4장 클래스와 객체

2020. 9. 14

순천향대학교 컴퓨터 공학과

세상의 모든 것이 객체이다.

- 세상 모든 것이 객체


TV


의자

책


집


카메라


컴퓨터

객체지향 프로그래밍

- **객체 지향 프로그래밍(OOP: object-oriented programming)**이란?

- 우리가 살고 있는 실세계가 객체(object)들로 구성되어 있는 것과 같이 소프트웨어도 객체로 구성하는 방법이다.

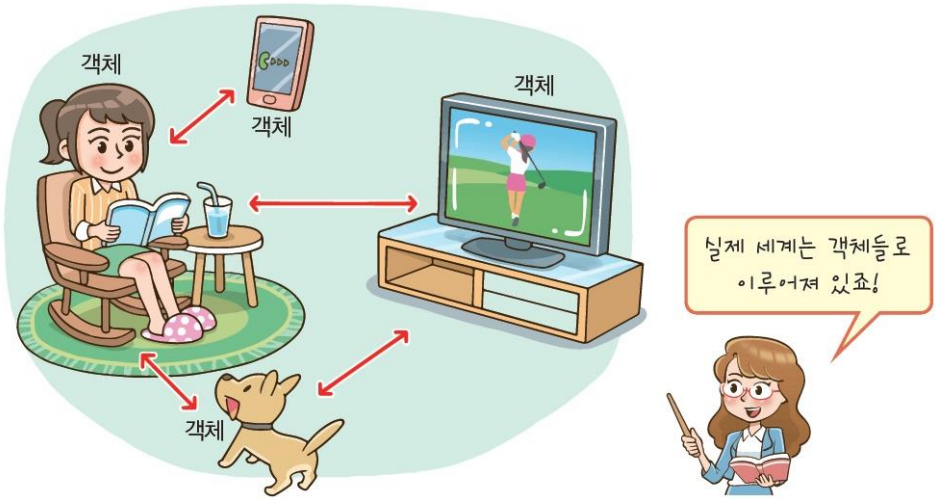


그림 4.1 실제 세계는 객체들로 이루어진다.

객체와 메시지

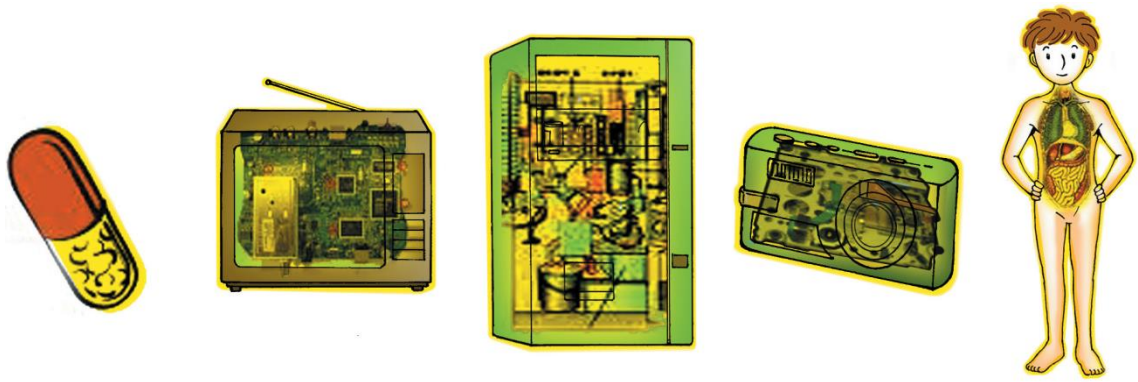
- 객체들은 메시지를 주고 받으면서 상호작용한다.

그림 4.2 객체들은 서로 메시지를 주고받으면서 상호작용한다.

객체는 캡슐화된다.

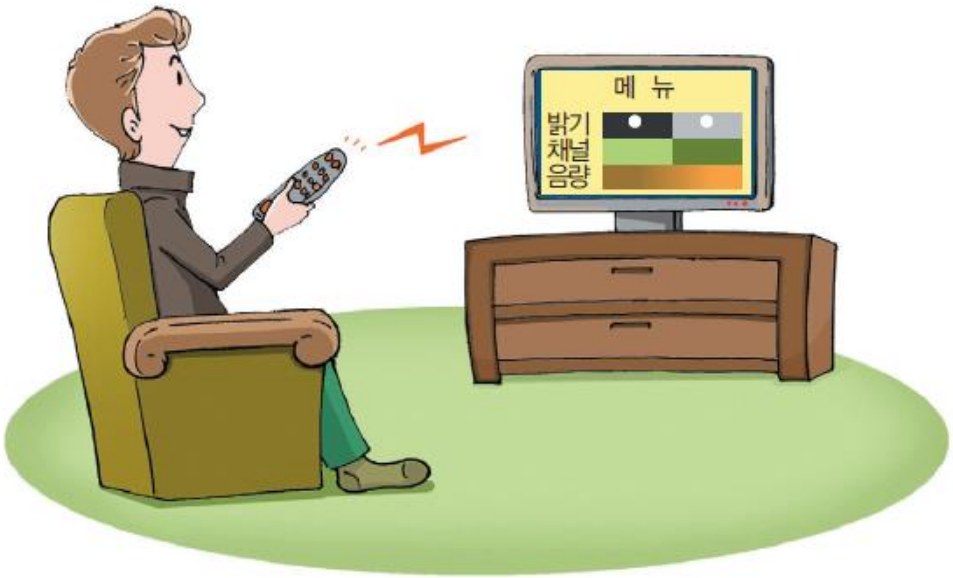
- 캡슐화(encapsulation)
  - 객체의 본질적인 특성
  - 객체를 캡슐로 싸서 그 내부를 보호하고 볼 수 없게 함

- 캡슐화의 목적
  - 객체 내 데이터에 대한 보안, 보호, 외부 접근 제한



객체에 대한 인터페이스 제공

- 객체 인터페이스
  - 외부와의 인터페이스(정보 교환 및 통신)를 위해 객체의 일부분 공개
  - TV 객체의 경우, On/Off 버튼, 밝기 조절, 채널 조절, 음량 조절 버튼 노출. 리모콘 객체와 통신하기 위함



객체의 구성

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

그림 4.7 자동차 객체의 예

C++ 객체의 멤버 변수와 멤버 함수

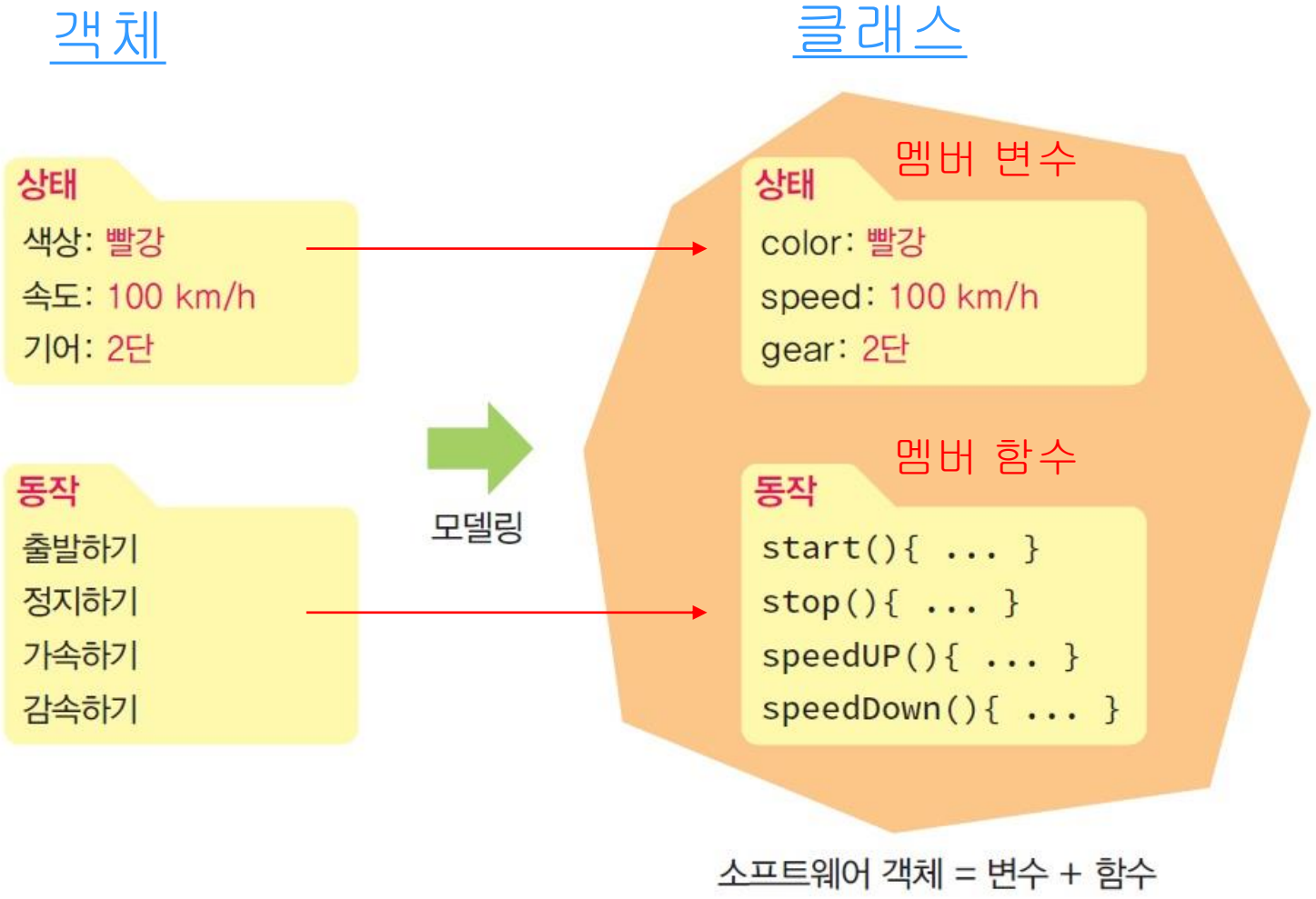


그림 4.8 멤버 변수와 멤버 함수

클래스

- 클래스(class)는 객체 설계 도구

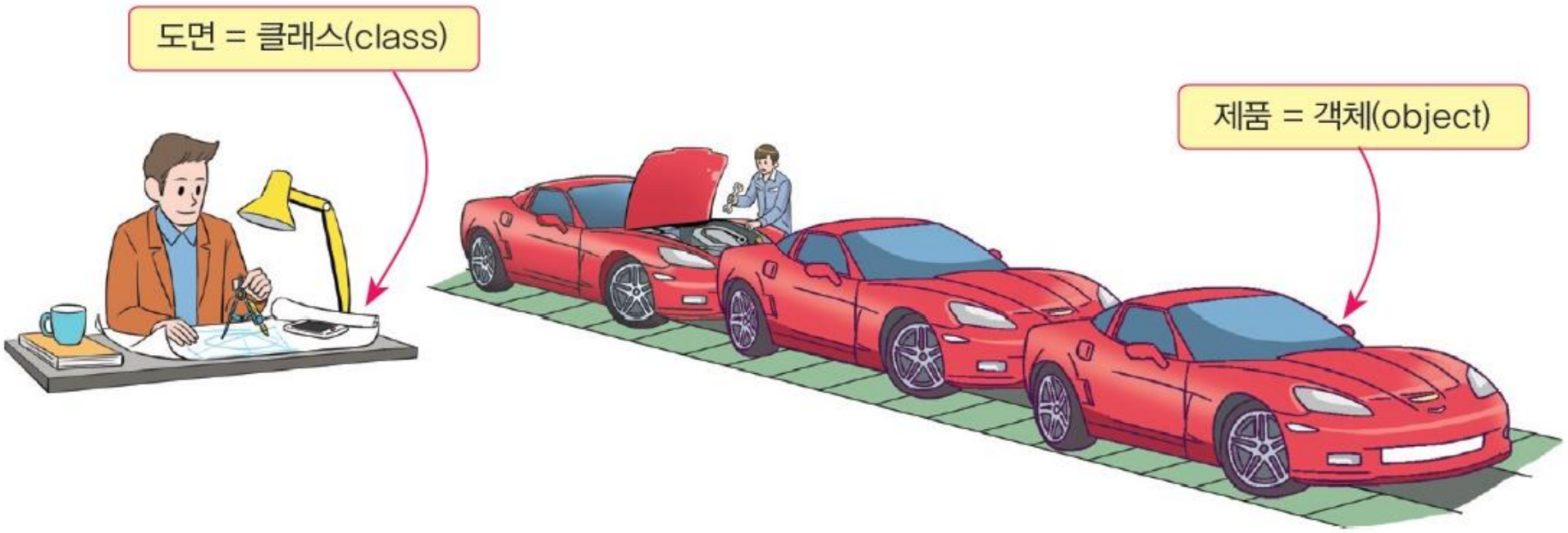


그림 4.9 객체를 클래스라는 설계도로 생성된다.

클래스와 객체

- 클래스
  - 객체를 만들어내기 위해 정의된 설계도, 틀
  - 클래스는 객체가 아님. 실체도 아님
  - 멤버 변수와 함수 선언

- 객체
  - 객체는 생성될 때 클래스의 모양을 그대로 가지고 탄생
  - 멤버 변수와 함수로 구성
  - 메모리에 생성, 실체(instance)라고도 부름
  - 클래스에서 여러 개의 객체 생성 가능
  - 객체들은 상호 별도의 공간에 생성

C++ 클래스 구조

문법 5.1 클래스 정의

```
class 클래스이름 {
    자료형 멤버변수1;
    자료형 멤버변수2;

    반환형 멤버함수1();
    반환형 멤버함수2();
};
```

멤버 변수

멤버 함수 선언부

클래스 작성 예

class 키워드로 클래스 선언

클래스 이름

```
class Circle {
public:
    int radius;
    string color;

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

접근 지정자

멤버 변수

멤버 함수

접근 지정자

- private 멤버는 클래스 내부에서만 접근(사용) 가능
- public 멤버는 클래스 내부와 외부에서 접근 가능
- protected 멤버는 클래스 내부와 상속된 클래스에서 접근 가능
- 디폴트는 private로 지정

- Java와의 차이점은?

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

* Java에서 객체 생성은?
그 차이점은?

객체의 멤버 접근

- 멤버에 접근하기 위해서는 도트(.) 연산자 사용

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

obj 객체의

radius 멤버 변수에 접근

```
class Circle {
public:
    int radius;
    string color;

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

```
#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;         // 객체 생성

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

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

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

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

```
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;
}
```

Lab: Car 클래스 작성

- 다음의 카를 표현하는 Car 클래스를 작성하라.

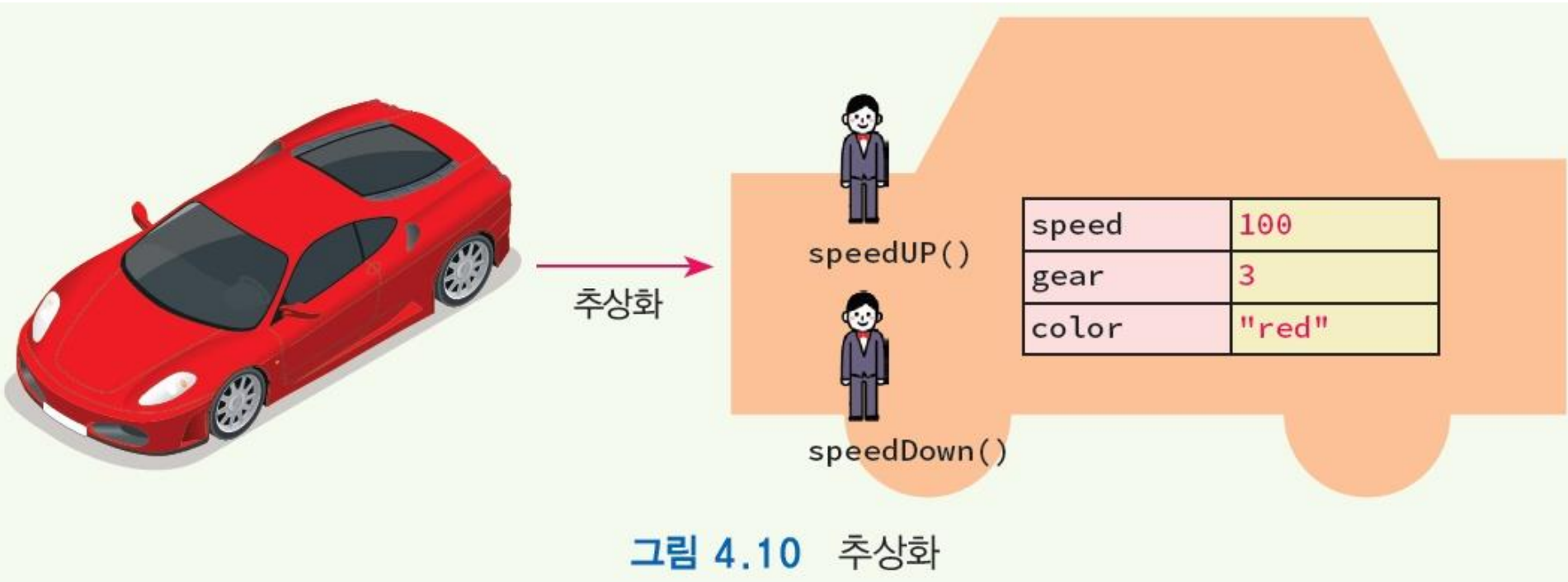


그림 4.10 추상화

```
#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  함수
                speed += 10;
        }

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

```
int main()
{
        Car myCar;

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

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

        return 0;
}
```

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;
}
```

클래스의 선언과 클래스의 정의 분리

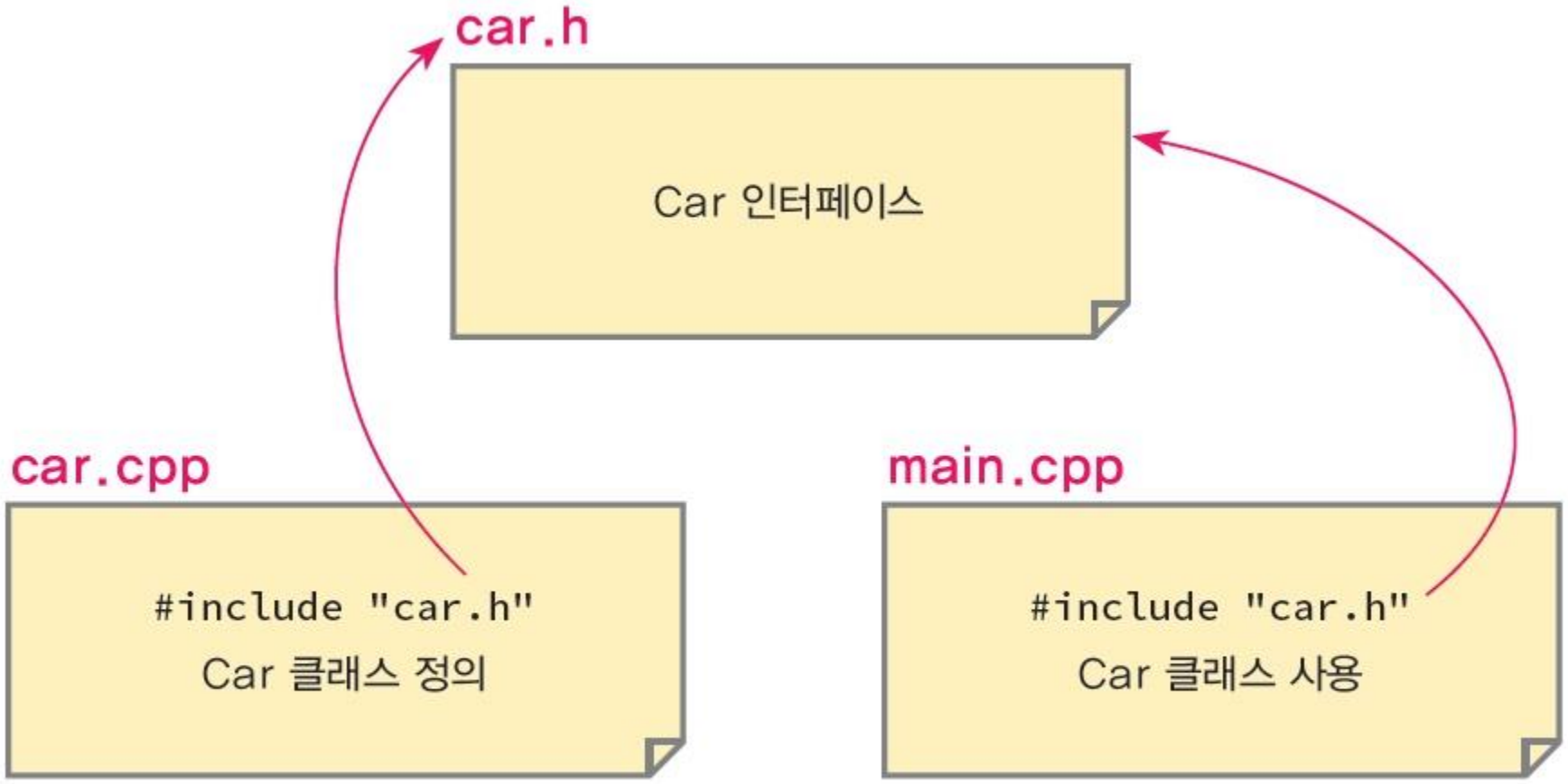


그림 4.11 클래스를 헤더 파일과 소스 파일로 분리

car.cpp

car.h

main.cpp

함수가 클래스 외부에 작성 가능

```
#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);
};
```

```
#include "car.h"

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

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

int main() {
   Car myCar;

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

   return 0;
}
```

예제: 클래스 작성


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

class Circle {
public:
   int radius;
   double getArea();
};

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

```
int main() {
   Circle donut;
   donut.radius = 1;
   double area = donut.getArea();
   cout << "donut 면적은 " << area << endl;

   Circle pizza;
   pizza.radius = 30;
   area = pizza.getArea();
   cout << "pizza 면적은 " << area << endl;
}
```

C++ 클래스 작성 (정리) (1)

- 클래스 작성
  - 멤버 변수와 멤버 함수로 구성
  - 클래스 선언부와 클래스 구현부로 구성하여 작성

- 클래스 구현부(class implementation)
  - 클래스에 정의된 모든 멤버 함수 구현

C++ 클래스 작성 (정리) (2)

- 클래스 선언부(class declaration)
  - class 키워드를 이용하여 클래스 선언
  - 멤버 변수와 멤버 함수 선언
    - 멤버 변수는 클래스 선언 내에서 초기화할 수 없음
    - 멤버 함수는 원형(prototype) 형태로 선언
  - 멤버에 대한 접근 권한 지정
    - private, public, protected 중의 하나
    - public : 다른 모든 클래스나 객체에서 멤버의 접근 가능
    - 디폴트는 private

바람직한 C++ 프로그램 작성법

- 클래스를 헤더 파일과 cpp 파일로 분리하여 작성
  - 클래스 선언 부
    - 헤더 파일(.h)에 저장
  - 클래스 구현 부
    - cpp 파일에 저장
    - 클래스가 선언된 헤더 파일 include
  - main() 등 전역 함수나 변수는 다른 cpp 파일에 저장
    - 필요하면 클래스가 선언된 헤더 파일 include
- 클래스 재사용이 용이함

실습

- 다음 프로그램을 헤더파일과 cpp 파일로 분리하여 작성하라.

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

class Circle {
private:
   int radius;
public:
   Circle();
   Circle(int r);
   double getArea();
};

Circle::Circle() {
   radius = 1;
   cout << "반지름 " << radius;
   cout << " 원 생성" << endl;
}

Circle::Circle(int r) {
   radius = r;
   cout << "반지름 " << radius;
   cout << " 원 생성" << endl;
}
```

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

int main() {
   Circle donut;
   double area = donut.getArea();
   cout << "donut 면적은 ";
   cout << area << endl;

   Circle pizza(30);
   area = pizza.getArea();
   cout << "pizza 면적은 ";
   cout << area << endl;
}
```

인라인 함수

- 자동 인라인 함수 : 클래스 선언부에 구현된 멤버 함수
  - inline으로 선언할 필요 없음
  - 컴파일러에 의해 자동으로 인라인 처리

```
class Circle {
private:
   int radius;
public:
   Circle() { // 자동 인라인 함수
      radius = 1;
   }

   Circle(int r);
   double getArea() { // 자동 인라인 함수
      return 3.14*radius*radius;
   }
};

Circle::Circle(int r) {
   radius = r;
}
```

=

```
class Circle {
private:
   int radius;
public:
   Circle();
   Circle(int r);
   double getArea();
};

inline Circle::Circle() {
   radius = 1;
}

Circle::Circle(int r) {
   radius = r;
}

inline double
Circle::getArea() {
   return
3.14*radius*radius;
}
```

UML

- **UML(Unified Modeling Language)**은 객체 지향 프로그래밍 설계 도구
  - 애플리케이션을 구성하는 클래스 정의
  - 클래스들간의 관계 표현

클래스 정의

```
class Car
{    int speed;
     int gear;
     string color;
public:
     int getSpeed() {..}
     void setSpeed(int s) {..}
};
```

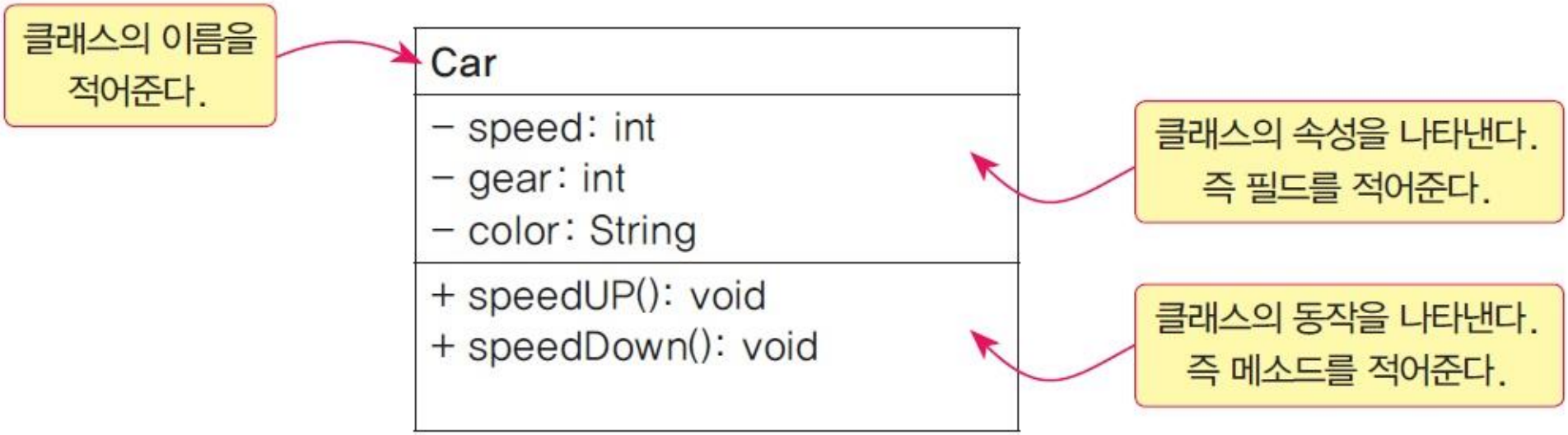


그림 4.13 UML의 예

클래스 관계 표현

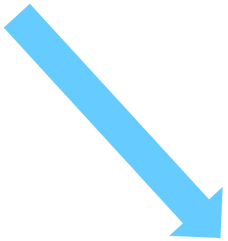
관계	화살표
일반화(generalization), 상속(inheritance)	
구현(realization)	
구성관계(composition)	
집합관계(aggregation)	
유향 연관(direct association)	
양방향 연관(bidirectional association)	
의존(dependency)	

그림 4.14 UML에서 사용되는 화살표의 종류

클래스 관계 표현 예

CarTest.cpp

```
int main() {
  Car myCar;
  myCar.speed = 30;
  myCar.speedup();
}
```



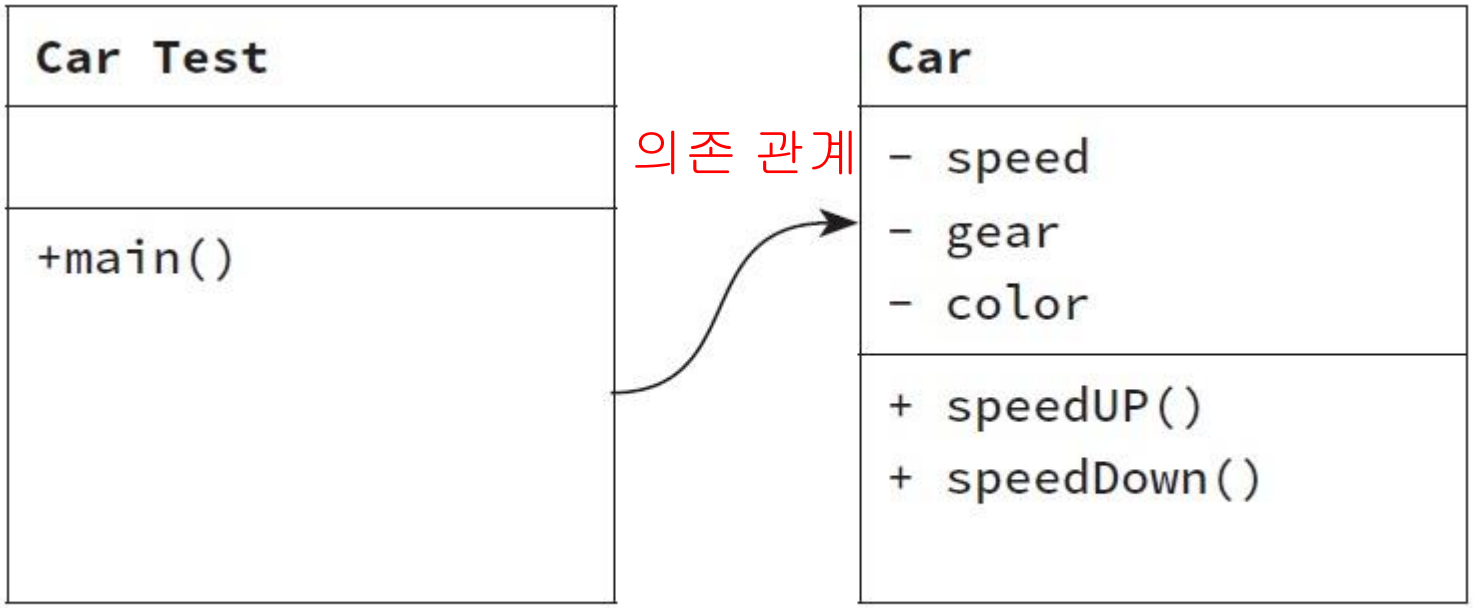


그림 4.15 Car 예제의 UML