

static 멤버와 non-static 멤버

1



사람은 모두 각자의 눈을 가지고 태어난다.



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

static 멤버와 non-static 멤버의 특성

2

□ 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

static 멤버 선언

3

□ 멤버의 static 선언

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

non-static 멤버 선언

static 멤버 변수 선언

static 멤버 함수 선언

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

static 변수 공간 할당.
프로그램의 전역 공간에 선언

□ static 멤버 변수 생성

- 전역 변수로 생성
- 전체 프로그램 내에 한 번만 생성

```
int Person::sharedMoney = 10; // sharedMoney를 10으로 초기화
```

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

컴파일
성공

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.

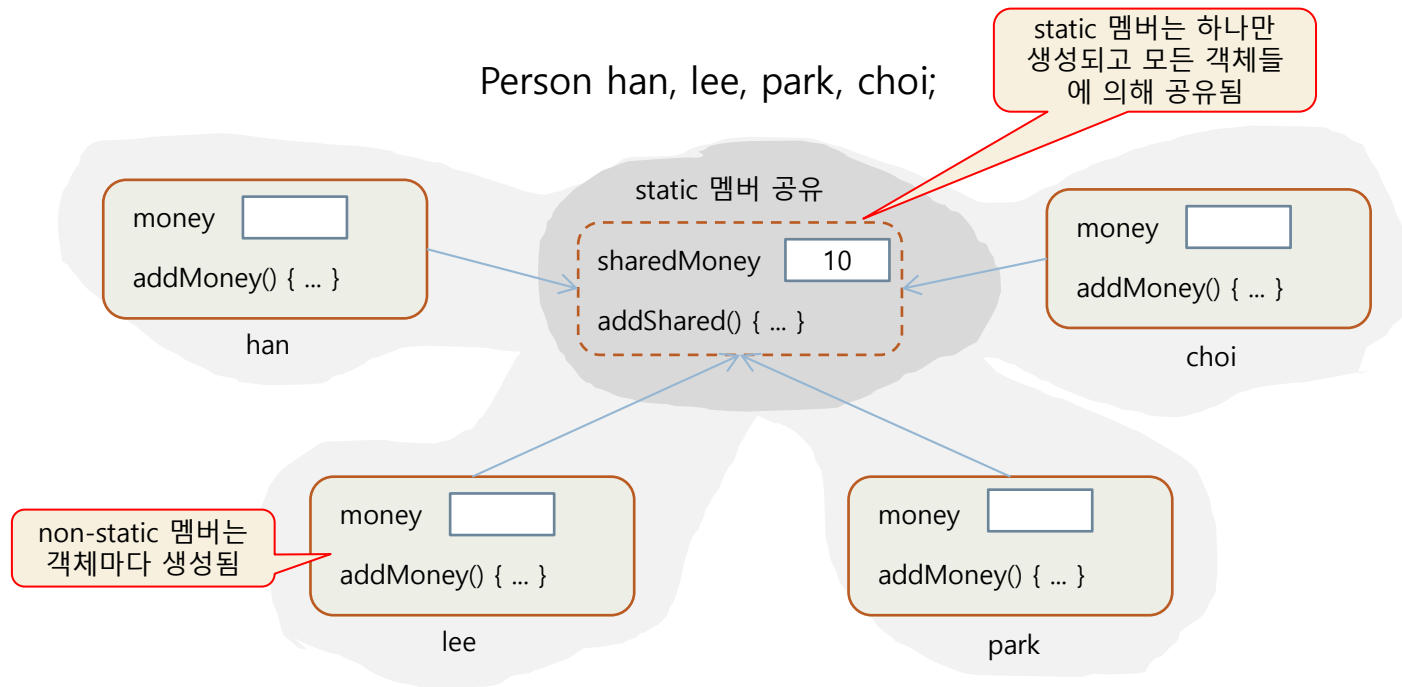
1>C:\WC++\Wchap6\Debug\그림 6-9.exe : fatal error LNK1120: 1개의 확인할 수 없는 외부 참조입니다.

===== 빌드: 성공 0, 실패 1, 최신 0, 생략 0 =====

링크
오류

static 멤버와 non-static 멤버의 관계

4



- 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

static 멤버와 non-static 멤버 비교

5

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 • 인스턴스 멤버라고 부름	멤버는 클래스 당 하나 생성 • 멤버는 객체 내부가 아닌 별도의 공간에 생성 • 클래스 멤버라고 부름
시간적 특성	객체와 생명을 같이 함 • 객체 생성 시에 멤버 생성 • 객체 소멸 시 함께 소멸 • 객체 생성 후 객체 사용 가능	프로그램과 생명을 같이 함 • 프로그램 시작 시 멤버 생성 • 객체가 생기기 전에 이미 존재 • 객체가 사라져도 여전히 존재 • 프로그램이 종료될 때 함께 소멸
공유의 특성	공유되지 않음 • 멤버는 객체 별로 따로 공간 유지	동일한 클래스의 모든 객체들에 의해 공유됨

static 멤버 사용 : 객체의 멤버로 접근

6

- 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

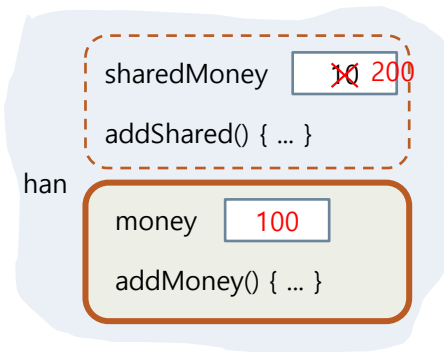
han과 lee의 money는 각각 100, 350

han과 lee의 sharedMoney는 공통 400

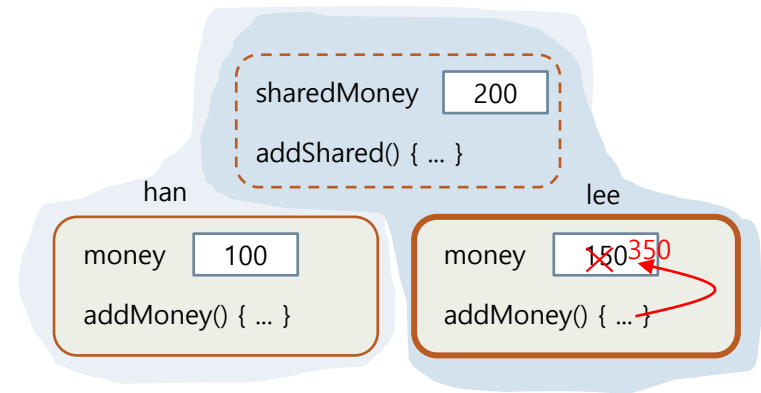
main()이 시작하기 직전



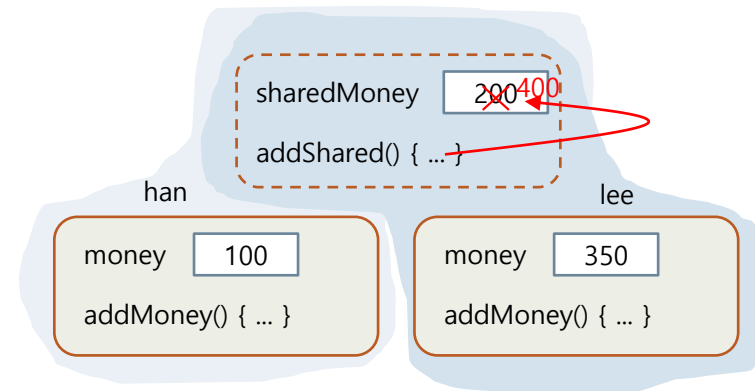
Person han;
han.money = 100;
han.sharedMoney = 200;



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



lee.addshared(200);



static 멤버 사용 : 클래스명과 범위 지정 연산자 (::)로 접근

8

- 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

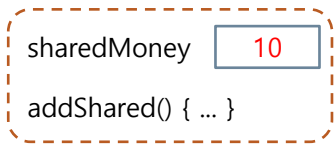
    cout << han.money << ' '
        << Person::sharedMoney << endl;
}
```

han 객체가 생기기전부터 static 멤버 접근

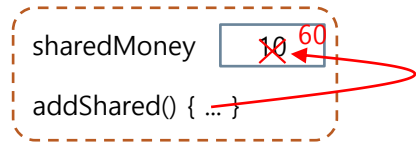
60 100 400 sharedMoney 400

han의 money 100

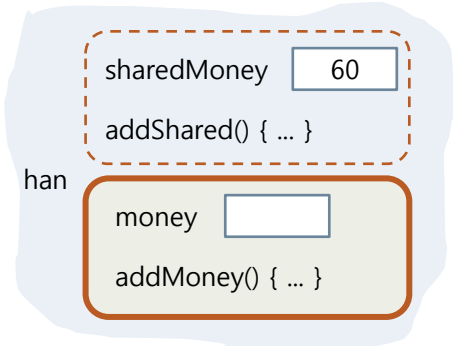
main()이 시작하기 직전



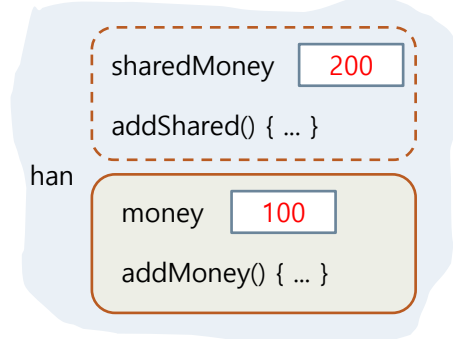
Person::addShared(50);



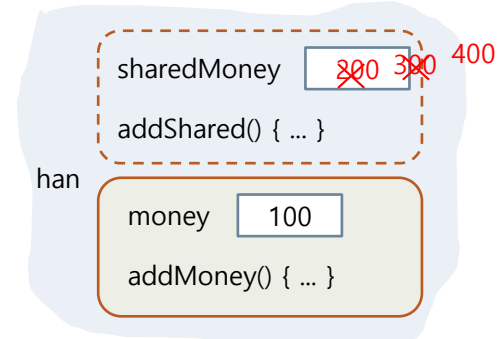
Person han;



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



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



static 활용

10

□ 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

예제 6-10 static 멤버를 가진 Math 클래스 작성

11

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;
}
```

```
#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;
}
```

5
10
-8

(a) 전역 함수들을 가진 좋지 않음 코딩 사례

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

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

```
#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 주목

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;
}
```

생성자가 10번 실행되어
numOfCircles = 10 이 됨

numOfCircles = 0 이 됨

numOfCircles = 1 이 됨

numOfCircles = 2 가 됨

static 멤버 함수는 static 멤버만 접근 가능

13

- ▣ 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

14

```
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 접근
할 수 없음.

main()이 시작하기 전

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

money는 아직 생
성되지 않았음.

n = PersonError::getMoney();

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

생성되지 않는 변수를 접
근하게 되는 오류를 범함

PersonError errorKim;

errorKim

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

errorKim 객체가 생길 때
money가 비로소 생성됨

money
setMoney() { ... }

non-static 멤버 함수는 static에 접근 가능

15

```
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 illustrates that a non-static function can access both non-static and static members. In the code, the `total()` function is non-static. It accesses the non-static member `money` and the static member `sharedMoney`. Red callout boxes with yellow backgrounds point to `money` and `sharedMoney`, labeled "non-static" and "static" respectively.

static 멤버 함수는 this 사용 불가

16

- 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