상속의 이해

박 종 혁 교수

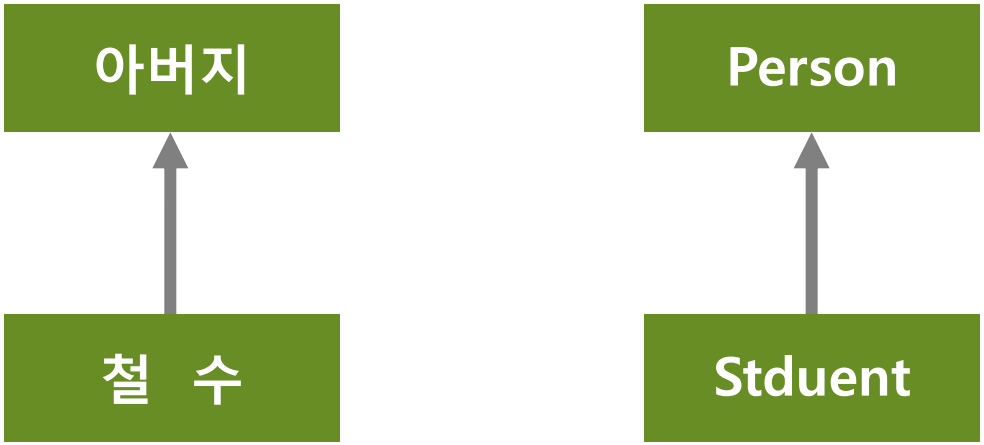
UCS Lab

Tel: 970-6702

Email: jhpark1@seoultech.ac.kr

상속의 기본 개념

- 상속의 예1
  - "철수는 아버지로부터 좋은 목소리와 큰 키를 물려 받았다."
- 상속의 예2
  - "Student 클래스가 Person 클래스를 상속한다."



파생 클래스(derived class)

- 상속(inheritance)
  - 한 클래스가 다른 클래스에서 정의된 속성(자료,함수)를 이어받아 그대로 사용
  - 이미 정의된 클래스를 바탕으로 필요한 기능을 추가하여 정의
  - 소프트웨어 재사용 지원

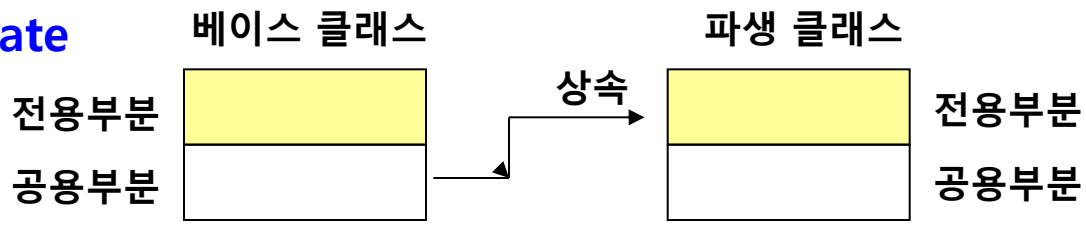
- 파생 클래스는 C++에서 각 클래스의 속성을 공유하고 물려받는 객체지향 프로그래밍의 상속(inheritance)을 구현한 것
  - 이미 만들어진 기존의 클래스를 베이스 클래스(base class) 또는 부모 클래스(parent class) or 슈퍼 클래스
  - 이를 상속 받아 새로 만들어지는 클래스를 파생 클래스(derived class) or 하위클래스 라고도 함.

공용부분 상속

```
class 파생클래스명 : public[private] 베이스클래스명  {
        .....
};
```

public

private



객체생성 순서

1.메모리 할당

2.베이스 클래스 생성자 실행

3.Derived클래스 생성자 실행

상속의 이해를 위한 이 책의 접근방식

✔ 1단계: 문제의 제시

상속과 더불어 다형성의 개념을 적용해야만 해결 가능한 문제를 먼저 제시한다.

✔ 2단계: 기본개념 소개

상속의 문법적 요소를 하나씩 소개해 나간다. 그리고 그 과정에서 앞서 제시한 문제의 해결책을 함께 고민해 나간다.

✔ 3단계: 문제의 해결

처음 제시한 문제를, 상속을 적용하여 해결한다. 그리고 이 때 여러분의 감탄사를 기대한다.

위의 흐름대로 상속을 이해하기 바랍니다.

상속은 '기존에 정의해 놓은 클래스의 재활용을 목적으로 만들어진 문법적 요소'라고 이해하는 경우가 많다. 하지만 상속에는 다른, 더 중요한 의미가 담겨있다.

문제의 제시

프로그램에 추가할 직급의 형태

- 영업직(Sales) 조금 특화된 형태의 고용직이다. 인센티브 개념이 도입
- 임시직(Temporary) 학생들을 대상으로 하는 임시고용 형태, 흔히 아르바이트라 함

확장 이후의 급여지급 방식

- 고용직 급여 연봉제! 따라서 매달의 급여가 정해져 있다.
- 영업직 급여 '기본급여 + 인센티브' 의 형태
- 임시직 급여 '시간당 급여 × 일한 시간' 의 형태

이 문제는 영업직과 임시직에 해당하는 클래스의 추가로 끝나지 않는다. 컨트롤 클래스인 EmployeeHandlder 클래스의 대대적인 변경으로 이어진다.

좋은 코드는 요구사항의 변경 및 기능의 추가에 따른 변경이 최소화되어야 한다. 그리고 이를 위한 해결책으로 상속이 사용된다.

EmployeeManager1.cpp

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

class PermanentWorker
{
private:
	char name[100];
	int salary;
public:
	PermanentWorker(char* name, int money)
		: salary(money)
	{
		strcpy(this->name, name);
	}
	int GetPay() const
	{
		return salary;
	}
	void ShowSalaryInfo() const
	{
		cout<<"name: "<<name<<endl;
		cout<<"salary: "<<GetPay()<<endl<<endl;
	}
};
```

```
class EmployeeHandler
{
private:
	PermanentWorker* empList[50];
	int empNum;
public:
	EmployeeHandler() : empNum(0)
	{ }
	void AddEmployee(PermanentWorker* emp)
	{
		empList[empNum++]=emp;
	}
	void ShowAllSalaryInfo() const
	{
		for(int i=0; i<empNum; i++)
			empList[i]->ShowSalaryInfo();
	}
	void ShowTotalSalary() const
	{
		int sum=0;
		for(int i=0; i<empNum; i++)
			sum+=empList[i]->GetPay();
		cout<<"salary sum: "<<sum<<endl;
	}
	~EmployeeHandler()
	{
		for(int i=0; i<empNum; i++)
			delete empList[i];
	}
};
```

EmployeeManager1.cpp

```
int main(void)
{
    // 직원관리를 목적으로 설계된 컨트롤 클래스의
    객체생성
    EmployeeHandler handler;

    // 직원 등록
    handler.AddEmployee(new
    PermanentWorker("KIM", 1000));
    handler.AddEmployee(new
    PermanentWorker("LEE", 1500));
    handler.AddEmployee(new
    PermanentWorker("JUN", 2000));

    // 이번 달에 지불해야 할 급여의 정보
    handler.ShowAllSalaryInfo();

    // 이번 달에 지불해야 할 급여의 총합
    handler.ShowTotalSalary();
    return 0;
}
```

상속의 방법과 그 결과

```
class Person
{
private:
    int age;          // 나이
    char name[50];    // 이름
public:
    Person(int myage, char * myname) : age(myage)
    {
        strcpy(name, myname);
    }
    void WhatYourName() const
    {
        cout<<"My name is "<<name<<endl;
    }
    void HowOldAreYou() const
    {
        cout<<"I'm "<<age<<" years old"<<endl;
    }
};
```

```
class UnivStudent : public Person
{
private:
    char major[50];   // 전공과목
public:
    UnivStudent(char * myname, int myage, char * mymajor)
        : Person(myage, myname)
    {
        strcpy(major, mymajor);
    }
    void WhoAreYou() const
    {
        WhatYourName();
        HowOldAreYou();
        cout<<"My major is "<<major<<endl<<endl;
    }
};
```

Person 클래스를 public 상속함

Person 클래스의 멤버

Person	↔	UnivStudent
상위 클래스	↔	하위 클래스
기초(base) 클래스	↔	유도(derived) 클래스
슈퍼(super) 클래스	↔	서브(sub) 클래스
부모 클래스	↔	자식 클래스

용어정리

상속받은 클래스의 생성자 정의

```
UnivStudent(char * myname, int myage, char * mymajor)
    : Person(myage, myname)
{
    strcpy(major, mymajor);
}
```

이니셜라이저를 통해서 유도 클래스는 기초 클래스의 생성자를 명시적으로 호출해야 한다.

유도 클래스의 생성자는 기초 클래스의 멤버를 초기화 할 의무를 갖는다. 단! 기초 클래스의 생성자를 명시적으로 호출해서 초기화해야 한다.

```
int main(void)
{
    UnivStudent ustd1("Lee", 22, "Computer eng.");
    ustd1.WhoAreYou();

    UnivStudent ustd2("Yoon", 21, "Electronic eng.");
    ustd2.WhoAreYou();
    return 0;
};
```

때문에 유도 클래스 UnivStudent는 기초 크래스의 생성자 호출을 위한 인자까지 함께 전달받아야 한다.

private 멤버는 유도 클래스에서도 접근이 불가능하므로, 생성자의 호출을 통해서 기초 클래스의 멤버를 초기화해야 한다.

UnivStudentInheri.cpp

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

class Person
{
private:
	int age;	// 나이
	char name[50];	// 이름
public:
	Person(int myage, char * myname) : age(myage)
	{
		strcpy(name, myname);
	}
	void WhatYourName() const
	{
		cout<<"My name is "<<name<<endl;
	}
	void HowOldAreYou() const
	{
		cout<<"I'm "<<age<<" years old"<<endl;
	}
};
```

```
class UnivStudent : public Person
{
private:
	char major[50];	// 전공과목
public:
	UnivStudent(char * myname, int myage, char * mymajor)
		: Person(myage, myname)
	{
		strcpy(major, mymajor);
	}
	void WhoAreYou() const
	{
		WhatYourName();
		HowOldAreYou();
		cout<<"My major is "<<major<<endl<<endl;
	}
};

int main(void)
{
	UnivStudent ustd1("Lee", 22, "Computer eng.");
	ustd1.WhoAreYou();

	UnivStudent ustd2("Yoon", 21, "Electronic eng.");
	ustd2.WhoAreYou();
	return 0;
};
```

유도 클래스의 객체생성 과정

```
class SoBase
{
private:
    int baseNum;
public:
    SoBase() : baseNum(20)
    {
        cout<<"SoBase()"<<endl;
    }
    SoBase(int n) : baseNum(n)
    {
        cout<<"SoBase(int n)"<<endl;
    }
    void ShowBaseData()
    {
        cout<<baseNum<<endl;
    }
};
```

```
class SoDerived : public SoBase
{
private:
    int derivNum;
public:
    SoDerived() : derivNum(30)
    {
        cout<<"SoDerived()"<<endl;
    }
    SoDerived(int n) : derivNum(n)
    {
        cout<<"SoDerived(int n)"<<endl;
    }
    SoDerived(int n1, int n2) : SoBase(n1), derivNum(n2)
    {
        cout<<"SoDerived(int n1, int n2)"<<endl;
    }
    void ShowDerivData()
    {
        ShowBaseData();
        cout<<derivNum<<endl;
    }
};
```

```
int main(void)
{
    cout<<"case1..... "<<endl;
    SoDerived dr1;
    dr1.ShowDerivData();
    cout<<"--------------------"<<endl;
    cout<<"case2..... "<<endl;
    SoDerived dr2(12);
    dr2.ShowDerivData();
    cout<<"--------------------"<<endl;
    cout<<"case3..... "<<endl;
    SoDerived dr3(23, 24);
    dr3.ShowDerivData();
    return 0;
};
```

```
case1.....
SoBase()
SoDerived()
20
30
--------------------
case2.....
SoBase()
SoDerived(int n)
20
12
--------------------
case3.....
SoBase(int n)
SoDerived(int n1, int n2)
23
24
```

실행결과

“유도클래스의 객체생성 과정에서 기초 클래스의 생성자는 100% 호출된다.”

“유도 클래스의 생성자에서 기초 클래스의 생성자 호출을 명시하지 않으면, 기초 클래스의 void생성자가 호출된다.”

DerivCreOrder.cpp

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

class SoBase
{
private:
	int baseNum;
public:
	SoBase() : baseNum(20)
	{
		cout<<"SoBase()"<<endl;
	}
	SoBase(int n) : baseNum(n)
	{
		cout<<"SoBase(int n)"<<endl;
	}
	void ShowBaseData()
	{
		cout<<baseNum<<endl;
	}
};

class SoDerived : public SoBase
{
private:
	int derivNum;
public:
	SoDerived() : derivNum(30)
	{
		cout<<"SoDerived()"<<endl;
	}
	SoDerived(int n) : derivNum(n)
	{
		cout<<"SoDerived(int n)"<<endl;
	}
	SoDerived(int n1, int n2) : SoBase(n1), derivNum(n2)
	{
		cout<<"SoDerived(int n1, int n2)"<<endl;
	}
	void ShowDerivData()
	{
		ShowBaseData();
		cout<<derivNum<<endl;
	}
};

int main(void)
{
	cout<<"case1..... "<<endl;
	SoDerived dr1;
	dr1.ShowDerivData();
	cout<<"-------------------"<<endl;
	cout<<"case2..... "<<endl;
	SoDerived dr2(12);
	dr2.ShowDerivData();
	cout<<"-------------------"<<endl;
	cout<<"case3..... "<<endl;
	SoDerived dr3(23, 24);
	dr3.ShowDerivData();
	return 0;
};
```

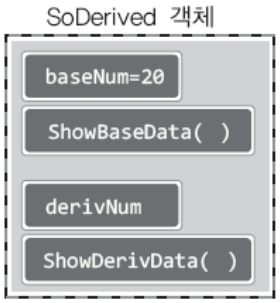
유도 클래스의 객체생성 과정 case1

순서 1. 메모리 공간의 할당

SoDerived dr1


```
SoDerived( ) : derivNum(30)
{
    cout<<"SoDerived()"<<endl;
}
```

순서 2. 유도 클래스의 void 생성자 호출



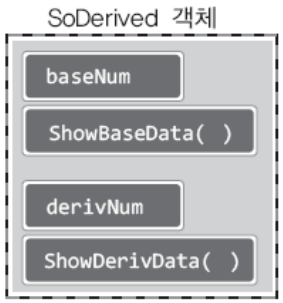

```
SoDerived( ) : derivNum(30)
{
    cout<<"SoDerived()"<<endl;
}
```

순서 3. 이니셜라이저를 통한 기초 클래스의 생성자 호출이 명시적으로 정의되어 있지 않으므로 void 생성자 호출

```
SoBase( ) : baseNum(20)
{
   cout<<"SoBase()"<<endl;
}
```

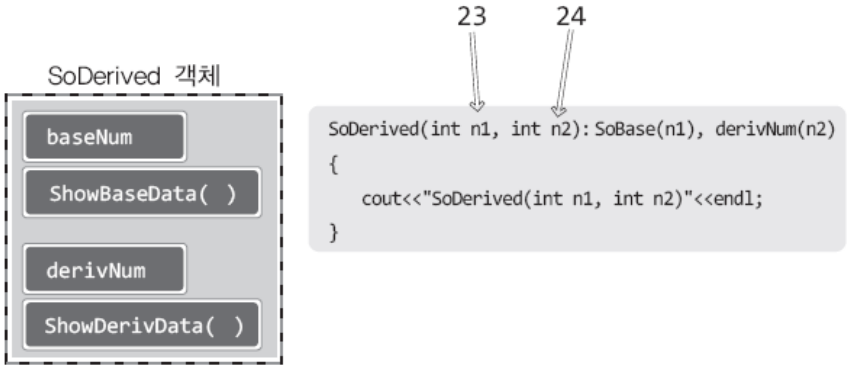
순서 4. 유도 클래스의 실행

유도 클래스의 객체생성 과정 case2

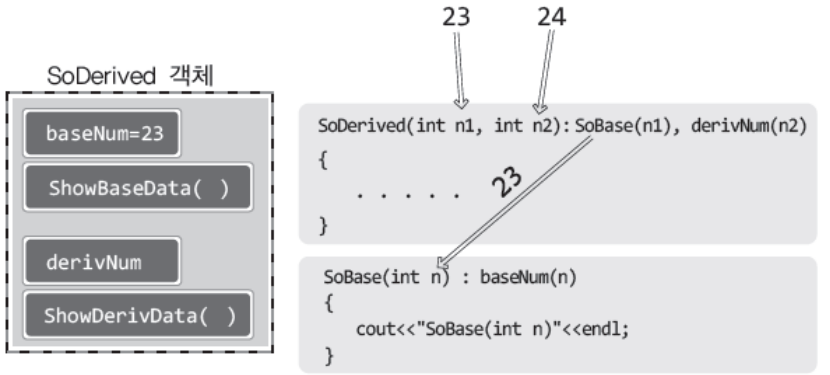


순서 1. 메모리 공간의 할당

```
SoDerived dr3(23, 24);
```



순서 2. 유도 클래스의 생성자 호출



순서 3. 기초 클래스의 생성자 호출 및 실행

순서 4. 유도 클래스의 생성자 실행

상속의 소멸자 예

```
#include <iostream>
using std::endl;
using std::cout;

class AAA    //Base 클래스
{
public:
   AAA() {
      cout<<"AAA() call!"<<endl;
  }
   ~AAA() {
      cout<<“~AAA(int i) call!"<<endl;
   }
};

class BBB : public AAA    //Derived 클래스
{
public:
   BBB(){
      cout<<"BBB() call!"<<endl;
  }
   ~BBB() {
      cout<<"~BBB() call!"<<endl;
   }
};
```

객체소멸 순서

----

1. Derived클래스 소멸자 실행

2. base 클래스 소멸자 실행

3. 메모리 반환(해제)

```
int main(void)
{
    BBB bbb;

    return 0;
}
```

```
AAA() call!
BBB() call!
~BBB() call!
~AAA() call!
```

유도 클래스 객체의 소멸과정

```
class SoBase
{
private:
    int baseNum;
public:
    SoBase(int n) : baseNum(n)
    {
        cout<<"SoBase() : "<<baseNum<<endl;
    }
    ~SoBase()
    {
        cout<<"~SoBase() : "<<baseNum<<endl;
    }
};

class SoDerived : public SoBase
{
private:
    int derivNum;
public:
    SoDerived(int n) : SoBase(n), derivNum(n)
    {
        cout<<"SoDerived() : "<<derivNum<<endl;
    }
    ~SoDerived()
    {
        cout<<"~SoDerived() : "<<derivNum<<endl;
    }
};
```

```
SoBase() : 15
SoDerived() : 15
SoBase() : 27
SoDerived() : 27
~SoDerived() : 27
~SoBase() : 27
~SoDerived() : 15
~SoBase() : 15
```

실행결과

유도 클래스의 소멸자가 실행된 이후에 기초 클래스의 소멸자가 실행된다.

스택에 생성된 객체의 소멸순서는 생성순서와 반대이다.

DerivDestOrder.cpp

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

class SoBase
{
private:
    int baseNum;
public:
    SoBase(int n) : baseNum(n)
    {
        cout<<"SoBase() : "<<baseNum<<endl;
    }
    ~SoBase()
    {
        cout<<"~SoBase() : "<<baseNum<<endl;
    }
};

class SoDerived : public SoBase
{
private:
    int derivNum;
public:
    SoDerived(int n) : SoBase(n), derivNum(n)
    {
        cout<<"SoDerived() :
    "<<derivNum<<endl;
    }
    ~SoDerived()
    {
        cout<<"~SoDerived() :
    "<<derivNum<<endl;
    }
};

int main(void)
{
    SoDerived drv1(15);
    SoDerived drv2(27);
    return 0;
};
```

유도 클래스 정의 모델

```
class Person
{
private:
    char * name;
public:
    Person(char * myname)
    {
        name=new char[strlen(myname)+1];
        strcpy(name, myname);
    }
    ~Person()
    {
        delete []name;
    }
    void WhatYourName() const
    {
        cout<<"My name is "<<name<<endl;
    }
};
```

```
class UnivStudent : public Person
{
private:
    char * major;
public:
    UnivStudent(char * myname, char * mymajor)
        : Person(myname)
    {
        major=new char[strlen(mymajor)+1];
        strcpy(major, mymajor);
    }
    ~UnivStudent()
    {
        delete []major;
    }
    void WhoAreYou() const
    {
        WhatYourName();
        cout<<"My major is "<<major<<endl<<endl;
    }
};
```

기초 클래스의 멤버 대상의 동적 할당은 기초 클래스의 생성자를 통해서, 소멸 역시 기초 클래스의 소멸자를 통해서

DestModel.cpp

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

class Person
{
private:
	char * name;
public:
	Person(char * myname)
	{
		name=new char[strlen(myname)+1];
		strcpy(name, myname);
	}
	~Person()
	{
		delete []name;
	}
	void WhatYourName() const
	{
		cout<<"My name is "<<name<<endl;
	}
};
```

```
class UnivStudent : public Person
{
private:
	char * major;
public:
	UnivStudent(char * myname, char * mymajor)
		:Person(myname)
	{
		major=new char[strlen(mymajor)+1];
		strcpy(major, mymajor);
	}
	~UnivStudent()
	{
		delete []major;
	}
	void WhoAreYou() const
	{
		WhatYourName();
		cout<<"My major is "<<major<<endl<<endl;
	}
};

int main(void)
{
	UnivStudent st1("Kim", "Mathmatics");
	st1.WhoAreYou();

	UnivStudent st2("Hong", "Physics");
	st2.WhoAreYou();
	return 0;
};
```

세 가지 형태의 상속

```
class Derived : public Base
{
    . . . . .
}
```

public 상속

접근 제어 권한을 그대로 상속한다!

단, private은 접근불가로 상속한다!

```
class Derived : protected Base
{
    . . . . .
}
```

protected 상속

protected보다 접근의 범위가 넓은 멤버는 protected로 상속한다.

단, private은 접근불가로 상속한다!

```
class Derived : private Base
{
    . . . . .
}
```

private 상속

private보다 접근의 범위가 넓은 멤버는 protected로 상속한다.

단, private은 접근불가로 상속한다!

세 가지 형태의 상속

- 접근권한 변경
  - Base 클래스의 멤버는 상속되는 과정에서 접근 권한 변경

상속 형태 / Base 클래스	public 상속	protected 상속	private 상속
public 멤버	public	protected	private
Protected 멤머	protected	protected	private
Private 멤버	접근 불가	접근 불가	접근 불가

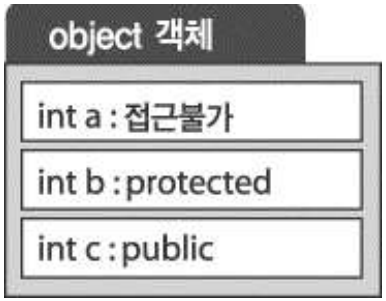
```
class 클래스명 {
[private:]              // 전용부분 멤버 정의
        .....           // 클래스 외부 접근 및 상속 안됨
protected:              // 보호부분 멤버 정의
        .....           // 클래스 외부 접근 안됨, 상속 됨
public:                 // 공용부분 멤버 정의
        .....           // 클래스 외부 접근 및 상속 됨
};
```

세 가지 형태의 상속

```
class Base
{
private:
    int a;
protected:
    int b;
public:
    int c;
};

class Derived : public Base  // public 상속 →
{
   // EMPTY
};
```

```
int main(void)
{
  Derived object;
  return 0;
};
```



protected ?

private ?

** 사용된 상속보다 넓은 범위는 그 범위로 맞춤.

protected로 선언된 멤버가 허용하는 접근의 범위

```
class Base
{
private:
    int num1;
protected:
    int num2;
public:
    int num3;
    void ShowData()
    {
        cout<<num1<<", "<<num2<<", "<<num3;
    }
};
```

```
class Derived : public Base
{
public:
    void ShowBaseMember()
    {
        cout<<num1;        // 컴파일 에러
        cout<<num2;        // 컴파일 OK!
        cout<<num3;        // 컴파일 OK!
    }
};
```

private < protected < public

private을 기준으로 보면,

protected는 private과 달리 상속관계에서의 접근을 허용한다!

protected 멤버 상속

- 파생 클래스의 멤버함수는 베이스 클래스의 공용멤버에 직접 접근이 가능하지만 베이스 클래스의 전용멤버에 대해서는 접근할 수 없음

- 파생 클래스의 멤버함수가 베이스 클래스의 전용멤버를 상속받아 자유로이 사용을 위해서는 다음 두 가지 방식 사용
  - 프렌드 함수를 사용
  - 베이스 클래스 정의 시 protected 키워드를 사용한 보호부분을 정의

- 보호부분에 있는 멤버는 전용멤버와 같이 외부에서는 직접 접근할 수 없지만 파생 클래스의 멤버함수에서는 직접 접근이 가능
  - 보호부분 멤버가 파생 클래스에서 public으로 상속 받으면 파생 클래스의 protected 멤버가 됨
  - private으로 상속 받으면 파생 클래스에서 전용부분 멤버가 됨

protected 멤버 상속 예

- 상속관계에 놓여있는 경우 접근 허용
- 그 이외는 private 멤버와 동일

```
class AAA
{
private:
    int a;
protected:
    int b;
};

class BBB: public AAA
{
public:
    void SetData() {
        a=10;  // private 멤버, 따라서 에러
          b=10; // protected 멤버, OK!
      }
};
```

```
int main(void)
{
    AAA aaa;
    aaa.a=10; // private 멤버, 따라서 에러 !!
    aaa.b=20; // protected 멤버, 따라서 에러!!

    BBB bbb;
    bbb.SetData();

    return 0;
}
```

protected 상속과 private 상속

```
class Base
{
private:
    int num1;
protected:
    int num2;
public:
    int num3;
};
```

```
class Derived : private Base
{
    // empty!
};
```

private
상속의 결과

```
class Derived : private Base
{
접근불가:
    int num1;
private:
    int num2;
private:
    int num3;
};
```

때문에 이 이상의 상속은 무의미할 수 있다.

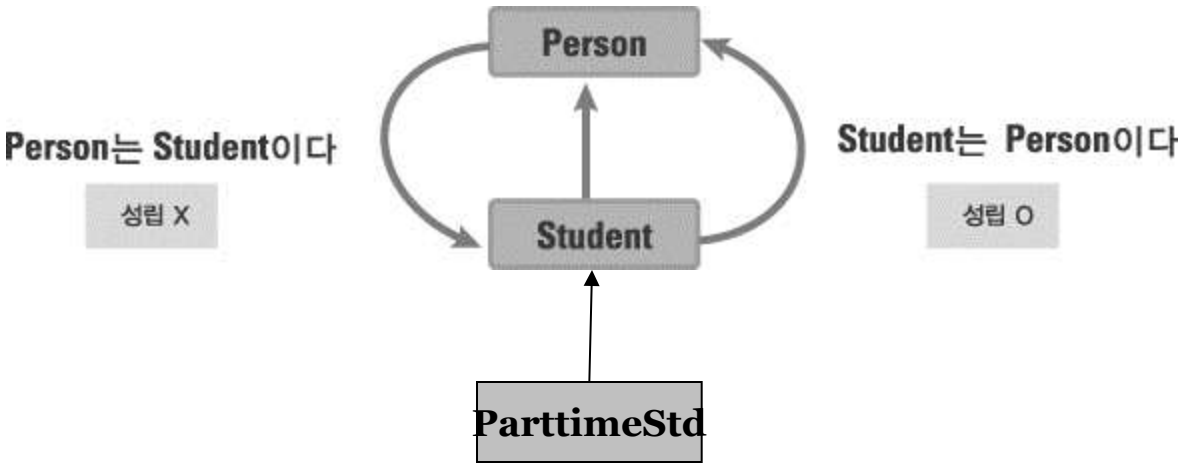
```
class Derived : protected Base
{
    // empty!
};
```

protected
상속의 결과

```
class Derived : protected Base
{
접근불가:
    int num1;
protected:
    int num2;
protected:
    int num3;
};
```

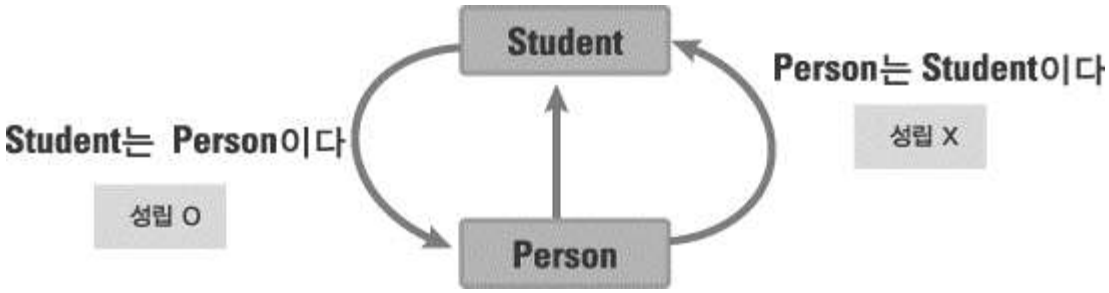
상속의 조건

- public 상속은 is-a 관계가 성립되도록 하자.



상속의 조건

- 잘못된 상속의 예



상속의 조건

- HAS-A(소유) 관계에 의한 상속!
  - 경찰은 몽둥이를 소유한다
  - The Police have a cudgel.

```
/*
  hasa1.cpp
*/
#include <iostream>
using std::endl;
using std::cout;

class Cudgel  //몽둥이
{
public:
        void Swing(){ cout<<"Swing a cudgel!"<<endl; }
};

class Police : public Cudgel  //몽둥이를 소유하는 경찰
{
public:
        void UseWeapon(){ Swing(); }
};
```

```
int main()
{
        Police pol;
        pol.UseWeapon();

        return 0;
}
```

Police is a Cudgel **(X)**
Police has a Cudgel **(O)**

상속의 조건

- HAS-A에 의한 상속 그리고 대안!
  - **포함 관계를** 통해서 소유 관계를 표현
  - **객체 멤버에** 의한 포함 관계의 형성
  - **객체 포인터 멤버에** 의한 포함 관계의 형성

Police 객체는 Cudgel 객체를 포함한다.

상속의 조건

```
/*    hasa2.cpp */
class Cudgel  //몽둥이
{
public:
      void Swing(){ cout<<"Swing a cudgel!"<<endl; }
};
class Police //몽둥이를 소유하는 경찰
{
      Cudgel cud;
public:
      void UseWeapon(){ cud.Swing(); }
};

int main()
{
      Police pol;
      pol.UseWeapon();
      return 0;
}
```

상속의 조건

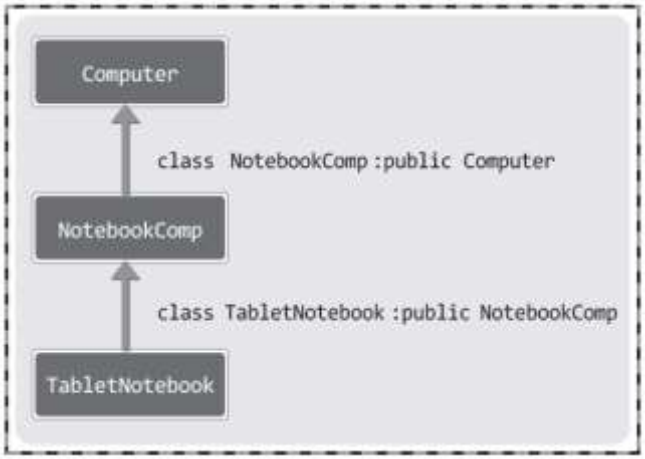
```
/*    hasa3.cpp */
class Cudgel  //몽둥이
{
public:
      void Swing(){ cout<<"Swing a cudgel!"<<endl; }
};
class Police //몽둥이를 소유하는 경찰
{
      Cudgel* cud;
public:
      Police(){ cud=new Cudgel; }
      ~Police(){ delete cud; }
      void UseWeapon(){ cud->Swing(); }
};
int main()
{
      Police pol;
      pol.UseWeapon();
      return 0;
}
```

상속의 기본 조건인 IS-A 관계의 성립

- 전화기 → 무선 전화기
- 컴퓨터 → 노트북 컴퓨터

- 무선 전화기는 일종의 전화기입니다.
- 노트북 컴퓨터는 일종의 컴퓨터입니다.

- 무선 전화기 is a 전화기
- 노트북 컴퓨터 is a 컴퓨터

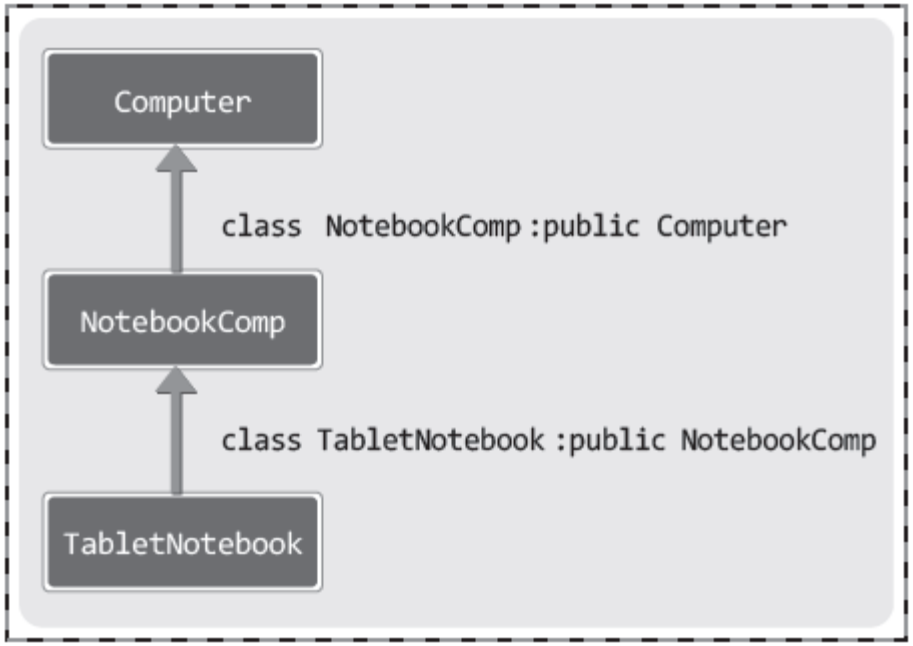


무선 전화기는 전화기의 기본 기능에 새로운 특성이 추가된 것이다.

노트북 컴퓨터는 컴퓨터의 기본 기능에 새로운 특성이 추가된 것이다.

이렇듯 is-a 관계는 논리적으로 상속을 기반으로 표현하기에 매우 적절하다.

IS-A 기반의 예제



예제는 도서 본문을 참조합시다!

- NotebookComp(노트북 컴퓨터)는 Computer(컴퓨터)이다.
- TabletNotebook(타블렛 컴퓨터)는 NotebookComp(노트북 컴퓨터)이다.
- TabletNotebook(타블렛 컴퓨터)는 Computer(컴퓨터)이다.

ISAInheritance.cpp

```
#include <iostream>
#include <cstring>
using namespace std;
class Computer
{
private:
    char owner[50];
public:
    Computer(char * name)
    {
        strcpy(owner, name);
    }
    void Calculate()
    {
        cout<<"요청 내용을 계산합니다."<<endl;
    }
};
```

```
class NotebookComp : public Computer
{
private:
    int battary;
public:
    NotebookComp(char * name, int initChag)
        : Computer(name), battary(initChag)
    { }
    void Charging() { battary+=5; }
    void UseBattary() { battary-=1; }
    void MovingCal()
    {
        if(GetBattaryInfo()<1)
        {
            cout<<"충전이 필요합니다."<<endl;
            return;
        }
        cout<<"이동하면서 ";
        Calculate();
        UseBattary();
    }
    int GetBattaryInfo() { return battary; }
};
```

ISAInheritance.cpp

```
class TabletNotebook : public NotebookComp
{
private:
	char regstPenModel[50];
public:
	TabletNotebook(char * name, int initChag, char * pen)
		: NotebookComp(name, initChag)
	{
		strcpy(regstPenModel, pen);
	}
	void Write(char * penInfo)
	{
		if(GetBattaryInfo()<1)
		{
			cout<<"충전이 필요합니다."<<endl;
			return;
		}
		if(strcmp(regstPenModel, penInfo)!=0)
		{
			cout<<"등록된 펜이 아닙니다.";
			return;
		}
		cout<<"필기 내용을 처리합니다."<<endl;
		UseBattary();
	}
};
```

```
int main(void)
{
	NotebookComp nc("이수종", 5);
	TabletNotebook tn("정수영", 5, "ISE-241-242");
	nc.MovingCal();
	tn.Write("ISE-241-242");
	return 0;
}
```

HAS-A 관계를 상속으로 구성하면

```
class Gun
{
private:
    int bullet;     // 장전된 총알의 수
public:
    Gun(int bnum) : bullet(bnum)
    { }
    void Shut()
    {
        cout<<"BBANG!";
        bullet--;
    }
};
```

```
class Police : public Gun
{
private:
    int handcuffs;      // 소유한 수갑의 수
public:
    Police(int bnum, int bcuff)
        : Gun(bnum), handcuffs(bcuff)
    { }
    void PutHandcuff()
    {
        cout<<"SNAP!";
        handcuffs--;
    }
};
```

경찰은 총을 소유한다.
경찰 has a 총!

has a 관계도 상속으로 구현이 가능하다. 하지만 이러한 경우 Police와 Gun은 강한 연관성을 띠게 된다.
따라서 총을 소유하지 않은 경찰이나, 다른 무기를 소유하는 경찰을 표현하기가 쉽지 않아진다.

HASInheritance.cpp

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

class Gun
{
private:
    int bullet;             // 장전된 총알의 수
public:
    Gun(int bnum) : bullet(bnum)
    { }
    void Shut()
    {
        cout<<"BBANG!"<<endl;
        bullet--;
    }
};
```

```
class Police : public Gun
{
private:
    int handcuffs;          // 소유한 수갑의 수
public:
    Police(int bnum, int bcuff)
        : Gun(bnum), handcuffs(bcuff)
    { }
    void PutHandcuff()
    {
        cout<<"SNAP!"<<endl;
        handcuffs--;
    }
};

int main(void)
{
    Police pman(5, 3); // 총알 5, 수갑 3
    pman.Shut();
    pman.PutHandcuff();
    return 0;
}
```

HAS-A 관계는 포함으로 표현한다

```
class Gun
{
private:
    int bullet;     // 장전된 총알의 수
public:
    Gun(int bnum) : bullet(bnum)
    { }
    void Shut()
    {
        cout<<"BBANG!"<<endl;
        bullet--;
    }
};
```

```
class Police
{
private:
    int handcuffs;      // 소유한 수갑의 수
    Gun * pistol;       // 소유하고 있는 권총
public:
    Police(int bnum, int bcuff)
        : handcuffs(bcuff)
    {
        if(bnum>0)
            pistol=new Gun(bnum);
        else
            pistol=NULL;
    }
    void PutHandcuff()
    {
        cout<<"SNAP!"<<endl;
        handcuffs--;
    }
    void Shut()
    {
        if(pistol==NULL)
            cout<<"Hut BBANG!"<<endl;
        else
            pistol->Shut();
    }
    ~Police()
    {
        if(pistol!=NULL)
            delete pistol;
    }
};
```

has a의 과계를 포함의 형태로 표현하면, 두 클래스간 연관성은 낮아지며, 변경 및 확장이 용이해진다.

즉, 총을 소유하지 않은 경찰의 표현이 쉬워지고, 추가로 무기를 소유하는 형태로의 확장도 간단해진다.

HASComposite.cpp

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

class Gun
{
private:
	int bullet;			// 장전된 총알의 수
public:
	Gun(int bnum) : bullet(bnum)
	{ }
	void Shut()
	{
		cout<<"BBANG!"<<endl;
		bullet--;
	}
};

class Police
{
private:
	int handcuffs;	// 소유한 수갑의 수
	Gun * pistol;	// 소유하고 있는 권총
public:
	Police(int bnum, int bcuff)
		: handcuffs(bcuff)
	{
		if(bnum>0)
			pistol=new Gun(bnum);
		else
			pistol=NULL;
	}
	void PutHandcuff()
	{
		cout<<"SNAP!"<<endl;
		handcuffs--;
	}
	void Shut()
	{
		if(pistol==NULL)
			cout<<"Hut BBANG!"<<endl;
		else
			pistol->Shut();
	}
	~Police()
	{
		if(pistol!=NULL)
			delete pistol;
	}
};
int main(void)
{
	Police pman1(5, 3);
	pman1.Shut();
	pman1.PutHandcuff();

	Police pman2(0, 3);	// 권총소유하지 않은 경찰
	pman2.Shut();
	pman2.PutHandcuff();
	return 0;
}
```

Q & A