항목 31: 파일 사이의 컴파일 의존성을 최대로 줄이자

구현부에 있는 코드 몇 줄 살짝 손봤을 뿐인데 건드리지 않은 코드까지 전부 다시 컴파일 되어 빌드 시간이 길어지는 문제를 해결하고자 합니다.

문제는 #include “헤더.h” 입니다. A라는 헤더파일을 include하면 컴파일 의존성이란 것을 엮어 버립니다. 엮여버렸기 때문에 A헤더 파일이 수정되면 include한 헤더까지 덩달아 컴파일 되어야 합니다.

컴파일러는 객체들의 크기를 전부 알아야 컴파일할 수 있습니다. 다음 예를 보여드리겠습니다.

int main( )

{

int x; // int 하나를 정의합니다.

Person p( params ); // Person 하나를 정의합니다.

}

컴파일러는 int x를 만나면 int를 담을 충분한 공간을 할당 해야 하는 것을 알고 있습니다. 그러나 Person에서 컴파일러는 얼마큼 공간을 할당해야 할지 모르는 것입니다. 컴파일러가 Person의 크기를 알려면 정의된 부분을 보는 방법 밖에 없습니다.

클래스를 두 개로 나눕니다. 한쪽은 인터페이스만 제공하고, 또 다른 한쪽은 그 인터페이스의 구현을 맡도록 만드는 것입니다.

인터페이스 클래스

#include<string>

#include<memory>

class PersonImpl; // Person의 구현 클래스에 대한 전방 선언

class Date; // Person 클래스 안에서 사용하는 부속 클래스 전방 선언

class Address;

class Person {

public:

Person(const std::string& name, const Date& birthday, const Address& addr);

std::string name( ) const;

std::string birthDate( ) const;

std::string address( ) const;

. . .

private: // pimpl (“pointer to implementation”)

std::tr1::shared\_ptr<PersonImpl> pImpl; // 구현 클래스 객체에 대한 포인터

};

이렇게 인터페이스 클래스를 만들면 Person class를 사용하는 사용자들은 구현부분 클래스가 바뀌더라도 재컴파일 없이 사용할 수 있습니다. 게다가 구현부를 들여다 볼 수 없기 때문에 캡슐화에 도움이 됩니다.

인터페이스와 구현을 둘로 나누는 것은 “정의부에 대한 의존성을 선언부에 대한 의존성으로 바꾸는 것” 입니다. 즉, 헤더 파일을 만들 때는 실용적으로 의미를 갖는 선언 형태로 만듭니다.

* 객체 참조자 및 포인터로 충분한 경우에는 객체를 직접 쓰지 않습니다.

어떤 타입에 대한 참조자 및 포인터를 정의할 때는 그 타입의 선언부만 필요합니다. 반면에 어떤 타입의 객체를 정의할 때는 그 타입의 정의가 준비되어 있어야 합니다.

* 할 수 있으면 클래스 정의 대신 클래스 선언에 최대한 의존하도록 만듭니다.

어떤 클래스를 사용하는 함수를 선언할 때는 그 클래스의 정의를 가져오지 않아도 됩니다. 심지어 그 클래스 객체를 값으로 전달하거나 반환하더라도 클래스 정의가 필요 없습니다.

class Date; // 클래스 선언 (정의 없이 가능)

Date today( ); // 가능. Date 클래스 today( )함수 선언

void clearAppointments(Date d); // 가능 매개변수