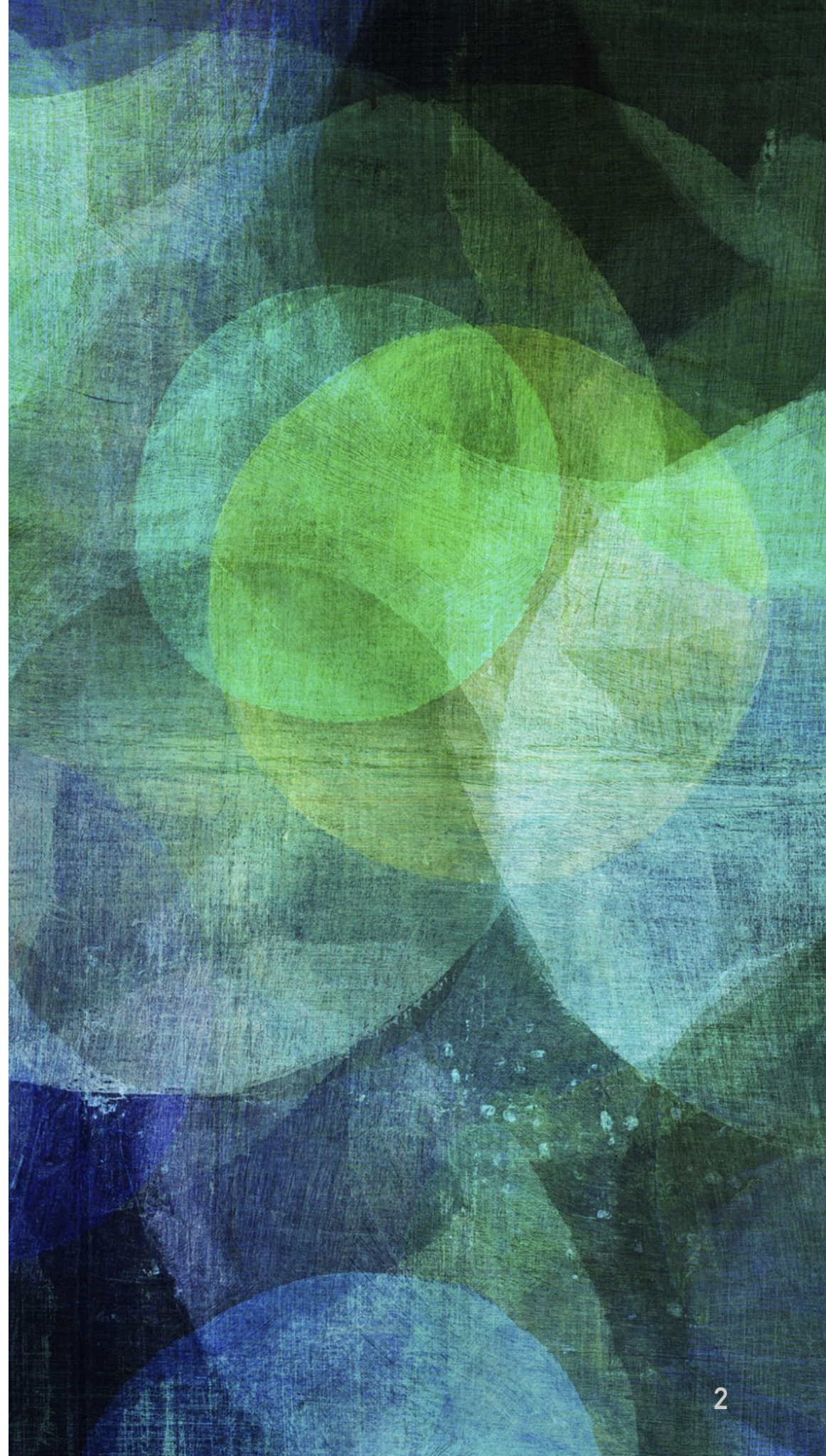


자바 기초 2

장서윤 pineai@cnu.ac.kr

문자열, 배열



문자열의 선언과 생성

```
String 변수;    // String 타입의 변수 선언  
변수 = "문자열"; // String 타입의 변수에 문자열 대입
```

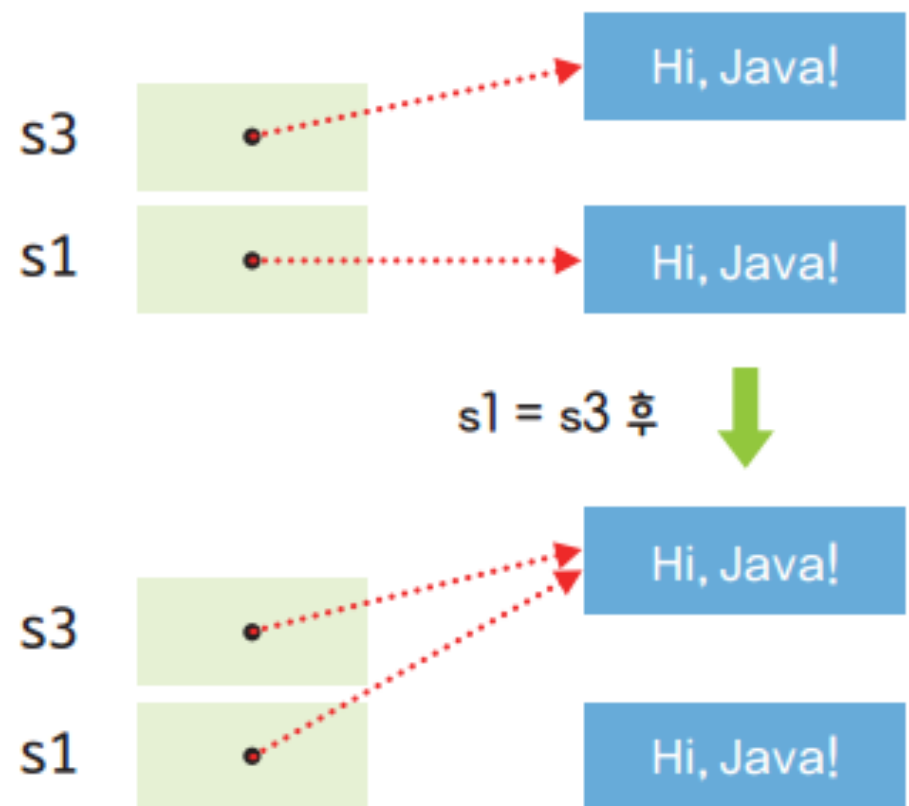
```
String s1 = "안녕, 자바!"; // String 타입의 변수 선언과 초기화  
String s2 = "안녕, 자바!"; // String 타입의 변수 선언과 초기화
```

문자열 리터럴이다.

- ▶ 문자열 리터럴은 내부적으로 `new String()`을 호출해 생성한 객체이다.
- ▶ 따라서 `s1`은 `new String("안녕, 자바!")`를 호출해서 생성한 객체를 가리킨다.

문자열의 비교(1)

- ▶ ==와 != 연산자는 두 문자열의 내용을 비교하는 것이 아니라 동일한 객체인지 검사



참조 변수가 없으므로 사용할 수 없는 객체가 된다.
따라서 후에 가비지 컬렉터로 자동 수거된다.

문자열의 비교(2)

.....

➤ String 클래스에서 제공하는 문자열 비교 메서드

메서드	설명
<code>int compareTo(String s)</code>	문자열을 사전 순으로 비교해 정수 값을 반환한다.
<code>int compareToIgnoreCase(String s)</code>	대 · 소문자를 무시하고, 문자열을 사전 순으로 비교한다.
<code>boolean equals(String s)</code>	주어진 문자열 s와 현재 문자열을 비교한 후 true/false를 반환한다.
<code>boolean equalsIgnoreCase(String s)</code>	주어진 문자열 s와 현재 문자열을 대 · 소문자 구분 없이 비교한 후 true/false를 반환한다.

➤ 예

```
// 문자열 s1과 s2가 가리키는 내용이 같으면 true, 다르면 false를 반환한다.
```

```
boolean result = s1.equals(s2);
```

s1은 원본 문자열, s2는 비교 문자열이다.

```
// 문자열 s1과 s2가 가리키는 내용이 같으면 0, 다르면 0이 아닌 정수를 반환한다.
```

```
int result = s1.compareTo(s2);
```

문자열의 비교(3)

.....

➤ 문자열의 조작

➤ String 클래스에서 제공하는 메서드

메서드	설명
char charAt(int index)	index가 지정한 문자를 반환한다.
String concat(String s)	주어진 문자열 s를 현재 문자열 뒤에 붙인다.
boolean contains(String s)	문자열 s를 포함하는지 조사한다.
boolean endsWith(String s)	끝나는 문자열이 s인지 조사한다.
boolean isEmpty()	문자열의 길이가 0이면 true를 반환한다.
int length()	문자열의 길이를 반환한다.
boolean startsWith(String s)	시작하는 문자열이 s인지 조사한다.
String substring(int index)	index부터 시작하는 문자열의 일부를 반환한다.
String toLowerCase()	문자열을 모두 소문자로 변환한다.
String toUpperCase()	문자열을 모두 대문자로 변환한다.
String trim()	문자열 앞뒤에 있는 공백을 제거한 후 반환한다.

이스케이프 시퀀스(ESCAPE SEQUENCE)

▶ 문자열 안에서 특별한 의미로 해석되는 문자

- \n 개행
- \t 탭(Tab)
- \" 큰 따옴표(Quotation mark)
- \\ 역슬래쉬(Backslash)

대표적인 이스케이프 시퀀스

```
System.out.println("제가 어제 "당신 누구세요?" 라고 물었더니");
```

문자열 안에 큰 따옴표가 들어가면 이는 문자열의 구분자로 인식된다.

```
System.out.println("제가 어제 \"당신 누구세요?\"라고 물었더니");
```

문자열 안에 큰 따옴표를 삽입하려면 이스케이프 시퀀스 사용!

SYSTEM.OUT.PRINTLN과 SYSTEM.OUT.PRINT

.....

```
class Friend
{
    String myName;
    public Friend(String name)
    {
        myName=name;
    }
    public String toString()
    {
        return "제 이름은 "+myName+"입니다.";
    }
}
```

```
class StringToString
{
    public static void main(String[] args)
    {
        Friend fnd1=new Friend("이종수");
        Friend fnd2=new Friend("현주은");
        System.out.println(fnd1);
        System.out.println(fnd2);
        System.out.print("출력이 ");
        System.out.print("종료되었습니다.");
        System.out.println("");
    }
}
```

실행결과

```
제 이름은 이종수입니다.
제 이름은 현주은입니다.
출력이 종료되었습니다.
```


JAVA의 입력

▶ Scanner 클래스

```
import java.util.Scanner;

class StringScanning
{
    public static void main(String[] args)
    {
        String source="1 5 7";
        Scanner sc=new Scanner(source);

        int num1=sc.nextInt();
        int num2=sc.nextInt();
        int num3=sc.nextInt();
        int sum=num1+num2+num3;

        System.out.printf(
            "문자열에 저장된 %d, %d, %d의 합은 %d \n",
            num1, num2,num3, sum);
    }
}
```

실행결과

문자열에 저장된 1, 5, 7의 합은 13

SCANNER 클래스를 구성하는 다양한 메소드

.....

- public boolean nextBoolean()
- public byte nextByte()
- public short nextShort()
- public int nextInt()
- public long nextLong()
- public float nextFloat()
- public double nextDouble()
- public String nextLine()

실행결과

당신의 이름은? 이은주

안녕하세요 이은주님

당신은 스파게티를 좋아한다는데, 진실입니까? true

오~ 좋아하는군요.

당신과 동생의 키는 어떻게 되나요? 162.4 170.9

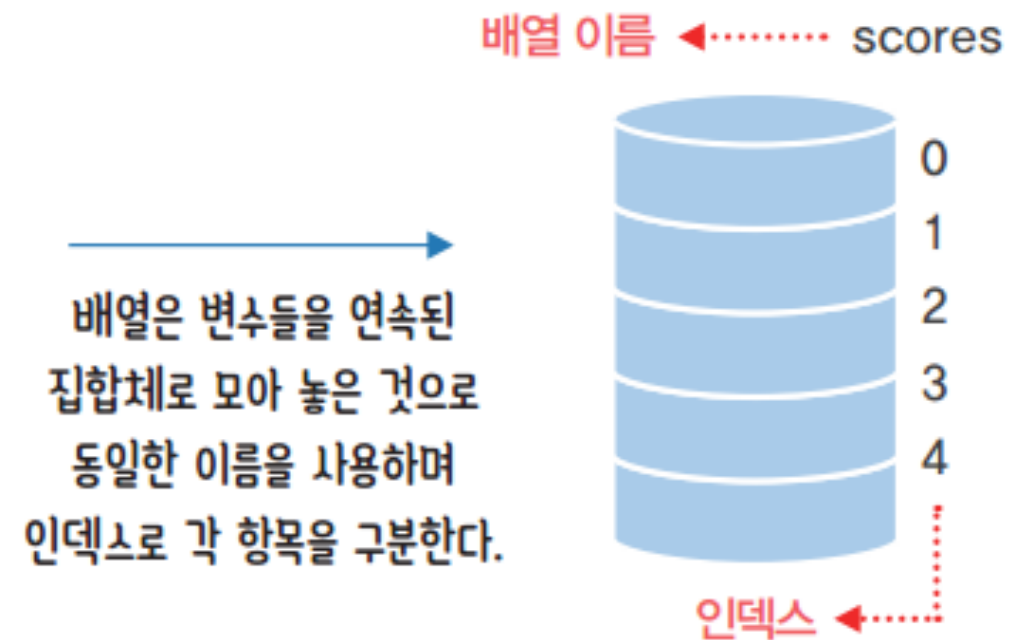
당신이 8.5만큼 작군요.

```
public static void main(String[] args)
{
    Scanner keyboard=new Scanner(System.in);
    System.out.print("당신의 이름은? ");
    String str=keyboard.nextLine();
    System.out.println("안녕하세요 "+str+'님');
    System.out.print("당신은 스파게티를 좋아한다는데, 진실입니까? ");
    boolean isTrue=keyboard.nextBoolean();
    if(isTrue==true)
        System.out.println("오~ 좋아하는군요.");
    else
        System.out.println("이런 아니었군요.");

    System.out.print("당신과 동생의 키는 어떻게 되나요? ");
    double num1=keyboard.nextDouble();
    double num2=keyboard.nextDouble();
    double diff=num1-num2;
    if(diff>0)
        System.out.println("당신이 "+diff+"만큼 크군요.");
    else
        System.out.println("당신이 "+(-diff)+"만큼 작군요.");
}
```

배열의 개념(1)

- ▶ 배열 (Array)은 동일한 데이터 타입의 집합을 쉽게 처리할 수 있는 데이터 구조



배열의 개념(2)

- ▶ 배열이 필요한 경우 : 5과목의 평균 점수를 구하는 경우

```
int score1 = 100;
int score2 = 90;
int score3 = 50;
int score4 = 95;
int score5 = 85;

int sum = score1;
sum += score2;
sum += score3;
sum += score4;
sum += score5;
double average = sum / 5;
```

(a) 배열을 사용하지 않을 때

```
int[] scores = { 100, 90, 50, 95, 85 };
int sum = 0;

for (int i = 0; i < 5; i++)
    sum += scores[i];
double average = sum / 5;
```

(b) 배열을 사용할 때

배열의 선언과 생성(1)

➤ 배열의 선언

➤ 실제로는 배열 변수의 선언

```
int[] scores;      혹은      int scores[];
```

```
int scores[5];
```

➤ 배열의 선언과 생성

➤ 실제로는 배열 변수의 선언과 초기화

배열의 크기

```
scores = new int[5];
```



배열의 선언과 생성(2)

▶ 배열의 선언과 생성 예

// 방법 ①

```
int[] scores = { 100, 90, 50, 95, 85 };
```

// 방법 ②

```
int[] scores = new int[] { 100, 90, 50, 95, 85 };
```

// 방법 ③

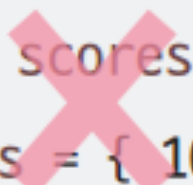
```
int[] scores;
```

```
scores = new int[] { 100, 90, 50, 95, 85 };
```

// 방법 ④

```
int[] scores;
```

```
scores = { 100, 90, 50, 95, 85 };
```



배열 원소의 접근과 배열의 크기

➤ 배열 원소의 접근

```
배열이름[인덱스];
```

➤ 배열의 크기

- 배열이 생성될 때 배열의 크기가 결정
- 배열의 length 필드가 배열의 크기를 나타냄.
 - 예를 들어, scores가 가리키는 배열의 크기는 scores.length

다차원 배열(1)

배열의 배열

예

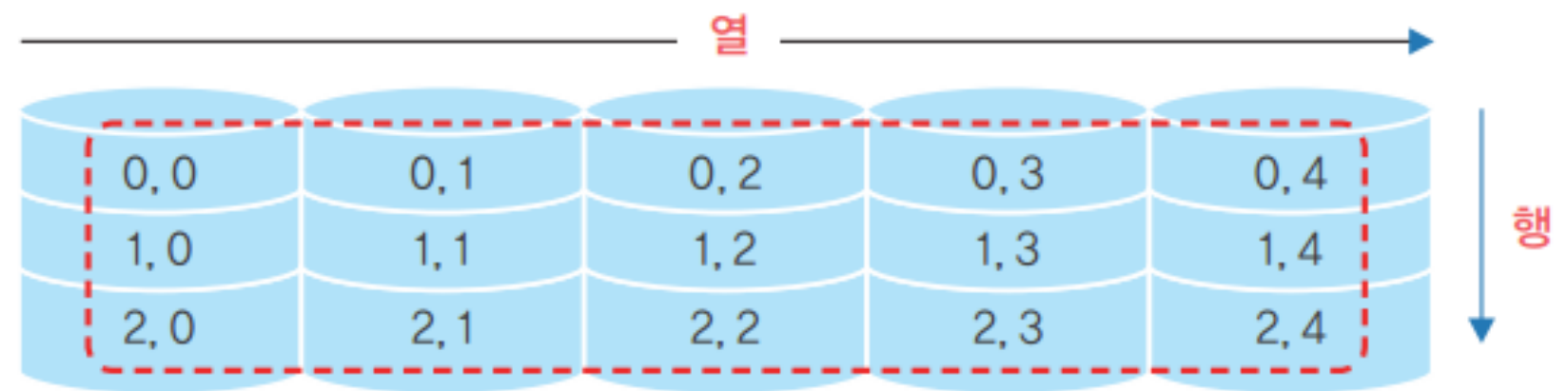
- ▶ 학생 3명의 5과목 성적을 처리하는 정수 타입 2차원 배열(3행 × 5열)인 scores

```
int[][] scores = new int[3][5];
```

2개의 대괄호는
2차원 배열을 표시

행의 개수

열의 개수



인덱스

첫 번째 인덱스는 행 번호이며, 두 번째 인덱스는 열 번호이다.

다차원 배열(2)

➤ 선언과 초기화

```
int[][] scores = {{100, 90, 50, 95, 85}, {70, 60, 82, 75, 40}, {90, 80, 70, 60, 50}};
```

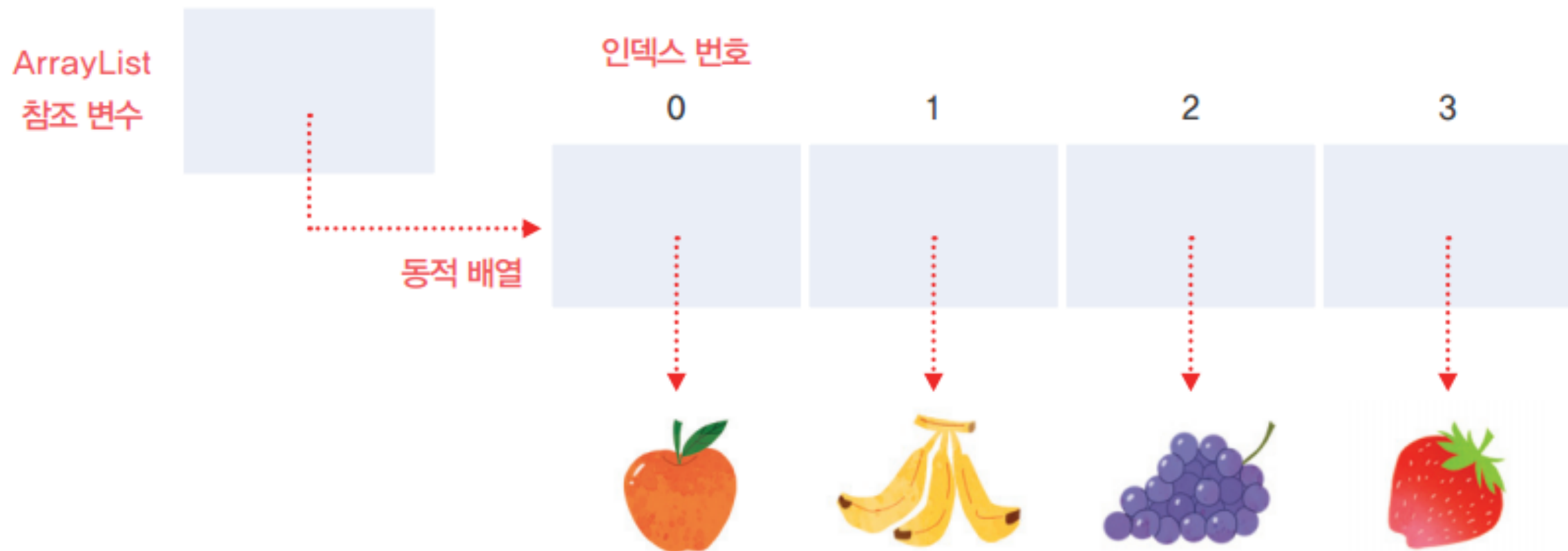
첫 번째 행의 원소이다.

두 번째 행의 원소이다.

세 번째 행의 원소이다.

동적 배열(1)

- ▶ 처리할 데이터의 개수가 고정된 경우가 아니라면 정적 배열은 자원을 낭비하거나 프로그램을 다시 컴파일 필요
- ▶ 자바는 크기가 유동적인 배열을 지원하기 위해 ArrayList 클래스 제공



동적 배열(2)

▶ ArrayList 객체 생성

```
ArrayList<참조타입> 참조변수 = new ArrayList<>();
```

숫자라면 Integer, Long, Short, Float, Double 등을 사용한다.

▶ ArrayList 원소 접근

```
참조변수.add(데이터)
```

```
참조변수.remove(인덱스번호)
```

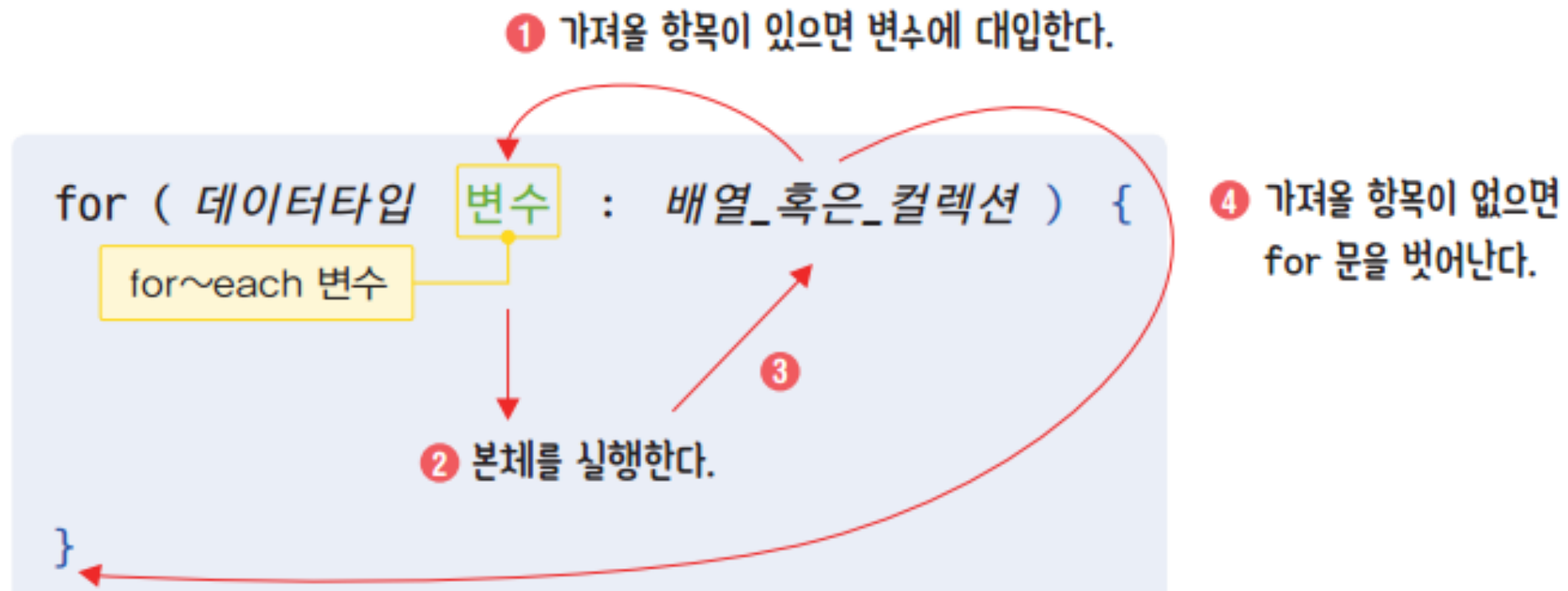
```
참조변수.get(인덱스번호)
```

```
참조변수.size()
```

배열을 위한 반복문

▶ for~each 반복문

▶ JDK 5부터 도입된 것으로 for 문을 개선한 방식

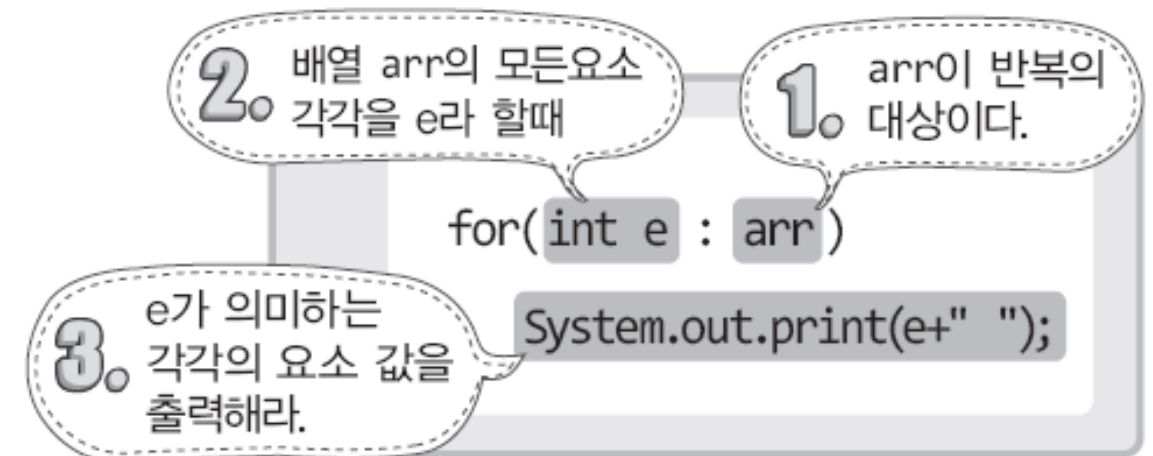


▶ 특정 원소를 나타내기 위한 인덱스를 사용하지 않음

FOR-EACH문의 활용

```
for(int i=0; i<arr.length; i++)  
    System.out.print(arr[i]+" ");
```

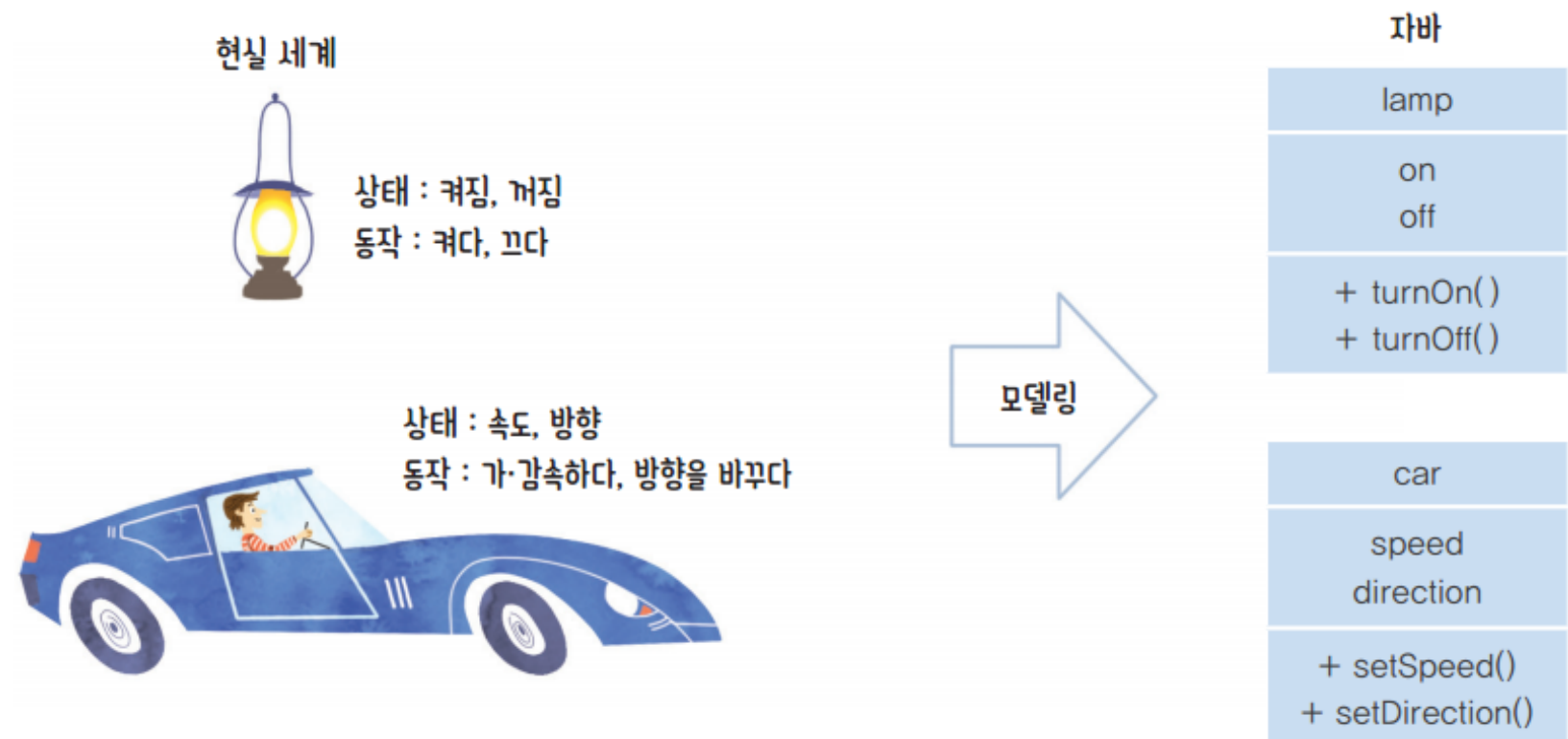
```
for(int e : arr)  
    System.out.print(e+" ");
```



객체 지향

객체의 개념(1)

- ▶ 소프트웨어 객체는 현실 세계의 객체를 필드와 메서드로 모델링한 것
 - ▶ 상태는 필드(Field)로, 동작은 메서드(Method)로 정의
 - ▶ 필드는 객체 내부에 선언된 변수
 - ▶ 메서드는 객체 내부에 정의된 동작



절차 지향과 객체 지향(1)

➤ 절차 지향 프로그래밍

- 일련의 동작을 순서에 맞추어 단계적으로 실행하도록 명령어를 나열
- 데이터를 정의하는 방법보다는 명령어의 순서와 흐름에 중점
- 수행할 작업을 예상할 수 있어 직관적인데, 규모가 작을 때는 프로그래밍과 이해하기가 용이
- 초기 소프트웨어는 계산 위주이므로 절차 지향 프로그래밍이 적합

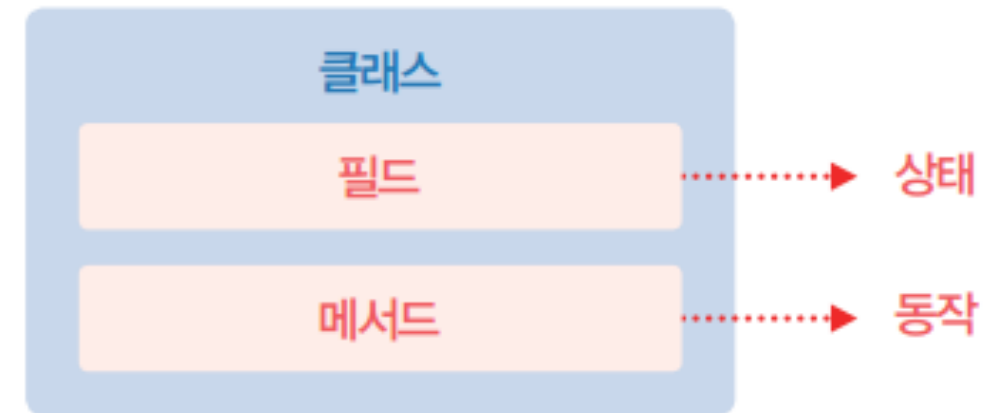
절차 지향과 객체 지향(2)

➤ 객체 지향 프로그래밍

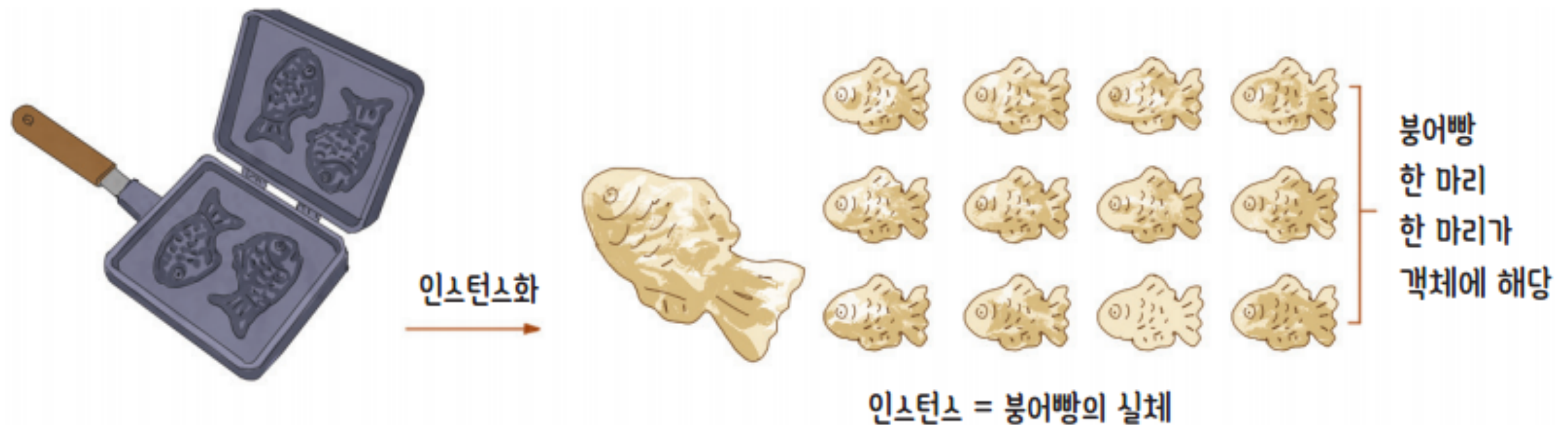
- 소프트웨어의 규모가 커지면서 동작과 분리되어 전 과정에서 서로 복잡하게 얽혀 있는 데이터를 사용했기 때문에 절차 지향 프로그래밍 방식의 한계
- 절차 지향 프로그램은 추후 변경하거나 확장하기도 어려움
- 현실 세계를 객체 단위로 프로그래밍하며, 객체는 필드(데이터)와 메서드(코드)를 하나로 묶어 표현

객체와 클래스

▶ 클래스와 객체의 개념

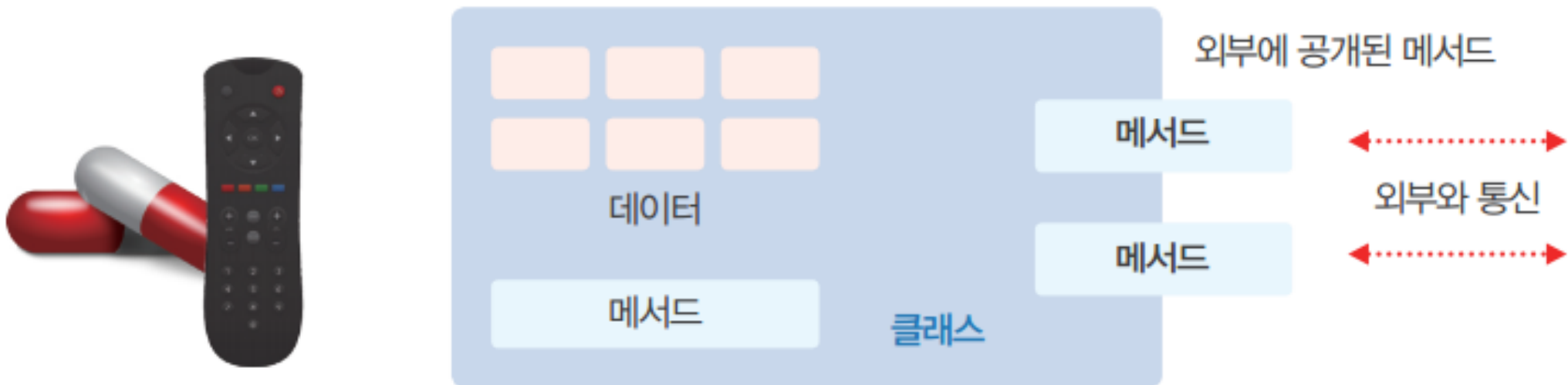


▶ 인스턴스화의 개념



객체 지향 프로그래밍(1)

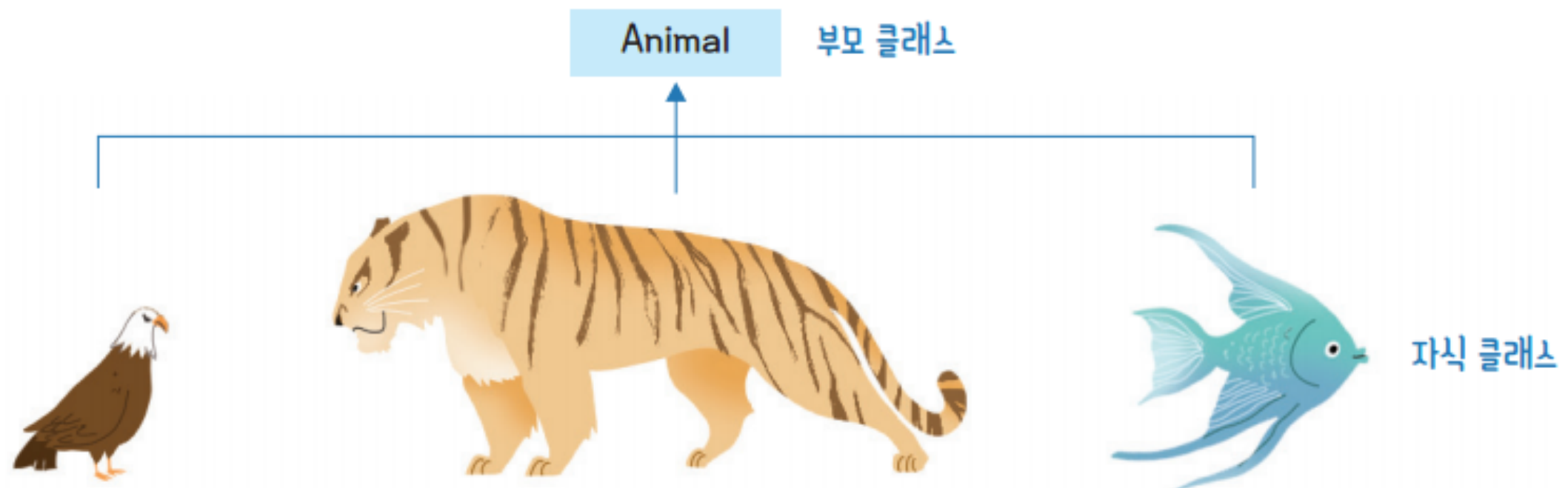
- ▶ 객체 지향 프로그래밍의 주요 특징은 캡슐화, 상속, 다형성
- ▶ 캡슐화(정보 은닉)
 - ▶ 관련된 필드와 메서드를 하나의 캡슐처럼 포장해 세부 내용을 외부에서 알 수 없도록 감추는 것



객체 지향 프로그래밍(2)

➤ 상속

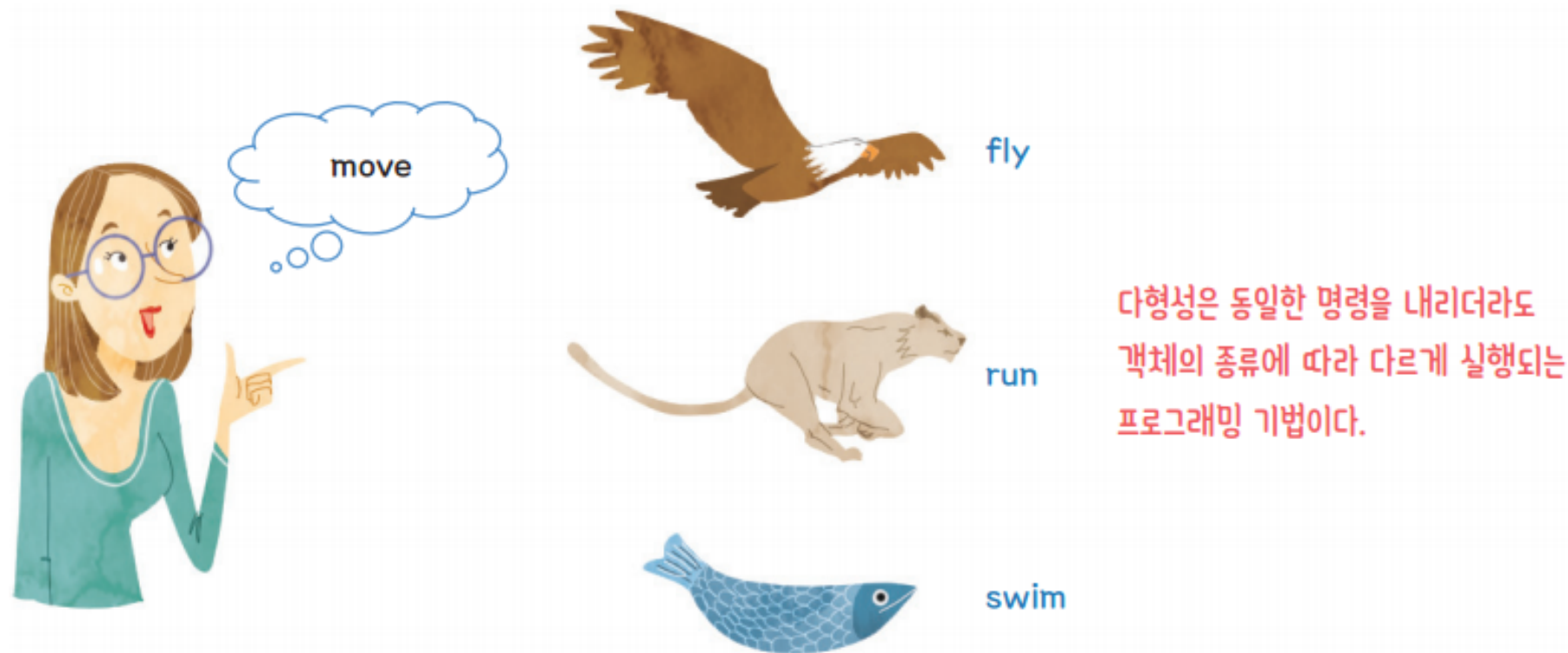
- 자녀가 부모 재산을 상속받아 사용하듯이 상위 객체를 상속받은 하위 객체가 상위 객체의 메서드와 필드를 사용하는 것
- 개발된 객체를 재사용하는 방법 중 하나



객체 지향 프로그래밍(3)

▶ 다형성

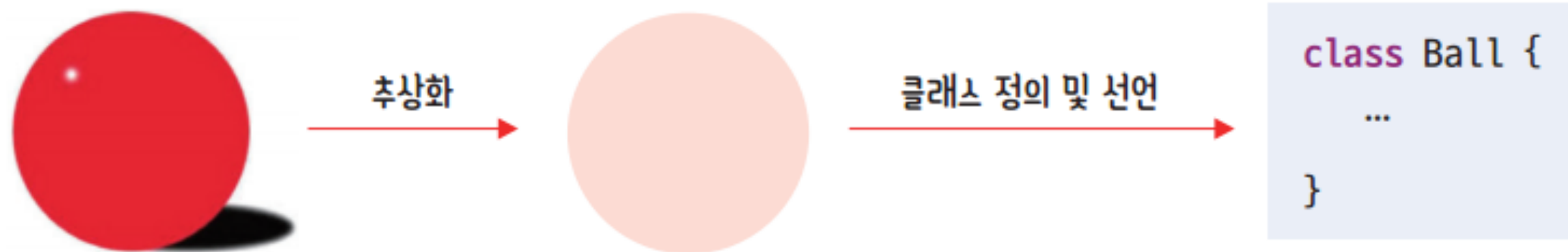
- ▶ 대입되는 객체에 따라서 메서드를 다르게 동작하도록 구현하는 기술
- ▶ 실행 도중 동일한 이름의 다양한 구현체 중에서 메서드를 선택 가능



클래스의 선언(1)

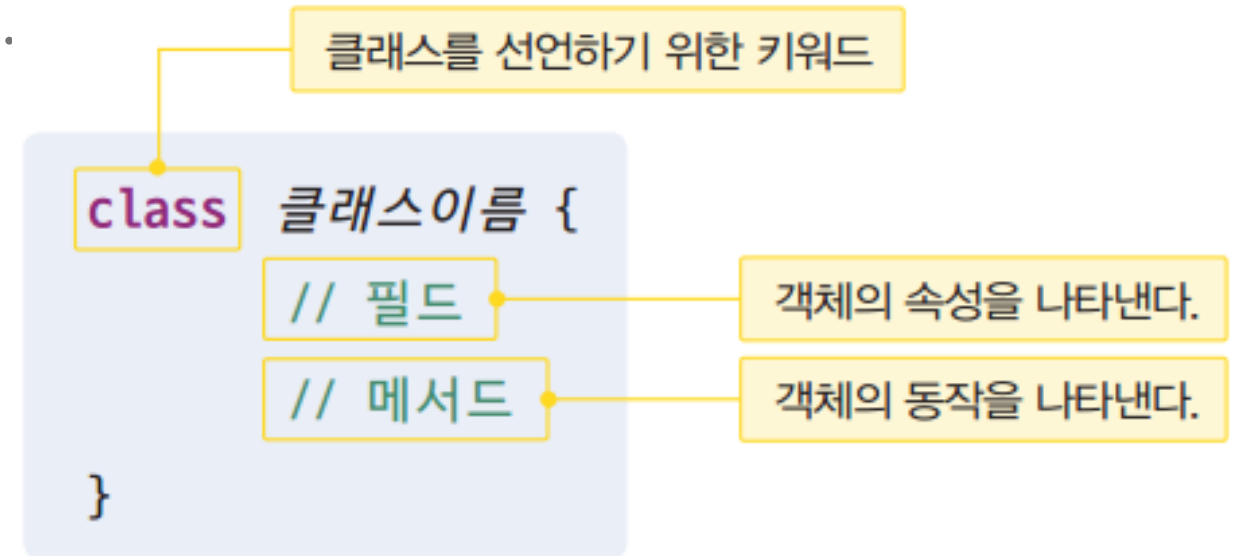
➤ 추상화

- 현실 세계의 객체는 수많은 상태가 있고 다양한 동작을 하지만, 클래스에 모두 포함하기는 어렵기에 추상화(Abstraction)하는 과정이 필요
- 추상화는 현실 세계의 객체에서 불필요한 속성을 제거하고 중요한 정보만 클래스로 표현하는 일종의 모델링 기법
- 따라서 사람마다 추상화하는 기법이 같지 않으므로 각 개발자는 클래스를 다르게 정의 가능

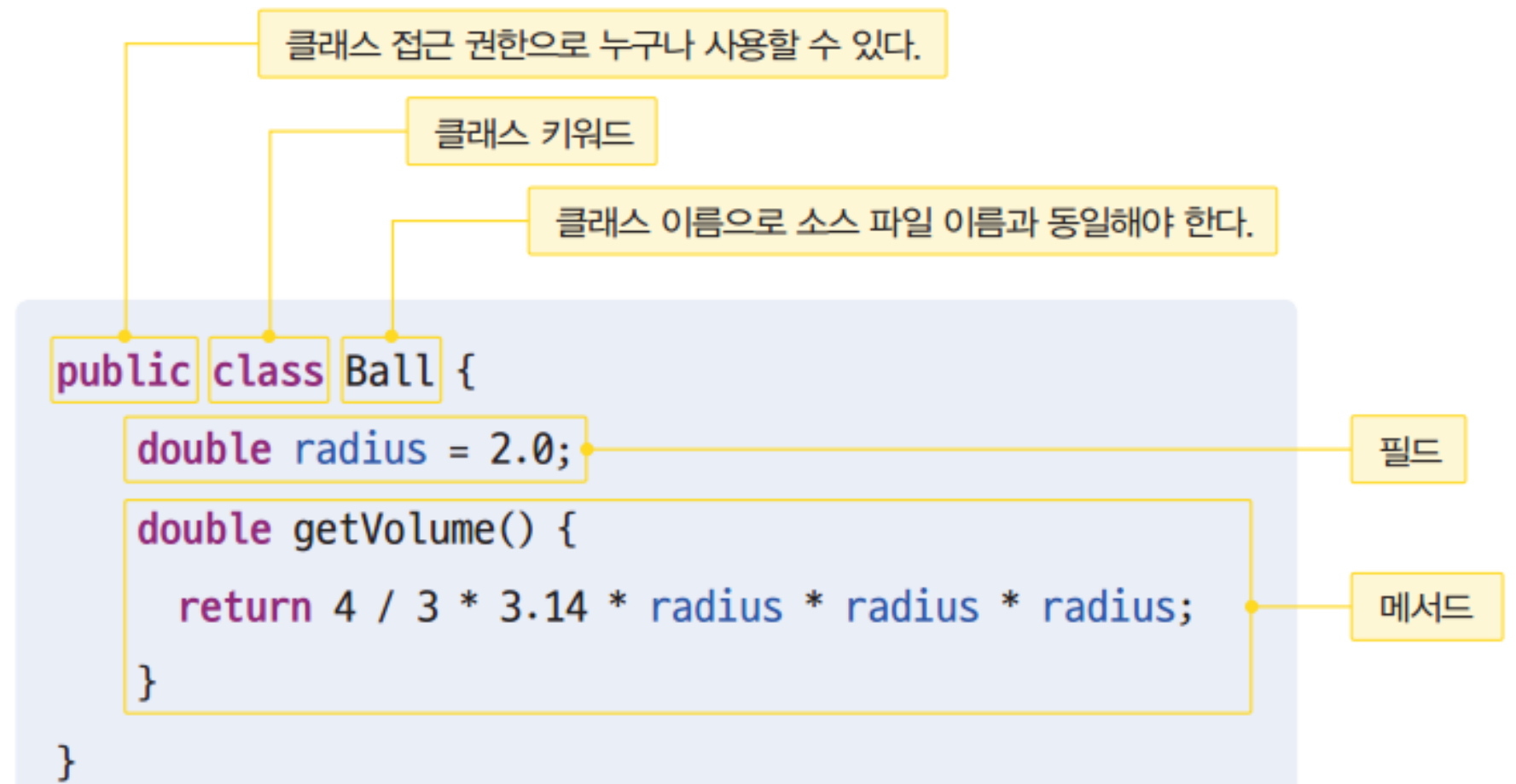


클래스 선언의 선언(2)

▶ 클래스 선언 형식



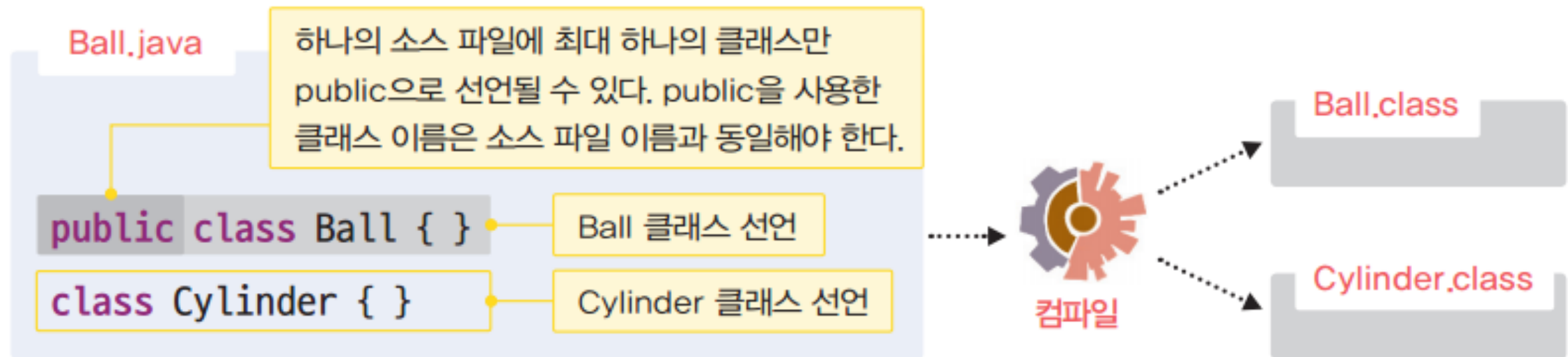
▶ 클래스 선언 예



클래스 선언의 선언(3)

▶ 클래스 선언과 파일

- ▶ 보통 소스 파일마다 하나의 클래스를 선언하지만, 2개 이상의 클래스를 하나의 파일로 선언 가능
- ▶ 하나의 파일에 클래스가 둘 이상 있다면 하나만 public으로 선언할 수 있고, 해당 클래스 이름은 소스 파일 이름과 동일해야 함



생성자의 개념과 선언

▶ 생성자의 역할

- ▶ 객체를 생성하는 시점에서 필드를 다양하게 초기화

▶ 생성자의 선언 방식

- ▶ 생성자 이름은 클래스 이름과 같다.
- ▶ 생성자의 반환 타입은 없다.
- ▶ 생성자는 new 연산자와 함께 사용하며, 객체를 생성할 때 호출한다.
- ▶ 생성자도 오버로딩할 수 있다

클래스이름 (...) { ... }

일반적으로 공개되어야 하므로 public으로 선언되지만 아닐 수도 있다.

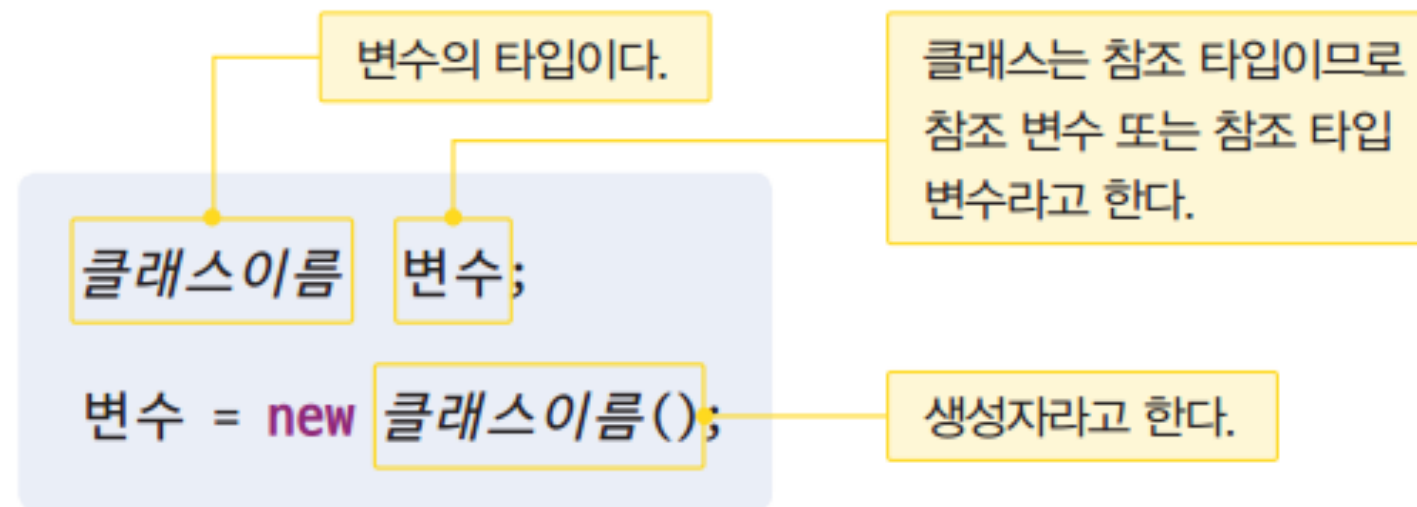
▶ 생성자의 사용

클래스이름 변수 = new 클래스이름(...);

생성자

객체 생성과 참조 변수(1)

▶ 객체 생성 형식



(a) 객체 변수 선언과 생성

```
new 클래스이름();
```

(b) 변수를 생략한 객체 생성

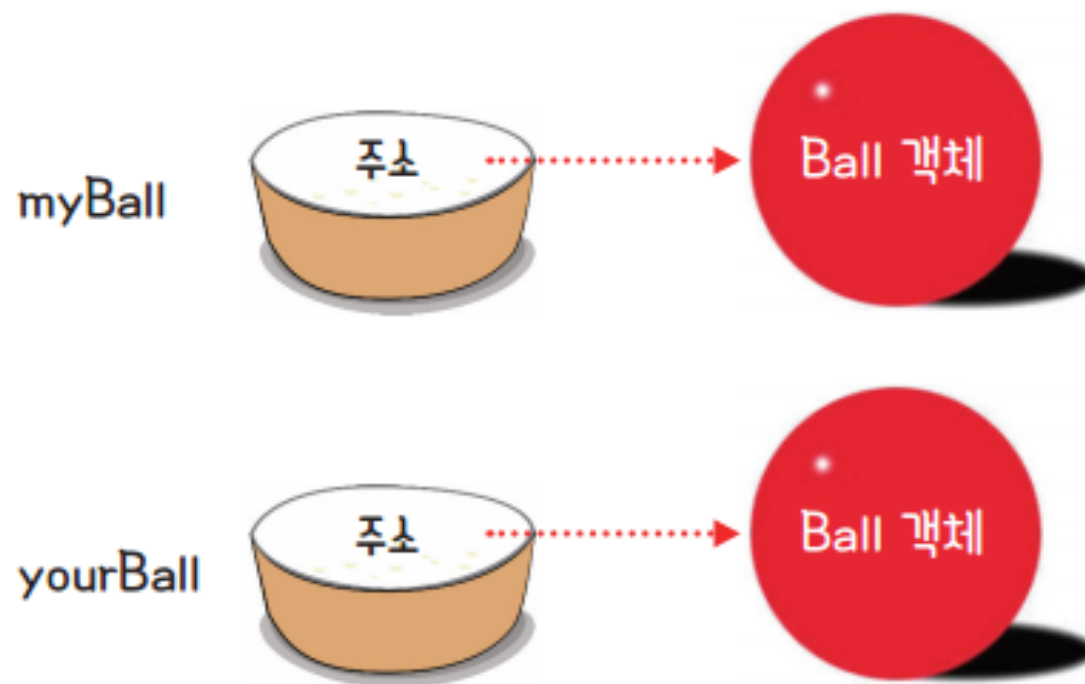
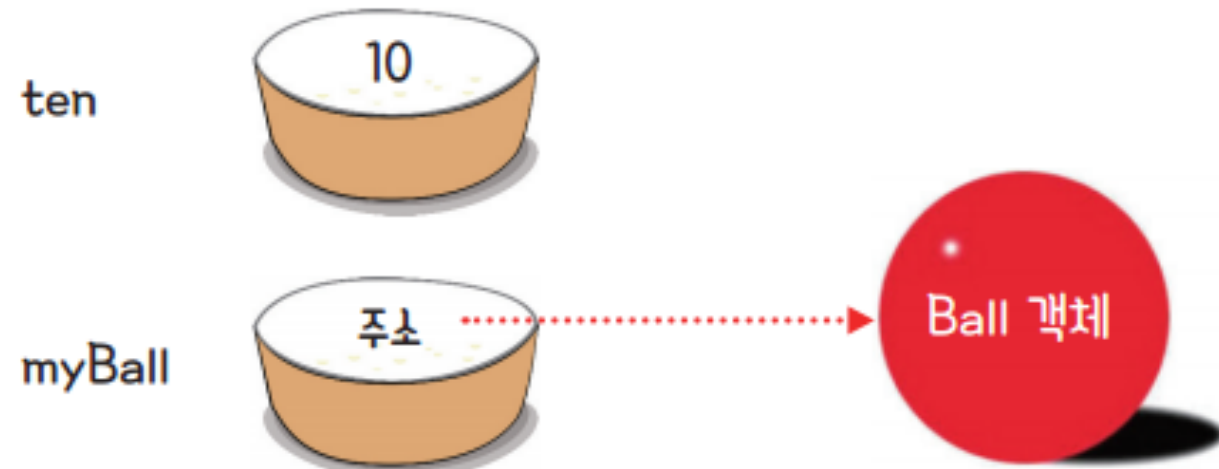
▶ 한 문장으로 변수 선언과 객체 생성 가능

```
클래스이름 변수 = new 클래스이름();
```

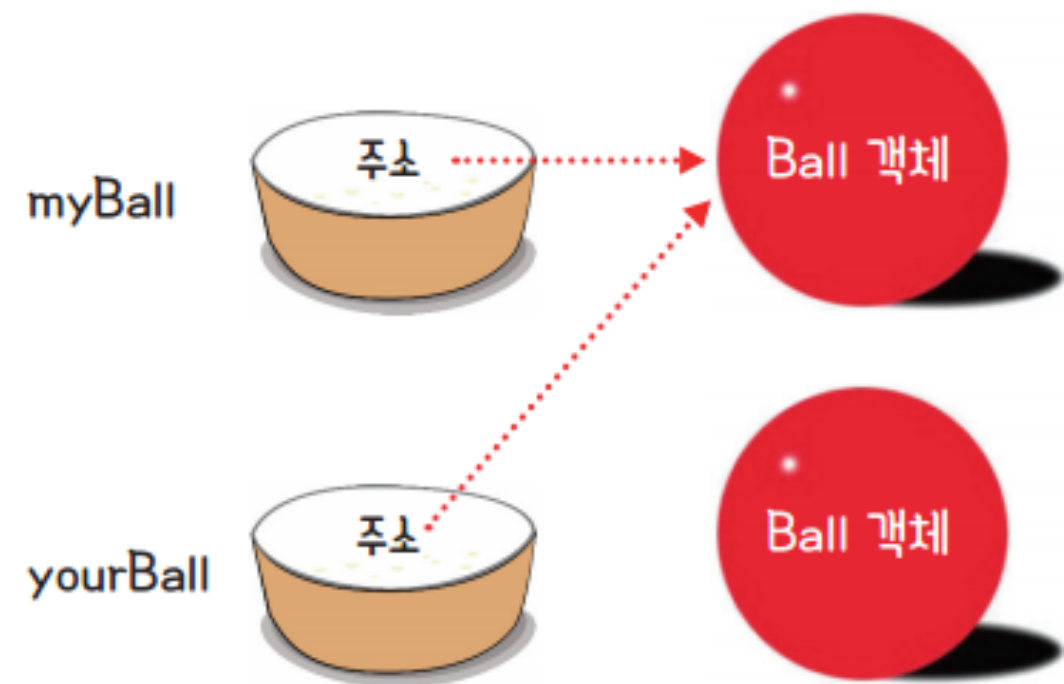
객체 생성과 참조 변수(2)

▶ 기초 타입과 참조 타입

```
int ten = 10;  
Ball myBall = new myBall();
```



(a) myBall = yourBall 연산 전



(b) myBall = yourBall 연산 후

클래스의 구성 요소

- ▶ 멤버 : 필드, 메서드
- ▶ 생성자

MAIN 메소드

- ▶ 메소드 이름은 main
- ▶ 메소드 중괄호 내에 존재하는 문장들이 위에서 아래로 순차적 실행

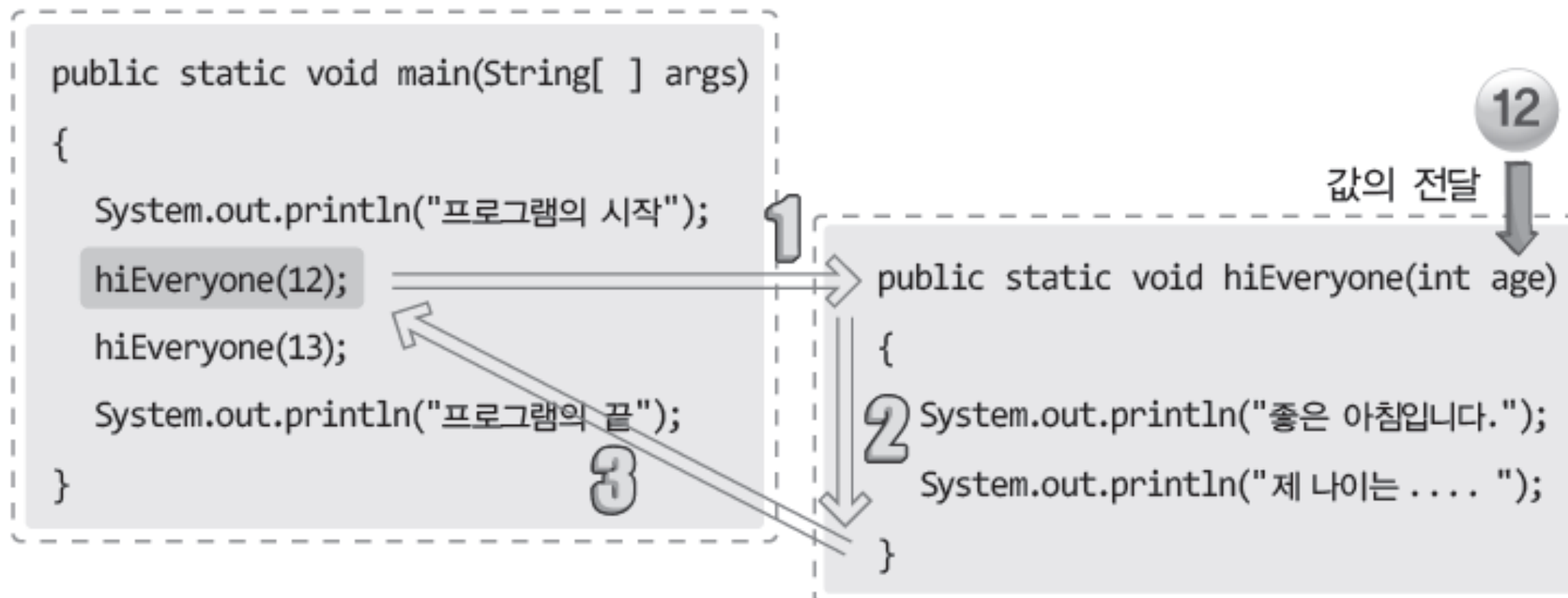
public, static, void
왜? 항상 main인가?
String[] args ?

```
public static void main(String[] args)
{
    int num1=5, num2=7;
    System.out.println("5+7="+num1+num2);
}
```

자바 프로그램의 시작은 main이라는 이름의 메소드를
실행하면서 시작한다!

다른 이름의 메소드 만들기

.....



다른 이름의 메소드 만들기

실행결과

```
class MethodDefAdd
{
    public static void main(String[] args)
    {
        System.out.println("프로그램의 시작");
        hiEveryone(12);
        hiEveryone(13);
        System.out.println("프로그램의 끝");
    }
    public static void hiEveryone(int age)
    {
        System.out.println("좋은 아침입니다.");
        System.out.println("제 나이는 "+ age+"세입니다.");
    }
}
```

메소드 실행(호출) 방법

매개변수

프로그램의 시작
좋은 아침입니다.
제 나이는 12세입니다.
좋은 아침입니다.
제 나이는 13세입니다.
프로그램의 끝

매개변수가 두 개인 형태의 메소드

.....

```
class Method2Param
{
    public static void main(String[] args)
    {
        double myHeight=175.9;
        hiEveryone(12, 12.5);
        hiEveryone(13, myHeight);
        byEveryone();
    }
    public static void hiEveryone(int age, double height)
    {
        System.out.println("제 나이는 "+ age+"세 입니다.");
        System.out.println("저의 키는 "+ height+"cm 입니다.");
    }
    public static void byEveryone()
    {
        System.out.println("다음에 뵙겠습니다.");
    }
}
```

전달 순서대로 저장

전달되는 것 없음

실행결과

제 나이는 12세 입니다.
저의 키는 12.5cm 입니다.
제 나이는 13세 입니다.
저의 키는 175.9cm 입니다.
다음에 뵙겠습니다.

값을 반환하는 메소드

```
class MethodReturns
{
    public static void main(String[] args)
    {
        int result=add(4, 5);
        System.out.println("4와 5의 합 : " + result);
        System.out.println("3.5의 제곱 : " + square(3.5));
    }
    public static int add(int num1, int num2)
    {
        int addResult=num1+num2;
        return addResult;
    }
    public static double square(double num)
    {
        return num*num;
    }
}
```

값을 반환하지 않겠다.

int형 데이터를 반환하겠다.

double형 데이터를 반환하겠다.

int result = add(4, 5) ;



int result = 9 ;

값의 반환이 의미하는 바

실행결과

4와 5의 합 : 9

3.5의 제곱 : 12.25

키워드 RETURN

- ▶ 값의 반환, 메소드의 종료, 두 가지의 의미를 지님

```
class OnlyExitReturn
{
    public static void main(String[] args)
    {
        divide(4, 2);
        divide(6, 2);
        divide(9, 0);
    }
    public static void divide(int num1, int num2)
    {
        if(num2==0)
        {
            System.out.println("0으로는 값을 나눌 수 없습니다.");
            return;
        }
        System.out.println("나눗셈 결과 : " + (num1/num2));
    }
}
```

실행결과

나눗셈 결과 : 2

나눗셈 결과 : 3

0으로는 값을 나눌 수 없습니다.

메소드의 종료만을 의미함

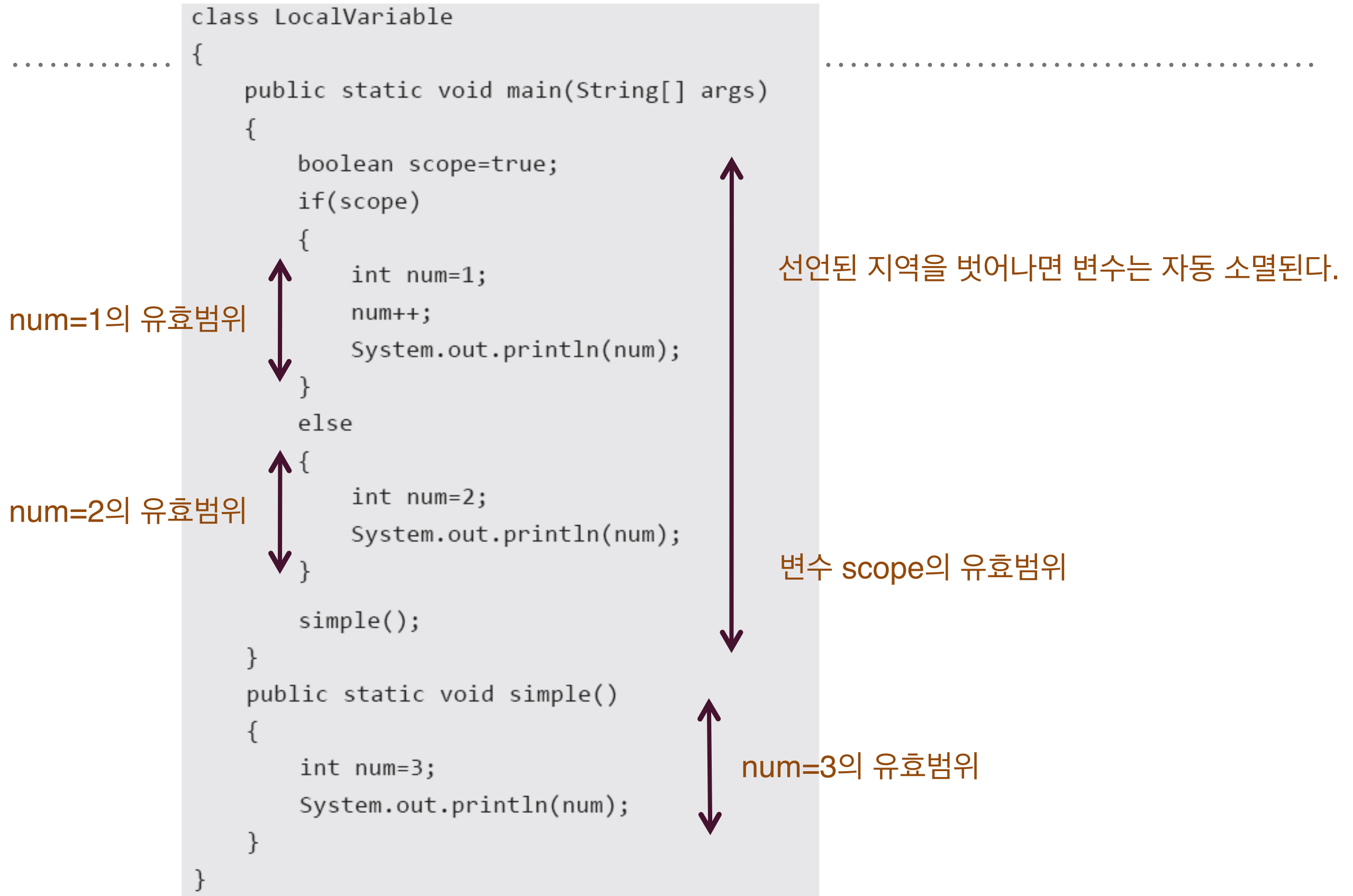
변수의 유효범위

```
for( int num=0 ; num<5; num++)  
{  
    /* 추가적인  
       변수 num 선언 불가 지역 */  
}
```

변수 num의
접근 가능지역

```
public static void myFunc( int num )  
{  
    /* 추가적인  
       변수 num 선언 불가 지역 */  
}
```

변수 num의
접근 가능지역



클래스 내부와 외부에서 멤버 접근

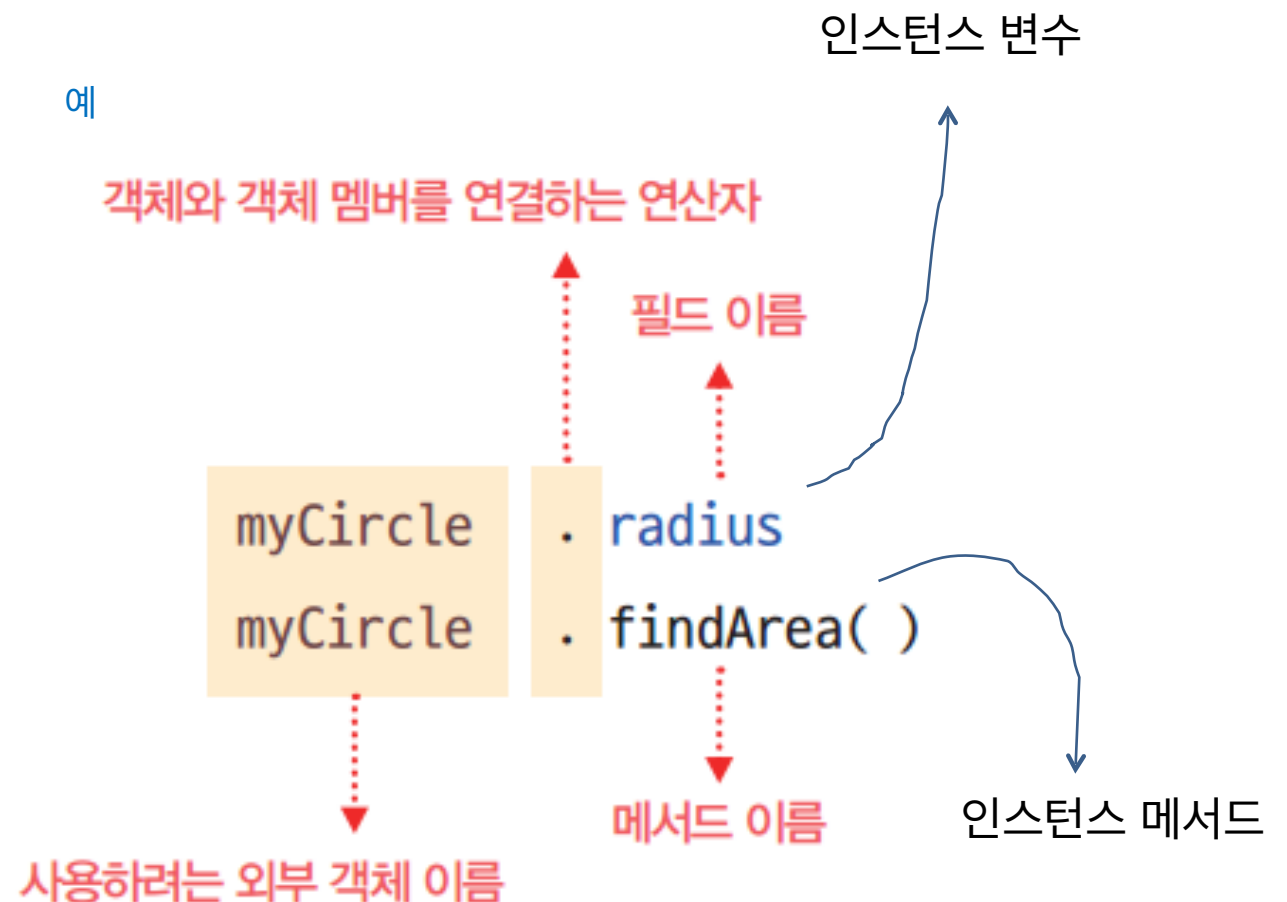
▶ 클래스 내부에서 멤버 접근

예

```
radius          // 필드 이름  
findArea()      // 메서드 이름
```

▶ 클래스 외부에서 멤버 접근

객체.필드
객체.메서드



접근자와 설정자

➤ 필요성

- 클래스 내부에 캡슐화된 멤버를 외부에서 사용할 필요

➤ 접근자와 설정자

- private으로 지정된 필드에 값을 반환하는 접근자와 값을 변경하는 설정자는 공개된 메서드
- 일반적으로 접근자는 get, 설정자는 set으로 시작하는 이름을 사용
- 필드 이름을 외부와 차단해서 독립시키기 때문에 필드 이름 변경이나 데이터 검증도 가능

PRIVATE과 PUBLIC

- ▶ private- 클래스 내부(메소드)에서만 접근 가능.
- ▶ public - 어디서든 접근 가능(접근을 제한하지 않는다).

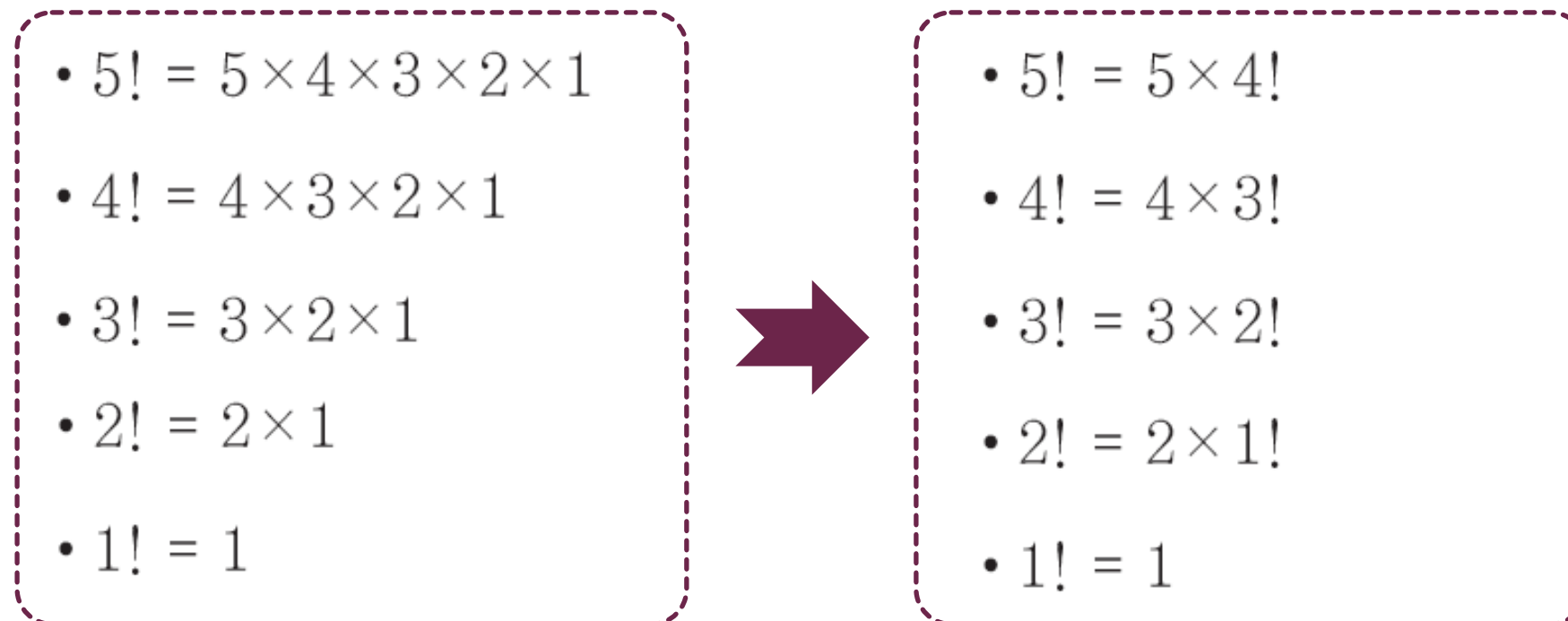
```
class AAA
{
    private int num;
    public void setNum(int n) { num=n; }
    public int getNum() { return num; }
    . . . .
}
```

setNum, getNum은 public이므로 호출 가능!

```
class BBB
{
    public accessAAA(AAA inst)
    {
        num은 private 멤버이므로 컴파일 불가!
        inst.num=20;
        inst.setNum(20);
        System.out.println(inst.getNum());
    }
    . . . .
}
```

수학적 측면에서의 재귀적(순환적) 사고

- ▶ 수학에서의 재귀가 허용되듯 자바는 재귀적 메소드 호출을 허용



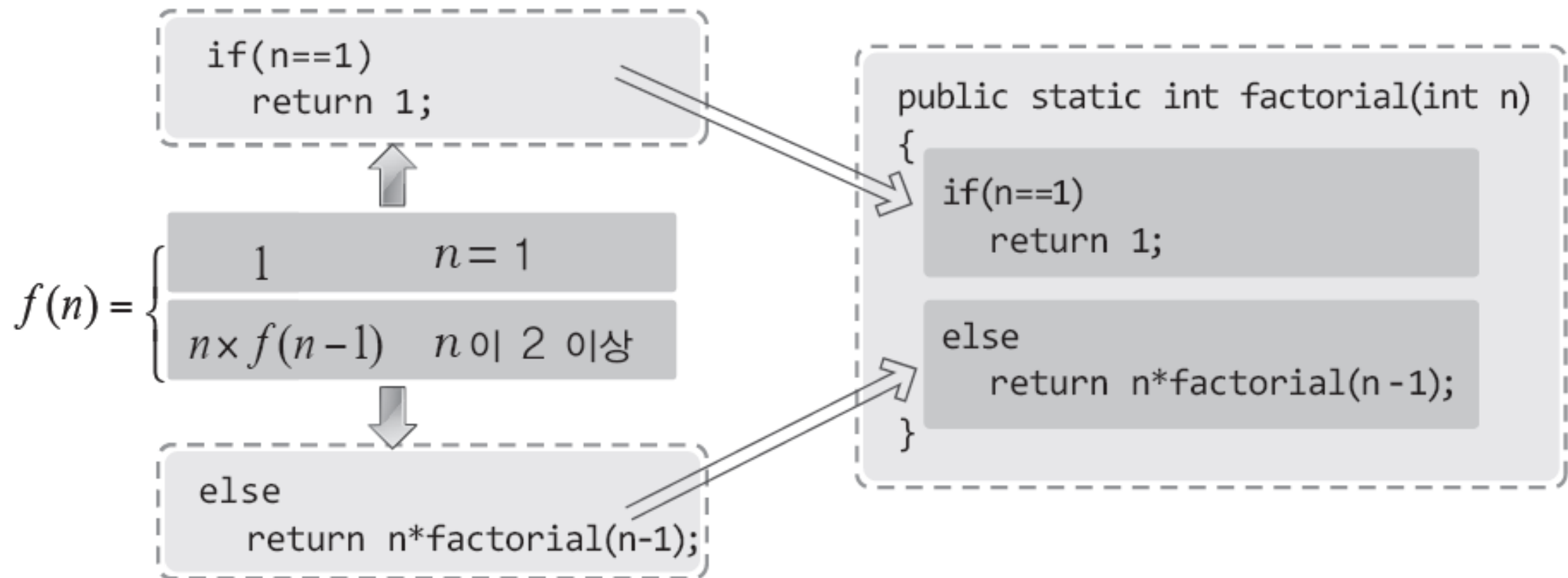
함수 정의

$$f(n) = \begin{cases} n \times f(n-1) & n \text{ 이 } 2 \text{ 이상} \\ 1 & n = 1 \end{cases}$$

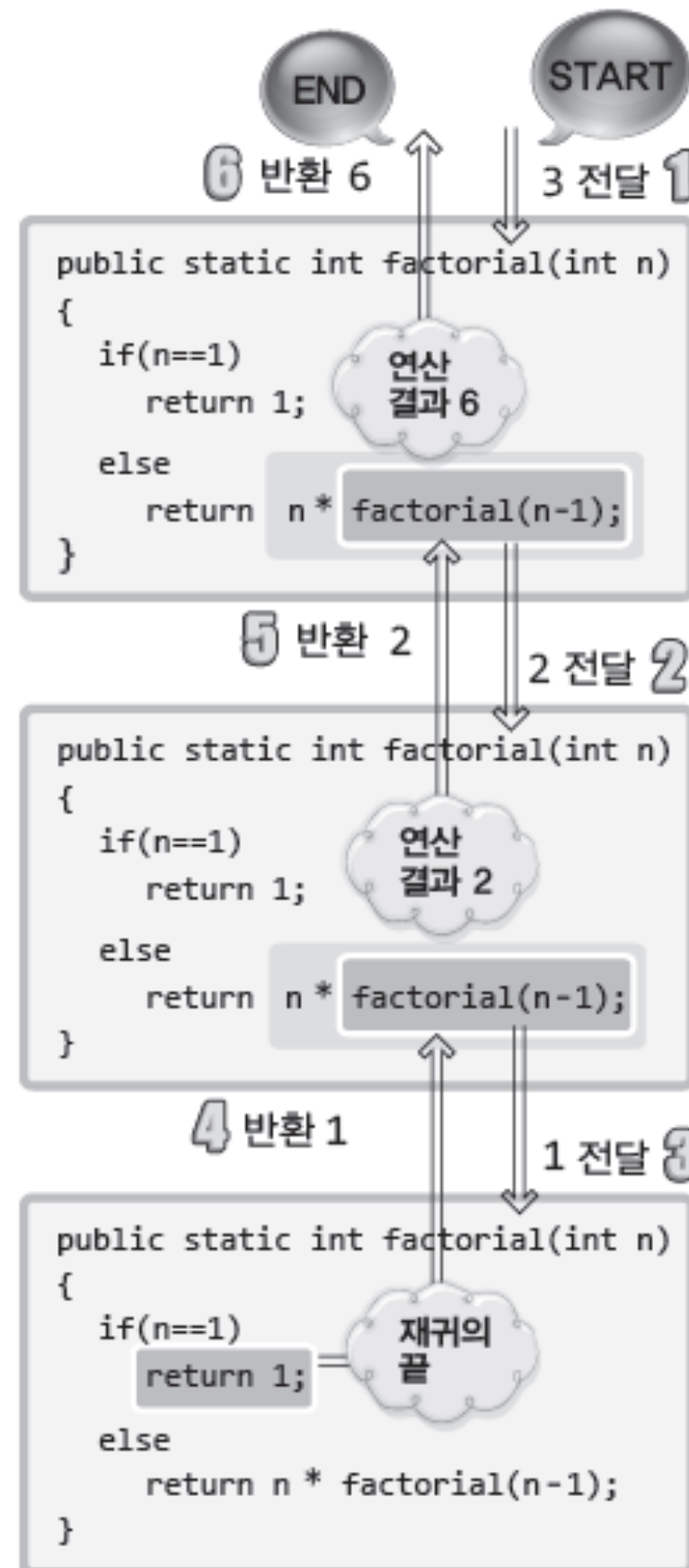
함수 f의 실행문

팩토리얼의 재귀적 메소드 정의

.....



재귀적 메소드



재귀적 메소드

```
class ReculFactorial
{
    public static void main(String[] args)
    {
        System.out.println("3 factorial : " + factorial(3));
        System.out.println("12 factorial : " + factorial(12));
    }
    public static int factorial(int n)
    {
        if(n==1)
            return 1;
        else
            return n*factorial(n-1);
    }
}
```

실행결과

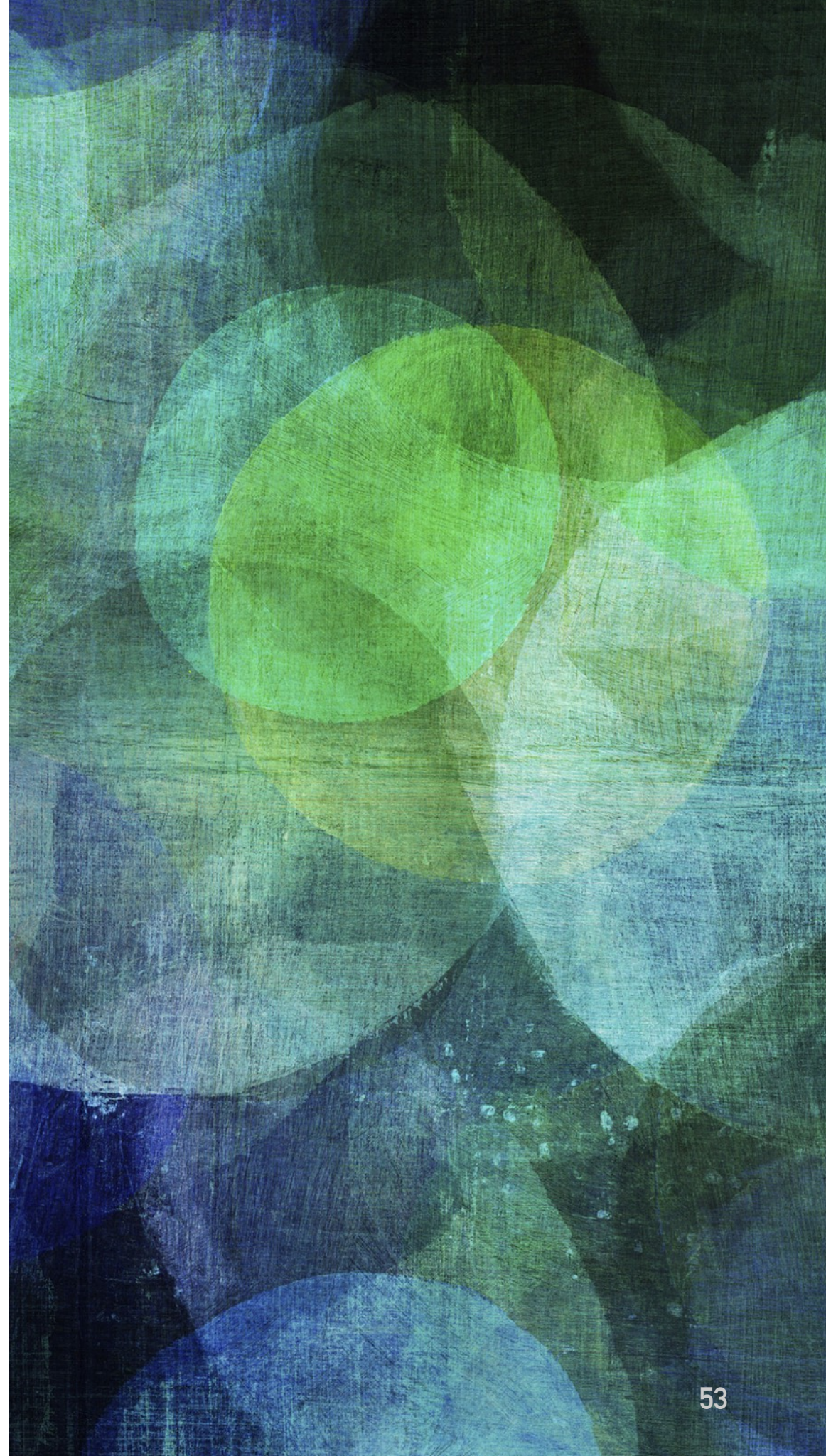
```
3 factorial : 6
12 factorial : 479001600
```

잘못된 재귀 메소드의 정의

```
class InfRecul
{
    public static void main(String[] args)
    {
        showHi(3);
    }
    public static void showHi(int cnt)
    {
        System.out.println("Hi~ ");
        showHi(cnt--);
        if(cnt==1)
            return;
    }
}
```


실습

.....

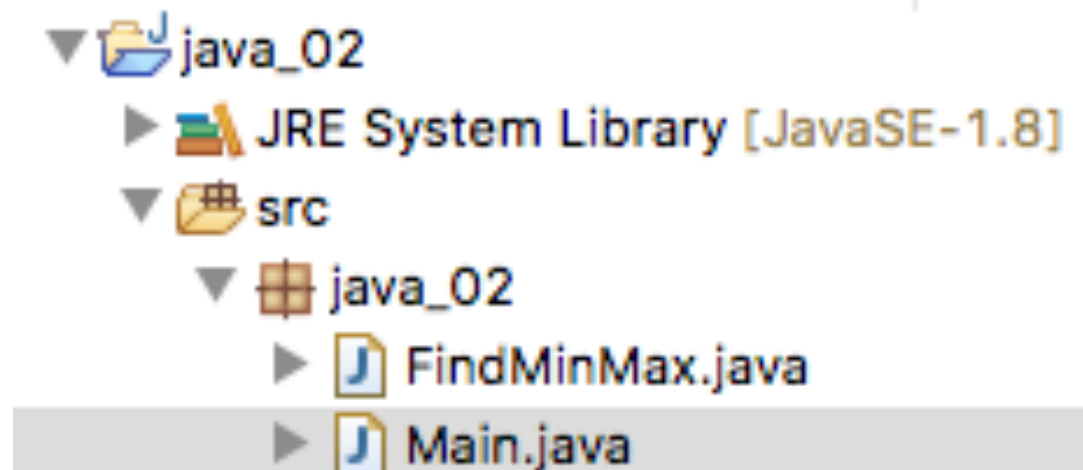


재귀적 문제 풀이

- ▶ 어떤 복잡한 문제들은 재귀 (recursion)을 사용하여 아주 간결하게 표현할 수 있다.
- ▶ 그러한 문제들은 그 자체가 재귀적(recursive)으로 정의된다.
- ▶ 원래 문제보다 크기가 작은 같은 형태의 문제의 답을 안다
 - ▶ 원래 문제의 답을 알 수 있다.
 - ▶ 문제를 재귀적으로 풀 수 있다!!

실습 1

- ▶ 배열에 저장된 데이터 중 가장 큰 값과 작은 값을 구하는 프로그램을 작성
- ▶ 클래스 구조는 다음과 같다.
 - ▶ Main - main 함수가 들어있는 클래스
 - ▶ main이 있는 클래스와 실제 프로그램이 동작하는 클래스는 분할 되는 것이 좋음
 - ▶ FindMinMax - 최소, 최대값을 찾는 클래스



실습 1

▶ Main 클래스

```
public class Main {  
    public static void main(String[] args) {  
        int arrayMaxNum = 10;  
        FindMinMax findMinMax = new FindMinMax(arrayMaxNum);  
        int max = findMinMax.findMax();  
        System.out.println("배열의 최대 값은 : " + max);  
        int min = findMinMax.findMin();  
        System.out.println("배열의 최소 값은 : " + min);  
    }  
}
```


실습 1-1

▶ FindMinMax 클래스

```
import java.util.Random;

public class FindMinMax {
    private int _maxArraySize;
    private int [] _intArray = null;

    public FindMinMax(int maxSize) {}
    private void makeArray() {}
    public int findMin() {}
    public int findMax() {}
}
```

실습 1-1

- `public FindMinMax(int maxSize)`
 - 생성자
 - 변수 `_maxArraySize`를 초기화
 - `makeArray`메소드를 호출

실습 1-1

- private void makeArray()
 - 검색 할 배열을 만드는 메소드
 - 검색 할 배열 객체화 (new)
 - 검색 할 배열에 값 저장
 - 배열은 내용이 랜덤하게 저장되도록 Random 클래스를 이용
 - <https://docs.oracle.com/javase/8/docs/api/java/util/Random.html>

```
Random random = new Random();  
this._intArray[i] = random.nextInt(this._maxArraySize);
```

실습 1-1

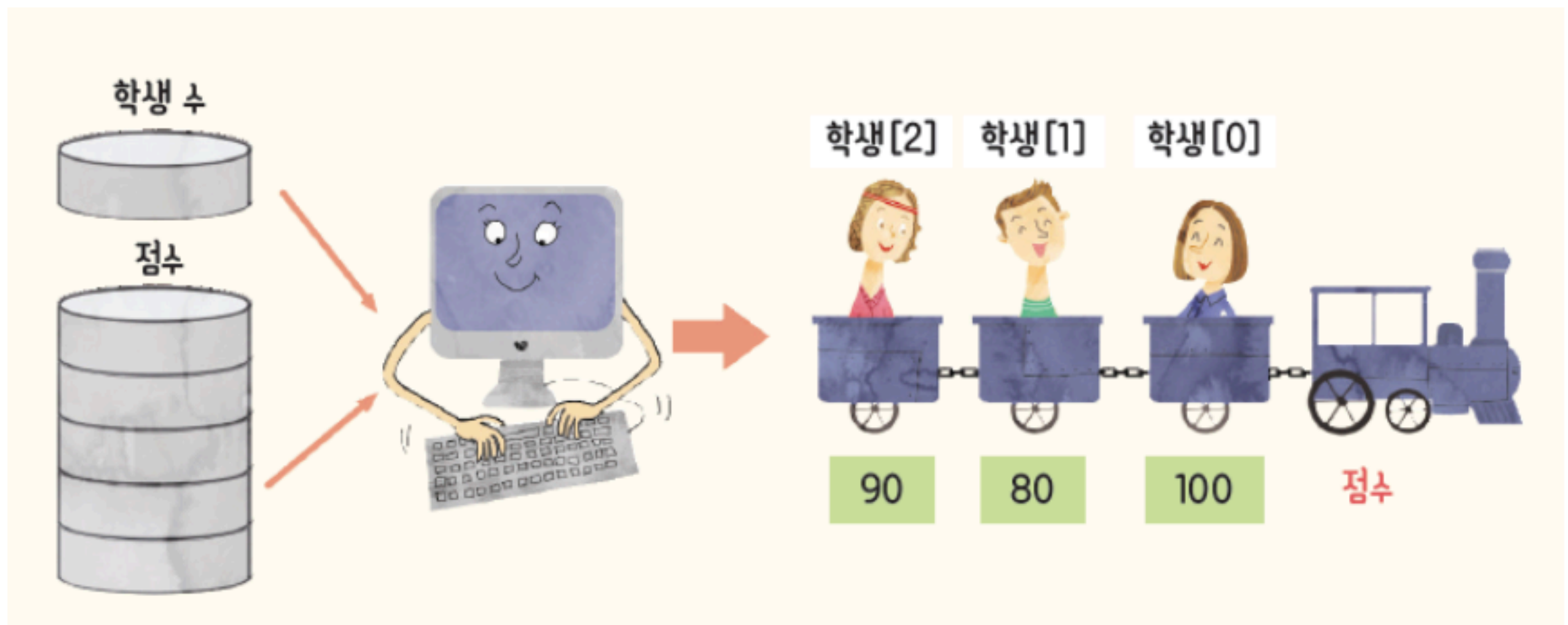
- `public int findMin()`
- `public int findMax()`
 - 배열에서 최소, 최대 값을 찾아 반환해주는 메소드

실습 1-2

- ▶ 실습 1에서 작성한 findMin/findMax 메소드를 재귀함수로 변경
 - ▶ 재귀 함수로 변경을 위해 매개 변수 변경이 필요함
 - ▶ 매개 변수의 개수는 재귀 형식에 따라 다름

실습 2-1

- ▶ 학생수와 점수를 입력하면 각 학생의 성적을 출력하는 프로그램 작성
- ▶ 학생수와 점수는 문자열로 입력 받음
- ▶ 학생의 점수는 정수형 배열에 저장



실습 2-2

▶ 실습 2-1을 입력 받은 학생의 등급을 출력하는 프로그램으로 변경

▶ for-each문을 사용

```
int[] scores;  
for (int score : scores) {  
    System.out.print(score + " ");  
}
```

