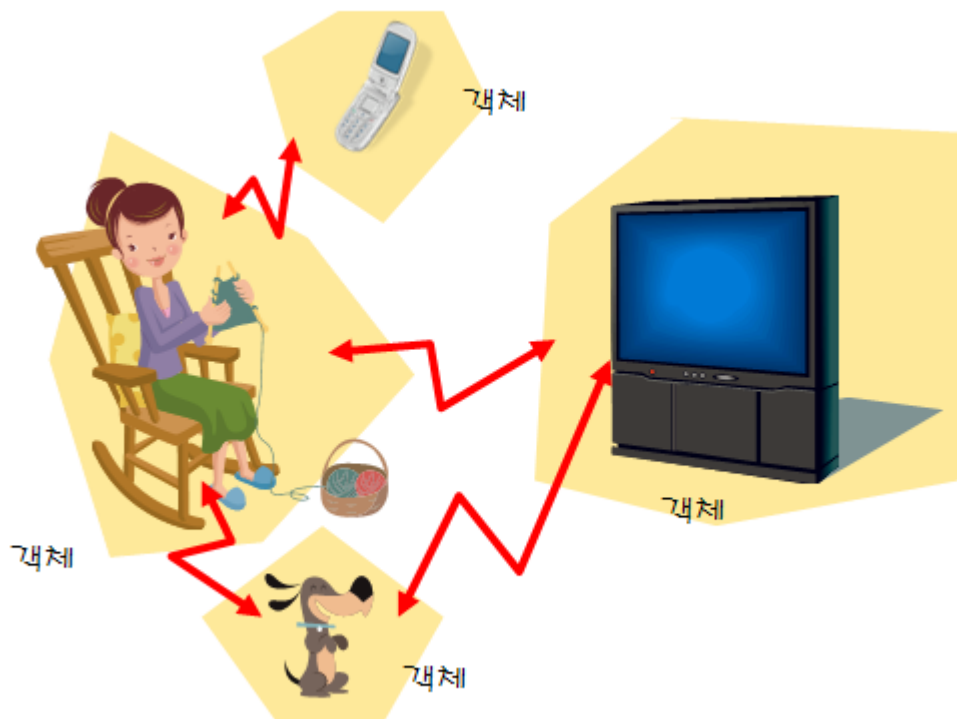




C++ Espresso

제7장 클래스의 활용





이번 장에서 학습할 내용



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

객체와
클래스의
활용에 필요한
사항들을
살펴봅니다.





객체들의 배열

- 객체 들의 배열
- 예: `Car objArray[3];`

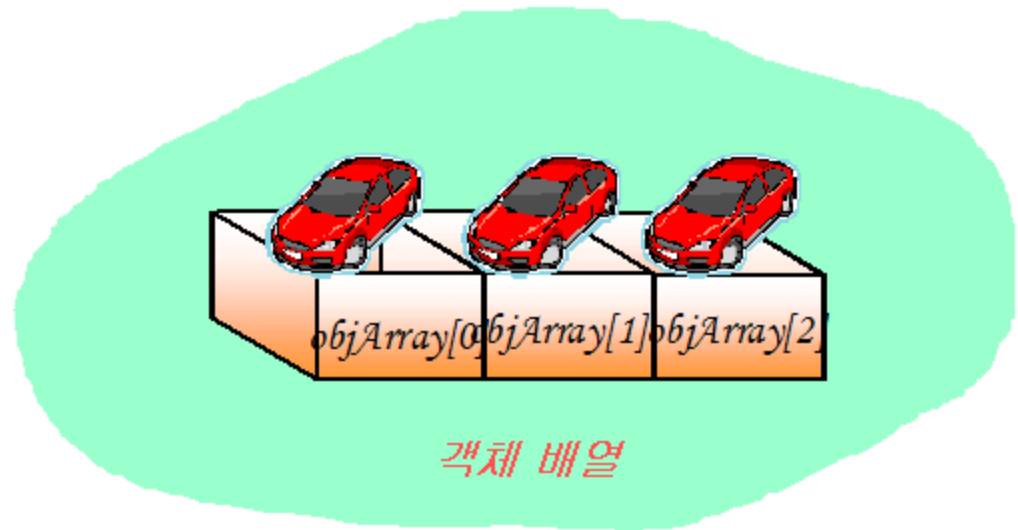


그림 10.8 객체 배열

```
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 print()
    {
        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].print();

    return 0;
}
```



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



객체의 동적 생성

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



객체 변수 `myCar`

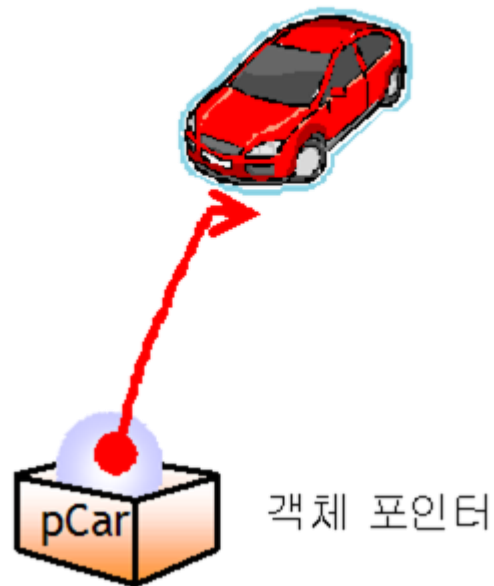


객체 포인터



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



예제



```
int main()
```

```
{
```

```
    Car myCar;
```

```
    myCar.print();
```

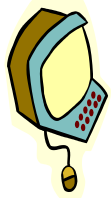
```
    pCar = new Car(0, 1, "blue");
```

```
    pCar->print();
```

```
    return 0;
```

```
}
```

객체 동적 생성



속도: 0 기어: 1 색상: white

속도: 0 기어: 1 색상: blue



this 포인터

- this는 현재 코드를 실행하는 객체를 가리키는 포인터

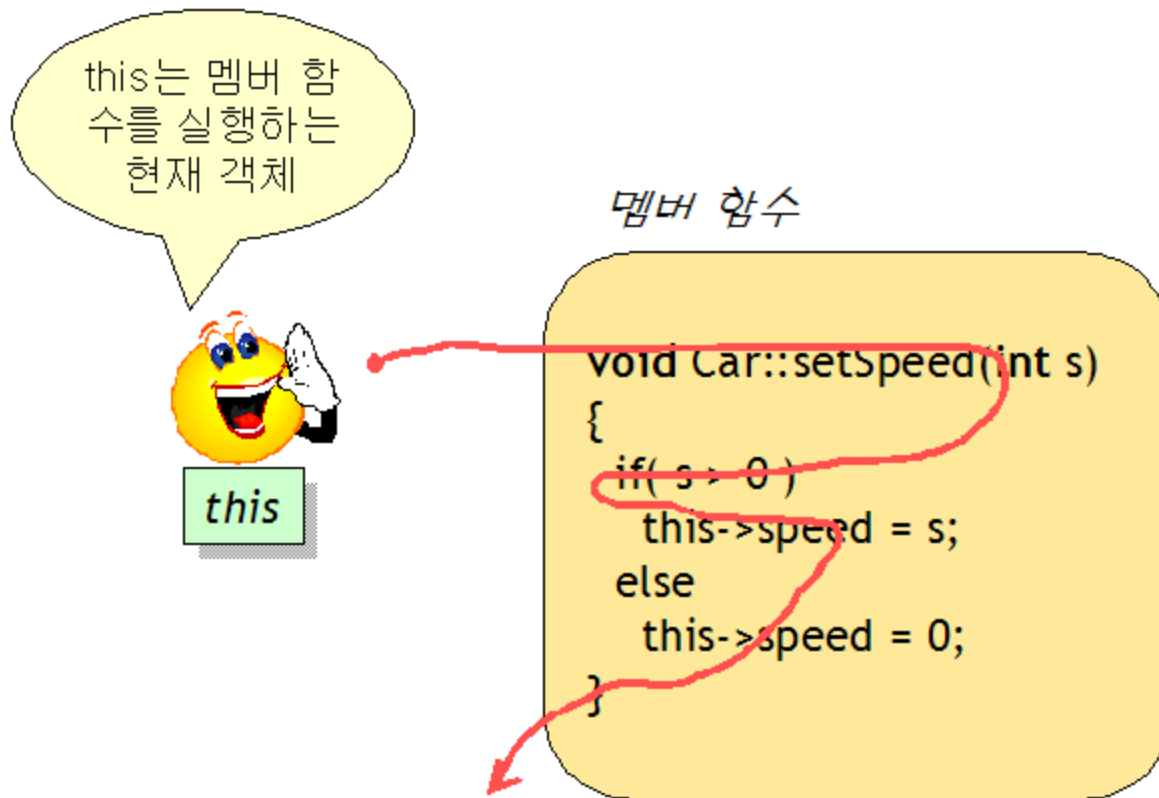


그림 10.2 this 포인터



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 Car {
    int speed; // 속도
    int gear; // 기어
    string color; // 색상
public:
    Car(int s=0, int g=1, string c="white") : speed(s), gear(g), color(c)
    {
    }
    int getSpeed()
    {
        return speed;
    }
}
```



예제



```
void setSpeed(int speed)
{
    if( speed > 0 )
        this->speed = speed;
    else
        this->speed = 0;
}
void print()
{
    cout << "속도: " << speed << " 기어: " << gear << " 색상: " << color;
}
void isFaster(Car *p)
{
    if( this->getSpeed() > p->getSpeed() )
        this->print();
    else
        p->print();
    cout << "의 자동차가 더 빠름" << endl;
}
};
```



예제



```
int main()
{
    Car c1(0, 1, "blue");
    Car c2(100, 3, "red");
    c1.isFaster(&c2);
    return 0;
}
```



속도: 100 기어: 3 색상: red의 자동차가 더 빠름



const 수식어

- 멤버 변수에 `const`를 붙이는 경우

이 멤버 변수의 값을
변경할 수 없다.

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




const 수식어

- 멤버 함수에 const를 붙이는 경우

이 함수 안에서는 멤버 변수의 값을 변경할 수 없다.

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



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;
    }
}
```



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; // 어떻게되는가?
    return 0;
}
```

c2 객체가 가지고 있는 변수의 값이 c1으로 복사된다..



객체와 연산자

- 객체에 할당 연산자(=)를 사용할 수 있는가?

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

연산자 중복이 되어
있지 않으면 오류!



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

```
int main()  
{  
    Car red(0, 1, "red");  
    Car blue(30, 2, "blue");  
    swapObjects(red, blue);  
    return 0;  
}
```



복사

복사



```
void swapObjects(Car c1, Car c2)  
{  
    ...  
}
```



객체가 함수의 매개 변수로 전달



```
#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 print()
    {
        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;  
}
```



속도: 0 기어: 1 색상: red
속도: 30 기어: 2 색상: blue

```
int main()  
{  
    Car red(0, 1, "red");  
    Car blue(30, 2, "blue");  
  
    swapObjects(red, blue);  
    red.print();  
    blue.print();  
    return 0;  
}
```



함수가 객체를 반환



// 전과동일

Car createCar()

```
{  
    Car tmp(0, 1, "metal");  
    return tmp;  
}
```

int main()

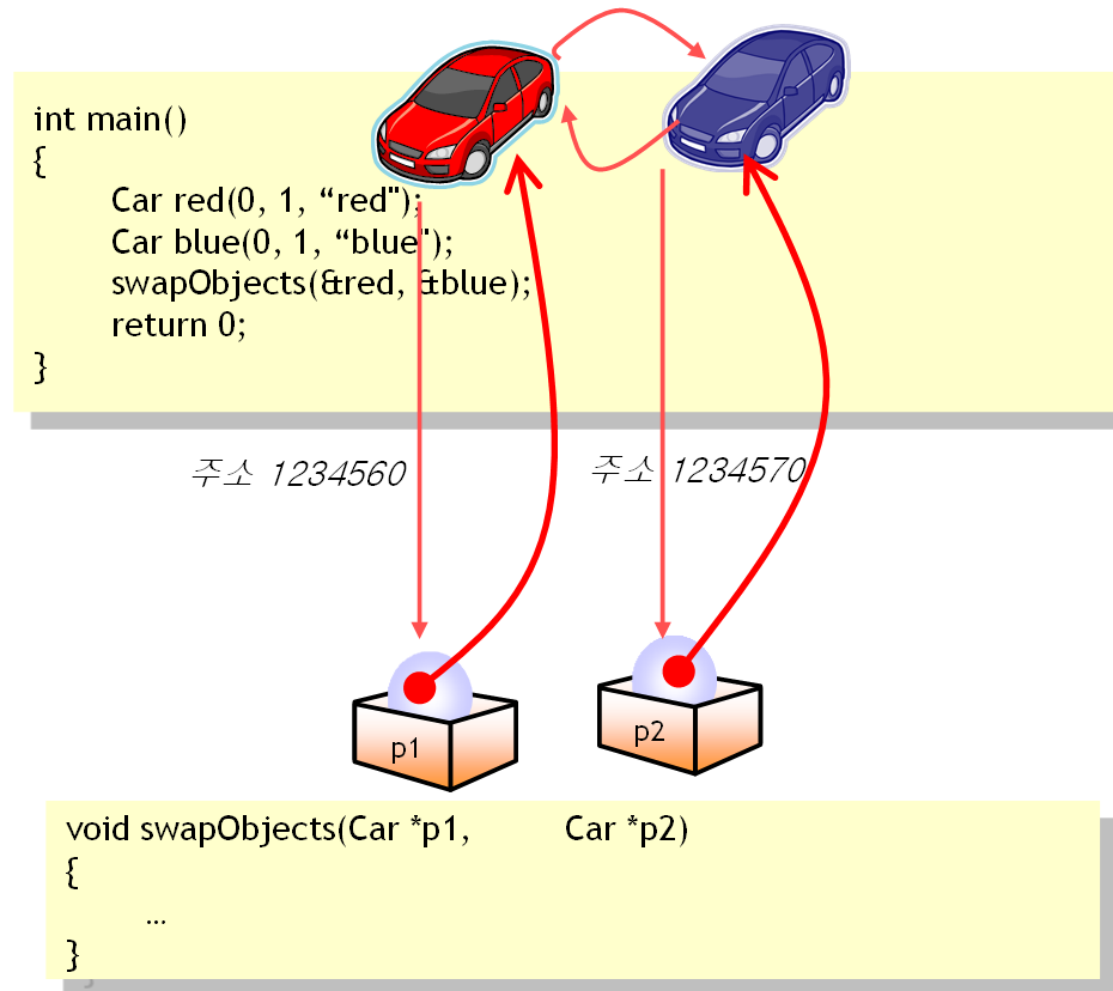
```
{  
    Car c;  
    c.print();  
    c = createCar();  
    c.print();  
  
    return 0;  
}
```



속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: metal



객체의 포인터가 함수에 전달





객체의 포인터가 함수에 전달



...// 전과 동일

```
void swapObjects(Car *p1, Car *p2)
```

```
{
```

```
    Car tmp;
```

```
    tmp = *p1;
```

```
    *p1 = *p2;
```

```
    *p2 = tmp;
```

```
}
```

```
int main()
```

```
{
```

```
    Car red(0, 1, "red");
```

```
    Car blue(0, 1, "blue");
```

```
    swapObjects(&red, &blue);
```

```
    red.print();
```

```
    blue.print();
```

```
    return 0;
```

```
}
```



속도: 0 기어: 1 색상: blue

속도: 0 기어: 1 색상: red



객체의 참조자가 함수에 전달

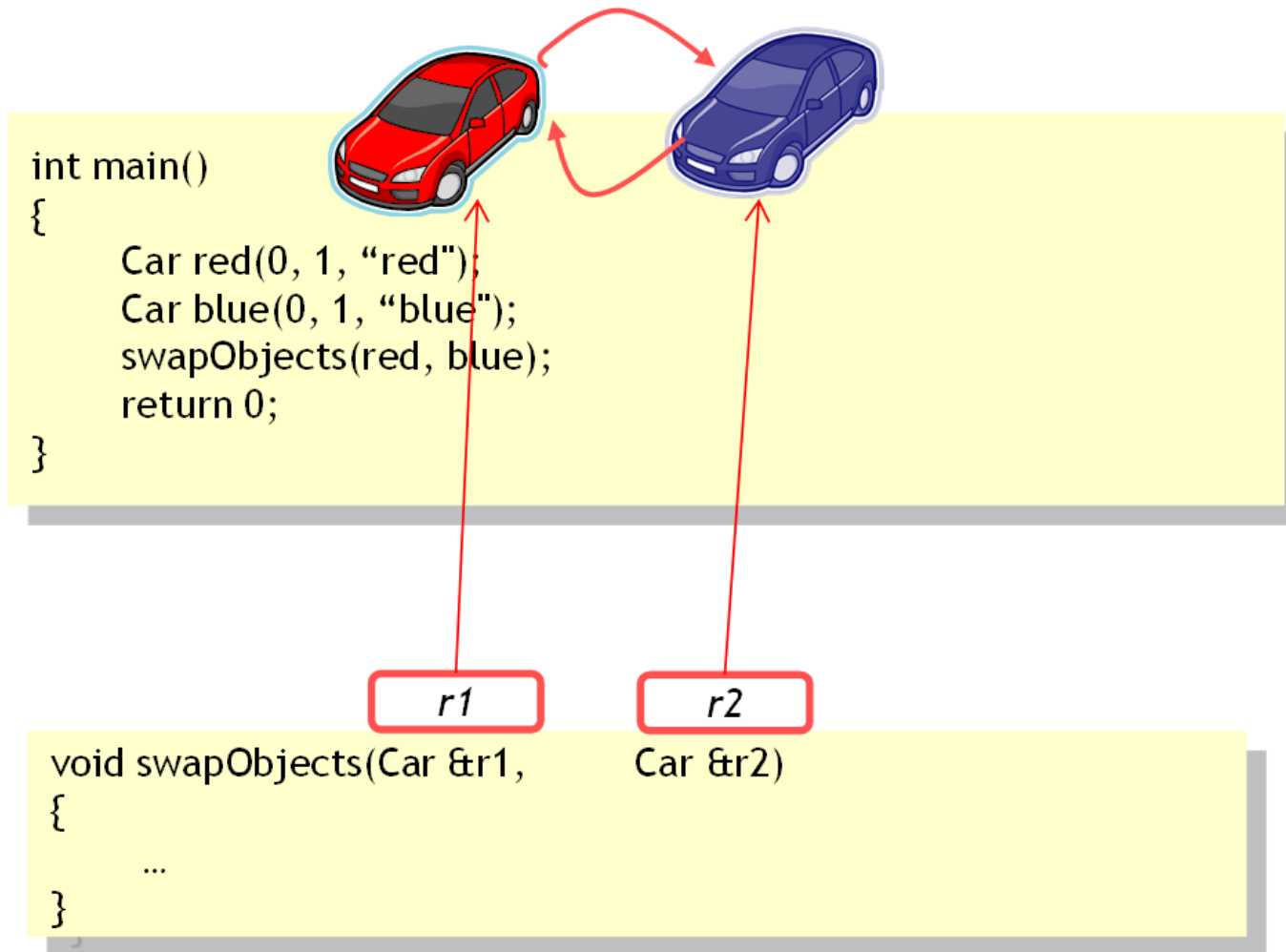


그림 7.5 객체의 참조자를 전달하는 경우



객체의 참조자가 함수에 전달



...// 전과동일

```
void swapObjects(Car &r1, Car &r2)
```

```
{
```

```
    Car tmp;
```

```
    tmp = r1;
```

```
    r1 = r2;
```

```
    r2 = tmp;
```

```
}
```

```
int main()
```

```
{
```

```
    Car red(0, 1, "red");
```

```
    Car blue(30, 2, "blue");
```

```
    swapObjects(red, blue);
```

```
    red.print();
```

```
    blue.print();
```

```
    return 0;
```

```
}
```



속도: 0 기어: 1 색상: blue

속도: 0 기어: 1 색상: red

계속하려면 아무 키나 누르십시오 ...



임시 객체

- 수식의 계산 도중에 중간 결과를 저장하기 위하여 임시적으로 만들어지는 객체

```
int main()
{
    string s1 = "Hello ";
    string s2 = "World";
    const char* p = (s1+s2).c_str();    // ①
    cout << p;

    return 0;
}
```

임시 객체가 생성된다.



임시 객체

- 함수가 객체를 반환하는 경우에도 생성

temp_obj.cpp

```
class Car {  
    ...  
};  
Car createCar()  
{  
    Car tmp(0, 1, "metal");  
    return tmp;  
}
```

```
int main()  
{  
    createCar().print();  
    return 0;  
}
```

반환된 임시객체를 통하
여 멤버 함수 호출



정적 멤버

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

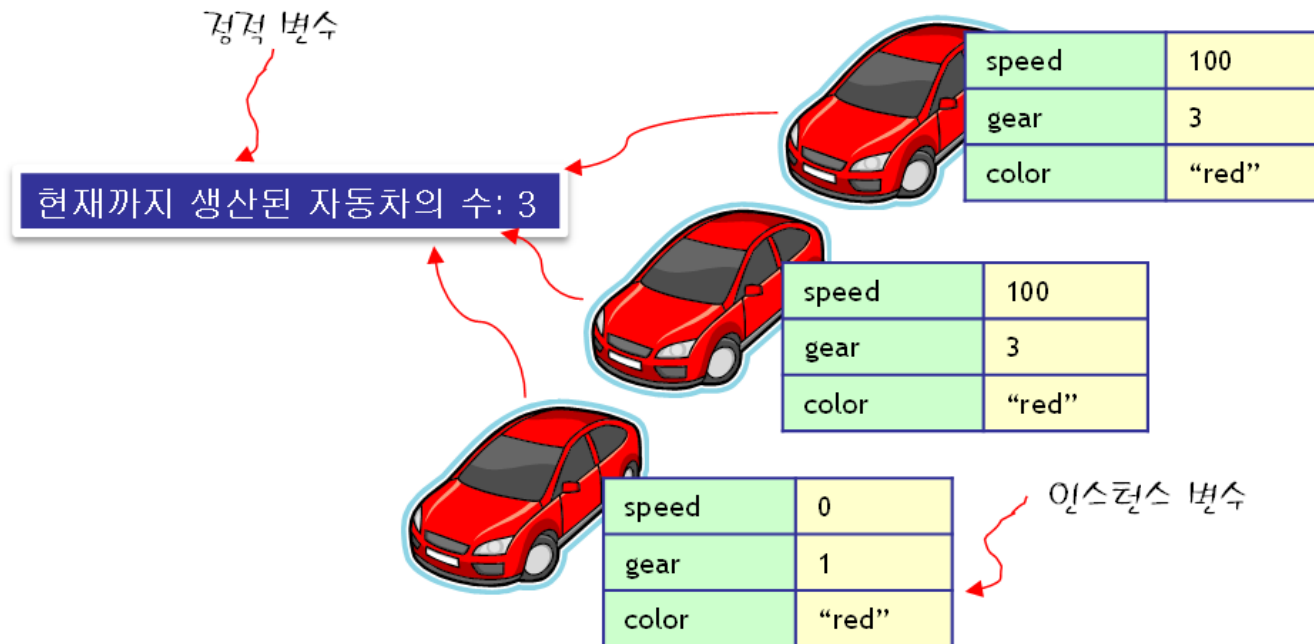


그림 7.6 정적 멤버



정적 멤버 변수



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

class Car {
    int speed;
    int gear;
    string color;

public:
    static int count;
    Car(int s=0, int g=1, string c="white"): speed(s), gear(g), color(c) {
        count++;
    }
    ~Car() {
        count--;
    }
};
```

정적 변수의 선언



정적 멤버 변수



```
int Car::count = 0; // ①
```

정적 변수의 정의

```
int main()
{
    cout << "지금까지 생성된 자동차 수 = " << Car::count << endl; // ②

    Car c1(100, 0, "blue");
    Car c2(0, 0, "white");
    cout << "지금까지 생성된 자동차 수 = " << Car::count << endl; //

    Car c3(0, 0, "red");
    cout << "지금까지 생성된 자동차 수 = " << c1.count << endl; //
    cout << "지금까지 생성된 자동차 수 = " << c2.count << endl; //

    return 0;
}
```



```
지금까지 생성된 자동차 수 = 0
지금까지 생성된 자동차 수 = 2
지금까지 생성된 자동차 수 = 3
지금까지 생성된 자동차 수 = 3
```



정적 멤버 함수

- 정적 멤버 함수는 **static** 수식자를 멤버 함수 선언에 붙인다.
- 클래스 이름을 통하여 호출
- 정적 멤버 함수도 클래스의 모든 객체들이 공유



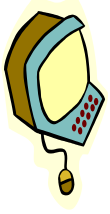
정적 멤버 함수



```
class Car {  
    ...  
public:  
    static int count;    // 정적변수의 선언  
  
    ...  
    // 정적 멤버 함수  
    static int getCount() {  
        return count;  
    }  
};  
int Car::count=0;    // 정적 변수의 정의  
  
int main()  
{  
    Car c1(100, 0, "blue");  
    Car c2(0, 0, "white");  
    int n = Car::getCount();  
    cout << "지금까지 생성된 자동차수= " << n << endl;  
    return 0;  
}
```



정적 멤버 함수



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



주의할 점

- 정적 멤버 함수에서 멤버 변수들은 사용할 수 없다.
- 정적 멤버 함수에서 일반 멤버 함수를 호출하면 역시 오류

```
class Car {  
    int speed;  
    ...  
public:  
    int getSpeed() {  
        return speed;  
    }  
    static int break() {  
        int s = getSpeed();  
        speed = 0;  
        return s;  
    }  
};
```

// 오류: 일반 멤버 함수는 호출할 수 없음

// 오류: 일반 멤버 변수는 접근할 수 없음

정적 멤버 함수에서 일반 멤버
는 사용할 수 없다.



객체들의 배열

- 객체 들의 배열
- 예: `Car objArray[3];`

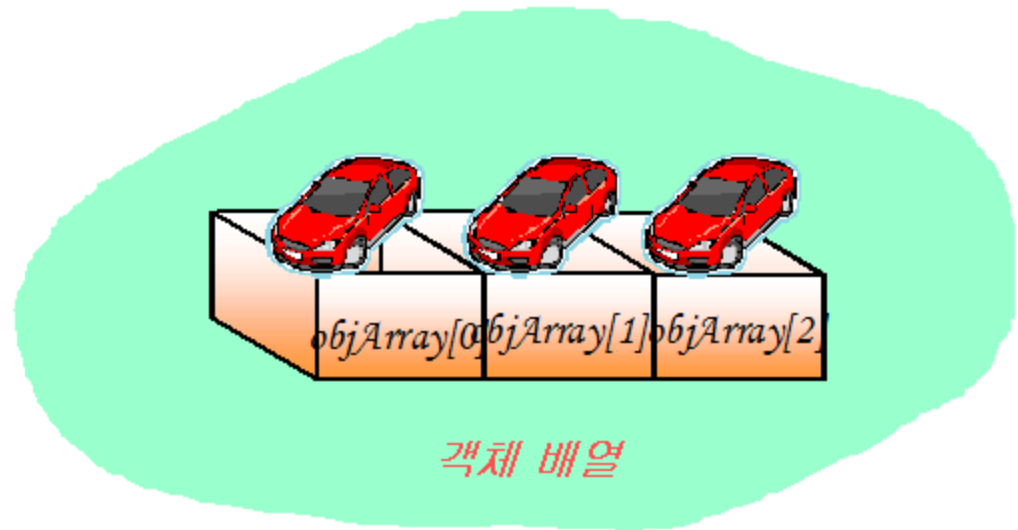


그림 10.8 객체 배열

```
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 print()
    {
        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].print();

    return 0;
}
```



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



객체 배열과 포인터

- 객체 배열의 이름은 포인터처럼 사용될 수 있다.

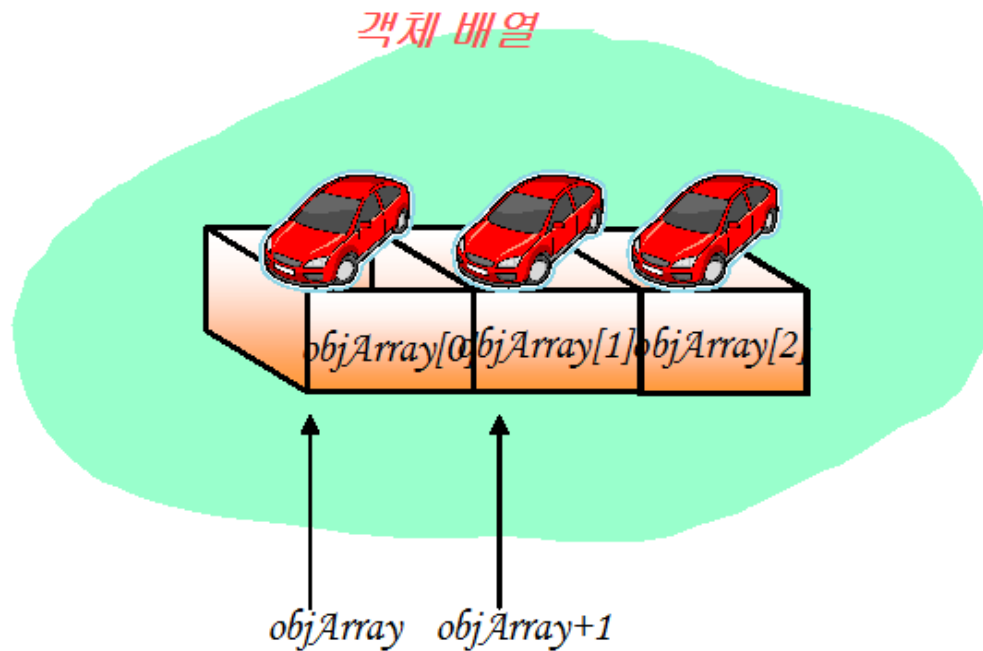


그림 7.8 객체의 이름은 포인터



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)->print();

    Car *p = objArray;
    for(int i=0; i< 3; i++){
        p->print();
        p++;
    }
    return 0;
}
```

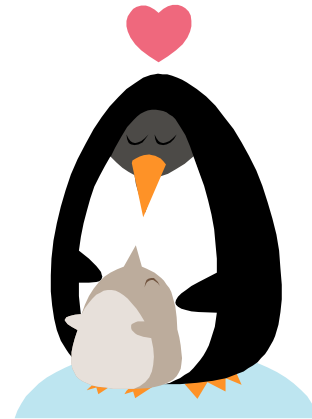


속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: red
속도: 0 기어: 1 색상: blue
속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: red
속도: 0 기어: 1 색상: blue



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.





사용 관계

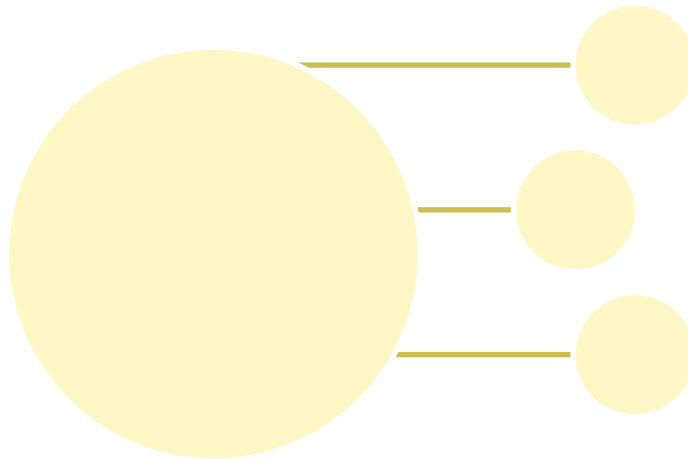
- 클래스 **A**의 멤버 함수에서 클래스 **B**의 멤버 함수들을 호출

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



포함 관계

- 하나의 객체 안에 다른 객체들이 포함





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() // 객체의정보를출력
{
    cout << time << "시" << minute << "분" << second << "초\n";
}

// 알람시계를나타낸다.
class AlarmClock {
private:
    Time currentTime; // 현재시각
    Time alarmTime;   // 알람시각
public:
    AlarmClock(Time a, Time c); // 생성자
    void print();               // 객체의정보출력
};

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



예제



```
void AlarmClock::print()
```

```
{  
    cout << "현재시각: ";  
    currentTime.print();  
    cout << "알람시각: ";  
    alarmTime.print();  
}
```



현재 시각: 12시 56분 34초
알람 시각: 6시 0분 0초

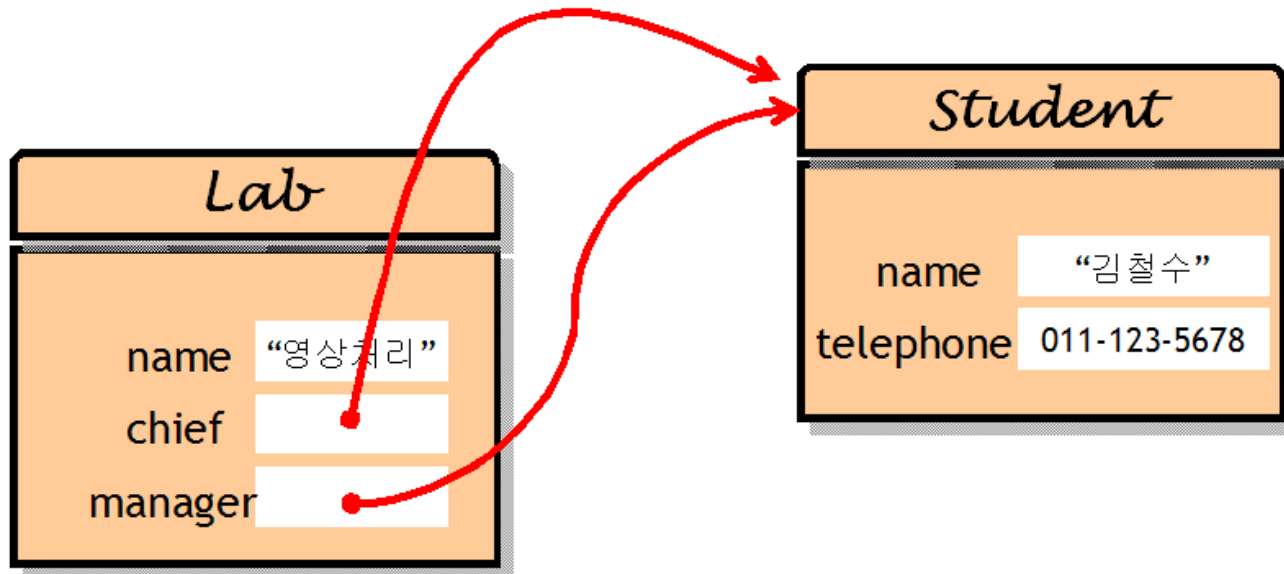
```
int main()
```

```
{  
    Time alarm(6, 0, 0);  
    Time current(12, 56, 34);  
    AlarmClock c(alarm, current);  
  
    c.print();  
    return 0;  
}
```



예제 #1 객체 포인터

- 만약 한 학생이 실험실의 실장과 총무를 겸하는 경우, 객체 포인터를 사용하여 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 << "실장은현재없습니다\n";
    if( manager != NULL )
        cout << "총무는" << manager->getName() << endl;
    else
        cout << "총무는현재없습니다\n";
}
```




객체 포인터



```
int main()
{
    Lab lab("영상처리");
    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();       // 실수부를반환한다.
    double getImag();       // 허수부를반환한다.
    Complex add(const Complex& c); // 복소수의덧셈연산을구현한다.
    void print();           // 복소수를출력한다.
};
```



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;    // 임시객체
    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: $4 + 6i$
 $z = x + y = 6 + 9i$