창의적 소프트웨어 설계

8주차 실습 – Polymorphism

노인우, inwoo13@hanyang.ac.kr
한중수, soohan@hanyang.ac.kr

Abstract Base Classes

목표

- Pure Virtual Functions
- Abstract Base Classes
- Overriding
- Appendix

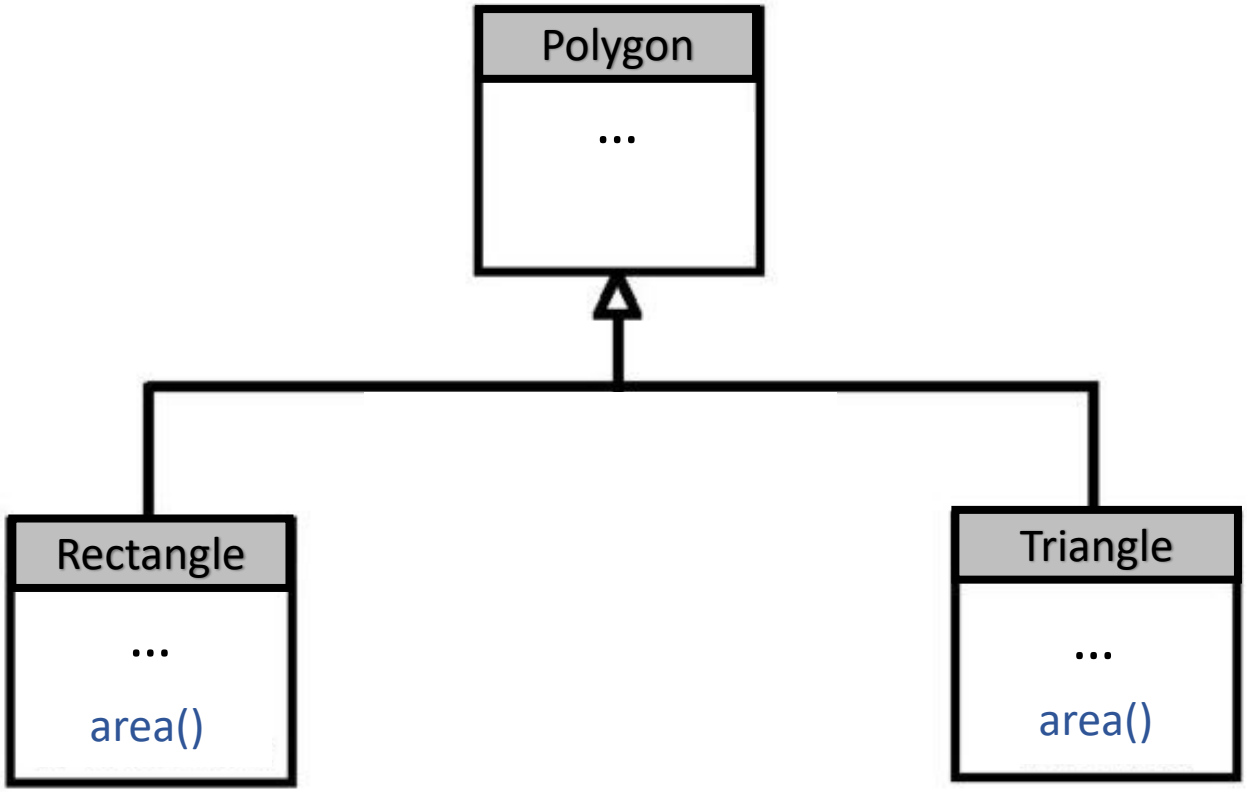
Inheritance & Polymorphism

- Inheritance without Polymorphism is possible!
  - Addition or Extension of base class

- Polymorphism without Inheritance is impossible!
  - Treat objects from different classes the same way
  - Needs virtual inheritance
  - class that declares or inherits a virtual function is called *polymorphic class*

Inheritance & Polymorphism

- Inheritance
  - Addition or Extension of base class



Example

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

class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b) { width=a; height=b; }
};

class Rectangle: public Polygon {
  public:
    int area() { return width*height; }
};
```

Example (Cont.)

```
class Triangle: public Polygon {
  public:
    int area() { return width*height/2; }
};

int main () {
  Rectangle rect;
  Triangle trgl;
  rect.set_values (4,5);
  trgl.set_values (4,5);
  cout << rect.area() << '\n';
  cout << trgl.area() << '\n';
  return 0;
}
```

Virtual Functions

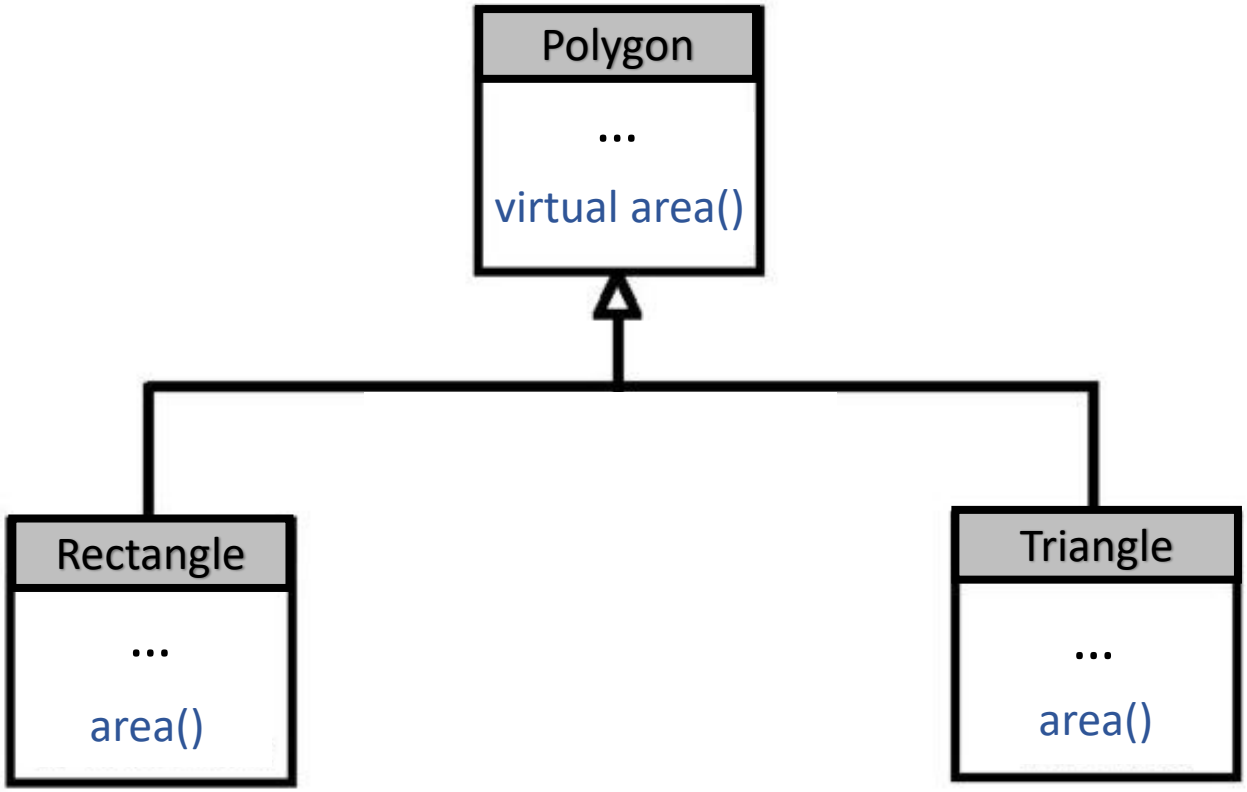
- Virtual Functions
  - Can be redefined in a derived class
  - Preserve calling properties through references
  - Abstract implementation!

```
class BaseClass {
  protected:
    int a, b;
  public:
    virtual int func () { return 0; }
};
```

```
class PolyClass: public BaseClass {
  public:
    int func () { return 1; }
};
```

Inheritance & Polymorphism

- Polymorphism
  - Treat objects from different classes the same way
  - Needs virtual inheritance



Example

```
#include <iostream>
using namespace std;
class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area () { return 0; }
};
class Rectangle: public Polygon {
  public:
    int area () { return width * height; }
};
```

Example (Cont.)

```
class Triangle: public Polygon {
  public:
    int area ()
      { return (width * height / 2); }
};
int main () {
  Rectangle rect;
  Triangle trgl;
```

```
  cout << ppoly1->area() << '\n';
  cout << ppoly2->area() << '\n';
return 0;
}
```

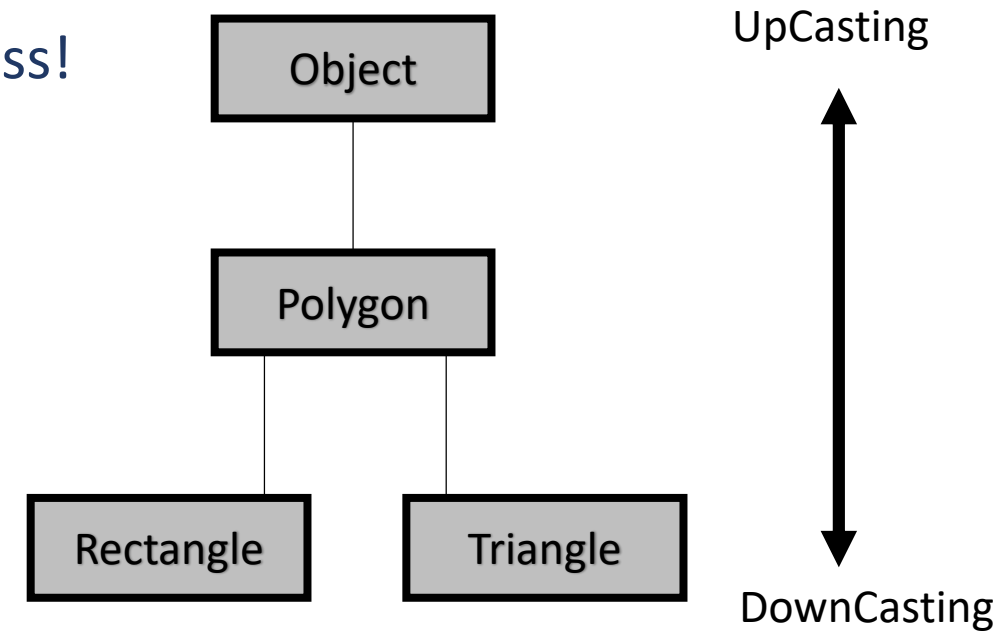
Inheritance & Polymorphism

- Key features
  - Pointer to a derived class is type-compatible with a pointer to its base class
  - Can preserve calling properties through references

Casting

- UpCasting
  - Up to Base Class!
  - Possible! Because derived class includes members of base class
  - ABC abc = abcd;

- DownCasting
  - Down to Derived Class!
  - ABCD abcd = abc;



Example

```
#include <iostream>

using namespace std;

class Base {
  public:
    void showBase() { cout << "Base Function" << endl; }
};

class Derived: public Base {
  public:
    void showDerived() { cout << "Derived Function" << endl; }
};
```

Example

```
int main(void) {
  Derived d2;
  Derived *d1;
  Base *b = &d2;          // UpCasting

  d1 = b;                 // Needs DownCasting

  d1->showDerived();
  d1->showBase();
}
```

```
example3.cpp: In function ‘int main()’:
example3.cpp:21:6: error: invalid conversion from ‘Base*’ to ‘Derived*’ [-fpermissive]
                d1 = b;
                     ^
```

Pure Virtual Function

- Pure virtual function
  - virtual functions with no definition
  - Start with ‘virtual’, ends with ‘= 0’
  - Can have constructors

```
// An abstract class with constructor
class Base {
  protected:
    int x;
  public:
    virtual void func() = 0;
    Base(int i) { x = I; }
};
```

Abstract Base Classes

- Abstract Class
  - Have at least one pure virtual function
  - Must implement own version of each derived class

Example

```
// pure virtual functions make a class abstract
#include<iostream>

using namespace std;

class Test
{
   int x;
    public:
        virtual void show() = 0;
         int getX() { return x; }
};

int main(void)
{
    Test t;
    return 0;
}
```

[본 선언문을 수정하지 않고 외부에 derived class를 선언하여 에러 해결]

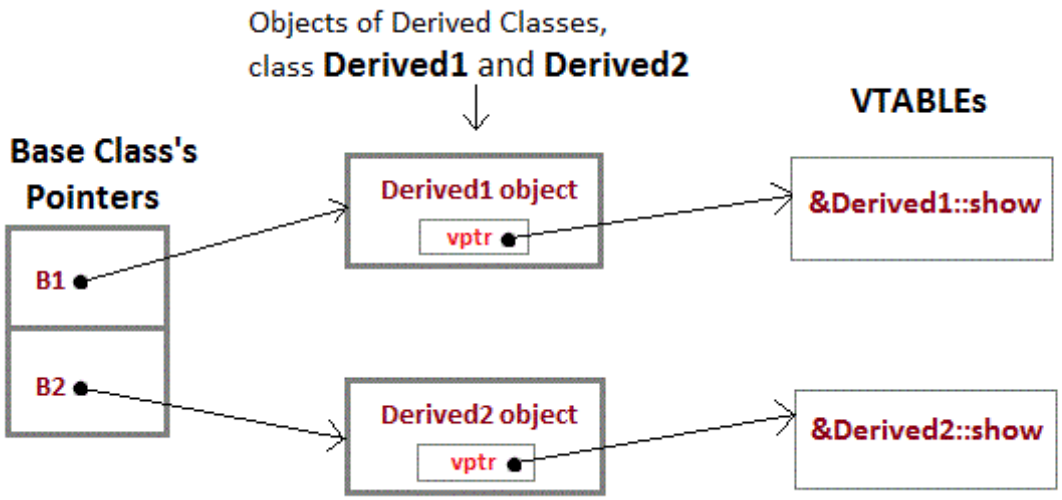
Output:

Compiler Error: cannot declare variable ‘t’ to be of abstract

Type ‘Test’ because the following virtual functions are pure within ‘Test’: note: virtual void Test::show()

Abstract Base Classes

- Why can't we create object of abstract class?
  - Reserve a slot for a function in the VTABLE
  - Doesn't put any address in the slot
  - Incomplete!



vptr, is the vpointer, which points to the Virtual Function for that object.

VTABLE, is the table containing address of Virtual Functions of each class.

Example

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

class A
{
    public:
    virtual void show() {
            cout << “Base class\n”;
    }
};

class B: public A
{
    private:
            virtual void show() {
            cout << “Derived class\n”;
    }
};

int main(void)
{
    A *a;                < base class pointer
    B b;
    a = &b;
    a -> show();         < Late binding occurs!
}
```

Output: Derived class

Abstract Base Classes

- But can have pointers and references of abstract class

```
#include<iostream>

using namespace std;

class Base
{
    public:
        virtual void show() = 0;
};

class Derived: public Base
{
    public:
        void show() { cout << "In Derived \n"; }
};

int main(void)
{
    Base *bp = new Derived();
    bp->show();
    return 0;
}
```

Output: In Derived

Overriding

- Similar to redefinition

- If we do not override the pure virtual function, Then derived class also becomes abstract class

참고자료

1. Polymorphism, https://en.wikipedia.org/wiki/Polymorphism_(computer_science)
2. Polymorphism, http://www.cplusplus.com/doc/tutorial/polymorphism
3. Casting, https://m.blog.naver.com/PostView.nhn?blogId=madplay&logNo=220203111905&proxyReferer=https%3A%2F%2Fwww.google.co.kr%2F
4. Casting, http://www.cs.utexas.edu/~cannata/cs345/Class%20Notes/14%20Java%20Upcasting%20Downcasting.htm


Q&A

Appendix #1. ostream&

◆ 날짜를 ostream 연산자 오버라이딩을 통해 출력하는 예제

```
#include<iostream>

using namespace std;

class Date
{
    public:
        int mo, da, yr;
        Date(int m, int d, int y) { mo = m; da = d; yr = y; }
};

ostream& operator<<(ostream& os, const Date& dt)
{
    os << dt.mo << '/' << dt.da << '/' << dt.yr << "\n";
    return os;
}

int main()
{
    Date dt(15, 11, 17);
    cout << dt;
    return 0;
}
```

Appendix #2. ACM-ICPC

◆ ACM ICPC ?

- acm 학회에서 1년마다 주최하는 알고리즘 대회
- 삼성, 네이버 등에서 따라서 시행하는데, 공신력이 있다.

◆ 어떻게 연습하나?

- Online judge site > https://www.acmicpc.net/
- 문제가 아주 많다. 기출문제, 알고리즘 별 분류까지 정리가 잘 되어있다.
- 네임드 프로그래밍 언어 대부분 지원,
  BrainFuck, Whitespace 등 ~~변태적인~~ 언어들까지 컴파일러를 지원

◆ 어떤 식으로 출제되나?

- 실습 과제랑 비슷
- 서버에 코드를 제출하여 통과하면 클리어
- 맞았습니다!! 모든 테스트 케이스를 통과했을 경우
- 그외엔 정답 처리되지 않는다. 물론 그 원인은 사용자가 잘못된 코드를 짜서

http://icpckorea.org/

Appendix #2. ACM-ICPC (Cont.)

◆ 추천문제

- 대표 쉬운 문제: A+B
  https://www.acmicpc.net/problem/1000

- 대표 간단한 문제: 거북이 (8911)
  https://www.acmicpc.net/problem/8911

◆ 숏코딩?

Golfscript	awk	아희
~+	{print$1+$2}	방다망해

```
int a, t;
for(scanf("%d",&t); t--; printf("%d\n",a))
{

        …
    a = (b < c ? c : b);
}
```

```
char a[501];
for(i=0; a[i]; i++)
{
   if(a[i]=='F') { … }
   if(a[i]=='B') { … }
   if(a[i]=='L') { … }
   if(a[i]=='R') { … }
}
```