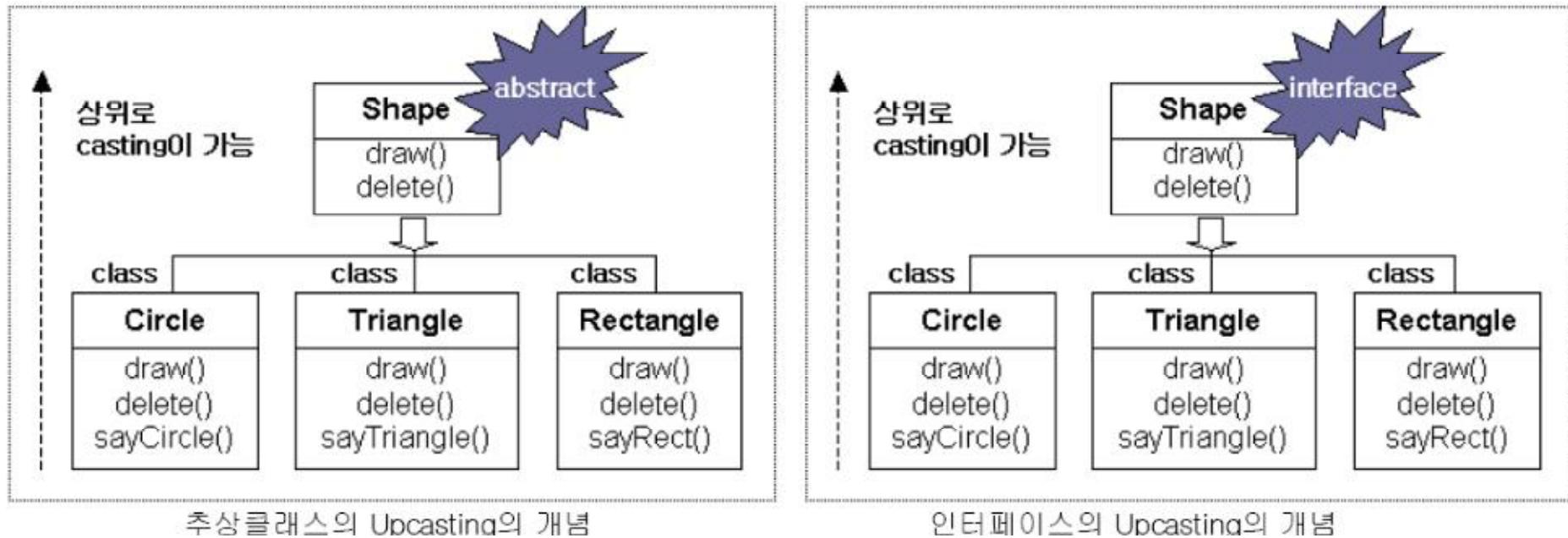
# Casting - 추상클래스와 Interface의 Upcasting

## ◆ 자체적으로 객체 생성이 불가능

- Upcasting 기법을 통해 객체 생성
- 가상 메서드를 통한 객체 은닉 구현



# Casting - Upcasting



# Casting - Downcasting

---

◆ 하위클래스로의 형 변환

◆ 상위클래스(Super Class) -> 하위클래스(Sub Class)

◆ 원칙적으로 불가능

(상위 클래스는 하위 클래스의 서브타입이 아니다.)

◆ Downcasting의 기법을 제공

▪ 명시적인 캐스팅

- 컴파일러에게 지금 이 객체는 타입정보면 볼 때는 서브타입이 아니지만, 실제 메모리에 담긴 놈은 서브타입인 놈이 담겨있을 거니까 여러 발생시키지 말고 그냥 서브타입 인 것처럼 쓰라고 지시하는 과정

# Casting- instanceof

- ◆ `instanceof` 키워드는 좌우의 객체, 클래스가 서로 같은 계층에 있지 않을 경우 컴파일 에러를 발생시킨다.
- ◆ 좌측의 객체가 우측의 클래스로 캐스팅 될 수 있으면 `true` 리턴

```
class Element { int atomicNumber; }
class Point { int x, y; }
class InstanceofTest {
    public static void main(String[] args) {
        Point p = new Point();
        Element e = new Element();
        if (e instanceof Point) { // compile-time error
            System.out.println("I get your point!");
            p = (Point)e; // compile-time error
        }
    }
}
```

```
InstanceofTest.java:8: inconvertible types
found   : Element
required: Point
    if (e instanceof Point) { // compile-time error
        ^

```

```
InstanceofTest.java:10: inconvertible types
found   : Element
required: Point
    p = (Point)e; // compile-time error
        ^

```

```
2 errors
```

# **접근 제어자와 주요 키워드의 이해**

# 접근 제어자(Access Modifier-한정자)와 변수의 유효 범위

---

## ◆ 변수의 유효 범위와 형식

- 정의된 변수가 프로그램 내에서 참조 가능한 범위를 변수의 유효 범위(scope rule)라고 한다.
- 그러나 접근 한정자에 의해서 범위가 바뀔 수 있다.
- 멤버 변수
  - 클래스 내에 메소드 밖에 선언된 변수를 가리키며 모든 멤버 변수들은 그 클래스 전체에서 유효한 범위를 갖음
- 메소드 지역 변수와 매개변수
  - 메소드 내에서만 사용 가능한 변수
- 예외처리기 매개 변수(exception handler parameter):
  - 예외 처리 기능을 하는 catch 절 내에서 사용 가능한 변수
- 클래스의 정의는 클래스의 선언을 이용해 표현한다.

# 접근 제어자(Access Modifier-한정자)와 변수의 유효 범위

## ◆ 변수의 유효 범위와 형식

- 접근 한정자는 클래스, 멤버 변수, 메소드에 접근을 제한할 수 있는 것으로 다른 클래스로부터 정보를 보호, 은폐, 캡슐화 등이 가능하다.

```
[클래스 접근한정자] class 클래스명 [extends 슈퍼클래스명]
                                [implements 인터페이스명] {
    ...
    [필드 접근한정자] 멤버 변수;
    [생성자 접근한정자][생성자];
    [메소드 접근한정자] 메소드;
    ...
}
```

# 접근 제어자(Access Modifier-한정자)와 변수의 유효 범위

## ◆ 은닉(Encapsulation)

- 은닉은 외부 사용자로부터 내부의 데이터 등의 구조를 보호하기 위하여 접근 금지 등을 하게하는 것을 말한다.
- 예를 들면 private 한정자는 해당 멤버의 참조 범위를 자신의 클래스 내로 한정시킨다.

```
public class Date {  
    public int day;  
    public int month;  
    public int year;  
}  
public class ProcDate {  
....  
Date kkk = new Date();  
kkk.day = 25;  
....  
}
```

public 접근 한정자사용

```
public class Date {  
    private int day;  
    private int month;  
    private int year;  
    public void setDay(int d) {  
        day = d;  
    }  
}  
public class ProcDate {  
....  
Date kkk = new Date();  
kkk.setDay(25);  
....  
}
```

private 접근 한정자사용

# 접근 제어자(Access Modifier-한정자)와 변수의 유효 범위

---

## ◆ 접근한정자의 종류

- 접근한정자의 종류

- 클래스 접근 한정자, 필드 접근 한정자, 생성자 접근 한정자 및 메소드 접근 한정자 등이 있다.

- 클래스 접근 한정자(class modifier) :

- 클래스 접근 한정자는 클래스에 대한 정보를 제공하는 예약어
  - 클래스 접근 한정자의 종류에는 public, abstract, final이 있으며  
public은 abstract, final과 같이 사용 가능하지만 abstract과 final은 같이 사용될 수 없다.

- 필드 접근한정자(field modifier) :

- 필드 접근한정자는 멤버 변수에 관하여 정보를 제공하는 예약어
  - 필드 접근 한정자의 종류에는 public, private, protected, static, final, transient, volatile이 있음

# 접근 제어자(Access Modifier-한정자)와 변수의 유효 범위

---

## ◆ 접근한정자의 종류

- 생성자 접근 한정자(constructor modifier) :
  - 생성자 접근 한정자는 다른 클래스로부터 정보를 보호, 은폐, 캡슐화를 제공하는 예약어
  - 생성자 접근 한정자의 종류에는 public, private, protected, default가 있다.
  
- 메소드 접근한정자(method modifier)
  - 메소드 접근 한정자는 메소드에 관하여 정보를 제공하는 예약어이다.
  - 메소드 접근 한정자의 종류에는 public, private, protected, static, final, abstract, synchronized가 있다.

# 접근 제어자(Access Modifier-한정자)와 변수의 유효 범위

## ◆ 클래스에서의 멤버변수, 멤버 메소드 간의 접근 한정 관계

- 은닉은 외부 사용자로부터 내부의 데이터 등의 구조를 보호하기 위하여 접근 금지 등을 하게하는 것을 말한다.
- 예를 들면 `private` 한정자는 해당 멤버의 참조 범위를 자신의 클래스 내로 한정시킨다.
- 클래스에서의 멤버변수, 멤버 메소드 간의 접근 한정 관계는 다음과 같다.

접근 한정자	클래스 내부	같은 패키지내의 클래스	서브 클래스	다른 모든 클래스
<code>public</code>	○	○	○	○
<code>private</code>	○	×	×	×
<code>protected</code>	○	○	○	×
<code>default (공백) 또는 package</code>	○	○	×	×

( ○: 접근가능 ×: 접근 불가능 )

# **예외(EXCEPTION)**

# 예외

---

## ◆ 예외처리에 대한 필요성과 이해

- 자바에서 프로그램의 실행하는 도중에 예외가 발생하면 발생된 그 시점에서 프로그램이 바로 종료가 된다.
- 때에 따라서는 예외가 발생 했을 때 프로그램을 종료시키는 것이 바른 판단일 수도 있다.
- 하지만 가벼운 예외이거나 예상을 하고 있었던 예외라면 프로그램을 종료시키는 것이 조금은 가혹(?)하다고 느껴진다.
- 그래서 '예외처리'라는 수단(mechanism)이 제안되었고 예외 처리를 통해 우선 프로그램의 비 정상적인 종료를 막고 발생한 예외에 대한 처리로 정상적인 프로그램을 계속 진행할 수 있도록 하는 것이 예외처리의 필요성이라 할 수 있다.

# 예외

---

## 예외란

- 실행 시에 발생하는 에러 이벤트
- try block에서 발생
- 클래스의 형태로 표현됨
- 발생하는 이벤트는 객체가 됨
- Exception클래스를 상속 받아 새로운 예외 클래스 생성 가능

# 예외- 유형

---

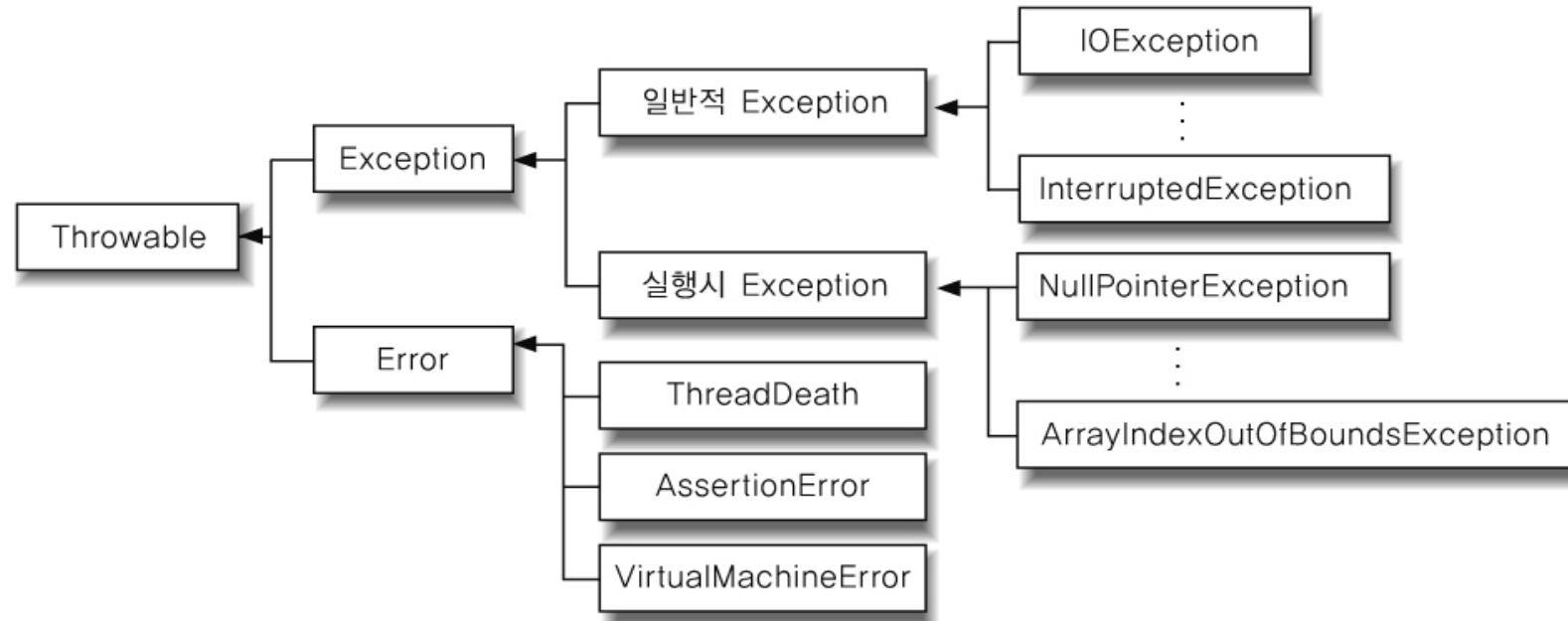
## ◆ RuntimeException

- 실행 시에 발생하는 예외 이벤트(Event)
- 시스템 상의 문제로 발생
- 실행해 봐야 알 수 있음

## ◆ RuntimeException 이외의 예외

- 컴파일 시에 발생하는 에러 이벤트
- 컴파일러가 문법적 오류로 간주해 발생
- 컴파일 시에 알 수 있음
- 자바 개발자들이 에러가 자주 발생하는 메서드에 throws 처리

# 예외 - 예외의 종류



[그림 6-5] 예외의 종류와 구조

[표 6-2] 오류의 구분

오류구분	설명
예외(Exception)	가벼운 오류이며 프로그램적으로 처리한다.
오류(Error)	치명적인 오류이며 JVM에 의존하여 처리한다.

# 예외

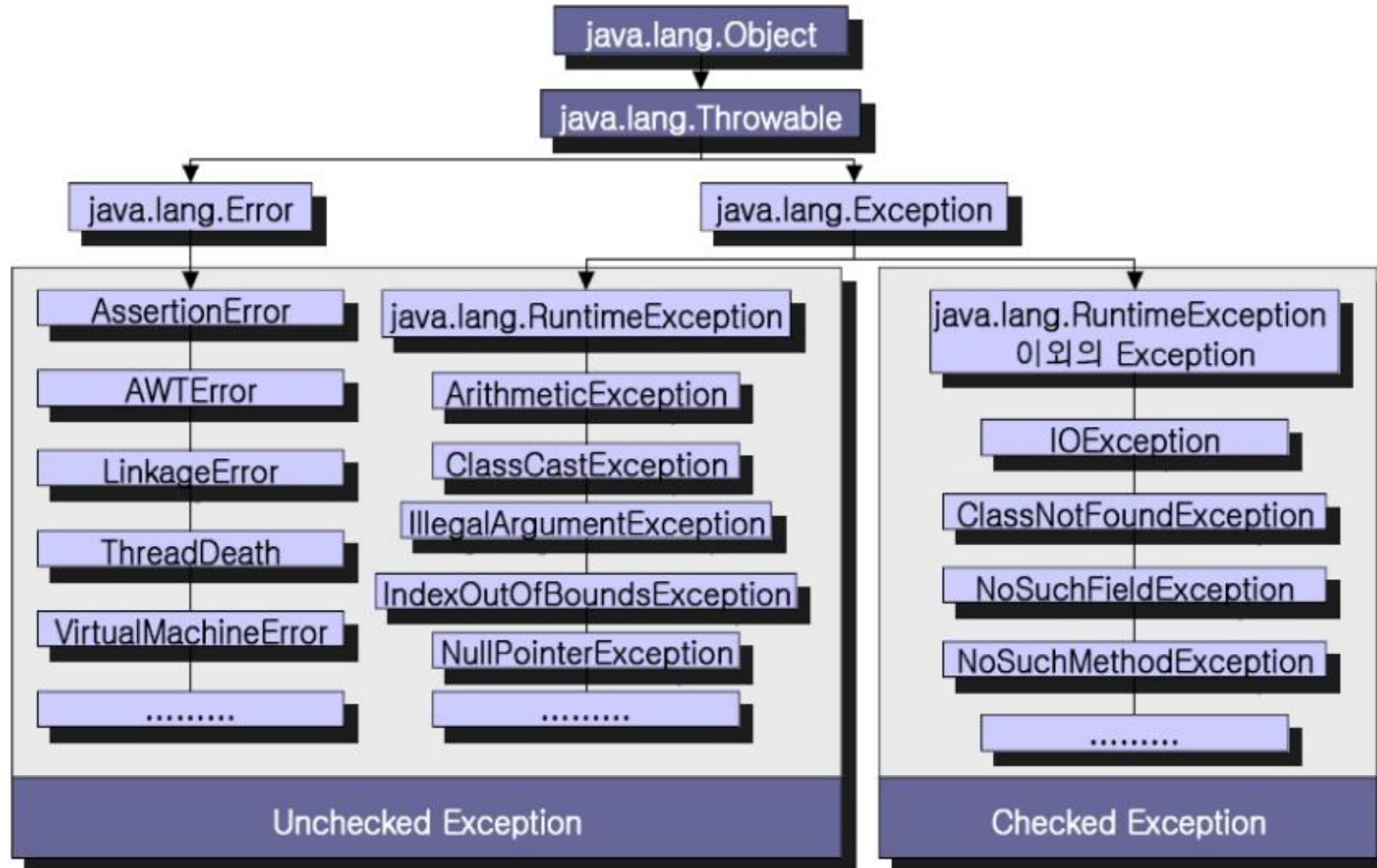
---

## ◆ 컴파일 시의 사전 검사 여부

- Unchecked Exception : 컴파일러가 예외 처리 여부를 검사하지 않음
  - 시스템 정의 예외
  - Error와 RuntimeException
- Checked Exception : 컴파일러가 예외 처리 여부를 검사함
  - 사용자 정의 예외  
(자바 개발자가 클래스 라이브러리에 정의)
  - RuntimeException 이외의 예외
- 예외의 복원 가능 여부
  - Error : 불가능 (java.lang.Error의 하부 클래스로 선언됨)
  - Exception : 가능 (java.lang.Exception의 하부 클래스로 선언됨)

# 예외 - 상속 계층도

## ◆ 자바 클래스 라이브러리에 정의된 예외 클래스들의 상속도



# 예외 - 여러가지 클래스

## ◆ RuntimeException의 하부에 정의된 예외클래스들

클래스 이름	예외 조건
ArithmaticException	정수 값을 0으로 나누려 하는 등의 유효하지 않는 계산 조건을 사용하는 경우.
IndexOutOfBoundsException	객체의 범위를 벗어난 색인을 사용하려 하는 경우, 배열이나 String 객체 또는 Vector 객체가 이에 해당한다.
NegativeArraySizeException	음수 차원의 배열을 정의하려 하는 경우.
NullPointerException	Null을 포함하는 객체 변수를 사용하려 하는 경우, 적절한 작업(예를 들어, 메서드를 호출하거나 데이터 멤버를 액세스 하는)을 하기 위해서는 변수가 객체를 참조해야 한다.
ArrayStoreException	배열 유형이 허락하지 않는 객체를 배열에 저장하려 하는 경우.
ClassCastException	객체를 부적절한 유형으로 형 변환하려 하는 경우. 즉, 객체가 지정한 클래스도 아니고, 지정한 클래스의 상위 클래스나 하위 클래스도 아닌 경우.
IllegalArgumentException	메서드에 매개변수 유형이 일치하지 않는 인수를 전달한 경우.
SecurityException	프로그램이 보안에 위배되는 부적절한 작업을 수행하려 하는 경우. 애플리케이션에서 로컬 머신의 파일을 읽으려 하는 경우 등이 이에 해당한다.
IllegalMonitorStateException	스레드가 스레드에 속하지 않는 객체를 모니터하려고 기다리는 경우.
IllegalStateException	적절하지 않은 때에 메서드를 호출하는 경우.
UnsupportedOperationException	객체가 지원하지 않는 작업을 수행하도록 요구하는 경우.

# 예외 처리와 Exception 클래스

## ◆ Exception 클래스의 하위 클래스의 종류 및 예외상황

클래스명	예외상황
<b>RuntimeException</b>	실행 시간 예외가 발생할 경우
<b>CloneNotSupportedException</b>	객체의 복제가 지원되지 않은 상황에서의 복제 시도의 경우
<b>IllegalAccessException</b>	클래스에 대한 부정 접근의 경우
<b>InstantiationException</b>	추상 클래스나 인터페이스로부터 객체 생성하려 할 경우
<b>InterruptedException</b>	쓰레드나 인터럽트 되었을 경우
<b>NoSuchMethodException</b>	메소드가 존재하지 않을 경우
<b>ClassNotFoundException</b>	클래스가 존재하지 않을 경우
<b>IOException</b>	입출력시에 파일 등이 존재하지 않을 경우

# 예외

---

## ◆ RuntimeException 클래스의 하위 클래스의 종류 및 예외상황

클래스명	예외상황
ArithmeticException	0으로 나눌 때 산술적인 예외의 경우
NegativeArraySizeException	배열의 크기를 음수로 지정할 경우
NullPointerException	null 객체의 메소드나 멤버 변수에 접근할 경우
ArrayStoreException	배열의 원소에 잘못된 형의 객체를 배정하였을 경우
IndexOutOfBoundsException	배열, 스트링, 벡터 등과 같이 인덱스를 사용하는 객체에서 인덱스의 범위를 벗어나는 경우
SecurityException	보안으로 인하여 메소드를 실행할 수 없는 경우

# 예외

## ◆ 컴파일 시에 예외 발생

- 자바 컴파일러가 `CompileTimeException.java` 파일을 컴파일 중에 URL 클래스 발견
- 클래스 라이브러리의 `java.net` 패키지의 `URL.class`를 로딩
- `URL` 클래스의 객체 생성 중에 `throws`로 `MalformedURLException`의 예외 처리가 미뤄진 것을 발견
- `URL` 클래스의 객체를 생성하려는 `CompileTimeException.java` 파일에 예외 처리가 안된 것을 확인한 후 예외 발생

### CompileTimeException.java (컴파일 시에 발생하는 예외)

```
import java.net.*;  
public class CompileTimeException {  
    public static void main(String args[]) {  
        URL url = new URL("http://www.yahoo.co.kr");  
    }  
}
```

c:\>javac CompileTimeException.java  
CompileTimeException .java:5: unreported exception java.net.MalformedURLException;  
must be caught or declared to be thrown  
 URL url = new URL("http://www.yahoo.co.kr");  
 ^  
1 error RuntimeExceptionTest.java:9

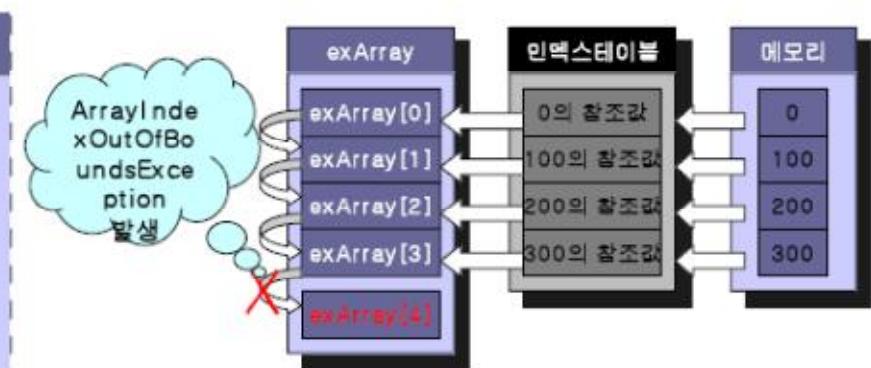
# 예외

## ◆ 컴파일은 되나 배열의 오류가 발생

- 선언된 배열의 범위를 넘은 요소에 접근
- ArrayIndexOutOfBoundsException 발생

### 실행 시에 발생하는 예외

```
public class RuntimeExceptionTest{  
    public static void main(String args[]) {  
        int[] exArray = new int[4];  
        exArray[0] = 0;  
        exArray[1] = 100;  
        exArray[2] = 200;  
        exArray[3] = 300;  
        for(int i=0; i<exArray.length+1; i++)  
            System.out.println("exArray["+i+"]=" + exArray[i]);  
    }  
}
```



ArrayIndexOutOfBoundsException의 예

```
C:\Wexception>javac RuntimeExceptionTest.java  
C:\Wexception>java RuntimeExceptionTest  
exArray[0]=0  
exArray[1]=100  
exArray[2]=200  
exArray[3]=300  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException  
at RuntimeExceptionTest.main
```

# 예외 처리

---

- ◆ try에서 발생된 예외를 처리하는 것
- ◆ catch block에서 처리됨
- ◆ **Checked Exception**
  - 컴파일 시에 컴파일러가 처리기 여부를 검사
  - 비교적 처리가 용이
- ◆ **RuntimeException**
  - 발생 전에는 알 수가 없음
  - 모든 발생 가능 상황에 대한 처리가 필요
    - 따라서 다중 catch block을 사용함
  - 비교적 처리가 어려움
  - 실행 중 예외 발생에 따른 위험
  - 프로그램의 신뢰성과 안전성 향상

# 예외

---

## ◆ try

- 예러 발생이 가능한 코드가 위치함

## ◆ catch

- try에서 발생한 에러이벤트를 처리
- 다중catch 사용시에 발생할 예외 클래스의 상속을 고려

## ◆ finally

- 프로그램 종료 전에 무조건 실행
- java.io패키지와 java.sql 패키지에서 자주 사용됨

# 예외

---

## ◆ try block에서 예외 발생

- 발생된 예외클래스의 객체 생성
- catch의 명시된 예외클래스들과 실제 발생된 예외클래스의 객체 형을 비교
- 만일 형이 같은 것이 있으면, 해당 catch block을 실행
- 만일 없다면, 그대로 프로그램 종료
- 예외의 발생유무와 관계없이 finally block은 무조건 실행

# 예외

## try, catch, finally를 이용한 예외 처리 예

```
try{
    //에러 발생 가능 코드
    throw new Exception();
} catch(에러 이벤트의 종류1 변수){
    //발생된 이벤트1의 객체가 매개변수의 형태로 catch{}으로 넘어온다.
    e.getMessage();
    e.printStackTrace();
    e.toString();
} catch(에러 이벤트의 종류2 변수){
    //발생된 이벤트2의 객체가 매개변수의 형태로 catch{}으로 넘어온다.
} catch(에러 이벤트의 종류3 변수){
    //여러 개의 catch 사용 가능
}finally{
    //무조건 실행
}
```

\* Throwable클래스의 멤버 메서드 정리

1. getMessage() : 에러 발생 클래스의 생성자에 매개 변수로  
넣어준 문자열을 출력
2. printStackTrace() : 에러가 발생된 곳의 위치를 알려줌
3. toString() : 발생된 에러의 종류[: getMessage의 내용]

# 예외

---

## ◆ throw

- 사용자가 에러 이벤트를 발생
- 발생된 이벤트는 반드시 catch로 받아야 함

```
try{  
    throw new 예외클래스명("예외클래스에 대한 설명");  
    //예외클래스를 발생  
} catch(예외클래스명 변수명){  
    //발생된 예외를 처리  
}
```

## ◆ throws

- 예외 처리 미루기
- 컴파일러에게 메서드 사용 시에 예외가 발생할 수도 있다는 것을 사전에 알림

```
메서드명() throws 발생 가능한 예외클래스{  
    //메서드의 작업 표시  
}
```

# 예외

---

## ◆ **throw**를 사용해 사용자 임의대로 예외 발생이 가능

- 컴파일러가 인식하지 못하는 예외를 사용자가 새롭게 정의해서 사용 가능
- 새로운 예외 클래스를 만들 때에는 무조건 Exception클래스를 상속 해야 함
- 컴파일러가 인식하는 예외를 사용자가 다른 예외 클래스로 바꿔 발생시키고자 할 때도 사용 가능

# 예외

## ◆ 인위적인 예ception의 발생

- 인위적인 예ception의 발생 방법
  - ✓ throw 키워드를 이용
  - ✓ 예ception 클래스를 이용하여 예ception 객체를 생성



# 예외

## ◆ 인위적인 예ception의 발생

-잔액이 부족할 때 예ception을 발생하는 Account 클래스 (완성)

```
1  class Account {  
2      String accountNo;  
3      String ownerName;  
4      int balance;  
5      Account(String accountNo, String ownerName, int balance) {  
6          this.accountNo = accountNo;  
7          this.ownerName = ownerName;  
8          this.balance = balance;  
9      }  
10     void deposit(int amount) {  
11         balance += amount;  
12     }  
13     int withdraw(int amount) throws Exception {  
14         if (balance < amount)  
15             throw new Exception("잔액이 부족합니다."),  
16         balance -= amount;  
17         return amount;  
18     }  
19 }
```

이 메소드가 발생하는 예ception의 종류를 표시하는 throws 절

# 예외

### ◆ 인위적인 익셉션의 발생

- 익셉션을 발생하는 메소드를 호출하는 프로그램

```
1 class MethodExample5 {  
2     public static void main(String args[]) {  
3         Account obj = new Account("777-777-77777777", "최대박", 10);  
4         try {  
5             int amount = obj.withdraw(100000000);  
6             System.out.println("인출액:" + amount);  
7         }  
8         catch (Exception e) {  
9             String msg = e.getMessage();  
10            System.out.println(msg);  
11        }  
12    }  
13 }
```

익셉션을 발생하는 메소드

메소드가 발생한  
익셉션을 처리합니다.

# **패키지(PACKAGE)**

# 패키지

---

## ◆ 패키지(Package)

- Java에서는 서로 관련된 클래스를 하나의 단위로 그룹화
- 계층 구조를 이루고 있음(이름만)
- 소프트웨어 재사용을 위한 매커니즘
- C/C++의 라이브러리와 동일
- 사용방법은 C/C++과 전혀 다름

## ◆ 목적

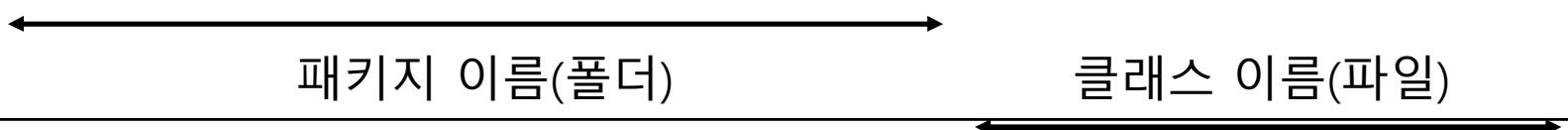
- 클래스, 인터페이스, 부 패키지 이름의 우연한 충돌 방지 및 관리 용이성
  - 2개의 클래스 혹은 인터페이스가 이름이 서로 같아도 서로 다른 이름의 패키지에 속하면 구분
  - name space 부여
- 패키지단위 접근권한지정

# 패키지

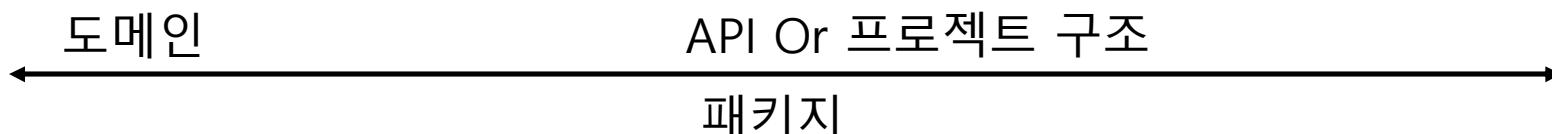
## ◆ 패키지 구성

- 패키지 이름 : 클래스 파일이 있는 폴더 구조
- 클래스 이름 : 실제 자바 api

```
com.bit.videoshop.customer.CustomerManager;
```



```
com.sun.java.awt.*;
com.sun.java.awt.Point;
com.sun.java.awt.event.ActionEvent;
com.sun.java.awt.image.renderable.RenderContext;
com.bit.videoshop.video.VideoManager;
com.bit.videoshop.customer.CustomerManager;
```

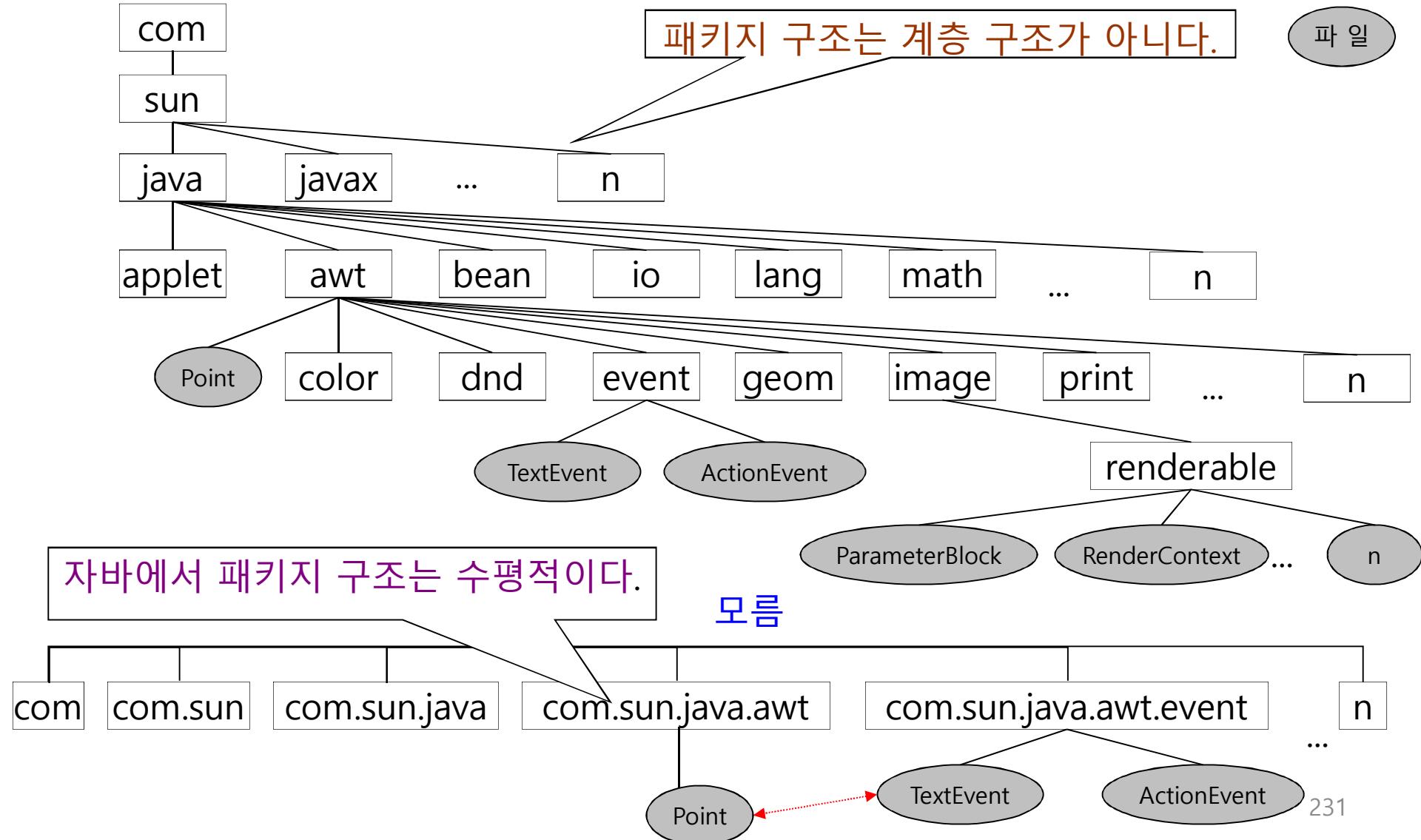


# 패키지

## ◆ 패키지 구조

풀 더

파일



## 패키지

### ◆ 패키지의 클래스와 인터페이스를 사용하기 위해서는

- 패키지 이름을 명시해야 한다 : 패키지이름. 클래스이름

```
Java.util.ArrayList vector1 = new java.util.ArrayList()
```

- 패키지 이름을 생략하고 싶다면

- import 문장을 사용하면 패키지이름을 생략하고, 클래스/인터페이스 이름만을 사용할 수 있다

- ✓ import 패키지이름.클래스이름

- ✓ import 패키지이름.\*

- 부 패키지는 import 되지 않는다.

- ✓ 별도로 import 해야 함

- ✓ import java.awt.\* 라고 했다고 해서 java.awt.color.\* 까지 import 되는 것은 아님

- ✓ 단지 java.awt 패키지 내의 클래스만 import

```
Import java.applet.Applet;  
Import java.util.*
```

### ◆ 컴파일 시 자동으로 import java.lang.\* 가 자동삽입

# 패키지

---

//PackageTest.java

```
import java.util.Date;
import java.util.Random;
// import java.util.*; // 위 2개의 import문 대신 사용할 수 있다.
// import java.sql.*; // 클래스의 중복이있어도 패키지가 다르면 별문제 없음
// import java.lang.*; // 컴파일러에 의해 자동 삽입.

class PackageTest
{
    public static void main(String[] args) {
        java.lang.System.out.println(new java.util.Date());
        System.out.println(new Date());
        System.out.println(new Random().nextInt());
    }
}
```

# 패키지

---

## ◆ 자바 원시 파일은 다음과 같은 순서로 이루어져져 있어야 한다.

Package 패키지이름; (패키지정의, 생략가능)

여러 개의 import 선언문; (생략가능)

1개 이상의 class 혹은 interface 정의

## ◆ 자바의 패키지는 디렉토리와 연관되어 있다.

- 특정 패키지에 속하도록 설정했다면 그 이름과 동일한 이름을 가진 디렉토리에 컴파일 되어야 한다.
- 패키지 선언이 없으면 해당 클래스들은 익명 패키지에 속한다.

Object, String, StringBuffer& StringBuilder  
Calender, Date, List, Map, Set

## JAVA 활용

# Java API Document

◆ 우측의 클래스를 선택하면 실제 API를 볼 수 있다.

실제 API 설명

Java™ 2 Platform  
Std. Ed. v1.4.1

Overview Package Class Use Tree Deprecated Index Help  
PREV NEXT FRAMES NO FRAMES

Java™ 2 Platform, Standard Edition, v 1.4.1  
API Specification

This document is the API specification for the Java 2 Platform, Standard Edition, version 1.4.1.

See:  
[Description](#)

### Java 2 Platform Packages

<a href="#">java.applet</a>	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
<a href="#">java.awt</a>	Contains all of the classes for creating user interfaces and for painting graphics and images.
<a href="#">java.awt.color</a>	Provides classes for color spaces.
<a href="#">java.awt.datatransfer</a>	Provides interfaces and classes for transferring data between and within applications.
<a href="#">java.awt.dnd</a>	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
<a href="#">java.awt.event</a>	Provides interfaces and classes for dealing with different types of events fired by AWT components.
<a href="#">java.awt.font</a>	Provides classes and interface relating to fonts.
<a href="#">java.awt.geom</a>	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.

# Java API Document 보는 방법

The image shows two side-by-side screenshots of Microsoft Internet Explorer displaying the Java API documentation for the `ActivationException` class.

**Left Window (ActivationException Class Page):**

- Package:** java.rmi.activation
- Class:** ActivationException
- Superclasses:** java.lang.Object → java.lang.Throwable → java.lang.Exception → java.rmi.activation.ActivationException
- Implemented Interfaces:** Serializable
- Subclasses:** UnknownGroupException, UnknownObjectException
- Description:** A general exception used by the activation interfaces.
- Since:** 1.2
- See Also:** Serialized Form

**Right Window (ActivationException Field Summary Page):**

- Field Summary:** Nested Exception to hold wrapped remote exceptions.
- Constructor Summary:**
  - `ActivationException()`: Constructs an ActivationException with no specified detail message.
  - `ActivationException(String s)`: Constructs an ActivationException with detail message, s.
  - `ActivationException(String s, Throwable ex)`: Constructs an ActivationException with detail message, s, and detail exception ex.
- Method Summary:**
  - `getCause()`: Returns the detail exception.
  - `getMessage()`: Returns the detail message, or from the detail exception if there is one.
- Inherited Methods:**
  - From `java.lang.Throwable`: fillInStackTrace, getLocalizedMessage, getStackTrace, initCause, printStackTrace, printStackTrace, setStackTrace, toString
  - From `java.lang.Object`: clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait
- Field Detail:**

# 자바 클래스의 상속 계층 구조

## ◆ Object 클래스에 대하여

- 우리가 직접 선언한 클래스도 Object 클래스를 상속받음

```
class GoodsInfo {  
    String goodsCode;  
    String name;  
}
```

extends 절이 없는  
클래스는 컴파일 과정에서  
자동으로 Object의  
서브클래스가 됩니다.

---

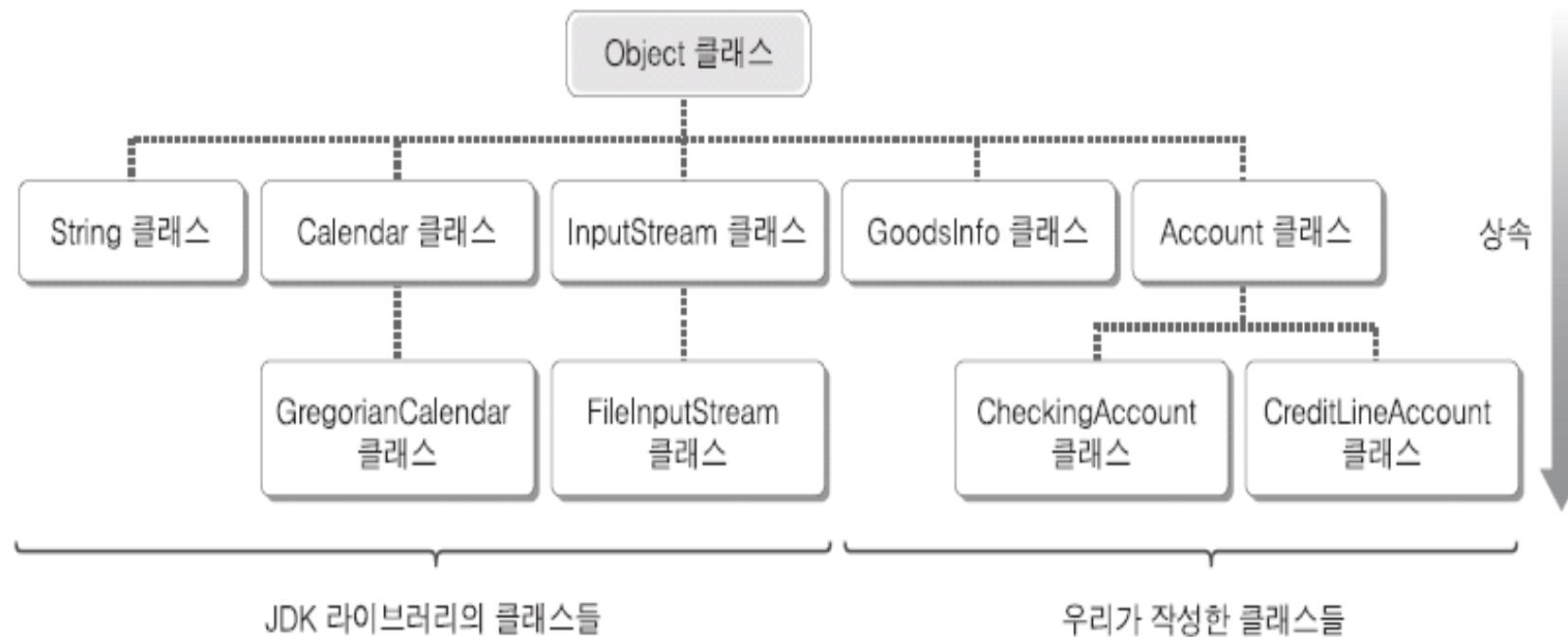
## ◆ Object 클래스에 대하여

클래스들의 공통 특성은 추출해서  
슈퍼클래스로 만듭니다.



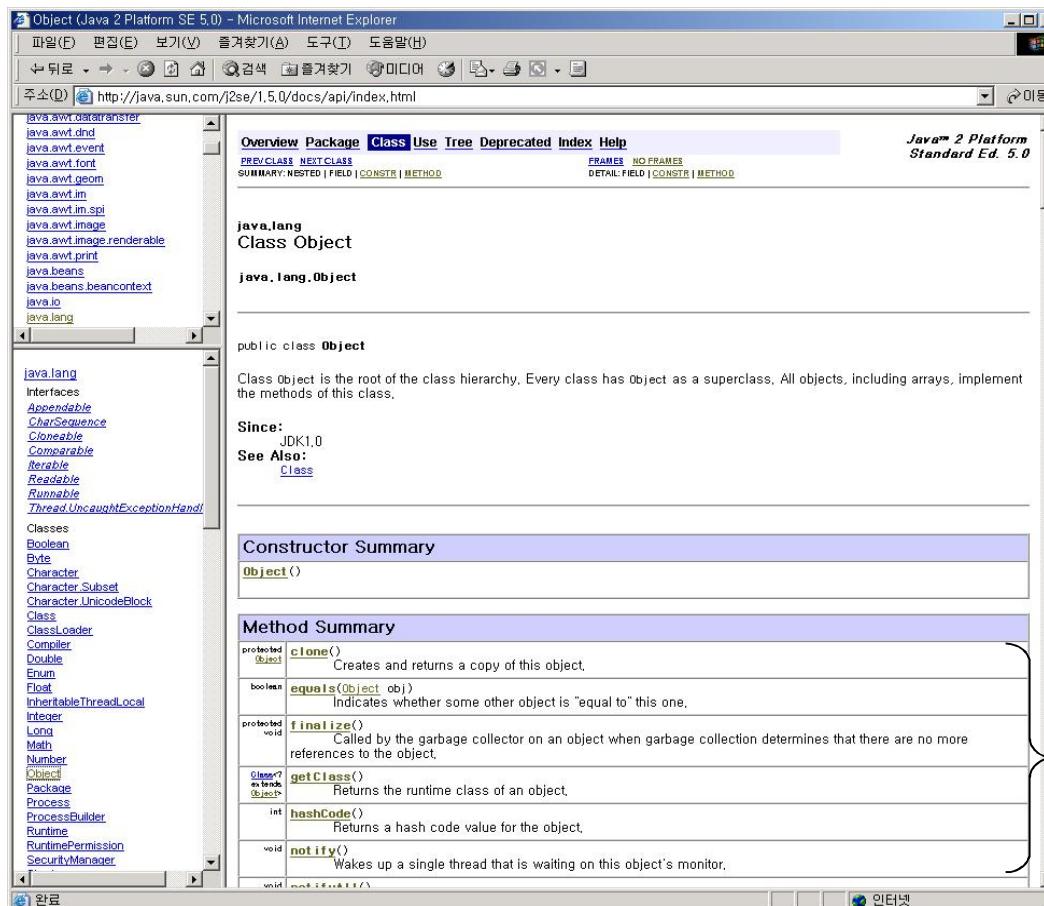
Object 클래스는 자바의 모든 클래스들의  
공통 특성을 추출해서 만든 슈퍼클래스입니다.

# 상속 계층 구조



# Object 클래스

## ◆ Object 클래스의 API 규격서



# Object 클래스- `toString`

- ◆ `toString` 메소드 : 객체가 가지고 있는 값을 문자열로 만들어서 리턴하는 메소드

- 호출 방법

```
String str = obj.toString();
```

↑  
객체가 가진 값을  
문자열로 만들어서 리턴하는 메소드

```
1 import java.io.File;  
2 class ObjectExample1 {  
3     public static void main(String args[]) {  
4         File file = new File("C:\\빼꾸기");           ----- File 객체를 생성합니다.  
5         String str = file.toString();             ----- File 객체에 대해 toString  
6         System.out.println(str);                  메소드를 호출합니다.  
7     }  
8 }
```



# Object 클래스- `toString`

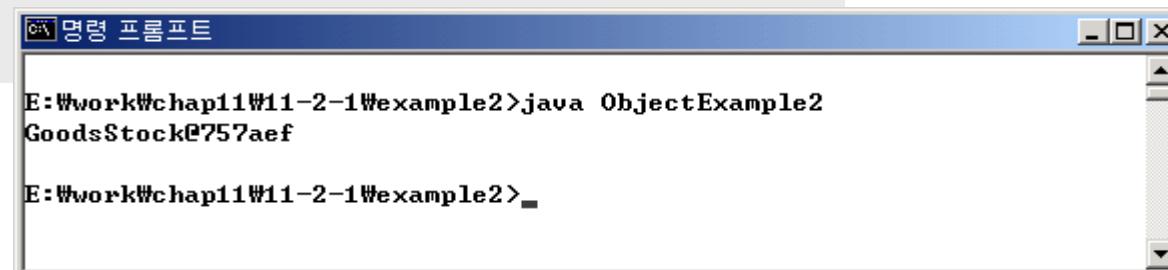
-`toString` 메소드의 또 다른 사용 예

상품 재고 클래스를 사용하는 프로그램

```
1  class ObjectExample2 {  
2      public static void main(String args[]) {  
3          GoodsStock obj = new GoodsStock("57293", 100);  
4          String str = obj.toString();  
5          System.out.println(str);  
6      }  
7  }
```

상품 재고 클래스

```
1  class GoodsStock {  
2      String goodsCode;    // 상품코드  
3      int stockNum;       // 재고수량  
4      GoodsStock(String goodsCode, int stockNum) {  
5          this.goodsCode = goodsCode;  
6          this.stockNum = stockNum;  
7      }  
8  }
```

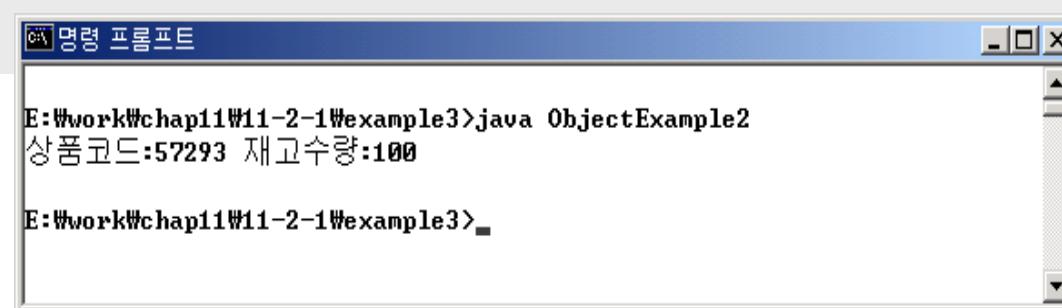


# Object 클래스- `toString`

## ◆ `toString` 메소드의 오버라이딩

상품 재고 클래스

```
1  class GoodsStock {  
2      String goodsCode;    // 상품코드  
3      int stockNum;       // 재고수량  
4      GoodsStock(String goodsCode, int stockNum) {  
5          this.goodsCode = goodsCode;  
6          this.stockNum = stockNum;  
7      }  
8      public String toString() {  
9          String str = "상품코드:" + goodsCode + " 재고수량:" + stockNum;  
10         return str;  
11     }  
12 }
```



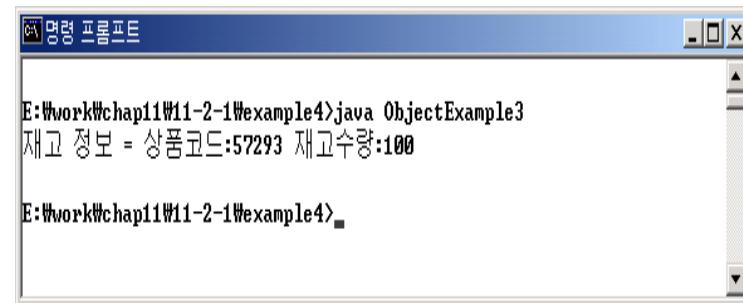
# Object 클래스- `toString`

```
File file = new File("C:\\\\독수리");
String str = "경로명:" + file;
```

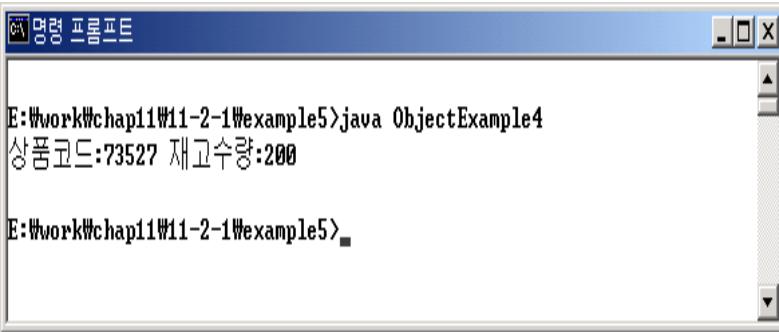
```
File file = new File("C:\\\\독수리");
String str = "경로명:" + file.toString();
```

똑같은 효과를 갖는 두 코드

```
1 class ObjectExample3 {
2     public static void main(String args[]) {
3         GoodsStock obj = new GoodsStock("57293",
4             100);
5         String str = "재고 정보 = " + obj;
6         System.out.println(str);
7     }
}
```



```
1 class ObjectExample4 {
2     public static void main(String args[]) {
3         GoodsStock obj = new GoodsStock("73527", 200);
4         System.out.println(obj);
5     }
6 }
```



# Object 클래스 - equals

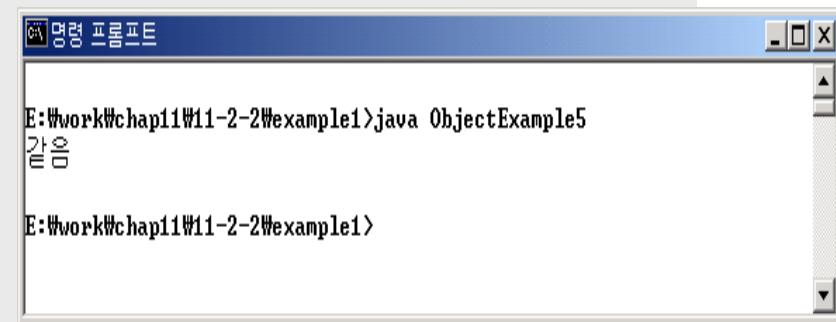
## ◆ equals 메소드

-equals 메소드 : 객체가 가지고 있는 값을 비교하는 메소드

-호출 방법

obj1.equals( obj2 )  
↑                   ↑  
이 객체와 이 객체의  
값을 비교합니다.

```
1 import java.util.GregorianCalendar;
2 class ObjectExample5 {
3     public static void main(String args[]) {
4         GregorianCalendar obj1 = new GregorianCalendar(2007, 0, 1);
5         GregorianCalendar obj2 = new GregorianCalendar(2007, 0, 1);
6         if (obj1.equals(obj2))
7             System.out.println("같음");
8         else
9             System.out.println("다름");
10    }
11 }
```

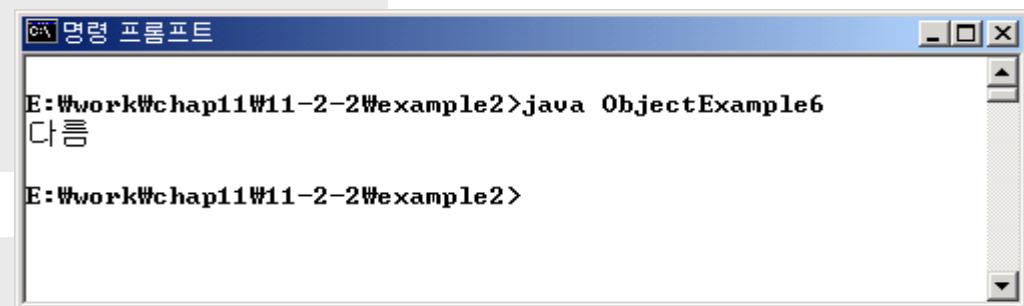


## Object 클래스 - equals

## 원 클래스를 사용하는 프로그램

원 클래스

```
1 class Circle {  
2     int radius;          // 반지름  
3     Circle(int radius) {  
4         this.radius = radius;  
5     }  
6 }
```



## Object 클래스 - equals

```
1 class Circle {  
2     int radius;          // 반지름  
3     Circle(int radius) {  
4         this.radius = radius;  
5     }  
6     public boolean equals(Object obj) { } }  
7     if (!(obj instanceof Circle)) { } } }  
8         return false; } } }  
9         Circle circle = (Circle) obj; } } }  
10        if (radius == circle.radius) { } } }  
11        return true; } } }  
12    else { } } }  
13        return false; } } }  
14    } } }  
15    //hashCode도 Override 필요 } } }  
16 }
```

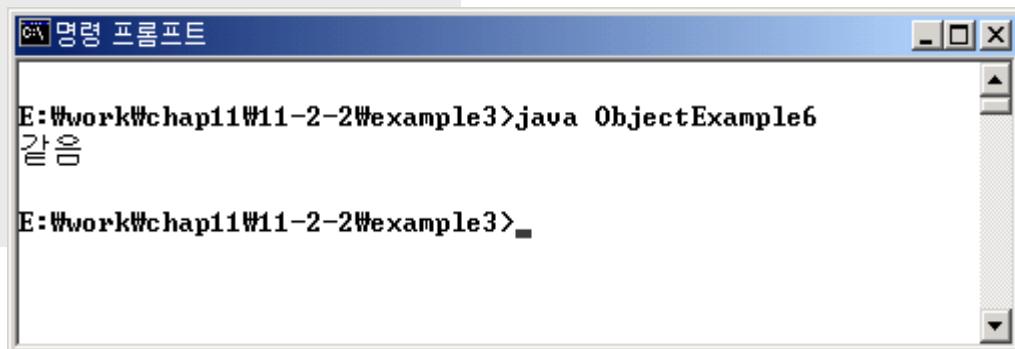
파라미터 객체가 Circle 타입인지 검사

파라미터 객체를 Circle 타입으로 캐스트 연산 수행

파라미터 객체와 객체 자신의 radius 필드 값 비교



The screenshot shows a terminal window with the following text:  
E:\work\chap11\11-2-2\example3>java ObjectExample6  
같음



# System 클래스 – 필드 및 메서드

---

◆ 프로그램 종료, 입출력, 시스템 시간 및 환경 변수 등을 읽는데 사용되는 클래스

◆ **fields**

- public static final InputStream in
- public static final PrintStream out
- public static final PrintStream err

◆ **method**

- public static void arraycopy([Object](#) src, int srcPos, [Object](#) dest, int destPos, int length)
- public static long currentTimeMillis()
- public static long nanoTime()
- public static void exit(int status)
- public static [String](#) getenv([String](#) name)
  - name : JAVA\_HOME 등
- public static [Properties](#) getProperties()
  - <https://docs.oracle.com/javase/8/docs/api/index.html>

# Class 클래스

---

## ◆ 실행중인 Java application의 클래스와 인터페이스를 나타내는 클래스

### ◆ Method

- public static [Class<?>](#) forName([String](#) className) throws [ClassNotFoundException](#)
- public [Class<?>\[\]](#) getClasses()
- public [Field\[\]](#) getFields() throws [SecurityException](#)
- public [String](#) getName()
- public [Package](#) getPackage()
- public [Class<?>\[\]](#) getInterfaces()
- public [T](#) newInstance() throws [InstantiationException](#), [IllegalAccessException](#)

# **문자열 관련 클래스**

# 문자열에 관련된 클래스들

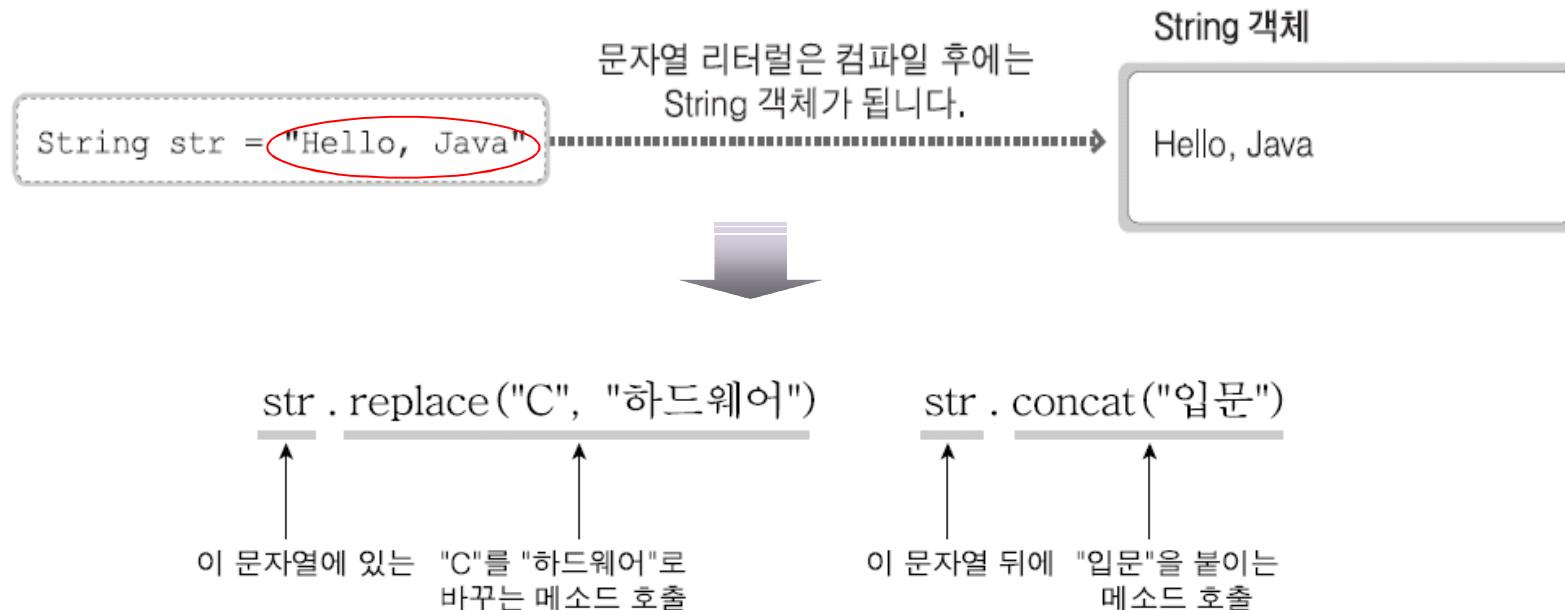
---

## ◆ 문자열 관련 클래스들

- String 클래스는 문자열 조작에 적합하지 않음
  - ✓ 너무 많은 String 객체를 만들기 때문
- 문자열 조작에 적합한 클래스 : StringBuilder 클래스, StringBuffer 클래스
  - ✓ 객체를 많이 만들지 않고 문자열 조작이 가능
- 문자열로부터 작은 단위 문자열을 추출하는 클래스 : StringTokenizer 클래스
  - ✓ "사과 배 복숭아"라는 문자열로부터 "사과", "배", "복숭아"를 추출

# 문자 관련 클래스 - String

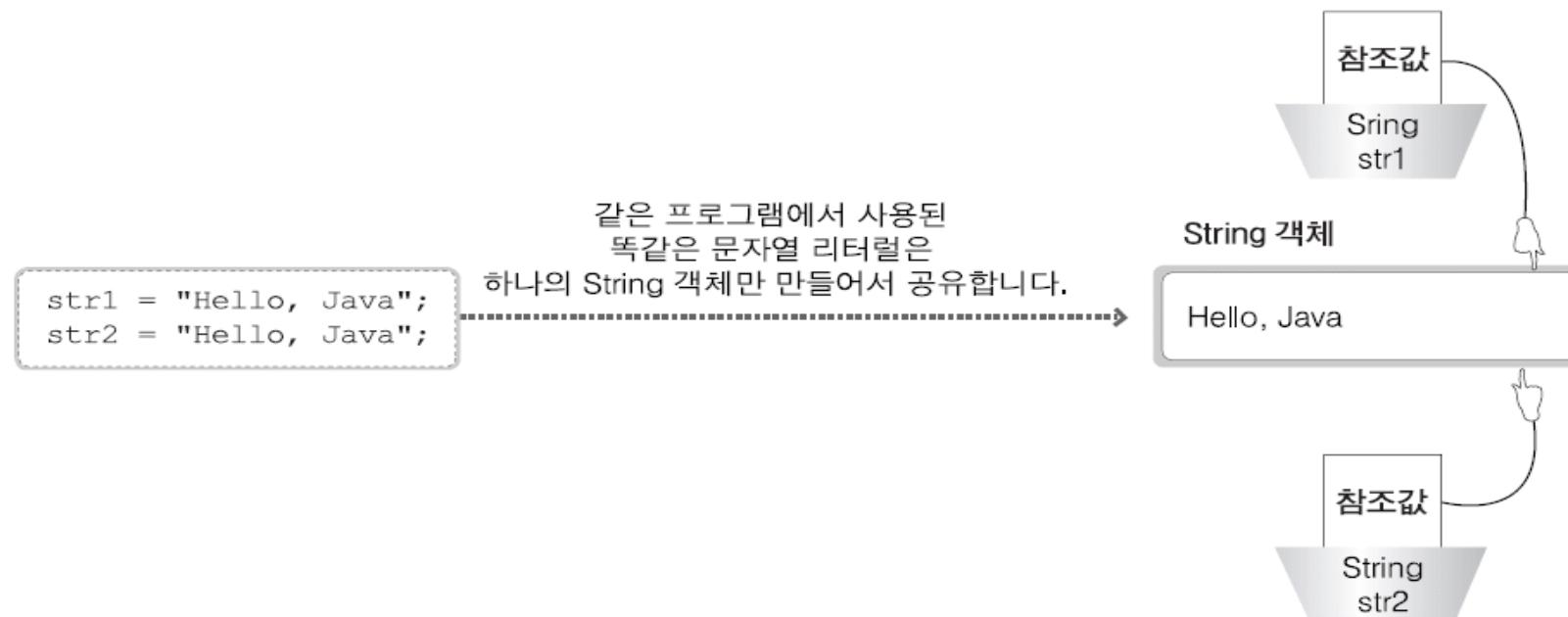
-자바 프로그램에 있는 문자열은 모두 **String** 클래스의 객체로 표현됨



# 문자열에 관련된 클래스들

## ◆ String 클래스

-문자열 리터럴이 String 객체로 만들어지는 방법

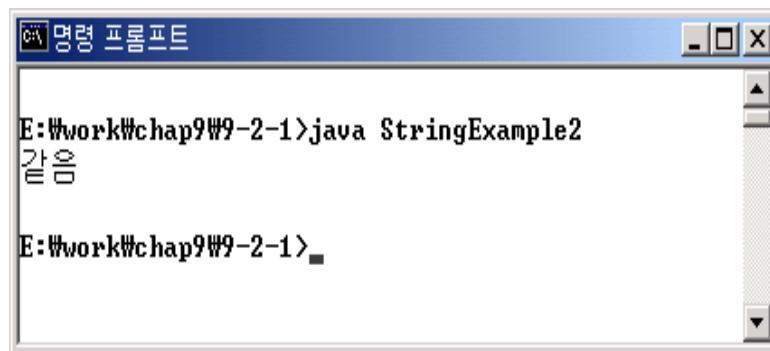


# String클래스 – 문자열 비교(동등 비교연산)

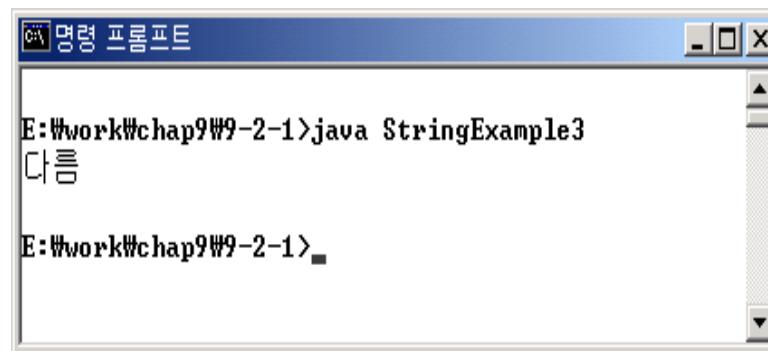
## ◆ 동등 연산자를 이용한 문자열 비교

```
class StringExample2 {  
    public static void main(String args[]) {  
        String str1 = "자바";  
        String str2 = "자바";  
        if (str1 == str2)  
            System.out.println("같음");  
        else  
            System.out.println("다름");  
    }  
}
```

```
class StringExample3 {  
    public static void main(String args[]) {  
        String str1 = new String("자바");  
        String str2 = new String("자바");  
        if (str1 == str2)  
            System.out.println("같음");  
        else  
            System.out.println("다름");  
    }  
}
```



```
E:\work\chap9\9-2-1>java StringExample2  
같음  
E:\work\chap9\9-2-1>
```



```
E:\work\chap9\9-2-1>java StringExample3  
다름  
E:\work\chap9\9-2-1>
```

# String클래스 – 문자열 비교>equals)

-문자열 내용을 비교하는 equals 메소드

str1.equals( str2 )  
↑                    ↗  
이 문자열과 이 문자열을 비교합니다.

```
class StringExample4 {  
    public static void main(String args[]) {  
        String str1 = "자바";  
        String str2 = "자바";  
        if (str1.equals(str2))  
            System.out.println("같음");  
        else  
            System.out.println("다름");  
    }  
}
```

E:\#work#\chap9#\9-2-1>java StringExample4  
같음  
E:\#work#\chap9#\9-2-1>

```
class StringExample5 {  
    public static void main(String args[]) {  
        String str1 = new String("자바");  
        String str2 = new String("자바");  
        if (str1.equals(str2))  
            System.out.println("같음");  
        else  
            System.out.println("다름");  
    }  
}
```

E:\#work#\chap9#\9-2-1>java StringExample5  
같음  
E:\#work#\chap9#\9-2-1>

# 문자 관련 클래스 - String

---

## ◆ Constructor(생성자)

- String()
- String(char chars[])
- String(char chars[], int startIndex, int numChars)
- String(String strObj) - 새로운 String 생성
- String(byte asciiChars[])
- String(byte asciiChars[], int startIndex, int numChars)

## ◆ Method

- public char charAt(int index)
- public int compareTo([String](#) anotherString)
- public int indexOf(int ch)
- public int indexOf(int ch, int fromIndex)
- public byte[] getBytes([String](#) charsetName) throws [UnsupportedEncodingException](#)

- 
- public `String` substring(int beginIndex)
  - public `String` substring(int beginIndex, int endIndex)
  - public `String` concat(`String` str)
  - public `String` replace(char oldChar, char newChar)
  - public boolean contains(`CharSequence` s)
  - public `String`[] split(`String` regex, int limit)
  - public `String` toLowerCase()
  - public `String` toUpperCase()
  - public `String` trim()
  - public `String` toString()
  - public char[] toCharArray()
  - public static `String` valueOf(`Object` obj)

# 문자열 관련 클래스 – StringBuilder, StringBuffer

## ◆ StringBuilder 클래스와 StringBuffer 클래스

- StringBuilder 클래스와 StringBuffer 클래스의 유사성

[StringBuilder 클래스의 API 규격서]

**Constructor Summary**

- `StringBuilder()` Constructs a string builder with no characters in it and an initial capacity of 16 characters.
- `StringBuilder(CharSequence seq)` Constructs a string builder that contains the same characters as the specified CharSequence.
- `StringBuilder(int capacity)` Constructs a string builder with no characters in it and an initial capacity specified by the capacity argument.
- `StringBuilder(String str)` Constructs a string builder initialized to the contents of the specified string.

**Method Summary**

- `append(boolean b)` Appends the string representation of the boolean argument to the sequence.
- `append(char c)` Appends the string representation of the char argument to this sequence.
- `append(char[] str)` Appends the string representation of the char array argument to this sequence.
- `append(char[] str, int offset, int len)` Appends the string representation of a subarray of the char array argument to this sequence.
- `append(CharSequence s)` Appends the specified character sequence to this Appendable.
- `append(CharSequence s, int start, int end)` Appends a subsequence of the specified CharSequence to this sequence.
- `append(double d)` Appends the string representation of the double argument to this sequence.
- `append(float f)` Appends the string representation of the float argument to this sequence.
- `append(int i)` Appends the string representation of the int argument to this sequence.
- `append(long lng)` Appends the string representation of the long argument to this sequence.
- `append(Object obj)` Appends the string representation of the object argument.
- `append(String str)` Appends the specified string to this character sequence.
- `append(StringBuffer sb)` Appends the specified StringBuffer to this sequence.
- `appendCodePoint(int codePoint)` Appends the string representation of the codePoint argument to this sequence.

생성자의 사용 방법이 같습니다.

[StringBuffer 클래스의 API 규격서]

**Constructor Summary**

- `StringBuffer()` Constructs a string buffer with no characters in it and an initial capacity of 16 characters.
- `StringBuffer(CharSequence seq)` Constructs a string buffer that contains the same characters as the specified CharSequence.
- `StringBuffer(int capacity)` Constructs a string buffer with no characters in it and the specified initial capacity.
- `StringBuffer(String str)` Constructs a string buffer initialized to the contents of the specified string.

**Method Summary**

- `append(boolean b)` Appends the string representation of the boolean argument to the sequence.
- `append(char c)` Appends the string representation of the char argument to this sequence.
- `append(char[] str)` Appends the string representation of the char array argument to this sequence.
- `append(char[] str, int offset, int len)` Appends the string representation of a subarray of the char array argument to this sequence.
- `append(CharSequence s)` Appends the specified CharSequence to this sequence.
- `append(CharSequence s, int start, int end)` Appends a subsequence of the specified CharSequence to this sequence.
- `append(double d)` Appends the string representation of the double argument to this sequence.
- `append(float f)` Appends the string representation of the float argument to this sequence.
- `append(int i)` Appends the string representation of the int argument to this sequence.
- `append(long lng)` Appends the string representation of the long argument to this sequence.
- `append(Object obj)` Appends the string representation of the Object argument.
- `append(String str)` Appends the specified string to this character sequence.
- `append(StringBuffer sb)` Appends the specified StringBuffer to this sequence.
- `appendCodePoint(int codePoint)` Appends the string representation of the codePoint argument to this sequence.

메소드의 기능과 사용 방법도 같습니다.

# 문자열 관련 클래스 –StringBuffer

---

- ◆ 변할 수 있는 문자열을 가진다. (mutable sequence of character)
- ◆ 동기화 처리가 되어 있다.( thread-safe)
- ◆ 생성자
  - StringBuffer()
    - 묵시적으로 16개의 문자를 저장할 수 있는 객체를 생성
  - StringBuffer(int size)
    - size 크기의 객체를 생성
  - StringBuffer(String str)
    - str로 지정된 문자열과 추가로 16개의 문자를 더 저장할 수 있는 객체를 생성
- ◆ StringBuffer 객체는 객체의 크기가 변할 때마다 메모리를 재할당 한다(16개의 문자를 저장할 수 있는 버퍼 단위로)
- ◆ 문자열을 조작하는 처리를 할 경우 String 보다 StringBuffer를 사용하는 것이 유리
  - 100배 이상 성능 차이 발생
  - String은 문자열을 수정하려 할 때마다 새로운 객체 생성, 그에 비해 StringBuffer는 객체에 담긴 내용만 수정

# 문자열 관련 클래스 –**StringBuffer**

---

## ◆ Method

- public [StringBuffer](#) append(boolean b) : 문자열 끝에 b를 추가
- public [StringBuffer](#) append(char c)
- public [StringBuffer](#) append(char[] str)
- public [StringBuffer](#) append([CharSequence](#) s)
- public int capacity() : 버퍼 크기를 반환
- public char charAt(int index) : index 위치의 문자를 반환
- public [StringBuffer](#) delete(int start, int end) start(inclusive)부터 end-1까지 삭제
- public [StringBuffer](#) deleteCharAt(int index)
- public int indexOf([String](#) str) 문자열이 나타나는 첫 위치
- public [StringBuffer](#) insert(int offset, char c) Offset위치에 문자 추가
- public int lastIndexOf([String](#) str) 끝에서부터 문자열이 나타나는 위치
- public int length() 버퍼에 있는 문자열 길이
- public [StringBuffer](#) reverse() 문자를 역순으로 반환
- public [CharSequence](#) subSequence(int start, int end) 문자열 start부터 end-1까지 반환
- public [String](#) substring(int start)
- public [String](#) substring(int start, int end)
- public void trimToSize() 버퍼크기를 문자열 길이 만큼으로 조정

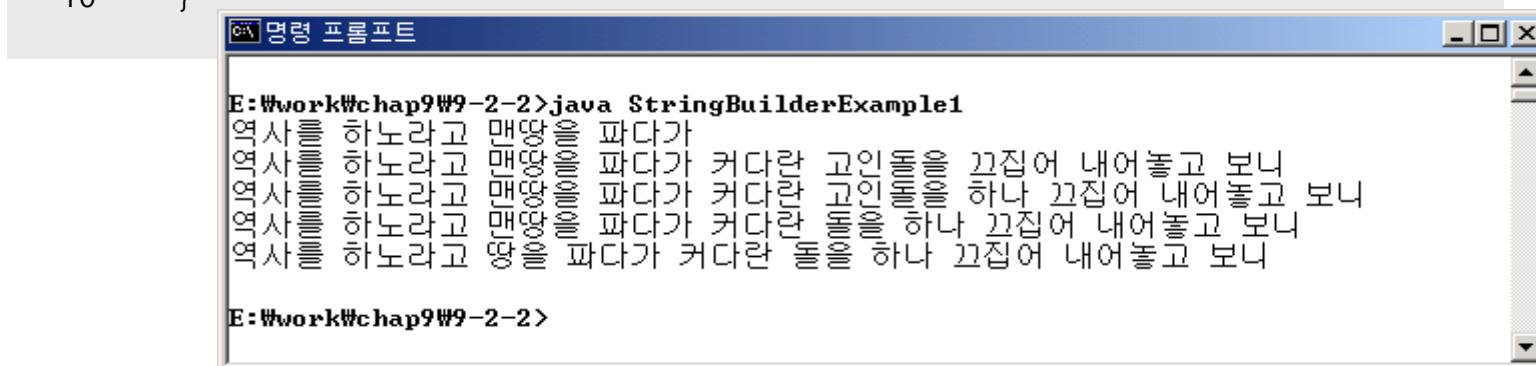
# 문자열 관련 클래스 –StringBuilder

---

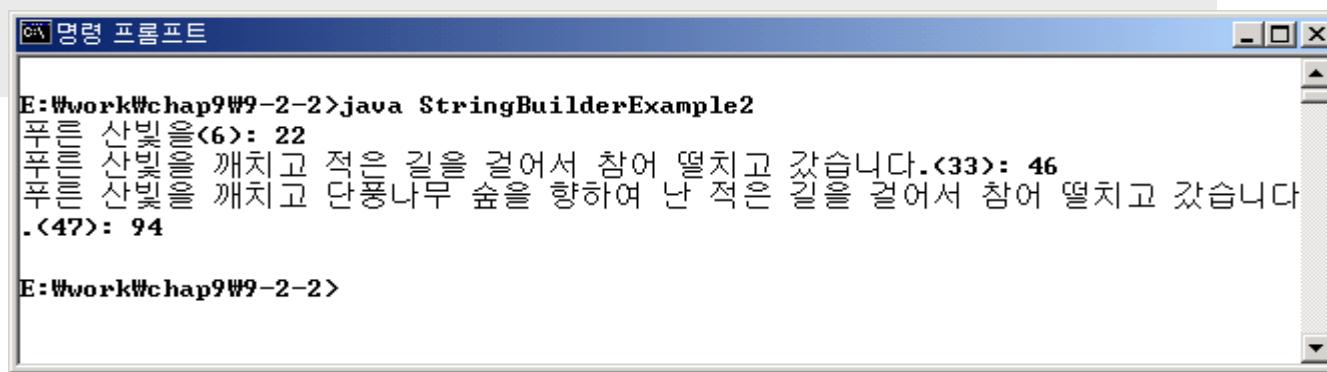
- ◆ 변할 수 있는 문자열을 가진다.
- ◆ 동기화가 보장되지 않음(no guarantee of synchronization)
- ◆ not safe for use by multiple threads
- ◆ 생성자
  - public StringBuilder()  
초기 용량이 16개인 문자를 저장할 수 있는 객체를 생성
  - public StringBuilder(int capacity)
  - public StringBuilder([String](#) str)
- 메서드는 API를 참조할것!

-StringBuilder 클래스를 이용하여 문자열을 조작하는 프로그램

```
1  class StringBuilderExample1 {  
2      public static void main(String args[]) {  
3          StringBuilder sb = new StringBuilder("역사를 하노라고 맨땅을 파다가 ");  
4          System.out.println(sb);  
5          System.out.println(sb.append("커다란 고인돌을 끄집어 내어놓고 보니"));  
6          System.out.println(sb.insert(26, "하나 "));  
7          System.out.println(sb.delete(21, 23));  
8          System.out.println(sb.deleteCharAt(9));  
9      }  
10 }
```



```
1 class StringBuilderExample2 {  
2     public static void main(String args[]) {  
3         StringBuilder sb = new StringBuilder("푸른 산빛을");  
4         printStringBuilder(sb);  
5         sb.append(" 깨치고 적은 길을 걸어서 참어 떨치고 갔습니다.");  
6         printStringBuilder(sb);  
7         sb.insert(10, " 단풍나무 숲을 향하여 난");  
8         printStringBuilder(sb);  
9     }  
10    static void printStringBuilder(StringBuilder sb) {  
11        String str = sb.toString();  
12        int len = sb.length();  
13        int bufSize = sb.capacity();  
14        System.out.printf("%s(%d): %d %n", str, len, bufSize);  
15    }  
16 }
```



# 문자열에 관련된 클래스들

-StringBuilder 객체의 버퍼 크기를 문자열에 맞게 줄이는 프로그램

```
1  class StringBuilderExample4 {  
2      public static void main(String args[]) {  
3          StringBuilder sb = new StringBuilder(100);  
4          sb.append("자바");  
5          System.out.println(sb + ": " + sb.capacity());  
6          sb.trimToSize(); - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
7          System.out.println(sb + ": " + sb.capacity());  
8      }  
9  }
```

과도한 버퍼 크기를 적당하게 줄이는  
메소드



# 문자열 관련 클래스 – StringTokenizer 클래스

- 문자열로부터 토큰(token)을 추출하는 기능이 있는 클래스

- 사용 방법

```
StringTokenizer stok = new StringTokenizer("사과 배 복숭아");
```



토큰을 추출할 문자열을 가지고  
StringTokenizer 객체를 생성합니다.

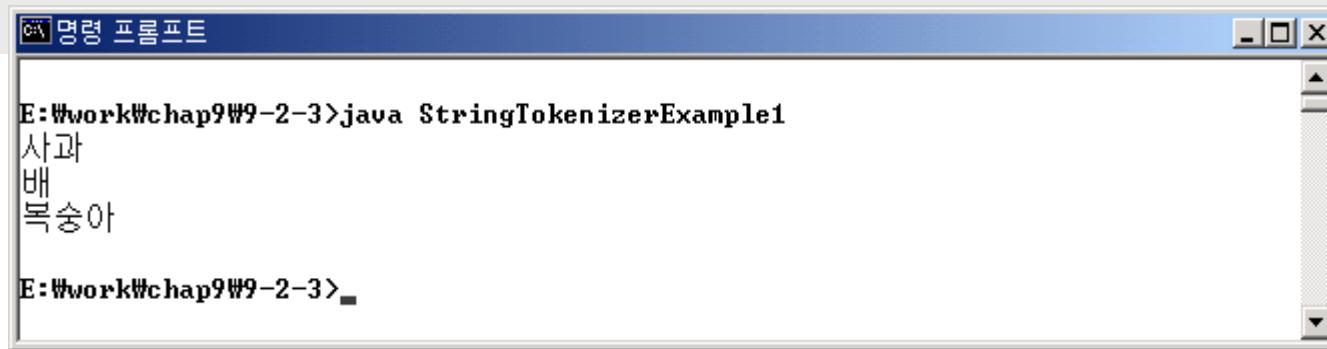
```
str1 = stok.nextToken(); // 첫 번째 토큰인 "사과"를 리턴  
str2 = stok.nextToken(); // 두 번째 토큰인 "배"를 리턴  
str3 = stok.nextToken(); // 세 번째 토큰인 "복숭아"를 리턴
```



더 이상 추출할 토큰이 없으면 NoSuchElementException 발생

```
while (stok.hasMoreTokens()) { // 토큰이 있는 동안만 while 문 안에서  
    str = stok.nextToken(); // 토큰을 추출하여  
    System.out.println(str); // 처리합니다.  
}
```

```
1 import java.util.StringTokenizer;
2 class StringTokenizerExample1 {
3     public static void main(String args[]) {
4         StringTokenizer stok = new StringTokenizer("사과 배 복숭아");
5         while (stok.hasMoreTokens()) {
6             String str = stok.nextToken();
7             System.out.println(str);
8         }
9     }
10 }
```



---

## ◆ StringTokenizer 클래스

### ◆ Constructor(생성자)

- public StringTokenizer([String](#) str)
- public StringTokenizer([String](#) str, [String](#) delim)
- public StringTokenizer([String](#) str, [String](#) delim, boolean returnDelims)

### ◆ method(메서드)

- public boolean hasMoreTokens()
- public [String](#) nextToken()
- public int countTokens()
- public boolean hasMoreElements()
- public [Object](#) nextElement()

-공백문자가 아닌 구획문자를 이용하여 토큰을 추출하는 예

```
1 import java.util.StringTokenizer;
2 class StringTokenizerExample2 {
3     public static void main(String args[ ]) {
4         StringTokenizer stok = new StringTokenizer("사과,배,복숭아", ",,");
5         while (stok.hasMoreTokens()) {
6             String str = stok.nextToken();
7             System.out.println(str);
8         }
9     }
10 }
```



The screenshot shows a Windows command prompt window titled "명령 프롬프트". The command E:\work\chap9\9-2-3>java StringTokenizerExample2 is entered, followed by four tokens: 사과, 배, and 복숭아.

```
E:\work\chap9\9-2-3>java StringTokenizerExample2
사과
배
복숭아

E:\work\chap9\9-2-3>
```

# 문자열에 관련된 클래스들

## -구획 문자(delimiter) 지정하기

```
stok = new StringTokenizer("사과,배,복숭아", ",");
```

이 문자열에 있는 콤마(,)를 가지고  
토큰을 추출하는 StringTokenizer 객체를 생성합니다.

```
stok = new StringTokenizer("사과,배 | 복숭아", ",|");
```

이 문자열에 있는 콤마(,)와 수직선(|)을 가지고  
토큰을 추출하는 StringTokenizer 객체를 생성합니다.

```
stok = new StringTokenizer("사과=10 | 초콜렛=3 | 샴페인=1", "=| ", true);
```

토큰을 추출할  
문자열

구획문자

구획문자도 토큰으로  
추출하도록 만드는 true 파라미터

---

## -구획문자를 토큰으로 추출하는 예

```
1 import java.util.*;
2 class StringTokenizerExample3 {
3     public static void main(String args[]) {
4         StringTokenizer stok = new StringTokenizer("사과=10|초콜렛=3|샴페인=1", "="|", true);
5         while (stok.hasMoreTokens()) {
6             String token = stok.nextToken();
7             if (token.equals("="))
8                 System.out.print("\t");
9             else if (token.equals("|"))
10                System.out.print("\n");
11             else
12                 System.out.print(token);
13         }
14     }
15 }
```

토큰이 "="이면 탭 문자,  
"|"이면 줄 바꿈 문자,  
그 밖의 문자열이면 토큰을 출력합니다.



# **WRAPPER 클래스**

# Wrapper 클래스

## ◆ Wrapper 클래스란?

-프리미티브 타입을 객체로 표현하는 데 사용되는 다음 클래스들의 통칭

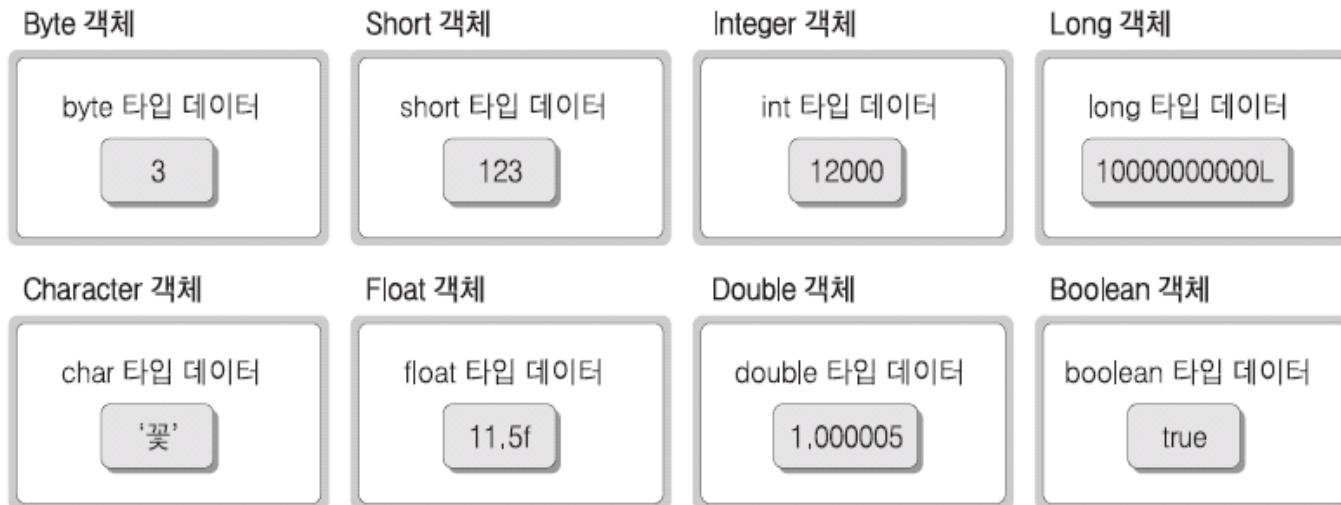
클래스 이름	해당 프리미티브 타입
Byte	byte
Short	short
Integer	int
Long	long
Character	char
Float	float
Double	double
Boolean	boolean

이름에 주의

✓ - 위 클래스들은 모두 java.lang 패키지에 속함

## ◆ Wrapper 클래스란?

- 프리미티브 타입을 데이터를 감싸는 역할을 하는 Wrapper 클래스



## ◆ Wrapper 클래스의 기본적인 사용 방법

- 래퍼 객체(wrapper object)를 만드는 방법

```
Byte    obj1 = new Byte((byte) 1);
Short   obj2 = new Short((short) 123);
Integer obj3 = new Integer(12345);
Long    obj4 = new Long(1234567890L);
Float   obj5 = new Float(1.5f);
Double  obj6 = new Double(1.00005);
Character obj7 = new Character('꽃');
Boolean  obj8 = new Boolean(true);
```

Wrapper 클래스의 생성자

- 래퍼 객체 안에 있는 프리미티브 값을 가져오는 방법

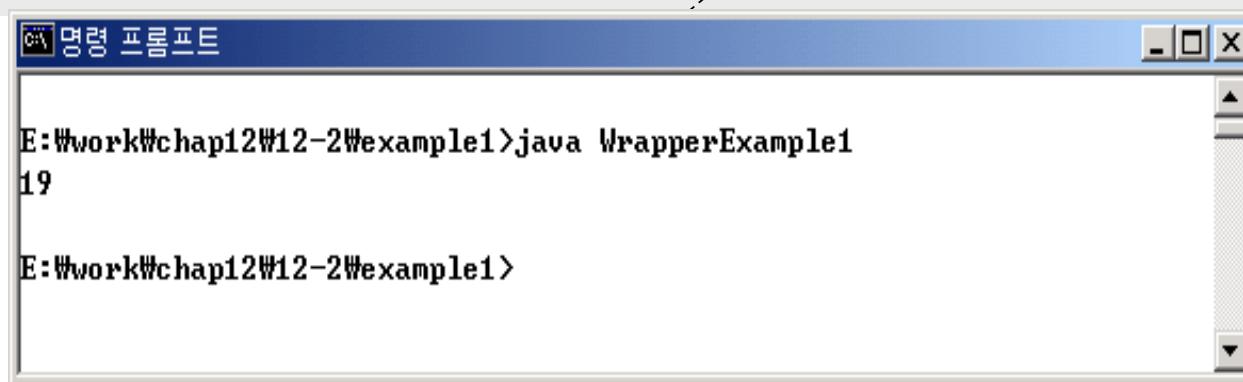
```
byte   num1 = obj1.byteValue();
short  num2 = obj2.shortValue();
int    num3 = obj3.intValue();
long   num4 = obj4.longValue();
float  num5 = obj5.floatValue();
double num6 = obj6.doubleValue();
char   ch   = obj7.charValue();
boolean truth = obj8.booleanValue();
```

Wrapper 객체 안에 있는  
프리미티브 타입의 값을  
가져오는 메소드

---

- Wrapper 클래스의 사용 예

```
1  class WrapperExample1 {  
2      public static void main(String args[]) {  
3          Integer obj1 = new Integer(12); } } // Integer 객체를 생성합니다.  
4          Integer obj2 = new Integer(7);  
5          int sum = obj1.intValue() + obj2.intValue();  
6          System.out.println(sum); } // Integer 객체 안에 있는 int 값을 반환  
7      }  
8  }
```



```
1 class WrapperExample2 {  
2     public static void main(String args[]) {  
3         int total = 0;  
4         for (int cnt = 0; cnt < args.length; cnt++) {  
5             Integer obj = new Integer(args[cnt]);  
6             total += obj.intValue();  
7         }  
8         System.out.println(total);  
9     }  
10 }
```

```
E:\work\chap12\12-2\example2>java WrapperExample2  
0  
  
E:\work\chap12\12-2\example2>java WrapperExample2 10 20  
30  
  
E:\work\chap12\12-2\example2>java WrapperExample2 100 100 100  
300  
  
E:\work\chap12\12-2\example2>
```

파라미터로 아무 값도 넘겨주지 않으면 0을 출력합니다.

파라미터로 정수를 넘겨주면 합계를 출력합니다.

---

-프리미티브 타입의 비트 패턴을 문자열로 리턴하는 메소드

```
String str1 = Integer.toBinaryString(9);           // "1001"을 리턴  
String str2 = Long.toBinaryString(100000000L);  
                                                // "10111110101110000100000000"을 리턴
```

-부동소수점수와 똑같은 비트 패턴을 갖는 정수를 리턴하는 메소드

```
int num1 = Float.floatToIntBits(1.5f);  
                                // 1.5f와 똑같은 비트 패턴의 int 값을 리턴  
long num2 = Double.doubleToLongBits(1.0005);  
                                // 1.0005와 똑같은 비트 패턴의 long 값을 리턴
```

---

### -문자열을 프리미티브 타입으로 바꾸는 메소드

```
byte    num1 = Byte.parseByte("1");
short   num2 = Short.parseShort("123");
int     num3 = Integer.parseInt("12345");
long    num4 = Long.parseLong("1234567890");
float   num5 = Float.parseFloat("1.5");
double  num6 = Double.parseDouble("1.00005");
boolean truth = Boolean.parseBoolean("true");
```

문자열을 프리미티브 타입의  
값으로 바꾸는 메소드

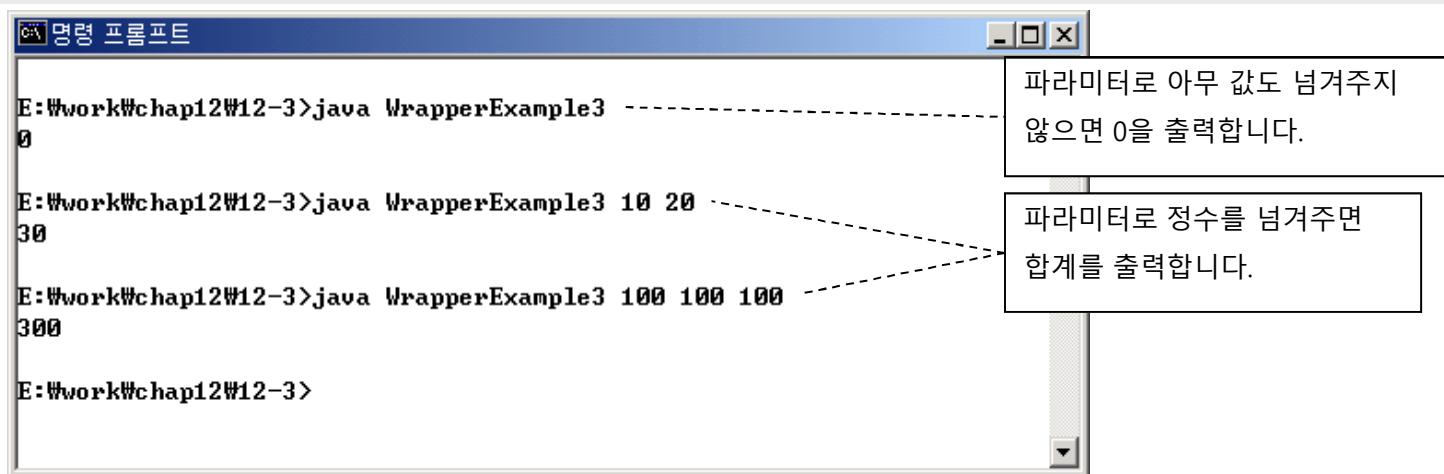
### -Wrapper 클래스의 생성자를 대신하는 메소드

```
Byte    obj1 = Byte.valueOf((byte) 1);
Short   obj2 = Short.valueOf((short) 123);
Integer obj3 = Integer.valueOf(12345);
Long    obj4 = Long.valueOf(1234567890L);
Float   obj5 = Float.valueOf(1.5f);
Double  obj6 = Double.valueOf(1.00005);
Character obj7 = Character.valueOf('꽃');
Boolean obj8 = Boolean.valueOf(true);
```

Wrapper 클래스의 생성자를  
대신하는 메소드

## ◆ parse-메소드의 사용 예

```
1  class WrapperExample3 {  
2      public static void main(String args[]) {  
3          int total = 0;  
4          for (int cnt = 0; cnt < args.length; cnt++)  
5              total = Integer.parseInt(args[cnt]);  
6          System.out.println(total);  
7      }  
8  }
```



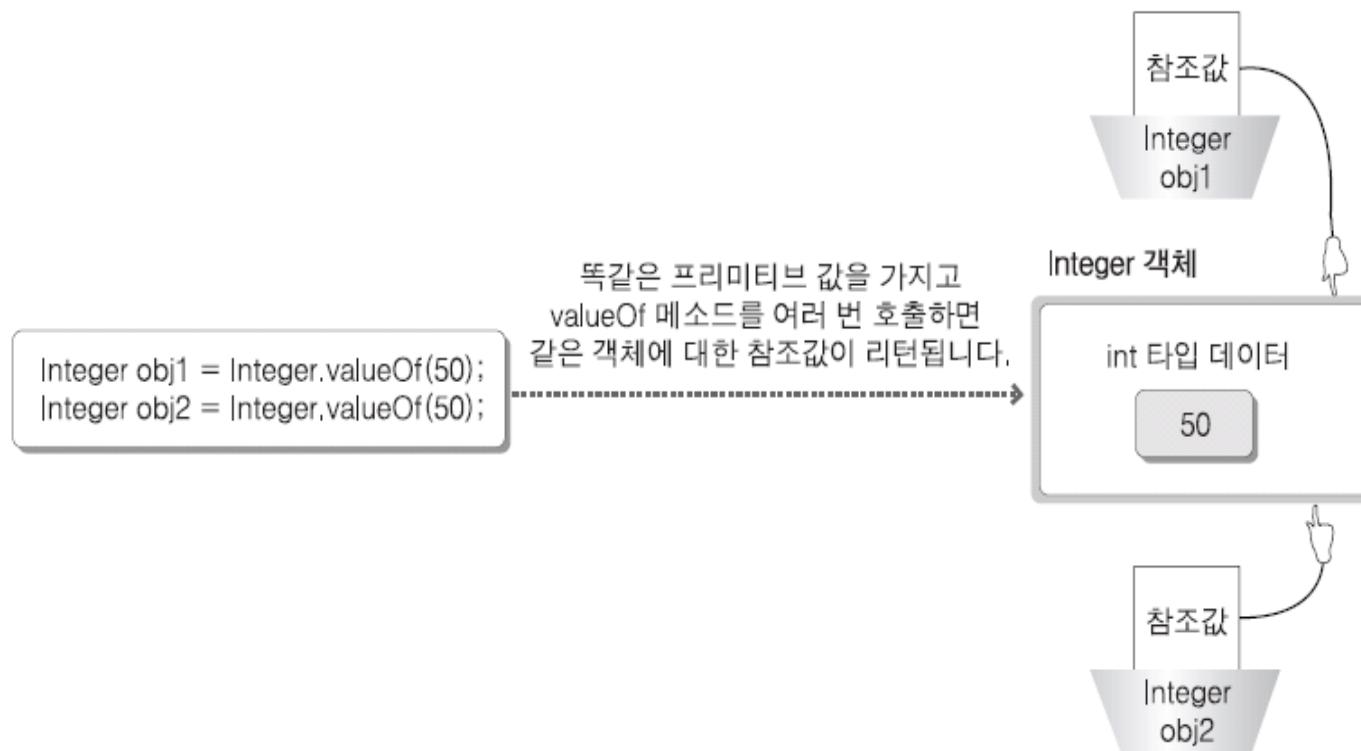
```
E:\work\chap12\12-3>java WrapperExample3 -----  
0  
  
E:\work\chap12\12-3>java WrapperExample3 10 20 -----  
30  
  
E:\work\chap12\12-3>java WrapperExample3 100 100 100 -----  
300
```

파라미터로 아무 값도 넘겨주지 않으면 0을 출력합니다.

파라미터로 정수를 넘겨주면 합계를 출력합니다.

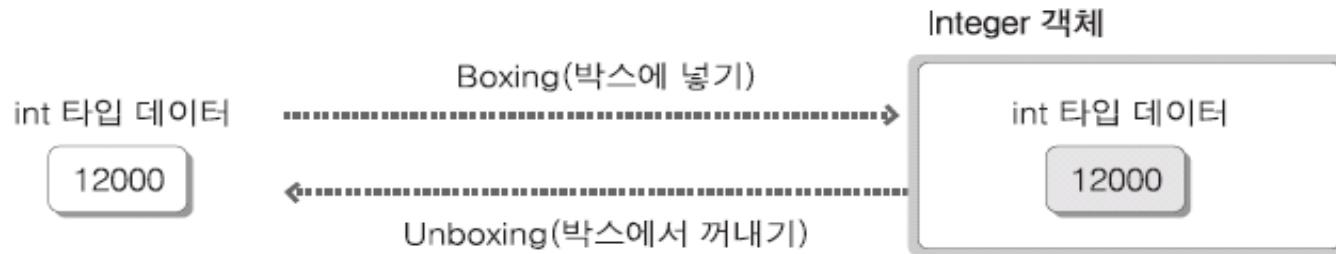
---

- `valueOf` 메소드의 작동 방식



# Boxing과 UnBoxing

## ◆ Boxing과 UnBoxing의 개념도



[예]

```
Integer obj = new Integer(12000);      // Boxing  
int num = obj.intValue();            // Unboxing
```

## ◆ 자동 Boxing과 자동 UnBoxing

### – 자동 Boxing

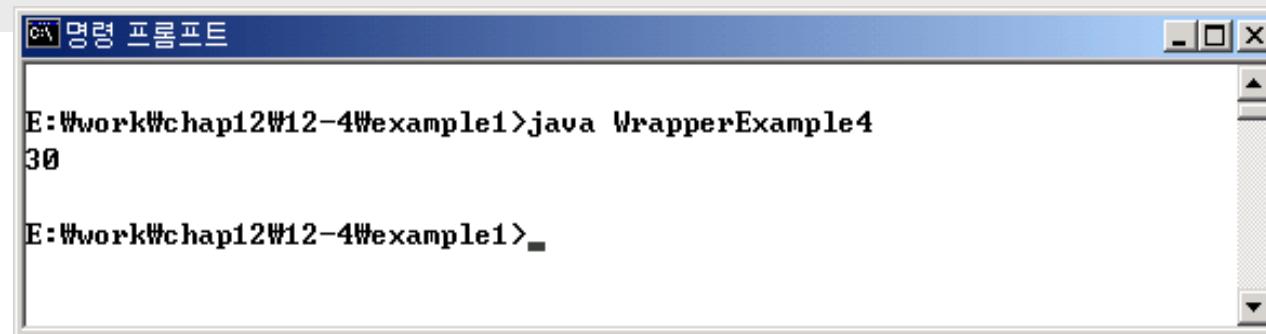
- ✓ 래퍼 객체를 써야할 자리에 프리미티브 값을 썼을 때 일어나는 래퍼 객체로의 자동 변환

### – 자동 UnBoxing

- ✓ 프리미티브 값을 써야할 자리에 래퍼 객체를 썼을 때 일어나는 프리미티브 값으로의 자동 변환

## ◆ 자동 UnBoxing

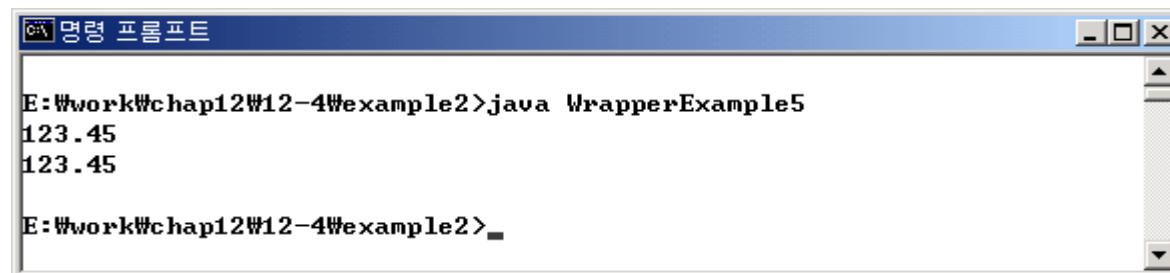
```
1  class WrapperExample4 {  
2      public static void main(String args[]) {  
3          Integer obj = new Integer("10");  
4          int sum = obj + 20;    ----- Integer 객체와 int 타입의 값을 더하는 명령문  
5          System.out.println(sum);  
6      }  
7  }
```



## ◆ 자동 Boxing이 일어나는 경우

```
1  class WrapperExample5 {  
2      public static void main(String args[]) {  
3          printDouble(new Double(123.45));  
4          printDouble(123.45);  
5      }  
6      static void printDouble(Double obj) {  
7          System.out.println(obj);  
8      }  
9  }
```

double 타입의 값을 가지고  
Double 타입 파라미터를 받는 메소드를 호출



# **자료구조로 사용되는 자바 클래스들**

# ArrayList 원소 정렬하기

---

- ◆ ArrayList 원소들을 정렬하려면 어떻게 해야 할까?
- ◆ API를 뒤져서 정렬 기능을 제공하는 메소드가 있는지 확인!!
  - ArrayList에는 정렬용 메소드가 없습니다.
- ◆ 혹시 다른 컬렉션 객체를 쓸 수 있는지 확인
  - 자동으로 정렬되는 TreeSet 같은 클래스를 활용하는 것도 좋은 대안이 될 수 있습니다.

## Collections.sort() 메소드

---

- ◆ **java.util.Collections** 클래스의 클래스 메소드인 **sort()** 메소드를 활용하면 됩니다.

```
public static void sort(List list)
```

- ◆ **java.util.Collections.sort()** 메소드를 이용하면 임의의 **List** 객체를 알파벳순으로 정렬할 수 있습니다.

- ◆ 하지만 **String** 객체가 아닌, 다른 복잡한 객체로 구성된 리스트는 어떻게 정렬할까요?

# Collections.sort() 메소드

---

<T extends Comparable<? super T>>

List<T> list

## Method Detail

### sort

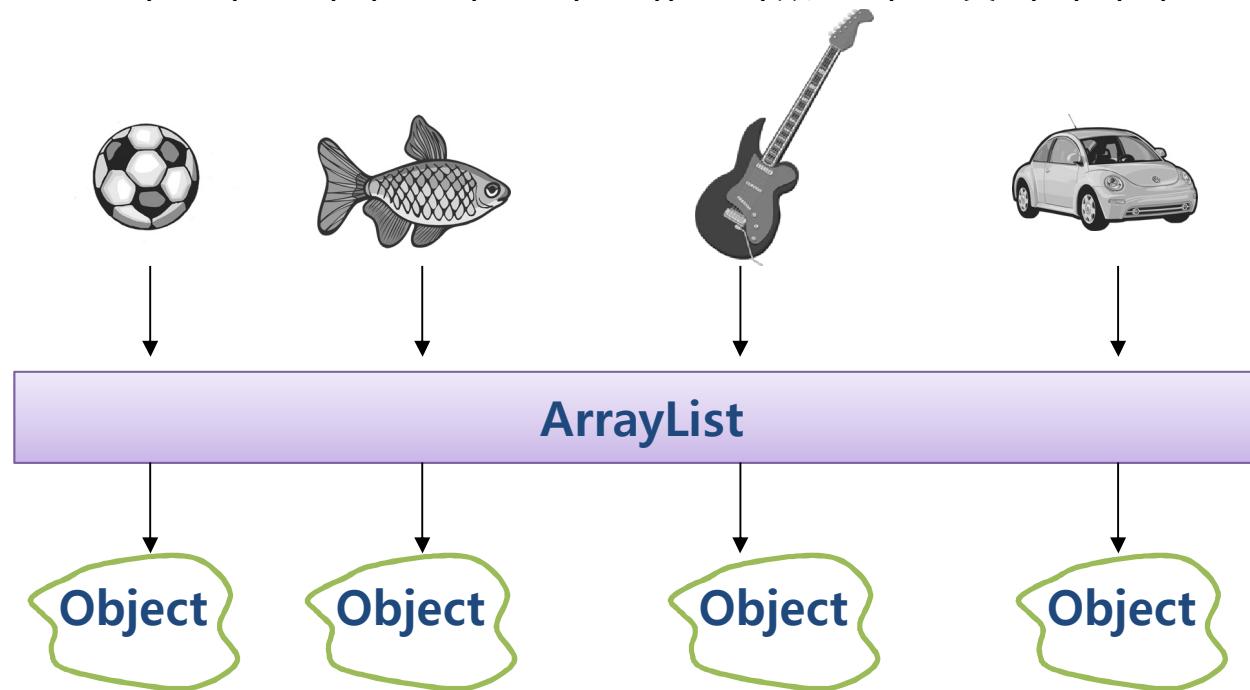
```
public static <T extends Comparable<? super T>> void sort(List<T> list)
```

Sorts the specified list into ascending order, according to the *natural ordering* of its elements. All elements in the list must implement the Comparable interface. Furthermore, all elements in the list must be *mutually comparable* (that is, `e1.compareTo(e2)` must not throw a `ClassCastException` for any elements `e1` and `e2` in the list).

# 제네릭과 형안전성

## ◆ 제네릭을 활용하지 않는 경우

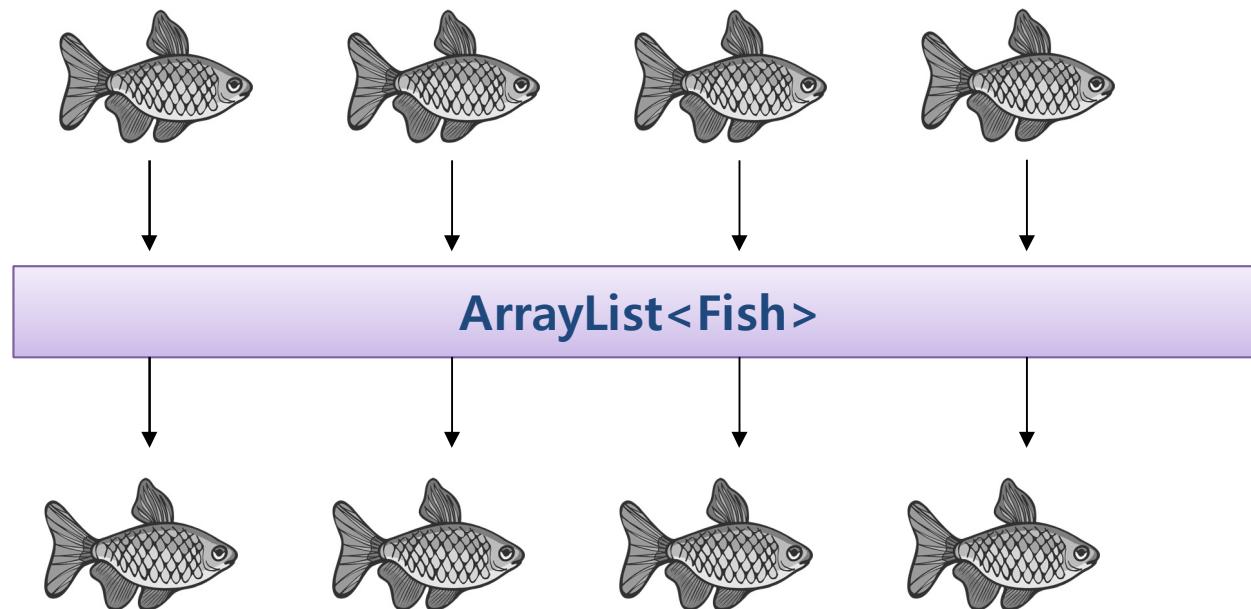
- 일단 컬렉션에 들어가고 나면 어떤 유형이었는지를 잊어버리게 됩니다.



---

- 제네릭을 활용하는 경우

- 유형이 엉뚱하게 바뀌지 않습니다.
- 형안전성이 확보됩니다.



# 제네릭 사용법

---

## ◆ 제네릭을 사용하는 클래스

```
new ArrayList<Song>()
```

## ◆ 제네릭형 변수 선언 및 대입

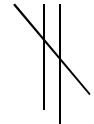
```
List<Song> songList = new ArrayList<Song>()
```

## ◆ 제네릭형을 받아들이는 메소드 선언 및 호출

```
void foo(List<Song> list)
```

```
x.foo(songList)
```

```
public <T extends Animal> void takeThing(ArrayList<T> list)
```



```
public void takeThing(ArrayList<Animal> list)
```

# 제네릭 사용법

---

- ◆ **extends 키워드의 의미**

- 제네릭에서의 extends 키워드는 확장과 구현을 모두 의미합니다.

- ◆ **<T extends Comparable<? super T>>**

- ◆ **위 코드가 무엇을 뜻하는지 답해보세요.**

# Comparator

---

- ◆ 정렬 순서를 정하는 기준을 직접 정하고 싶다면?  
    또는
  - ◆ Comparable을 구현하지 않는 클래스를 정렬하고 싶다면?
- 
- ◆ java.util.Comparator 사용
  - ◆ sort(List o, Comparator c) 메소드 사용

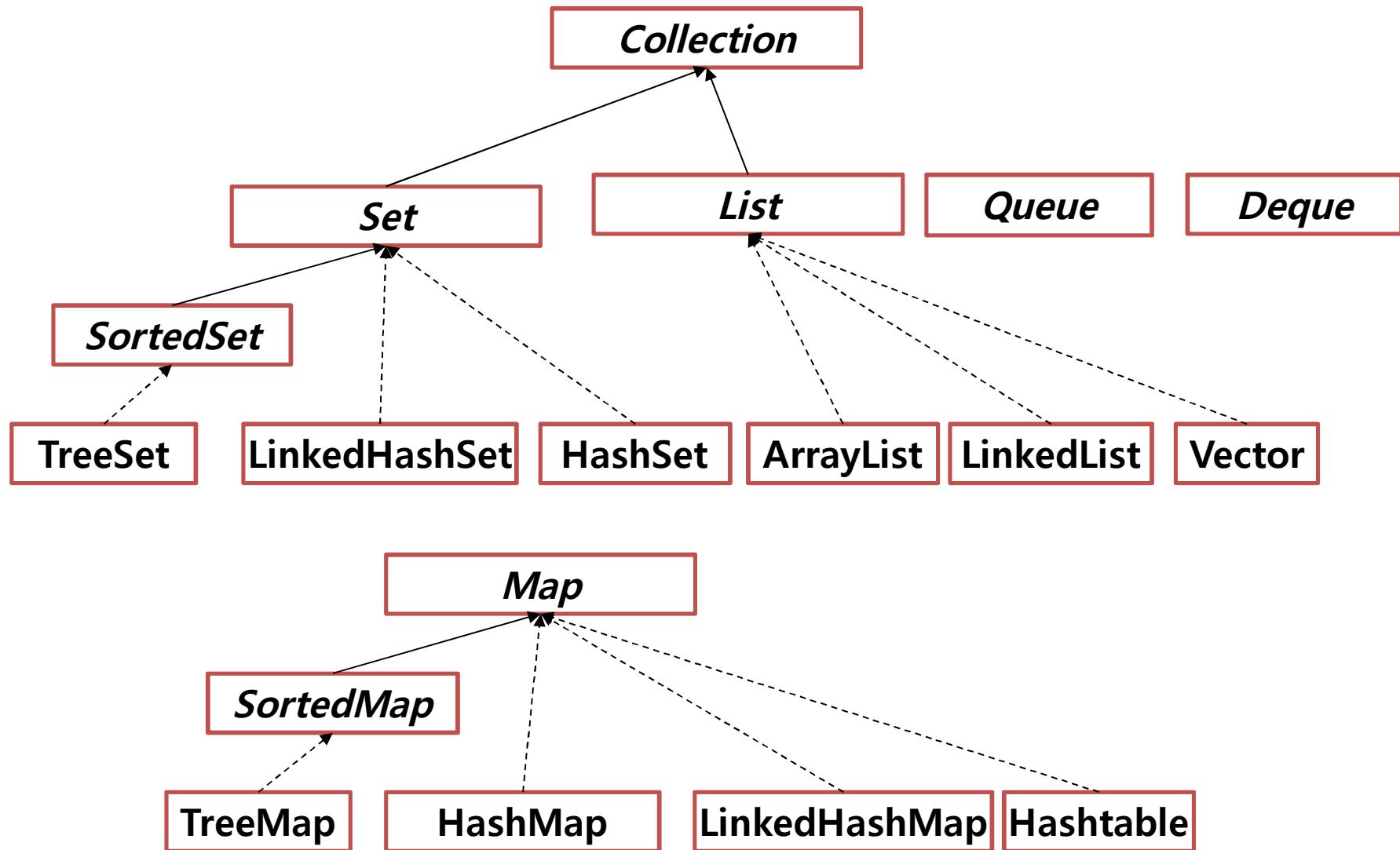
## Comparator 활용

---

```
class ArtistCompare implements Comparator<Song> {  
    public int compare(Song one, Song two) {  
        return one.getArtist().compareTo(two.getArtist());  
    }  
}
```

```
ArtistCompare artistCompare = new ArtistCompare();  
Collections.sort(songList, artistCompare);
```

# 자료구조 - 컬렉션의 종류



# 자료구조 - List

---

- ◆ 인덱스 제공
- ◆ 순서보장, 중복적으로 객체 저장이 가능
- ◆ **ListIterator**를 줄 수 있음

- **ArrayList**
  - 상당히 빠르고 크기를 마음대로 조절할 수 있는 초강력 배열
- **Vector**
  - **ArrayList**의 구형 버전. 모든 메소드가 동기화되어 있음 - 배열
  - 요즘은 대부분 **ArrayList**를 사용함
- **LinkedList**
  - 목록 끝에 원소를 추가하거나 끝에 있는 원소를 쉽게 제거할 수 있는 메소드 제공
  - 스택, 큐, 양방향 큐 등을 만들기 위한 용도로 많이 쓰임

# 자료구조 - Set

---

- ◆ 중복된 항목이 들어가는 것을 방지함
- ◆ equals() 메소드를 이용하여 중복 확인

- HashSet
  - 가장 빠른 임의 접근 속도
  - 순서를 전혀 예측할 수 없음
- LinkedHashSet
  - 추가된 순서, 또는 가장 최근에 접근한 순서대로 접근 가능
- TreeSet
  - 정렬된 순서대로 보관. 정렬 방법을 지정할 수 있음

# 자료구조 - Map

---

- ◆ 키와 값을 대응시키는 기능을 제공
- ◆ 키와 값은 모두 객체여야만 함
  - HashMap
    - 가장 빠른 임의 접근 기능을 제공
  - Hashtable
    - HahsMap의 구형 버전
  - LinkedHashMap
    - LinkedHashMap과 유사. 입력된 순서 또는 가장 최근에 접근된 순서대로 보관
  - TreeMap
    - 정렬된 순서대로 유지하기에 좋음

# 객체가 같은지 확인하는 방법

---

## ◆ 객체 동치

- equals() 메소드에 대해 true가 리턴되는 것
- 똑같은 객체를 참조해야 하는 것은 아님

## ◆ 레퍼런스 동치

- 두 레퍼런스가 같은 객체를 참조하는 경우
- `a == b` 가 참이 되어야 함

```
String a = "abc";
if (a.equals(new String("abc")) {
```

- ◆ 어떤 조건에서 두 객체가 같다고 인정할 수 있을지 결정한다  
음 equals() 메소드를 오버라이드
- ◆ 이 때 hashCode() 메소드도 반드시 오버라이드해야 함

## 와일드카드

---

```
public <T extends Animal> void takeThing(ArrayList<T> list)
```

||

```
public void takeThing(ArrayList<? extends Animal> list)
```

- ◆ 와일드카드를 쓸 때는 목록에 새로운 원소를 추가할 수 없음

# 자료구조

---

## ◆ 자료구조란?

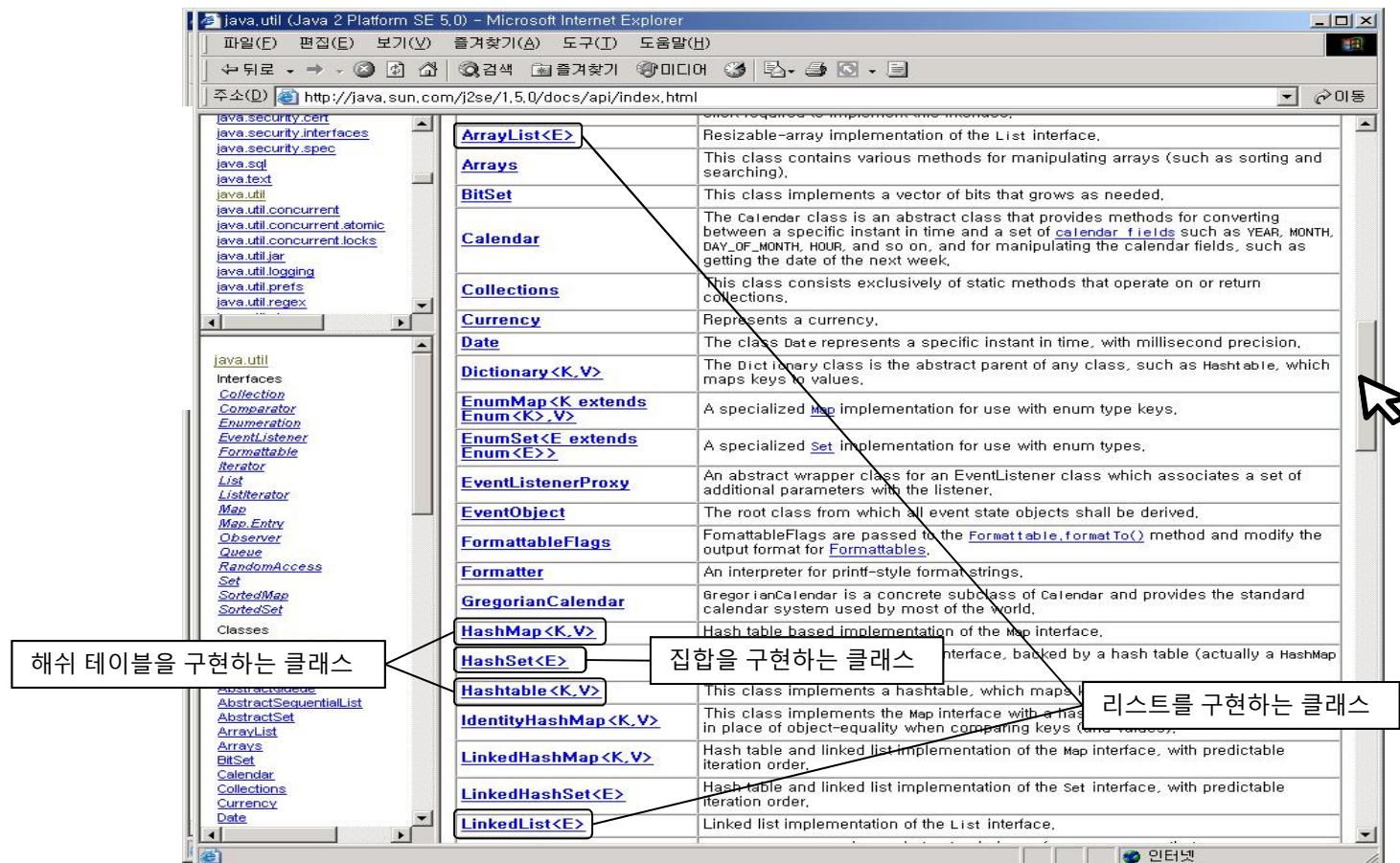
-자료구조(data structure)

✓ 데이터를 효율적으로 사용할 수 있도록 구조를 만들어서 저장해둔 것

-자료구조의 종류

- ✓ 리스트(list) : 배열 리스트(array list), 연결 리스트(linked list)로 세분됨
- ✓ 스택(stack)
- ✓ 큐(queue)
- ✓ 해시 테이블(hashtable)
- ✓ 집합(set) \* 엄밀히 말하면 자료구조가 아님

## -자료구조 클래스의 API 규격서

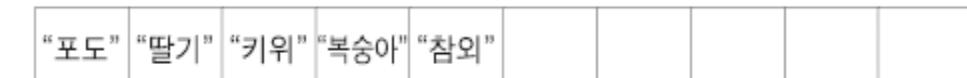


# 자료구조 - list

**리스트(list)** : 데이터를 일렬로 늘어놓은 자료구조

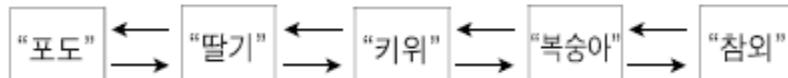
-리스트로 사용할 수 있는 클래스 : ArrayList 클래스와 LinkedList 클래스

ArrayList 객체



ArrayList 클래스는 내부에 있는 배열에 데이터를 저장합니다.

LinkedList 객체



LinkedList 클래스는 인접 데이터가 서로 가리키는 식으로 데이터를 저장합니다.

## ◆ 자료구조 클래스의 사용 방법

```
ArrayList<String> list = new ArrayList<String>();
```

ArrayList 객체를 대입할 변수도 이렇게 선언해야 합니다.

String 객체를 담을 수 있는 ArrayList 객체를 생성합니다.

타입 파라미터(type parameter)

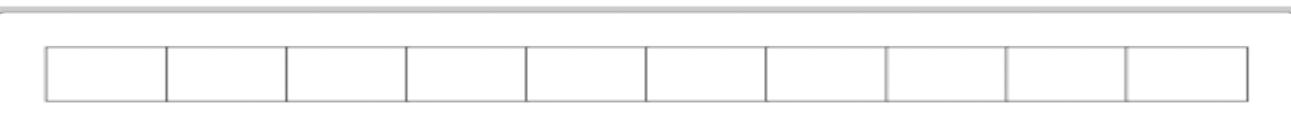
```
ArrayList<String> list = new ArrayList<String>();
```

타입 파라미터

타입 파라미터

ArrayList 객체 안에 String 객체 10개를 담을 수 있는 배열이 생성됩니다.

ArrayList 객체



## ◆ 자료구조 클래스의 사용 방법

-타입 파라미터에 의해 저장 데이터의 타입이 제한됨

list.add("포도");

String 타입의 데이터를  
추가하는 것은 가능합니다.

list.add(new Integer(52));

Integer 타입의 데이터를  
추가하는 것은 불가능합니다.

ArrayList 객체

“포도”	“딸기”	“복숭아”							
------	------	-------	--	--	--	--	--	--	--

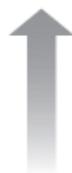
타입 파라미터와 맞는  
데이터 값을 넘겨주어야 합니다.

add 메소드 호출 순서대로  
데이터가 저장됩니다.

## ◆ 데이터를 가져오는 방법

```
String str = list.get(2);
```

ArrayList 객체



인덱스 2 위치에 있는  
"복숭아"를 리턴합니다.

“포도”	“딸기”	“복숭아”							
------	------	-------	--	--	--	--	--	--	--

0      1      2

## ◆ 데이터 수를 가져오는 방법

```
int num = list.size();
```

ArrayList 객체



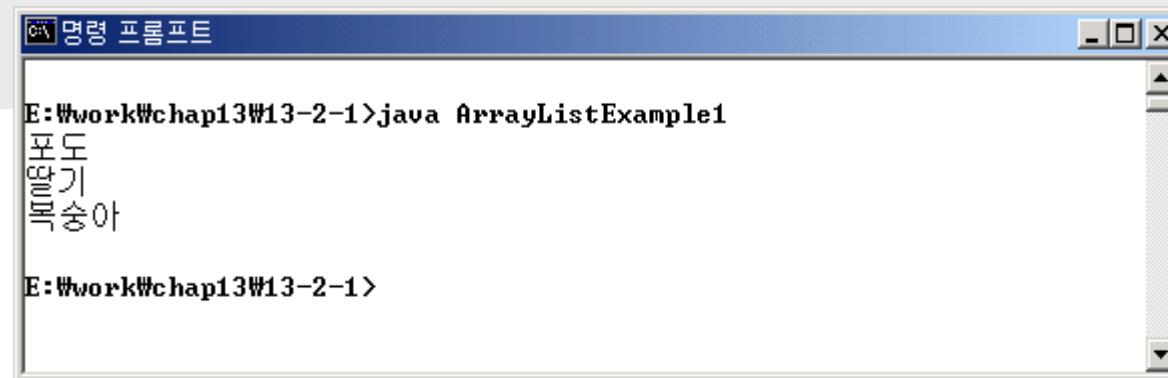
리스트에 있는 데이터의 수  
3을 리턴합니다.

“포도”	“딸기”	“복숭아”							
------	------	-------	--	--	--	--	--	--	--

```

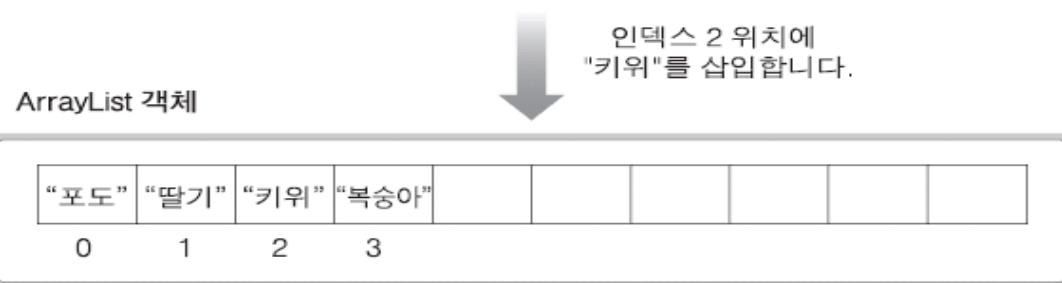
1 import java.util.*;
2 class ArrayListExample1 {
3     public static void main(String args[]) {
4         ArrayList<String> list = new ArrayList<String>(); ----- ArrayList 객체를 생성합니다.
5         list.add("포도");
6         list.add("딸기"); } } 리스트에 3개의 데이터를 추가
7         list.add("복숭아"); 합니다.
8         int num = list.size();
9         for (int cnt = 0; cnt < num; cnt++) { } } 리스트에 있는 데이터의 수만큼
10            String str = list.get(cnt); 루프를 돌면서 데이터를 읽어와서
11            System.out.println(str); 출력합니다.
12        }
13    }
14 }

```



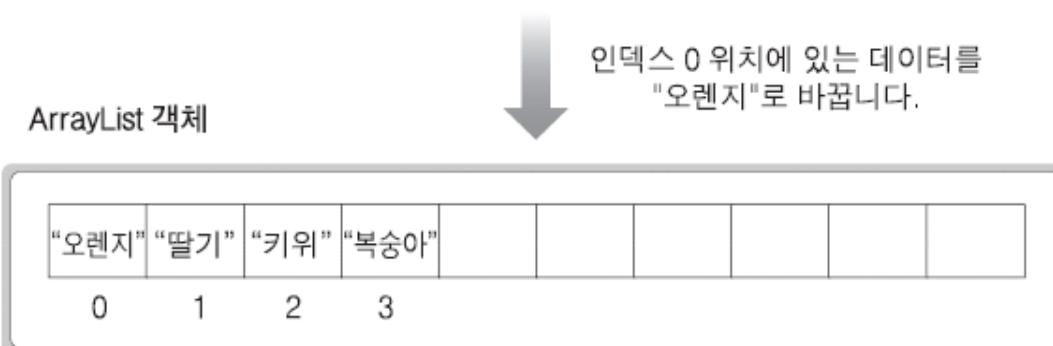
## ◆ 데이터를 중간에 삽입하는 방법

```
list.add(2, "키위");
```



## ◆ 기존 데이터를 교체하는 방법

```
list.set(0, "오렌지");
```



## ◆ 데이터를 삭제하는 방법

list.remove(1);

ArrayList 객체

인덱스 1 위치에 있는  
데이터를 삭제합니다.

“오렌지”	“키위”	“복숭아”						
0	1	2						

0 1 2

list.remove("키위");

LinkedList 객체

리스트에서 "키위"라는  
데이터를 찾아서 삭제합니다.

“오렌지”	“복숭아”							
0	1							

0 1

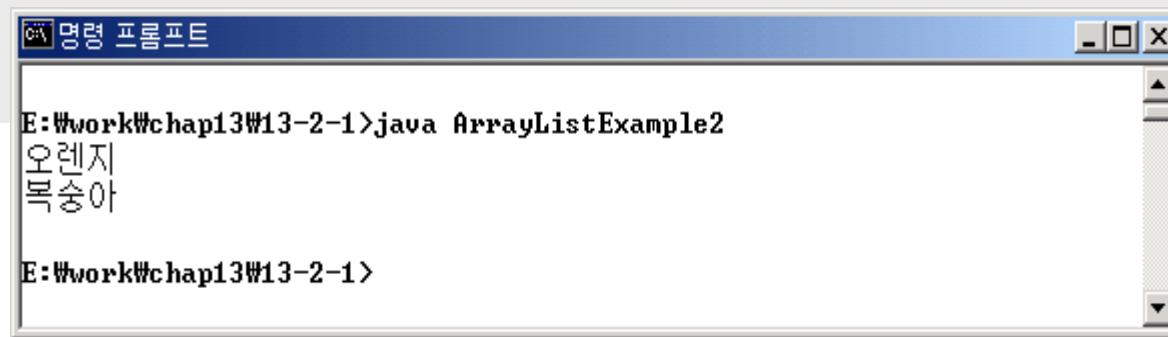
```

1 import java.util.*;
2 class ArrayListExample2 {
3     public static void main(String args[]) {
4         ArrayList<String> list = new ArrayList<String>();
5         list.add("포도");
6         list.add("딸기");
7         list.add("복숭아");
8         list.add(2, "키위");
9         list.set(0, "오렌지");
10        list.remove(1);
11        list.remove("키위");
12        int num = list.size();
13        for (int cnt = 0; cnt < num; cnt++) {
14            String str = list.get(cnt);
15            System.out.println(str);
16        }
17    }
18 }

```

add, set, remove 메소드를 이용하여  
리스트에 데이터를 삽입/수정/삭제

리스트의 데이터를 순서대로  
가져와서 출력



## ◆ 데이터를 검색하는 방법

```
int index = list.indexOf("사과");
```

ArrayList 객체

↑  
첫번째 "사과"의  
위치 1을 리턴합니다.

"머루"	"사과"	"앵두"	"자두"	"사과"					
0	1	2	3	4					

0 1 2 3 4

```
int index = list.lastIndexOf("사과");
```

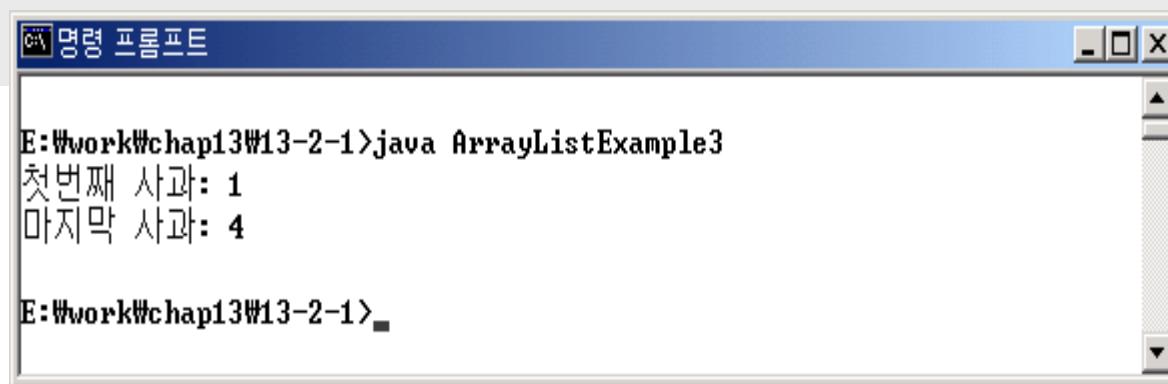
ArrayList 객체

↑  
마지막 "사과"의  
위치 4를 리턴합니다.

"머루"	"사과"	"앵두"	"자두"	"사과"					
0	1	2	3	4					

0 1 2 3 4

```
1 import java.util.*;
2 class ArrayListExample3 {
3     public static void main(String args[]) {
4         ArrayList<String> list = new ArrayList<String>();
5         list.add("머루");
6         list.add("사과");
7         list.add("앵두");
8         list.add("자두"); } } } } }
9 list.add("사과"); } } } }
10 int index1 = list.indexOf("사과");
11 int index2 = list.lastIndexOf("사과");
12 System.out.println("첫번째 사과: " + index1);
13 System.out.println("마지막 사과: " + index2);
14 }
15 }
```



```
1 import java.util.*;
2 class LinkedListExample1 {
3     public static void main(String args[]) {
4         LinkedList<String> list = new LinkedList<String>(); ----- LinkedList 객체를 생성
5         list.add("포도");
6         list.add("딸기");
7         list.add("복숭아");
8         int num = list.size();
9         for (int cnt = 0; cnt < num; cnt++) {
10             String str = list.get(cnt);
11             System.out.println(str);
12         }
13     }
14 }
```



---

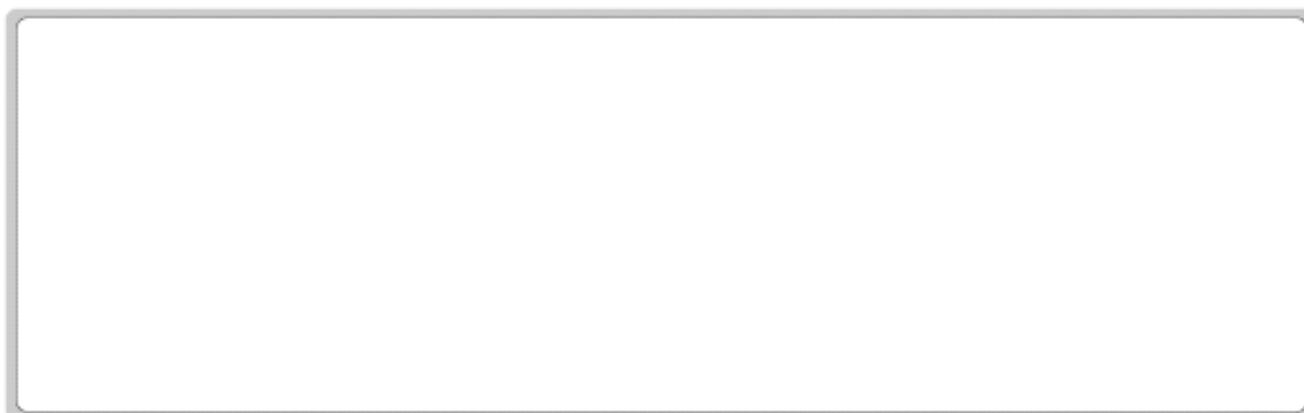
## ◆ LinkedList 객체를 생성할 때 일어나는 일

```
LinkedList<String> list = new LinkedList<String>();
```

LinkedList 객체



LinkedList 객체를 생성한다고 해서  
데이터 저장 영역이 생기지는 않습니다.



## ◆ add 메소드를 처음으로 호출할 때 일어나는 일

list.add("포도");  
↓  
LinkedList 객체

처음으로 add 메소드 호출하면  
그 데이터가 저장됩니다.

“포도”

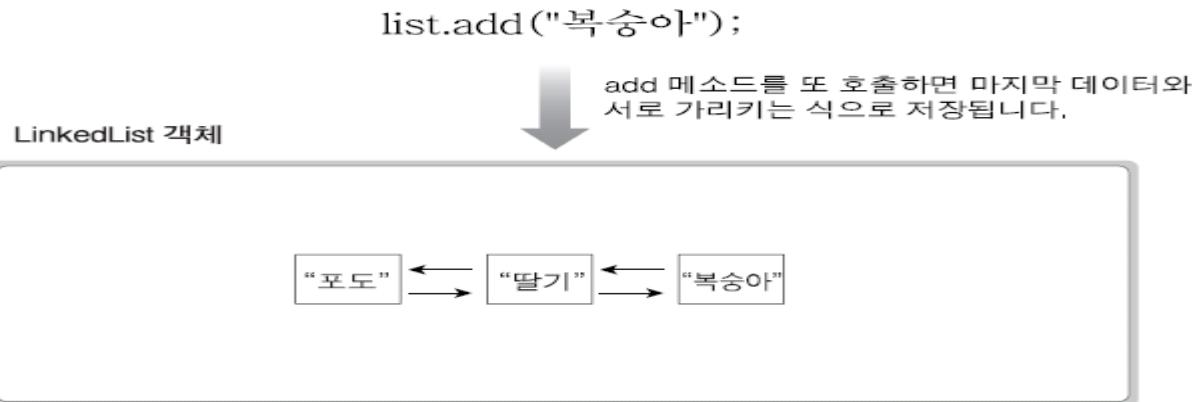
## ◆ add 메소드를 두번째로 호출할 때 일어나는 일

list.add("딸기");  
↓  
LinkedList 객체

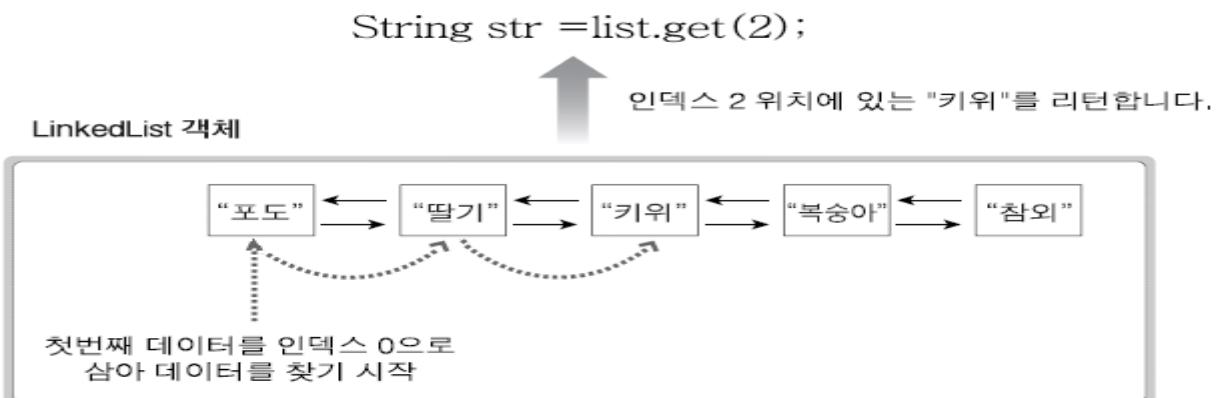
add 메소드를 또 호출하면 마지막 데이터와  
서로 가리키는 식으로 저장됩니다.

“포도” ← → “딸기”

## ◆ add 메소드를 세번째로 호출할 때 일어나는 일

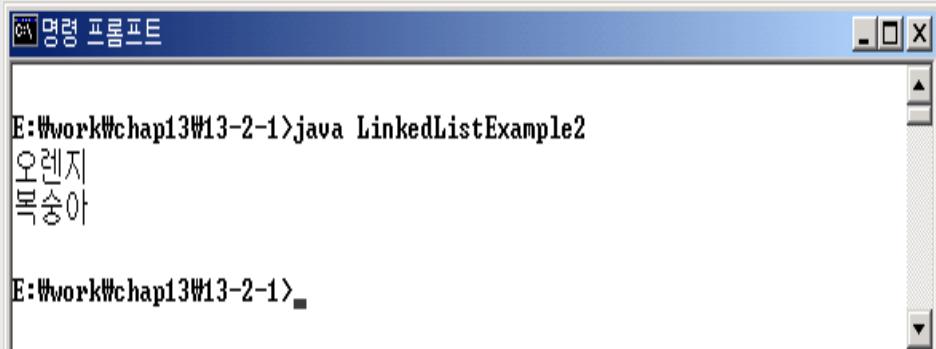


## ◆ get 메소드를 호출할 때 일어나는 일



## ◆ LinkedList에 데이터를 삽입/수정/삭제하는 예

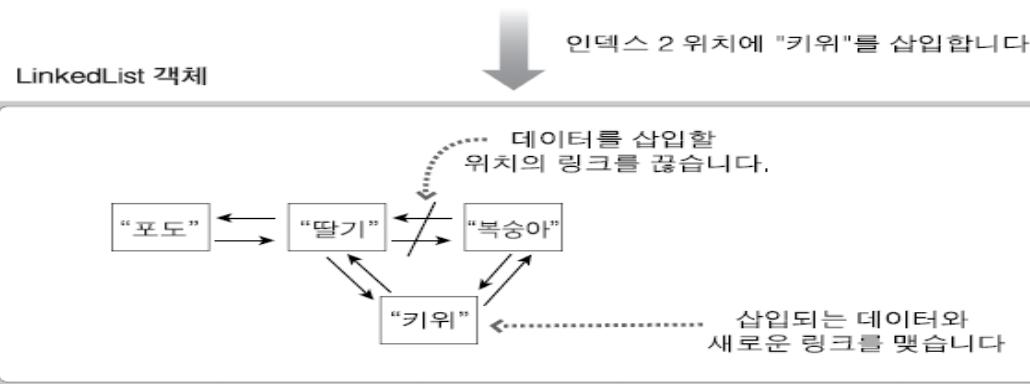
```
1 import java.util.*;
2 class LinkedListExample2 {
3     public static void main(String args[]) {
4         LinkedList<String> list = new LinkedList<String>();           LinkedList 객체를 생성
5         list.add("포도");
6         list.add("딸기");
7         list.add("복숭아");
8         list.add(2, "키위");
9         list.set(0, "오렌지");
10        list.remove(1);
11        list.remove("키위");
12        int num = list.size();
13        for (int cnt = 0; cnt < num; cnt++) {
14            String str = list.get(cnt);
15            System.out.println(str);
16        }
17    }
18 }
```



E:\work\chap13\13-2-1>java LinkedListExample2  
오렌지  
복숭아  
E:\work\chap13\13-2-1>

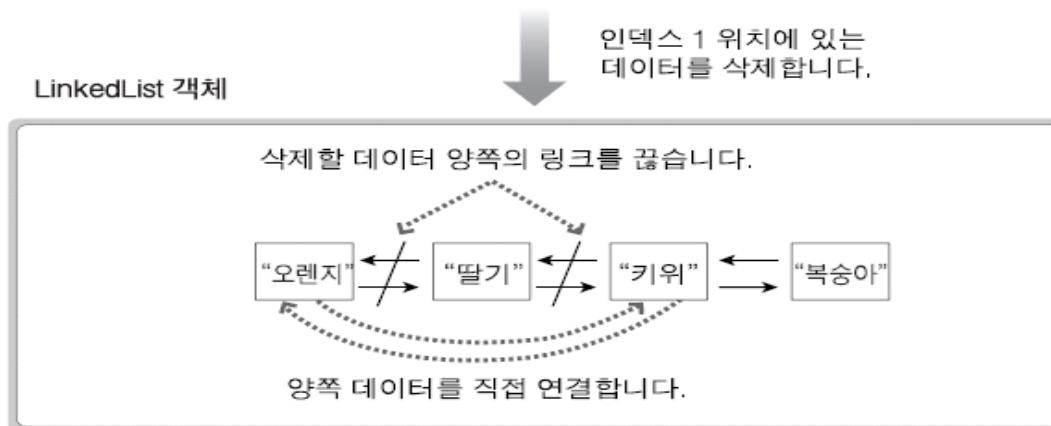
## ◆ 데이터를 중간에 삽입할 때 일어나는 일

list.add(2, "키위");



## ◆ 데이터를 중간에서 삭제할 때 일어나는 일

list.remove(1);



## ◆ 데이터 순차 접근을 효율적으로 하는 방법

- ✓ 1) iterator 메소드를 호출합니다.

```
Iterator<String> iterator = list.iterator();
```



- ✓ 2) Iterator 객체에 대해 next 메소드를 호출합니다.

```
String str = iterator.next();
```



next 메소드는 더 이상 데이터가 없으면  
NoSuchElementException을 발생

```
while (iterator.hasNext()) {  
    String str = iterator.next();  
}
```

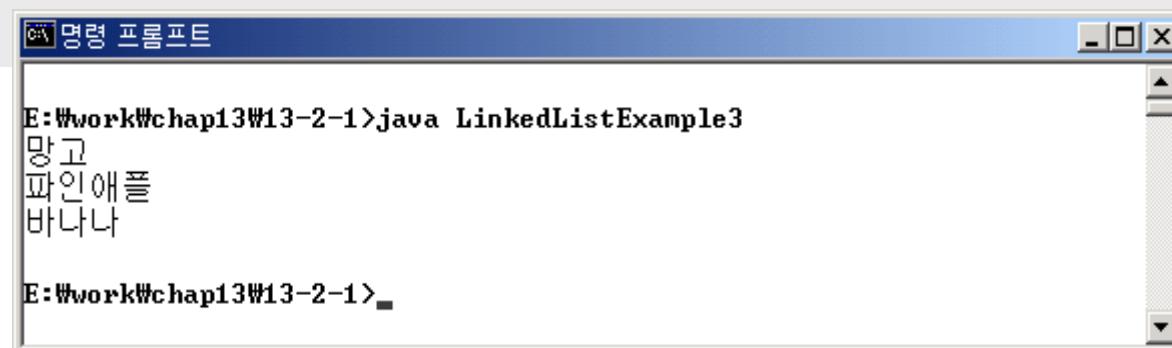
hasNext 메소드의 리턴 값이 true일 동안  
next 메소드를 반복 호출합니다.  
next 메소드로 가져온  
데이터를 처리하는 부분

```
1 import java.util.*;
2 class LinkedListExample3 {
3     public static void main(String args[]) {
4         LinkedList<String> list = new LinkedList<String>();
5         list.add("망고");
6         list.add("파인애플");
7         list.add("바나나");
8         Iterator<String> iterator = list.iterator();
9         while (iterator.hasNext()) {
10             String str = iterator.next();
11             System.out.println(str);
12         }
13     }
14 }
```

LinkedList 객체를 생성하여  
3개의 데이터를 저장

iterator 메소드를 호출하여  
Iterator 객체를 얻음

Iterator 객체를 이용하여 리스트에 있는  
데이터를 순서대로 가져와서 출력



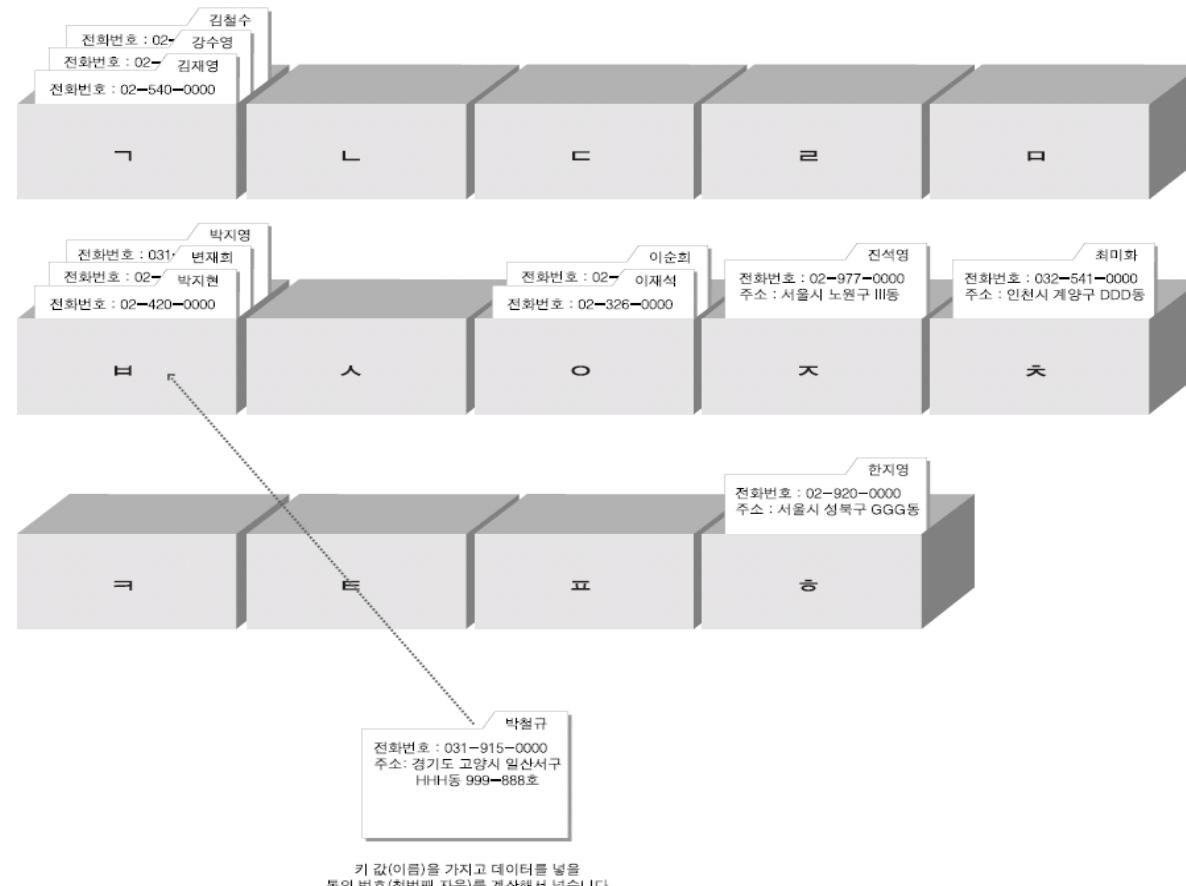
## ◆ 향상된 for 문으로 리스트를 사용하는 방법

```
변수 타입    변수 이름    리스트 객체  
↓           ↓           ↓  
for (String str : list) {  
    -----  
    }  
                                         └── 반복 실행 부분
```

# 자료구조 - map

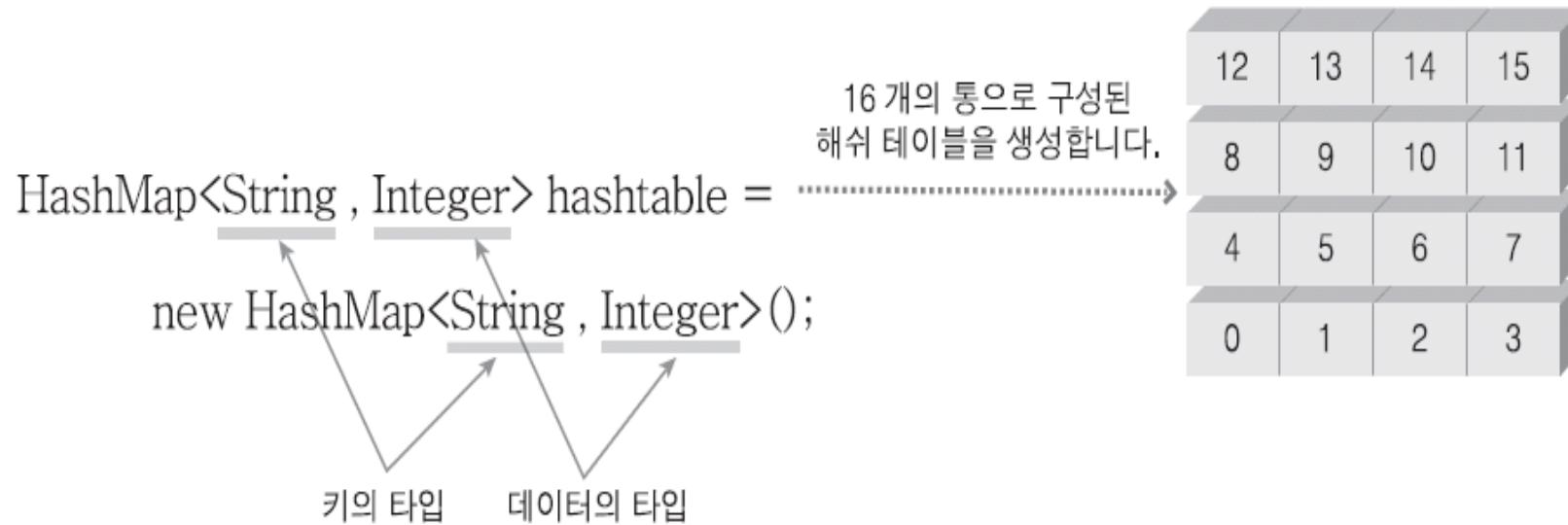
## ◆ 해쉬 테이블(hash table)

- ✓ 여러 개의 통(bucket)을 만들어 두고 키 값을 이용하여 데이터를 넣을 때 번호를 계산



## ◆ HashMap

- 해쉬 테이블로 사용할 수 있는 클래스 : HashMap 클래스
- 해쉬 테이블 생성 방법



---

## ◆ 해쉬 테이블 : **HashMap** 클래스

-100 개의 통으로 구성된 해쉬 테이블 생성하기

```
HashMap<String , Integer> hashtable = new HashMap<String , Integer>(100);
```

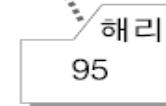
  
100 개의 통으로 구성된  
해쉬 테이블을 생성합니다.

## ◆ 데이터 넣기

hashtable.put("해리", new Integer(95));



키 값("해리")으로 통 번호(5)를  
계산하여 그 통에 키 값과  
데이터를 넣습니다.



## ◆ 데이터 찾기

Integer num = hashtable.get("해리");



키 값으로 통 번호를 계산하고, 그 통 안에서  
키 값과 일치하는 데이터를 찾아서 리턴합니다.

## ◆ 데이터 삭제하기

hashtable.remove("해리");

↑  
키 값

키 값으로 통 번호를 계산하고,  
해당 통에서 키 값과 일치하는  
데이터를 찾아서 삭제합니다.

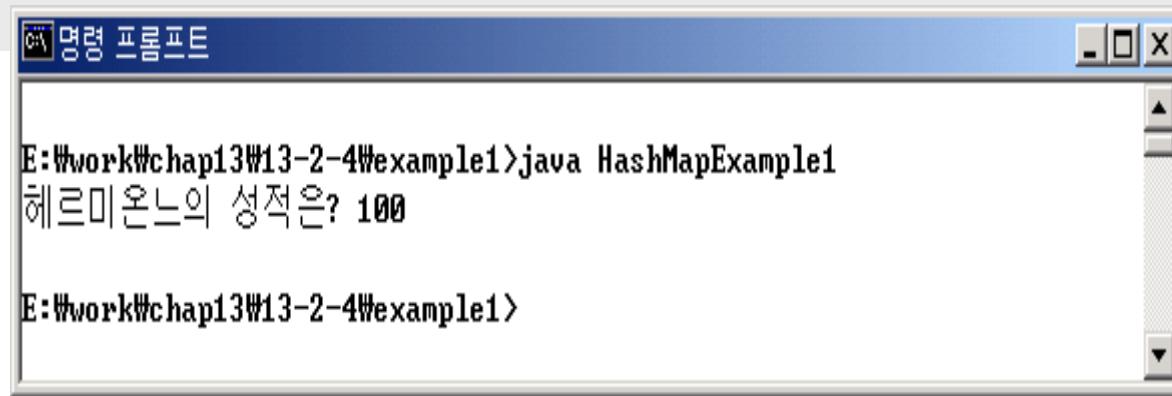


```
1 import java.util.*;
2 class HashMapExample1 {
3     public static void main(String args[]) {
4         HashMap<String, Integer> hashtable = new HashMap<String, Integer>();
5         hashtable.put("해리", new Integer(95));
6         hashtable.put("헤르미온느", new Integer(100));
7         hashtable.put("론", new Integer(85));
8         hashtable.put("드레이코", new Integer(93));
9         hashtable.put("네빌", new Integer(70));
10        Integer num = hashtable.get("헤르미온느");
11        System.out.println("헤르미온느의 성적은? " + num);
12    }
13 }
```

해시 테이블로 사용할  
HashMap 객체를 생성

해시 테이블에 5개의 데이터를  
추가

키 값으로 해시 테이블의 데이터를  
찾아서 출력

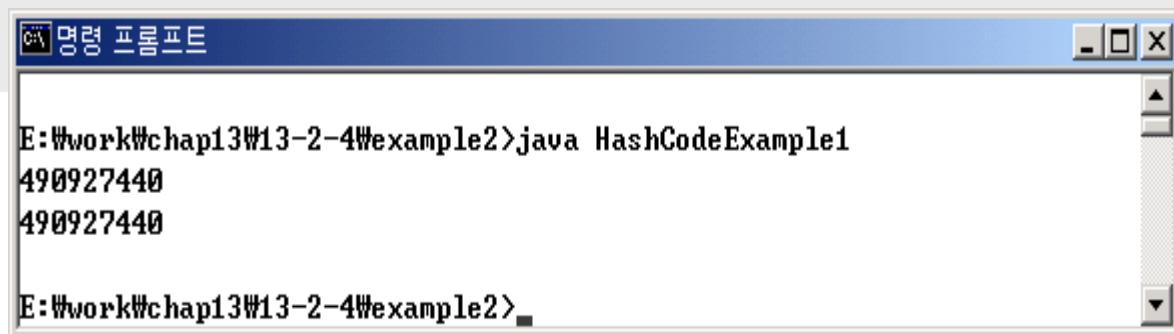


---

## ◆ 해쉬 테이블

- 키값을 가지고 해쉬 테이블의 통번호를 계산하는 공식은?
  - ✓ 프로그래머가 알 필요가 없음
  
- 하지만 다음 사실은 꼭 알아두어야 함
  - ✓ 해쉬 테이블 계산에는 hashCode 메소드가 사용됨
  
  - ✓ 키를 문자여로 사용하면  
그 문자열(String 객체)에 대해 hashCode 메소드가 호출됨

```
1 class HashCodeExample1 {  
2     public static void main(String args[]) {  
3         String obj1 = new String("헤르미온느"); } } 두 개의 String 객체를 생성  
4         String obj2 = new String("헤르미온느"); } } 각각의 객체에 대해 hashCode 메소드를 호출  
5     int hash1 = obj1.hashCode(); } } hashCode 메소드의 리턴 값을 출력  
6     int hash2 = obj2.hashCode(); } }  
7     System.out.println(hash1); }  
8     System.out.println(hash2);  
9 }  
10 }
```



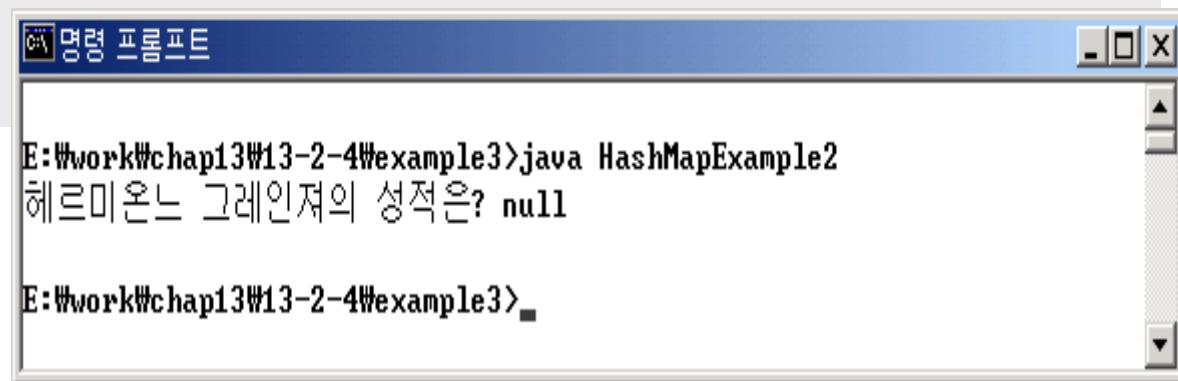
---

## ◆ hashCode 메소드

- 성과 이름을 모두 키로 사용하고 싶을 때는?
  - ✓ 다음과 같이 직접 선언한 클래스를 사용하면 됩니다.
- 사람의 이름을 표현하는 클래스

```
1  class Name {  
2      String firstName;      // 이름  
3      String lastName;       // 성  
4      Name(String firstName, String lastName) {  
5          this.firstName = firstName;  
6          this.lastName = lastName;  
7      }  
8  }
```

```
1 import java.util.*;
2 class HashMapExample2 {
3     public static void main(String args[]) {
4         HashMap<Name, Integer> hashtable = new HashMap<Name, Integer>();
5         hashtable.put(new Name("해리", "포터"), new Integer(95));
6         hashtable.put(new Name("헤르미온느", "그레인저"), new Integer(100));
7         hashtable.put(new Name("론", "위즐리"), new Integer(85));
8         hashtable.put(new Name("드레이코", "말포이"), new Integer(93));
9         hashtable.put(new Name("네빌", "魯버頓"), new Integer(70));
10        Integer num = hashtable.get(new Name("헤르미온느", "그레인저"));
11        System.out.println("헤르미온느 그레인저의 성적은? " + num);
12    }
13 }
```



원인? & 해결?

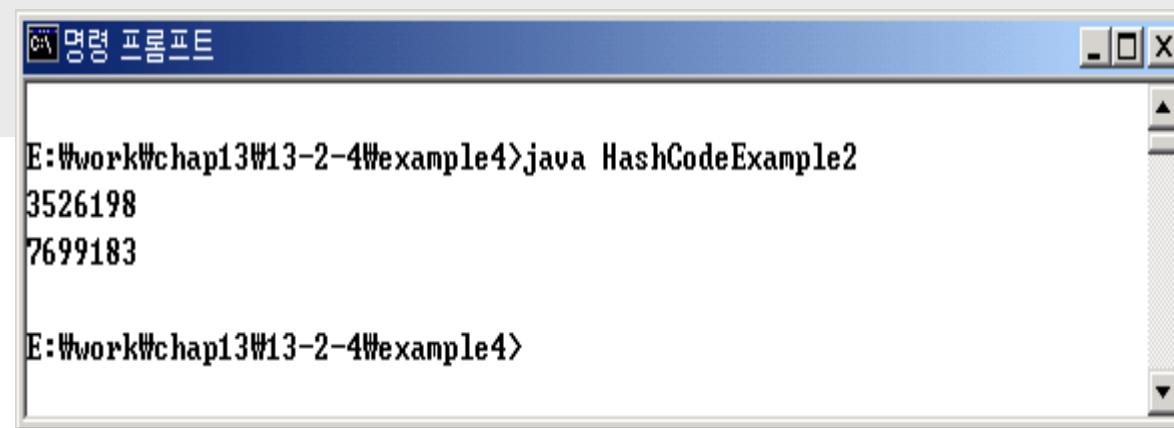
## ◆ Name 클래스에 대해 hashCode 메소드를 호출하는 프로그램

```
1  class HashCodeExample2 {  
2      public static void main(String args[]) {  
3          Name obj1 = new Name("헤르미온느", "그레인저"); }  
4          Name obj2 = new Name("헤르미온느", "그레인저"); }  
5          int hash1 = obj1.hashCode(); }  
6          int hash2 = obj2.hashCode(); }  
7          System.out.println(hash1); }  
8          System.out.println(hash2); }  
9      }  
10 }
```

두 개의 Name 객체를 생성

각각의 객체에 대해 hashCode 메소드를 호출

hashCode 메소드의 리턴 값을 출력



## ◆ hashCode 메소드의 오버라이드 방법

```
public int hashCode() {  
    return firstName.hashCode() + lastName.hashCode();  
}
```

String 타입 필드의 hashCode 메소드를  
가지고 리턴 값을 계산하면 좀 더 고른  
분포를 갖는 리턴 값을 만들 수 있습니다.

바람직한 방법

```
명령 프롬프트
E:\work\chap13\13-2-4\example6>java HashMapExample2
해르미온느 그레인저의 성적은? 100
E:\work\chap13\13-2-4\example6>

1  class Name {
2      String firstName;
3      String lastName;
4      Name(String firstName, String lastName) {
5          this.firstName = firstName;
6          this.lastName = lastName;
7      }
8      public boolean equals(Object obj) {
9          if (!(obj instanceof Name))
10             return false;
11          Name name = (Name) obj;
12          if (firstName.equals(name.firstName) && lastName.equals(name.lastName))
13              return true;
14          else
15              return false;
16      }
17      public int hashCode() {
18          return firstName.hashCode() + lastName.hashCode();
19      }
20  }
```

추가된 equals 메소드

# 자료구조 - Set

## ◆ 집합

- 집합(Set) : 수학에서 말하는 집합처럼 데이터를 중복 저장하지 않음
- 집합으로 사용할 수 있는 클래스 : HashSet 클래스

## ◆ 집합 : HashSet 클래스

- 집합 생성 방법

```
HashSet<String> set = new HashSet<String>();
```



## ◆ 데이터 추가 방법

```
set.add("자바");  
set.add("카푸치노");  
set.add("에스프레소");
```

집합에 데이터를 저장합니다.



이미 있는 데이터를 저장하면  
집합에 변동이 일어나지 않습니다.  
set.add("자바");

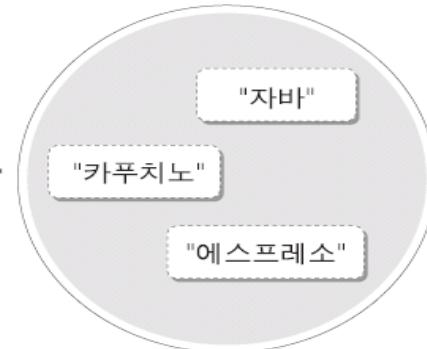
집합에 데이터를 저장하면  
집합에 변동이 일어나지 않습니다.



## ◆ 데이터의 수를 가져오는 방법

```
int num = set.size();
```

집합에 있는 데이터의 수  
3을 리턴합니다.



## ◆ 모든 데이터를 읽어오는 방법

```
Iterator<String> iterator = set.iterator();
while (iterator.hasNext()) {
    String str = iterator.next();
}
```

데이터 처리 부분

iterator 메서드를 호출하여  
Iterator 객체를 가져옵니다.

Iterator 객체의 데이터를  
순서대로 가져와서 처리합니다.

```

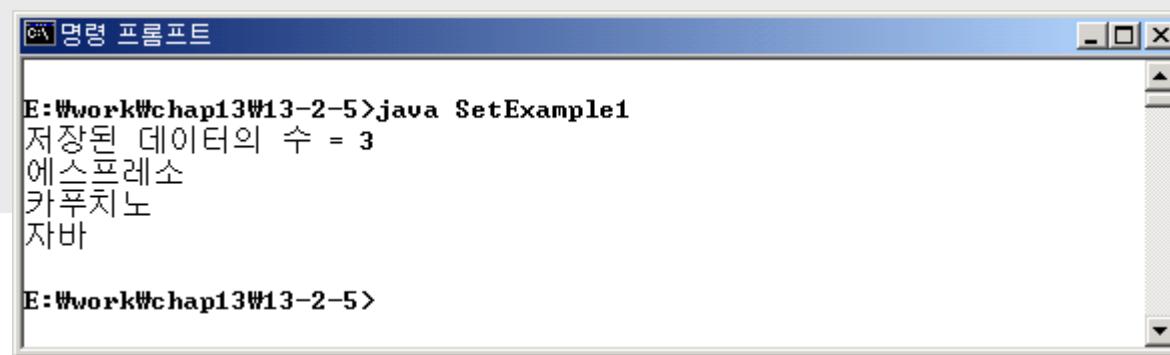
1 import java.util.*;
2 class SetExample1 {
3     public static void main(String args[]) {
4         HashSet<String> set = new HashSet<String>();
5         set.add("자바");
6         set.add("카푸치노");
7         set.add("에스프레소");
8         set.add("자바");
9         System.out.println("저장된 데이터의 수 = " + set.size());
10        Iterator<String> iterator = set.iterator();
11        while (iterator.hasNext()) {
12            String str = iterator.next();
13            System.out.println(str);
14        }
15    }
16 }

```

집합으로 사용할 HashSet 객체를  
생성

집합에 데이터를 저장

집합에 있는 데이터를  
모두 가져와서 출력



# 자료구조 - 열거자

---

- ◆ 데이터를 순서대로 나열
- ◆ 데이터를 순서대로 추출할 수 있는 기능만을 함
- ◆ Enumeration과 Iterator인터페이스를 정의하여 사용
- ◆ Vector, Hashtable등의 객체저장을 위한 클래스와 함께 사용
  - Enumeration의 사용법 - 동기화 처리됨

```
Enumeration enum = vector.elements();
while(enum.hasMoreElements()){
    String temp = (String)enum.nextElement();
}
```

- Iterator의 사용법

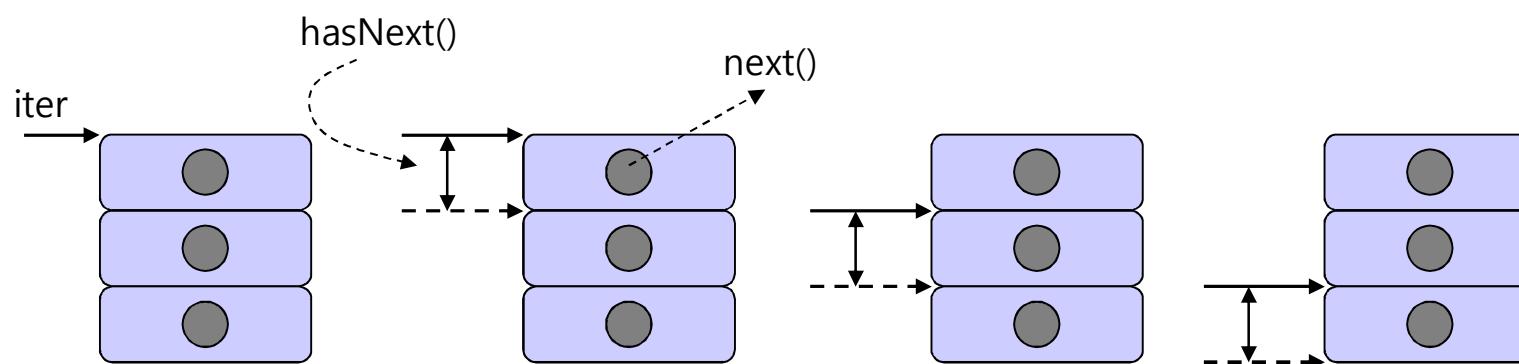
```
Iterator itr = vector.iterator();
while(itr.hasNext()){
    String temp = (String)itr.next();
}
```

# Iterator

---

## ◆ Iterator

- JFC의 Set, List 계열에 대입된 모든 것을 얻기 위해
- Set, List 계열에 대입된 모든 것을 Iterator로 넘기면 iterator메서드를 사용하여 추출할 수 있다
- Iterator 는 인덱스 0번보다 위쪽을 가리킨다.
- 인덱스의 개수를 모르므로 while를 사용.
- hasNext는 1칸 이동하는데 데이터가 있으면 true 없으면 false 반환



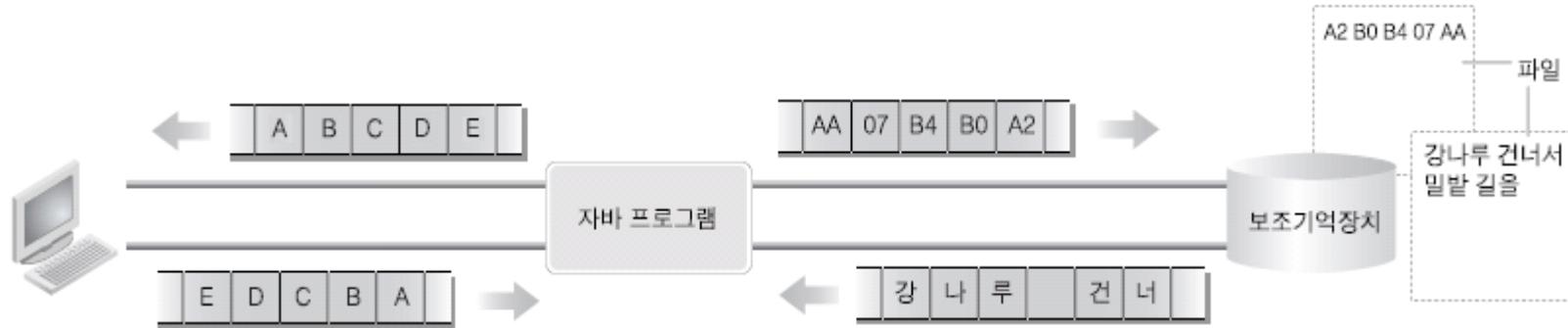
```
HashSet set=new HashSet();
Iterator iter=set.iterator();
while(iter.hasNext()){
    String str=(String)iter.next();
    ...
}
```

# 파일 입출력에 사용되는 자바 클래스들

# 파일 관련 클래스

## ◆ 스트림이란?

- 일차원적인 데이터의 흐름



- 흐름의 방향에 따른 분류

- 입력 스트림(input stream)
- 출력 스트림(output stream)

- 데이터의 형태에 따른 분류

- 문자 스트림(character stream)
- 바이트 스트림(byte stream)

# 파일 입출력

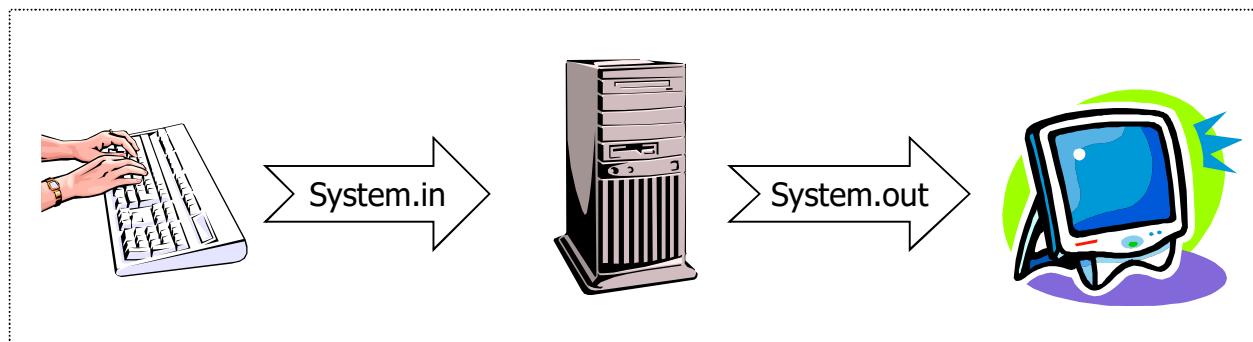
## ◆ 입력

- 데이터를 안으로 읽어 들이는 것
- System.in
  - 키보드의 입력을 담당

## ◆ 출력

- 데이터를 밖으로 기록하는 것
- System.out
- Console화면에 출력

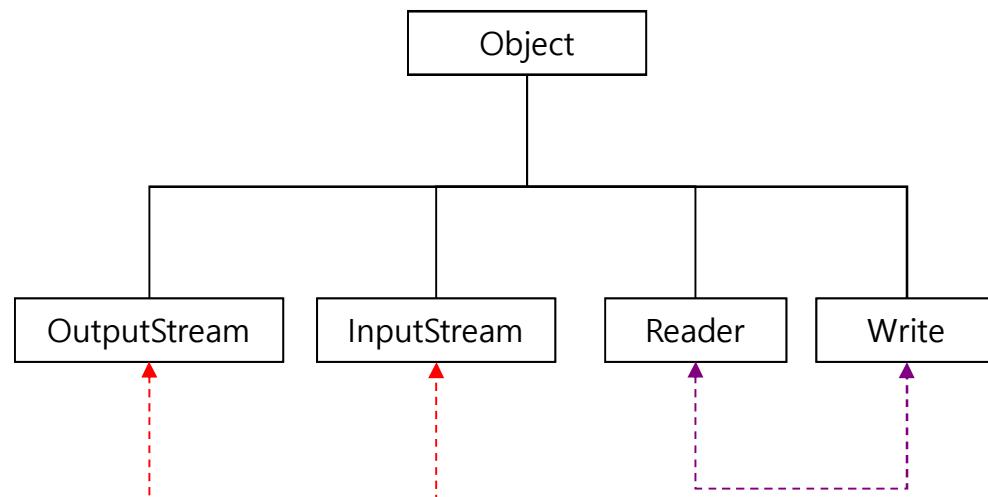
시작점( 소스, source )과 종료점( 싱크, sink )



# 파일 관련 클래스-스트림의 분류

- 기본적인 분류
  - 입력 스트림
    - ex) InputStream, Reader
  - 출력 스트림
    - ex) OutputStream, Writer
- 실제적인 분류
  - 바이트 스트림
    - 바이트, 바이트 배열, 정수, 데이터 등의 흐름
    - 1byte씩 처리
      - ex) InputStream, OutputStream
  - 문자 스트림
    - 문자, 문자 배열, 문자열의 흐름
    - 2byte씩 처리
      - ex) Reader, Writer

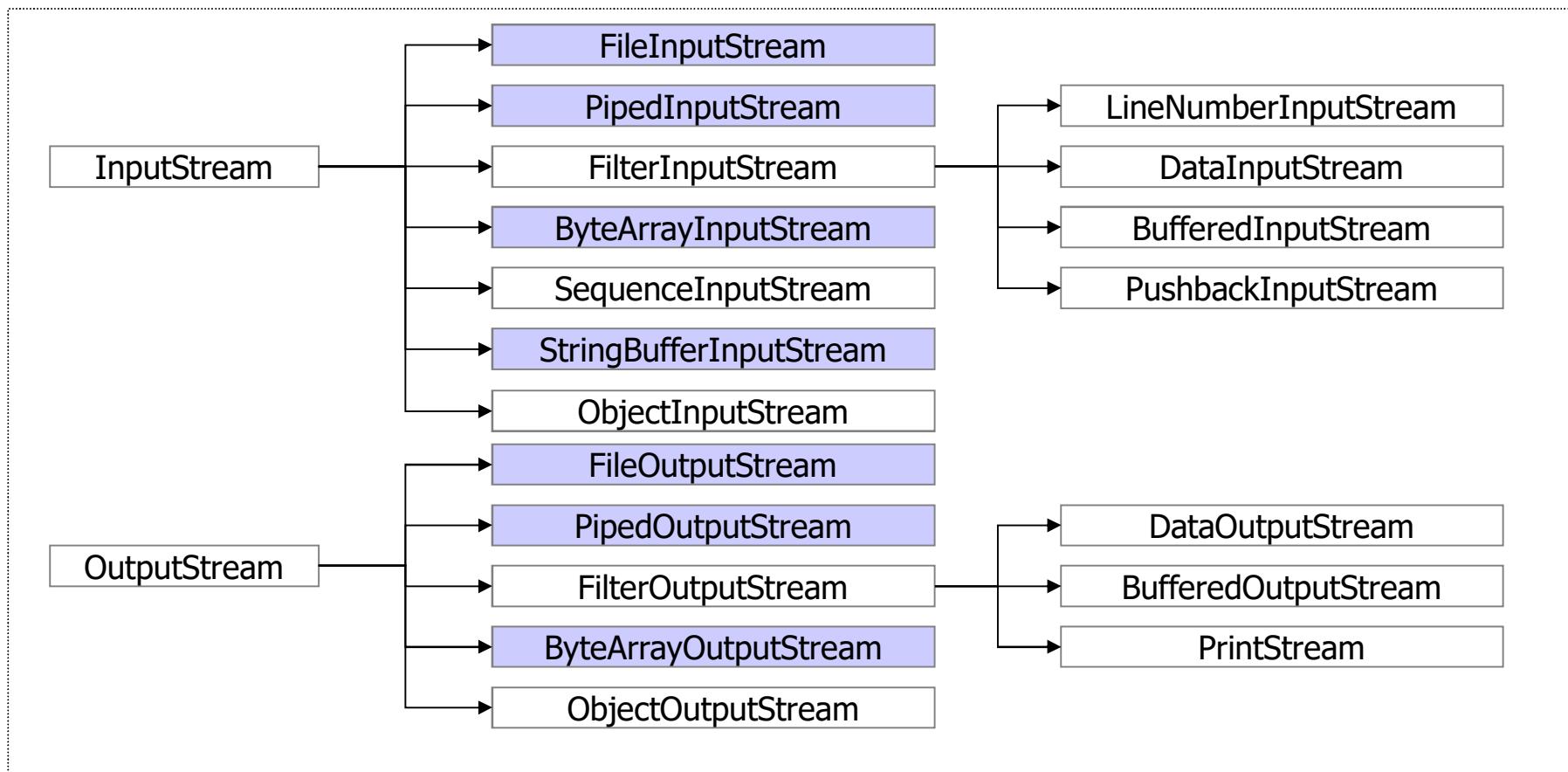
	입력	출력
바이트	<b>InputStream</b>	<b>OutputStream</b>
문자	<b>Reader</b>	<b>Writer</b>



# 파일 관련 클래스- 바이트 스트림

## ◆ 바이트 스트림

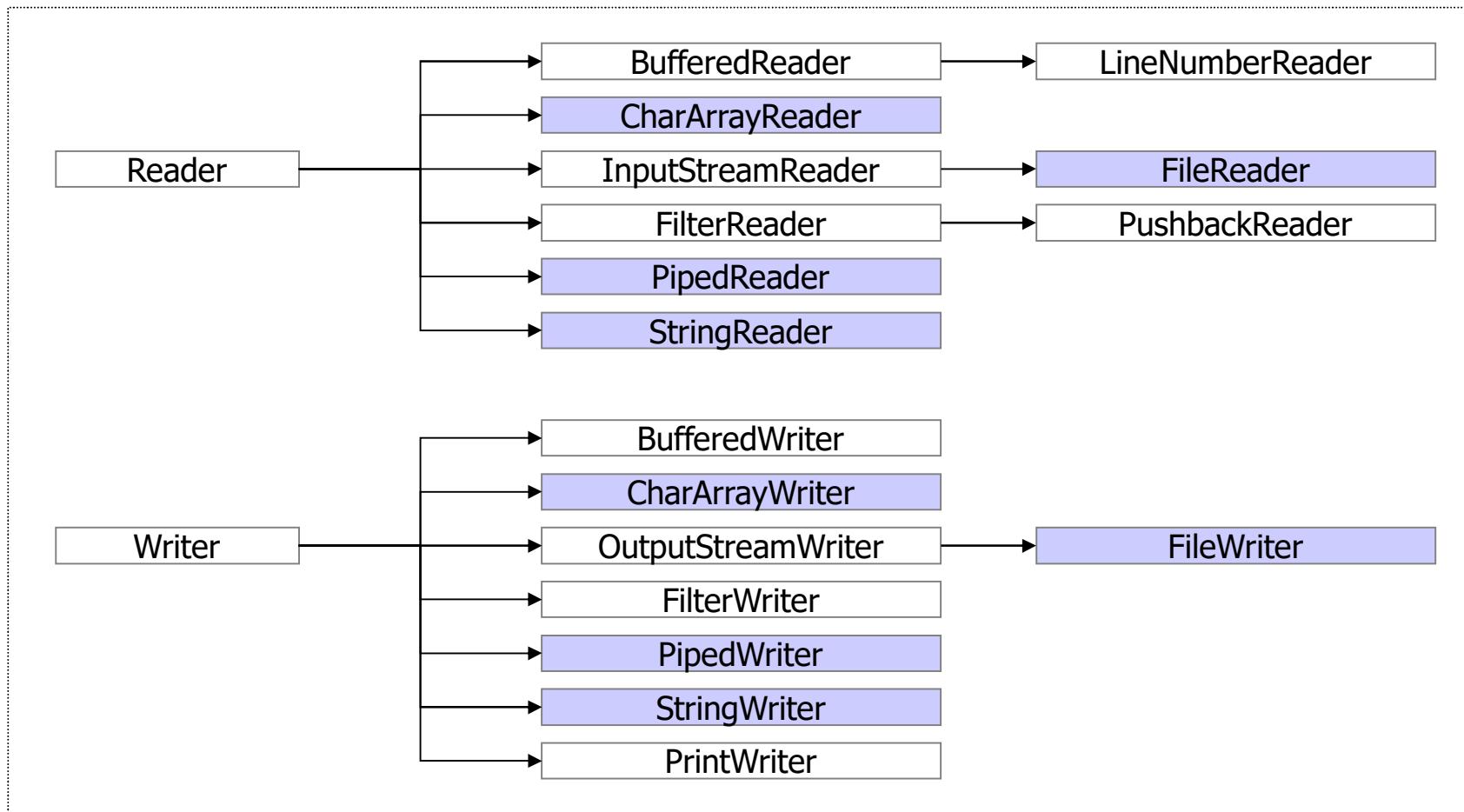
- 1byte씩 처리
- InputStream
- OutputStream



# 파일 관련 클래스- 문자 스트림

## ◆ 문자 스트림

- Reader : 입력문자 스트림
- Writer : 출력문자 스트림



# 파일 관련 클래스 – 표준 입력

## ◆ 표준 입력 필드 : in

- 키보드로부터 입력을 받는 방법

```
InputStreamReader reader = new InputStreamReader(System.in);
```

System 클래스의 in 필드를 InputStreamReader 클래스의  
생성자 파라미터로 넘겨주어야 합니다.

```
char ch = (char) reader.read();
```

read 메서드의 리턴 값은 int 타입이므로  
char 타입으로 캐스트해야 합니다.

```
1 import java.io.*;
2 class SystemExample1 {
3     public static void main(String args[]) {
4         InputStreamReader reader =
5             new InputStreamReader(System.in);
6         try {
7             while(true) {
8                 char ch = (char) reader.read();
9                 System.out.println("입력된 문자: " + ch);
10                if (ch == '.')
11                    break;
12            }
13        catch (IOException e) {
14            System.out.println("키보드 입력 에러");
15        }
16    }
17 }
```

System 클래스의 in 필드를 가지고 InputStreamReader 객체를 생성합니다.

} 키보드로부터 한 글자씩 입력 받아서 출력하는 일을 마침표(.)가 입력될 때까지 반복합니다.

} read 메서드가 발생하는 IOException을 처리합니다.

```
InputStreamReader reader1 = new InputStreamReader(System.in);
```

InputStreamReader 객체를 BufferedReader 생성자의  
파라미터로 사용해야 합니다.

```
BufferedReader reader2 = new BufferedReader(reader1);
```

```
String str = reader2.readLine();
```

입력 문자를 행 단위로  
읽어들이는 메소드

```
1 import java.io.*;
2 class SystemExample2 {
3     public static void main(String args[]) {
4         BufferedReader reader = new BufferedReader(
5             new InputStreamReader(System.in));           ----- System 클래스의 in 필드를 가지고
6         try {                                         BufferedReader 객체를 생성
7             String str = reader.readLine();          }
8             System.out.println("입력된 문자열: " + str);   키보드로부터 한 줄을 입력받아서 출력
9         } catch (IOException e) {                   }
10            System.out.println("키보드 입력 에러");    }
11        }
12    }
13 }
```

System 클래스의 in 필드를 가지고  
BufferedReader 객체를 생성

} 키보드로부터 한 줄을 입력받아서 출력

} readLine 메서드가 발생하는  
IOException을 처리

# 파일 관련 클래스 – 표준 출력, 에러출력

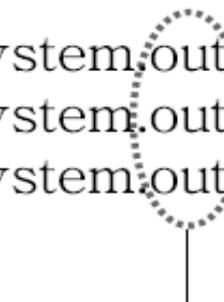
---

## ◆ 표준 출력 필드 : out

- 모니터로 출력을 하는 방법

```
System.out.println("Hello, Java");  
System.out.print(12.5);  
System.out.printf("%d", 27);
```

System 클래스의 out 필드



## ◆ 표준 에러 출력 필드 : err

- err 필드 : 표준 에러 출력(모니터로의 에러 메시지 출력)에 사용되는 필드
- err 필드의 사용 방법
  - ✓ out 필드의 사용 방법과 동일

```
System.err.println("잘못된 포맷입니다.");  
System.err.print(-2.5);  
System.err.printf("%d", 0);
```

```
1  class SystemExample3 {  
2      public static void main(String args[]) {  
3          int arr1[] = { 77, 82, 99, 100, -27, 0, 42, -35, 60, 72 };  
4          int arr2[] = { 7, 0, 3, 0, -1, 2, 11, 5, 0, 9 };  
5          for (int cnt = 0; cnt < arr1.length; cnt++) {  
6              try {  
7                  int result = arr1[cnt] / arr2[cnt];  
8                  System.out.printf("%d / %d = %d %n",  
9                      arr1[cnt], arr2[cnt], result);  
10             }  
11             catch (java.lang.ArithmaticException e) {  
12                 System.err.println("잘못된 연산입니다. (index=" + cnt + ")");  
13             }  
14         }  
15     }
```

나눗셈의 정상적인 결과  
를 out 필드를 통해 출력

나눗셈 중 발생한 에러에  
대한 메시지를 err 필드를  
통해 출력

# 파일 관련 클래스 - File 클래스

## ◆ File 클래스

- File 클래스 : (파일의 내용이 아니라) 파일 자체를 관리하는 클래스
- 다음과 같은 메소드 제공
  - 파일 정보를 가져오는 메소드
  - 파일 정보를 수정하는 메소드
  - 파일을 생성/삭제하는 메소드
- 디렉토리 관리에도 사용됨

```
File file = new File("poem.txt");
```

현재 디렉토리에 있는 poem.txt에 대한  
File 객체를 생성합니다.

```
File file = new File("C:\\work\\chap10");
```

C 드라이브의 work 디렉토리 아래에 있는  
chap10에 대한 File 객체를 생성합니다.

# 파일관련 클래스 -File 클래스

## ◆ 파일/디렉토리 정보 가져오기

```
Boolean isThere = file.exists();
```

파일 또는 디렉토리가 있으면 true,  
없으면 false를 리턴

```
Boolean isFile = file.isFile();
```

파일이면 true,  
아니면 false를 리턴

```
Boolean isDir = file.isDirectory();
```

디렉토리면 true,  
아니면 false를 리턴

```
String name = file.getName();           // 이름을 리턴  
long size = file.length();             // 크기를 리턴  
long time = file.lastModified();       // 최종 수정일시를 리턴  
boolean readMode = file.canRead();     // 읽기 가능 여부를 리턴  
boolean writeMode = file.canWrite();    // 쓰기 가능 여부를 리턴  
boolean hiddenMode = file.isHidden();   // 숨김 여부를 리턴  
String parent = file.getParent();      // 부모 디렉토리 경로명을 리턴
```

```
Files child[] = file.listFiles();
```

서브디렉토리와 파일들의 목록을  
리턴하는 메소드

```
1 import java.io.*;
2 import java.util.*;
3 class FileExample1 {
4     public static void main(String args[]) { 현재 디렉토리 경로명을 가지고 File 객체를 생성
5         File file = new File(".");
6         File arr[] = file.listFiles(); 서브디렉토리와 파일 목록
7         for (int cnt = 0; cnt < arr.length; cnt++) {
8             String name = arr[cnt].getName();
9             if (arr[cnt].isFile())
10                 System.out.printf("%-25s %7d ", name, arr[cnt].length()); } 가져온 서브디렉토리와 파일의
11             else
12                 System.out.printf("%-25s <DIR> ", name);
13             long time = arr[cnt].lastModified();
14             GregorianCalendar calendar = new GregorianCalendar();
15             calendar.setTimeInMillis(time);
16             System.out.printf("%1$tF %1$tT %n", calendar);
17         }
18     }
19 }
```

## ◆ 파일/디렉토리 생성/삭제하기

### -- 파일을 생성하고 삭제하는 메소드

```
{ File file1 = new File("poem.txt");
  file1.createNewFile(); }  
현재 디렉토리에 poem.txt라는  
이름의 파일을 생성합니다.
```

```
{ File file2 = new File("C:\\doc\\회의록.hwp");
  file2.delete(); }  
C:\\doc\\회의록.hwp 파일을  
삭제합니다.
```

### -- 디렉토리를 생성하고 삭제하는 메소드

```
{ File file1 = new File("C:\\울빼미");
  file1.mkdir(); }  
C 드라이브의 루트 디렉토리에  
“울빼미”라는 디렉토리를 생성합니다.
```

```
{ File file2 = new File("두루미");
  file2.delete(); }  
현재 디렉토리에 있는  
“두루미”라는 디렉토리를 삭제합니다
```

## ◆ 임시 파일 생성하기

```
File tmpFile = file.createTempFile("tmp", ".txt", tmpDir);
```

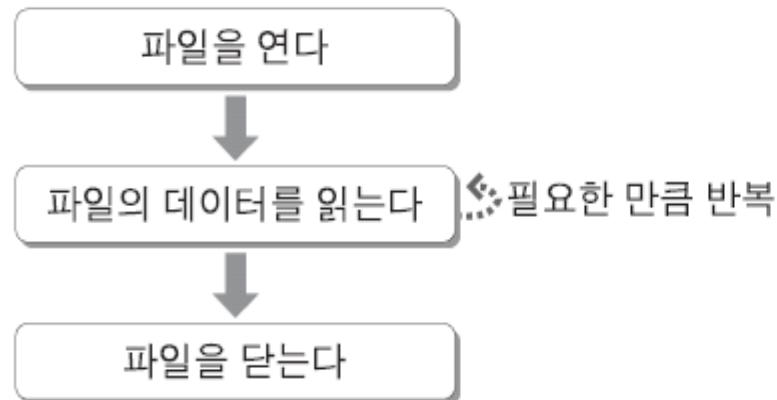
↑      ↑      ↑  
tmp로 시작하고 .txt로 끝나는 임시파일을  
이 디렉토리에 생성합니다.

## ◆ 임시 파일 생성

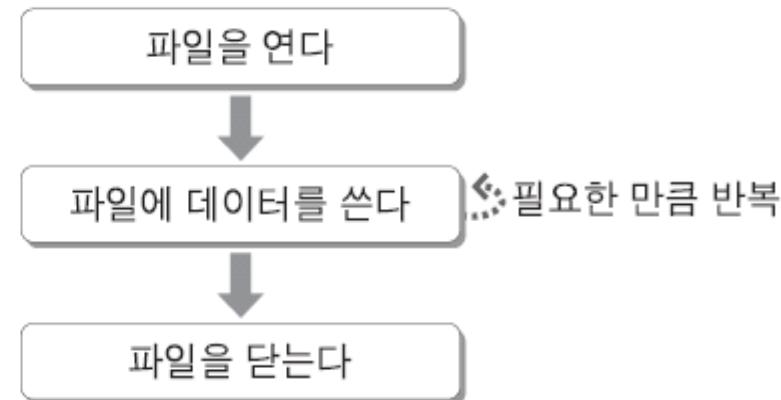
```
1 import java.io.*;
2 class FileExample2 {
3     public static void main(String args[]) {
4         FileWriter writer = null;
5         try {
6             File file = File.createTempFile("tmp", ".txt", new File("C:\temp"));
7             writer = new FileWriter(file);
8             writer.write('자'); }           }   임시 파일을 생성
9             writer.write('바');
10        }
11        catch (IOException ioe) {
12            System.out.println("임시 파일에 쓸 수 없습니다.");
13        }
14        finally {
15            try { writer.close(); }           }   임시 파일을 열어서 데이터를 쓰기
16            catch (Exception e) {
17            }
18        }
19    }
20}
21}
22}
```

---

a) 파일로부터 데이터를 읽는 3단계



b) 파일에 데이터를 쓰는 3단계



# 파일 관련 클래스 - FileOutputStream

## ◆ FileOutputStream 클래스

- FileOutputStream 클래스 : 바이트 데이터를 파일에 쓰는 클래스

```
1 import java.io.*;
2 class OutputStreamExample1 {
3     public static void main(String args[]) {
4         FileOutputStream out = null;
5         try {
6             out = new FileOutputStream("output.dat");           파일 열기
7             byte arr[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
8                 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 };   파일에 반복해서 byte 탑입
9             for (int cnt = 0; cnt < arr.length; cnt++)
10                 out.write(arr[cnt]);                      데이터 쓰기
11         }
12         catch (IOException ioe) {
13             System.out.println("파일로 출력할 수 없습니다.");
14         }
15         finally {
16             try {
17                 out.close();                         파일을 닫기
18             }
19             catch (Exception e) {
20             }
21         }
22     }
23 }
```

# 파일 관련 클래스 - FileInputStream

## ◆ FileInputStream 클래스

-FileInputStream 클래스 : 파일로부터 바이트 단위로 데이터를 읽는 클래스

```
while (true) {  
    int data = inputStream.read(); ─────────── 데이터를 읽어서  
    if (data < 0) } ─────────── -1이면 반복을 종단하고  
    break; ──────────────────  
    byte b = (byte) data; ─────────── 아니면 byte 타입으로 캐스트  
    ...  
}
```

- 한꺼번에 여러 바이트를 읽을 때

```
byte arr = new byte[16];  
  
int num = inputStream.read(arr);
```

byte 타입의 배열을 생성해서 넘겨줘야 합니다.

```
1 import java.io.*;
2 class FileDump {
3     public static void main(String args[]) {
4         if (args.length < 1) {
5             System.out.println("Usage: java FileDump <filename>");
6             return;
7         }
8         FileInputStream in = null;
9         try {
10             in = new FileInputStream(args[0]);
11             byte arr[] = new byte[16];
12             while (true) {
13                 int num = in.read(arr);
14                 if (num < 0)
15                     break;
16                 for (int cnt = 0; cnt < num; cnt++)
17                     System.out.printf("%02X ", arr[cnt]);
18                 System.out.println();
19             }
20         } catch (FileNotFoundException fnfe) {
21             System.out.println(args[0] + " 파일이 존재하지 않습니다.");
22         } catch (IOException ioe) {
23             System.out.println(args[0] + " 파일을 읽을 수 없습니다.");
24         } finally {
25             try { in.close(); } catch (Exception e) { }
26         }
27     }
28 }
```

# 파일 관련 클래스 – FileReader클래스

## ◆ FileReader 클래스

- FileReader 클래스 : 텍스트 파일을 읽는 클래스

```
FileReader reader = new FileReader("poem.txt");
```

생성자 안에서 현재 디렉토리의  
poem.txt 파일을 엽니다.

```
data = reader.read();
```

이 메서드는 파일에 있는  
문자 하나를 읽어서 리턴합니다.



```
while (true) {  
    int data = reader.read(); ─────────── 데이터를 읽어서  
    if (data < 0) } ─────────── 마이너스 값이면 반복을 종단하고,  
    break; ─────────── 아니면 char 타입으로 캐스트합니다.  
    char ch = (char) data; ─────────── 데이터 처리 로직이 들어가는 부분  
}
```

```
reader.close();
```

파일을 닫는 메소드

```
import java.io.*;
class ReaderExample1 {
    public static void main(String args[]) {
        FileReader reader = null;
        try {
            reader = new FileReader("poem.txt");
            while (true) {
                int data = reader.read();
                if (data == -1)
                    break;
                char ch = (char) data;
                System.out.print(ch);
            }
        } catch (FileNotFoundException fnfe) {
            System.out.println("파일이 존재하지 않습니다.");
        } catch (IOException ioe) {
            System.out.println("파일을 읽을 수 없습니다.");
        } finally {
            try {
                reader.close();
            } catch (Exception e) {
            }
        }
    }
}
```

```
1 import java.io.*;
2 class ReaderExample1 {
3     public static void main(String args[]) {
4         FileReader reader = null;
5         try {
6             reader = new FileReader("poem.txt");
7             while (true) {
8                 int data = reader.read();
9                 if (data == -1)
10                     break;
11                 char ch = (char) data;
12                 System.out.print(ch);
13             }
14         }
15         catch (FileNotFoundException fnfe) {
16             System.out.println("파일이 존재하지 않습니다.");
17         }
18         catch (IOException ioe) {
19             System.out.println("파일을 읽을 수 없습니다.");
20         }
21         finally {
22             try {
23                 reader.close();
24             }
25             catch (Exception e) {
26             }
27         }
28     }
29 }
```

# 파일 관련 클래스 - FileReader

## ◆ FileReader 클래스

- 한꺼번에 여러 문자를 읽는 read 메소드

```
int num = reader.read(arr);
```



파일로부터 읽은 문자를 담을 char 배열

[올바른 예]

```
char arr[] = new char[100];  
int num = reader.read(arr);
```

[잘못된 예]

```
char arr[];  
int num = reader.read(arr);
```

# 파일 관련 클래스 - **FileWriter**

---

## ◆ **FileWriter** 클래스

- **FileWriter** 클래스 : 텍스트를 파일에 쓰는 클래스

```
FileWriter writer = new FileWriter("output.txt");
```

↑  
현재 디렉토리에 output.txt라는  
파일을 새로 만들어서 엽니다.

```
writer.write(ch);
```

↑  
이 문자를 파일에 씁니다.

```
writer.close();
```

↑  
파일을 닫는 메소드

```
1 import java.io.*;
2 class WriterExample1 {
3     public static void main(String args[]) {
4         FileWriter writer = null;
5         try {
6             writer = new FileWriter("output.txt");
7             char arr[] = { '안', '녕', '하', '세', '요', ' ', 'h', 'e', 'l', 'l', 'o' };
8             for (int cnt = 0; cnt < arr.length; cnt++)
9                 writer.write(arr[cnt]);
10        }
11        catch (IOException ioe) {
12            System.out.println("파일로 출력할 수 없습니다.");
13        }
14        finally {
15            try {
16                writer.close();
17            }
18            catch (Exception e) {
19            }
20        }
21    }
22 }
23 }
```

---

## ◆ **FileWriter** 클래스

- 한꺼번에 여러 문자를 출력하는 write 메소드

```
writer.write(arr);
```

↑  
char 배열

arr 배열에 있는 모든 문자들을 파일로 출력

# 데이터 포맷을 통한 출력

## ◆ PrintWriter 클래스와 PrintStream 클래스

-PrintWriter 클래스 : 데이터를 포맷해서 파일로 출력하는 클래스

```
PrintWriter writer = new PrintWriter("output.txt");
```

생성자 안에서 이 파일을 엽니다.

```
writer.print(12); // "12"라는 문자열을 출력
```

```
writer.print(10000L); // "10000"이라는 문자열을 출력
```

```
writer.println(1.5); // "1.5"라는 문자열과 줄바꿈 문자를 출력
```

```
writer.println(true); // "true"라는 문자열과 줄바꿈 문자를 출력
```

포맷 명세자(format specifier)

```
writer.printf("%d년 %d월 %d일", 2006, 4, 19);
```

// "2006년 4월 19일"이라는 문자열을 출력

```
writer.printf("파이=%4.2f", Math.PI);
```

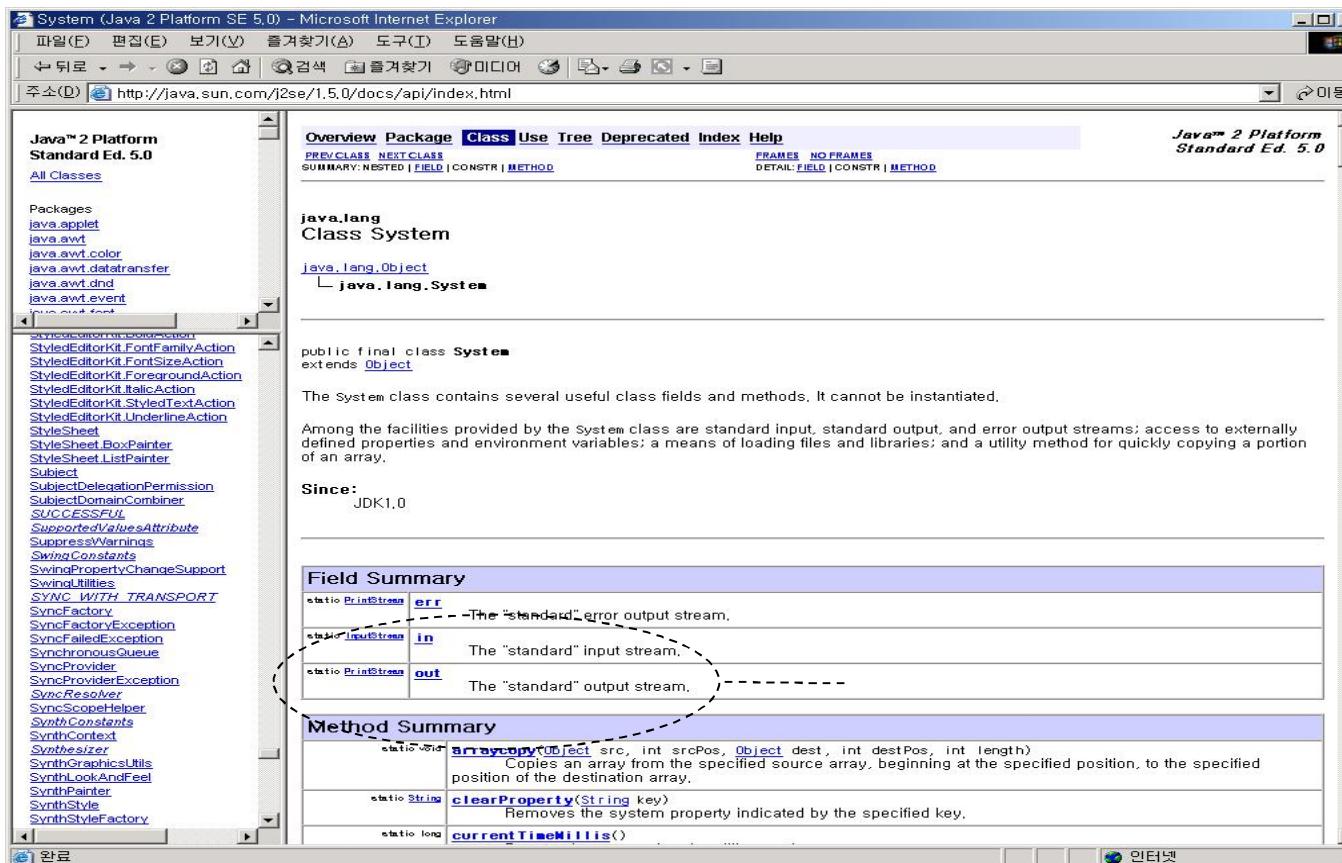
// "파이=3.14"라는 문자열을 출력

포맷 문자열(format string)

```
1 import java.io.*;
2 import java.util.*;
3 class PrintWriterExample1 {
4     public static void main(String args[]) {
5         PrintWriter writer = null;
6         try {
7             writer = new PrintWriter("output.txt");
8             writer.println("    *** 성적표 ***    ");
9             writer.printf("%3s : %3d %3d %3d %5.1f %n", "김지영", 92, 87, 95, 91.3f);
10            writer.printf("%3s : %3d %3d %3d %5.1f %n", "박현식", 100, 90, 88, 92.7f);
11            writer.printf("%3s : %3d %3d %3d %5.1f %n", "최민재", 75, 88, 85, 82.7f);
12
13        }
14        catch (IOException ioe) {
15            System.out.print("파일로 출력할 수 없습니다.");
16        }
17        finally {
18            try {
19                writer.close();
20            }
21            catch (Exception e) {
22            }
23        }
24    }
25 }
```

## ◆ PrintStream 클래스

- PrintStream 클래스 : 데이터를 포맷해서 파일로 출력하는 클래스



# 입출력 기능/성능 향상

---

## ◆ 입출력 기능/성능을 향상시키는 클래스들

클래스 이름	설명
DataInputStream	프리미티브 타입의 데이터를 입출력하는 클래스
DataOutputStream	
ObjectInputStream	프리미티브 타입과 레퍼런스 타입의 데이터를 입출력하는 클래스
ObjectOutputStream	
BufferedReader	데이터를 한꺼번에 읽어서 버퍼에 저장해두는 클래스
BufferedInputStream	
BufferedWriter	데이터를 버퍼에 저장해두었다가 한꺼번에 출력하는 클래스
BufferedOutputStream	
LineNumberReader	텍스트 파일의 각 행에 번호를 붙여가면서 읽는 클래스

- ✓ - 이 클래스들은 모두 java.io 패키지에 속함
- ✓ - 이 클래스들은 단독으로는 사용될 수 없음

# 파일관련 - 입출력 기능/성능 향상 클래스들

## ◆ DataOutputStream 클래스

-DataOutputStream 클래스 : 프리미티브 타입 데이터를 출력하는 클래스

```
FileOutputStream out1 = new FileOutputStream("output.dat");
DataOutputStream out2 = new DataOutputStream(out1);
```

FileOutputStream 객체를 생성해서  
DataOutputStream 생성자의 파라미터로 사용합니다.

메소드	설명
void writeByte(int value)	value의 마지막 1바이트 출력
void writeShort(int value)	value의 마지막 2바이트 출력
void writeChar(int value)	value의 마지막 2바이트 출력
void writeInt(int value)	value 출력
void writeLong(long value)	value 출력
void writeFloat(float value)	value 출력
void writeDouble(double value)	value 출력
void writeBoolean(boolean value)	value 출력

```
1 import java.io.*;
2 class DataOutputExample1 {
3     public static void main(String args[]) {
4         DataOutputStream out = null;
5         try {
6             out = new DataOutputStream(new FileOutputStream("output.dat"));
7             int arr[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
8             for (int cnt = 0; cnt < arr.length; cnt++)
9                 out.writeInt(arr[cnt]);
10        }
11        catch (IOException ioe) {
12            System.out.println("파일로 출력할 수 없습니다.");
13        }
14        finally {
15            try {
16                out.close();
17            }
18            catch (Exception e) {
19            }
20        }
21    }
22 }
```

## ◆ DataInputStream 클래스

- DataInputStream 클래스 : 프리미티브 타입 데이터를 읽는 클래스

```
FileInputStream in1 = new FileInputStream("input.dat");
```

FileInputStream 객체를 생성해서  
DataInputStream 생성자의 파라미터로 사용합니다.

```
DataInputStream in2 = new DataInputStream(in1);
```

메소드	설명
byte readByte()	1바이트를 읽어서 byte 타입으로 리턴
short readShort()	2바이트를 읽어서 short 타입으로 리턴
char readChar()	2바이트를 읽어서 char 타입으로 리턴
int readInt()	4바이트를 읽어서 int 타입으로 리턴
long readLong()	8바이트를 읽어서 long 타입으로 리턴
float readFloat()	4바이트를 읽어서 float 타입으로 리턴
double readDouble()	8바이트를 읽어서 double 타입으로 리턴
boolean readBoolean()	1바이트를 읽어서 boolean 타입으로 리턴 (0이면 false, 아니면 true)

```
1 import java.io.*;
2 class DataInputStreamExample1 {
3     public static void main(String args[]) {
4         DataInputStream in = null;
5         try {
6             in = new DataInputStream(new FileInputStream("output.dat"));
7             while (true) {
8                 int data = in.readInt();
9                 System.out.println(data);
10            }
11        }
12        catch (FileNotFoundException fnfe) {
13            System.out.println("파일이 존재하지 않습니다.");
14        }
15        catch (EOFException eofe) {
16            System.out.println("끝");
17        }
18        catch (IOException ioe) {
19            System.out.println("파일을 읽을 수 없습니다.");
20        }
21        finally {
22            try {
23                in.close();
24            }
25            catch (Exception e) {
26            }
27        }
28    }
29 }
```

## ◆ 문자열을 읽고 쓰는 메소드

- DataOutputStream 클래스의 문자열을 쓰는 메소드

```
out.writeUTF("Hello, java");  
                ↑               ↑               ↑  
                |               |               |  
                이 메소드를 호출하면 이 문자열을 출력합니다.  
DataOutputStream 객체
```

- DataInputStream 클래스의 문자열을 읽는 메소드

```
String str = in.readUTF();  
                ↑               ↑  
                |               |  
                이 메소드를 호출하면  
                문자열을 읽어서 리턴합니다.  
DataInputStream 객체
```

## ◆ ObjectOutputStream 클래스

-- ObjectOutputStream 클래스 : 객체를 출력하는 클래스

```
FileOutputStream out1 = new FileOutputStream("output.dat");
```

FileOutputStream 객체를 생성해서  
ObjectOutputStream 생성자의 파라미터로 사용합니다.

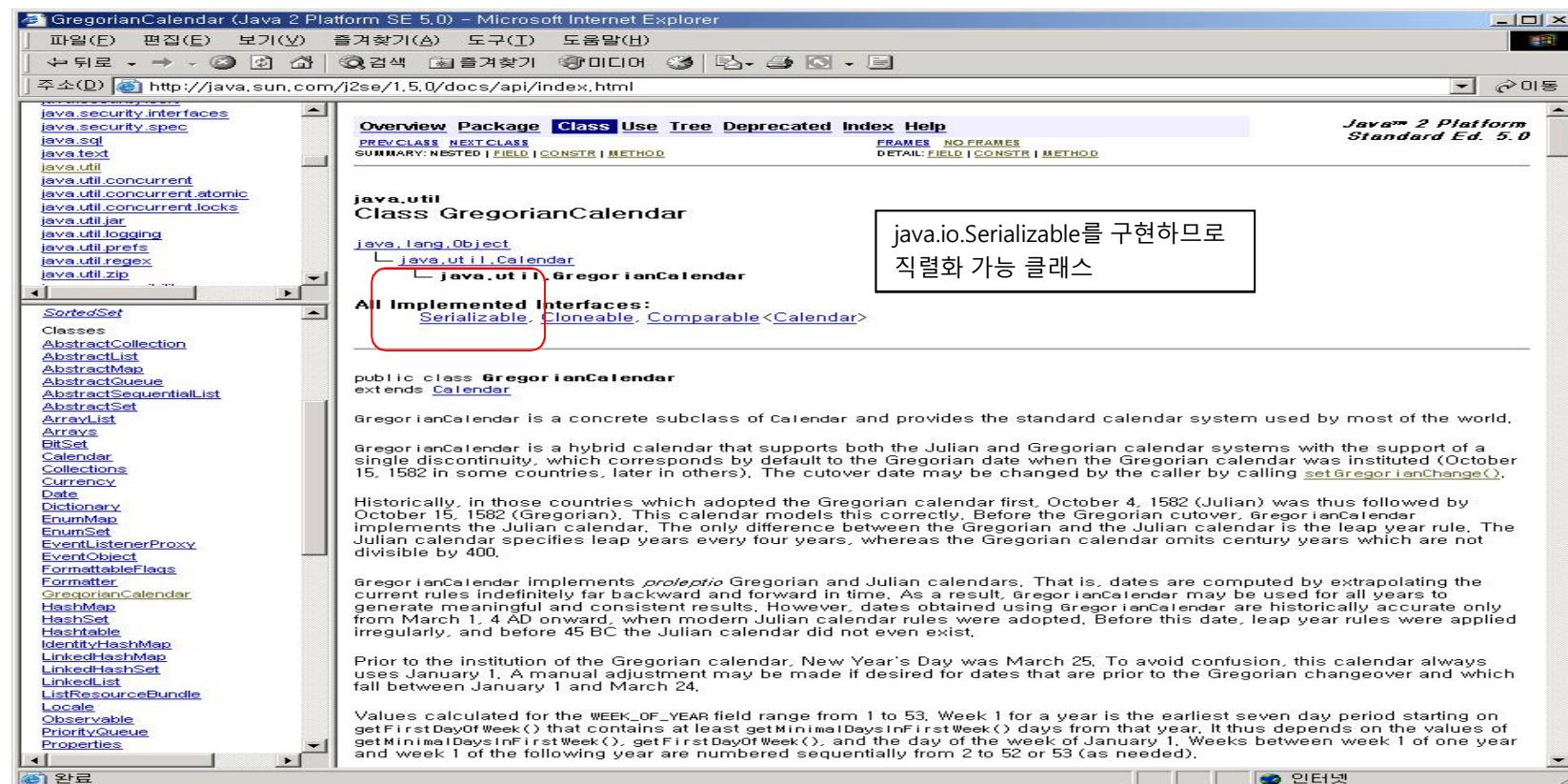
```
ObjectOutputStream out2 = new ObjectOutputStream(out1);
```

```
out2.writeObject(obj);
```



이렇게 객체를 스트림으로 만드는 것을  
직렬화(serialization)라고 합니다.

## • 직렬화 가능 클래스와 직렬화 불가능 클래스



```
1 import java.io.*;
2 import java.util.GregorianCalendar;
3 class ObjectOutputStreamExample1 {
4     public static void main(String args[]) {
5         ObjectOutputStream out = null;
6         try {
7             out = new ObjectOutputStream(new FileOutputStream("output.dat"));
8             out.writeObject(new GregorianCalendar(2006, 0, 14));
9             out.writeObject(new GregorianCalendar(2006, 0, 15));
10            out.writeObject(new GregorianCalendar(2006, 0, 16));
11        }
12        catch (IOException ioe) {
13            System.out.println("파일로 출력할 수 없습니다.");
14        }
15        finally {
16            try {
17                out.close();
18            }
19            catch (Exception e) {
20            }
21        }
22    }
23 }
```

## ◆ ObjectInputStream 클래스

- ObjectInputStream 클래스 : 객체를 읽는 클래스

```
FileInputStream in1 = new FileInputStream("input.dat");
```

FileInputStream 객체를 생성해서  
ObjectInputStream 생성자의 파라미터로 사용합니다.

```
ObjectInputStream in2 = new ObjectInputStream(in1);
```

```
GregorianCalendar obj = (GregorianCalendar) in2.readObject();
```

캐스트 연산이 필요

객체를 읽어서 리턴하는 메소드

```
1  class ObjectInputExample1 {
2      public static void main(String args[]) {
3          ObjectInputStream in = null;
4          try {
5              in = new ObjectInputStream(new FileInputStream("output.dat"));
6              while (true) {
7                  GregorianCalendar calendar = (GregorianCalendar) in.readObject();
8
9                  int year = calendar.get(Calendar.YEAR);
10                 int month = calendar.get(Calendar.MONTH) + 1;
11                 int date = calendar.get(Calendar.DATE);
12                 System.out.println(year + "/" + month + "/" + date);
13             }
14         } catch (FileNotFoundException fnfe) {
15             System.out.println("파일이 존재하지 않습니다.");
16         } catch (EOFException eofe) {
17             System.out.println("끝");
18         } catch (IOException ioe) {
19             System.out.println("파일을 읽을 수 없습니다.");
20         } catch (ClassNotFoundException cnfe) {
21             System.out.println("해당 클래스가 존재하지 않습니다.");
22         } finally {
23             try {
24                 in.close();
25             }
26             catch (Exception e) {
27             }
28         }
29     }
30 }
```

---

### ◆ “객체의 상태”란 정확하게 무엇을 의미할까?

- 인스턴스 변수 중에 원시 변수가 아닌 레퍼런스 변수가 있다면?
- 그 레퍼런스 변수가 참조하는 객체에 또 다른 객체를 참조하는 인스턴스 변수가 있다면?
- 저장했을 때와 똑같은 상태의 객체를 얻기 위해 어떤 것이 필요할지 생각해봅시다.

### ◆ 직렬화할 때 정적 변수도 저장되나요?

- 그렇지 않습니다. 정적 변수는 클래스마다 하나씩 있는 변수이므로 현재 정적 변수에 들어 있는 값을 받게 됩니다.
- 직렬화할 수 있는 객체를 만들 때는 동적으로 바뀔 수 있는 정적 변수에 의존하지 않도록 만들어야 합니다.

---

## ◆ Serializable 인터페이스

- 클래스를 직렬화할 수 있도록 하고 싶다면 Serializable 인터페이스를 구현해야 합니다.
- Serializable 같은 인터페이스는 표지 인터페이스, 또는 태그 인터페이스라고 부르기도 합니다.

```
objectOutputStream.writeObject(myBox);
```

```
import java.io.*;
```

```
public class Box implements Serializable {  
    ...  
}
```

- 
- ◆ 어떤 객체를 저장할 때 그 객체와 관련된 것들이 모두 제대로 직렬화되지 않으면 그 직렬화 작업은 제대로 완료되지 않습니다.
  - ◆ 직렬화하고자 하는 모든 객체가 직렬화할 수 있는(즉 Serializable을 구현하는) 클래스에 속해야 합니다.
  - ◆ 직렬화가 제대로 되지 않는 경우에는 NotSerializableException 예외가 발생합니다.
  - ◆ 특정 변수를 생략하고 싶다면
  - ◆ **transient** 키워드
    - 특정 인스턴스 변수가 직렬화되지 않도록 하고 싶다면 transient 키워드를 쓰면 됩니다.

```
import java.net.*;
import java.io.*;
class Chat implements Serializable {
    transient String currentID;
    String userName;
    // 나머지 코드
}
```

---

## ◆ 역직렬화(deserialization)

- 직렬화된 객체를 원래 상태로 돌려놓는 것

### 1. **FileInputStream** 만들기

```
FileInputStream fileStream = new FileInputStream("MyGame.ser");
```

### 2. **ObjectInputStream** 만들기

```
ObjectInputStream os = new ObjectInputStream(fileStream);
```

---

### 3. 객체 읽기

```
Object one = os.readObject();
Object two = os.readObject();
Object three = os.readObject();
```

### 4. 객체 캐스팅

```
GameCharacter elf = (GameCharacter) one;
GameCharacter troll = (GameCharacter) two;
GameCharacter magician = (GameCharacter) three;
```

### 5. **ObjectInputStream** 닫기

```
os.close();
```

---

## ◆ transient로 지정했던 변수는 나중에 역직렬화할 때 어떤 값으로 복구되나요?

- 무조건 null 값을 가지게 됩니다.
- null이 들어있으면 안 되는 경우의 해결책
  1. 특정 기본값을 정해놓고 복구할 때 항상 기본값을 대입하는 방법
  2. 해당 변수에서 중요한 역할을 하는 값을 따로 저장했다가 그 값을 이용하여 복구하는 방법

## ◆ 직렬화 가능 클래스의 버전 관리

- 버전 번호의 충돌을 최소화하는 방법

The screenshot shows a Windows desktop environment with two windows and a command prompt.

**Rectangle.java - 메모장** window:

```
class Rectangle implements java.io.Serializable {
    int width, height;
    Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }
}
```

A callout box points to the code with the text: "1) 직렬화 가능 클래스를 작성합니다."

**명령 프롬프트** window:

```
E:\work\chap17\17-2-4\example3>javac Rectangle.java
```

```
E:\work\chap17\17-2-4\example3>serialver Rectangle
Rectangle: static final long serialVersionUID = 1292117344561223774L;
```

A callout box points to the command with the text: "2) 직렬화 가능 클래스를 컴파일합니다."

A callout box points to the output of the serialver command with the text: "3) 컴파일한 디렉토리에서 serialver 명령을 실행하면 버전 번호가 생성됩니다."

The word "버전 번호" is centered at the bottom of the command prompt window.

## ◆ 직렬화 가능 클래스의 버전 관리

-serialver 명령이 생성한 버전 번호를 붙인 사각형 클래스

```
1  class Rectangle implements java.io.Serializable {
2      static final long serialVersionUID = 1292117344561223774L;
3      int width, height;
4      Rectangle(int width, int height) {
5          this.width = width;
6          this.height = height;
7      }
8      int getArea() {
9          return width * height;
10     }
11 }
```

버전 번호

---

## ◆ 버퍼를 이용해서 입출력 성능을 향상시키는 클래스들

- 스트림의 종류에 따라 4개의 클래스가 있음

클래스 이름	설명
BufferedInputStream	바이트 입력 스트림을 버퍼링하는 클래스
BufferedOutputStream	바이트 출력 스트림을 버퍼링하는 클래스
BufferedReader	문자 입력 스트림을 버퍼링하는 클래스
BufferedWriter	문자 출력 스트림을 버퍼링하는 클래스

## ◆ **BufferedInputStream** 클래스

- **BufferedInputStream** 클래스 : 바이트 입력 스트림의 성능을 향상시키는 클래스

```
FileInputStream in1 = new FileInputStream("input.dat");
```

FileInputStream 객체를 생성해서  
BufferedInputStream 생성자의 파라미터로 사용합니다.

```
BufferedInputStream in2 = new BufferedInputStream(in1);
```

- 버퍼의 크기를 지정하는 방법

```
BufferedInputStream in2 = new BufferedInputStream(in1, 1024);
```



```
1 import java.io.*;
2 class FileDump {
3     public static void main(String args[]) {
4         if (args.length < 1) {
5             System.out.println("Usage: java FileDump <filename>");
6             return;
7         }
8         BufferedInputStream in = null;
9         try {
10            in = new BufferedInputStream(new FileInputStream(args[0]));
11            byte arr[] = new byte[16];
12            while (true) {
13                int num = in.read(arr);
14                if (num < 0)
15                    break;
16                for (int cnt = 0; cnt < num; cnt++)
17                    System.out.printf("%02X ", arr[cnt]);
18                System.out.println();
19            }
20        } catch (FileNotFoundException fnfe) {
21            System.out.println(args[0] + " 파일이 존재하지 않습니다.");
22        } catch (IOException ioe) {
23            System.out.println(args[0] + " 파일을 읽을 수 없습니다.");
24        } finally {
25            try { in.close(); } catch (Exception e) { }
26        }
27    }
28 }
```

## ◆ BufferedWriter 클래스

- BufferedWriter 클래스 : 문자 출력 스트림의 성능을 향상시키는 클래스

```
FileWriter writer1 = new FileWriter("output.txt");
```

FileWriter 객체를 생성해서  
BufferedWriter 생성자의 파라미터로 사용합니다.

```
BufferedWriter writer2 = new BufferedWriter(writer1);
```

- 버퍼에 있는 데이터를 파일에 즉시 출력하는 flush메소드

```
writer.flush();
```

호출되는 즉시 버퍼의 데이터를  
모두 출력하는 메소드

```
1 import java.io.*;
2 class WriterExample1 {
3     public static void main(String args[]) {
4         BufferedWriter writer = null;
5         try {
6             writer = new BufferedWriter(new FileWriter("output.txt"));
7             char arr[] = {'안', '녕', '하', '세', '요', 'j', 'a', 'v', 'a' };
8             for (int cnt = 0; cnt < arr.length; cnt++)
9                 writer.write(arr[cnt]);
10        }
11        catch (IOException ioe) {
12            System.out.println("파일로 출력할 수 없습니다.");
13        }
14        finally {
15            try {
16                writer.close();
17            }
18            catch (Exception e) {
19            }
20        }
21    }
22 }
```

## ◆ LineNumberReader 클래스

- LineNumberReader 클래스 : 텍스트를 한 줄씩 번호를 매기면서 읽는 클래스

```
FileReader reader1 = new FileReader("input.txt");
```

FileReader 객체를 생성해서 LineNumberReader  
생성자의 파라미터로 사용합니다.

```
LineNumberReader reader2 = new LineNumberReader(reader1);
```

```
String str = reader2.readLine();
```

↑  
텍스트 한 줄을 읽어서  
리턴하는 메소드

```
int num = reader2.getLineNumber();
```

↑  
바로 전에 읽은 행 번호를  
리턴하는 메소드

```
1 import java.io.*;
2 class LineNumberExample1 {
3     public static void main(String args[]) {
4         LineNumberReader reader = null;
5         try {
6             reader = new LineNumberReader(new FileReader("poem.txt"));
7             while (true) {
8                 String str = reader.readLine();
9                 if (str == null)
10                     break;
11                 int lineNo = reader.getLineNumber();
12                 System.out.println(lineNo + ": " + str);
13             }
14         }
15         catch (FileNotFoundException fnfe) {
16             System.out.println("파일이 존재하지 않습니다.");
17         }
18         catch (IOException ioe) {
19             System.out.println("파일을 읽을 수 없습니다.");
20         }
21         finally {
22             try {
23                 reader.close();
24             }
25             catch (Exception e) {
26             }
27         }
28     }
29 }
```