

# 자바 프로그래밍 기초

임성국 ([eventia@gmail.com](mailto:eventia@gmail.com))

# 자바 : 클래스와 객체



## 액체지향개념 I-1

- 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
- 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

# 1. 객체지향언어란?

## 1.1 객체지향언어의 역사

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



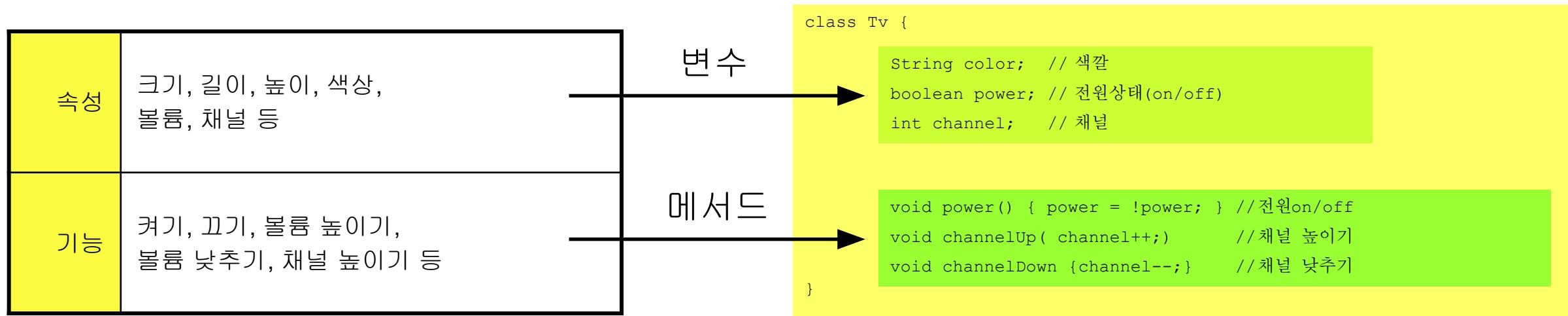
## 2.3 객체의 구성요소 – 속성과 기능

▶ 객체는 속성과 기능으로 이루어져 있다.

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

▶ 속성은 변수로, 기능은 메서드로 정의한다.

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



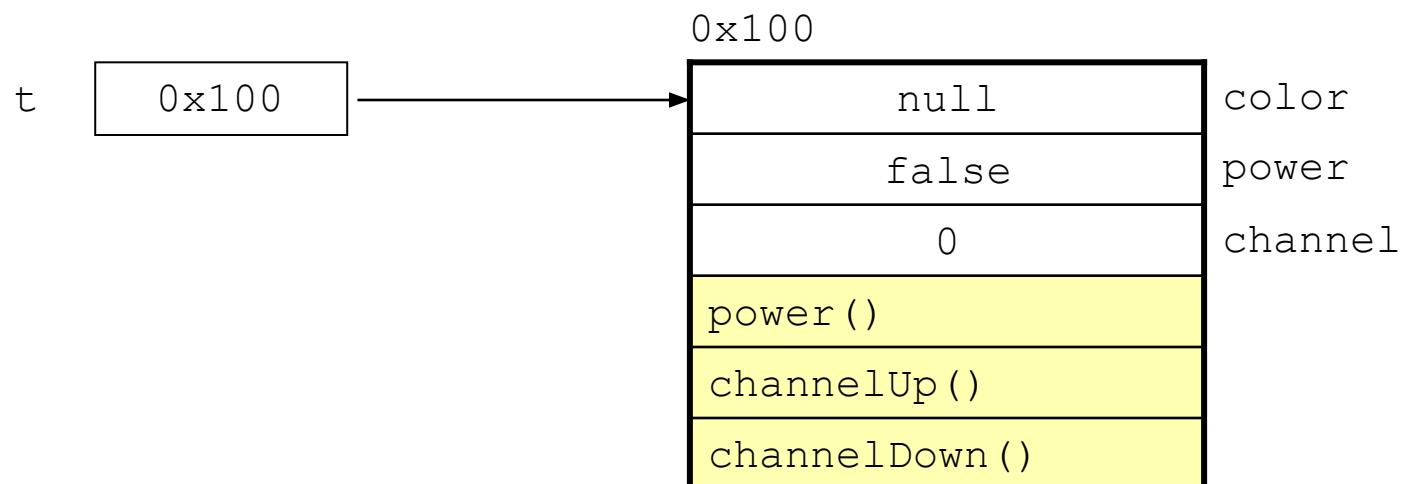
## 2.4 인스턴스의 생성과 사용(1/4)

### ▶ 인스턴스의 생성방법

```
클래스명 참조변수명;           // 객체를 다루기 위한 참조변수 선언  
참조변수명 = new 클래스명 (); // 객체생성 후, 생성된 객체의  
                           // 주소를 참조변수에 저장
```

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

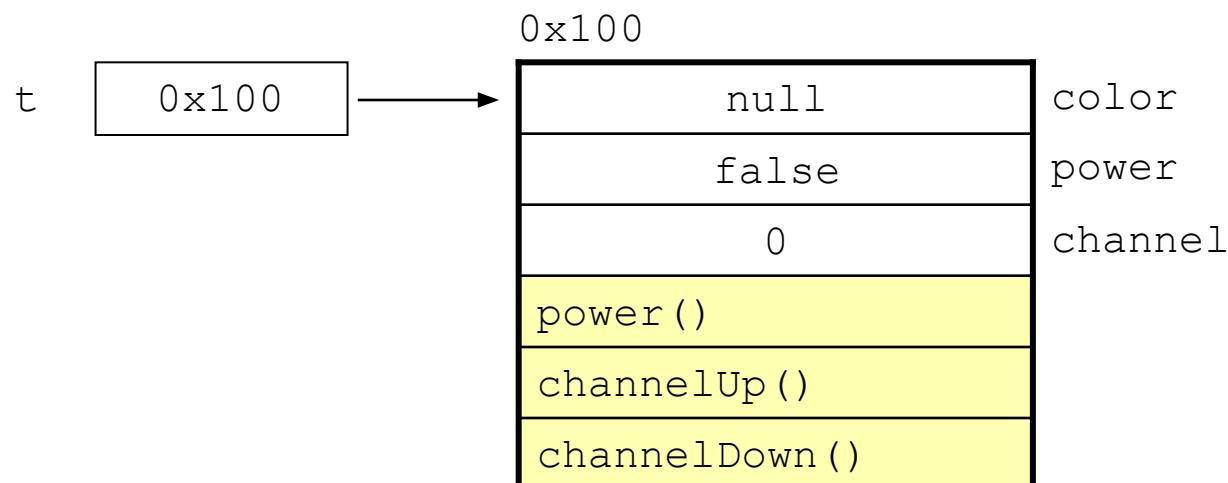
```
Tv t = new Tv();
```



## 2.4 인스턴스의 생성과 사용(2/4)

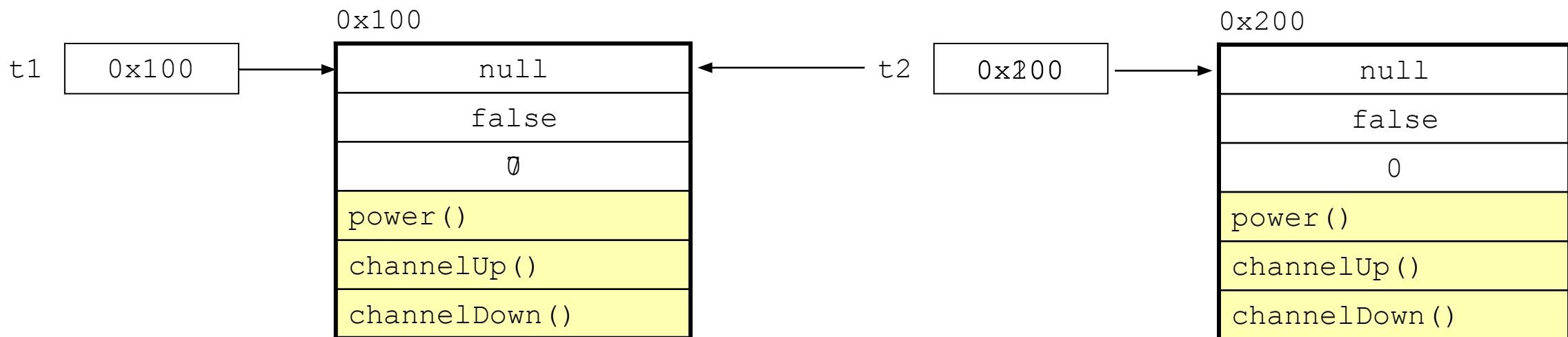
```
Tv t;  
t = new Tv();  
t.channel = 7;  
t.channelDown();  
  
System.out.println(t.channel);
```

```
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  
}
```

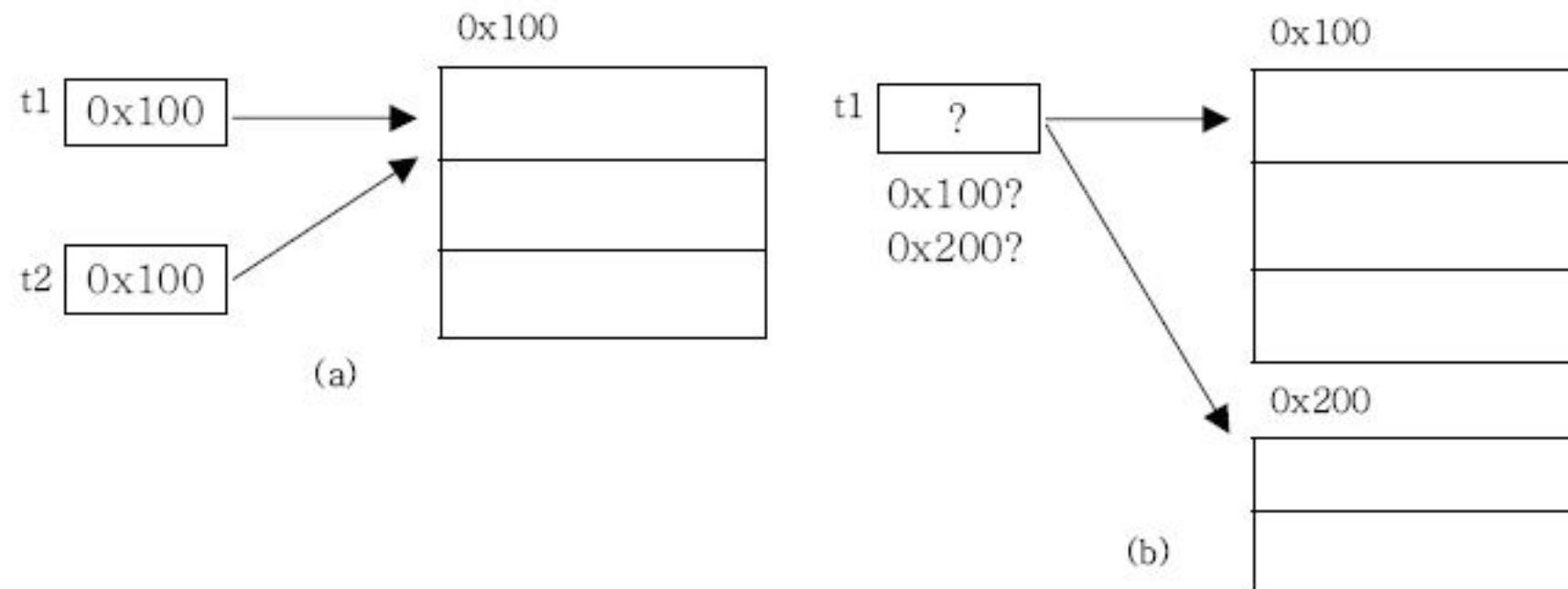


## 2.4 인스턴스의 생성과 사용(3/4)

```
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)

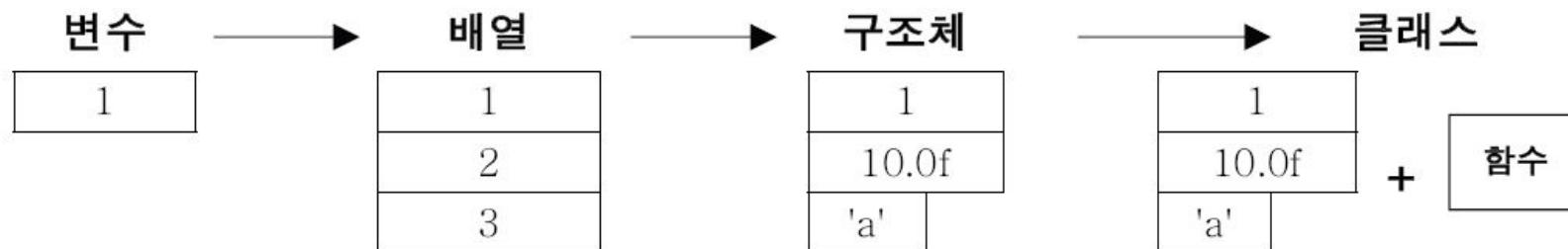


- (a) 하나의 인스턴스를 여러 개의 참조변수가 가리키는 경우(가능)
- (b) 여러 개의 인스턴스를 하나의 참조변수가 가리키는 경우(불가능)

【그림6-2】 참조변수와 인스턴스의 관계

## 2.5 클래스의 또 다른 정의

### 1. 클래스 – 데이터와 함수의 결합



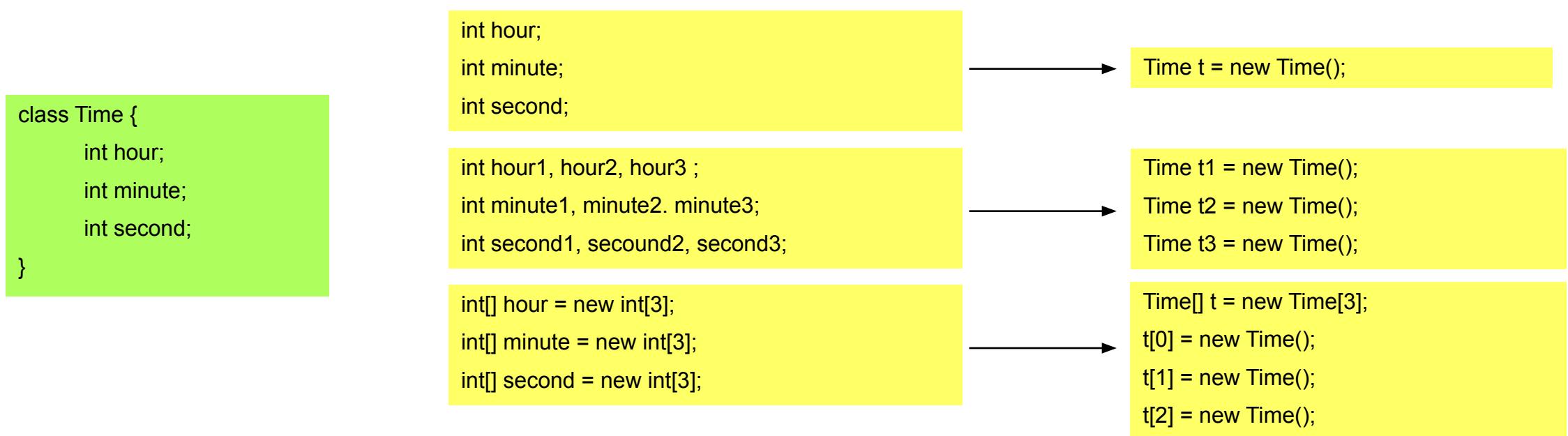
[그림6-3] 데이터 저장개념의 발전과정

- ▶ 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.



## 2.5 클래스의 또 다른 정의

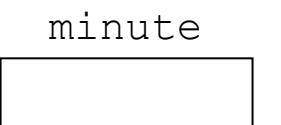
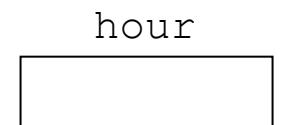
### 2. 클래스 – 사용자 정의 타입(User-defined type)

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



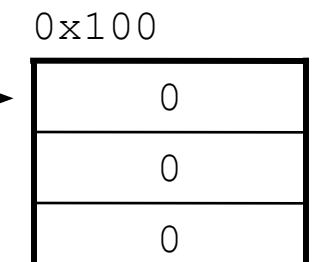
```
Time t = new Time();
```

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



t

0x100



hour  
minute  
second

## 2.5 클래스의 또 다른 정의

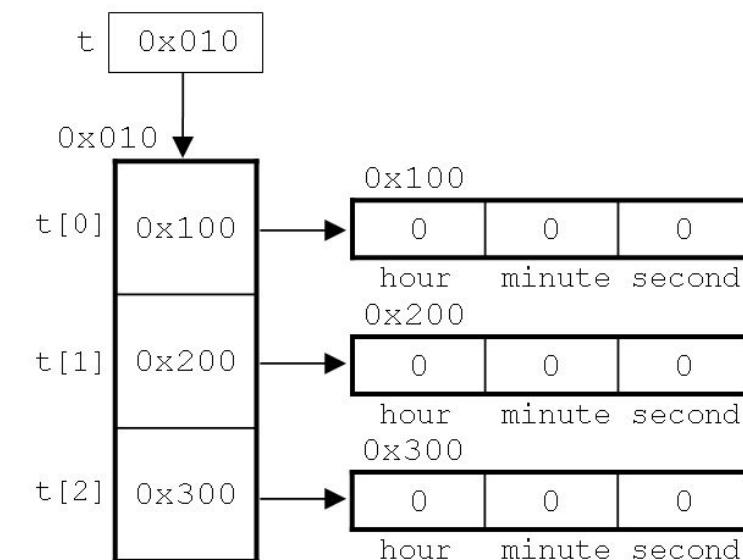
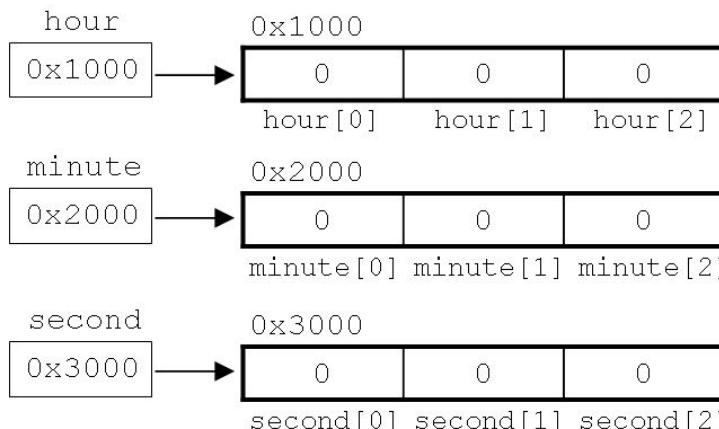
### 2. 클래스 – 사용자 정의 타입(User-defined type)

```
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;
}
```





## 액체지향개념 I-2

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

### 3. 변수와 메서드

- 3.1 선언위치에 따른 변수의 종류
- 3.2 클래스변수와 인스턴스변수
- 3.3 메서드
- 3.4 return문
- 3.5 메서드 호출
- 3.6 JVM의 메모리구조
- 3.7 기본형 매개변수와 참조형 매개변수
- 3.8 재귀호출
- 3.9 클래스 메서드와 인스턴스 메서드
- 3.10 멤버간의 참조와 호출

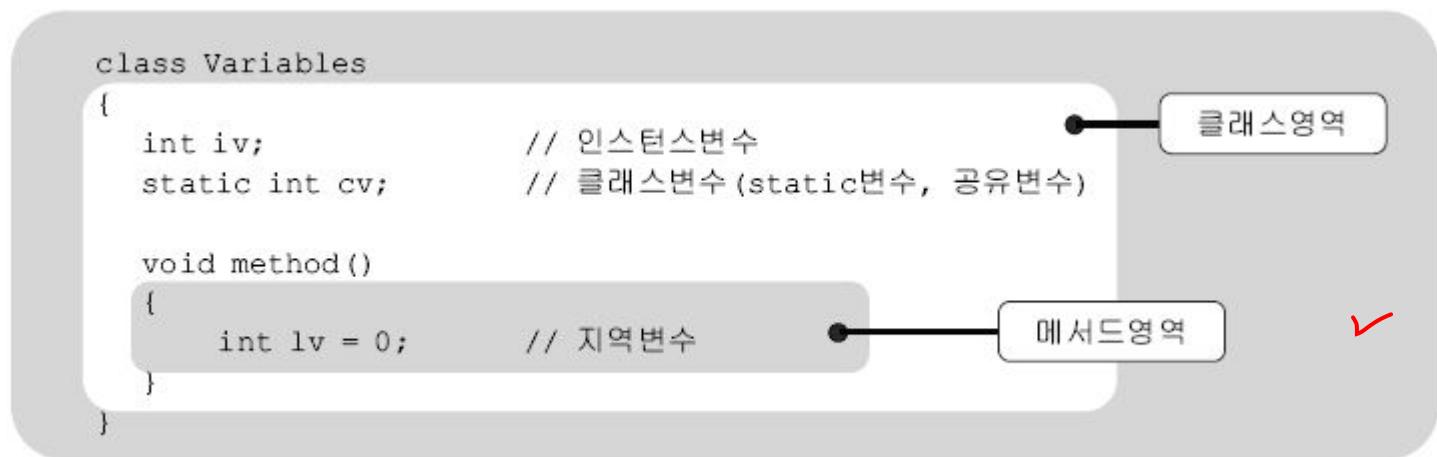
### 4. 메서드 오버로딩(method overloading)

- 4.1 메서드 오버로딩이란?
- 4.2 오버로딩의 조건
- 4.3 오버로딩의 예

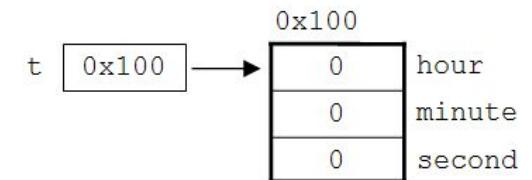
### 3. 변수와 메서드

### 3.1 선언위치에 따른 변수의 종류

“변수의 선언위치가 변수의 종류와 범위(scope)을 결정한다.”



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



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

### 3.1 선언위치에 따른 변수의 종류

#### ▶ 인스턴스변수(instance variable)

- 각 인스턴스의 개별적인 저장공간. 인스턴스마다 다른 값 저장 가능
- 인스턴스 생성 후, ‘참조변수.인스턴스변수명’으로 접근
- 인스턴스를 생성할 때 생성되고, 참조변수가 없을 때 가비지컬렉터에 의해 자동제거됨

#### ▶ 클래스변수(class variable)

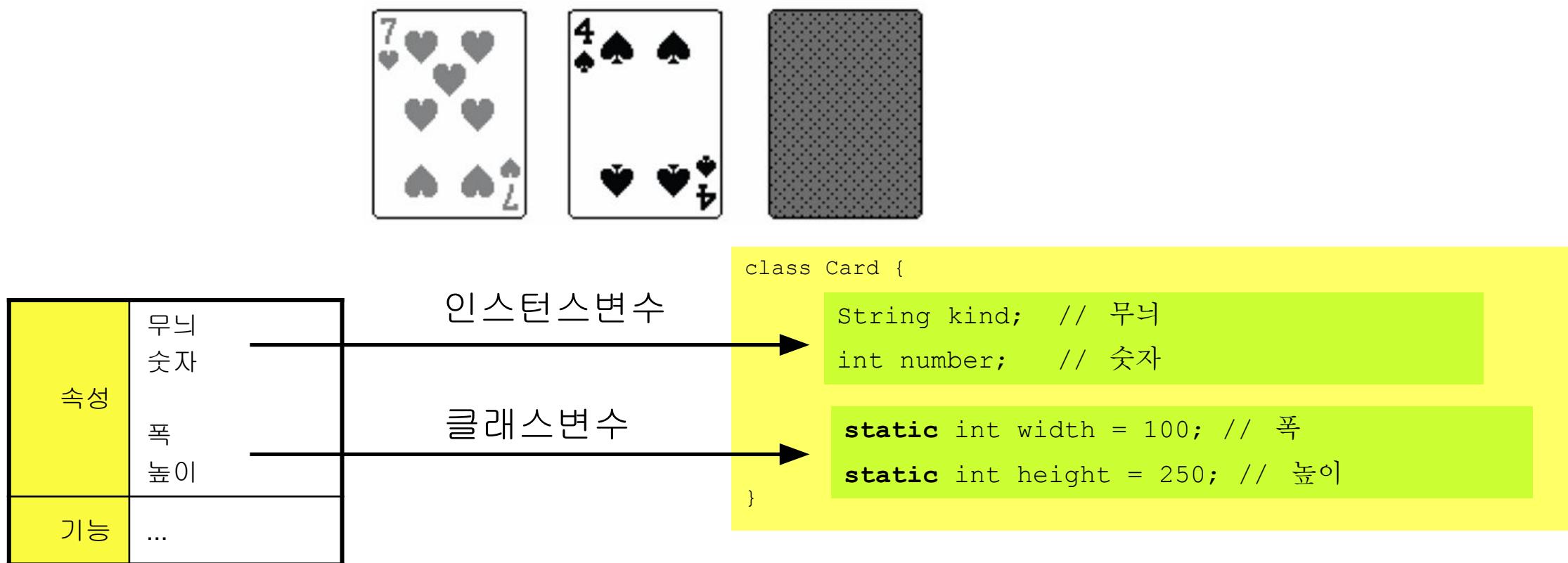
- 같은 클래스의 모든 인스턴스들이 공유하는 변수
- 인스턴스 생성없이 ‘클래스이름.클래스변수명’으로 접근
- 클래스가 로딩될 때 생성되고 프로그램이 종료될 때 소멸

#### ▶ 지역변수(local variable)

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

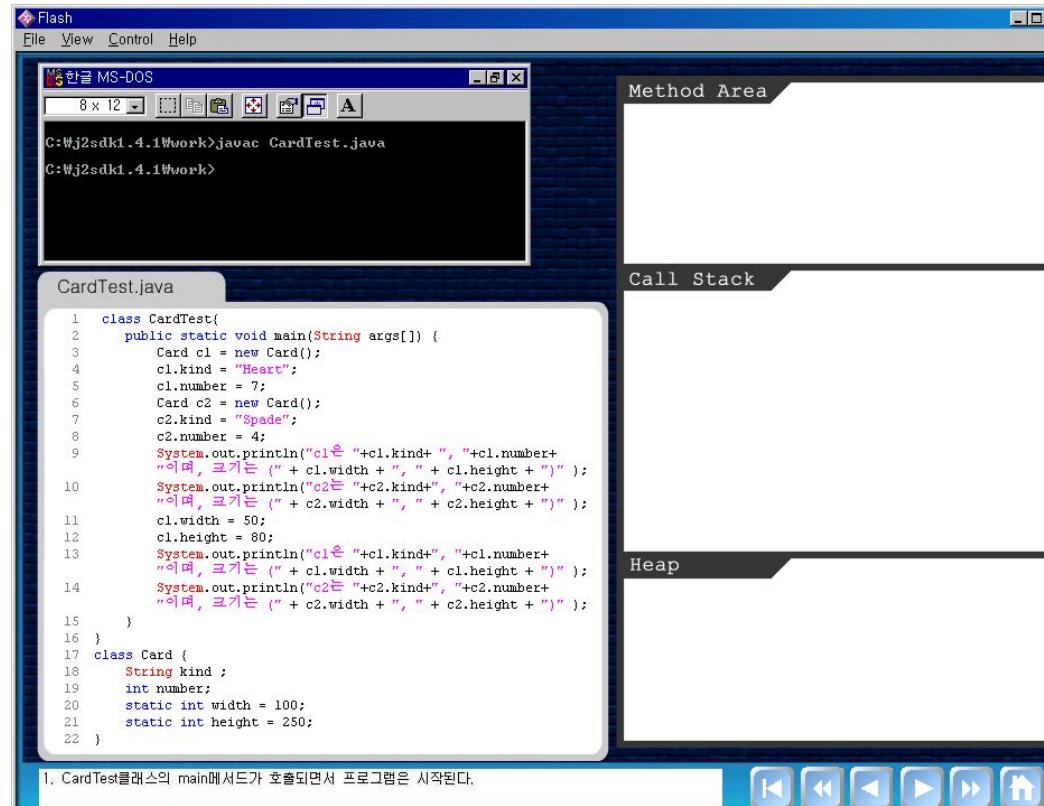
## 3.2 클래스변수와 인스턴스변수

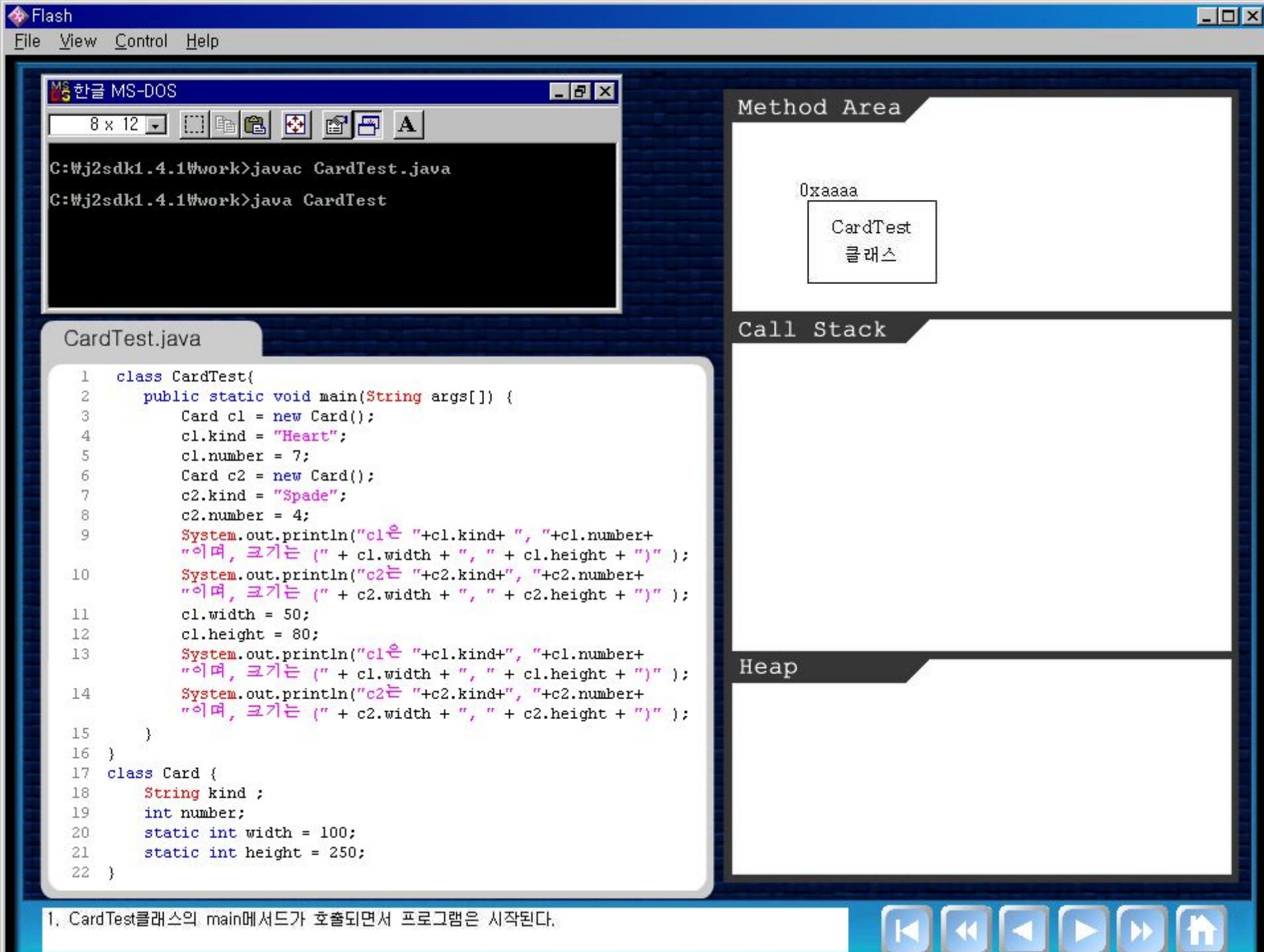
“인스턴스변수는 인스턴스가 생성될 때마다 생성되므로 인스턴스마다 각기 다른 값을 유지할 수 있지만, 클래스변수는 모든 인스턴스가 하나의 저장공간을 공유하므로 항상 공통된 값을 갖는다.”



## 3.2 클래스변수와 인스턴스변수

\* 플래시 동영상 : MemberVar.exe 또는 MemberVar.swf  
(java\_jungsuk\_src.zip의 flash폴더에 위치)





MS 한글 MS-DOS

8 x 12 A

```
C:\Wj2sdk1.4.1\work>javac CardTest.java
C:\Wj2sdk1.4.1\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+
10             "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
11          System.out.println("c2는 "+c2.kind+", "+c2.number+
12             "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
13          c1.width = 50;
14          c1.height = 80;
15          System.out.println("c1은 "+c1.kind+", "+c1.number+
16             "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
17          class Card {
18              String kind ;
19              int number;
20              static int width = 100;
21              static int height = 250;
22      }

```

Method Area

Call Stack

Heap

2. Card클래스의 인스턴스를 생성하기 위해 먼저 Card클래스가 메모리에 로드된다.

이 때, Card클래스의 클래스변수인 width와 height가 메모리에 생성되고 각각 100, 250으로 초기화 된다.

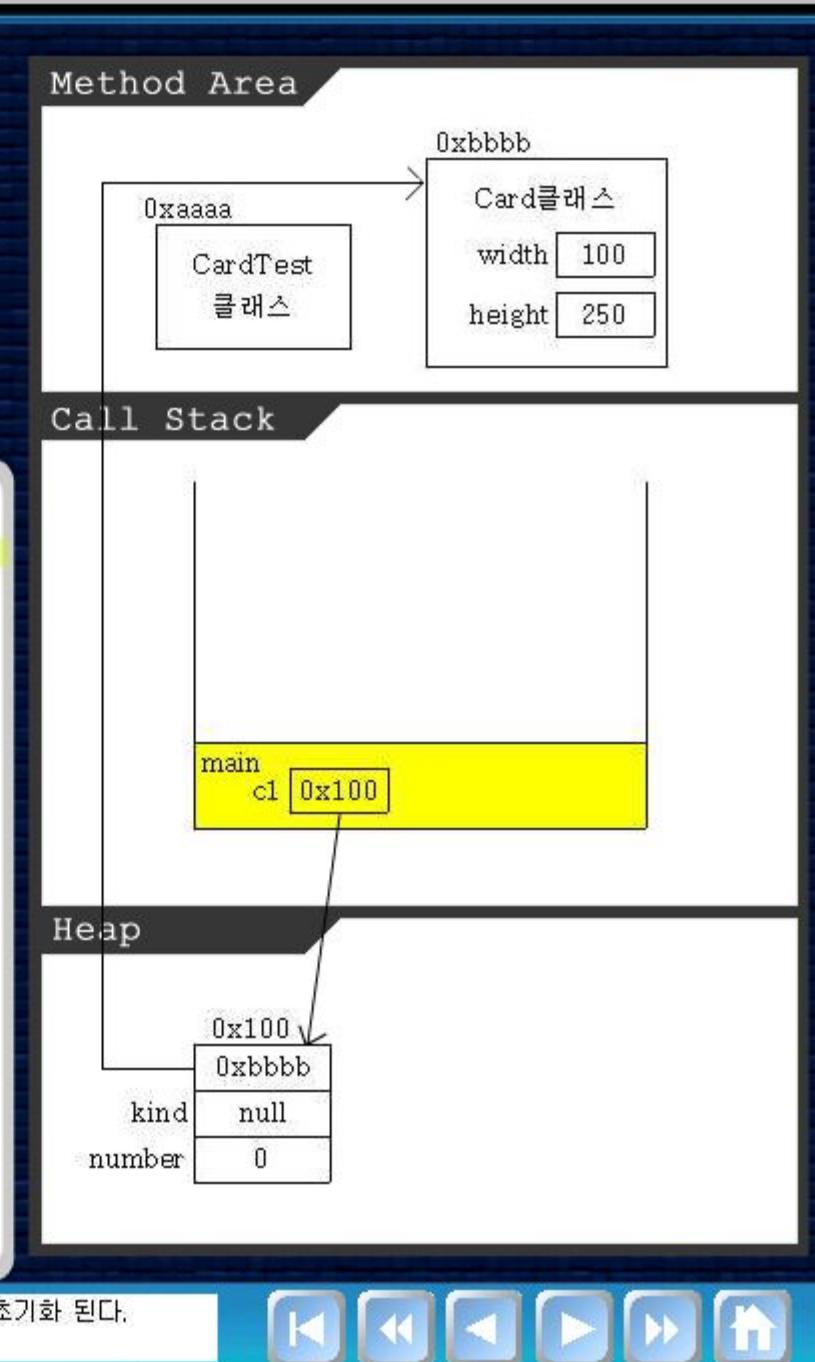
MS 한글 MS-DOS

```
8 x 12 A
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 }
```



3. Card인스턴스가 생성되고, 멤버변수인 kind와 number가 기본값인 null과 0으로 각각 초기화 된다.

그리고 생성된 인스턴스의 주소가 참조변수 c1에 저장된다.

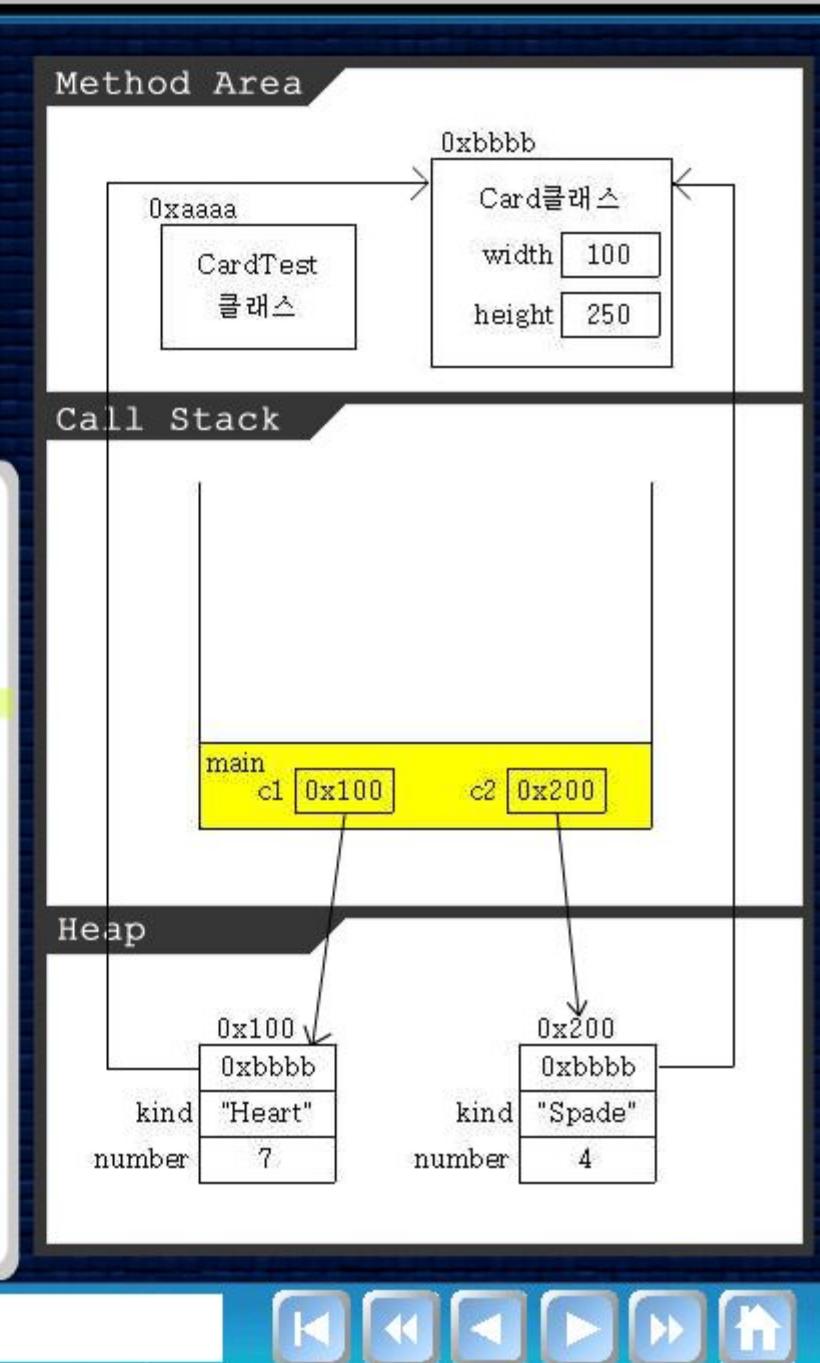
MS 한글 MS-DOS

```
8 x 12 A
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.

MS 한글 MS-DOS

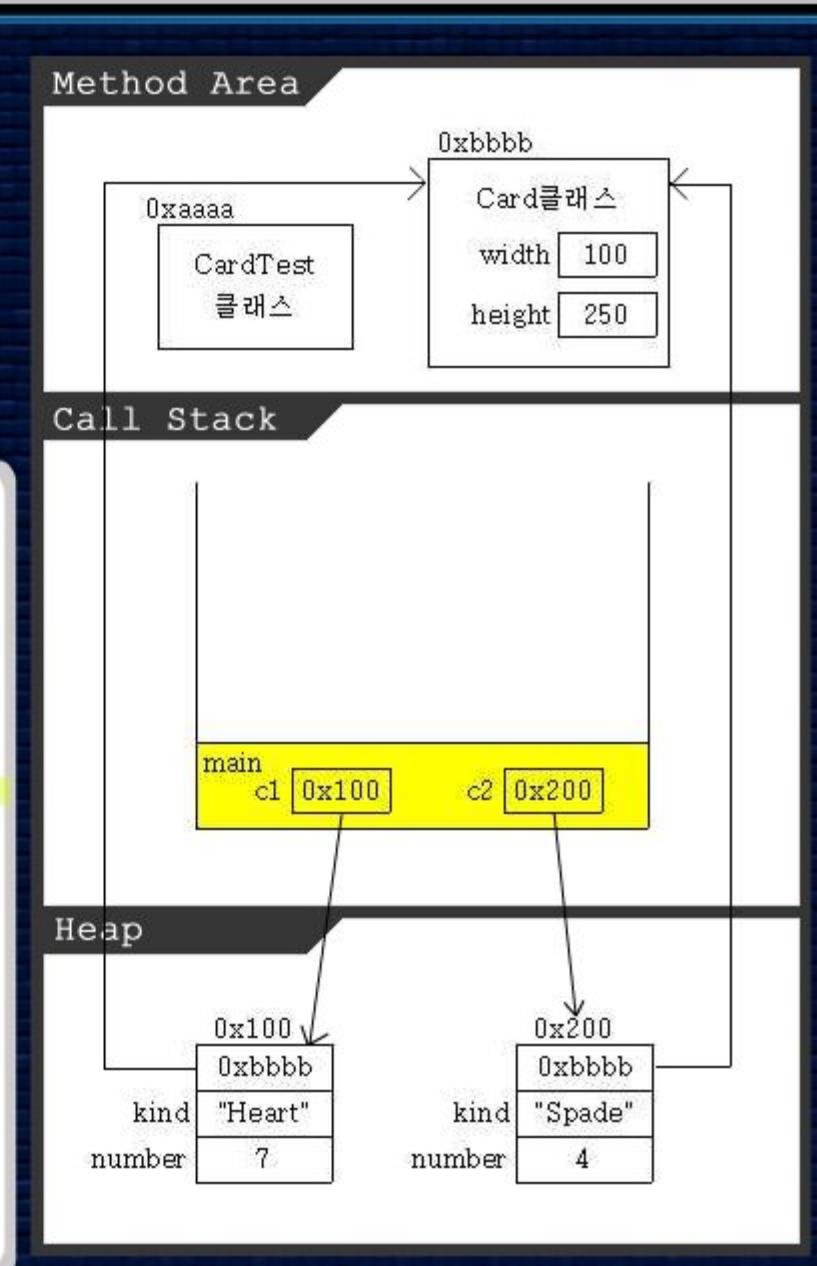
```
8 x 12 A
```

```
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+
10         "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
11         System.out.println("c2는 "+c2.kind+", "+c2.number+
12         "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
13         c1.width = 50;
14         c1.height = 80;
15         System.out.println("c1은 "+c1.kind+", "+c1.number+
16         "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
17         System.out.println("c2는 "+c2.kind+", "+c2.number+
18         "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
19     }
20 }
21 class Card {
22     String kind ;
23     int number;
24     static int width = 100;
25     static int height = 250;
26 }
```



7. 인스턴스 `c1`과 `c2`의 값을 화면에 출력한다. 모든 인스턴스는 자신을 생성한 클래스의 주소를 갖고 있으므로, 참조변수를 사용해서도 클래스변수에 접근할 수 있다.

MS 한글 MS-DOS

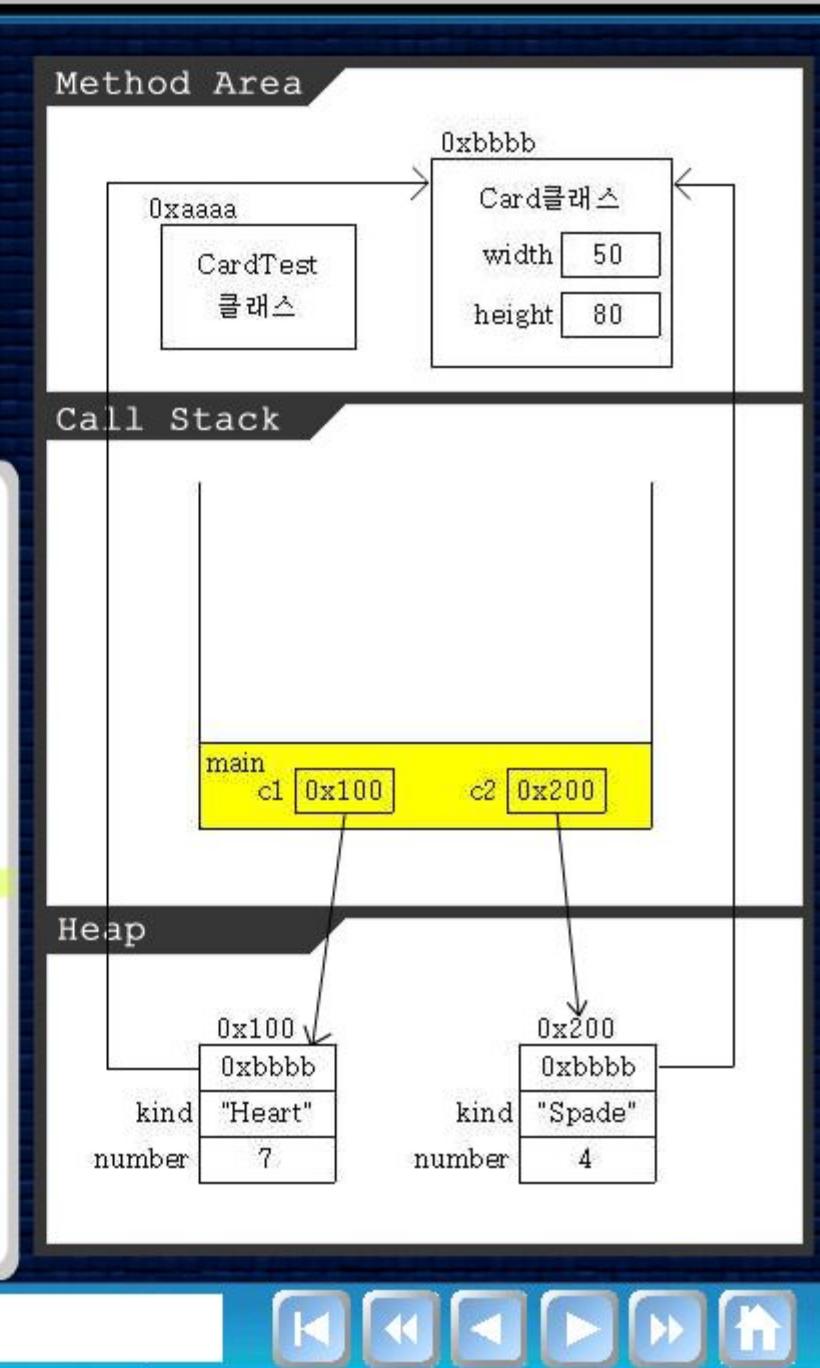
```
8 x 12 A
```

```
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+
10             "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
11          System.out.println("c2는 "+c2.kind+", "+c2.number+
12             "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
13          c1.width = 50;
14          c1.height = 80;
15          System.out.println("c1은 "+c1.kind+", "+c1.number+
16             "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
17          System.out.println("c2는 "+c2.kind+", "+c2.number+
18             "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
19      }
20  }
```



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

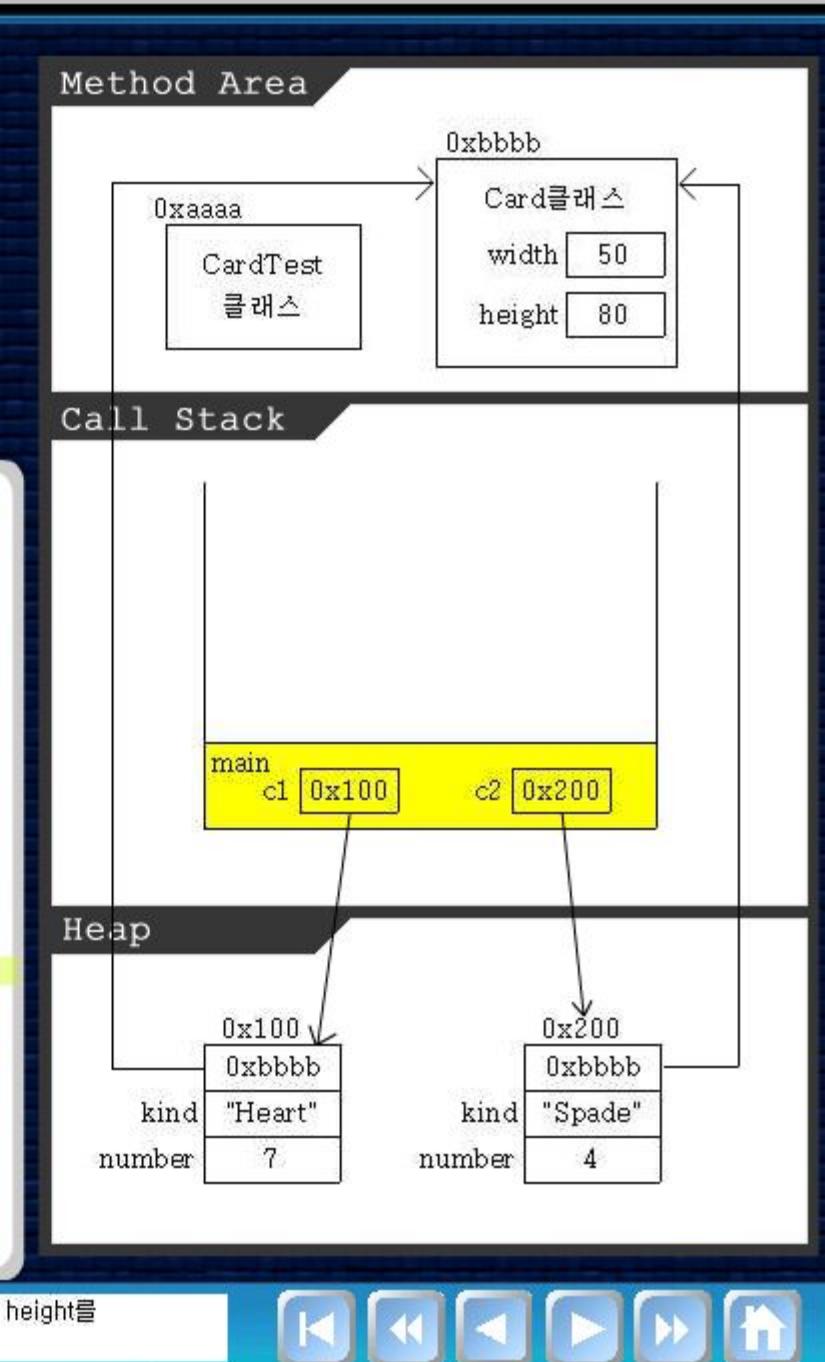
MS 한글 MS-DOS

```
8 x 12 A
C:\w2sdk1.4.1\work>javac CardTest.java
C:\w2sdk1.4.1\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+
10             "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
11          System.out.println("c2는 "+c2.kind+", "+c2.number+
12             "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
13          c1.width = 50;
14          c1.height = 80;
15          System.out.println("c1은 "+c1.kind+", "+c1.number+
16             "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
17          System.out.println("c2는 "+c2.kind+", "+c2.number+
18             "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
19      }
20  }
21  class Card {
22      String kind ;
23      int number;
24      static int width = 100;
25      static int height = 250;
26  }
```



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

### 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.)

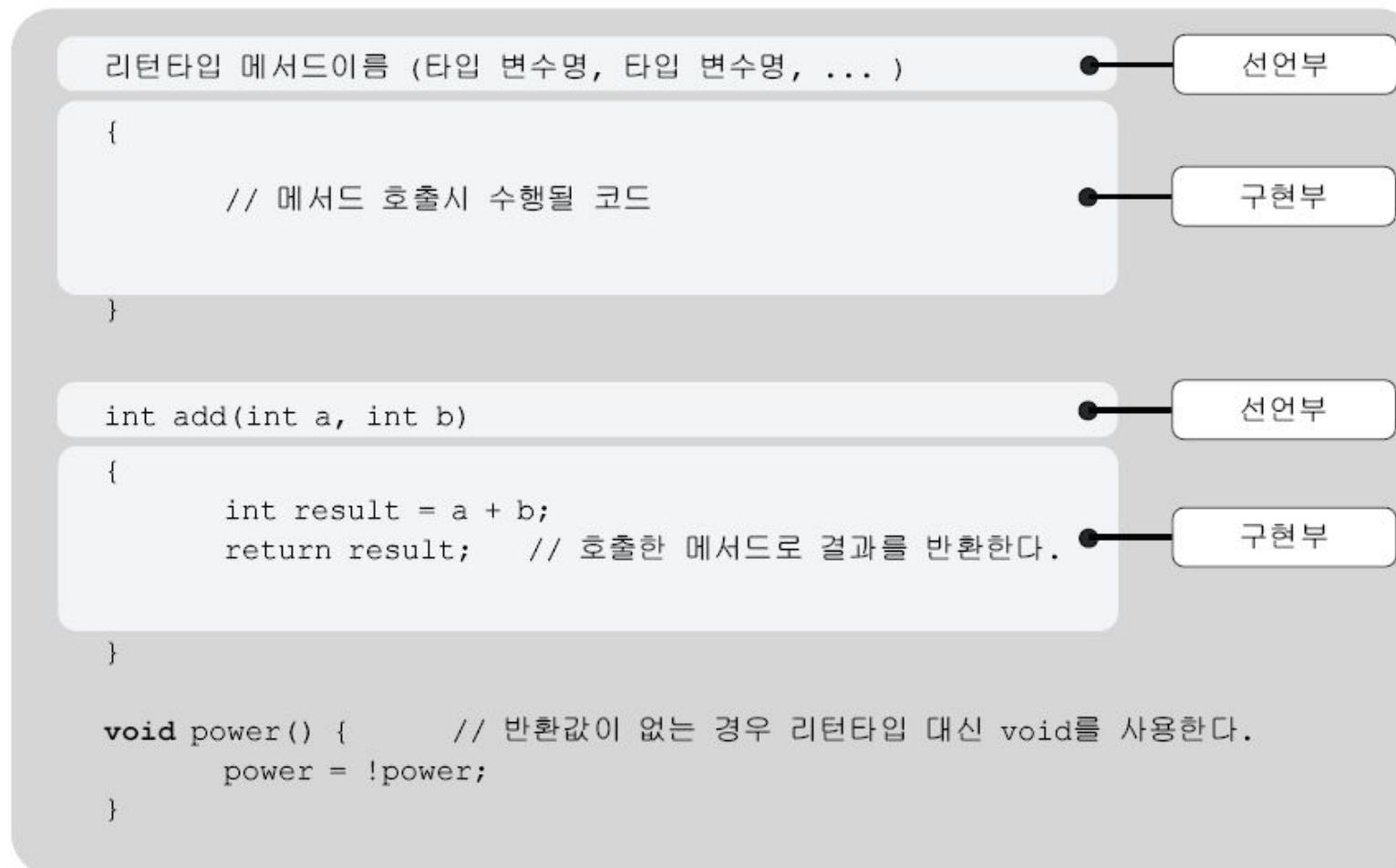
#### ▶ 메서드의 장점과 작성지침

- 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.

```
public static void main(String args[]) {  
    while(true) {  
        switch(displayMenu()) { // 화면에 메뉴를 출력한다.  
            case 1 :  
                inputRecord(); // 데이터를 입력 받는다.  
                break;  
            case 2 :  
                deleteRecord(); // 데이터를 삭제한다.  
                break;  
            case 3 :  
                sortRecord(); // 데이터를 정렬한다.  
                break;  
            case 4 :  
                System.out.println("프로그램을 종료합니다. `");  
                System.exit(0);  
        }  
    } // while(true)  
} // main메서드의 끝
```

### 3.3 메서드(method)

- ▶ 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 **return**문을 만났을 때

### ▶ **return**문

- 현재 실행 중인 메서드를 종료하고 호출한 메서드로 되돌아간다.

1. 반환값이 없는 경우 - **return**문만 써주면 된다.

```
return;
```

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

```
return 반환값;
```

```
int add(int a, int b)
{
    int result = a + b;
    return result;
}
```

타입이 일치해야한다.

## 3.4 return문 - 주의사항

- ▶ 반환값이 있는 메서드는 모든 경우에 return문이 있어야 한다.

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

```
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;  
}
```

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

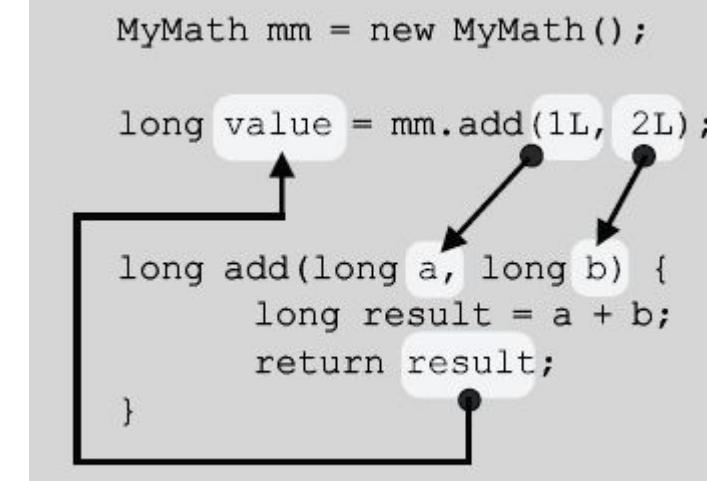
## 3.5 메서드의 호출

### ▶ 메서드의 호출방법

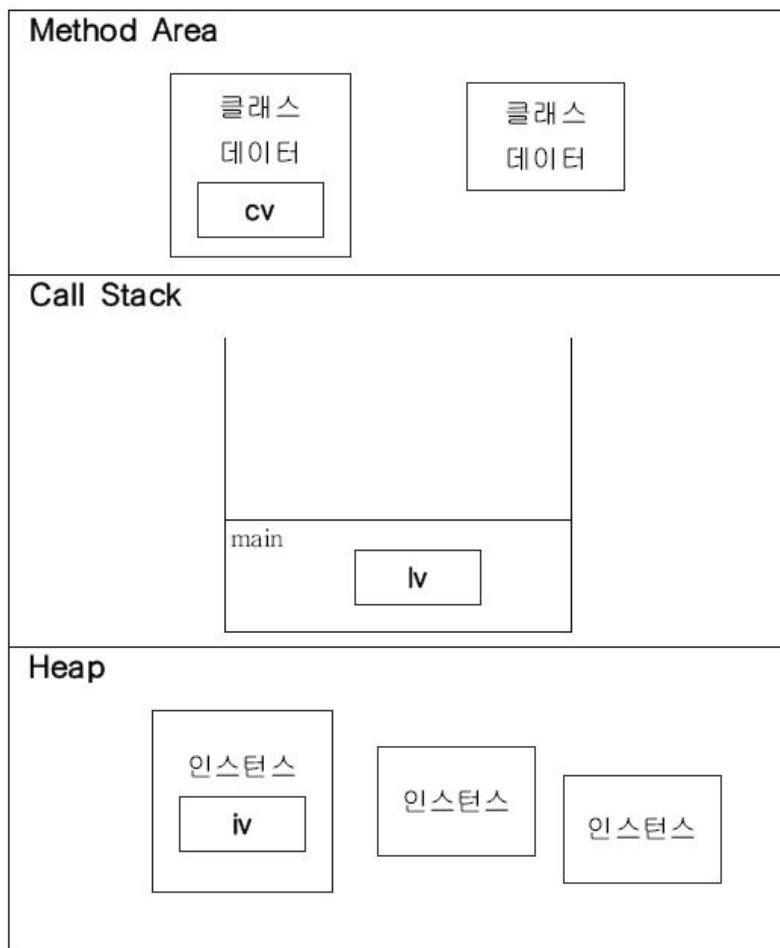
```
참조변수.메서드 이름();           // 메서드에 선언된 매개변수가 없는 경우
```

```
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;  
    }  
    ...  
}
```



## 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.

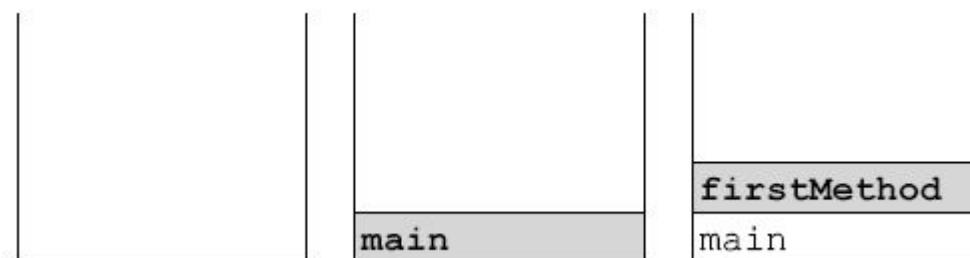
### ▶ 힙(Heap)

- 인스턴스가 생성되는 공간. `new`연산자에 의해서 생성되는 배열과 객체는 모두 여기에 생성된다.

## 3.6 JVM의 메모리 구조 - 호출스택

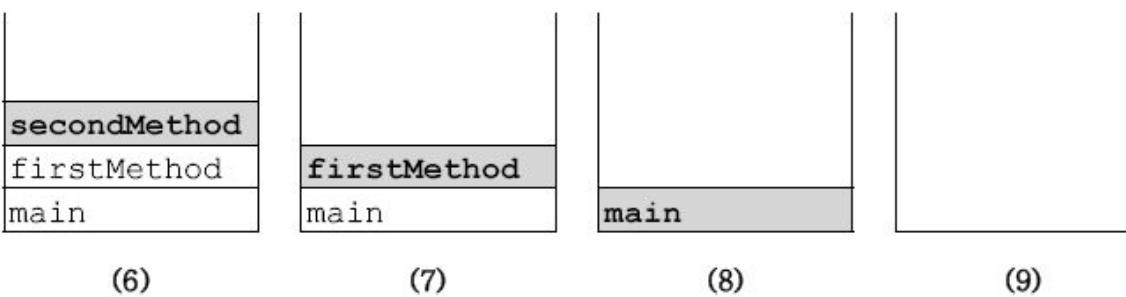
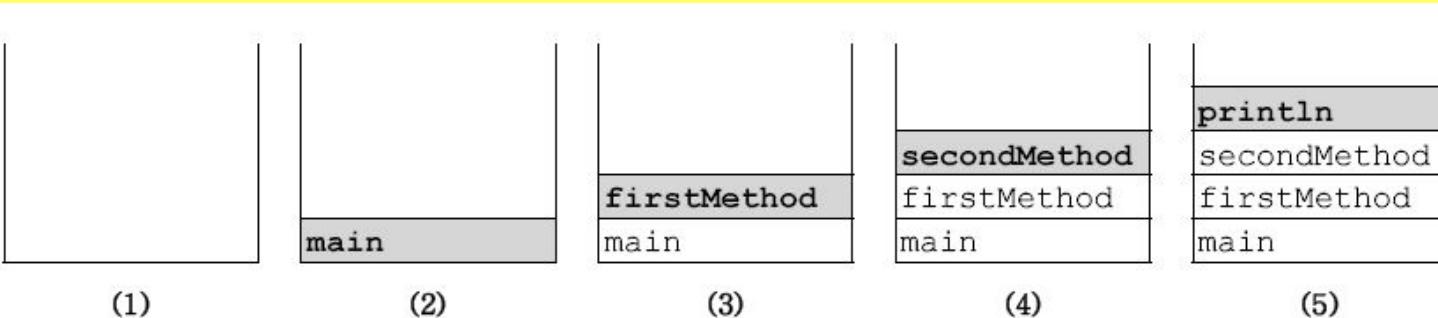
### ▶ 호출스택의 특징

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



## 3.6 JVM의 메모리 구조 - 호출스택

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

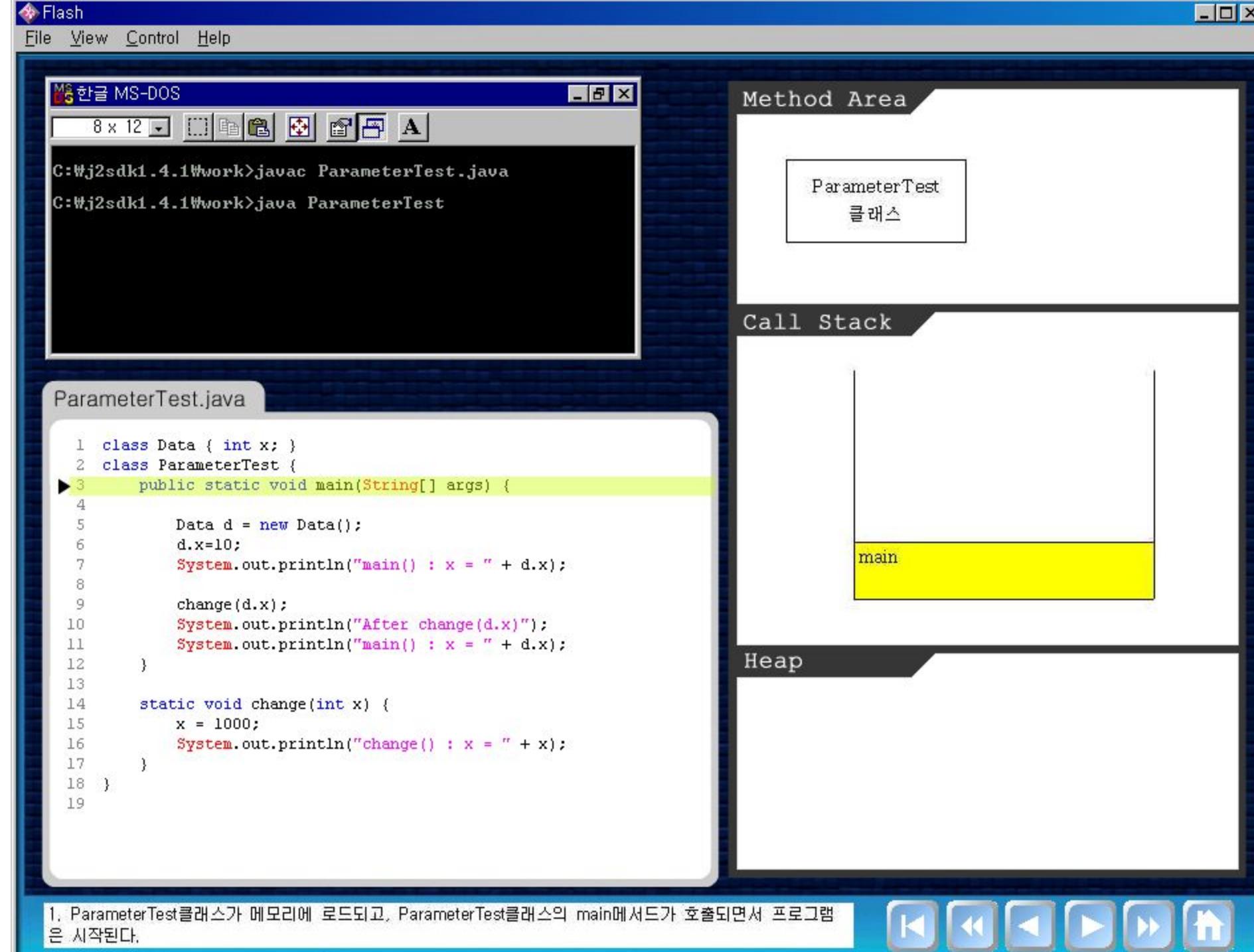


### 3.7 기본형 매개변수와 참조형 매개변수

- ▶ 기본형 매개변수 – 변수의 값을 읽기만 할 수 있다.(read only)
- ▶ 참조형 매개변수 – 변수의 값을 읽고 변경할 수 있다.(read & write)

\* 플래시 동영상(java\_jungsuk\_src.zip의 flash폴더에 위치)

- 기본형 매개변수 예제 : PrimitiveParam.exe
- 참조형 매개변수 예제 : ReferenceParam.exe



Flash  
File View Control Help

MS 한글 MS-DOS  
8 x 12 A

```
C:\j2sdk1.4.1\work>javac ParameterTest.java
C:\j2sdk1.4.1\work>java ParameterTest
```

ParameterTest.java

```

1 class Data { int x; }
2 class ParameterTest {
3     public static void main(String[] args) {
4
5         Data d = new Data();
6         d.x=10;
7         System.out.println("main() : x = " + d.x);
8
9         change(d.x);
10        System.out.println("After change(d.x)");
11        System.out.println("main() : x = " + d.x);
12    }
13
14    static void change(int x) {
15        x = 1000;
16        System.out.println("change() : x = " + x);
17    }
18 }
```

Method Area

ParameterTest 클래스  
Data 클래스

Call Stack

Heap

2. Data클래스가 메모리에 로드되고, Data타입의 참조변수 d가 main에서 d의 지역변수로 생성된다.  
Data클래스의 인스턴스가 생성되고, 생성된 인스턴스의 주소가 참조변수 d에 저장된다.

MS 한글 MS-DOS

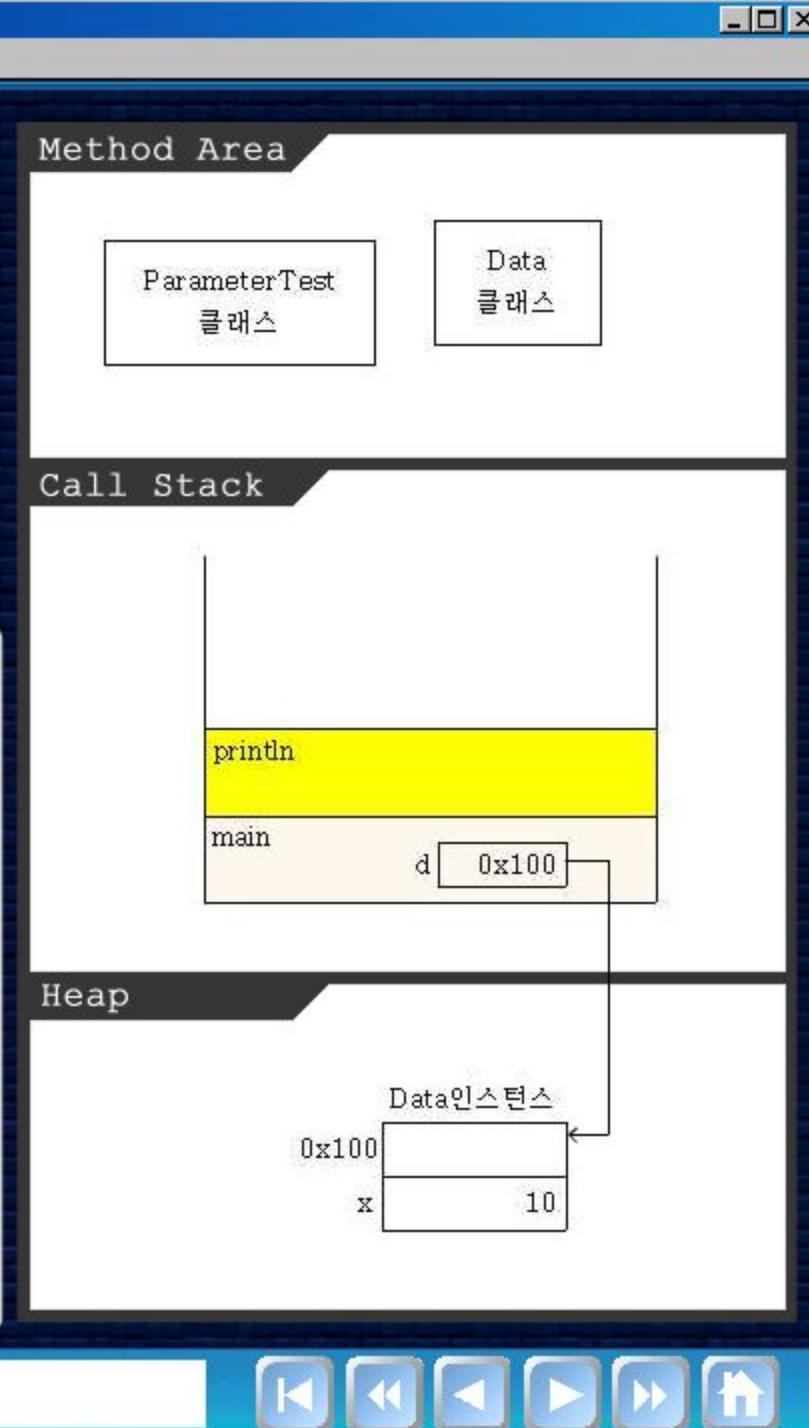
8 x 12

```
C:\Wj2sdk1.4.1\work>javac ParameterTest.java
C:\Wj2sdk1.4.1\work>java ParameterTest
main() : x = 10
```

## ParameterTest.java

```

1 class Data { int x; }
2 class ParameterTest {
3     public static void main(String[] args) {
4
5         Data d = new Data();
6         d.x=10;
7         System.out.println("main() : x = " + d.x);
8
9         change(d.x);
10        System.out.println("After change(d.x)");
11        System.out.println("main() : x = " + d.x);
12    }
13
14    static void change(int x) {
15        x = 1000;
16        System.out.println("change() : x = " + x);
17    }
18 }
```



MS 한글 MS-DOS

8 x 12

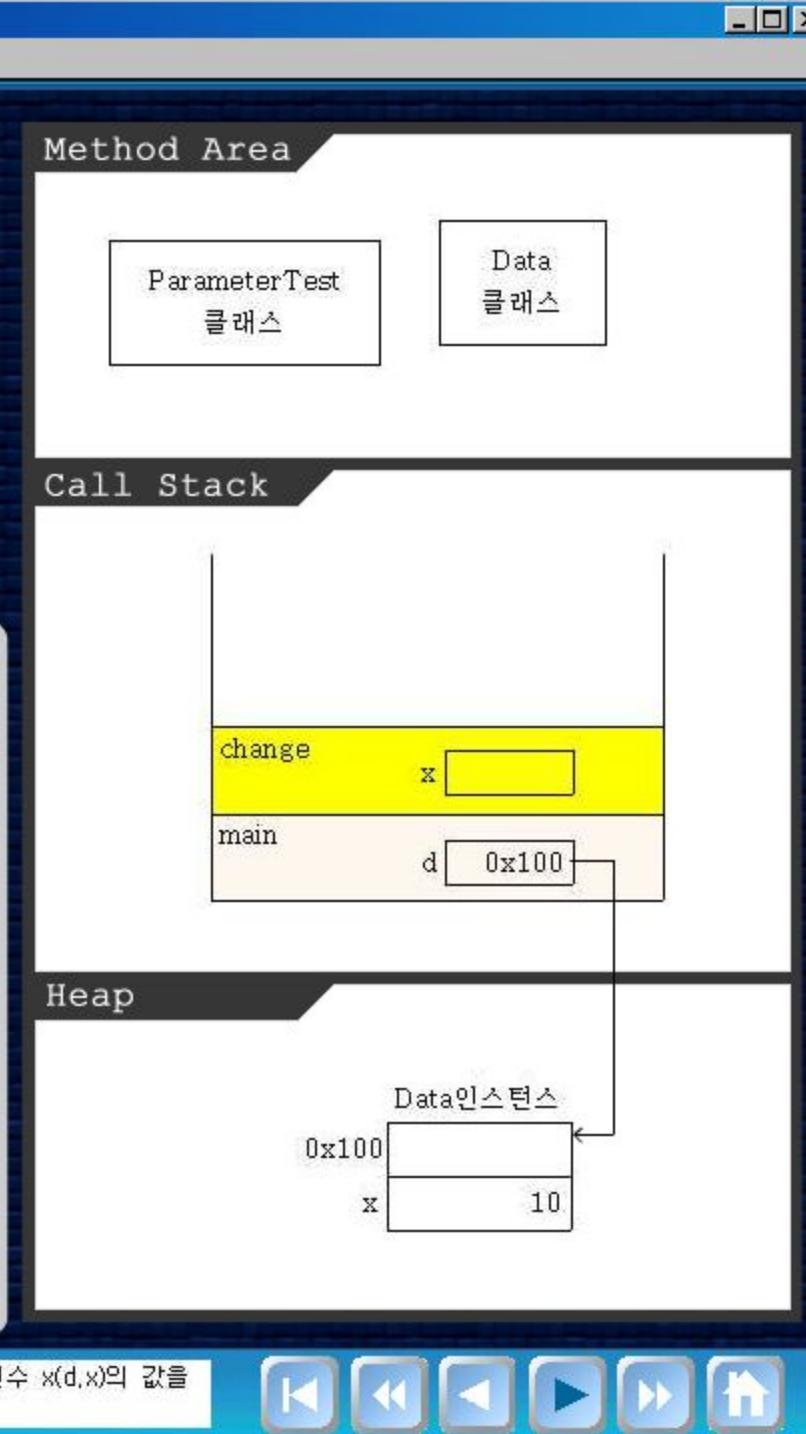
```
C:\Wj2sdk1.4.1\work>javac ParameterTest.java
C:\Wj2sdk1.4.1\work>java ParameterTest
main() : x = 10
```

## ParameterTest.java

```

1 class Data { int x; }
2 class ParameterTest {
3     public static void main(String[] args) {
4
5         Data d = new Data();
6         d.x=10;
7         System.out.println("main() : x = " + d.x);
8
9         change(d.x);
10        System.out.println("After change(d.x)");
11        System.out.println("main() : x = " + d.x);
12    }
13
14    static void change(int x) {
15        x = 1000;
16        System.out.println("change() : x = " + x);
17    }
18 }
19

```



5. change메서드를 호출하면서 매개변수로 참조변수 d가 가리키고 있는 인스턴스의 멤버변수 x(d.x)의 값을 넘겨준다. d.x의 값인 10이 change의 매개변수 x에 복사된다.



MS 한글 MS-DOS

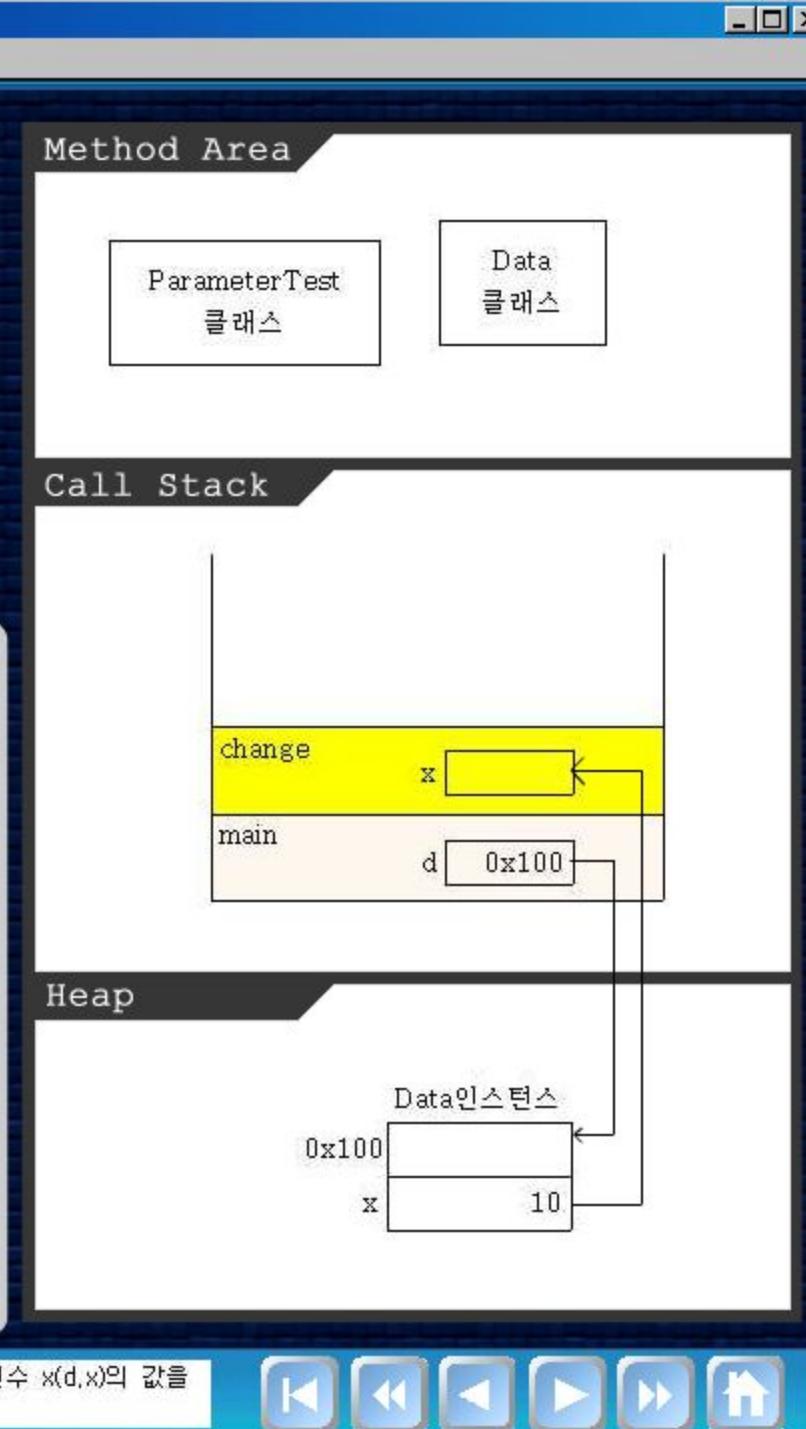
8 x 12 A

```
C:\j2sdk1.4.1\work>javac ParameterTest.java
C:\j2sdk1.4.1\work>java ParameterTest
main() : x = 10
```

## ParameterTest.java

```

1 class Data { int x; }
2 class ParameterTest {
3     public static void main(String[] args) {
4
5         Data d = new Data();
6         d.x=10;
7         System.out.println("main() : x = " + d.x);
8
9         change(d.x);
10        System.out.println("After change(d.x)");
11        System.out.println("main() : x = " + d.x);
12    }
13
14    static void change(int x) {
15        x = 1000;
16        System.out.println("change() : x = " + x);
17    }
18 }
```



5. change메서드를 호출하면서 매개변수로 참조변수 d가 가리키고 있는 인스턴스의 멤버변수 x(d.x)의 값을 넘겨준다. d.x의 값인 10이 change의 매개변수 x에 복사된다.



Flash  
File View Control Help

MS 한글 MS-DOS  
8 x 12 A

```
C:\j2sdk1.4.1\work>javac ParameterTest.java
C:\j2sdk1.4.1\work>java ParameterTest
main() : x = 10
```

ParameterTest.java

```

1 class Data { int x; }
2 class ParameterTest {
3     public static void main(String[] args) {
4
5         Data d = new Data();
6         d.x=10;
7         System.out.println("main() : x = " + d.x);
8
9         change(d.x);
10        System.out.println("After change(d.x)");
11        System.out.println("main() : x = " + d.x);
12    }
13
14    static void change(int x) {
15        x = 1000;
16        System.out.println("change() : x = " + x);
17    }
18 }
19

```

Method Area

- ParameterTest 클래스
- Data 클래스

Call Stack

```

graph TD
    main[main] --> change[change]
    subgraph call_stack [Call Stack]
        main
        change
    end

```

Heap

```

graph TD
    subgraph heap [Heap]
        direction TB
        subgraph Data_instant [Data인스턴스]
            direction LR
            subgraph d_scope [d]
                d_x[0x100]
                d_x_val[x 10]
            end
            d_scope --- Data_instant
        end
        subgraph change_scope [change]
            direction LR
            subgraph x_scope [x]
                x_val[x 1000]
            end
            x_scope --- change_scope
        end
        Data_instant --- change_scope
    end

```

6. change에서 d의 지역변수 x에 1000을 저장한다.  
x의 값이 10에서 1000으로 변경된다.

MS 한글 MS-DOS

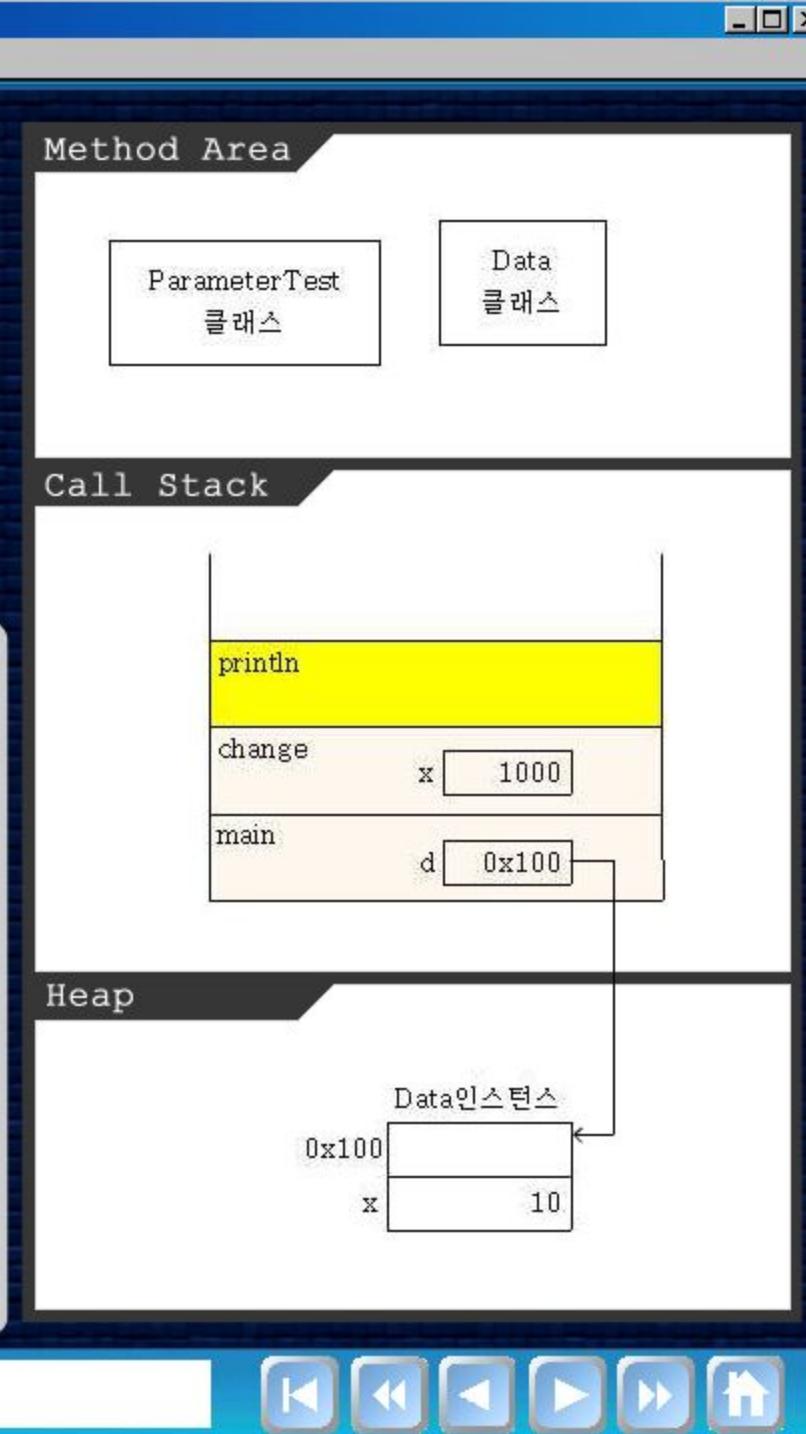
8 x 12

```
C:\Wj2sdk1.4.1\work>javac ParameterTest.java
C:\Wj2sdk1.4.1\work>java ParameterTest
main() : x = 10
change() : x = 1000
```

## ParameterTest.java

```

1 class Data { int x; }
2 class ParameterTest {
3     public static void main(String[] args) {
4
5         Data d = new Data();
6         d.x=10;
7         System.out.println("main() : x = " + d.x);
8
9         change(d.x);
10        System.out.println("After change(d.x)");
11        System.out.println("main() : x = " + d.x);
12    }
13
14    static void change(int x) {
15        x = 1000;
16        System.out.println("change() : x = " + x);
17    }
18 }
```



MS 한글 MS-DOS

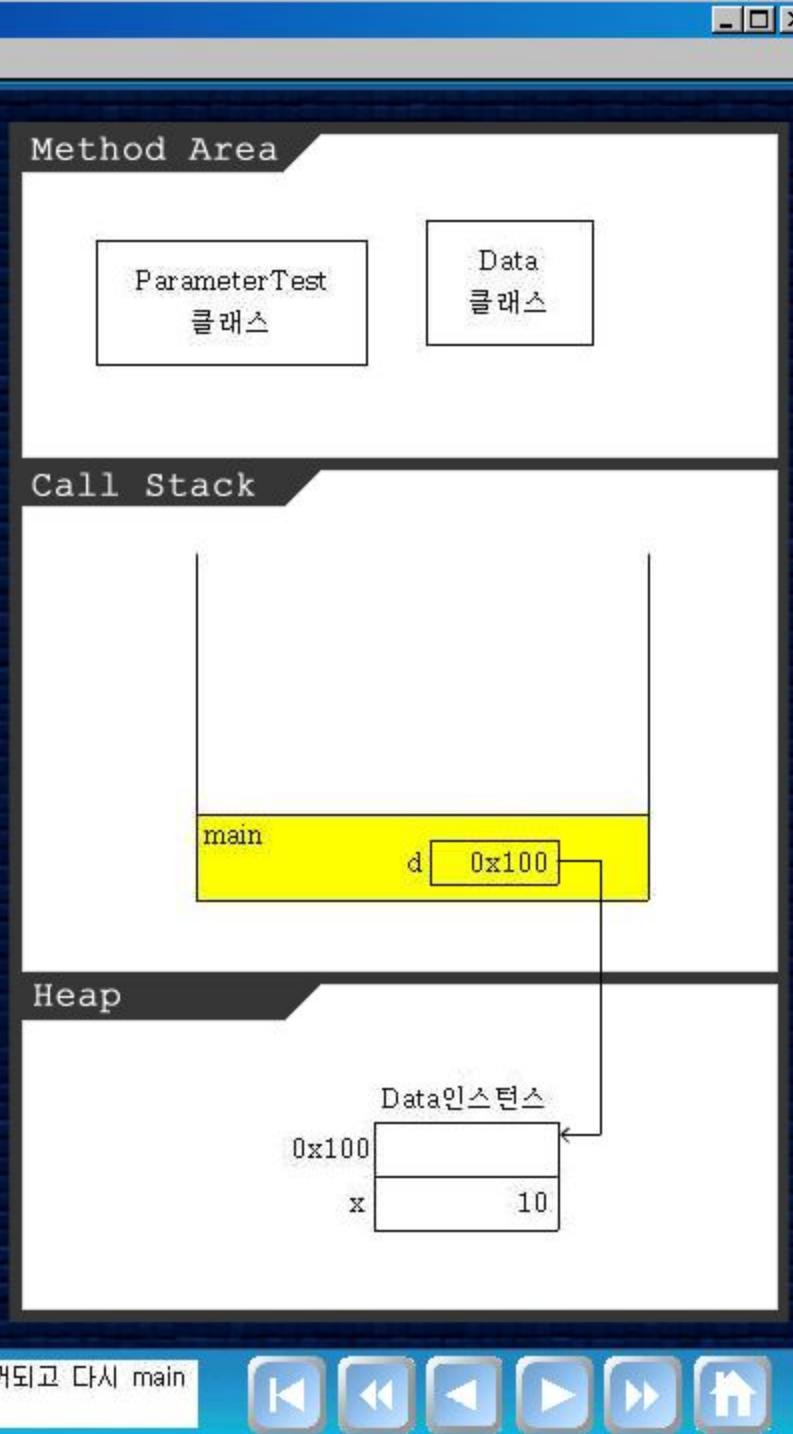
8 x 12

```
C:\Wj2sdk1.4.1\work>javac ParameterTest.java
C:\Wj2sdk1.4.1\work>java ParameterTest
main() : x = 10
change() : x = 1000
```

## ParameterTest.java

```

1  class Data { int x; }
2  class ParameterTest {
3      public static void main(String[] args) {
4
5          Data d = new Data();
6          d.x=10;
7          System.out.println("main() : x = " + d.x);
8
9          change(d.x);
10         System.out.println("After change(d.x)");
11         System.out.println("main() : x = " + d.x);
12     }
13
14     static void change(int x) {
15         x = 1000;
16         System.out.println("change() : x = " + x);
17     }
18 }
```



MS 한글 MS-DOS

8 x 12

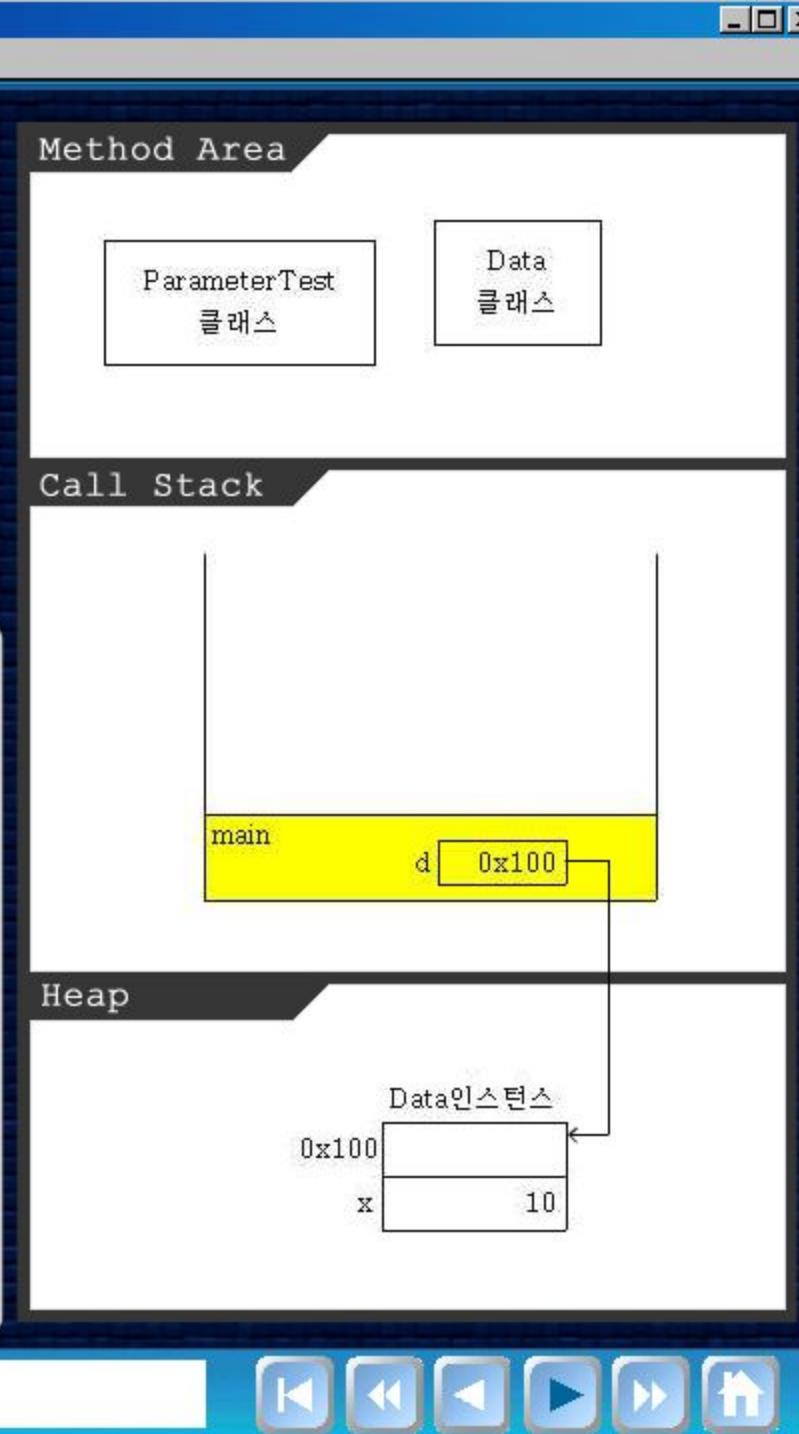
```
C:\Wj2sdk1.4.1\work>javac ParameterTest.java
C:\Wj2sdk1.4.1\work>java ParameterTest
main() : x = 10
change() : x = 1000
After change(d.x)
main() : x = 10
```

## ParameterTest.java

```

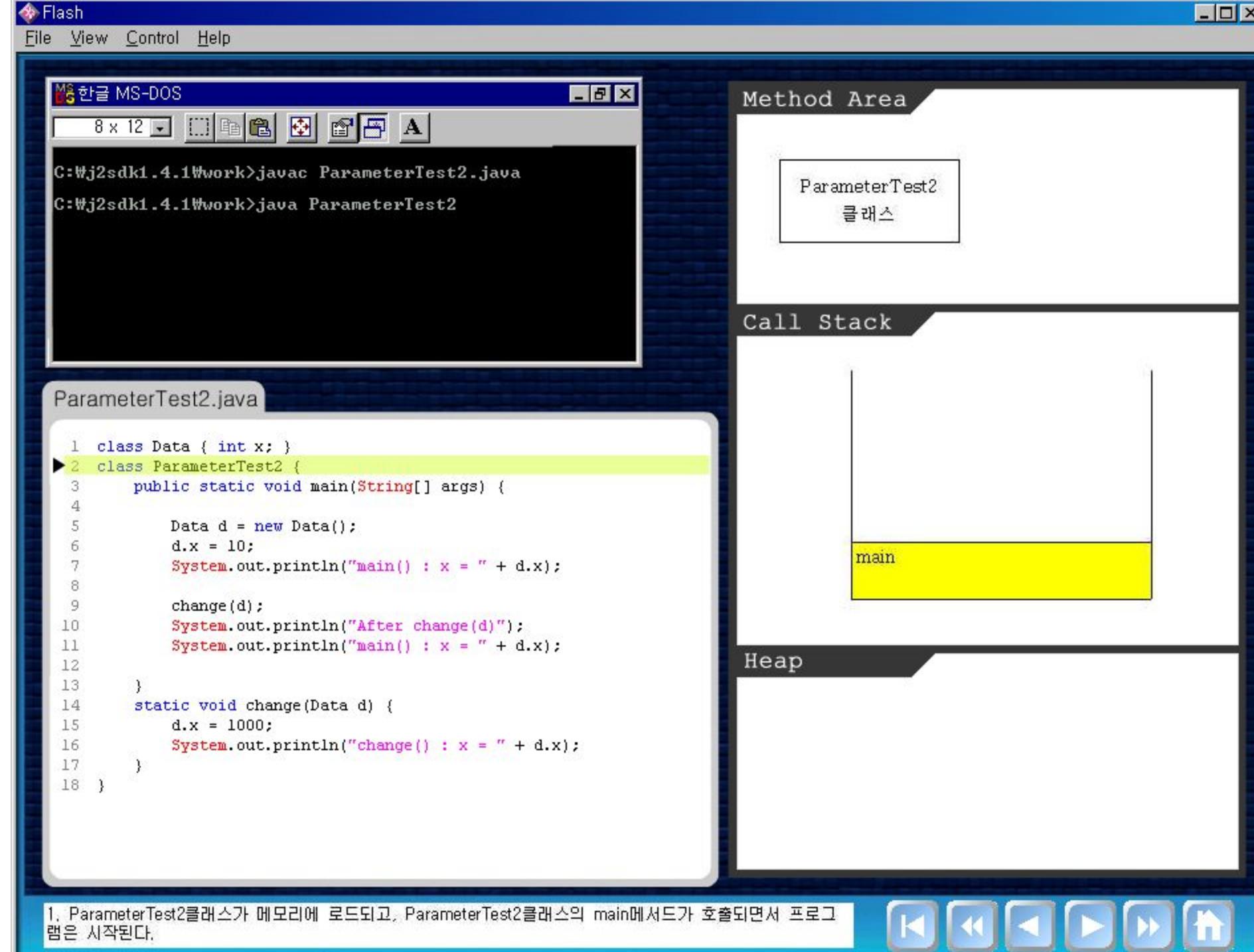
1 class Data { int x; }
2 class ParameterTest {
3     public static void main(String[] args) {
4
5         Data d = new Data();
6         d.x=10;
7         System.out.println("main() : x = " + d.x);
8
9         change(d.x);
10        System.out.println("After change(d.x)");
11        System.out.println("main() : x = " + d.x);
12    }
13
14    static void change(int x) {
15        x = 1000;
16        System.out.println("change() : x = " + x);
17    }
18 }
19

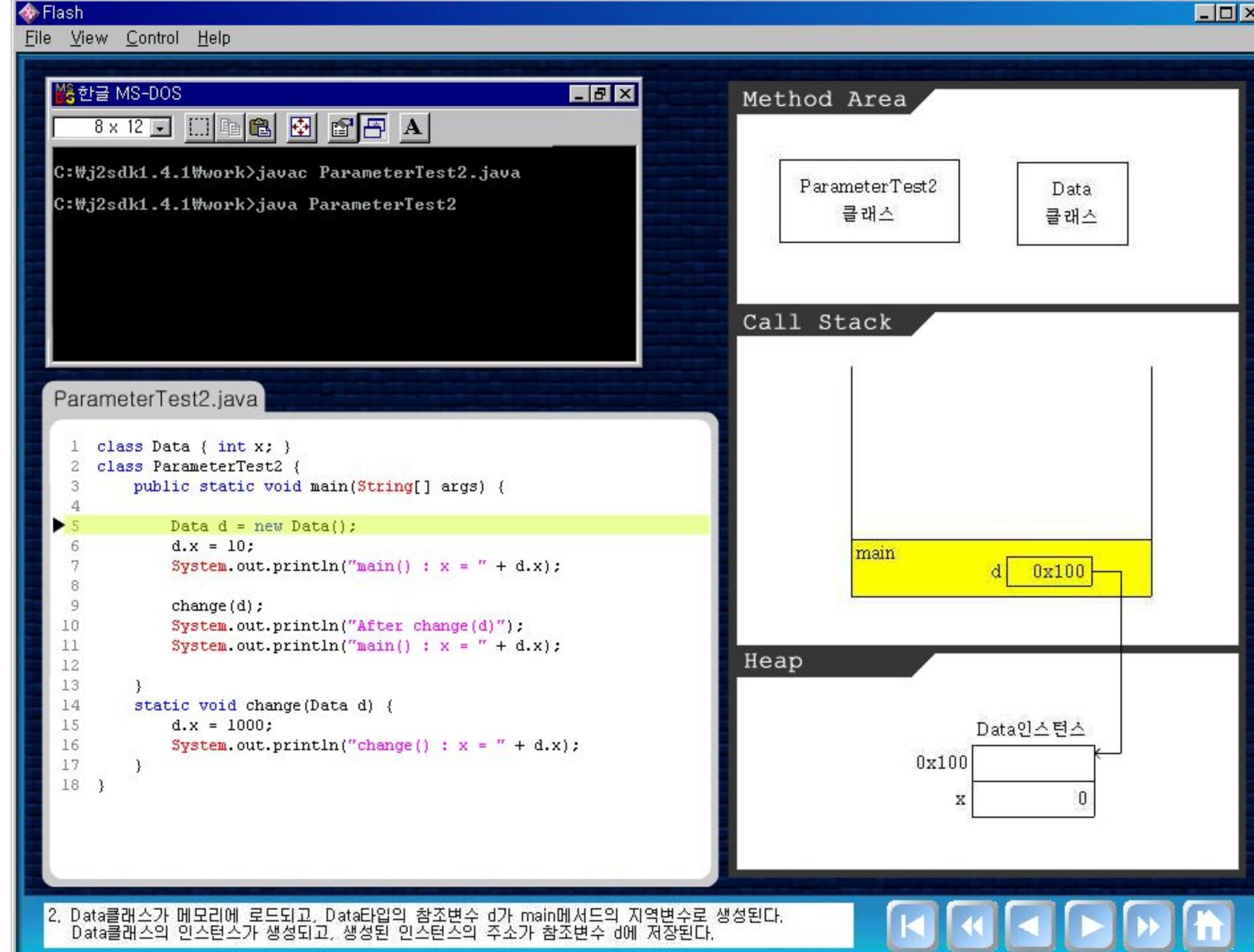
```



9. println메서드를 호출하여 d.x의 값을 출력한다. d.x의 값은 10이므로  
"main() : x = 10"을 출력한다.







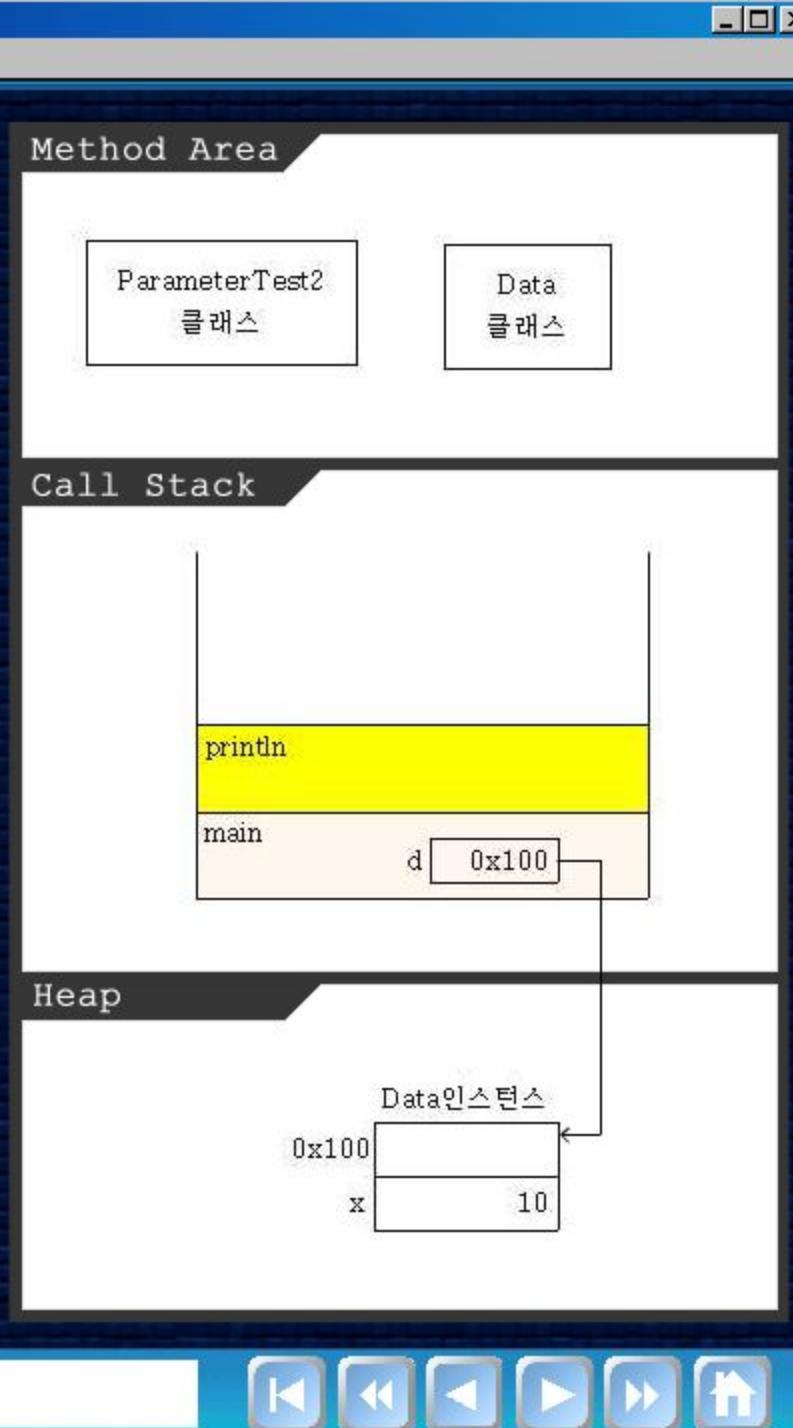
MS 한글 MS-DOS

8 x 12

```
C:\Wj2sdk1.4.1\work>javac ParameterTest2.java
C:\Wj2sdk1.4.1\work>java ParameterTest2
main() : x = 10
```

ParameterTest2.java

```
1 class Data { int x; }
2 class ParameterTest2 {
3     public static void main(String[] args) {
4
5         Data d = new Data();
6         d.x = 10;
7         System.out.println("main() : x = " + d.x);
8
9         change(d);
10        System.out.println("After change(d)");
11        System.out.println("main() : x = " + d.x);
12    }
13
14    static void change(Data d) {
15        d.x = 1000;
16        System.out.println("change() : x = " + d.x);
17    }
18 }
```



4. println메서드를 호출해서 d.x의 값을 출력한다.(println메서드의 수행내용은 생략했다.)



Flash  
File View Control Help

MS 한글 MS-DOS  
8 x 12 A

```
C:\Wj2sdk1.4.1\work>javac ParameterTest2.java
C:\Wj2sdk1.4.1\work>java ParameterTest2
main() : x = 10
```

ParameterTest2.java

```

1  class Data { int x; }
2  class ParameterTest2 {
3      public static void main(String[] args) {
4          Data d = new Data();
5          d.x = 10;
6          System.out.println("main() : x = " + d.x);
7
8          change(d);
9          System.out.println("After change(d)");
10         System.out.println("main() : x = " + d.x);
11     }
12 }
13
14 static void change(Data d) {
15     d.x = 1000;
16     System.out.println("change() : x = " + d.x);
17 }
18 }
```

Method Area

- ParameterTest2 클래스
- Data 클래스

Call Stack

Heap

5. change에서 d를 호출하면서 매개변수로 참조변수 d를 넘겨준다.  
main메서드의 참조변수 d의 값(Data인스턴스의 주소)은 change메서드의 매개변수 d에 복사된다.

Flash  
File View Control Help

MS 한글 MS-DOS

8 x 12

C:\Wj2sdk1.4.1\work>javac ParameterTest2.java  
C:\Wj2sdk1.4.1\work>java ParameterTest2  
main() : x = 10  
change() : x = 1000

ParameterTest2.java

```

1  class Data { int x; }
2  class ParameterTest2 {
3      public static void main(String[] args) {
4          Data d = new Data();
5          d.x = 10;
6          System.out.println("main() : x = " + d.x);
7
8          change(d);
9          System.out.println("After change(d)");
10         System.out.println("main() : x = " + d.x);
11     }
12 }
13 static void change(Data d) {
14     d.x = 1000;
15     System.out.println("change() : x = " + d.x);
16 }
17 }
18 }
```

Method Area

ParameterTest2 클래스  
Data 클래스

Call Stack

Heap

7. println메서드를 호출해서 d.x의 값을 출력한다.

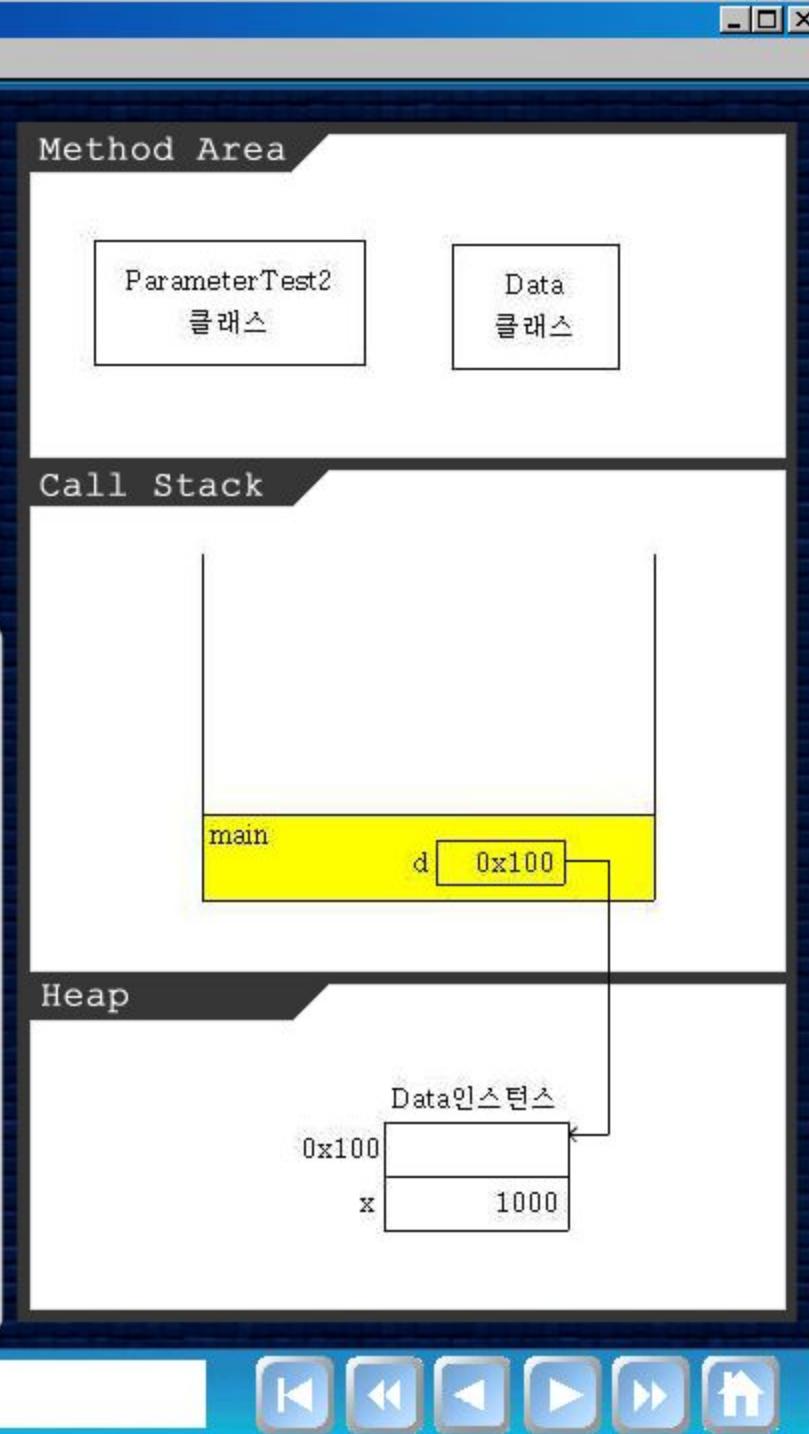
MS 한글 MS-DOS

8 x 12

```
C:\Wj2sdk1.4.1\work>javac ParameterTest2.java
C:\Wj2sdk1.4.1\work>java ParameterTest2
main() : x = 10
change() : x = 1000
After change(d)
main() : x = 1000
```

ParameterTest2.java

```
1  class Data { int x; }
2  class ParameterTest2 {
3      public static void main(String[] args) {
4
5          Data d = new Data();
6          d.x = 10;
7          System.out.println("main() : x = " + d.x);
8
9          change(d);
10         System.out.println("After change(d)");
11         System.out.println("main() : x = " + d.x);
12     }
13 }
14 static void change(Data d) {
15     d.x = 1000;
16     System.out.println("change() : x = " + d.x);
17 }
```



9. println메서드를 호출하여 d.x의 값을 출력한다. d.x의 값은 1000이므로 "main() : x = 1000"을 출력한다.

## 3.8 재귀 호출(recursive call)

### ▶ 재귀 호출이란?

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

### ▶ 재귀 호출의 예(例)

- 팩토리얼, 제곱, 트리운행, 폴더목록표시 등

\* 팩토리얼 (factorial)

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

$$f(n) = n * f(n-1) \text{ 단, } f(1) = 1$$

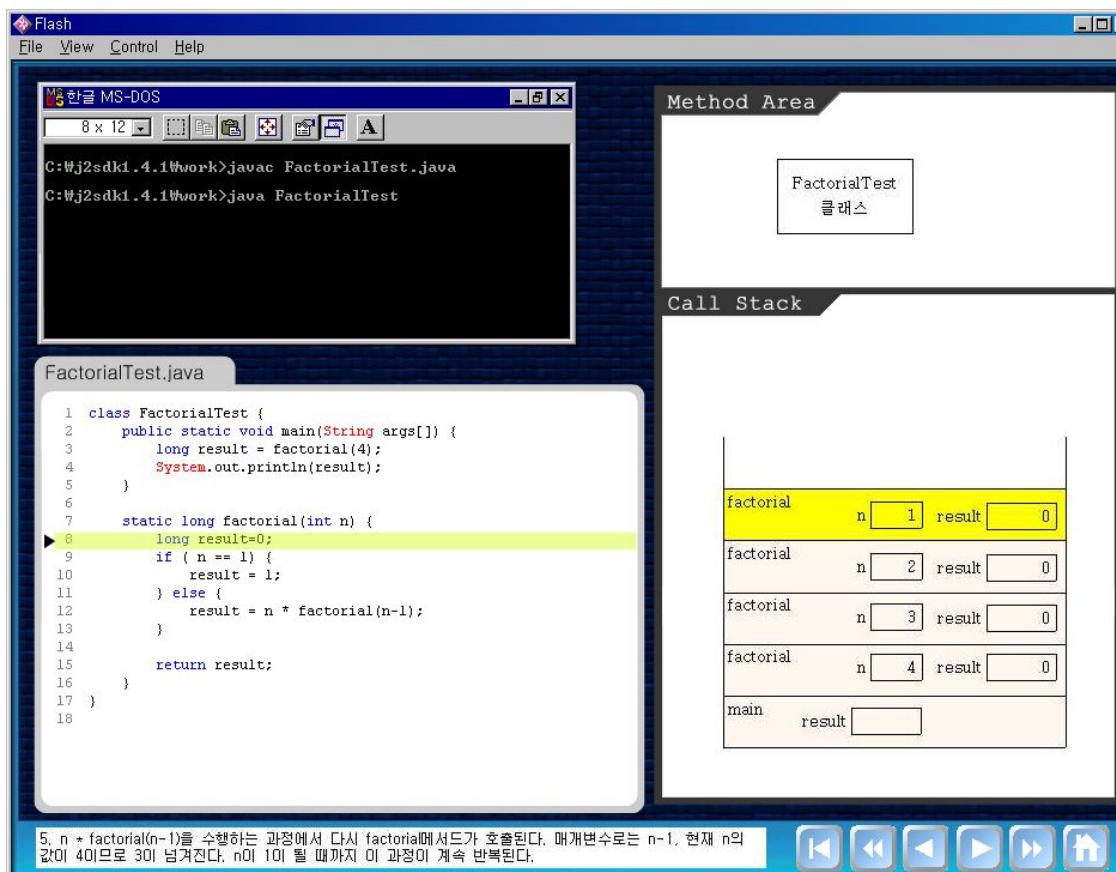


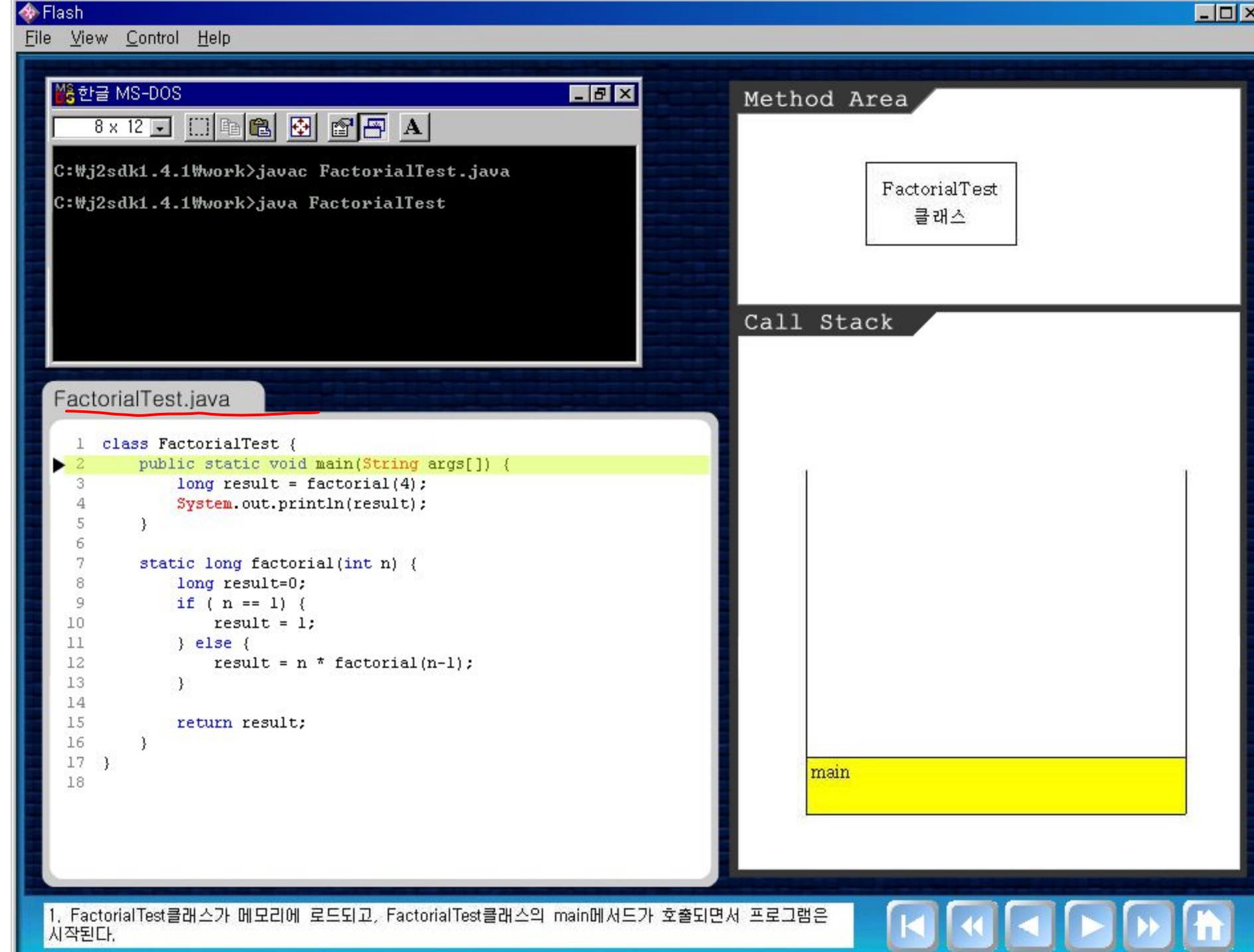
```
long factorial(int n) {  
    long result = 0;  
  
    if(n==1) {  
        result = 1;  
    } else {  
        result = n * factorial(n-1);  
    }  
  
    return result;  
}
```

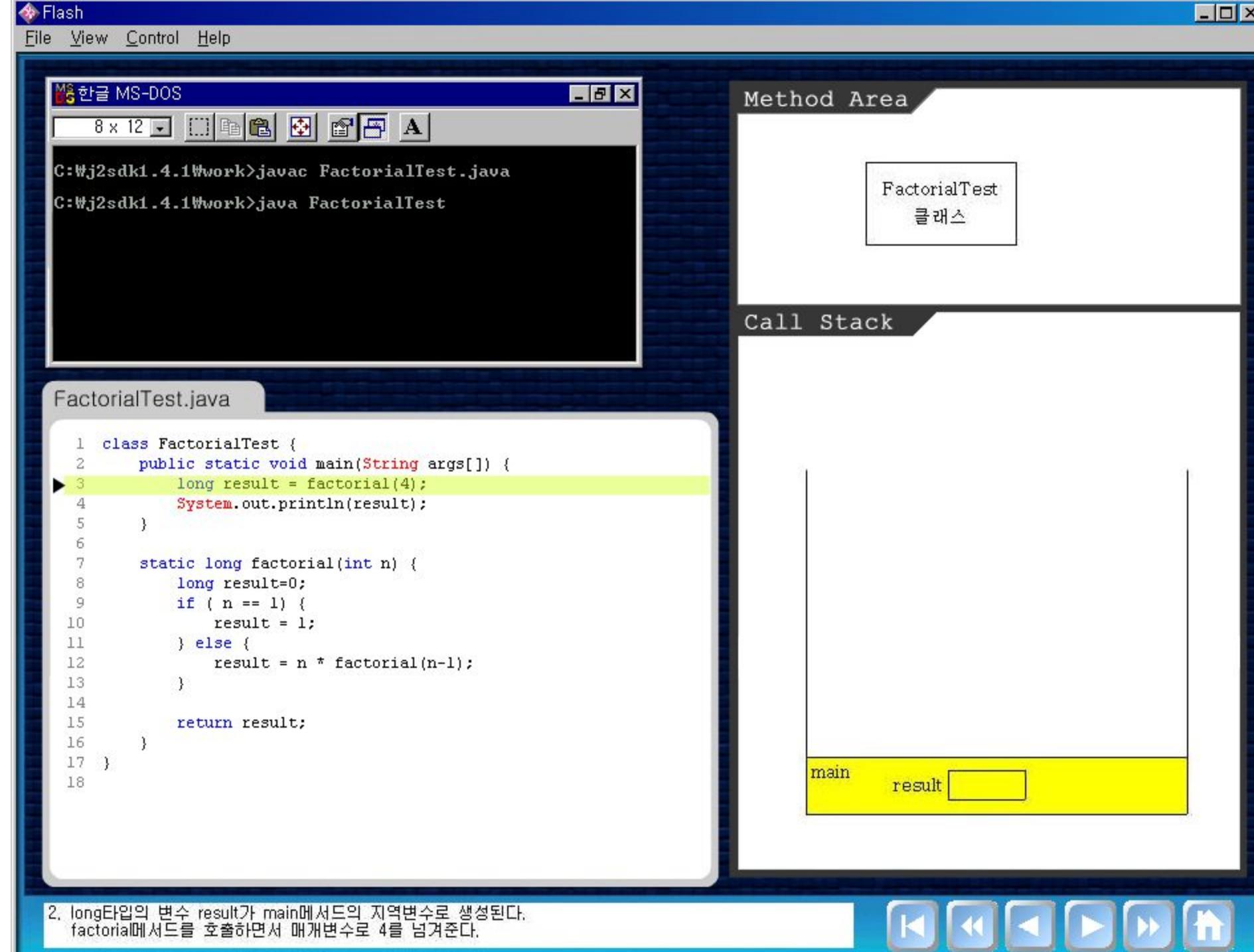
## 3.8 재귀 호출(recursive call)

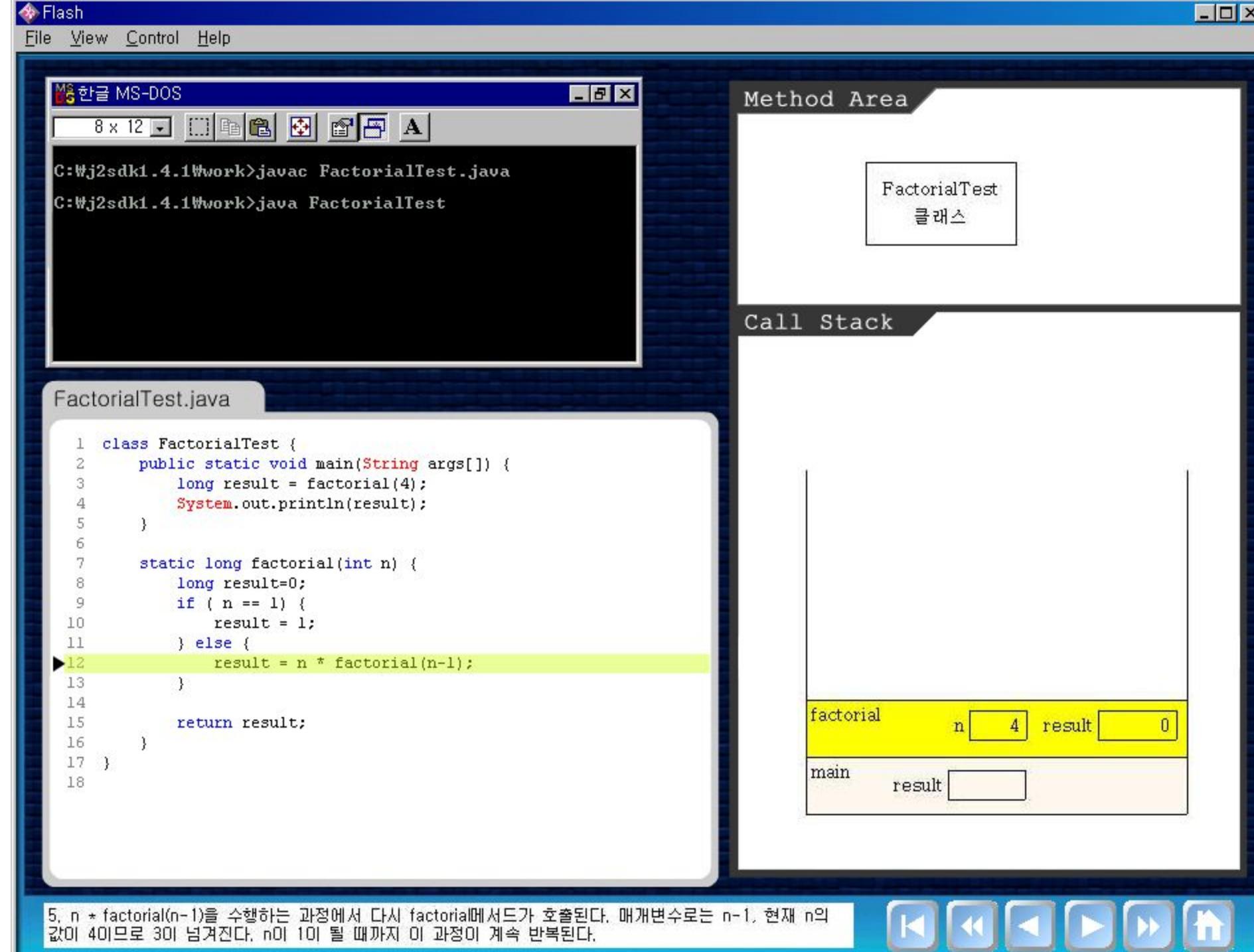
\* 플래시 동영상 : RecursiveCall.exe

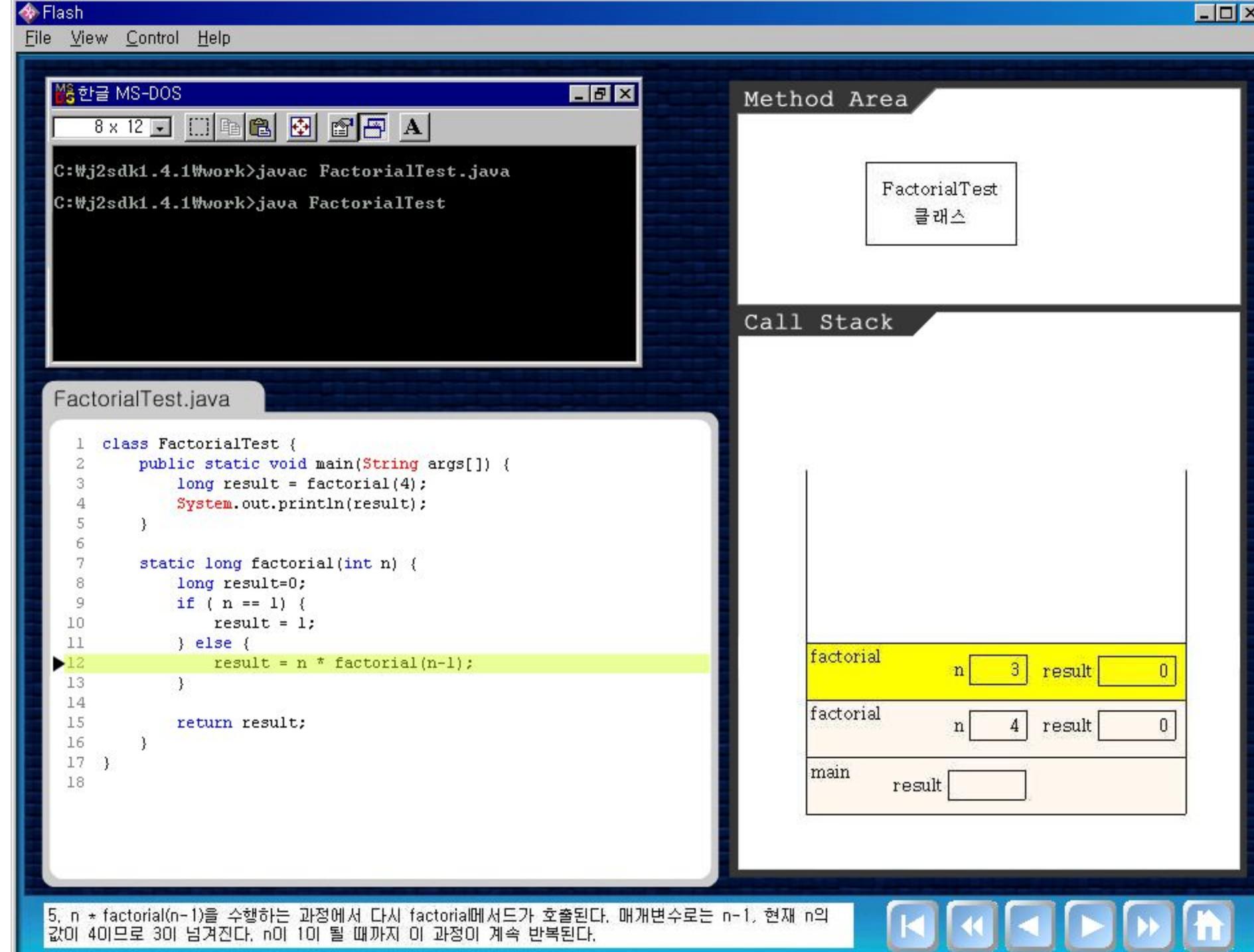
(java\_jungsuk\_src.zip의 flash폴더에 위치)

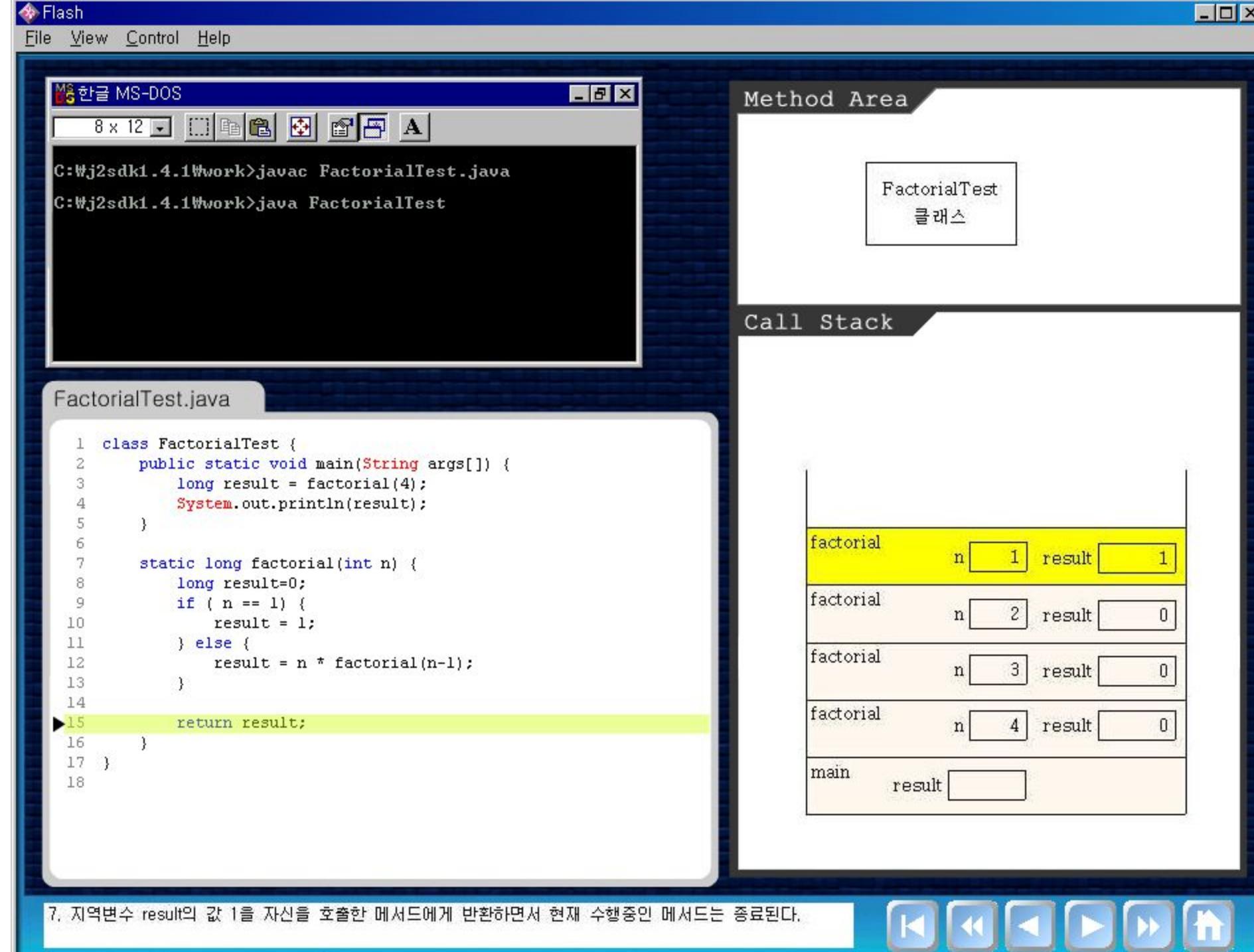


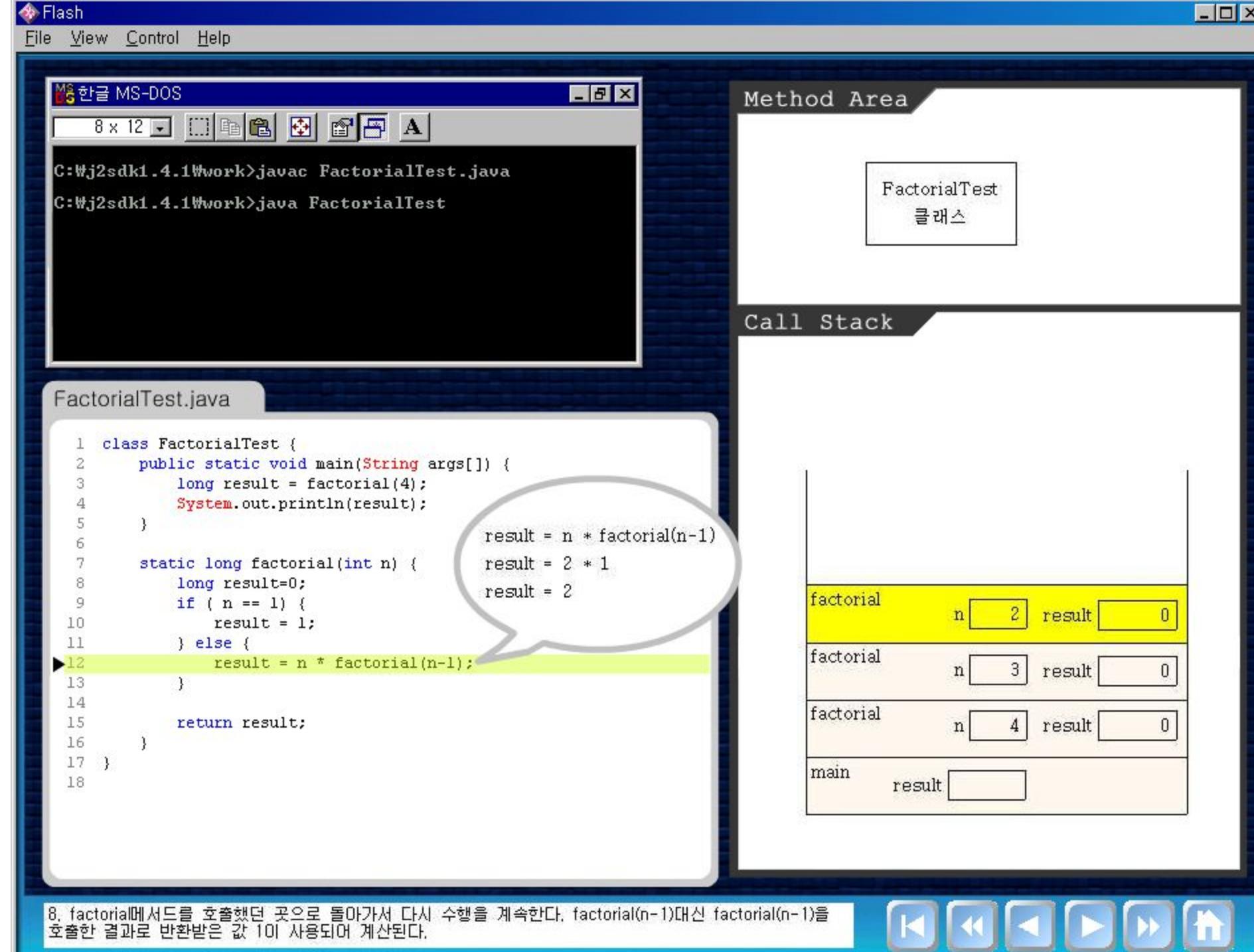












Flash  
File View Control Help

MS 한글 MS-DOS  
8 x 12 A

C:\Wj2sdk1.4.1\work>javac FactorialTest.java  
C:\Wj2sdk1.4.1\work>java FactorialTest

Method Area  
FactorialTest  
클래스

Call Stack

FactorialTest.java

```
1 class FactorialTest {  
2     public static void main(String args[]) {  
3         long result = factorial(4);  
4         System.out.println(result);  
5     }  
6     static long factorial(int n) {  
7         long result=0;  
8         if ( n == 1) {  
9             result = 1;  
10        } else {  
11            result = n * factorial(n-1);  
12        }  
13    }  
14    return result;  
15 }  
16 }  
17 }
```

▶ 12 result = n \* factorial(n-1);

result = n \* factorial(n-1)  
result = 3 \* 2  
result = 6

factorial n 3 result 0  
factorial n 4 result 0  
main result

9. 지역변수 result의 값 2를 자신을 호출한 메서드에게 반환하면서 현재 수행중인 메서드는 종료된다.  
main메서드에서 factorial메서드를 처음 호출한 곳으로 돌아갈 때까지 이 과정이 되풀이 된다.

Flash  
File View Control Help

MS 한글 MS-DOS  
8 x 12 A

```
C:\Wj2sdk1.4.1\work>javac FactorialTest.java
C:\Wj2sdk1.4.1\work>java FactorialTest
```

Method Area

FactorialTest  
클래스

Call Stack

FactorialTest.java

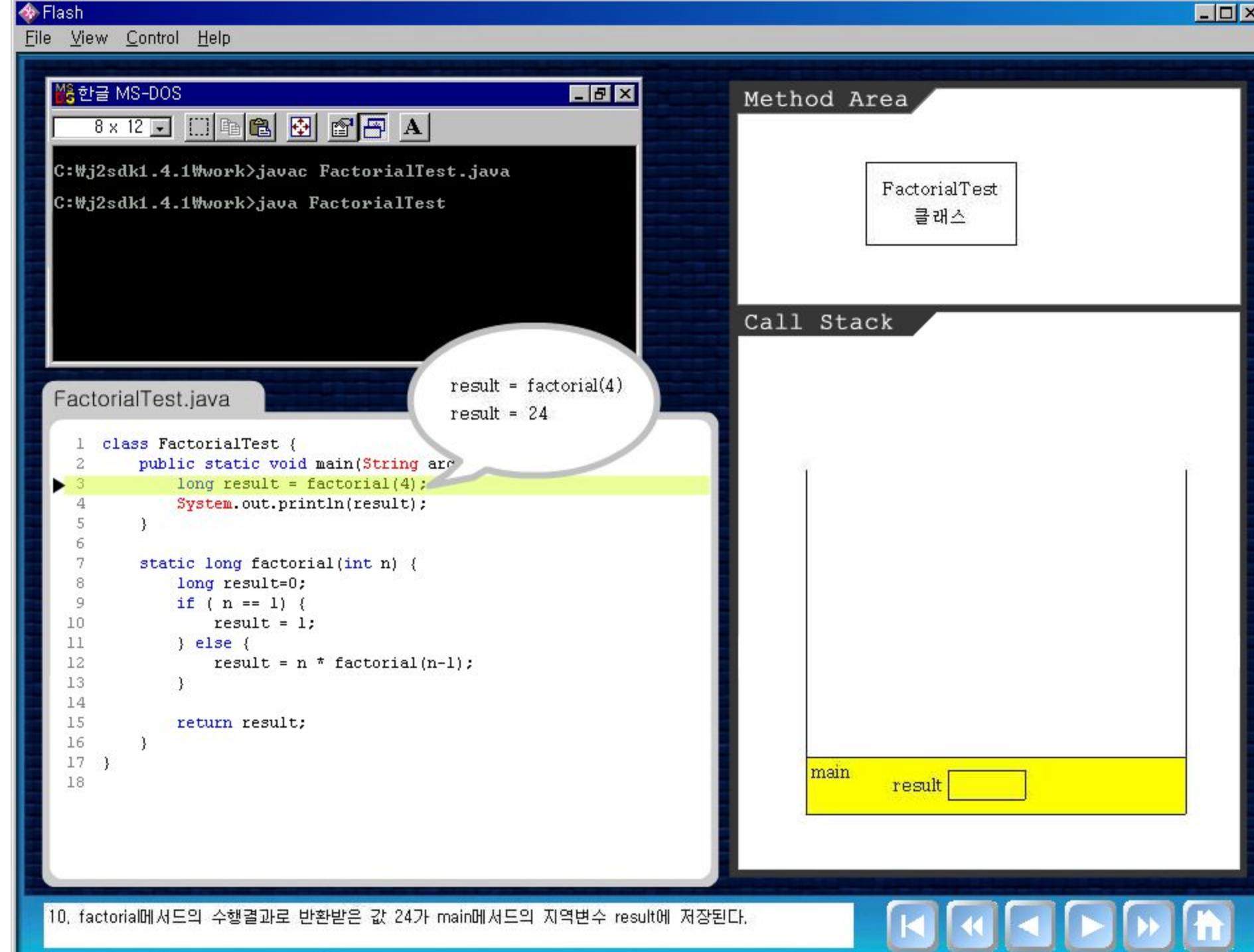
```
1 class FactorialTest {
2     public static void main(String args[]) {
3         long result = factorial(4);
4         System.out.println(result);
5     }
6
7     static long factorial(int n) {
8         long result=0;
9         if ( n == 1) {
10             result = 1;
11         } else {
12             result = n * factorial(n-1); // This line is highlighted with a yellow bar
13         }
14
15         return result;
16     }
17 }
18 }
```

▶ 12 result = n \* factorial(n-1);

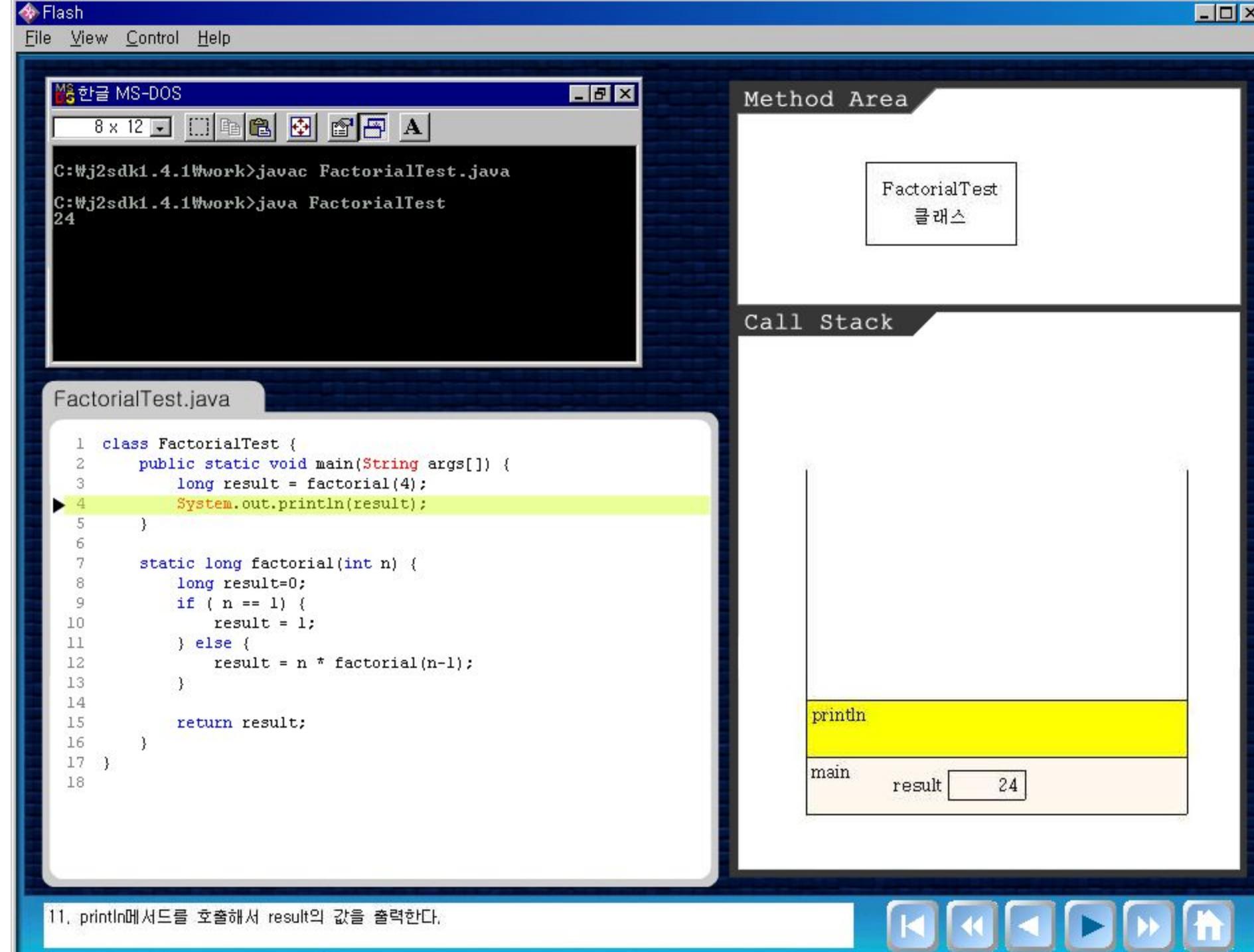
result = n \* factorial(n-1)  
result = 4 \* 6  
result = 24

factorial	n	4	result	0
main			result	

9. 지역변수 result의 값 2를 자신을 호출한 메서드에게 반환하면서 현재 수행중인 메서드는 종료된다.  
main메서드에서 factorial메서드를 처음 호출한 곳으로 돌아갈 때까지 이 과정이 되풀이 된다.



10. factorial메서드의 수행결과로 반환받은 값 24가 main메서드의 지역변수 result에 저장된다.



## 3.9 클래스메서드(static메서드)와 인스턴스메서드

### ▶ 인스턴스메서드

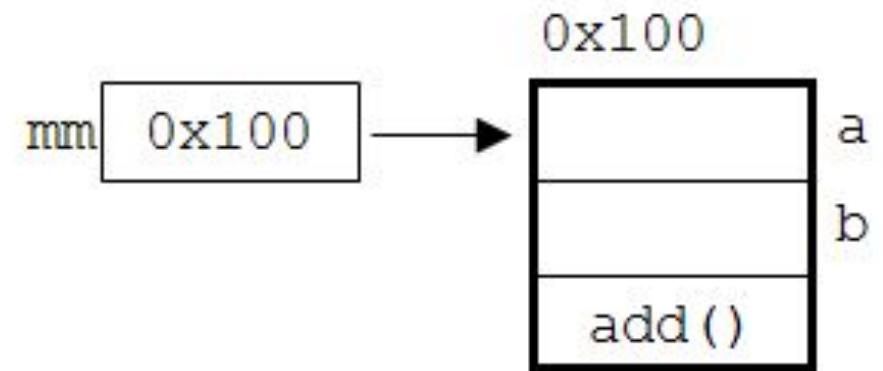
- 인스턴스 생성 후, ‘참조변수.메서드이름()’으로 호출
- 인스턴스변수나 인스턴스메서드와 관련된 작업을 하는 메서드
- 메서드 내에서 인스턴스변수 사용 가능

### ▶ 클래스메서드(static메서드)

- 객체생성없이 ‘클래스이름.메서드이름()’으로 호출
- 인스턴스변수나 인스턴스메서드와 관련없는 작업을 하는 메서드
- 메서드 내에서 인스턴스변수 사용불가
- 메서드 내에서 인스턴스변수를 사용하지 않는다면 static을 붙이는 것을 고려한다.

### 3.9 클래스메서드(static메서드)와 인스턴스메서드

```
class MyMath2 {  
    long a, b;  
  
    long add() { // 인스턴스메서드  
        return a + b;  
    }  
    .  
    static long add(long a, long b) { // 클래스메서드(static메서드)  
        return a + b;  
    }  
}
```



```
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  
    }  
}
```

## 3.10 멤버간의 참조와 호출(1/2) – 메서드의 호출

“같은 클래스의 멤버간에는 객체생성이나 참조변수 없이 참조할 수 있다. 그러나 static멤버들은 인스턴스멤버들을 참조할 수 없다.”

```
class TestClass {  
    void instanceMethod() {}          // 인스턴스메서드  
    static void staticMethod() {} // static메서드  
  
    void instanceMethod2() {           // 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
```

### 3.10 멤버간의 참조와 호출(2/2) – 변수의 접근

“같은 클래스의 멤버간에는 객체생성이나 참조변수 없이 참조할 수 있다. 그러나 static멤버들은 인스턴스멤버들을 참조할 수 없다.”

```
class TestClass2 {  
    int iv;          // 인스턴스변수  
    static int cv;   // 클래스변수  
  
    void instanceMethod() {      // 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. 에서드 오버로딩

## 4.1 메서드 오버로딩(method overloading)이란?

“하나의 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(1/3)

- ▶ `System.out.println` 메서드
  - 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)
```

## 4.3 오버로딩의 예(1/2)

- ▶ 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); }
```

## 4.3 오버로딩의 예(1/3)

- ▶ 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

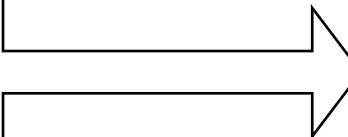
### [보기4]

```
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;
}
```

## 액체지향개념 I-3

1. 객체지향언어란?

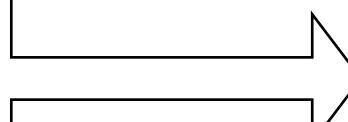


객체지향개념 I-1

2. 클래스와 객체



3. 변수와 메서드

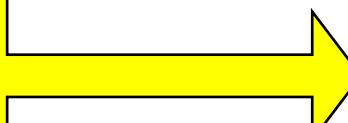


객체지향개념 I-2

4. 메서드 오버로딩



5. 생성자



객체지향개념 I-3

6. 변수의 초기화

## 5. 생성자

5.1 생성자란?

5.2 생성자의 조건

5.3 기본 생성자

5.4 매개변수가 있는 생성자

5.5 생성자에서 다른 생성자 호출하기 – `this()`

5.6 참조변수 `this`

5.7 생성자를 이용한 인스턴스의 복사

## 6. 변수의 초기화

6.1 변수의 초기화

6.2 멤버변수의 초기화

6.3 초기화 블럭

6.4 멤버변수의 초기화 시기와 순서

## 5. 생성자

## 5.1 생성자(constructor)란?

### ▶ 생성자란?

- 인스턴스가 생성될 때마다 호출되는 ‘인스턴스 초기화 메서드’
- 인스턴스 변수의 초기화 또는 인스턴스 생성시 수행할 작업에 사용
- 몇 가지 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 `new`에 의해서 메모리(heap)에 `Card`클래스의 인스턴스가 생성된다.
2. 생성자 `Card()`가 호출되어 수행된다.
3. 연산자 `new`의 결과로, 생성된 `Card`인스턴스의 주소가 반환되어 참조변수 `c`에 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.”

## 5.3 기본 생성자(default constructor)

“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 error 발생  
    }  
}
```

```
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);
```

## 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);  
}
```

## 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         //Car("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  
this 사용

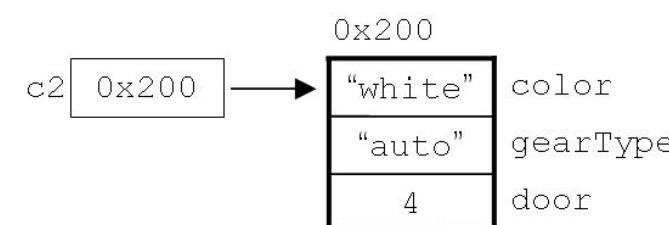
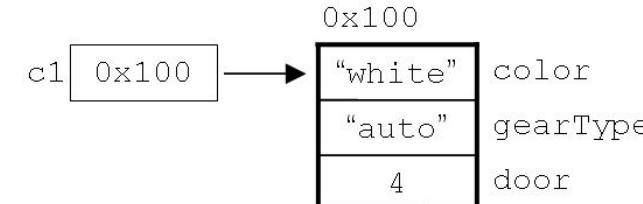
```
Car(String color, String gearType, int door) {  
    this.color = color;  
    this.gearType = gearType;  
    this.door = door;  
}
```



## 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);  
}
```

## 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.

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

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

## 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 공통 수행 작업에 사용되며 인스턴스 생성시 우선적 실행

```
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 }
```

## 6.4 멤버변수의 초기화 시기와 순서

- ▶ 클래스변수 초기화 시점 : 클래스가 처음 로딩될 때 단 한번
- ▶ 인스턴스변수 초기화 시점 : 인스턴스가 생성될 때마다

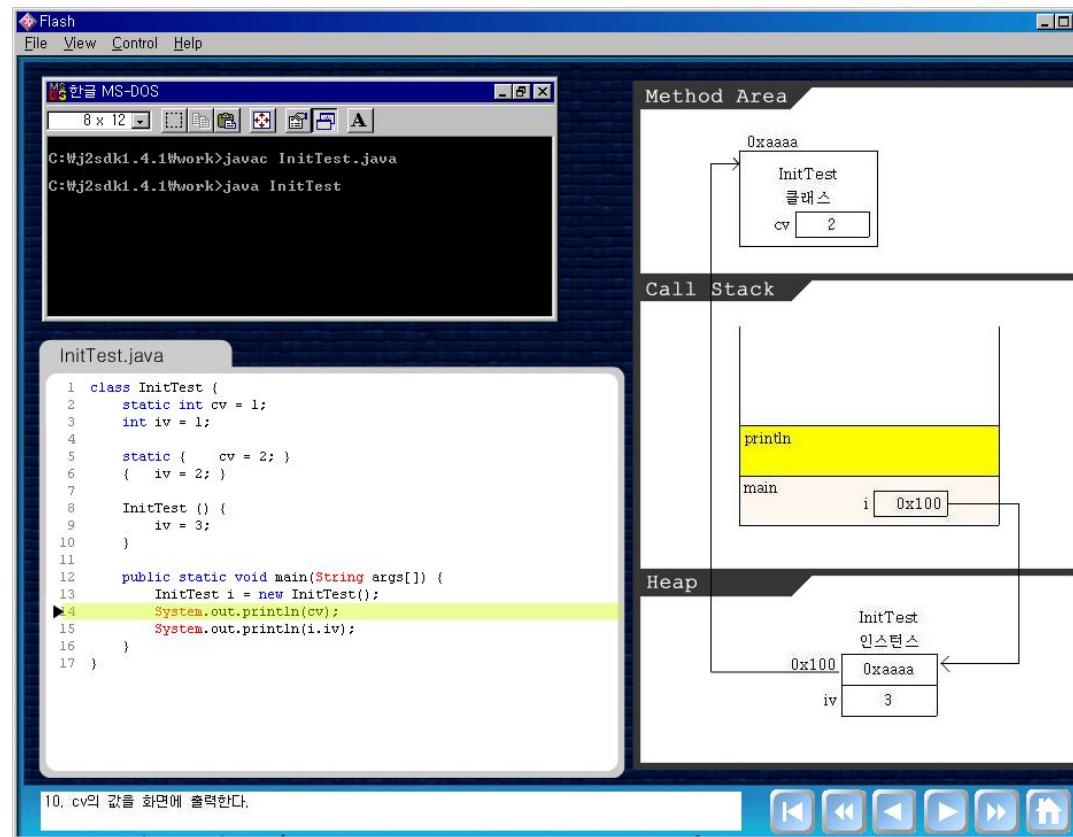
```
1 class InitTest {  
2     static int cv = 1; // 명시적 초기화  
3     int iv = 1;       // 명시적 초기화  
4  
5     static {    cv = 2; } // 클래스 초기화 블럭  
6     {      iv = 2; }   // 인스턴스 초기화 블럭  
7  
8     InitTest() { // 생성자  
9         iv = 3;  
10    }  
11 }
```

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 ]	cv [ 2 ]	cv [ 2 ]	cv [ 2 ]
			iv [ 0 ]	iv [ 1 ]	iv [ 2 ]	iv [ 3 ]
1	2	3	4	5	6	7

## 6.4 멤버변수의 초기화 시기와 순서

\* 플래시 동영상 : Initialization.exe 또는 Initialization.swf  
(java\_jungsuk\_src.zip의 flash폴더에 위치)



## 6.4 멤버변수의 초기화 시기와 순서 - 예제설명

```
1 class Product {  
2     static int count = 0;      // 생성된 인스턴스의 수를 저장하기 위한 변수  
3     int serialNo;           // 인스턴스 고유의 번호  
4  
5     { // 인스턴스 초기화 블럭 : 모든 생성자에서 공통적으로 수행될 코드  
6         ++count;  
7         serialNo = count;  
8     }  
9  
10    public Product() {}  
11 }  
12  
13 class ProductTest {  
14     public static void main(String args[]) {  
15         Product p1 = new Product();  
16         Product p2 = new Product();  
17         Product p3 = new Product();  
18  
19         System.out.println("p1의 제품번호(serial no)는 " + p1.serialNo);  
20         System.out.println("p2의 제품번호(serial no)는 " + p2.serialNo);  
21         System.out.println("p3의 제품번호(serial no)는 " + p3.serialNo);  
22         System.out.println("생산된 제품의 수는 모두 "+Product.count+"개 입니다.");  
23     }  
24 }
```

