객체지향설계
Fall 2018

임성수 교수

Lecture 6: Inheritance & Composition

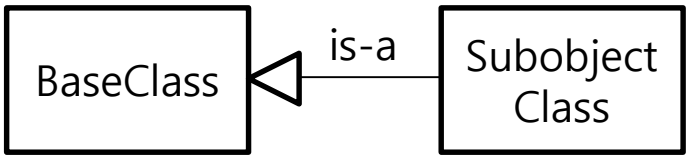
수업 내용

1. 상속과 구성
2. 생성자
3. 이름 은닉
4. 접근 제어
5. 하위 타입
6. 업캐스팅
7. 복사 생성자

1. 상속과 구성

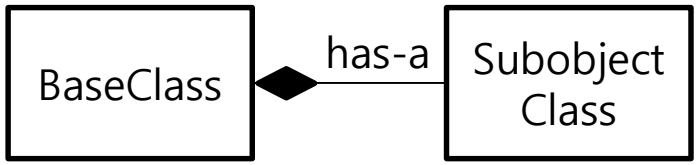
상속 (Inheritance)

- is-a 관계
- 파생 클래스가 기존 클래스의 멤버를 받아서 사용 가능
- 기능을 확장할 때 유리함



구성 (Composition)

- has-a 관계
- 다른 클래스의 일부 기능을 사용, 전체 기능 상속은 안 해도 됨
- 재사용할 때 유리함
- **포함 객체** (embedded object) : 클래스 내에 다른 클래스를 선언하여 활용



1. 상속과 구성

[예제] 구성 (Composition)

```
// Useful.h
#ifndef USEFUL_H
#define USEFUL_H

class X {
  int i;
public:
  X() { i = 0; }
  void set(int ii) { i = ii; }
  // 함수 선언에 붙는 const:
  // 값을 쓸 수는 있지만 변경은 불가
  int read() const { return i; }
  int permute() { return i = i * 47; }
};

#endif
```

```
// Composition.cpp
#include "pch.h"
#include "Useful.h"
class Y {
  int i;
public:
  X x; // 포함 객체 x를 public으로 생성
  Y() { i = 0; }
  void f(int ii) { i = ii; }
  int g() const { return i; }
};

int main()
{
  Y y; // 클래스 Y로부터 생성된 객체 y
  y.f(47); // 클래스 Y에서 정의된 함수 실행
  y.x.set(37); // 포함 객체 x의 멤버 접근 가능
}
```

1. 상속과 구성

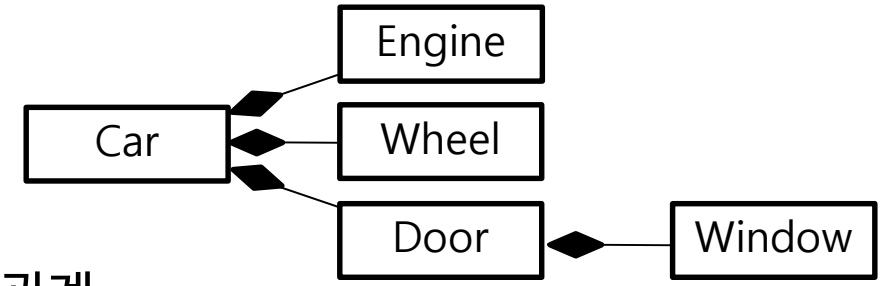
[예제] 구성 (Composition)

- 포함 객체를 private로 생성
- 멤버 함수 활용 (y.x 접근 불가)

```
// Useful.h - 앞 페이지와 동일함
#ifndef USEFUL_H
#define USEFUL_H

class X {
  int i;
public:
  X() { i = 0; }
  void set(int ii) { i = ii; }
  int read() const { return i; }
  int permute() { return i = i * 47; }
};

#endif
```

```
// Composition2.cpp
#include "pch.h"
#include "Useful.h"
class Y {
  int i;
  X x; // 포함 객체 x를 private으로 생성
public:
  Y() { i = 0; }
  void f(int ii) { i = ii; x.set(ii); }
  int g() const { return i * x.read(); }
  void permute() { x.permute(); } // Redefine
};

int main()
{
  Y y; // 포함 객체 x에 대해 y.x로 접근 불가
  y.f(47); // 클래스 Y에서 정의된 함수 실행
  y.permute(); // Y에서 재정의된 함수 실행
}
```

1. 상속과 구성



[예제] 구성 (Composition): has-a 관계

```
#include "pch.h"
#include <iostream>

class Engine {
public:
  void start() {} // 출발
  void rev() {} // 엔진 회전
  void stop() {} //정지
};

class Wheel {
public:
  void inflate(int psi) {} // 공기압
};

class Window {
public:
  void rollup() {} // 창문 올리기
  void rolldown() {} // 창문 내리기
};
```

```
class Door {
public:
  Window window;
  void open() {} // 문 열기
  void close() {} // 문 닫기
};

class Car {
public:
  Engine engine; // 엔진 1개
  Wheel wheel[4]; // 4륜 구동
  Door left, right; // 양쪽 문
};

int main() {
  Car car;
  // 왼쪽 문에 달린 창문 올리기
  car.left.window.rollup();
}
```

1. 상속과 구성

[예제] 상속 (Inheritance)

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

class X {// 기존(Base) 클래스
  int i;
public:
  X() { i = 0; }
  void set(int ii) { i = ii; }
  int read() const { return i; }
  int permute() { return i = i * 47; }
};
```

실행 결과

```
sizeof(X) = 4
sizeof(Y) = 8
```

// X: int 하나 포함(4 bytes)

// Y: X의 int까지 둘 포함

```
class Y : public X { // 파생(Derived) 클래스
  int i; // Different from X's i
public:
  Y() { i = 0; }
  int change() { // 새로운 함수 정의
    i = permute(); // 다른 호출명(name call)
    return I;
  }
  void set(intii) { // 함수 재정의
    I = ii;
    X::set(ii); // 동일한 호출명
  }
};

int main()
{
  cout << "sizeof(X) = " << sizeof(X) << endl;
  cout << "sizeof(Y) = " << sizeof(Y) << endl;
  Y D;
  D.read(); // X의 함수 실행
  D.permute(); // X의 함수 실행
  D.change(); // Y에서 새로 정의된 함수 실행
  D.set(12); // Y에서 재정의된 함수 실행
}
```

1. 상속과 구성

다중 상속

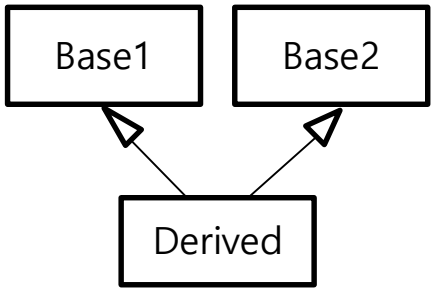
- 두 개 이상의 클래스로부터 멤버를 상속받아 파생 클래스를 생성 가능
- 여러 기존 클래스의 멤버를 상속받을 수 있는 강력한 방법
- 쉼표를 사용하여 상속받을 기존 클래스 명시

주의

- 상속받은 여러 기존 클래스에 같은 이름의 멤버 존재 가능
- 하나의 클래스를 간접적으로 두 번 이상 상속받을 가능성 존재

```
class 파생클래스명 : 접근제어지시자1 기존클래스명1, 접근제어지시자2 기존클래스명2 …
{
   // 파생 클래스 멤버 리스트
}
```

1. 상속과 구성



[예제] 다중 상속

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

class Base1 {
  int i;
public:
  Base1(int ii) : i(ii) {}
  int getI() { return i; }
};

class Base2 {
  int j;
public:
  Base2(int jj) : j(jj) {}
  int getJ() { return j; }
};
```

```
// 다중 상속 받은 파생 클래스
class Derived : public Base1, public Base2 {
  bool k;
public:
  // 다중 상속 받은 파생 클래스의 생성자
  Derived(int i, int j) : Base1(i), Base2(j) {}
  void print() {
    cout << getI()+getJ();
  }
};

int main() {
  Derived d(10,20);
  d.print();
}
```

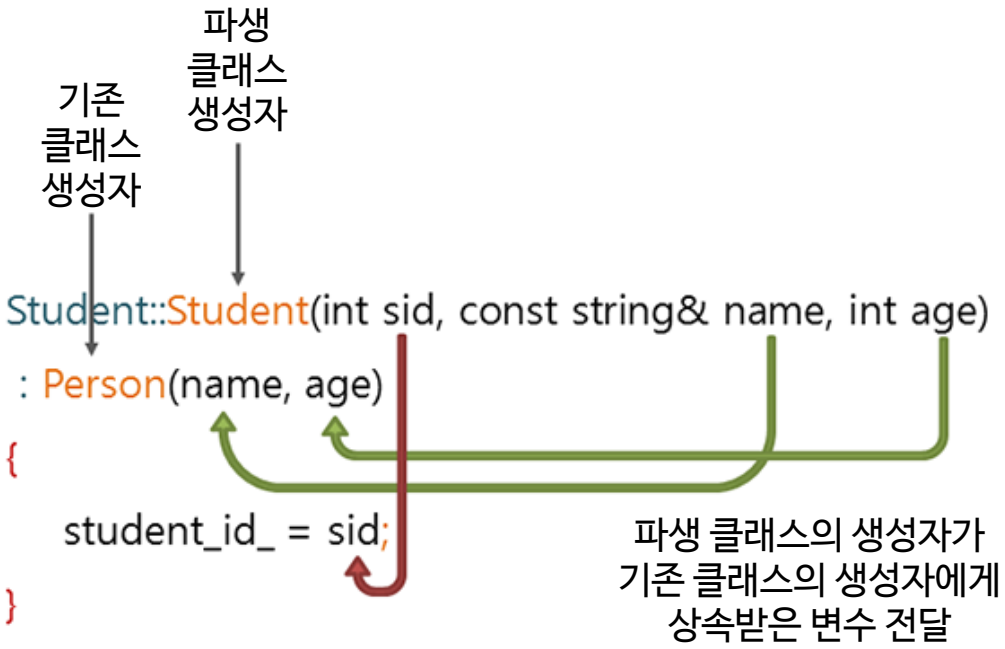
실행 결과

```
30
```

2. 생성자

생성자 초기화 목록 (Constructor initializer list)

- 생성자, 소멸자는 자동 상속되지 않음 (not automatically inherit)
- 파생 클래스의 생성자에서 기존 클래스 멤버를 호출하여 초기화



2. 생성자

[예제] 생성자 초기화 목록

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

class X { // 기존(Base) 클래스
  int i;
public:
  X(int ii) { i = ii; }
  void set(int ii) { i = ii; }
  int read() const { return i; }
  int permute() { return i = i * 47; }
};
```

```
class Y : public X { // 파생(Derived) 클래스
  int i; // Different from X's i
public:
  // 생성자 초기화 목록
  Y(int ix, int iy) : X(ix), i(iy) {}
  void print(){
    cout << "Member of X:" << read() << endl;
    cout << "Member of Y:" << i << endl;
  }
};

int main()
{
  Y D(42,43);
  D.print();
}
```

실행 결과

```
Member of X: 42
Member of Y: 43
```

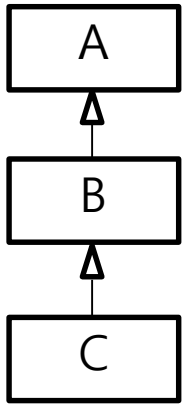
// X의 i값 출력

// Y의 i값 출력

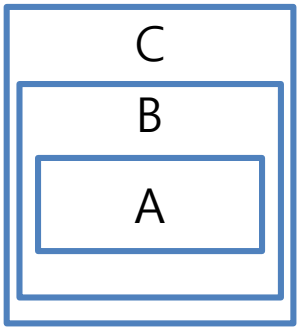
2. 생성자

클래스의 생성 및 소멸

- 객체 생성
  - 기존 클래스 생성자 호출 후 파생 클래스 생성자 호출
  - 부분 클래스 생성자 호출 후 전체 클래스 생성자 호출
- 객체 소멸: 객체 생성의 역순으로 진행



기존 클래스부터 생성

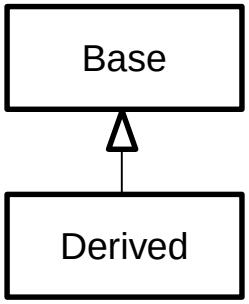


구성 클래스부터 생성
(부분 클래스부터)

A() → B() → C()
~C() → ~B() → ~A()

2. 생성자

[예제] 파생 클래스
생성자 초기화




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

class Base { // 기존 클래스
public:
  int i;
  // 클래스 생성자
  Base(int x){
    cout << "Constructor Base" << endl;
    i = x;
  }
};
```

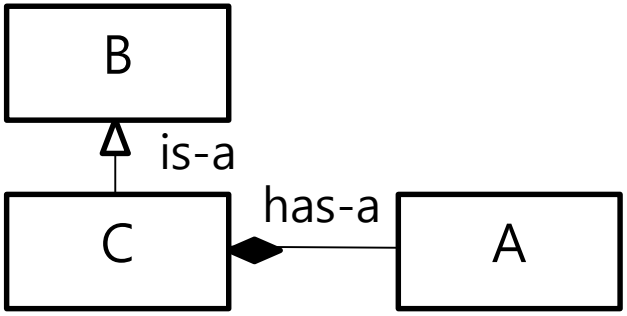
```
class Derived :public Base { // 파생 클래스
  int m;
public:
  Derived(int i) : Base(i) {
    m = i + 1;
    cout << "Constructor Derived" << endl;
  }
};
```

=

```
class Derived :public Base { // 파생 클래스
  int m;
public:
  Derived(int i) : Base(i), m(i + 1) {
    cout << "Constructor Derived" << endl;
  }
};
```

2. 생성자

[예제] 상속과 구성의 결합




```
//: C14:Combined.cpp
#include "pch.h"

class A {
  int i;
public:
  A(int ii) : i(ii) {}
  ~A() {}
  void f() const {}
};

class B {
  int i;
public:
  B(int ii) : i(ii) {}
  ~B() {}
  void f() const {}
};
```

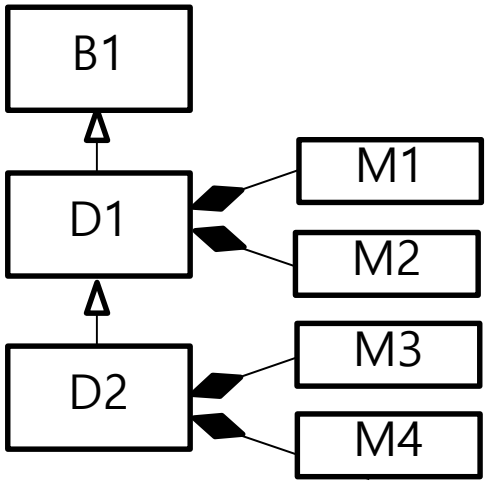
```
class C : public B { // C는 B의 파생 클래스
  A a; // A는 C의 부분 클래스
public:
  C(int ii) : B(ii), a(ii) {} // 생성자
  ~C() {} // 소멸자
  void f() const { // 함수 f 재정의
    a.f();
    B::f();
  }
};

int main()
{
  C c(47);
}
```

생성자
소멸자
실행 순서

```
B()
A()
C()
~C()
~A()
~B()
```

2. 생성자

[예제]
상속과
구성의
결합




```
class Derived1 : public Base1 {
  Member1 m1;
  Member2 m2;
  …
}
```

```
class Derived2 : public Derived2 {
  Member3 m3;
  Member4 m4;
  …
}
```

생성자 & 소멸자 실행 순서

```
Base1 constructor
Member1 constructor
Member2 constructor
Derived1 constructor
Member3 constructor
Member4 constructor
Derived2 constructor
Derived2 destructor
Member4 destructor
Member3 destructor
Derived1 destructor
Member2 destructor
Member1 destructor
Base1 destructor
```

3. 이름 은닉

이름 은닉

- 기존 클래스의 멤버 함수에 대해 상속 받은 파생 클래스에서 멤버 함수에 대해 **재정의**를 하면 기존 클래스의 함수가 **숨겨짐**
- 기존에 오버로딩 된 함수라고 해도 한 번만 재정의하면 모두 숨겨짐

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

class Base {
public:
  int f() { // 오버로딩
    cout << "Base::f()" << endl;
    return 0;
  }
  int f(string) { // 오버로딩
    return 0;
  }
};
```

```
class Derived1 : public Base {
public:
  // 재정의
  int f() {
    cout << "Derived2::f()" << endl;
    return 0;
  }
};

int main() {
  Derived1 d;
  d.f(); // Derived1의 f()가 실행됨
  // d.f("string"); 함수 호출에 인수가 너무 많습니다.
}
```

3. 이름 은닉

[예제] 이름 은닉

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

class Base {
public:
  int f() {
    cout << "Base::f()" << endl;
    return 0;
  }
  int f(string) { return 0; }
};
```

```
class Derived2 :public Base {
public:
  // Change return type -> 모든 기존의 f 함수 은닉
  void f() {
    cout << "Derived3::f()" << endl;
  }
};

class Derived3 :public Base {
public:
  // Change argument list -> 모든 기존의 f 함수 은닉
  int f(int) {
    cout << "Derived4::f()" << endl;
    return 0;
  }
};
```

오버로딩 된 함수도 재정의하면 모두 숨겨짐
- 반환 타입을 바꿔도, 인수를 바꿔도 성립

4. 접근 제어

[예제]

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

class Base {
  int i;
protected: // 파생 클래스에서 접근 가능
  void set(int ii) { i = ii; cout << i; }
public:
  Base(int ii = 0) :i(ii) {}
};

class Derived :public Base {
  int j;
public:
  Derived(int jj = 0) :j(jj) {}
  void change(int x) { set(x); } // set함수 사용
};

int main() {
  Derived d;
  d.change(10);
}
```

접근 제어 지시자

public	모든 접근 허용
protected	파생 클래스 허용
private (default)	모든 접근 제한

5. Subtyping

하위 클래스 및 타입

- 하위 클래스 (Subclass): 클래스의 상속
- 하위 타입 (Subtype): 다른 클래스의 기능을 활용하는 하위 타입
  아래 예제와 같이 구성을 통해 적용이 잘 안 됨

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

class FName { // ifstream의 하위 타입
  ifstream file; // 기능 활용이 목표
  string fileName;
public:
  FName(const string& fname) : fileName(fname), file(fname.c_str()) {}
  string name() { return fileName; }
};

int main() {
  FName file("FName1.cpp");
  cout << "name: " << file.name() << endl;
  string s;
  // getline(file, s); 에러 발생
  // file.close(); 에러: 멤버 함수가 아닙니다
}
```

5. Subtyping

[예제] 하위 타입: 상속을 통해 정의 가능

- FName 타입으로 ifstream 기능 활용 (getline(), close() 등)

실행 결과

```
name: Composition.cpp
#include "pch.h"
#include <iostream>
```

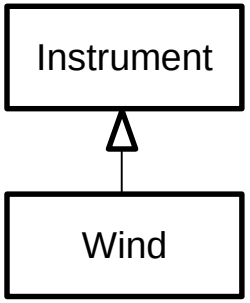
```cpp
int main() {
  FName file("Composition.cpp");
  cout << "name: " << file.name() << endl;
  string s;
  getline(file, s); // first line - OK
  cout << s << endl;
  getline(file, s); // second line - OK
  cout << s << endl;
  file.close(); // OK
}
```

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

class FName : public ifstream {// ifstream의 하위 타입
  string fileName;
public:
  FName(const string& fname) :ifstream(fname.c_str()), fileName(fname) {}
  string name() { return fileName; }
};
```

6. Upcasting

업캐스팅

- 기존 클래스와 파생 클래스 사이의 관계
- 파생 클래스는 기존 클래스의 (특수한) 타입
- 파생 클래스로부터 기존 클래스로의 캐스팅 (형 변환)




```
// Instrument.cpp
#include "pch.h"
// Inheritance & upcasting
enum note { middleC, Csharp, Cflat };

class Instrument { // 악기
public:
  void play(note) {}
};

// Wind (관악기) is a Instrument (악기)
// because they have the same interface:
class Wind : public Instrument {};
```

```
void tune(Instrument& i)
{
  i.play(middleC);
}

int main()
{
  Wind flute;
  tune(flute); // Upcasting
}
```

7. 복사 생성자

복사 생성자 (Copy-Constructor)

- 선언되는 객체와 같은 타입의 객체를 인수(argument)로 전달
- 인수(argument)는 참조자(&객체명)로 전달하지 않으면 무한루프

디폴트 복사 생성자

- 복사 생성자 정의 생략 시
  자동으로 복사 생성자 삽입
- 인수로 전달된 객체의
  모든 멤버 변수를
  선언되는 객체의
  모든 멤버 변수로 복사

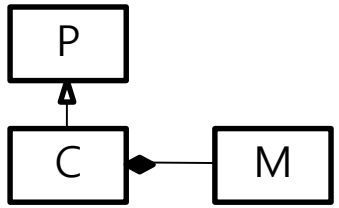
```
class AAA
{
public:
  AAA() {
    cout << "AAA() 호출" << endl;
  }
  AAA(int i) {
    cout << "AAA(int i) 호출" << endl;
  }
  AAA(AAA& a) { // 복사 생성자
    cout << "AAA(const AAA& a) 호출" << endl;
  }
};
```

[예제] 복사 생성자

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

class Parent {
  int i;
  public:
  Parent(int ii) : i(ii) {
    cout << "Parent(int)" << endl;
  }
  Parent(Parent& b) : i(b.i) {
    cout << "Parent(Parent&)" << endl;
  }
};

class Member {
  int i;
public:
  Member(int ii) : i(ii) {
    cout << "Member(int ii)" << endl;
  }
  Member(Member& m) : i(m.i) {
    cout << "Member(const Member&)" << endl;
  }
};
```

실행 결과

```
Parent(int)
Member(int ii)
Child(int)
Calling copy-constructor:
Parent(Parent&)
Member(const Member&)
```




```
class Child : public Parent {
  int i;
  Member m;
public:
  Child(int ii) : Parent(ii), i(ii), m(ii) {
    cout << "Child(int)" << endl;
  }
};

int main() {
  Child c(2);
  cout << "Calling copy-constructor: " << endl;
  Child c2 = c; // 복사 생성자 호출
}
```

7. 복사 생성자

[예제] 디폴트 복사 생성자

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

class Point
{
  int i, j;
public:
  Point(int ii, int jj) {
    i = ii;
    j = jj;
  }
  void ShowData() {
    cout << i << ' ' << j << endl;
  }
};
```

```
int main(void)
{
  Point p1(10, 20);
  Point p2(p1); // 디폴트 복사 생성자 호출
  p1.ShowData();
  p2.ShowData();
  return 0;
}
```

실행 결과

```
10 20
10 20
```

```
// 디폴트 복사 생성자
Point(const Point& p) {
  i = p.i;
  j = p.j;
}
```

공지

평가 기준(100%)

- 시험 (50%): **중간고사 (25%)**, 기말고사 (25%)
- 학기말 과제(30%): Term Project 제안/발표/결과
- 기타 (20%): 출석 (10%), 실습 과제 (10%)

중간고사 공지

- 일정 및 장소: 10/26(금), 강의실
- 범위: **1주~6주 수업 내용**
  - 슬라이드 범위를 벗어난 내용을 포함하지 않음
    (교재에 나왔어도 수업에서 다루지 않은 내용은 포함 안 함)
  - 개념 이해 및 용어에 대해 묻는 문제, 객관식 및 단답형 위주
  - 코드의 해석, 빈 칸 채우기, 결과를 묻는 문제 포함
    (슬라이드에 나온 예제 코드 또는 약간 변형된 수준)

문제 예시

다음 시간에 제안서 발표 후 내용을 간략히 리뷰하고 예시 문제를 설명할 예정

문항 예시

- 객관식
  - (개념/용어) ~의 주요 개념을 설명한 것 중 거리가 먼 것은?
  - (코드 문법) ~클래스를 정의한 것이다. ()안에 알맞은 것을 적은 것은?
  - (코드 해석) ~와 같이 하면 문제가 생긴다. 문제점/해결책으로 알맞은 것은?
- 단답형
  - (개념/용어) 다음 설명에 해당하는 ~를 쓰시오.
  - (코드 문법) 다음과 같은 출력이 나오도록 ()에 알맞은 내용을 쓰시오.
- 서술형
  - (개념/용어) ~에 대해/~를 사용하는 이유에 대해 간략히 설명하시오.
  - (코드 해석) 다음 코드의 실행 결과를 쓰시오. / 개선 방안을 설명하시오.

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**	

질문 및 답변