

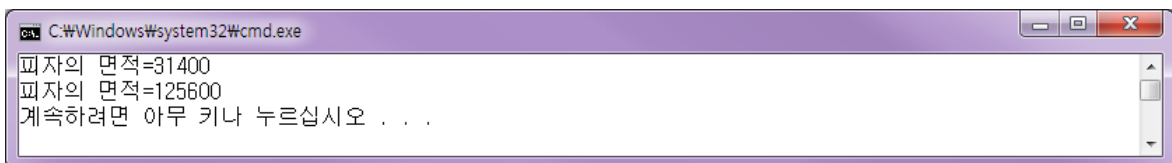
제4장 클래스와 객체

1. 객체 지향 기법을 이해한다.
2. 클래스를 작성할 수 있다.
3. 클래스에서 객체를 생성할 수 있다.
4. 생성자를 이용하여 객체를 초기화할 수 있다.
5. 접근자와 설정자를 사용할 수 있다.

1

이번 장에서 만들어볼 프로그램

피자를 나타내는 클래스 **Pizza**를 작성하고 객체를 생성



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

원을 나타내는 클래스 **Circle**을 정의하고 화면에 원을 그려보자.



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

2

4.2 객체 지향이란?

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

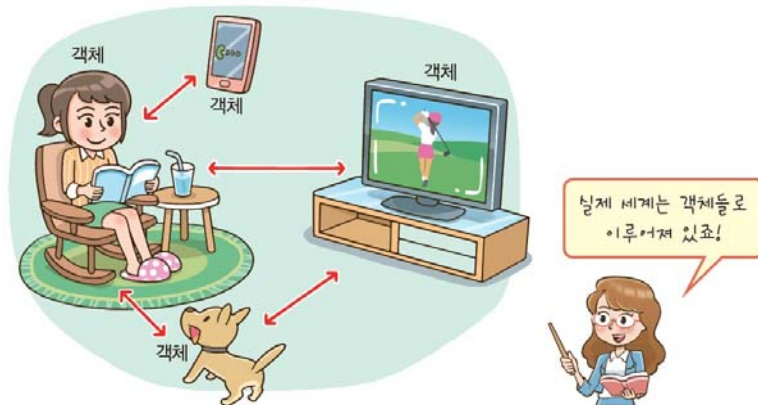


그림 4.1 실제 세계는 객체들로 이루어진다.

3

객체와 메시지

- 객체들은 메시지를 주고 받으면서 상호작용한다.



그림 4.2 객체들은 서로 메시지를 주고받으면서 상호작용한다.

4

절차 지향과 객체 지향

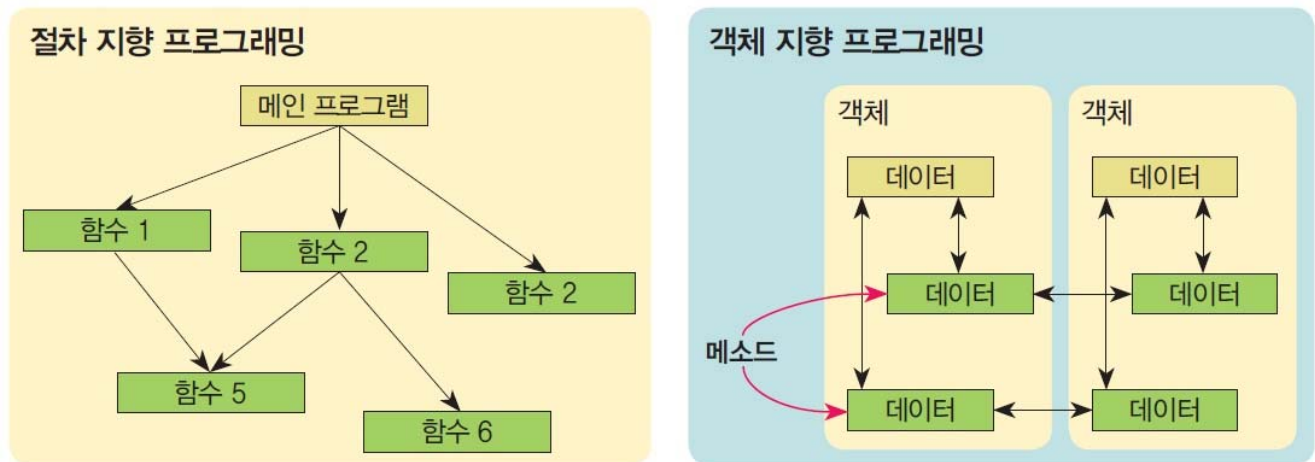


그림 4.3 절차 지향 프로그래밍과 객체 지향 프로그래밍의 비교

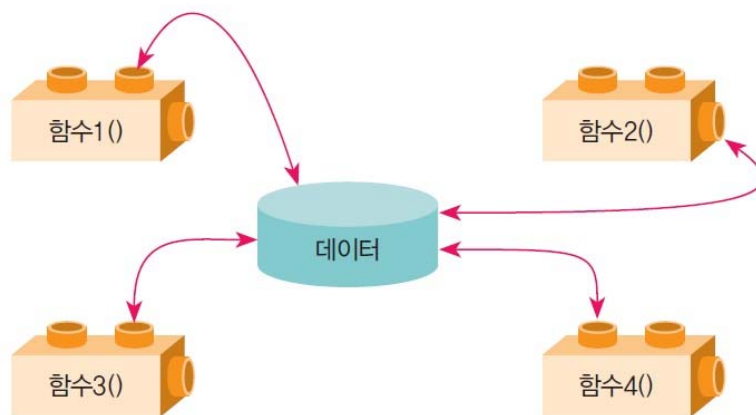
5

절차 지향 프로그래밍

- 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.

6

절차지향의 문제점

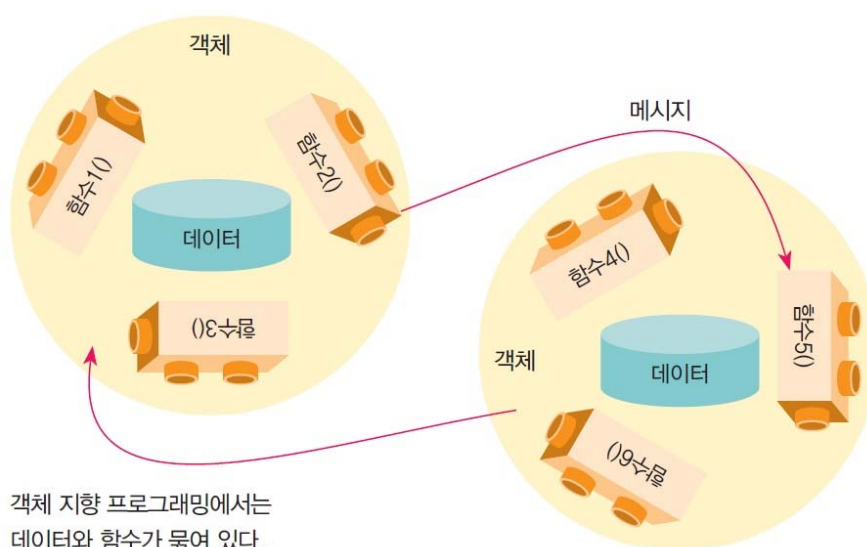


절차 지향 프로그래밍에서는 데이터와 함수가 묶여 있지 않다.

그림 4.4 절차 지향 프로그래밍

7

객체 지향 프로그래밍



객체 지향 프로그래밍에서는
데이터와 함수가 묶여 있다.

그림 4.5 객체 지향 프로그래밍

8

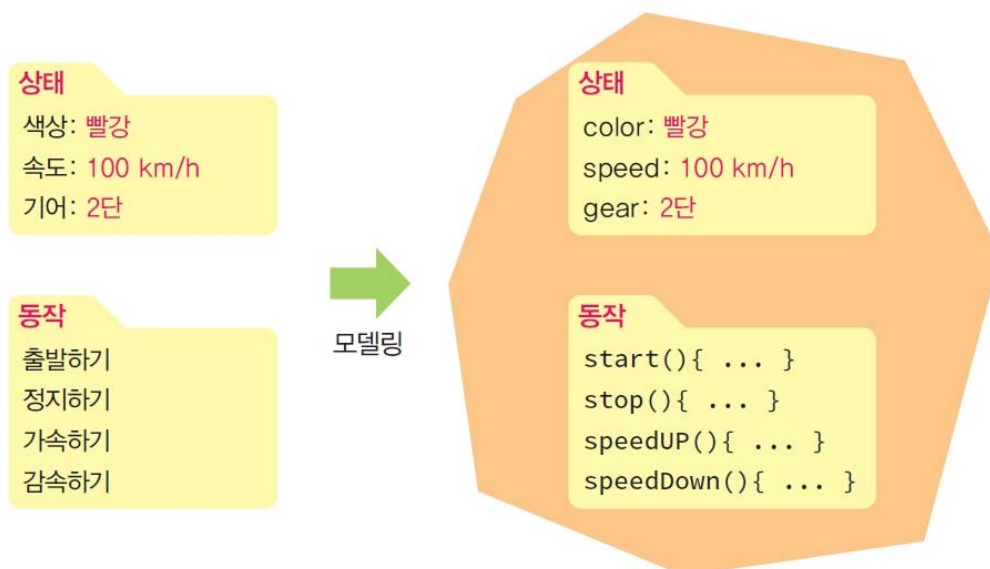
4.3 객체의 구성

- 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



소프트웨어 객체 = 변수 + 함수

그림 4.8 멤버 변수와 멤버 함수

4.4 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.

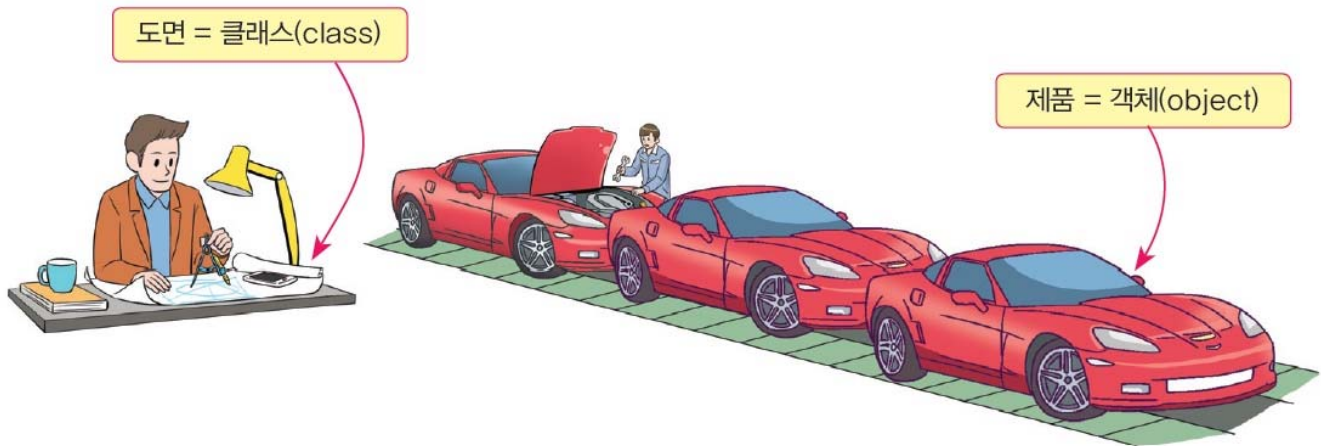


그림 4.9 객체를 클래스라는 설계도로 생성된다.

11

클래스 작성하기

문법 5.1

클래스 정의

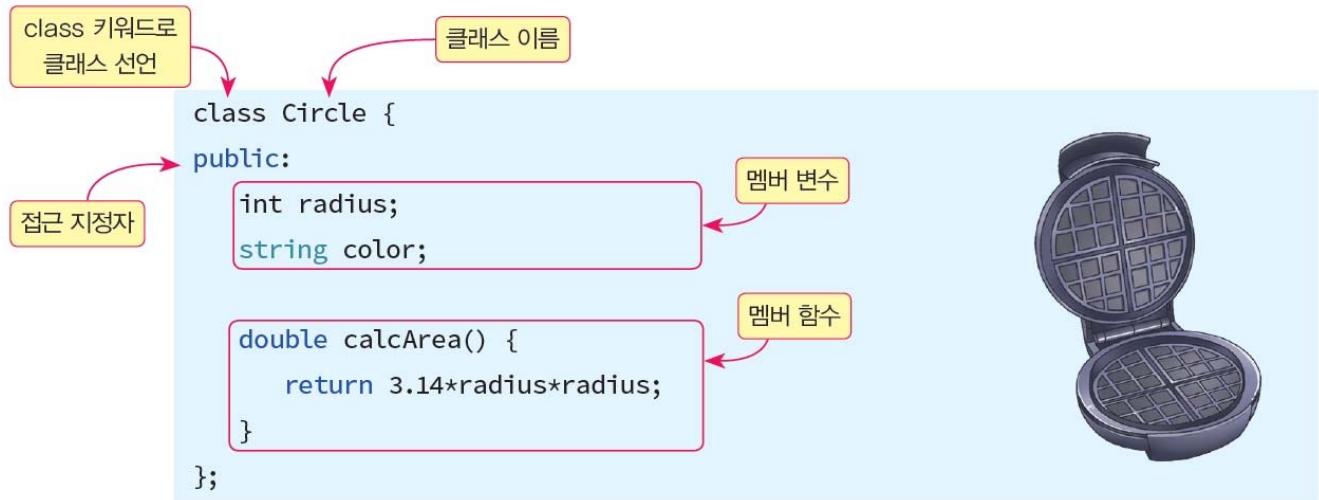
```
class 클래스이름 {  
    자료형 멤버변수1;  
    자료형 멤버변수2;  
  
    반환형 멤버함수1();  
    반환형 멤버함수2();  
};
```

멤버 변수

멤버 함수 선언부

12

클래스 작성의 예



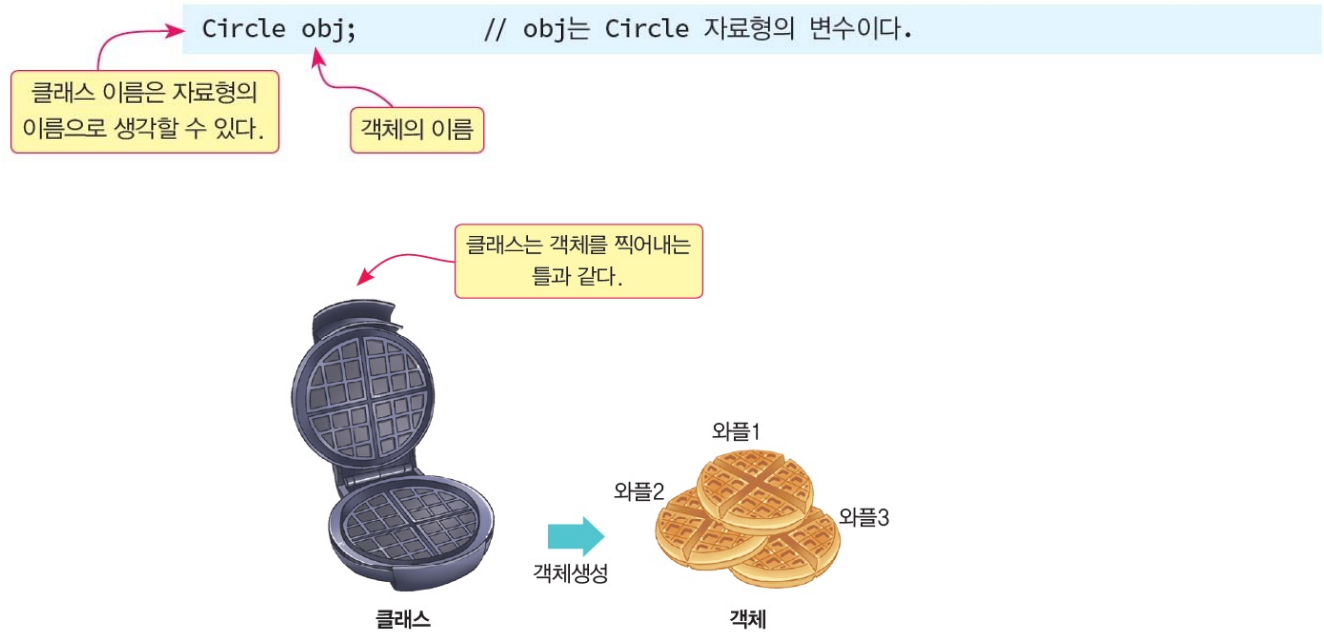
13

접근지정자

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

14

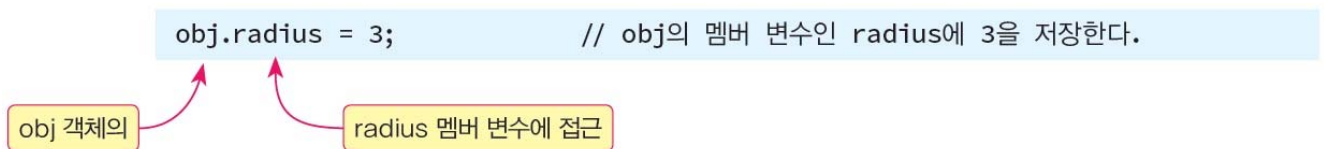
객체 생성하기



15

객체의 멤버 접근

- 멤버에 접근하기 위해서는 도트(.) 연산자를 사용한다.



16


```

#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;
}

```

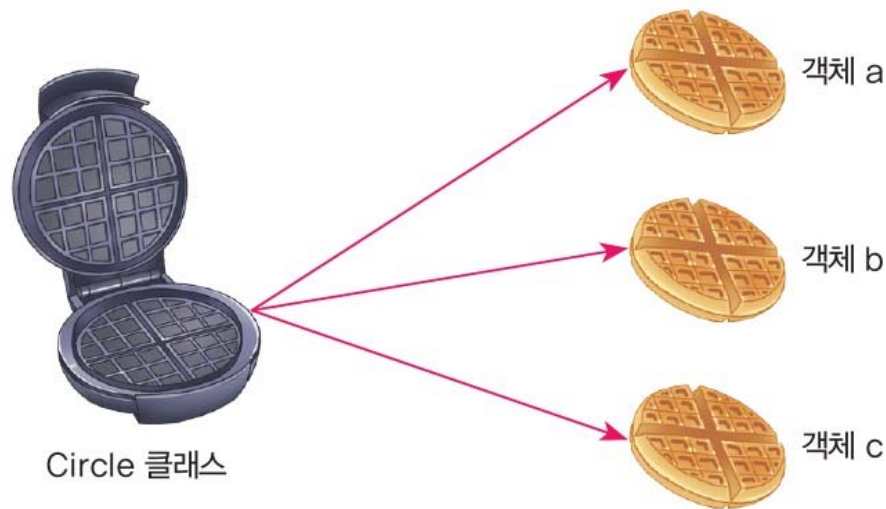
17

실습

- 사각형을 클래스 **Rectangle**로 표현하라. width, height를 멤버 변수로 가지고 면적을 계산하는 calcArea()도 정의하고 실행 예를 보이시오.

18

하나의 클래스로 많은 객체 생성 가능



19

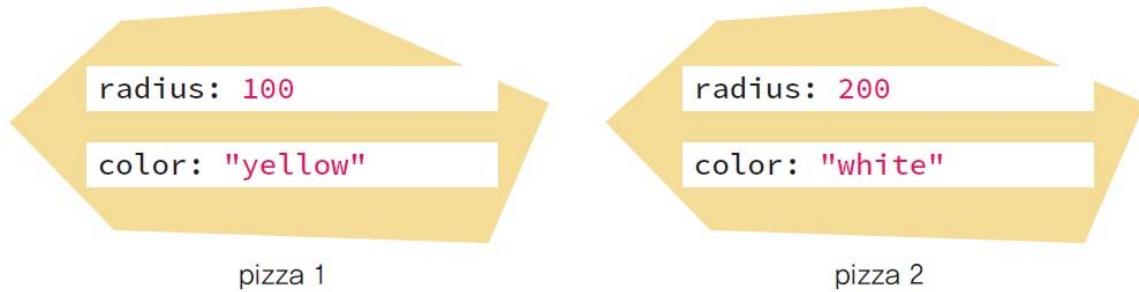
여러 개의 객체 생성 예제

```
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;
}
```

20

각 객체 상태

- 각 객체의 멤버 변수 값은 서로 다르다.



21

Lab: 사각형 클래스

- 아래 클래스를 가지고 하나의 객체를 생성하는 프로그램을 작성해보자.

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

22

solution

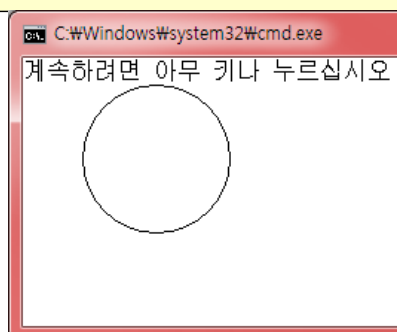
```
#include <iostream>
using namespace std;
class Rectangle {
public:
    int width, height;
    int calcArea() {
        return width*height;
    }
};
int main() {
    Rectangle obj;

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

23

Lab: 원 객체 그리기

```
#include <windows.h>
#include <stdio.h>
int main()
{
    HDC hdc = GetWindowDC(GetForegroundWindow());
    Ellipse(hdc, 100, 100, 180, 180);
}
```



24

solution

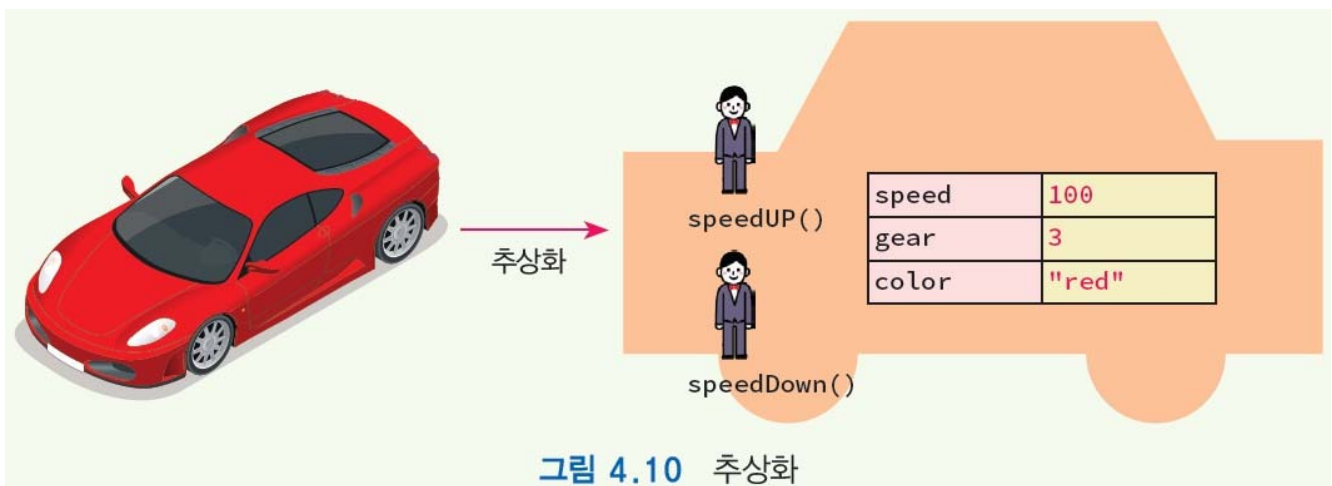
```
#include <iostream>
#include <windows.h>

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);
    }
};
```

25

Lab: Car 클래스 작성



26

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;
    }
};
```

27

solution

```
int main()
{
    Car myCar;

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

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

    return 0;
}
```

28

4.5 멤버 함수 중복 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;
}
```

29

실행 결과



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The output of the program is displayed as follows:

```
1
3.14
C++14 is cool.
No Data!
계속하려면 아무 키나 누르십시오 . . .
```

4.6 클래스의 인터페이스와 구현의 분리

- 복잡한 클래스인 경우에는 멤버 함수를 클래스 외부에서 정의

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

class Circle {
public:
    double calcArea();

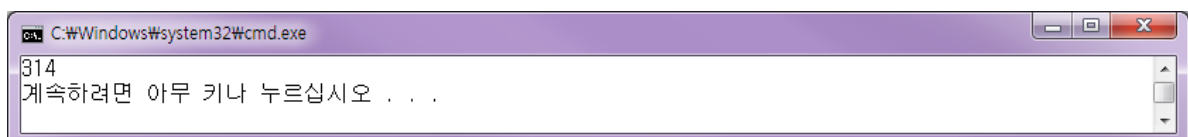
    int radius;      // 반지름
    string color;    // 색상
};
```

31

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;
}
```



32

이름 공간

- 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;
```

33

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;
}
```

34

클래스의 선언과 클래스의 정의 분리

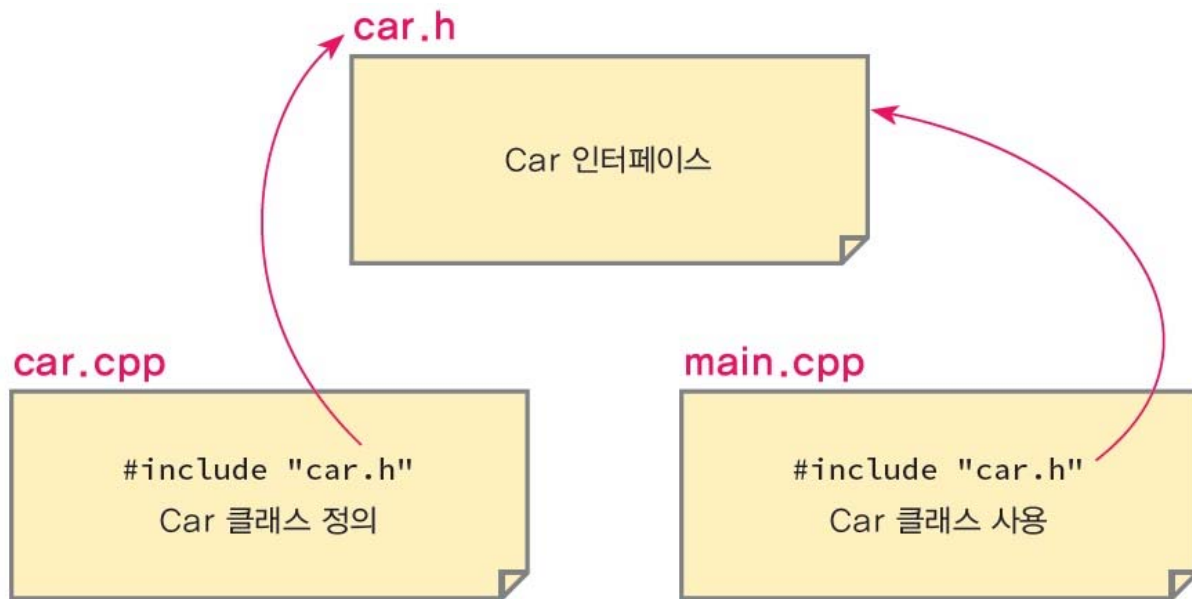


그림 4.11 클래스를 헤더 파일과 소스 파일로 분리

35

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);
};
```

36

car.cpp

```
#include "car.h"

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

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

37

main.cpp

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

int main()
{
    Car myCar;

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

    return 0;
}
```



38

- 2차원 공간에서 하나의 점을 나타내는 **Point** 클래스를 작성하라.

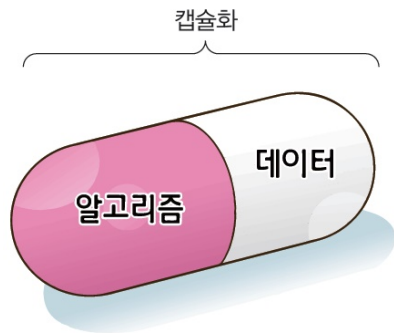
```
class Point {  
    int x, y;  
public:  
    int getX() // 접근자  
    { ...; }  
    void setX(int x) // 설정자  
    { ...; }  
};
```

- 위의 빈칸을 채우고 멤버 변수 **y**에 대해서도 접근자와 설정자를 추가하라.
- 접근자와 설정자를 클래스 외부에 정의하라.
- **p1** 이라는 이름의 객체를 생성하고 **p1**을 통하여 설정자를 호출하여 **p1**의 **x**좌표를 100, **y** 좌표를 200으로 설정하라.

4.7 객체 지향의 개념들

- 캡슐화
 - 정보은닉
- 상속
- 다형성

캡슐화



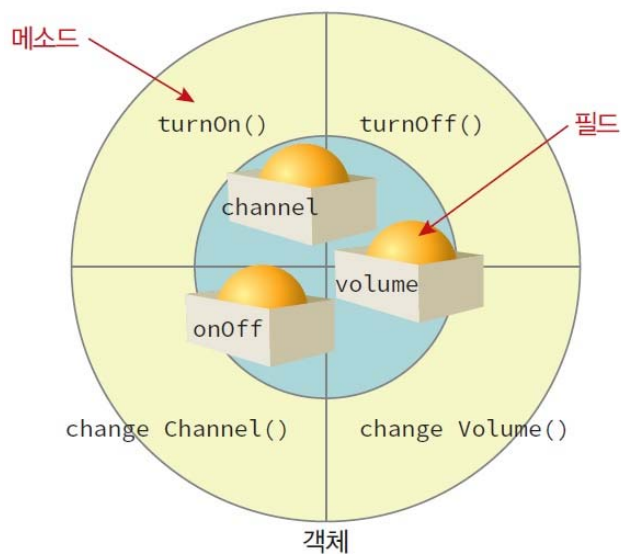
캡슐화는 데이터와 알고리즘을 하나로 묶는 것입니다.



캡슐화 되어 있지 않은 데이터와 코드는 사용하기 어렵다.



정보 은닉



보통은 데이터들은 공개되지 않고 몇 개의 메소드 만이 외부로 공개됩니다.



상속

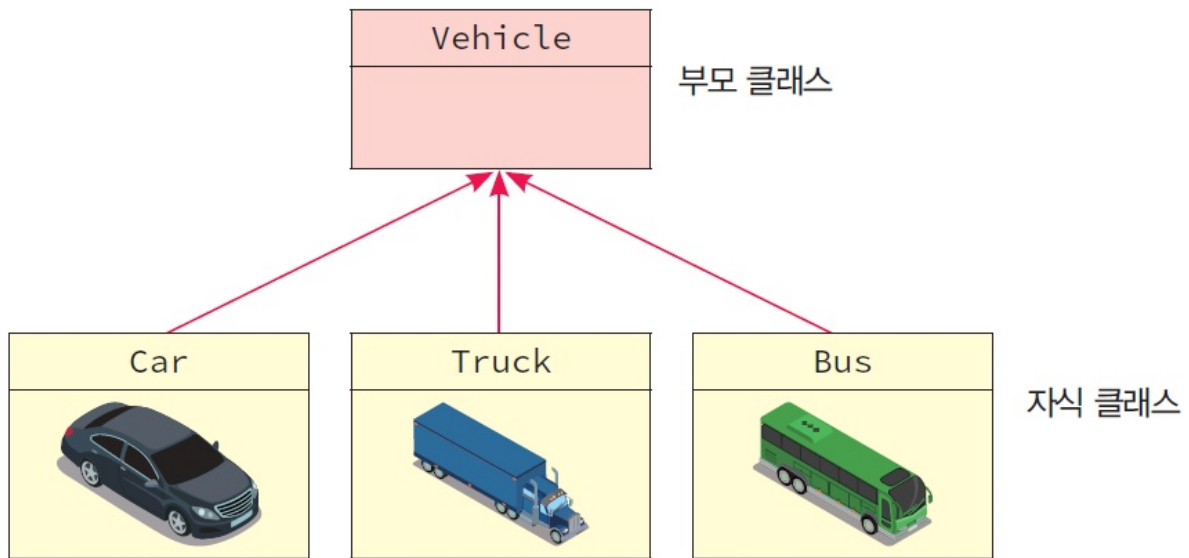
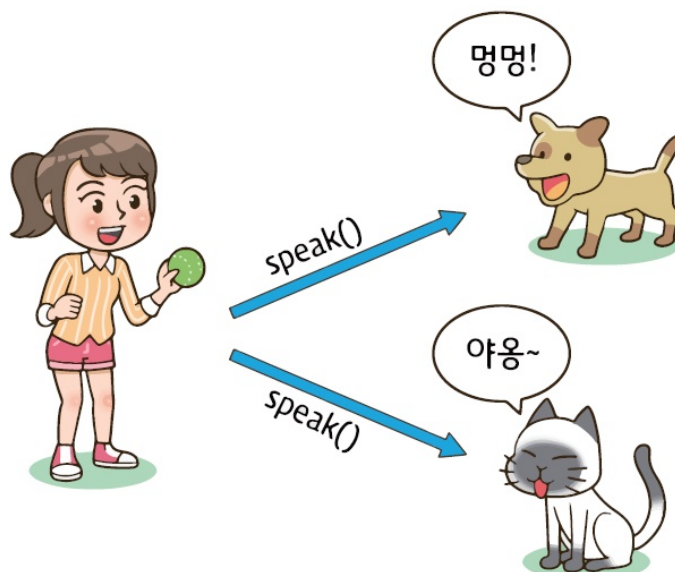


그림 4.12 상속의 개념

43

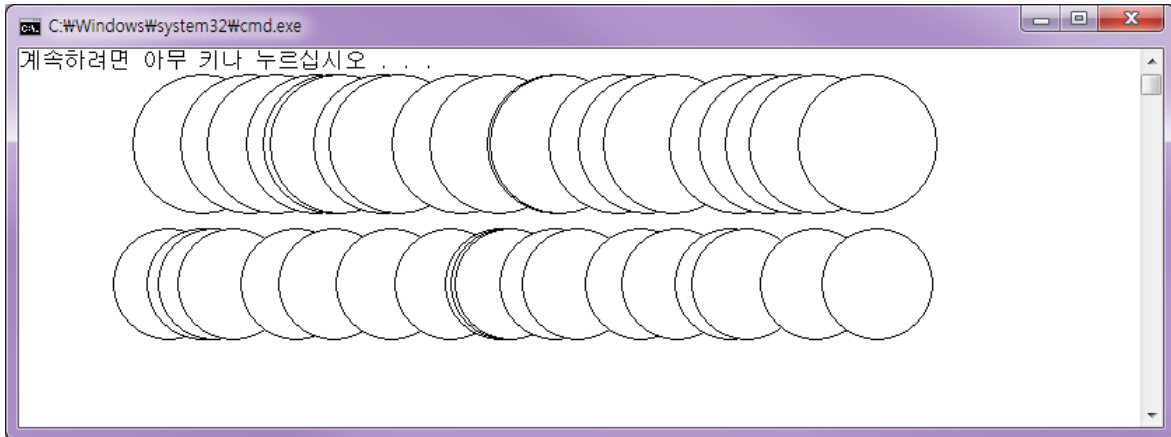
다형성



44

Lab: 원들의 경주

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



45

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;
}
```

46

```

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;
}

```

47

실습

- 직원을 나타내는 **Employee** 클래스를 다음의 단계에 따라서 작성하라.
 1. **name, age, salary, years** 등을 멤버 변수로 가지는 클래스를 작성한다.
 2. 모든 멤버 변수를 **private**로 정의하고 접근자와 설정자를 제공하라.
 3. **Employee** 클래스의 객체를 생성하고 설정자를 통하여 {"홍길동", 26, 1000000, 1}로 설정하고 접근자를 이용하여서 멤버 변수의 값을 화면에 출력한다.

4.8 UML

- 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.



49

UML

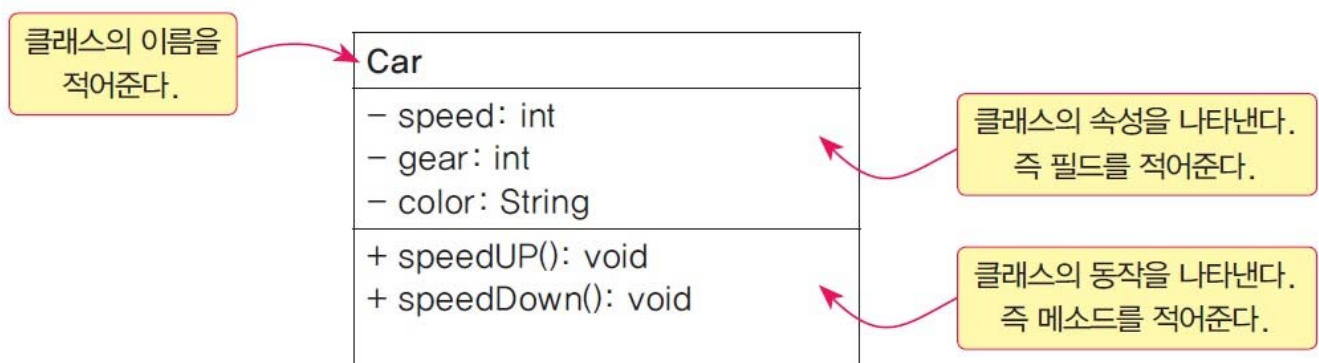


그림 4.13 UML의 예

50

UML

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

그림 4.14 UML에서 사용되는 화살표의 종류

UML의 예

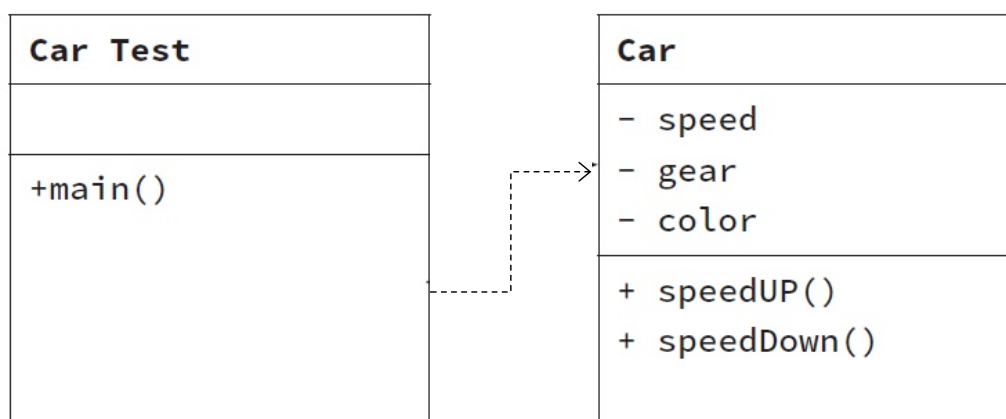


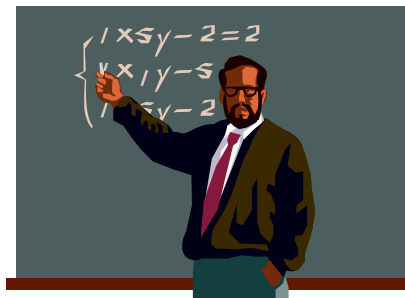
그림 4.15 Car 예제의 UML

과제

- Programming Exercise 10 (교재 214쪽)
 - ▣ eclass에 제출
 - ▣ 제출물: 소스코드, 실행 캡처 화면

53

Q & A



54