Chapter 02. 클래스의 기본

박 종 혁 교수

UCS Lab

Tel: 970-6702

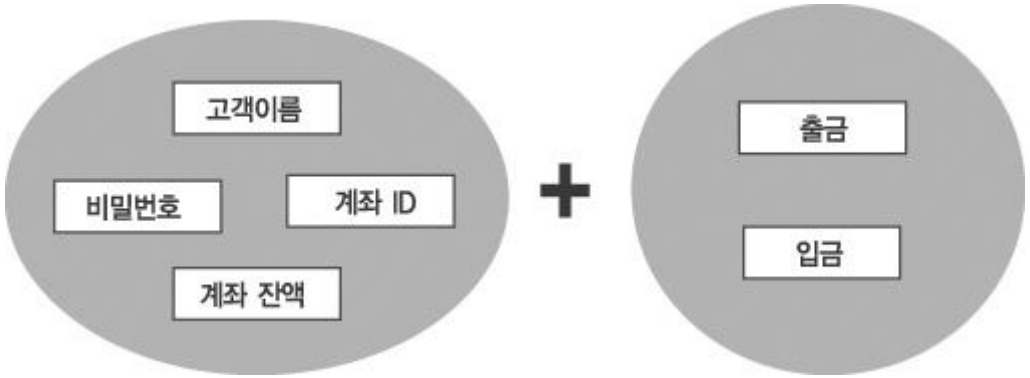
Email: jhpark1@seoultech.ac.kr

객체지향 프로그래밍

- 객체지향 프로그래밍 시각
  - 문제의 영역을 단순한 자료의 처리 흐름으로 보지 않음
    - 구조적 프로그래밍 서로 관련된 자료와 연산(함수)들이 서로 독립적으로 정의되어 취급
  - 문제 영역 내에 존재하는 여러 연관된 객체들을 정의하고 이들 객체들이 서로 정보를 주고 받는다고 보는 시각 (객체 간의 관계)
- 객체지향 프로그래밍에서는 프로그램은 여러 개의 객체로 구성
  - 객체(object)는 자료(특성(attribute))와 이를 대상으로 처리하는 동작인 연산(함수,메쏘드(method))을 하나로 묶어 만든 요소로 프로그램을 구성하는 실체
  - 객체란 단순히 자료를 표현하는 변수 만을 가지는 것이 아니라 그 객체가 무엇을 할 수 있는가를 정의한 함수(메쏘드)로 구성

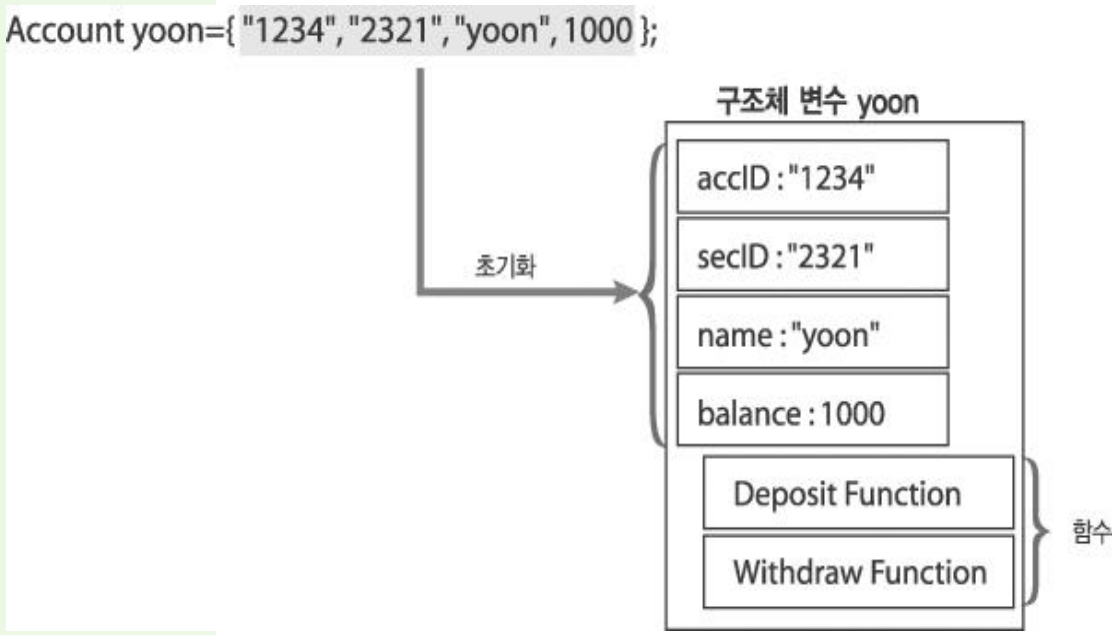
구조체 + 함수

- 함수를 넣으면 좋은 구조체
  - 프로그램=데이터+데이터 조작 루틴(함수)
  - 잘 구성된 프로그램은 데이터와 더불어 함수들도 그룹화
  - 객체지향 프로그래밍 기법



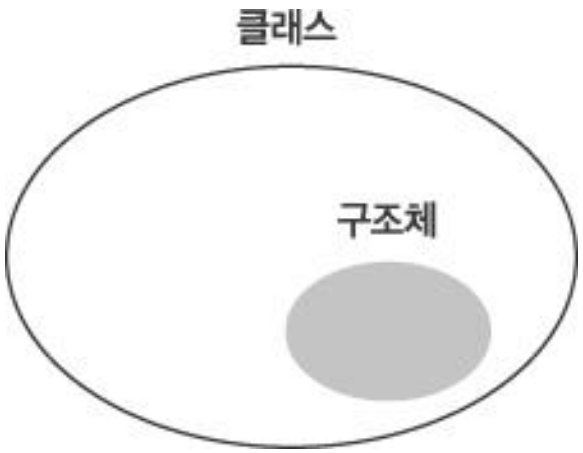
구조체 + 함수 예

```
struct Account {
      char accID[20];
      char secID[20];
      char name[20];
      int balance;

      void Deposit(int money){
          balance+=money;
      }
      void Withdraw(int money){
          balance-=money;
      }
};

int main(void)
{
    Account yoon={"1234", "2321", "yoon", 1000};
    yoon.Deposit(100);
    cout<<"잔     액 : "<<yoon.balance<<endl;
    return 0;
}
```



클래스

- 객체지향 프로그래밍에서는 구조체가 아니라 클래스(class)
  - 클래스 = 멤버 변수 + 멤버 함수
  - 변수가 아니라 객체(object)



데이터 추상화와 클래스

- 사물의 관찰 이후의 데이터 추상화
  - 현실 세계의 사물을 데이터적인 측면과 기능적인 측면을 통해서 정의하는 것

예제) 코끼리를 자료형으로 정의
→ 특징들을 뽑아내야한다.

1. 발이 네 개 →데이터
2. 코의 길이가 5미터 내외 →데이터
3. 몸무게는 1톤 이상 →데이터
4. 코를 이용해서 목욕을 함 →기능
5. 코를 이용해서 물건을 집기도 함 →기능

프로그램

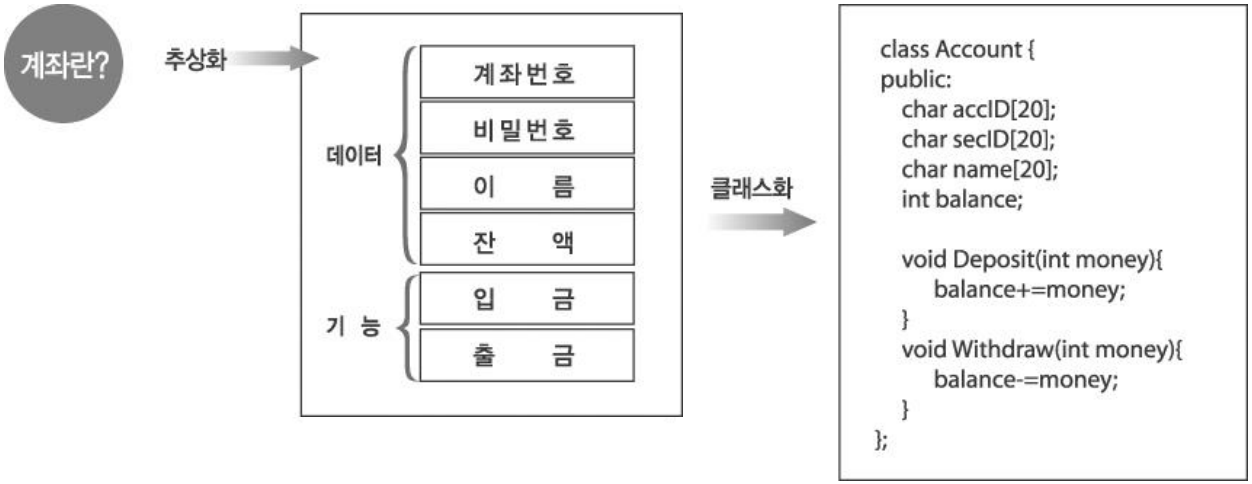
변수

함수

데이터 추상화

데이터 추상화와 클래스

- 데이터 추상화 이후의 클래스화
  - 추상화된 데이터를 가지고 사용자 정의 자료형을 정의하는 것



객체 생성

- 선언된 클래스를 사용하여 객체를 생성
  - 클래스화 이후의 인스턴스화(instantiation)

```
class Account {
public:
   char accID[20];
   char secID[20];
   char name[20];
   int balance;

   void Deposit(int money){
       balance+=money;
   }
   void Withdraw(int money){
       balance-=money;
   }
};
```

인스턴스화

```
int main(void
{
   Account yoon={"1234", "2321", "yoon", 1000};
   .....
}
```

클래스 선언 형식

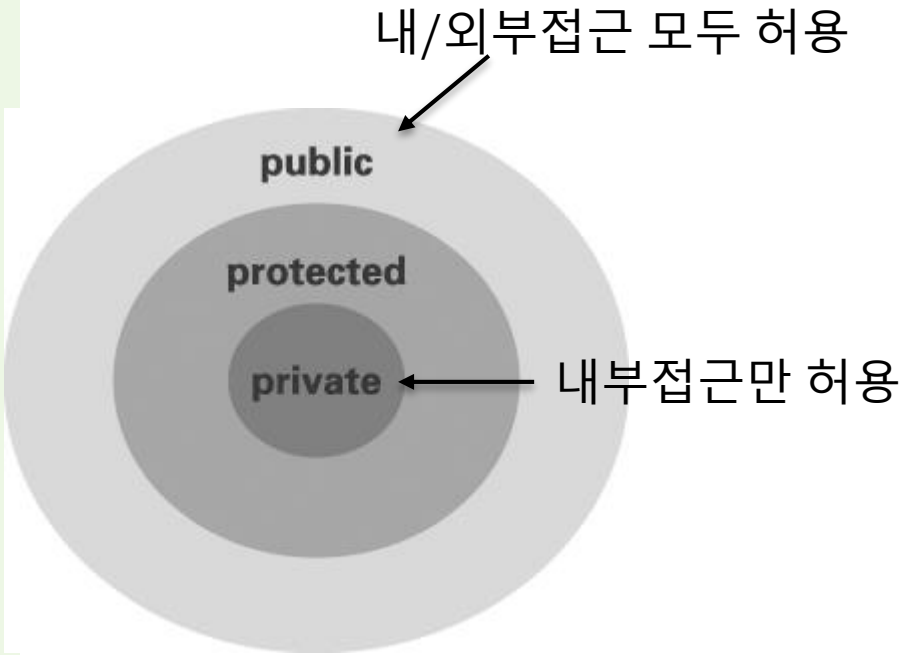
- C++ 언어에서 클래스의 정의는 C 언어의 구조체(struct) 선언과 비슷
  - 다른 점은 클래스의 멤버로서 자료 뿐만 아니라 연산을 위한 함수도 포함된다는 점

```
class 클래스명 {
[private:]
        자료 선언;
        함수 선언;
public:
        자료 선언;
        함수 선언;
} [객체 변수];
클래스명 객체변수, ……;
```

```
class Test {
 private:
        int a;
        void inc();
public:
        char s[10];
} var;

Test t1, t2;
```

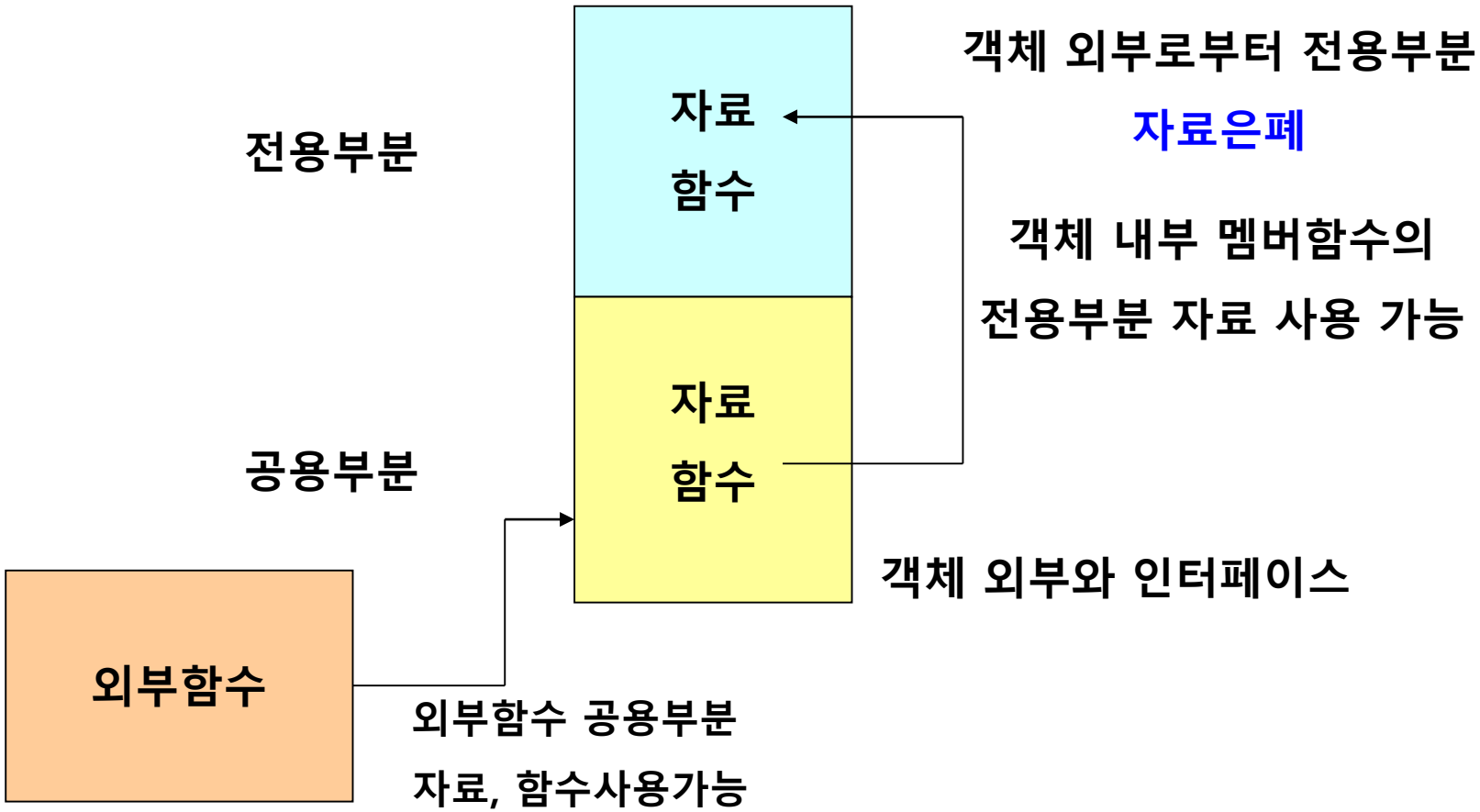
클래스 멤버의 접근 제어

```
const int OPEN=1;
const int CLOSE=2;

class Door{
private:
    int state;
public:
    void Open(){ state=OPEN; }
    void Close(){ state=CLOSE; }
    void ShowState(){ ...생략...  }
};

int main()
{
    Door d;
    //d.state=OPEN;
    d.Open();
    d.ShowState();
    return 0;
}
```



C++의 접근 제어 키워드 : 접근의 범위

클래스 영역


전용부분
자료
함수
객체 외부로부터 전용부분
자료은폐
객체 내부 멤버함수의
전용부분 자료 사용 가능
공용부분
자료
함수
객체 외부와 인터페이스
외부함수
외부함수 공용부분
자료, 함수사용가능

클래스 영역

- 클래스를 정의할 때 private과 public 영역으로 구분

- private은 전용부분
  - private 영역에서 정의된 자료형이나 함수는 오직 해당 객체 내부의 멤버함수만이 사용
  - 외부에 대해 자료의 정보가 은폐
  - 전용부분에는 함부로 변경되어서는 안될 자료와 객체 외부에서 호출되어서는 안될 멤버함수를 정의

- public은 공용부분
  - 객체 외부에서 사용될 수 있는 자료나 함수가 정의
  - 클래스 정의 시, 접근제어 키워드를 생략하면 private으로 간주
- 클래스 영역으로는 위 두 영역 외에 protected로 구분되는 보호부분이 있는 파생 클래스를 설명할 때 소개

클래스 멤버함수의 선언

- 클래스의 멤버로서 자료 뿐만 아니라 함수도 정의
  - 함수의 본체 내용은 직접 클래스 내부나 또는 클래스 외부에서 기술
  - 클래스 내부에서 함수의 본체가 기술된 멤버함수는 모두 인라인 함수(inline function)로 정의

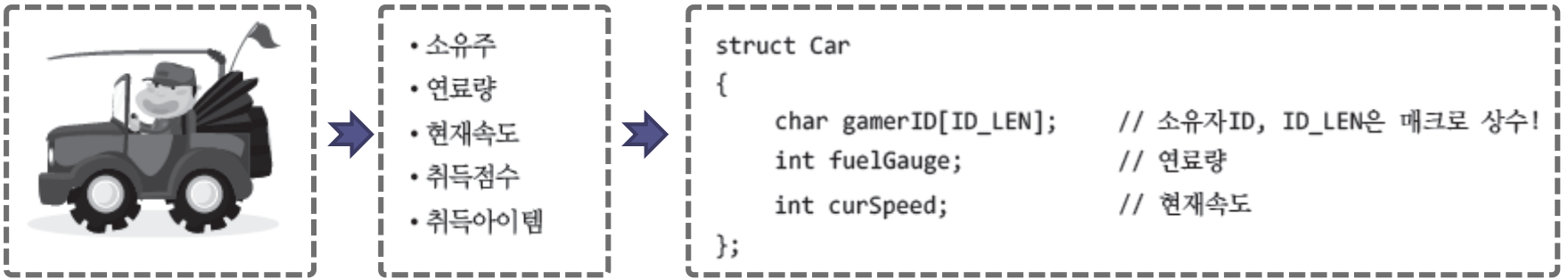
```
class 클래스명 {
        ……
        자료형 함수명(인수 선언) { 함수 본체 }
        ……
}
```

C++에서의 구조체

C++에서의 구조체

구조체의 등장배경

- 연관 있는 데이터를 하나로 묶으면 프로그램의 구현 및 관리가 용이하다.
- 구조체는 연관 있는 데이터를 하나로 묶는 문법적 장치이다.



연관 있는 데이터들은 생성 및 소멸의 시점이 일치하고, 이동 및 전달의 시점 및 방법이 일치하기 때문에 하나의 자료형으로 묶어서 관리하는 것이 용이하다.

C++에서의 구조체 변수 선언

```
struct Car basicCar;
struct Car simpleCar;
```

```
Car basicCar;
Car simpleCar;
```

따라서 C++에서는 구조체 변수 선언시 struct 키워드의 생략을 위한 typedef 선언이 불필요하다.

C 스타일 구조체 변수 초기화　　C++ 스타일 구조체 변수 초기화

```
struct Car
{
    char gamerID[ID_LEN];   // 소유자ID
    int fuelGauge;          // 연료량
    int curSpeed;           // 현재속도
};
```

Car와 관련된 연관된 데이터들의 모임

```
void ShowCarState(const Car &car)
{
    . . . .
}
void Accel(Car &car)
{
    . . . .
}
void Break(Car &car)
{
    . . . .
}
```

Car와 관련된 연관된 함수들의 모임

데이터 뿐만 아니라, 해당 데이터와 연관된 함수들도 함께 그룹을 형성하기 때문에 함수도 하나로 묶는 것에 대해 나름의 가치를 부여할 수 있다.

구조체 안에 함수 삽입하기

```
struct Car
{
    char gamerID[ID_LEN];
    int fuelGauge;
    int curSpeed;

    void ShowCarState()
    {
        . . . .
    }
    void Accel()
    {
        . . . .
    }
    void Break()
    {
        . . . .
    }
};
```

C++에서는 구조체 안에 함수를 삽입하는 것이 가능하다.

따라서 C++에서는 구조체가 아닌, 클래스라 한다.

```
void ShowCarState()
{
    cout<<"소유자ID: "<<gamerID<<endl;    // 위에 선언된 gamerID에 접근
    cout<<"연료량: "<<fuelGauge<<"%"<<endl;
    cout<<"현재속도: "<<curSpeed<<"km/s"<<endl<<endl;
}
```

```
void Break()
{
    if(curSpeed<BRK_STEP)
    {
        curSpeed=0;    // 위에 선언된 curSpeed에 접근
        return;
    }

    curSpeed-=BRK_STEP;
}
```

함께 선언된 변수에는 직접 접근이 가능하다.

예제 RacingCar

```
#include <iostream>
using namespace std;
#define ID_LEN 20
#define MAX_SPD 200
#define FUEL_STEP 2
#define ACC_STEP 10
#define BRK_STEP 10
struct Car
{
	char gamerID[ID_LEN];
	int fuelGauge;
	int curSpeed;
};
void ShowCarState(const Car &car)
{
	cout<<"소유자ID: "<<car.gamerID<<endl;
	cout<<"연료량: "<<car.fuelGauge<<"%"<<endl;
	cout<<"현재속도: "<<car.curSpeed<<"km/s"<<endl<<endl;
}
void Accel(Car &car)
{
	if(car.fuelGauge<=0)
		return;
	else
		car.fuelGauge-=FUEL_STEP;
	if(car.curSpeed+ACC_STEP>=MAX_SPD)
	{
		car.curSpeed=MAX_SPD;
		return;
	}
	car.curSpeed+=ACC_STEP;
}
```

```
void Break(Car &car)
{
	if(car.curSpeed<BRK_STEP)
	{
		car.curSpeed=0;
		return;
	}
		car.curSpeed-=BRK_STEP;
}

int main(void)
{
	Car run99={"run99",100,0};
	Accel(run99);
	Accel(run99);
	ShowCarState(run99);
	Break(run99);
	ShowCarState(run99);

	Car sped77{"sped77",100,0};
	Accel(sped77);
	Break(sped77);
	ShowCarState(sped77);
	return 0;
}
```

C++에서의 구조체 변수 선언

변수의 생성

```
Car run99={"run99", 100, 0};

Car sped77={"sped77", 100, 0};
```

실제로는 구조체 변수마다 함수가 독립적으로 존재하는 구조는 아니다. 그러나 논리적으로는 독립적으로 존재하는 형태로 보아도 문제가 없으니, 위의 그림의 형태로 변수(객체)를 이해하자!

예제 RacingCarFuncAdd

```
#include <iostream>
using namespace std;
#define ID_LEN          20
#define MAX_SPD         200
#define FUEL_STEP       2
#define ACC_STEP        10
#define BRK_STEP        10
struct Car
{
 char gamerID[ID_LEN];     // 소유자ID
 int fuelGauge;                          // 연료량
 int curSpeed;             // 현재속도
 void ShowCarState()
 {
  cout<<"소유자ID: "<<gamerID<<endl;
  cout<<"연료량: "<<fuelGauge<<"%"<<endl;
  cout<<"현재속도: "<<curSpeed<<"km/s"<<endl<<endl;
 }
 void Accel()
 {
  if(fuelGauge<=0)
   return;
  else
   fuelGauge-=FUEL_STEP;
  if(curSpeed+ACC_STEP>=MAX_SPD)
  {
   curSpeed=MAX_SPD;
   return;
  }
  curSpeed+=ACC_STEP;
 }
 void Break()
 {
  if(curSpeed<BRK_STEP)
  {
   curSpeed=0;
   return;
  }
  curSpeed-=BRK_STEP;
 }
};
int main(void)
{
  Car run99={"run99", 100, 0};
  run99.Accel();
  run99.Accel();
  run99.ShowCarState();
  run99.Break();
  run99.ShowCarState();
  Car sped77={"sped77", 100, 0};
  sped77.Accel();
  sped77.Break();
  sped77.ShowCarState();
  return 0;
}
```

함수는 외부로 뺄 수 있다.

```
struct Car
{
    . . . . .
    void ShowCarState();
    void Accel();
    . . . . .
};
```

구조체 안에 삽입된 함수의 선언!

```
void Car::ShowCarState()
{
    . . . . .
}
void Car::Accel()
{
    . . . . .
}
```

구조체 밖에 선언된 함수의 정의!

구조체 안에 정의된 함수는 inline 선언된 것으로 간주

따라서 필요하다면,

함수의 정의를 외부로 뺄 때에는 다음과 같이 명시적으로 inline 선언을 해야 한다.

```
inline void Car::ShowCarState() { . . . . . }
inline void Car::Accel() { . . . . }
inline void Car::Break() { . . . . }
```

클래스(Class)와 객체(Object)

클래스와 구조체의 유일한 차이점

키워드 struct를 대신해서 class를 사용한것이 유일한 외형적 차이점이다.

```
class Car
{
    char gamerID[CAR_CONST::ID_LEN];
    int fuelGauge;
    int curSpeed;

    void ShowCarState() { . . . . }
    void Accel() { . . . . }
    void Break() { . . . . }
};
```

```
int main(void)
{
    Car run99;
    strcpy(run99.gamerID, "run99");     (×)
    run99.fuelGauge=100;                (×)
    run99.curSpeed=0;                   (×)
    . . . .
```

왼쪽과 같이 단순히 키워드만 class로 바꾸면 선언된 멤버의 접근이 불가능하다.

따라서 별도의 접근제어와 관련된 선언을 추가해야 한다.

접근제어 지시자

접근제어 지시자

- public 어디서든 접근허용
- protected 상속관계에 놓여있을 때, 유도 클래스에서의 접근허용
- private 클래스 내(클래스 내에 정의된 함수)에서만 접근허용

```
class Car
{
private:
    char gamerID[CAR_CONST::ID_LEN];
    int fuelGauge;
    int curSpeed;
public:
    void InitMembers(char * ID, int fuel);
    void ShowCarState();
    void Accel();
    void Break();
};
```

private!

public!

```
int main(void)
{
    Car run99;
    run99.InitMembers("run99", 100);
    run99.Accel();
    run99.Accel();
    run99.Accel();
    run99.ShowCarState();
    run99.Break();
    run99.ShowCarState();
    return 0;
}
```

Car의 멤버함수는 모두 public이므로 클래스의 외부에 해당하는 main 함수에서 접근가능!

예제 RacingCarClassBase

```
#include <iostream>
#include <cstring>
using namespace std;
namespace CAR_CONST
{
 enum
 {
  ID_LEN=20, MAX_SPD=200, FUEL_STEP=2, ACC_STEP=10,
BRK_STEP=10
 };
}
class Car
{
private:
 char gamerID[CAR_CONST::ID_LEN];
 int fuelGauge;
 int curSpeed;
public:
 void InitMembers(char * ID, int fuel);
 void ShowCarState();
 void Accel();
 void Break();
};
void Car::InitMembers(char * ID, int fuel)
{
 strcpy(gamerID, ID);
 fuelGauge=fuel;
 curSpeed=0;
};
void Car::ShowCarState()
{
 cout<<"소유자ID: "<<gamerID<<endl;
 cout<<"연료량: "<<fuelGauge<<"%"<<endl;
 cout<<"현재속도: "<<curSpeed<<"km/s"<<endl<<endl;
}
void Car::Accel()
{
 if(fuelGauge<=0)
  return;
 else
  fuelGauge-=CAR_CONST::FUEL_STEP;
 if((curSpeed+CAR_CONST::ACC_STEP)>=CAR_CONST::MAX_SPD)
 {
  curSpeed=CAR_CONST::MAX_SPD;
  return;
 }
 curSpeed+=CAR_CONST::ACC_STEP;
}
void Car::Break()
{
 if(curSpeed<CAR_CONST::BRK_STEP)
 {
  curSpeed=0;
  return;
 }
 curSpeed-=CAR_CONST::BRK_STEP;
}
int main(void)
{
 Car run99;
 run99.InitMembers("run99", 100);
 run99.Accel();
 run99.Accel();
 run99.Accel();
 run99.ShowCarState();
 run99.Break();
 run99.ShowCarState();
 return 0;
}
```

용어정리: 객체(Object), 멤버변수, 멤버함수

```
class Car
{
private:
    char gamerID[CAR_CONST::ID_LEN];
    int fuelGauge;
    int curSpeed;
public:
    void InitMembers(char * ID, int fuel);
    void ShowCarState();
    void Accel();
    void Break();
};
```

왼쪽의 Car 클래스를 대상으로 생성된 변수를 가리켜 '객체'라 한다.

왼쪽의 Car 클래스 내에 선언된 변수를 가리켜 '멤버변수'라 한다.

왼쪽의 Car 클래스 내에 정의된 함수를 가리켜 '멤버함수'라 한다.

C++에서의 파일 분할

```
class Car
{
private:
    char gamerID[CAR_CONST::ID_LEN];
    int fuelGauge;
    int curSpeed;
public:
    void InitMembers(char * ID, int fuel);
    void ShowCarState();
    void Accel();
    void Break();
};
```

클래스의 선언은 일반적으로 헤더파일에 삽입한다. 객체생성문 및 멤버의 접근문장을 컴파일하기 위해서 필요하다.
클래스의 이름을 때서 Car.h로 헤더파일의 이름정의하기도 한다.

단! 인라인 함수는 컴파일 과정에서 함수의 호출문을 대체해야 하기 때문에 헤더파일에 함께 정의되어야 한다

```
void Car::InitMembers(char * ID, int fuel) { . . . . }
void Car::ShowCarState() { . . . . }
void Car::Accel() { . . . . }
void Car::Break() { . . . . }
```

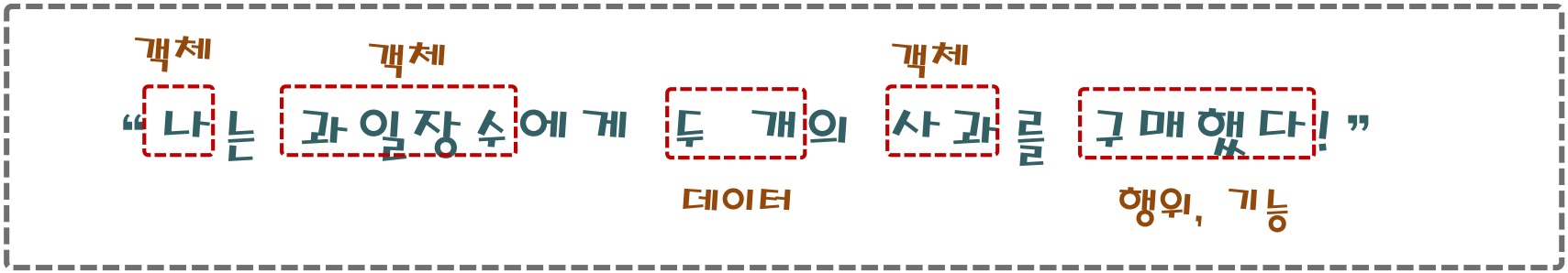
Car 클래스의 멤버함수의 몸체는 다른 코드의 컴파일 과정에서 필요한 게 아니다. 링크의 과정을 통해서 하나의 바이너리로 구성만 되면 된다. 따라서 cpp 파일에 정의하는 것이 일반적이다.
클래스의 이름을 따서 Car.cpp로 소스파일의 이름을 정의하기도 한다.

객체지향 프로그래밍의 이해

객체지향 프로그래밍의 이해

객체에 대한 간단한 정의

- 사전적 의미 물건 또는 대상
- 객체지향 프로그래밍 객체 중심의 프로그래밍



객체지향 프로그래밍에서는 나, 과일장수, 사과라는 객체를 등장시켜서 두 개의 사과 구매라는 행위를 실체화한다.

객체지향 프로그래밍은 현실에 존재하는 사물과 대상, 그리고 그에 따른 행동을 있는 그대로 실체화 시키는 형태의 프로그래밍이다.

객체를 이루는 것은 데이터와 기능입니다.

과일장수 객체의 표현

- 과일장수는 과일을 팝니다. **행위**
- 과일장수는 사과 20개, 오렌지 10개를 보유하고 있습니다. **상태**
- 과일장수의 과일판매 수익은 현재까지 50,000원입니다. **상태**

과일장수의 데이터 표현

- 보유하고 있는 사과의 수 → int numOfApples;
- 판매 수익 → int myMoney;

과일장수의 행위 표현

```
int SaleApples(int money)      // 사과 구매액이 함수의 인자로 전달
{
     int num = money/1000;     // 사과가 개당 1000원이라고 가정
     numOfApples -= num;       // 사과의 수가 줄어들고,
     myMoney += money;         // 판매 수익이 발생한다.
     return num;               // 실제 구매가 발생한 사과의 수를 반환
}
```

이제 남은 것은 데이터와 행위를 한데 묶는 것!

'과일장수'의 정의와 멤버변수의 상수화

```
class FruitSeller
{
private:
    int APPLE_PRICE;
    int numOfApples;
    int myMoney;

public:
    int SaleApples(int money)
    {
        int num=money/APPLE_PRICE;
        numOfApples-=num;
        myMoney+=money;
        return num;
    }
};
```

변수 선언

함수 정의

과일 값은 변하지 않는다고 가정할 때
APPLE_PRICE는 다음과 같이 선언하는 것이 좋다!

```
const int APPLE_PRICE;
```

그러나 상수는 선언과 동시에 초기화 되어야 하기 때문에 이는 불가능하다. 물론 클래스를 정의하는 과정에서 선언과 동시에 초기화는 불가능하다.

추가

```
void InitMembers(int price, int num, int money)
{
    APPLE_PRICE=price;
    numOfApples=num;
    myMoney=money;
}
```

초기화를 위한 추가

추가

```
void ShowSalesResult()
{
    cout<<"남은 사과: "<<numOfApples<<endl;
    cout<<"판매 수익: "<<myMoney<<endl;
}
```

얼마나 파셨어요? 라는 질문과 답변을 위한 함수

'나(me)'를 표현하는 클래스의 정의와 객체생성

'나'의 클래스 정의

```
class FruitBuyer
{
    int myMoney;       // private:
    int numOfApples;   // private:
public:
    void InitMembers(int money)
    {
        myMoney=money;
        numOfApples=0;     // 사과구매 이전이므로!
    }
    void BuyApples(FruitSeller &seller, int money)
    {
        numOfApples+=seller.SaleApples(money);
        myMoney-=money;
    }
    void ShowBuyResult()
    {
        cout<<"현재 잔액: "<<myMoney<<endl;
        cout<<"사과 개수: "<<numOfApples<<endl;
    }
};
```

상태

행위

일반적인 변수 선언 방식의 객체생성

```
FruitSeller seller;

FruitBuyer buyer;
```

동적 할당 방식의 객체생성

```
FruitSeller * objPtr1=new FruitSeller;

FruitBuyer * objPtr2=new FruitBuyer;
```

사과장수 시뮬레이션 완료

```
int main(void)
{
    FruitSeller seller;
    seller.InitMembers(1000, 20, 0);
    FruitBuyer buyer;
    buyer.InitMembers(5000);
    buyer.BuyApples(seller, 2000);

    cout<<"과일 판매자의 현황"<<endl;
    seller.ShowSalesResult();
    cout<<"과일 구매자의 현황"<<endl;
    buyer.ShowBuyResult();
    return 0;
}
```

아저씨 사과 2000원어치 주세요.

아저씨 오늘 얼마나 파셨어요.. 라는 질문의 대답

너 사과 심부름 하고 나머지 잔돈이 얼마야.. 라는 질문의 대답

```
void BuyApples(FruitSeller &seller, int money)
{
    numOfApples+=seller.SaleApples(money);
    myMoney-=money;
}
```

FruitBuyer 객체가 FruitSeller 객체의 SaleApples 함수를 호출하고 있다. 그리고 객체지향에서는 이것을 '두 객체가 대화하는 것'으로 본다. 따라서 이러한 형태의 함수호출을 가리켜 '메시지 전달'이라 한다.

예제 FruitSalesSim1

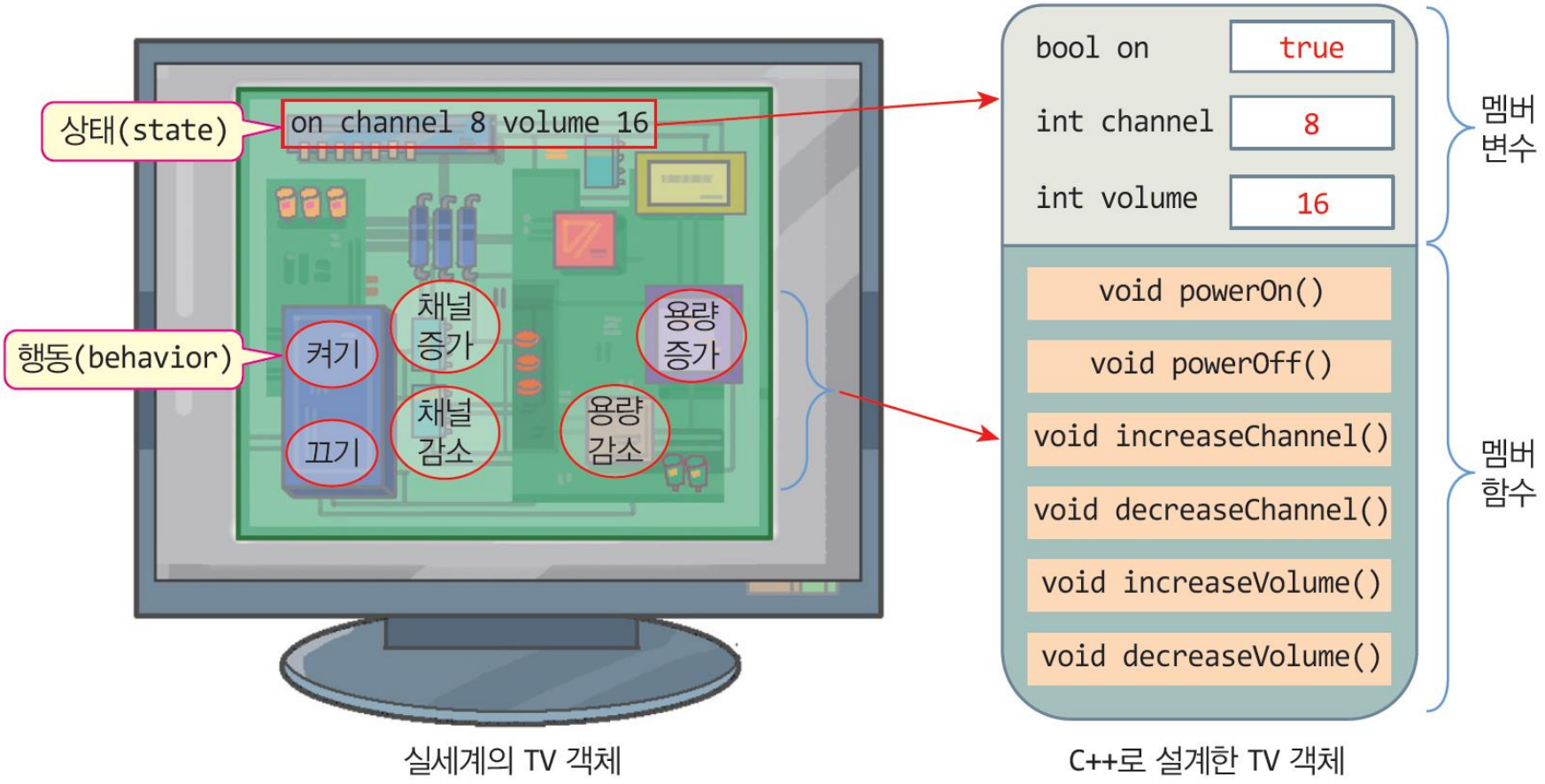
```
#include <iostream>
using namespace std;

class FruitSeller
{
private:
        int APPLE_PRICE;
        int numOfApples;
        int myMoney;

public:
        void InitMembers(int price, int num, int money)
        {
                APPLE_PRICE=price;
                numOfApples=num;
                myMoney=money;
        }
        int SaleApples(int money)
        {
                int num=money/APPLE_PRICE;
                numOfApples-=num;
                myMoney+=money;
                return num;
        }
        void ShowSalesResult()
        {
                cout<<"남은 사과: "<<numOfApples<<endl;
                cout<<"판매 수익: "<<myMoney<<endl<<endl;
        }
};
```

```
class FruitBuyer
{
        int myMoney;            // private:
        int numOfApples;        // private:

public:
        void InitMembers(int money)
        {
                myMoney=money;
                numOfApples=0;
        }
        void BuyApples(FruitSeller &seller, int money)
        {
                numOfApples+=seller.SaleApples(money);
                myMoney-=money;
        }
        void ShowBuyResult()
        {
                cout<<"현재 잔액: "<<myMoney<<endl;
                cout<<"사과 개수:
"<<numOfApples<<endl<<endl;
        }
};

int main(void)
{
        FruitSeller seller;
        seller.InitMembers(1000, 20, 0);
        FruitBuyer buyer;
        buyer.InitMembers(5000);
        buyer.BuyApples(seller, 2000);

        cout<<"과일 판매자의 현황"<<endl;
        seller.ShowSalesResult();
        cout<<"과일 구매자의 현황"<<endl;
        buyer.ShowBuyResult();
        return 0;
}
```

다른 예제) TV 객체 사례

- 객체는 상태(state)와 행동(behavior)으로 구성
- TV 객체 사례
  - 상태
    - on/off 속성 – 현재 작동 중인지 표시
    - 채널(channel) - 현재 방송중인 채널
    - 음량(volume) – 현재 출력되는 소리 크기
  - 행동
    - 켜기(power on)
    - 끄기(power off)
    - 채널 증가(increase channel)
    - 채널 감소(decrease channel)
    - 음량 증가(increase volume)
    - 음량 줄이기(decrease volume)

TV와 C++로 설계된 TV 객체



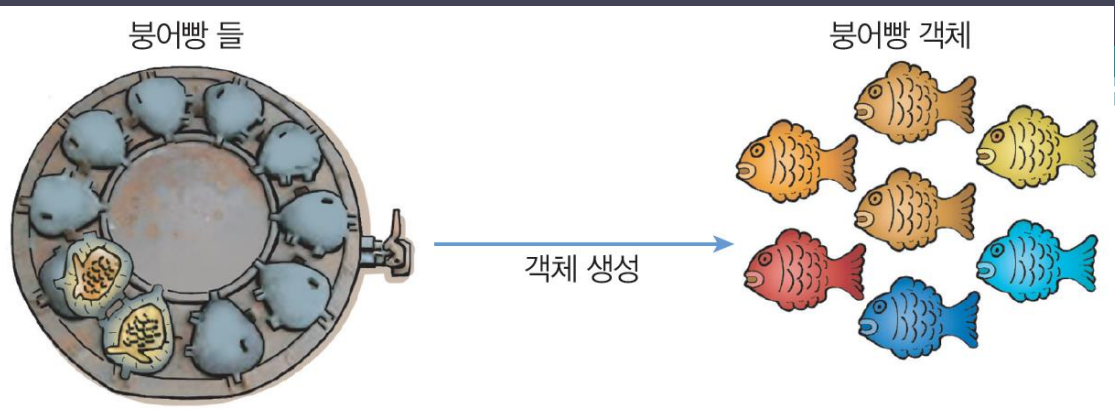
실세계의 TV 객체

C++로 설계한 TV 객체

C++클래스와 C++객체

- 클래스
  - 객체를 만들어내기 위해 정의된 설계도, 틀
  - 클래스는 객체가 아님. 실체도 아님
  - 멤버 변수와 멤버 함수 선언
- 객체
  - 객체는 생성될 때 클래스의 모양을 그대로 가지고 탄생
  - 멤버 변수와 멤버 함수로 구성
  - 메모리에 생성, 실체(instance)라고도 부름
  - 하나의 클래스 틀에서 찍어낸 여러 개의 객체 생성 가능
  - 객체들은 상호 별도의 공간에 생성

클래스와 객체 관계



(a) 붕어빵들과 붕어빵 객체들

```
class TV {
   bool on;
   int channel;
   int volume;
public:
   void powerOn() { ... }
   void powerOff() { ... }
   void increaseChannel() { ... }
   void decreaseChannel() { ... }
   void increaseVolume() { ... }
   void decreaseVolume() { ... }
};
```

TV 클래스

객체 생성

객체들은 서로 별도의 영역에 멤버들을 만든다.

TV 객체 1

bool on	true
int channel	8
int volume	16
void powerOn()	
void powerOff()	
void increaseChannel()	
void decreaseChannel()	
void increaseVolume()	
void decreaseVolume()	

TV 객체 2

bool on	false
int channel	13
int volume	5
void powerOn()	
void powerOff()	
void increaseChannel()	
void decreaseChannel()	
void increaseVolume()	
void decreaseVolume()	

(b) C++로 표현한 TV 클래스와 TV 객체들

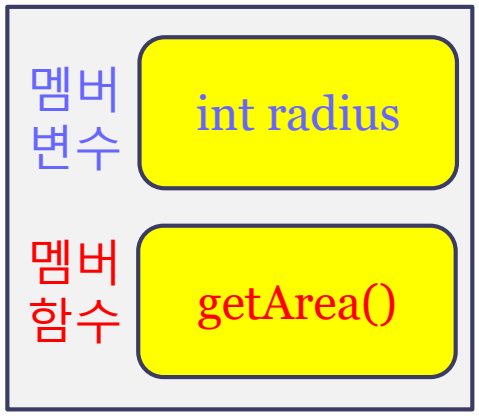
정리) 클래스 만들기

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

- 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 접근제어 지시자 이용 권한 설정
    - private, public, protected 중의 하나
    - 디폴트는 private
    - public : 다른 모든 클래스나 객체에서 멤버의 접근이 가능함을 표시

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

클래스 구성



클래스의 선언은 class 키워드 이용

클래스 이름

멤버에 대한 접근 지정자

세미콜론으로 끝남

```
class Circle {
public:
   int radius; // 멤버 변수
   double getArea(); // 멤버 함수
};
```

클래스 선언부

클래스 선언과 클래스 구현으로 분리하는 이유는 클래스를 다른 파일에서 활용하기 위함

함수의 리턴 타입

클래스 이름

범위지정 연산자

멤버 함수명과 매개변수

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

클래스 구현부

클래스와 친해지기
- 예제 중심

예제1) 사각형 클래스

- 아래 클래스를 가지고 하나의 객체를 생성하는 프로그램을 작성해보자.

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

- 실행결과

```
사각형의 넓이: 12
계속하려면 아무 키나 누르십시오 ....
```

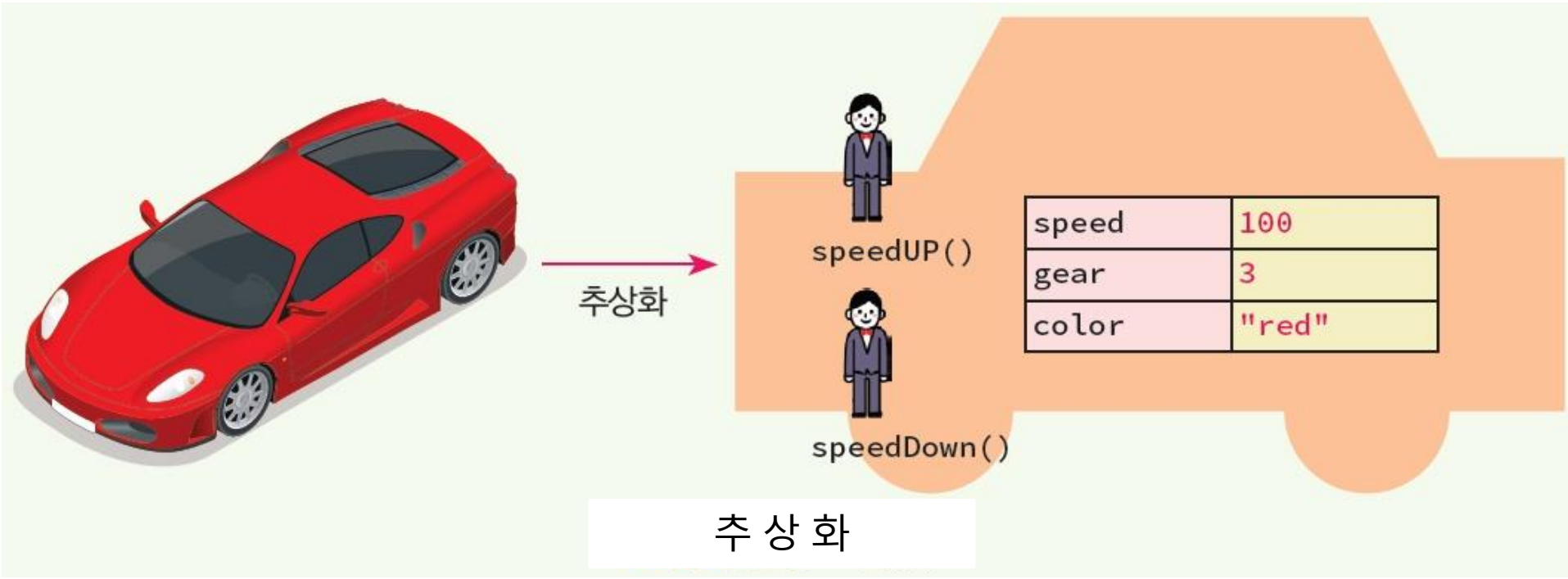
solution

```
#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;
}
```

예제2) Car 클래스 작성

- 먼저 추상화 해보자

[자동차의 추상화]



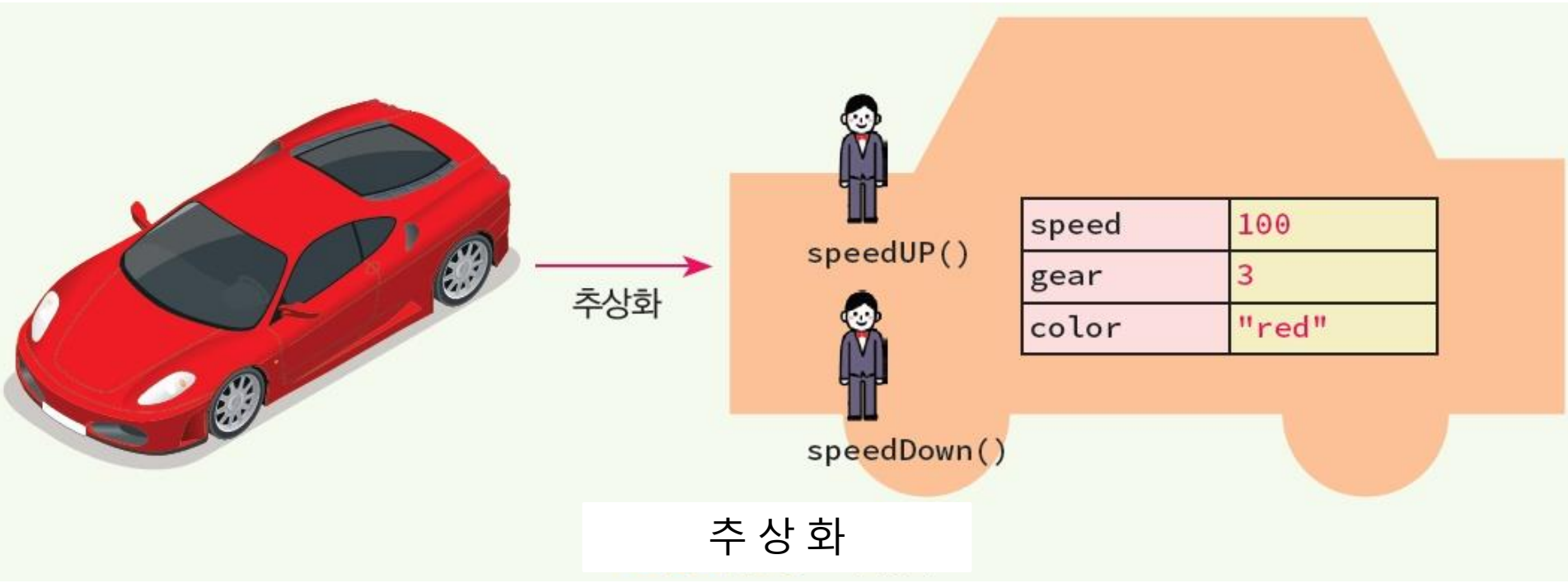
추 상 화

예제2) Car 클래스 작성

- 먼저 추상화 해보자

자동차의 추상화

- 속성: 모델, 색상, 가격, 속도, 기어, 주행거리, 승차인원 등
- 동작: 출발, 멈추기, 가속하기, 감속하기, 방향선택 등



추 상 화

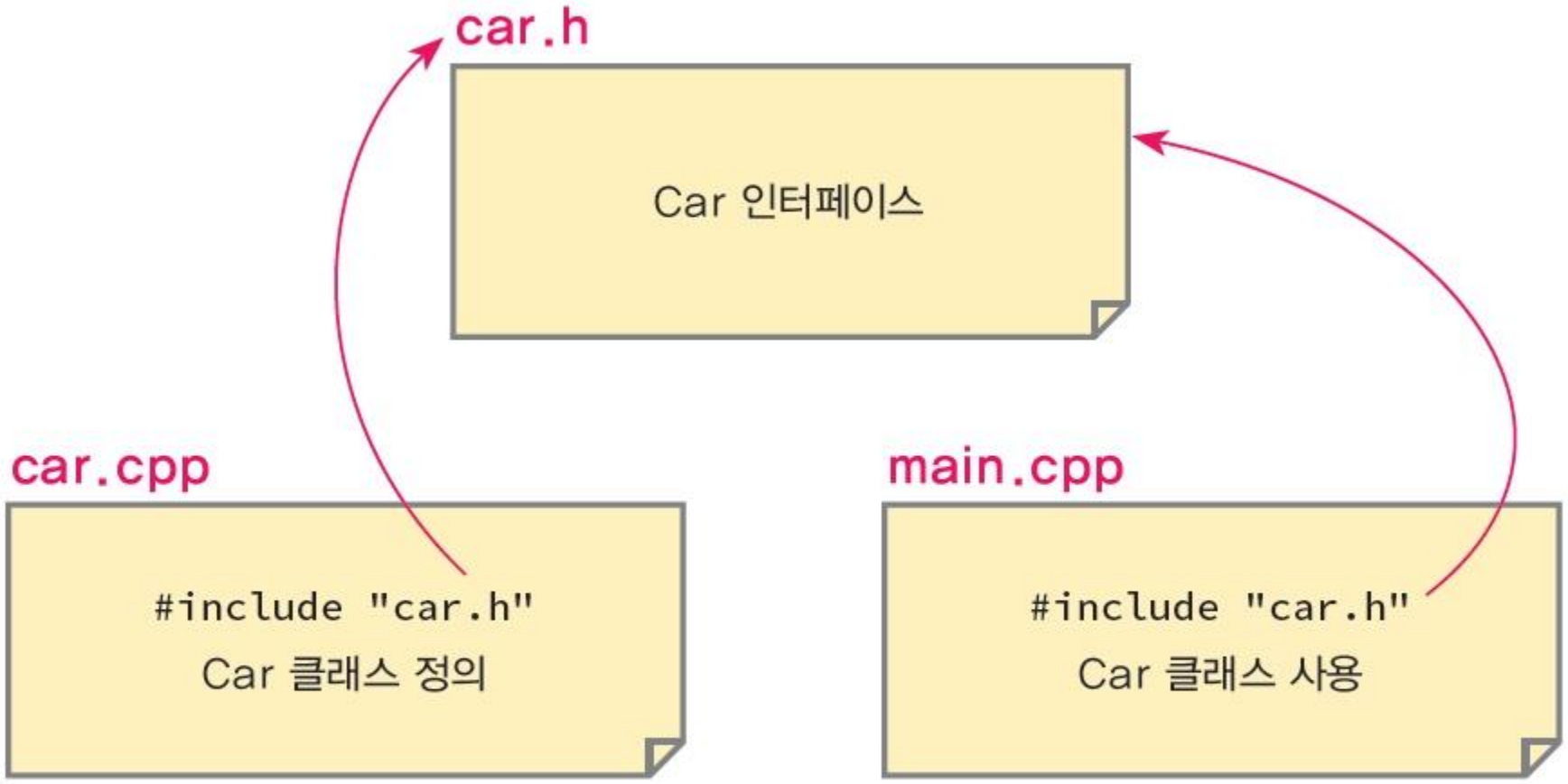
```
#include <iostream>
#include <string>
using namespace std;

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;
}
```

추가 예제)

클래스의 선언과 클래스의 정의 분리



클래스를 헤더 파일과 소스 파일로 분리

car.h

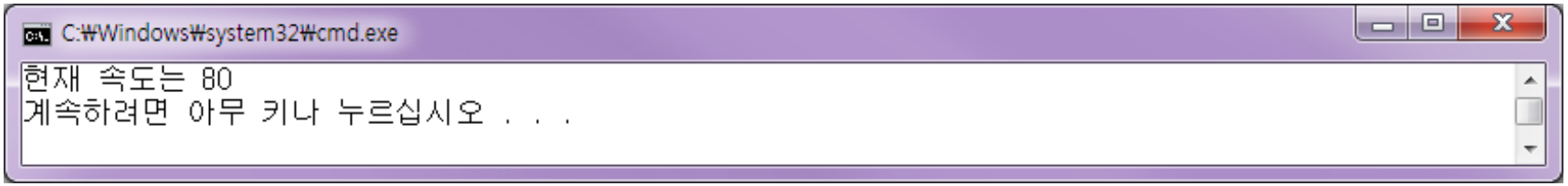
```
#include <iostream>
#include <string>
using namespace std;

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;
}
```



실행 결과


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

참고문헌

- 뇌를 자극하는 C++ 프로그래밍, 이현창, 한빛미디어, 2011
- 열혈 C++ 프로그래밍(개정판), 윤성우, 오렌지미디어, 2012
- 객체지향 원리로 이해하는 ABSOLUTE C++ , 최영근 외 4명 , 교보문고, 2013
- C++ ESPRESSO, 천인국, 인피니티북스, 2011
- 명품 C++ Programming, 황기태 , 생능출판사, 2018
- 어서와 C++는 처음이지, 천인국, 인피니티북스, 2018

Q&A