

C++ Programming

10th Study: Object-Oriented Programming (6/8)

- Virtual function / virtual inheritance
- Polymorphism

C++ Korea 옥찬호 (utilForever@gmail.com)



Virtual function

A function or method whose behavior can be overridden within an inheriting class by a function with the same signature

Virtual function

- 가상함수(Virtual function)?

- 부모 클래스의 멤버 함수 앞에 “virtual” 이라는 키워드를 붙이고, 자식 클래스에서 그 함수를 오버라이딩(Overriding)할 때, 포인터 또는 참조자 타입의 업 캐스팅(Up-casting)하여 객체를 사용하는 경우 자식 클래스의 함수를 호출 해 준다.
- 업 캐스팅시 “virtual” 키워드를 붙여주지 않으면 부모 클래스의 함수가 호출 되는데, 그 이유는 함수의 호출 위치를 컴파일 타임(‘early time’)에 결정하기 때문이다.
- “virtual” 키워드를 붙여 준다면, 해당 함수의 호출 위치를 코드가 컴파일 되는 시점에 결정하는 것이 아니라 런타임(‘late time’)에 결정되도록 하기 때문에 하위 클래스의 오버라이딩 된 함수를 호출해 주게 된다.

Virtual function: Example

```
class Person {
public:
    Person()
    {
        cout << "Call Person()" << endl;
    }
    ~Person()
    {
        cout << "Call ~Person()" << endl;
    }

    virtual void doWork()
    {
        cout << "Working ..." << endl;
    }
};
```

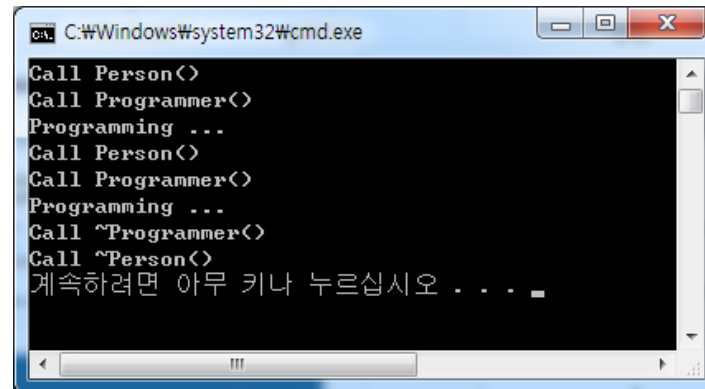
```
class Programmer : public Person {
public:
    Programmer() : Person()
    {
        cout << "Call Programmer()" << endl;
    }
    ~Programmer()
    {
        cout << "Call ~Programmer()" << endl;
    }

    void doWork()
    {
        cout << "Programming ..." << endl;
    }
};

int main() {
    Programmer pro;
    Person &p = pro; // 참조자 형식
    p.doWork();

    Person *per = new Programmer; // 포인터 형식
    per->doWork();
}
```

Virtual function: Example



```
C:\Windows\system32\cmd.exe
Call Person()
Call Programmer()
Programming ...
Call Person()
Call Programmer()
Programming ...
Call ~Programmer()
Call ~Person()
계속하려면 아무 키나 누르십시오 . . .
```

Virtual function

- 소멸자에서 Virtual function
 - 소멸자 역시 함수 이므로 오버라이딩 된 소멸자가 호출되게 하려면 반드시 상위 클래스에 “virtual” 키워드를 붙여줘야 한다.
 - 소멸자의 경우 제대로 처리 하지 않으면 메모리 누수(Memory leak)이 발생할 수 있으니 주의 깊게 다뤄야 한다.

Virtual function: Example

```
class Person {
public:
    Person()
    {
        cout << "Call Person()" << endl;
    }
    ~Person()
    {
        cout << "Call ~Person()" << endl;
    }

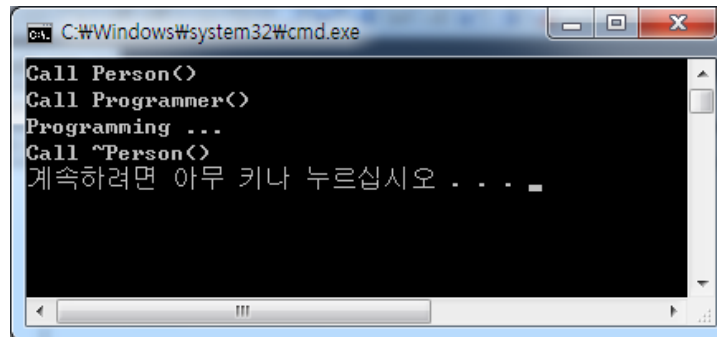
    virtual void doWork()
    {
        cout << "Working ..." << endl;
    }
};
```

```
class Programmer : public Person {
public:
    Programmer() : Person()
    {
        cout << "Call Programmer()" << endl;
    }
    ~Programmer()
    {
        cout << "Call ~Programmer()" << endl;
    }

    void doWork()
    {
        cout << "Programming ..." << endl;
    }
};

int main() {
    Person *per = new Programmer; // 포인터 형식
    per->doWork();
    delete per;
}
```

Virtual function: Example



```
C:\Windows\system32\cmd.exe
Call Person()
Call Programmer()
Programming ...
Call ~Person()
계속하려면 아무 키나 누르십시오 . . .
```


Virtual function: Example

```
class Person {
public:
    Person()
    {
        cout << "Call Person()" << endl;
    }
    virtual ~Person()
    {
        cout << "Call ~Person()" << endl;
    }

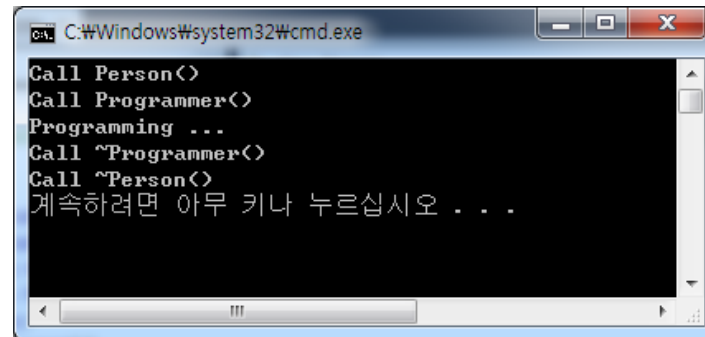
    virtual void doWork()
    {
        cout << "Working ..." << endl;
    }
};
```

```
class Programmer : public Person {
public:
    Programmer() : Person()
    {
        cout << "Call Programmer()" << endl;
    }
    ~Programmer()
    {
        cout << "Call ~Programmer()" << endl;
    }

    void doWork()
    {
        cout << "Programming ..." << endl;
    }
};

int main() {
    Person *per = new Programmer; // 포인터 형식
    per->doWork();
    delete per;
}
```

Virtual function: Example



```
C:\Windows\system32\cmd.exe
Call Person<>
Call Programmer<>
Programing ...
Call ~Programmer<>
Call ~Person<>
계속하려면 아무 키나 누르십시오 . . .
```

Virtual function

- 순수 가상 함수(Pure virtual function)?
 - 부모 클래스에서 상속받을 자식 클래스의 공통 기능(함수)의 정의만 선언해 놓고, 실제 구현은 자식 클래스에게 위임하기 위한 함수이다.
- 다중 상속 문법의 기본 형태
 - **virtual** return_type functionName(parameter) = 0;
 - 예
 - virtual void doWork() = 0;

Virtual function

- 추상 클래스 (abstract class)
 - 순수 가상 함수를 하나 이상 포함하는 클래스를 추상 클래스라고 한다.
 - 추상클래스의 대표적인 특징은 객체화 할 수 없다는 것이다.
 - 만약 추상 클래스를 객체화 하려고 시도한다면 “추상 클래스를 인스턴스화할 수 없습니다.” 라는 컴파일 오류가 발생한다.
 - 상위 클래스에 대한 객체 생성 요구가 발생하지 않는 상황이라면 상위 클래스에 순수 가상 함수를 선언하고 추상 클래스로 만들면 된다.

Virtual function: Example

```
class Person {  
public:  
    Person()  
    {  
        cout << "Call Person()" << endl;  
    }  
    virtual ~Person()  
    {  
        cout << "Call ~Person()" << endl;  
    }  
  
    virtual void doWork() = 0;  
};
```

```
class Programmer : public Person {  
public:  
    Programmer() : Person()  
    {  
        cout << "Call Programmer()" << endl;  
    }  
    ~Programmer()  
    {  
        cout << "Call ~Programmer()" << endl;  
    }  
  
    void doWork()  
    {  
        cout << "Programming ..." << endl;  
    }  
};  
  
int main() {  
    Person per;  
}
```

Virtual function: Example

error C2259: 'Person' : 추상 클래스를 인스턴스화할 수 없습니다.
다음 멤버가 원인입니다.
'void Person::doWork(void)' : abstract입니다.
'Person::doWork' 선언을 참조하십시오.

Virtual function: Example

```
class Person {
public:
    Person()
    {
        cout << "Call Person()" << endl;
    }
    virtual ~Person()
    {
        cout << "Call ~Person()" << endl;
    }

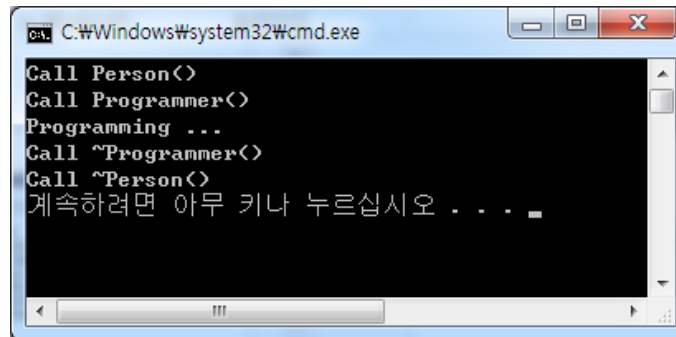
    virtual void doWork() = 0;
};
```

```
class Programmer : public Person {
public:
    Programmer() : Person()
    {
        cout << "Call Programmer()" << endl;
    }
    ~Programmer()
    {
        cout << "Call ~Programmer()" << endl;
    }

    void doWork()
    {
        cout << "Programming ..." << endl;
    }
};

int main() {
    Person *per = new Programmer;
    per->doWork();
    delete per;
}
```

Virtual function: Example

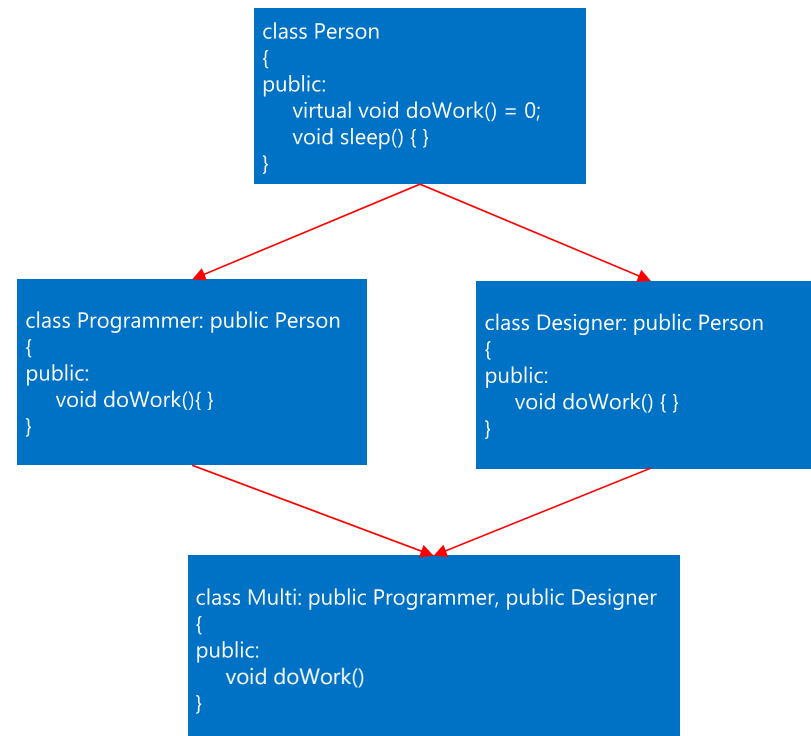


A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. The text shows a sequence of C++ virtual function calls: "Call Person()", "Call Programmer()", "Programming ...", "Call ~Programmer()", and "Call ~Person()". Below these, there is a line of Korean text: "계속하려면 아무 키나 누르십시오 . . .". The window includes standard Windows window controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

```
C:\Windows\system32\cmd.exe
Call Person()
Call Programmer()
Programming ...
Call ~Programmer()
Call ~Person()
계속하려면 아무 키나 누르십시오 . . .
```


Virtual function

- 가상 상속(Virtual inheritance)
- 다중 상속을 구현하다 보면 다음과 같은 애매한 문제가 발생한다.
 - Multi 클래스의 sleep() 함수는 Programmer에서 상속받은 Person의 sleep()인가 Designer에서 상속받은 sleep()인가를 결정하는 문제이다.
 - 실제로 이를 컴파일 해보면 에러가 발생한다.
 - 또한 sleep()을 사용하지 않더라도 Programmer클래스가 상속하는 Person과 Designer 클래스가 상속하는 Person이 둘 다 존재 하여 Person 생성자와 소멸자가 두 번 호출된다.
- 이를 해결 하기 위한 방법이 가상 상속이다.
- 상속 문법에서 접근 제한자 앞에 “virtual”이라는 키워드를 적으면 된다.



Virtual function: Example

```
class Person {
public:
    Person() {
        cout << "Call Person()" << endl;
    }
    virtual ~Person() {
        cout << "Call ~Person()" << endl;
    }
    void sleep() { cout << "Sleep..." << endl; }
};

class Programmer : public Person {
public:
    Programmer() : Person() {
        cout << "Call Programmer()" << endl;
    }
    virtual ~Programmer() {
        cout << "Call ~Programmer()" << endl;
    }
};
```

```
class Designer : public Person {
public:
    Designer() : Person() {
        cout << "Call Designer()" << endl;
    }
    virtual ~Designer(){
        cout << "Call ~Designer()" << endl;
    }
};

class Multi : public Programmer, public Designer {
public:
    Multi() : Programmer(), Designer() {
        cout << "Call Multi()" << endl;
    }
    ~Multi() {
        cout << "Call ~Multi()" << endl;
    }
};

int main() {
    Multi m1;
    m1.sleep();
}
```

Virtual function: Example

error C2385: 'sleep' 액세스가 모호합니다.
기본 'Person'의 'sleep'일 수 있습니다.
또는 기본 'Person'의 'sleep'일 수 있습니다.

Virtual function: Example

```
class Person {
public:
    Person() {
        cout << "Call Person()" << endl;
    }
    virtual ~Person() {
        cout << "Call ~Person()" << endl;
    }
    void sleep() { cout << "Sleep..." << endl; }
};

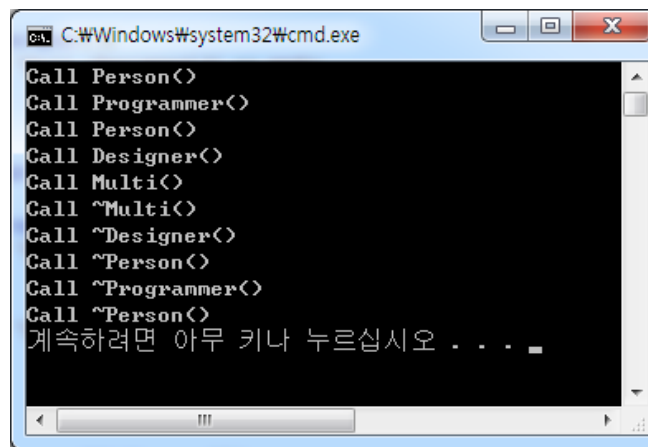
class Programmer : public Person {
public:
    Programmer() : Person() {
        cout << "Call Programmer()" << endl;
    }
    virtual ~Programmer() {
        cout << "Call ~Programmer()" << endl;
    }
};
```

```
class Designer : public Person {
public:
    Designer() : Person() {
        cout << "Call Designer()" << endl;
    }
    virtual ~Designer(){
        cout << "Call ~Designer()" << endl;
    }
};

class Multi : public Programmer, public Designer {
public:
    Multi() : Programmer(), Designer() {
        cout << "Call Multi()" << endl;
    }
    ~Multi() {
        cout << "Call ~Multi()" << endl;
    }
};

int main() {
    Multi m1;
}
```

Virtual function: Example



```
C:\Windows\system32\cmd.exe
Call Person()
Call Programmer()
Call Person()
Call Designer()
Call Multi()
Call ~Multi()
Call ~Designer()
Call ~Person()
Call ~Programmer()
Call ~Person()
계속하려면 아무 키나 누르십시오 . . . .
```

Virtual function: Example

```
class Person {
public:
    Person() {
        cout << "Call Person()" << endl;
    }
    virtual ~Person() {
        cout << "Call ~Person()" << endl;
    }
    void sleep() { cout << "Sleep..." << endl; }
};

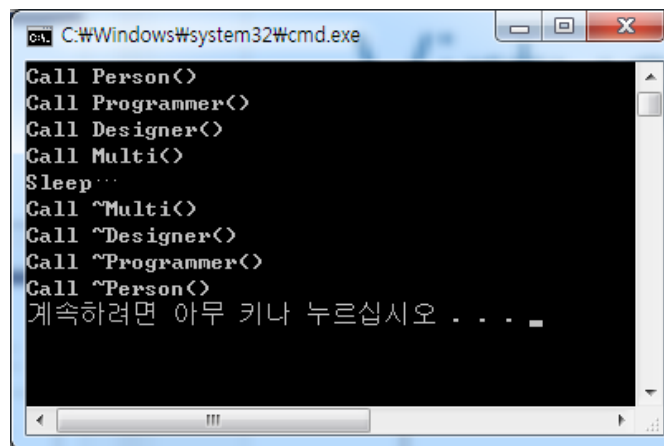
class Programmer : public virtual Person {
public:
    Programmer() : Person() {
        cout << "Call Programmer()" << endl;
    }
    virtual ~Programmer() {
        cout << "Call ~Programmer()" << endl;
    }
};
```

```
class Designer : public virtual Person {
public:
    Designer() : Person() {
        cout << "Call Designer()" << endl;
    }
    virtual ~Designer(){
        cout << "Call ~Designer()" << endl;
    }
};

class Multi : public Programmer, public Designer {
public:
    Multi() : Programmer(), Designer() {
        cout << "Call Multi()" << endl;
    }
    ~Multi() {
        cout << "Call ~Multi()" << endl;
    }
};

int main() {
    Multi m1;
    m1.sleep();
}
```

Virtual function: Example



```
cmd: C:\Windows\system32\cmd.exe
Call Person()
Call Programmer()
Call Designer()
Call Multi()
Sleep...
Call ~Multi()
Call ~Designer()
Call ~Programmer()
Call ~Person()
계속하려면 아무 키나 누르십시오 . . .
```

Polymorphism

A type whose operations can also be applied to values of some other type, or types.

Polymorphism

- 다형성 (Polymorphism)
 - 하나의 모습으로 다양한 형태를 가질 수 있는 성질을 다형성이라 한다.
 - 클래스를 상속받아 사용하는 가장 큰 이유는 다형성이다.
 - 예제
 - `Person *per = new Programmer;`
 - `Person *per = new Designer;`
 - 와 같이 Person은 Programmer또는 Designer라는 다양한 형태를 가질 수 있다.

Polymorphism

- 다형성을 사용하는 이유

```
class Programmer
{
    pub class CProgrammer
    v {
    }
    pub class CPPProgrammer
    v {
    }
    publ class JAVAProgrammer
    vc {
    }
    public:
        void doWork();
}
```

```
int main
{
    Project(Programmer *p);
    Project(Cprogrammer *cp);
    Project(CPPProgrammer *cpp);
    Project(JAVAProgrammer *java);
    ...
}
```



Python Programmer 추가

```
class Programmer
{
    pub class CProgrammer
    v {
    }
    pub class CPPProgrammer
    v {
    }
    publ class JAVAProgrammer
    vc {
    }
    pub class PythonProgrammer
    v {
    }
    public:
        void doWork();
}
```

```
int main
{
    Project(Programmer *p);
    Project(Cprogrammer *cp);
    Project(CPPProgrammer *cpp);
    Project(JAVAProgrammer *java);
    Project(PythonProgrammer *py);
    ...
}
```

Polymorphism

• 다형성을 사용하는 이유

```
class Programmer
{
public:
    void doWork();
}
```

```
class Cprogrammer:
Programmer
{
public:
    void doWork();
}
```

```
class CPPProgrammer:
Programmer
{
public:
    void doWork();
}
```

```
class JAVAProgrammer:
Programmer
{
public:
    void doWork();
}
```

```
int main
{
    Project(Programmer *p);
    ...
}
```



Python Programmer 추가

```
class Programmer
{
public:
    void doWork();
}
```

```
class Cprogrammer:
Programmer
{
public:
    void doWork();
}
```

```
class CPPProgrammer:
Programmer
{
public:
    void doWork();
}
```

```
class JAVAProgrammer:
Programmer
{
public:
    void doWork();
}
```

```
class PythonProgrammer
{
}
```

```
int main
{
    Project(Programmer *p);
    ...
}
```