객체지향프로그래밍
실습과제보고서 #11

과목_ 객체지향 프로그래밍
교수님_ 최경주 교수님
제출일_ 2019 년 5 월 28 일
제출자_ 소프트웨어학과 김예원

실습과제 #11: 객체지향 방식의 성적처리프로그램 #4 작성

: IOInterface Class를 작성하고 Subject 및 Student Class에 상속해 기본클래스와 파생클래스의 관계에 대해 터득한다. 또한 기존의 Modify() 함수를 변경해 검색된 과목만 수정하도록 한다.

1. 기본 클래스 IOInterface 클래스 작성

***Student Class와 함수 정의- 기본클래스 작성(1), (2)**

```
class IOInterface {
protected:
        string m_name;
        int m_data;

public:
        IOInterface();
        IOInterface(string m_name);
        ~IOInterface();

        void InputData();
        void PrintData() const;
        void Modify();

        string GetName() const;
        int GetData() const;
};
```

```
IOInterface::IOInterface()
{
        cout << "IOInterface의 디폴트 생성자 호출" << endl;
        m_name = " ";
        m_data = 0;
}

IOInterface::IOInterface(string m_name)
{
        cout << "IOInterface의 인자있는 생성자 호출" << endl;
        this->m_name = m_name;
}

//소멸자
IOInterface::~IOInterface()
{
        cout << "IOInterface 소멸자 호출₩n";
        //아무것도 하지 않는다
```

```
}

//멤버함수
void IOInterface::InputData() //화면으로부터 입력받는다
{
        InputUtil::InputValue(m_name);
        InputUtil::InputValue(m_data);
}

void IOInterface::PrintData() const //값을 화면에 출력한다
{
        cout << this->m_name;
        cout << this->m_data;
}

void IOInterface::Modify() //화면에서 입력받은 값으로 수정
{
        InputUtil::InputValue(m_name);
        InputUtil::InputValue(m_data);
}

string IOInterface::GetName() const //이름 정보를 출력해준다
{
        return this->m_name;
}

int IOInterface::GetData() const //데이터의 정보를 출력해준다
{
        return this->m_data;
}
```

m_name 과 m_data 멤버변수를 가진 IOInterface() 클래스를 생성했다.

PPT에 나온 테스트 케이스를 작성할 땐 기본 클래스의 멤버함수가 필요해서, 상속받은 멤버 함수들이 오버라이딩 되지 않도록 파생 클래스의 멤버함수 몇 개를 주석처리 했다.

2. 파생클래스 작성

*파생클래스 Subject, Student

```
class Student:public IOInterface
```

```
class Subject: public IOInterface
```

***파생클래스 생성자 수정**

```
Student::Student():IOInterface()

Student::Student(string name, int hakbun, int subnum, Subject *
sub):IOInterface(name)
```

```
Subject::Subject():IOInterface()

Subject::Subject(string name, int hakjum, string grade):IOInterface(name)
```

멤버변수를 초기화하는 디폴트 생성자를 포함해서 인자 있는 생성자를 상속했다. 값을 받는 변수가 겹치는 m_name=name 부분은 기본클래스를 상속하면서 삭제했다.

***테스트**

```
void main() {
        Student std;
        std.InputData();
        cout << std.GetData() << "₩n";

}
```

m_data는 초기화 되지 않은 멤버변수이기 때문에 원래의 소스대로 출력하면 쓰레기 값이 나온다. 따라서 초기의 소스코드에 InputData함수를 이용해 값을 입력하기도 했고, 생성자에 멤버변수를 초기화 하게 만들어 m_data를 0으로 초기화했다.

```
<<< Student 디폴트 생성자 호출 >>>
김예원
300
300
<<< Student 소멸자 호출 >>>
IOInterface 소멸자 호출

C:₩Users₩김예원₩source₩repos₩객프과제11₩Debug₩객프과제11.exe(63
60 프로세스)이(가) 0 코드로 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.
```

```
<<< Student 디폴트 생성자 호출 >>>
0
<<< Student 소멸자 호출 >>>
IOInterface 소멸자 호출

C:₩Users₩김예원₩source₩repos₩객프과제11₩Debug₩객프과제11.exe(55
56 프로세스)이(가) 0 코드로 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.
```

추가로 0으로 초기화 된 m_data를 출력한 모습이다.

3. 기본클래스와 파생클래스의 관계

***소스코드**

```
void main() {
	Subject sub("컴퓨터", 3, "C");
	Subject sub2(100); //생성자를 호출해서 m_data를 초기화
	cout << "----------------------------₩n";
	cout << "m_data : ";
	cout << sub2.GetData()<< endl; //이상함
	cout << "교과목 이름 : ";
	cout << sub.GetName() << endl;
	cout << "부모클래스의 이름 : ";
	cout << sub.GetName() << endl; //출력이 아예 안됨
	cout << "----------------------------₩n";

	Student std("홍길동", 2018038032, 1, &sub);
	Student std2(300); //생성자를 호출해서 m_data를 초기화
	cout << "----------------------------₩n";
	cout << "m_data : ";
	cout << std2.GetData()<<endl;
	cout << "학생 이름 : ";
	cout << std.GetName()<<endl;
	cout << "부모클래스의 이름 : ";
	cout << std.GetName()<<endl;
	cout << "----------------------------₩n";

}
```

m_data에 접근해 m_data를 초기화 하기 위해 기본클래스와 파생클래스에 새로운 생성자를 만들었고, 상속해주었다.

***IOInterface 클래스(기본클래스)**

```
IOInterface::IOInterface(int m_data) {
	this->m_data = m_data;
}
```

***Student 클래스**

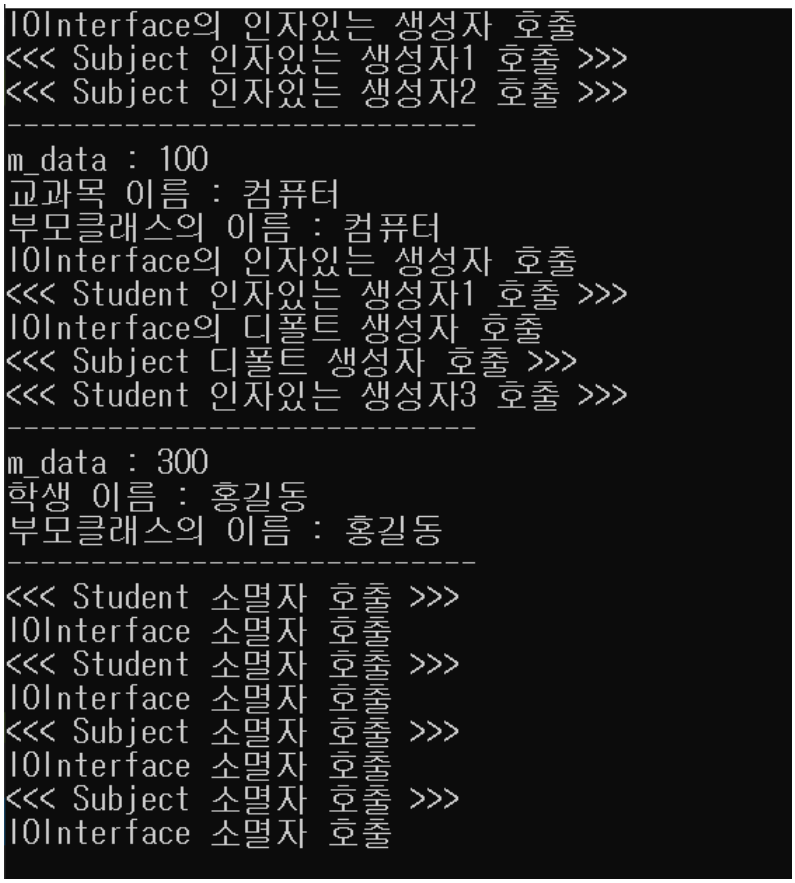
```
Student::Student(int data) :IOInterface(data) //data를 초기화하기 위한 생성자
{
	cout << "<<< Student 인자있는 생성자3 호출 >>>" << endl;
```

```
}
```

***Subject 클래스**

```
Subject::Subject(int data):IOInterface(data) { //data를 초기화하기 위한 생성자
	cout << "<<< Subject 인자있는 생성자2 호출 >>>" << endl;

}
```

***실행결과**



4. Student Class의 Modify() 멤버함수 수정

***소스코드**

<과목정보를 화면에 입력했을 때>

```
else if (Type == "과목정보") { //검색된 과목 정보 수정
		string sname;
```

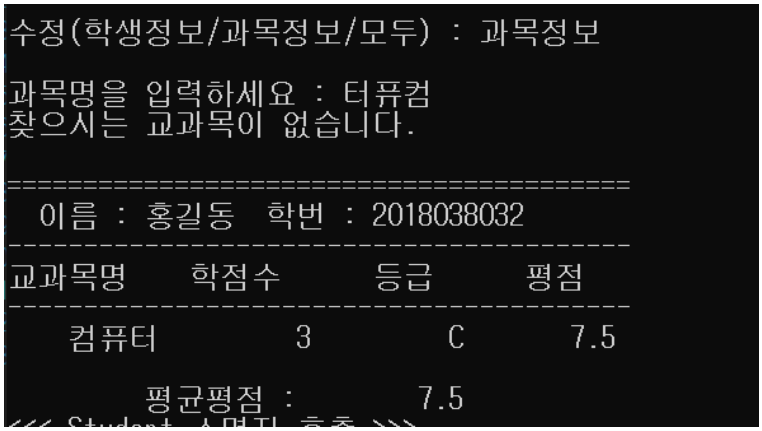
```
		int i;
		cout << "과목명을 입력하세요 : ";
		InputUtil::InputValue(sname);

		Subject *p = NULL;
		p = SubSearch(sname);
		if (p != NULL) { //주소를 넘겨받았을 경우 수정한다
			p->Modify();
		}

		cout << endl;
	}
```

***테스트**

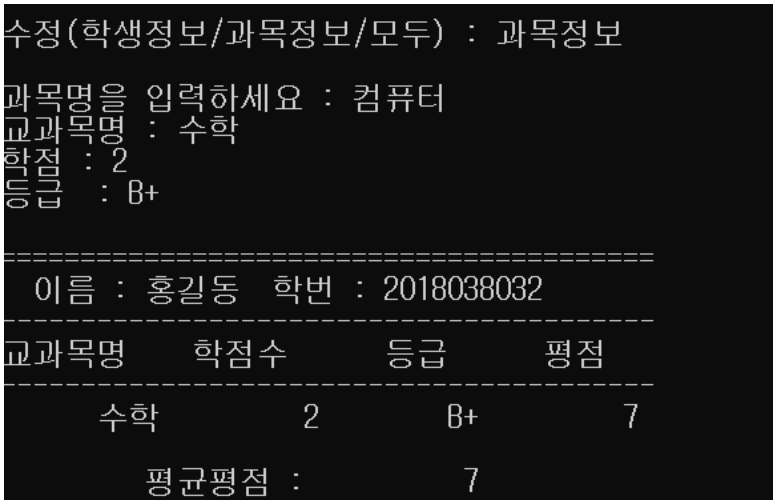
```
void main() {
	Subject sub("컴퓨터", 3, "C");
	Student std("홍길동", 2018038032, 1, &sub);

	std.PrintData();
	std.Modify();
	std.PrintData();

}
```

```
========================================
  이름 : 홍길동  학번 : 2018038032
----------------------------------------
교과목명    학점수      등급      평점
----------------------------------------
   컴퓨터         3         C       7.5

        평균평점 :        7.5


수정(학생정보/과목정보/모두) : 학생정보

<학생 정보 수정>
이름 : 이소온
학번 : 201820182018
잘못된 정보가 입력됨
다시 입력하세요: 2018

========================================
  이름 : 이소온  학번 : 2018
----------------------------------------
교과목명    학점수      등급      평점
----------------------------------------
   컴퓨터         3         C       7.5

        평균평점 :        7.5
```

1. "학생정보"를 입력했을 때



2. "과목정보" 입력 후 잘못된 값을 입력했을 때



3. "과목정보" 입력 후 올바른 값 입력했을 때

느낀점

상속에 대해서 깊이 공부할 수 있는 기회였다. 사실 수업시간에는 생성자의 상속에 대해서 잘 이해가 되지 않았었다. 기본 클래스의 생성자가 정확히 파생클래스의 생성자에 무슨 일을 하는지 몰랐다가, 전공책을 천천히 다시 읽어보면서 실습을 했다가 변수명을 교체해보면서 느끼게 되었다.

과제를 할 때 마다 느끼는 거지만, 여러가지 테스트 케이스 별로 소스코드를 조작해야하는데, 세세하게 모두 조작하지 못해서 오류가 생긴다. 이번 과제도 마찬가지였는데, 기본클래스와 파생클래스의 관계에 대해 과제를 수행하면서 문제를 겪었다. m_data 때문이었다. m_data가 상속됨을 확인하기 위해 파생클래스에 생성자를 만들었고, 호출했다. 생성자도 호출되고, 출력되는 값도 정상적이었으나, 소멸자가 호출되지 않았다. 소멸자가 호출되지 않는 이유를 알고 싶어서 디버깅해봤지만, 디버깅이 걸리는 곳은 this 연산자와 관련된 곳이었다. 하지만 사실 동적할당 해제가 필요 없는데 소멸자에 동적할당 하도록 코딩 되어있어서 문제였다. 동적할당 부분에 주석처리 해주니까 제대로 소멸자가 호출됨을 확인할 수 있었다. 사실 근본적인 문제는 어쩌면 아직도 동적할당과 생성자와 관련된 부분의 개념이 부족해서가 아닐까 하는 생각이 든다. 이번 기회를 통해서 제대로 다시 한 번 점검하게 된 것 같아서 좋다.