Inheritance

다음의 각 개념을 설명하는 예제 소스 코드를 각각 작성하고 해당 소스 코드를 바탕으로 그 개념을 설명하시오.

(단 교재의 코드를 사용하지 않고 새로운 소스 코드를 개발해야 함.)

코드1: Constructor delegation, Constructor inheritance

(1) Constructor delegation

```
class Animal {
    string name;
    int age;
public:
    Animal(const string& _name, const int _age)
        :name(_name), age(_age) {}
    Animal(const string& _name)
        :name(_name), age(0) {}
};
```

↓

```
class Animal {
    string name;
    int age;
public:
    Animal(const string& _name, const int _age)
        :name(_name), age(_age) {}
    Animal(const string& _name)
        : Animal(_name, 0) {}
};
```

한 클래스의 Constructor에는 대게 객체를 초기화하는 비슷한 코드가 중복된다. 위쪽코드의 Constructor의 경우 class의 멤버변수가 늘어날수록 Constructor의 개수와 초기화할 멤버변수의 개수는 증가할 것이고, 그로 인하여 반복되는 코드의 수가 늘어날 것이다. 그렇기에 하나의 Constructor를 정의하고, 다른 Constructor에서 정의된 Constructor을 아래코드처럼 호출한다면 중복된 코드는 간소화되고, 효율 또한 상승한다. 최근의 C++11부터는 이러한 기능을 제공한다.

(2) Constructor inheritance

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

class Animal {
    string name;
    int age;
public:
    Animal(const string& _name, const int _age)
        :name(_name), age(_age) {}
    Animal(const string& _name)
        : Animal(_name, 0) {}
};

class Dog : public Animal {
    string cryingSound;
public:
    Dog(const string& _name, const int _age, const string& _cryingSound)
        :Animal(_name, _age), cryingSound(_cryingSound) {}
    using Animal::Animal;
};

int main() {
    Dog object1("허니", 3, "왈왈");
    Dog object2("여름", 2);
    Dog object3("도토");

    return 0;
}
```

상속받는 클래스의 경우 상위클래스의 Constructor는 상속받지 않는다. 그렇기에 빨간색으로 표시해둔 Animal::Animal 코드가 없다면, object2와 object3객체는 Constructor의 parameter의 개수가 일치하지 않기 때문에, 오류를 발생한다. 그래서 상속받는 하위클래스에 상위클래스의 Constructor을 표시하는 코드인 Animal::Animal를 명시해줌으로써 하위클래스에서는 상위클래스의 Constructor까지 상속받는 형태가 되는 것이다.

물론 하위클래스의 Constructor을 간편하게 정의하는 장점을 갖는다.
하지만 object2와 object3와 같은 경우는 상위클래스의 Constructor을 호출하는 형태이기에, 하위클래스의 멤버변수를 초기화 할 수 없다. cryingSound의 경우는 string의 클래스의 default Constructor이 호출되어서 괜찮지만 다른 경우는 주의를 기울여 사용해야 한다.

코드2: Final virtual function, Final class, Virtual function override

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

class Animal {
    string name;
    int age;
public:
    Animal(const string& _name, const int _age)
        :name(_name), age(_age) {}

    virtual void shot() = 0;
    virtual void print() {
        cout << "name : " << name << endl;
        cout << "age : " << age << endl;
    }
    void classInformation() {
        cout << "Animal";
    }
};

class Dog : public Animal {
    string cryingSound;
public:
    Dog(const string& _name, const int _age, const string& _cryingSound)
        :Animal(_name, _age), cryingSound(_cryingSound) {}

    void shot() override { cout << cryingSound << endl; }
    void print() final {
        Animal::print();
        cout << "crying sound : " << cryingSound << endl;
    }
    void classInformation() override { //error
        Animal::classInformation();
        cout << "->" << "Dog";
    }
};

class Buldog final : public Dog { //불독 클래스 //error
    int length;
public:
    Buldog(const string& _name, const int _age,
        const string& _cryingSound, int _length)
        :Dog(_name, _age, _cryingSound), length(_length) {}

    void print() override { //error
        Dog::print();
        cout << "length : " << length << endl;
    }
};

class FrenchBuldog : public Buldog { //error

};
```

(1) Final virtual function

print() 함수를 살펴보면 virtual로 선언되어 하위클래스에서 overriding하고 있는 것을 볼 수 있다.

Dog class의 28번 라인에서 print()함수를 final 선언하여 현재 클래스를 상속받는 하위 클래스에서는 더 이상 print()함수를 overriding 할 수 없게 선언하였고, 그 예로 Buldog class의 45번 라인에서 print()를 overriding 하였지만, 빨간 밑줄로 error를 표시하고 있음을 확인할 수 있다.

(2) Final class

38번 라인에서 Buldog class를 final로 선언하였다.

Final의 선언이 뜻하는 것은 (1)Final virtual function에서 볼 수 있듯이, 하위 클래스에서는 더 이상 final선언된 것을 상속할 수 없다는 것을 의미하며, 현재가 마지막이라는 뜻을 내포하고 있다.

그렇기에 51번 라인에서 FrenchBuldog class를 정의할 때, Buldog class를 상속하려 했지만, final 선언으로 상속하지 못하고 error가 발생함을 확인할 수 있다.

(3) Virtual function override

Override 선언은 virtual function을 명시적으로 override 한다는 선언이다.

11번 라인을 보면 Animal 클래스안에 shot() 함수가 virtual로 선언되어 있고, 27번 라인을 보면 Animal 클래스를 public로 상속받는 Dog 클래스 안에 shot() override 선언이 정상적으로 구현됨을 확인할 수 있다.

그런데 override 선언의 경우 virtual function이 아닌 경우에는 error가 발생한다.

Animal 클래스에서 16번 라인에 선언된 classInformation() 함수를 Dog클래스의 32번 라인에 선언된 classInformation() 함수로 overriding 하려 시도하였지만, 이 함수의 경우에는 virtual 함수가 아니기에, error가 발생함을 확인할 수 있다.