객체지향프로그래밍 강의소개

황원주 교수

인제대학교 전자IT기계자동차공학부

2019년 가을학기

교과목 개요

교과목 개요

본 과목에서는 객체지향언어인 Java언어를 사용해서 객체지향언어를 이해하고 기본적인 프로그래밍 기법을 익히며, 스스로 창의적인 문제를 도출하여 팀웍과 팀원 간의 커뮤케이션을 통해 문제를 해결해나가는 방법을 함양한다.

수업진행방법

- 이론: 화 2,3교시
- 실습: 화 4,5교시
  - 실습시간에는 교재의 모든 연습문제와 보충문제를 직접 프로그래밍한 후 담당교수에게 검사 받은 후 귀가
- 시험: 제출된 시험 문제에 따라 각자 PC상에서 직접 프로그램 작성
  - 연습문제와 보충문제와 유사한 형식의 시험문제
  - 약 2시간 소요

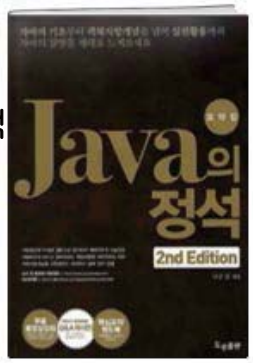
교과목 개요

성적평가방법

- 중간고사: 40%
- 기말고사: 50%
- 설계: 10%

교재

- 도서명: Java 의 정석
- 저자: 남궁 성
- 출판사: 도우출판



부교재

- 도서명: Java 프로그래밍
- 저자: 김일민, 조세홍
- 출판사: 홍릉과학출판사

2

주간진도계획

주간진도계획(1)

주	수 업 내 용	수업방법	평가방법
1	강의 소개 제1장 자바를 시작하기 전에 제2장 변수	강의/실습	
2	제3장 연산자 제4장 조건문과 반복문 제5장 배열	강의/실습	
3	제6장 객체지향프로그래밍I	강의/실습	
4	제6장 객체지향프로그래밍I	강의/실습	
5	제7장 객체지향프로그래밍II	강의/실습	
6	제7장 객체지향프로그래밍II	강의/실습	
7	제8장 예외처리	강의/실습	
8	중간고사		시험

3

주간진도계획(2)

주	수 업 내 용	수업방법	평가방법
9	제13장 AWT	강의/실습	
10	제13장 AWT	강의/실습	
11	제13장 AWT	강의/실습	
12	제12장 쓰레드	강의/실습	설계계획서
13	제12장 쓰레드	강의/실습	
14	설계 발표	발표	설계보고서
15	기말고사		시험

4

Thank you

제 2 장
변수(Variable)

인제대학교 전자IT기계자동차공학부
황 원 주

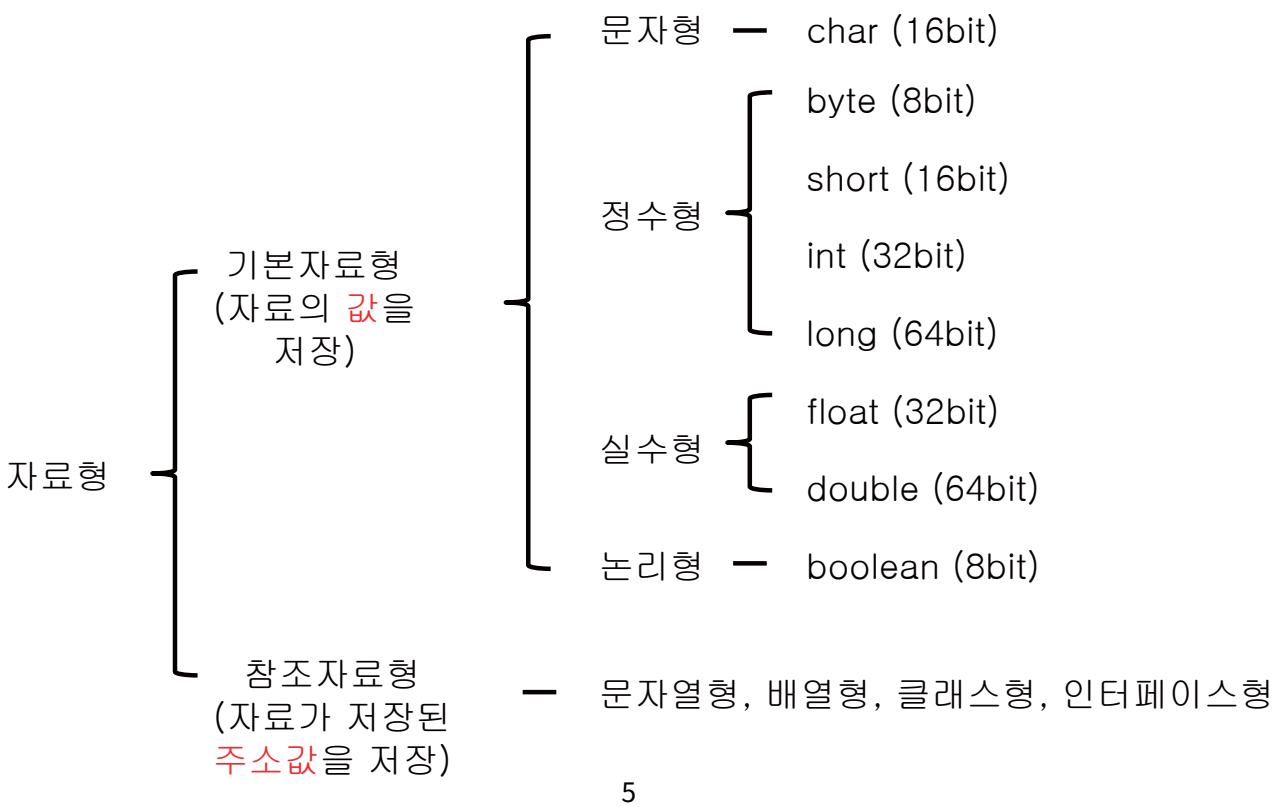
1. 변수(Variable)란?
2. 변수의 자료형(Data Type)
3. 변수의 선언방법
4. 명명규칙(Naming Convention)
5. 변수, 상수, 리터럴
6. 리터럴과 접미사
7. 변수의 기본값과 초기화
8. 문자와 문자열
9. 정수의 오버플로우(Overflow)
10. 형변환(Casting)

1. 변수(Variable)란?

변하는 수?

하나의 값을 저장할 수 있는 기억공간

2. 변수의 자료형(Data type)



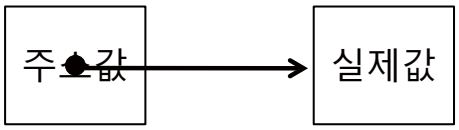
▶ 기본 자료형(Primitive type)

- 정수형, 실수형, 문자형, 논리형
- 자료의 실제 값을 저장

▶ 참조 자료형(Reference type)

- 기본형을 제외한 나머지(문자열형, 배열형, 클래스형, 인터페이스형)
-자료가 저장된 곳의 주소값을 저장

(4 byte, 0x00000000~0xffffffff)



기본 자료형(Primitive type)

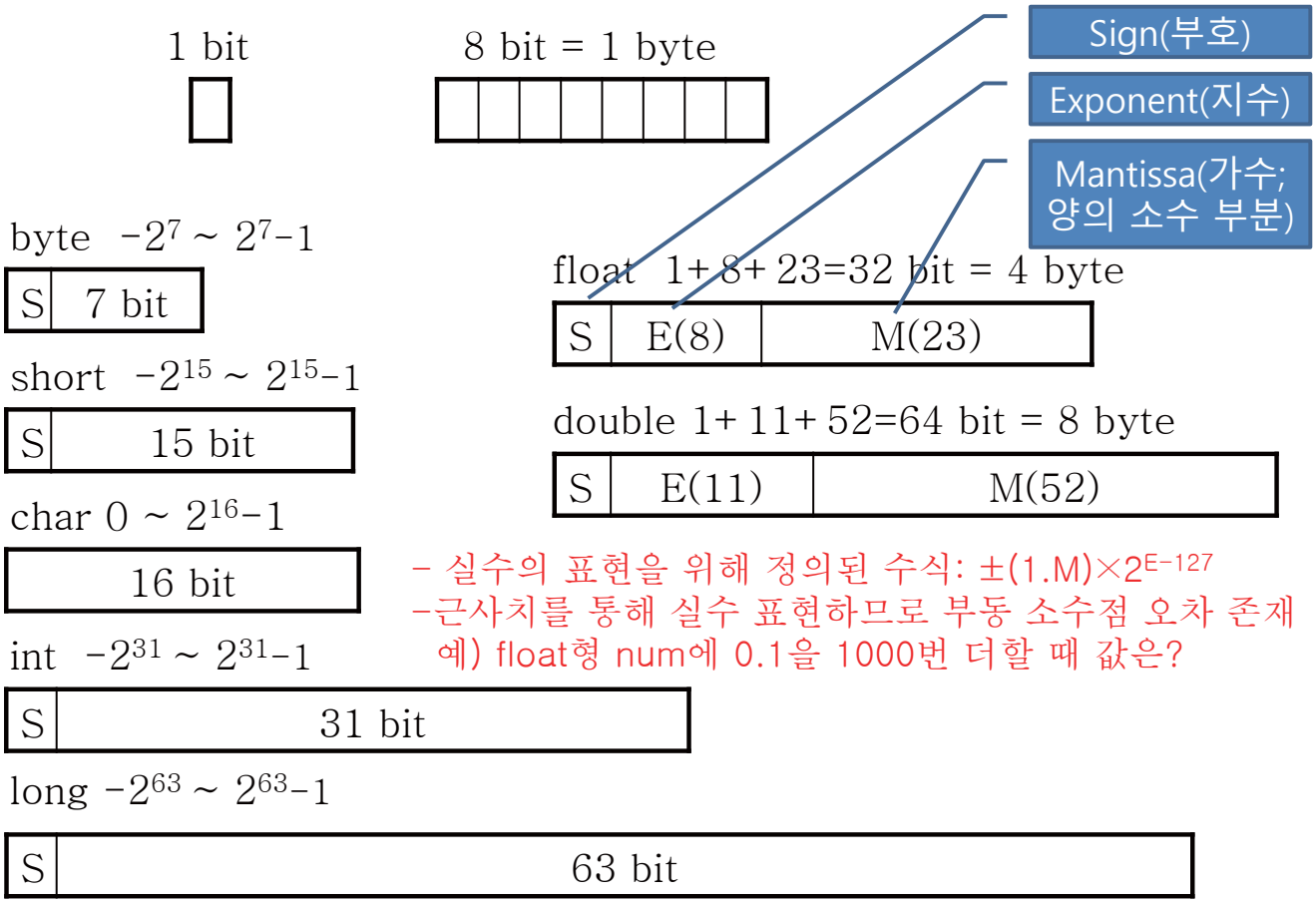
▶ 논리형 - true와 false중 하나를 값으로 갖으며, 조건식과 논리적 계산에 사용된다.

▶ 문자형 - 문자를 저장하는데 사용되며, 변수 당 하나의 문자만을 저장할 수 있다.

▶ 정수형 - 정수 값을 저장하는데 사용된다. 주로 사용하는 것은 int와 long이며,
byte는 이진데이터를 다루는데 사용되며,
short은 c언어와의 호환을 위해 추가되었다.

▶ 실수형 - 실수 값을 저장하는데 사용된다. float와 double이 있다.

종류 \ 크기(byte)	1	2	4	8
논리형	boolean			
문자형		char		
정수형	byte	short	int	long
실수형			float	double



3. 변수의 선언방법

타입 변수명;

기본자료형

```
int   score ;          (변수선언)
score = 100;           (변수초기화)
int score = 100;       (변수선언+변수초기화)
```

참조자료형

```
String str = new String("abc");
       str = null;     (참조자료형이 null이라는 말은 어떠한 객체도 참조하고 있지 않음을 의미: 초기화)
```

1. 참조변수 str 선언: 모든 참조변수는 4byte 메모리공간할당
2. "abc"라는 문자열을 담은 String객체 생성
3. 객체의 주소가 변수 str에 저장

4. 명명규칙(Naming convention)

1. 대소문자가 구분되며 길이에 제한이 없다.
   - True와 true는 서로 다른 것으로 간주된다.
2. 예약어(Reserved word)를 사용해서는 안 된다.
   - true는 예약어라 사용할 수 없지만, True는 가능하다.
3. 숫자로 시작해서는 안 된다.
   - top10은 허용하지만, 7up은 허용되지 않는다.
4. 특수문자는 '_'와 '$'만을 허용한다.
   - $harp은 허용되지만 S#arp는 허용되지 않는다.

4. 명명규칙 - 권장사항

1. 클래스 이름의 첫 글자는 항상 대문자로 한다.
   - 변수와 메서드 이름의 첫 글자는 항상 소문자로 한다.
2. 여러 단어 이름은 단어의 첫 글자를 대문자로 한다.
   - lastIndexOf, StringBuffer
3. 상수의 이름은 대문자로 한다. 단어는 '_'로 구분한다.
   - PI, MAX_NUMBER

5. 변수, 상수, 리터럴

- 변수(variable) – 하나의 값을 저장하기 위한 공간
- 상수(constant) – 한 번만 값을 저장할 수 있는 공간
- 리터럴(literal) – 변수 또는 상수에 저장되는 값

(100, 'A', "abc" 등)

```
int score = 100;
    score = 200;
char ch = 'A';
String str = "abc";
final int MAX = 100;
MAX = 200; // 에러
```

11

6. 리터럴과 접미사

```
boolean power = true;
char ch = 'A';
char ch = '\u0041';
char tab = '\t';
byte b = 127;
short s = 32767;
int i = 100;
int oct = 0100;
int hex = 0x100;
```

유니코드 (16진수 41->'A')

10진수 100

16진수 100

```
long l = 10000000000L;
float f = 3.14f
double d = 3.14d
float f = 100f;
10.   ⟶  10.0
.10   ⟶  0.10
10f   ⟶  10.0f
3.14e3f ⟶ 3140.0f
1e1   ⟶  10.0
```

7. 변수의 기본값과 초기화

변수의 초기화 : 변수에 처음으로 값을 저장하는 것

* 지역변수는 사용되기 전에 반드시 초기화해주어야 한다.

자료형	기본값
boolean	false
char	'\u0000'
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d 또는 0.0
참조형 변수	null

```
boolean isGood = false;
char grade = ' ';  // 공백
byte b = 0;
short s = 0;
int i = 0;
long l = 0;   // 0L로 자동변환
float f = 0;  // 0.0f로 자동변환
double d = 0; // 0.0로 자동변환
String s1 = null;
String s2 = ""; // 빈 문자열
```

8. 문자와 문자열

```
char ch = 'A';
char ch = 'AB'; // 에러
String s1 = "AB";
```

```
String s1 = "A" + "B"; // "AB"
```

"" + 7 ⟶ "" + "7" ⟶ "7"

```
char ch = ''; // 에러 (반드시 한 문자 저장해야 함)
String s1 = "";
```

""+ 7+ 7 ⟶ "7"+ 7 ⟶ "7"+ "7" ⟶ "77"

7+ 7+ "" ⟶ 14+ "" ⟶ "14"+ "" ⟶ "14"

문자열 + any type ⟶ 문자열

any type + 문자열 ⟶ 문자열

Java의 문자형도 C의 문자형과 마찬가지로 정수형의 부분집합

char형 변수에 저장되는 값은 부호 없는 정수의 형태로 저장 (모든 데이터는 숫자로 저장)

→ char형 변수 ch에는 문자 'A'가 저장되는 것이 아니라 'A'의 unicode인 65 (10진수)가 저장

예제 (C언어)

```
#include <stdio.h>

void main() {
   char ch = 'A';
   printf("%c₩n", ch);
   printf("%d₩n", ch);
}
```

실행결과

```
A
65
```

예제 2-1

```
class CharToCode {
	public static void main(String[] args) {
		char ch = 'A';		// char ch = '₩u0041';로 바꿔 써도 같다.
		int code = (int)ch; // ch에 저장된 값을 int형으로 변환하여 저장한다.
		System.out.println(ch);
		System.out.println(code);
	}
}
```

c1+1을 계산할 때, c1을 int형으로 변환한 후 덧셈연산 수행하므로 int값 98이 반환되므로 char c2=(char)(c1+1)과 같이 형변환이 필요

Java의 문자형도 C의 문자형과 마찬가지로 정수형의 부분집합

→ 문자형 변수에 사칙연산 가능 (예, 소문자↔대문자 프로그램)

예제 3-14

```
class OperatorEx14 {
	public static void main(String[] args) {
		char c1 = 'a';
		char c2 =  c1+1;	// 라인 5 : 컴파일 에러발생!!!
//수식에 변수가 있는 경우, 변수는  연산 도중에 언제든지 변할 수 있으므로, 컴파일러
//가 미리 계산 할 수 없다. 따라서 char+int의 결과값은 int이므로 형변환이 필요
		char c2 = 'a'+1;	// 라인 6 : 컴파일 에러 없음,
//소문자 a는 코드값이 10진수로는 97이고 16진수로는 61
//'a'+1은 리터럴간의 연산이므로, 리터럴은 연산 도중 변하는 값이 아니므로 컴파일 시
//에 컴파일러가 계산해서 그 결과인 char c2 = 'b'로 대체

		System.out.println(c2);
	}
}
```

컴파일 전	컴파일 후
char c='a'+1;	char c='b';
int i=100*100;	int i=10000;

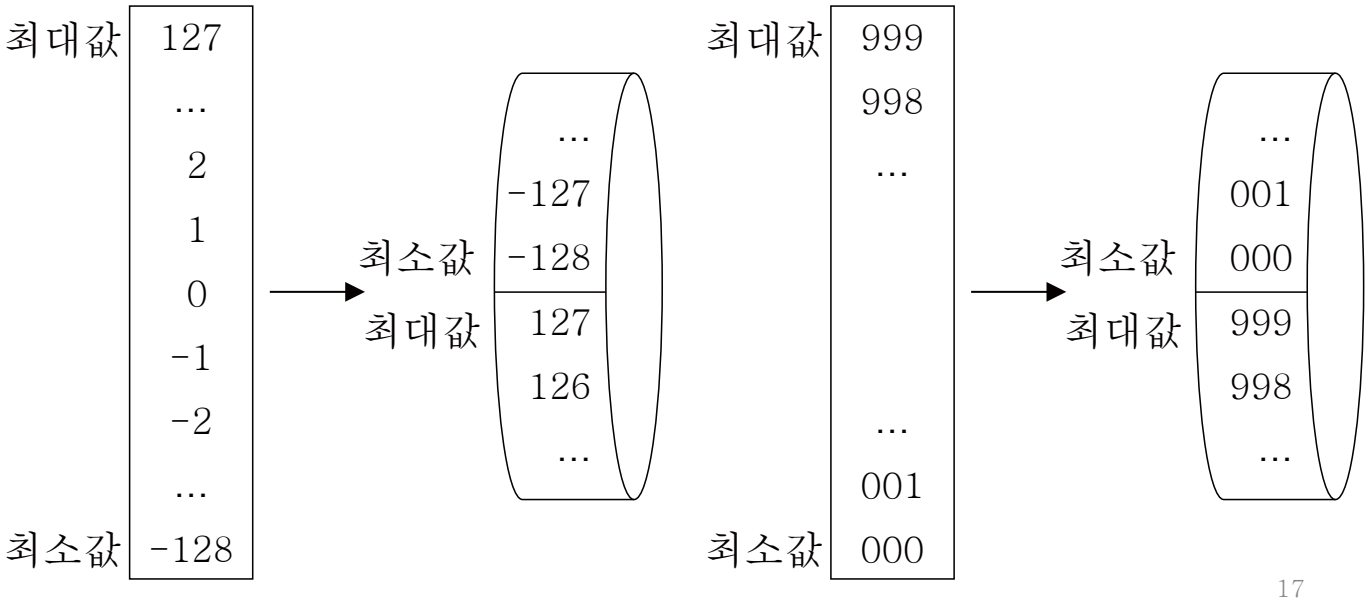
실행결과

```
b
```

9. 정수의 오버플로우(Overflow)

: 각 변수형이 저장할 수 있는 값의 범위를 초과할 때 발생

byte b = 127;　　byte b = 128; //에러
b = b + 1; // b에 저장된 값을 1증가



17

예제 3-14

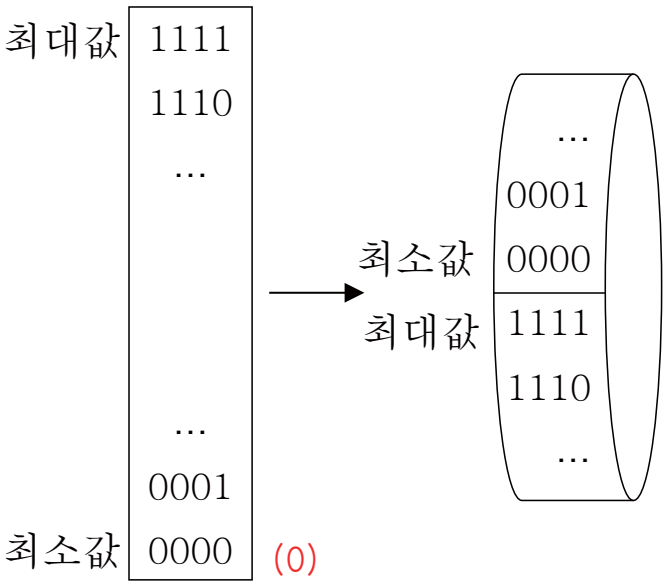
```
class IntTest2 {
  public static void main(String[] args) {
                    int i;
                    byte b;
                    short s, temp=128;
                    b = (byte) temp; // narrowing conversion
                    System.out.println(b);
                    s = (short) (temp*100); // overflow 발생가능성이
                                            // 있으므로 형변환 명시
                    System.out.println(s);
                    i = temp;          // widening conversion
                    System.out.println(i);
  }
}
```

실행결과

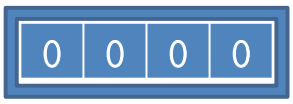
```
-128
12800
128
```


18

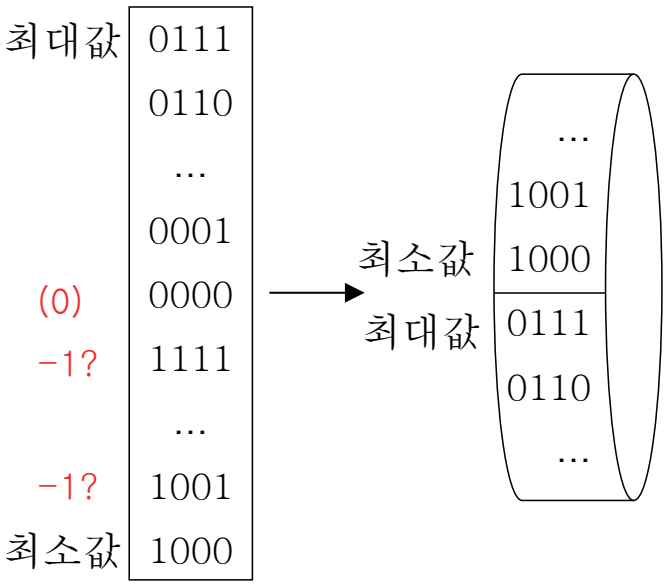
1. 부호가 없는 정수



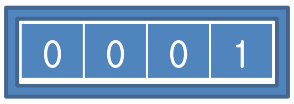
1씩 증가하는 2진 카운터



2. 부호가 있는 정수



1씩 감소하는 2진 카운터



19

►부호가 있는 정수에서 음수 만들기

- 음수는 일반적으로 2의 보수(2's complement)를 사용해서 표현
  1. 해당하는 양의 숫자를 표현한다.
  2. 모든 bit들을 반대로 바꾸고 1을 더한다.
  3. 부호 bit를 1로 지정한다.

- 예) -9
  1. 0000 1001 (9를 표현)
  2. 1111 0111 (2의 보수)
  3. 1111 0111 (부호 bit 지정)

20

10. 형변환(Casting)

형변환이란?

- 값의 타입을 다른 타입으로 변환하는 것이다.
- boolean을 제외한 7개의 기본형은 서로 형변환이 가능하다.

```
float f = 1.6f;
int i = (int)f;
```

변 환	수 식	결 과
int → char	(char)65	'A'
char → int	(int)'A'	65
float → int	(int)1.6f	1
int → float	(float)10	10.0f

1. 확대변환(widening conversion)

:시스템이 스스로 처리

byte → int

```
byte b = 10;
int i = (int) b; // 생략가능
```

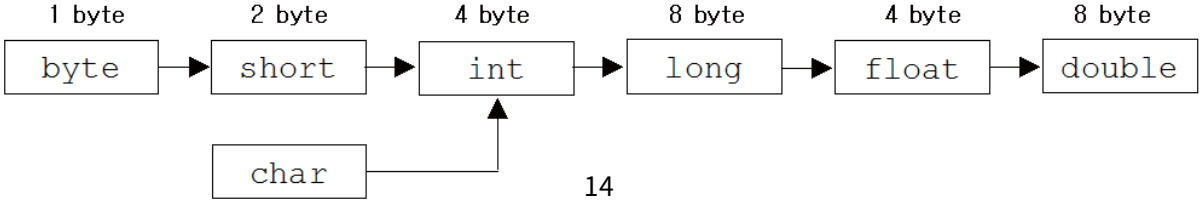
2. 축소변환(narrowing conversion)

:사용자 명시 필요 (컴파일러는 에러 메시지를 출력하고 형변환 명시요구)

int → byte

```
int i2 = 300;
byte b2 = (byte)i2; // 생략불가
```

변환	2진수	10진수	값손실
byte ↓ int	00001010 0000000000000000000000000000000000001010	10 10	없음
int ↓ byte	0000000000000000000000000000000100101100 00101100	300 44	있음



예제 2-8

```
class CastingEx2
{
	public static void main(String[] args)
	{
		byte b = 10;
		int i = (int)b;
		System.out.println("i=" + i);
		System.out.println("b=" + b);

		int i2 = 300;
		byte b2 = (byte)i2;
		System.out.println("i2=" + i2);
		System.out.println("b2=" + b2);
	}
}
```

실행결과

```
i=10
b=10
i2=300
b2=44 (Overflow발생)
```

제 3 장
연산자(Operator)

인제대학교 전자IT기계자동차공학부

황 원 주

1. 연산자(Operator)란?

▶ 연산자(Operator)

- 어떠한 기능을 수행하는 기호(+ ,-,*,/ 등)

▶ 피연산자(Operand)

- 연산자의 작업 대상(변수,상수,리터럴,수식)

a + b

2. 연산자의 종류

▶ 단항 연산자 : + - (타입) ++ -- ~ !

▶ 이항 연산자 <
- 산술 : + - * / % << >> >>>
- 비교 : > < >= <= == != } 연산결과로 true또는 false 반환
- 논리 : && || & ^ |

▶ 삼항 연산자 : ? :

▶ 대입 연산자 : =

3. 연산자의 우선순위

종 류	연산방향	연산자	우선순위
단항 연산자	←	++ -- + - ~ ! (타입)	높음
산술 연산자	→	* / %	
	→	+ -	
	→	<< >> >>>	
비교 연산자	→	< > <= >= instanceof	
	→	== !=	
논리 연산자	→	&	
	→	^	
	→	|	
	→	&&	
	→	||	
삼항 연산자	→	?:	
대입 연산자	←	= *= /= %= += -= <<= >>= >>>= &= ^= |=	낮음

[표3-1] 연산자의 종류와 우선순위

-괄호의 우선순위가 제일 높다.

(우선순위를 잘 모르겠다 → 무조건 괄호 쳐라!)

- 산술 > 비교 > 논리 > 대입

- 단항 > 이항 > 삼항

- 연산자의 연산 진행방향은 왼쪽에서 오른쪽(→)이다.

단, 단항, 대입 연산자만 오른쪽에서 왼쪽(←)이다.

3 * 4 * 5 x = y = 3

- 상식적으로 생각하라. 우리는 이미 다 알고 있다.

ex1) –x + 3 단항 > 이항

ex2) x + 3 * y 곱셈, 나눗셈 > 덧셈, 뺄셈

ex3) x + 3 > y - 2 산술 > 비교

ex4) x > 3 && x < 5 비교 > 논리

ex5) int result = x + y * 3; 항상 대입은 맨 끝에

- 그러나 몇 가지 주의해야 할 것이 있다.

1. <<, >>, >>>는 덧셈연산자보다 우선순위가 낮다.

ex5) x << 2 + 1 x << (2 + 1) 과 같다.

2. ||, |(OR)는 &&, &(AND)보다 우선순위가 낮다.

ex6) x < -1 || x > 3 && x < 5

x < -1 || (x > 3 && x < 5) 와 같다.

4. 증감연산자 - ++, --

- 증가연산자(++) : 피연산자의 값을 1 증가시킨다.
- 감소연산자(--) : 피연산자의 값을 1 감소시킨다.

```
int i = 5;
int j = 0;
```

전위형	j = ++i;	++i; j = i;	값이 대입(참조)되기 전에 증가시킨다.
후위형	j = i++;	j = i; i++;	값이 대입(참조)된 후에 증가시킨다.

예제 3-2

```
class OperatorEx2 {
        public static void main(String args[]) {
                int i=5;
                int j=0;
                j = i++;
                System.out.println("j=i++; 실행 후, i=" + i +", j="+ j);

                i=5;  // 결과 비교를 위해, i와 j의 값을 다시 5와 0으로 변경
                j=0;
                j = ++i;
                System.out.println("j=++i; 실행 후, i=" + i +", j="+ j);
        }
}
```

실행결과

```
j=i++; 실행 후, i=6, j=5
j=++i; 실행 후, i=6, j=6
```

5. 부호연산자(+ ,-)와 논리부정연산자(!)

▶ 부호연산자(+ ,-) : '+'는 피연산자에 1을 곱하고
'-'는 피연산자에 -1을 곱한다.

▶ 논리부정연산자(!) : true는 false로, false는 true로
피연산자가 boolean일 때만 사용가능

```
int i = -10;
i = +i;
i = -i;
```

```
boolean power = false;
power = !power;
power = !power;
```

6. 비트전환연산자 - ~

- 정수를 2진수로 표현했을 때, 1을 0으로 0은 1로 바꾼다.
정수형에만 사용가능.

2진수	10진수
0 0 0 0 1 0 1 0	10
1 1 1 1 0 1 0 1	-11
1 1 1 1 0 1 0 1	-11
0 0 0 0 0 0 0 1	+) 1
1 1 1 1 0 1 1 0	-10

[표3-5] 음수를 2진수로 표현하는 방법

7. 이항연산자의 특징

이항연산자는 연산을 수행하기 전에 피연산자의 타입을 일치시킨다.

- int보다 크기가 작은 타입은 int로 변환한다.

  (byte, char, short → int)

- 피연산자 중 표현범위가 큰 타입으로 형변환 한다.

  byte + short → int + int → int

  char + int → int + int → int

  float + int → float + float → float

  long + float → float + float → float

  float + double → double + double → double

```
byte a = 10;
byte b = 20;
byte c = a + b;
```

byte + byte → int + int → int

// 컴파일 에러가 발생! 명시적 형변환 필요

```
byte c = (byte)a + b;
```

// 에러: 캐스트연산자는 단항연산자이므로 연산 순위가 이항연산자인 덧셈연산자보다 높다. 따라서 (byte)a가 먼저 수행된 다음 덧셈이 수행되므로 캐스트연산자에 의해 byte로 변환된 다음 덧셈연사자에 의해 int로 변환된다

```
byte c = (byte)(a + b);
```

// OK

```
int a = 1000000;  // 1,000,000
int b = 2000000;  // 2,000,000
long c = a * b;   // c는 2,000,000,000,000 ?

        // c는 -1454759936 !!!-> 이미 오버플로우가 발생한 결과를
        // 아무리 long형 변수에 저장한다고 하더라도 int형 변수에
        // 저장한 것과 차이가 없음
```

int * int → int

```
long c = (long)a * b; // c는 2,000,000,000,000 !
```

long * int → long * long → long

```
long a = 1000000 * 1000000;  // 결과가 int므로 a는 -727,379,968
                             // (2^40->오버플로우)
long b = 1000000 * 1000000L; // b는 1,000,000,000,000
        // int형과 long(1000000L은 long형 리터럴)의
        // 연산이므로 그 결과가 long

int c = 1000000 * 1000000 / 1000000;  // c는 -727
        // -727,379,968을 1,000,000으로 나누므로

int d = 1000000 / 1000000 * 1000000; // d는 1,000,000
        // 나눗셈을 먼저 하므로 오버플로우가 발생하지 않으므로
```

```
char c1 = 'a';
char c2 = c1 +  1;  // 결과값이 int므로 에러
char c2 = (char)(c1 +  1);  // OK
char c2 = + + c1;  // OK
int i = 'B' – 'A';
int i = '2' – '0';
```

문자	코드
…	…
0	48
1	49
2	50
…	…
A	65
B	66
C	67
…	…
a	97
b	98
c	99
…	…

Pi의 값을 소수 셋째 자리까지 표시하는 방법

```
float pi = 3.141592f;
float shortPi = (int)(pi * 1000) / 1000f;
            (int)(3.141592f * 1000) / 1000f;
            (int)(3141.592f) / 1000f;
                3141 / 1000f;
              3141.0f / 1000f
                3.141f
```

- int/int -> int (결과가 float나 double가 아님)
- 나눗셈의 결과를 반올림하는 것이 아니라 버림. 즉, 3/4->1(1.5나 2가 아님)

* Math.round() : 소수점 첫째자리에서 반올림한 값을 반환

```
float pi = 3.141592f;
float shortPi = Math.round(pi * 1000) / 1000f;
```

Math.round(3.141592f * 1000) / 1000f;

Math.round(3141.592f) / 1000f;

3142 / 1000f;

3142.0f / 1000f

3.142f

8. 나머지연산자 - %

- 나누기한 나머지를 반환한다.
- 홀수, 짝수 등 배수검사에 주로 사용.

```
int share = 10 / 8;
int remain = 10 % 8;
```

10 % 8 → 2

10 % -8 → 2

-10 % 8 → -2

-10 % -8 → -2

9. 쉬프트연산자 - <<, >>, >>>

-사용처

▶ 2^n으로 곱하거나 나눈 결과를 반환한다.

▶ 곱셈, 나눗셈보다 빠르다.

▶그러나, 곱셈이나 나눗셈보다 가독성(readability)이 떨어지므로 특별한 경우를 제외하고는 사용하지 않는것이 좋다.

x << n 은 x * 2^n과 같다.

x >> n 은 x / 2^n과 같다.

8 << 2 는 8 * 2^2과 같다.

8 >> 2 는 8 / 2^2과 같다.

* 참고 : p.63 표3-11 쉬프트연산의 예

10. 비교연산자 - > < >= <= == !=

- 피연산자를 같은 타입으로 변환한 후에 비교한다.
  연산결과는 true 또는 false이다.
- 기본형(boolean제외)과 참조형에 사용할 수 있으나
  참조형에는 ==와 !=만 사용할 수 있다.

수 식	연 산 결 과
x > y	x가 y보다 클 때 true, 그 외에는 false
x < y	x가 y보다 작을 때 true, 그 외에는 false
x >= y	x가 y보다 크거나 같을 때 true, 그 외에는 false
x <= y	x가 y보다 작거나 같을 때 true, 그 외에는 false
x == y	x와 y가 같을 때 true, 그 외에는 false
x != y	x와 y가 다를 때 true, 그 외에는 false

[표3-11] 비교연산자의 연산결과

'A' < 'B' → 65 < 66 → true

'0' == 0 → 48 == 0 → false

'A' != 65 → 65 != 65 → false

10.0d == 10.0f → 10.0d == 10.0d → true

0.1d == 0.1f → 0.1d == 0.1d → true? false?

```
double d = (double)0.1f;
System.out.println(d);  // 0.10000000149011612
```

(float)0.1d == 0.1f → 0.1f == 0.1f → true

11. 비트연산자 - & | ^

- 피연산자를 비트단위로 연산한다.
  실수형(float, double)을 제외한 모든 기본형에 사용가능

▶ OR연산자(|) : 피연산자 중 어느 한 쪽이 1이면 1이다.

▶ AND연산자(&) : 피연산자 양 쪽 모두 1이면 1이다.

▶ XOR연산자(^) : 피연산자가 서로 다를 때 1이다.

x	y	x | y	x & y	x ^ y
1	1	1	1	0
1	0	1	0	1
0	1	1	0	1
0	0	0	0	0

-사용처

▶ 플래그 비트(flag bit): 해당되는 비트 플래그를 비교함으로써 어떤 종류의 플래그인지 판별할 수 있는 비트

예) 두개의 플래그를 비교: XOR연산자(^)

▶ 마스크 비트(mask bit): 플래그 비트의 특정 비트만은 처리하는 비트

예) 플래그를 set: OR연산자 (|), 플래그를 clear: AND연산자 (&)

-연산결과

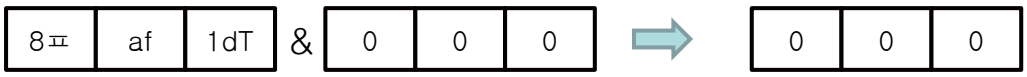
식	2진수	10진수
3 | 5 = 7	0 0 0 0 0 0 1 1	3
	|) 0 0 0 0 0 1 0 1	5
	0 0 0 0 0 1 1 1	7
3 & 5 = 1	0 0 0 0 0 0 1 1	3
	&) 0 0 0 0 0 1 0 1	5
	0 0 0 0 0 0 0 1	1
3 ^ 5 = 6	0 0 0 0 0 0 1 1	3
	^) 0 0 0 0 0 1 0 1	5
	0 0 0 0 0 1 1 0	6

[표3-14] 비트연산자의 연산결과

예) 정보통신공학과 학생의 졸업요건 (학점, 토익, 봉사)

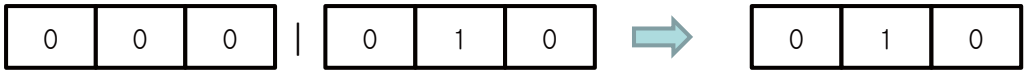
학점	토익	봉사

1. 입학: 초기화 (모든 플래그 비트를 clear)



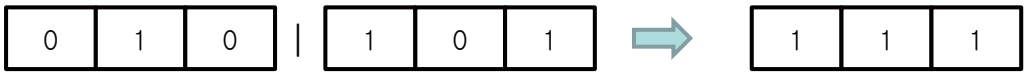
(플래그 비트) (마스크 비트)

2. 2학년: 토익점수 700점 획득 (토익 플래그 비트를 set)



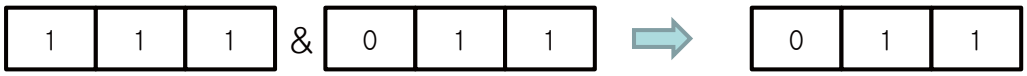
(플래그 비트) (마스크 비트)

3. 4학년: 학점이수 및 봉사시간 만족 (학점과 봉사 플래그 비트를 set)



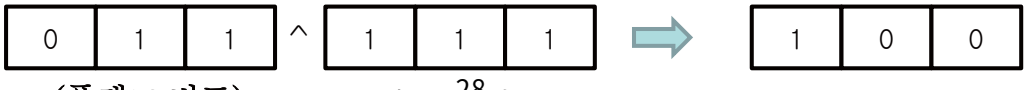
(플래그 비트) (마스크 비트)

4. 학과 졸업사정: 졸업과제 미이수 발견 (학점 플래그 비트를 clear)



(플래그 비트) (마스크 비트)

5. 학교 졸업사정: 전체 학생들 중에서 졸업할 학생 선별 (xor사용하여 결과가 000일때 졸업가능)

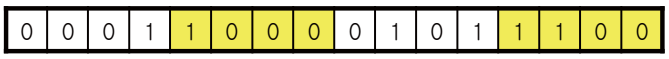


(플래그 비트) (마스크 비트)

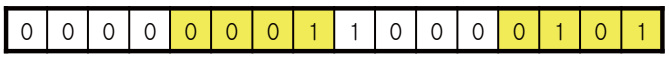
예제

```
class DrawShape {
  public static void main(String args[]) {
        final byte TRIANGLE=1;     //0001
        final byte RECTANGLE=2;  //0010
        final byte CIRCLE=4;         //0100
        final byte ELLIPSE=8;        //1000

   // 생략

        DrawPolygon(byte type)  {    //type=0010인 경우
          if( !(TRIANGLE&~type) )
           //삼각형을 그린다.
          if( !(RECTANGLE&~type) ) //!((0010)&(~(0010)))
           //사각형을 그린다.    //!((0010)&(1101)) -> &: 둘 다 1일 때만 1
          if( !(CIRCLE&~type) ) //!(0000) -> 1111
           //원을 그린다.
          if( !(ELLIPSE&~type) )
           //타원을 그린다.
}
  }
        }
```

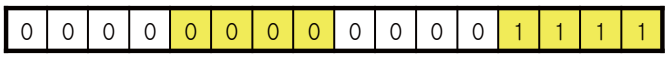
(플래그 비트) 0x185C



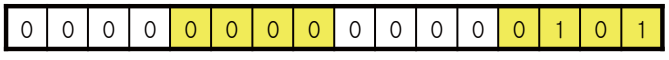
0x185C >> 4 → 0x0185



(마스크 비트) 0x000F



0x0185 & 0x000F → 0x0005



이 부분이 clear

나머지 부분은 그대로

0x185C >> 4 & 0x000F → 0x0005

0x185C >> 8 & 0x000F → 0x0008

10진수에서 나눗셈과 나머지 연산자 이용하여 동일한 결과를 얻을 수 있다.

12345 / 100 % 10 → 3

12345 / 1000 % 10 → 2

12. 논리연산자 - && ||

-피연산자가 반드시 boolean이어야 하며 연산결과도 boolean (true 또는 false)이다.

&&가 || 보다 우선순위가 높다. 같이 사용되는 경우 괄호를 사용하자

▶ OR연산자(||) : 피연산자 중 어느 한 쪽이 true이면 true이다.

▶ AND연산자(&&) : 피연산자 양 쪽 모두 true이면 true이다.

x	y	x || y	x && y
true	true	true	true
true	false	true	false
false	true	true	false
false	false	false	false

int i = 7;

i > 3 && i < 5

i > 3 || i < 0

x	y	x || y	x && y
true	true	true	true
true	false	true	false
false	true	true	false
false	false	false	false

char x = 'j';

x >= 'a' && x <= 'z'

(x >= 'a' && x <= 'z') || (x >= 'A' && x <= 'Z')

13. 삼항연산자 - ? :

- 조건식의 연산결과가 true이면 '식1'의 결과를 반환하고
  false이면 '식2'의 결과를 반환한다.

(조건식) ? 식1 : 식2

- 따라서 보통 조건식에는 true 또는 false를 반환하는 비교연산자 또는 논리연산자가 온다.

```
int x = -10;
int absX = x >= 0 ? x : -x;
```

```
if(x>=0) {
   absX = x;
} else {
   abxX = -x;
}
```

```
int score = 50;
char grade = score >= 90 ? 'A' : (score >= 80? 'B' : 'C');
```

14. 대입연산자 - = op=

- 오른쪽 피연산자의 값을 왼쪽 피연산자에 저장한다.
  단, 왼쪽 피연산자는 상수가 아니어야 한다.

```
int i = 0;
i = i +  3;

final int MAX = 3;
MAX = 10;  // 에러
```

op=	=
i +=3;	i = i + 3;
i -= 3;	i = i - 3;
i *= 3;	i = i * 3;
i /= 3;	i = i / 3;
i %= 3;	i = i % 3;
i <<= 3;	i = i << 3;
i >>= 3;	i = i >> 3;
i >>>= 3;	i = i >>> 3;
i &= 3;	i = i & 3;
i ^= 3;	i = i ^ 3;
i |= 3;	i = i | 3;
i *= 10 + j;	**i = i * (10+j);**

제 4 장
조건문과 반복문

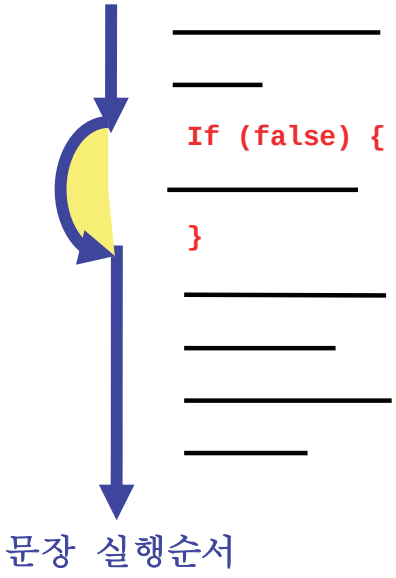
인제대학교 전자IT기계자동차공학부

황 원 주

제어문(control statement): 문장 실행순서(왼쪽에서 오른쪽으로, 위에서 아래로 순서대로 한번씩 빠짐없이 실행)의 제어

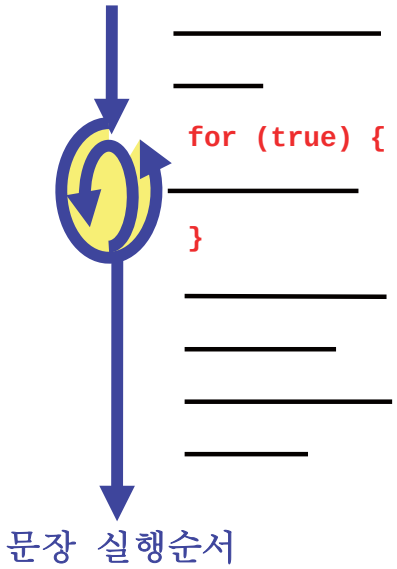
- 조건문
- 반복문

1. 조건문
   - 1.1 조건문(if, switch)
   - 1.2 if문
   - 1.3 중첩 if문
   - 1.4 switch문
   - 1.5 중첩 switch문
   - 1.6 if문과 switch문의 비교
   - 1.7 Math.random()



2. 반복문

2.1 반복문(for, while, do-while)
2.2 for문
2.3 중첩 for문
2.4 while문
2.5 중첩 while문
2.6 do-while문
2.7 break문
2.8 continue문
2.9 이름 붙은 반복문과
break, continue



1. 조건문(if, switch)

1.1 조건문 – if, switch

```
if(조건식) { 문장들 }
```

- 조건문은 조건식과 실행될 하나의 문장 또는 블럭{}으로 구성
- Java에서 조건문은 if문과 switch문 두 가지 뿐이다.
- if문이 주로 사용되며, 경우의 수가 많은 경우 switch문을 사용할 것을 고려한다.
- 모든 switch문은 if문으로 변경이 가능하지만, if문은 switch문으로 변경할 수 없는 경우가 많다.

```
if(num==1) {
    System.out.println("SK");
} else if(num==6) {
    System.out.println("KTF");
} else if(num==9) {
    System.out.println("LG");
} else {
    System.out.println("UNKNOWN");
}
```



```
switch(num) {
    case 1:
        System.out.println("SK");
        break;
    case 6:
        System.out.println("KTF");
        break;
    case 9:
        System.out.println("LG");
        break;
    default:
        System.out.println("UNKNOWN");
}
```

1.2 if문

- if문은 if, if-else, if-else if의 세가지 형태가 있다.
- 조건식의 결과는 반드시 true 또는 false이어야 한다.

```
if(조건식) {
    // 조건식의 결과가 true일 때 수행될 문장들
}
```

```
if(조건식) {
    // 조건식의 결과가 true일 때 수행될 문장들
} else {
    // 조건식의 결과가 false일 때 수행될 문장들
}
```

```
if(조건식1) {
    // 조건식1의 결과가 true일 때 수행될 문장들
} else if(조건식2) {
    // 조건식2의 결과가 true일 때 수행될 문장들
    // (조건식1의 결과는 false)
} else if(조건식3) {
    // 조건식3의 결과가 true일 때 수행될 문장들
    // (조건식1과 조건식2의 결과는 false)
} else {
    // 모든 조건식의 결과가 false일 때 수행될 문장들
}
```

```
if(score > 60) {
    System.out.println("합격입니다.");
}

if(score > 60)
    System.out.println("합격입니다.");

if(score > 60) {
    System.out.println("합격입니다.");
} else {
    System.out.println("불합격입니다.");
}

if(num > 0) {
    System.out.println("양수입니다.");
} else if(num < 0) {
    System.out.println("음수입니다.");
} else {
    System.out.println("영입니다.");
}

if(score >=90) {
    System.out.println("A등급");
} else if(score >= 80 && score < 90 ){ // 80<=score<90
    System.out.println("B등급");
} else if(score >= 70 && score < 80 ){ // 70<=score<80
    System.out.println("C등급");
} else { // score < 70
    System.out.println("F등급");
}
```

1.2 if문 – 조건식의 예(example)

```
int i = 0;
if(i%2==0) { }
if(i%3==0) { }
```

```
if(i=0) { }
if(i==0) { }
```

```
String str = "";
char ch = ' ';
if(ch==' ' || ch=='₩t') { }
if(ch=='c' || ch=='C') { }
if(str=="c" || str=="C") { }
if(str.equals("c") || str.equals("C")) { }
if(str.equalsIgnoreCase("c")) { }

if(ch>='0' && ch<='9') { }
if(!(ch>='0' && ch<='9')) { }
if(ch<'0' || ch>'9')) { }
```

```
if(('a'<=ch && ch<='z')||
  ('A'<=ch && ch<='Z')) { }

if( i<-1 || i>3 && i<5 ) {}

str="3";
if(str!=null && !str.equals("")) {
    ch = str.charAt(0);
}

boolean powerOn=false;
if(!powerOn) {
    // 전원이 꺼져있으면...
}
```

문자열 "3" → 문자 '3'

1.3 중첩 if문

- if문 안에 또 다른 if문을 중첩해서 넣을 수 있다.
- if문의 중첩횟수에는 거의 제한이 없다.

```
if (조건식1) {
    // 조건식1의
    if (조건식2)
        // 조
    } else {
        // 조
    }
} else {
    // 조건식1이
}
```

```
if (score >= 90)    {       // score가 90점 보다 같거나 크면 A학점(grade)
    grade = "A";

    if ( score >= 98) {     // 90점 이상 중에서도 98점 이상은 A+
        grade += "+";       // grade = grade + "+";
    } else if ( score < 94) {
        grade += "-";
    }
} else if  (score >= 80){   // score가 80점 보다 같거나 크면 B학점(grade)
    grade = "B";

    if ( score >= 88) {
        grade += "+";
    } else if ( score < 84) {
        grade += "-";
    }
} else {                    // 나머지는 C학점(grade)
    grade = "C";
}
```

1.4 switch문

- if문의 조건식과 달리, 조건식의 계산결과가 정수형에 포함될 수 있는 유형(int형, char형, byte형)만 가능하다.
- 조건식의 계산결과와 일치하는 case문으로 이동 후 break문을 만날 때까지 문장들을 수행한다.(break문 반드시 넣을 것, break문이 없으면 switch문의 끝까지 진행한다.)
- 일치하는 case문의 값이 없는 경우 default문으로 이동한다.
  (default문 생략가능하나 생략하지 말 것)
- case문의 값으로 변수를 사용할 수 없다.(리터럴, 상수만 가능)
- 중복된 case문의 값은 사용할 수 없다.

```
switch (조건식) {
    case 값1 :
        // 조건식의 결과가 값1과 같을 경우 수행될 문장들
        //...
        break;
    case 값2 :
        // 조건식의 결과가 값2와 같을 경우 수행될 문장들
        //...
        break;
    //...
    default :
        // 조건식의 결과와 일치하는 case문이 없을 때 수행될 문장들
        //...
}
```

```
switch(num) {
    case 1:
    case 7:
        System.out.println("SK");
        break;
    case 6:
    case 8:
        System.out.println("KTF");
        break;
    case 9:
        System.out.println("LG");
        break;
    default:
        System.out.println("UNKNOWN");
        break;
}
```

1.4 switch문 – 사용예(examples)

```
int level = 3;

switch(level) {
    case 3 :
        grantDelete(); // 삭제권한을 준다.
    case 2 :
        grantWrite();  // 쓰기권한을 준다.
    case 1 :
        grantRead();   // 읽기권한을 준다.

}
```

```
char op = '*';

switch(op) {
    case '+':
        result = num1 + num2;
        break;
    case '-':
        result = num1 - num2;
        break;
    case '*':
        result = num1 * num2;
        break;
    case '/':
        result = num1 / num2;
        break;
}
```

```
switch(score) {
    case 100: case 99: case 98: case 97: case 96:
    case 95:  case 94: case 93: case 92: case 91:
    case 90 :
        grade = 'A';
        break;
    case 89: case 88: case 87: case 86:
    case 85: case 84: case 83: case 82: case 81:
    case 80 :
        grade = 'B';
        break;
    case 79: case 78: case 77: case 76:
    case 75: case 74: case 73: case 72: case 71:
    case 70 :
        grade = 'C';
        break;
    case 69: case 68: case 67: case 66:
    case 65: case 64: case 63: case 62: case 61:
    case 60 :
        grade = 'D';
        break;
    default :
        grade = 'F';
} // end of switch
```

```
switch(score/10) {
    case 10:
    case  9 :
        grade = 'A';
        break;
    case 8 :
        grade = 'B';
        break;
    case 7 :
        grade = 'C';
        break;
    case 6 :
        grade = 'D';
        break;
    default :
        grade = 'F';
}
```

예제 4-8

```
class FlowEx8
{
  public static void main(String[] args)
  {
    int score = 1;

    switch(score*100) {
            case 100 :
                    System.out.println("당신의 점수는 100이고, 상품은 자전거입니다.");
            case 200 :
                    System.out.println("당신의 점수는 200이고, 상품은 TV입니다.");
            case 300 :
                    System.out.println("당신의 점수는 300이고, 상품은 노트북입니다.");
            case 400 :
                    System.out.println("당신의 점수는 400이고, 상품은 자동차입니다.");
            default :
                    System.out.println("죄송하지만 당신의 점수에 해당상품이 없습니다.");
    }
  }
}
```

실행결과

```
당신의 점수는 100이고, 상품은 자전거입니다.
당신의 점수는 200이고, 상품은 TV입니다.
당신의 점수는 300이고, 상품은 노트북입니다.
당신의 점수는 400이고, 상품은 자동차입니다.
죄송하지만 당신의 점수에 해당상품이 없습니다.
```

1.5 중첩 switch문

- switch문 안에 또 다른 switch문을 중첩해서 넣을 수 있다.
- switch문의 중첩횟수에는 거의 제한이 없다.

```
switch(num) {
    case 1:
    case 7:
        System.out.println("SK");
        switch(num) {
            case 1:
                System.out.println("1");
                break;
            case 7:
                System.out.println("7");
                break;
        }
        break;
    case 6:
        System.out.println("KTF");
        break;
    case 9:
        System.out.println("LG");
        break;
    default:
        System.out.println("UNKNOWN");
}
```



```
switch(num) {
    case 1:
    case 7:
        System.out.println("SK");
        if(num==1) {
            System.out.println("1");
        } else if(num==7) {
            System.out.println("7");
        }
        break;
    case 6:
        System.out.println("KTF");
        break;
    case 9:
        System.out.println("LG");
        break;
    default:
        System.out.println("UNKNOWN");
}
```

1.6 if문과 switch문의 비교

- if문이 주로 사용되며, 경우의 수가 많은 경우 switch문을 사용할 것을 고려한다.
- 모든 switch문은 if문으로 변경이 가능하지만, if문은 switch문으로 변경할 수 없는 경우가 많다.
- if문 보다 switch문이 더 간결하고 효율적이다.

```
if(num==1) {
    System.out.println("SK");
} else if(num==6) {
    System.out.println("KTF");
} else if(num==9) {
    System.out.println("LG");
} else {
    System.out.println("UNKNOWN");
}
```



```
switch(num) {
    case 1:
        System.out.println("SK");
        break;
    case 6:
        System.out.println("KTF");
        break;
    case 9:
        System.out.println("LG");
        break;
    default:
        System.out.println("UNKNOWN");
}
```

1.7 Math.random()

- Math클래스에 정의된 난수(亂數) 발생함수
- 0.0과 1.0 사이의 double값을 반환한다.(0.0 <= Math.random() < 1.0)

예) 1~10범위의 임의의 정수를 얻는 식 만들기

1. 각 변에 10을 곱한다.

```
0.0 * 10 <= Math.random() * 10 < 1.0 * 10
    0.0 <= Math.random() * 10 < 10.0
```

2. 각 변을 int형으로 변환한다.

```
(int)0.0 <= (int)(Math.random() * 10) < (int)10.0
       0 <= (int)(Math.random() * 10) < 10
```

3. 각 변에 1을 더한다.

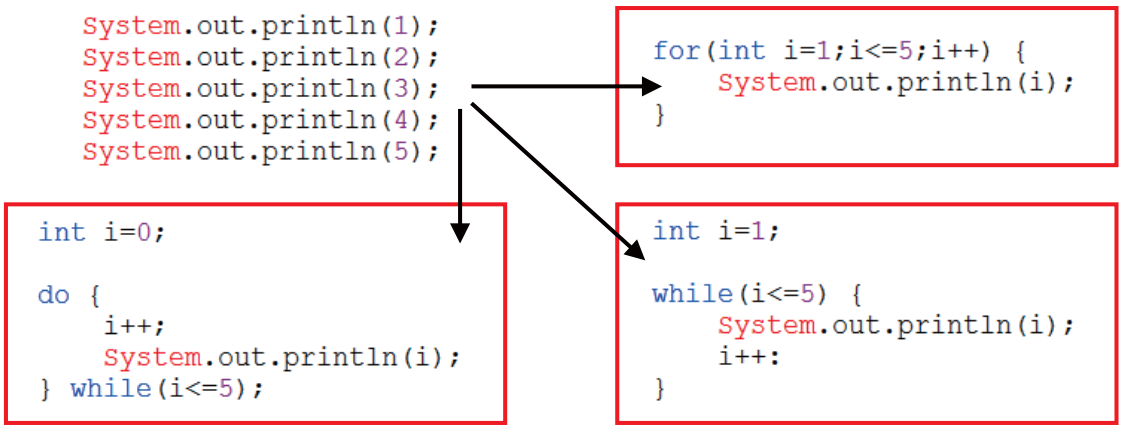
```
int score = (int)(Math.random() * 10)+1;
```

```
0 + 1 <= (int)(Math.random() * 10) + 1 < 10 + 1
    1 <= (int)(Math.random() * 10) + 1 < 11
```

2. 반복문(for, while, do-while)

2.1 반복문 – for, while, do-while

- 문장 또는 문장들을 반복해서 수행할 때 사용
- 조건식과 수행할 블럭{} 또는 문장으로 구성
- 반복회수가 중요한 경우에 for문을 그 외에는 while문을 사용한다.
- for문과 while문은 서로 변경가능하다.
- do-while문은 while문의 변형으로 블럭{}이 최소한 한번은 수행될 것을 보장한다.

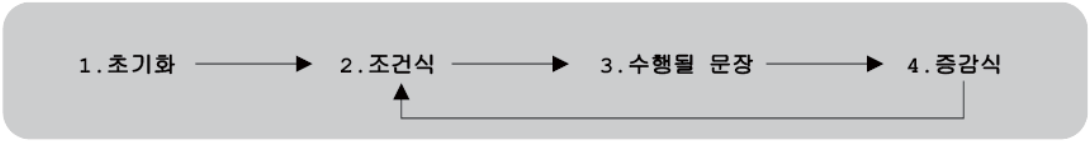


2.2 for문

- 초기화, 조건식, 증감식 그리고 수행할 블럭{} 또는 문장으로 구성

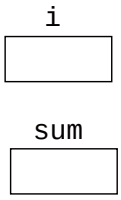
```
for (초기화;조건식;증감식) {
      // 조건식이 true일 때 수행될 문장들을 적는다.
}
```

[참고] 반복하려는 문장이 단 하나일 때는 중괄호{}를 생략할 수 있다.



예) 1부터 10까지의 정수를 더하기

```
int sum = 0;

for(int i=1; i<=10; i++) {
    sum += i; // sum = sum + i;
}
```



i	sum
1	
2	
3	
4	
...	
10	

2.2 for문 – 작성예(examples)

for문 작성 예	설 명

for(;;)
{
/* 반복해서 수행할
}
	조건식이 없기 때문에 결과가 true로 간주되

for(int i=0;;)
{
/* 반복해서 수행할
}	

for(int i=1,j=1;i<10 &&
{
/* 반복해서 수행할
}	

```
public static void main(String[] args)
{
    int sum = 0;
    int i;

    for(i=1; i<=10;i++) {
        sum += i; // sum = sum + i;
    }

    System.out.println(i-1 + "까지의 합: " + sum);

}
```

```
public static void main(String[] args)
{
    int sum = 0

    for(int i=1; i <= 10; i++) {
        sum += i ;        //  sum = sum + i;
    }
    System.out.println( i-1 + " 까지의 합: " + sum);
}
```

변수 sum 이 유효한 범위

변수 i가 유효한 범위

←에러발생

2.3 중첩for문

- for문 안에 또 다른 for문을 포함시킬 수 있다.
- for문의 중첩횟수에는 거의 제한이 없다.

```
for(int i=2; i<=9; i++) {
    for(int j=1; j<=9; j++) {
        System.out.println(i+" * "+j+" = "+i*j);
    }
}
```

```
for(int i=2; i<=9; i++)
    for(int j=1; j<=9; j++)
        System.out.println(i+" * "+j+" = "+i*j);
```

```
i * j = i*j
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
   …
2 * 9 = 18
3 * 1 = 3
3 * 2 = 6
   …
9 * 8 = 72
9 * 9 = 81
```

```
for(int i=1; i<=3; i++) {
    for(int j=1; j<=3; j++) {
        for(int k=1; k<=3; k++) {
            System.out.println(""+i+j+k);
        }
    }
}
```

```
for(int i=1; i<=3; i++)
    for(int j=1; j<=3; j++)
        for(int k=1; k<=3; k++)
            System.out.println(""+i+j+k);
```

```
ijk
111
112
113
121
122
123
…
331
332
333
```

2.4 while문

- 조건식과 수행할 블럭{} 또는 문장으로 구성

```
while (조건식) {
    // 조건식의 연산결과가 true일 때 수행될 문장들을 적는다.
}
```

```
int i=10;

while(i >= 0) {
    System.out.println(i--);
}
```

```
for(int i=10;i>=0;i--) {
    System.out.println(i);
}
```

```
int i=0;

while(i >= 0) {
    i=10;
    System.out.println(i--);
}
```

```
int i=10;

while(i < 10) {
    System.out.println(i--);
}
```

2.5 중첩while문

- while문 안에 또 다른 while문을 포함시킬 수 있다.
- while문의 중첩횟수에는 거의 제한이 없다.

```
for(int i=2; i<=9; i++) {
    for(int j=1; j<=9; j++) {
        System.out.println(i+" * "+j+" = "+i*j);
    }
}
```

```
int i=2;
while(i <= 9) {
    int j=1;
    while(j <= 9) {
        System.out.println(i+" * "+j+" = "+i*j);
        j++;
    }
    i++;
}
```

2.6 do-while문

- while문의 변형. 블럭{}을 먼저 수행한 다음에 조건식을 계산한다.
- 블럭{}이 최소한 1번 이상 수행될 것을 보장한다.

```
do {
    // 조건식의 연산결과가 true일 때 수행될 문장들을 적는다.
} while (조건식);
```

```
class FlowEx24 {
    public static void main(String[] args) throws java.io.IOException {
        int input=0;

        System.out.println("문장을 입력하세요.");
        System.out.println("입력을 마치려면 x를 입력하세요.");
        do {
            input = System.in.read();
            System.out.print((char)input);
        } while(input!=-1 && input !='x');
    }
}
```

문자	코드
…	…
A	65
B	66
C	67
…	…
a	97
b	98
c	99
…	…
x	120
…	…

2.7 break문

- 자신이 포함된 하나의 반복문 또는 switch문을 빠져 나온다.
- 주로 if문과 함께 사용해서 특정 조건을 만족하면 반복문을 벗어나게 한다.

```
class FlowEx25
{
   public static void main(String[] args)
   {
        int sum = 0;
        int i = 0;

        while(true) {
            if(sum > 100)
                break;
            i++;
            sum += i;
        } // end of while

        System.out.println("i=" + i);
        System.out.println("sum=" + sum);
   }
}
```

break문이 수행되면 이 부분은 실행되지 않고 while문을 완전히 벗어난다.

i	sum
0	0
1	1
2	3
3	6
...	...
13	91
14	105

2.8 continue문

- 자신이 포함된 반복문의 끝으로 이동한다.(다음 반복으로 넘어간다.)
- continue문 이후의 문장들은 수행되지 않는다.

```
class FlowEx26
{
   public static void main(String[] args)
   {
        for(int i=0;i <= 10;i++) {
            if (i%3==0)
                continue;
            System.out.println(i);
        }
   }
}
```

조건식이 true가 되어 continue문이 수행되면 반복문의 끝으로 이동한다.
break문과 달리 반복문 전체를 벗어나지 않는다.

[실행결과]

```
1
2
4
5
7
8
10
```

2.9 이름 붙은 반복문과 break, continue

- 반복문 앞에 이름을 붙이고, 그이름을 break, continue와 같이 사용함으로써 둘 이상의 반복문을 벗어나거나 반복을 건너뛰는 것이 가능하다.
- 이와 같은 프로그래밍은 하지 말것! -> 스파게티코드(spaghetti code)

```
class FlowEx27
{
   public static void main(String[] args)
   {
        // for문에 Loop1이라는 이름을 붙였다.
        Loop1 : for(int i=2;i <=9;i++) {
                  for(int j=1;j <=9;j++) {
                       if(j==5)
                            break Loop1;
                       System.out.println(i+"*"+ j +"="+ i*j);
                  } // end of for i
                  System.out.println();
        } // end of Loop1
   }
}
```

[실행결과]

```
2*1=2
2*2=4
2*3=6
2*4=8
```

제 5 장
배열(Array)

인제대학교 전자IT기계자동차공학부

황 원 주

1. 배열(array)
    1.1 배열(array)이란?
    1.2 배열의 선언과 생성
    1.3 배열의 초기화
    1.4 배열의 활용
    1.5 다차원 배열의 선언과 생성
    1.6 가변배열
    1.7 배열의 복사
    1.8 사용자 입력받기 – 커맨드라인, InputDialog

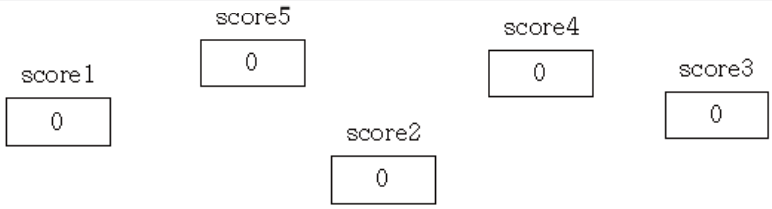
► 7장의 Vector과 비교

1. 배열(array)

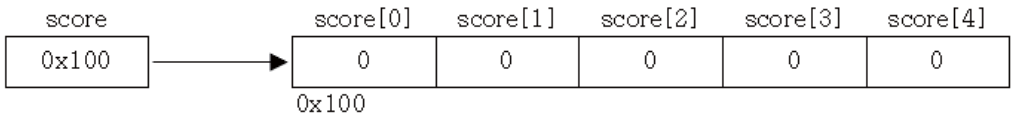
1.1 배열(array)이란?

- 같은 타입의 여러 변수를 하나의 묶음으로 다루는 것
- 많은 양의 값(데이터)을 다룰 때 유용하다.
- 배열의 각 요소는 서로 연속적이다.

```
int score1=0, score2=0, score3=0, score4=0, score5=0 ;
```




```
int[] score = new int[5]; // 5개의 int 값을 저장할 수 있는 배열을 생성한다.
```

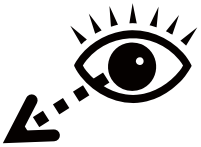


▶ C언어에서 값을 저장하는 장소는?

- -변수: 다른 타입의 여러 변수를 개별로 다루는 것
- -배열: 같은 타입의 여러 변수를 하나의 묶음으로 다루는 것
- -구조체: 다른 타입의 여러 변수를 하나의 묶음으로 다루는 것

```
struct book {
    char title [50]; // 제목
    char author [50]; // 저자
    char publish [50]; // 출판사
    int pages; // 페이지 수
    int price; // 가격
};
```

인제대학교 도서관의 도서정보를 관리하는 프로그램



정보통신공학과 2학년 학생 40명에 대한 객체지향프로그래밍 중간, 기말 성적을 가지고 개인 성적의 합과 중간, 기말고사 각각 모든 학생의 합을 구하는 프로그램

int grade[41][3]

중간고사	기말고사	합
10	20	행합
33	35	행합
13	79	행합
60	45	행합
열합	열합	

0에서부터 9까지의 임의로 뽑은 30 개의 수의 총합과 평균, 그리고 원소의 출현 빈도수를 계산하는 프로그램

```
int frequency[10] = { 0 };
int response[60] = {
        5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
        4, 8, 0, 6, 3, 7, 0, 2, 0, 8,
        7, 8, 0, 5, 8, 7, 3, 9, 7, 8};
```

복소수는 a+ bi로 표현된다(실수부 a와 허수부 b는 실수 값). 두 복소수 a+ bi와 c+ di의 합을 계산하는 프로그램

```
struct complex {
        float real; //실수부
        float img; //허수부
};
```

1.2 배열의 선언과 생성

▶ 1단계: 배열의 선언 (int [] score;)

- 타입 또는 변수이름 뒤에 대괄호[]를 붙여서 배열을 선언한다.
- score가 int형 1차원 배열임을 나타낸다.
- 그러나, 실제 int형 값이 저장될 공간은 아직 할당되지 않았다.

선언방법	선언 예
타입[] 변수이름;	`int[] score;` `String[] name;`
타입 변수이름[];	`int score[];` `String name[];`

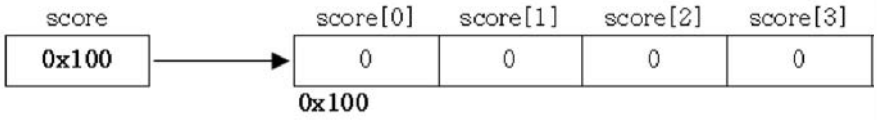
[표5-1] 배열의 선언방법과 선언 예

▶ **2단계: 배열의 생성 (score=new int[5];)**

- 실제 값을 저장할 메모리상의 공간을 할당하기 위해서는 키워드 new를 사용해서 배열을 생성한다.
- 이제 5개의 int형 값을 저장할 수 있는 메모리상의 공간이 할당된다.

```
int[] score;        // 배열을 선언한다.(생성된 배열을 다루는데 사용될 참조변수 선언)
score = new int[5]; // 배열을 생성한다.(5개의 int값을 저장할 수 있는 공간생성)
```

[참고] 위의 두 문장은int[] score = new int[5];와 같이 한 문장으로 줄여 쓸 수 있다.

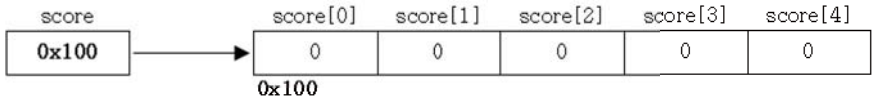


자료형	기본값
boolean	false
char	'\u0000'
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d 또는 0.0
참조형 변수	null

1.3 배열의 초기화

- 생성된 배열에 처음으로 값을 저장하는 것

```
int[] score = new int[5]; // 크기가 5인 int형 배열을 생성한다.
score[0] = 100;           // 각 요소에 직접 값을 저장한다.
score[1] = 90;
score[2] = 80;
score[3] = 70;
score[4] = 60;
```



- 초기화 방법 (2가지): 가능한 간단한 1번 방법으로 초기화하자.

```
int[] score = { 100, 90, 80, 70, 60};          // 1번
int[] score = new int[]{ 100, 90, 80, 70, 60}; // 2번
```

- 그러나 아래의 경우에는 2번의 방법으로만 초기화해야 한다.

```
int[] score;
score = { 100, 90, 80, 70, 60}; // 에러 발생!!!

int[] score;
score = new int[]{ 100, 90, 80, 70, 60}; // OK
```

```
int add(int[] arr) { /* 내용 생략 */}

int result = add({ 100, 90, 80, 70, 60});          // 에러 발생!!!
int result = add(new int[]{ 100, 90, 80, 70, 60}); // OK
```

1.4 배열의 활용

▶ 배열에 값을 저장하고 읽어오기

```
score[3] = 100;       // 배열 score의 4번째 요소에 100을 저장한다.
int value = score[3]; // 배열 score의 4번째 요소에 저장된 값을 읽어서 value에 저장.
```

▶'배열이름.length'는 배열의 크기를 알려준다.

```
int[] score = { 100, 90, 80, 70, 60, 50 };

for(int i=0; i < 6; i++) {
    System.out.println(socre[i]);
}

for(int i=0; i < score.length; i++) {
    System.out.println(socre[i]);
}
```

▶로또번호 생성하는 예제

[예제5-4]/ch5/ArrayEx4.java

```
class ArrayEx4 {
  public static void main(String[] args)
  {
      // 45개의 정수값을 저장하기 위한 배열 생성.
      int[] ball = new int[45];

      // 배열의 각 요소에 1~45의 값을 저장한다.
      for(int i=0; i < ball.length; i++)
          ball[i] = i+1;    // ball[0]에 1이 저장된다.

      int temp = 0;  // 두 값을 바꾸는데 사용할 임시변수
      int j = 0;     // 임의의 값을 얻어서 저장할 변수

      // 배열에 저장된 값이 잘 섞이도록 충분히 큰 반복횟수를 지정한다.
      // 배열의 첫 번째 요소와 임의의 요소에 저장된 값을 서로 바꿔서 값을 섞는다.
      for(int i=0; i < 100; i++) {
          j = (int)(Math.random() * 45); // 배열 범위(0~44)의 임의의 값을 얻는다.
          temp = ball[0];
          ball[0] = ball[j];
          ball[j] = temp;
      }
      // 배열 ball의 앞에서 부터 6개의 요소를 출력한다.
      for(int i=0; i < 6; i++)
          System.out.print(ball[i]+" ");
  }
}
```

ball[0]과 ball[j]의 값을 서로 바꾼다.

▶16진수(CAFE) -> 2진수(1100101011111110) 예제

[예제5-7]/ch5/ArrayEx7.java

```
class ArrayEx7
{
   public static void main(String[] args)
   {
       char[] hex = { 'C', 'A', 'F', 'E'}; //변환하고자 하는 16진수

       String[] binary = {   "0000", "0001", "0010", "0011"
                           , "0100", "0101", "0110", "0111"
                           , "1000", "1001", "1010", "1011"
                           , "1100", "1101", "1110", "1111" };

       String result="";

       for (int i=0; i < hex.length ; i++ ) {
           if(hex[i] >='0' && hex[i] <='9') {
               result +=binary[hex[i]-'0'];    // '8'-'0'의 결과는 8이다.
           } else { // A~F이면
               result +=binary[hex[i]-'A'+10]; // 'C'-'A'의 결과는 2
           }
       }

       System.out.println("hex:"+ new String(hex));
       System.out.println("binary:"+result);
   }
}
```

문자	코드
...	...
0	48
1	49
2	50
...	...
A	65
B	66
C	67
D	68
E	69
F	70
...	...

1.5 다차원 배열의 선언과 생성

-'[]'의 개수가 차원의 수를 의미한다.

선언방법	선언예
타입[][] 변수이름;	int[][] score;
타입 변수이름[][];	int score[][];
타입[] 변수이름[];	int[] score[];

[표5-3] 2차원 배열의 선언

```
int[][] score = new int[5][3];      // 5행 3열의 2차원 배열을 생성한다.
```

```
int[][] score = {
                    {100,100,100},
                    { 20, 20, 20},
                    { 30, 30, 30},
                    { 40, 40, 40},
                    { 50, 50, 50},
                };
```

- 배열의 생성과 초기화를 동시에
- new int[][]는 생략 가능

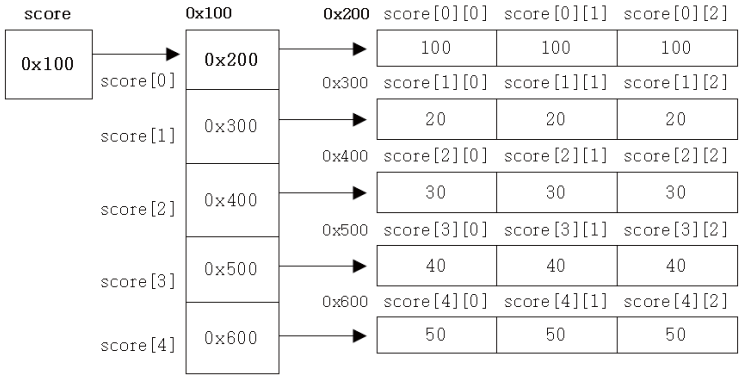
부분배열: score[0], score[1], score[2], score[3], score[4]

-배열명(score)과 부분배열: 주소상수

-score[0][0]: 값

	국어	영어	수학
1	100	100	100
2	20	20	20
3	30	30	30
4	40	40	40
5	50	50	50

```
for (int i=0; i < score.length; i++) {
    for (int j=0; j < score[i].length; j++) {
        score[i][j] = 10;
    }
}
```



[그림5-2] 2차원 배열

1.6 가변배열

- 다차원 배열에서 마지막 차수의 크기를 지정하지 않고 각각 다르게 지정.

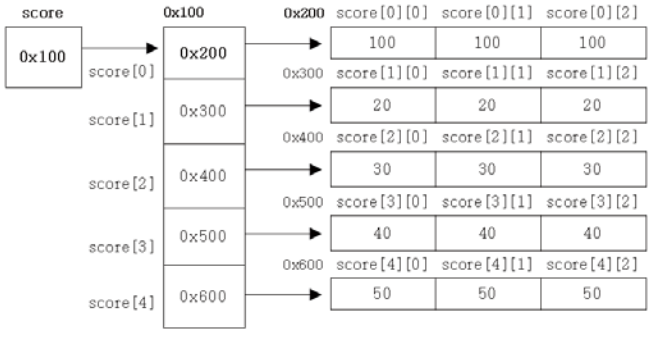
```
int[][] score = new int[5][3];      // 5행 3열의 2차원 배열을 생성한다.
```

```
int[][] score = new int[5][];
score[0] = new int[3];
score[1] = new int[3];
score[2] = new int[3];
score[3] = new int[3];
score[4] = new int[3];
```

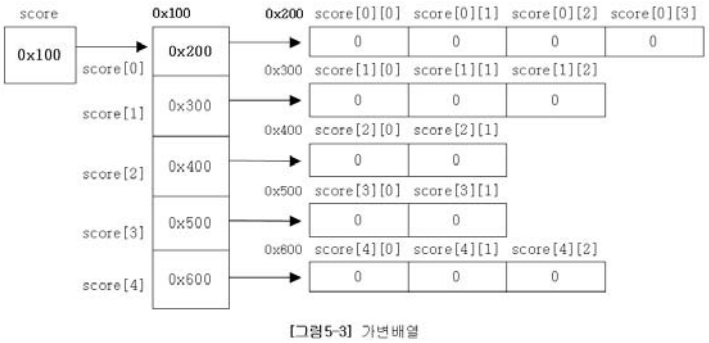
```
int[][] score =
    {
        {100,100,100},
        { 20, 20, 20},
        { 30, 30, 30},
        { 40, 40, 40},
        { 50, 50, 50},
    };
```

```
int[][] score = new int[5][];
score[0] = new int[4];
score[1] = new int[3];
score[2] = new int[2];
score[3] = new int[2];
score[4] = new int[3];
```

```
int[][] score =
    {
        {100,100,100,100},
        { 20, 20, 20},
        { 30, 30},
        { 40, 40},
        { 50, 50, 50},
    };
```



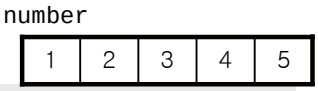
[그림5-2] 2차원 배열

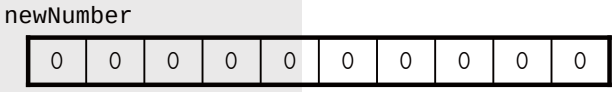


[그림5-3] 가변배열

1.7 배열의 복사

▶ for문을 이용한 배열의 복사





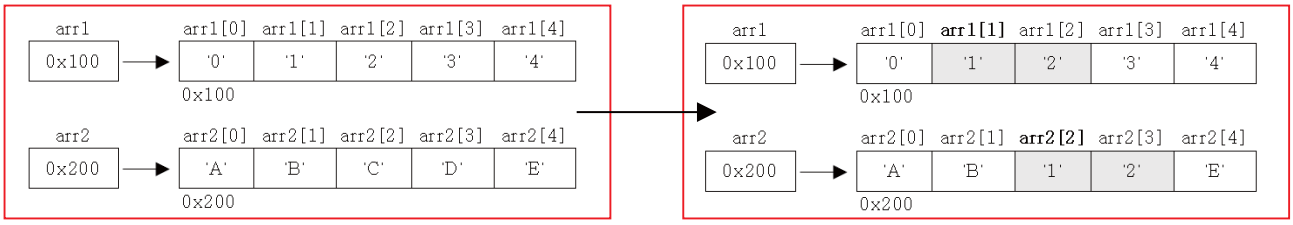

```
int[] number = {1,2,3,4,5};
int[] newNumber = new int[10];

for(int i=0; i<number.length;i++) {
    newNumber[i] = number[i];// 배열 number의 값을 newNumber에 저장한다.
}
```

▶ System.arraycopy()를 이용한 배열의 복사

```
System.arraycopy(arr1, 0, arr2, 0, arr1.length);
```

arr1[0]에서 arr2[0]으로 arr1.length개의 데이터를 복사

```
System.arraycopy(arr1, 1, arr2, 2, 2);
```



1.8 사용자 입력받기 - 커맨드라인

▶ 커맨드라인에서 입력된 값들은 문자열 배열에 담겨 main메서드에 전달된다.

[예제5-13]/ch5/ArrayEx13.java

```
class ArrayEx13
{
   public static void main(String[] args)
   {
        System.out.println("매개변수의 개수:"+args.length);
        for(int i=0;i< args.length;i++) {
            System.out.println("args[" + i + "] = \""+ args[i] + "\"");
        }

   }
}
```

[실행결과]

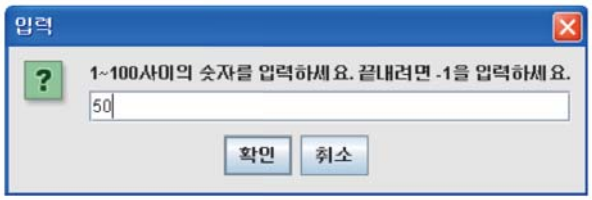
```
C:\jdk1.5\work>java ArrayEx13 abc 123 "Hello world"
매개변수의 개수:3
args[0] = "abc"
args[1] = "123"
args[2] = "Hello world"
```

1.9 사용자 입력받기 – 입력창(InputDialog)

▶ Swing패키지의 JOptionPane.showInputDialog()를 사용

[예제5-16]/ch5/ArrayEx16.java

```
import javax.swing.*;           // JOptionPane클래스를 사

class ArrayEx16 {
  public static void main(String[] args)
  {
      // 1~100사이의 임의의값을 얻어서 answer에 저장한다.
      int answer = (int)(Math.random() * 100) + 1;
      int input = 0;            // 사용자입력을 저장할 공간
      String temp = "";         // 사용지입력을 저장할 임시공간
      int count = 0;            // 시도횟수를 세기위한 변수

      do {
          count++;
          temp = JOptionPane.showInputDialog("1~100사이의 숫자를 입력하세요."
                                    + " 끝내려면 -1을 입력하세요.");

          // 사용자가 취소버튼을 누르거나 -1을 입력하면 do-while문을 빗어난다.
          if(temp==null || temp.equals("-1")) break;

          System.out.println("입력값 : "+temp);

          // 사용자입력을 문자열로 받아오기 때문에 int로 변환해 주어야한다.
          input = Integer.parseInt(temp);
```

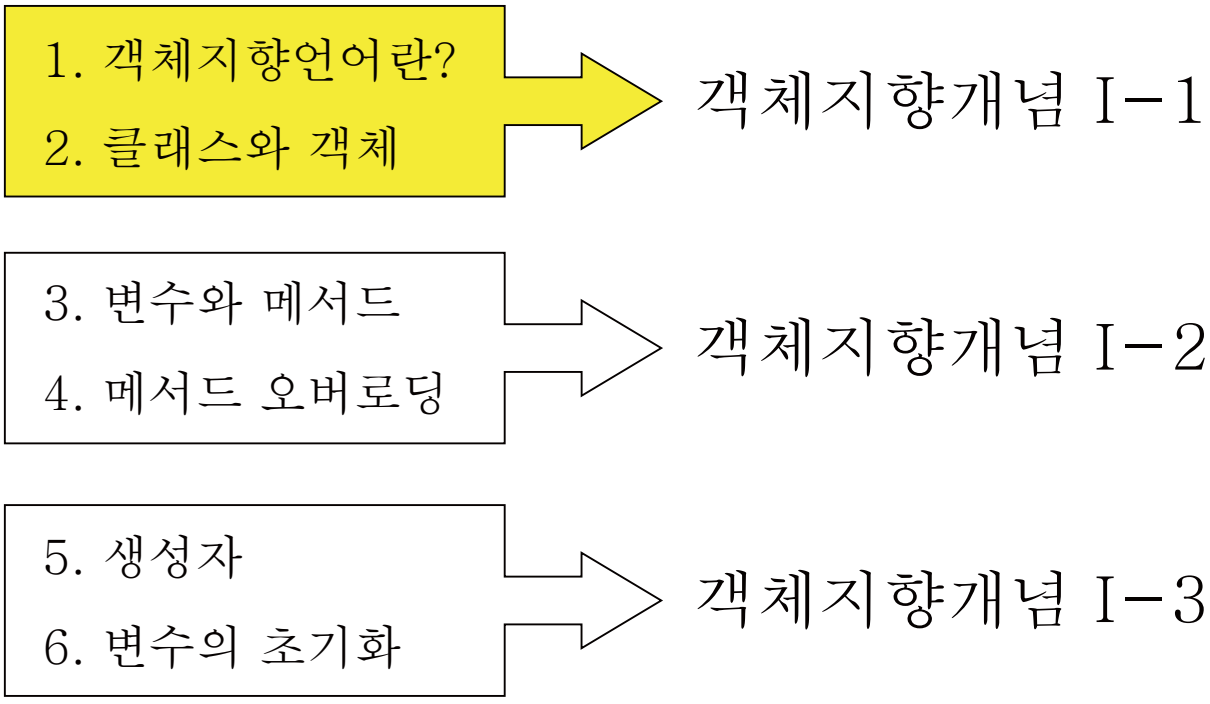


[그림5-4] JOptionPane.showInputDialog()에 의해서 생성된 입력창

제 6 장
객체지향개념 I-1

인제대학교 전자IT기계자동차공학부

황 원 주



1. 객체지향언어란?

1.1 객체지향언어의 역사

- 과학, 군사적 모의실험(simulation)을 위해 컴퓨터를 이용한 가상세계를 구현하려는 노력으로부터 객체지향이론이 시작됨
- 1960년대 최초의 객체지향언어 Simula탄생
- 1980년대 절차방식의 프로그래밍의 한계를 객체지향방식으로 극복하려고 노력함.(C++, Smalltalk과 같은 보다 발전된 객체지향언어가 탄생)
- 1995년 말 Java탄생. 객체지향언어가 프로그래밍 언어의 주류가 됨.

1.2 객체지향언어의 특징

▶ 정보은닉(Information hiding)

- 객체 내부의 변수는 객체 외부에 감춘다.
- 클래스 멤버 변수는 private으로 선언

▶ 데이터 캡슐화(Data encapsulation)

-객체 외부에 감추어진 객체 내부의 변수는 메서드를 통해 액세스한다.

-클래스 멤버 메서드는 public으로 선언

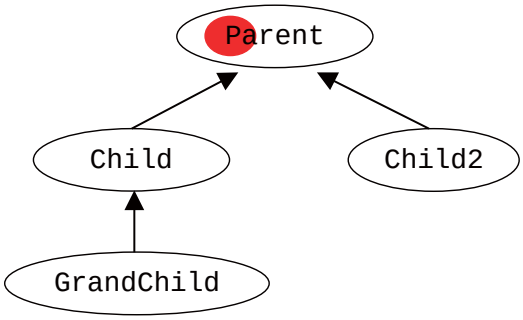
▶ 상속성(Inheritance)

- 객체 구성은 계층구조로 되어 있으며 하위클래스 객체는 상위클래스 객체의 특성을 물려 받는다.

-extends를 이용하여 상속

▶ 다형성(Polymorphism)

- 하나의 연산자나 함수가 여러 가지 기능을 가질 수 있다.
- 연산자 오버로딩(+ 연산자), 메서드 오버로딩



1.2 객체지향언어의 특징

▶ 기존의 프로그래밍언어와 크게 다르지 않다.

– 기존의 프로그래밍 언어에 몇가지 규칙을 추가한 것일 뿐이다.

▶ 코드의 재사용성이 높다.

– 새로운 코드를 작성할 때 기존의 코드를 이용해서 쉽게 작성할 수 있다.

▶ 코드의 관리가 쉬워졌다.

– 코드간의 관계를 맺어줌으로써 보다 적은 노력으로 코드변경이 가능하다.

▶ 신뢰성이 높은 프로그램의 개발을 가능하게 한다.

– 제어자와 메서드를 이용해서 데이터를 보호하고, 코드의 중복을 제거하여 코드의 불일치로 인한 오류를 방지할 수 있다.

1.3 Object can be everything!

▶ 객체의 예: 내 여자친구 M

–내 여자친구 M은 약간 통통하고 귀여운 여자입니다. M은 먹는 것을 매우 좋아하고, 배고플 때에는 매우 신경질적으로 나들 대합니다. 만일 그녀가 배고플 때 말을 걸면 그녀는 거친 말이 나오고, 그렇지 않은 경우에는 고운 말이 나왔습니다.

–내가 데이트를 신청하기 위해서는 그녀가 배고픈지 여부가 매우 중요합니다. 그러나 겉으로는 배고픈지를 직접 볼 수 없으므로 넌지시 말을 걸어보아서 알아봅니다. 배고픈 상태의 M은 신경질을 내면서 데이트 신청을 받아 주질 않으므로 캔디를 주어서 배를 부르게 한 후에 데이트를 신청할 수 있었습니다.

객체 M

```
boolean hungry;
talk() {
  if (hungry==true)
    return (거친말)
  else return(고운말)
}
give_candy() {
  hungry = false;
}
```

2. 클래스와 객체

2.1 클래스와 객체의 정의와 용도

▶ 클래스의 정의 – 클래스란 객체를 정의해 놓은 것이다.

▶ 클래스의 용도 – 클래스는 객체를 생성하는데 사용된다.

▶ 객체의 정의 – 실제로 존재하는 것. 사물 또는 개념.

▶ 객체의 용도 – 객체의 속성과 기능에 따라 다름.

클래스	객체
제품 설계도	제품
TV설계도	TV
붕어빵기계	붕어빵

2.2 객체와 인스턴스

▶ 객체 ≒ 인스턴스

– 객체(object)는 인스턴스(instance)를 포함하는 일반적인 의미

책상은 인스턴스다.	책상은 책상 클래스의 객체다.
책상은 객체다.	책상은 책상 클래스의 인스턴스다.

좀 더 자연스러운 표현

▶ 인스턴스화(instantiate, 인스턴스化)

– 클래스로부터 인스턴스를 생성하는 것.

클래스 —인스턴스화→ 인스턴스(객체)

2.3 객체의 구성요소 – 속성과 기능

▶ 객체는 속성과 기능으로 이루어져 있다.

- 객체는 속성과 기능의 집합이며, 속성과 기능을 객체의 멤버(member, 구성요소)라고 한다.

▶ 속성은 변수로, 기능은 메서드로 정의한다.

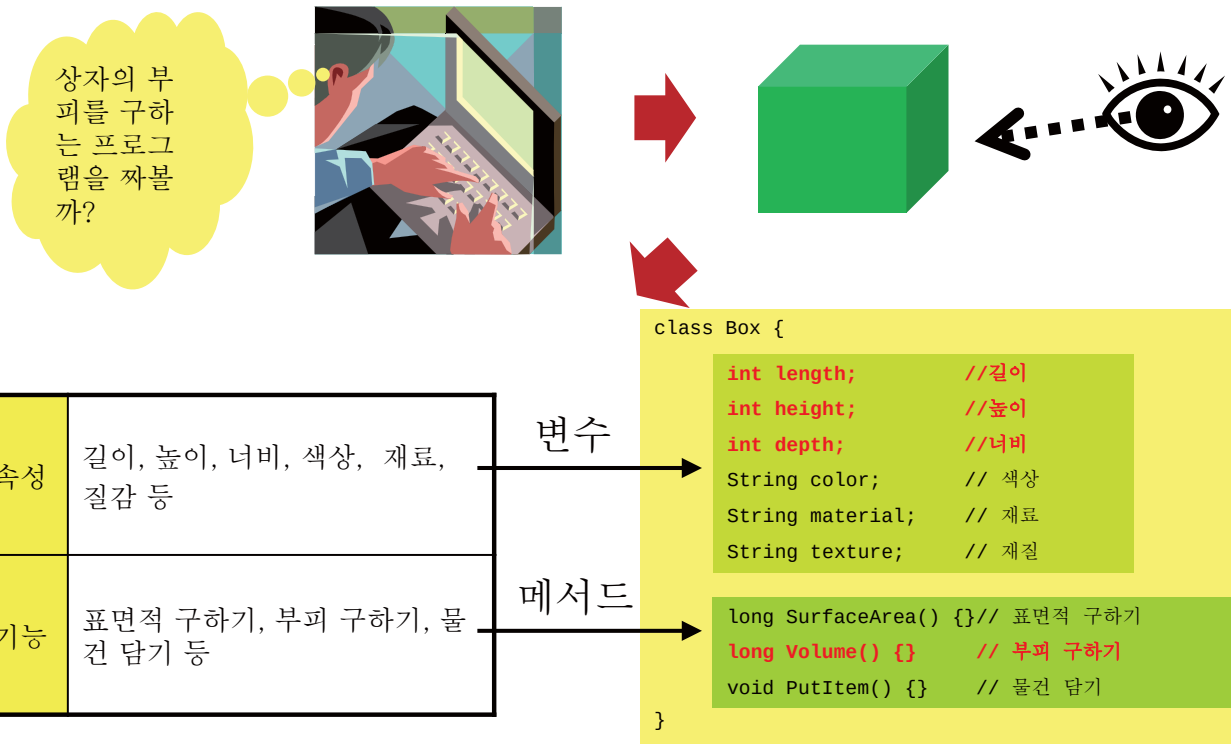
- 클래스를 정의할 때 객체의 속성은 변수로, 기능은 메서드로 정의한다.

예: TV

속성	크기, 길이, 높이, 색상, 볼륨, 채널 등	변수
기능	켜기, 끄기, 볼륨 높이기, 볼륨 낮추기, 채널 높이기 등	메서드

```
class Tv {
    String color;  // 색깔
    boolean power; // 전원상태(on/off)
    int channel;   // 채널

    void power() { power = !power; } // 전원on/off
    void channelUp( channel++;)      // 채널 높이기
    void channelDown {channel--;}    // 채널 낮추기
}
```

다른 예: 상자



매우 중요한 예제

```
class Box {
  private int width;          // 멤버 변수
  private int height;
  private int depth;
  Box(int a, int b, int c) {  // 생성자
    width = a;
    height = b;
    depth = c;
  }
  public long Volume() { // 메소드
         return(width * height * depth);
  }
}
class BoxExample {
   public static void main(String args[])
   { long vol;
     Box b = new Box(2, 2, 3);  // box 객체의 선언 및 할당
     vol=b.Volume();  // 객체 b의 Volume()을 호출
     System.out.println("Volume="+vol);
   }
}
```

2.4 인스턴스의 생성과 사용

① class 정의

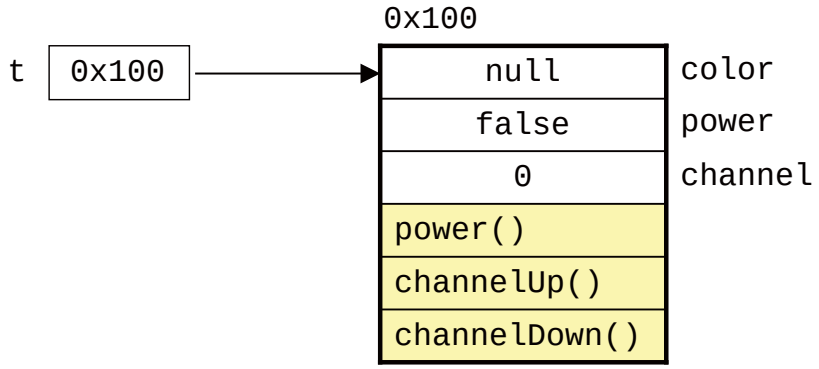
② 객체참조변수 선언

```
클래스명 참조변수명; // 객체를 다루기 위한 객체참조변수 선언
```

③ 객체 할당

```
참조변수명 = new 클래스명();   // new를 이용하여 객체생성 후, 생성된 객체의 주소를 객체참조변수에 저장
```

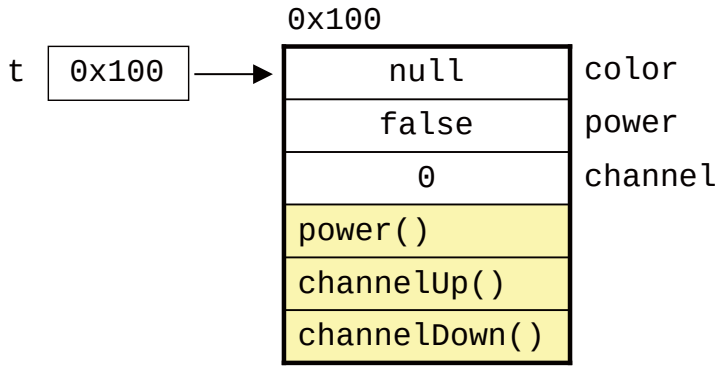
```
Tv t;
t = new Tv();
```




```
Tv t = new Tv();
```

```
Tv t;
t = new Tv();
t.channel = 7;
t.channelDown();
System.out.println(t.channel);
```

```
class Tv {
      String color;  // 색깔
      boolean power; // 전원상태(on/off)
      int channel;   // 채널
      void power() { power = !power; } // 전원on/off
      void channelUp( channel++;)      // 채널 높이기
      void channelDown {channel--;}    // 채널 낮추기
}
```



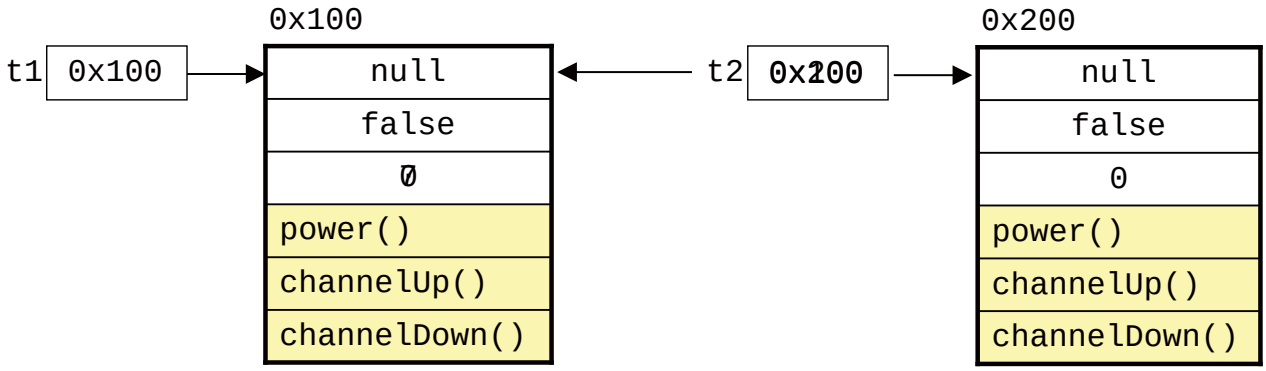
예제 6-1

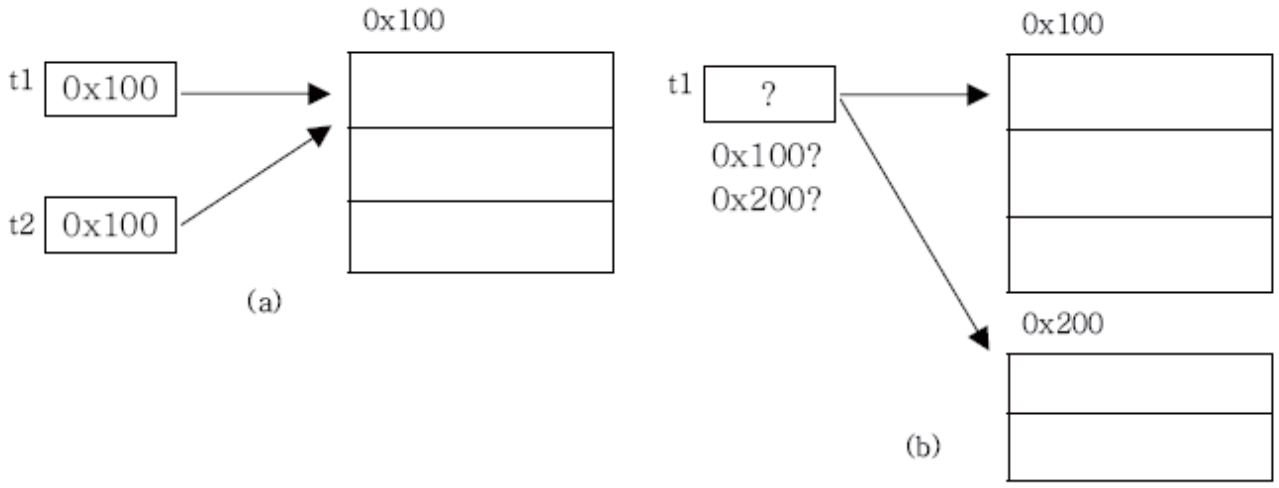
```
class Tv {
    // Tv의 속성(멤버변수)
    String color;                  // 색상
    boolean power;                 // 전원상태(on/off)
    int channel;                   // 채널

    // Tv의 기능(메서드)
    void power() {   power = !power; }   /* TV를 켜거나 끄는 기능을 하는 메서드 */
    void channelUp() {     ++channel; }   /* TV의 채널을 높이는 기능을 하는 메서드 */
    void channelDown() {    --channel; } /* TV의 채널을 낮추는 기능을 하는 메서드 */
}

class TvTest {
     public static void main(String args[]) {
          Tv t;                  // Tv인스턴스를 참조하기 위한 객체참조변수 t를 선언
          t = new Tv();          // Tv인스턴스를 생성한다.
          t.channel = 7;         // Tv인스턴스의 멤버변수 channel의 값을 7로 한다.
          t.channelDown();       // Tv인스턴스의 메서드 channelDown()을 호출한다.
          System.out.println("현재 채널은 " + t.channel + " 입니다.");
     }
}
```

실행결과

```
현재 채널은 6 입니다.
```

```
Tv t1 = new Tv();
Tv t2 = new Tv();
t2 = t1;
t1.channel = 7;
System.out.println(t1.channel);
System.out.println(t2.channel);
```





(a) 하나의 인스턴스를 여러 개의 참조변수가 가리키는 경우(가능)
(b) 여러 개의 인스턴스를 하나의 참조변수가 가리키는 경우(불가능)

[그림6-2] 참조변수와 인스턴스의 관계

예제 6-3

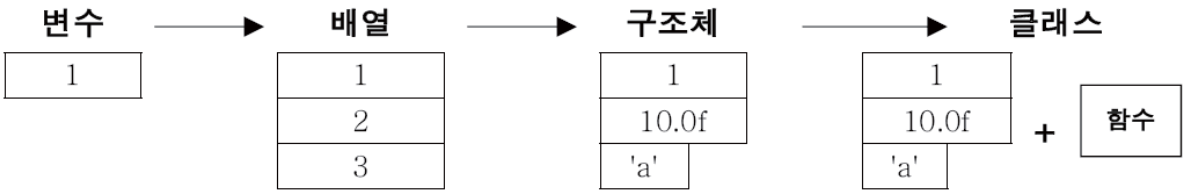
```
class Tv {
     // Tv의 속성(멤버변수)
     String color;          // 색상
     boolean power;         // 전원상태(on/off)
     int channel;           // 채널

     // Tv의 기능(메서드)
     void power() {   power = !power; }   /* TV를 켜거나 끄는 기능을 하는 메서드 */
     void channelUp() {     ++channel; }   /* TV의 채널을 높이는 기능을 하는 메서드 */
     void channelDown() {    --channel; } /* TV의 채널을 낮추는 기능을 하는 메서드 */
}

class TvTest2 {
        public static void main(String args[]) {
                Tv t1 = new Tv();
                Tv t2 = new Tv();
                System.out.println("t1의 channel값은 " + t1.channel + "입니다.");
                System.out.println("t2의 channel값은 " + t2.channel + "입니다.");

                t2 = t1; // t1이 저장하고 있는 값(주소)을 t2에 저장한다.
                t1.channel = 7;  // channel 값을 7으로 한다.
                System.out.println("t1의 channel값을 7로 변경하였습니다.");

                System.out.println("t1의 channel값은 " + t1.channel + "입니다.");
                System.out.println("t2의 channel값은 " + t2.channel + "입니다.");
        }
}
```

실행결과

```
t1의 channel값은 0입니다.
t2의 channel값은 0입니다.
t1의 channel값을 7로 변경하였습니다.
t1의 channel값은 7입니다.
t2의 channel값은 7입니다.
```

2.5 클래스의 또 다른 정의

1. 클래스 – 데이터와 함수의 결합



[그림6-3] 데이터 저장개념의 발전과정

- ▶ 변수 – 하나의 데이터를 저장할 수 있는 공간
- ▶ 배열 – 같은 타입의 여러 데이터를 저장할 수 있는 공간
- ▶ 구조체 – 타입에 관계없이 서로 관련된 데이터들을 저장할 수 있는 공간
- ▶ 클래스 – 데이터와 함수의 결합(구조체+ 함수)

2. 클래스 – 사용자 정의 타입(User-defined type)

- 프로그래머가 직접 새로운 타입을 정의할 수 있다.
- 서로 관련된 값을 묶어서 하나의 타입으로 정의한다.

```
class Time {
        int hour;
        int minute;
        int second;
}
```

```
int hour;
int minute;
int second;
```

→

```
Time t = new Time();
```

```
int hour1, hour2, hour3 ;
int minute1, minute2. minute3;
int second1, secound2, second3;
```

→

```
Time t1 = new Time();
Time t2 = new Time();
Time t3 = new Time();
```

```
int[] hour = new int[3];
int[] minute = new int[3];
int[] second = new int[3];
```

→

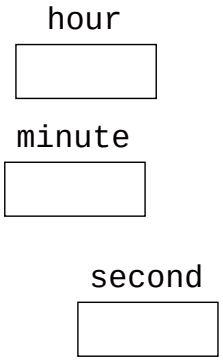
```
Time[] t = new Time[3];
t[0] = new Time();
t[1] = new Time();
t[2] = new Time();
```

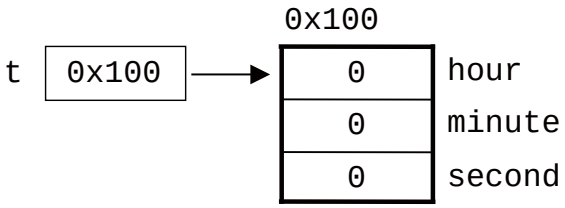
```
int hour;
int minute;
int second;
```

→

```
Time t = new Time();
```

```
class Time {
        int hour;
        int minute;
        int second;
}
```

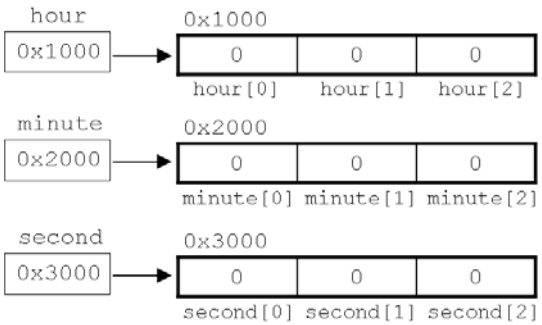


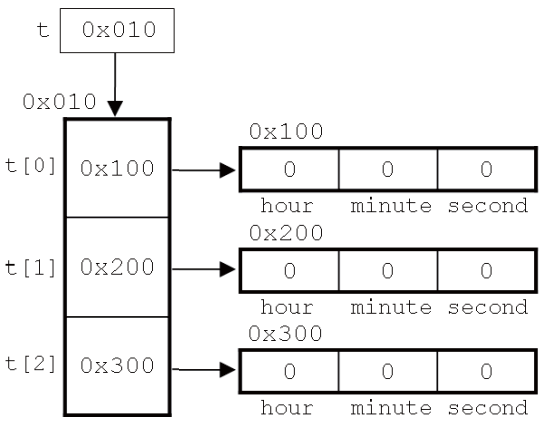



```
int[] hour = new int[3];
int[] minute = new int[3];
int[] second = new int[3];
```

```
Time[] t = new Time[3];
t[0] = new Time();
t[1] = new Time();
t[2] = new Time();
```

```
class Time {
      int hour;
      int minute;
      int second;
}
```

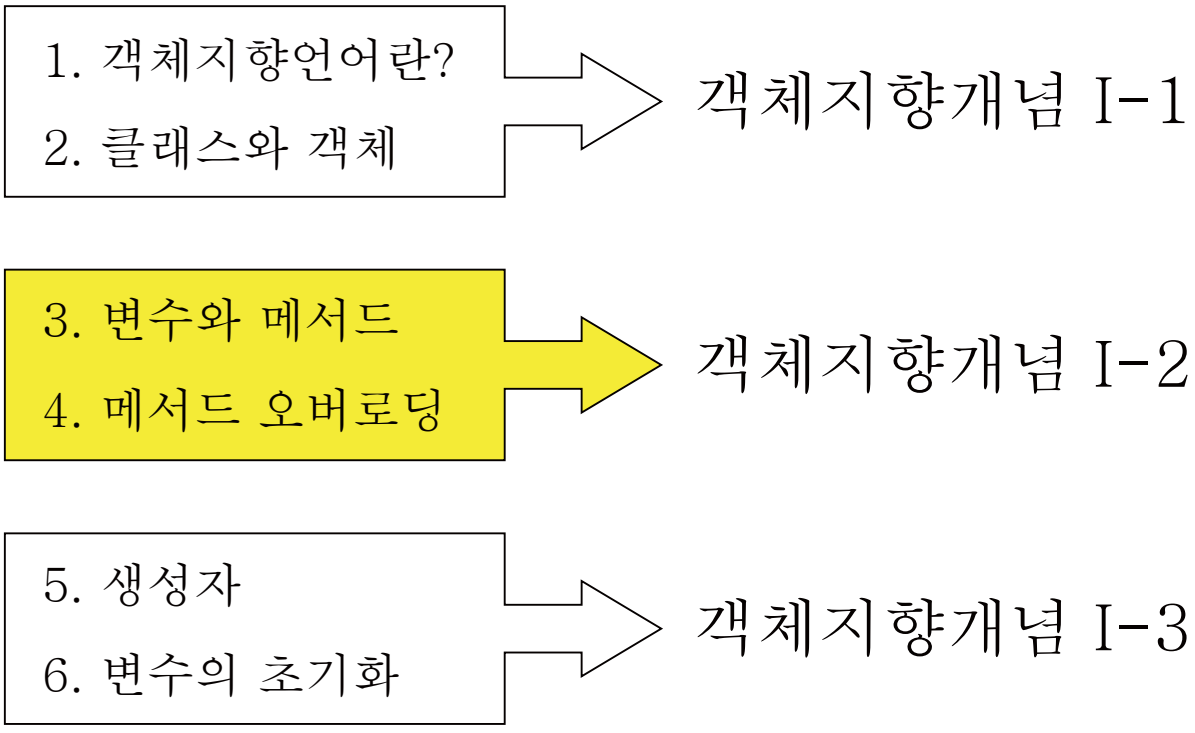




제 6 장
객체지향개념 I-2

인제대학교 전자IT기계자동차공학부

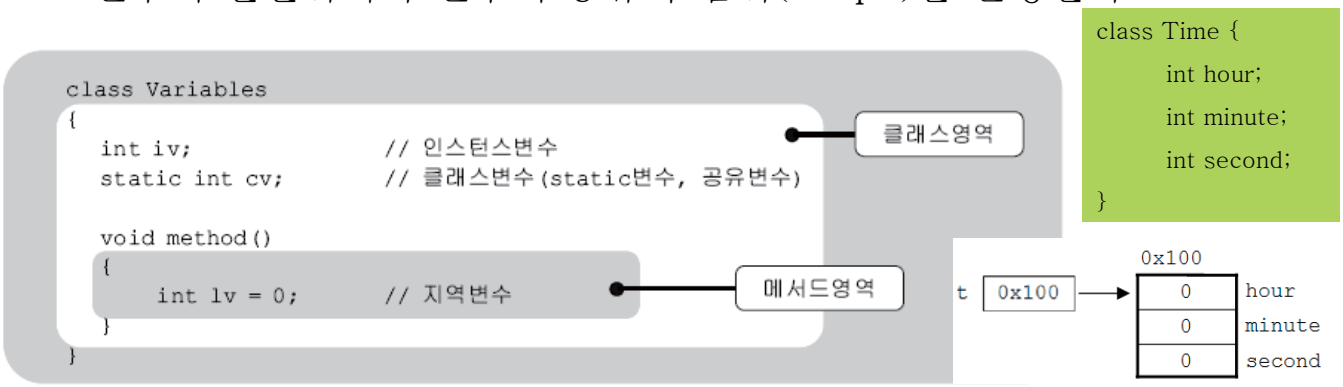
황 원 주



3. 변수와 메서드

3.1 선언위치에 따른 변수의 종류

"변수의 선언위치가 변수의 종류와 범위(scope)을 결정한다."



변수의 종류	선언위치	생성시기
클래스변수	클래스 영역	클래스가 메모리에 올라갈 때
인스턴스변수		인스턴스 생성시
지역변수	메서드 영역	변수 선언문 수행시

▶ 인스턴스변수(instance variable)

- 각 인스턴스의 개별적인 저장공간. 인스턴스마다 다른 값 저장가능
- 인스턴스 생성 후, '참조변수.인스턴스변수명'으로 접근
- 인스턴스를 생성할 때 생성되고, 참조변수가 없을 때 가비지컬렉터에 의해자동 제거됨

▶ 클래스변수(class variable)

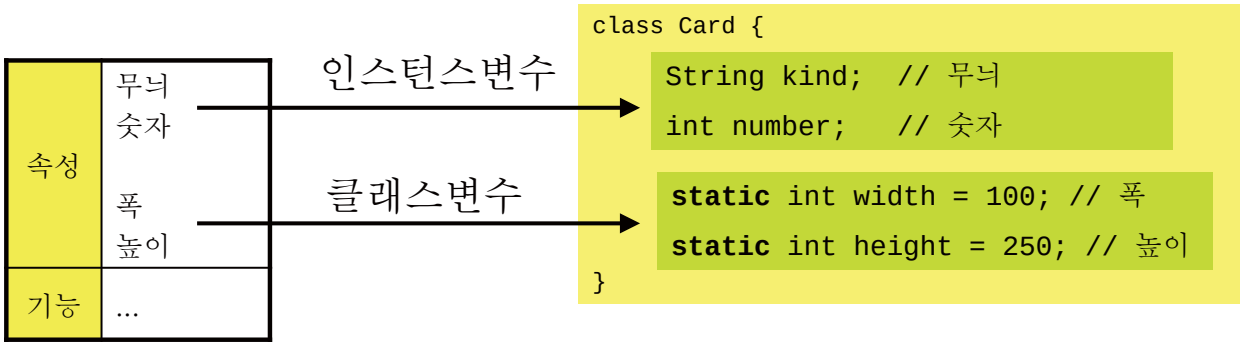
- 같은 클래스의 모든 인스턴스들이 공유하는 변수
- 인스턴스 생성없이 '클래스이름.클래스변수명'으로 접근
- 클래스가 로딩될 때 생성되고 프로그램이 종료될 때 소멸

▶ 지역변수(local variable)

- 메서드 내에 선언되며, 메서드의 종료와 함께 소멸
- 조건문, 반복문의 블럭{} 내에 선언된 지역변수는 블럭을 벗어나면 소멸

3.2 클래스변수와 인스턴스변수

"인스턴스변수는 인스턴스가 생성될 때마다 생성되므로 인스턴스마다 각기 다른 값을 유지할 수 있지만, 클래스변수는 모든 인스턴스가 하나의 저장공간을 공유하므로 항상 공통된 값을 갖는다."



예제 6-4

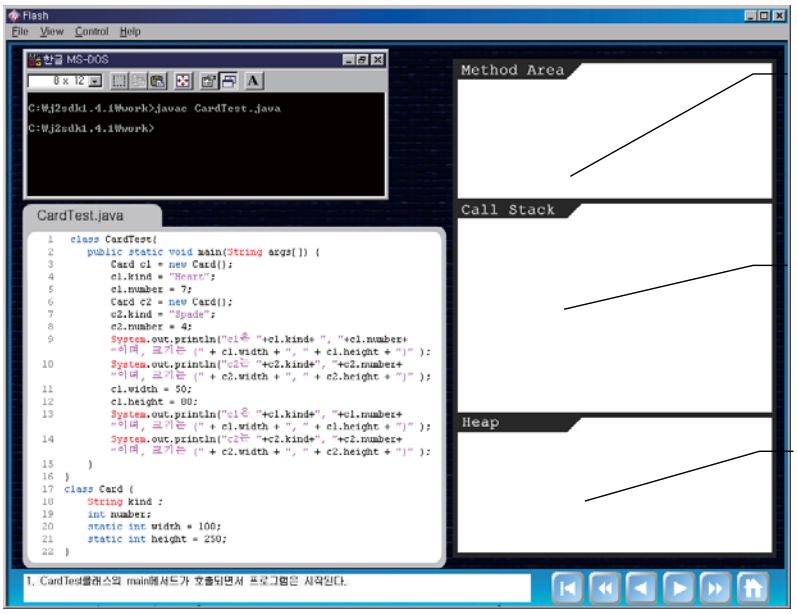
```
class CardTest{
   public static void main(String args[]) {
      // 클래스변수(static 변수)는 객체생성없이 '클래스이름.클래스변수'로 직접 사용 가능하다.
      System.out.println("Card.width = " + Card.width);
      System.out.println("Card.height = " + Card.height);

      Card c1 = new Card();
      c1.kind = "Heart";
      c1.number = 7;

      Card c2 = new Card();
      c2.kind = "Spade";
      c2.number = 4;

      System.out.println("c1은 " + c1.kind + ", " + c1.number + "이며,
                                          크기는 (" + c1.width + ", " + c1.height + ")" );
      System.out.println("c2는 " + c2.kind + ", " + c2.number + "이며,
                                          크기는 (" + c2.width + ", " + c2.height + ")" );
      System.out.println("이제 c1의 width와 height를 각각 50, 80으로 변경합니다.");
      c1.width = 50;       // 클래스변수는 참조변수를 통해서 변경하면 인스턴스변수로 착각할 수 있기 때문에
      c1.height = 80;      // 반드시 클래스이름.클래스변수이름으로 접근!

      System.out.println("c1은 " + c1.kind + ", " + c1.number + "이며,
                                          크기는 (" + c1.width + ", " + c1.height + ")" );
      System.out.println("c2는 " + c2.kind + ", " + c2.number + "이며,
                                          크기는 (" + c2.width + ", " + c2.height + ")" );
      }
}

class Card {
     String kind ;                    // 카드의 무늬 - 인스턴스 변수
     int number;                      // 카드의 숫자 - 인스턴스 변수
     static int width = 100;          // 카드의 폭 - 클래스 변수
     static int height = 250;         // 카드의 높이 - 클래스 변수
}
```

실행결과

```
Card.width = 100
Card.height = 250
c1은 Heart, 7이며, 크기는 (100, 250)
c2는 Spade, 4이며, 크기는 (100, 250)
이제 c1의 width와 height를 각각 50, 80으로 변경합니다.
c1은 Heart, 7이며, 크기는 (50, 80)
c2는 Spade, 4이며, 크기는 (50, 80)
```

3.2 클래스변수와 인스턴스변수

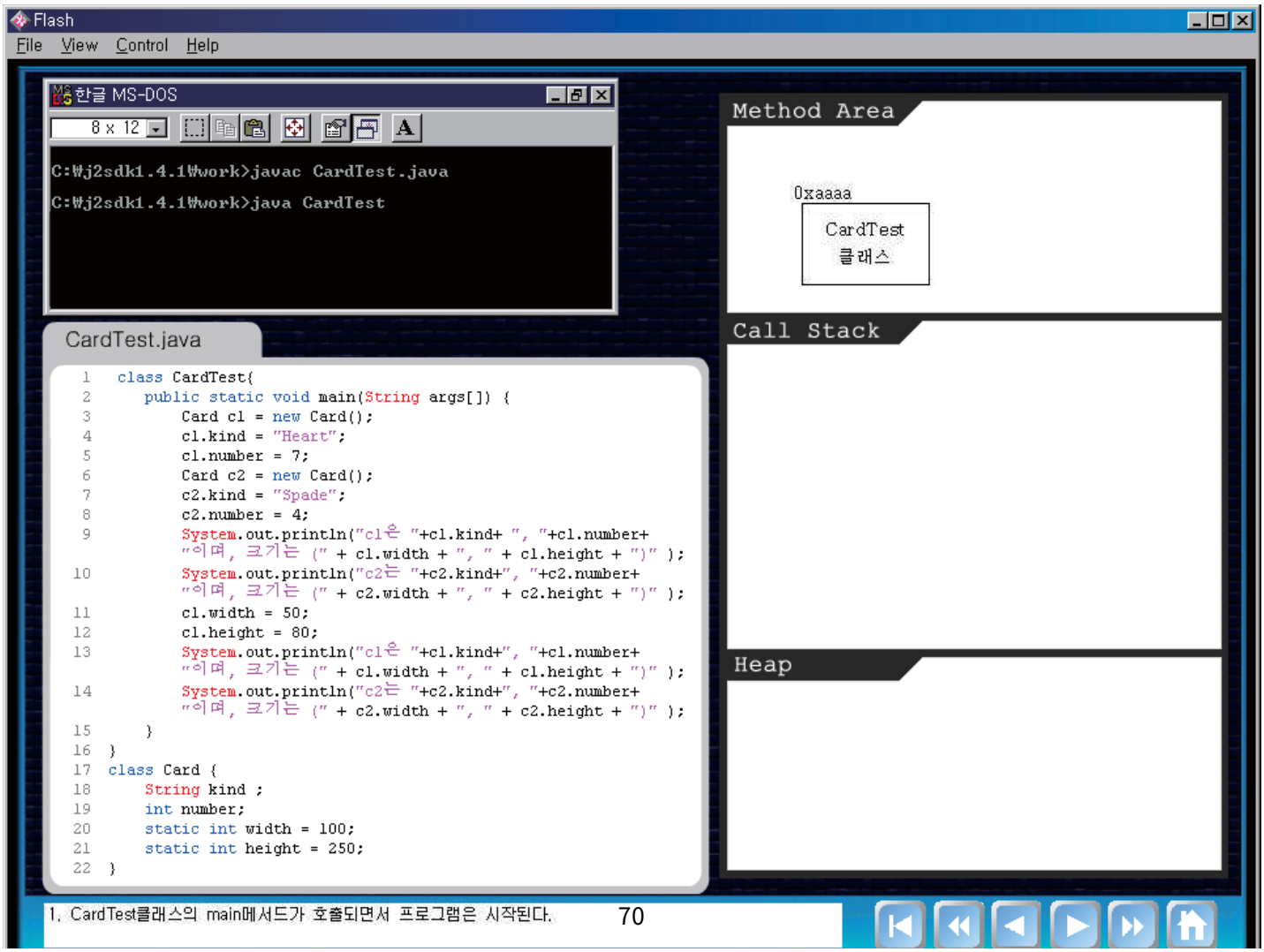
▶ 프로그램의 메모리 영역은 기본적으로 정적(static)영역, 힙(heap)영역, 스택(stack)영역의 3가지로 나뉘어 관리된다.



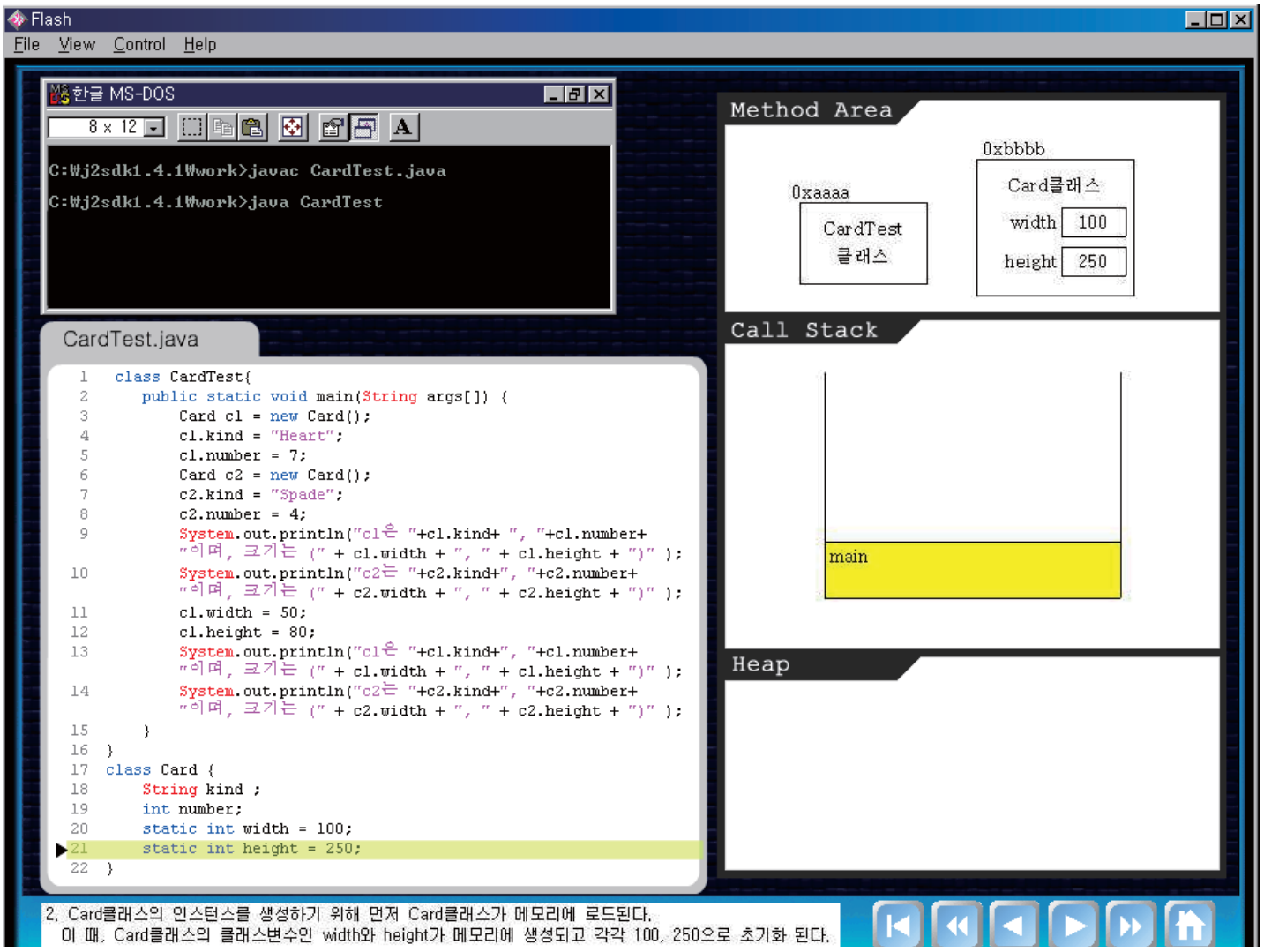
정적영역(Method Area)은 프로그램이 시작될 때 확보되어, 이후 프로그램이 종료될 때까지 배치가 고정된 영역. JVM이 해당 클래스를 로드하면 클래스정보, 클래스 변수가 정적영역에 올라감

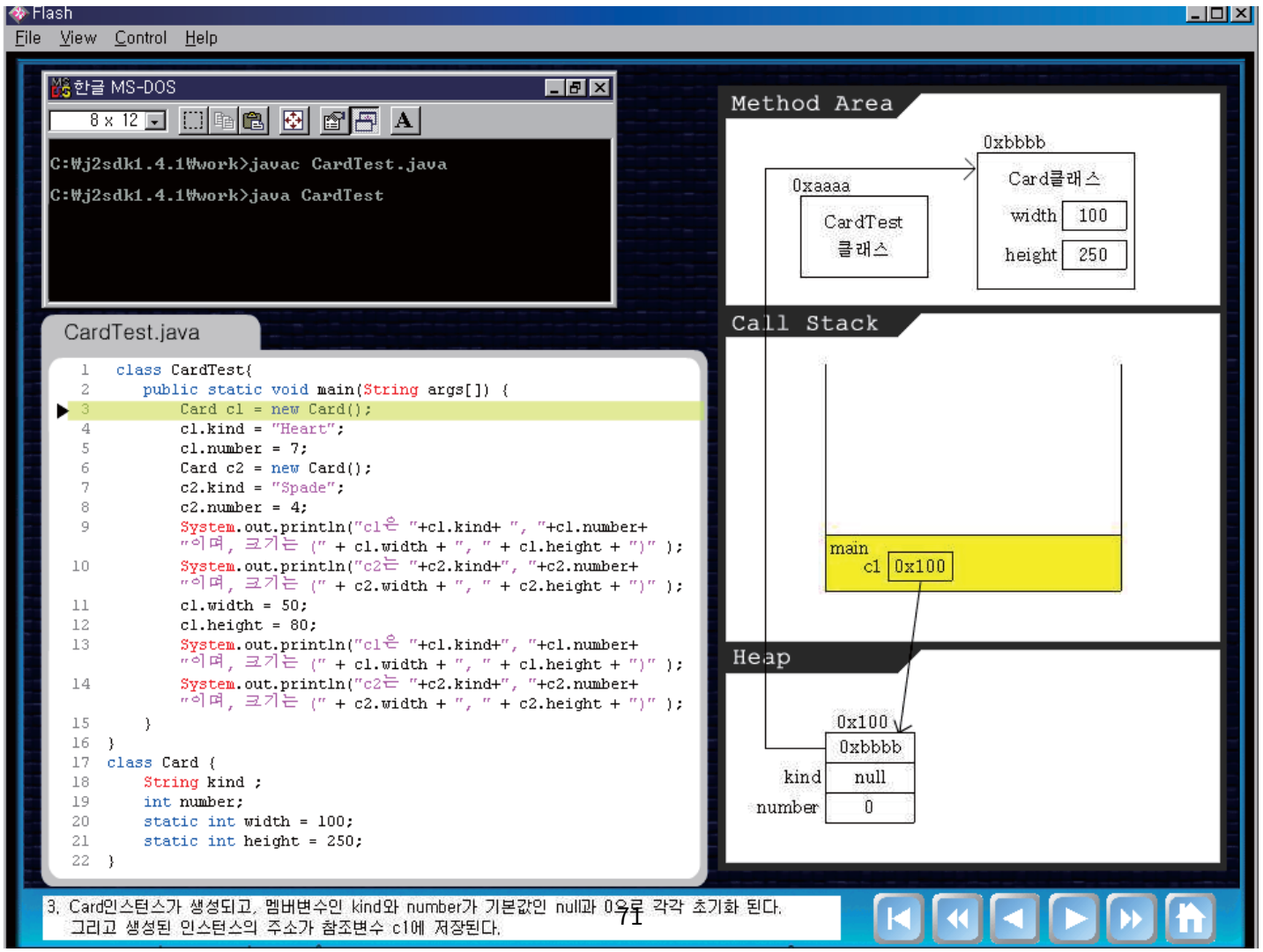
정적영역에 저장되어 있다가 호출되면 (클래스메서드든 인스턴스메서드든 메서드의 종류에 관계없이) **스택영역(Call Stack)**에 올라감

힙영역(Heap)은 프로그램을 실행할 때 동적으로 확보하기 위해 미리 확보해둔 메모리 영역. 인스턴스 변수와 배열만 올라감




```
1  class CardTest{
2     public static void main(String args[]) {
3         Card c1 = new Card();
4         c1.kind = "Heart";
5         c1.number = 7;
6         Card c2 = new Card();
7         c2.kind = "Spade";
8         c2.number = 4;
9         System.out.println("c1은 "+c1.kind+ ", "+c1.number+
          "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
10        System.out.println("c2는 "+c2.kind+", "+c2.number+
          "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
11        c1.width = 50;
12        c1.height = 80;
13        System.out.println("c1은 "+c1.kind+", "+c1.number+
          "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
14        System.out.println("c2는 "+c2.kind+", "+c2.number+
          "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
15    }
16 }
17 class Card {
18    String kind ;
19    int number;
20    static int width = 100;
21    static int height = 250;
22 }
```


Flash
File View Control Help
한글 MS-DOS
8 x 12
C:₩j2sdk1.4.1₩work>javac CardTest.java
C:₩j2sdk1.4.1₩work>java CardTest
CardTest.java
1 class CardTest{
2 public static void main(String args[]) {
3 Card c1 = new Card();
4 c1.kind = "Heart";
5 c1.number = 7;
6 Card c2 = new Card();
7 c2.kind = "Spade";
8 c2.number = 4;
9 System.out.println("c1은 "+c1.kind+ ", "+c1.number+
"이며, 크기는 (" + c1.width + ", " + c1.height + ")");
10 System.out.println("c2는 "+c2.kind+", "+c2.number+
"이며, 크기는 (" + c2.width + ", " + c2.height + ")");
11 c1.width = 50;
12 c1.height = 80;
13 System.out.println("c1은 "+c1.kind+", "+c1.number+
"이며, 크기는 (" + c1.width + ", " + c1.height + ")");
14 System.out.println("c2는 "+c2.kind+", "+c2.number+
"이며, 크기는 (" + c2.width + ", " + c2.height + ")");
15 }
16 }
17 class Card {
18 String kind ;
19 int number;
20 static int width = 100;
21 static int height = 250;
22 }
Method Area
0xaaaa
CardTest
클래스
0xbbbb
Card클래스
width 100
height 250
Call Stack
main
Heap
2. Card클래스의 인스턴스를 생성하기 위해 먼저 Card클래스가 메모리에 로드된다.
이 때, Card클래스의 클래스변수인 width와 height가 메모리에 생성되고 각각 100, 250으로 초기화 된다.


Flash
File View Control Help
한글 MS-DOS
8 x 12
C:₩j2sdk1.4.1₩work>javac CardTest.java
C:₩j2sdk1.4.1₩work>java CardTest
CardTest.java
1 class CardTest{
2 public static void main(String args[]) {
3 Card c1 = new Card();
4 c1.kind = "Heart";
5 c1.number = 7;
6 Card c2 = new Card();
7 c2.kind = "Spade";
8 c2.number = 4;
9 System.out.println("c1은 "+c1.kind+ ", "+c1.number+
"이며, 크기는 (" + c1.width + ", " + c1.height + ")");
10 System.out.println("c2는 "+c2.kind+", "+c2.number+
"이며, 크기는 (" + c2.width + ", " + c2.height + ")");
11 c1.width = 50;
12 c1.height = 80;
13 System.out.println("c1은 "+c1.kind+", "+c1.number+
"이며, 크기는 (" + c1.width + ", " + c1.height + ")");
14 System.out.println("c2는 "+c2.kind+", "+c2.number+
"이며, 크기는 (" + c2.width + ", " + c2.height + ")");
15 }
16 }
17 class Card {
18 String kind ;
19 int number;
20 static int width = 100;
21 static int height = 250;
22 }
Method Area
0xaaaa
CardTest
클래스
0xbbbb
Card클래스
width 100
height 250
Call Stack
main
c1 0x100
Heap
0x100
0xbbbb
kind null
number 0
3. Card인스턴스가 생성되고, 멤버변수인 kind와 number가 기본값인 null과 0으로 각각 초기화 된다.
그리고 생성된 인스턴스의 주소가 참조변수 c1에 저장된다.

Flash

File View Control Help

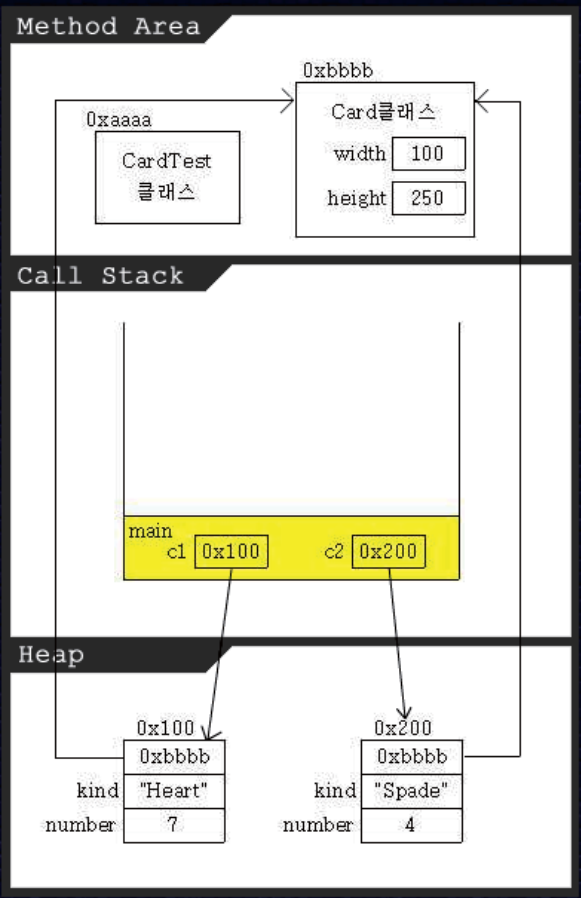
한글 MS-DOS

8 x 12

```
C:₩j2sdk1.4.1₩work>javac CardTest.java

C:₩j2sdk1.4.1₩work>java CardTest
```

CardTest.java

```
1  class CardTest{
2     public static void main(String args[]) {
3         Card c1 = new Card();
4         c1.kind = "Heart";
5         c1.number = 7;
6         Card c2 = new Card();
7         c2.kind = "Spade";
8         c2.number = 4;
9         System.out.println("c1은 "+c1.kind+ ", "+c1.number+
          "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
10        System.out.println("c2는 "+c2.kind+", "+c2.number+
          "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
11        c1.width = 50;
12        c1.height = 80;
13        System.out.println("c1은 "+c1.kind+", "+c1.number+
          "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
14        System.out.println("c2는 "+c2.kind+", "+c2.number+
          "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
15    }
16 }
17 class Card {
18    String kind ;
19    int number;
20    static int width = 100;
21    static int height = 250;
22 }
```



6. c2.kind를 "Spade"로, 그리고 c2.number를 4로 변경한다.

Flash

File View Control Help

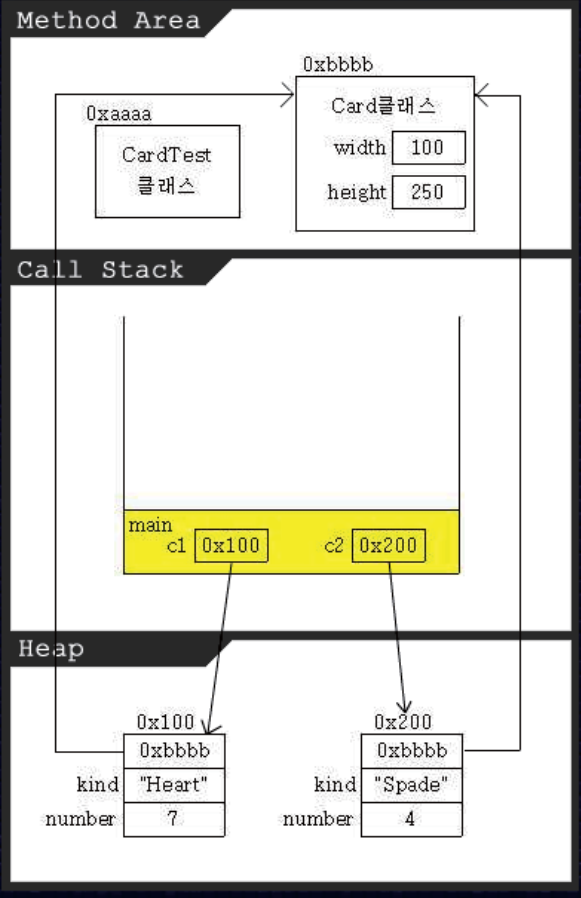
한글 MS-DOS

8 x 12

```
C:₩j2sdk1.4.1₩work>javac CardTest.java

C:₩j2sdk1.4.1₩work>java CardTest
c1은 Heart, 7이며, 크기는 (100, 250)
c2는 Spade, 4이며, 크기는 (100, 250)
```

CardTest.java

```
1  class CardTest{
2     public static void main(String args[]) {
3         Card c1 = new Card();
4         c1.kind = "Heart";
5         c1.number = 7;
6         Card c2 = new Card();
7         c2.kind = "Spade";
8         c2.number = 4;
9         System.out.println("c1은 "+c1.kind+ ", "+c1.number+
          "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
10        System.out.println("c2는 "+c2.kind+", "+c2.number+
          "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
11        c1.width = 50;
12        c1.height = 80;
13        System.out.println("c1은 "+c1.kind+", "+c1.number+
          "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
14        System.out.println("c2는 "+c2.kind+", "+c2.number+
          "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
15    }
16 }
17 class Card {
18    String kind ;
19    int number;
20    static int width = 100;
21    static int height = 250;
22 }
```



7. 인스턴스 c1과 c2의 값을 화면에 출력한다. 모든 인스턴스는 자신을 생성한 클래스의 주소를 갖고 있으므로, 참조변수를 사용해서도 클래스변수에 접근할 수 있다.

Flash

File View Control Help

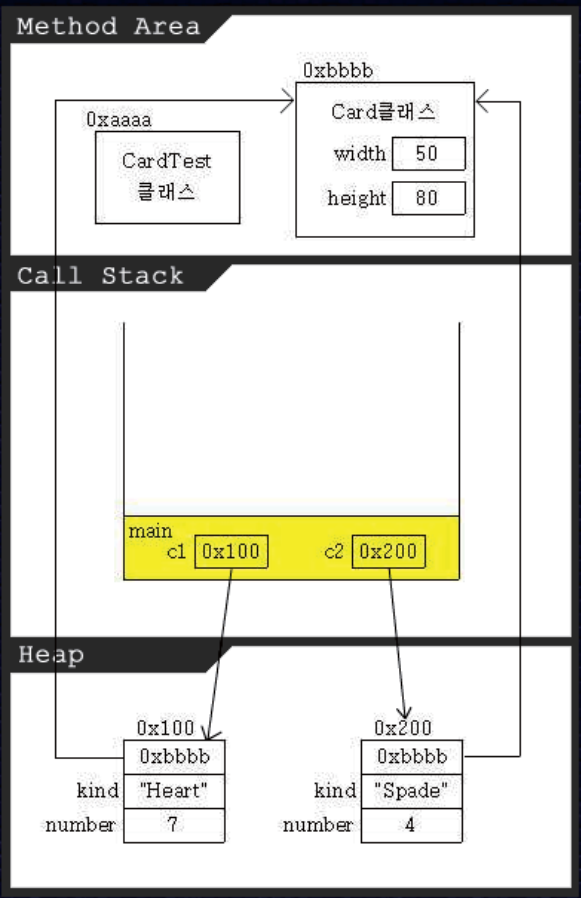
한글 MS-DOS

8 x 12

```
C:₩j2sdk1.4.1₩work>javac CardTest.java

C:₩j2sdk1.4.1₩work>java CardTest
c1은 Heart, 7이며, 크기는 (100, 250)
c2는 Spade, 4이며, 크기는 (100, 250)
```

CardTest.java

```
1   class CardTest{
2      public static void main(String args[]) {
3          Card c1 = new Card();
4          c1.kind = "Heart";
5          c1.number = 7;
6          Card c2 = new Card();
7          c2.kind = "Spade";
8          c2.number = 4;
9          System.out.println("c1은 "+c1.kind+ ", "+c1.number+
           "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
10         System.out.println("c2는 "+c2.kind+", "+c2.number+
           "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
11         c1.width = 50;
12         c1.height = 80;
13         System.out.println("c1은 "+c1.kind+", "+c1.number+
           "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
14         System.out.println("c2는 "+c2.kind+", "+c2.number+
           "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
15     }
16  }
17  class Card {
18      String kind ;
19      int number;
20      static int width = 100;
21      static int height = 250;
22  }
```



8. 클래스변수 c1.width와 c1.height의 값을 각각 50, 80으로 변경한다.

Flash

File View Control Help

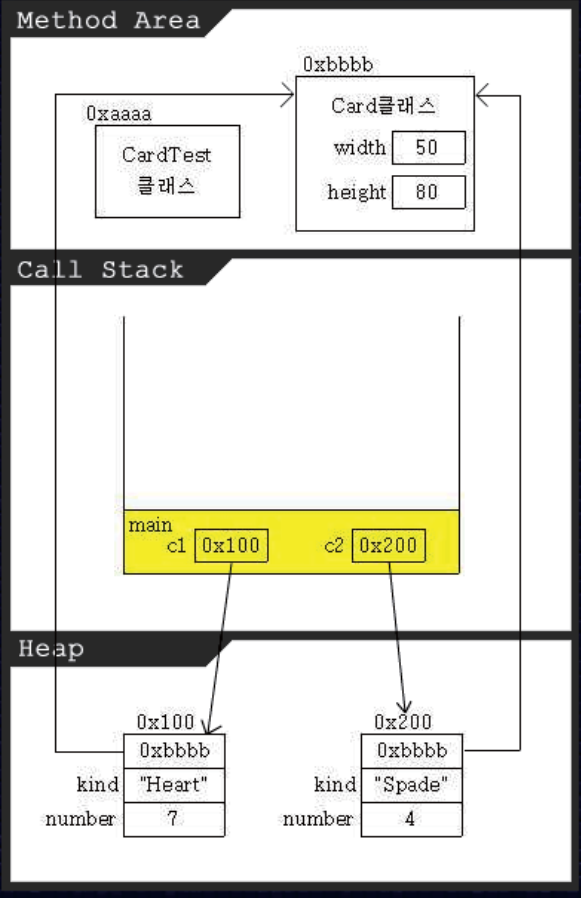
한글 MS-DOS

8 x 12

```
C:₩j2sdk1.4.1₩work>javac CardTest.java

C:₩j2sdk1.4.1₩work>java CardTest
c1은 Heart, 7이며, 크기는 (100, 250)
c2는 Spade, 4이며, 크기는 (100, 250)
c1은 Heart, 7이며, 크기는 (50, 80)
c2는 Spade, 4이며, 크기는 (50, 80)
```

CardTest.java

```
1   class CardTest{
2      public static void main(String args[]) {
3          Card c1 = new Card();
4          c1.kind = "Heart";
5          c1.number = 7;
6          Card c2 = new Card();
7          c2.kind = "Spade";
8          c2.number = 4;
9          System.out.println("c1은 "+c1.kind+ ", "+c1.number+
           "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
10         System.out.println("c2는 "+c2.kind+", "+c2.number+
           "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
11         c1.width = 50;
12         c1.height = 80;
13         System.out.println("c1은 "+c1.kind+", "+c1.number+
           "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
14         System.out.println("c2는 "+c2.kind+", "+c2.number+
           "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
15     }
16  }
17  class Card {
18      String kind ;
19      int number;
20      static int width = 100;
21      static int height = 250;
22  }
```



9. 인스턴스 c1과 c2의 값을 화면에 출력한다. 인스턴스 c1과 c2는 클래스변수 width와 height를 공유하므로 같은 값이 출력된다.

예제

```
class Human {
  int id;
  int income;
  static int total=0; // 클래스 변수
  Human() { // Human 객체가 생성될 때마다
    total++;  // total이 1씩 증가한다.
//이와 같이 ID, serial number 등을 부여할 때도 사용
  }
}
class HumanTest {
  public static void main(String[] args) {
    Human m = new Human();
    Human n = new Human();
    Human p = new Human();
    System.out.println("The total man is " + Human.total);
  }
}
```

실행결과

```
The total man is 3
```

3.3 메서드(method)

▶ 메서드란?

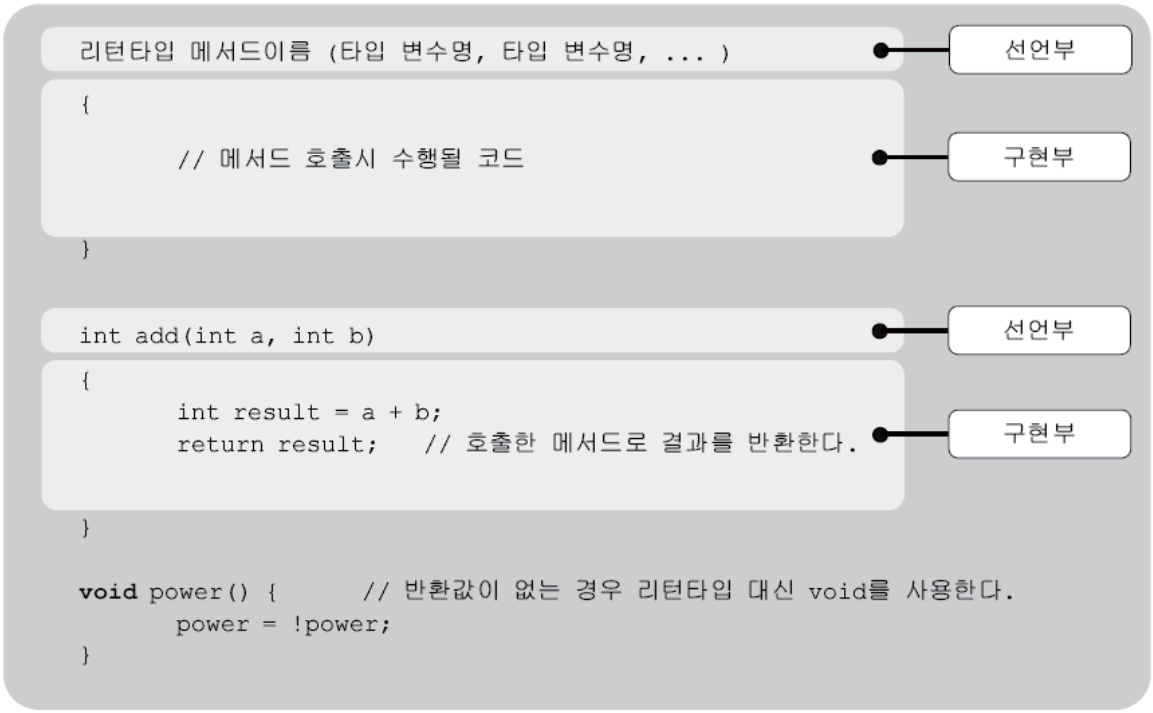
- 작업을 수행하기 위한 명령문의 집합
- 어떤 값을 입력받아서 처리하고 그 결과를 돌려준다.

(입력받는 값이 없을 수도 있고 결과를 돌려주지 않을 수도 있다.)

▶ 메서드의 장점과 작성지침

- 반복적인 코드를 줄이고 코드의 관리가 용이하다.
- 반복적으로 수행되는 여러 문장을 메서드로 작성한다.
- 하나의 메서드는 한 가지 기능만 수행하도록 작성하는 것이 좋다.
- 관련된 여러 문장을 메서드로 작성한다.

▶ 메서드를 정의하는 방법 – 클래스 영역에만 정의할 수 있음



3.4 return문

▶ 메서드가 정상적으로 종료되는 경우
- 메서드의 블럭{}의 끝에 도달했을 때
- 메서드의 블럭{}을 수행 도중 return문을 만났을 때

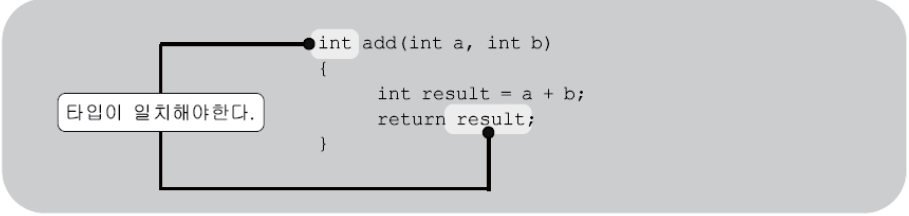
▶ return문
- 현재 실행 중인 메서드를 종료하고 호출한 메서드로 되돌아간다.

1. **반환값이 없는 경우** - return문만 써주면 된다.

```
return;
```

2. **반환값이 있는 경우** - return문 뒤에 반환값을 지정해 주어야 한다.

```
return 반환값;
```



▶ return문 - 주의사항

- 반환값이 있는 메서드는 모든 경우에 return문이 있어야 한다.

```
int max(int a, int b) {
     if(a > b)
          return a;
}
```

거짓일 경우 컴파일시 에러발생

```
int max(int a, int b) {
     if(a > b)
          return a;
     else
          return b;
}
```

- return문의 개수는 최소화하는 것이 좋다.

```
int max(int a, int b) {
     if(a > b)
          return a;
     else
          return b;
}
```

컴파일시 에러발생은 없으나…

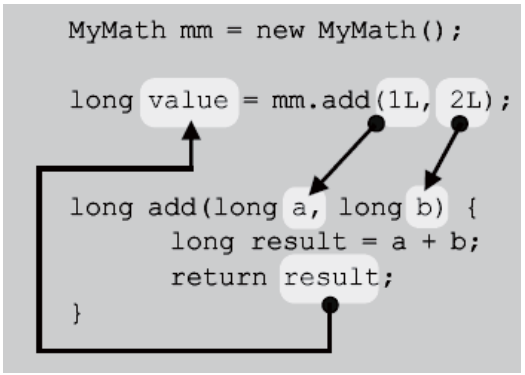
```
int max(int a, int b) {
     int result = 0;
     if(a > b)
          result = a;
     else
          result = b;
     return result;
}
```

3.5 메서드의 호출

▶ 메서드의 호출방법

```
참조변수.메서드 이름();                  // 메서드에 선언된 매개변수가 없는 경우
참조변수.메서드 이름(값1, 값2, ... );    // 메서드에 선언된 매개변수가 있는 경우
```

```
class MyMath {
     long add(long a, long b) {
          long result = a + b;
          return result;
//        return a + b;
     }
  ...
}
```



예제 6-6

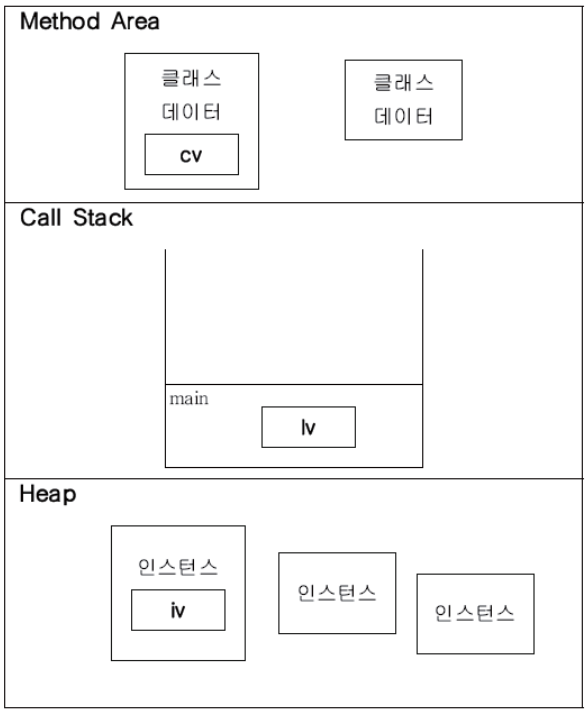
```
class MyMathTest {
        public static void main(String args[]) {
                MyMath mm = new MyMath();
                long result1 = mm.add(5L, 3L);
                long result2 = mm.subtract(5L, 3L);
                long result3 = mm.multiply(5L, 3L);
                double result4 = mm.divide(5L, 3L);
                System.out.println("add(5L, 3L) = " + result1);
                System.out.println("subtract(5L, 3L) = " + result2);
                System.out.println("multiply(5L, 3L) = " + result3);
                System.out.println("divide(5L, 3L) = " + result4);
        }
}

class MyMath {
        long add(long a, long b) {
                long result = a+b;
                return result;
        //      return a + b;       // 위의 두 줄을 이와 같이 한 줄로 간단히 할 수 있다.
        }

        long subtract(long a, long b) {
                return a - b;
        }

        long multiply(long a, long b) {
                return a * b;
        }

        double divide(double a, double b) {
                return a / b;
        }
}
```

double대신, long값으로 호출하였다. 이 값은 double로 자동 형변환된다.

실행결과

```
add(5L, 3L) = 8
subtract(5L, 3L) = 2
multiply(5L, 3L) = 15
divide(5L, 3L) = 1.6666666666666667
```

3.6 JVM의 메모리 구조



▶ 메서드영역(Method Area)

- 클래스 정보와 클래스변수가 저장되는 곳

▶ 호출스택(Call Stack)

- 메서드의 작업공간. 메서드가 호출되면 메서드 수행에 필요한 메모리공간을 할당받고 메서드가 종료되면 사용하던 메모리를 반환한다.

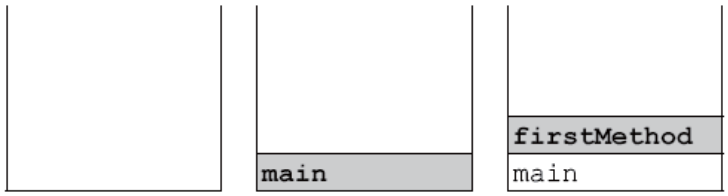
▶ 힙(Heap)

- 인스턴스가 생성되는 공간. new연산자에 의해서 생성되는 배열과 객체는 모두 여기에 생성된다.

▶ 호출스택

- 호출스택의 특징

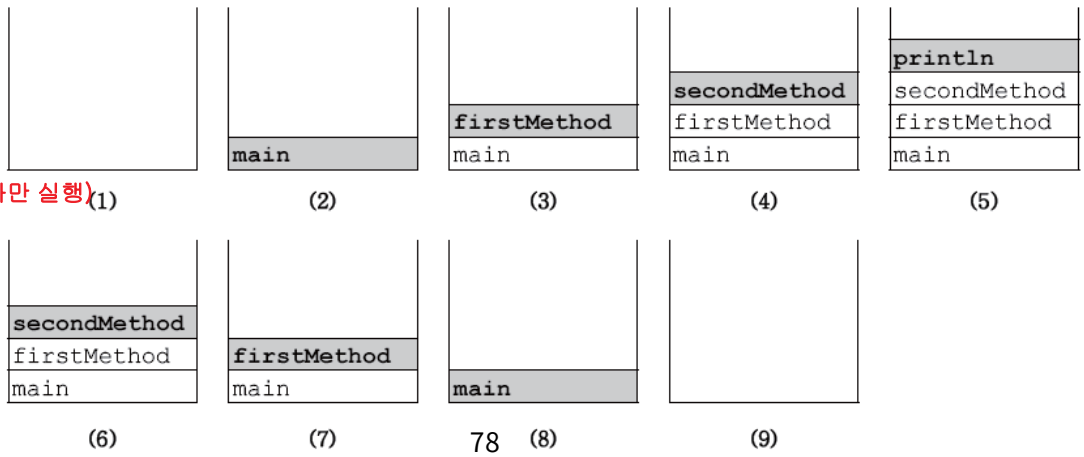
- 메서드가 호출되면 수행에 필요한 메모리를 스택에 할당 받는다.
- 메서드가 수행을 마치면 사용했던 메모리를 반환한다.
- 호출스택의 제일 위에 있는 메서드가 현재 실행중인 메서드다.
- 아래에 있는 메서드가 바로 위의 메서드를 호출한 메서드다.



- 호출스택의 변화

```
class CallStackTest {
	public static void main(String[] args) {
		firstMethod();
	}
	static void firstMethod() {
		secondMethod();
	}
	static void secondMethod() {
		System.out.println("secondMethod()");
	}
}
```

실행 중인 메서드
(회색, 맨 위의 하나만 실행)



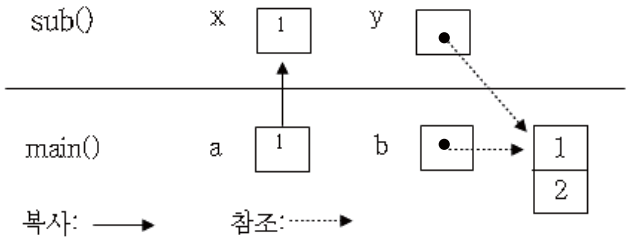
3.7 기본형 매개변수와 참조형 매개변수

- 기본형 매개변수 – Call by value
- 참조형 매개변수 – Call by reference

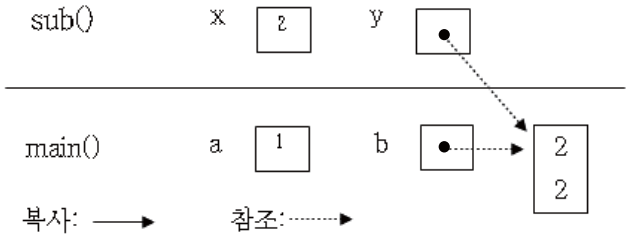
예제

```
class PassingTest {
  static void sub(int x, int[] y)
  // x, y는 형식인자
  {
    x++;
    y[0]++;
  }

  public static void main(String[] arg)
  {
    int a=1, b[]={1,2};
    System.out.println("a="+a+" b="+b[0]);
    sub(a, b); // a, b는 실인자
    System.out.println("a="+a+" b="+b[0]);
  }
}
```

① 메소드 **sub()** 호출시



② 메소드 **sub()**에서 **return**시



실행결과

```
a=1 b=1
a=1 b=2
```

3.8 재귀호출(recursive call)

▶ 재귀호출이란?

- 메서드 내에서 자기자신을 반복적으로 호출하는 것
- 재귀호출은 반복문으로 바꿀 수 있으며 반복문보다 성능이 나쁨
- 이해하기 쉽고 간결한 코드를 작성할 수 있다

▶ 재귀호출의 예(例)

- 팩토리얼, 제곱, 트리운행, 폴더목록표시 등

*팩토리얼(factorial)

5! = 5 * 4 * 3 * 2 * 1

f(n) = n * f(n-1) 단, f(1) = 1

```
long factorial(int n) {
    long result = 0;
    if(n==1) {
        result = 1;
    } else {
        result = n * factorial(n-1);
    }
    return result;
}
```

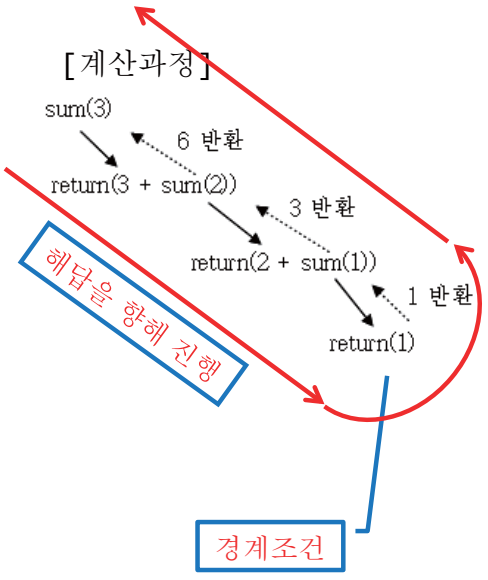
▶ 재귀함수(recursive method)의 구현 방법

① 재귀호출이 종료될 수 있도록 경계조건(boundary condition)을 설정한다.

② 매 호출마다 해답을 향해 한 단계씩 가까워지게끔 구현한다.

예제

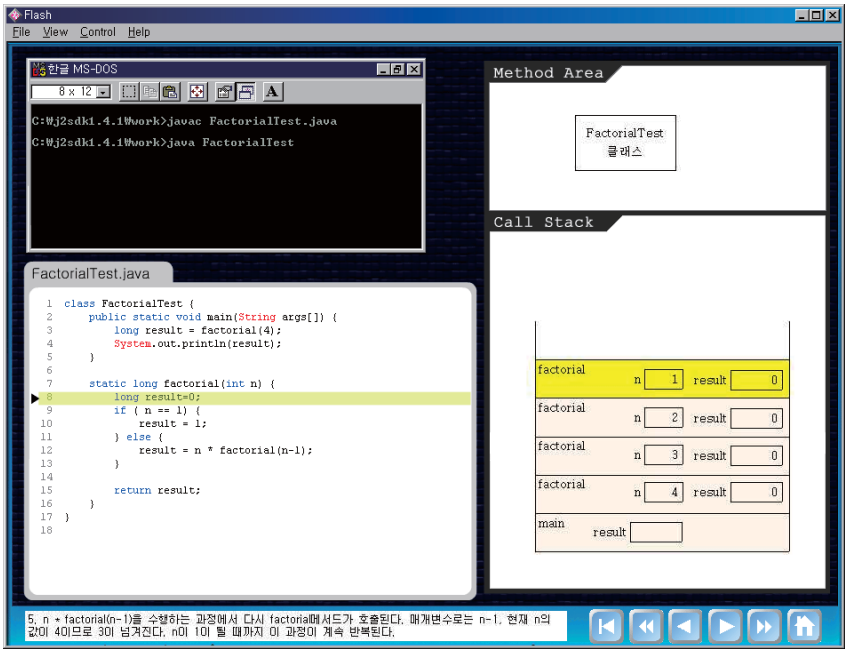
```
class Recursion {
  static int sum(int n) {
     if(n>1) return(n + sum(n-1)); // 해답을 향해 진행
     else return 1; // 경계조건
  }
  public static void main(String a[])
  {
     System.out.println("The sum(10)="+sum(10));
     System.out.println("The sum(50)="+sum(50));
  }
}
```



실행결과

```
The sum(10)=55
The sum(50)=127
```

3.8 재귀호출(recursive call)



3.9 클래스메서드(static메서드)와 인스턴스메서드

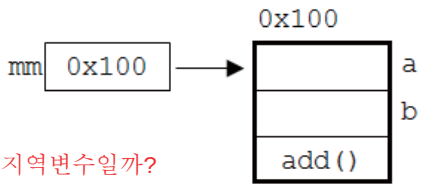
▶ 인스턴스메서드

- 인스턴스 생성 후, '참조변수.메서드이름()'으로 호출
- 인스턴스변수나 인스턴스메서드와 관련된 작업을 하는 메서드
- 메서드 내에서 인스턴스변수 접근 가능

▶ 클래스메서드(static메서드)

- 객체생성없이 '클래스이름.메서드이름()'으로 호출
- 인스턴스변수나 인스턴스메서드와 관련없는 작업을 하는 메서드
- 메서드 내에서 인스턴스변수 접근 불가
- 즉, 클래스변수만 접근 가능

```
class MyMath2 {
	long a, b;

	long add() {	// 인스턴스메서드
		return a + b;	// 변수 a와 b는 인스턴스변수일까 지역변수일까?
	}

	static long add(long a, long b) { // 클래스메서드(static메서드)
		return a + b;	// 변수 a와 b는 인스턴스변수일까 지역변수일까?
	}
}
```




```
class MyMathTest2 {
	public static void main(String args[]) {
		System.out.println(MyMath2.add(200L,100L); // 클래스메서드 호출
		MyMath2 mm = new MyMath2(); // 인스턴스 생성
		mm.a = 200L;
		mm.b = 100L;
		System.out.println(mm.add()); // 인스턴스메서드 호출
	}
}
```

예제 6-15

```
class MyMath2 {
	long a, b;

	// 인스턴스변수 a, b만을 이용해서 작업하므로 매개변수가 필요없다.
	long add() {	return a + b; } // a, b는 인스턴스변수
	long subtract() {	return a - b; }
	long multiply() {	return a * b; }
	double divide() {	return a / b; }

	// 인스턴스변수와 관계없이 매개변수만으로 작업이 가능하다.
	static long add(long a, long b) {	return a + b; }	// a, b는 지역변수
	static long subtract(long a, long b) {	return a - b; }
	static long multiply(long a, long b) {	return a * b; }
	static double divide(double a, double b) {	return a / b; }
}

class MyMathTest2 {
	public static void main(String args[]) {
		// 클래스메서드 호출
		System.out.println(MyMath2.add(200L, 100L));
		System.out.println(MyMath2.subtract(200L, 100L));
		System.out.println(MyMath2.multiply(200L, 100L));
		System.out.println(MyMath2.divide(200.0, 100.0));

		MyMath2 mm = new MyMath2();
		mm.a = 200L;
		mm.b = 100L;
		// 인스턴스메서드는 객체생성 후에만 호출이 가능함.
		System.out.println(mm.add());
		System.out.println(mm.subtract());
		System.out.println(mm.multiply());
		System.out.println(mm.divide());
	}
}
```

실행결과

```
300
100
20000
2.0
3000
100
20000
2.0
```

예제

```
class Count {
  static int total = 10; // static variable
  int cnt = 6;
  public static void st_inc() { // 클래스메서드
                total ++;       // 변수 total는 인스턴스변수일까 클래스변수일까?
  }
  public void inc() {  // 인스턴스메서드
                cnt ++;
  }
  public void out() { // print two variables
                System.out.println("total="+total+"\tcnt="+cnt);
  }
}
public class CountTest {
  public static void main(String[] args) {
                Count m = new Count();
                m.out();          // print 메서드 호출
                m.inc();
                m.st_inc();       // 클래스메서드 호출
                Count.st_inc();
                m.out();          // print 메서드 호출
  }
}
```

실행결과

```
total=10        cnt=6
total=12        cnt=7
```

3.10 멤버간의 참조와 호출

▶ 메서드의 호출

"같은 클래스의 멤버간에는 객체생성이나 참조변수 없이 참조할 수 있다. 그러나 클래스멤버들은 인스턴스멤버들을 참조할 수 없다."

```
class TestClass {
   void instanceMothod() {}      // 인스턴스메서드
   static void staticMethod() {} // static메서드

   void instanceMothod2() {      // 인스턴스메서드
       instanceMethod();         // 다른 인스턴스메서드를 호출한다.
       staticMethod();           // static메서드를 호출한다.
   }

   static void staticMethod2() { // static메서드
       instanceMethod();         // 에러!!! 인스턴스메서드를 호출할 수 없다.
       staticMethod();           // static메서드는 호출 할 수 있다.
   }
} // end of class
```

▶ 변수의 접근

"같은 클래스의 멤버간에는 객체생성이나 참조변수 없이 참조할 수 있다. 그러나 클래스멤버들은 인스턴스멤버들을 참조할 수 없다."

```
class TestClass2 {
   int iv;          // 인스턴스변수
   static int cv;   // 클래스변수

   void instanceMothod() {       // 인스턴스메서드
       System.out.println(iv);   // 인스턴스변수를 사용할 수 있다.
       System.out.println(cv);   // 클래스변수를 사용할 수 있다.
   }

   static void staticMethod() {  // static메서드
       System.out.println(iv);   // 에러!!! 인스턴스변수를 사용할 수 없다.
       System.out.println(cv);   // 클래스변수를 사용할 수 있다.
   }
} // end of class
```

4. 메서드 오버로딩

4.1 메서드 오버로딩(method overloading)이란?

"하나의(동일한) 클래스에 같은 이름의 메서드를 여러 개 정의하는 것을 메서드 오버로딩, 간단히 오버로딩이라고 한다."

* overload - vt. 과적하다. 부담을 많이 지우다.

4.2 오버로딩의 조건

- 메서드의 이름이 같아야 한다.
- 매개변수의 개수 또는 타입이 달라야 한다.
- 매개변수는 같고 리턴타입이 다른 경우는 오버로딩이 성립되지 않는다.

(리턴타입은 오버로딩을 구현하는데 아무런 영향을 주지 못한다.)

4.3 오버로딩의 예

▸ System.out.println메서드

- 다양하게 오버로딩된 메서드를 제공함으로써 모든 변수를 출력할 수 있도록 설계

```
void println()
void println(boolean x)
void println(char x)
void println(char[] x)
void println(double x)
void println(float x)
void println(int x)
void println(long x)
void println(Object x)
void println(String x)
```

▸ 매개변수의 이름이 다른 것은 오버로딩이 아니다.

```
[보기1]
int add(int a, int b) { return a+b; }
int add(int x, int y) { return x+y; }
```

▸ 리턴타입은 오버로딩의 성립조건이 아니다.

```
[보기2]
int add(int a, int b) { return a+b; }
long add(int a, int b) { return (long)(a + b); }
```

▸ 매개변수의 타입이 다르므로 오버로딩이 성립한다.

```
[보기3]
long add(int a, long b) { return a+b; }
long add(long a, int b) { return a+b; }
```

▸ 오버로딩의 올바른 예 – 매개변수는 다르지만 같은 의미의 기능수행

```
[보기4]
int add(int a, int b) { return a+b; }
long add(long a, long b) { return a+b; }
int add(int[] a) {
     int result =0;

     for(int i=0; i < a.length; i++) {
          result += a[i];
    }
     return result;
}
```

예제 6-17

```
class OverloadingTest {
	public static void main(String args[]) {
		MyMath3 mm = new MyMath3();
		System.out.println("mm.add(3, 3) 결과:" + mm.add(3,3));
		System.out.println("mm.add(3L, 3) 결과: " + mm.add(3L,3));
		System.out.println("mm.add(3, 3L) 결과: " + mm.add(3,3L));
		System.out.println("mm.add(3L, 3L) 결과: " + mm.add(3L,3L));

		int[] a = {100, 200, 300};
		System.out.println("mm.add(a) 결과: " + mm.add(a));
   }
}

class MyMath3 {
	int add(int a, int b) {
		System.out.print("int a
		return a+b;
	}

	long add(int a, long b) {
		System.out.print("long add(int a, long b) - ");
		return a+b;
	}

	long add(long a, int b) {
		System.out.print("long add(long a, int b) - ");
		return a+b;
	}

	long add(long a, long b) {
		System.out.print("long add(long a, long b) - ");
		return a+b;
	}

	int add(int[] a) {		// 배열의 모든 요소의 합을 결과로 돌려준다.
		System.out.print("int add(int[] a) - ");
		int result = 0;
		for(int i=0; i < a.length;i++) {
			result += a[i];
		}
		return result;
	}
}
```

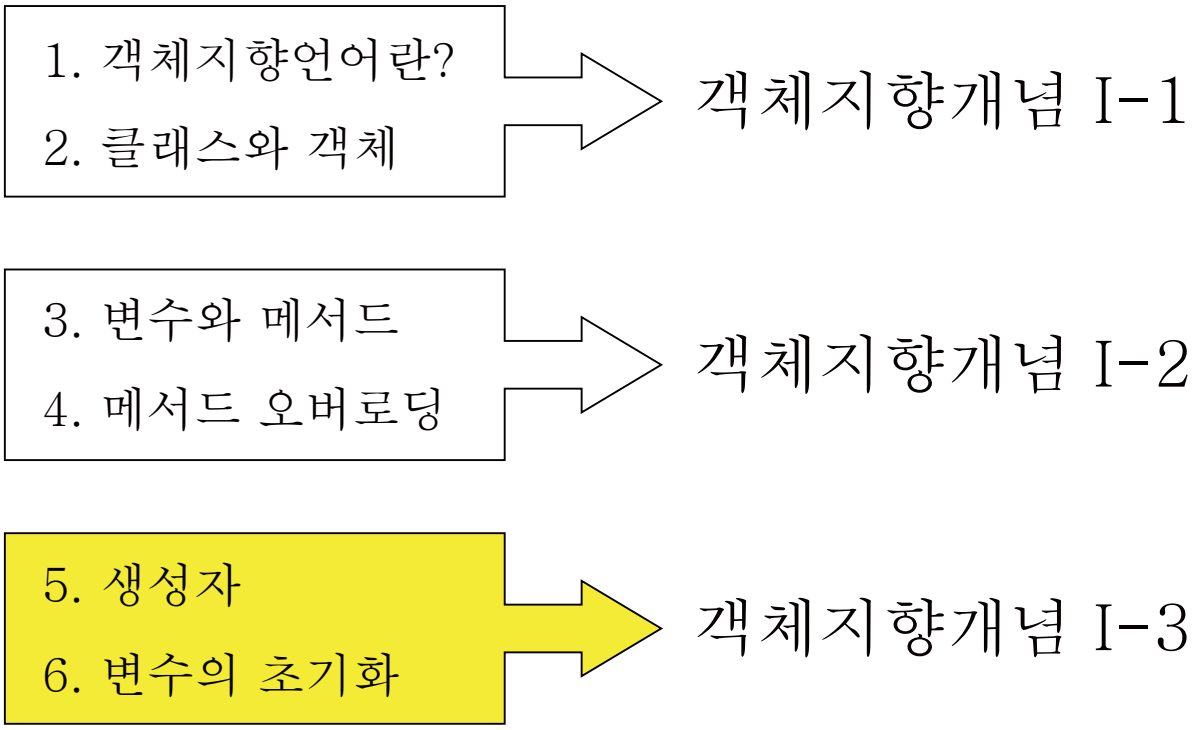
실행결과

```
int add(int a, int b) - mm.add(3, 3) 결과: 6
long add(long a, int b) - mm.add(3L, 3) 결과: 6
long add(int a, long b) - mm.add(3, 3L) 결과: 6
long add(long a, long b) - mm.add(3L, 3L) 결과: 6
int add(int[] a) - mm.add(a) 결과: 600
```

제 6 장
객체지향개념 I-3

인제대학교 전자IT기계자동차공학부

황 원 주



5. 생성자

5.1 생성자(constructor)란?

▸ 생성자란?

- 인스턴스가 생성될 때마다 호출되는 '**인스턴스 초기화 메서드**'
- 인스턴스 변수의 초기화 또는 인스턴스 생성시 수행할 작업에 사용
- 몇가지 조건을 제외하고는 메서드와 같다.
- 모든 클래스에는 반드시 하나 이상의 생성자가 있어야 한다.

* 인스턴스 초기화 – 인스턴스 변수에 적절한 값을 저장하는 것.

```
Card c = new Card();
```

1. 연산자 new에 의해서 메모리(heap)에 Card클래스의 인스턴스가 생성된다.
2. 생성자 Card()가 호출되어 수행된다.
3. 연산자 new의 결과로, 생성된 Card인스턴스의 주소가 반환되어 참조변수 c에 저장된다.

5.2 생성자의 조건

▸ 생성자의 조건

- 생성자의 이름은 클래스의 이름과 같아야 한다.
- 생성자는 리턴값이 없다. (하지만 void를 쓰지 않는다.)

```
클래스이름(타입 변수명, 타입 변수명, ... ) {
    // 인스턴스 생성시 수행될 코드
    // 주로 인스턴스 변수의 초기화 코드를 적는다.
}
```

```
class Card {
    ...
    Card() {  // 매개변수가 없는 생성자.
        // 인스턴스 초기화 작업
    }

    Card(String kind, int number) { // 매개변수가 있는 생성자
        // 인스턴스 초기화 작업
    }
}
```

5.3 기본 생성자(default constructor)

▸ 기본 생성자란?

- 매개변수가 없는 생성자
- 클래스에 생성자가 하나도 없으면 컴파일러가 기본 생성자를 추가한다.
  (생성자가 하나라도 있으면 컴파일러는 기본 생성자를 추가하지 않는다.)

```
클래스이름() { }
Card() { } // 컴파일러에 의해 추가된 Card클래스의 기본 생성자. 내용이 없다.
```

"모든 클래스에는 반드시
하나 이상의 생성자가 있어야 한다."

"모든 클래스에는 반드시 하나 이상의 생성자가 있어야 한다."

[예제6-18]/ch6/ConstructorTest.java

```
class Data1 {
   int value;
}

class Data2 {
   int value;
   Data2(int x) {    // 매개변수가 있는 생성자.
        value = x;
   }
}

class ConstructorTest {
   public static void main(String[] args) {
        Data1 d1 = new Data1();
        Data2 d2 = new Data2();        // compile error발생
   }
}
```

```
class Data1 {
    int value;
    Data1() {} // 기본생성자
}
```

5.4 매개변수가 있는 생성자

```
class Car {
      String color;          // 색상
      String gearType;       // 변속기 종류 - auto(자동), manual(수동)
      int door;              // 문의 개수

      Car() {}  // 생성자
      Car(String c, String g, int d) { // 생성자
            color = c;
            gearType = g;
            door = d;
      }
}
```

```
Car c = new Car();
c.color = "white";
c.gearType = "auto";
c.door = 4;
```

→

```
Car c = new Car("white","auto",4);
```

예제 6-19

```
class Car {
        String color;       // 색상
        String gearType;    // 변속기 종류 - auto(자동), manual(수동)
        int door;           // 문의 개수

        Car() {}
        Car(String c, String g, int d) {
                color = c;
                gearType = g;
                door = d;
        }
}

class CarTest {
        public static void main(String[] args) {
                Car c1 = new Car();
                c1.color = "white";
                c1.gearType = "auto";
                c1.door = 4;

                Car c2 = new Car("white", "auto", 4);
                System.out.println("c1의 color=" + c1.color + ", gearType="
                     + c1.gearType+ ", door="+c1.door);
                System.out.println("c2의 color=" + c2.color + ", gearType="
                     + c2.gearType+ ", door="+c2.door);
        }
}
```

실행결과

```
c1의 color=white, gearType=auto, door=4
c2의 color=white, gearType=auto, door=4
```

5.5 생성자에서 다른 생성자 호출하기 – this()

▶ this() – 생성자, 같은 클래스의 다른 생성자를 호출할 때 사용

다른 생성자 호출은 생성자의 첫 문장에서만 가능

```
1  class Car {
2      String color;
3      String gearType;
4      int door;
5
6      Car() {
7          color = "white";
8          gearType = "auto";
9          door = 4;
10     }
11
12     Car(String c, String g, int d) {
13         color = c;
14         gearType = g;
15         door = d;
16     }
17
18 }
19
```

* 코드의 재사용성을 높인 코드

```
Car() {
    //Card("white","auto",4);
    this("white","auto",4);
}
```

```
Car() {
    door = 5;
    this("white","auto",4);
}
```

예제 6-20

```
class Car {
        String color;                   // 색상
        String gearType;                // 변속기 종류 - auto(자동), manual(수동)
        int door;                       // 문의 개수

        Car() {
                this("white", "auto", 4);
        }

        Car(String color) {
                this(color, "auto", 4);
        }

        Car(String color, String gearType, int door) {
                this.color = color;
                this.gearType = gearType;
                this.door = door;
        }
}

class CarTest2 {
        public static void main(String[] args) {
                Car c1 = new Car();
                Car c2 = new Car("blue");

                System.out.println("c1의 color=" + c1.color + ", gearType="
                                + c1.gearType+ ", door="+c1.door);
                System.out.println("c2의 color=" + c2.color + ", gearType="
                                + c2.gearType+ ", door="+c2.door);
        }
}
```

실행결과

```
c1의 color=white, gearType=auto, door=4
c2의 color=blue, gearType=auto, door=4
```

5.6 참조변수 this

▸ this – 인스턴스 자신을 가리키는 참조변수. 인스턴스의 주소가 저장되어있음
모든 인스턴스 메서드에 지역변수로 숨겨진 채로 존재

```
1  class Car {
2      String color;
3      String gearType;
4      int door;
5
6      Car() {
7          //Card("white","auto",4);
8          this("white","auto",4);
9      }
10
11     Car(String c, String g, int d){
12         color = c;
13         gearType = g;
14         door = d;
15     }
16 }
17
```

* 인스턴스변수와 지역변수를 구별하기 위해 참조변수 this사용

```
Car(String color, String gearType, int door){
    this.color = color;
    this.gearType = gearType;
    this.door = door;
}
```

인스턴스변수 / 지역변수

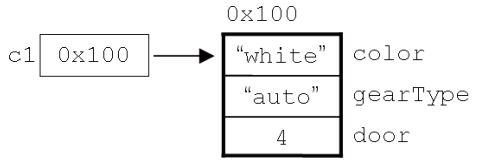
5.7 생성자를 이용한 인스턴스의 복사

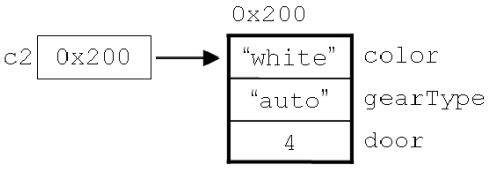
– 인스턴스간의 차이는 인스턴스변수의 값 뿐 나머지는 동일하다.

– 생성자에서 참조변수를 매개변수로 받아서 인스턴스변수들의 값을 복사한다.

– 똑같은 속성값을 갖는 독립적인 인스턴스가 하나 더 만들어진다.

```
1  class Car {
2      String color;       // 색상
3      String gearType;    // 변속기 종류 - auto(자동), manual(수동)
4      int door;           // 문의 개수
5
6      Car() {
7          this("white", "auto", 4);
8      }
9
10     Car(String color, String gearType, int door) {
11         this.color = color;
12         this.gearType = gearType;
13         this.door = door;
14     }
15
16     Car(Car c) {    // 인스턴스의 복사를 위한 생성자.
17         color = c.color;
18         gearType = c.gearType;
19         door = c.door;
20     }
21 }
22
23 class CarTest3 {
24     public static void main(String[] args) {
25         Car c1 = new Car();
26         Car c2 = new Car(c1); // Car(Car c)를 호출
27     }
28 }
```






```
Car(Car c) {
    this(c.color, c.gearType, c.door);
}
```

예제 6-21

```
class Car {
		String color; // 색상
		String gearType;		// 변속기 종류 - auto(자동), manual(수동)
		int door;			// 문의 개수

		Car() {
			this("white", "auto", 4);
		}

		Car(Car c) { // 인스턴스의 복사를 위한 생성자.
			color = c.color;
			gearType = c.gearType;
			door = c.door;
		}

		Car(String color, String gearType, int door) {
			this.color = color;
			this.gearType = gearType;
			this.door = door;
		}
}
class CarTest3 {
		public static void main(String[] args) {
			Car c1 = new Car();
			Car c2 = new Car(c1);	// c1의 복사본 c2를 생성한다.
			System.out.println("c1의 color=" + c1.color + ", gearType="
		+ c1.gearType+ ", door="+c1.door);
			System.out.println("c2의 color=" + c2.color + ", gearType="
		+ c2.gearType+ ", door="+c2.door);

			c1.door=100;			// c1의 인스턴스변수 door의 값을 변경한다.
			System.out.println("c1.door=100; 수행 후");
			System.out.println("c1의 color=" + c1.color + ", gearType="
		+ c1.gearType+ ", door="+c1.door);
			System.out.println("c2의 color=" + c2.color + ", gearType="
		+ c2.gearType+ ", door="+c2.door);
		}
}
```

실행결과

```
c1의 color=white, gearType=auto, door=4
c2의 color=white, gearType=auto, door=4
c1.door=100; 수행 후
c1의 color=white, gearType=auto, door=100
c2의 color=white, gearType=auto, door=4
```

6. 변수의 초기화

6.1 변수의 초기화

- 변수를 선언하고 처음으로 값을 저장하는 것
- 멤버변수(인스턴스변수,클래스변수)와 배열은 각 타입의 기본값으로 자동초기화되므로 초기화를 생략할 수 있다.
- 지역변수는 사용전에 꼭!!! 초기화를 해주어야 한다.

자료형	기본값
boolean	false
char	'\u0000'
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d 또는 0.0
참조형 변수	null

```
class InitTest {
      int x;           // 인스턴스변수
      int y = x;       // 인스턴스변수

      void method1() {
            int i;       // 지역변수
            int j = i;   // 컴파일 에러!!! 지역변수를 초기화하지 않고 사용했음.
      }
}
```

▶ 변수의 초기화 예

선언예	설 명
int i=10; int j=10;	int형 변수 i를 선언하고 10으로 초기화 한다. int형 변수 j를 선언하고 10으로 초기화 한다.
int i=10, j=10;	같은 타입의 변수는 콤마(,)를 사용해서 함께 선언하거나 초기화 할 수 있다.
int i=10, long j=0;	타입이 다른 변수는 함께 선언하거나 초기화할 수 없다.
int i=10; int j=i;	변수 i에 저장된 값으로 변수 j를 초기화 한다. 변수 j는 i의 값인 10으로 초기화 된다.
int j=i; int i=10;	변수 i가 선언되기 전에 i를 사용할 수 없다.

```
class Test
{
    int j = i;
    int i = 10; // 에러!!!
}
```

```
class Test
{
    int i = 10;
    int j = i;  // OK
}
```

6.2 멤버변수의 초기화

▶ 멤버변수의 초기화 방법

1. 명시적 초기화(explicit initialization)

```
class Car {
     int door = 4;               // 기본형(primitive type) 변수의 초기화
     Engine e = new Engine();    // 참조형(reference type) 변수의 초기화

     //...
}
```

2. 생성자(constructor)

```
Car(String color, String gearType, int door){
    this.color = color;
    this.gearType = gearType;
    this.door = door;
}
```

3. 초기화 블럭(initialization block)

- 인스턴스 초기화 블럭 : { }
- 클래스 초기화 블럭 : static { }

6.3 초기화 블럭(initialization block)

▶ 클래스 초기화 블럭 – 클래스변수의 복잡한 초기화에 사용되며 클래스가 로딩될 때 실행된다.

▶ 인스턴스 초기화 블럭 – 생성자에서 공통적으로 수행되는 작업에 사용되며 인스턴스가 생성될 때 마다 (생성자보다 먼저) 실행된다.

```
class InitBlock {
     static { /* 클래스 초기화블럭 입니다. */ }

     { /* 인스턴스 초기화블럭 입니다. */ }

     // ...
}
```

```
1  class StaticBlockTest {
2      static int[] arr = new int[10]; // 명시적 초기화
3
4      static { // 배열 arr을 1~10사이의 값으로 채운다.
5          for(int i=0;i<arr.length;i++) {
6              arr[i] = (int)(Math.random()*10) + 1;
7          }
8      }
9      //...
10 }
```

예제 6-22

```
class BlockTest {

	static {
		System.out.println("static { }");
	}

	{
		System.out.println("{ }");
	}

	public BlockTest() {
		System.out.println("생성자");
	}

	public static void main(String args[]) {
		System.out.println("BlockTest bt = new BlockTest(); ");
		BlockTest bt = new BlockTest();

		System.out.println("BlockTest bt2 = new BlockTest(); ");
		BlockTest bt2 = new BlockTest();
	}
}
```

실행결과

```
arr[0] :4
arr[1] :8
arr[2] :7
arr[3] :2
arr[4] :2
arr[5] :10
arr[6] :7
arr[7] :10
arr[8] :1
arr[9] :7
```

6.4 멤버변수의 초기화 시기와 순서

▶ 클래스변수 초기화 시점 : 클래스가 처음 로딩될 때 단 한번

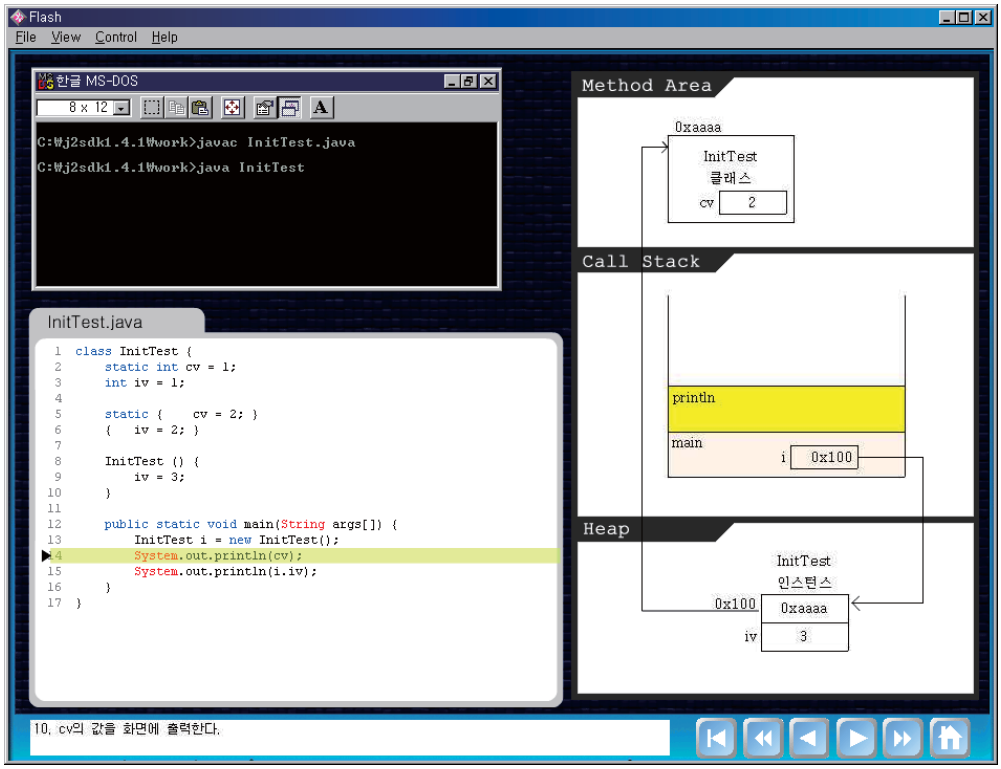
▶ 인스턴스변수 초기화 시점 : 인스턴스가 생성될 때 마다

```
1  class InitTest {
2      static int cv = 1;  // 명시적 초기화
3      int iv = 1;         // 명시적 초기화
4
5      static {    cv = 2; }   // 클래스 초기화 블럭
6      {   iv = 2; }           // 인스턴스 초기화 블럭
7
8      InitTest() { // 생성자
9          iv = 3;
10     }
11 }
```

```
InitTest it = new InitTest();
```

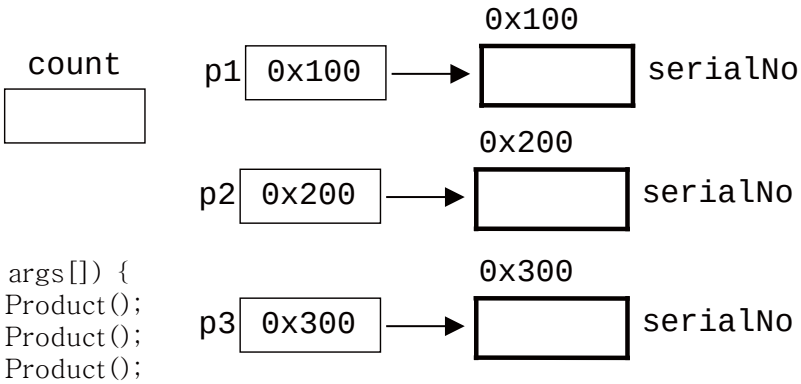
클래스 초기화			인스턴스 초기화			
기본값	명시적 초기화	클래스 초기화블럭	기본값	명시적 초기화	인스턴스 초기화블럭	생성자
cv 0	cv 1	cv 2	cv 2 iv 0	cv 2 iv 1	cv 2 iv 2	cv 2 iv 3
1	2	3	4	5	6	7

6.4 멤버변수의 초기화 시기와 순서



예제 6-24

```
class Product {
        static int count = 0;    // 생성된 인스턴스의 수를 저장하기 위한 변수
        int serialNo;                    // 인스턴스 고유의 번호

        {
                ++count;
                serialNo = count;
        }

        public Product() {}
}

class ProductTest {
        public static void main(String args[]) {
                Product p1 = new Product();
                Product p2 = new Product();
                Product p3 = new Product();

                System.out.println("p1의 제품번호(serial no)는 " + p1.serialNo);
                System.out.println("p2의 제품번호(serial no)는 " + p2.serialNo);
                System.out.println("p3의 제품번호(serial no)는 " + p3.serialNo);
                System.out.println("생산된 제품의 수는 모두 "+Product.count+"개 입니다.");
        }
}
```



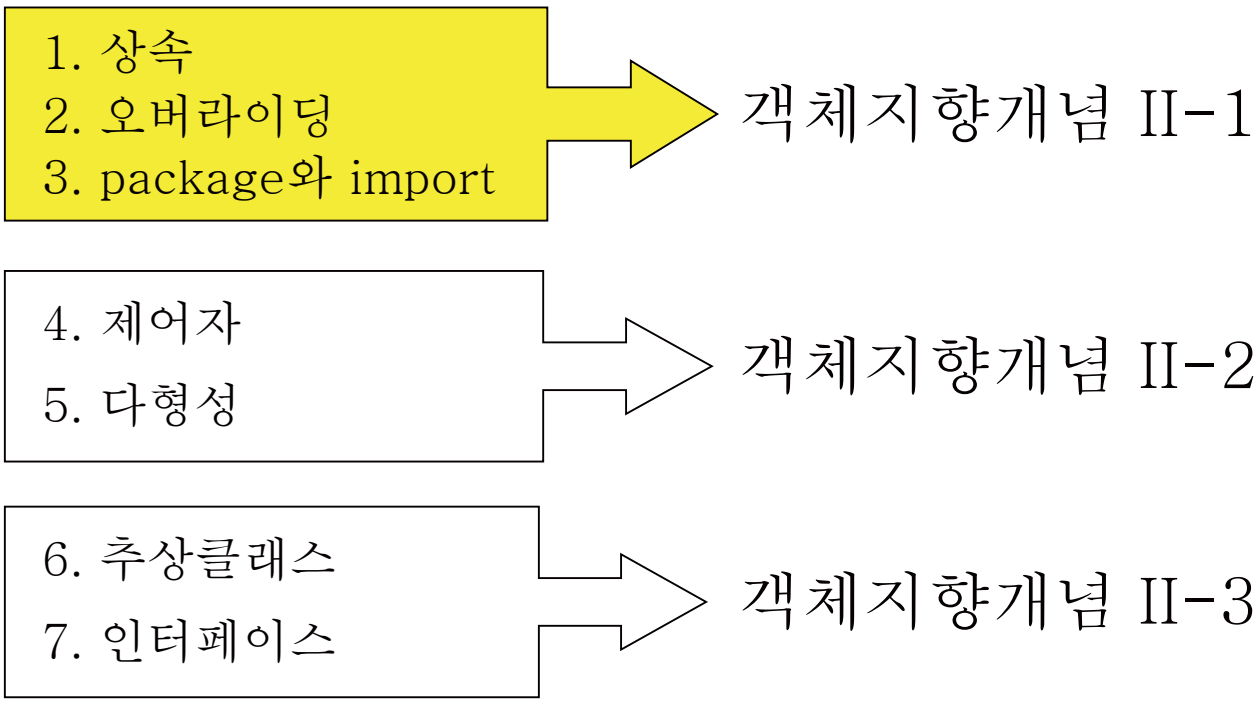
실행결과

```
p1의 제품번호(serial no)는 1
p2의 제품번호(serial no)는 2
p3의 제품번호(serial no)는 3
생산된 제품의 수는 모두 3개 입니다
```

제 7 장
객체지향개념 II-1

인제대학교 전자IT기계자동차공학부

황 원 주



1. 상속(inheritance)

1.1 상속(inheritance)의 정의와 장점

▸ 상속이란?

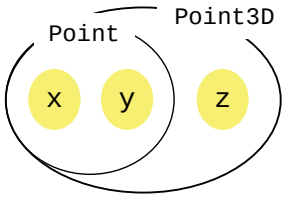
- 기존의 클래스를 재사용해서 새로운 클래스를 작성하는 것.
- 두 클래스를 조상과 자손으로 관계를 맺어주는 것.
- 자손은 조상의 모든 멤버를 상속받는다.(생성자, 초기화블럭 제외)
- 자손의 멤버개수는 조상보다 적을 수 없다.(같거나 많다.)

```
class 자손클래스 extends 조상클래스 {
    // ...
}
```

```
class Point {
    int x;
    int y;
}
```

```
class Point3D {
    int x;
    int y;
    int z;
}
```

```
class Point3D extends Point {
    int z;
}
```



예제 7-1

```
class Tv {
        boolean power;      // 전원상태(on/off)
        int channel;        // 채널

        void power() {      power = !power; }
        void channelUp() {          ++channel; }
        void channelDown() {        --channel;        }
}

class CaptionTv extends Tv {
        boolean caption;   // 캡션상태(on/off)
        void displayCaption(String text) {
                if (caption) {     // 캡션 상태가 on(true)일 때만 text를 보여 준다.
                        System.out.println(text);
                }
        }
}

class CaptionTvTest {
        public static void main(String args[]) {
                CaptionTv ctv = new CaptionTv();
                ctv.channel = 10;                       // 조상 클래스로부터 상속받은 멤버
                ctv.channelUp();                        // 조상 클래스로부터 상속받은 멤버
                System.out.println(ctv.channel);
                ctv.displayCaption("Hello, World");
                ctv.caption = true;                     // 캡션기능을 켠다.
                ctv.displayCaption("Hello, World");  // 캡션을 화면에 보여 준다.
        }
}
```

실행결과

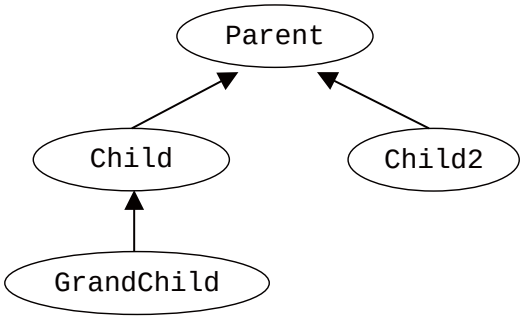
```
11
Hello, World
```

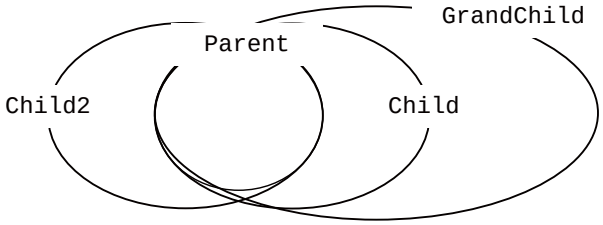
1.2 클래스간의 관계

▶ 상속관계(inheritance)

- 하위 클래스의 객체가 상위클래스의 객체를 물려 받는 것
- 공통부분은 조상에서 관리하고 개별부분은 자손에서 관리한다
- 조상의 변경은 자손에 영향을 미치지만, 자손의 변경은 조상에 아무런 영향을 미치지 않는다.

```
class Parent {}
class Child extends Parent {}
class Child2 extends Parent {}
class GrandChild extends Child {}
```





▶ 포함관계(composite)

- 한 클래스의 멤버변수로 다른 클래스를 선언하는 것
- 작은 단위의 클래스를 먼저 만들고, 이 들을 조합해서 하나의 커다란 클래스를 만든다.

```
class Circle {
    int x; // 원점의 x좌표
    int y; // 원점의 y좌표
    int r; // 반지름(radius)
}
```

```
class Point {
    int x;
    int y;
}
```

```
class Circle {
    Point c = new Point(); // 원점
    int r; // 반지름(radius)
}
```

```
class Car {
    Engine e = new Engine(); // 엔진
    Door[] d = new Door[4];  // 문, 문의 개수를 넷으로 가정하고 배열로 처리했다.
    //...
}
```

1.3 클래스간의 관계결정하기 – 상속 vs. 포함

- 가능한 한 많은 관계를 맺어주어 재사용성을 높이고 관리하기 쉽게 한다.->(예제 클래스가 간단해서 이 두 코드의 차이가 별로 크게 와 닿지 않겠지만) 클래스가 복잡할 수록 관련된 멤버들을 묶어서 여러 개의 작은 클래스로 정의하고 이들을 포함시키는 것이 코드를 간결하고 이해하기 쉽게 만들어줄 뿐만 아니라 이 클래스들은 다른 곳에서 재사용될 수도 있고 나중에 변경사항이 생기더라도 코드의 유지 및 관리가 쉽다는 장점이 있다.

-'is-a'와 'has-a'를 가지고 문장을 만들어 본다.

원(Circle)은 점(Point)**이다**. - Circle **is a** Point.
원(Circle)은 점(Point)을 **가지고 있다**. - Circle **has a** Point.

상속관계 - '~은 ~이다.(is-a)'
포함관계 - '~은 ~을 가지고 있다.(has-a)'

```
class Point {
    int x;
    int y;
}
```

```
class Circle extends Point{
    int r; // 반지름(radius)
}
```

```
class Circle {
    Point c = new Point(); // 원점
    int r; // 반지름(radius)
}
```

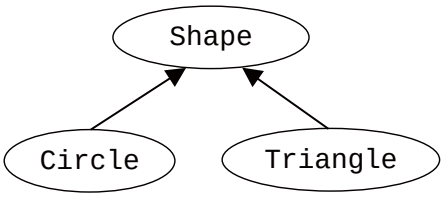
▶ 클래스간의 관계결정하기의 예: [예제7-2]

- 원(Circle)은 도형(Shape)이다.(A Circle is a Shape.) : 상속관계
- 원(Circle)은 점(Point)를 가지고 있다.(A Circle has a Point.) : 포함관계

```
class Shape {
    String color = "blue";
    void draw() {
        // 도형을 그린다.
    }
}
```

```
class Point {
    int x;
    int y;

    Point() {
        this(0,0);
    }

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```
class Circle extends Shape {
    Point center;
    int r;

    Circle() {
        this(new Point(0,0),100);
    }

    Circle(Point center, int r) {
        this.center = center;
        this.r = r;
    }
}

class Triangle extends Shape {
    Point[] p;

    Triangle(Point[] p) {
        this.p = p;
    }

    Triangle(Point p1, Point p2, Point p3) {
        p = new Point[]{p1,p2,p3};
    }
}
```




```
Circle c1 = new Circle();
Circle c2 = new Circle(new Point(150,150),50);

Point[] p = {new Point(100,100),
             new Point(140,50),
             new Point(200,100)
            };
Triangle t1 = new Triangle(p);
```

예제 7-2 (1)

```
import java.awt.Frame;
import java.awt.Graphics;

class DrawShape extends Frame {
	public static void main(String[] args) {
		DrawShape win = new DrawShape("도형그리기");
	}

	public void paint(Graphics g) {
		Point[] p = { new Point(100, 100)
		  , new Point(140, 50)
		  , new Point(200, 100)};
		Triangle t = new Triangle(p);
		Circle c = new Circle(new Point(150, 150), 50);

		// 원을 그린다.
		g.drawOval(c.center.x, c.center.y, c.r, c.r);

		// 직선 3개로 삼각형을 그린다.
		g.drawLine(t.p[0].x, t.p[0].y, t.p[1].x, t.p[1].y);
		g.drawLine(t.p[1].x, t.p[1].y, t.p[2].x, t.p[2].y);
		g.drawLine(t.p[2].x, t.p[2].y, t.p[0].x, t.p[0].y);
	}

	DrawShape(String title) {
		super(title);
		setSize(300, 300);
		setVisible(true);
	}
}
```

예제 7-2 (2)

```
class Point {
	int x;
	int y;

	Point(int x, int y) {
		this.x = x;
		this.y = y;
	}

	Point() {
		this(0,0);
	}
}

class Circle {
	Point center;		// 원의 원점좌표
	int r;			// 반지름

	Circle() {
		this(new Point(0, 0), 100);
	}

	Circle(Point center, int r) {
		this.center = center;
		this.r = r;
	}
}

class Triangle {
	Point[] p = new Point[3];

	Triangle(Point[] p) {
		this.p = p;
	}

	Triangle(Point p1, Point p2, Point p3) {
		p[0] = p1;
		p[1] = p2;
		p[2] = p3;
	}
}
```

▶ 클래스간의 관계결정하기의 예: [예제7-3]

```
class Deck {
    final int CARD_NUM = 52;    // 카드의 개수
    Card c[] = new Card[CARD_NUM];

    Deck () {   // Deck의 카드를 초기화한다.
        int i=0;

        for(int k=Card.KIND_MAX; k > 0; k--) {
            for(int n=1; n < Card.NUM_MAX + 1 ; n++) {
                c[i++] = new Card(k, n);
            }
        }
    }

    Card pick(int index) {  // 지정된 위치(index)에 있는 카드 하나를 선택한다.
        return c[index%CARD_NUM];
    }

    Card pick() {           // Deck에서 카드 하나를 선택한다.
        int index = (int)(Math.random() * CARD_NUM);
        return pick(index);
    }

    void shuffle() {            // 카드의 순서를 섞는다.
        for(int n=0; n < 1000; n++) {
            int i = (int)(Math.random() * CARD_NUM);
            Card temp = c[0];
            c[0] = c[i];
            c[i] = temp;
        }
    }
} // Deck클래스의 끝
```

```
public static void main(String[] args) {
    Deck d = new Deck();
    Card c = d.pick();

    d.shuffle();
    Card c2 = d.pick(55);
}
```

예제 7-3 (1)

```
class DeckTest
{
        public static void main(String args[])
    {
                Deck d = new Deck();        // 카드 한 벌(Deck)을 만든다.
                Card c = d.pick(0);         // 섞기 전에 제일 위의 카드를 뽑는다.
                System.out.println(c);
                d.shuffle();                // 카드를 섞는다.
                c = d.pick(0);              // 섞은 후에 제일 위의 카드를 뽑는다.
                System.out.println(c);
        }
}

// Deck클래스
class Deck {
        final int CARD_NUM = 52;    // 카드의 개수
        Card c[] = new Card[CARD_NUM];

        Deck () {// Deck의 카드를 초기화한다.
                int i=0;

                for(int k=Card.KIND_MAX; k > 0; k--) {
                        for(int n=1; n < Card.NUM_MAX + 1 ; n++) {
                                c[i++] = new Card(k, n);
                        }
                }
        }

        Card pick(int index) {      // 지정된 위치(index)에 있는 카드 하나를 선택한다.
                if(0 <= index && index < CARD_NUM)
                        return c[index];
                else
                        return pick();
        }
```

예제 7-3 (2)

```
        Card pick() {                           // Deck에서 카드 하나를 선택한다.
                int index = (int)(Math.random() * CARD_NUM);
                return pick(index);
        }

        void shuffle() {                        // 카드의 순서를 섞는다.
                for(int n=0; n < 1000; n++) {
                        int i = (int)(Math.random() * CARD_NUM);
                        Card temp = c[0];
                        c[0] = c[i];
                        c[i] = temp;
                }
        }
} // Deck클래스의 끝

// Card클래스
class Card {
        static final int KIND_MAX = 4;          // 카드 무늬의 수
        static final int NUM_MAX  = 13;         // 무늬별 카드 수

        static final int SPADE   = 4;
        static final int DIAMOND = 3;
        static final int HEART   = 2;
        static final int CLOVER  = 1;

        int kind;
        int number;

        Card() {
                this(SPADE, 1);
        }

        Card(int kind, int number) {
                this.kind = kind;
                this.number = number;
        }
```

예제 7-3 (3)

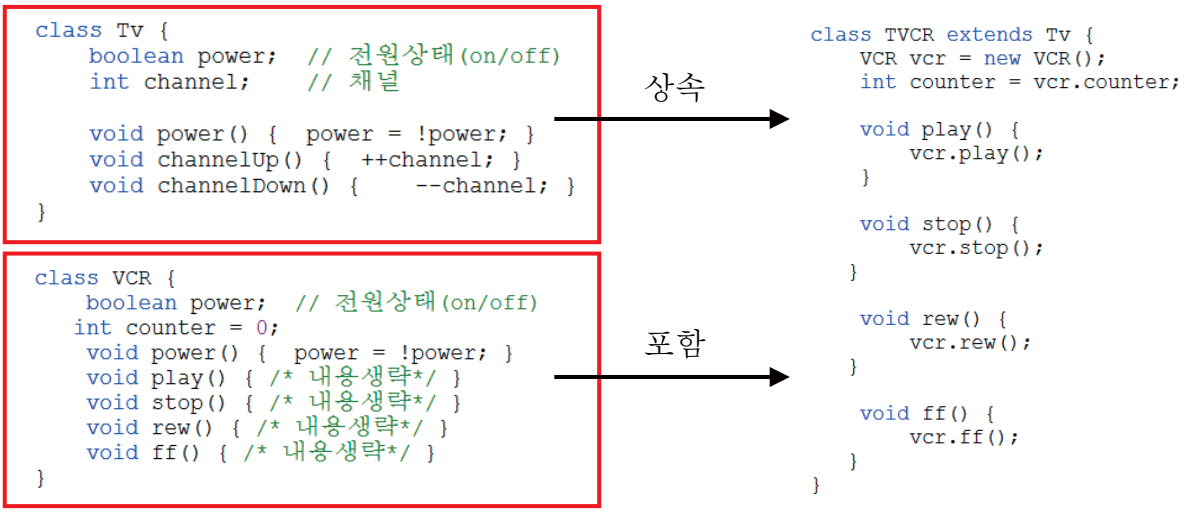
```
public String toString() {
                String kind="";
                String number="";

                switch(this.kind) {
                        case 4 :
                                kind = "SPADE";
                                break;
                        case 3 :
                                kind = "DIAMOND";
                                break;
                        case 2 :
                                kind = "HEART";
                                break;
                        case 1 :
                                kind = "CLOVER";
                                break;
                        default :
                }

                switch(this.number) {
                        case 13 :
                                number = "K";
                                break;
                        case 12 :
                                number = "Q";
                                break;
                        case 11 :
                                number = "J";
                                break;
                        default :
                                number = this.number + "";
                }
                return "kind : " + kind + ", number : " + number;
        } // toString()의 끝
} // Card클래스의 끝
```

1.4 단일상속(single inheritance)

- Java는 단일상속만을 허용한다.(C++은 다중상속 허용)

```
class TVCR extends TV, VCR {       // 이와 같은 표현은 허용하지 않는다.
    //...
}
```

- 비중이 높은 클래스 하나만 상속관계로, 나머지는 포함관계로 한다.



예제 7-4

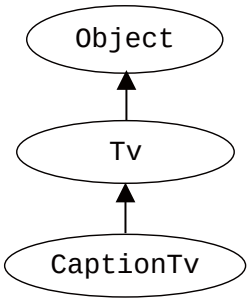
```
class Tv {
        boolean power;      // 전원상태(on/off)
        int channel;                // 채널

        void power() {      power = !power; }
        void channelUp() {          ++channel; }
        void channelDown() {        --channel; }
}

class VCR {
        boolean power;      // 전원상태(on/off)
   int counter = 0;
        void power() {      power = !power; }
        void play() { /* 내용생략*/ }
        void stop() { /* 내용생략*/ }
        void rew() { /* 내용생략*/ }
        void ff() { /* 내용생략*/ }
}

class TVCR extends Tv {         //비중이 높은 TV클래스는 상속시켜서 사용한다.
        VCR vcr = new VCR();  //VCR클래스를 포함시켜서 사용한다.
        int counter = vcr.counter;

        void play() {
                vcr.play();
        }

        void stop() {
                vcr.stop();
   }

        void rew() {
                vcr.rew();
   }

        void ff() {
                vcr.ff();
   }
}
```

1.5 Object클래스 – 모든 클래스의 최고조상

- 조상이 없는 클래스는 자동적으로 Object클래스를 상속받게 된다.
- 상속계층도의 최상위에는 Object클래스가 위치한다.
- 모든 클래스는 Object클래스에 정의된 11개의 메서드를 상속받는다.
  toString(), equals(Object obj), hashCode(), ...

```
class Tv {
    // ...
}

class CaptionTv extends Tv {
    // ...
}
```

```
class Tv extends Object {
    // ...
}

class CaptionTv extends Tv {
    // ...
}
```



2. 오버라이딩(overriding)

2.1 오버라이딩(overriding)이란?

"조상클래스로부터 상속받은 메서드의 내용을 상속받는 클래스에 맞게 변경하는 것을 오버라이딩이라고 한다."

* override - vt. '~위에 덮어쓰다(overwrite).', '~에 우선하다.'

```
class Point {
     int x;
     int y;

     String getLocation() {
          return "x :" + x + ", y :"+ y;
     }
}

class Point3D extends Point {
     int z;
     String getLocation() {       // 오버라이딩
          return "x :" + x + ", y :"+ y + ", z :" + z;
     }
}
```

2.2 오버라이딩의 조건

1. 선언부가 같아야 한다.(이름, 매개변수, 리턴타입)
2. 접근제어자를 좁은 범위로 변경할 수 없다.
   - 조상의 메서드가 protected라면, 범위가 같거나 넓은 protected나 public으로만 변경할 수 있다.
3. 조상클래스의 메서드보다 많은 수의 예외를 선언할 수 없다.

```
class Parent {
    void parentMethod() throws IOException, SQLException {
        // ...
    }
}

class Child extends Parent {
    void parentMethod() throws IOException {
        //..
    }
}

class Child2 extends Parent {
    void parentMethod() throws Exception {
        //..
    }
}
```

2.3 오버로딩 vs. 오버라이딩

오버로딩(overloading) - 기존에 없는 새로운 메서드를 정의하는 것(new)
오버라이딩(overriding) - 상속받은 메서드의 내용을 변경하는 것(change, modify)

▶ 오버로딩

- 동일 클래스 내에서 동일 이름으로 메서드를 중복 정의
- 메서드의 이름을 동일하나 매개변수의 개수나 타입은 달라야 한다.

▶ 오버라이딩

- 계층관계에서 하위 클래스에서 상위 클래스의 메서드를 동일 이름으로 중복 정의
- 메서드의 이름, 매개변수의 개수나 타입도 모두 동일하다.

```
class Parent {
    void parentMethod() {}
}

class Child extends Parent {
    void parentMethod() {}         // 오버라이딩
    void parentMethod(int i) {}    // 오버로딩

    void childMethod() {}
    void childMethod(int i) {}     // 오버로딩
    void childMethod() {}          // 에러!!! 중복정의임
}
```

2.4 super – 참조변수

※ 클래스에서 선언된 멤버변수와 동일한 이름을 가진 지역변수가 선언되었을 때, 멤버변수는 가려져서 접근이 불가능하게 된다.

▶ this – 자기자신 클래스 인스턴스를 가리키는 참조변수. 인스턴스의 주소가 저장되어있음

- 모든 인스턴스 메서드(비교:클래스 메서드)에 지역변수로 숨겨진 채로 존재
- 멤버변수와 지역변수를 구분하기 위해 사용

※ 상위 클래스에서 선언된 변수와 동일한 이름을 가진 변수가 하위 클래스에서 선언되었을 때, 하위 클래스에서 상위 클래스의 변수는 가려져서 접근이 불가능하게 된다.

※ 상위 클래스에서 정의된 메서드가 하위 클래스에서 오버라이딩되었을 때, 하위 클래스에서 상위 클래스의 메서드는 접근할 수 없다.

▶ super – 조상 클래스 인스턴스를 가리키는 참조변수. 인스턴스의 주소가 저장되어있음

- 모든 인스턴스 메서드에 지역변수로 숨겨진 채로 존재
- 조상의 멤버변수(또는 멤버메서드)와 자신의 멤버변수(또는 멤버메서드)를 구분하는 데 사용

예제 7-5

```
class SuperTest {
   public static void main(String args[]) {
      Child c = new Child();
      c.method();
   }
}

class Parent {
   int x=10;
}

class Child extends Parent {
   void method() {
      System.out.println("x=" + x);
      System.out.println("this.x=" + this.x);
      System.out.println("super.x="+ super.x);

  // 자신의 멤버가 조상의 멤버와 중복되지 않기 때문에... super를 사용할 필요가 없음. 상속받은 멤버도 자신의 멤버이므로 this를 사용하는 것만으로도 충분함. 단지 super를 사용했을 때 어떤 결과가 나오는지 보여주기 위한 것임.

   }
}
```

실행결과

```
x=10
this.x=10
super.x=10
```

예제 7-6

```
class SuperTest2 {
   public static void main(String args[]) {
      Child c = new Child();
      c.method();
   }
}

class Parent {
   int x=10;
}

class Child extends Parent {
   int x=20;

   void method() {
      System.out.println("x=" + x);
      System.out.println("this.x=" + this.x);
      System.out.println("super.x="+ super.x);
   }
}
```

실행결과

```
x=20
this.x=20
super.x=10
```

예제

```
class Super1 {
  int x = 12;
  void p() {
              System.out.println("Super1 p() is called");
  }
}
class Super2 extends Super1 {
  int x = 24;
  void p() {
              System.out.println("Super2 p() is called");
  }
  void test() {
              System.out.println(" super x  = "+super.x);
              System.out.println("current x = "+x);
              super.p();              // 상위 클래스 p() 호출
              this.p();  // 현재 클래스 p() 호출
  }
}
class SuperTest {
  public static void main(String[] a) {
              Super2 s2 = new Super2();
              s2.test();
  }
}
```

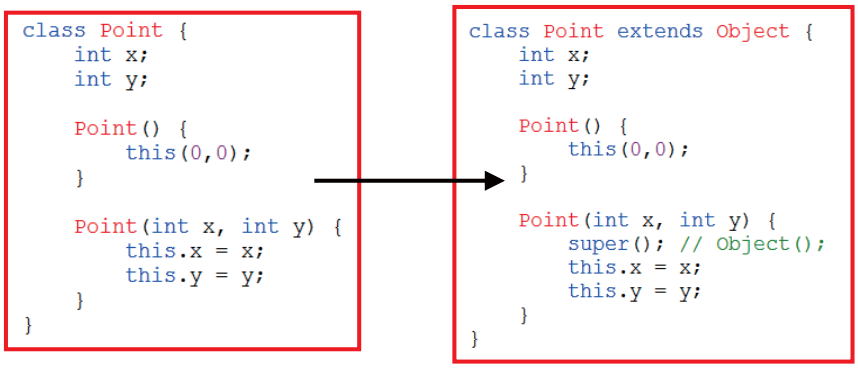
실행결과

```
super x = 12

current x = 24

Super1 p() is called

Super2 p() is called
```

2.5 super() – 조상의 생성자

- (비교) this(): 같은 클래스의 다른 생성자 호출
- 자손클래스의 인스턴스를 생성하면, 자손의 멤버와 조상의 멤버가 합쳐진 하나의 인스턴스가 생성된다.
- 조상의 멤버들도 초기화되어야 하기 때문에 자손의 생성자의 첫 문장에서 조상의 생성자를 호출해야 한다.

Object클래스를 제외한 모든 클래스의 생성자 첫 줄에는 생성자(같은 클래스의 다른 생성자 또는 조상의 생성자)를 호출해야한다.
그렇지 않으면 컴파일러가 자동적으로 'super();'를 생성자의 첫 줄에 삽입한다.



```
class Point {
    int x;
    int y;

    Point() {
        this(0,0);
    }

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```
class Point extends Object {
    int x;
    int y;

    Point() {
        this(0,0);
    }

    Point(int x, int y) {
        super(); // Object();
        this.x = x;
        this.y = y;
    }
}
```

예제7-8

```
class PointTest2 {
        public static void main(String argsp[]) {
                Point3D p3 = new Point3D();
                System.out.println("p3.x=" + p3.x);
                System.out.println("p3.y=" + p3.y);
                System.out.println("p3.z=" + p3.z);
        }
}

class Point {
        int x=10;
        int y=20;

        Point(int x, int y) {
                this.x = x;
                this.y = y;
        }
}

class Point3D extends Point {
        int z=30;

        Point3D() {
                this(100, 200, 300);   // Point3D(int x, int y, int z)를 호출한다.
        }

        Point3D(int x, int y, int z) {
                super(x, y);           // Point(int x, int y)를 호출한다.
                this.z = z;
        }
}
```

실행결과

```
p3.x=100
p3.y=200
p3.z=300
```

```java
class Point {
    int x;
    int y;

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    String getLocation() {
        return "x :" + x + ", y :"+ y;
    }
}

class Point3D extends Point {
    int z;

    Point3D(int x, int y, int z) {

        this.x = x;
        this.y = y;
        this.z = z;
    }

    String getLocation() {  // 오버라이딩
        return "x :" + x + ", y :"+ y + ", z :" + z;
    }
}
```

```java
Point(int x, int y) {
    super(); // Object();
    this.x = x;
    this.y = y;
}
```

```java
class PointTest {
    public static void main(String args[]) {
        Point3D p3 = new Point3D(1,2,3);
    }
}
```

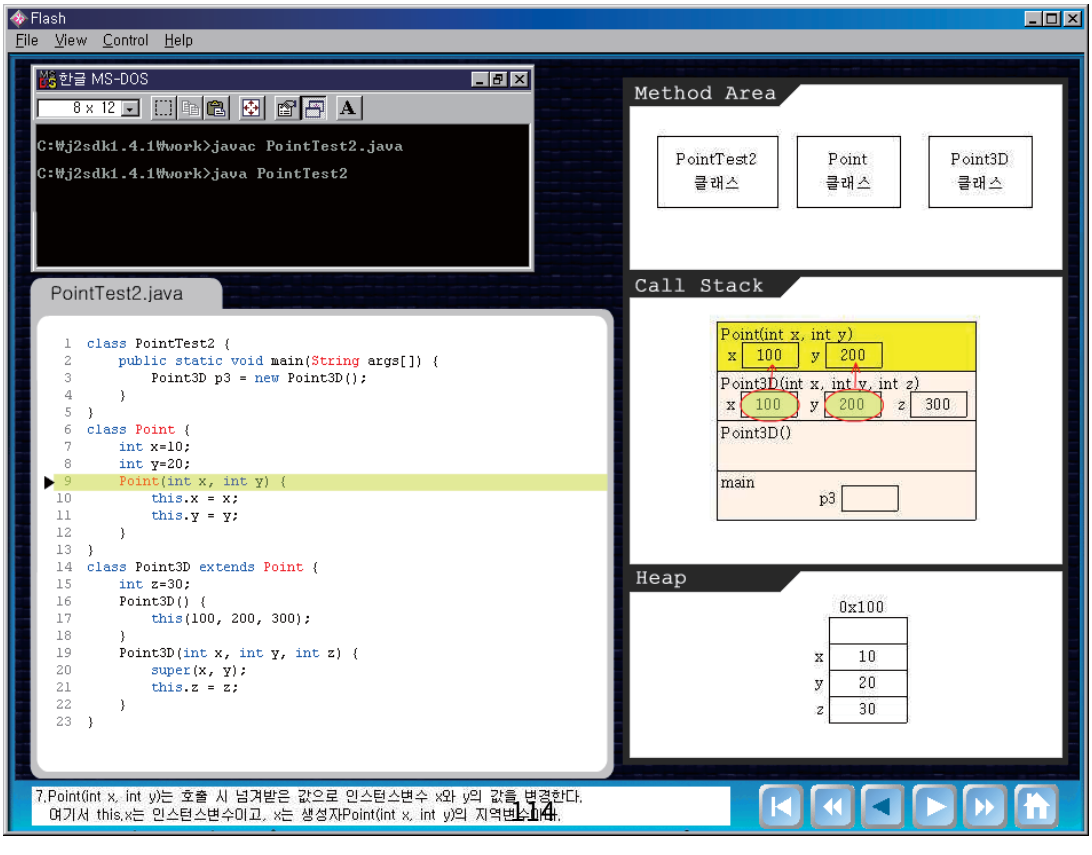
```
---------- javac ----------
PointTest.java:24: cannot find symbol
symbol  : constructor Point()
location: class Point
        Point3D(int x, int y, int z) {
                                     ^
1 error
```

```java
Point3D(int x, int y, int z) {
    super(); // Point()를 호출
    this.x = x;
    this.y = y;
    this.z = z;
}
```

```java
Point3D(int x, int y, int z) {
    // 조상의 생성자 Point(int x, int y)를 호출
    super(x,y);
    this.z = z;
}
```

* 플래시 동영상 : Super.exe 또는 Super.swf

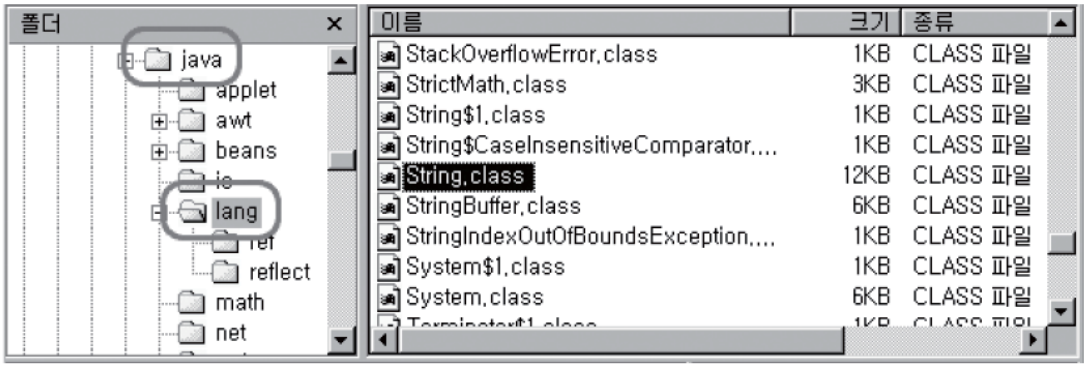
(java_jungsuk_src.zip의 flash폴더에 위치)



3. package와 import

3.1 패키지(package)

- 서로 관련된 클래스와 인터페이스의 묶음.
- 클래스가 물리적으로 클래스파일(*.class)인 것처럼, 패키지는 물리적으로 폴더이다. 패키지는 서브패키지를 가질 수 있으며, '.'으로 구분한다.
- 클래스의 실제 이름(full name)은 패키지명이 포함된 것이다.
  (String클래스의 full name은 java.lang.String)

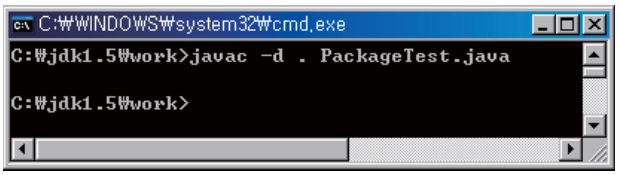


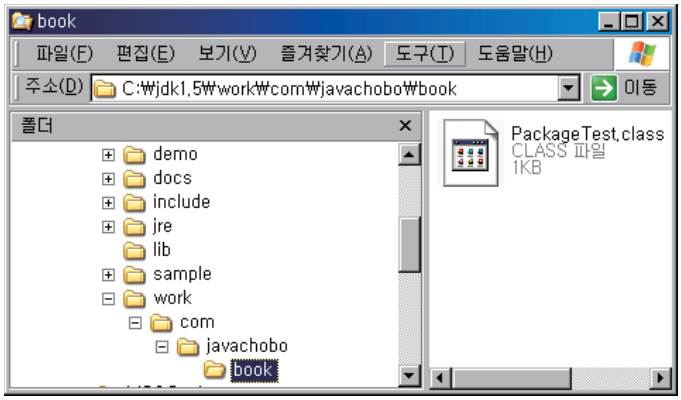
3.2 패키지의 선언

- 패키지는 소스파일에 첫 번째 문장(주석 제외)으로 단 한번 선언한다.
- 하나의 소스파일에 둘 이상의 클래스가 포함된 경우, 모두 같은 패키지에 속하게 된다.(하나의 소스파일에 단 하나의 public클래스만 허용한다.)
- 사용자 정의 패키지 형식

  package packageName;

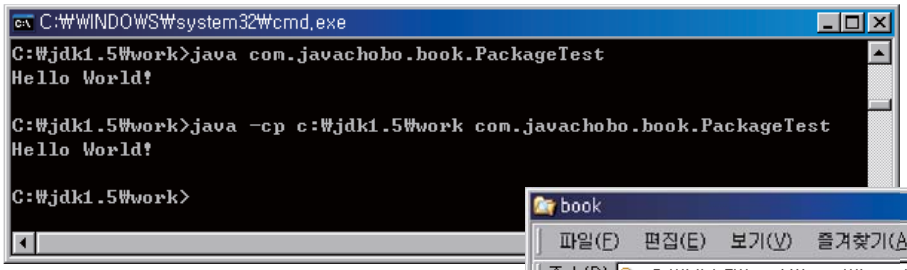
```
1  // PackageTest.java
2  package com.javachobo.book;
3
4  public class PackageTest {
5      public static void main(String[] args) {
6          System.out.println("Hello World!");
7      }
8  }
9
10 public class PackageTest2 {}
```

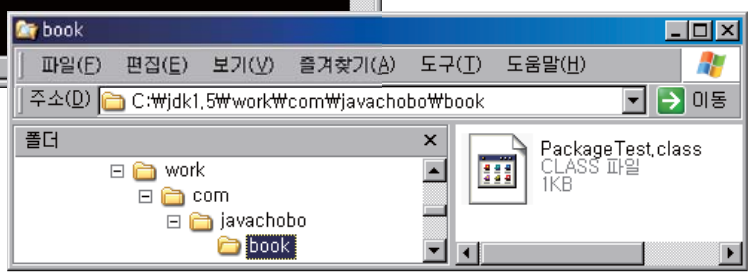




3.3 클래스패스(classpath) 설정

- 클래스패스(classpath)는 클래스파일(*.class)를 찾는 경로. 구분자는 ';'
- 클래스패스에 패키지가 포함된 폴더나 jar파일을(*.jar) 나열한다.
- 클래스패스가 없으면 자동적으로 현재 폴더가 포함되지만 클래스패스를 지정할 때는 현재 폴더(.)도 함께 추가해주어야 한다.





[참고] java.exe의'cp'옵션을 이용해서 일시적으로 클래스패스를 지정해 줄 수도 있다.

3.4 import문

- 사용할 클래스가 속한 패키지를 지정하는데 사용.
- import문을 사용하면 클래스를 사용할 때 패키지명을 생략할 수 있다.

```
class ImportTest {
    java.util.Date today = new java.util.Date();
    // ...
}
```

→

```
import java.util.*;

class ImportTest {
    Date today = new Date();
}
```

- java.lang패키지의 클래스는 import하지 않고도 사용할 수 있다.

String, Object, System, Thread ...

```
import java.lang.*;

class ImportTest2
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

```
public static void main(java.lang.String[] args)
{
    java.lang.System.out.println("Hello World!");
}
```

3.5 import문의 선언

- import문은 패키지문과 클래스선언의 사이에 선언한다.

일반적인 소스파일(*.java)의 구성은 다음의 순서로 되어 있다.

① package문
② import문
③ 클래스 선언

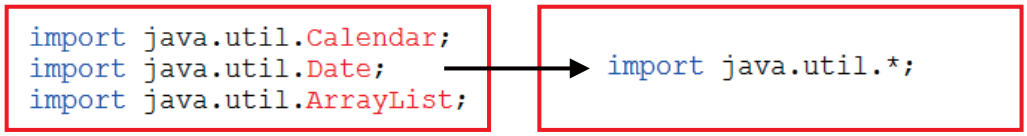
- import문을 선언하는 방법은 다음과 같다.

```
import 패키지명.클래스명;
   또는
import 패키지명.*;
```

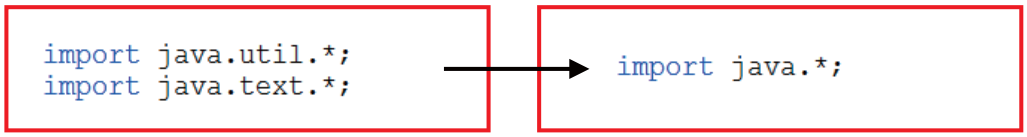
```
1  package com.javachobo.book;
2
3  import java.text.SimpleDateFormat;
4  import java.util.*;
5
6  public class PackageTest {
7      public static void main(String[] args) {
8          // java.util.Date today = new java.util.Date();
9          Date today = new Date();
10         SimpleDateFormat date = new SimpleDateFormat("yyyy/MM/dd");
11     }
12 }
```

▶ import문의 선언의 예

- import문은 컴파일 시에 처리되므로 프로그램의 성능에 아무런 영향을 미치지 않는다.



- 다음의 두 코드는 서로 의미가 다르다.



- 이름이 같은 클래스가 속한 두 패키지를 import할 때는 클래스 앞에 패키지명을 붙여줘야 한다.

```
import java.sql.*;  // java.sql.Date
import java.util.*; // java.util.Date

public class ImportTest {
    public static void main(String[] args) {
        java.util.Date today = new java.util.Date();
    }
}
```

사용자 정의 패키지의 예

① classpath의 설정
c:>SET CLASSPATH =.;C:\MyPackage;

② 패키지 선언 클래스의 컴파일
c:>javac –d c:\MyPackage Sample.java

```
package First.Test;
public class Sample {
   public void print() {
      System.out.println("Sample class Print");
   }
}
```

위 명령을 실행하면, C:\MyPackage\First\Test 디렉토리에 Sample.class 파일이 저장됨

③ 절대 경로의 사용

```
public class Absolute {
  public static void main(String[] arg) {
    First.Test.Sample s1 = new
    First.Test.Sample(); // 절대 경로 클래스명
    s1.print();
  }
}
```

또는 import문의 사용

```
import First.Test.Sample; // 또는 import First.Test.*;
public class Import {
  public static void main(String[] arg) {
    Sample s1 = new Sample();
    s1.print();
  }
}
```

[실행 결과]

```
C:\java\c6>java Import
Sample class Print
```

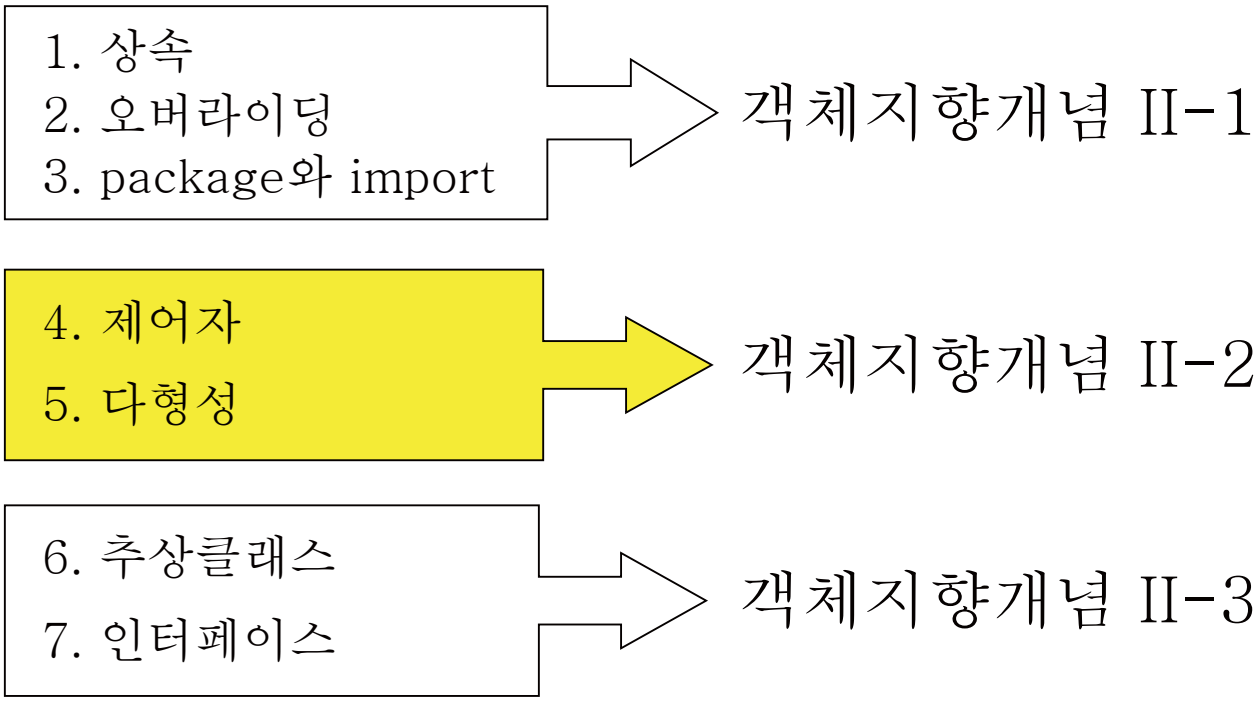
예제

```
class Complex {
   double real;
   double imagine;

   Complex(){
            real=0.0;
            imagine=0.0;
   }
   Complex(double r, double i) {
            real=r;
            imagine=i;
   }
}

class calcComplex {
          private Complex comp=new Complex();

          public Complex addComplex(Complex a, Complex b){
                    comp.real=a.real+b.real;
                    comp.imagine=a.imagine+b.imagine;

                    return comp;
          }
}

class ComplexMain{
          public static void main(String[] args){
                    calcComplex cc=new calcComplex();
                    Complex res, loperand, roperand;

                    res=new Complex();
                    loperand=new Complex(3.0,4.0);
                    roperand=new Complex(5.0,-2.0);

                    res=cc.addComplex(loperand, roperand);
                    System.out.println(res.imagine);
          }
}
```

제 7 장
객체지향개념 II-2

인제대학교 전자IT기계자동차공학부

황 원 주



4. 제어자(modifiers)

4.1 제어자(modifier)란?

- 클래스, 변수, 메서드의 선언부에 사용되어 부가적인 의미를 부여한다.
- 제어자는 크게 접근 제어자와 그 외의 제어자로 나뉜다.
- 하나의 대상에 여러 개의 제어자를 조합해서 사용할 수 있으나,
  접근제어자는 단 하나만 사용할 수 있다.

```
접근 제어자 - public, protected, default, private
그      외 - static, final, abstract, native, transient, synchronized,
             volatile, strictfp
```

4.2 static – (변화 없이) 고정된(→클래스의, 공통적

static이 사용될 수 있는 곳 - 멤버변수, 메서드, 초기화 블럭

제어자	대상	의 미
static	멤버변수	- 모든 인스턴스에 공통적으로 사용되는 클래스변수가 된다. - 클래스변수는 인스턴스를 생성하지 않고도 사용 가능하다. - 클래스가 메모리에 로드될 때 생성된다.
	메서드	- 인스턴스를 생성하지 않고도 호출이 가능한 static 메서드가 된다. - static메서드 내에서는 인스턴스멤버들을 직접 사용할 수 없다.

```java
class StaticTest {
    static int width = 200;
    static int height = 120;

    static { // 클래스 초기화 블럭
        // static변수의 복잡한 초기화 수행
    }

    static int max(int a, int b) {
        return a > b ? a : b;
    }
}
```

4.3 final – 마지막의, 변경될 수 없는

final이 사용될 수 있는 곳 - 클래스, 메서드, 멤버변수, 지역변수

제어자	대상	의 미
final	클래스	변경될 수 없는 클래스, 확장될 수 없는 클래스가 된다. 그래서 final로 지정된 클래스는 다른 클래스의 조상이 될 수 없다.
	메서드	변경될 수 없는 메서드, final로 지정된 메서드는 오버라이딩을 통해 재정의 될 수 없다.
	멤버변수	변수 앞에 final이 붙으면, 값을 변경할 수 없는 상수가 된다.
	지역변수	

[참고] 대표적인 final클래스로는 String과 Math가 있다.

```
final class FinalTest {
    final int MAX_SIZE = 10; // 멤버변수

    final void getMaxSize() {
        final LV = MAX_SIZE; // 지역변수
        return MAX_SIZE;
    }
}

class Child extends FinalTest {
    void getMaxSize() {} // 오버라이딩
}
```

4.4 생성자를 이용한 final 멤버변수 초기화

- final이 붙은 변수는 상수이므로 보통은 선언과 초기화를 동시에 하지만, 인스턴스변수의 경우 생성자에서 초기화 할 수 있다.

```
class Card {
    final int NUMBER;       // 상수지만 선언과 함께 초기화 하지 않고
    final String KIND;      // 생성자에서 단 한번만 초기화할 수 있다.
    static int width = 100;
    static int height = 250;

    Card(String kind, int num) {
        KIND = kind;
        NUMBER = num;
    }

    Card() {
        this("HEART", 1);
    }

    public String toString() {
        return "" + KIND +" "+ NUMBER;
    }
}
```

```
public static void main(String args[]) {
    Card c = new Card("HEART", 10);
//  c.NUMBER = 5;    에러!!!
    System.out.println(c.KIND);
    System.out.println(c.NUMBER);
}
```

4.5 abstract – 추상의, 미완성의

abstract가 사용될 수 있는 곳 - 클래스, 메서드

제어자	대상	의 미
abstract	클래스	클래스 내에 추상메서드가 선언되어 있음을 의미한다.
	메서드	선언부만 작성하고 구현부는 작성하지 않은 추상메서드임을 알린다.

[참고] 추상메서드가 없는 클래스도 abstract를 붙여서 추상클래스로 선언하는 것이 가능하기는 하지만 그렇게 해야 할 이유는 없다.

```
abstract class AbstractTest { // 추상클래스
    abstract void move();     // 추상메서드
}
```

4.6 접근 제어자(access modifier)

- 멤버 또는 클래스에 사용되어, 외부로부터의 접근을 제한한다.

접근 제어자가 사용될 수 있는 곳 - 클래스, 멤버변수, 메서드, 생성자

private - **같은 클래스** 내에서만 접근이 가능하다.
default - **같은 패키지** 내에서만 접근이 가능하다.
protected - 같은 패키지 내에서, 그리고 다른 패키지의 자손클래스에서 접근이 가능하다.
public - 접근 제한이 전혀 없다.

제어자	같은 클래스	같은 패키지	자손클래스	전 체
public	■	■	■	■
protected	■	■	■	
default	■	■		
private	■			

public
(default)

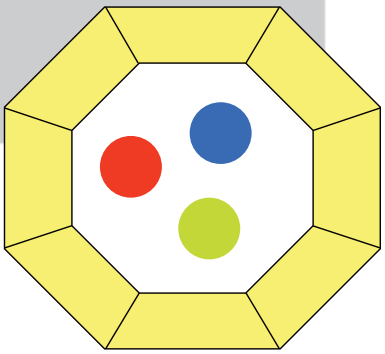
public
protected
(default)
private

```
class AccessModifierTest {
    int iv;         // 멤버변수(인스턴스변수)
    static int cv; // 멤버변수(클래스변수)

    void method() {}
}
```

4.7 접근 제어자를 이용한 캡슐화

접근 제어자를 사용하는 이유

- 외부로부터 데이터를 보호하기 위해서
- 외부에는 불필요한, 내부적으로만 사용되는, 부분을 감추기 위해서



```
class Time {
	private int hour;
	private int minute;
	private int second;

	Time(int hour, int minute, int second) {
		setHour(hour);
		setMinute(minute);
		setSecond(second);
	}

	public int getHour() {		return hour; }

	public void setHour(int hour) {
		if (hour < 0 || hour > 23) return;
		this.hour = hour;
	}

	... 중간 생략 ...

	public String toString() {
		return hour + ":" + minute + ":" + second;
	}
}
```

```
public static void main(String[] args) {
	Time t = new Time(12, 35, 30);
	// System.out.println(t.toString());
	System.out.println(t);
	// t.hour = 13;  에러!!!

	// 현재시간보다 1시간 후로 변경한다.
	t.setHour(t.getHour()+1);
	System.out.println(t);
}
```

```
---------- java ----------
12:35:30
13:35:30

출력 완료 (0초 경과)
```

예제 7-12 (중요)

```
public class TimeTest {
	public static void main(String[] args)
	{
		Time t = new Time(12, 35, 30);
		System.out.println(t);
//		t.hour = 13;
		t.setHour(t.getHour()+1);	// 현재시간보다 1시간 후로 변경한다.
		System.out.println(t);
	}
}

class Time {
	private int hour;
	private int minute;
	private int second;

	Time(int hour, int minute, int second) {
		setHour(hour);
		setMinute(minute);
		setSecond(second);
	}

	public int getHour() {		return hour; }
	public void setHour(int hour) {
		if (hour < 0 || hour > 23) return;
		this.hour = hour;
	}
	public int getMinute() {		return minute; }
	public void setMinute(int minute) {
		if (minute < 0 || minute > 59) return;
		this.minute = minute;
	}
	public int getSecond() {		return second; }
	public void setSecond(int second) {
		if (second < 0 || second > 59) return;
		this.second = second;
	}
	public String toString() {
		return hour + ":" + minute + ":" + second;
	}
}
```

변수 hour의 접근 제어자가 private이므로 접근할 수 없다.

멤버 변수는 접근 제어자를 private으로 하여 외부에서 직접 접근하지 못하도록 한다.(Information hiding)

멤버 메서드는 접근 제어자를 public으로 하여 외부에서 접근 제한자가 private인 멤버 변수에 접근한다. (Data encapsulation)

실행결과

```
12:35:30
13:35:30
```

4.8 생성자의 접근 제어자

- 일반적으로 생성자의 접근 제어자는 클래스의 접근 제어자와 일치한다.
- 생성자에 접근 제어자를 사용함으로써 인스턴스의 생성을 제한할 수 있다.

```
final class Singleton {
    private static Singleton s = new Singleton(

    private Singleton() { // 생성자
        //...
    }

    public static Singleton getInstance() {
        if(s==null) {
            s = new Singleton();
        }
        return s;
    }

    //...
}
```

getInstance()에서 사용될 수 있도록 객체 미리 생성→static

생성자의 접근제어자를 private으로 하면,
-new를 이용하여 객체생성 불가
-자손클래스의 객체를 생성하면 조상클래스의 생성자를 호출하므로 조상클래스 못됨→final

```
class SingletonTest {
    public static void main(String args[]) {
//      Singleton s = new Singleton();  에러!!!
        Singleton s1 = Singleton.getInstance();
    }
}
```

4.9 제어자의 조합

대 상	사용가능한 제어자
클래스	public, (default), final, abstract
메서드	모든 접근 제어자, final, abstract, static
멤버변수	모든 접근 제어자, final, static
지역변수	final

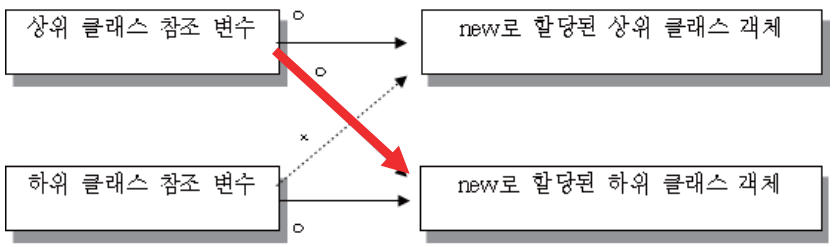
1. 메서드에 static과 abstract를 함께 사용할 수 없다.
   - static메서드는 몸통(구현부)이 있는 메서드에만 사용할 수 있기 때문이다.
2. 클래스에 abstract와 final을 동시에 사용할 수 없다.
   - 클래스에 사용되는 final은 클래스를 확장할 수 없다는 의미이고, abstract는 상속을 통해서 완성되어야 한다는 의미이므로 서로 모순되기 때문이다.
3. abstract메서드의 접근제어자가 private일 수 없다.
   - abstract메서드는 자손클래스에서 구현해주어야 하는데 접근 제어자가 private이면, 자손클래스에서 접근할 수 없기 때문이다.
4. 메서드에 private과 final을 같이 사용할 필요는 없다.
   - 접근 제어자가 private인 메서드는 오버라이딩될 수 없기 때문이다. 이 둘 중 하나만 사용해도 의미가 충분하다.

5. 다형성(polymorphism)

5.1 다형성(polymorphism)이란?

- 한 이름으로 다른 메서드를 참조할 수 있는 것
- 다형성을 통하여 프로그램은 유연성 있게 확장될 수 있다.
- 자바에는 세 가지 형태의 다형성이 존재한다: Overloading, Overriding, Dynamic method binding
- 교과서에 나오는 다형성 설명은 잘 못된 것임

▶ 상위 클래스 변수의 하위 클래스 객체 참조



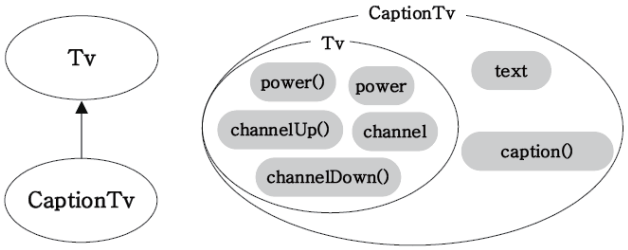
5.1 상위 클래스 변수의 하위 클래스 객체 참조

- 계층관계에 있는 클래스간에서 상위 클래스 참조변수를 사용하여 하위 클래스 객체를 참조하는 것이 가능하다.

-"하나의 참조변수로 여러 타입의 객체를 참조할 수 있는 것"

즉, 조상타입의 참조변수로 자손타입의 객체를 다룰 수 있다.

```
class Tv {
    boolean power;  // 전원상태(on/off)
    int channel;    // 채널

    void power(){   power = !power;}
    void channelUp(){   ++channel; }
    void channelDown(){ --channel; }
}

class CaptionTv extends Tv {
    String text;    // 캡션내용
    void caption() { /* 내용생략 */}
}
```




```
Tv        t = new Tv();
CaptionTv c = new CaptionTv();
```

```
Tv        t = new CaptionTv();
```

```
CaptionTv c = new CaptionTv();

Tv        t = new CaptionTv();
```

"하나의 참조변수로 여러 타입의 객체를 참조할 수 있는 것"

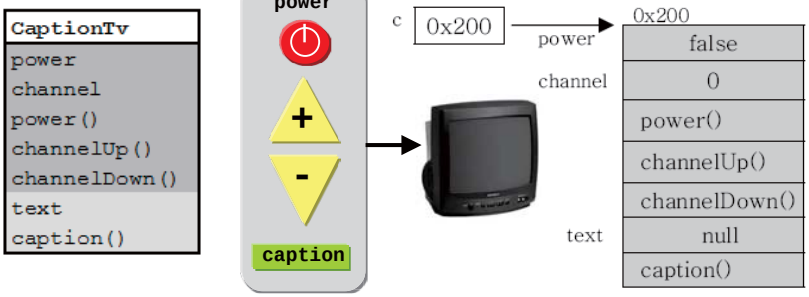
즉, 조상타입의 참조변수로 자손타입의 객체를 다룰 수 있다.

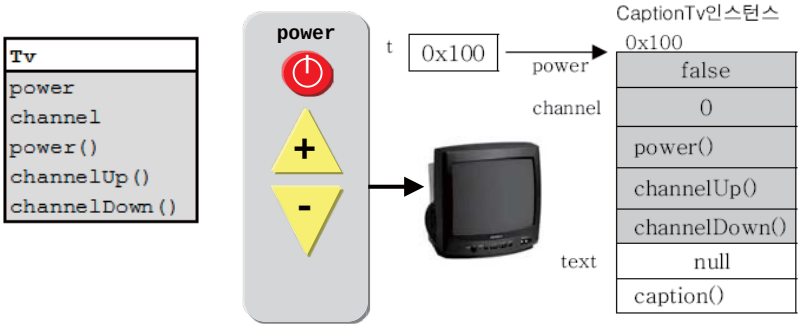
다른 말로, 리모컨 하나로 텔레비전, 에어컨, 오디오 다 조작하고 싶다.

```
CaptionTv c = new CaptionTv();

Tv        t = new CaptionTv();
```

Tv타입의 객체참조변수로는 CaptionTv클래스의 모든 멤버 사용 못함!

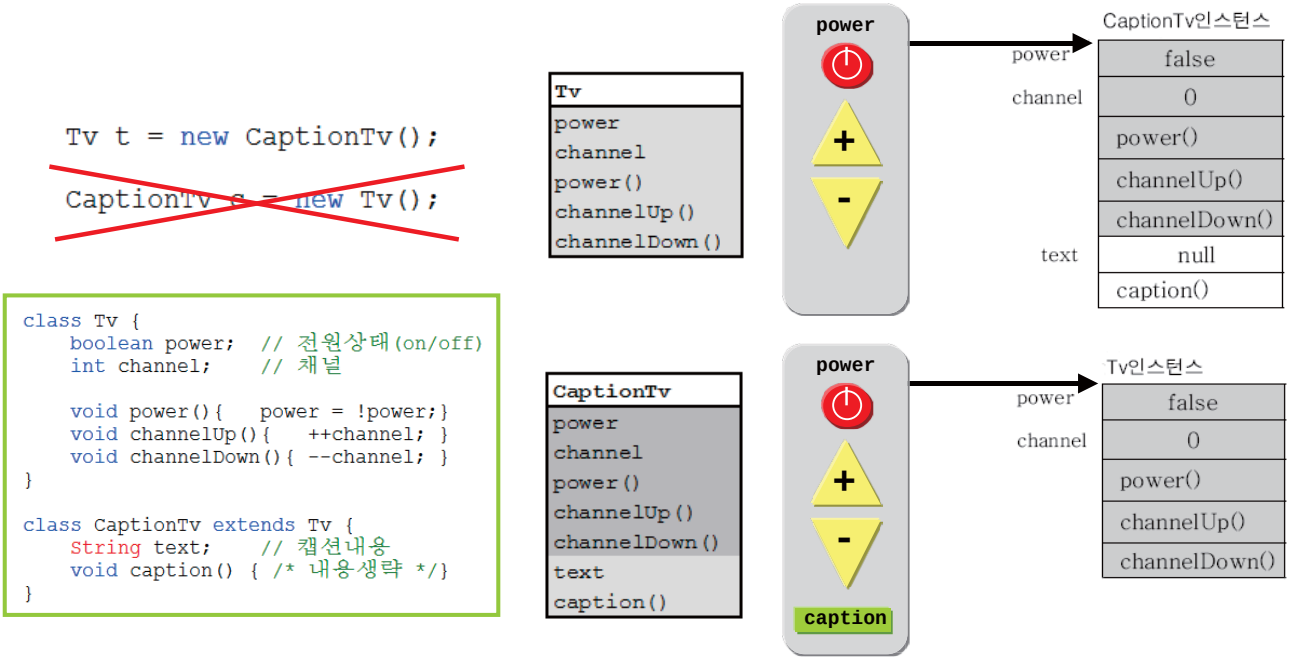
→왜 쓸까? 하나의 그릇에 여러 물건 담기 위함






```
class Tv {
    boolean power;  // 전원상태(on/off)
    int channel;    // 채널

    void power(){   power = !power;}
    void channelUp(){   ++channel; }
    void channelDown(){ --channel; }
}

class CaptionTv extends Tv {
    String text;    // 캡션내용
    void caption() { /* 내용생략 */}
}
```

"조상타입의 참조변수로 자손타입의 인스턴스를 참조할 수 있지만,
반대로 자손타입의 참조변수로 조상타입의 인스턴스를 참조할 수는 없다."



예제

```
class Tv {
	boolean power;		// 전원상태(on/off)
	int channel;		// 채널

	void power() {		power = !power; }
	void channelUp() {		++channel; }
	void channelDown() {		--channel;		}
}

class CaptionTv extends Tv {
	boolean caption;	// 캡션상태(on/off)
	void displayCaption(String text) {
		if (caption) {		// 캡션 상태가 on(true)일 때만 text를 보여 준다.
			System.out.println(text);
		}
	}
}

class CaptionTvTest {
	public static void main(String args[]) {
		CaptionTv ctv = new CaptionTv();
		ctv.channel = 10;				// 조상 클래스로부터 상속받은 멤버
		ctv.channelUp();				// 조상 클래스로부터 상속받은 멤버
		System.out.println(ctv.channel);
		ctv.displayCaption("Hello, World");
		ctv.caption = true;				// 캡션기능을 켠다.
		ctv.displayCaption("Hello, World");	// 캡션을 화면에 보여 준다.
	}
}
```

- Dynamic method binding: 자바에서 오버라이딩된 메서드를 실행시간에 바인딩하여 실행

예제

```
class Super1 {
  void f() {
    System.out.println("Super1's f() method");
  }
}
class Super2 extends Super1 {
  void f() {
    System.out.println("Super2's f() method");
  }
}
class Super3 extends Super2 {
  void f() {
    System.out.println("Super3's f() method");
  }
}
class Test {
  public static void main(String[] args) {
1:    Super1 s1 = new Super1();
2:    s1.f();      // Super1 클래스의 메소드 f() 호출
3:    s1 = new Super2();
4:    s1.f();      // Super2 클래스의 메소드 f() 호출
5:    s1 = new Super3();
6:    s1.f();      // Super3 클래스의 메소드 f() 호출
  }
}
```

실행결과

```
Super1's f() method
Super2's f() method
Super3's f() method
```

Test 의 내용을 살펴보면 s1이라는 이름의 Super1형의 객체가 세 개가 생성되어 있는 것을 볼 수 있다.

하지만 실제로 1번째 줄의 s1은 Super1의 인스턴스를, 세 번째 줄의 s1은 Super2의 인스턴스를, 다섯 번째 줄의 s1은 Super3의 인스턴스를 나타내고 있다.

그리고 두 번째 줄, 네 번 째 줄과 여섯 번째 줄에서 같은 이름의 f()를 호출하고 있다.

f()가 오버라이딩 되어있기 때문에 두 번째 줄, 네 번 째 줄과 여섯 번째 줄의 f()가 어떤 클래스의 f()를 나타내는지는 컴파일시에 결정되지 않는다.

f()가 어떤 클래스의 메서드를 나타내는지는 실행시간, 즉 Test 클래스 파일이 실행하는 시점에서 결정이 되며, 이것이 동적바인딩이라는 개념이다.

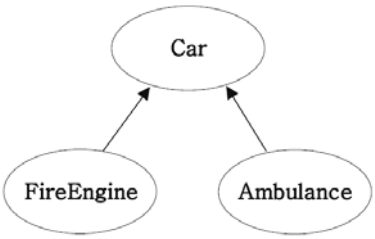
그러나, 모든 인스턴스 메서드는 런타임시에 결정되지만 정적(static) 메소드는 컴파일 시에 결정이 된다. 인스턴스 변수 또한 컴파일시에 결정된다.

5.2 참조변수의 형변환

- 서로 상속관계에 있는 타입간의 형변환만 가능하다.
- 자손 타입에서 조상타입으로 형변환하는 경우, 형변환 생략가능

자손타입 → 조상타입 (Up-casting) : 형변환 생략가능
자손타입 ← 조상타입 (Down-casting) : 형변환 생략불가

```
class Car {
    String color;
    int door;

    void drive() {  // 운전하는 기능
        System.out.println("drive, Brrrr~");
    }

    void stop() {   // 멈추는 기능
        System.out.println("stop!!!");
    }
}

class FireEngine extends Car {  // 소방차
    void water() {  // 물뿌리는 기능
        System.out.println("Water!!!");
    }
}

class Ambulance extends Car {   // 구급차
    void siren() {  // 사이렌을 울리는 기능
        System.out.println("siren~~~");
    }
}
```




```
FireEngine f
Ambulance a;

a = (Ambulance)f;
f = (FireEngine)a;
```

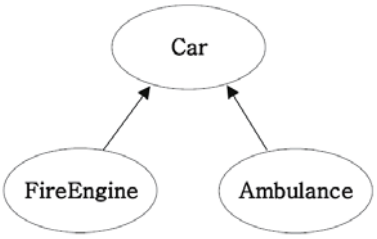
컴파일 에러: FireEngine과 Ambulance간에는 형변환 할 수 없다.

```
Car c1=(Car)f;
```
생략가능

```
FireEngine f2=(FireEngine) Car
```
생략불가

5.2 참조변수의 형변환 - 예제설명

```
class Car {
    String color;
    int door;

    void drive() {  // 운전하는 기능
        System.out.println("drive, Brrrr~");
    }

    void stop() {   // 멈추는 기능
        System.out.println("stop!!!");
    }
}

class FireEngine extends Car {  // 소방차
    void water() {  // 물뿌리는 기능
        System.out.println("water!!!");
    }
}

class Ambulance extends Car {   // 구급차
    void siren() {  // 사이렌을 울리는 기능
        System.out.println("siren~~~");
    }
}
```




```
public static void main(String args[]) {
    Car car = null;
    FireEngine fe = new FireEngine();
    FireEngine fe2 = null;

    fe.water();
    car = fe;   // car = (Car)fe; 조상 <- 자손
//  car.water();
    fe2 = (FireEngine)car; // 자손 <- 조상
    fe2.water();
}
```

예제 7-14

```
class CastingTest1 {
        public static void main(String args[]) {
                Car car = null;
                FireEngine fe = new FireEngine();
                FireEngine fe2 = null;

                fe.water();
                car = fe;     // car =(Car)fe;에서 형변환이 생략된 형태다.
//              car.water();  // 컴파일 에러!!! Car타입의 참조변수로는 water()를 호출할 수 없다.
                fe2 = (FireEngine)car; // 자손타입 ← 조상타입
                fe2.water();
        }
}

class Car {
        String color;
        int door;

        void drive() {                  // 운전하는 기능
                System.out.println("drive, Brrrr~");
        }

        void stop() {                   // 멈추는 기능
                System.out.println("stop!!!");
        }
}

class FireEngine extends Car {          // 소방차
        void water() {                  // 물을 뿌리는 기능
                System.out.println("water!!!");
        }
}
```

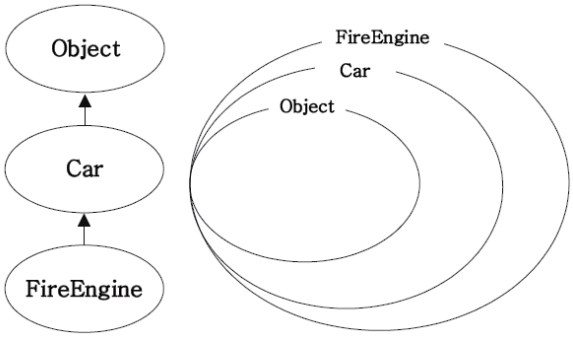
실행결과

```
water!!!
water!!!
```

5.3 instanceof연산자

- 참조변수가 참조하는 인스턴스의 실제 타입을 체크하는데 사용.
- 이항연산자이며 피연산자는 참조형 변수와 타입. 연산결과는 true, false.
- instanceof의 연산결과가 true이면, 해당 타입으로 형변환이 가능하다.

```
class InstanceofTest {
    public static void main(String args[]) {
        FireEngine fe = new FireEngine();

        if(fe instanceof FireEngine) {
            System.out.println("This is a FireEngine instance.");
        }

        if(fe instanceof Car) {
            System.out.println("This is a Car instance.");
        }

        if(fe instanceof Object) {
            System.out.println("This is an Object instance.");
        }
    }
}
```




```
---------- java ----------
This is a FireEngine instance.
This is a Car instance.
This is an Object instance.

출력 완료 (0초 경과)
```

```
void method(Object obj) {
    if(c instanceof Car) {
        Car c = (Car)obj;
        c.drive();
    } else if(c instanceof FireEngine) {
        FireEngine fe = (FireEngine)obj;
        fe.water();
    }
}
```

5.4 참조변수와 인스턴스변수의 연결

- 멤버변수가 중복정의된 경우, 참조변수의 타입에 따라 연결되는 멤버변수가 달라진다. (참조변수타입에 영향받음)
- 메서드가 중복정의된 경우, 참조변수의 타입에 관계없이 항상 실제 인스턴스의 타입에 정의된 메서드가 호출된다.(참조변수타입에 영향받지 않음)

```
class Parent {
    int x = 100;

    void method() {
        System.out.println("Parent Method");
    }
}

class Child extends Parent {
    int x = 200;

    void method() {
        System.out.println("Child Method");
    }
}
```

```
p.x = 100
Child Method
c.x = 200
Child Method
```

```
class Parent {
    int x = 100;

    void method() {
        System.out.println("Parent Method");
    }
}

class Child extends Parent { }
```

```
p.x = 100
Parent Method
c.x = 100
Parent Method
```

```
public static void main(String[] args) {
    Parent p = new Child();
    Child c = new Child();

    System.out.println("p.x = " + p.x);
    p.method();

    System.out.println("c.x = " + c.x);
    c.method();
}
```

5.5 매개변수의 다형성

- 참조형 매개변수는 메서드 호출시, 자신과 같은 타입 또는 자손타입의 인스턴스를 넘겨줄 수 있다.

```
class Product {
    int price;      // 제품가격
    int bonusPoint; // 보너스점수
}

class Tv extends Product {}
class Computer extends Product {}
class Audio extends Product {}

class Buyer { // 물건사는 사람
    int money = 1000;   // 소유금액
    int bonusPoint = 0; // 보너스점수
}
```

```
Buyer b = new Buyer();

Tv tv = new Tv();
Computer com = new Computer();

b.buy(tv);
b.buy(com);
```

```
Product p1 = new Tv();
Product p2 = new Computer();
Product p3 = new Audio();
```

```
void buy(Tv t) {
    money -= t.price;
    bonusPoint += t.bonusPoint;
}
```

→

```
void buy(Product p) {
    money -= p.price;
    bonusPoint += p.bonusPoint;
}
```

함수 정의는 void buy(Product p); -> 뒤의 예제 참조

예제 7-20 (1)

```
class Product
{
        int price;                      // 제품의 가격
        int bonusPoint;                 // 제품구매 시 제공하는 보너스점수

        Product(int price) {
                this.price = price;
                bonusPoint =(int)(price/10.0);          // 보너스점수는 제품가격의 10%
        }
}

class Tv extends Product {
        Tv() {
                // 조상클래스의 생성자 Product(int price)를 호출한다.
                super(100);                             // Tv의 가격을 100만원으로 한다.
        }

        public String toString() {    // Object클래스의 toString()을 오버라이딩한다.
                return "Tv";
        }
}

class Computer extends Product {
        Computer() {
                super(200);
        }

        public String toString() {
                return "Computer";
        }
}
```

예제 7-20 (2)

```
class Buyer {                           // 고객, 물건을 사는 사람
          int money = 1000;             // 소유금액
          int bonusPoint = 0; // 보너스점수

          void buy(Product p) {
                    if(money < p.price) {
                              System.out.println("잔액이 부족하여 물건을 살수 없습니다.");
                              return;
                    }

                    money -= p.price;                       // 가진 돈에서 구입한 제품의 가격을 뺀
다.
                    bonusPoint += p.bonusPoint;   // 제품의 보너스 점수를 추가한다.
                    System.out.println(p + "을/를 구입하셨습니다.");
          }
}

class PolyArgumentTest {
          public static void main(String args[]) {
                    Buyer b = new Buyer();
                    Tv tv = new Tv();
                    Computer com = new Computer();

                    b.buy(tv);
                    b.buy(com);

                    System.out.println("현재 남은 돈은 " + b.money + "만원입니다.");
                    System.out.println("현재 보너스점수는 " + b.bonusPoint + "점입니다.");
          }
}
```

실행결과

```
Tv을/를 구입하셨습니다.
Computer을/를 구입하셨습니다.
현재 남은 돈은 700만원입니다.
현재 보너스점수는 30점입니다.
```

5.6 여러 종류의 객체를 하나의 배열로 다루기

- 조상타입의 배열에 자손들의 객체를 담을 수 있다.

```
Product p1 = new Tv();
Product p2 = new Computer();
Product p3 = new Audio();
```

→

```
Product p[] = new Product[3];
p[0] = new Tv();
p[1] = new Computer();
p[2] = new Audio();
```

```
class Buyer { // 물건사는 사람
    int money = 1000;   // 소유금액
    int bonusPoint = 0; // 보너스점수

    Product[] cart = new Product[10]; // 구입한 물건을 담을 배열

    int i=0;

    void buy(Product p) {
        if(money < p.price) {
            System.out.println("잔액부족");
            return;
        }

        money -= p.price;
        bonusPoint += p.bonusPoint;
        cart[i++] = p;
    }
}
```

예제 7-21 (1)

```
class Product
{
	int price;				// 제품의 가격
	int bonusPoint;			// 제품구매 시 제공하는 보너스점수

	Product(int price) {
		this.price = price;
		bonusPoint =(int)(price/10.0);
	}

	Product() {
		price = 0;
		bonusPoint = 0;
	}
}

class Tv extends Product {
	Tv() {
		super(100);
	}

	public String toString() {
		return "Tv";
	}
}

class Computer extends Product {
	Computer() {
		super(200);
	}

	public String toString() {
		return "Computer";
	}
}

class Audio extends Product {
	Audio() {
		super(50);
	}

	public String toString() {
		return "Audio";
	}
}
```

예제 7-21 (2)

```
class Buyer {				// 고객, 물건을 사는 사람
	int money = 1000;	// 소유금액
	int bonusPoint = 0;	// 보너스점수
	Product[] item = new Product[10];		// 구입한 제품을 저장하기 위한 배열
	int i =0;				// Product배열에 사용될 카운터

	void buy(Product p) {
		if(money < p.price) {
			System.out.println("잔액이 부족하여 물건을 살수 없습니다.");
			return;
		}

		money -= p.price;			// 가진 돈에서 구입한 제품의 가격을 뺀다.
		bonusPoint += p.bonusPoint;	// 제품의 보너스 점수를 추가한다.
		item[i++] = p;				// 제품을 Product[] item에 저장한다.
		System.out.println(p + "을/를 구입하셨습니다.");
	}

	void summary() {			// 구매한 물품에 대한 정보를 요약해서 보여 준다.
		int sum = 0;				// 구입한 물품의 가격합계
		String itemList ="";  // 구입한 물품목록

		// 반복문을 이용해서 구입한 물품의 총 가격과 목록을 만든다.
		for(int i=0; i<item.length;i++) {
			if(item[i]==null) break;
			sum += item[i].price;
			itemList += item[i] + ", ";
		}

		System.out.println("구입하신 물품의 총금액은 " + sum + "만원입니다.");
		System.out.println("구입하신 제품은 " + itemList + "입니다.");
	}
}

class PolyArgumentTest2 {
	public static void main(String args[]) {
		Buyer b = new Buyer();
		Tv tv = new Tv();
		Computer com = new Computer();
		Audio audio = new Audio();
		b.buy(tv);
		b.buy(com);
		b.buy(audio);
		b.summary();
	}
}
```

실행결과

```
Tv을/를 구입하셨습니다.
Computer을/를 구입하셨습니다.
Audio을/를 구입하셨습니다.
구입하신 물품의 총금액은 350만원입니다.
구입하신 제품은 Tv, Computer, Audio, 입니다.
```

▶ java.util.Vector – 모든 종류의 객체들을 저장할 수 있는 클래스

메서드 / 생성자	설 명
Vector()	10개의 객체를 저장할 수 있는 Vector인스턴스를 생성한다. 10개 이상의 인스턴스가 저장되면, 자동적으로 크기가 증가된다.
boolean add(Object o)	Vector에 객체를 추가한다. 추가에 성공하면 결과값으로 true, 실패하면 false를 반환한다.
boolean remove(Object o)	Vector에 저장되어 있는 객체를 제거한다. 제거에 성공하면 true, 실패하면 false를 반환한다.
boolean isEmpty()	Vector가 비어있는지 검사한다. 비어있으면 true, 비어있지 않으면 false를 반환한다.
Object get(int index)	지정된 위치(index)의 객체를 반환한다. 반환타입이 Object타입이므로 적절한 타입으로의 형변환이 필요하다.
int size()	Vector에 저장된 객체의 개수를 반환한다.

```
public class Vector extends AbstractList implements List, Cloneable,
                    java.io.Serializable {
    protected Object elementData[];
    ...
}
```

```
Product[] cart = new Product[10];
//...

void buy(Product p) {
    //...
    cart[i++] = p;
}
```

→

```
Vector cart = new Vector();
//...

void buy(Product p) {
    //...
    cart.add(p);
}
```

메서드 / 생성자
Vector()
boolean add(Object o)
boolean remove(Object o)
boolean isEmpty()
Object get(int index)
int size()

```
void summary() {                    // 구매한 물품에 대한 정보를 요약해서 보여준다.
    int sum = 0;                    // 구입한 물품의 가격합계
    String cartList ="";            // 구입한 물품목록

    if(cart.isEmpty()) {            //
        System.out.println("구입
        return;
    }

    // 반복문을 이용해서 구입한 물품의 총 가격과 목록을 만든다.
    for(int i=0; i<cart.size();i++) {
        Product p = (Product)cart.get(i);
        sum += p.price;
        cartList += (i==0) ? "" + p : ", " + p;
    }
    System.out.println("구입하신 물품의 총금액은 " + sum + "만원입니다.");
    System.out.println("구입하신 제품은 " + cartList + "입니다.");
}
```

```
class Tv extends Product {
    Tv() { super(100); }
    public String toString() { return "Tv"; }
}
```

```
Object obj = cart.get(i);
sum += obj.price; // 에러
```

예제 7-22 (1)

```
import java.util.*;                    // Vector클래스를 사용하기 위해서 추가해주었다.

class Tv extends Product {
        Tv() { super(100); }
        public String toString() { return "Tv"; }
}

class Computer extends Product {
        Computer() { super(200); }
        public String toString() { return "Computer"; }
}

class Audio extends Product {
        Audio() { super(50); }
        public String toString() { return "Audio"; }
}

class Buyer {                  // 고객, 물건을 사는 사람
        int money = 1000;      // 소유금액
        int bonusPoint = 0; // 보너스점수
        Vector item = new Vector();   // 구입한 제품을 저장하는데 사용될 Vector객체

        void buy(Product p) {
                if(money < p.price) {
                        System.out.println("잔액이 부족하여 물건을 살수 없습니다.");
                        return;
                }
                money -= p.price;              // 가진 돈에서 구입한 제품의 가격을 뺀다.
                bonusPoint += p.bonusPoint;    // 제품의 보너스 점수를 추가한다.
                item.add(p);                   // 구입한 제품을 Vector에 저장한다.
                System.out.println(p + "을/를 구입하셨습니다.");
        }

        void refund(Product p) {               // 구입한 제품을 환불한다.
                if(item.remove(p)) {           // 제품을 Vector에서 제거한다.
                        money += p.price;
                        bonusPoint -= p.bonusPoint;
                        System.out.println(p + "을/를 반품하셨습니다.");
                } else {                       //  제거에 실패한 경우
                        System.out.println("구입하신 제품 중 해당 제품이 없습니다.");

                }
        }
```

예제 7-22 (2)

```
        void summary() {                       // 구매한 물품에 대한 정보를 요약해서 보여준다.
                int sum = 0;                   // 구입한 물품의 가격합계
                String itemList ="";           // 구입한 물품목록
                // 반복문을 이용해서 구입한 물품의 총 가격과 목록을 만든다.

                if(item.isEmpty()) {           // Vector가 비어있는지 확인한다.
                        System.out.println("구입하신 제품이 없습니다.");
                        return;
                }
                // Vector의 i번째에 있는 객체를 얻어 온다.
                for(int i=0; i<item.size();i++) {
                        Product p = (Product)item.get(i);
                        sum += p.price;
                        itemList += (i==0) ? "" + p : ", " + p;
                }
                System.out.println("구입하신 물품의 총금액은 " + sum + "만원입니다.");
                System.out.println("구입하신 제품은 " + itemList + "입니다.");
        }
}

class PolyArgumentTest3 {
        public static void main(String args[]) {
                Buyer b = new Buyer();
                Tv tv = new Tv();
                Computer com = new Computer();
                Audio audio = new Audio();

                b.buy(tv);
                b.buy(com);
                b.buy(audio);
                b.summary();
                System.out.println();
                b.refund(com);
                b.summary();
        }
}
```

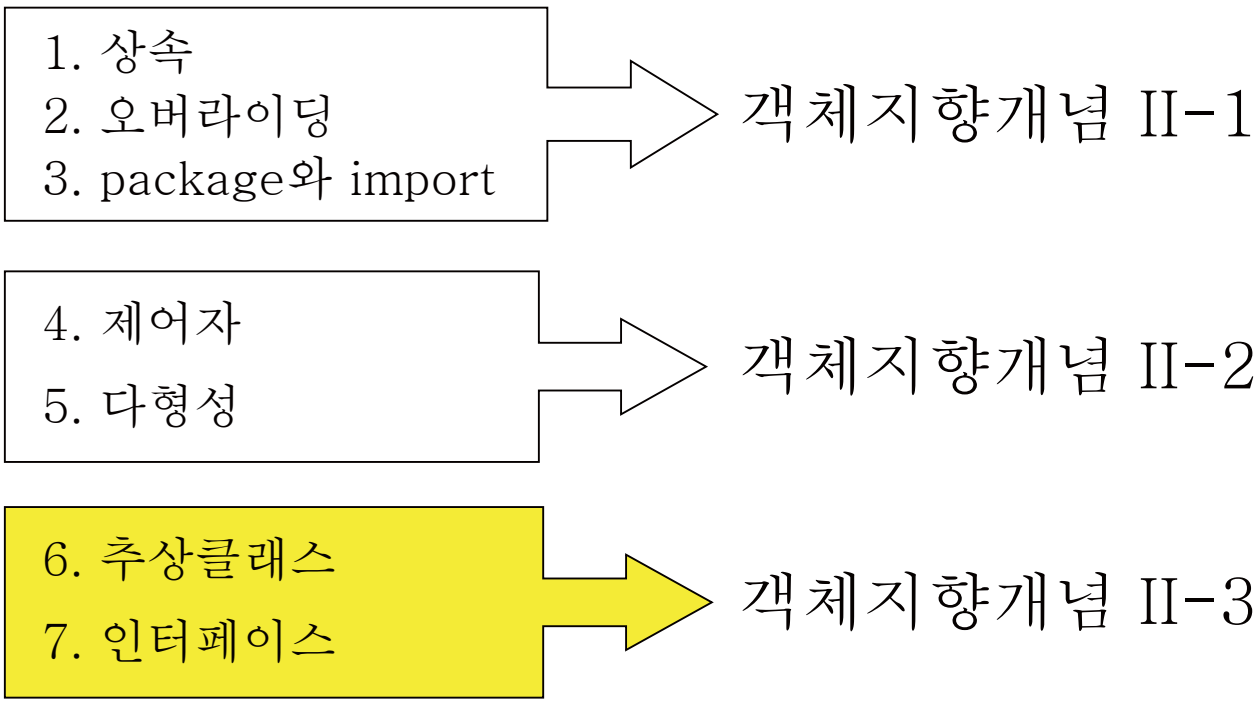
실행결과

```
Tv을/를 구입하셨습니다.
Computer을/를 구입하셨습니다.
Audio을/를 구입하셨습니다.
구입하신 물품의 총금액은 350만원입니다.
구입하신 제품은 Tv, Computer, Audio입니다.

Computer을/를 반품하셨습니다.
구입하신 물품의 총금액은 150만원입니다.
구입하신 제품은 Tv, Audio입니다.
```

제 7 장
객체지향개념 II-3

인제대학교 전자IT기계자동차공학부

황 원 주



6. 추상클래스 (abstract class)

6.1 추상클래스(abstract class)란?

- 하위 클래스에서 구현할 메서드의 선언부만을 정의하고 구체적인 구현부를 생략한 클래스
- 클래스가 설계도라면 추상클래스는 '미완성 설계도'
- 추상클래스는 반드시 하나 이상의 추상메서드(미완성 메서드)를 포함하고 있어야 한다.

  * 추상메서드 : 선언부만 있고 구현부(몸통, body)가 없는 메서드

```
abstract class Player {
     int currentPos;          // 현재 Play되고 있는 위치를 저장하기 위한 변수

     Player() {               // 추상클래스도 생성자가 있어야 한다.
          currentPos = 0;
     }

     abstract void play(int pos);      // 추상메서드
     abstract void stop();             // 추상메서드

     void play() {
          play(currentPos);            // 추상메서드를 사용할 수 있다.
     }
     ...
}
```

- 일반메서드가 추상메서드를 호출할 수 있다.(호출할 때 필요한 건 선언부)
- 완성된 설계도가 아니므로 인스턴스를 생성할 수 없다.
- 다른 클래스를 작성하는 데 도움을 줄 목적으로 작성된다.

6.2 추상메서드(abstract method)란?

- 선언부만 있고 구현부(몸통, body)가 없는 메서드

```
/* 주석을 통해 어떤 기능을 수행할 목적으로 작성하였는지 설명한다. */
abstract 리턴타입 메서드이름();

Ex)
/* 지정된 위치(pos)에서 재생을 시작하는 기능이 수행되도록 작성한다.*/
abstract void play(int pos);
```

- 꼭 필요하지만 자손마다 다르게 구현될 것으로 예상되는 경우에 사용
- 추상클래스를 상속받는 자손클래스에서는 추상메서드의 구현부를 오버라이딩을 통하여 완성해야 한다.

```
abstract class Player {
     ...
     abstract void play(int pos);      // 추상메서드
     abstract void stop();             // 추상메서드
     ...
}

class AudioPlayer extends Player {
     void play(int pos) { /* 내용 생략 */ }
     void stop() { /* 내용 생략 */ }
}

abstract class AbstractPlayer extends Player {
     void play(int pos) { /* 내용 생략 */ }
}
```

6.3 추상클래스의 작성

- 여러 클래스에 공통적으로 사용될 수 있는 추상클래스를 바로 작성하거나 기존클래스의 공통 부분을 뽑아서 추상클래스를 만든다.

```
class Marine {    // 보병
    int x, y;     // 현재 위치
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stop()             { /* 현재 위치에 정지 */ }
    void stimPack()         { /* 스팀팩을 사용한다.*/}
}

class Tank {      // 탱크
    int x, y;     // 현재 위치
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stop()             { /* 현재 위치에 정지 */ }
    void changeMode()       { /* 공격모드를 변환한다. */}
}

class Dropship {  // 수송선
    int x, y;     // 현재 위치
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stop()             { /* 현재 위치에 정지 */ }
    void load()             { /* 선택된 대상을 태운다.*/ }
    void unload()           { /* 선택된 대상을 내린다.*/ }
}
```

```
abstract class Unit {
    int x, y;
    abstract void move(int x, int y);
    void stop() {      /* 현재 위치에 정지 */ }
}

class Marine extends Unit {  // 보병
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stimPack()         { /* 스팀팩을 사용한다.*/}
}

class Tank extends Unit {    // 탱크
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void changeMode()       { /* 공격모드를 변환한다. */}
}

class Dropship extends Unit { // 수송선
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void load()             { /* 선택된 대상을 태운다.*/ }
    void unload()           { /* 선택된 대상을 내린다.*/ }
}
```

```
Unit[] group = new Unit[4];
group[0] = new Marine();
group[1] = new Tank();
group[2] = new Marine();
group[3] = new Dropship();

for(int i=0;i< group.length;i++) {
    group[i].move(100, 200);
}
```

추상메서드가 호출되는 것이 아니라 각 자손들에 실제로 구현된 move(int x, int y)가 호출된다.

7. 인터페이스(interface)

7.1 인터페이스(interface)란?

- 일종의 추상클래스. 추상클래스(미완성 설계도)보다 추상화 정도가 높다.
- 실제 구현된 것이 전혀 없는 기본 설계도.(알맹이 없는 껍데기)
- 추상메서드와 상수만을 멤버로 가질 수 있다.
- 인스턴스를 생성할 수 없고, 클래스 작성에 도움을 줄 목적으로 사용된다.
- 미리 정해진 규칙에 맞게 구현하도록 표준을 제시하는 데 사용된다.

7.2 인터페이스의 작성

- 'class'대신 'interface'를 사용한다는 것 외에는 클래스 작성과 동일하다.

```
interface 인터페이스이름 {
    public static final 타입 상수이름 = 값;
    public abstract 메서드이름(매개변수목록);
}
```

- 하지만, 구성요소(멤버)는 추상메서드와 상수만 가능하다.

- 모든 멤버변수는 public static final 이어야 하며, 이를 생략할 수 있다.
- 모든 메서드는 public abstract 이어야 하며, 이를 생략할 수 있다.

```
interface PlayingCard {
    public static final int SPADE = 4;
    final int DIAMOND = 3;      // public static final int DIAMOND = 3;
    static int HEART = 2;       // public static final int HEART = 2;
    int CLOVER = 1;             // public static final int CLOVER = 1;

    public abstract String getCardNumber();
    String getCardKind();  // public abstract String getCardKind();
}
```

7.3 인터페이스의 구현

- 인터페이스를 구현하는 것은 클래스를 상속받는 것과 같다.
  다만, 'extends' 대신 'implements'를 사용한다.

```
class 클래스이름 implements 인터페이스이름 {
      // 인터페이스에 정의된 추상메서드를 구현해야한다.
}
```

- 인터페이스에 정의된 추상메서드를 완성해야 한다.

예제

```
interface Animal {        // 인터페이스 Animal 정의
  void make_descendant();  // 추상 메소드
}
class Bird implements Animal {
  public void make_descendant() { // 재정의된 메소드
    System.out.println("Bird lay egg");
  }
}
class Mammal implements Animal {
   public void make_descendant() { // 재정의된 메소드
     System.out.println("Mammal lay children");
   }
}
class InterfaceTest {
   public static void main(String[] arg) {
     Bird b = new Bird();
     Mammal m = new Mammal();
     b.make_descendant();
     m.make_descendant();
   }
}
```

실행결과

```
Bird lay egg
Mammal lay children
```

7.4 인터페이스의 상속

- 인터페이스도 클래스처럼 상속이 가능하다.(클래스와 달리 다중상속 허용)

예제

```
interface Bird {
              void fly();
}
interface Mammal {
              void produce_milk();
}
class Bat implements Bird, Mammal {
              public void live_in() { System.out.println("Bat lives in Cave"); }
              public void produce_milk() { System.out.println("Bat breeds with milk"); }
              public void fly() { System.out.println("Bat can fly"); }
}

class MultipleInterface {
              public static void main(String[] arg) {
                            Bat b = new Bat();
                            b.fly();
                            b.live_in();
                            b.produce_milk();
              }
}
```

실행결과

```
Bat can fly
Bat lives in Cave
Bat breeds with milk.
```

- 상속과 구현이 동시에 가능하다.

```
class Fighter extends Unit implements Fightable {
      public void move(int x, int y) { /* 내용 생략 */}
      public void attack(Unit u) { /* 내용 생략 */}
}
```

7.5 인터페이스를 이용한 다형성

- 인터페이스 타입의 변수로 인터페이스를 구현한 클래스의 인스턴스를 참조할 수 있다.

```
class Fighter extends Unit implements Fightable {
     public void move(int x, int y)  { /* 내용 생략 */ }
     public void attack(Fightable f){ /* 내용 생략 */ }
}
```

```
Fighter   f = new Fighter();
Fightable f = new Fighter();
```

- 인터페이스를 메서드의 매개변수 타입으로 지정할 수 있다.

```
void attack(Fightable f) {  // Fightable인터페이스를 구현한 클래스의 인스턴스를
    //...                   // 매개변수로 받는 메서드
}
```

- 인터페이스를 메서드의 리턴타입으로 지정할 수 있다.

```
Fightable method() { // Fightable인터페이스를 구현한 클래스의 인스턴스를 반환
    // ...
    return new Fighter();
}
```

7.6 인터페이스의 장점

1. 개발시간을 단축시킬 수 있다.

일단 인터페이스가 작성되면, 이를 사용해서 프로그램을 작성하는 것이 가능하다. 메서드를 호출하는 쪽에서는 메서드의 내용에 관계없이 선언부만 알면 되기 때문이다.

그리고 동시에 다른 한 쪽에서는 인터페이스를 구현하는 클래스를 작성하도록 하여, 인터페이스를 구현하는 클래스가 작성될 때까지 기다리지 않고도 양쪽에서 동시에 개발을 진행할 수 있다.

2. 표준화가 가능하다.

프로젝트에 사용되는 기본 틀을 인터페이스로 작성한 다음, 개발자들에게 인터페이스를 구현하여 프로그램을 작성하도록 함으로써 보다 일관되고 정형화된 프로그램의 개발이 가능하다.

3. 서로 관계없는 클래스들에게 관계를 맺어 줄 수 있다.

서로 상속관계에 있지도 않고, 같은 조상클래스를 가지고 있지 않은 서로 아무런 관계도 없는 클래스들에게 하나의 인터페이스를 공통적으로 구현하도록 함으로써 관계를 맺어 줄 수 있다.

4. 독립적인 프로그래밍이 가능하다.

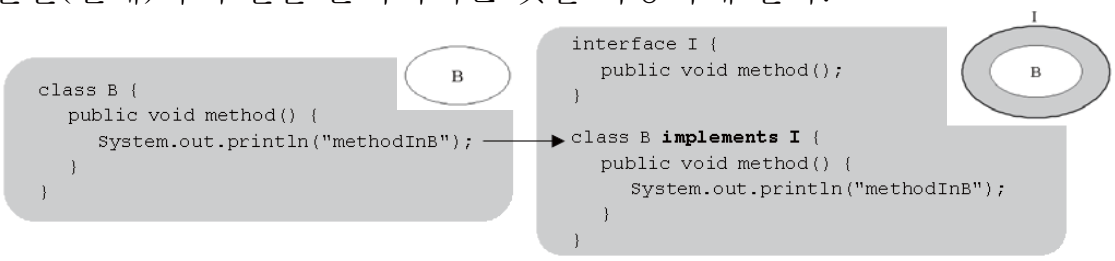
인터페이스를 이용하면 클래스의 선언과 구현을 분리시킬 수 있기 때문에 실제구현에 독립적인 프로그램을 작성하는 것이 가능하다.

클래스와 클래스간의 직접적인 관계를 인터페이스를 이용해서 간접적인 관계로 변경하면, 한 클래스의 변경이 관련된 다른 클래스에 영향을 미치지 않는 독립적인 프로그래밍이 가능하다.

7.7 인터페이스의 이해

▶ 인터페이스는...

- 두 대상(객체) 간의 '연결, 대화, 소통'을 돕는 '중간 역할'을 한다.
- 선언(설계)와 구현을 분리시키는 것을 가능하게 한다.



▶ 인터페이스를 이해하려면 먼저 두 가지를 기억하자.

- 클래스를 사용하는 쪽(User)과 클래스를 제공하는 쪽(Provider)이 있다.
- 메서드를 사용(호출)하는 쪽(User)에서는 사용하려는 메서드(Provider)의 선언부만 알면 된다.

▶ 직접적인 관계의 두 클래스(A-B)

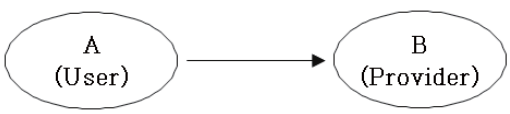
:클래스 A는 클래스 B에 직접 접근(→인스턴스를 생성하고 메서드를 호출)

```
class A {
	public void methodA(B b) {
		b.methodB();
	}
}
```

```
class B {
	public void methodB() {
		System.out.println("methodB()");
	}
}
```

```
class InterfaceTest {
	public static void main(String args[]) {
		A a = new A();
		a.methodA(new B());
	}
}
```

클래스 A는 인터페이스 I를 사용해서 작성되었으면, 매개변수를 통해서 인터페이스 I를 구현한 클래스의 인스턴스를 동적으로 제공받는다. (다형성: Dynamic Method Binding)



▶ 간접적인 관계의 두 클래스(A-I-B)

:클래스 A는 클래스 B에 직접 접근하지 않고 인터페이스를 통해서 간접 접근

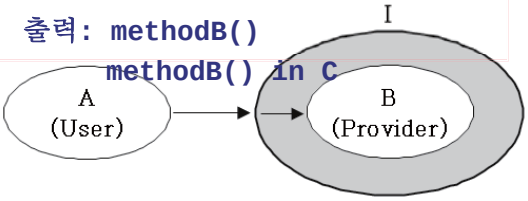
```
class A {
	public void methodA(I i) {
		i.methodB();
	}
}
```

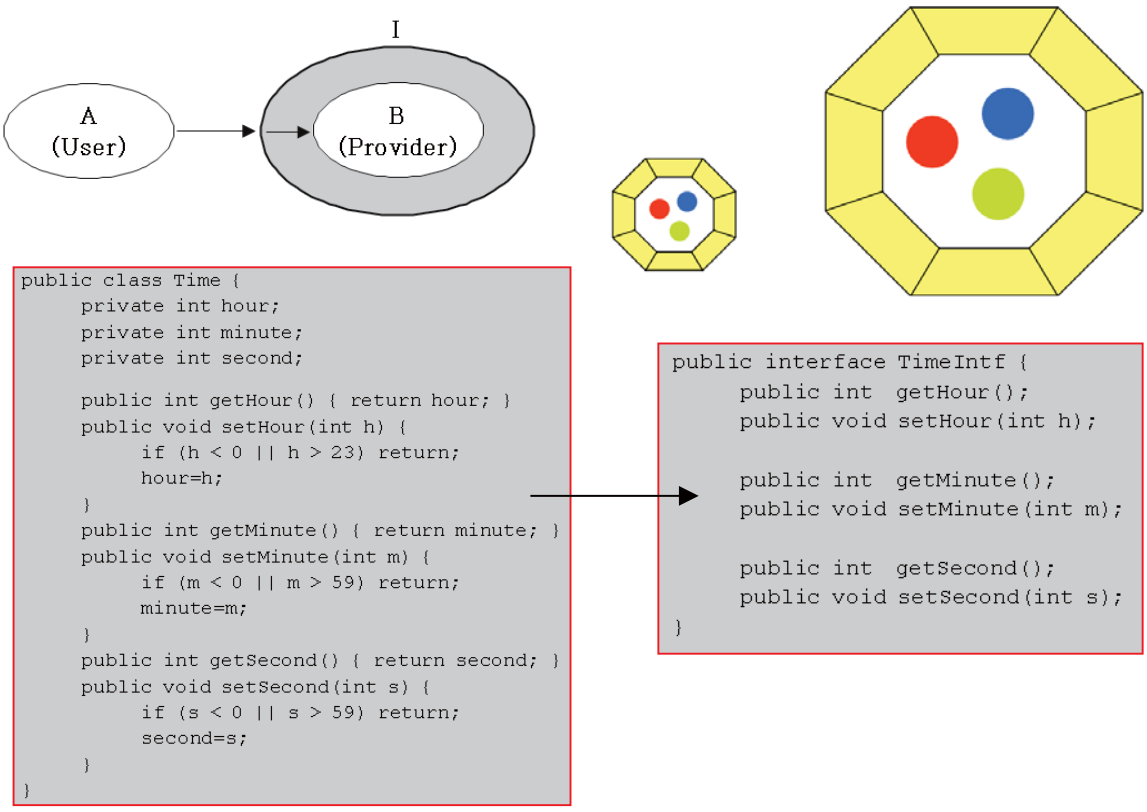
```
interface I { void methodB(); }

class B implements I {
	public void methodB() {
		System.out.println("methodB()");
	}
}
```

```
class C implements I {
	public void methodB() {
		System.out.println("methodB() in C");
	}
}
```

```
class InterfaceTest {
	public static void main(String args[]) {
		A a = new A();
		a.methodA(new B());
		a.methodA(new C());
	}
}
```

출력: methodB()
methodB() in C





※ 추상클래스 vs. 인터페이스

- 추상클래스나 인터페이스로부터 객체 생성하면 컴파일 시 에러 발생 (InstatiationException)
- 추상클래스와 인터페이스는 껍데기만 제공하므로 오버라이딩을 통한 재정의 필요

▶ 추상클래스

- 하나 이상의 추상메서드로 구성
- 구현방법: 키워드 extends을 이용하여 하위 클래스에서 오버라이딩을 이용하여 구현

▶ 인터페이스

- 추상메서드와 상수만으로 구성
- 구현방법: 키워드 implements를 이용하여 인터페이스를 사용한 클래스에서 오버라이딩을 이용하여 구현 (인터페이스에서 정의된 모든 메서드는 반드시 재정의되어야 한다.)

제 8 장
예외처리

인제대학교 전자IT기계자동차공학부

황 원 주

1. 예외처리(Exception handling)
    1.1 프로그램 오류
    1.2 예외처리의 정의와 목적
    1.3 예외처리구문 – try-catch
    1.4 try-catch문에서의 흐름
    1.5 예외 발생시키기
    1.6 예외클래스의 계층구조
    1.7 예외의 발생과 catch블럭
    1.8 finally블럭
    1.9 메서드에 예외 선언하기
    1.10 예외 되던지기(re-throwing)
    1.11 사용자정의 예외 만들기

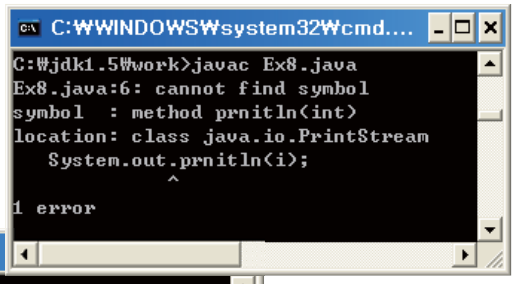
1. 예외처리(Exception handling)

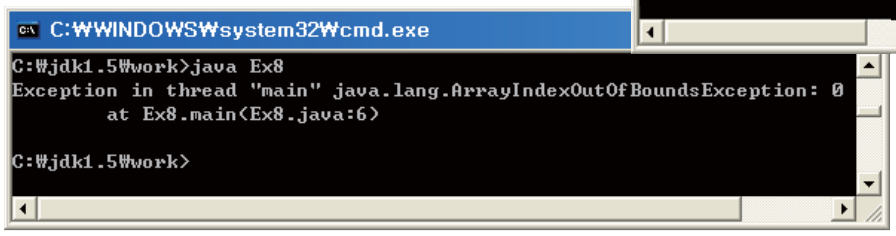
1.1 프로그램 오류

▶ 컴파일 에러(compile-time error)와 런타임 에러(runtime error)

. 컴파일 에러 – 컴파일할 때 발생하는 에러

. 런타임 에러 – 실행할 때 발생하는 에러





▶ Java의 런타임 에러 – 에러(error)와 예외(exception)

에러(error) - 프로그램 코드에 의해서 수습될 수 없는 심각한 오류

예외(exception) - 프로그램 코드에 의해서 수습될 수 있는 다소 미약한 오류

1.2 예외처리의 정의와 목적

▶ 에러(error)는 어쩔 수 없지만, 예외(exception)는 처리해야 한다.

- 컴파일시 소스코드의 문법적인 오류(잘못된 구문 이나 자료형 등)는 걸러낼 수 있으나, 논리적인 오류(무한 루프로 인한 동작 중단 등)은 걸러낼 수 없다.

```
에러(error) - 프로그램 코드에 의해서 수습될 수 없는 심각한 오류
예외(exception) - 프로그램 코드에 의해서 수습될 수 있는 다소 미약한 오류
```

▶ 예외처리의 정의와 목적

```
예외처리(exception handling)의
정의 - 프로그램 실행 시 발생할 수 있는 예외의 발생에 대비한 코드를 작성하는 것
목적 - 프로그램의 비정상 종료를 막고, 정상적인 실행상태를 유지하는 것
```

[참고] 에러와 예외는 모두 실행 시(runtime) 발생하는 오류이다.

5

1.3 예외처리구문 – try-catch

- 예외를 처리하려면 try-catch문을 사용해야 한다.

```
try {
    // 예외가 발생할 가능성이 있는 문장들을 넣는다.
} catch (Exception1 e1) {
    // Exception1이 발생했을 경우, 이를 처리하기 위한 문장을 적는다.
} catch (Exception2 e2) {
    // Exception2가 발생했을 경우, 이를 처리하기 위한 문장을 적는다.
...
} catch (ExceptionN eN) {
    // ExceptionN이 발생했을 경우, 이를 처리하기 위한 문장을 적는다.
}
```

[참고] if문과 달리 try블럭이나 catch블럭 내에 포함된 문장이 하나라고 해서 괄호{}를 생략할 수는 없다.

```
public static void main(String[] args)
{
    try  {
        try {    } catch (Exception e) {
            //...
        }
    } catch (Exception e) {
        try {    } catch (Exception e) { // 컴파일 에러 발생 !!!
            //...
        }
    } // try-catch의 끝
}   // main메서드의 끝
```

6

1.4 try-catch문에서의 흐름

▶ try블럭 내에서 예외가 발생한 경우,
1. 발생한 예외와 일치하는 catch블럭이 있는지 확인한다.
2. 일치하는 catch블럭을 찾게 되면, 그 catch블럭 내의 문장들을 수행하고 전체 try-catch문을 빠져나가서 그 다음 문장을 계속해서 수행한다. 만일 일치하는 catch블럭을 찾지 못하면, 예외는 처리되지 못한다.

▶ try블럭 내에서 예외가 발생하지 않은 경우,
1. catch블럭을 거치지 않고 전체 try-catch문을 빠져나가서 수행을 계속한다.

```
class ExceptionEx4 {
   public static void main(String args[]) {
            System.out.println(1);
            System.out.println(2);

            try {
                System.out.println(3);
                System.out.println(4);
            } catch (Exception e) {
                System.out.println(5);
            } // try-catch의 끝
            System.out.println(6);
   }    // main메서드의 끝
}
```

[실행결과]
```
1
2
3
4
6
```

```
class ExceptionEx5 {
   public static void main(String args[]) {
            System.out.println(1);
            System.out.println(2);
            try {
                System.out.println(3);
                System.out.println(0/0);
                System.out.println(4);
            } catch (ArithmeticException ae)  {
                System.out.println(5);
            }    // try-catch의 끝
            System.out.println(6);
   }    // main메서드의 끝
}
```

[실행결과]
```
1
2
3
5
6
```


1.5 예외 발생시키기 (생략)

1. 먼저, 연산자 new를 이용해서 발생시키려는 예외 클래스의 객체를 만든 다음

```
Exception e = new Exception("고의로 발생시켰음");
```

2. 키워드 throw를 이용해서 예외를 발생시킨다.

```
throw e;
```

[예제8-6]/ch8/ExceptionEx6.java

```
class ExceptionEx6
{
   public static void main(String args[])
   {
        try {
            Exception e = new Exception("고의로 발생시켰음.");
            throw e;  // 예외를 발생시킴
        //  throw new Exception("고의로 발생시켰음.");

        } catch (Exception e) {
            System.out.println("에러 메시지 : " + e.getMessage());
            e.printStackTrace();
        }
        System.out.println("프로그램이 정상 종료되었음.");
   }
}
```

위의 두 줄을 한 줄로 줄여 쓸 수 있다.

[실행결과]
```
에러 메시지 : 고의로 발생시켰음.
java.lang.Exception: 고의로 발생시켰음.
        at ExceptionEx6.main(ExceptionEx6.java:6)
프로그램이 정상 종료되었음.
```

※ 그럼, 예외처리는 어떻게 미리 알고 처리하나?

(1) API Specification에서 확인 후 예외처리

(2) 런타임 에러가 발생하면 예외처리

(3) 예외 발생 가능한 문장에 대해 예외처리(경험)

▶ API Specification에서 확인 후 예외처리

- 라이브러리 메서드(Library method 또는 Built-in method)를 사용하기 전에 API Specification에서 확인 후 예외처리 (Standard Edition 6 API Specification, http://download.oracle.com/javase/6/docs/api/index.html)

readLine

```
public String readLine()
                throws IOException
```

Reads a line of text. A line is considered to be terminated by any one of a line feed ('₩n'), a carriage return ('₩r'), or a carriage return followed immediately by a linefeed.

Returns:

A String containing the contents of the line, not including any line-termination characters, or null if the end of the stream has been reached

Throws:

IOException - If an I/O error occurs

9

▶ 런타임 에러가 발생하면 예외처리

- 아래 예제는 컴파일 시에는 에러가 발생하지 않아 실행된다. 그러나 실행 중 0으로 나누어지는 순간 다음과 같은 메시지를 출력하며 런타임 에러가 발생한다.

예제 8-2

```
class ExceptionEx2 {
        public static void main(String args[]) {
                int number = 100;
                int result = 0;

                for(int i=0; i < 10; i++) {
                        result = number / (int)(Math.random() * 10); // 7번째 라인
                        System.out.println(result);
                }
        }
}
```

실행결과

```
20
100
java.lang.ArithmeticException: / by zero
     at ExceptionEx2.main(ExceptionEx2.java:7)
```

10

▶ 예외 발생 가능한 문장에 대해 예외처리(경험)

- 앞과 같이 런타임 에러를 자주 경험하다 보면, 미리 예외 발생 가능한 문장에 대해 예외처리하게 된다.

예제 8-3

```
class ExceptionEx3 {
  public static void main(String args[]) {
            int number = 100;
            int result = 0;

            for(int i=0; i < 10; i++) {
                        try {
                                    result = number / (int)(Math.random() * 10);
                                    System.out.println(result);
                        } catch (ArithmeticException e)      {
                                    System.out.println("0"); // ArithmeticException이 발생하면 실행되는 코드
                        } // try-catch의 끝
            } // for의 끝
  }
}
```

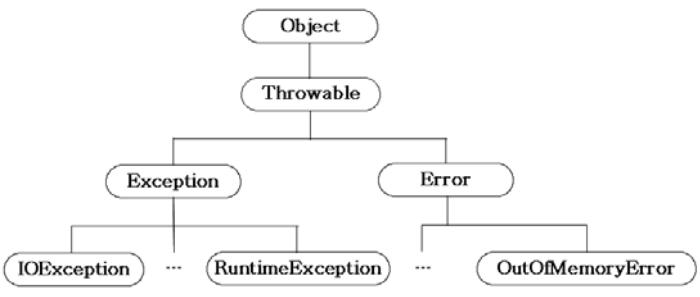
실행결과

```
16
20
11
0 ← ArithmeticException이 발생해서 0이 출력되었다.
25
...
```

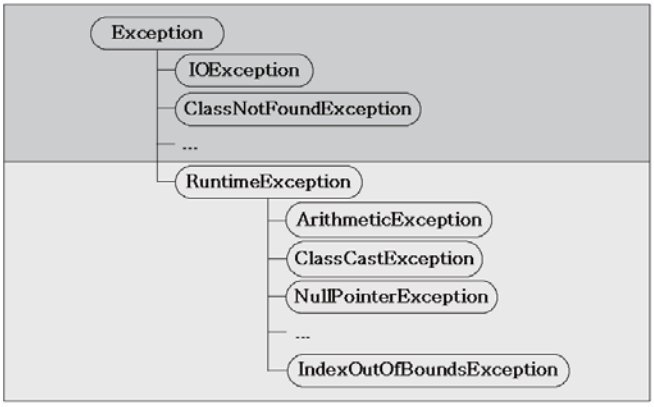

1.6 예외 클래스의 계층구조

- 예외 클래스는 크게 두 그룹으로 구분한다.

RuntimeException클래스들 - 프로그래머의 실수로 발생하는 예외 예외처리 필수
Exception클래스들 - 사용자의 실수와 같은 외적인 요인에 의해 발생하는 예외예외처리 선택



[그림8-1] 예외클래스 계층도



[그림8-2] Exception클래스와 RuntimeException클래스 중심의 상속계층도

RuntimeException클래스들 - 프로그래머의 실수로 발생하는 예외 ← 예외처리 필수

Exception클래스들 - 사용자의 실수와 같은 외적인 요인에 의해 발생하는 예외 ← 예외처리 선택

```
C:₩WINDOWS₩system32₩cmd.exe
C:₩jdk1.5₩work>javac ExceptionEx7.java
ExceptionEx7.java:5: unreported exception java.lang.Exception; must be caught or
 declared to be thrown
                throw new Exception();          // Exception을 강제로 발생시킨다
                ^
1 error
```

[예제8-7]/ch8/ExceptionEx7.java

```
class ExceptionEx7
{
   public static void main(String[] args)
   {
       throw new Exception();        // Exception을 강제로 발생시킨다.
   }
}
```

[예제8-8]/ch8/ExceptionEx8.java

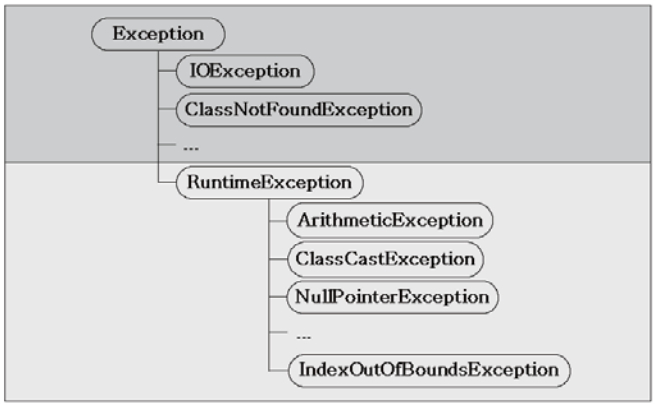
```
class ExceptionEx8 {
   public static void main(String[] args)
   {
       try {
           throw new Exception();
       } catch (Exception e) {
           System.out.println("Exception이 발생했습니다.");
       }
   }   // main메서드의 끝
}
```

[예제8-9]/ch8/ExceptionEx9.java

```
class ExceptionEx9 {
   public static void main(String[] args)
   {
       throw new RuntimeException(); // RuntimeException을 강제로 발생시킨다.
   }
}
```

```
C:₩WINDOWS₩system32₩cmd.exe
C:₩jdk1.5₩work>javac ExceptionEx9.java

C:₩jdk1.5₩work>
```

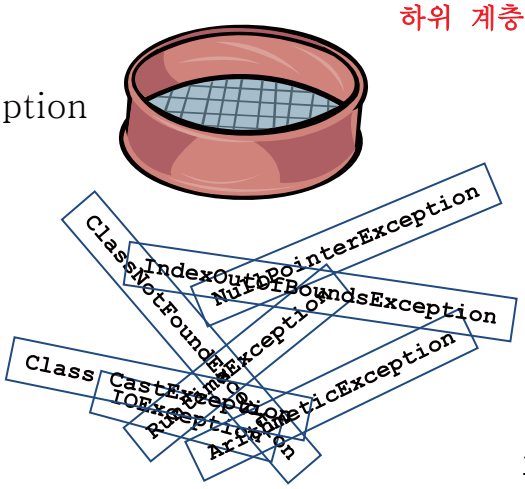
```
C:₩WINDOWS₩system32₩cmd.exe
C:₩jdk1.5₩work>java ExceptionEx9
Exception in thread "main" java.lang.RuntimeException
        at ExceptionEx9.main(ExceptionEx9.java:4)

C:₩jdk1.5₩work>
```



[그림8-2] Exception클래스와 RuntimeException클래스 중심의 상속계층도

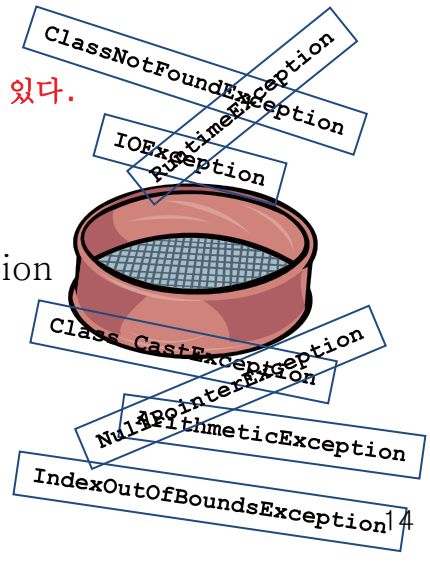
모든 예외를 다 걸러낸다.

Exception



하위 계층의 예외만 걸러낼 수 있다.

RuntimeException



1.7 예외의 발생과 catch블럭

- try블럭에서 예외가 발생하면, 발생한 예외를 처리할 catch블럭을 찾는다.
- 첫번째 catch블럭부터 순서대로 찾아 내려가며, 일치하는 catch블럭이 없으면 예외는 처리되지 않는다.
- catch블럭은 좁은 범위에서 넓은 범위 순서로 배치한다. (가장 처음의 catch블럭이 포괄적인 경우에는 이후의 catch블럭이 실행되지 않기 때문이다.)
- 예외의 최고 조상인 Exception을 처리하는 catch블럭은 모든 종류의 예외를 처리할 수 있다.(반드시 마지막 catch블럭이어야 한다.)

[예제8-11]/ch8/ExceptionEx11.java

```
class ExceptionEx11 {
    public static void main(String args[]) {
        System.out.println(1);
        System.out.println(2);
        try {
            System.out.println(3);
            System.out.println(0/0);
            System.out.println(4);        // 실행되지 않는다.
        } catch (ArithmeticException ae)  {
            if (ae instanceof ArithmeticException)
                System.out.println("true");
                System.out.println("ArithmeticException");
        } catch (Exception e) {
            System.out.println("Exception");
        }    // try-catch의 끝
        System.out.println(6);
    }    // main메서드의 끝
}
```

0으로 나눠서 ArithmeticException을 발생시킨다.

ArithmeticException을 제외한 모든 예외가 처리된다.

```
C:₩jdk1.5₩work>java ExceptionEx11
1
2
3
true
ArithmeticException
6
```

15

- 발생한 예외 객체를 catch블럭의 참조변수로 접근할 수 있다.

printStackTrace() - 예외발생 당시의 호출스택(Call Stack)에 있었던 메서드의 정보와 예외 메시지를 화면에 출력한다.
getMessage() - 발생한 예외클래스의 인스턴스에 저장된 메시지를 얻을 수 있다.

[예제8-12]/ch8/ExceptionEx12.java

```
class ExceptionEx12 {
    public static void main(String args[]) {
        System.out.println(1);
        System.out.println(2);
        try {
            System.out.println(3);
            System.out.println(0/0); // 예외발생!!!
            System.out.println(4);   // 실행되지 않는다.
        } catch (ArithmeticException ae)  {
            ae.printStackTrace();
            System.out.println("예외메시지 : " + ae.getMessage());
        }    // try-catch의 끝
        System.out.println(6);
    }    // main메서드의 끝
}
```

참조변수 ae를 통해, 생성된 ArithmeticException인 스턴스에 접근할 수 있다.

```
C:₩jdk1.5₩work>java ExceptionEx12
1
2
3
java.lang.ArithmeticException: / by zero
        at ExceptionEx12.main(ExceptionEx12.java:7)
예외메시지 : / by zero
6
```

16

1.8 finally블럭

- 예외의 발생여부와 관계없이 실행되어야 하는 코드를 넣는다.
- 선택적으로 사용할 수 있으며, try-catch-finally의 순서로 구성된다.
- 예외 발생시, try → catch → finally의 순서로 실행되고
  예외 미발생시, try → finally의 순서로 실행된다.
- try 또는 catch블럭에서 return문을 만나도 finally블럭은 수행된다.

```
try {
    // 예외가 발생할 가능성이 있는 문장들을 넣는다.
} catch (Exception1 e1) {
    // 예외처리를 위한 문장을 적는다.
} finally {
    // 예외의 발생여부에 관계없이 항상 수행되어야하는 문장들을 넣는다.(옵션)
    // finally블럭은 try-catch문의 맨 마지막에 위치해야한다.
}
```

17

▶ finally블럭의 예

[예제8-15]/ch8/FinallyTest.java

```
class FinallyTest {
  public static void main(String args[]) {
      try {
          startInstall();       // 프로그램 설치에 필요한 준비를 한다.
          copyFiles();          // 파일들을 복사한다.
          deleteTempFiles();    // 프로그램 설치에 사용된 임시파일들을 삭제한다.
      } catch (Exception e) {
          e.printStackTrace();
          deleteTempFiles();    // 프로그램 설치에 사용된 임
      } // try-catch의 끝
  } // main의 끝
  static void startInstall() {
      /* 프로그램 설치에 필요한 준비를 하는 코드를 적는다.*/
  }
  static void copyFiles() { /* 파일들을 복사하는 코드를 적는다. */ }
  static void deleteTempFiles() { /* 임시파일들을 삭제하는 코드를 적는다.*/ }
}
```

```
try {
    startInstall();
    copyFiles();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    deleteTempFiles();
} // try-catch의 끝
```

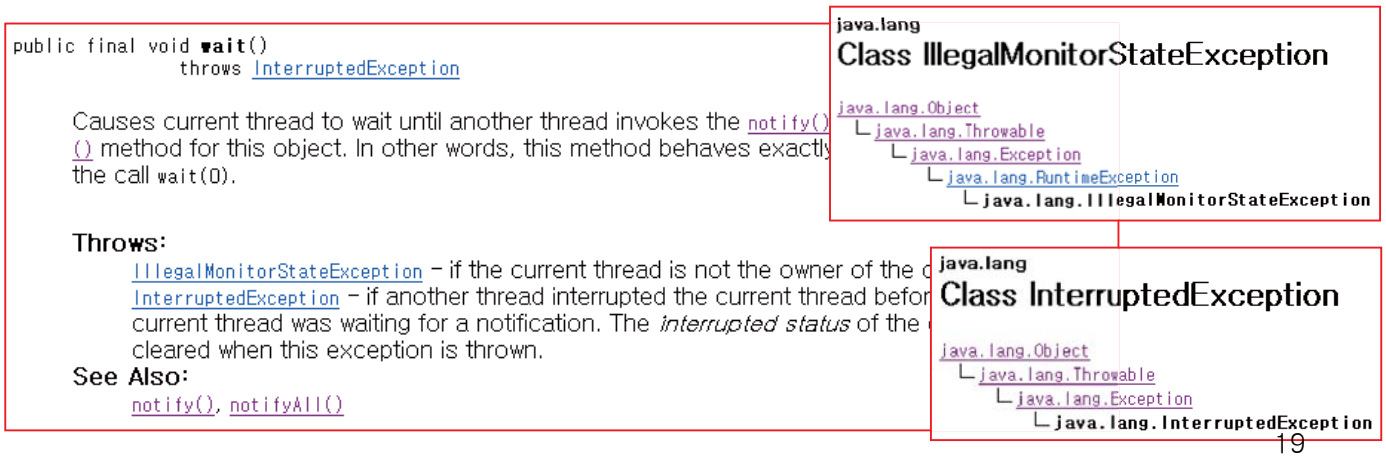
18

1.9 메서드에 예외 선언하기

- 예외를 처리하는 또 다른 방법
- 사실은 예외를 처리하는 것이 아니라, 호출한 메서드로 전달해주는 것
- 호출한 메서드에서 예외처리를 해야만 할 때 사용

```
void method() throws Exception1, Exception2, ... ExceptionN {
    // 메서드의 내용
}
```

[참고] 예외를 발생시키는 키워드 throw와 예외를 메서드에 선언할 때 쓰이는 throws를 잘 구별하자.

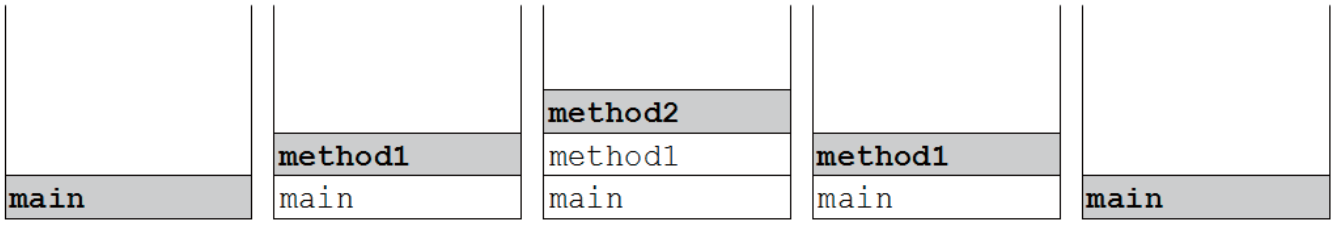


▶ 메서드에 예외 선언하기의 예 (1)

[예제8-18]/ch8/ExceptionEx18.java

```
class ExceptionEx18 {
    public static void main(String[] args) throws Exception {
        method1();    // 같은 클래스내의 static멤버이므로 객체생성없이 직접 호출가능.
    }    // main메서드의 끝

    static void method1() throws Exception {
        method2();
    }    // method1의 끝

    static void method2() throws Exception {
        throw new Exception();
    }    // method2의 끝
}
```

```
C:\WINDOWS\system32\CMD.exe
C:\jdk1.5\work>java ExceptionEx18
Exception in thread "main" java.lang.Exception
        at ExceptionEx18.method2(ExceptionEx18.java:11)
        at ExceptionEx18.method1(ExceptionEx18.java:7)
        at ExceptionEx18.main(ExceptionEx18.java:3)
```

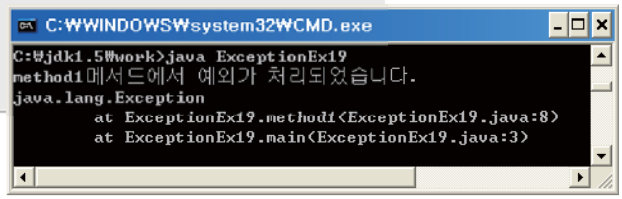


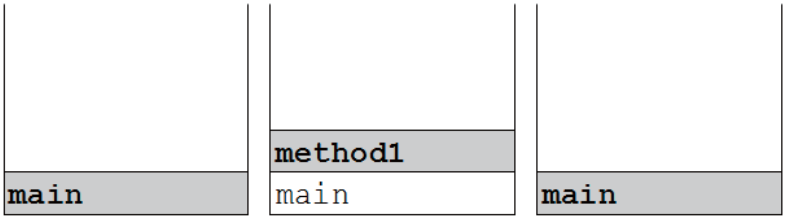
- 예외가 발생했을 때, 모두 3개의 메서드(main, method1, method2)가 호출스택에 있었으며,
- 예외가 발생한 곳은 제일 윗줄에 있는 method2()라는 것과
- main메서드가 method1()를, 그리고 method1()은 method2()를 호출했다는 것을 알 수 있다.

▶ 메서드에 예외 선언하기의 예 (2)

[예제8-19]/ch8/ExceptionEx19.java

```
class ExceptionEx19 {
  public static void main(String[] args) {
      method1();     // 같은 클래스내의 static멤버이므로 객체생성없이 직접 호출가능.
  }   // main메서드의 끝

  static void method1() {
      try {
          throw new Exception();
      } catch (Exception e) {
          System.out.println("method1메서드에서 예외가 처리되었습니다.");
          e.printStackTrace();
      }
  }   // method1의 끝
}
```





21

▶ 메서드에 예외 선언하기의 예 (3)

[예제8-21]/ch8/ExceptionEx21.java

```
import java.io.*;

class ExceptionEx21 {
  public static void main(String[] args) {
      // command line에서 입력받은 값을 이름으로 갖는 파일을 생성한다.
      File f = createFile(args[0]);
      System.out.println( f.getName() + " 파일이 성공적으로 생성되었
  } // main메서드의 끝

  static File createFile(String fileName) {
      try {
          if (fileName==null || fileName.equals(""))
              throw new Exception("파일이름이 유효하지 않습니다.");
      } catch (Exception e) {
          // fileName이 부적절한 경우, 파일 이름을 '제목없음.txt'로 한
          fileName = "제목없음.txt";
      } finally {
          File f = new File(fileName); // File클래스의 객체를 만든다.
          createNewFile(f);            // 생성된 객체를 이용해서 파일을 생성한다.
          return f;
      }
  }   // createFile메서드의 끝

  static void createNewFile(File f) {
      try {
          f.createNewFile();       // 파일을 생성한다.
      } catch(Exception e){ }
  }   // createNewFile메서드의 끝
} // 클래스의 끝
```

[실행결과]

```
C:\jdk1.5\work>java ExceptionEx21 "test.txt"
test.txt 파일이 성공적으로 생성되었습니다.

C:\jdk1.5\work>java ExceptionEx21 ""
제목없음.txt 파일이 성공적으로 생성되었습니다.

C:\jdk1.5\work>dir *.txt

드라이브 C에 레이블이 없습니다
볼륨 일련 번호 251C-08DD
디렉터리 C:\jdk1.5\work

제목없음 TXT 0 03-01-24 0:47 제목없음.txt
TEST TXT 0 03-01-24 0:47 test.txt
```

22

▶ 메서드에 예외 선언하기의 예 (4)

[예제8-22]/ch8/ExceptionEx22.java

```
import java.io.*;

class ExceptionEx22 {
   public static void main(String[] args)
   {
       try {
           File f = createFile(args[0]);
           System.out.println( f.getName()+"파일이 성공적으로 생성되었습니다.");
       } catch (Exception e) {
           System.out.println(e.getMessage()+" 다시 입력해 주시기 바랍니다.");
       }
   }   // main메서드의 끝

   static File createFile(String fileName) throws Exception {
       if (fileName==null || fileName.equals(""))
           throw new Exception("파일이름이 유효하지 않습니다.");
       File f = new File(fileName);       //  File클래스의 객체를 만든다.
       // File객체의 createNewFile메서드를 이용해서 실제 파일을 생성한다.
       f.createNewFile();
       return f;         // 생성된 객체의 참조를 반환한다.
   }   // createFile메서드의 끝
} // 클래스의 끝
```

[실행결과]

```
C:\jdk1.5\work>java ExceptionEx22 test2.txt
test2.txt파일이 성공적으로 생성되었습니다.

C:\jdk1.5\work>java ExceptionEx22 ""
파일이름이 유효하지 않습니다. 다시 입력해 주시기 바랍니다.
```

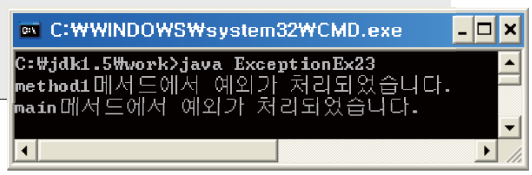
23

1.10 예외 되던지기(re-throwing) (생략)

- 예외를 처리한 후에 다시 예외를 생성해서 호출한 메서드로 전달하는 것
- 예외가 발생한 메서드와 호출한 메서드, 양쪽에서 예외를 처리해야 하는 경우에 사용.

[예제8-23]/ch8/ExceptionEx23.java

```
class ExceptionEx23 {
   public static void main(String[] args)
   {
       try  {
           method1();
       } catch (Exception e) {
           System.out.println("main메서드에서 예외가 처리되었습니다.");
       }
   }   // main메서드의 끝

   static void method1() throws Exception {
       try {
           throw new Exception();
       } catch (Exception e) {
           System.out.println("method1메서드에서 예외가 처리되었습니다.");
           throw e;          // 다시 예외를 발생시킨다.
       }
   }   // method1메서드의 끝
}
```



24

1.11 사용자정의 예외 만들기

- 기존의 예외 클래스를 상속받아서 새로운 예외 클래스를 정의할 수 있다.

```
class MyException extends Exception {
     MyException(String msg) {  // 문자열을 매개변수로 받는 생성자
       super(msg); // 조상인 Exception클래스의 생성자를 호출한다.
     }
}
```

- 에러코드를 저장할 수 있게 ERR_CODE와 getErrCode()를 멤버로 추가

```
class MyException extends Exception {
   // 에러 코드 값을 저장하기 위한 필드를 추가 했다.
   private final int ERR_CODE;

   MyException(String msg, int errCode) {  // 생성자.
      super(msg);
      ERR_CODE = errCode;
   }

   MyException(String msg) {  // 생성자.
      this(msg,100);           // ERR_CODE를 100(기본값)으로 초기화한다.
   }

   public int getErrCode() {  // 에러 코드를 얻을 수 있는 메서드도 추가했다.
       return ERR_CODE;     // 이 메서드는 주로 getMessage()와 함께 사용될 것이다.
   }
}
```

25

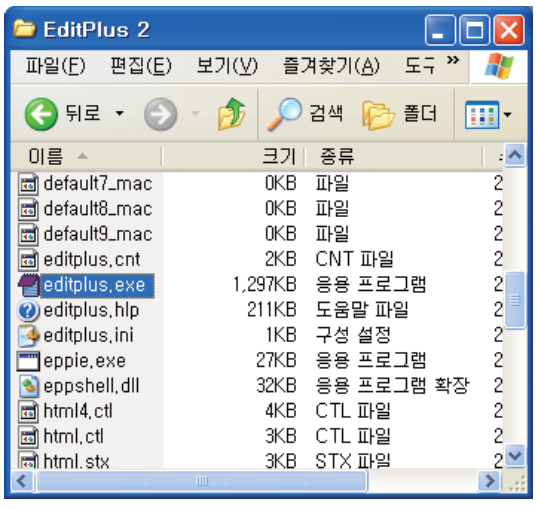
제 12 장
쓰레드(Thread)

인제대학교 전자IT기계자동차공학부

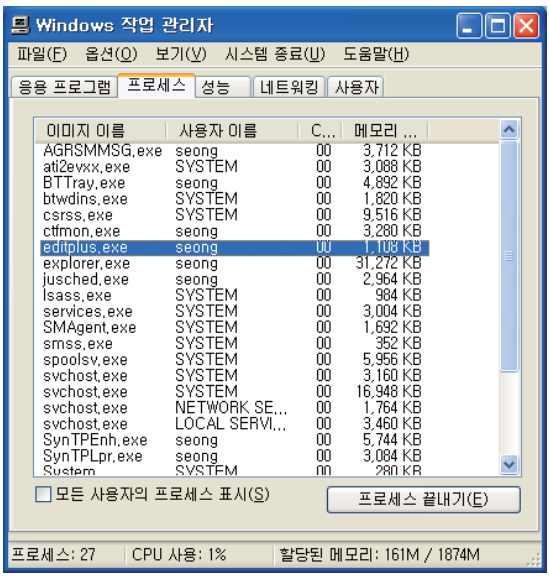
황 원 주

1.1 프로세스와 쓰레드(process & thread)

▶ 프로그램 : 실행 가능한 파일(HDD)



▶ 프로세스 : 실행 중인 프로그램 (working program) (메모리)



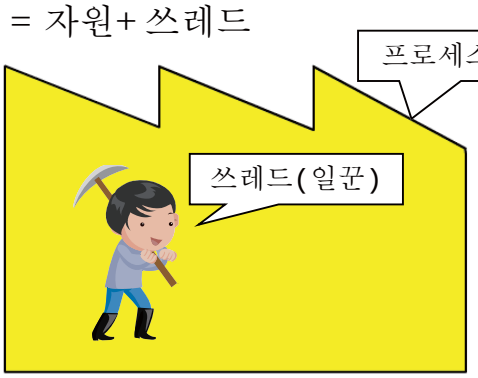
2

▶ 프로세스 : 실행 중인 프로그램, 자원(resources)과 쓰레드로 구성

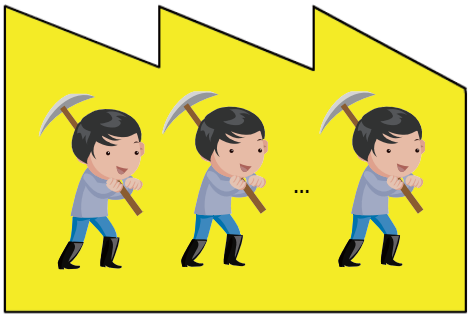
▶ 쓰레드 : 프로세스 내에서 실제 작업을 수행.
모든 프로세스는 하나 이상의 쓰레드를 가지고 있다.

프로세스 : 쓰레드 = 공장 : 일꾼

▶ 싱글 쓰레드 프로세스
= 자원+ 쓰레드



▶ 멀티 쓰레드 프로세스
= 자원+ 쓰레드+ 쓰레드+ …+ 쓰레드

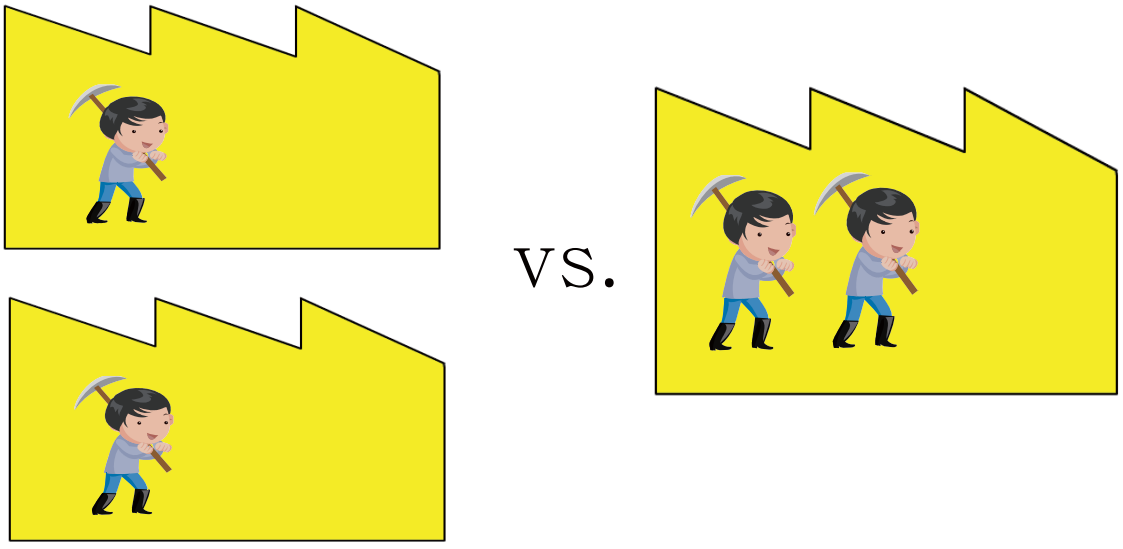


3

1.2 멀티프로세스 vs. 멀티쓰레드

**"하나의 새로운 프로세스를 생성하는 것보다
하나의 새로운 쓰레드를 생성하는 것이 더 적은 비용이 든다."**

- 2 프로세스 1 쓰레드 vs. 1 프로세스 2 쓰레드



4

1.3 멀티쓰레드의 장단점

"많은 프로그램들이 멀티쓰레드로 작성되어 있다.
그러나, 멀티쓰레드 프로그래밍이 장점만 있는 것은 아니다."

장점	- 자원을 보다 **효율적**으로 사용할 수 있다. - 사용자에 대한 **응답성(responseness)이 향상**된다. - 작업이 분리되어 **코드가 간결**해 진다. **"여러 모로 좋다."**
단점	- 동기화(synchronization)에 주의해야 한다. - 교착상태(dead-lock)가 발생하지 않도록 주의해야 한다. - 각 쓰레드가 효율적으로 고르게 실행될 수 있게 해야 한다. **"프로그래밍할 때 고려해야 할 사항들이 많다."**

5

1.4 쓰레드의 구현과 실행 (2가지 방법)

1) Thread 클래스를 상속받아 새로운 클래스 정의

```
class MyThread extends Thread {
   ....
   public void run() {
      // 쓰레드가 수행할 일을 여기에 기술한다.
   }
}

MyThread t1 = new MyThread();
t1.start();
```

6

1. 자바는 다중상속을 제공하지 않으므로
2. Thread클래스를 상속받으면 오버헤드가 크므로

2) Runnable 인터페이스를 구현하여 클래스를 정의

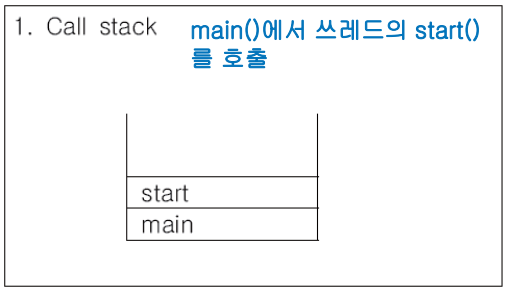
```
Class MyRunnable extends Applet implements Runnable {
   .......
   public void run() {
     // 쓰레드가 수행할 일을 여기에 기술한다.
   }
}

MyRunnable r1 = new MyRunnable();
Thread t1 = new Thread(r1);
t1.start();
```

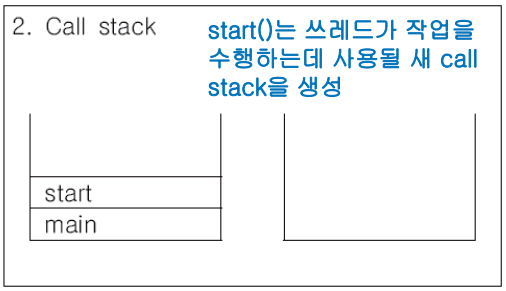
7

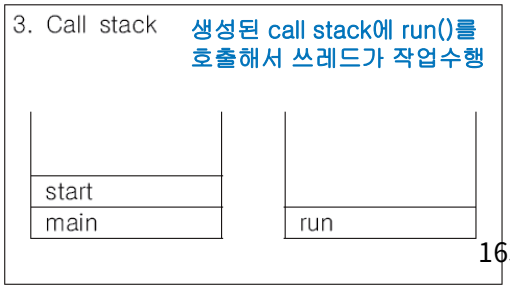
1.5 start() & run()

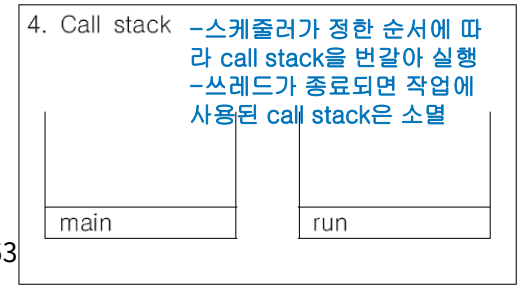
```
class ThreadTest {
    public static void main(String args[]){
        MyThread t1 = new MyThread();
        t1.start();
    }
}
```

```
class MyThread extends Thread {
    public void run() {
        //...
    }
}
```





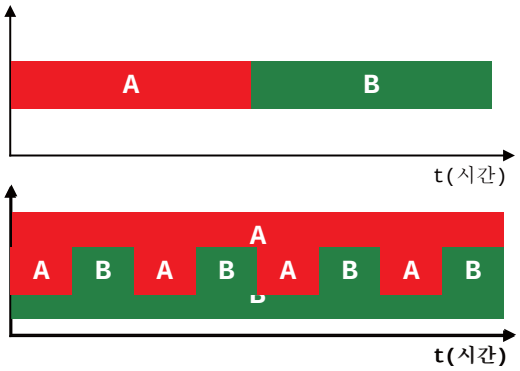




1.6 싱글쓰레드 vs. 멀티쓰레드

▶ 싱글쓰레드

```
class ThreadTest {
    public static void main(String args[]){
        for(int i=0;i<300;i++) {
            System.out.println("-");
        }

        for(int i=0;i<300;i++) {
            System.out.println("|");
        }
    } // main
}
```



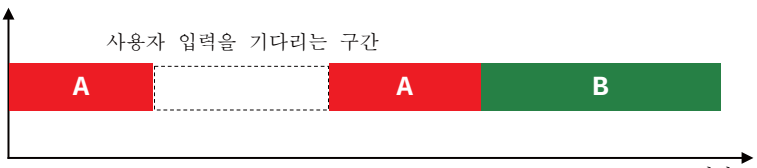
▶ 멀티쓰레드

```
class ThreadTest {
    public static void main(String args[]){
        MyThread1 th1 = new MyThread1();
        MyThread2 th2 = new MyThread2();
        th1.start();
        th2.start();
    }
}

class MyThread1 extends Thread {
    public void run() {
        for(int i=0;i<300;i++) {
            System.out.println("-");
        }
    } // run()
}

class MyThread2 extends Thread {
    public void run() {
        for(int i=0;i<300;i++) {
            System.out.println("|");
        }
    } // run()
}
```



```
class ThreadEx6 {
    public static void main(String[] args){
        String input = JOptionPane.showInputDialog("아무 값이나 입력하세요.");
        System.out.println("입력하신 값은 " + input + "입니다.");

        for(int i=10; i > 0; i--) {
            System.out.println(i);
            try { Thread.sleep(1000); } ca
        }
    } // main
}
```

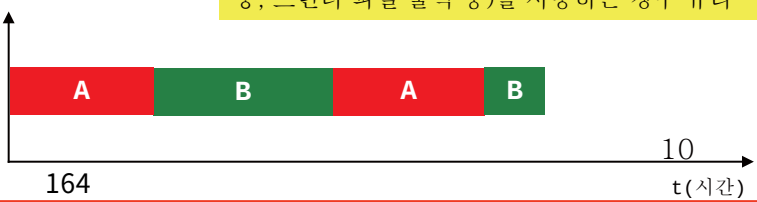
▶ 싱글쓰레드




```
class ThreadEx7 {
    public static void main(String[] args) {
        ThreadEx7_1 th1 = new ThreadEx7_1();
        th1.start();

        String input = JOptionPane.showInputDialog("아무 값이나 입력하세요.");
        System.out.println("입력하신 값은 " + input + "입니다.");
    }
}

class ThreadEx7_1 extends Thread {
    public void run() {
        for(int i=10; i > 0; i--) {
            System.out.println(i);
            try { sleep(1000); } catch(Exception e ) {}
        }
    } // run()
}
```

- CPU만을 사용하는 경우 쓰레드간의 작업전환(context switch)때문에 오히려 속도가 늦다.
- CPU이외의 자원(데이터 입력, 네트워크 파일전송, 프린터 파일 출력 등)을 사용하는 경우 유리

▶ 멀티쓰레드

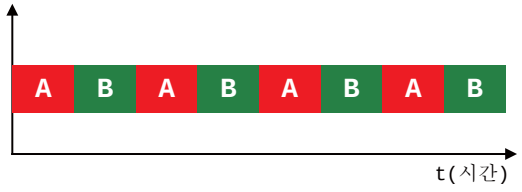


1.7 쓰레드의 우선순위(priority of thread)

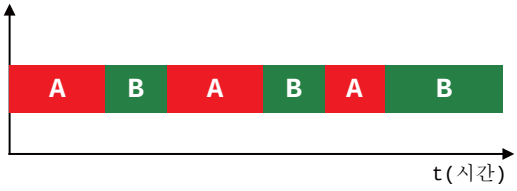
"작업의 중요도에 따라 쓰레드의 우선순위를 다르게 하여
특정 쓰레드가 더 많은 작업시간을 갖도록 할 수 있다."

```
void setPriority(int newPriority) : 쓰레드의 우선순위를 지정한 값으로 변경한다.
int getPriority() : 쓰레드의 우선순위를 반환한다.

public static final int MAX_PRIORITY = 10 // 최대우선순위
public static final int MIN_PRIORITY = 1  // 최소우선순위
public static final int NORM_PRIORITY = 5 // 보통우선순위
```

▶ 우선순위가 같은 경우



▶ A의 우선순위가 높은 경우



11

1.10 쓰레드의 실행제어

생성자 / 메서드	설 명
void interrupt()	sleep()이나 join()에 의해 일시정지상태인 쓰레드를 실행대기상태로 만든다. 해당 쓰레드에서는 InterruptedException이 발생함으로써 일시정지상태를 벗어나게 된다.
void join() void join(long millis) void join(long millis, int nanos)	지정된 시간동안 쓰레드가 실행되도록 한다. 지정된 시간이 지나거나 작업이 종료되면 join()을 호출한 쓰레드로 다시 돌아와 실행을 계속한다.
void resume()	suspend()에 의해 일시정지상태에 있는 쓰레드를 실행대기상태로 만든다.
static void sleep(long millis) static void sleep(long millis, int nanos)	지정된 시간(천분의 일초 단위)동안 쓰레드를 일시정지시킨다. 지정한 시간이 지나고 나면, 자동적으로 다시 실행대기상태가 된다.
void stop()	쓰레드를 즉시 종료시킨다. 교착상태(dead-lock)에 빠지기 쉽기 때문에 deprecated되었다.
void suspend()	쓰레드를 일시정지시킨다. resume()을 호출하면 다시 실행대기상태가 된다.
static void yield()	실행 중에 다른 쓰레드에게 양보(yield)하고 실행대기상태가 된다.

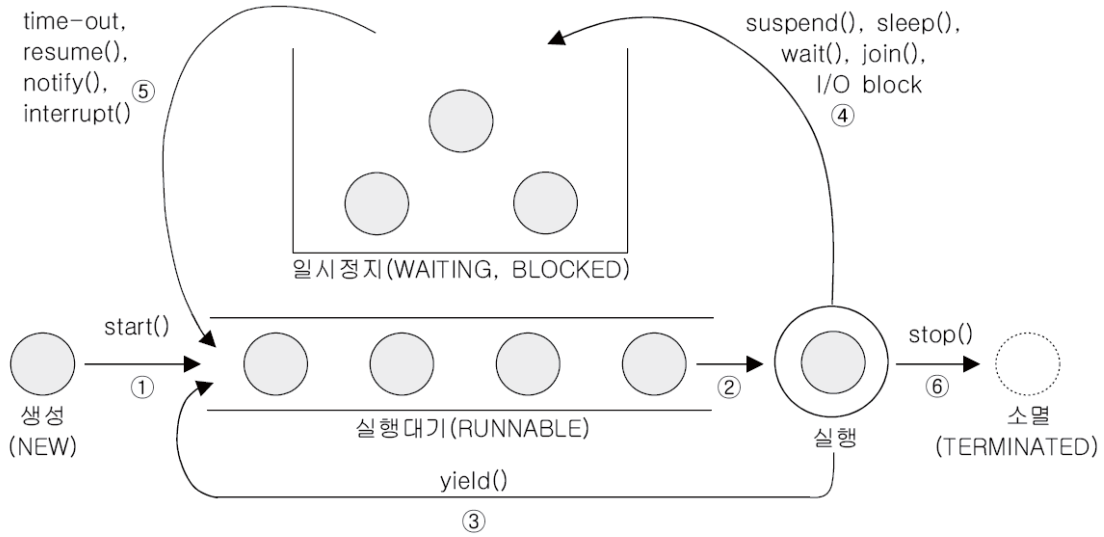
[표 12-2] 쓰레드의 스케줄링과 관련된 메서드

* resume(), stop(), suspend()는 쓰레드를 교착상태로 만들기 쉽기 때문에 deprecated되었다.

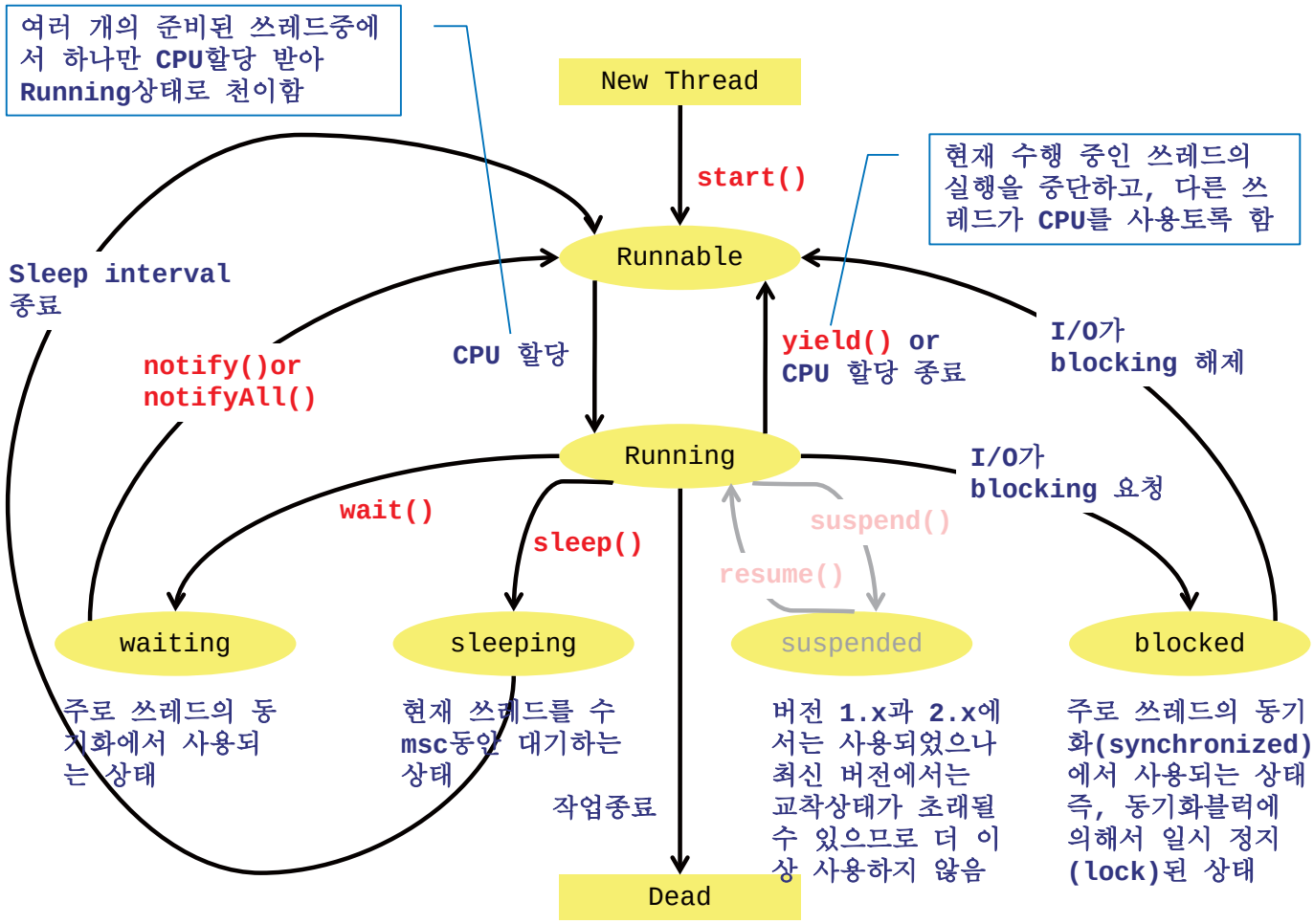
12

1.11 쓰레드의 상태(state of thread)

상태	설명
NEW	쓰레드가 생성되고 아직 start()가 호출되지 않은 상태
RUNNABLE	실행 중 또는 실행 가능한 상태
BLOCKED	동기화블럭에 의해서 일시정지된 상태(lock이 풀릴 때까지 기다리는 상태)
WAITING, TIMED_WAITING	쓰레드의 작업이 종료되지는 않았지만 실행가능하지 않은(unrunnable) 일시정지상태. TIMED_WAITING은 일시정지시간이 지정된 경우를 의미한다.
TERMINATED	쓰레드의 작업이 종료된 상태



13



14

1.12 쓰레드의 실행제어

▶ 예제 12-18

```
class MyThreadEx18 implements Runnable {
    boolean suspended = false;
    boolean stopped = false;

    Thread th;

    MyThreadEx18(String name) {
        th = new Thread(this, name);
    }

    public void run() {
        while(!stopped) {
            if(!suspended) {
                /*
                    작업수행
                */
                try {
                    Thread.sleep(1000);
                } catch(InterruptedException e) {}
            } // if
        } // while
    }

    public void suspend() {
        suspended = true;
    }

    public void resume() {
        suspended = false;
    }

    public void stop() {
        stopped = true;
    }

    public void start() {
        th.start();
    }
```

15

▶ 예제 12-20

```
public void run() {
  while(true) {
      try {
          Thread.sleep(10 * 1000);  // 10초를 기다린다.
      } catch(InterruptedException e) {
          System.out.println("Awaken by interrupt().");
      }

      gc();    // garbage collection을 수행한다.
      System.out.println("Garbage Collected. Free Memory :"
                                                  + freeMemory());
  }
}
```

```
for(int i=0; i < 20; i++) {
  requiredMemory = (int)(Math.random() * 10) * 20;

  // 필요한 메모리가 사용할 수 있는 양보다 적거나 전체 메모리의 60%이상을 사용했을 경우 gc를 깨운다.
  if(gc.freeMemory() < requiredMemory
  || gc.freeMemory() < gc.totalMemory() * 0.4) {
      gc.interrupt();       // 잠자고 있는 쓰레드 t1을 깨운다.
  }

  gc.usedMemory += requiredMemory;
  System.out.println("usedMemory:"+gc.usedMemory);
}
```

16

1.13 쓰레드의 동기화 - synchronized

- 여러 쓰레드가 하나의 객체를 공유할 때, 한 번에 하나의 쓰레드만 객체에 접근할 수 있도록 객체에 락(lock)을 걸어서 데이터의 일관성을 유지하는 것.

1. 특정한 객체에 lock을 걸고자 할 때

```
synchronized(객체의 참조변수) {
    //...
}
```

2. 메서드에 lock을 걸고자할 때

```
public synchronized void calcSum() {
    //...
}
```

(추천) 한 쓰레드가 synchronized메서드를 호출해서 수행하고 있으면 이 메서드가 종료될 때까지 다른 쓰레드가 이 메서드를 호출해서 수행할 수 없다.

```
public synchronized void withdraw(int money) {
    if(balance >= money) {
        try {
            Thread.sleep(1000);
        } catch(Exception e) {}

        balance -= money;
    }
}
```

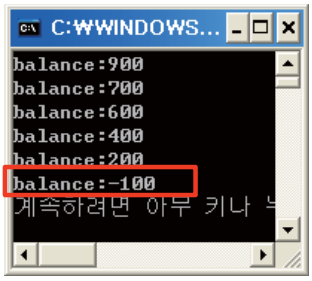
```
public void withdraw(int money) {
    synchronized(this) {
        if(balance >= money) {
            try {
                Thread.sleep(1000);
            } catch(Exception e) {}

            balance -= money;
        }
    } // synchronized(this)
}
```

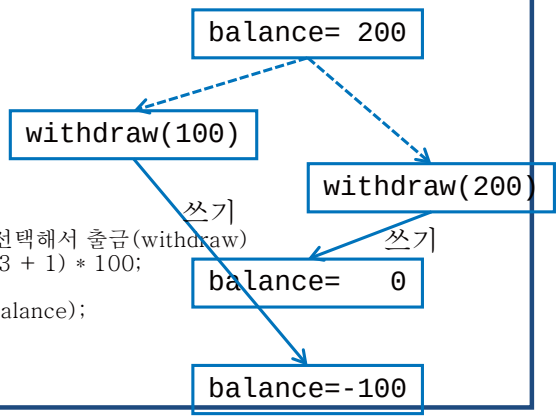
17

예제 12-24

```
class ThreadEx24 {
    public static void main(String args[]) {
        Runnable r = new RunnableEx24();
        Thread t1 = new Thread(r);
        Thread t2 = new Thread(r);

        t1.start();
        t2.start();
    }
}

class Account {
    int balance = 1000;

    public void withdraw(int money) {
        if(balance >= money) {
            try { Thread.sleep(1000);} catch(Exception e) {}
            balance -= money;
        }
    } // withdraw
}

class RunnableEx24 implements Runnable {
    Account acc = new Account();

    public void run() {
        while(acc.balance > 0) {
            // 100, 200, 300중의 한 값을 임으로 선택해서 출금(withdraw)
            int money = (int)(Math.random() * 3 + 1) * 100;
            acc.withdraw(money);
            System.out.println("balance:"+acc.balance);
        }
    } // run()
}
```

▶Synchronized없을 때



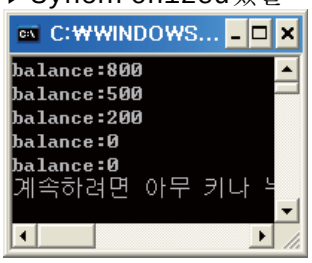


- **두 쓰레드의 객체(balance) 공유**. 즉, 한 쓰레드가 if문의 조건식을 계산하였을 때는 balance=200이고 출금하려는 금액이 100이어서 조건식이 참이 되어 출금하려는 순간 다른 쓰레드에게 제어권이 넘어가서 다른 쓰레드가 200을 출금하여 balance가 0이 되었다. 다시 이전의 쓰레드로 제어권이 넘어오면 if문 다음부터 다시 수행하게 되므로 balance가 0인 상태에서 100을 출금하여 balance가 결국 -100되었다. -> if문과 출금하는 문장(balance-=money)은 하나의 동기화블록으로 묶어야 한다.

예제 12-24 (매우 중요)

```
class ThreadEx25 {
        public static void main(String args[]) {
                Runnable r = new RunnableEx101();
                Thread t1 = new Thread(r);
                Thread t2 = new Thread(r);

                t1.start();
                t2.start();
        }
}

class Account {
        int balance = 1000;

        public synchronized void withdraw(int money) {
                if(balance >= money) {
                        try { Thread.sleep(1000);} catch(Exception e) {}
                        balance -= money;
                }
        } // withdraw
}

class RunnableEx101 implements Runnable {
        Account acc = new Account();

        public void run() {
                while(acc.balance > 0) {
                        int money = (int)(Math.random() * 3 + 1) * 100;
                        acc.withdraw(money);
                        System.out.println("balance:"+acc.balance);
                }
        } // run()
}
```

▶Synchronized있을 때



예제-동기화 없음

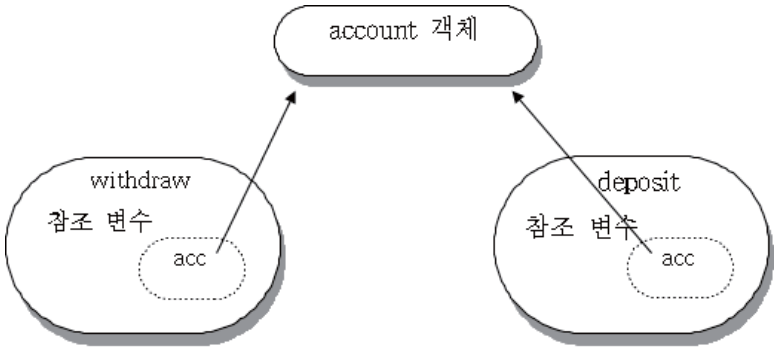
```
class Account {
  int bal=10000;
}
class Deposit extends Thread {
  Account account;
  int temp;
  Deposit(Account a) { // 생성자
    account = a;
  }
  public void run() {
    for(int i=1; i<=10; i++) {
        temp = account.bal;
        temp += 100;
        System.out.println("dep # "+i+" Current = " + temp);
        account.bal = temp;
    }
  }
}
class Withdraw extends Thread {
  Account account;
  int temp;
  Withdraw(Account a) { // 생성자
    account = a;
  }
  public void run() {
    for(int i=1; i<=10; i++) {
        temp = account.bal;
        temp -= 100;
        System.out.println("with# "+i+" Current = "+temp);
        account.bal = temp;
}}}
class ConCurrent {
  public static void main(String args[]) {
    Account acc  = new Account();
    Deposit deposit  = new Deposit(acc);
    Withdraw withdraw = new Withdraw(acc);
    deposit.start();
    withdraw.start();
    try {
        deposit.join(); //deposit 쓰레드가 종료할 때까지 대기
        withdraw.join(); //withdraw 쓰레드가 종료할 때까지 대기
    } catch(InterruptedException e) { e.printStackTrace(); }
    System.out.println("The balance = " + acc.bal);
  }
}
```

실행결과

```
dep # 1 Current = 10100
dep # 2 Current = 10200
with# 1 Current = 10000
dep # 3 Current = 10300
with# 2 Current = 9900
dep # 4 Current = 10400
with# 3 Current = 9800
dep # 5 Current = 10500
with# 4 Current = 9700
dep # 6 Current = 10600
with# 5 Current = 9600
dep # 7 Current = 10700
with# 6 Current = 9500
dep # 8 Current = 10800
with# 7 Current = 9400
dep # 9 Current = 10900
with# 8 Current = 9300
dep # 10 Current = 11000
with# 9 Current = 9200
with# 10 Current = 9100
The balance = 9100
```

▶ 두 쓰레드의 객체 공유

- withdraw 쓰레드와 deposit 쓰레드의 account 객체 공유 방법



21

예제-동기화 있음 (매우 중요)

```
class Account {  int bal=10000; }
class Deposit extends Thread {
  Account account;
  int temp;
  Deposit(Account a) {     account = a; }
  synchronized int add(int amt) {  // 동기화된 메소드
                account.bal += amt;
                return (account.bal);
  }
  public void run() {
    for(int i=1; i<=10; i++) {
                                temp = add(100);
                                System.out.println("dep # " + i + " Current = " + temp);
    }
  }
}
class Withdraw extends Thread {
  Account account;
  int temp;
  Withdraw(Account a) {   account = a; }
  synchronized int sub(int amt) {  // 동기화된 메소드
                                account.bal -= amt;
                                return (account.bal);
  }
  public void run() {
    for(int i=1; i<=10; i++) {
                                temp = sub(100);
                                System.out.println("with# " + i + " Current = " + temp);
    }
  }
}

class SynConCurrent {
  public static void main(String args[]) {
    Account acc  = new Account();
    Deposit deposit  = new Deposit(acc);  // 두 쓰레드의 객체 공유
    Withdraw withdraw = new Withdraw(acc);
    deposit.start();
    withdraw.start();
    try {
                                deposit.join();
                                withdraw.join();
    } catch(InterruptedException e) {                    e.printStackTrace(); }
    System.out.println("The balance = " + acc.bal);
}}
```

실행결과

```
dep # 1 Current = 10100
dep # 2 Current = 10200
with# 1 Current = 10100
dep # 3 Current = 10200
with# 2 Current = 10100
dep # 4 Current = 10200
with# 3 Current = 10100
dep # 5 Current = 10200
with# 4 Current = 10100
dep # 6 Current = 10200
with# 5 Current = 10100
dep # 7 Current = 10200
with# 6 Current = 10100
dep # 8 Current = 10200
with# 7 Current = 10100
dep # 9 Current = 10200
with# 8 Current = 10100
dep # 10 Current = 10200
with# 9 Current = 10100
with# 10 Current = 10000
The balance = 10000
```

1.14 쓰레드의 동기화–wait(), notify(), notifyAll()

- 두 개 이상의 쓰레드가 동시에 객체에 접근하는 것 방지(Synchronized 동기화) + 두 개 이상의 쓰레드가 특정 순서에 따라 실행하고자 할 때 wait(), notify()를 사용

- Object클래스에 정의되어 있으며, 동기화 블록 내에서만 사용할 수 있다.

▶wait() – 객체의 lock을 풀고 해당 객체의 쓰레드를 waiting pool에 넣는다.

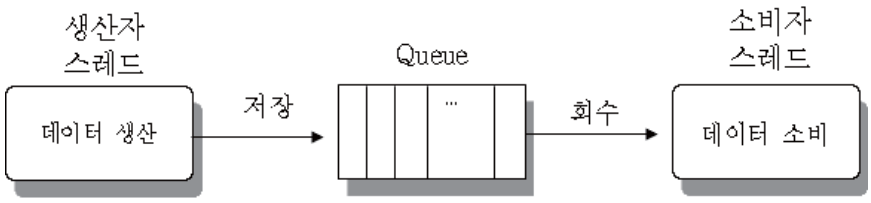
▶notify() – waiting pool에서 대기중인 쓰레드 중의 하나를 깨운다.

▶notifyAll() – waiting pool에서 대기중인 모든 쓰레드를 깨운다.

```
class Account {
    int balance = 1000;

    public synchronized void withdraw(int money){
        while(balance < money) {
            try {
                wait();
            } catch(InterruptedException e) {}
        }

        balance -= money;
    } // withdraw

    public synchronized void deposit(int money) {
        balance += money;
        notify();
    }
}
```

23

▶ 생산자 / 소비자 문제



① 생산자와 소비자는 서로 독립적으로 실행된다.

② 생산자는 데이터를 생산해서 Queue에 저장한다.

③ 소비자는 Queue에서 데이터를 읽어 사용한다.

④ Queue에 데이터를 추가 저장할 공간이 없을 때, 생산자는 데이터 생산을 멈추고 대기한다.

⑤ Queue에 저장된 데이터가 없으면 소비자는 대기한다.

⑥ Queue는 생산자와 소비자가 공유하는 객체로서 어느 한 순간에는 하나의 쓰레드만 접근할 수 있다.

예제-1/2 (매우 중요)

```
class Producer extends Thread {
  Queue queue;
  Producer(Queue queue) { // 생성자
    this.queue = queue;
  }
  public void run() {
    int x = 1;
    int idx = 0;
    while(++idx <= 10) {
      queue.add(x);
      System.out.println("Producer: "+ x);
      x *= 2;              // 새로운 데이터 생산
    }
  }
}
class Consumer extends Thread {
  Queue queue;
  Consumer(Queue queue) { // 생성자
    this.queue = queue;
  }
  public void run() {
    int idx = 0;
    while(++idx <= 10) {
      System.out.println("\tConsumer: " + queue.remove());
    }
  }
}
```

예제-2/2 (매우 중요)

```
class Queue {
  int buf;
  boolean available = true;
    synchronized void add(int data) {
    while(!available) { // 여유 공간이 없으면 대기
      try {
        wait();
      }
      catch(Exception e) {}
    }
    buf = data;          // Producer이 데이터를 큐에 저장한 후에는
    available = false;  // 반드시 대기하고 있을 지도 모르는
    notify();            // Consumer 쓰레드를 깨움
  }
  synchronized int remove() {
    while(available) { // 데이터가 없으면 대기
      try {
        wait();
      }
      catch(Exception e) {}
    }
    available = true;  // Consumer가 큐에서 데이터를 회수한
    notify();          // 후에는 반드시 대기하고 있을 지도 모르는
    return buf;        // Producer 쓰레드를 깨움
  }
}
class SimplePC {
  public static void main(String args[]) {
    Queue q1 = new Queue();
    new Producer(q1).start(); // 두 쓰레드의 객체 공유
    new Consumer(q1).start();
  }
}
```

실행결과

```
Producer: 1
          Consumer: 1
Producer: 2
          Consumer: 2
Producer: 4
          Consumer: 4
Producer: 8
          Consumer: 8
Producer: 16
          Consumer: 16
Producer: 32
          Consumer: 32
Producer: 64
          Consumer: 64
Producer: 128
          Consumer: 128
Producer: 256
          Consumer: 256
Producer: 512
          Consumer: 512
```

1.8 쓰레드 그룹(ThreadGroup)

- 서로 관련된 쓰레드를 그룹으로 묶어서 다루기 위한 것
- 모든 쓰레드는 반드시 하나의 쓰레드 그룹에 포함되어 있어야 한다.
- 쓰레드 그룹을 지정하지 않고 생성한 쓰레드는 'main쓰레드 그룹'에 속한다.
- 자신을 생성한 쓰레드(조상 쓰레드)의 그룹과 우선순위를 상속받는다.

생성자 / 메서드	설 명
ThreadGroup(String name)	지정된 이름의 새로운 쓰레드 그룹을 생성한다.
ThreadGroup(ThreadGroup parent, String name)	지정된 쓰레드 그룹에 포함되는 새로운 쓰레드 그룹을 생성한다.
int activeCount()	쓰레드 그룹에 포함된 활성상태에 있는 쓰레드의 수를 반환한다.
void destroy()	쓰레드 그룹과 하위 쓰레드 그룹까지 모두 삭제한다.단, 쓰레드 그룹이나 하위 쓰레드 그룹이 비어있어야한다.
int getMaxPriority()	쓰레드 그룹의 최대우선순위를 반환한다.
String getName()	쓰레드 그룹의 이름을 반환한다.
ThreadGroup getParent()	쓰레드 그룹의 상위 쓰레드그룹을 반환한다.
void interrupt()	쓰레드 그룹에 속한 모든 쓰레드를 interrupt한다.
boolean isDaemon()	쓰레드 그룹이 데몬 쓰레드그룹인지 확인한다.
boolean isDestroyed()	쓰레드 그룹이 삭제되었는지 확인한다.
void list()	쓰레드 그룹에 속한 쓰레드와 하위 쓰레드그룹에 대한 정보를 출력한다.
boolean parentOf(ThreadGroup g)	지정된 쓰레드 그룹의 상위 쓰레드그룹인지 확인한다.
void setDaemon(boolean daemon)	쓰레드 그룹을 데몬 쓰레드그룹으로 설정/해제한다.
void setMaxPriority(int pri)	쓰레드 그룹의 최대우선순위를 설정한다.

27

1.9 데몬 쓰레드(daemon thread)

- 일반 쓰레드(non-daemon thread)의 작업을 돕는 보조적인 역할을 수행.
- 일반 쓰레드가 모두 종료되면 자동적으로 종료된다.
- 가비지 컬렉터, 자동저장, 화면자동갱신 등에 사용된다.
- 무한루프와 조건문을 이용해서 실행 후 대기하다가 특정조건이 만족되면 작업을 수행하고 다시 대기하도록 작성한다.

```
boolean isDaemon()  - 쓰레드가 데몬 쓰레드인지 확인한다. 데몬 쓰레드이면 true를 반환한다.
void setDaemon(boolean on)  - 쓰레드를 데몬 쓰레드로 또는 사용자 쓰레드로 변경한다.
                              (매개변수 on의 값을 true로 지정하면 데몬 쓰레드가 된다.)
```

* setDaemon(boolean on)은 반드시 start()를 호출하기 전에 실행되어야 한다.
  그렇지 않으면 IllegalThreadStateException이 발생한다.

```
public void run() {
    while(true) {
        try {
            Thread.sleep(3 * 1000); // 3초마다
        } catch(InterruptedException e) {}

        // autoSave의 값이 true이면 autoSave()를 호출한다.
        if(autoSave) {
            autoSave();
        }
    }
}
```

28

제 13 장
AWT

인제대학교 전자IT기계자동차공학부

황 원 주

5. 이벤트 처리

(Event handling)

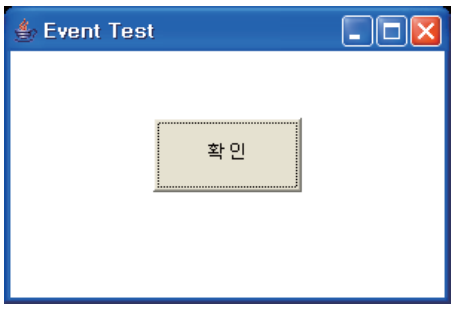
5.1 이벤트(Event)란?

- 사용자 또는 프로그램에 의해 발생할 수 있는 하나의 사건.

종류	설명
이벤트 소스 (Event Source, 이벤트 발생지)	이벤트가 발생한 컴포넌트. 사용자가 Button을 눌렀을 때 이벤트가 발생하고, Button은 이 이벤트의 이벤트 소스가 된다.
이벤트 핸들러 (Event Handler, 이벤트 처리기)	이벤트가 발생했을 때 실행될 코드를 구현해 놓은 클래스
이벤트 리스너 (Event Listener, 이벤트 감지기)	이벤트를 감지하고 처리한다. 이벤트 핸들러를 이벤트 리스너로 이벤트 소스에 연결해야 이벤트가 발생했을 때 이벤트가 처리된다.

```
class EventTest extends Frame {
    Button b = new Button("확 인");

    EventTest(String title) {
          super(title);
          setLayout(null);
          b.setBounds(100,75,100,50);
          // 이벤트 핸들러를 버튼의 이벤트 리스너를 등록한다.
          b.addActionListener(new EventHandler());

          add(b); // 버튼을 Frame에 추가한다.
          setSize(300, 200);
          setVisible(true);
    }

    public static void main(String args[]) {
          EventTest mainWin = new EventTest("Event Test");
    } // main
}
```

```
class EventHandler implements ActionListener { // 이벤트 핸들러
      public void actionPerformed(ActionEvent ae) {
            System.out.println("버튼이 눌러졌습니다.");
      }
}
```



5.2 이벤트 처리(Event handling)

- 이벤트가 발생했을 때, 어떤 작업이 수행되도록 코드를 작성하는 것

▶ 이벤트 처리 방법

1. 표13-25에 있는 메서드 중에서 필요한 것을 찾는다.

```
windowClosing(WindowEvent we)
```

2. 선택한 메서드가 속해있는 인터페이스를 구현하는 클래스를 작성한다.(표13-24 참고)

```
class EventHandler implements WindowListener {
      public void windowClosing(WindowEvent e) { /* 코드 작성 */  }
       ...
}
```

3. 위에서 구현한 클래스의 인스턴스를 생성해서 이벤트 소스에 Listener로 등록한다.

```
f.addWindowListener(new EventHandler());
```

이벤트	인터페이스	메서드
ActionEvent	ActionListener	actionPeformed(ActonEvent ae)
…	…	…
WindowEvent	WindowListener	windowClosing(WindowEvent we)
		windowOpened(WindowEvent we)
		windowIconified(WindowEvent we)
		windowDeiconified(WindowEvent we)
		windowClosed(WindowEvent we)
		windowActivated(WindowEvent we)
		windowDeactivated(WindowEvent we)
…	…	…

메서드	호출시기
actionPeformed(ActonEvent ae)	Button을 클릭했을 때, Menu를 클릭했을 때, TextField에서 Enter키를 눌렀을 때, List의 item하나를 선택하여 더블클릭했을 때
…	…
windowClosing(WindowEvent we)	윈도우가 닫힐 때(닫기 버튼을 눌렀을 때)
…	…

5.2 이벤트 처리(Event handling) - 예제

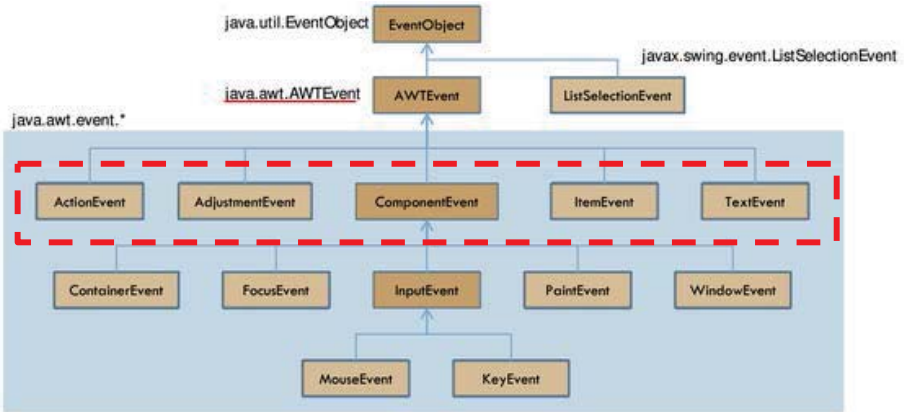
```
class FrameTest3 {
    public static void main(String args[]) {
        Frame f = new Frame("Login");
        f.setSize(300, 200);

        // EventHandler클래스의 객체를 생성해서 Frame의 WindowListener로 등록한다.
        f.addWindowListener(new EventHandler());
        f.setVisible(true);
    }
}
```


```
class EventHandler implements WindowListener {
    public void windowOpened(WindowEvent e) {}
    public void windowClosing(WindowEvent e) { // Frame의 닫기 버튼을 눌렀을 때 호출된다.
        e.getWindow().setVisible(false);       // Frame을 화면에서 보이지 않도록 하고
        e.getWindow().dispose();               // 메모리에서 제거한다.
        System.exit(0);                        // 프로그램을 종료한다.
    }
    public void windowClosed(WindowEvent e) {} // 아무내용도 없는 메서드 구현
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
}
```

사용자가 Frame의 닫기 버튼을 누르면,
1. WindowEvent가 발생하고(WindowEvent의 인스턴스가 생성됨),
2. Frame에 WindowListener로 등록되어 있는 이벤트 핸들러의 windowClosing메서드를 호출한다. 이 메서드 내에서는 이벤트 발생시 생성된 WindowEvent인스턴스의 참조를 사용할 수 있어서 WindowsEvent인스턴스의 메서드들을 사용할 수 있다.

5.3 ActionEvent

- 컴포넌트에 정의된 특정 동작이 수행되었을 때 발생하는 고수준 이벤트
- Button을 누르는 방법은 두 가지(마우스 클릭, spacebar누르기)가 있으며, Button을 누르면, MouseEvent나 KeyEvent가 발생하지만, ActionEvent도 발생한다.
- MouseEvent와 KeyEvent에 각각 별도의 이벤트처리를 하는 것보다 ActionEvent에만 이벤트처리를 하는 것이 낫다.(코드의 중복제거)



▶ ActionEvent가 발생하는 경우

- Button이 눌러졌을 때
- Menu를 클릭했을 때
- TextField에서 Enter키를 눌렀을 때
- List의 item하나를 선택하여 더블클릭했을 때

1. AWT
(Abstract Window Toolkit)

1.1 AWT(Abstract Window Toolkit)란?

▶ AWT

- GUI프로그래밍(윈도우 프로그래밍)을 위한 도구
- GUI프로그래밍에 필요한 다양한 컴포넌트를 제공한다.
- Java로 구현하지 않고, OS의 컴포넌트를 그대로 사용한다.

▶ Swing

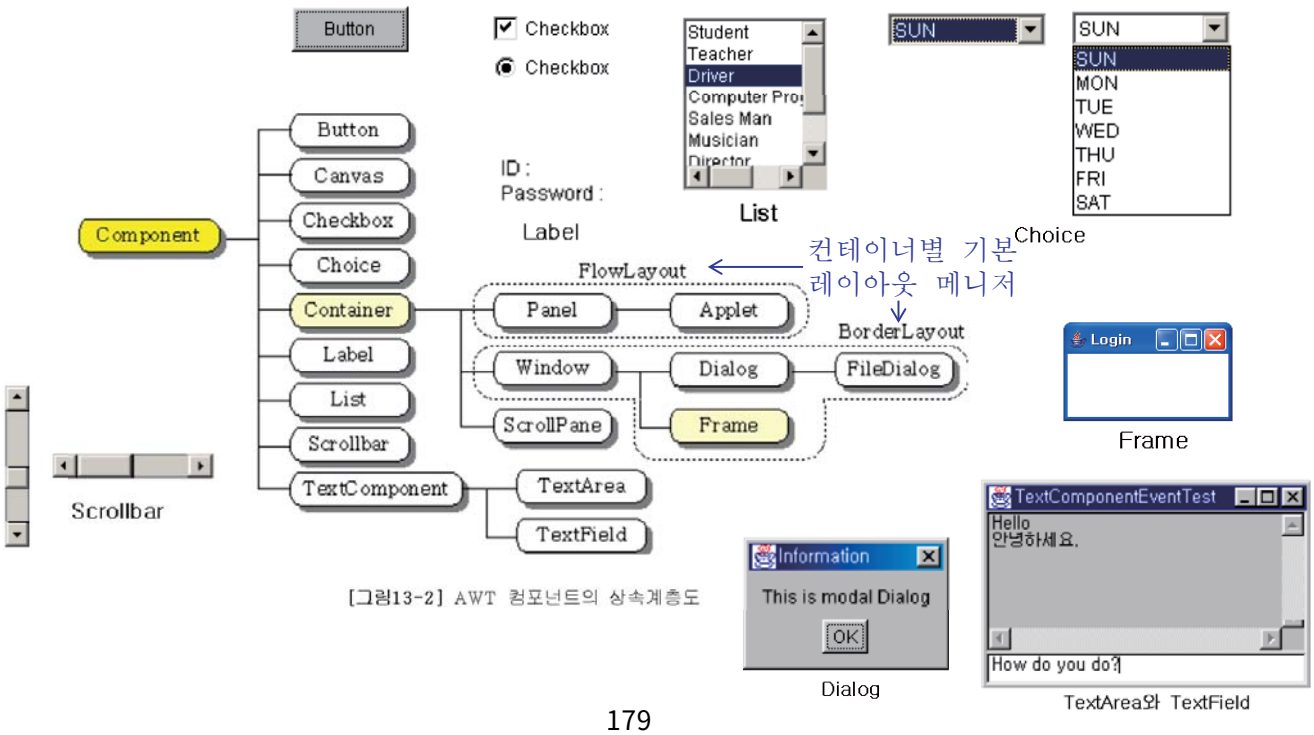
- AWT를 확장한 GUI프로그래밍 도구
- AWT보다 더 많은 종류의 컴포넌트를 제공한다.
- OS의 컴포넌트를 사용하지 않고, 순수한 Java로 구현하였다.

1.2 AWT의 구성

- AWT관련 패키지는 모두 'java.awt'로 시작한다.
- 'java.awt'패키지와 'java.awt.event'패키지가 AWT의 핵심이다.

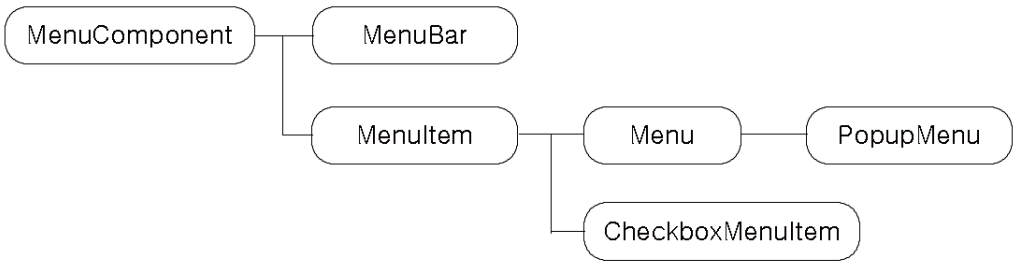
패키지명	설명
java.awt	AWT를 이용한 GUI어플리케이션을 작성하는데 필요한 기본적인 클래스와 컴포넌트를 제공한다.
java.awt.datatransfer	여러 어플리케이션 사이의, 또는 단일 어플리케이션 내에서의 데이터 전송을 구현하는데 필요한 클래스와 인터페이스를 제공한다.
java.awt.dnd	GUI의 장점 중의 하나인 끌어놓기(Drag and Drop)기능을 구현하는데 필요한 클래스들을 제공한다.
java.awt.event	GUI어플리케이션에서 발생하는 이벤트를 처리하는데 필요한 클래스와 인터페이스를 제공한다.
java.awt.font	폰트와 관련된 클래스와 인터페이스를 제공한다.
java.awt.image	이미지를 생성하거나 변경하는데 사용되는 클래스를 제공한다.
java.awt.print	출력에 관련된 클래스와 인터페이스를 제공한다.

- 모든 AWT컴포넌트의 최고 조상은 java.awt.Component클래스이다.
  (메뉴관련 컴포넌트 제외)
- Container는 다른 컴포넌트를 담을 수 있는 컴포넌트이다.

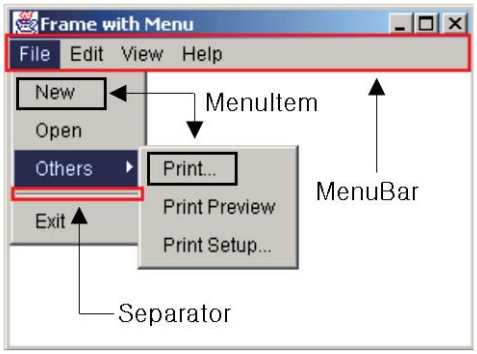


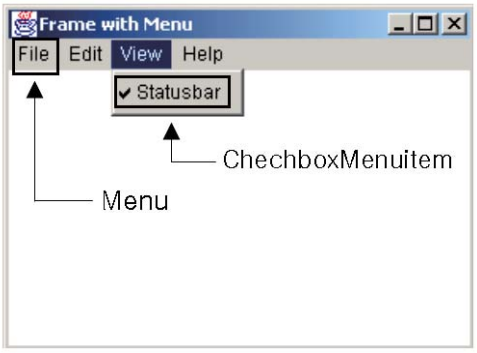
[그림13-2] AWT 컴포넌트의 상속계층도

- 메뉴관련 컴포넌트의 최고 조상은 java.awt.MenuComponent클래스이다.



[그림13-4] AWT 메뉴컴포넌트의 상속계층도





1.3 컴포넌트(Component)

- 모든 AWT컴포넌트(메뉴관련 컴포넌트 제외)의 최고 조상
- 컴포넌트라면 반드시 있어야 하는 공통적인 메서드들이 정의되어 있다.

메서드	설 명
Color getBackground() void setBackground(Color c)	컴포넌트의 배경색을 얻는다. 컴포넌트의 배경색을 지정된 색으로 변경한다.
Cursor getCursor() void setCursor(Cursor c)	컴포넌트에 지정된 커서를 얻는다. 컴포넌트의 커서(마우스포인터)를 지정한다.
Font getFont() void setFont(Font f)	컴포넌트에 지정되어 있는 Font를 얻는다. 컴포넌트의 Font를 지정한다.
Color getForeground() void setForeground(Color c)	컴포넌트의 전경색을 얻는다. 컴포넌트의 전경색을 지정한다.
int getHeigtht() int getWidth()	컴포넌트의 높이(Height)를 얻는다. 컴포넌트의 폭(Width)를 얻는다.
void setBounds(int x, int y, int width, int height) Rectangle getBounds()	컴포넌트의 위치(x, y)와 크기(width, height)를 지정한다. 컴포넌트의 위치와 크기(Rectangle객체)를 얻는다.
Point getLocation() void setLocation(int x, int y)	컴포넌트의 위치를 얻는다. 컴포넌트의 위치를 지정한다.
Dimension getSize() void setSize(int width, int height)	컴포넌트의 크기를 얻는다. 컴포넌트의 크기를 지정한다.
boolean hasFocus() void requestFocus()	컴포넌트가 현재 focus를 갖고 있는지 알려준다. 컴포넌트가 focus를 갖도록 한다.
void paint(Graphics g) void repaint()	컴포넌트를 화면에 그린다. 컴포넌트를 화면에 다시 그린다.
void setEnabled(boolean b)	컴포넌트를 사용가능(true)/불가능(false) 하게 한다.
Container getParent()	컴포넌트가 포함되어져 있는 컨테이너(parent)를 얻는다.
void setVisible(boolean b)	컴포넌트가 화면에 보이게(true)/보이지 않게(false) 한다.

1.4 컨테이너(Container)

- 다른 컴포넌트를 포함할 수 있는 컴포넌트. Container클래스와 그 자손들

1. 독립적인 컨테이너 – 독립적으로 사용될 수 있으며, 다른 컴포넌트나 종속적인 컨테이너를 포함할 수 있다.

컨테이너	설 명
Frame	가장 일반적인 컨테이너로 윈도우와 모양이 같다. titlebar와 크기조절버튼, 닫기버튼을 가지고 있다. 그리고 메뉴를 추가할 수 있다.
Window	Frame의 조상이며, 경계선, titlebar, 크기조절버튼, 닫기 버튼이 없으며, 메뉴도 추가할 수 없다. 단지 컴포넌트를 담을 수 있는 평면공간만을 갖는다.
Dialog	Frame처럼, titlebar와 닫기버튼을 갖고 있지만, 메뉴는 가질 수 없으며 기본적으로 크기를 변경할 수 없다. 주로 프로그램사용자에게 메시지를 보여 주거나, 응답을 받는데 사용한다.

2. 종속적인 컨테이너 – 독립적으로 사용될 수 없으며, 다른 컨테이너에 포함되어야 한다. 다른 컴포넌트나 종속적인 컨테이너를 포함할 수 있다.

컨테이너	설 명
Panel	평면공간으로 Frame과 같이 여러 컴포넌트를 담을 수 있으나 단독적으로 사용될 수는 없다.
ScrollPane	Panel과 같은 평면공간이지만, Panel과는 달리 단 하나의 컴포넌트만 포함할 수 있으며 자신보다 큰 컴포넌트가 포함되면 스크롤바가 자동적으로 나타난다.

▶ 컨테이너의 주요 메서드

- add()를 사용해서 컴포넌트를 컨테이너에 담는다.

- 컨테이너에 담긴 컴포넌트는 컨테이너의 전경색, 배경색, 폰트 등의 설정을 그대로 따르게 된다.(나중에 변경가능)

메서드	설 명
Component[] getComponents()	컨테이너에 포함되어 있는 모든 컴포넌트를 얻는다.
Component getComponent(int n)	컨테이너에 n번째로 추가된 컴포넌트를 얻는다.
Component getComponentAt(int x, int y)	컨테이너의 지정된 위치(x, y)에 있는 컴포넌트를 얻는다.
Component add(Component comp)	컨테이너에 컴포넌트를 추가한다.
void remove(Component comp)	컨테이너에서 지정된 컴포넌트를 제거한다.
Insets getInsets()	컨테이너의 경계의 크기를 알 수 있는 Inset객체를 얻는다.
LayoutManager getLayout()	컨테이너에 설정되어 있는 LayoutManager를 얻는다.
void setLayout(LayoutManager mgr)	컨테이너에 LayoutManager를 설정한다.

▶ AWT를 이용한 GUI 프로그램 작성 순서

1. AWT 프로그램 작성에 필요한 라이브러리를 import한다.

   import java.awt.*;

   import java.awt.event.*;

2. 반드시 적절한 "독립적인 컨테이너"를 설정한다. -> 가장 일반적인 컨테이너는 Frame
3. 필요에 따라 "종속적인 컨테이너"를 설정한다.
4. 컨테이너의 레이아웃을 설정한다. 독립적인 컨테이너에 컴포넌트를 부착하거나 레이아웃을 설정할 때는 먼저 ContentPane을 구하여야한다.
5. GUI 컴포넌트에 대해, 독립적인 컨테이너를 상속받은 클래스의

   (1) 멤버 변수에서

   - 객체 참조변수 선언

   (2) 생성자에서

   - 객체 생성
   - 컨테이너에 객체 포함 (add())
   - 이벤트 처리가 필요한 컴포넌트는 리스너에 등록 (addActionListener())
   - 기타 초기화와 관련된 내용 기술

6. 이벤트 처리 루틴 작성: 각 이벤트에 따라 자동적으로 호출되는 메서드에

   - GUI 컴포넌트의 종류에 따라 다름
   - 이벤트 발생 시 필요한 동작 구현

7. 컨테이너의 위치와 크기를 설정하고, setVisible(true) 메서드를 사용하여 컨테이너가 보여지도록 한다.

2. AWT의 주요 컴포넌트

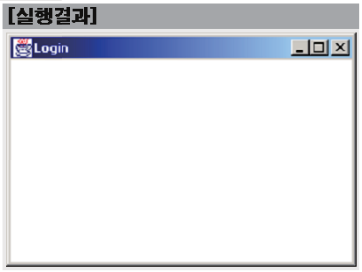
2.1 Frame

- titlebar와 최대화, 최소화, 닫기 버튼을 가진 윈도우(독립적인 컨테이너)

메서드 또는 생성자	설 명
Frame(String title)	Frame을 생성한다. title - Frame의 titlebar에 보여 질 text
String getTitle() void setTitle(String title)	titlebar에 있는 text를 얻는다. titlebar의 text를 변경한다.
void setState(int state)	Frame의 상태를 변경할 수 있으며, state에는 아래의 두 가지 값중 하나를 사용할 수 있다. Frame.ICONIFIED - Frame을 최소화 상태가 되게 한다. Frame.NORMAL - Frame을 정상적인 상태(최소화 이전상태)가 되게 한다.
int getState()	Frame의 현재 상태를 알 수 있다.
void setResizable(boolean resizable)	Frame의 크기를 변경가능 또는 불가능하게 한다. (resizable의 값이 false로 하면 사용자가 Frame의 크기를 변경할 수 없다.)

```
import java.awt.*;

class FrameTest {
    public static void main(String args[]) {
        Frame f = new Frame("Login");     // Frame객체를 생성한다.
        f.setSize(300, 200);              // Frame의 크기를 설정한다.
        f.setVisible(true);               // 생성한 Frame을 화면에 보이도록
    }
}
```

[실행결과]



2.2 Button

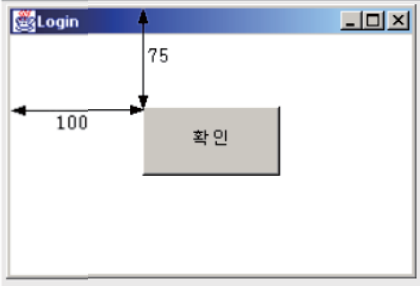
1. 언제 어떤 event가 발생되는가?
2. event발생시 어떤 메서드가 호출되는가?

- 사용자가 클릭했을 때 어떤 작업이 수행되도록 할 때 사용하는 컴포넌트
- 버튼을 마우스로 클릭하는 경우 ActionEvent가 발생되고, actionPerformed() 메서드가 호출

메서드 또는 생성자	설 명
Button(String label)	지정된 label을 가진 Button을 생성한다. label - Button위에 나타날 text
String getLabel() void setLabel(String label)	Button에 나타나있는 text를 얻는다. Button에 나타나있는 text를 변경한다.

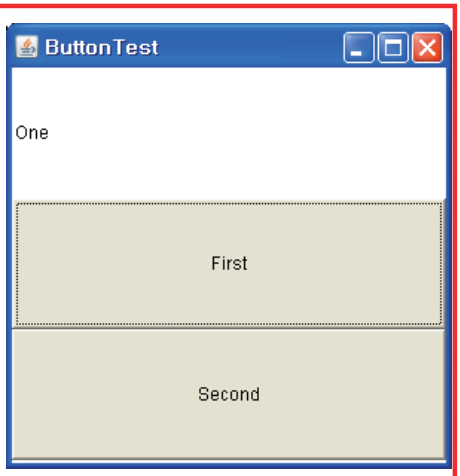
```
import java.awt.*;

class ButtonTest2 {
    public static void main(String args[]) {
        Frame f = new Frame("Login");
        f.setSize(300, 200);
        f.setLayout(null);          // 레이아웃 매니저의 설정을 해제한다.

        Button b = new Button("확 인");
        b.setSize(100, 50);         // Button의 크기를 설정한다.
        b.setLocation(100, 75);     // Frame내에서의 Button의 위치를 설정한다.

        f.add(b);
        f.setVisible(true);
    }
}
```

[실행결과]



►First와 Second라는 레이블을 가진 2개의 버튼과 1개의 레이블이 생성된다. 각 버튼을 클릭하면, One 또는 Two라는 문자열이 레이블에 출력된다.

예제 (매우 중요)




```
// 1. AWT 프로그램 작성에 필요한 라이브러리를 import한다.
import java.awt.*;
import java.awt.event.*;

//  2. 독립적인 컨테이너(Frame)를 상속받고, 리스너를 구현한다.
public class ButtonTest extends Frame implements ActionListener {

        //5-1. 객체 참조변수 선언
        Button btn1, btn2;
        Label lb;

        ButtonTest(String title) {
                super(title); // Frame(String title)을 호출한다.

                // 5-2. 객체 생성
                btn1 = new Button("First");
                btn2 = new Button("Second");
                lb = new Label();

                // 5-3. 컨테이너에 객체 포함
                add(lb);
                add(btn1);
                add(btn2);

                // 5-4. 이벤트 처리가 필요한 컴포넌트는 리스너에 등록
                btn1.addActionListener(this);
                btn2.addActionListener(this);

                // 5-5. 기타 초기화와 관련된 내용 기술

                //7. 컨테이너의 위치와 크기를 설정하고, 컨테이너가 보여지도록 한다.
                setSize(300, 300);
                setLayout(new GridLayout(3, 1));
                setVisible(true);
        }
```

예제 (매우 중요)

```
        // 6. 이벤트 처리 루틴 작성
        public void actionPerformed(ActionEvent e) {
                if (e.getActionCommand() == "First") {
                        lb.setText("One");
                } else if (e.getActionCommand() == "Second") {
                        lb.setText("Two");
                }
        }
}

class ButtonMain {
        public static void main(String[] args) {
                ButtonTest bt = new ButtonTest("ButtonTest");

                bt.addWindowListener(
                        new WindowAdapter() {
                                // 프레임의 닫기 버튼을 눌렀을 때 이벤트 처리
                                public void windowClosing(WindowEvent e) {
                                        System.exit(0);
                                }
                        }
                );
        }
}
```

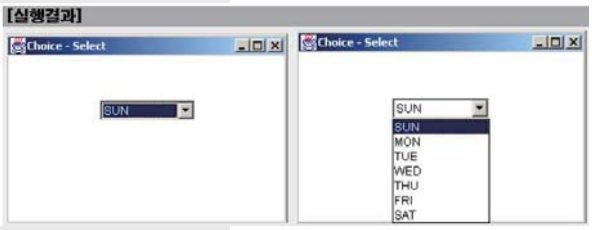
2.3 Choice

- 여러 item 중에서 하나를 선택할 수 있게 해주는 컴포넌트
- Choice에서 item을 선택하면 ItemEvent가 발생되고, itemStateChanged() 메서드가 호출

메서드 또는 생성자	설 명
void add(String item)	Choice에 item을 추가한다.
void remove(String item)	Choice에서 item을 제거한다.
void remove(int index)	지정된 순서에 있는 item을 제거한다. index는 0부터 시작
void removeAll()	Choice의 모든 item을 제거한다.
void insert(String item, int index)	지정된 순서에 item을 추가한다. index는 0부터 시작
String getItem(int index)	지정된 순서의 item을 얻는다. index는 0부터 시작
int getItemCount()	현재 Choice에 추가되어 있는 item이 몇 개인지 알려준다.
int getSelectedIndex()	현재 선택되어져 있는 item의 index값을 얻는다.
String getSelectedItem()	현재 선택되어져 있는 item을 얻는다.

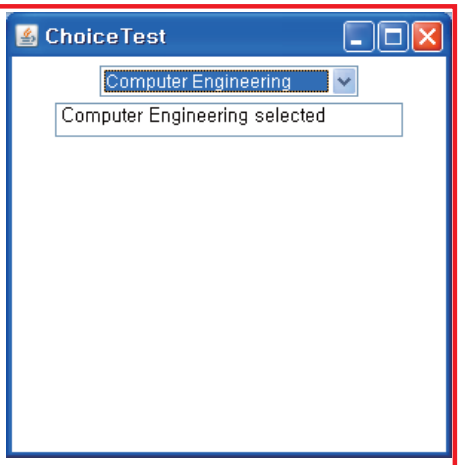
```
public static void main(String args[]) {
      Frame f = new Frame("Choice - Select");
      f.setSize(300, 200);
      f.setLayout(null);

      Choice day = new Choice();  // Choice객체를 생성한 다음
      day.add("SUN");             // Choice의 목록에 나타날 값들을 추가한다.
      day.add("MON");       day.add("TUE");
      day.add("WED");       day.add("THU");
      day.add("FRI");       day.add("SAT");

      day.setSize(100, 50);
      day.setLocation(100, 70);

      f.add(day);
      f.setVisible(true);
}
```

[실행결과]



►1개의 초이스와 1개의 텍스트필드가 생성된다. 각 초이스에서 하나의 메뉴를 선택하면, 전공명이 텍스트필드에 출력된다.

예제 (매우 중요)




```
// 1. AWT 프로그램 작성에 필요한 라이브러리를 import한다.
import java.awt.*;
import java.awt.event.*;

//  2. 독립적인 컨테이너(Frame)를 상속받고, 리스너를 구현한다.
public class ChoiceTest extends Frame implements ItemListener{
        //5-1. 객체 참조변수 선언
        Choice ch;
        TextField tf;

        ChoiceTest(String title) {
                super(title); // Frame(String title)을 호출한다.

                // 5-2. 객체 생성
                ch = new Choice();
                tf = new TextField(30);

                // 5-3. 컨테이너에 객체 포함
                add(ch);
                add(tf);

                // 5-4. 이벤트 처리가 필요한 컴포넌트는 리스너에 등록
                ch.addItemListener(this);

                // 5-5. 기타 초기화와 관련된 내용 기술
                ch.addItem("Computer Engineering");
                ch.addItem("Multimedia");
                ch.addItem("Computer Communication");
                ch.addItem("DataBase Design");
                ch.select(2); //초기 값을  Computer Communication으로 선택
```

예제 (매우 중요)

```
                //7. 컨테이너의 위치와 크기를 설정하고, 컨테이너가 보여지도록 한다.
                        setSize(300, 300);
                        setLayout(new FlowLayout());
                        setVisible(true);
                }

                // 6. 이벤트 처리 루틴 작성
                public void itemStateChanged(ItemEvent e) {
                        Choice tmp = (Choice) e.getItemSelectable();
                        tf.setText((tmp.getSelectedItem()+ " selected"));
                }
}

class ChoiceMain{
                public static void main(String[] args){
                        ChoiceTest ct = new ChoiceTest("ChoiceTest");

                        ct.addWindowListener(
                                new WindowAdapter(){
                                        // 프레임의 닫기 버튼을 눌렀을 때 이벤트 처리
                                        public void windowClosing(WindowEvent e){
                                                System.exit(0);
                                        }
                                }
                        );
                }
}
```

2.4 List - 메서드

- 여러 item 중에서 하나 또는 여러 개를 선택할 수 있게 해주는 컴포넌트
- List에서 item을 선택하면 ItemEvent가 발생되고,

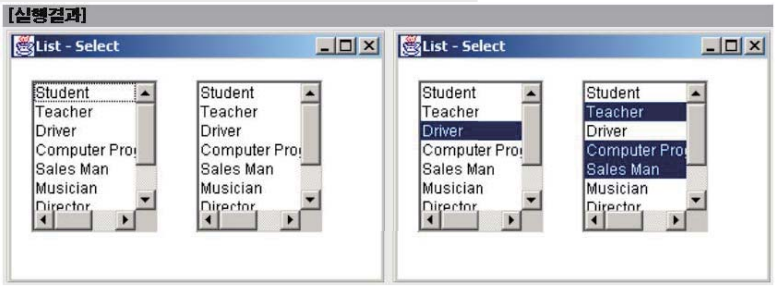
  itemStateChanged() 메서드가 호출

메서드 또는 생성자	설 명
List(int rows, boolean multipleMode)	rows - 몇 줄짜리 크기의 List를 보여줄 것인지 지정한다. multipleMode - List목록 다중선택이 가능하도록 할 것인가를 지정할 수 있다. true로 하면 여러 개의 item을 선택할 수 있다.
List(int rows)	List에 보여줄 item 수만 지정한다. multipleMode의 값은 false로 지정되어 하나의 목록만 선택가능하다.
List()	rows의 값은 기본값인 4로 지정되고, multipleMode의 값은 false가 되어 하나의 item만 선택가능하다.
void add(String item)	item을 List에 추가한다.
void add(String item, int index)	지정된 위치(index)에 item을 추가한다.
void replaceItem(String newValue, int index)	지정된 위치(index)의 item을 새로운 item(newValue)로 바꾼다.
void remove(String item)	List에서 해당 item을 제거한다.
void remove(int index)	index - 지정된 위치에 있는 item을 제거한다.
void removeAll()	List의 모든 item을 제거한다.
int getRows()	List에 scroll없이 볼 수 있는 item의 수를 얻는다.
String getItem(int index)	index - 지정된 위치에 있는 item을 얻는다.
String[] getItems()	List에 있는 모든 item을 얻는다.
int getItemCount()	List에 있는 item이 모두 몇 개인지 알려준다.(getRows()와 비교해볼 것)
void select(int index)	지정된 위치에 있는 item을 선택한다.
void deselect(int index)	지정된 위치에 있는 item을 선택해제한다.
int getSelectedIndex()	현재 선택되어 있는 item의 index값을 얻는다.
int[] getSelectedIndexes()	현재 선택되어 있는 item들의 index값을 얻는다.(List의 multipleMode가 true인 경우)
String getSelectedItem()	현재 선택되어 있는 item을 얻는다.
String[] getSelectedItems()	현재 선택되어 있는 item들을 얻는다.(multipleMode가 true인 경우)
boolean isIndexSelected(int index)	지정된 위치의 item이 선택되어있는지 알려준다.
void setMultipleMode(boolean b)	List를 multipleMode로 할 것인지 결정한다.b - true면, List를 multipleMode로 설정한다.

2.4 List - 예제

```
public static void main(String args[]) {
      Frame f = new Frame("List - Select");
      f.setSize(300, 200);
      f.setLayout(null);

      List selectOne = new List(6);     // 6개 목록을 보여줄 수 있는 크기의 List를 만든다.
      selectOne.setLocation(20,40);
      selectOne.setSize(100, 120);
      selectOne.add("Student");
      selectOne.add("Teacher");
      selectOne.add("Driver");
      selectOne.add("Computer Programmer");
      selectOne.add("Sales Man");
      selectOne.add("Musician");
      selectOne.add("Director");

      // 생성자의 두번째 인자값을 true로 설정해서 List의 목록에서 여러 개를 선택할 수 있게 했다.
      List selectMany = new List(6, true);
      selectMany.setLocation(150, 40);
      selectMany.setSize(100,120);
      selectMany.add("Student");
      selectMany.add("Teacher");
      // ... 내용 중간생략...

      f.add(selectOne);
      f.add(selectMany);
      f.setVisible(true);
}
```

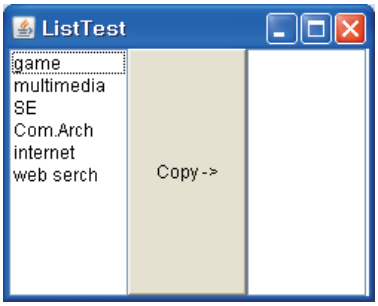
[실행결과]



►2개의 리스트와 1개의 버튼이 생성된다. 왼쪽 리스트에서 선택한 아이템들이 카피 버튼을 클릭하면 오른쪽 리스트로 복사된다.

예제 (매우 중요)

```
// 1. AWT 프로그램 작성에 필요한 라이브러리를 import한다.
import java.awt.*;
import java.awt.event.*;

//2. 독립적인 컨테이너(Frame)를 상속받고, 리스너를 구현한다.
public class ListTest extends Frame implements ActionListener{

        //5-1. 객체 참조변수 선언
        List fromlist,tolist;
        Button copybutton;
        String majorNames[] = {"game", "multimedia", "SE", "Com.Arch", "internet","web serch"};

        ListTest(String title) {
                super(title); // Frame(String title)을 호출한다.

                // 5-2. 객체 생성
                fromlist = new List(5,true);
                tolist = new List(5,false);
                copybutton = new Button("Copy ->");

                // 5-3. 컨테이너에 객체 포함
                add(fromlist);
                add(copybutton);
                add(tolist);

                // 5-4. 이벤트 처리가 필요한 컴포넌트는 리스너에 등록
                copybutton.addActionListener(this);
```



예제 (매우 중요)

```
                        // 5-5. 기타 초기화와 관련된 내용 기술
                        for(int i=0;i<majorNames.length;i++) {
                                fromlist.add(majorNames[i]);
                        }

                        //7. 컨테이너의 위치와 크기를 설정하고, 컨테이너가 보여지도록 한다.
                                setSize(250, 200);
                                setLayout(new FlowLayout());
                                setVisible(true);
                        }

        //  6. 이벤트 처리 루틴 작성
        public void actionPerformed(ActionEvent e)  {
                        String colors[];
                        colors = fromlist.getSelectedItems();
                        tolist.clear();
                        for(int i=0;i<colors.length;i++) {
                                tolist.add(colors[i]);
                        }
        }
}

class ListMain{
        public static void main(String[] args) {
                        ListTest lt = new ListTest("ListTest");

                        lt.addWindowListener(
                                new WindowAdapter() {
                                        // 프레임의 닫기 버튼을 눌렀을 때 이벤트 처리
                                        public void windowClosing(WindowEvent e) {
                                                System.exit(0);
                                        }
                                }
                        );
        }
}
```

2.5 Label

- 화면에 텍스트를 표시하는데 사용되는 컴포넌트

메서드 또는 생성자	설 명
Label(String text, int alignment)	text - 화면에 나타낼 글(text)을 String으로 넣는다. alignment -text의 정렬방식을 지정한다. Label.LEFT, Label.CENTER, Label.RIGHT 중 하나를 사용.
Label(String text)	text - 화면에 나타낼 text를 지정한다. alignment의 기본값인 Label.LEFT로 설정된다.
String getText()	Label의 text를 얻어온다.
void setText(String text)	Label의 text를 주어진 값으로 변경한다.
void setAlignment(int alignment)	Label의 text 정렬을 지정한다. Label.LEFT, Label.CENTER, Label.RIGHT 중 하나를 사용.

```
public static void main(String args[]) {
      Frame f = new Frame("Login");
      f.setSize(300, 200);
      f.setLayout(null);

      Label id = new Label("ID :");      // Label을 생성하고 크기와 위치를 지정한다.
      id.setBounds(50, 50, 30, 10);      // 50, 50위치에 크기가 가로 30, 세로 10

      Label pwd = new Label("Password :");
      pwd.setBounds(50, 65, 100, 10);

      f.add(id);                         // 생성한 Label을 Frame에 포함시킨다.
      f.add(pwd);
      f.setVisible(true);
}
```

[실행결과]

2.6 Checkbox - 메서드

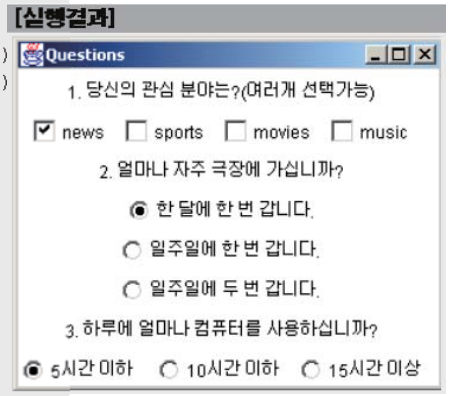
- '선택/비선택'의 상태를 표현하는데 사용되는 컴포넌트. (여러 개 선택 가능)
- CheckboxGroup을 사용하면 여러 값 중의 하나만 선택 가능
- Checkbox의 상태가 바뀔 때 마다 ItemEvent가 발생되고, itemStateChanged() 메서드가 호출

메서드 또는 생성자	설 명
Checkbox(String text, boolean state)	text - Checkbox와 함께 보여질 text를 지정한다. state - true이면 Checkbox가 선택된 상태로 생성되고, false이면 Checkbox가 선택해제된 상태로 생성된다.
Checkbox(String text)	Checkbox와 함께 보여질 text를 지정한다. Checkbox는 선택해제된 상태로 생성된다.
Checkbox()	text없이 Checkbox만 나타나고, Checkbox는 선택해제된 상태로 생성된다.
Checkbox(String text, CheckboxGroup group, boolean state)	group - CheckboxGroup객체의 참조. CheckboxGroup을 이용해서 radio button으로 만든다.
String getLabel()	Checkbox의 label을 얻는다.
void setLabel(String label)	Checkbox의 label을 주어진 값으로 변경한다.
boolean getState()	Checkbox의 상태를 얻는다. 결과값이 true면 체크되어있는 상태이다.
void setState(boolean state)	Checkbox의 상태를 설정한다. state값을 true로 하면 Checkbox가 체크되어 있는 상태가 되도록 한다.

2.6 Checkbox - 예제

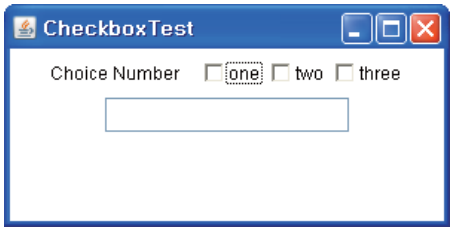
```
public static void main(String args[]) {
      Frame f = new Frame("Questions");
      f.setSize(305, 250);
      // Frame의 LayoutManager를 FlowLayout으로 설정한다.
      f.setLayout(new FlowLayout());

      Label q1 = new Label("1. 당신의 관심 분야는?(여러개 선택가능)");
      Checkbox news = new Checkbox("news", true); // 선택된 상태로 생성
      Checkbox sports = new Checkbox("sports");
      Checkbox movies = new Checkbox("movies");
      Checkbox music = new Checkbox("music");

      f.add(q1); f.add(news); f.add(sports); f.add(movies); f.add(music);

      Label q2 = new Label("2. 얼마나 자주 극장에 가십니까?");
      CheckboxGroup group1 = new CheckboxGroup();
      Checkbox movie1 = new Checkbox("한 달에 한 번 갑니다.", group1, true);
      Checkbox movie2 = new Checkbox("일주일에 한 번 갑니다.", group1, false);
      Checkbox movie3 = new Checkbox("일주일에 두 번 갑니다.", group1, false);

      f.add(q2); f.add(movie1); f.add(movie2); f.add(movie3);

      Label q3 = new Label("3. 하루에 얼마나 컴퓨터를 사용하십니까?");
      CheckboxGroup group2 = new CheckboxGroup();
      Checkbox com1 = new Checkbox("5시간 이하 ", group2, true);
      Checkbox com2 = new Checkbox("10시간 이하", group2, false);
      Checkbox com3 = new Checkbox("15시간 이상", group2, false);

      f.add(q3); f.add(com1); f.add(com2); f.add(com3);
      f.setVisible(true);
}
```

[실행결과]



►3개의 체크박스와 1개의 텍스트 필드가 생성된다. 체크박스가 선택되거나 선택 취소될 경우 상태변환을 나타내는 메시지가 텍스트 필드에 출력된다.

예제 (매우 중요)




```
// 1. AWT 프로그램 작성에 필요한 라이브러리를 import한다.
import java.awt.*;
import java.awt.event.*;

//2. 독립적인 컨테이너(Frame)를 상속받고, 리스너를 구현한다.
public class CheckboxTest extends Frame implements ItemListener{

        //5-1. 객체 참조변수 선언
        Label lb;
        Checkbox one, two, three;
        TextField tf;
        String msg = "CheckBox";


        CheckboxTest(String title) {
                super(title); // Frame(String title)을 호출한다.

                // 5-2. 객체 생성
                lb = new Label("Choice Number");
                one = new Checkbox("one",false);
                two = new Checkbox("two",false);
                three = new Checkbox("three",false);
                tf = new TextField(20);

                // 5-3. 컨테이너에 객체 포함
                add(lb);      add(one);
                add(two);     add(three);
                add(tf);

                // 5-4. 이벤트 처리가 필요한 컴포넌트는 리스너에 등록
                one.addItemListener(this);
                two.addItemListener(this);
                three.addItemListener(this);

                // 5-5. 기타 초기화와 관련된 내용 기술
```

예제 (매우 중요)

```
                //7. 컨테이너의 위치와 크기를 설정하고, 컨테이너가 보여지도록 한다.
                setSize(300, 150);
                setLayout(new FlowLayout());
                setVisible(true);
        }

        //  6. 이벤트 처리 루틴 작성
        public void itemStateChanged(ItemEvent e)  {
                if(e.getItem().equals("one")) {
                        msg = msg +"1";
                }
                else if(e.getItem().equals("two")) {
                        msg = msg +"2";
                }
                else {
                        msg = msg +"3";
                }

                if(e.getStateChange() == ItemEvent.SELECTED) {
                        msg=msg+" is selected";
                } else {
                        msg = msg + " is deselected";
                }
                tf.setText(msg);
                msg = "CheckBox ";
        }

}

class view{
        public static void main(String[] args) {
                CheckboxTest CT = new CheckboxTest("CheckboxTest");
                CT.addWindowListener(
                                new WindowAdapter() {
                                        // 프레임의 닫기 버튼을 눌렀을 때 이벤트 처리
                                        public void windowClosing(WindowEvent e) {
                                                System.exit(0);
                                        }
                                }
                        );
                }
        }
```

2.7 TextField - 메서드

- 사용자로부터 데이터를 자유롭게 입력받을 수 있는 컴포넌트
- 한 줄만 입력할 수 있어서 비교적 길지 않은 값의 입력에 사용된다.
- TextField에 text를 입력한 후 엔터키를 치면 ActionEvent가 발생되고, actionPerformed() 메서드가 호출

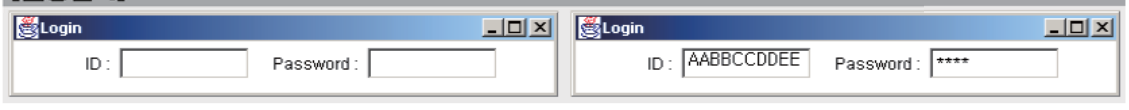
메서드 또는 생성자	설 명
TextField(String text, int col)	text - TextField에 보여질 text를 지정한다. col - 입력 받을 글자의 수를 적는다. col의 값에 따라서 TextField의 크기가 결정된다.
TextField(int col)	col - 입력 받을 글자의 수를 적는다.
TextField(String text)	text - TextField에 보여질 text를 지정한다.
void setEchoChar(char c)	지정된 문자를 EchoChar로 한다. (비밀번호 입력에 주로 사용됨)
int getColumns()	TextField의 칼럼 수를 얻는다.
void setText(String t)*	지정된 문자열을 TextField의 text로 한다.
String getText()*	TextField의 text를 얻는다.
void select(int selectionStart, int selectionEnd)*	selectionStart부터 selectionEnd까지의 text를 선택(하이라이트)한다.
void selectAll()*	TextField의 모든 text를 선택된 상태가 되도록 한다.
String getSelectedText()*	TextField의 text중 선택되어진 부분을 얻는다.
void setEditable(boolean b)*	TextField의 text를 편집가능(true)/불가능(false) 하도록 한다.

[참고] '*' 표시가 되어있는 것은 TextField의 조상인 TextComponent로부터 상속받은 것이다.

2.7 TextField - 예제

```
import java.awt.*;

class TextFieldTest {
    public static void main(String args[]) {
          Frame f = new Frame("Login");
          f.setSize(400, 65);
          f.setLayout(new FlowLayout()); // LayoutManager를 FlowLayout으로 한다.

          Label lid = new Label("ID :", Label.RIGHT); // 정렬을 오른쪽으로.
          Label lpwd = new Label("Password :", Label.RIGHT);

          TextField id = new TextField(10);//약 10개의 글자를 입력할 수 있는 TextField 생성
          TextField pwd = new TextField(10);
          pwd.setEchoChar('*');      // 입력한 값 대신 '*'가 보이도록 한다.

          f.add(lid);      // 생성한 컴포넌트들을 Frame에 포함시킨다.
          f.add(id);
          f.add(lpwd);
          f.add(pwd);
          f.setVisible(true);
    }
}
```

[실행결과]



►2개의 텍스트필드가 생성된다. 한쪽의 텍스트필드에 텍스트를 입력하고 엔터키를 치면 입력한 문자열이 반대편 텍스트필드에 이동되어 출력된다.

예제 (매우 중요)

```
// 1. AWT 프로그램 작성에 필요한 라이브러리를 import한다.
import java.awt.*;
import java.awt.event.*;

//2. 독립적인 컨테이너(Frame)를 상속받고, 리스너를 구현한다.
public class TextFieldTest extends Frame implements ActionListener{

        //5-1. 객체 참조변수 선언
        TextField tf1, tf2;

        TextFieldTest(String title) {
                super(title); // Frame(String title)을 호출한다.

                // 5-2. 객체 생성
                tf1 = new TextField(20);
                tf2 = new TextField(20);

                // 5-3. 컨테이너에 객체 포함
                add(tf1);
                add(tf2);

                // 5-4. 이벤트 처리가 필요한 컴포넌트는 리스너에 등록
                tf1.addActionListener(this);
                tf2.addActionListener(this);

                // 5-5. 기타 초기화와 관련된 내용 기술
```

예제 (매우 중요)

```
                //7. 컨테이너의 위치와 크기를 설정하고, 컨테이너가 보여지도록 한다.
                setSize(500, 150);
                setLayout(new GridLayout());
                setVisible(true);
        }

        //  6. 이벤트 처리 루틴 작성
        public void actionPerformed(ActionEvent e) {
                if(e.getSource() instanceof TextField) {
                        TextField t = (TextField)e.getSource();
                        if(t==tf1) {
                                tf2.setText(t.getText());
                                tf1.setText("");
                        }else if(t==tf2) {
                                tf1.setText(t.getText());
                                tf2.setText("");
                        }
                }
        }
}

class TextFieldMain{
        public static void main(String[] args) {
                TextFieldTest tft = new TextFieldTest("TextFieldTest");

                tft.addWindowListener(
                        new WindowAdapter() {
                                // 프레임의 닫기 버튼을 눌렀을 때 이벤트 처리
                                public void windowClosing(WindowEvent e) {
                                        System.exit(0);
                                }
                        }
                );
        }
}
```

2.8 TextArea - 메서드

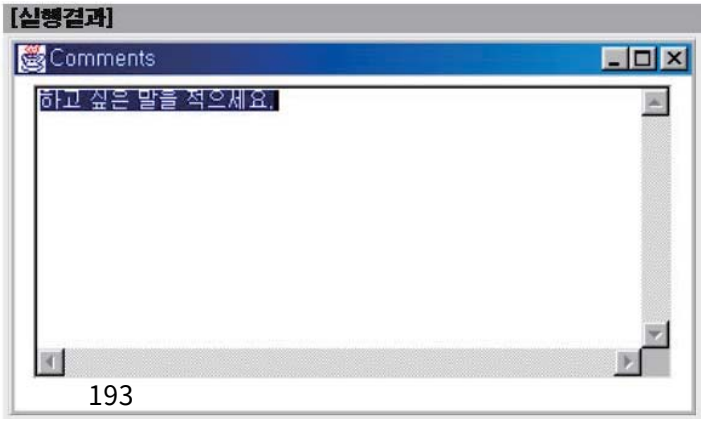
- 여러 줄의 텍스트를 입력하거나 보여줄 수 있는 편집가능한 컴포넌트
- TextArea에 문자 하나를 입력할 때마다 TextEvent가 발생되고, textValueChanged() 메서드가 호출

메서드 또는 생성자	설 명
TextArea(String text, int row, int col, int scrollbar)	text - TextArea에 보여질 text를 지정한다. row - TextArea의 줄(row) 수를 지정한다. col - TextArea의 열(column) 수를 적는다. scrollbar - TextArea에 사용할 scrollbar의 종류와 사용여부 지정. (아래의 값들 중 하나) TextArea.SCROLLBARS_BOTH,TextArea.SCROLLBARS_NONE TextArea.SCROLLBARS_HORIZONTAL_ONLY, TextArea.SCROLLBARS_VERTICAL_ONLY
TextArea(int row, int col)	row - TextArea의 줄(row) 수를 지정한다. col - TextArea의 열(column) 수를 적는다. 아무런 text도 없는 빈 TextArea를 생성한다. scrollbar는 수평(HORIZONTAL), 수직(VERTICAL) 모두 갖는다.
int getRows()	TextArea의 행(row)의 개수를 얻는다.
int getColumns()	TextArea의 열(column)의 개수를 얻는다.
void setRows(int rows)	지정된 값으로 TextArea의 행(row)의 개수를 설정한다.
void setColumns(int columns)	지정된 값으로 TextArea의 열(column)의 개수를 설정한다.
void append(String str)	TextArea에 있는 text의 맨 마지막에 문자열을 덧붙인다.
void insert(String str, int pos)	TextArea에 있는 text의 지정된 위치에 문자열을 넣는다.
void replaceRange(String str, int start, int end)	TextArea에 있는 text의 start부터 end범위에 있는 문자열을 str에 지정된 값으로 변경한다.
void setText(String t)*	지정된 문자열을 TextArea의 text로 한다.
String getText()*	TextArea의 text를 얻는다.
void select(int selectionStart, int selectionEnd)*	selectionStart부터 selectionEnd까지의 text가 선택되게 한다.
void selectAll()*	TextArea의 모든 text를 선택되게 한다.
String getSelectedText()*	TextArea의 text중 선택된 부분을 얻는다.
void setEditable(boolean b)*	TextArea의 text를 편집가능(true)/불가능(false) 하도록 한다.

2.8 TextArea - 예제

```
import java.awt.*;

class TextAreaTest {
    public static void main(String args[]) {
          Frame f = new Frame("Comments");
          f.setSize(400, 220);
          f.setLayout(new FlowLayout());

          TextArea comments = new TextArea("하고 싶은 말을 적으세요.", 10, 50);

          f.add(comments);
          comments.selectAll();  // TextArea의 text 전체가 선택 되도록 한다.
          f.setVisible(true);
    }
}
```

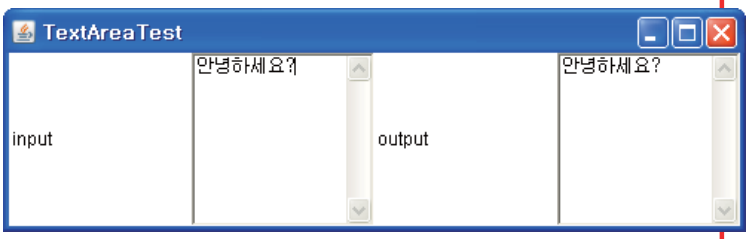
[실행결과]



►2개의 텍스트영역과 2개의 레이블이 생성된다. 왼쪽 텍스트영역에 문자를 입력하면 오른쪽 텍스트영역에 입력한 내용을 복사하여 출력한다.

예제 (매우 중요)

```
// 1. AWT 프로그램 작성에 필요한 라이브러리를 import한다.
import java.awt.*;
import java.awt.event.*;

//2. 독립적인 컨테이너(Frame)를 상속받고, 리스너를 구현한다.
public class TextAreaTest extends Frame implements TextListener {

        //5-1. 객체 참조변수 선언
        TextArea ts,td;
        Label lb1,lb2;

        TextAreaTest(String title) {
                super(title); // Frame(String title)을 호출한다.

                // 5-2. 객체 생성
                ts = new TextArea(5,20);
                td = new TextArea(5,20);
                lb1 = new Label("input");
                lb2 = new Label("output");

                // 5-3. 컨테이너에 객체 포함
                add(lb1);
                add(ts);
                add(lb2);
                add(td);

                // 5-4. 이벤트 처리가 필요한 컴포넌트는 리스너에 등록
                ts.addTextListener(this);

                // 5-5. 기타 초기화와 관련된 내용 기술
```



예제 (매우 중요)

```
        //7. 컨테이너의 위치와 크기를 설정하고, 컨테이너가 보여지도록 한다.
                setSize(500, 150);
                setLayout(new GridLayout());
                setVisible(true);
        }

        //  6. 이벤트 처리 루틴 작성
        public void textValueChanged(TextEvent e) {
                TextComponent source = (TextComponent) e.getSource();
                td.setText(source.getText());
        }
}

class TextAreaMain{
        public static void main(String[] args) {
                TextAreaTest tat = new TextAreaTest("TextAreaTest");

                tat.addWindowListener(
                        new WindowAdapter() {
                                // 프레임의 닫기 버튼을 눌렀을 때 이벤트 처리
                                public void windowClosing(WindowEvent e) {
                                        System.exit(0);
                                }
                        }
                );
        }
}
```

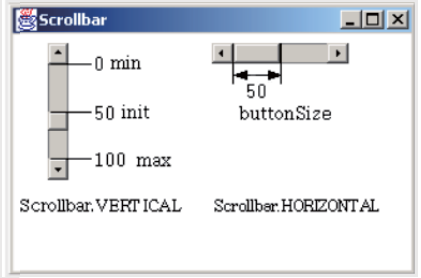
2.9 Scrollbar

- 사용자가 정해진 범위에서 값을 조절할 수 있게 해주는 컴포넌트
- 사용자가 Scrollbar를 움직여서 원하는 위치에 놓으면 AdjustmentEvent가 발생되고, adjustmentValueChanged() 메서드가 호출

메서드 또는 생성자	설 명
Scrollbar() Scrollbar(int orientation) Scrollbar(int orientation, int value, int visible, int min, int max)	orientation - Scrollbar의 종류를 지정해준다. Scrollbar.VERTICAL, Scrollbar.HORIZONTAL 중의 하나 value - Scrollbar의 초기값 visible - Scroll버튼(bubble)의 크기 min - Scrollbar가 가질 수 있는 최소값 max - Scrollbar가 가질 수 있는 최대값
int getValue()	현재 설정된 Scrollbar의 값을 얻어온다.
void setValue(int newValue)	Scrollbar의 값을 지정된 값(newValue)으로 설정한다.

```
public static void main(String args[]) {
	Frame f = new Frame("Scrollbar");
	f.setSize(300, 200);
	f.setLayout(null);

	Scrollbar hor = new Scrollbar(Scrollbar.HORIZONTAL, 0, 50, 0, 100);
	hor.setSize(100, 15);
	hor.setLocation(60, 30);
	Scrollbar ver = new Scrollbar(Scrollbar.VERTICAL, 50, 20, 0, 100);
	ver.setSize(15, 100);
	ver.setLocation(30, 30);

	f.add(hor);
	f.add(ver);
	f.setVisible(true);
}
```

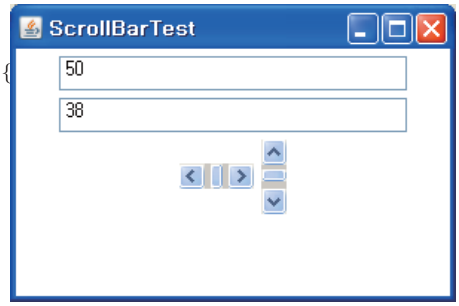
[실행결과]



►1개의 수평 스크롤바와 1개의 수직 스크롤바, 그리고 2개의 텍스트필드가 생성된다. 스크롤바를 움직여서 놓으면 해당 값이 텍스트필드에 출력된다.

예제 (매우 중요)

```
// 1. AWT 프로그램 작성에 필요한 라이브러리를 import한다.
import java.awt.*;
import java.awt.event.*;

//2. 독립적인 컨테이너(Frame)를 상속받고, 리스너를 구현한다.
public class ScrollBarTest extends Frame implements AdjustmentListener{

		//5-1. 객체 참조변수 선언
		Scrollbar hs,vs;
		TextField tf1,tf2;

		ScrollBarTest(String title) {
			super(title); // Frame(String title)을 호출한다.

			// 5-2. 객체 생성
			hs = new Scrollbar(Scrollbar.HORIZONTAL,100,100,0,200);
			vs = new Scrollbar(Scrollbar.VERTICAL,100,100,0,200);
			tf1 = new TextField(30);
			tf2 = new TextField(30);

			// 5-3. 컨테이너에 객체 포함
			add(tf1);
			add(tf2);
			add(hs);
			add(vs);

			// 5-4. 이벤트 처리가 필요한 컴포넌트는 리스너에 등록
			hs.addAdjustmentListener(this);
			vs.addAdjustmentListener(this);
```



예제 (매우 중요)

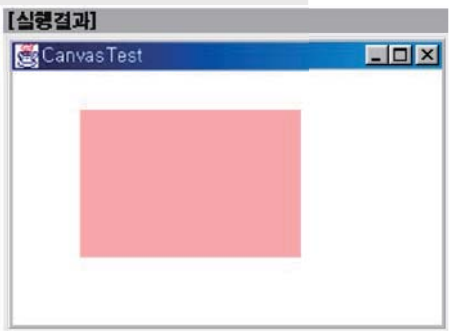
```
                    // 5-5. 기타 초기화와 관련된 내용 기술

                    //7. 컨테이너의 위치와 크기를 설정하고, 컨테이너가 보여지도록 한다.
                    setSize(300, 200);
                    setLayout(new FlowLayout());
                    setVisible(true);
        }

        // 6. 이벤트 처리 루틴 작성
        public void adjustmentValueChanged(AdjustmentEvent e) {
                    Scrollbar s = (Scrollbar)e.getSource();
                    if(s.getOrientation() == Scrollbar.HORIZONTAL){
                                tf1.setText(String.valueOf(s.getValue()));
                    }
                    else{
                                tf2.setText(String.valueOf(s.getValue()));
                    }
        }

}

class ScroolBarMain{
        public static void main(String[] args){
                    ScrollBarTest sbt = new ScrollBarTest("ScrollBarTest");
                    sbt.addWindowListener(
                                new WindowAdapter(){
                                            // 프레임의 닫기 버튼을 눌렀을 때 이벤트 처리
                                            public void windowClosing(WindowEvent e){
                                                        System.exit(0);
                                            }
                                }
                    );
        }
}
```

2.10 Canvas

- 주로 그림을 그리거나 이미지를 위한 공간으로 사용되는 컴포넌트

```
import java.awt.*;

class CanvasTest {
    public static void main(String args[]) {
          Frame f = new Frame("CanvasTest");
          f.setSize(300, 200);
          f.setLayout(null);          // Frame의 Layout Manager설정을 해제한다.

          Canvas c = new Canvas();
          c.setBackground(Color.pink);    // Canvas의 배경을 분홍색(pink)으로 한다.
          c.setBounds(50, 50, 150, 100);

          f.add(c);         // Canvas을 Frame에 포함시킨다.
          f.setVisible(true);
    }
}
```

[실행결과]



4. 레이아웃 매니저
(Layout Manager)

4.1 레이아웃 매니저를 이용한 컴포넌트 배치

- 레이아웃 매니저는 컨테이너에 포함된 컴포넌트의 배치를 자동관리한다.
- 레이아웃 매니저를 사용하면 컨테이너의 크기가 변경되거나 새로운 컴포넌트가 추가될 때, 컴포넌트를 재배치하는 코드를 작성할 필요가 없다.

- AWT에서는 아래와 같이 5개의 레이아웃 매니저를 제공한다.

BorderLayout, FlowLayout, GridLayout, CardLayout, GridbagLayout

► 컨테이너별 기본 레이아웃 매니저

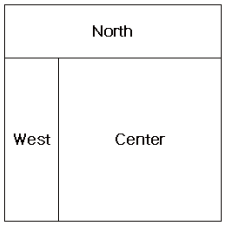
FlowLayout - Panel, Applet

BorderLayout - Window, Dialog, Frame

4.2 BorderLayout

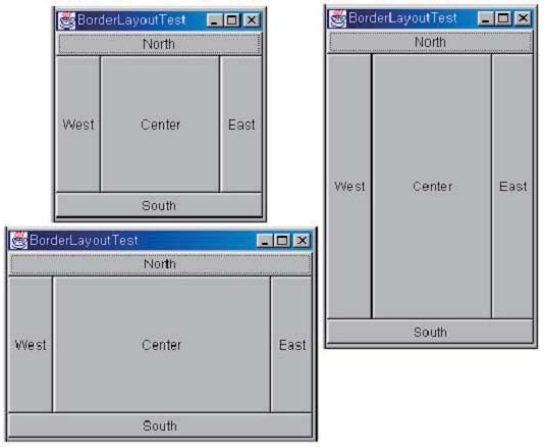
- 모두 5개의 영역으로 나누고, 각 영역에 하나의 컴포넌트만 넣을 수 있다.
- 한 영역에 하나 이상의 컴포넌트를 넣으려면 Panel을 사용하면 된다.

메서드 또는 생성자	설 명
BorderLayout(int hgap, int vgap)	각 영역 사이에 간격이 있는 BorderLayout을 생성한다. hgap - 각 영역의 사이에 간격을 준다.(좌우) vgap - 각 영역의 사이에 간격을 준다.(위아래)
BorderLayout()	영역 사이에 간격이 없는 BorderLayout을 생성한다.
add(String name, Component c) 또는 add(Component c, String name)	c - 추가하려는 컴포넌트 name- "North", "South", "East", "West", "Center" 중의 하나 또는 BorderLayout.NORTH, BorderLayout.SOUTH, BorderLayout.EAST, BorderLayout.WEST, BorderLayout.CENTER 중의 하나




```
Frame f = new Frame("BorderLayoutTest");
f.setSize(200, 200);
// Frame은 기본적으로 BorderLayout로 설정되어 있으므로 따로 설정하지 않아도 됨
f.setLayout(new BorderLayout());
Button north = new Button("North");
Button south = new Button("South");
Button east = new Button("East");
Button west = new Button("West");
Button center = new Button("Center");

// Frame의 5개의 각 영역에 Button을 하나씩 추가한다.
f.add(north, "North"); // f.add("North",north);와 같이 쓸 수도 있다.
f.add(south, "South"); // South의 대소문자 정확히
f.add(east, "East");   // East대신, BorderLayout.EAST 사용가능
f.add(west, "West");
f.add(center, "Center");
```

위치



4.3 FlowLayout

- 컴포넌트를 워드프로세서와 같은 방식, 즉 왼쪽에서 오른쪽으로 배치한다.
- 3가지 정렬방식(왼쪽, 가운데, 오른쪽)이 가능하다.

메서드 또는 생성자	설 명
FlowLayout(int align, int hgap, int vgap)	align - 컴포넌트들의 정렬 방법을 지정한다. FlowLayout.LEFT(왼쪽), FlowLayout.CENTER(가운데),FlowLayout.RIGHT(오른쪽) 중 하나 hgap - 각 컴포넌트간의 사이에 간격을 준다.(좌우) vgap - 각 컴포넌트간의 사이에 간격을 준다.(위아래)
FlowLayout(int align)	align - 컴포넌트들의 정렬 방법을 지정한다. hgap과 vgap이 5픽셀인 FlowLayout을 생성한다.
FlowLayout()	가운데 정렬이면서 hgap과 vgap이 5픽셀인 FlowLayout을 생성한다.

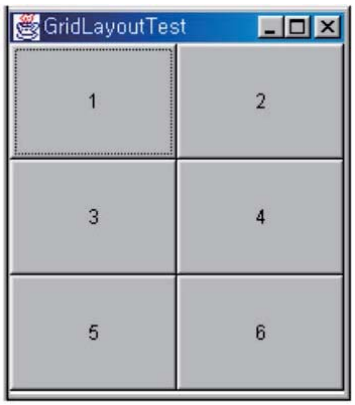
```
Frame f = new Frame("FlowLayoutTest");
f.setSize(250, 100);
f.setLayout(new FlowLayout(FlowLayout.LEFT));
f.add(new Button("첫 번째"));
f.add(new Button("두 번째"));
f.add(new Button("세 번째"));
f.add(new Button("네 번째"));
f.add(new Button("다섯 번째"));
f.setVisible(true);
```

4.4 GridLayout

- 컴포넌트를 워드프로세서와 같은 방식, 즉 왼쪽에서 오른쪽으로 배치한다.
- 3가지 정렬방식(왼쪽, 가운데, 오른쪽)이 가능하다.

메서드 또는 생성자	설 명
GridLayout(int row, int col, int hgap, int vgap)	영역들 간의 사이에 간격이 있는 GridLayout을 생성한다. row - 컨테이너를 몇 개의 행(row)으로 나눌 것인지 적는다. col - 컨테이너를 몇 개의 열(column)로 나눌 것인지 적는다. hgap - 각 영역간의 사이에 간격을 준다.(좌우) vgap - 각 영역간의 사이에 간격을 준다.(위아래)
GridLayout(int row, int col)	영역들 간의 사이에 간격이 없는 GridLayout을 생성한다. row - 컨테이너를 몇 개의 행(row)으로 나눌 것인지 적는다. col - 컨테이너를 몇 개의 열(column)로 나눌 것인지 적는다.

```
Frame f = new Frame("GridLayoutTest");
f.setSize(150, 150);
f.setLayout(new GridLayout(3, 2)); // 3행 2열의 테이블을 만든다.
f.add(new Button("1")); // 추가되는 순서대로 Button에 번호를 붙였다.
f.add(new Button("2"));
f.add(new Button("3"));
f.add(new Button("4"));
f.add(new Button("5"));
f.add(new Button("6"));
```



7. 애플릿(Applet)

▶ 애플릿과 AWT를 이용한 GUI 프로그램 작성 순서

1. 애플릿과 AWT 프로그램 작성에 필요한 라이브러리를 import한다.

   import java.awt.*;

   import java.awt.event.*;

   import java.applet.Applet;

2. 필요에 따라 "독립적인 컨테이너"를 설정한다. -> Applet클래스는 Container클래스에서 상속되었으므로 보통은 애플릿상에 컴포넌트를 부착한다.
3. 필요에 따라 "종속적인 컨테이너"를 설정한다.
4. 애플릿(또는 컨테이너)의 레이아웃을 설정한다. AWT 컨테이너의 레이아웃 관리자를 이용한다.
5. 애플릿의 init()에
   - 객체생성
   - 애플릿에 객체 포함 (add())
   - 이벤트 처리가 필요한 컴포넌트는 리스너에 등록 (addActionListener())
   - 기타 초기화와 관련된 내용 기술
6. 이벤트 처리 루틴 작성: 각 이벤트에 따라 자동적으로 호출되는 메서드에
   - GUI 컴포넌트의 종류에 따라 다름
   - 이벤트 발생 시 필요한 동작 구현

7.1 애플릿(Applet)이란?

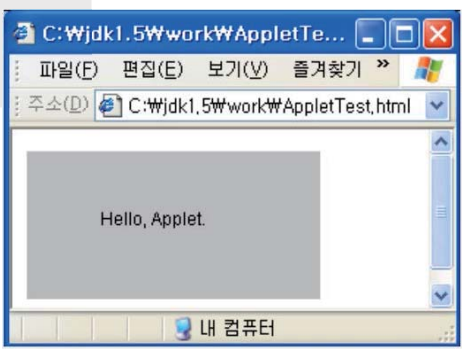
- 웹브라우저를 통해 실행될 수 있는 '작은 어플리케이션(Applet)'

1. 애플릿 관련정보가 포함된 HTML문서를 작성해야한다.
2. java.exe가 아닌 웹브라우저를 통해 실행된다.
3. public static void main(String args[])이 필요없다.
4. 애플릿은 java.applet.Applet을 상속하는 public클래스이어야 한다.

```
import java.awt.*;

public class HelloApplet extends java.applet.Applet
{
    public void paint(Graphics g) {
        setBackground(Color.lightGray); // 애플릿의 바탕을 밝은 회색으로 설정
        g.drawString("Hello, Applet.",50,50);
    }
}
```

AppletTest.html

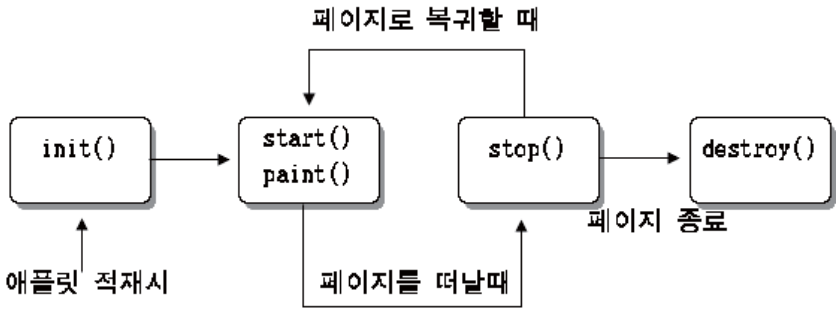
```
<html>
    <applet code="HelloApplet.class" width=200 height=100>
    </applet>
</html>
```



7.2 Applet의 생명주기(Life cycle)

- 애플릿이 담긴 HTML페이지가 브라우저에 로딩되면서 애플릿은 시작된다.

1. 웹브라우저가 애플릿이 포함된 HTML문서를 읽는다.
2. 웹브라우저가 애플릿(자바클래스)을 다운로드한다.
3. 웹브라우저가 애플릿의 인스턴스를 생성한다.
4. 애플릿이 초기화된다. - init()이 호출된다.
5. 애플릿이 실행된다. - start()가 호출된다.



애플릿의 메서드	설 명
init()	애플릿이 생성될 때 호출된다. 객체생성이나 이미지 로딩 등, 애플릿의 초기화 작업에 사용된다. 생성자 다음에 호출된다.
start()	애플릿의 실행이 시작 또는 재시작될 때 호출된다. 웹브라우저가 아이콘화되었다가 화면에 다시 나타날 때 그리고 init()이 호출된 직 후에는 반드시 호출된다.
stop()	애플릿의 실행을 중지시킨다. 웹브라우저가 아이콘화되던가, 다른 페이지로 이동할 때, 그리고 destroy()가 호출되기 직전에 반드시 호출된다.
destroy()	웹브라우저가 닫히면서 애플릿이 소멸되기 직전에 stop() 다음으로 호출된다. 애플릿이 사용하던 자원을 반환하는 용도로 사용된다.

[표13-29] 애플릿의 생명주기와 관련된 메서드