Power C++

제11장 클래스의 활용

이번 장에서 학습할 내용

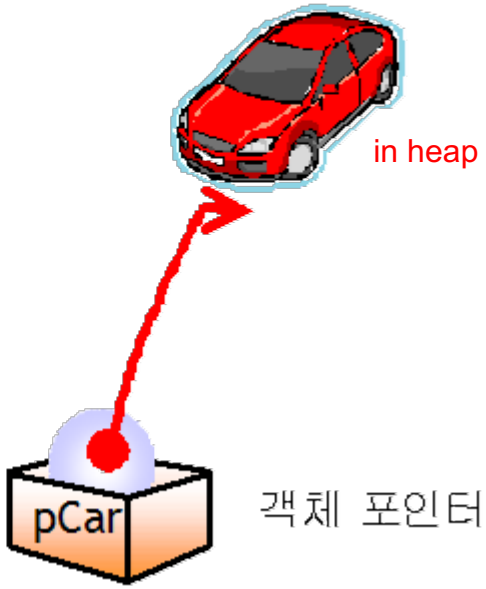
- 객체의 동적 생성
- this
- const
- 객체와 연산자
- 객체와 함수
- 정적 멤버
- 객체의 배열

객체의 동적 생성

- 객체도 동적으로 생성할 수 있다.
- Car myCar; // 정적 메모리 할당으로 객체 생성
- Car *pCar = new Car(); // 동적 메모리 할당으로 객체 생성

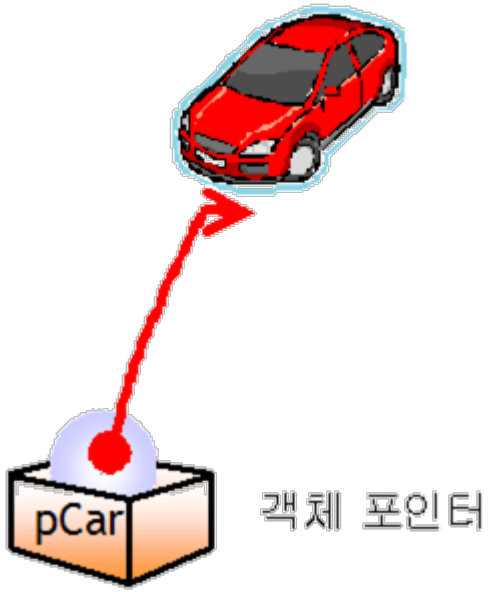
객체는 무명씨로 힙에 저장되고, 그 위치를 알려주는
포인터pCar로 접근

객체 변수 myCar



객체 포인터를 통한 멤버 접근

- pCar->speed = 100;
- pCar->speedUp();



예제

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

class Car {
        int speed;
        int gear;
        string color;
 public:
        Car(int s=0, int g=1, string c="white") : speed(s), gear(g), color(c) {
        }
      void display();
};
void Car::display()
{
      cout << "속도: " << speed << " 기어: " << gear << " 색상: " << color << endl;
}
```

예제

```
int main()
{
    Car myCar;                              // 정적 메모리 할당으로 객체 생성
    myCar.display();

    Car *pCar = &myCar;                     // 객체 포인터로 객체를 가리키게 함
    pCar->display();

    pCar = new Car(0, 1, "blue");   delete[] pCar 필요   // 동적 메모리 할당으로 객체 생성
    pCar->display();
    return 0;
}
```

```
속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: blue
계속하려면 아무 키나 누르십시오 . . .
```

중간 점검 문제

1. 클래스로부터 객체를 생성할 수 있는 방법을 열거하여 보라.

2. 객체 포인터로는 반드시 동적 생성된 객체만을 가리켜야 하는가?

this 포인터

- this는 현재 코드를 실행하는 객체를 가리키는 포인터

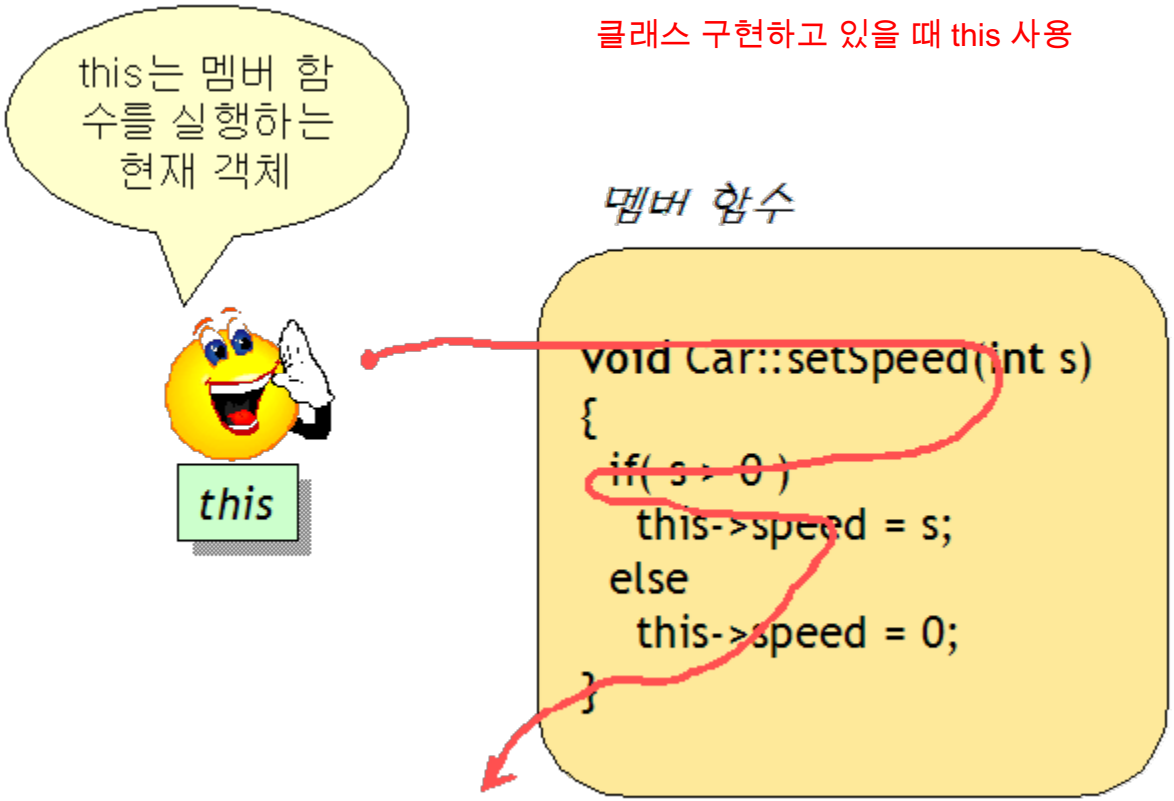


그림 10.2 this 포인터

this를 사용하는 예

```
void Car::setSpeed(int speed)
{
      if( speed > 0 )
            this->speed = speed; // speed는 매개 변수, this->speed는 멤버 변수
      else
            this->speed = 0;
}
```

```
// 생성자
Car::Car(int s) {
      this->setSpeed(s);          // this는 없어도 된다. 멤버 함수임을 강조
      this->gear = 1;
      this->color = "white";
}
```

예제

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

class Person {
	string lastName;
	string firstName;
public:
	Person(string lastName, string firstName);
	string getLastName() {
		return lastName;
	};
	string getFirstName() {
		return firstName;
	}
	string buildName();
};
```

예제

```
Person::Person(string lastName, string firstName)
{
    this->lastName = lastName;          // this는 현재 객체를 가리킨다.
    this->firstName = firstName;        // this는 현재 객체를 가리킨다.
}
string Person::buildName() {
    return this->getLastName() + this->getFirstName();      // ①
}
int main()
{
    Person person("홍", "길동");
    cout << person.buildName() << endl;
}
```

```
홍길동
계속하려면 아무 키나 누르십시오 . . .
```

중간 점검 문제

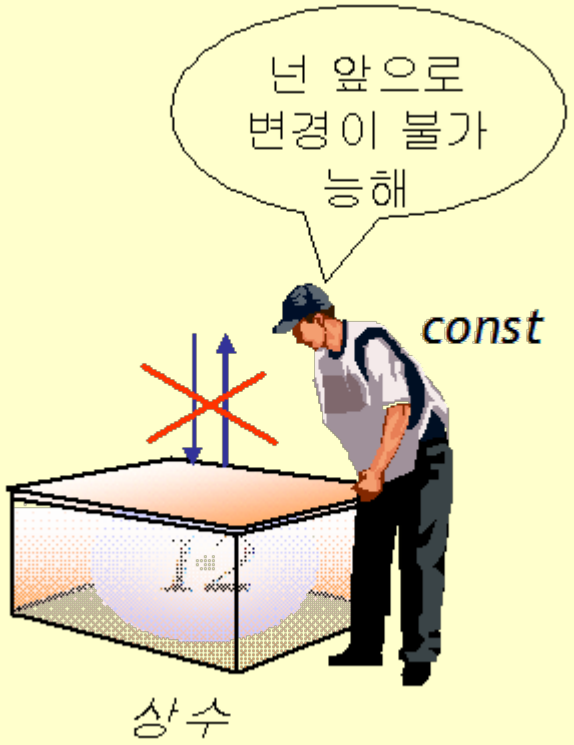
1. this 포인터는 무엇을 가리키는가?

2. this 포인터가 꼭 필요한 경우는?

const 수식어

- 멤버 변수에 const를 붙이는 경우

```
class Car
{
    const int serial;
    string color;
    …
public:
    Car(int s, string c) : serial(s)
    {
        color = c;
    }
}
```



const 수식어

- 멤버 함수에 const를 붙이는 경우

```
void displayInfo() const
{
        cout << "속도: " << speed << endl;
        cout << "기어: " << gear << endl;
        cout << "색상: " << color << endl;
}
```

이 함수 안에서는 멤버 변수의 값을 변경할 수 없다.

멤버 변수를 변화시키지 않겠다는 걸 명시

const 수식어

- 객체에 const를 붙이는 경우

```
int main()
{
    const Car c1(0, 1, "yellow");
    c1.setSpeed();              // 오류!
    return 0;
}
```

이 객체를 통해서는
멤버 변수의 값을
변경할 수 없다.

const 수식어

- 함수에 const가 붙어 있으면 중복이 가능

```
class Car
{
    …
    void printInfo() const
    {
        cout << "속도: " << speed << endl;
        cout << "기어: " << gear << endl;
        cout << "색상: " << color << endl;
    }
    void printInfo()
    {
        cout << "-----------------------------------" << endl;
        cout << "속도: " << speed << endl;
        cout << "기어: " << gear << endl;
        cout << "색상: " << color << endl;
        cout << "-----------------------------------" << endl;
    }
}
```

중간 점검 문제

1. 객체 선언시에 const가 붙으면 어떤 의미인가? const float PI = 3.14f; 선언함과 동시에 그 객체를 update하지 않겠다.
2. 멤버 ~~변수~~ 함수 getSpeed()에 const를 붙여보라. 어떤 의미인가? getSpeed()에서 멤버 변수들을 update시키지 않겠다.

객체와 연산자

- 객체에 할당 연산자(=)를 사용할 수 있는가?

```
class Car
{
    … //생략
};

int main()
{
    Car c1(0, 1, "white");
    Car c2(0, 1, "red");
    c1 = c2;          // 어떻게 되는가?
    return 0;
}
```

c2 객체가 가지고 있는 변수의 값이 c1으로 복사된다..

객체와 연산자

- 객체에 비교 연산자(==)를 사용할 수 있는가?

```
class Car
{
    … //생략
};

int main()
{
    Car c1(0, 1, "white");
    Car c2(0, 1, "red");
    if( c1 == c2 ){
        cout << "같습니다“ << endl;
    }
    else {
        cout << "같습니다“ << endl;
    }
    return 0;
}
```

연산자 중복이 되어 있지 않으면 오류!-> 뒤에 학습

중간 점검 문제

1. = 연산자를 이용하여서 하나의 객체를 다른 객체에 할당할 수 있는가?
2. == 연산자를 이용하여서 하나의 객체와 다른 객체를 비교할 수 있는가?

객체와 함수

① 객체가 함수의 매개 변수로 전달되는 경우
② 함수가 객체를 반환하는 경우
③ 객체의 포인터가 함수의 매개 변수로 전달되는 경우
④ 객체의 레퍼런스가 함수의 매개 변수로 전달되는 경우

객체가 함수의 매개 변수로 전달

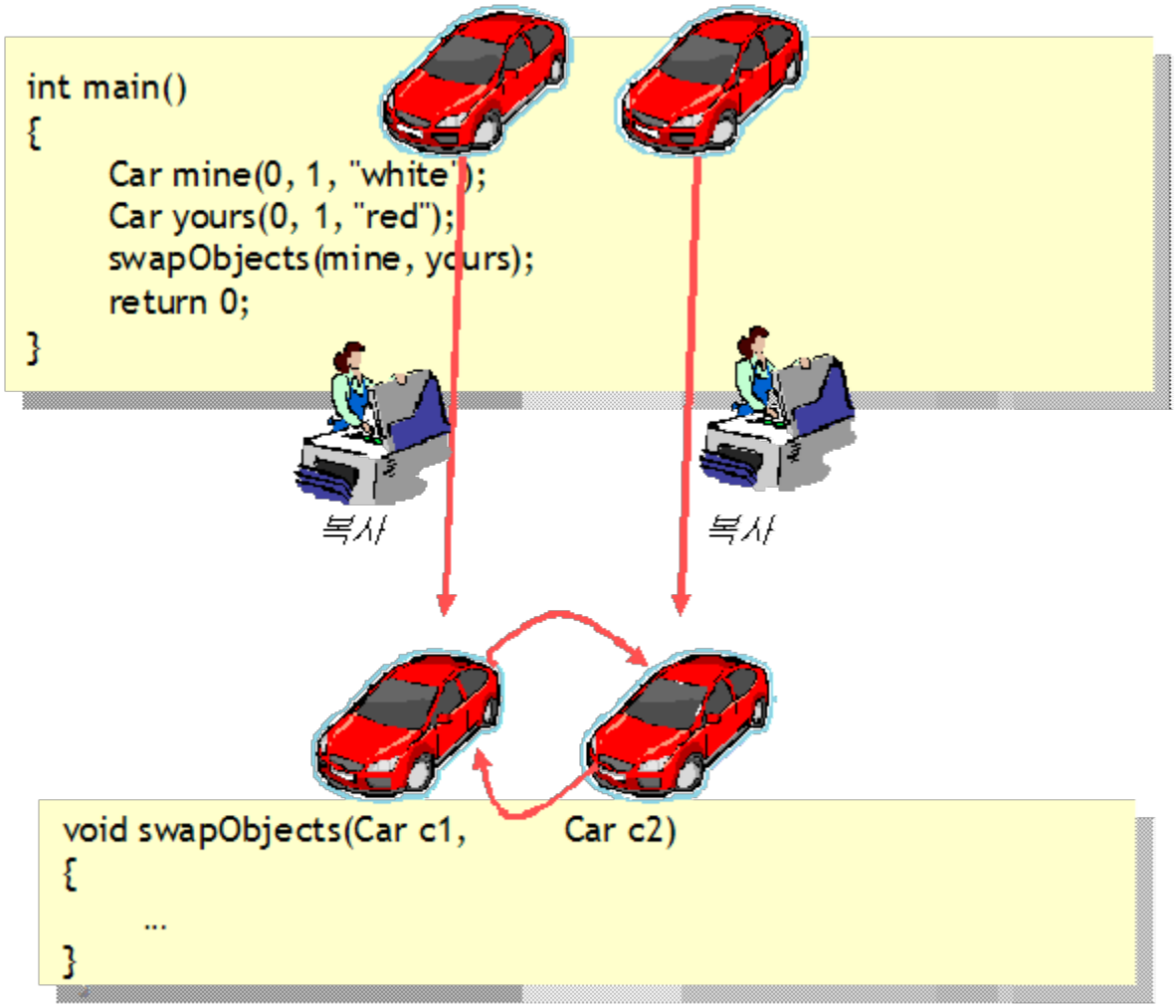


그림 10.4 객체는 값으로 전달된다.

객체가 함수의 매개 변수로 전달

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

class Car {
        int speed;
        int gear;
        string color;
 public:
        Car(int s=0, int g=1, string c="white") : speed(s), gear(g), color(c) {
        }
                void display();
};

void Car::display()
{
     cout << "속도: " << speed << " 기어: " << gear << " 색상: " << color << endl;
}
```

객체가 함수의 매개 변수로 전달

```
void swapObjects(Car c1, Car c2)
{
    Car tmp;
    tmp = c1;
    c1 = c2;
    c2 = tmp;
    c1.display();
    c2.display();
}

int main()
{
    Car mine(0, 1, "white");
    Car yours(0, 1, "red");
    swapObjects(mine, yours);
    mine.display();
    yours.display();

    return 0;
}
```

```
속도: 0 기어: 1 색상: red
속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: red
계속하려면 아무 키나 누르십시오 . . .
```

함수가 객체를 반환

```
...// 전과 동일
Car buyCar()
{
    Car tmp(0, 1, "metal");
    return tmp;
}

int main()
{
    Car c1;
    c1.display();
    c1 = buyCar();
    c1.display();

    return 0;
}
```

```
속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: metal
계속하려면 아무 키나 누르십시오 . . .
```

객체의 포인터가 함수에 전달

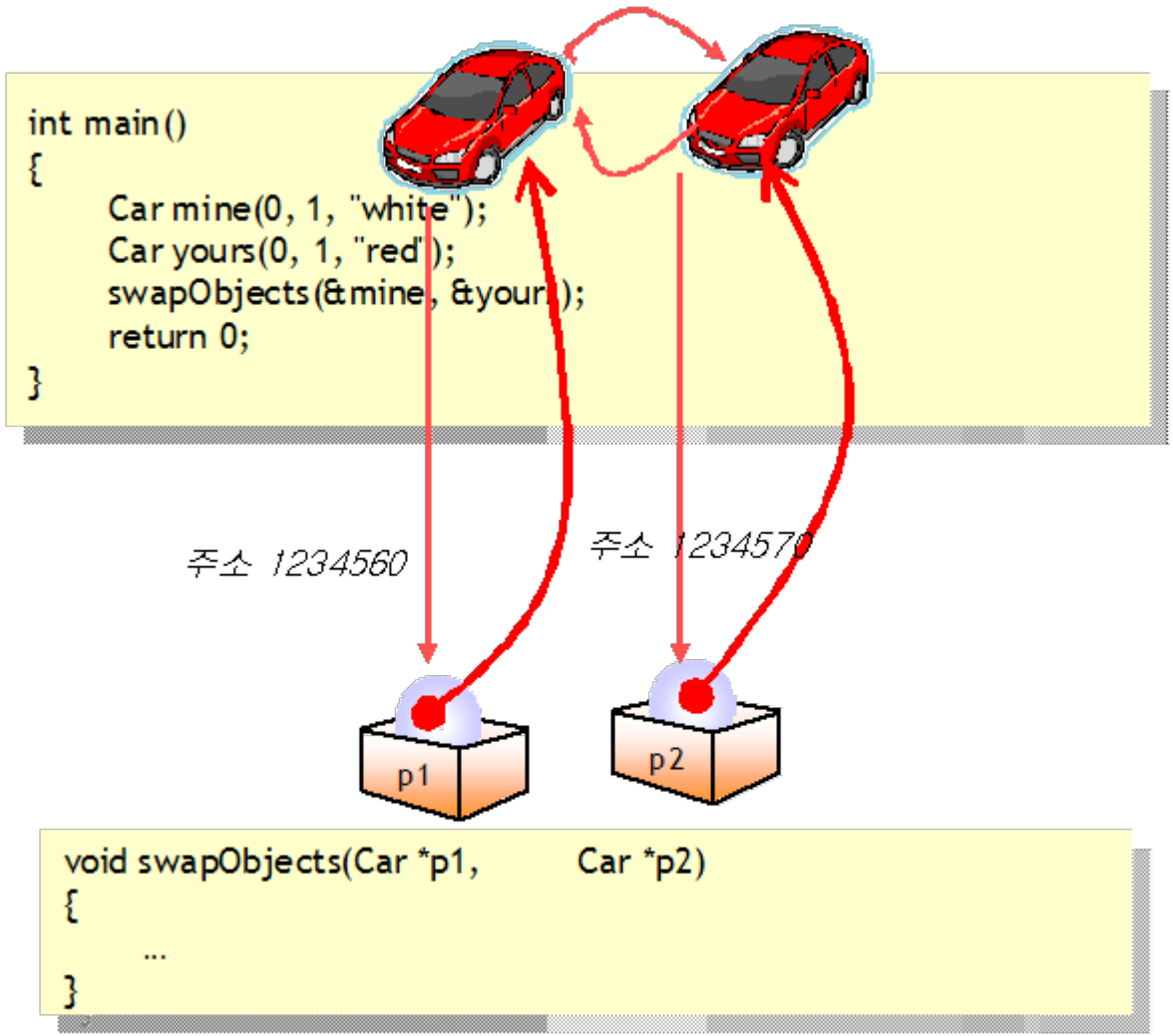


그림 10.5 객체의 포인터를 전달하는 경우

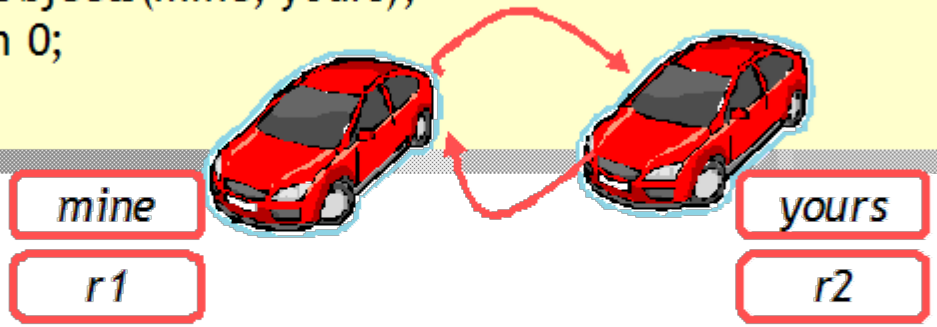
객체의 포인터가 함수에 전달

```
...// 전과 동일
void swapObjects(Car *p1, Car *p2)
{
    Car tmp;
    tmp = *p1;
    *p1 = *p2;
    *p2 = tmp;
    p1->display();
    p2->display();
}

int main()
{
    Car mine(0, 1, "white");
    Car yours(0, 1, "red");
    swapObjects(&mine, &yours);
    mine.display();
    yours.display();

    return 0;
}
```

```
속도: 0 기어: 1 색상: red
속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: red
속도: 0 기어: 1 색상: white
계속하려면 아무 키나 누르십시오 . . .
```

객체의 레퍼런스가 함수에 전달

```
int main()
{
        Car mine(0, 1, "white");
        Car yours(0, 1, "red");
        swapObjects(mine, yours);
        return 0;
}
```




```
void swapObjects(Car &r1,       Car &r2)
{
        ...
}
```

const가 없으니 update되겠구나를 암시

그림 10.6 객체의 레퍼런스를 전달하는 경우

객체의 포인터가 함수에 전달

```
...// 전과 동일
void swapObjects(Car &r1, Car &r2)
{
    Car tmp;
    tmp = r1;
    r1 = r2;
    r2 = tmp;
    r1.display();
    r2.display();
}

int main()
{
    Car mine(0, 1, "white");
    Car yours(0, 1, "red");
    swapObjects(mine, yours);
    mine.display();
    yours.display();

    return 0;
}
```

```
속도: 0 기어: 1 색상: red
속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: red
속도: 0 기어: 1 색상: white
계속하려면 아무 키나 누르십시오 . . .
```

중간 점검 문제

1. 함수 안에서 매개 변수로 전달받은 객체의 내용을 수정하려면 매개 변수를 어떤 타입으로 선언하여야 하는가?
2. 매개 변수로 포인터와 레퍼런스를 사용하는 경우를 비교하여 보자.

정적 멤버

- 인스턴스 변수(instance variable): 객체마다 하나씩 있는 변수
- 정적 변수(static variable): 모든 객체를 통틀어서 하나만 있는 변수

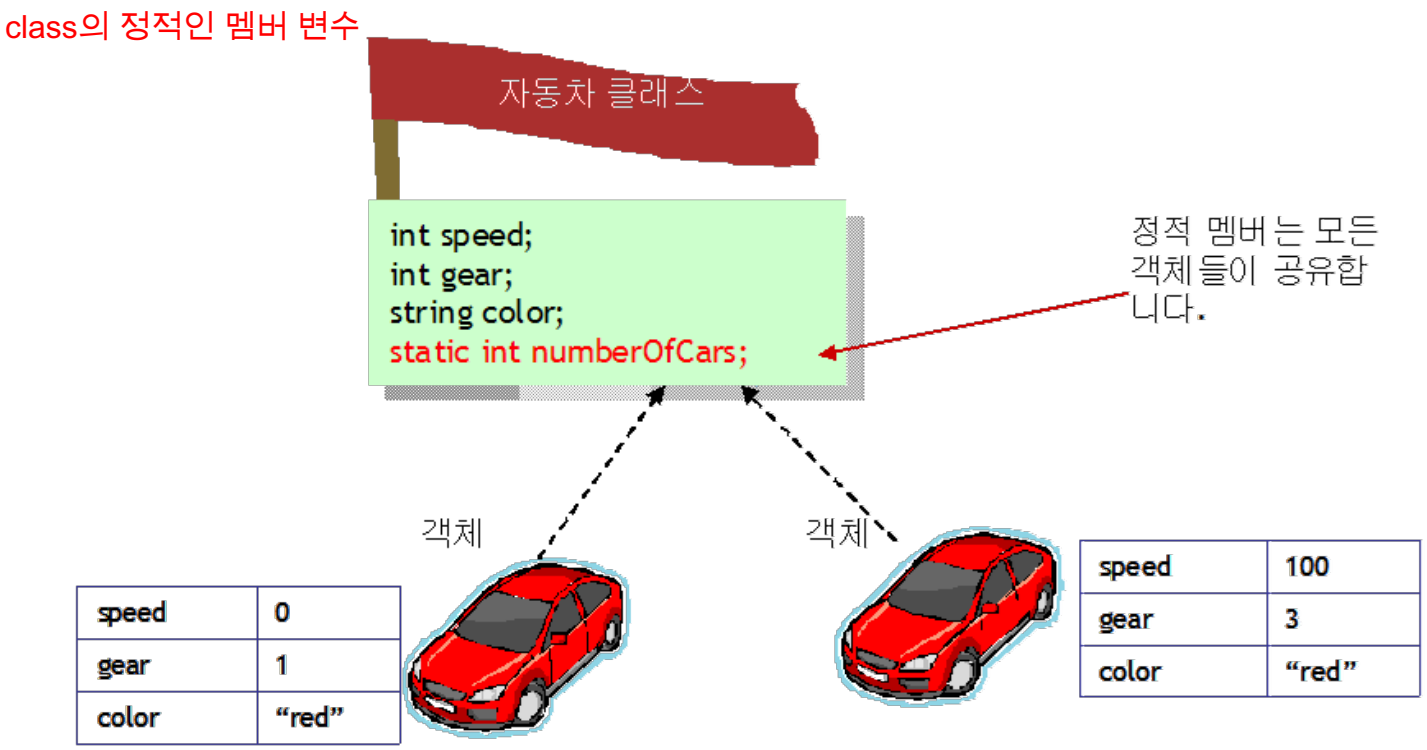


그림 10.7 정적 멤버

정적 멤버 변수

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

class Car {
      int speed;
      int gear;
      string color;
      int id;        // 자동차의 시리얼 번호

public:
      // 실체화된 Car 객체의 개수를 위한 정적 변수
      static int numberOfCars;
      Car(int s=0, int g=1, string c="white"): speed(s), gear(g), color(c) {
            // 자동차의 개수를 증가하고 id 번호를 할당한다.
            id = ++numberOfCars;
      }
};
```

정적 변수의 선언

정적 멤버 변수

```
int Car::numberOfCars = 0;

int main()
{
    Car c1;
    cout << Car::numberOfCars << endl;

    Car c2;
    cout << c2.numberOfCars << endl;
}
```

정적 변수의 정의

```
1
2
계속하려면 아무 키나 누르십시오 . . .
```

정적 멤버 함수

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

class Car {
    int speed;
    int gear;
    string color;
    int id;        // 자동차의 시리얼 번호

public:
    // 실체화된 Car 객체의 개수를 위한 정적 변수
    static int numberOfCars;  // 정적 변수의 선언
    Car(int s=0, int g=1, string c="white"): speed(s), gear(g), color(c) {
            // 자동차의 개수를 증가하고 id 번호를 할당한다.
            id = ++numberOfCars;
    }
    // 정적 멤버 함수
    static int getNumberOfCars() {
            return numberOfCars; // OK!
    }
};
```

정적 멤버 변수

```
int Car::numberOfCars=0;      // 정적 변수의 정의

int main()
{
    Car c1(100, 0, "blue");      // 첫 번째 생성자 호출
    Car c2(0, 0, "white");       // 첫 번째 생성자 호출
    int n = Car::getNumberOfCars();      // 정적 멤버 함수 호출
    cout <<"지금까지 생성된 자동차 수 = " << n << endl;
    return 0;
}
```

```
지금까지 생성된 자동차 수 = 2
계속하려면 아무 키나 누르십시오 . . .
```

예제

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

class Employee {
    string name;
    double salary;

    static int count;              // 정적 변수
public:
    // 생성자
    Employee(string n="", double s=0.0): name(n), salary(s) {
        count++; // 정적 변수인 count를 증가
    }

    // 객체가 소멸될 때 호출된다.
    ~Employee() {
        count--; // 직원이 하나 줄어드는 것이므로 count를 하나 감소
    }
```

정적 멤버 변수

```
    // 정적 멤버 함수
    static int getCount() {
        return count;
    }
};
int Employee::count=0;          // 정적 변수

int main()
{
    Employee e1("김철수", 35000);
    Employee e2("최수철", 50000);
    Employee e3("김철호", 20000);

    int n = Employee::getCount();
    cout << "현재의 직원수=" << n << endl;
    return 0;
}
```

```
현재의 직원수=3
```

중간 점검 문제

1. 정적 변수는 어떤 경우에 사용하면 좋은가?
2. 정적 변수나 정적 멤버 함수를 사용할 때, 클래스 이름을 통하여 접근하는 이유는 무엇인가?
3. 정적 멤버 함수 안에서 인스턴스 멤버 함수를 호출할 수 없는 이유는 무엇인가?

객체들의 배열

- Car objArray[3];

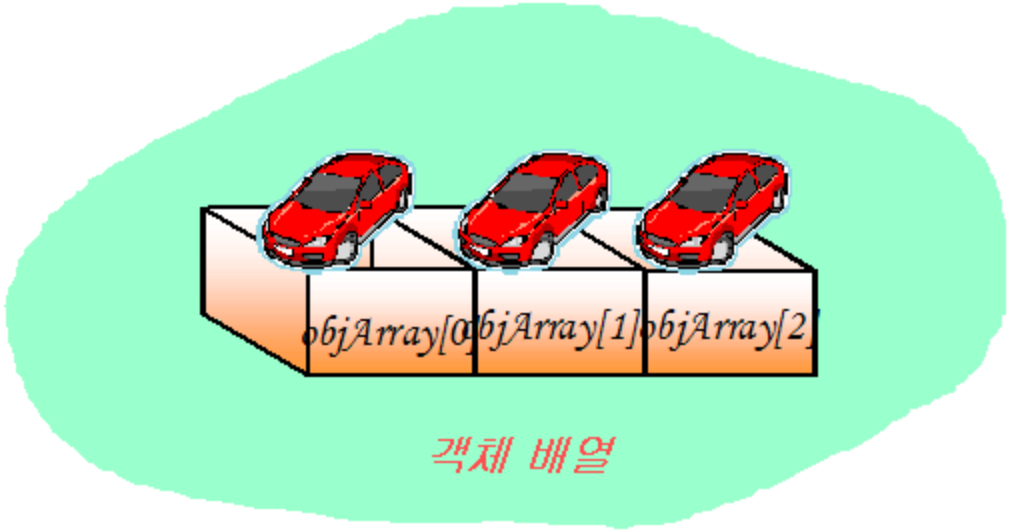


그림 10.8 객체 배열

```
objArray[0].speed = 0;// 멤버 변수 접근
objArray[1].speedUp();// 멤버 함수 호출
```

객체 배열의 초기화

```
Car objArray[3] = {
    Car(0, 1, "white"),
    Car(0, 1, "red"),
    Car(0, 1, "blue"),
};
```

객체 별로 생성자를
호출할 수 있다.

예제

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

class Car {
      int speed;
      int gear;
      string color;
public:
      Car(int s=0, int g=1, string c="white"): speed(s), gear(g), color(c) {
      }
      void display();
};

void Car::display()
{
      cout << "속도: " << speed << " 기어: " << gear << " 색상: " << color << endl;
}
```

예제

```
int main()
{
    Car objArray[3] = {
        Car(0, 1, "white"),
        Car(0, 1, "red"),
        Car(0, 1, "blue"),
    };
    for(int i=0; i< 3; i++)
        objArray[i].display();

    return 0;
}
```

```
속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: red
속도: 0 기어: 1 색상: blue
계속하려면 아무 키나 누르십시오 . . .
```

클래스와 클래스 간의 관계

- 사용(use): 하나의 클래스가 다른 클래스를 사용한다.
- 포함(has-a): 하나의 클래스가 다른 클래스를 포함한다.
- 상속(is-a): 하나의 클래스가 다른 클래스를 상속한다.

면접 많이 물어봄!!!!!!!

사용 관계

```
ClassA::func()
{
    ClassB obj;   // 사용 관계
    obj.func();
    …
}
```

포함 관계

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

// 시각을 나타내는 클래스
class Time {
private:
    int time;                   // 시간
    int minute;                 // 분
    int second;                 // 초
public:
    Time();                     // 디폴트 생성자
    Time(int t, int m, int s);  // 생성자
    void print();               // 객체의 정보 출력
};

Time::Time() {                  // 디폴트 생성자
    time = 0;
    minute = 0;
    second = 0;
}
```

포함 관계

```
Time::Time(int t, int m, int s) {          // 생성자
    time = t;
    minute = m;
    second = s;
}
void Time::print()   // 객체의 정보를 출력
{
    cout << time << "시 " << minute << "분 " << second << "초 \n";
}

// 알람 시계를 나타낸다.
class AlarmClock {
private:
    Time currentTime;          // 현재 시각
    Time alarmTime;            // 알람 시각
public:
    AlarmClock(Time a, Time c);          // 생성자
    void print();              // 객체의 정보 출력
};

AlarmClock::AlarmClock(Time a, Time c) {          // 생성자
    alarmTime = a;             // 객체가 복사된다.
    currentTime = c;                        // 객체가 복사된다.
}
```

예제

```
void AlarmClock::print()
{
    cout << "현재 시각: ";
    currentTime.print();
    cout << "알람 시각: ";
    alarmTime.print();
}

int main()
{
    Time alarm(6, 0, 0);
    Time current(12, 56, 34);
    AlarmClock c(alarm, current);

    c.print();
    return 0;
}
```

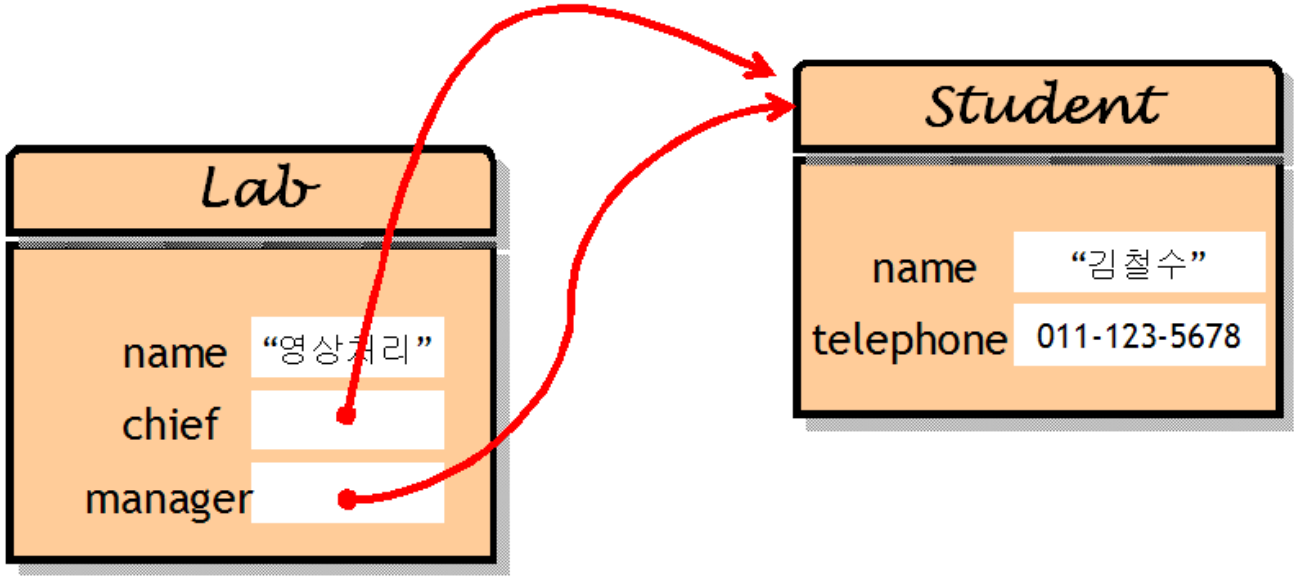
```
현재 시각: 12시 56분 34초
알람 시각: 6시 0분 0초
```

중간 점검 문제

1. 사용 관계와 포함 관계는 어떻게 다른가?
2. 사용 관계와 포함 관계의 예를 더 들어보자.

예제 #1 객체 포인터

- 만약 한 학생이 실험실의 실장과 총무를 겸하는 경우, 객체 포인터를 사용하여서 중복을 줄인다.



예제

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

// 학생을 나타낸다.
class Student {
private:
	string name;
	string telephone;
public:
	Student(const string n="", const string t="");
	string getTelephone() const;
	void setTelephone(const string t);
	string getName() const;
	void setName(const string n);
};

Student::Student(const string n, const string t)
{
	name = n;
	telephone = t;
}
```

객체 포인터

```
string Student::getTelephone() const
{
        return telephone;
}
void Student::setTelephone(const string t)
{
        telephone = t;
}

string Student::getName() const
{
        return name;
}
void Student::setName(const string n)
{
        name = n;
}
```

객체 포인터

```
// 연구실을 나타낸다.
class Lab {
       string name;
         Student *chief;
         Student *manager;
public:
      Lab(string n="");
      void setChief(Student *p);
      void setManager(Student *p);
      void print() const;
};

Lab::Lab(const string n)
{
      name = n;
      chief = NULL;
      manager = NULL;
}

void Lab::setChief(Student *p)
{
      chief = p;
}
```

객체 포인터

```
void Lab::setManager(Student *p)
{
        manager = p;
}

void Lab::print() const
{
    cout << name << "연구실" << endl;
    if( chief != NULL )
        cout << "실장은 " << chief->getName() << endl;
    else
        cout << "실장은 현재 없습니다\n";
    if( manager != NULL )
        cout << "총무는 " << manager->getName() << endl;
    else
        cout << "총무는 현재 없습니다\n";
}
```

객체 포인터

```
int main()
{
    Lab lab("영상 처리");
    Student *p= new Student("김철수", "011-123-5678");

    lab.setChief(p);
    lab.setManager(p);
    lab.print();

    delete p;
    return 0;
}
```

```
영상 처리연구실
실장은 김철수
총무는 김철수
```

예제#2 복소수

복소수: $a + bi$

복소수

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

class Complex
{
private:
    double real;                // 실수부
    double imag;    // 허수부

public:
    Complex();                  // 생성자
    Complex(double a, double b); // 생성자
    ~Complex();                 // 소멸자

    double getReal();           // 실수부를 반환한다.
    double getImag();           //허수부를 반환한다.
    Complex add(const Complex& c);    // 복소수의 덧셈 연산을 구현한다.
    void print();               // 복소수를 출력한다.
};
```

복소수

```
Complex::Complex()
{
    real = 0;
    imag = 0;
}

Complex::Complex(double a, double b)
{
    real = a;
    imag = b;
}

Complex::~Complex()
{
}

double Complex::getReal()
{
    return(real);
}
```

복소수

```
double Complex::getImag()
{
    return(imag);
}
// 복소수의 덧셈 연산 구현
Complex Complex::add(const Complex& c)
{
    Complex temp;  // 임시 객체
    temp.real = this->real + c.real;
    temp.imag = this->imag + c.imag;

    return(temp);    // 객체를 반환한다.
}

void Complex::print()
{
    cout << real << " + " << imag << "i" << endl;
}
```

복소수

```
int main(void)
{
    Complex x(2, 3), y(4, 6), z;

    cout << "첫번째 복소수 x: ";
    x.print();

    cout << "두번째 복소수 y: ";
    y.print();

    z = x.add(y);                    // z = x + y

    cout << " z = x + y = ";
    z.print();

    return(0);
}
```

```
첫번째 복소수 x: 2 + 3i
두번째 복소수 y: 4 + 6i
 z = x + y = 6 + 9i
```

Q & A