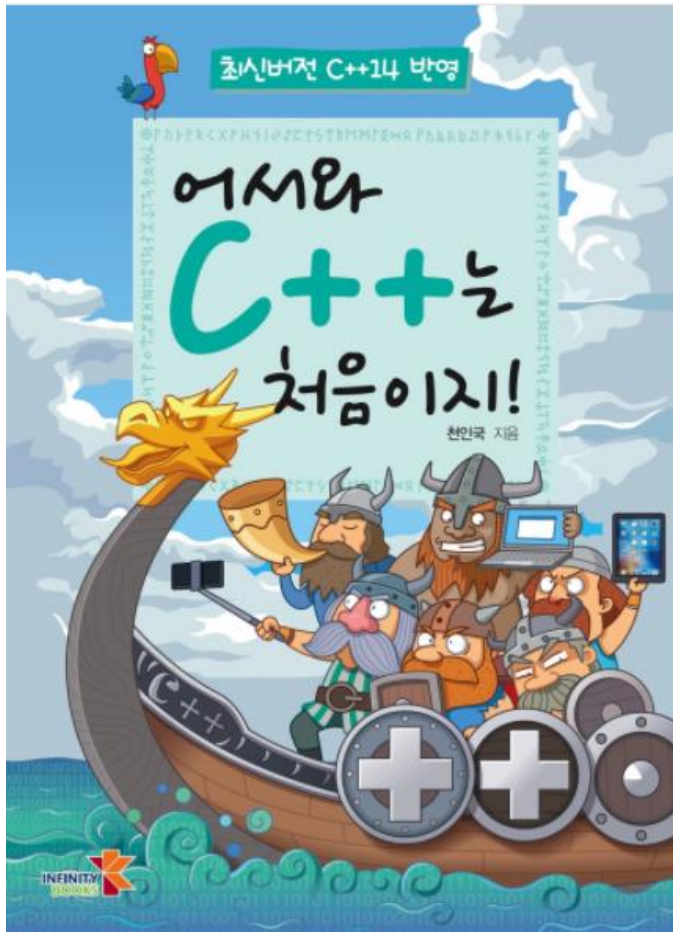


제5장 생성자와 접근제어



- 생성자(contructor): 객체가 생성될 때에 필드에게 초기값을 제공하고 필요한 초기화 절차를 실행하는 멤버 함수



객체의 일생



그림 10.2 객체의 일생

- 클래스 이름과 동일하다
- 반환값이 없다.
- 반드시 **public** 이어야 한다.
- 중복 정의할 수 있다.

```
class Car
{
    ...
    public:
        Car()
        {
            ...
        }
};
```

생성자



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

```
class Car {
private:
    int speed; // 속도
    int gear; // 기어
    string color; // 색상
```

```
public:
```

```
    Car()
    {
```

```
        cout << "디폴트 생성자 호출" << endl;
        speed = 0;
        gear = 1;
        color = "white";
    }
```

```
};
```

```
int main()
```

```
{
```

```
    Car c1; // 디폴트생성자호출
```

```
    return 0;
```

```
}
```



speed	0
gear	1
color	"white"

↑
c1

생성자의 외부 정의

```
Car::Car()
{
    cout << "디폴트 생성자 호출" << endl;
    speed = 0;
    gear = 1;
    color = "white";
}
```

매개 변수를 가지는 생성자



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

class Car {
private:  int speed;           // 속도
         int gear;            // 기어
         string color;        // 색상

public:  Car(int s, int g, string c)
        {
            speed = s;
            gear = g;
            color = c;
        }

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

매개 변수를 가지는 생성자



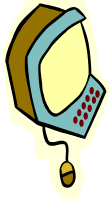
```
#include <iostream>
#include <string>
using namespace std;

class Car {
private:
    int speed;           // 속도
    int gear;            // 기어
    string color;        // 색상
public:
    Car(int s, int g, string c)
    {
        speed = s;
        gear = g;
        color = c;
    }
    void printInfo();
};
```


매개 변수를 가지는 생성자



```
int main()
{
    Car c1(0, 1, "red");           // 생성자 호출
    Car c2(0, 1, "blue");         // 생성자 호출
    c1.print();
    c2.print();
    return 0;
}
```



```
=====
속도: 0
기어: 1
색상: red
=====
=====
속도: 0
기어: 1
색상: blue
=====
```



c1

speed	0
gear	1
color	"red"



c2

speed	0
gear	1
color	"blue"

생성자의 중복 정의

- 생성자도 메소드이므로 중복 정의가 가능하다.



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

class Car {
private:
    int speed;           // 속도
    int gear;            // 기어
    string color;        // 색상
public:
    Car();
    Car(int s, int g, string c) ;
};
```

생성자의 중복 정의



매개 변수가 있는 생성자 호출
디폴트 생성자 호출
계속하려면 아무 키나 누르십시오 ...

```
Car::Car()
{
    cout << "디폴트 생성자 호출<< endl;
    speed = 0;
    gear = 1;
    color = "white"
}
Car::Car(int s, int g, string c)
{
    cout << "매개변수가 있는 생성자 호출<< endl;
    speed = s;
    gear = g;
    color = c;
}
int main()
{
    Car c1;                // 디폴트 생성자 호출
    Car c2(100, 0, "blue"); // 매개변수가 있는 생성자 호출
    return 0;
}
```

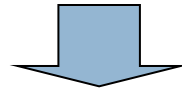
생성자 호출의 다양한 방법

```
int main()
{
    Car c1;    // ①디폴트 생성자 호출
    Car c2();  // ②이것은 생성자 호출이 아니라 c2()라는 함수의 원형 선언
    Car c3(100, 3, "white");    // ③생성자 호출
    Car c4 = Car(0, 1, "blue");// ④이것은 먼저 임시 객체를 만들고 이것을
                                c4에 복사
    Car c4{100, 3, "white"};    // 생성자 호출(최신 버전)

    return 0;
}
```

생성자를 하나도 정의하지 않으면?

```
class Car {  
    int speed;      // 속도  
    int gear;       // 기어  
    string color;   // 색상  
};
```



컴파일러가 비어있는 디폴트 생성자를 자동으로 추가한다.

```
class Car {  
    int speed;      // 속도  
    int gear;       // 기어  
    string color;   // 색상  
public:  
    Car() { }  
}
```

```
Car(int s=0, int g=1, string c="red")  
{  
    speed = s;  
    gear = g;  
    color = c;  
}
```

디폴트 생성자를 정의한 것과 같은 효과를 낸다.

생성자에서 다른 생성자 호출하기



```

...
class Car {
    int speed; // 속도
    int gear; // 기어
    string color; // 색상
public:
    // 첫 번째 생성자
    Car(int s, int g, string c) {
        speed = s;
        gear = g;
        color = c;
    }
    // 색상만 주어진 생성자
    Car(string c) {
        Car(0, 0, c); // 첫 번째 생성자를 호출한다.
    }
};
int main()
{
    Car c1("white");
    return 0;
}
  
```

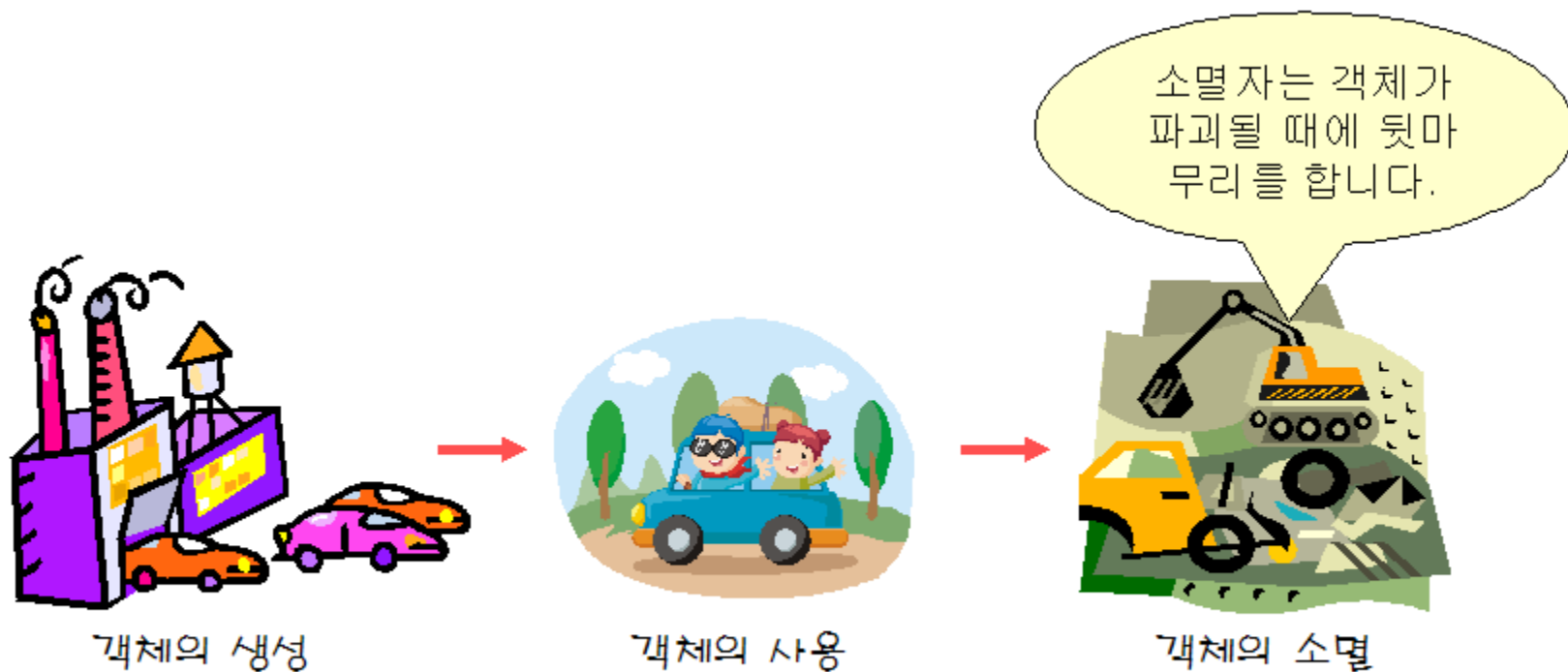


그림 10.4 소멸자의 개념

- 소멸자는 클래스 이름에 ~가 붙는다.
- 값을 반환하지 않는다.
- **public** 멤버 함수로 선언된다.
- 소멸자는 매개 변수를 받지 않는다.
- 중복 정의도 불가능하다.

```
class Car
{
    ...
    public:
        ~Car()
        {
            ...
        }
};
```

소멸자



```
class Car {
private:
    int speed;           // 속도
    int gear;            // 주행거리
    string color;        // 색상
```

public:

```
    Car()
    {
        cout << "생성자 호출" << endl;
        speed = 0;
        gear = 1;
        color = "white";
    }
```

생성자

```
    ~Car()
    {
        cout << "소멸자 호출" << endl;
    }
```

소멸자

```
};
int main()
{
    Car c1;
    return 0;
}
```



생성자 호출
소멸자 호출

- 만약 프로그래머가 소멸자를 정의하지 않았다면 어떻게 되는가?
- 디폴트 소멸자가 자동으로 삽입되어서 호출된다

```
class Time {  
    int hour, minute, second;  
public:  
    print() { ... }  
}
```

~Time()을 넣어준다.


```
#include <string.h>

class MyString {
private:
    char *s;
    int size;
public:
    MyString(char *c) {
        size = strlen(c)+1;
        s = new char[size];
        strcpy(s, c);
    }
    ~MyString() {
        delete[] s;
    }
};

int main() {
    MyString str("abcdefghijkl");
}
```

- 멤버 변수를 간단히 초기화할 수 있는 형식

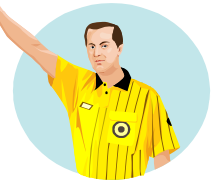
```
Car(int s, int g, string c) : speed(s), gear(g), color(c) {  
    ...// 만약 더 하고 싶은 초기화가 있다면 여기에  
}
```



- 멤버가 상수인 경우에는 어떻게 초기화하여야 하는가?

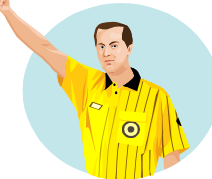
```
class Car
{
    const int MAX_SPEED = 300;
    int speed;
    ...
}
```

아직 생성이 안됐음!



```
class Car
{
    const int MAX_SPEED;
    int speed; // 속도
public:
    Car()
    {
        MAX_SPEED = 300;
    }
}
```

상수를 변경할 수 없음!



상수 멤버의 초기화

```
class Car
{
    const int MAX_SPEED;
    int speed;        // 속도
public:
    Car() : MAX_SPEED(300)
    {
    }
};
```

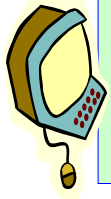
상수 멤버의 초기화는
이렇게,

참조자 멤버의 초기화



```
#include <iostream>
#include <string>
using namespace std;
class Car
{
    string& alias;
    int speed; // 속도
public:
    Car(string s) : alias(s)
    {
        cout << alias << endl;
    }
};

int main()
{
    Car c1("꿈의 자동차");
    return 0;
}
```



꿈의 자동차
계속하려면 아무 키나 누르십시오 ...

객체 멤버의 경우

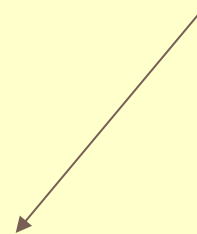


```
#include <iostream>
#include <string>
using namespace std;
```

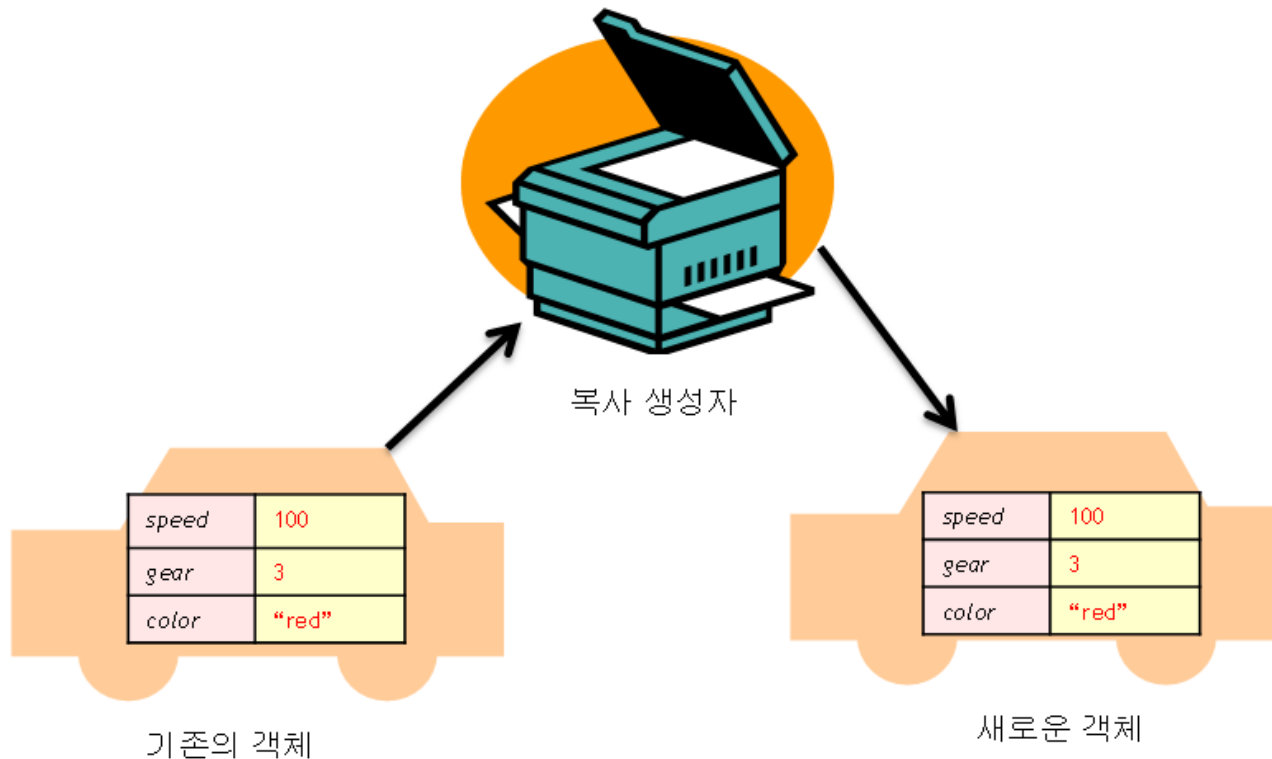
```
class Point
{
    int x, y;
public:
    Point(int a, int b) : x(a), y(b)
    {
    }
};
```

```
class Rectangle
{
    Point p1, p2;
public:
    Rectangle(int x1, int y1, int x2, int y2) : p1(x1, y2), p2(x2, y2)
    {
    }
};
```

생성자 호출



- 한 객체의 내용을 다른 객체로 복사하여서 생성



복사 생성자의 특징

- 자동으로 디폴트 복사 생성자가 생성된다.
- 자신과 같은 타입의 객체를 매개 변수로 받는다.

```
Car(Car& obj);  
Car(const Car& obj);
```

복사 생성자



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

class Car {
    int speed; // 속도
    int gear; // 기어
    string color; // 색상

public:
    Car(int s, int g, string c) : speed(s), gear(g), color(c)
    {
        cout << "생성자 호출" << endl;
    }
    Car(const Car &obj) : speed(obj.speed), gear(obj.gear), color(obj.color)
    {
        cout << "복사 생성자 호출" << endl;
    }
    ...
};
```

복사 생성자



```
int main()
```

```
{
```

```
    Car c1(0, 1, "yellow");
```

```
    Car c2(c1);
```

```
    c1.print();
```

```
    c2.print();
```

```
    return 0;
```

```
}
```

복사 생성자 호출



```
=====
```

```
속도: 0
```

```
기어: 1
```

```
색상: yellow
```

```
=====
```

```
=====
```

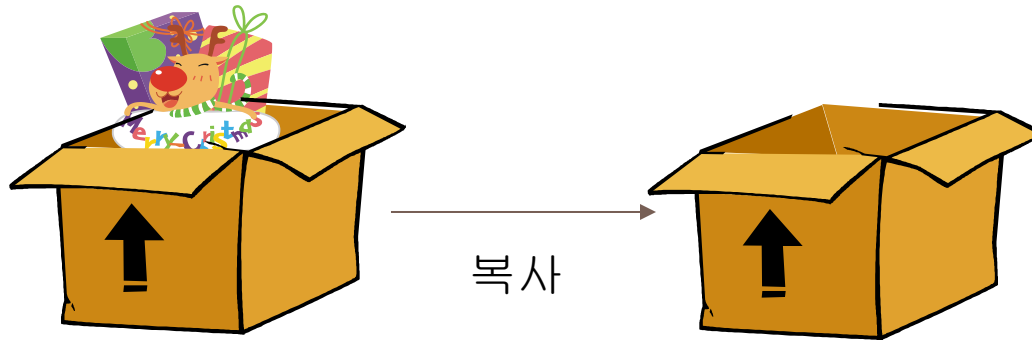
```
속도: 0
```

```
기어: 1
```

```
색상: yellow
```

```
=====
```

- 멤버의 값만 복사하면 안되는 경우가 발생한다.
- 얕은 복사(shallow copy) 문제



예제



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

class Student {
    char *name; // 이름
    int number;
public:
    Student(char *p, int n) {
        cout << "메모리 할당" << endl;
        name = new char[strlen(p)+1];
        strcpy(name, p);
        number = n;
    }
    ~Student() {
        cout << "메모리 소멸" << endl;
        delete [] name;
    }
};

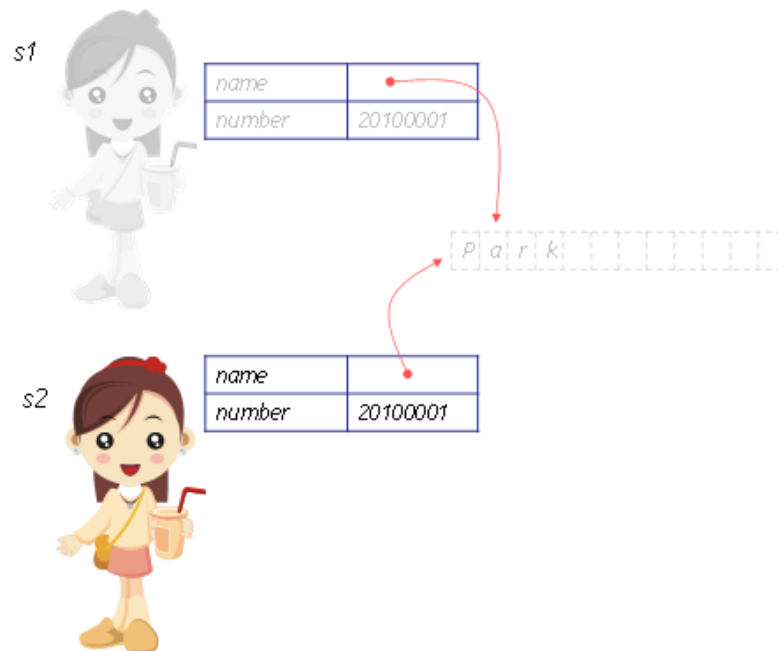
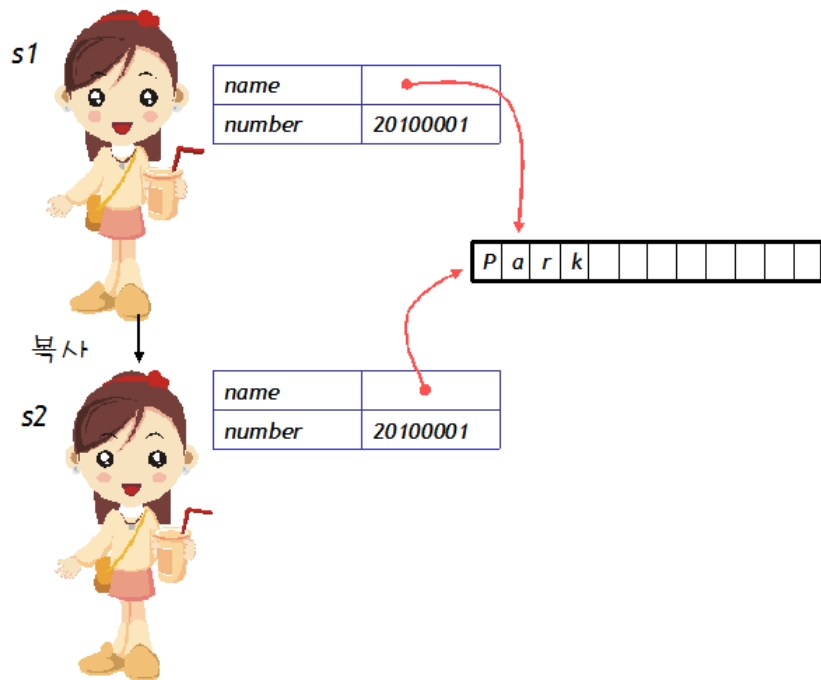
int main()
{
    Student s1("Park", 201000001);
    Student s2(s1); // 복사 생성자 호출
    return 0;
}
```

예제



메모리 할당
메모리 소멸
메모리 소멸

메모리의 소멸이 2번
호출되었음



이름을 저장하는 동적
메모리 공간이 별도로
할당되지 않았음



```
class Student {  
    ....  
    Student(const Student& s) {  
        cout << "메모리 할당<< endl;  
        name = new char[strlen(s.name)+1];  
        strcpy(name, s.name);  
        number = s.number;  
    }  
    ....  
};
```



메모리 할당
메모리 할당
메모리 소멸
메모리 소멸

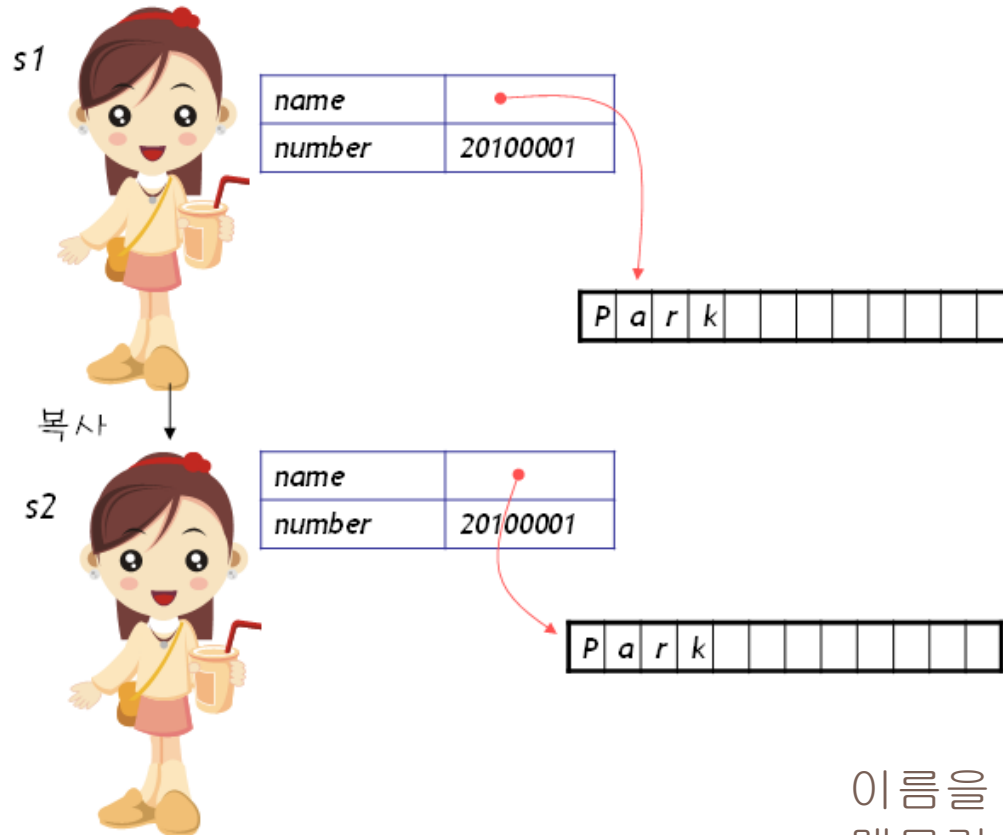


그림 6.9 깊은 복사

이름을 저장하는 동적
메모리 공간을 별도로
할당

복사 생성자가 호출되는 경우

- 기존의 객체의 내용을 복사하여서 새로운 객체를 만드는 경우
- 객체를 값으로 매개 변수로 전달하는 경우
- 객체를 값으로 반환하는 경우



복사 생성자

생각보다 많이
사용됩니다.

예제



```
class Car {  
...  
    Car(const Car &obj) : speed(obj.speed), gear(obj.gear), color(obj.color)  
    {  
        cout << "복사 생성자 호출" << endl;  
    }  
...  
};  
void isMoving(Car obj)  
{  
    if( obj.getSpeed() > 0 )  
        cout << "움직이고 있습니다" << endl;  
    else  
        cout << "정지해 있습니다" << endl;  
}  
int main()  
{  
    Car c(0, 1, "white");  
    isMoving(c);  
    return 0;  
}
```



일반 생성자 호출
복사 생성자 호출
정지해 있습니다



복사 생성자는
어디에서
호출되었을까?

디폴트 멤버 함수

자동으로 추가된다.

- 디폴트 생성자
- 디폴트 소멸자
- 디폴트 복사 생성자
- 디폴트 할당 연

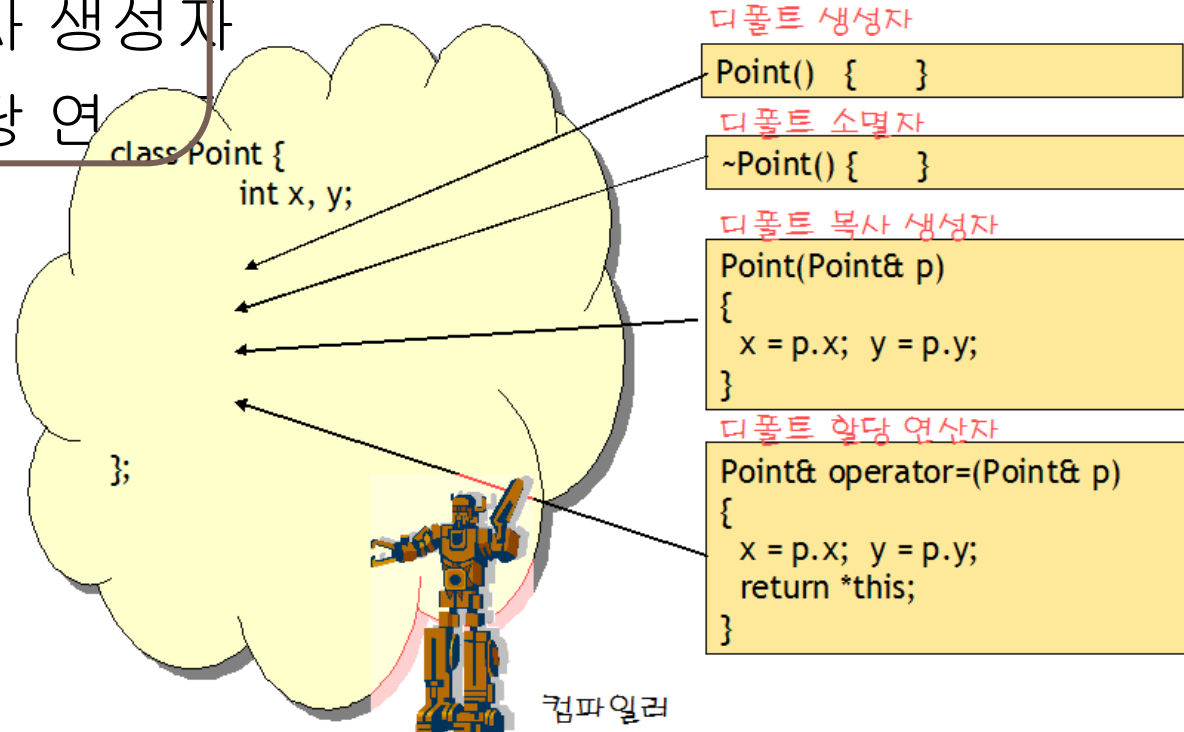
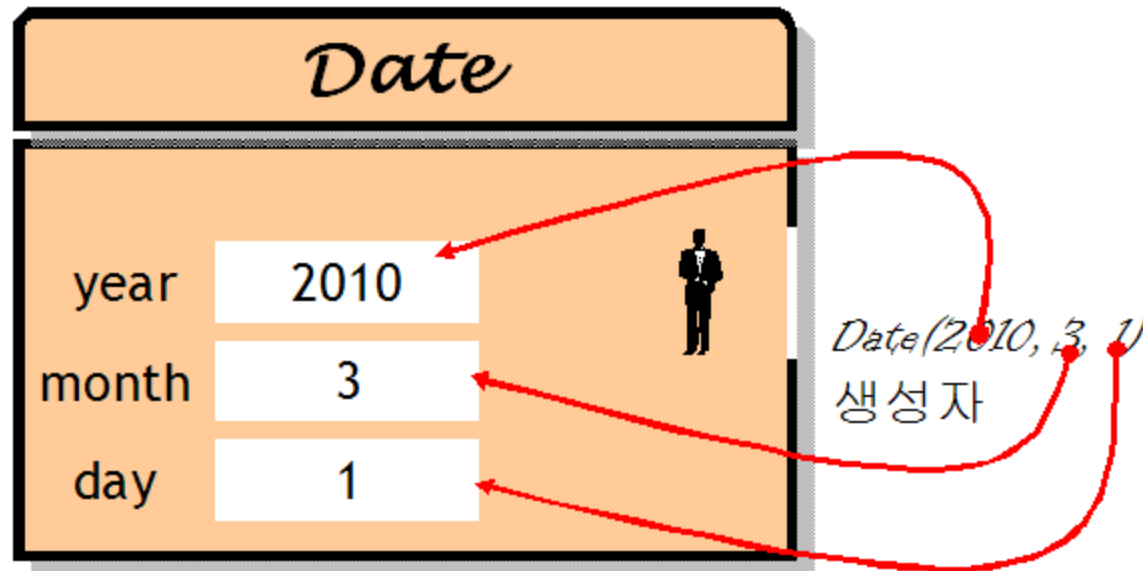


그림 10.7 디폴트 멤버 함수의 추가

- Date 클래스에 생성자와 소멸자를 추가



예제



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

class Date {
private:
    int year;
    int month;
    int day;
public:
    Date();
    Date(int year);
    Date(int year, int month, int day);
    void setDate(int year, int month, int day);
    void print();
};

Date::Date() // 디폴트생성자
{
    year = 2010;
    month = 1;
    day = 1;
}
```

예제



```
Date::Date(int year) // 생성자
{
    setDate(year, 1, 1);
}
Date::Date(int year, int month, int day) // 생성자
{
    setDate(year, month, day);
}
void Date::setDate(int year, int month, int day)
{
    this->month = month;           // this는 현재 객체를 가리킨다.
    this->day = day;
    this->year = year;
}
void Date::print()
{
    cout << year << "년" << month << "월" << day << "일" << endl;
}
```

예제



```
int main()
{
    Date date1(2009, 3, 2);    // 2009.3.2
    Date date2(2009);          // 2009.1.1
    Date date3;                // 2010.1.1
    date1.print();
    date2.print();
    date3.print();
    return 0;
}
```



2009년 3월 2일
2009년 1월 1일
2010년 1월 1일

- Time 클래스에 생성자와 소멸자를 추가



예제



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

class Time {
private:
    int hour; // 0 - 23
    int minute; // 0 - 59
    int second; // 0 - 59

public:
    Time();
    // 생성자
    Time(int h, int m, int s);
    void setTime(int h, int m, int s);
    void print();
};
// 첫번째 생성자
Time::Time()
{
    setTime(0, 0, 0);
}
```

예제



```
// 두번째 생성자
Time::Time(int h, int m, int s)
{
    setTime(h, m, s);
}

// 시간설정 함수
void Time::setTime(int h, int m, int s)
{
    hour = ((h >= 0 && h < 24) ? h : 0); // 시간검증
    minute = ((m >= 0 && m < 60) ? m : 0); // 분검증
    second = ((s >= 0 && s < 60) ? s : 0); // 초검증
}

// "시:분:초"의 형식으로 출력
void Time::print()
{
    cout << hour << ":" << minute << ":" << second << endl;
}
```

예제



```
int main()
{
    Time time1;

    cout << "기본생성자호출후시간: ";
    time1.print();

    // 두번째생성자호출
    Time time2(13, 27, 6);
    cout << "두번째생성자호출후시간: ";
    time2.print();

    // 올바른지않은시간으로설정해본다.
    Time time3(99, 66, 77);
    cout << "올바르지않은시간설정후시간: ";
    time3.print();

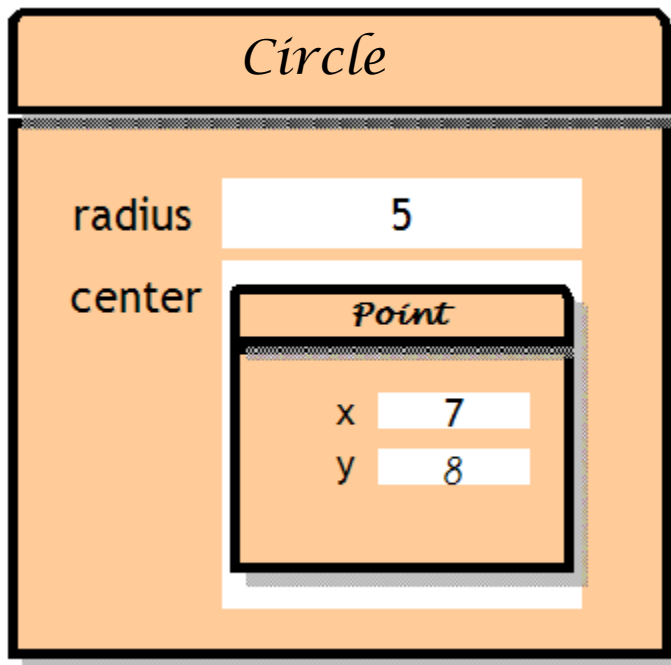
    return 0;
}
```

예제



기본 생성자 호출 후 시간: 0:0:0
두번째 생성자 호출 후 시간: 13:27:6
올바르지 않은 시간 설정 후 시간: 0:0:0

- Circle 객체 안에 Point 객체가 들어 있는 경우



예제



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

class Point {
private:
    int x;
    int y;
public:
    Point();
    Point(int a, int b);
    void print();
};

Point::Point() : x(0), y(0)
{
}

Point::Point(int a, int b) : x(a), y(b)
{
}
```

예제



```
void Point::print()
{
    cout << "(" << x << ", " << y << " )\n";
}

class Circle {
private:
    int radius;
    Point center; // Point 객체가 멤버변수로 선언되어 있다.
public:
    Circle();
    Circle(int r);
    Circle(Point p, int r);
    Circle(int x, int y, int r);
    void print();
};

// 생성자
Circle::Circle(): radius(0), center(0, 0)
{
}
```

예제



```
Circle::Circle(int r) : radius(r), center(0, 0)
{
}
Circle::Circle(Point p, int r) : radius(r), center(p)
{
}
Circle::Circle(int x, int y, int r) : radius(r), center(x, y)
{
}
void Circle::print()
{
    cout << "중심: ";
    center.print();
    cout << "반지름: " << radius << endl << endl;
}
```

예제



```
int main()
{
    Point p(5, 3);

    Circle c1;
    Circle c2(3);
    Circle c3(p, 4);
    Circle c4(9, 7, 5);

    c1.print();
    c2.print();
    c3.print();
    c4.print();
    return 0;
}
```



중심: (0, 0)
반지름: 0
중심: (0, 0)
반지름: 3
중심: (5, 3)
반지름: 4
중심: (9, 7)
반지름: 5

객체와 함수

- 객체가 함수의 매개 변수로 전달되는 경우
- 객체의 참조자가 함수의 매개 변수로 전달되는 경우
- 함수가 객체를 반환하는 경우

객체가 함수의 매개 변수로 전달되는 경우

- 값을 전달한다.
- 어떤 피자 체인점에서 미디엄 크기의 피자를 주문하면 무조건 라지 피자로 변경해준다고 하자. 다음과 같이 프로그램을 작성하면 피자의 크기가 커질까?

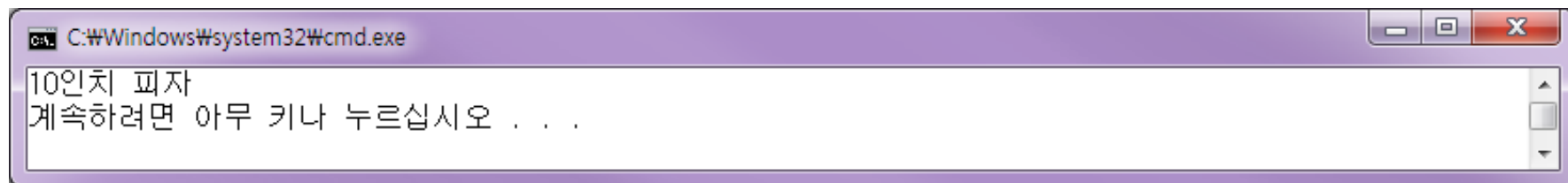
```
#include <iostream>
using namespace std;

class Pizza {
public:
    Pizza(int s) : size(s) { }
    int size;                // 단위: 인치
};

void makeDouble(Pizza p){
    p.size *= 2;
}

int main(){
    Pizza pizza(10);
    makeDouble(pizza);
    cout << pizza.size << "인치 피자" << endl;
    return 0;
}
```


실행결과



C:\Windows\system32\cmd.exe

```
10인치 피자  
계속하려면 아무 키나 누르십시오 . . .
```

객체의 참조자가 함수의 매개 변수로 전달되는 경우

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

class Pizza {
public:
    Pizza(int s) : size(s) {}
    int size;                // 단위: 인치
};

void makeDouble(Pizza& p) {
    p.size *= 2;
}

int main() {
    Pizza pizza(10);
    makeDouble(pizza);
    cout << pizza.size << "인치 피자" << endl;

    return 0;
}
```

실행결과



```
C:\Windows\system32\cmd.exe
```

20인치 피자
계속하려면 아무 키나 누르십시오 . . .

함수가 객체를 반환하는 경우

- 함수가 객체를 반환할 때도 객체의 내용이 복사될 뿐 원본이 전달되지 않는다.

예제

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

class Pizza {
public:
    Pizza(int s) : size(s) {}
    int size;                // 단위: 인치
};

Pizza createPizza() {
    Pizza p(10);
    return p;
}

int main() {
    Pizza pizza = createPizza();
    cout << pizza.size << "인치 피자" << endl;

    return 0;
}
```

실행결과



```
C:\Windows\system32\cmd.exe
```

10인치 피자
계속하려면 아무 키나 누르십시오 . . .