

06

함수 중복과 static 멤버

■ 함수 중복

- 동일한 이름의 함수가 공존
 - 다형성
 - C 언어에서는 불가능
- function overloading
- 함수 중복이 가능한 범위
 - 보통 함수들 사이
 - 클래스의 멤버 함수들 사이
 - 상속 관계에 있는 기본 클래스와 파생 클래스의 멤버 함수들 사이

■ 함수 중복 성공 조건

- 중복된 함수들의 이름 동일
- 중복된 함수들의 매개 변수 타입이 다르거나 개수가 달라야 함
- 리턴 타입은 함수 중복과 무관

```
int sum(int a, int b, int c) {  
    return a + b + c;  
}
```

```
double sum(double a, double b) {  
    return a + b;  
}
```

```
int sum(int a, int b) {  
    return a + b;  
}
```

성공적으로 중복된 sum() 함수들

```
int main(){  
    cout << sum(2, 5, 33);
```

```
    cout << sum(12.5, 33.6);
```

```
    cout << sum(2, 6);  
}
```

중복된 sum() 함수 호출.
컴파일러가 구분

- 리턴 타입이 다르다고 함수 중복이 성공하지 않는다.

```
int sum(int a, int b) {  
    return a + b;  
}  
double sum(int a, int b) {  
    return (double)(a + b);  
}
```

함수 중복 실패

```
int main() {  
    cout << sum(2, 5);  
}
```

컴파일러는 어떤 sum() 함수를
호출하는지 구분할 수 없음

- 동일한 이름을 사용하면 함수 이름을 구분하여 기억할 필요 없고, 함수 호출을 잘못하는 실수를 줄일 수 있음

```
void msg1() {  
    cout << "Hello";  
}  
void msg2(string name) {  
    cout << "Hello, " << name;  
}  
void msg3(int id, string name) {  
    cout << "Hello, " << id << " " << name;  
}
```

(a) 함수 중복하지 않는 경우



```
void msg() {  
    cout << "Hello";  
}  
void msg(string name) {  
    cout << "Hello, " << name;  
}  
void msg(int id, string name) {  
    cout << "Hello, " << id << " " << name;  
}
```

(b) 함수 중복한 경우

함수 중복하면 함수 호출의
편리함.오류 가능성 줄임

예제 6-1 big() 함수 중복 연습

6

큰 수를 리턴하는 다음 두 개의 big 함수를 중복 구현하라.

```
int big(int a, int b);           // a와 b 중 큰 수 리턴
int big(int a[], int size);      // 배열 a[]에서 가장 큰 수 리턴
```

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

int big(int a, int b) {          // a와 b 중 큰 수 리턴
    if(a>b) return a;
    else return b;
}

int big(int a[], int size) {     // 배열 a[]에서 가장 큰 수 리턴
    int res = a[0];
    for(int i=1; i<size; i++)
        if(res < a[i]) res = a[i];
    return res;
}

int main() {
    int array[5] = {1, 9, -2, 8, 6};
    cout << big(2,3) << endl;
    cout << big(array, 5) << endl;
}
```

3
9

예제 6-2(실습) sum() 함수 중복 연습

함수 sum()을 호출하는 경우가 다음과 같을 때, 함수 sum()을 중복 구현하라. sum()의 첫 번째 매개 변수는 두 번째 매개 변수보다 작은 정수 값으로 호출된다고 가정한다.

```
sum(3,5);           // 3~5까지의 합을 구하여 리턴
sum(3);             // 0~3까지의 합을 구하여 리턴
sum(100);           // 0~100까지의 합을 구하여 리턴
```

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

int sum(int a, int b) { // a에서 b까지 합하기
    int s = 0;
    for(int i=a; i<=b; i++)
        s += i;
    return s;
}

int sum(int a) {        // 0에서 a까지 합하기
    int s = 0;
    for(int i=0; i<=a; i++)
        s += i;
    return s;
}

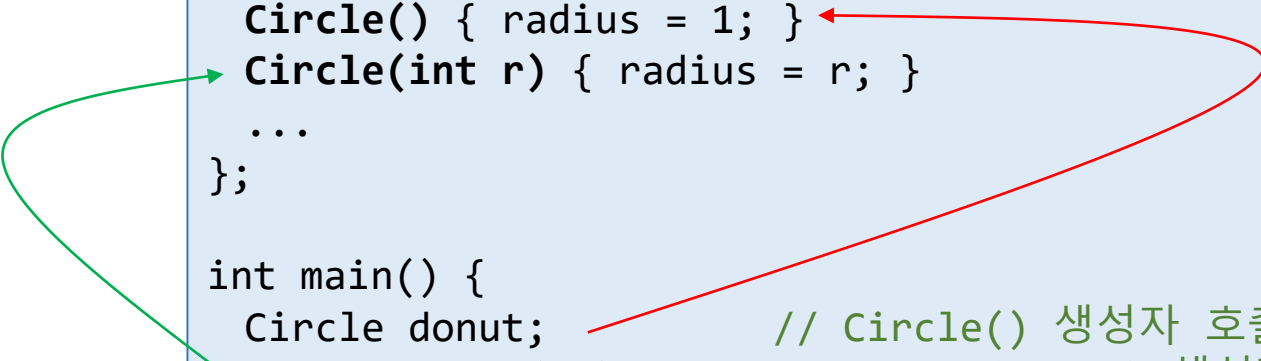
int main() {
    cout << sum(3, 5) << endl;
    cout << sum(3) << endl;
    cout << sum(100) << endl;
}
```

12
6
5050

■ 생성자 함수 중복 가능

- 생성자 함수 중복 목적
 - 객체 생성시, 매개 변수를 통해 다양한 형태의 초깃값 전달

```
class Circle {  
    .....  
public:  
    Circle() { radius = 1; }  
    Circle(int r) { radius = r; }  
    ...  
};  
  
int main() {  
    Circle donut;           // Circle() 생성자 호출  
    Circle pizza(30);       // Circle(int r) 생성자 호출  
}
```




```
class string {  
    .....  
public:  
    string();           // 빈 문자열을 가진 스트링 객체 생성  
    string(char* s);    // '\0'로 끝나는 C-스트링 s를 스트링 객체로 생성  
    string(string& str); // str을 복사한 새로운 스트링 객체 생성  
    .....  
};
```

```
string str;           // 빈 문자열을 가진 스트링 객체  
string address("서울시 성북구 삼선동 389");  
string copyAddress(address); // address의 문자열을 복사한 별도의 copyAddress 생성
```

■ 소멸자 함수 중복 불가

- 소멸자는 매개 변수를 가지지 않음
- 한 클래스 내에서 소멸자는 오직 하나만 존재

■ 디폴트 매개 변수(default parameter)

- 매개 변수에 값이 넘어오지 않는 경우, 디폴트 값을 받도록 선언된 매개 변수
 - '매개 변수 = 디폴트값' 형태로 선언

■ 디폴트 매개 변수 선언 사례

■ 디폴트 매개 변수를 가진 함수 호출

```
void star(int a=5); // a의 디폴트 값은 5
```

```
star(); // 매개 변수 a에 디폴트 값 5가 전달됨. star(5);와 동일  
star(10); // 매개 변수 a에 10을 넘겨줌
```

■ 사례 1

```
void msg(int id, string text="Hello"); // text의 디폴트 값은 "Hello"
```

```
msg(10); // msg(10, "Hello"); 호출과 동일. id에 10, text에 "Hello" 전달
```

```
msg(20, "Good Morning"); // id에 20, text에 "Good Morning" 전달
```

```
msg(); // 컴파일 오류. 첫 번째 매개 변수 id에 반드시 값을 전달하여야 함
```

```
msg("Hello"); // 컴파일 오류. 첫 번째 매개 변수 id에 값이 전달되지 않았음
```

호출 오류

■ 디폴트 매개 변수는 보통 매개 변수 앞에 선언될 수 없음

- 디폴트 매개 변수는 끝 쪽에 몰려 선언되어야 함

컴파일 오류

```
void calc(int a, int b=5, int c, int d=0);    // 컴파일 오류
void sum(int a=0, int b, int c);              // 컴파일 오류

void calc(int a, int b=5, int c=0, int d=0);  // 컴파일 성공
```

■ 사례 2

```
void square(int width=1, int height=1);
```

디폴트 매개 변수를 가진
square()

```
void square(int width=1, int height=1);
```

square();



square(__, __);



square(**1**, **1**);

square(5);



square(5, __);



square(5, **1**);

square(3, 8);



square(3, 8);



square(3, 8);

컴파일러에 의해 변환
되는 과정

■ 사례 3

```
void g(int a, int b=0, int c=0, int d=0);
```

디폴트 매개 변수를 가진 함수

```
void g(int a, int b=0, int c=0, int d=0);
```

<code>g(10);</code>	\longrightarrow	<code>g(10, <u> </u> , <u> </u> , <u> </u>);</code>	\longrightarrow	<code>g(10, 0, 0, 0);</code>
<code>g(10, 5);</code>	\longrightarrow	<code>g(10, 5 , <u> </u> , <u> </u>);</code>	\longrightarrow	<code>g(10, 5, 0, 0);</code>
<code>g(10, 5, 20);</code>	\longrightarrow	<code>g(10, 5 , 20 , <u> </u>);</code>	\longrightarrow	<code>g(10, 5, 20, 0);</code>
<code>g(10, 5, 20, 30);</code>	\longrightarrow	<code>g(10, 5 , 20 , 30);</code>	\longrightarrow	<code>g(10, 5, 20, 30);</code>

컴파일러에 의해 변환
되는 과정

예제 6-3 디폴트 매개 변수를 가진 함수 선언 및 호출

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

// 원형 선언

```
void star(int a=5);
void msg(int id, string text="");
```

디폴트
매개 변수 선언

// 함수 구현

```
void star(int a) {
    for(int i=0; i<a; i++)
        cout << '*';
    cout << endl;
}
```

```
void msg(int id, string text) {
    cout << id << ' ' << text << endl;
}
```

```
int main() {
    // star() 호출
    star();
    star(10);
```

star(5);

```
    // msg() 호출
    msg(10);
    msg(10, "Hello");
}
```

msg(10, "");

동일한코드

```
void star(int a=5) {
    for(int i=0; i<a; i++)
        cout << '*';
    cout << endl;
}

void msg(int id, string text="") {
    cout << id << ' ' << text << endl;
}
```

```
*****
*****
10
10 Hello
```


예제 6-4(실습) 디폴트 매개 변수를 가진 함수 만들기 연습

함수 f()를 호출하는 경우가 다음과 같을 때
f()를 디폴트 매개 변수를 가진 함수로
작성하라.

빈 칸이 10개 출력됨

```
%%%%%%%%  
aaaaaaaa  
aaaaaaaa  
aaaaaaaa  
aaaaaaaa  
aaaaaaaa
```

```
f();           // 한 줄에 빈칸을 10개 출력한다.  
f('%');        // 한 줄에 '%'를 10개 출력한다.  
f('@', 5);     // 다섯 줄에 '@'를 10개 출력한다.
```

```
#include <iostream>  
using namespace std;  
  
// 원형 선언  
void f(char c=' ', int line=1);  
  
// 함수 구현  
void f(char c, int line) {  
    for(int i=0; i<line; i++) {  
        for(int j=0; j<10; j++)  
            cout << c;  
        cout << endl;  
    }  
}  
  
int main() {  
    f();           // 한 줄에 빈칸을 10개 출력한다.  
    f('%');        // 한 줄에 '%'를 10개 출력한다.  
    f('@', 5);     // 5 줄에 '@' 문자를 10개 출력한다.  
}
```

- 디폴트 매개 변수의 장점 - 함수 중복 간소화

```
class Circle {  
    .....  
public:  
    Circle() { radius = 1; }  
    Circle(int r) { radius = r; }  
    .....  
};
```

```
class Circle {  
    .....  
public:  
    Circle(int r=1) { radius = r; }  
    .....  
};
```

2 개의 생성자 함수를
디폴트 매개 변수를 가진 하나의
함수로 간소화

- 중복 함수들과 디폴트 매개 변수를 가진 함수를 함께 사용 불가

```
class Circle {  
    .....  
public:  
    Circle() { radius = 1; }  
    Circle(int r) { radius = r; }  
    Circle(int r=1) { radius = r; }  
    .....  
};
```

중복된 함수와 동시 사
용 불가

다음 두 개의 중복 함수를 디폴트 매개 변수를 가진 하나의 함수로 작성하라.

```
void fillLine() { // 25 개의 '*' 문자를 한 라인에 출력
    for(int i=0; i<25; i++) cout << '*';
    cout << endl;
}
void fillLine(int n, char c) { // n개의 c 문자를 한 라인에 출력
    for(int i=0; i<n; i++) cout << c;
    cout << endl;
}
```

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

void fillLine(int n=25, char c='*') { // n개의 c 문자를 한 라인에 출력
    for(int i=0; i<n; i++) cout << c;
    cout << endl;
}

int main() {
    fillLine();           // 25개의 '*'를 한 라인에 출력
    fillLine(10, '%');    // 10개의 '%'를 한 라인에 출력
}
```

```
*****
%%%%%%%%%
```

다음 클래스에 중복된 생성자를 디폴트 매개 변수를 가진 하나의 생성자로 작성하라.

```
class MyVector{
    int *p;
    int size;
public:
    MyVector() {
        p = new int [100];
        size = 100;
    }
    MyVector(int n) {
        p = new int [n];
        size = n;
    }
    ~MyVector() { delete [] p; }
};
```

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

```
class MyVector{
    int *p;
    int size;
public:
    /*
```

→ 이곳에 디폴트 매개변수를 가진 생성자 작성하라

```
*/
    ~MyVector() { delete [] p; }
};
```

```
int main() {
    MyVector *v1, *v2;
    v1 = new MyVector(); // 디폴트로 정수 100개의 배열 동적 할당
    v2 = new MyVector(1024); // 정수 1024개의 배열 동적 할당

    delete v1;
    delete v2;
}
```

예제 6-6(실습) 생성자 함수의 중복 간소화(정답)

```
class MyVector{  
    int *p;  
    int size;  
public:  
    MyVector() {  
        p = new int [100];  
        size = 100;  
    }  
    MyVector(int n) {  
        p = new int [n];  
        size = n;  
    }  
    ~MyVector() { delete [] p; }  
};
```

정답

```
#include <iostream>  
using namespace std;  
  
class MyVector{  
    int *p;  
    int size;  
public:  
    MyVector(int n=100) {  
        p = new int [n];  
        size = n;  
    }  
    ~MyVector() { delete [] p; }  
};  
  
int main() {  
    MyVector *v1, *v2;  
    v1 = new MyVector(); // 디폴트로 정수 100개의 배열 동적 할당  
    v2 = new MyVector(1024); // 정수 1024개의 배열 동적 할당  
  
    delete v1;  
    delete v2;  
}
```

위임생성자로 작성할 수도 있음(3.4절참고)

```
MyVector() : MyVector(100) { }  
MyVector(int n) {  
    p = new int [n];  
    size = n;  
}
```

■ 함수 중복이 모호하여 컴파일러가 어떤 함수를 호출하는지 판단하지 못하는 경우

- 형 변환으로 인한 모호성
- 참조 매개 변수로 인한 모호성
- 디폴트 매개 변수로 인한 모호성

■ 매개 변수의 형 변환으로 인한 중복 함수 호출의 모호성

```
double square(double a) {  
    return a*a;  
}  
int main() {  
    cout << square(3);  
}
```

int 타입 3이 double
로 자동 형 변환

(a) 정상 컴파일

```
float square(float a) {  
    return a*a;  
}  
double square(double a) {  
    return a*a;  
}  
int main() {  
    cout << square(3.0);  
    cout << square(3);  
}
```

3.0은 double 타입이
므로
모호하지 않음

int 타입 3을 double로
변환할지 float로 변환할
지 모호함

(b) 모호한 호출, 컴파일 오류

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

float square(float a) {
    return a*a;
}

double square(double a) {
    return a*a;
}

int main() {
    cout << square(3.0); // square(double a); 호출
    cout << square(3); // 컴파일 오류
}
```

square
오버로드된 함수 "square"의 인스턴스 중 두 개 이상이 인수 목록과 일치합니다.
함수 "square(float a)"
함수 "square(double a)"
인수 형식이 (int) 입니다.

예제 6-8 참조 매개 변수로 인한 함수 중복의 모호성

두 함수는 근본적으로
중복 시킬 수 없다.

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

int add(int a, int b) {
    return a + b;
}

int add(int a, int &b) {
    b = b + a;
    return b;
}

int main(){
    int s=10, t=20;
    cout << add(s, t); // 컴파일 오류
}
```

call by value인지
call by reference인지 모호

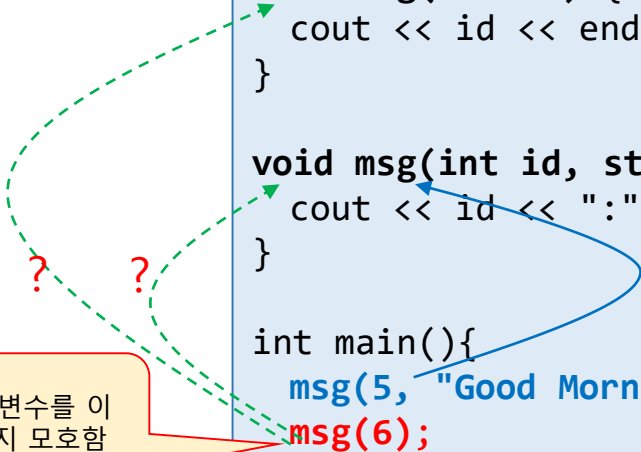
```
#include <iostream>
#include <string>
using namespace std;

void msg(int id) {
    cout << id << endl;
}

void msg(int id, string s="") {
    cout << id << ":" << s << endl;
}

int main(){
    msg(5, "Good Morning"); // 정상 컴파일. 두 번째 msg() 호출
    msg(6);                // 함수 호출 모호. 컴파일 오류
}
```

디폴트 매개 변수를 이
용하고 있는지 모호함





사람은 모두 각자의 눈을 가지고 태어난다.



사람이 태어나기 전에 공기가 있으며, 모든 사람은 공기를 공유한다. 공기 역시 각 사람의 것이다.

■ static

- 변수와 함수에 대한 기억 부류의 한 종류
 - 생명 주기 – 프로그램이 시작될 때 생성, 프로그램 종료 시 소멸
 - 사용 범위 – 선언된 범위, 접근 지정에 따름

■ 클래스의 멤버

- static 멤버
 - 프로그램이 시작할 때 생성
 - 클래스 당 하나만 생성, 클래스 멤버라고 불림
 - 클래스의 모든 인스턴스(객체)들이 공유하는 멤버
- non-static 멤버
 - 객체가 생성될 때 함께 생성
 - 객체마다 객체 내에 생성
 - 인스턴스 멤버라고 불림

■ this

- 포인터, 객체 자신 포인터
- 클래스의 멤버 함수 내에서만 사용
- 개발자가 선언하는 변수가 아니고, 컴파일러가 선언한 변수
 - 멤버 함수에 컴파일러에 의해 묵시적으로 삽입 선언되는 매개 변수

```
Circle *Circle = new Circle;
```



```
Circle * Circle = new Circle();  
(*Circle).radius = 1;  
Circle ->radius = 1;  
this -> radius = 1;
```

```
class Circle {  
    int radius;  
public:  
    Circle() { this->radius=1; }  
    Circle(int radius) { this->radius = radius; }  
    void setRadius(int radius) { this->radius = radius; }  
    ....  
};
```

■ 멤버의 static 선언

```
class Person {  
public:  
    int money; // 개인 소유의 돈  
    void addMoney(int money) {  
        this->money += money;  
    }  
  
    static int sharedMoney; // 공금  
    static void addShared(int n) {  
        sharedMoney += n;  
    }  
};  
  
int Person::sharedMoney = 10; // sharedMoney를 10으로 초기화
```

non-static 멤버 선언

static 멤버 변수 선언

static 멤버 함수 선언

static 변수 공간 할당.
프로그램의 전역 공간에 선언

■ static 멤버 변수 생성

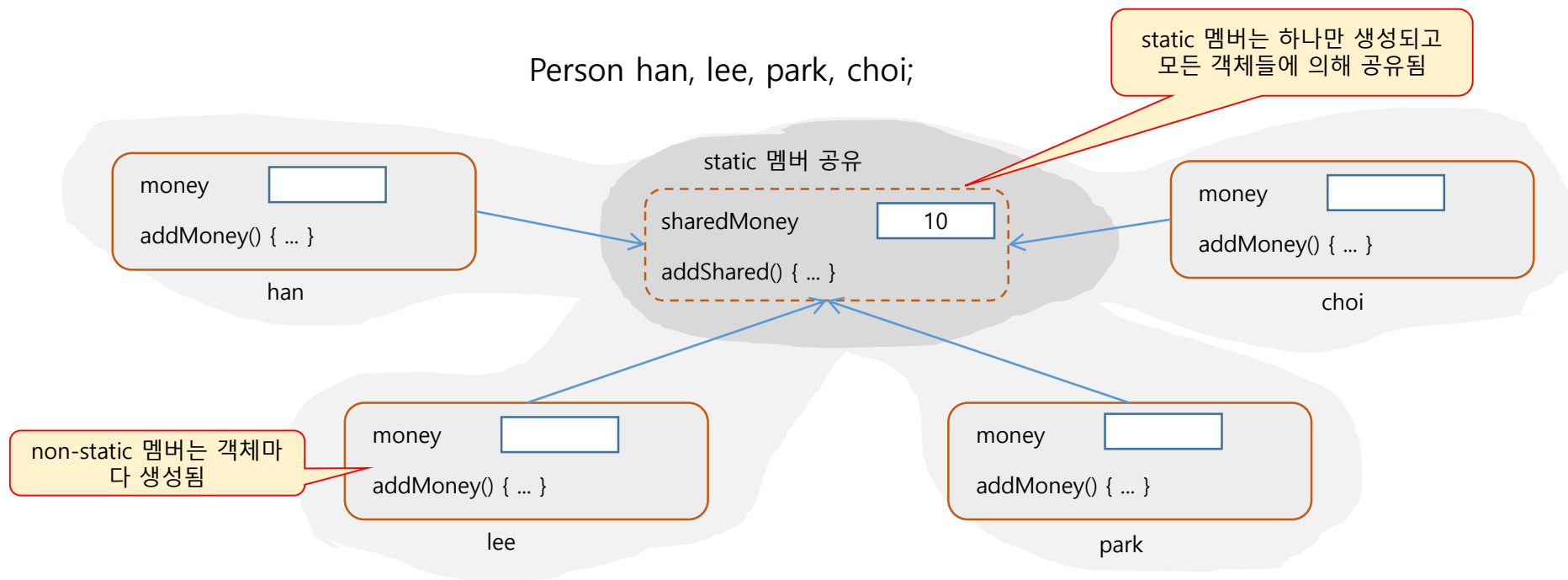
- 전역 변수로 생성
- 전체 프로그램 내에 한 번만 생성

■ static 멤버 변수에 대한 외부 선언이 없으면 다음과 같은 링크 오류

컴파일 성공

링크 오류

```
1>----- 빌드 시작: 프로젝트: StaticSample1, 구성: Debug Win32 -----  
1> StaticSample1.cpp  
1>StaticSample1.obj : error LNK2001: "public: static int Person::sharedMoney" (?sharedMoney@Person@@2HA) 외부 기호를 확인할 수 없습니다.  
1>C:\WC++\Wchap6\Debug\그림 6-9.exe : fatal error LNK1120: 1개의 확인할 수 없는 외부 참조입니다.  
===== 빌드: 성공 0, 실패 1, 최신 0, 생략 0 =====
```



- han, lee, park, choi 등 4 개의 Person 객체 생성
- sharedMoney와 addShared() 함수는 하나만 생성되고 4 개의 객체들의 의해 공유됨
- sharedMoney와 addShared() 함수는 han, lee, park, choi 객체들의 멤버임

항목	non-static 멤버	static 멤버
선언 사례	<pre>class Sample { int n; void f(); };</pre>	<pre>class Sample { static int n; static void f(); };</pre>
공간 특성	멤버는 객체마다 별도 생성 <ul style="list-style-type: none"> • 인스턴스 멤버라고 부름 	멤버는 클래스 당 하나 생성 <ul style="list-style-type: none"> • 멤버는 객체 내부가 아닌 별도의 공간에 생성 • 클래스 멤버라고 부름
시간적 특성	객체와 생명을 같이 함 <ul style="list-style-type: none"> • 객체 생성 시에 멤버 생성 • 객체 소멸 시 함께 소멸 • 객체 생성 후 객체 사용 가능 	프로그램과 생명을 같이 함 <ul style="list-style-type: none"> • 프로그램 시작 시 멤버 생성 • 객체가 생기기 전에 이미 존재 • 객체가 사라져도 여전히 존재 • 프로그램이 종료될 때 함께 소멸
공유의 특성	공유되지 않음 <ul style="list-style-type: none"> • 멤버는 객체 별로 따로 공간 유지 	동일한 클래스의 모든 객체들에 의해 공유됨

■ static 멤버는 객체 이름이나 객체 포인터로 접근

- 보통 멤버처럼 접근할 수 있음

```
객체.static멤버  
객체포인터->static멤버
```

- Person 타입의 객체 lee와 포인터 p를 이용하여 static 멤버를 접근하는 예

```
Person lee;  
lee.sharedMoney = 500; // 객체.static멤버 방식  
  
Person *p;  
p = &lee;  
p->addShared(200); // 객체포인터->static멤버 방식
```

```

#include <iostream>
using namespace std;

class Person {
public:
    int money; // 개인 소유의 돈
    void addMoney(int money) {
        this->money += money;
    }

    static int sharedMoney; // 공금
    static void addShared(int n) {
        sharedMoney += n;
    }
};

// static 변수 생성. 전역 공간에 생성
int Person::sharedMoney=10; // 10으로 초기화

// main() 함수
int main() {
    Person han;
    han.money = 100; // han의 개인 돈=100
    han.sharedMoney = 200; // static 멤버 접근, 공금=200

    Person lee;
    lee.money = 150; // lee의 개인 돈=150
    lee.addMoney(200); // lee의 개인 돈=350
    lee.addShared(200); // static 멤버 접근, 공금=400

    cout << han.money << ' '
         << lee.money << endl;
    cout << han.sharedMoney << ' '
         << lee.sharedMoney << endl;
}

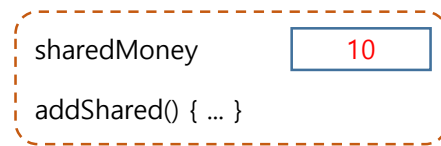
```

100 350
400 400

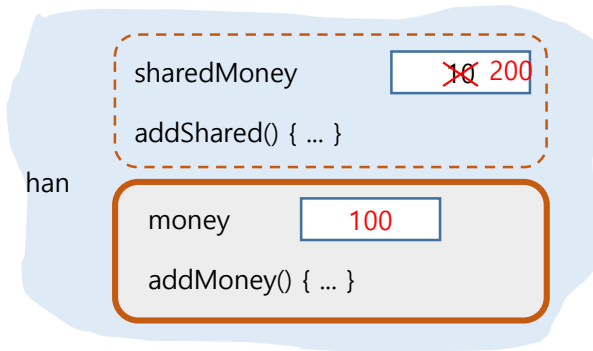
han과 lee의 money는 각각 100, 350

han과 lee의 sharedMoney는 공통 400

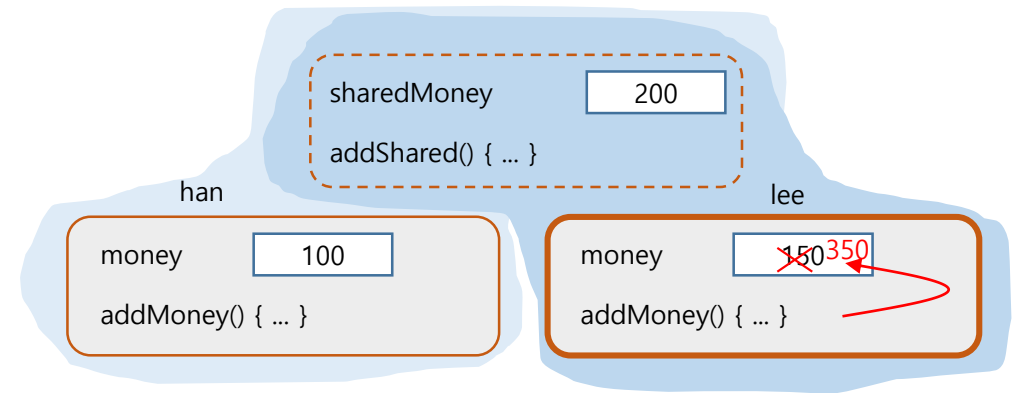
main()이 시작하기 직전



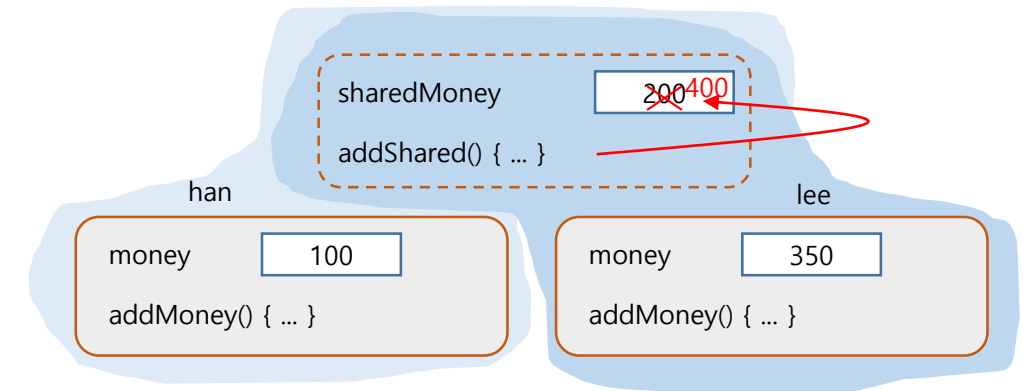
Person han;
han.money = 100;
han.sharedMoney = 200;



Person lee;
lee.money = 150;
lee.addMoney(200);



lee.addshared(200);



■ 클래스 이름과 범위 지정 연산자(::)로 접근 가능

- static 멤버는 클래스마다 오직 한 개만 생성되기 때문

클래스명::static멤버

han.sharedMoney = 200;	<->	Person::sharedMoney = 200;
lee.addShared(200);	<->	Person::addShared(200);

- non-static 멤버는 클래스 이름을 접근 불가

Person::money = 100;	// 컴파일 오류. non-static 멤버는 클래스 명으로 접근불가
Person::addMoney(200);	// 컴파일 오류. non-static 멤버는 클래스 명으로 접근불가

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

class Person {
public:
    int money; // 개인 소유의 돈
    void addMoney(int money) {
        this->money += money;
    }

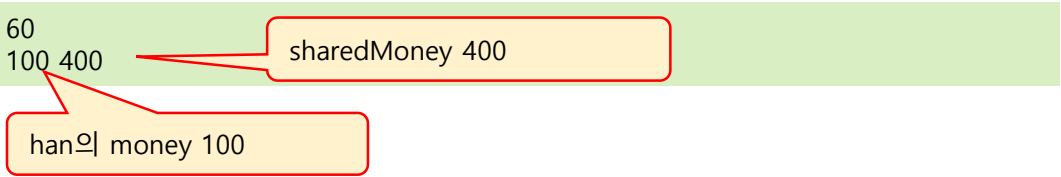
    static int sharedMoney; // 공금
    static void addShared(int n) {
        sharedMoney += n;
    }
};

// static 변수 생성. 전역 공간에 생성
int Person::sharedMoney=10; // 10으로 초기화

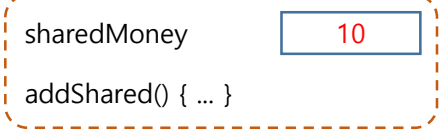
// main() 함수
int main() {
    Person::addShared(50); // static 멤버 접근, 공금=60
    cout << Person::sharedMoney << endl;

    Person han;
    han.money = 100;
    han.sharedMoney = 200; // static 멤버 접근, 공금=200
    Person::sharedMoney = 300; // static 멤버 접근, 공금=300
    Person::addShared(100); // static 멤버 접근, 공금=400

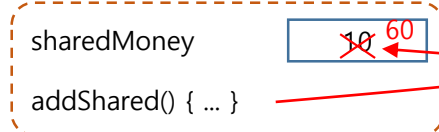
    cout << han.money << ' '
        << Person::sharedMoney << endl;
}
```



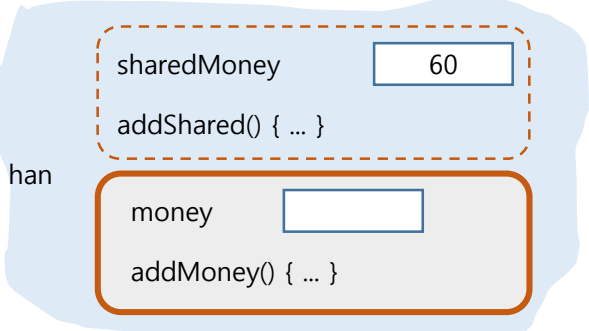
main()이 시작하기 직전



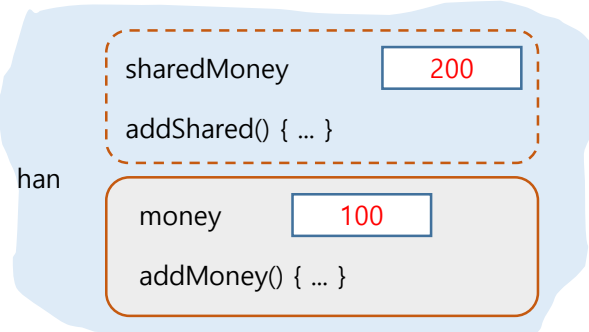
Person::addShared(50);



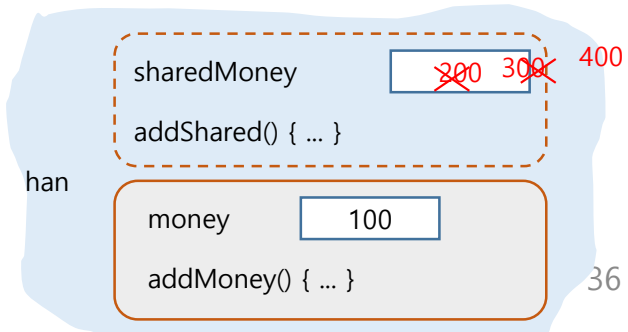
Person han;



han.money = 100;
han.sharedMoney = 200;



Person::sharedMoney = 300;
Person::addShared(100);



■ static의 주요 활용

- 전역 변수나 전역 함수를 클래스에 캡슐화
 - 전역 변수나 전역 함수를 가능한 사용하지 않도록
 - 전역 변수나 전역 함수를 static으로 선언하여 클래스 멤버로 선언
- 객체 사이에 공유 변수를 만들고자 할 때
 - static 멤버를 선언하여 모든 객체들이 공유

왼쪽 코드를 static 멤버를 가진 Math 클래스로 작성하고 멤버 함수를 호출하라.

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

int abs(int a) { return a>0?a:-a; }
int max(int a, int b) { return a>b?a:b; }
int min(int a, int b) { return (a>b)?b:a; }

int main() {
    cout << abs(-5) << endl;
    cout << max(10, 8) << endl;
    cout << min(-3, -8) << endl;
}
```

(a) 전역 함수들을 가진 좋지 않은 코딩 사례

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

class Math {
public:
    static int abs(int a) { return a>0?a:-a; }
    static int max(int a, int b) { return (a>b)?a:b; }
    static int min(int a, int b) { return (a>b)?b:a; }
};

int main() {
    cout << Math::abs(-5) << endl;
    cout << Math::max(10, 8) << endl;
    cout << Math::min(-3, -8) << endl;
}
```

(b) Math 클래스를 만들고 전역 함수들을 static 멤버로 캡슐화한 프로그램

5
10
-8

예제 6-11 static 멤버를 공유의 목적으로 사용하는 예

생존하고 있는 원의 개수 = 10
생존하고 있는 원의 개수 = 0
생존하고 있는 원의 개수 = 1
생존하고 있는 원의 개수 = 2

생성자가 10번 실행되어
numOfCircles = 10 이 됨

numOfCircles = 0 이 됨

numOfCircles = 1 이 됨

numOfCircles = 2 가 됨

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

class Circle {
private:
    static int numOfCircles;
    int radius;
public:
    Circle(int r=1);
    ~Circle() { numOfCircles--; } // 생성된 원의 개수 감소
    double getArea() { return 3.14*radius*radius;}
    static int getNumOfCircles() { return numOfCircles; }
};

Circle::Circle(int r) {
    radius = r;
    numOfCircles++; // 생성된 원의 개수 증가
}

int Circle::numOfCircles = 0; // 0으로 초기화

int main() {
    Circle *p = new Circle[10]; // 10개의 생성자 실행
    cout << "생존하고 있는 원의 개수 = " << Circle::getNumOfCircles() << endl;

    delete [] p; // 10개의 소멸자 실행
    cout << "생존하고 있는 원의 개수 = " << Circle::getNumOfCircles() << endl;

    Circle a; // 생성자 실행
    cout << "생존하고 있는 원의 개수 = " << Circle::getNumOfCircles() << endl;

    Circle b; // 생성자 실행
    cout << "생존하고 있는 원의 개수 = " << Circle::getNumOfCircles() << endl;
}
```

주목

- static 멤버 함수가 접근할 수 있는 것
 - static 멤버 함수
 - static 멤버 변수
 - 함수 내의 지역 변수
- static 멤버 함수는 non-static 멤버에 접근 불가
 - 객체가 생성되지 않은 시점에서 static 멤버 함수가 호출될 수 있기 때문

static 멤버 함수 getMoney()가 non-static 멤버 변수 money를 접근하는 오류

```
class PersonError {  
    int money;  
public:  
    static int getMoney() { return money; }  
  
    void setMoney(int money) { // 정상 코드  
        this->money = money;  
    }  
};  
  
int main(){  
    int n = PersonError::getMoney();  
  
    PersonError errorKim;  
    errorKim.setMoney(100);  
}
```

컴파일 오류.
static 멤버 함수는 non-static 멤버에 접근할 수 없음.

main()이 시작하기 전

```
static int getMoney() {  
    return money;  
}
```

money는 아직 생성되지 않았음.

n = PersonError::getMoney();

```
static int getMoney() {  
    return money;  
}
```

생성되지 않는 변수를 접근하게 되는 오류를 범함

PersonError errorKim;

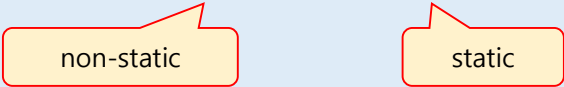
errorKim

```
static int getMoney() {  
    return money;  
}
```

errorKim 객체가 생길 때
money가 비로소 생성됨

money
setMoney() { ... }

```
class Person {  
    public: double money;    // 개인 소유의 돈  
    static int sharedMoney; // 공금  
    ....  
    int total() { // non-static 함수는 non-static이나 static 멤버에 모두 접근 가능  
        return money + sharedMoney;  
    }  
};
```



The diagram consists of two yellow callout boxes with red outlines. The first box, labeled 'non-static', has an arrow pointing to the 'total()' method. The second box, labeled 'static', has an arrow pointing to the 'sharedMoney' variable.

■ static 멤버 함수는 객체가 생기기 전부터 호출 가능

- static 멤버 함수에서 this 사용 불가

```
class Person {  
public:  
    double money; // 개인 소유의 돈  
    static int sharedMoney; // 공금  
    ....  
    static void addShared(int n) { // static 함수에서 this 사용 불가  
        this->sharedMoney + = n; // this를 사용하므로 컴파일 오류  
    }  
};
```

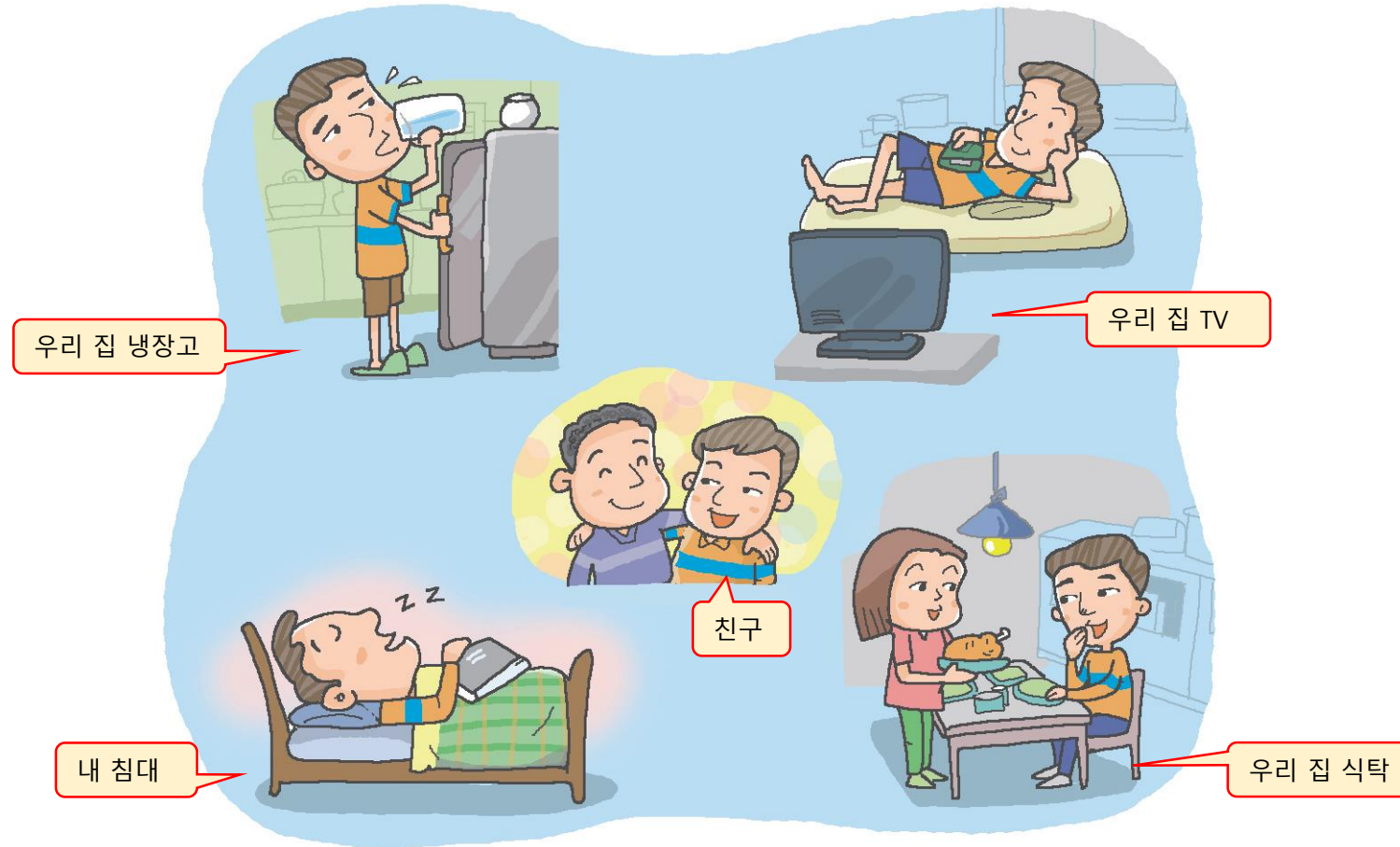
sharedMoney += n;으로 하면 정상 컴파일

07

프렌드와 연산자 중복

친구?

내 가족의 일원은 아니지만 내 가족과 동일한 권한을 가진 일원으로 인정받은 사람

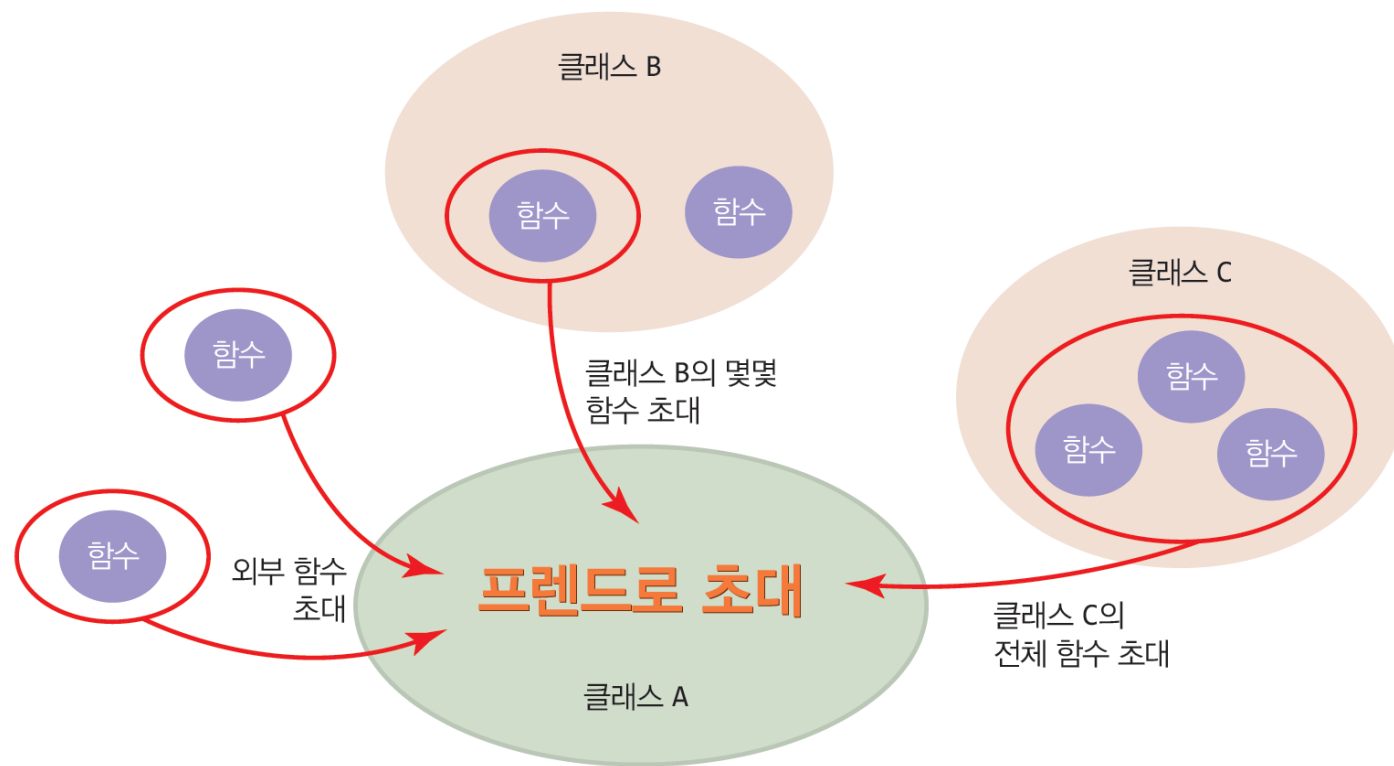


■ 프렌드 함수

- 클래스의 멤버 함수가 아닌 외부 함수
 - 전역 함수
 - 다른 클래스의 멤버 함수
- friend 키워드로 클래스 내에 선언된 함수
 - 클래스의 모든 멤버를 접근할 수 있는 권한 부여
 - 프렌드 함수라고 부름
- 프렌드 선언의 필요성
 - 클래스의 멤버로 선언하기에는 무리가 있고, 클래스의 모든 멤버를 자유롭게 접근할 수 있는 일부 외부 함수 작성 시

항목	세상의 친구	프렌드 함수
존재	가족이 아님. 외부인	클래스 외부에 작성된 함수. 멤버가 아님
자격	가족의 구성원으로 인정받음. 가족의 모든 살림살이에 접근 허용	클래스의 멤버 자격 부여. 클래스의 모든 멤버에 대해 접근 가능
선언	친구라고 소개	클래스 내에 friend 키워드로 선언
개수	친구의 명수에 제한 없음	프렌드 함수 개수에 제한 없음

- 프렌드 함수가 되는 3 가지
 - 전역 함수 : 클래스 외부에 선언된 전역 함수
 - 다른 클래스의 멤버 함수 : 다른 클래스의 특정 멤버 함수
 - 다른 클래스 전체 : 다른 클래스의 모든 멤버 함수



1. 외부 함수 equals()를 Rect 클래스에 프렌드로 선언

```
class Rect { // Rect 클래스 선언
    ...
    friend bool equals(Rect r, Rect s);
};
```

2. RectManager 클래스의 equals() 멤버 함수를 Rect 클래스에 프렌드로 선언

```
class Rect {
    .....
    friend bool RectManager::equals(Rect r, Rect s);
};
```

3. RectManager 클래스의 모든 멤버 함수를 Rect 클래스에 프렌드로 선언

```
class Rect {
    .....
    friend RectManager;
};
```

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

```
class Rect;
bool equals(Rect r, Rect s); // equals() 함수 선언
```

Rect 클래스가 선언되기 전에 먼저 참조되는 컴파일 오류(forward reference)를 막기 위한 선언문(forward declaration)

```
class Rect { // Rect 클래스 선언
    int width, height;
public:
    Rect(int width, int height) { this->width = width; this->height = height; }
    friend bool equals(Rect r, Rect s);
};
```

equals() 함수를 프렌드로 선언

```
bool equals(Rect r, Rect s) { // 외부 함수
    if(r.width == s.width && r.height == s.height) return true;
    else return false;
}
```

equals() 함수는 private 속성을 가진 width, height에 접근할 수 있다.

```
int main() {
    Rect a(3,4), b(4,5);
    if(equals(a, b)) cout << "equal" << endl;
    else cout << "not equal" << endl;
}
```

객체 a와 b는 동일한 크기의 사각형이므로 "not equal" 출력

not equal

예제 7-2 다른 클래스의 멤버 함수를 프렌드로 선언

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

class Rect;

class RectManager { // RectManager 클래스 선언
public:
    bool equals(Rect r, Rect s);
};

class Rect { // Rect 클래스 선언
    int width, height;
public:
    Rect(int width, int height) { this->width = width; this->height = height; }
    friend bool RectManager::equals(Rect r, Rect s);
};

bool RectManager::equals(Rect r, Rect s) {
    if(r.width == s.width && r.height == s.height) return true;
    else return false;
}

int main() {
    Rect a(3,4), b(3,4);
    RectManager man;

    if(man.equals(a, b)) cout << "equal" << endl;
    else cout << "not equal" << endl;
}
```

Rect 클래스가 선언되기 전에 먼저 참조되는 컴파일 오류(forward reference)를 막기 위한 선언문(forward declaration)

RectManager 클래스의 equals() 멤버를 프렌드로 선언

객체 a와 b는 동일한 크기의 사각형이므로 "equal" 출력

equal

예제 7-3 다른 클래스 전체를 프렌드로 선언

52

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

```
class Rect;
```

Rect 클래스가 선언되기 전에 먼저 참조되는 컴파일 오류(forward reference)를 막기 위한 선언문(forward declaration)

```
class RectManager { // RectManager 클래스 선언
```

```
public:
```

```
    bool equals(Rect r, Rect s);
```

```
    void copy(Rect& dest, Rect& src);
```

```
};
```

```
class Rect { // Rect 클래스 선언
```

```
    int width, height;
```

```
public:
```

```
    Rect(int width, int height) { this->width = width; this->height = height; }
```

```
    friend RectManager;
```

```
};
```

RectManager 클래스를 프렌드 함수로 선언

```
bool RectManager::equals(Rect r, Rect s) { // r과 s가 같으면 true 리턴
```

```
    if(r.width == s.width && r.height == s.height) return true;
```

```
    else return false;
```

```
}
```

```
void RectManager::copy(Rect& dest, Rect& src) { // src를 dest에 복사
```

```
    dest.width = src.width; dest.height = src.height;
```

```
}
```

```
int main() {
```

```
    Rect a(3,4), b(5,6);
```

```
    RectManager man;
```

객체 b의 width, height 값이 a와 같아진다.

```
    man.copy(b, a); // a를 b에 복사한다.
```

```
    if(man.equals(a, b)) cout << "equal" << endl;
```

```
    else cout << "not equal" << endl;
```

```
}
```

equal

man.copy(b,a)를 통해 객체 b와 a의 크기가 동일하므로 "equal" 출력