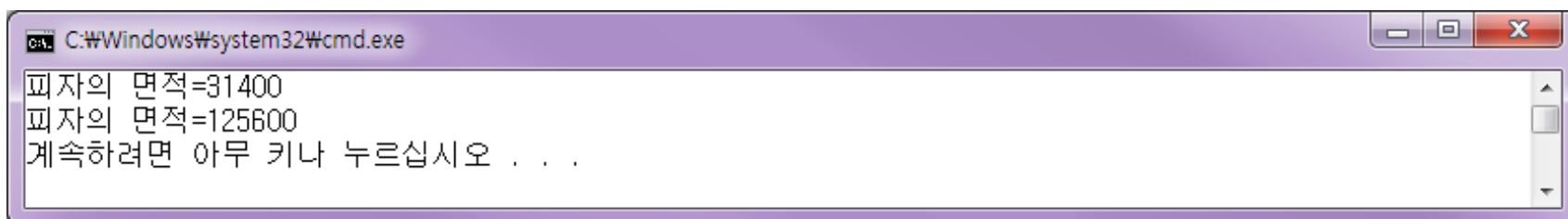


제4장 클래스와 객체

이번 장에서 만들어볼 프로그램



```
C:\Windows\system32\cmd.exe
피자의 면적=31400
피자의 면적=125600
계속하려면 아무 키나 누르십시오 . . .
```



```
C:\Windows\system32\cmd.exe
계속하려면 아무 키나 누르십시오 . . .
```

객체지향이란?

- 객체 지향 프로그래밍(OOP: **object-oriented programming**)은 우리가 살고 있는 실제 세계가 객체(object)들로 구성되어 있는 것과 비슷하게, 소프트웨어도 객체로 구성하는 방법이다.

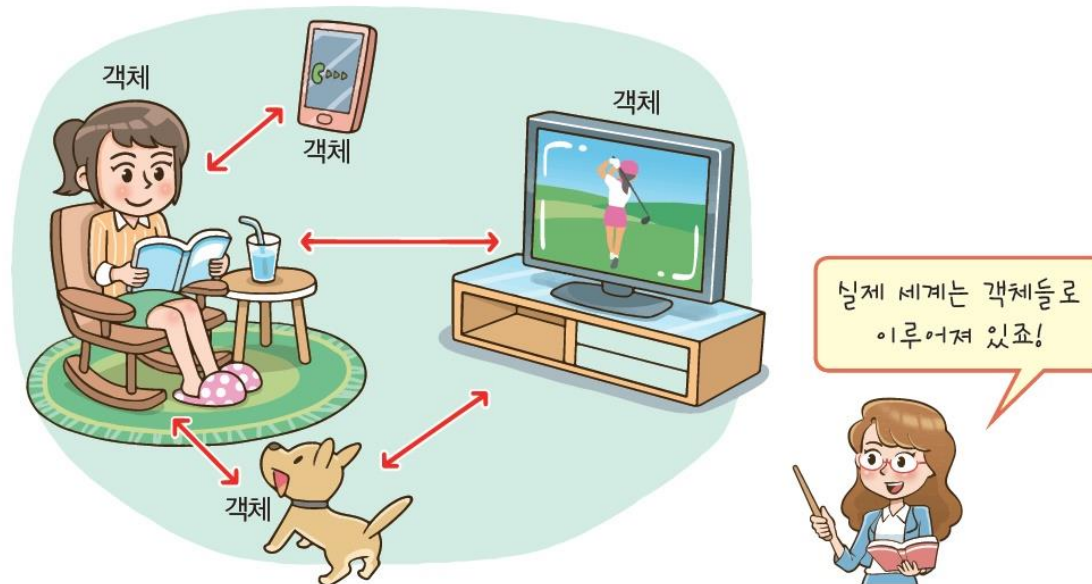


그림 4.1 실제 세계는 객체들로 이루어진다.

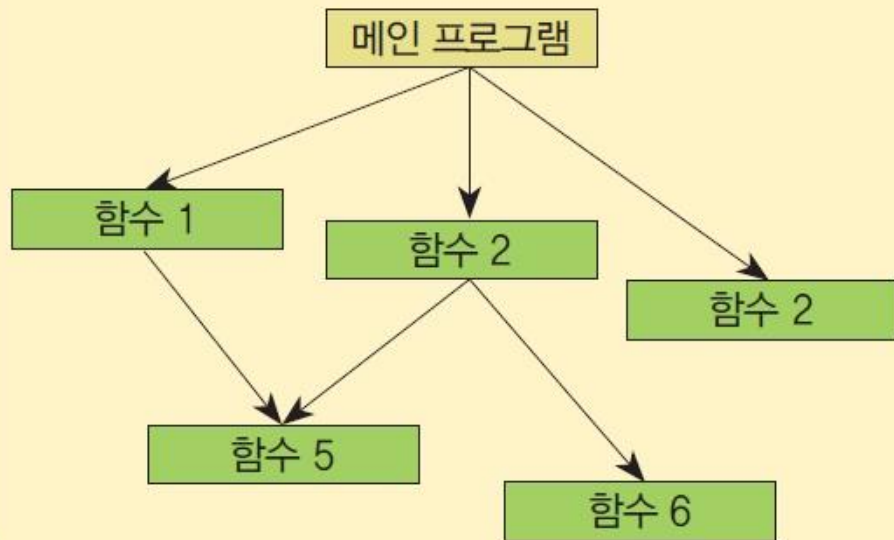
- 객체들은 메시지를 주고 받으면서 상호작용한다.



그림 4.2 객체들은 서로 메시지를 주고받으면서 상호작용한다.

절차 지향과 객체 지향

절차 지향 프로그래밍



객체 지향 프로그래밍

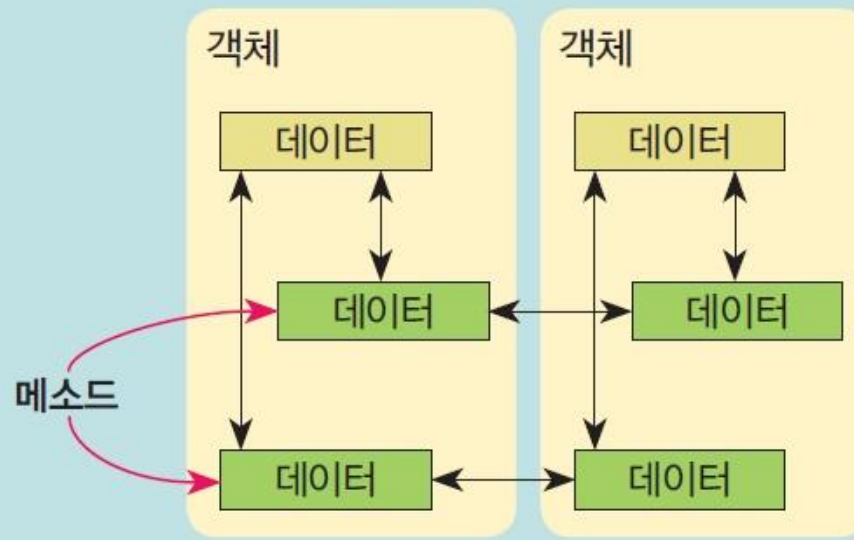
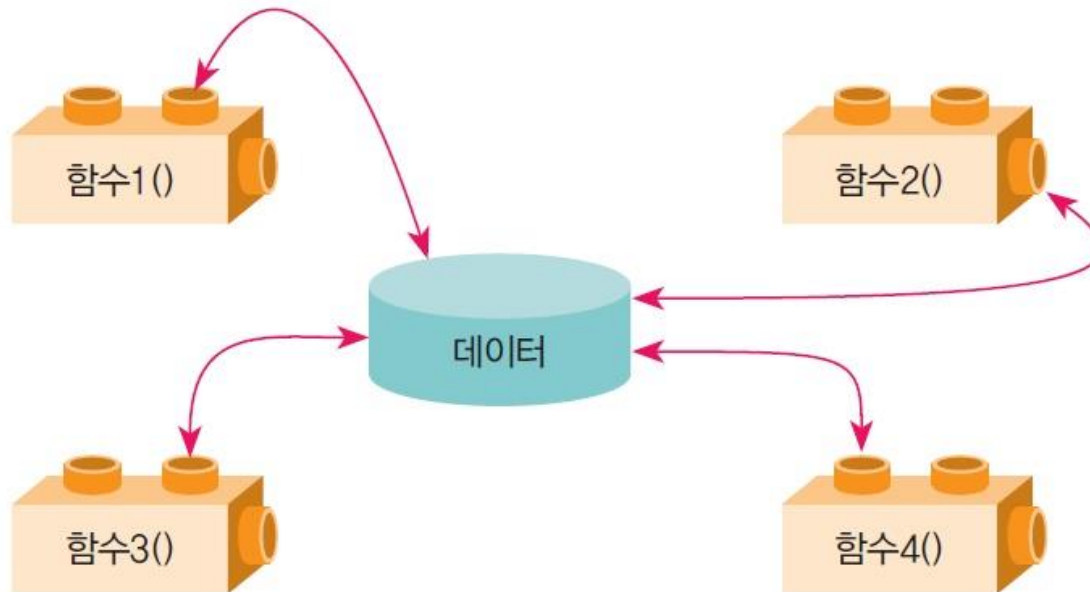


그림 4.3 절차 지향 프로그래밍과 객체 지향 프로그래밍의 비교

- 절차 지향 프로그래밍(**procedural programming**)은 프로시저(**procedure**)를 기반으로 하는 프로그래밍 방법이다.
- 프로시저는 일반적으로 함수를 의미한다.
- 절차 지향 프로그래밍에서 전체 프로그램은 함수들의 집합으로 이루어진다.

절차 지향의 문제점



절차 지향 프로그래밍에서는 데이터와 함수가 묶여 있지 않다.

그림 4.4 절차 지향 프로그래밍

객체지향 프로그래밍

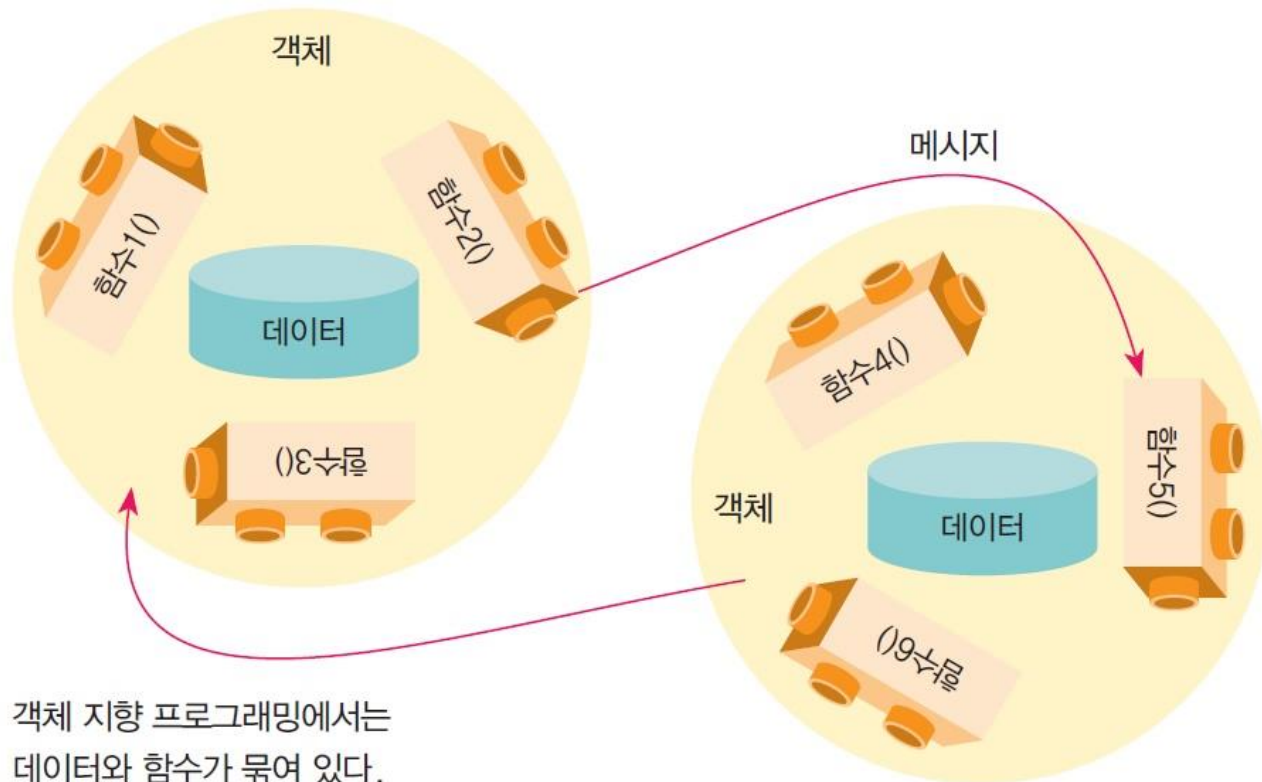


그림 4.5 객체 지향 프로그래밍

- 객체는 상태와 동작을 가지고 있다. 객체의 상태(**state**)는 객체의 속성이다. 객체의 동작(**behavior**)은 객체가 취할 수 있는 동작이다.



그림 4.7 자동차 객체의 예

멤버 변수와 멤버 함수

상태

색상: 빨강
속도: 100 km/h
기어: 2단

동작

출발하기
정지하기
가속하기
감속하기



상태

color: 빨강
speed: 100 km/h
gear: 2단

동작

```
start(){ ... }
stop(){ ... }
speedUP(){ ... }
speedDown(){ ... }
```

소프트웨어 객체 = 변수 + 함수

그림 4.8 멤버 변수와 멤버 함수

클래스 = 객체의 설계도

- 객체 지향 소프트웨어에서도 같은 객체들이 여러 개 필요한 경우도 있다. 이러한 객체들은 모두 하나의 설계도로 만들어진다. 바로 이 설계도를 클래스(class)라고 한다.

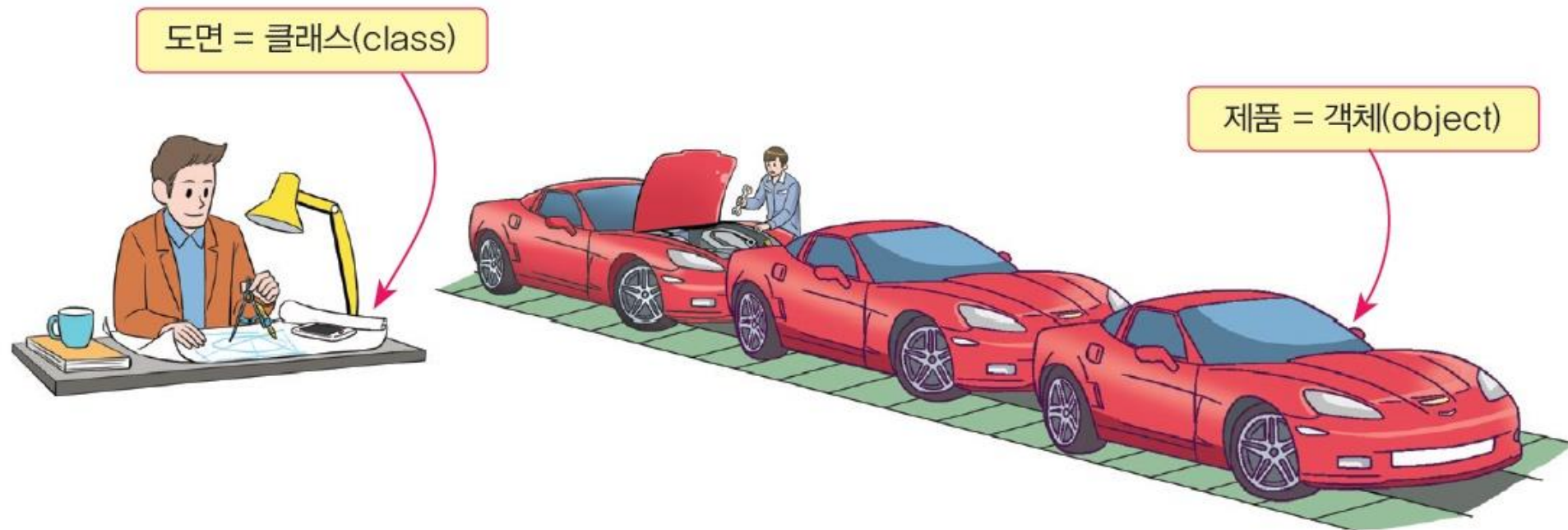


그림 4.9 객체를 클래스라는 설계도로 생성된다.

클래스 작성하기

문법 5.1

클래스 정의

```
class 클래스이름 {
```

```
자료형 멤버변수1;
```

```
자료형 멤버변수2;
```

멤버 변수

```
반환형 멤버함수1();
```

```
반환형 멤버함수2();
```

멤버 함수 선언부

```
};
```

클래스 작성의 예

class 키워드로
클래스 선언

클래스 이름

```
class Circle {  
public:
```

접근 지정자

```
    int radius;  
    string color;
```

멤버 변수

```
    double calcArea() {  
        return 3.14*radius*radius;  
    }
```

멤버 함수

```
};
```



- **private** 멤버는 클래스 안에서만 접근(사용)될 수 있다.
- **protected** 멤버는 클래스 안과 상속된 클래스에서 접근이 가능하다(상속은 아직 학습하지 않았다).
- **public** 멤버는 어디서나 접근이 가능하다.

객체 생성하기

```
Circle obj; // obj는 Circle 자료형의 변수이다.
```

클래스 이름은 자료형의
이름으로 생각할 수 있다.

객체의 이름

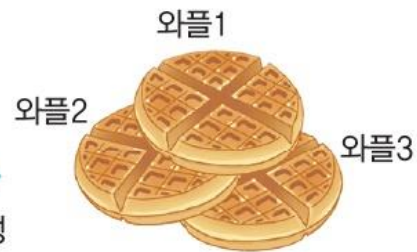
클래스는 객체를 찍어내는
틀과 같다.



클래스



객체생성



객체

객체의 멤버 접근

- 멤버에 접근하기 위해서는 도트(.) 연산자를 사용한다.

```
obj.radius = 3;           // obj의 멤버 변수인 radius에 3을 저장한다.
```

obj 객체의

radius 멤버 변수에 접근


```
#include <iostream>
using namespace std;

class Circle {
public:
    int radius;        // 반지름
    string color;       // 색상

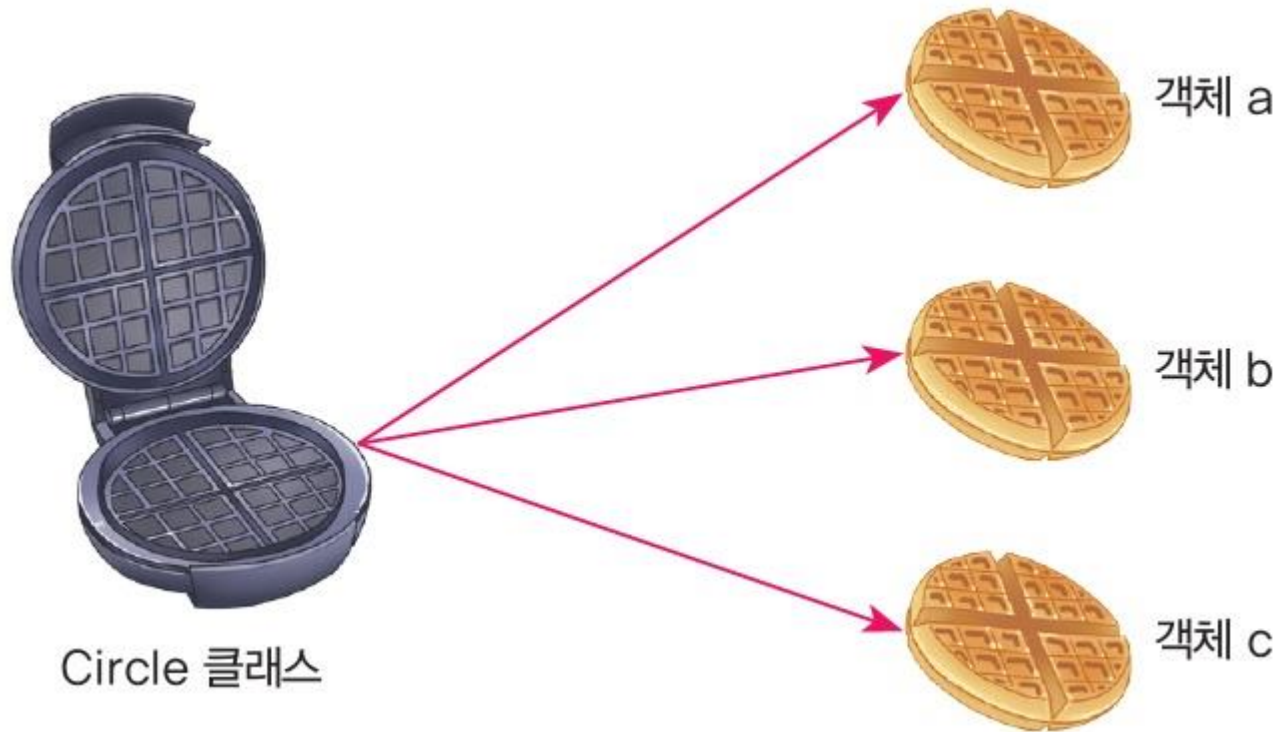
    double calcArea() {
        return 3.14*radius*radius;
    }
};

int main() {
    Circle obj;

    obj.radius = 100;
    obj.color = "blue";

    cout << "원의 면적=" << obj.calcArea() << "\n";
    return 0;
}
```

하나의 클래스로 많은 객체 생성 가능

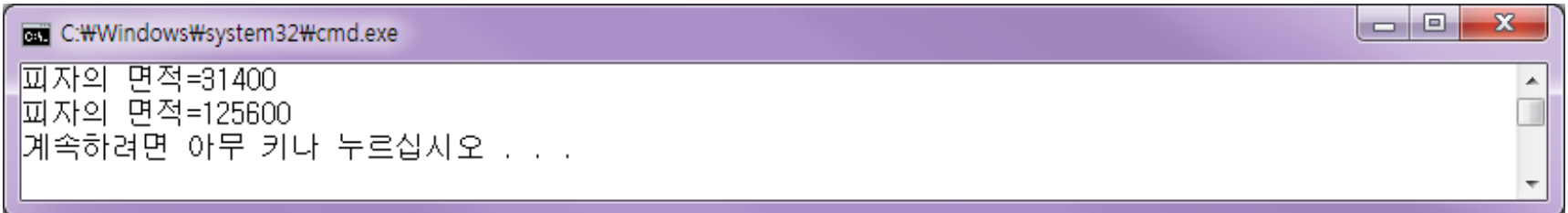


여러 개의 객체 생성 예제

```
int main()
{
    Circle pizza1, pizza2;

    pizza1.radius = 100;
    pizza1.color = "yellow";
    cout << "피자의 면적=" << pizza1.calcArea() << "\n";

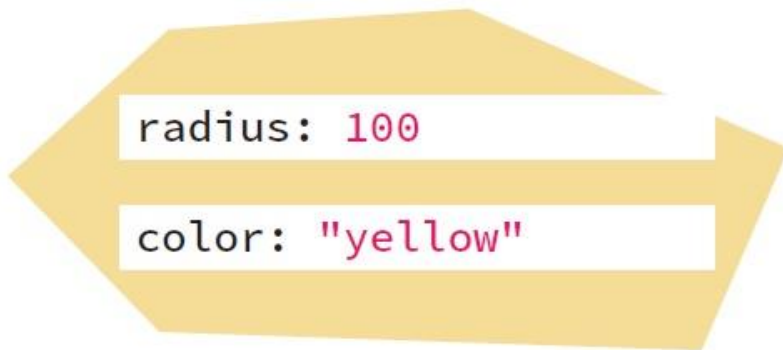
    pizza2.radius = 200;
    pizza2.color = "white";
    cout << "피자의 면적=" << pizza2.calcArea() << "\n";
    return 0;
}
```



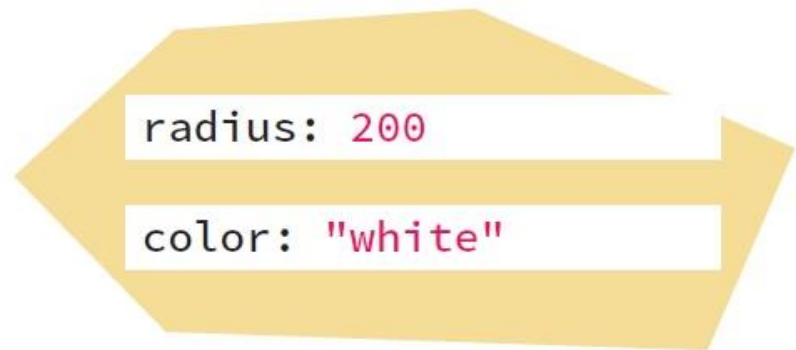
```
C:\Windows\system32\cmd.exe
피자의 면적=31400
피자의 면적=125600
계속하려면 아무 키나 누르십시오 . . .
```

각 객체 상태

- 각 객체의 멤버 변수 값은 서로 다르다.



pizza 1



pizza 2

Lab: 사각형 클래스

- 아래 클래스를 가지고 하나의 객체를 생성하는 프로그램을 작성해보자.

```
class Rectangle {  
public:  
    int width, height;  
    int calcArea() {  
        return width*height;  
    }  
};
```

solution

```
#include <iostream>
using namespace std;

class Rectangle {
public:
    int width, height;
    int calcArea() {
        return width*height;
    }
};

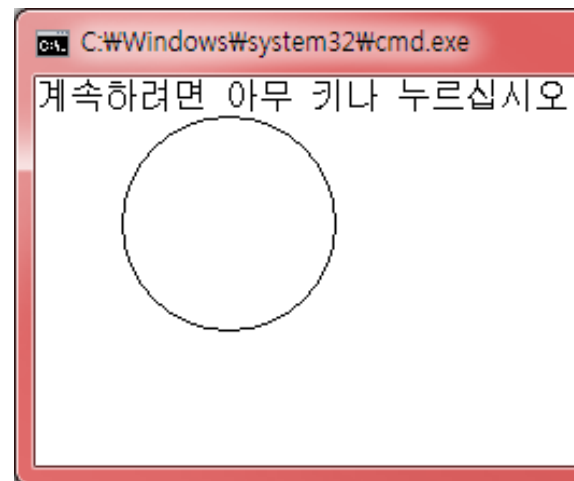
int main() {
    Rectangle obj;

    obj.width = 3;
    obj.height = 4;
    int area = obj.calcArea();
    cout << "사각형의 넓이: " << area<<endl;
    return 0;
}
```

Lab: 원 객체 그리기

```
#include <windows.h>
#include <stdio.h>

int main()
{
    HDC hdc = GetWindowDC(GetForegroundWindow());
    Ellipse(hdc, 100, 100, 180, 180);
}
```



solution

```
#include <iostream>
#include <windows.h>

using namespace std;

class Circle {
public:
    int x, y, radius; // 원의 중심점과 반지름
    string color;      // 원의 색상
    double calcArea() { // 원의 면적을 계산하는 함수
        return 3.14*radius*radius;
    }
    void draw() { // 원을 화면에 그리는 함수
        HDC hdc = GetWindowDC(GetForegroundWindow());
        Ellipse(hdc, x - radius, y - radius, x + radius, y + radius);
    }
};
```


Lab: Car 클래스 작성

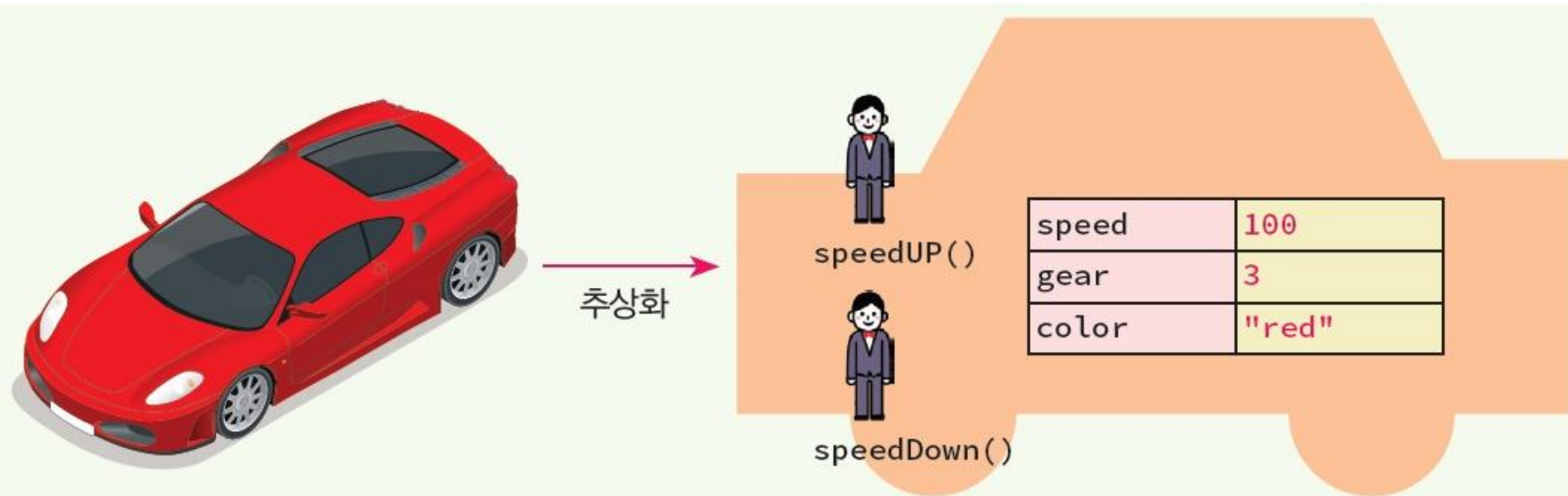


그림 4.10 추상화

solution

```
#include <iostream>
#include <string>
using namespace std;

class Car {
public:
    // 멤버 변수 선언
    int speed; // 속도
    int gear; // 기어
    string color; // 색상

    // 멤버 함수 선언
    void speedUp() { // 속도 증가 멤버 함수
        speed += 10;
    }

    void speedDown() { // 속도 감소 멤버 함수
        speed -= 10;
    }
};
```

solution

```
int main()
{
    Car myCar;

    myCar.speed = 100;
    myCar.gear = 3;
    myCar.color = "red";

    myCar.speedUp();
    myCar.speedDown();

    return 0;
}
```

멤버 함수의 중복 정의

- 멤버 함수도 중복 정의(오버로딩)가 가능함

```
class Car {  
private:  
    int speed;           //속도  
    int gear;            //기어  
    string color;        //색상  
public:  
    int getSpeed();  
    void setSpeed(int s);  
    void setSpeed(double s);  
};
```

멤버 함수 정보 정의

```
#include <iostream>
#include <string>
using namespace std;

class PrintData {
public:
    void print(int i) { cout << i << endl; }
    void print(double f) { cout << f << endl; }
    void print(string s = "No Data!") { cout << s << endl; }
};

int main() {
    PrintData obj;

    obj.print(1);
    obj.print(3.14);
    obj.print("C++14 is cool.");
    obj.print();
    return 0;
}
```

실행 결과



```
C:\Windows\system32\cmd.exe
1
3.14
C++14 is cool.
No Data!
계속하려면 아무 키나 누르십시오 . . .
```

클래스의 인터페이스와 구현의 분리

- 복잡한 클래스인 경우에는 멤버 함수를 클래스 외부에서 정의

```
#include <iostream>
using namespace std;


class Circle {
public:
    double calcArea();

    int radius;        // 반지름
    string color;       // 색상
};
```

클래스의 인터페이스와 구현의 분리

// 클래스 외부에서 멤버 함수들이 정의된다.

```
double Circle::calcArea() {  
    return 3.14*radius*radius;  
}  
  
int main()  
{  
    Circle c;  
    c.radius = 10;  
    cout << c.calcArea() << endl;  
    return 0;  
}
```



cmd C:\Windows\system32\cmd.exe

```
314  
계속하려면 아무 키나 누르십시오 . . .
```


- 이름 공간(name space)는 식별자 (자료형, 함수, 변수 등의 이름)의 영역
- 이름 공간은 코드를 논리적 그룹으로 구성하고 특히 코드에 여러 라이브러리가 포함되어 있을 때 발생할 수 있는 이름 충돌을 방지하는 데 사용된다.

```
using namespace std;
```

using 문장을 사용하지 않으면

```
#include <iostream>

class Circle {
public:
    double calcArea();

    int radius;      // 반지름
    std::string color; // 색상
};

double Circle::calcArea() {
    return 3.14*radius*radius;
}

int main() {
    Circle c;
    c.radius = 10;
    std::cout << c.calcArea() << std::endl;
    return 0;
}
```

클래스의 선언과 클래스의 정의 분리

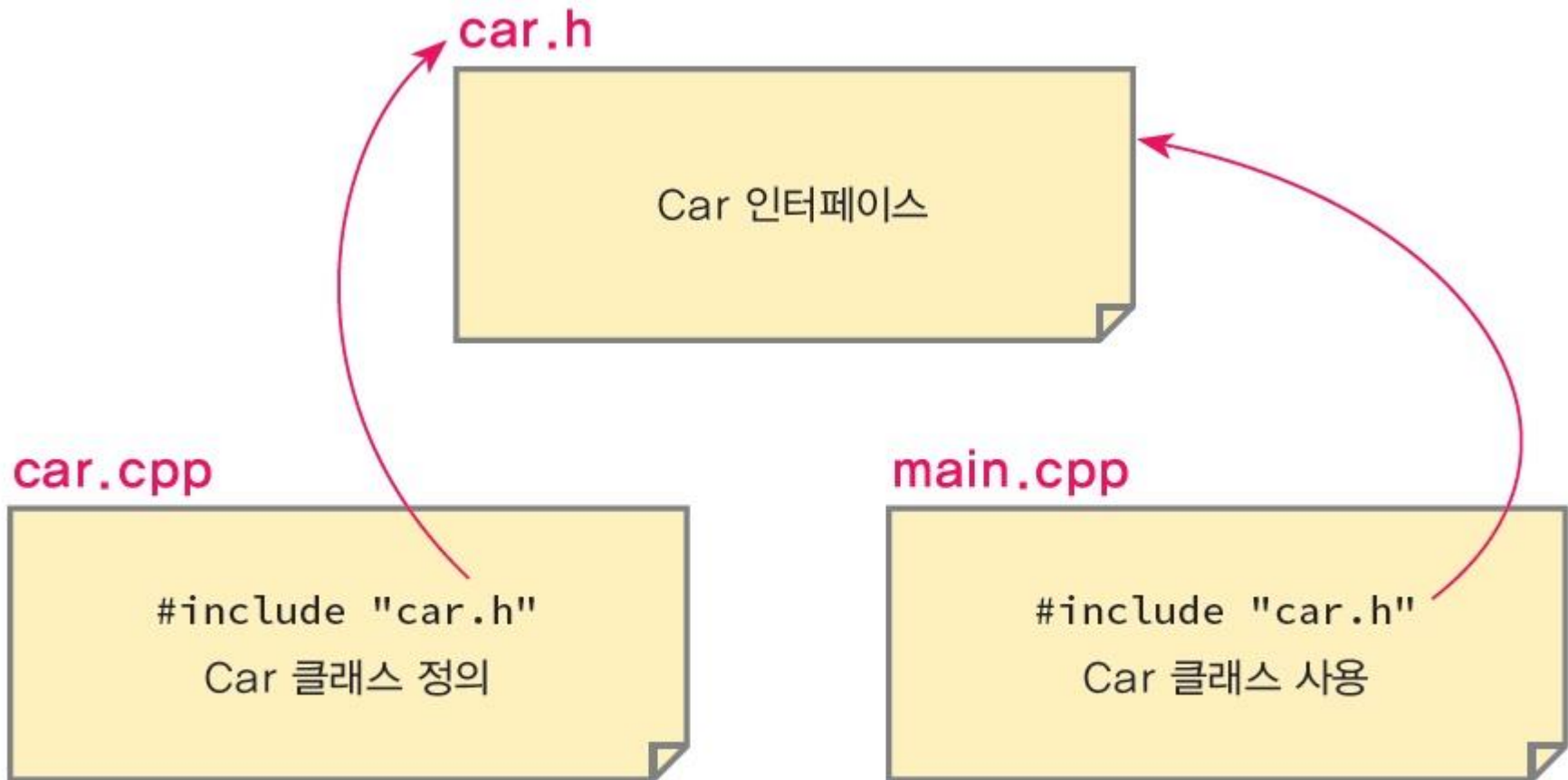


그림 4.11 클래스를 헤더 파일과 소스 파일로 분리

car.h

```
#include <iostream>
#include <string>
using namespace std;

class Car
{
    int speed;        //속도
    int gear;         //기어
    string color;     //색상
public:
    int getSpeed();
    void setSpeed(int s);
};
```

car.cpp

```
#include "car.h"

int Car::getSpeed()
{
    return speed;
}

void Car::setSpeed(int s)
{
    speed = s;
}
```

main.cpp

```
#include "car.h"
using namespace std;

int main()
{
    Car myCar;

    myCar.setSpeed(80);
    cout << "현재 속도는 " << myCar.getSpeed() << endl;

    return 0;
}
```



C:\Windows\system32\cmd.exe

```
현재 속도는 80
계속하려면 아무 키나 누르십시오 . . .
```

□ 구조체(structure) = 클래스

```
struct BankAccount { // 은행계좌
    int accountNumber; // 계좌번호
    int balance; // 잔액을표시하는변수
    double interest_rate; // 연이자
    double get_interrest(int days){
        return (balance*interest_rate)*((double)days/365.0);
    }
};
```

모든 멤버가 디폴트로 public이 된다.

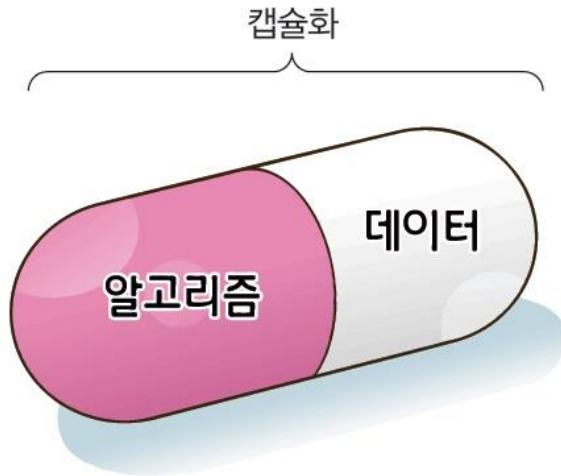
개체 지향의 개념들

- 자료 추상화
- 캡슐화
- 정보은닉
- 상속
- 다형성

자료 추상화 (data abstraction)

- (객체를 사용하는데 있어서) 객체의 중요한 핵심적인 특징 (속성과 행위) 들을 추출하는 과정 => 구현과 분리

캡슐화 (encapsulation)

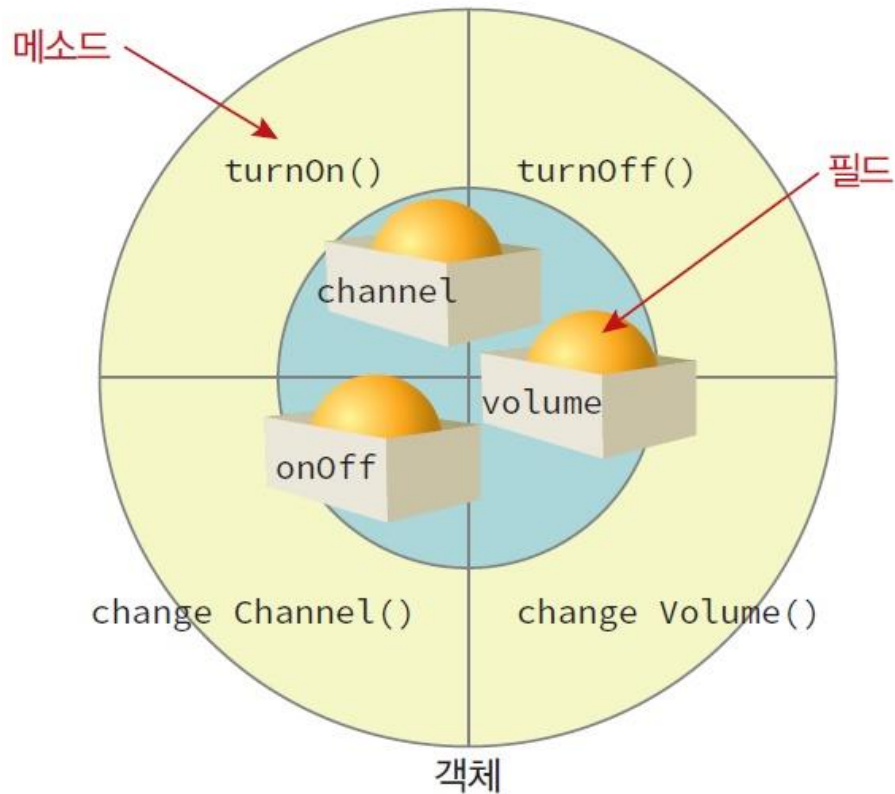


캡슐화는 데이터와 알고리즘을 하나로 묶는 것입니다.



캡슐화 되어 있지 않은 데이터와 코드는 사용하기 어렵다.





보통은 데이터들은 공개되지
않고 몇 개의 메소드 만이
외부로 공개됩니다.



- 은닉이란 내부 데이터, 내부 연산을 외부에서 접근하지 못하도록 은닉(hiding) 혹은 격리(isolation)시키는 것 => 접근 지정자를 **private**로 함
- 객체 외부에서 객체내의 자료로의 접근을 제한함
- 공개된 멤버함수(메소드)를 통해 데이터를 접근할 수 있음
- 데이터에 대한 불필요한 접근을 차단하여서 데이터를 보호

- 한 클래스가 기존의 다른 클래스에서 정의된 속성(자료, 함수)를 이어받아 그대로 사용
- 이미 정의된 클래스를 바탕으로 필요한 기능을 추가하여 정의
- 소프트웨어 재사용 지원
- **is-a** 관계

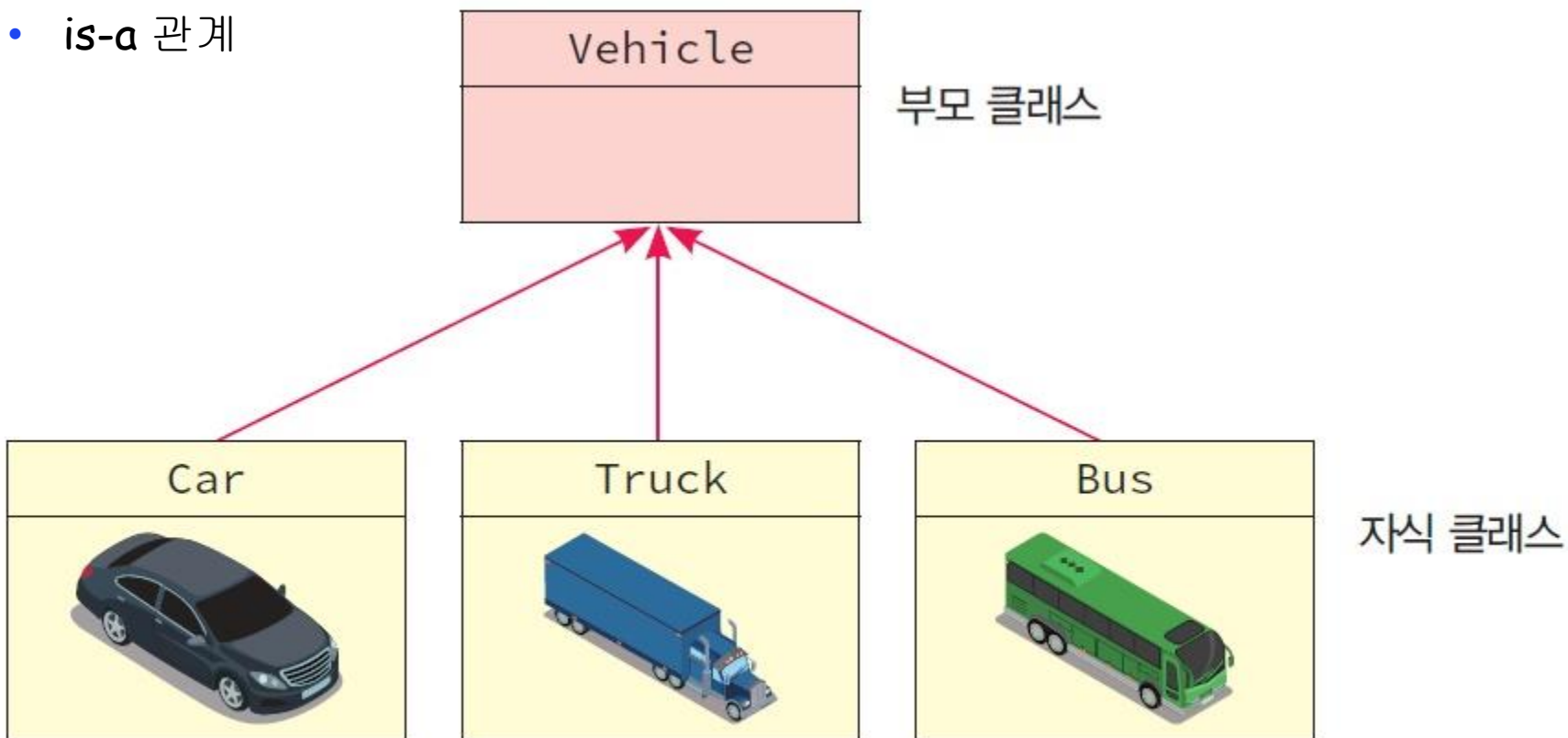
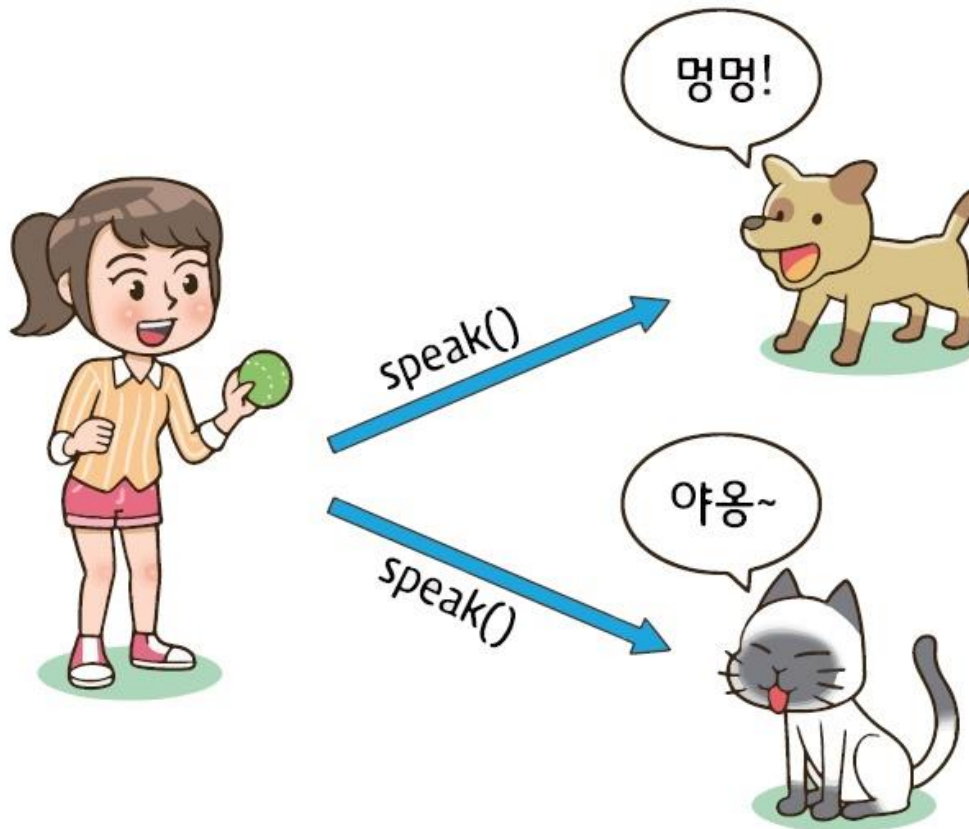


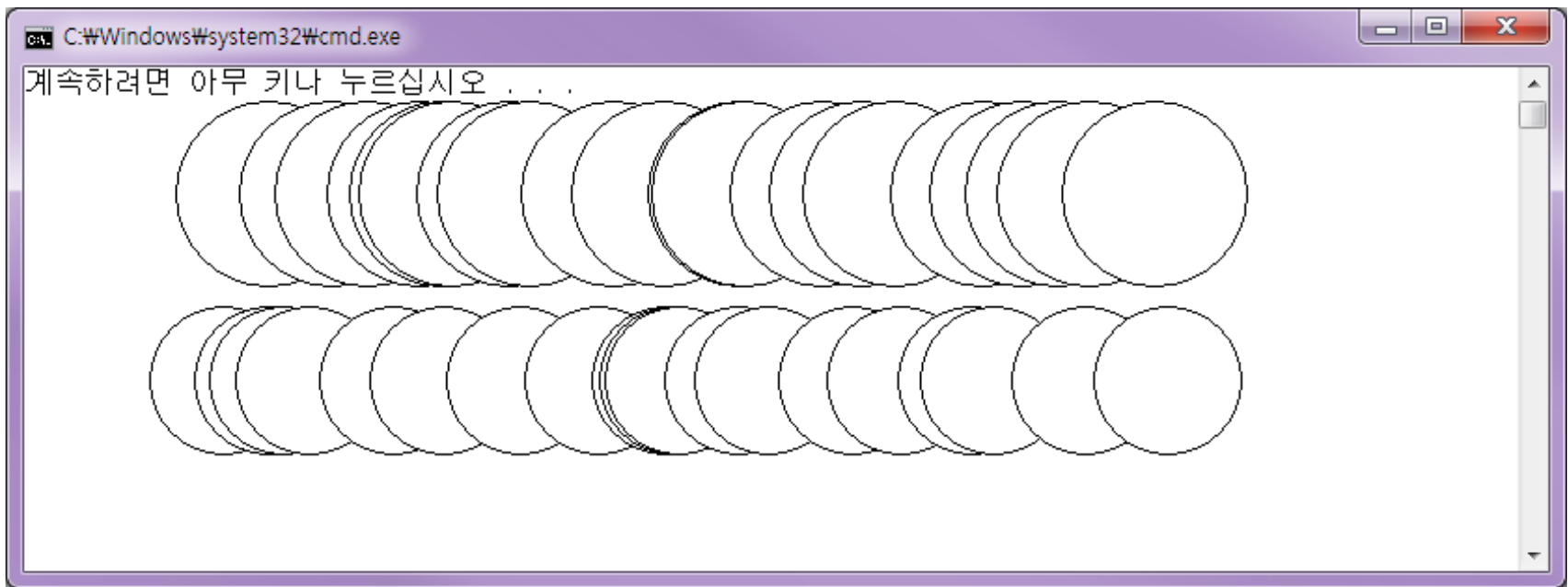
그림 4.12 상속의 개념

- 객체들의 타입이 다르면 동일한 메소드에 대하여 서로 다른 동작을 하는 것



Lab: 원들의 경주

- 두 대의 원을 생성한 후에 난수를 발생하여 원들을 움직인다. 원을 화면에 그리는 `draw()` 함수와 난수를 발생하여 원을 움직이는 함수 `move()`를 클래스에 추가한다.



solution

```
#include <iostream>
#include <windows.h>
using namespace std;

class Circle {
public:
    void init(int xval, int yval, int r);
    void draw();
    void move();
private:
    int x, y, radius;
};

// 아직 생성자를 학습하지 않았기 때문에 init() 함수 사용
void Circle::init(int xval, int yval, int r) {
    x = xval;
    y = yval;
    radius = r;
}
```

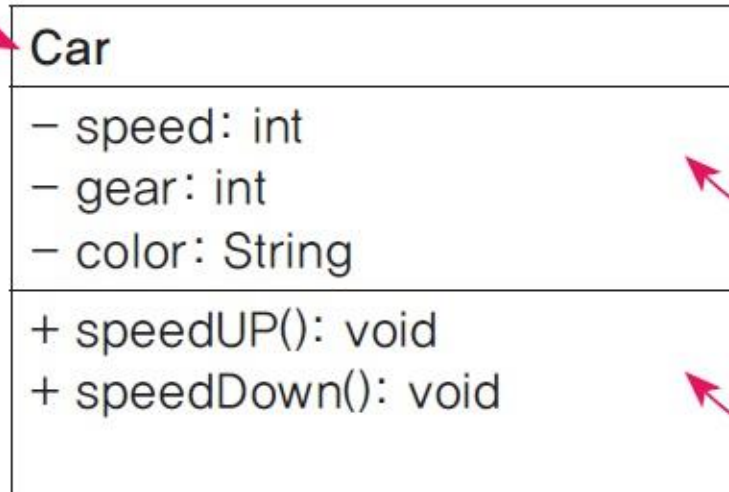
```
void Circle::draw() {  
    HDC hdc = GetWindowDC(GetForegroundWindow());  
    Ellipse(hdc, x - radius, y - radius, x + radius, y + radius);  
}  
void Circle::move() {  
    x += rand() % 50;  
}  
  
int main() {  
    Circle c1;  
    Circle c2;  
  
    c1.init(100, 100, 50);  
    c2.init(100, 200, 40);  
    for (int i = 0; i < 20; i++) {  
        c1.move();  
        c1.draw();  
        c2.move();  
        c2.draw();  
        Sleep(1000);  
    }  
    return 0;  
}
```


- 객체 지향 프로그래밍에서도 프로그래머들은 애플리케이션을 구성하는 클래스들 간의 관계를 그리기 위하여 클래스 다이어그램(class diagram)을 사용한다. 가장 대표적인 클래스 다이어그램 표기법은 **UML(Unified Modeling Language)**이다.



UML

클래스의 이름을
적어준다.



클래스의 속성을 나타낸다.
즉 필드를 적어준다.

클래스의 동작을 나타낸다.
즉 메소드를 적어준다.

그림 4.13 UML의 예

관계	화살표
일반화(generalization), 상속(inheritance)	
구현(realization)	
구성관계(composition)	
집합관계(aggregation)	
유향 연관(direct association)	
양방향 연관(bidirectional association)	
의존(dependency)	

그림 4.14 UML에서 사용되는 화살표의 종류

UML의 예

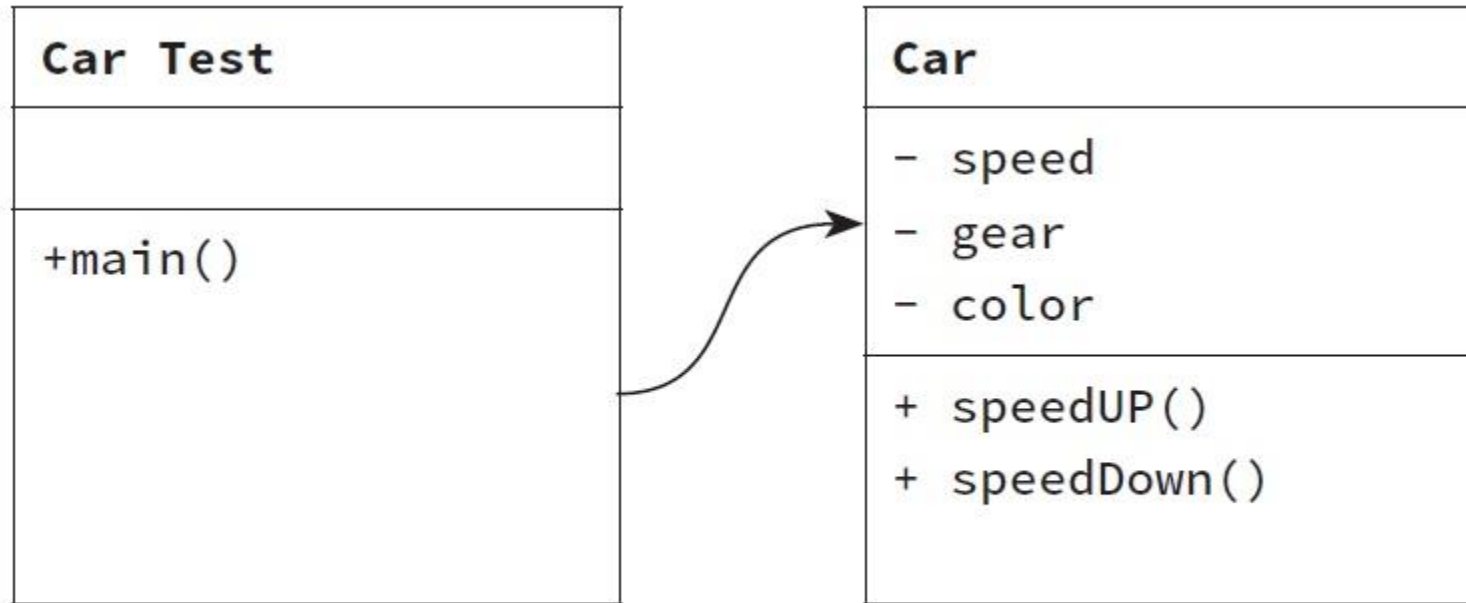


그림 4.15 Car 예제의 UML