객체지향설계
Fall 2018

임성수 교수

Lecture 5: Abstraction & Implementation Hiding

수업 내용

1. 추상 자료형
2. 접근 제어
3. 프렌드
4. 인터페이스와 구현

1. 추상 자료형

추상 자료형 (Abstract Data Type, ADT)

- 기능의 구현 부분을 나타내지 않은 자료형
- 필요한 **자료와 기능의 집합을 나열**
- 구현의 세부 사항은 정의되지 않음

추상 자료형의 활용

- 추상자료형을 헤더 파일(.h)을 통해 선언
- 헤더 파일을 include하고
  연산자를 통해 추상 자료형 활용
- **정보 은닉**: 인터페이스만 제공하며,
  내부 구현은 숨겨짐

Light.h

```
class Light
{
  ... // 인터페이스 제공
  void on(int s);
  void off(int s);
};
```

Light.cpp

```
...
void Light::on(int s)
{
   ... // 실제 구현
}
void Light::off(int s)
{
   ... // 실제 구현
}
```

1. 추상 자료형

C의 추상 자료형

- 구조체를 사용하여 표현
- 멤버 변수로 구성

C++의 추상 자료형

- 클래스를 사용하여 표현
- **캡슐화**: 멤버 변수와 함수가 묶임

추상화: 효율성 높이고 복잡성 제거를 위해 관련 없는 세부 사항을 숨기는 것

캡슐화: 캡슐에 자료와 기능을 묶어서 외부로부터 접근을 제한하는 것

C의 추상 자료형 (CLib.h)

```
typedef struct CStashTag
{ // 변수 선언
  int size;
  int quantity;
  ...
} CStash;          구조체 변수
// 함수 선언
void initialize(CStash s, int sz);
void cleanup(CStash s);
...
```

C++의 추상 자료형 (CLib.h)

```
class Stash
{ // 변수 및 함수 선언
  int size;
  int quantity;
  ...
  public:
   // 멤버 함수
    void initialize(int sz);
    void cleanup();
};
```

1. 추상 자료형

```
MyClass *p;
p = new MyClass;
p->grade = "A"; // 서로
(*p).grade = "A"; // 동일
```

클래스 멤버 접근 방법

1. 클래스 범위 안에서 접근
2. 멤버 접근 연산자(. 과 ->)
   [예제] 객체.멤버
   객체->멤버 (또는 (*객체).멤버)
3. 범위 지정 연산자(클래스::변수 또는 함수)

참조자와 포인터

- 참조 변수(&변수): 변수를 입력 받음
  [예제] 초기화: `int &r = a;  // 대상을 직접 할당`
- 포인터 변수(*변수): 변수 주소값을 입력 받음
  [예제] 초기화: `int *p = &a; // &연산을 통해 주소값 할당`

```
#include "pch.h"
#include <iostream>
using namespace std;

int main()
{
  int num1 = 10;
  int &num2 = num1;
  int *num3 = &num1;
  num2 += 10;
  cout << num1 << " "
  << num2 << " " << *num3;
  return 0;
}
```

실행 결과

```
20 20 20
```

1. 추상 자료형

예제 구성

- 구조체를 사용한 추상 자료형 – C 스타일에서 C++ 스타일로 변환
- 클래스를 사용한 추상 자료형 – 구조체에서 클래스로 변환

예제 내용

- CLib.h (추상 자료형), CLib.cpp, CLibTest.cpp로 구성
- 구조체를 통한 추상 자료형에서 시작

CLib.h

```
typedef struct 태그명
{
  타입이름 변수명;
  ...
} 구조체타입명; // 새로운 타입을 정의

반환타입명 함수명(구조체타입명 구조체변수명, ...);
...
```

CLib.cpp

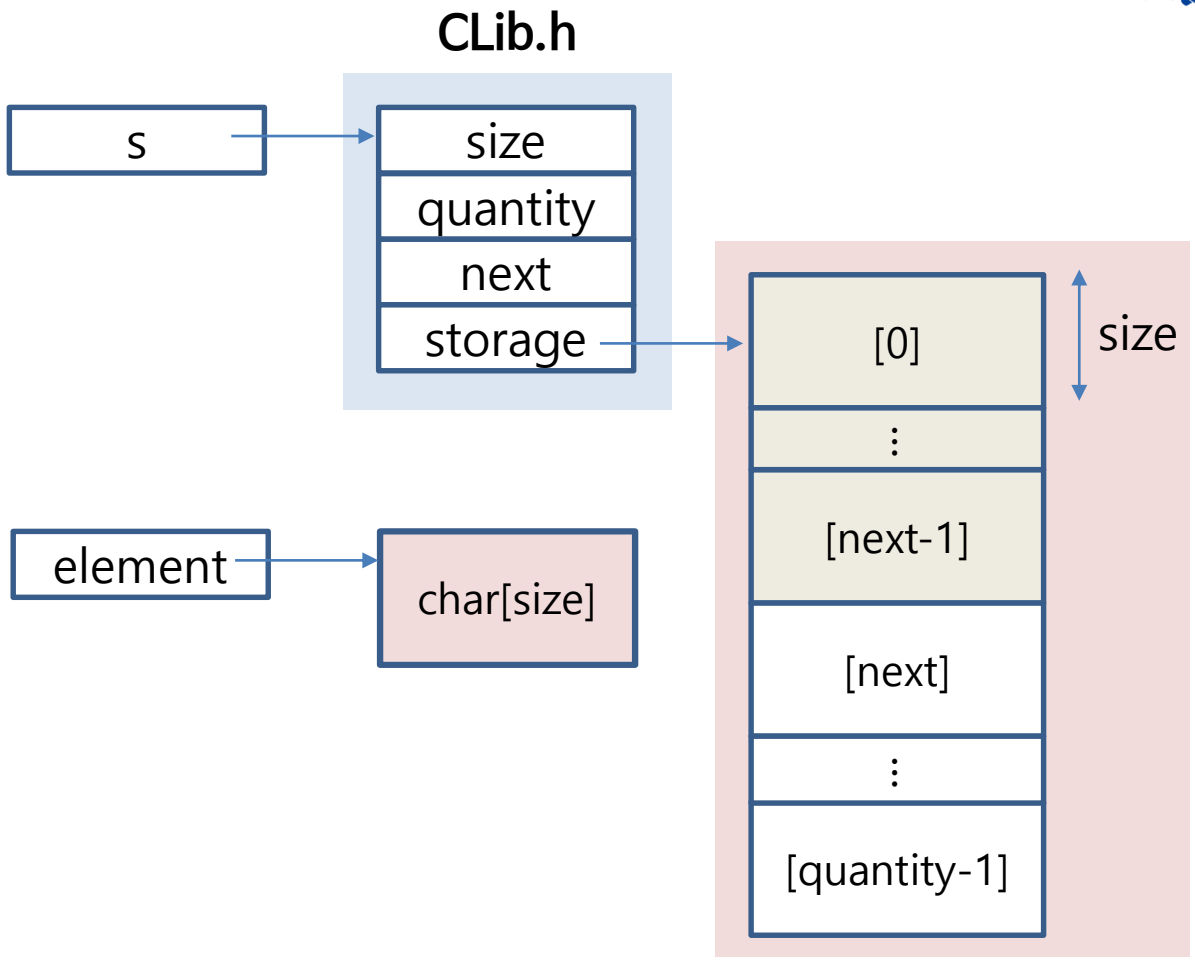
함수 구현

CLibTest.cpp

메인 함수

1. 추상 자료형

예제 내용


CLib.h
s
size
quantity
next
storage
[0]
size
[next-1]
[next]
[quantity-1]
element
char[size]
CLib.cpp
initialize()
cleanup()
add()
fetch()
count()
inflate()

1. 추상 자료형

[예제] C 추상 자료형

- CLib.h: typdef struct를 활용하여 CStash라는 추상 자료형 선언

```
// CLib.h
typedef struct CStashTag {
  int size;      // Size of each space
  int quantity;  // Number of storage spaces
  int next;      // Next empty space
  // Dynamically allocated array of bytes:
  unsigned char* storage;
} CStash;

void initialize(CStash* s, int sz);
void cleanup(CStash* s);
int add(CStash* s, const void* element);
void* fetch(CStash* s, int index);
int count(CStash* s);
void inflate(CStash* s, int increase);
```

구조체 Cstash 내부 멤버 정의

signed char: -127~127
unsigned char: 0~255
주로 문자 저장 시 사용하는 자료형

CStash를 자료형으로 활용
함수는 struct 바깥에 위치

[예제] CLib.cpp

```
#include "pch.h"
#include "CLib.h"
#include <iostream>
#include <cassert>
using namespace std;

// Quantity of elements to add
// when increasing storage:
const int increment = 100;

void initialize(CStash* s, int sz)
{
  s->size = sz;
  s->quantity = 0;
  s->next = 0;
  s->storage = 0;
}

void cleanup(CStash* s) {
  if (s->storage != 0) {
    cout << "freeing storage" << endl;
    delete[]s->storage;
  }
}
```

storage 공간이 부족할 때
increment 100씩 증가시킴

추상자료형 CStash 변수 s에 대해
size 값 sz, 나머지 0으로 초기화

종료 후 storage를 제거하는 역할

[예제] CLib.cpp (cont'd)

```
int add(CStash* s, const void* element){
  //Enough space left?
  if (s->next >= s->quantity)
    inflate(s, increment);
  // Copy element into storage,
  // starting at next empty space:
  int startBytes = s->next * s->size;
  unsigned char* e = (unsigned char*)element;
  for (int i = 0; i < s->size; i++)
    s->storage[startBytes + i] = e[i];
  s->next++;
  return(s->next - 1); // Index number
}

void* fetch(CStash* s, int index) {
  // Check index boundaries:
  assert(0 <= index);
  if (index >= s->next)
    return 0; // To indicate the end
  // Produce pointer to desired element:
  return &(s->storage[index * s->size]);
}
```

Next가 quantity보다 작지 않으면
현재 storage가 꽉 찼다는 의미

현재 storage 크기보다 더 많은
작업이 필요하면 inflate 함수를 통해
storage 크기를 increment 증가시킴

element를 가져와서 storage에 저장
아직 쓰지 않은 entry에 순차적 저장
next의 index 값을 +1 증가시킴

index가 음수이면 프로그램 종료
index가 next 이상인 경우 return 0;
index에 대해 storage 해당 부분 가리킴

[예제] CLib.cpp (cont'd)

```
int count(CStash* s) {
  return s->next;  // Elements in CStash
}

void inflate(CStash* s, int increase){
  assert(increase > 0);
  int newQuantity = s->quantity + increase;
  int newBytes = newQuantity * s->size;
  int oldBytes = s->quantity * s->size;
  unsigned char* b = new unsigned char[newBytes];
  for (int i = 0; i < oldBytes; i++)
    b[i] = s->storage[i]; // Copy old to new
  delete[](s->storage); // Old storage
  s->storage = b; // Point to new memory
  s->quantity = newQuantity;
}
```

다음 storage index를 반환

increase가 양수가 아니면 종료

quantity: increment만큼 추가

storage: quantity * size 크기의 배열에 기존 storage값 저장

[예제] CLibTest.cpp

```
int main() {
  // Define variables at the beginning
  // of the block, as in C:
  CStash intStash, stringStash;
  int i;
  char* cp;
  ifstream in;
  string line;
  const int bufsize = 80;

  // Now remember to initialize the variables:
  initialize(&intStash, sizeof(int));
  for (i = 0; i < 100; i++)
    add(&intStash, &i);
  for (i = 0; i < count(&intStash); i++)
    cout << "fetch(&intStash, " << i << ") = "
    << *(int*)fetch(&intStash, i)
    << endl;
```

```
#include "pch.h"
#include "CLib.h"
#include <fstream>
#include <iostream>
#include <string>
#include <cassert>
#include <iostream>
#include <string>
#include <cassert>
using namespace std;
```

intStash: 0부터 99까지 순차적으로 storage 추가 후 출력

[예제] CLibTest.cpp (cont'd)

```
// Holds 80-character strings:
  initialize(&stringStash, sizeof(char)*bufsize);
  in.open("CLibTest.cpp");
  assert(in);
  while (getline(in, line))
    add(&stringStash, line.c_str());
  i = 0;
  while ((cp = (char*)fetch(&stringStash, i++)) != 0)
    cout << "fetch(&stringStash, " << i << ") = "
    << cp << endl;

  cleanup(&intStash);
  cleanup(&stringStash);
}
```

stringStash: 파일을 줄마다 읽고 저장 후 각 줄마다 출력

c_str 함수: string을 char*로 변환 함수를 사용할 수 있게 함

사용한 storage들 모두 제거

실행 결과

```
fetch(&intStash, 0) = 0
fetch(&intStash, 1) = 1
fetch(&intStash, 2) = 2
…
fetch(&intStash, 98) = 97
fetch(&intStash, 98) = 98
fetch(&intStash, 99) = 99
```

```
fetch(&stringStash, 1) = #include "pch.h"
fetch(&stringStash, 2) = #include "CLib.h"
fetch(&stringStash, 3) = #include <fstream>
…
fetch(&stringStash, 39) =     cleanup(&intStash);
fetch(&stringStash, 40) =     cleanup(&stringStash);
fetch(&stringStash, 41) = }
freeing storage
freeing storage
```

1. 추상 자료형

C 스타일의 추상 자료형의 문제점

- 각 함수마다 구조체의 주소를 참조(&intStash, &stringStash 등)
- 프로그램이 제공하는 기능 파악을 위해 **세부 사항을 알아야 함**

C++스타일의 추상 자료형으로 수정

- **구조체 안에 함수**를 집어넣을 수 있음 (C: 불가/C++: 가능 – 캡슐화)
- main 함수에서 구조체 변수를 선언하여 내부 함수 기능 활용

1. 추상 자료형

[예제] C++ 추상 자료형

- CLib.h: typdef struct를 활용하여 CStash라는 추상 자료형 선언

```
struct Stash {
  int size; // Size of each space
  int quantity;// # of storage spaces
  int next; // Next empty space
  // Dynamically allocated array of bytes:
  unsigned char* storage;

  // Functions!
  void initialize(int size);
  void cleanup();
  int add(const void* element);
  void* fetch(int index);
  int count();
  void inflate(int increase);
};
```

구조체 안에 함수까지 작성

함수 선언에 구조체 변수 생략 가능
e.g., C스타일 예제: CStash* s

[예제] CLib.cpp

```
#include "pch.h"
#include "CLib.h"
#include <iostream>
#include <cassert>
using namespace std;

// Quantity of elements to add
// when increasing storage:
const int increment = 100;

void Stash::initialize(int sz)
{
  size = sz;
  quantity = 0;
  storage = 0;
  next = 0;
}

void Stash::cleanup() {
  if (storage != 0) {
    cout << "freeing storage" << endl;
  delete[]storage;
  }
}
```

```
void initialize(CStash* s, int sz)
{
  s->size = sz;
  s->quantity = 0;
  s->next = 0;
  s->storage = 0;
}
```

C스타일과 달리 구조체 변수 참조 안함
범위 지정 연산자를 통해 접근 가능

[예제] CLib.cpp (cont'd)

```
int Stash::add(const void* element) {
  if (next >= quantity) // Enough space left?
    inflate(increment);
  // Copy element into storage,
  // starting at next empty space:
  int startBytes = next * size;
  unsigned char* e = (unsigned char*)element;
  for (int i = 0; i < size; i++)
    storage[startBytes + i] = e[i];
  next++;
  return(next - 1);// Index number
}

void* Stash::fetch(int index) {
  // Check index boundaries:
  assert(0 <= index);
  if (index >= next)
    return 0; // To indicate the end
  // Produce pointer to desired element:
  return &(storage[index * size]);
}
```

Stash
int size; int quantity; int next; unsigned char* storage;

void initialize(int size);
void cleanup();
int add(const void* element);
void* fetch(int index);
int count();
void inflate(int increase);

[예제] CLib.cpp (cont'd)

```
int Stash::count() {
  return next; // Number of elements in CStash
}

void Stash::inflate(intincrease) {
  assert(increase > 0);
  int newQuantity = quantity + increase;
  int newBytes = newQuantity * size;
  int oldBytes = quantity * size;
  unsigned char* b = new unsigned char[newBytes];
  for (int i = 0; i < oldBytes; i++)
    b[i] = storage[i];// Copy old to new
  delete[] storage; // Old storage
  storage = b;// Point to new memory
  quantity = newQuantity;
}
```

[예제] CLibTest.cpp

```
#include "pch.h"
#include "CLib.h"
#include <fstream>
#include <iostream>
#include <string>
#include <cassert>
#include <iostream>
#include <string>
#include <cassert>
using namespace std;
int main() {
  Stash intStash;
  intStash.initialize(sizeof(int));
  for (int i = 0; i < 100; i++)
    intStash.add(&i);
  for (int j = 0; j < intStash.count(); j++)
    cout << "intStash.fetch(" << j << ") = "
    << *(int*)intStash.fetch(j) << endl;
```

구조체 변수(Stash) 선언
내부 함수 기능 활용 가능

[예제] CLibTest.cpp (cont'd)

```
// Holds 80-character strings:
  Stash stringStash;
  const int bufsize = 80;
  stringStash.initialize(bufsize);
  ifstream in("CLibTest.cpp");
  string line;
  while (getline(in, line))
    stringStash.add(line.c_str());
  int k = 0;
  char* cp;
  while ((cp = (char*)stringStash.fetch(k++)) != 0)
    cout << "stringStash.fetch(" << k << ") = "
    << cp << endl;
  intStash.cleanup();
  stringStash.cleanup();
}
```

실행 결과

```
fetch(&intStash, 0) = 0
fetch(&intStash, 1) = 1
fetch(&intStash, 2) = 2
...
fetch(&intStash, 98) = 97
fetch(&intStash, 98) = 98
fetch(&intStash, 99) = 99
```

```
fetch(&stringStash, 1) = #include "pch.h"
fetch(&stringStash, 2) = #include "CLib.h"
fetch(&stringStash, 3) = #include <fstream>
...
fetch(&stringStash, 39) =     cleanup(&intStash);
fetch(&stringStash, 40) =     cleanup(&stringStash);
fetch(&stringStash, 41) = }
freeing storage
freeing storage
```

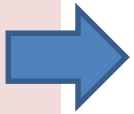
1. 추상 자료형

[예제] C++ 추상 자료형

- CLib.h: 구조체 → 클래스 수정 // 접근 제어 지시자의 차이가 있음

```
struct Stash
{
  int size;
  int quantity;
  int next;
  unsigned char* storage;

  void initialize(int size);
  void cleanup();
  int add(const void* element);
  void* fetch(int index);
  int count();
  void inflate(int increase);
};
```



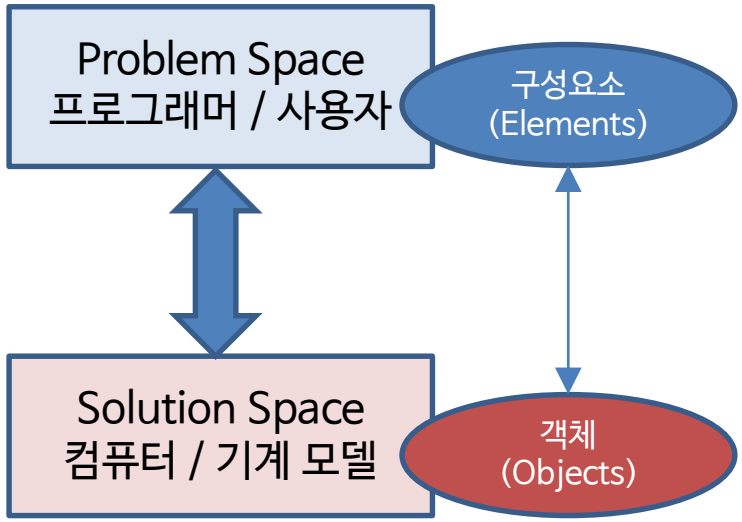
```
class Stash
{
  int size;
  int quantity;
  int next;
  unsigned char* storage;

  void inflate(int increase);
  public:
    void initialize(int size);
    void cleanup();
    int add(void* element);
    void* fetch(int index);
    int count();
};
```

1. 추상 자료형

C 의 구조체

- 자료의 집합으로 구성
- 기본적으로 자료와 함수가 분리



C++의 클래스

- **캡슐화**
  - 자료와 함수가 결합
- **추상화**
  - 문제 공간의 개념을 해결 공간으로 추상화

2. 접근 제어

C 프로그램 접근 제어

- 클라이언트 프로그래머가 구조체를 활용하여 뭐든지 할 수 있음
  - 디폴트 접근 제어: public
- 특정 행동에 대해 **제어하기 어려움**

멤버를 제어해야 하는 이유

- 클라이언트 프로그래머가 건들면 안 되는 부분을 유지
  - 인터페이스에 해당되지 않는 **내부 구조 보호**
- 라이브러리 디자이너에게 내부 구조를 변경할 수 있도록 허용
  - 클라이언트 프로그래머에게 어떤 영향을 끼칠지 걱정 없이

2. 접근 제어

C++ 프로그램 접근 제어

- **접근 제어 지시자**: 구조의 경계를 정하는 세 가지 키워드 사용
  - **public, private, protected**
  - 디폴트 접근 제어: private
- public : 선언된 멤버들에 대한 모든 외부 접근 허용
- private : 만든 사람을 제외하고 모든 사람의 멤버 접근을 차단
- protected : 상속된 구조까지 허용하고 모든 사람의 멤버 접근을 차단

2. 접근 제어

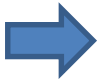
객체 레이아웃

- **접근 제어**: 정보 은닉을 위해 인터페이스와 구현을 분리
- 접근 제어 지시자를 통해 권한을 구분하고 별도 접근 및 수정
- 구조체는 public이 디폴트, 클래스는 private이 디폴트

[예제]

```
struct A
{
  int f();
  void g();
private:
  int i, j, k;
public:};
```

```
class B
{
  int i, j, k;
public:
  int f();
  void g();
};
```



동일한 접근 제어

2. 접근 제어

[예제] 접근 제어 지시자: private

```
#include "pch.h"
// Setting the boundary
struct B
{
private:
  char j;
  float f;
public:
  int i;
  void func();
};
```

```
void B::func()
{
  i = 0;
  j = '0';
  f = 0.0;
}
int main()
{
  B b;
  b.i = 1; // OK, public
  // b.j = '1'; // Illegal, private
  // b.f = 1.0; // Illegal, private
}
```

멤버 "B::j"에 엑세스할 수 없습니다.
멤버 "B::f"에 엑세스할 수 없습니다.

2. 접근 제어

[예제] 접근 제어 지시자: protected

```
#include "pch.h"
#include <iostream>
using namespace std;
class B {
private:
  int b_pri;
  void b_fpri() { cout <<
    "private 멤버 접근₩n"; }
protected:
  int b_pro;
  void b_fpro() { cout <<
    "protected 멤버 접근₩n"; }
};
```

```
class D : public B
{ // B는 D의 파생 클래스
public:
  void d_fpub() {
    // b_pri = 1;  // private 멤버 접근 불가
    // b_fpri();
    b_pro = 2;  // protected 멤버 접근 가능
    b_fpro();
  }
};

int main()
{
  D d;
  d.d_fpub();
}
```

실행 결과

```
Protected 멤버 접근
```

3. 프렌드

프렌드 함수

- 클래스의 **멤버 함수는 아니지만** 멤버 함수처럼 private 멤버에 **접근 가능**
- 전역 함수, 멤버 함수, 클래스 형태로 선언 가능

[예제]

```
#include "pch.h"
#include <iostream>
using namespace std;

class Test {
private:
  int pri;
public:
  Test(int val) { pri = val; }
  friend void friendF(Test test);
};
```

```
void friendF(Test test)
{
  cout << test.pri << endl;
}

int main()
{
  Test pritest(3);
  friendF(pritest);
}
```

friend 안 써줄 경우:
멤버 "Test::pri"에
액세스할 수 없습니다

실행 결과

```
3
```

3. 프렌드

[예제] 프렌드 활용

```
#include "pch.h"
#include <iostream>
using namespace std;

struct X;
struct Y {
  void f(X*);
};
struct X { // Definition
private:
  int i;
public:
  void initialize();
  friend void g(X*, int); // Global friend
  friend void Y::f(X*); // Struct member friend
  friend struct Z; // Entire struct is a friend
  friend void h();
};
```

다양한 형태의 프렌드 함수 활용
구조체를 프렌드로 할 수 있음

3. 프렌드

[예제] 프렌드 활용

```
void X::initialize() { i = 0; }
void g(X* x, int i) { x->i = i; }
void Y::f(X* x) { x->i = 47; }

struct Z {
private:
  int j;
public:
  void initialize();
  void g(X* x);
};

void Z::initialize() { j = 99; }
void Z::g(X* x) {
  x->i += j;
}
void h() {
  X x;
  x.i = 100;
}
```

X의 private 멤버 접근 및 수정 가능

프렌드 구조체 정의

```
int main() {
  X x;
  x.initialize();
  Z z;
  z.initialize();
  z.g(&x);
  cout << z.j;
}
```

실행 결과

```
99
```

3. 프렌드

중첩 클래스 (nested-/ inner-)

- **중첩 클래스**: 하나의 클래스 내부에 선언된 클래스
- 서로 관련 있는 클래스를 묶어서 코드의 캡슐화를 증가
- 외부에서는 내부 클래스의 멤버에 자유롭게 접근할 수 없음

중첩 프렌드 함수

- 외부에서는 내부 클래스의 멤버에 접근 가능하도록 friend로 정의
- 두 클래스가 서로 private 멤버를 자유롭게 읽어야 하는 상황에 지정

3. 프렌드

[예제]
중첩
클래스

```
#include "pch.h"
#include <iostream>
#include <string>
using namespace std;

class foo {
private:
  int attribute_0 = 100;
public:
  void action()
{ attribute_2.PrintFooBar(this); }

  friend class bar;
  class bar {
  private:
    int attribute_0 = 200;
  public:
    void PrintFooBar(foo* f) {
      cout << f->attribute_0 << " "
      << attribute_0;
    }
  } attribute_2;
};
```

```
int main() {
  foo f;
  f.action();
  return 0;
}
```

실행 결과

```
100 200
```

this 포인터: 객체가 생성되었을 때
자기 자신을 가리키는 포인터

attribute_0에 접근 가능하며
서로 다른 값 접근하여 출력

3. 프렌드

하이브리드 언어

- C++은 객체지향 개념을 추구하지만 실용성에 충실한 **하이브리드 언어**
- 실용적인 문제 해결을 위해 friend 키워드를 활용
  - 언어의 순수성은 다소 떨어지지만 문제 없음
- 추상적인 이상(ideal)보다 실리적으로 설계
- JAVA는 비교적 순수 객체지향 언어

강의 계획

주차	내용	
~~1~~	~~객체지향 개요 (lecture note)~~	~~개발 환경 구축~~
~~2~~	~~클래스 (lecture note)~~	~~C++ 기초 문법~~
~~3~~	~~객체 (Chap. 1)~~	~~**프로젝트** 팀 구성~~
~~4~~	~~객체 생성과 활용 (Chap. 2)~~	
~~5~~	~~추상화 및 정보 은닉 (Chap. 4, Chap. 5)~~	
6	상속과 구성 (Chap. 14)	**프로젝트** 제안서 제출
7	중간고사 전 내용 정리	**프로젝트** 제안서 발표
8	**중간고사**	

강의 계획

주차	내용	
9	다형성과 가상함수 (Chap. 15)	
10	템플릿의 소개 (Chap. 16)	
11	예외 처리 (lecture note)	**프로젝트** 중간 발표
12	디자인 패턴의 소개 (Ref. 2)	
13	주요 디자인 패턴 (Ref. 2)	
14	**프로젝트** 최종 발표 및 데모 시연	**프로젝트** 보고서 제출
15	**기말고사**	

질문 및 답변