# 클래스와 객체

컴퓨터공학전공
박요한

# 절차 지향 프로그래밍

- 절차 지향 프로그래밍
  - ✓ procedural programming
  - ✓ 작업 순서 표현
  - ✓ 작업을 함수로 작성한 함수들의 집합

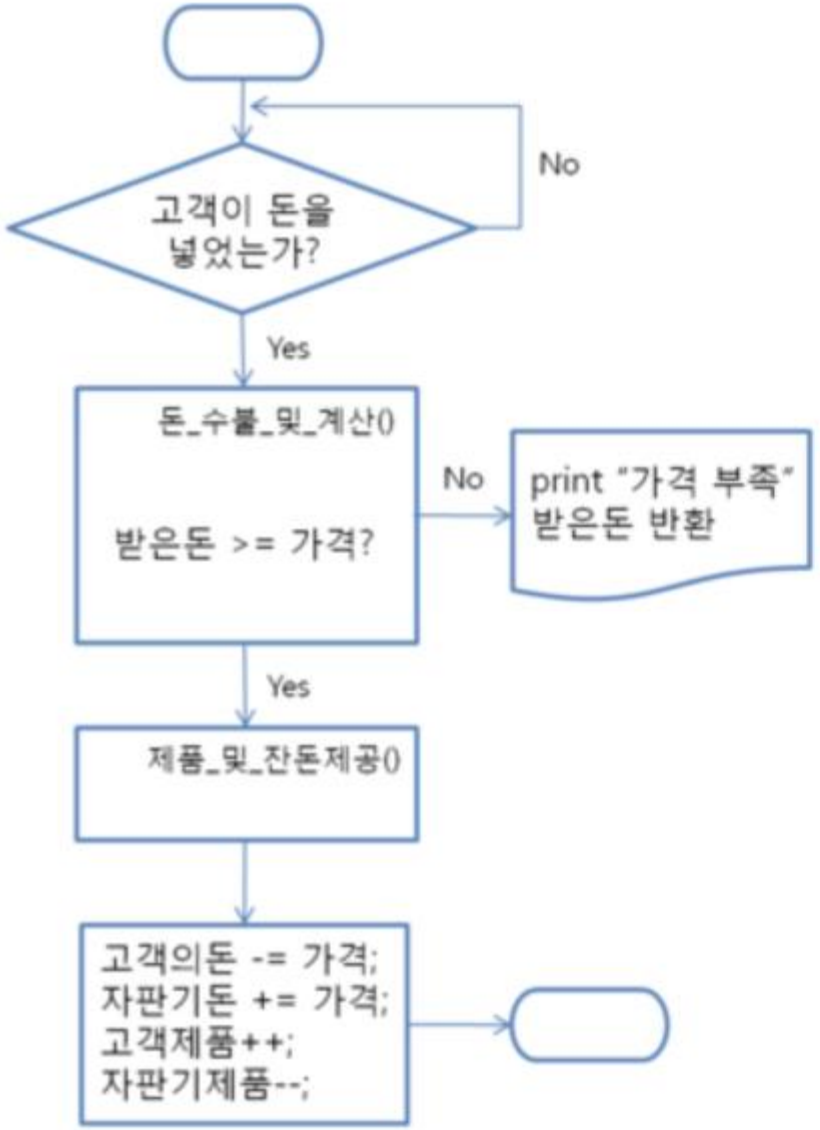
```java
int sum = 0;
int result = 0;

for (int i = 0; i <= 10; i++) {
    sum = sum + i;
}
System.out.println("1부터 10까지의 정수의 합은 " + sum + "입니다.");

result = sum % 2;
if(result == 0){
    System.out.println("짝수 입니다.");
} else {
    System.out.println("홀수 입니다.");
}
```

```java
public int sum(int i, int j) {
    // TODO Auto-generated method stub

    int r = 0;

    for (int h = i; h <= j; h++) {
        r = r + h;
    }

    return r;
}


public String evenOdd(int sum) {
    // TODO Auto-generated method stub
    String r = new String();

    if((sum % 2) == 0){
        r = "짝수 입니다.";
    } else {
        r = "홀수 입니다.";
    }

    return r;
}
```

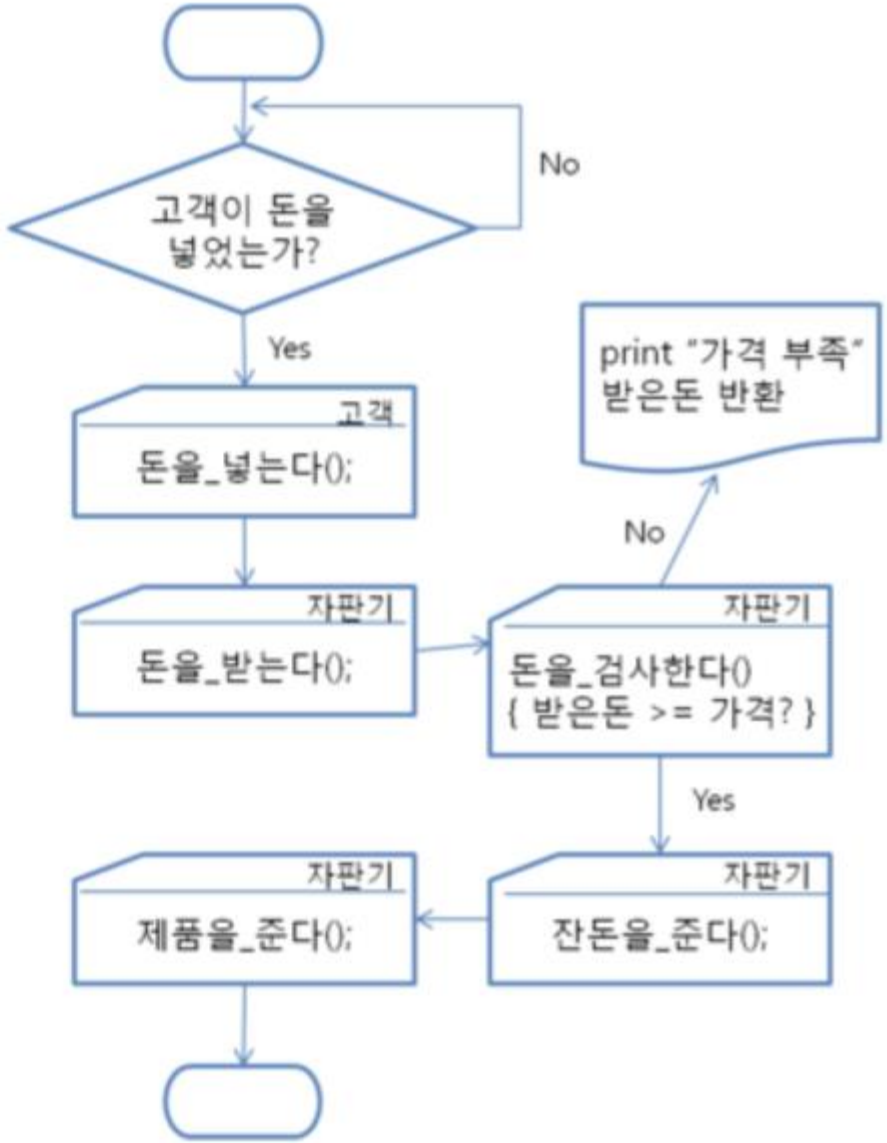
# 객체 지향 프로그래밍

- 객체 지향 프로그래밍
  - ✓OOP: Object Oriented Programming
  - ✓부품 객체를 먼저 만들고 이것들을 하나씩 조립해 완성된 프로그램을 만드는 기법
  - ✓클래스 혹은 객체들의 집합으로 프로그램 작성

## 절차지향 방식




## 객체지향 방식



# 객체 지향 언어의 목적

1. 소프트웨어의 생산성 향상
   - ✓ 컴퓨터 산업 발전에 따라 소프트웨어의 생명 주기(life cycle) 단축
     - 소프트웨어를 빠른 속도로 생산할 필요성 증대
   - ✓ 객체 지향 언어
     - 상속, 다형성, 객체, 캡슐화 등 소프트웨어 재사용을 위한 여러 장치 내장
     - 소프트웨어 재사용과 부분 수정 빠름
     - 소프트웨어를 다시 만드는 부담 대폭 줄임
     - 소프트웨어 생산성 향상

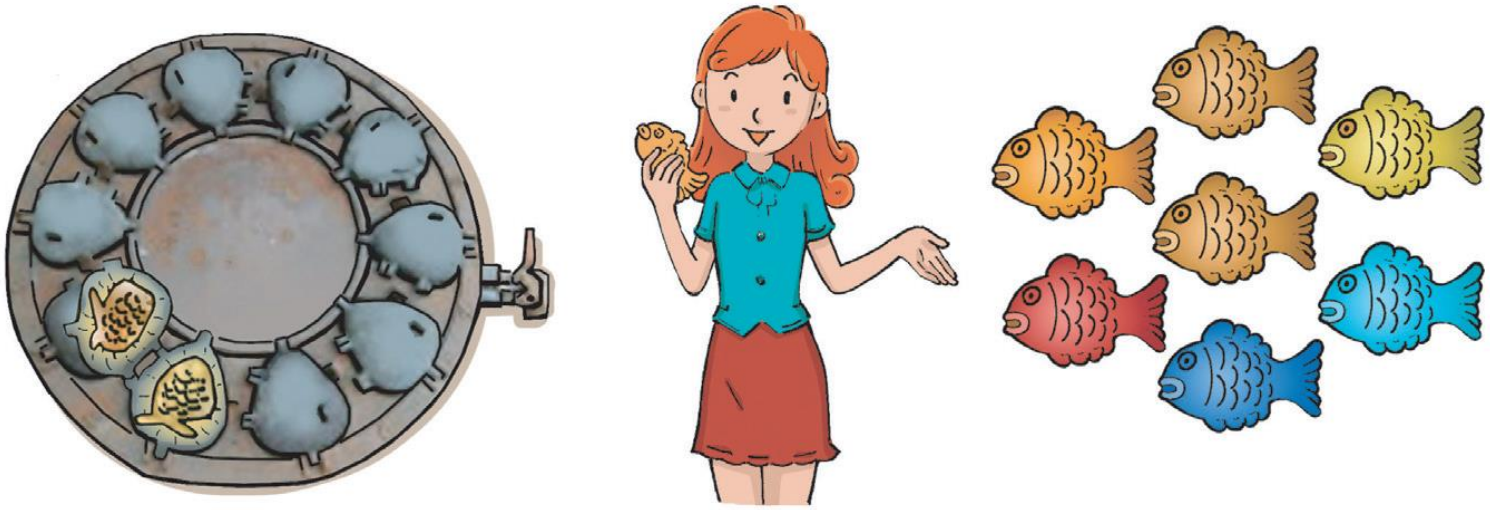
2. 실세계에 대한 쉬운 모델링
   - ✓ 컴퓨터 초기 시대의 프로그래밍
     - 수학 계산/통계 처리를 하는 등 처리 과정, 계산 절차 중요
   - ✓ 현대의 프로그래밍
     - 컴퓨터가 산업 전반에 활용
     - 실세계에서 발생하는 일을 프로그래밍
     - 실세계에서는 절차나 과정보다 물체(객체)들의 상호 작용으로 묘사하는 것이 용이
   - ✓ 객체 지향 언어
     - 실세계의 일을 보다 쉽게 프로그래밍하기 위한 객체 중심적 언어

# 클래스와 객체

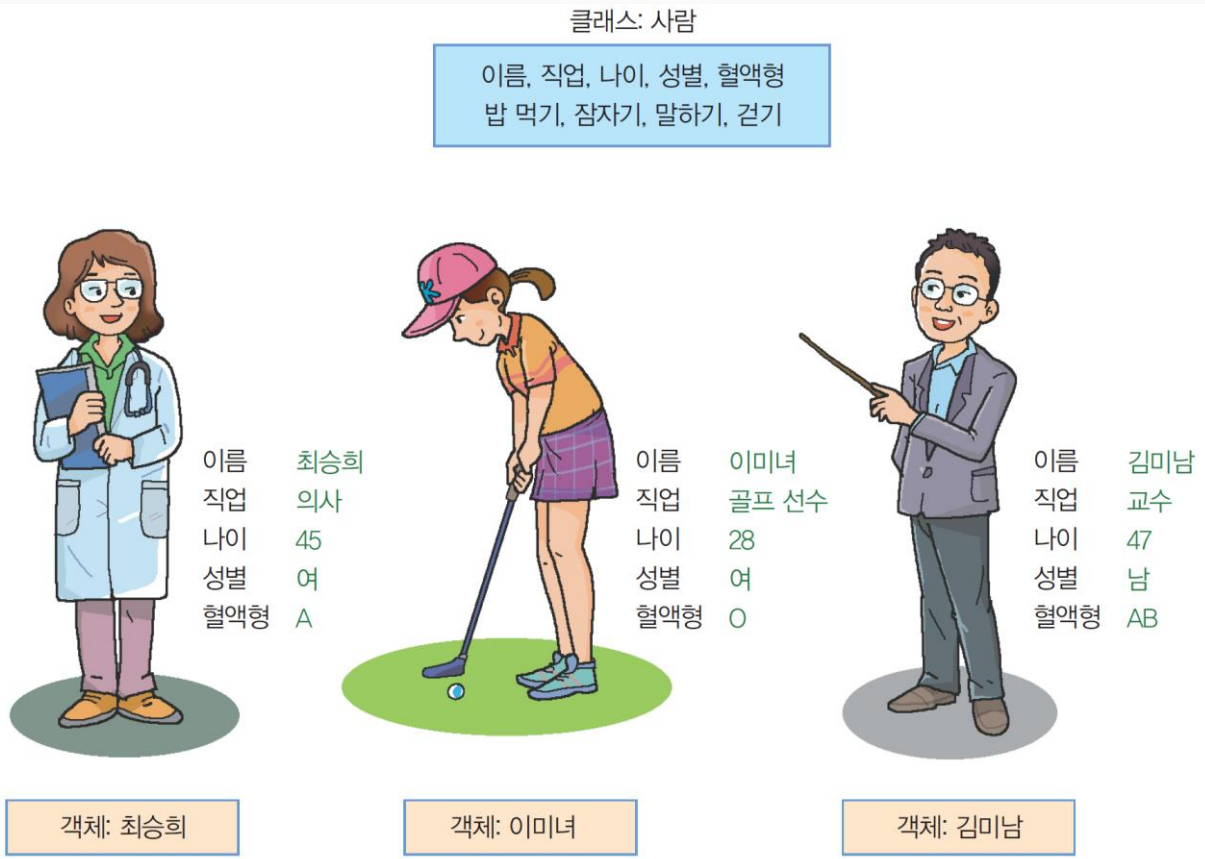
- 클래스
  - ✓객체를 만들어내기 위한 설계도 혹은 **틀**
  - ✓객체의 **속성(state)**과 **행동(behavior)** 포함
- 객체
  - ✓클래스의 모양 그대로 찍어낸 실체
    - 프로그램 실행 중에 생성되는 실체
    - 메모리 공간을 갖는 구체적인 실체
    - 인스턴스(instance)라고도 부름
- 사례
  - ✓클래스: 자동차, 객체: 그랜저, 소나타, K5
  - ✓클래스: 소나타자동차, 객체: 회색 소나타, 흰색 소나타
  - ✓클래스: 사람, 객체: 나, 너, 윗집사람, 아랫집사람
  - ✓클래스: 붕어빵틀, 객체: 구워낸 붕어빵들

# 클래스와 객체와의 관계

붕어빵 틀은 클래스이며, 이 틀의 형태로 구워진 붕어빵은 바로 객체입니다. 붕어빵은 틀의 모양대로 만들어지지만 서로 조금씩 다릅니다.
치즈붕어빵, 크림붕어빵, 앙코붕어빵 등이 있습니다.
그래도 이들은 모두 붕어빵입니다.

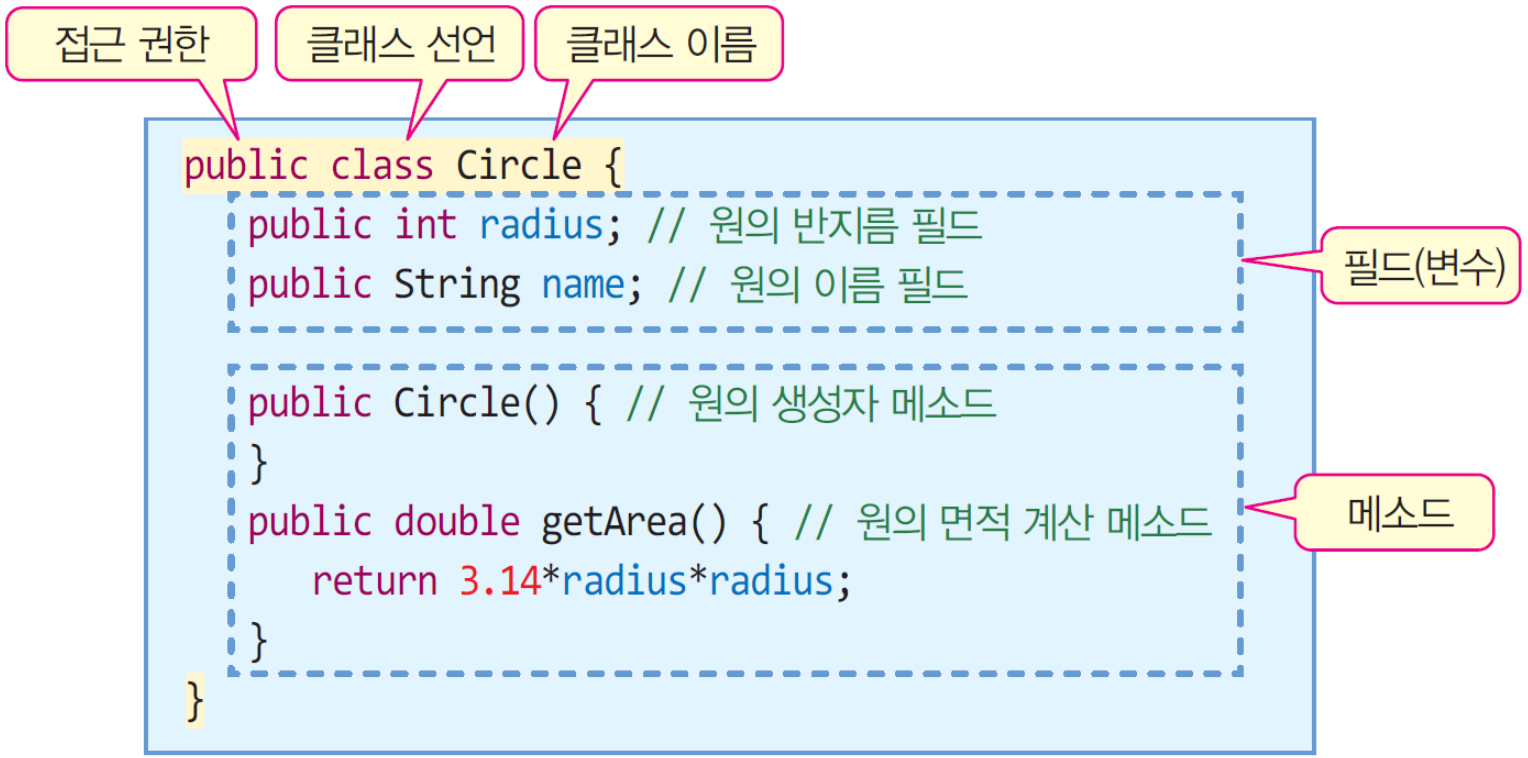


# 사람을 사례로 든 클래스와 객체 사례




* 객체들은 클래스에 선언된 **동일한 속성**을 가지지만, 객체마다 **서로 다른 고유한 값**으로 구분됨

# 클래스 구성

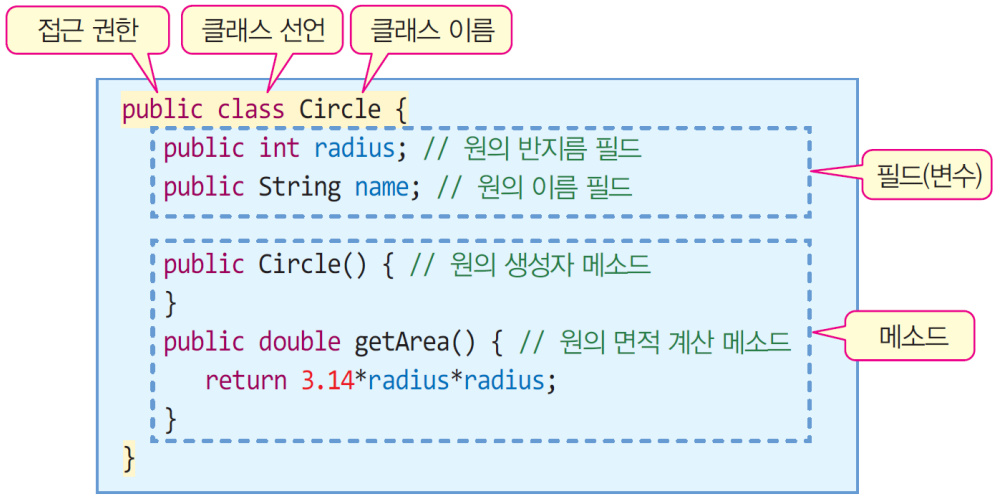

접근 권한
클래스 선언
클래스 이름
public class Circle {
public int radius; // 원의 반지름 필드
public String name; // 원의 이름 필드
public Circle() { // 원의 생성자 메소드
}
public double getArea() { // 원의 면적 계산 메소드
return 3.14*radius*radius;
}
}
필드(변수)
메소드

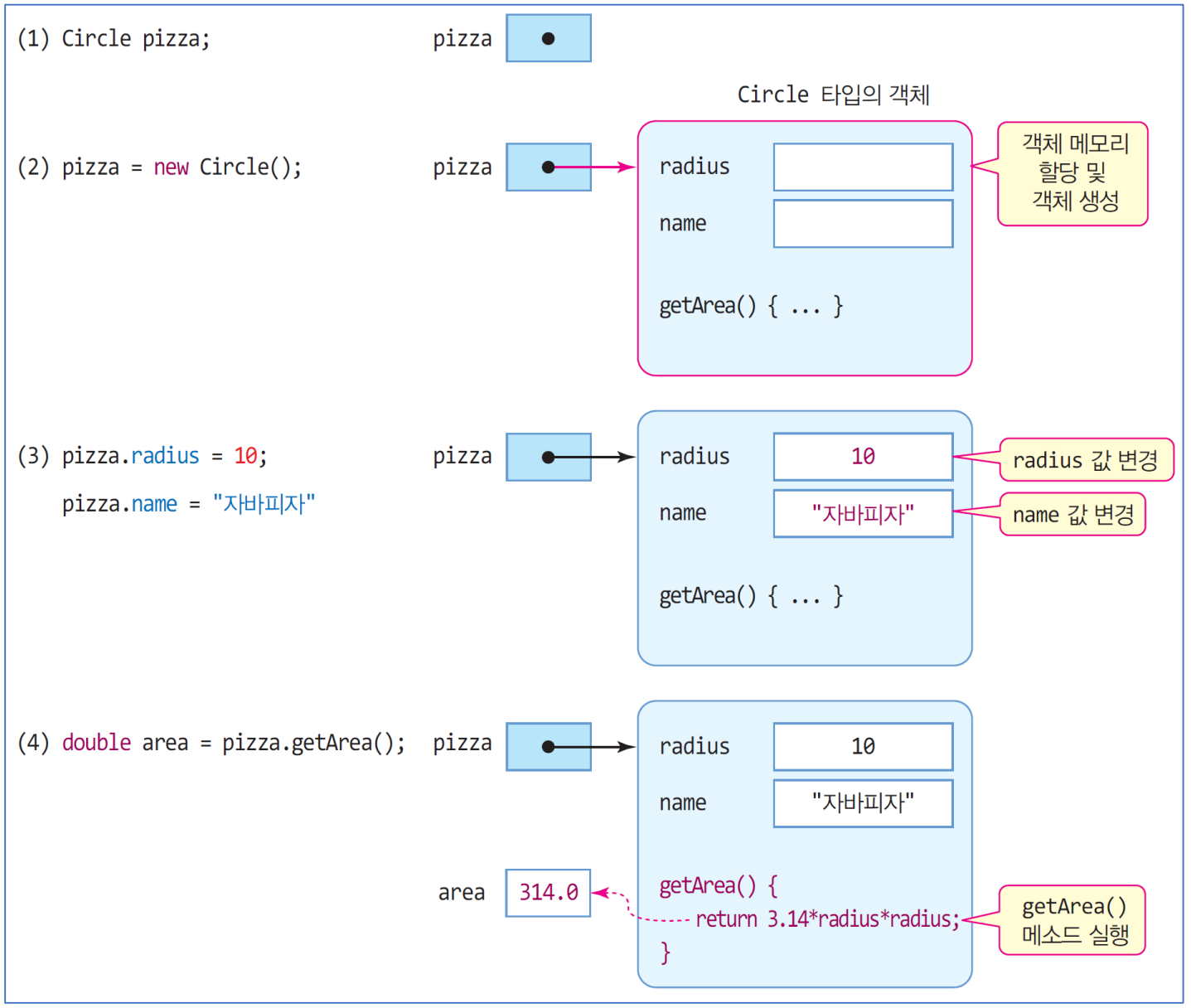
# <객체 생성과 접근>

## 1. 레퍼런스 변수 선언

## 2. 객체 생성
- new 연산자 이용

## 3. 객체 멤버 접근
- 점(.) 연산자 이용

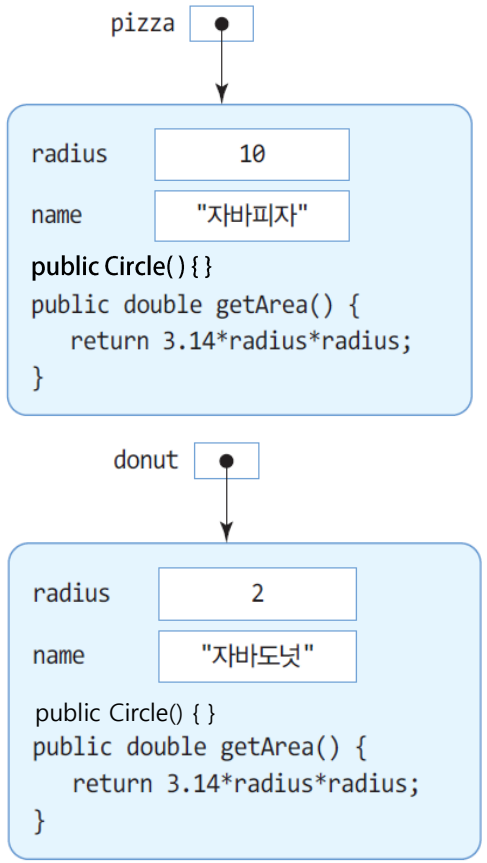


# 예제 4-1 : Circle 클래스의 객체 생성 및 활용

<반지름과 이름을 가진 Circle 클래스를 작성하고, Circle 클래스의 객체를 생성하라. 그리고 객체가 생성된 모습을 그려보라.>

```
public class Circle {
    int radius;                 // 원의 반지름 필드
    String name;                // 원의 이름 필드

    public Circle() { }         // 원의 생성자

    public double getArea() {  // 원의 면적 계산 메소드
        return 3.14*radius*radius;
    }

    public static void main(String[] args) {
        Circle pizza;
        pizza = new Circle();               // Circle 객체 생성
        pizza.radius = 10;                  // 피자의 반지름을 10으로 설정
        pizza.name = "자바피자";             // 피자의 이름 설정
        double area = pizza.getArea();      // 피자의 면적 알아내기
        System.out.println(pizza.name + "의 면적은 " + area);

        Circle donut = new Circle();        // Circle 객체 생성
        donut.radius = 2;                   // 도넛의 반지름을 2로 설정
        donut.name = "자바도넛";             // 도넛의 이름 설정
        area = donut.getArea();             // 도넛의 면적 알아내기
        System.out.println(donut.name + "의 면적은 " + area);
    }
}
```




```
자바피자의 면적은 314.0
자바도넛의 면적은 12.56
```