Chapter 8

Index

friend 함수 및 클래스 : 클래스에서 선언하는 접근 제어 지시자의 영향을 받지 않도록 해준다. 기존 클래스와 클래스 상속 관계에 영향을 받지 않고 새로운 관계 형성 가능

집합 관계 : friend 함수와 클래스 선언을 이용해서 각 요소를 의미에 따라 묶는 구조화

friend

```
friend class 클래스이름;
friend 함수원형선언;
```

- friend로 선언한 함수나 클래스는 접근 제어 지시자의 영향을 안 받는다

friend 함수

▼ **friend 함수 코드**

```
/* friend 함수 */
#include "pch.h"
#include <iostream>
using namespace std;
```

```
class CMyData
{
public:
    CMyData(int nParam) : m_nData(nParam) { }
    int GetData() const { return m_nData; }
    void SetData(int nParam) { m_nData = nParam; }

    // friend 함수로 선언한다.
    friend void PrintData(const CMyData &);

private:
    int m_nData = 0;
};

void PrintData(const CMyData& rData)
{
    // 프랜드 함수이므로 접근 제어 지시자의 영향을 받지 않고
    // private 멤버에 직접 접근한다.
    cout << "PrintData():" << rData.m_nData << endl;
}

int main()
{
    CMyData a(5);
    PrintData(a);

    return 0;
}
```

freind 클래스

▼ friend 예약어를 선언한 클래스 코드

```
/* frined 예약어를 선언한 클래스 */
#include "pch.h"
#include <iostream>
using namespace std;

class CNode
{
    // friend 클래스 선언
    friend class CMyList;

public:
    explicit CNode(const char* pszName)
    {
        strcpy_s(m_szName, sizeof(m_szName), pszName);
    }
```

```
private:
    // 단일 연결 리스트로 관리할 데이터
    char m_szName[32];
    CNode* pNext = nullptr;
};

class CMyList
{
public:
    CMyList() : m_HeadNode("Dummy Head") { }
    ~CMyList()
    {
        // 리스트에 담긴 데이터들을 모두 출력하고 삭제
        CNode* pNode = m_HeadNode.pNext;
        CNode* pDelete = nullptr;

        while (pNode)
        {
            pDelete = pNode;
            pNode = pNode->pNext;

            cout << pDelete->m_szName << endl;
            delete pDelete;
        }

        m_HeadNode.pNext = nullptr;
    }

    void AddNewNode(const char* pszName)
    {
        CNode* pNode = new CNode(pszName);

        // 리스트에 새로운 노드를 추가
        pNode->pNext = m_HeadNode.pNext;
        m_HeadNode.pNext = pNode;
    }

private:
    CNode m_HeadNode;
};

// 사용자 코드
int main()
{
    // 메모리 추가/삭제 코드가 등장하지 않는다.
    CMyList list;
    list.AddNewNode("길동");
    list.AddNewNode("철수");
    list.AddNewNode("영희");
```

```
    return 0;
}
```

집합 관계

- 여럿이 모여 새로운 하나를 이루는 경우
- **Composition** : 인스턴스가 소멸하면 구성 요소들도 모두 ‘함께 소멸’한다
- **Aggregation** : 모두 모여 하나의 시스템을 이루지만 개별적으로도 분리되어 독립적으로 활용

> 모여서 뗄 수 없는 한 덩어리를 이루는가 아니면 각자 독립적인 것들이 모여서 만들어진 분리 가능 집합체인가

Composition 관계

▼ Composition 관계 코드

```
/* Composition 관계 */
#include "pch.h"
#include <iostream>
using namespace std;

// 자료구조 클래스
class CNode
{
    // friend 클래스 선언
    friend class CMyList;

public:
    explicit CNode(const char* pszName)
    {
        strcpy_s(m_szName, sizeof(m_szName), pszName);
    }

private:
    // 단일 연결 리스트로 관리할 데이터
    char m_szName[32];
    CNode* pNext = nullptr;
};

class CMyList
{
public:
```

```
    CMyList() : m_HeadNode("Dummy Head") { }
    ~CMyList()
    {
        // 리스트에 담긴 데이터들을 모두 출력하고 삭제
        CNode* pNode = m_HeadNode.pNext;
        CNode* pDelete = nullptr;

        while (pNode)
        {
            pDelete = pNode;
            pNode = pNode->pNext;
            delete pDelete;
        }
        m_HeadNode.pNext = nullptr;
    }

    // 리스트에 새로운 노드 추가
    void AddNewNode(const char* pszName)
    {
        CNode* pNode = new CNode(pszName);

        pNode->pNext = m_HeadNode.pNext;
        m_HeadNode.pNext = pNode;
    }

    // 리스트의 모든 노드 값을 출력
    void Print()
    {
        CNode* pNode = m_HeadNode.pNext;
        while (pNode)
        {
            cout << pNode->m_szName << endl;
            pNode = pNode->pNext;
        }
    }

private:
    CNode m_HeadNode;
};

// UI클래스
class CMyUI
{
public:
    // 메뉴 출력 및 사용자 입력 확인
    int PrintMenu()
    {
        system("cls");
        cout << "[1]Add\t" << "[2]Print\t" << "[0]Exit\n";
        cout.flush();
        int nInput = 0;
        cin >> nInput;
```

```
        return nInput;
    }

    // 지속적으로 메뉴를 출력하는 메인 이벤트 반복문
    void Run()
    {
        char szName[32];
        int nInput = 0;

        while ((nInput = PrintMenu()) > 0)
        {
            switch (nInput)
            {
            case 1:   // Add
                cout << "이름: ";
                cout.flush();
                cin >> szName;
                m_list.AddNewNode(szName);
                break;

            case 2:   // Print
                m_list.Print();
                break;

            default:
                break;
            }
        }
    }

private:
    // UI 클래스 내부에 자료구조가 포함된다.
    CMyList m_list;
};

// 프로그램 시작
int main()
{
    CMyUI ui;
    ui.Run();

    return 0;
}
```

Aggregation 관계

▼ Aggregation 관계 코드

```
/* Aggregation 관계 */
#include "pch.h"
#include <iostream>
using namespace std;

// 자료구조 클래스
class CNode
{
    // friend 클래스 선언
    friend class CMyList;

public:
    explicit CNode(const char* pszName)
    {
        strcpy_s(m_szName, sizeof(m_szName), pszName);
    }

private:
    // 단일 연결 리스트로 관리할 데이터
    char m_szName[32];
    CNode* pNext = nullptr;
};

class CMyList
{
public:
    CMyList() : m_HeadNode("Dummy Head") { }
    ~CMyList()
    {
        // 리스트에 담긴 데이터들을 모두 출력하고 삭제
        CNode* pNode = m_HeadNode.pNext;
        CNode* pDelete = nullptr;

        while (pNode)
        {
            pDelete = pNode;
            pNode = pNode->pNext;
            delete pDelete;
        }
        m_HeadNode.pNext = nullptr;
    }

    // 리스트에 새로운 노드 추가
    void AddNewNode(const char* pszName)
    {
        CNode* pNode = new CNode(pszName);

        pNode->pNext = m_HeadNode.pNext;
        m_HeadNode.pNext = pNode;
    }
```

```
    // 리스트의 모든 노드 값을 출력
    void Print()
    {
        CNode* pNode = m_HeadNode.pNext;
        while (pNode)
        {
            cout << pNode->m_szName << endl;
            pNode = pNode->pNext;
        }
    }

private:
    CNode m_HeadNode;
};

// UI클래스
class CMyUI
{
public:
    //참조 멤버는 반드시 초기화 목록을 이용해 초기화해야 한다.
    CMyUI(CMyList &rList) : m_list(rList) { }

    // 메뉴 출력 및 사용자 입력 확인
    int PrintMenu()
    {
        system("cls");
        cout << "[1]Add\t" << "[2]Print\t" << "[0]Exit\n";
        cout.flush();
        int nInput = 0;
        cin >> nInput;

        return nInput;
    }

    // 지속적으로 메뉴를 출력하는 메인 이벤트 반복문
    void Run()
    {
        char szName[32];
        int nInput = 0;

        while ((nInput = PrintMenu()) > 0)
        {
            switch (nInput)
            {
            case 1:   // Add
                cout << "이름: ";
                cout.flush();
                cin >> szName;
                m_list.AddNewNode(szName);
                break;
```

```
            case 2:   // Print
                m_list.Print();
                break;

            default:
                break;
            }
        }
    }

private:
    // UI 클래스 내부에 자료구조 객체에 대한 참조만 존재한다.
    CMyList &m_list;
};

// 프로그램 시작
int main()
{
    // 자료구조와 UI 객체를 별도로 선언하고 연결한다.
    CMyList list;
    CMyUI ui(list);
    ui.Run();

    return 0;
}
```

> 프로그램의 제어 시스템, 사용자 인터페이스, 원본 데이터는 분리하는 것이 원칙

문제

1. **friend 함수가 가지는 특권?**

   접근제어지시자를 무시하고 접근 가능

2. **A 클래스의 멤버로 B 클래스가 사용됐다면 두 클래스는 어떤 관계?**

   Composition 관계