Java

제 7 강

다차원배열 & 객체지향개념 I-1

0. 다차원배열(array)
    0.1 다차원 배열 예제
    0.2 다차원 배열과 마방진
1. 객체지향언어란?
    1.1 객체지향언어의 역사
    1.2 객체지향언어의 특징
2. 클래스와 객체
    2.1 클래스와 객체의 정의와 용도
    2.2 객체와 인스턴스
    2.3 객체의 구성요소 – 속성과 기능
    2.4 인스턴스의 생성과 사용
    2.5 클래스의 또 다른 정의

0.1 다차원 배열의 예제

-'[]'의 개수가 차원의 수를 의미한다.

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

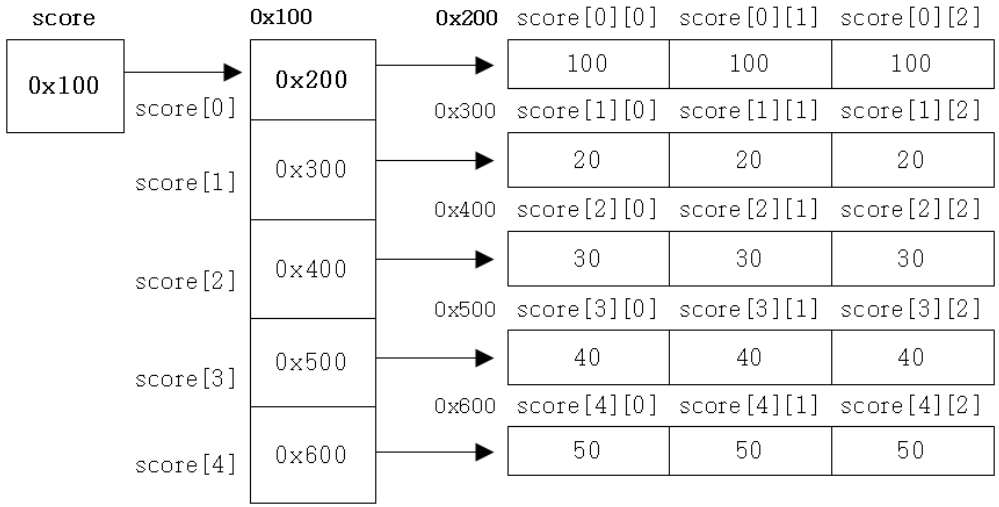
[표5-3] 2차원 배열의 선언

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

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

	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;
    }
}
```



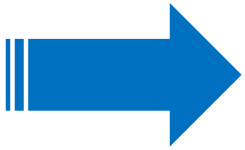
[그림5-2] 2차원 배열

0.2 다차원 배열과 예제

-5*5 배열을 선언하고 다음과 같이 초기값을 입력하고 오른쪽과 같이 출력되도록 자바 프로그램을 작성하기

그림

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25



1	2	3	4	5
10	9	8	7	6
11	12	13	14	15
20	19	18	17	16
21	22	23	24	25

0.3 다차원 배열과 마방진

-마방진이란? 아래 그림처럼 가로 세로 대각선의 합이 같은 배열의 형태이다

8	1	6
3	5	7
4	9	2

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

0.3 마방진의 조건

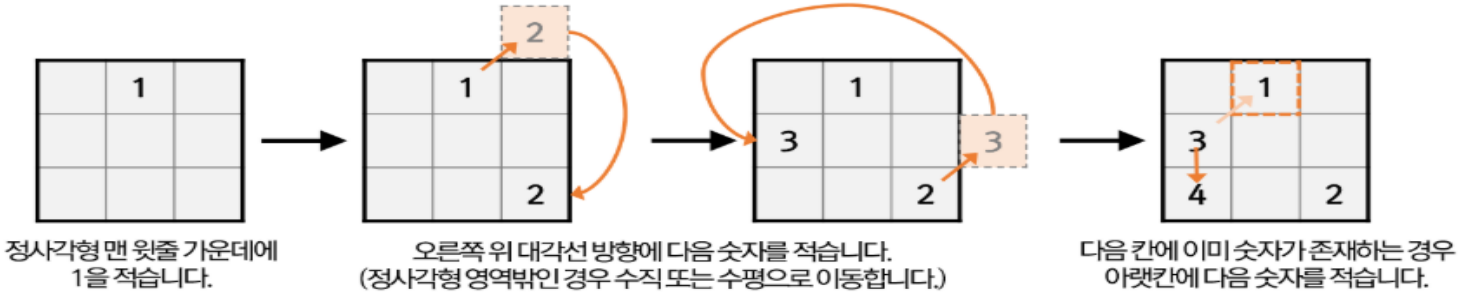
- 가로 세로가 홀수인 마방진을 다룬다
- 행의 길이가 N 이라고 하면 NxN 인 배열이 된다
- 가로 세로 대각선의 합의 결과값은 N(NxN+1) / 2
- 1 부터 NxN 까지의 값이 들어간다

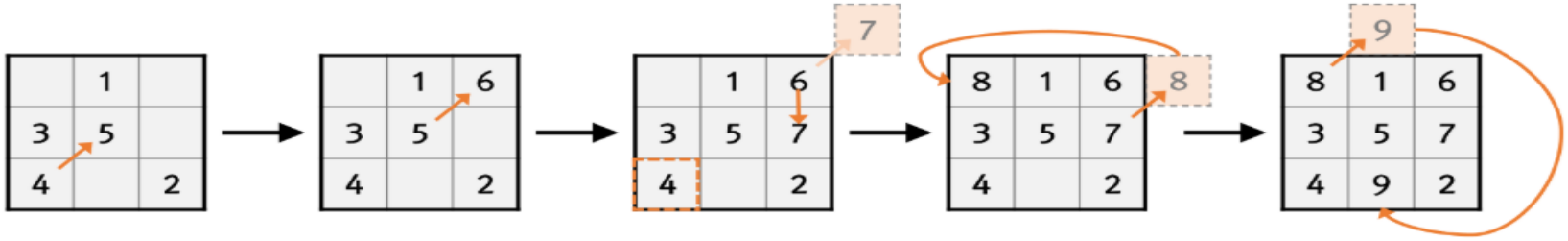
0.3 마방진의 알고리즘

1 – 첫번째 숫자인 1을 맨윗 행 중간열에 배치

2 – 두번째 수부터는 오른쪽 대각선 위로 이동하며 숫자를 배치. 행렬의 범위를 넘어가면 반대쪽 끝으로 돌아옴

3 – 만약 숫자를 놓아야하는 자리에 이미 다른 숫자가 배치되어 있다면 현재 위치에서 행을 하나 내린 같은 열의 위치에 숫자를 놓음. 다음 숫자는 다시 2번을 수행





1. 객체지향언어란?
2. 클래스와 객체

→ 객체지향개념 I-1

3. 변수와 메서드
4. 메서드 오버로딩

→ 객체지향개념 I-2

5. 생성자
6. 변수의 초기화

→ 객체지향개념 I-3

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

▶ 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.

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

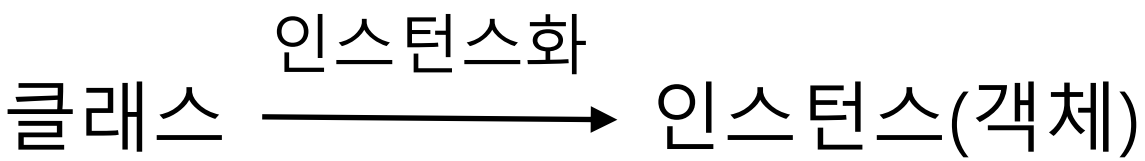
- 객체(object)는 인스턴스(instance)를 포함하는 일반적인 의미

책상은 인스턴스다.
책상은 객체다.

책상은 책상 클래스의 객체다.
책상은 책상 클래스의 인스턴스다.

▶ 인스턴스화(instantiate, 인스턴스化)

- 클래스로부터 인스턴스를 생성하는 것.



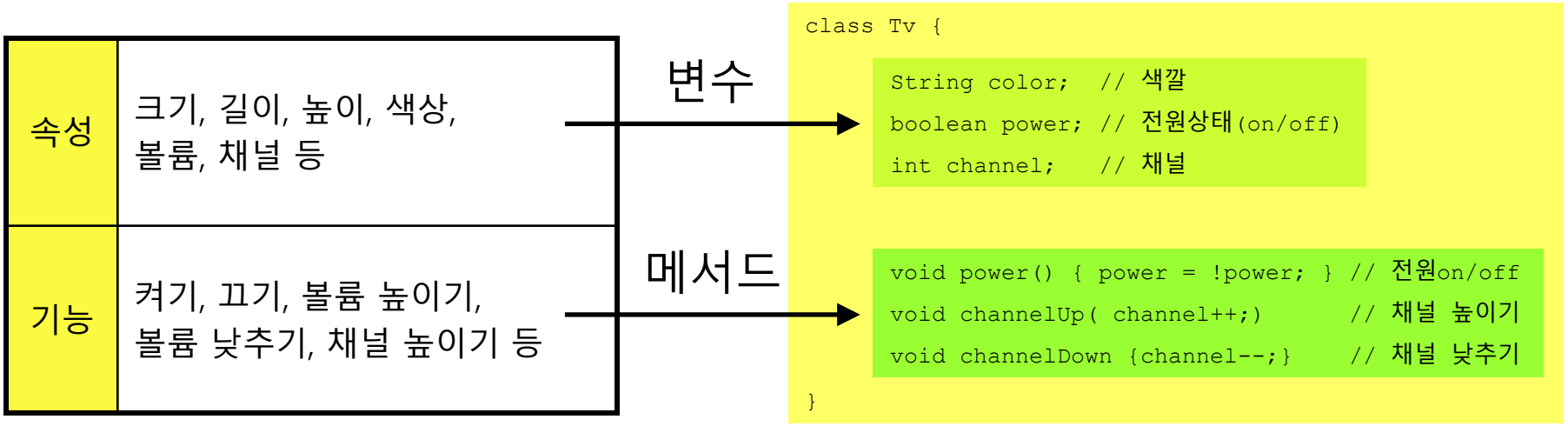
2.3 객체의 구성요소 – 속성과 기능

▶ 객체는 속성과 기능으로 이루어져 있다.

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

▶ 속성은 변수로, 기능은 메서드로 정의한다.

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



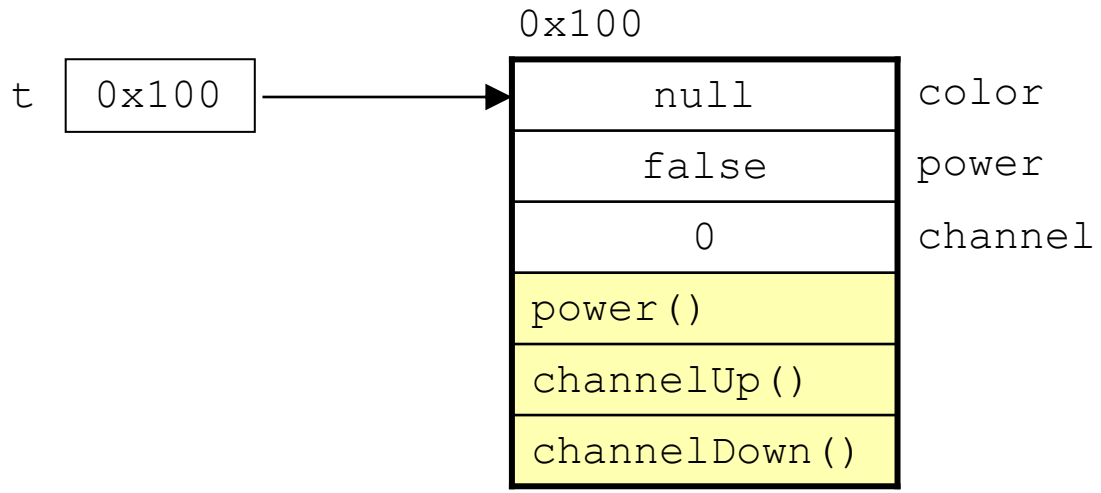
2.4 인스턴스의 생성과 사용(1/4)

▶ 인스턴스의 생성방법

클래스명 참조변수명; // 객체를 다루기 위한 참조변수 선언

참조변수명 = new 클래스명(); // 객체생성 후, 생성된 객체의 주소를 참조변수에 저장

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

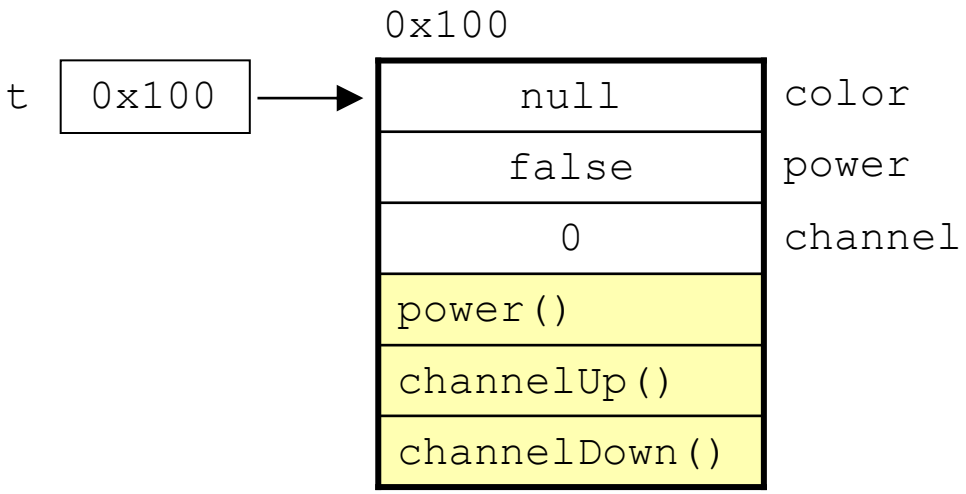



```
Tv t = new Tv();
```

2.4 인스턴스의 생성과 사용(2/4)

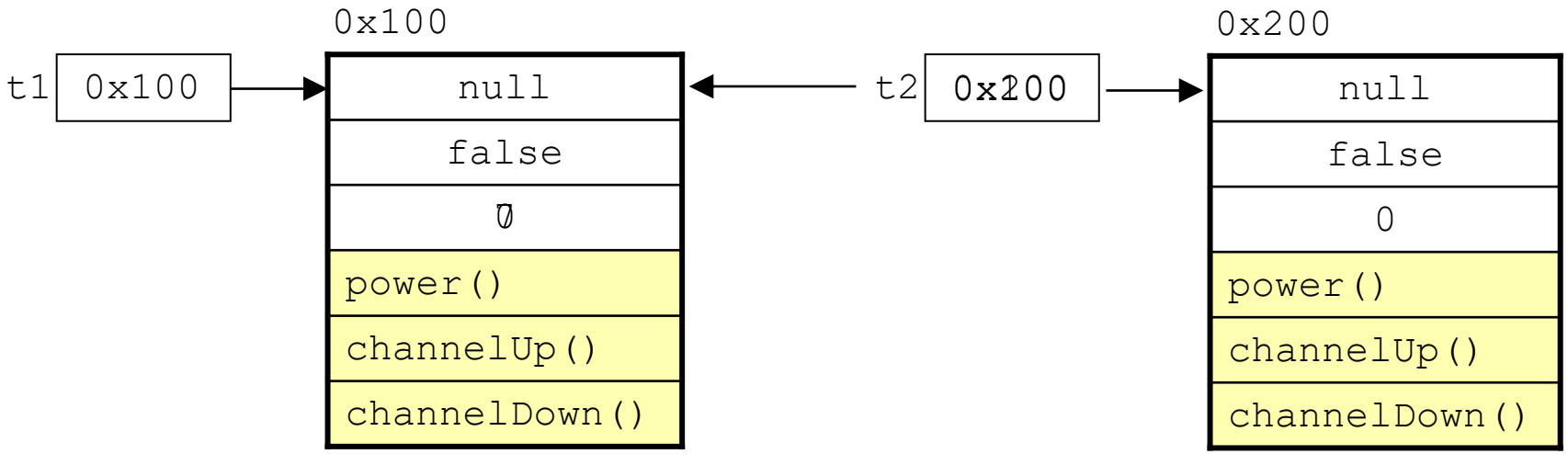
```
Tv t;
t = new Tv();
t.channel = 7;
t.channelDown();
System.out.println(t.channel);
```

```
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
}
```

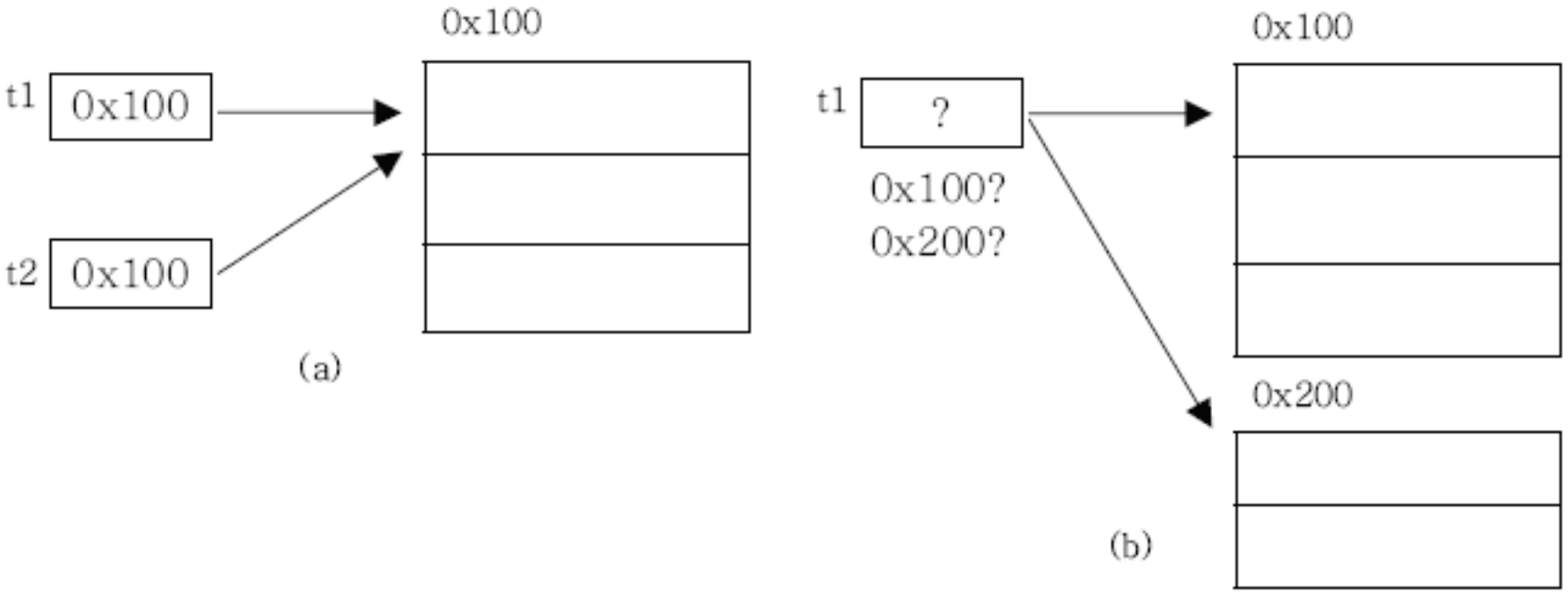


2.4 인스턴스의 생성과 사용(3/4)

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



2.4 인스턴스의 생성과 사용(4/4)

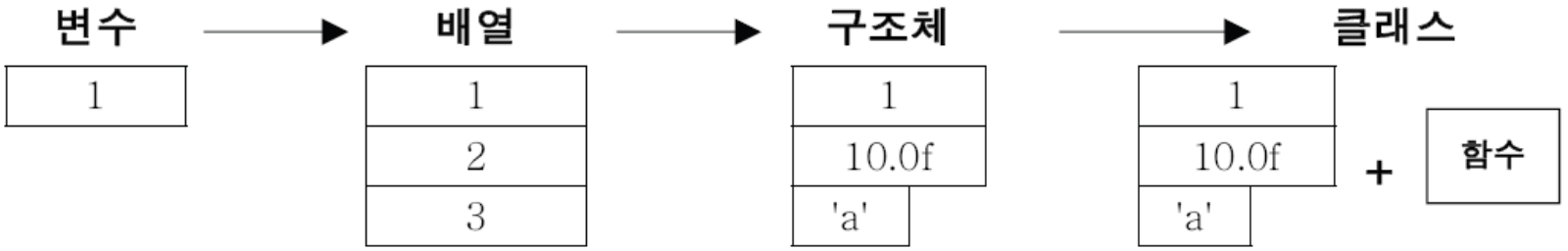


(a) 하나의 인스턴스를 여러 개의 참조변수가 가리키는 경우(가능)
(b) 여러 개의 인스턴스를 하나의 참조변수가 가리키는 경우(불가능)

[그림6-2] 참조변수와 인스턴스의 관계

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)

2.5 클래스의 또 다른 정의

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();
```

2.5 클래스의 또 다른 정의

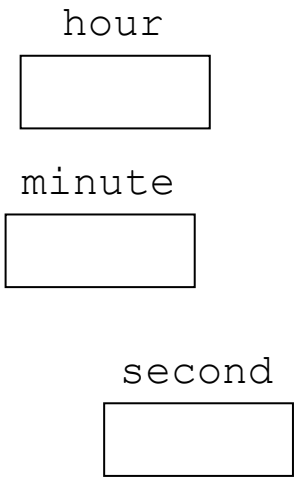
2. 클래스 – 사용자 정의 타입(User-defined type)

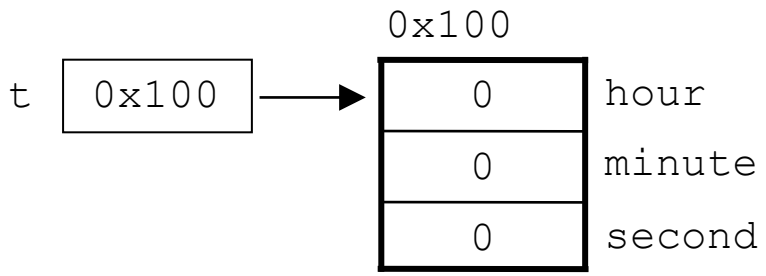
```
int hour;
int minute;
int second;
```

→

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

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





2.5 클래스의 또 다른 정의

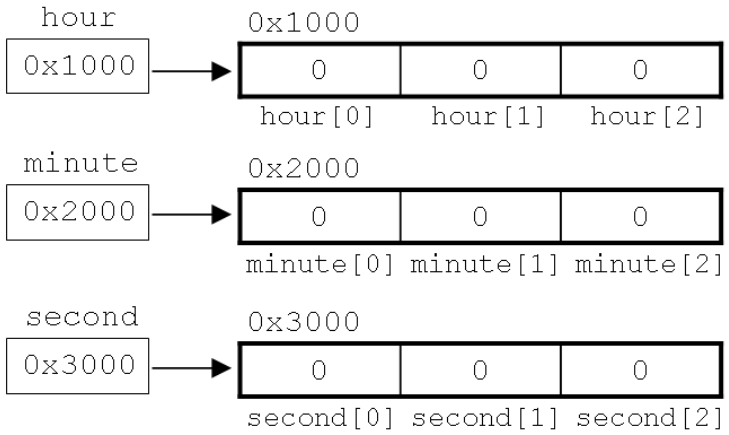
2. 클래스 – 사용자 정의 타입(User-defined type)

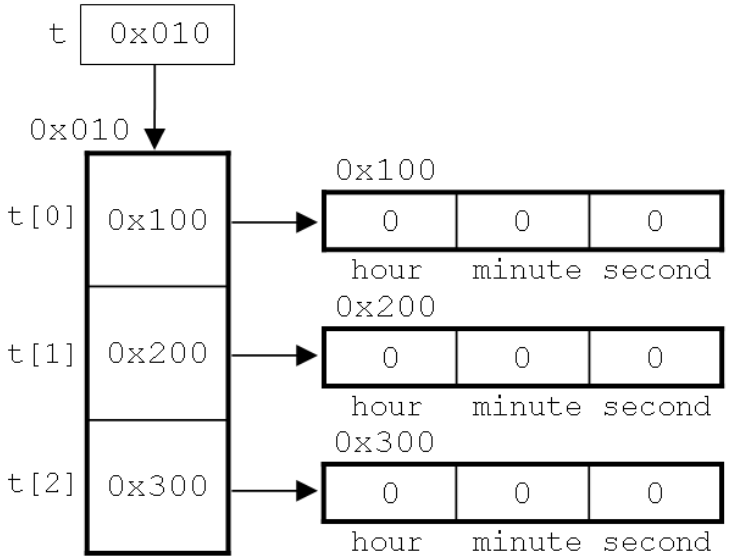
```
int[] hour = new int[3];
int[] minute = new int[3];
int[] second = new int[3];
```

→

```
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;
}
```





감사합니다.