[객체지향 프로그래밍]

[과제3 다형성 활용하기]

과제 체크리스트	확인
1. 이 과제물은 내가 직접 연구하고 작성한 것이다.	예
2. 인용한 모든 자료(책, 논문, 보고서, 인터넷 자료 등)의 출처를 밝혔다.	예
3. 정확한 출처 제시나 사용 허락 없이 다른 사람의 아이디어를 가져오지 않았다.	예
4. 이 과제물을 다른 사람으로부터 받거나 구매하여 제출하지 않았다.	예
5. 이 과제의 결과물에 나의 학번 이름이 출력되어 있다.	예

■ 담당 교수 : 지정희 교수님

■ 학 번 : 201711430

■ 이 름 : 차준혁

■ 제출일 : 2021년 05월 14일

1 문제정의 및 분석

Ticket의 판매현황을 관리할 수 있는 TicketManager 클래스를 생성해야 합니다. TicketManager에서 Ticket객체의 포인터를 담을 수 있는 배열을 동적 할당하여 생성합니다. 그리고 Ticket클래스의 자식 객체인 AdvancedTicket객체와 GeneralTicket객체를 동적 할당하여 생성하고 부모 클래스 Ticket의 포인터로 자식 객체를 가리킵니다. Ticket클래스가 기본 클래스의 역할을 하게 만들기 위해서 getPrice() 함수를 순수 가상 함수로 선언하여 Ticket클래스를 추상 클래스로 만듭니다. TicketManager에서 Ticket의 포인터로 자식 객체를 가리킬 때 AdvancedTicket객체나 GeneralTicket객체에서 타입에 따라 호출 함수를 결정하게 하기 위해서 virtual이란 예약어를 사용해서 동적 바인딩에 의해 함수를 결정하게 만듭니다.

2 주요 소스코드 설명

```
#pragma once
class Ticket
{
protected:
	int number;
	double price;
	double finalprice;
public:

	static int order;
	Ticket();
	Ticket(double price);
	virtual ~Ticket();
	int getNumber() const;
	virtual  double getPrice() const=0;
	void setPrice(const double& price);
	virtual void show() const;
	virtual double setFinalPrice()=0;
};
```

티켓이 생성될 때마다 티켓 번호를 부여하기 위해서 static멤버인 order를 선언합니다. 전역 변수로 생성하고 Ticket객체가 생성될 때마다 +1 합니다.

판매가격과 지불금액이 다를 수 있기 때문에 finalprice라는 변수를 선언하여 지불금액을 따로 저장합니다.

Ticket소멸자를 virtual로 만듭니다.

getPrice(), show(), setFinalPrice()를 virtual 예약어를 붙여서 가상 함수로 만들어서 함수 호출을 동적 바인딩을 통해 하도록 만듭니다.

```
#pragma once
#include "Ticket.h"
class AdvancedTicket :
        public Ticket
{
private:
        int advanceDays;


public:
        AdvancedTicket();
        AdvancedTicket(double price, int advanceDays);
        ~AdvancedTicket();
        double getPrice() const;
        int getAdvanceDays() const;
        void show() const;
        double setFinalPrice();
};

double AdvancedTicket::setFinalPrice()
{
        if (advanceDays >= 30) {
                this->finalprice = this->price * 0.5;
        }
        else if (advanceDays >= 20) {
                this->finalprice = this->price * 0.7;
        }
        else if (advanceDays >= 10) {
                this->finalprice = this->price * 0.9;
        }
        else {
                this->finalprice = this->price;
        }
        return this->finalprice;
}
```

setFinalPrice() 함수에서 기간에 따라서 달라지는 지불 금액을 finalprice에 저장합니다.

```
#pragma once
#include "Ticket.h"
class GeneralTicket :
        public Ticket
{
private:
        bool payByCredit;
public:
        GeneralTicket();
        GeneralTicket(double price, bool b);
        ~GeneralTicket();
        double getPrice() const;
        bool getPayByCredit() const;
```

```
        void show() const;
        double setFinalPrice();
};
double GeneralTicket::setFinalPrice()
{
        if (this->payByCredit) {

                this->finalprice = price * 1.1;
        }
        else {
                this->finalprice = price;
        }
        return this->finalprice;
}
```

setFinalPrice()함수에서도 카드 결제 여부에 따라 달라지는 지불 금액을 finalprice 변수에 저장합니다.

```
#pragma once

#include "Ticket.h"
#include <iostream>
#include <string>
using namespace std;
class TicketManager
{
private:
        string m_name;
        int totalNumber;
        Ticket** ticket = nullptr;
        int count = 0;
        double sum = 0;
public:
        TicketManager() = delete;
        ~TicketManager();
        TicketManager(const string& name, const int number);
        void buy(Ticket* t);
        void show() const;
        void showGeneralTicket(const bool& card);
        void showAdvancedTicket();
        int returnCount() const;
        friend ostream& operator<<(ostream& out, const TicketManager& manager);

};
```

생성되는 총 ticket의 개수를 저장하기 위한 count 변수와 총 예약금액을 저장하기 위해서 sum 변수를 선언합니다.

returnCount()함수는 TicketManager가 생성한 총 Ticket객체의 개수를 반환하는 함수입니다.

```
#include "TicketManager.h"
#include "Ticket.h"
#include "AdvancedTicket.h"
#include "GeneralTicket.h"
#include <iostream>
using namespace std;
```

```
TicketManager::~TicketManager()
{
	if (ticket != nullptr) {
		for (int i = 0; i < (this->count); i++) {
			delete ticket[i];
		}
		delete[] ticket;
	}
	ticket = nullptr;
}

TicketManager::TicketManager(const string& name, const int number)
{
	this->m_name = name;
	this->totalNumber = number;
	this->ticket = new Ticket * [this->totalNumber];
}
```

Ticket포인터를 원소로 가지는 배열을 동적 할당하여 생성합니다.

```
void TicketManager::buy(Ticket* t)
{
	ticket[this->count] = t;
	this->count = this->count + 1;
	this->sum += t->setFinalPrice();
}
```

Ticket객체를 가리키는 포인터로 Ticket객체의 자식클래스를 가리키고 생성합니다. 총 예약금액을 구하기 위해서 setFinalPrice()함수를 활용합니다.

```
void TicketManager::show() const
{
	for (size_t i = 0; i < this->count; i++) {
		(this->ticket[i])->show();
		cout << endl;
	}
	cout << "-------------------------------------" << endl;
	cout << "총 예약 금액 : "<<this->sum << endl;
	cout << "-------------------------------------" << endl;

}
```

Ticket객체를 가리키는 포인터를 통해서 show()함수를 호출하는데 이 때 show()함수는 virtual 함수이므로 동적 바인딩에 의해 Ticket객체를 가리키는 타입인 포인터가 가리키는 Ticket클래스의 자식 객체에 따라서 호출되는 함수가 달라지게 됩니다.

```
void TicketManager::showGeneralTicket(const bool& card)
{
	double temp = 0;
	cout << "------------일반 예약 현황-----------" << endl;
	for (size_t i = 0; i < this->count; i++) {
		if (dynamic_cast<GeneralTicket*>(ticket[i])) {
			if (static_cast<GeneralTicket*>(ticket[i])->getPayByCredit() == card) {
				ticket[i]->show();
				temp += ticket[i]->setFinalPrice();
			}
		}
```

```
        }
        if (temp) {
                cout << "-------------------------------------" << endl;
                cout << "총 예약 금액 : " << temp << endl;
                cout << "-------------------------------------" << endl;
        }
}
```

GeneralTicket객체의 예약 현황만 보여주기 위해서 dynamic_cast를 활용합니다. 포인터 타입인 경우 GeneralTicket 타입이 아니면 nullptr을 반환하기 때문에 GeneralTicket타입일때만 show()함수를 호출하게 됩니다.

```
void TicketManager::showAdvancedTicket()
{
        double temp = 0;
        cout << "------------사전 예약 현황-----------" << endl;
        for (size_t i = 0; i < this->count; i++) {
                if (dynamic_cast<AdvancedTicket*>(ticket[i])) {
                        ticket[i]->show();
                        temp += ticket[i]->setFinalPrice();
                        cout << endl;
                }
        }
        if (temp) {
                cout << "-------------------------------------" << endl;
                cout << "총 예약 금액 : " << temp << endl;
                cout << "-------------------------------------" << endl;
        }
}
```

AdvancedTicket객체의 예약 현황만 보여주기 위해서 dynamic_cast를 활용합니다. 포인터 타입인 경우 AdvancedTicket 타입이 아니면 nullptr을 반환하기 때문에 AdvancedTicket타입일때만 show()함수를 호출하게 됩니다.

```
int TicketManager::returnCount() const
{
        return this->count;
}

ostream& operator<<(ostream& out, const TicketManager& manager)
{
        out << "-----------티켓 예약 현황------------" << endl<<endl;
        out << "총 예약 매수 : " << manager.returnCount() << endl;
        out << "-------------------------------------" << endl;
        manager.show();
        out << endl;
        return out;
}
```

<< 연산자 오버라이딩을 통해 티켓 예약 현황을 보여줍니다.

```
#include "TicketManager.h"
#include "Ticket.h"
```

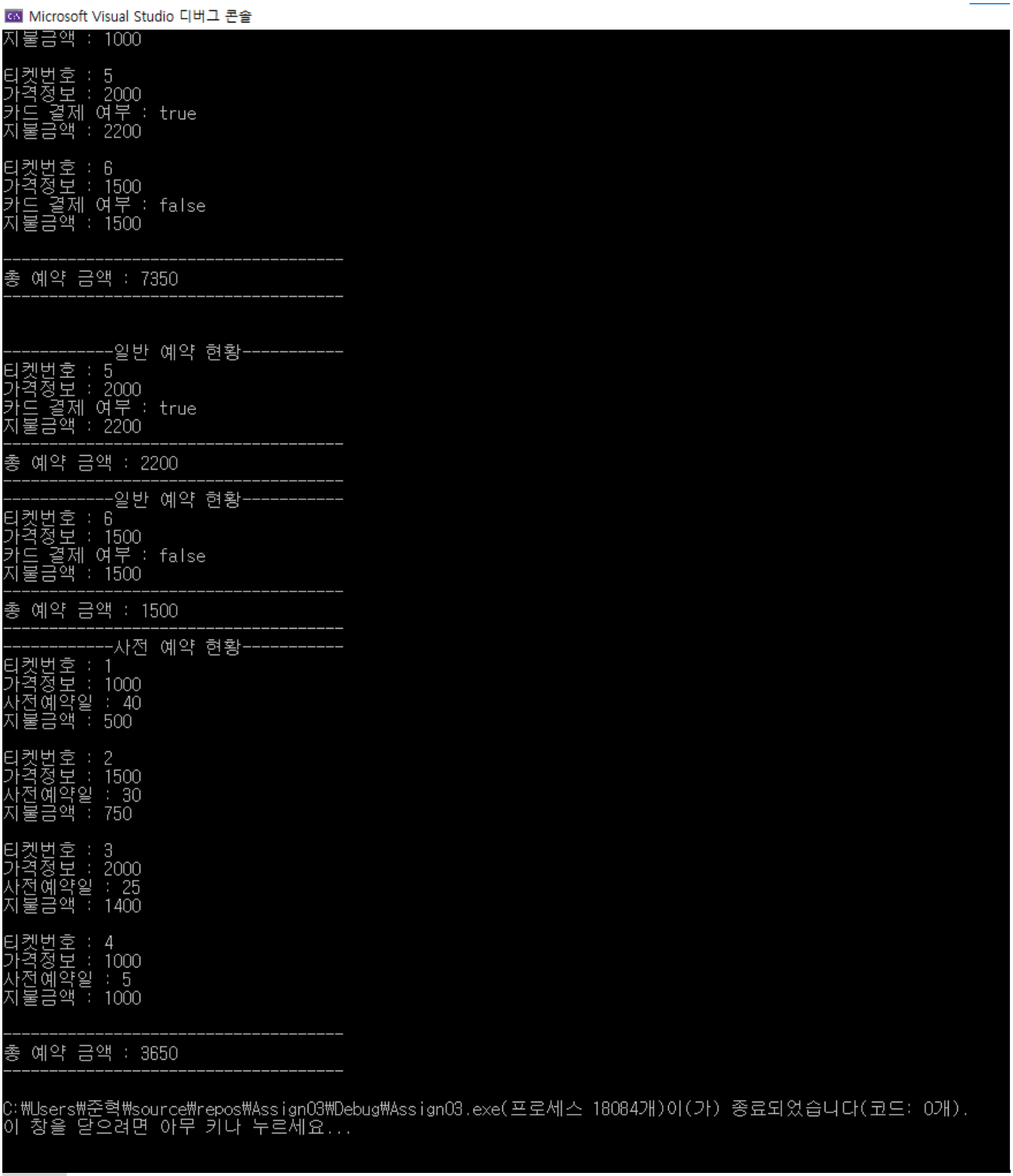
```
#include "AdvancedTicket.h"
#include "GeneralTicket.h"
#include <iostream>
using namespace std;

int Ticket::order = 0;

namespace CJH {
	void printName() {
		cout << "학번 : 201711430 컴퓨터공학부 차준혁" << endl << endl;
	}
}
#ifdef _DEBUG
#ifndef DBG_NEW
#define DBG_NEW new ( _NORMAL_BLOCK , __FILE__ , __LINE__ )
#define new DBG_NEW
#endif // !DBG_NEW
#endif


int main() {
	_CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
	CJH::printName();
	TicketManager manager("아이유콘서트", 10);

	manager.buy(new AdvancedTicket(1000, 40));
	manager.buy(new AdvancedTicket(1500, 30));
	manager.buy(new AdvancedTicket(2000, 25));
	manager.buy(new AdvancedTicket(1000, 5));
	manager.buy(new GeneralTicket(2000, true));
	manager.buy(new GeneralTicket(1500, false));

	cout << manager << endl;

	manager.showGeneralTicket(true);
	manager.showGeneralTicket(false);
	manager.showAdvancedTicket();
}
```

메인 구동 함수입니다.

3 실행결과

Microsoft Visual Studio 디버그 콘솔

```
학번 : 201711430 컴퓨터공학부 차준혁

----------티켓 예약 현황----------

총 예약 매수 : 6
---------------------------------------
티켓번호 : 1
가격정보 : 1000
사전예약일 : 40
지불금액 : 500

티켓번호 : 2
가격정보 : 1500
사전예약일 : 30
지불금액 : 750

티켓번호 : 3
가격정보 : 2000
사전예약일 : 25
지불금액 : 1400

티켓번호 : 4
가격정보 : 1000
사전예약일 : 5
지불금액 : 1000

티켓번호 : 5
가격정보 : 2000
카드 결제 여부 : true
지불금액 : 2200

티켓번호 : 6
가격정보 : 1500
카드 결제 여부 : false
지불금액 : 1500

---------------------------------------
총 예약 금액 : 7350
---------------------------------------


----------일반 예약 현황----------
티켓번호 : 5
가격정보 : 2000
카드 결제 여부 : true
지불금액 : 2200
---------------------------------------
총 예약 금액 : 2200
---------------------------------------
----------일반 예약 현황----------
티켓번호 : 6
가격정보 : 1500
카드 결제 여부 : false
지불금액 : 1500
---------------------------------------
총 예약 금액 : 1500
---------------------------------------
----------사전 예약 현황----------
티켓번호 : 1
가격정보 : 1000
사전예약일 : 40
지불금액 : 500
```



4 느낀점 및 토의사항

티켓 매니저가 티켓을 생성하기만 하고 티켓구매를 취소할 수 있는 기능이 없어서 아쉽다고 생각했습니다. virtual함수를 통해 만들 수 있는 다형성이 흥미롭고 유용하다고 느꼈습니다.