객체지향프로그래밍 정리노트

202004029 김정호, 202004040 노성민

Q1. 상속 관계로 클래스의 간결화 사례 (p.7)

김정호 : 위에 Student, StudentWorker, Researcher, Professor 클래스들을 보면 말하기, 먹기, 걷기, 잠자기는 공통으로 가지고 있는 것을 확인할 수 있어. 그래서 이 공통된 부분을 하나의 Person 클래스로 작성해서 상속을 받아가 간결하게 표현할 수 있는 거야.

노성민 : 그러면 공부하기가 포함된 Student, StudentWorker 클래스들도 위와 같이 상속받고 연구하기가 포함된 Researcher, Professor 클래스들도 위와 같이 상속을 통해 클래스를 간결화 할 수 있겠다.

김정호 : 위에서 말한 방식으로 코드 예시를 들어보자.

```
class Student : public person{
 // person을 상속받는 student 선언
    ...
};
class StudentWorker: public Student{
 // Student를 상속받는 StudentWorker 선언
    ...
};
```

Student 클래스는 person 클래스의 멤버를 물려받고 StudentWorker 클래스는 Student의 멤버를 물려받고 Student가 물려받은 Person의 멤버도 함께 물려받아.

노성민 : 그러면 Researcher 클래스랑 Professor 클래스도 똑같은 구조로 되어있겠네.

김정호 : 맞아.

Q2. 업캐스팅과 다운캐스팅

김정호: 업캐스팅부터 알아볼까?

노성민: 업캐스팅은 파생 클래스 포인터가 기본 클래스 포인터에 치환되는 것을 말해.

김정호 : 코드 예시를 들어보자.

```
#include<iostream>
#include<string>
using namespace std;

class Point {
	int x, y;
public:
	void set(int x, int y) {
		this->x = x;
		this->y = y;
	}
	void showPoint() {
		cout << "(" << x << "," << y << ")" << endl;
	}
};

class ColorPoint :public Point {
	string color;
public:
	void setColor(string color) {
		this->color = color;
	}
	void showColorPoint();
};

void ColorPoint::showColorPoint() {
	cout << color << ":";
	showPoint();
}

int main() {
	ColorPoint cp;
	ColorPoint* pDer = &cp;
	Point* pBase = pDer; // 업캐스팅
	pDer->set(3, 4);
	pBase->showPoint();
	pDer->setColor("Red");
	pDer->showColorPoint();
	//pBase->showColorPoint(); // 컴파일 오류
}
```

먼저 ColorPoint에 객체 cp를 생성하고 cp의 주소를 가리키는 포인터 pDer를 생성했어.
이것을 통해 pDer의 3,4 좌표, Red가 cp.showColor();에도 똑같이 출력될 것이야.
다음은 pDer을 Point 포인터 pBase로 업캐스팅 했어.
pBase->showPoint(); 여기서 포인터 pBase 는 pDer을 가리키기 때문에 똑같이 (3,4)가 출력되는거야. 이게 업캐스팅을 했기 때문에 오류가 나지 않는거지. 이처럼 업캐스팅을 통해 Point 클래스의 멤버 함수에 접근할 수 있어.

노성민: 그럼 이제 다운캐스팅을 알아보자. 다운캐스팅은 기본 클래스의 포인터가 파생 클래스의 포인터에 치환되는 것이야.

김정호: 코드 예시를 들어볼게.

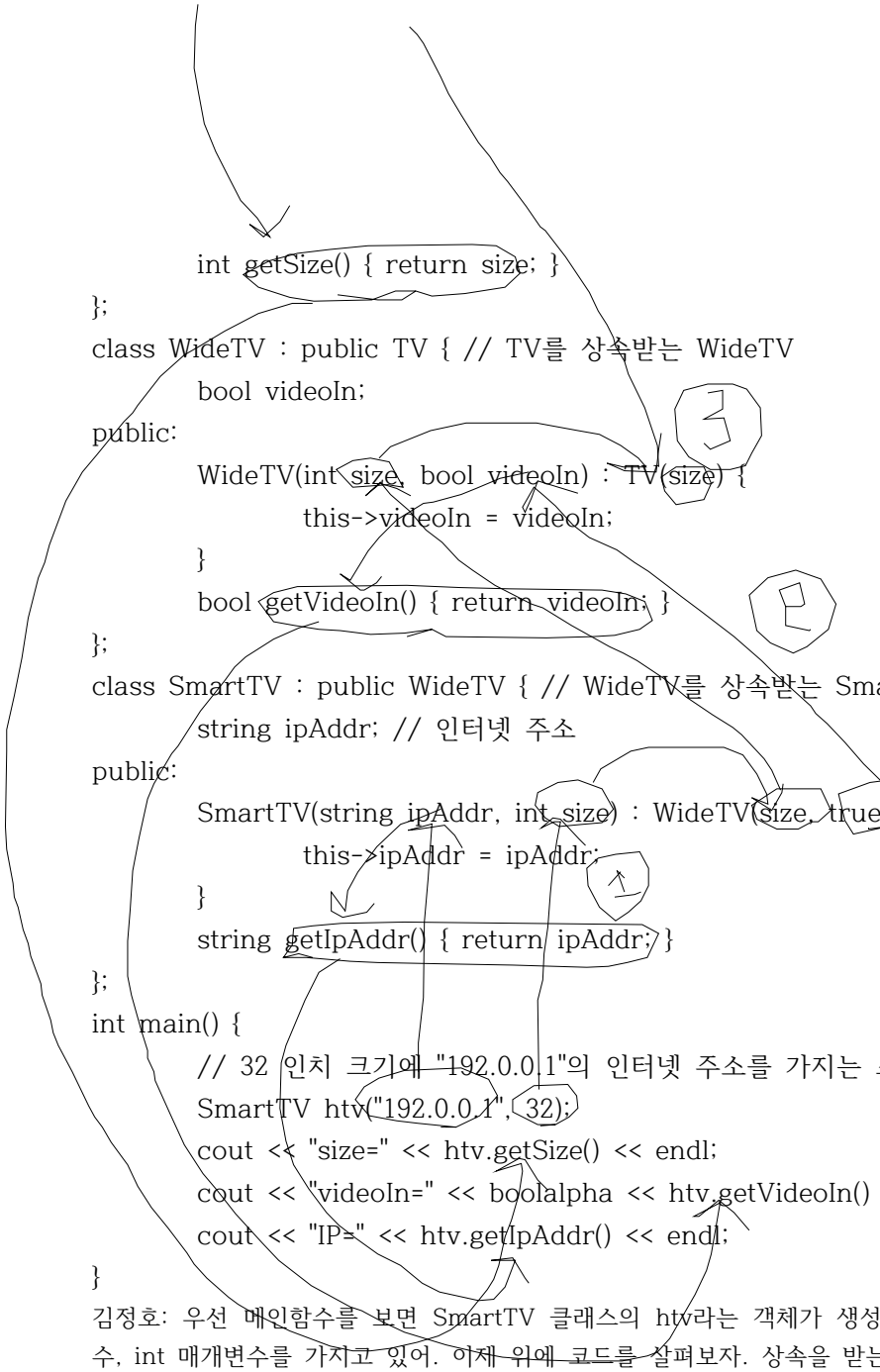
```
int main() {
ColorPoint cp;
ColorPoint *pDer;
Point* pBase = &cp; // 업캐스팅
pBase->set(3,4);
pBase->showPoint();
pDer = (ColorPoint *)pBase; // 다운캐스팅
pDer->setColor("Red"); // 정상 컴파일
pDer->showColorPoint(); // 정상 컴파일
}
```

우선 업캐스팅을 통해 파란색 부분이 가능하게 됐어. 이 부분은 사실 pBase가 실제로 ColorPoint 객체를 가리키고 있는 거야. 여기서 다운 캐스팅을 한 이유가 나와.
pBase가 Point 클래스를 가리키는 포인터인데, 여기에 저장된 객체를 ColorPoint 객체로 다시 가리키기 위해서입니다.

따라서 다운 캐스팅을 수행하여 pBase를 다시 ColorPoint 포인터 pDer로 가리키면, ColorPoint 클래스의 멤버 함수인 setColor("Red")와 showColorPoint()를 호출할 수 있게 되는거지. 이를 통해 ColorPoint 객체에 대한 기능을 사용할 수 있게 돼.

Q3. TV, WideTV, SmartTV 생성자 매개 변수 전달 - 결과 예측해 보기

```
#include <iostream>
#include <string>
using namespace std;
class TV {
        int size; // 스크린 크기
public:
        TV() { size = 20; }
        TV(int size) { this->size = size; }
```



```
        int getSize() { return size; }
};
class WideTV : public TV { // TV를 상속받는 WideTV
        bool videoIn;
public:
        WideTV(int size, bool videoIn) : TV(size) {
                this->videoIn = videoIn;
        }
        bool getVideoIn() { return videoIn; }
};
class SmartTV : public WideTV { // WideTV를 상속받는 SmartTV
        string ipAddr; // 인터넷 주소
public:
        SmartTV(string ipAddr, int size) : WideTV(size, true) {
                this->ipAddr = ipAddr;
        }
        string getIpAddr() { return ipAddr; }
};
int main() {
        // 32 인치 크기에 "192.0.0.1"의 인터넷 주소를 가지는 스마트 TV 객체 생성
        SmartTV htv("192.0.0.1", 32);
        cout << "size=" << htv.getSize() << endl;
        cout << "videoIn=" << boolalpha << htv.getVideoIn() << endl;
        cout << "IP=" << htv.getIpAddr() << endl;
}
```

김정호: 우선 메인함수를 보면 SmartTV 클래스의 htv라는 객체가 생성되고 그 객체는 string 매개변수, int 매개변수를 가지고 있어. 이제 위에 코드를 살펴보자. 상속을 받는 과정에서 위에서 아래로 코드를 보면 이해하기 편할 것 같아.
1,2,3번 그림을 통해 각각 상속받아 코드 오류 없이 정상작동하게 돼.
따라서 결과 값이 size=32
videoIn=true
IP=192.0.0.1 이 나올 것이야.

Q5. protected 상속 사례 - 결과 예측 해보기

Q. 다음에서 컴파일 오류가 발생하는 부분을 찾으시오.

```
#include <iostream>
using namespace std;
class Base {
        int a;
protected:
        void setA(int a) { this->a = a; }
```

```
public:
        void showA() { cout << a; }
};
class Derived : protected Base {
        int b;
protected:
        void setB(int b) { this->b = b; }
public:
        void showB() { cout << b; }
};
int main() {
        Derived x;
        x.a = 5; // ①
        x.setA(10); // ②
        x.showA(); // ③
        x.b = 10; // ④
        x.setB(10); // ⑤
        x.showB(); // ⑥
}
```

김정호: 우선 1, 2, 3번은 protected 멤버 취급받으므로 접근할 수 없어 따라서 모두 오류가 발생하게 돼, 4, 5번도 private, protected 멤버 취급이기 때문에 외부에서 접근할 수 없어.

Q6. 다중 상속

```
#include <iostream>
using namespace std;
class Adder {
protected:
        int add(int a, int b) { return a + b; }
};
class Subtractor {
protected:
        int minus(int a, int b) { return a - b; }
};

class Calculator : public Adder, public Subtractor {
public:
        int calc(char op, int a, int b);
};

int Calculator::calc(char op, int a, int b) {
        int res = 0;
        switch (op) {
        case '+': res = add(a, b); break;
```

```
        case '-': res = minus(a, b); break;
        }
        return res;
}
int main() {
        Calculator handCalculator;
        cout << "2 + 4 = "
                << handCalculator.calc('+', 2, 4) << endl;
        cout << "100 - 8 = "
                << handCalculator.calc('-', 100, 8) << endl;
}
```

노성민: Calculator 클래스에서 public으로 Adder, Subtractor를 상속받았기 때문에 Calculator 클래스에서 add, minus 함수를 사용할 수 있는거야. 다중 상속은 그냥 상속을 여러개 받을 수 있구나 정도로 생각하면 될 것 같아.

김정호: 하지만 p.36을 보면 기본 클래스의 맴버가 중복으로 상속받게 되면 다중 상속 과정 중에서 어떤 변수를 지정하고 싶은지에 대한 오류가 발생할 수 있어. 이를 해결하기 위해 가상 상속을 알아보자.
우선 가상 상속은 기본 클래스 앞에 virtual로 선언하고 파생 클래스의 객체가 생성될 때 기본 클래스의 맴버는 오직 한 번만 생성해. 이를 통해 중복을 방지할 수 있어.
그래서 p.38을 보면 In, Out 클래스에서 virtual을 이용하여 mode가 하나만 생성되게 해서 p.36에서 나타났던 문제점을 보완할 수 있는거야.