

C++ Programming

9th Study: Object-Oriented Programming (5/8)

- Inheritance
- Overriding
- Up / Down casting

C++ Korea 옥찬호 (utilForever@gmail.com)



Inheritance

When an object or class is based on another object or class, using the same implementation specifying implementation to maintain the same behavior.

Inheritance

- 상속이란?

- 클래스를 구현할 때, 다른 클래스나 객체에 구현되어 있는 동작을 유지하기 위해 그것을 기반으로 만드는 것으로 일반적으로는 기반이 되는 것을 부모(Parent) 클래스, 기반을 활용하여 만드는 클래스를 자식(Child)클래스라고 함.

- 상속을 하는 이유는

- 기반 클래스의 코드를 재활용 하기 위해서
- 다형성(Polymorphism)을 활용하기 위해서

Inheritance

- 상속의 종류

- 단일 상속(Single Inheritance)

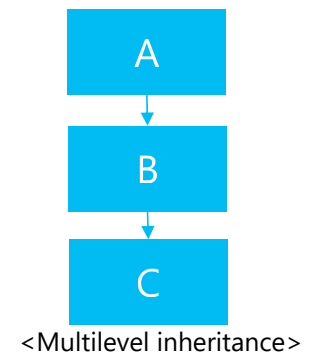
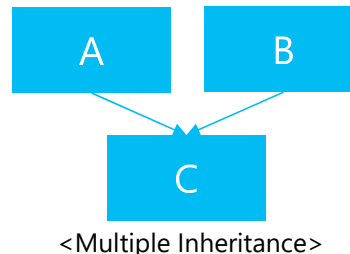
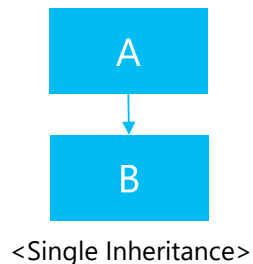
- 하위 클래스에서 한 개의 상위 클래스로부터 상속 받는 것

- 다중 상속(Multiple Inheritance)

- 하위 클래스에서 두 개 이상의 상위 클래스로부터 상속 받는 것

- 다 단계 상속(Multilevel Inheritance)

- 하위 클래스에서 다른 하위 클래스로부터 상속 받는 것



Inheritance

- 상속 문법의 기본 형태
 - `class` ClassName : [public/protected/private] ParentClassName
 { };
- 예
 - `class` Programmer : `public` Person
 { };
 - Person이라는 클래스를 부모 클래스로 하는 Programmer라는 자식 클래스를 선언

Inheritance

• 상속의 이해

- Programmer와 Designer는 Person를 상속받아 만든 클래스이므로, Programmer와 Designer 클래스가 가지고 있는 고유한 멤버 변수나 함수 뿐만 아니라 Person 클래스의 멤버 변수와 함수까지 가질 수 있게 된다.

```
class Person
{
private:
    char* name;
    int _age;
    int _height;
    int _weight;
public:
    Person() { ... .. }
    ~ Person() { ... .. }
}
```

```
class Programmer: public Person
{
private:
    int numOfLanguage;
public:
    Programmer() { ... .. }
    ~ Programmer() { ... .. }
    void doCoding() { ... .. }
}
```

```
class Designer: public Person
{
private:
    int numOfTools;
public:
    Designer() { ... .. }
    ~ Designer() { ... .. }
    void doDesign() { ... .. }
}
```

Inheritance

• 상속의 이해

- Programmer p; 로 Programmer의 객체를 생성한다면 그림과 같이 p는 Person과 Programmer의 모든 내용을 포함하는 객체로 생성된다.

```
class Person
{
Private:
    char* name;
    int _age;
    int _height;
    int _weight;
Public:
    Person() { ... .. }
    ~ Person() { ... .. }
}
```

```
class Programmer: public Person
{
Private:
    int numOfLanguage;
Public:
    Programmer() { ... .. }
    ~ Programmer() { ... .. }
    void doCoding() { ... .. }
}
```

```
Programmer{
    char* name;
    int _age;
    int _height;
    int _weight;
    int numOfLanguage;
    Person() { ..... }
    ~Person() { ..... }
    Programmer() { ..... }
    ~Prorammer() { ..... }
    void doCoding() { ... .. }
}
```

Inheritance

- 상속관계에서의 생성자와 소멸자
 - 상속 관계에서 하위 클래스는 상위클래스의 멤버 변수들을 모두 갖기 때문에, 하위 클래스의 객체를 생성할 때 하위 클래스와 상위 클래스의 생성자가 모두 호출 되어야 하며, 객체가 파괴될 때 상위 클래스와 하위 클래스의 소멸자가 모두 호출 되어야 한다.
 - 생성자와 소멸자는 다음과 같은 순서로 호출 된다.
 - 생성자: 상위 클래스의 생성자 호출 -> 하위 클래스의 생성자 호출
 - 소멸자: 하위 클래스의 소멸자 호출 -> 상위 클래스의 소멸자 호출

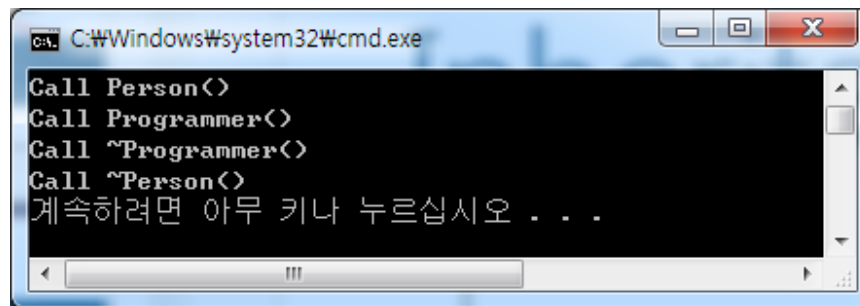
Inheritance: Example

```
class Person {
private:
    double height;
    double weight;
public:
    Person(double _height, double _weight)
    {
        height = _height;
        weight = _weight;
        cout << "Call Person()" << endl;
    }
    ~Person()
    {
        cout << "Call ~Person()" << endl;
    }
};
```

```
class Programmer : public Person {
private:
    int numOfLanguage;
public:
    Programmer(double _height, double _weight, int lang)
        : Person(_height, _weight)
    {
        numOfLanguage = lang;
        cout << "Call Programmer()" << endl;
    }
    ~Programmer()
    {
        cout << "Call ~Programmer()" << endl;
    }
};

int main() {
    Programmer p1(183.4, 78.5, 3);
}
```

Inheritance: Example



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

Inheritance

- 상속의 접근 범위
 - 클래스의 멤버 변수나 함수에 접근 제한자를 두는 것처럼, 상속 관계 안에도 접근 제한자를 통해, 하위 클래스에서 상위 클래스의 멤버 변수의 접근 범위를 정할 수 있다.
 - 상속 관계 안에서의 접근 제한자에 따라 3가지의 상속 관계로 분류할 수 있다.
 - public 상속
 - protected 상속
 - private 상속

Inheritance

- public 상속
 - public으로 상속을 받으면 하위 클래스에서는 상위 클래스의 접근 제한자의 특성을 그대로 물려 받는다.
 - 예제에서처럼 Person의 private 영역의 data는 보이지 않고 나머지 모든 Data의 접근 제한자는 그대로 상속된다.

```
class Person
{
private:
    int weight;
protected:
    int age;
public:
    char* name;
}
```



```
class Programmer: public Person
{
// Private 영역은 보이지 않음
protected:
    int age;
public:
    char* name;
}
```



```
class OOPProgrammer: public Programmer
{
// Private 영역은 보이지 않음
protected:
    int age;
public:
    char* name;
}
```

Inheritance

- Protected 상속
 - protected로 상속을 받으면 하위 클래스에서는 상위 클래스의 public 접근 제한자도 protected로 바뀌어 물려 받는다.
 - 예제에서처럼 Person의 private 영역의 data는 보이지 않고 public은 protected로 변경되어 상속된다.

```
class Person
{
private:
    int weight;
protected:
    int age;
public:
    char* name;
}
```

↓

```
class Programmer: protected Person
{
// Private 영역은 보이지 않음
protected:
    int age;
protected:
    char* name;
}
```

↓

```
class OOPProgrammer: protected Programmer
{
// Private 영역은 보이지 않음
protected:
    int age;
protected:
    char* name;
}
```

Inheritance

- Private 상속
 - Private으로 상속을 받으면 하위 클래스에서는 상위 클래스의 모든 접근 제한자를 private으로 바꿔 물려 받는다.
 - 예제에서처럼 Person의 private 영역의 data는 보이지 않고 public과 protected는 private으로 변경되어 상속된다.

```
class Person
{
private:
    int weight;
protected:
    int age;
public:
    char* name;
}
```

↓

```
class Programmer: private Person
{
// Private 영역은 보이지 않음
private:
    int age;
private:
    char* name;
}
```

↓

```
class OOPProgrammer: private Programmer
{
// Private 영역은 보이지 않음

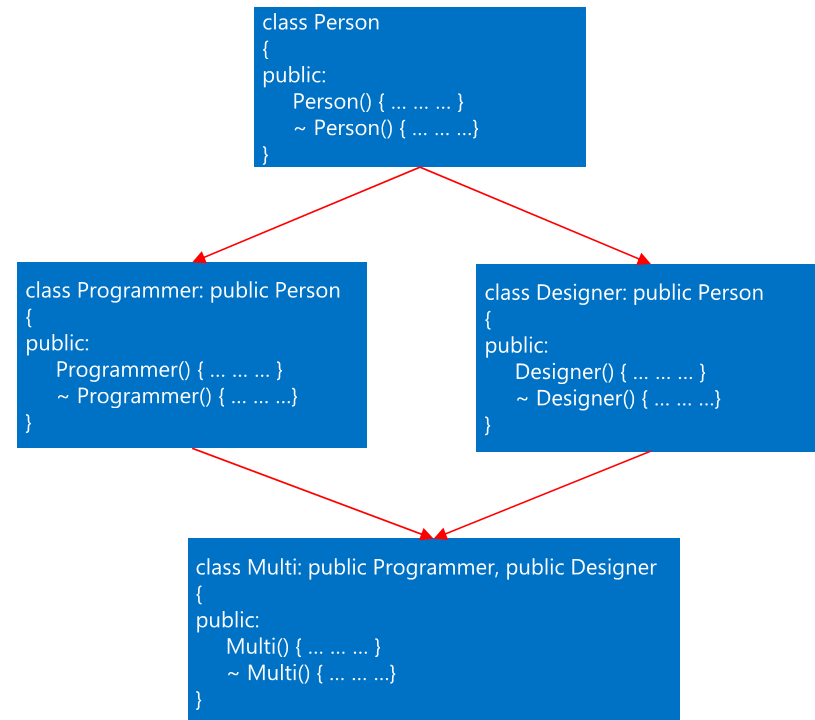
}
```

Inheritance

- 다중 상속 (Multiple Inheritance)
 - C++에서는 여러 개의 부모로부터 상속을 받을 수 있다.
- 다중 상속 문법의 기본 형태
 - `class` ClassName : [public/protected/private] ParentClassName, ...
 { };
 - 예
 - `class` Multiplayer : `public` Programmer, `public` Designer { };
 - Programmer와 Designer라는 클래스를 부모 클래스로 하는 Multiplayer라는 자식 클래스를 선언

Inheritance

- 다중 상속 (Multiple Inheritance)
 - Multi 클래스는 Programmer와 Designer의 멤버 변수 및 함수를 모두 갖는다.
 - 생성자와 소멸자가 불리는 순서는 단일 상속과 같은 순서를 갖는다.



Inheritance: Example

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

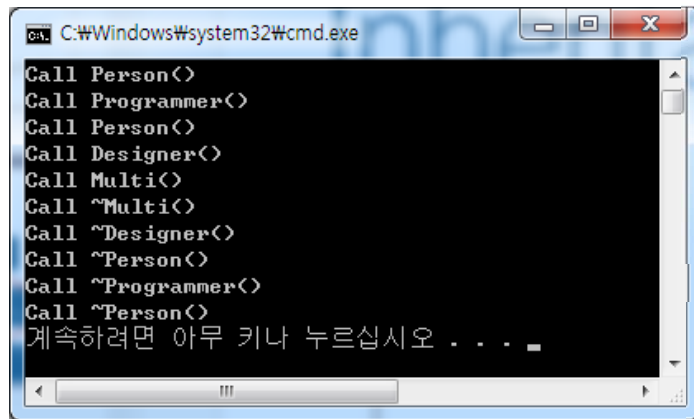
class Programmer : public Person {
public:
    Programmer() : Person()
    {
        cout << "Call Programmer()" << endl;
    }
    ~ Programmer()
    {
        cout << "Call ~Programmer()" << endl;
    }
};
```

```
class Designer : public Person {
public:
    Designer() : Person() {
        cout << "Call Designer()" << endl;
    }
    ~ 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 p1;
}
```

Inheritance: 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 . . .
```

???

- Person()의 생성자가 2번이나 호출
- Person()의 소멸자도 2번이나 호출
- ➔ 이에 대한 해답은 다음 강의에서 virtual keyword 대한 내용을 다룰 때 자세하...

Overriding

A language feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its superclasses or parent classes.

Overriding

- Overriding 이란?
 - 사전적 의미는 무효로 하다. 뒤엎다 라는 뜻.
 - 상속관계의 상위클래스의 함수를 무효화 한다 또는 재정의 한다는 의미이다.
- 방법은?
 - 상위 클래스의 함수 이름과 리턴 타입, 파라미터의 형태가 같은 함수를 동일하게 하위 클래스에서 정의 하면 된다.

Overriding: Example

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

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

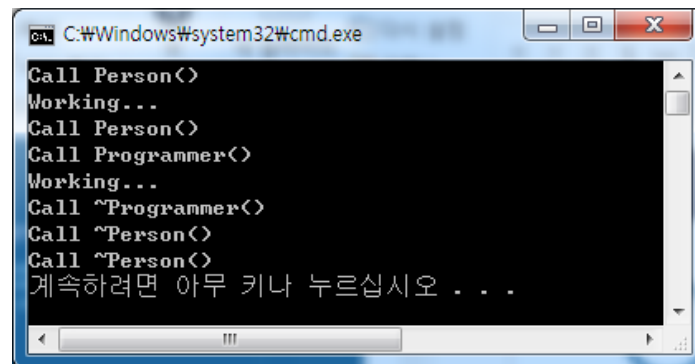
```
class Programmer : public Person {
public:
    Programmer() : Person()
    {
        cout << "Call Programmer()" << endl;
    }
    ~Programmer()
    {
        cout << "Call ~Programmer()" << endl;
    }

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

};

int main() {
    Person p;
    p.doWork();
    Programmer pr;
    pr.doWork();
}
```

Overriding: Example



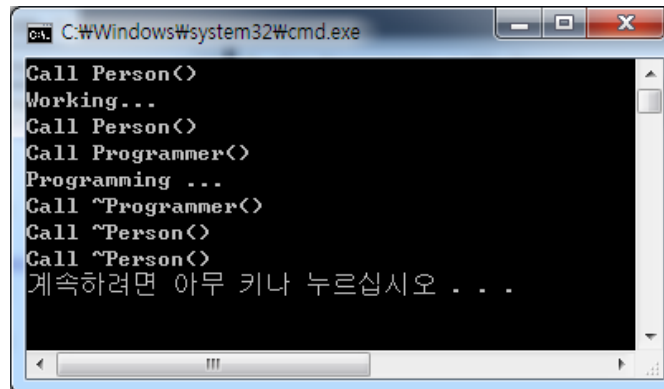
```
C:\Windows\system32\cmd.exe
Call Person<>
Working...
Call Person<>
Call Programmer<>
Working...
Call ~Programmer<>
Call ~Person<>
Call ~Person<>
계속하려면 아무 키나 누르십시오 . . .
```

Inheritance: Example

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

Overriding: Example



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


Up-casting

Converting a derived-class reference or pointer to a base-class

Up-casting

- Up-casting 이란?
 - 하위 클래스의 객체를 상위 클래스의 객체에 대입할 때 발생하는 형 변환을 업 캐스팅(Up-casting)이라고 한다.
 - 일반적인 방법으로 업 캐스팅을 하게 될 경우 하위 클래스의 데이터가 손실 될 수 있다.
 - 데이터 손실을 방지하기 위해 참조자 또는 포인터를 사용하여 업 캐스팅을 한다.

Up-casting: Example

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

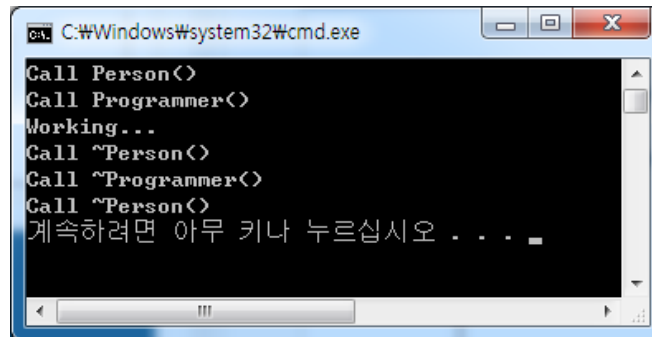
    void doWork()
    {
        cout << "Working ..." << endl;
    }
};
```

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

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

int main() {
    Programmer pr;
    Person p = pr;
    p.doWork();
}
```

Up-casting: Example



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

Up-casting: Example

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

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

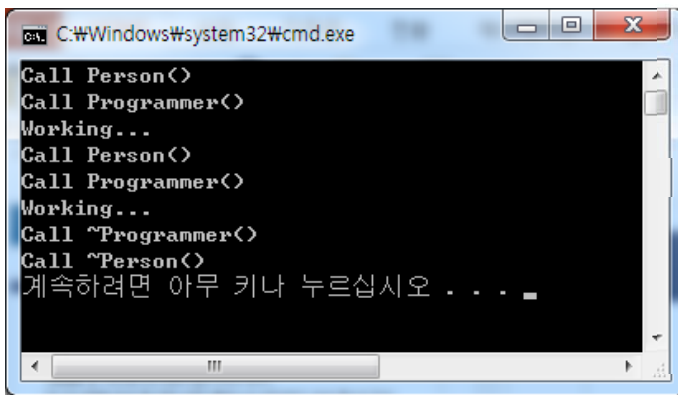
```
class Programmer : public Person {
public:
    Programmer() : Person()
    {
        cout << "Call Programmer()" << endl;
    }
    ~Programmer()
    {
        cout << "Call ~Programmer()" << endl;
    }

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

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

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

Up-casting: Example



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

- Overriding은 일어나지 않음.
- Overriding을 하려면?
- ➔ 이에 대한 해답은 다음 강의에서 virtual keyword 대한 내용을 다룰 때 자세하...

Down-casting

Converting a base-class pointer (reference) to a derived-class pointer (reference)

Down-casting

- Down-casting 이란?
 - 상위 클래스의 객체를 하위 클래스의 객체에 대입할 때 발생하는 형 변환을 다운 캐스팅(Down-casting)이라고 한다.
 - 일반적으로 상위 클래스의 객체를 하위 클래스 객체로 형 변환 하는 것은 상대적으로 메모리가 작은 상위 클래스 객체에서 메모리가 큰 하위 클래스의 객체로 변환하기 때문에 정상 동작을 보장하지 않는다.

Down-casting: Example

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

    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 p;
    Programmer pr = p;
    p.doWork();
}
```

Down-casting: Example

error C2440: '초기화 중' : 'Person'에서 'Programmer'(으)로 변환할 수 없습니다.

Down-casting: Example

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

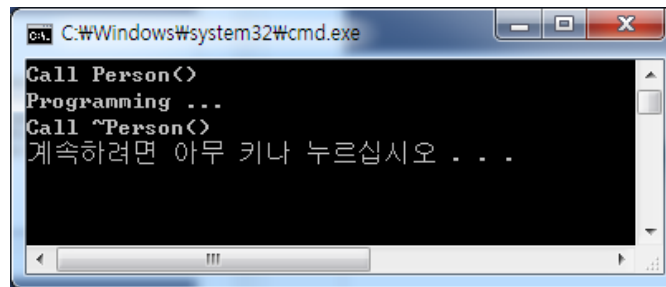
    void doWork()
    {
        cout << "Working ..." << endl;
    }
};
```

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

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

int main() {
    Person p;
    Programmer *pr = reinterpret_cast<Programmer*>(&p);
    pr->doWork();
}
```

Down-casting: Example



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