객체지향형 프로그래밍과 자료구조

Ch 6. 다형성 (Polymorphism)과 가상함수 (Virtual Function)

정보통신공학과

교수 김 영 탁

(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

- 다형성 (polymorphism)
- 가상함수 (virtual function),
  지연 바인딩 (late binding)
- 상속관계의 객체 인스탄스간의
  upcasting, downcasting
- 다형성과 가상함수 기반의 그래픽 응용 프로그램
  - Windows 환경에서의 pixel 단위 그래픽

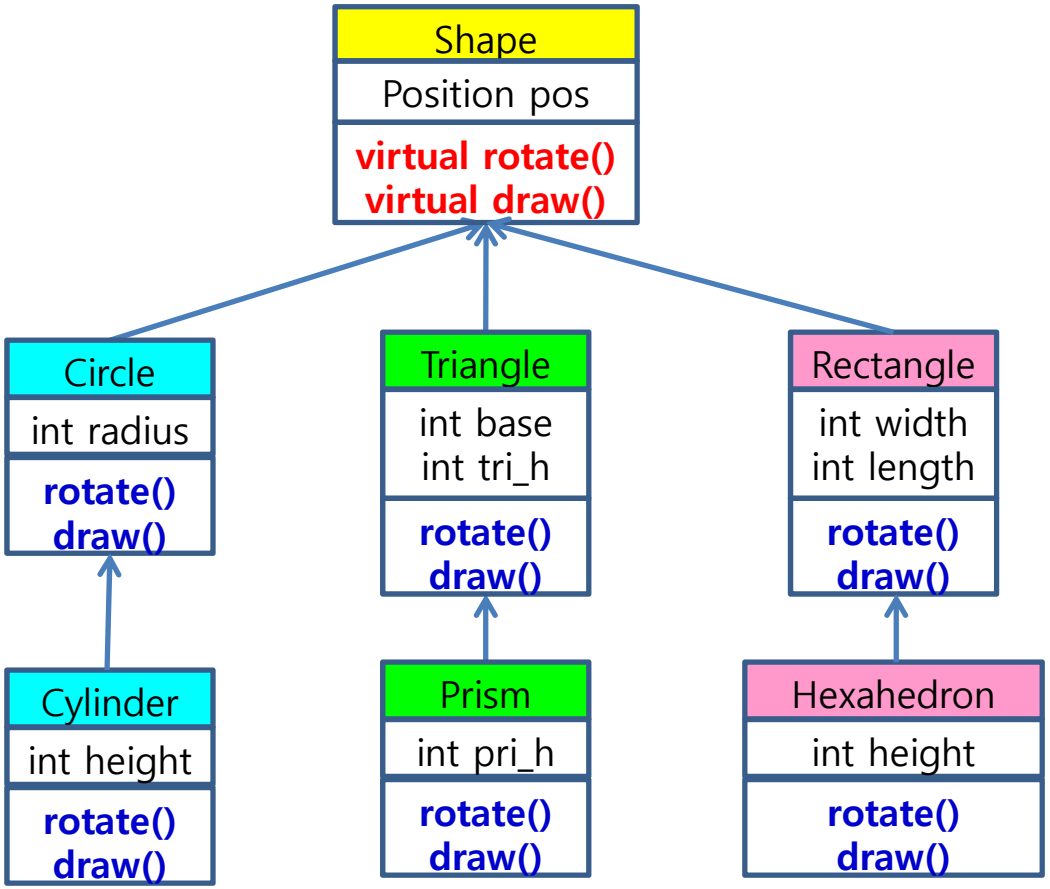
다형성 (Polymorphism)과 가상함수 (Virtual Function)

다형성 (Polymorphism)

- ◆Polymorphism (同質異像)
- ◆가상함수 (Virtual Function)
  - 지연 바인딩 (Late binding)
    - 프로그램 실행 단계에서 실제 실행될 함수가 연결됨
  - 가상함수의 구현 및 활용
  - 추상화 클래스와 순수 가상함수
- ◆포인터와 가상함수 (Virtual Function)
  - 확장된 상호 운용성 (type compatibility)
  - Downcasting vs. upcasting
  - C++ "under the hood" with virtual functions

다형성과 유사한 객체들의 관리 용이성

◆ 동일한 부모 클래스로부터 파생된 자손 클래스 객체들을 하나의 체계로 관리 (1)

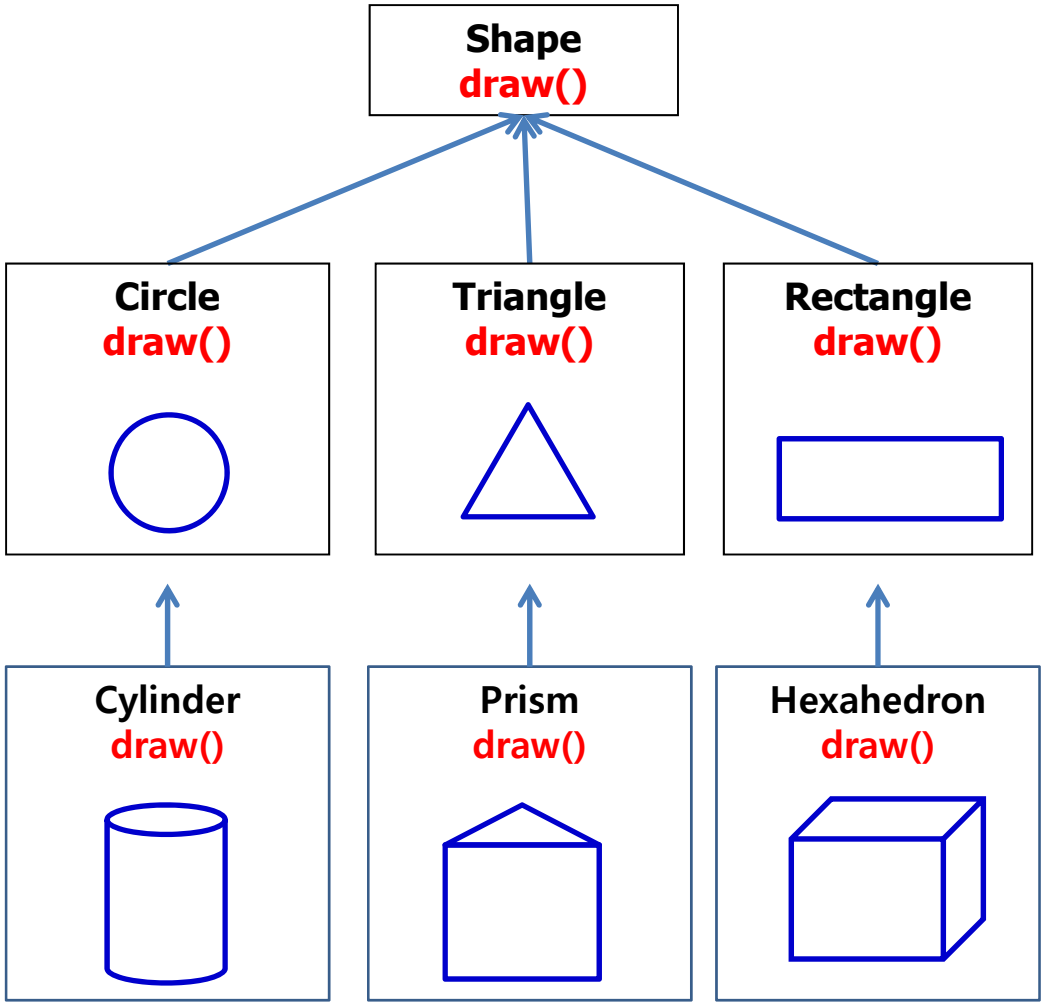


◆ 동일한 부모 클래스로부터 파생된 자손 클래스 객체들을 하나의 체계로 관리 (2)

```
main()
{
   . . . . . .
   Shape  *shapes[3];
   Rectangle  rec(1, 2, 3, 4);
   Circle   cl(4, 5, 6);
   Hexahedron  hexhdrn(0, 0, 7, 8, 9)

   shapes[0] = &rec; // up-casting
   shapes[1] = &cl; // up-casting
   shapes[2] = &hexhdrn; // up-casting
   for (int i = 0; i< 3; i++) {
      shapes[i] -> draw();
      cout << "area is : " << shapes[i] -> area() << endl;
   }
   . . . .
}
```

다형성 (Polymorphism)

- **Class Shape, Circle, Triangle, Rectangle, Cylinder, Prism, Hexahedron의 다층 상속 관계에서 draw() 멤버함수**
  - each class draws its own shape, such as "circle", "triangle", "rectangle", "cylinder", "prism" , and "hexahedron"



상속관계 클래스의 멤버함수 재정의 (Member Function Re-definition)

```
class Shape
{
 public:
    void draw() { cout << "Shape::draw()"; }
}

class Circle : public Shape
{
 public:
    void draw() { cout << "Circle::draw()"; } // re-definition
}

class Cylinder : public Circle
{
 public:
    void draw() { cout << "Cylinder::draw()"; } // re-definition
}
```

다형성과 가상함수 (Virtual Function)

◆ 다형성 (Polymorphism)

- 동일한 이름을 가지는 하나의 함수가 적용되는 객체에 따라 다른 의미 (기능)을 가질 수 있게 함
- 가상함수를 사용하여 다형성을 구현
- 객체지향 프로그래밍의 기본 기능 중 하나로 제공

◆가상 함수 (Virtual Function)

- 실제 실행되는 기능이 결정되기 이전에 해당 함수의 사용을 프로그램에 구성할 수 있음
- 프로그램 소스에 포함된 가상함수가 실제 어떤 함수로 실행될 것인지는 나중에 (실행단계에서) 결정됨 (late binding)

왜 다형성이 필요한가 ?

◆ 동일한 기반 클래스로부터 상속받아 생성된 파생클래스의 객체들을 하나의 체계로 쉽게 관리

- 하나의 부모 클래스로 부터 상속받아 생성되는 다양한 파생클래스들을 부모 (조상) 클래스의 동일한 멤버함수를 사용하여 쉽게 관리할 수 있도록 구성
- E.g.)
  - 부모 클래스: Shape, Shape::draw();
  - 파생 클래스: Circle, Rectangle, Triangle, Cylinder, Hexahedron, Prism
    - class Shape으로 부터 파생된 모든 클래스들은 가상함수 "draw()"를 구현
    - 모든 파생 클래스들은 부모 클래스 (Shape)의 "draw()" 함수를 사용하여 관리되도록 for-loop을 구성하며, 부모 클래스의 draw() 함수를 통하여 파생 클래스의 draw() 함수가 실행되도록 구성함
    - 부모 클래스 (Shape)의 draw() 함수에 연결된 파생 클래스의 draw() 함수는 각 파생클래스의 속성에 따라 다른 행동을 할 수 있도록 dynamic/late binding 됨

class Shape의 예 (1)

◆다양한 도형의 공통적인 속성을 가지는 class Shape의 설계 및 이를 상속하는 파생클래스 설계

- 다양한 도형의 예: Rectangles, circles, triangles, etc.
- 각 도형은 서로다른 속성을 가짐
  - Rectangle data: height, width, center point
  - Circle data: center point, radius
- 다양한 도형들은 하나의 기반(부모) 클래스 class Shape으로 부터 상속받을 수 있도록 설계
- 다양한 도형을 하나의 체계로 관리할 수 있도록 draw()를 구현하며, 실제 파생 클래스의 도형을 그릴 때에는 해당 파생클래스의 draw() 함수가 되도록 late binding

class Shape의 예 (2)

- 각 파생클래스 별로 별도의 ***draw()*** 함수가 구현됨
- 각 파생클래스에 동일한 함수 이름 **draw()**을 사용

```
Rectangle r;
Circle c;
r.draw();  //Calls Rectangle class's draw
c.draw(); //Calls Circle class's draw
```

- 각 도형의 **base class**인 **class Shape**의 **draw()** 멤버함수를 사용하여 다양하게 생성된 파생 클래스의 도형그리기 함수 **draw()**가 실행되도록 하는 방법은 ?

class Shape의 예 (3) : rotate()

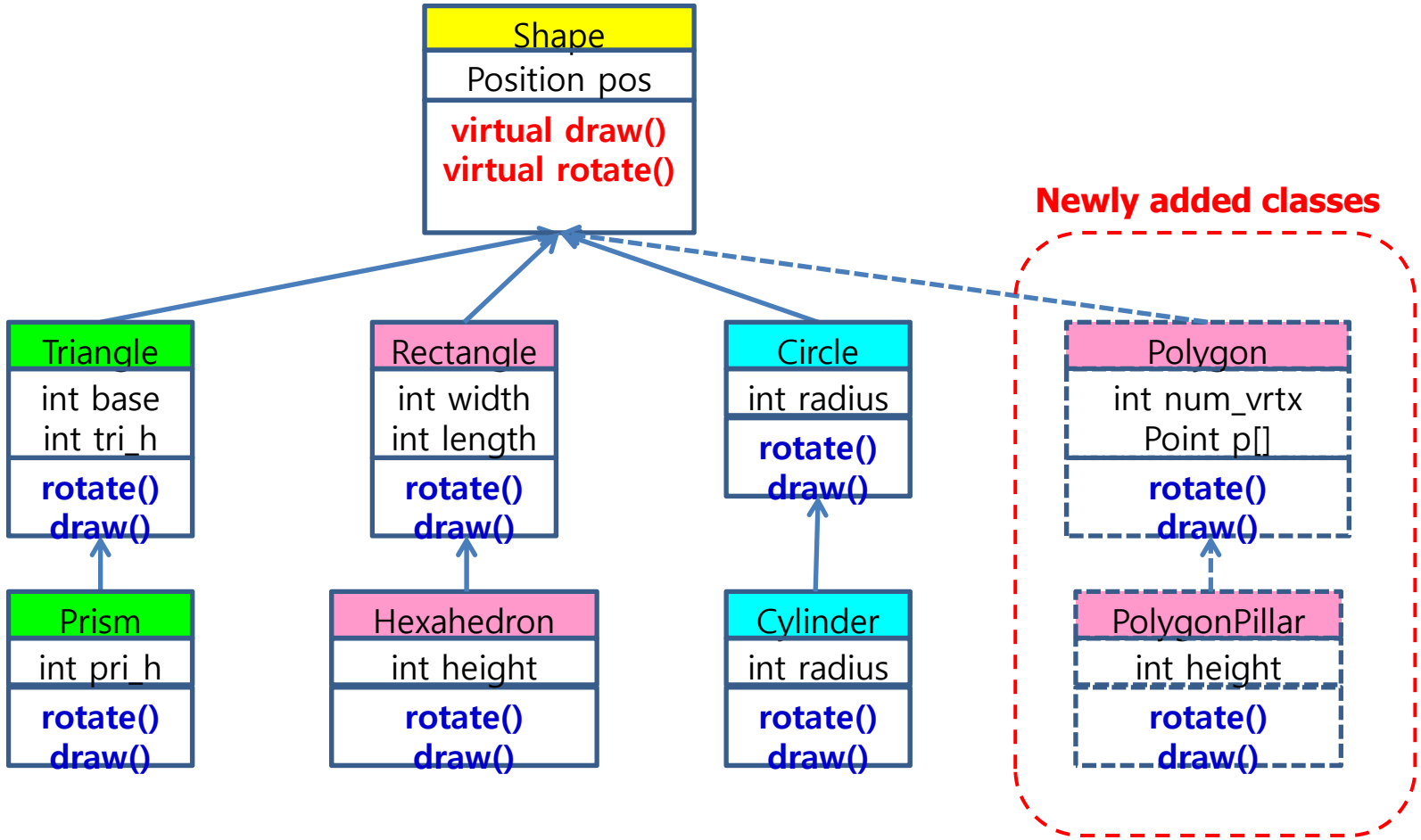
◆**Base class인 class Shape에는 다양한 파생 클래스들에 공통적으로 사용되는 기능을 함께 포함**

(예) rotate(): 도형을 지정된 각도로 회전시켜주는 함수

- 현재 표시된 도형을 지운 후, 변경된 각도로 다시 그려줌
- 따라서 Shape::**rotate()** 멤버함수는 해당 도형을 다시 그리기 위하여 **draw()** 멤버함수를 사용하게 됨
- 컴파일 단계에서 :
  - 어떤 클래스의 **draw()** 함수가 실행되어야 하나? : class rectangle 또는 circle

다형성과 시스템 확장성

◆ 기존의 상속관계에서 함께 개발된 파생 클래스이외에 나중에 추가되는 파생 클래스 들도 동일한 체계로 관리



class Shape의 예 (4) : 새로운 도형을 추가

- **class Shape을 상속하여 새로운 클래스 class Polygon을 추가**
  - **class Polygon** 는 class Shape으로 부터 상속 받아 생성되는 파생 클래스
- **class Shape으로 부터 파생된 클래스들은 class Shape의 rotate() 함수를 상속**
  - 다양하게 파생된 도형들의 class는 draw()를 사용하며, 각 도형에 따라 다른 기능을 가짐
  - 클래스 이름을 함께 표시하면 (예: Shape::draw()) 해당 클래스의 기능이 실행됨
- **class Polygon은 상속된 함수 rotate() 를 사용할 때 Shape::draw() 함수 대신 Polygon::draw()가 실행되도록 구성**

class Shape의 예 (5) : 가상함수

- **class Shape**이 설계 및 구현될 때에는 **class Polygon**의 존재를 모르며, **class Polygon**의 멤버 함수 **Polygon::draw()**가 아직 구현되지 않은 상태임
- 가상함수 **(Virtual function)**를 사용하여 아직 구현되지 않은 파생 클래스의 멤버 함수로 연결될 수 있도록 구성
- ***Late binding*** 또는 ***dynamic binding***
  - 컴파일러에게 실제 실행될 함수가 어떻게 구현되어 있는지 아직 모르며, 실제 프로그램이 실행될 때 까지 기다리도록 하며, 실제 프로그램이 실행될 때 결정되도록 프로그램을 구성
  - 가상 함수로 선언하여 late binding (dynamic binding)이 실행되도록 구성

순수 가상 함수 (Pure Virtual Function)

- **기반 클래스 (base class)는 일부 멤버 함수를 구체적으로 구현하지 않고, 상속받은 클래스의 멤버함수를 연결할 수 있는 기능만 제공하도록 구현**
- **class Shape으로 부터 상속받아 생성되는 다양한 도형 클래스들에서**
  - 다양한 파생클래스의 객체들 : (예) Rectangles, circles, triangles, etc.
  - class Shape의 draw() 함수는 어떻게 특정 도형을 그릴지 모르는 상태임
- **class Shape의 멤버함수를 순수 가상 함수로 선언**

```
virtual void draw() = 0;
```

추상적 기반 클래스 (Abstract Base Class)

- **기반클래스의 순수 가상 함수는 구체적으로 구현되지 않음**
  - class Shape을 상속받아 생성되는 파생 클래스들은 자신의 도형에 적합한 멤버함수를 구현
- **추상적 기반 클래스 (abstract base class)란?**
  - 하나이상의 순수 가상 함수를 가지는 기반 클래스
  - 추상적 기반 클래스에는 구체적으로 구현되지 않은 멤버함수가 포함되어 있음
  - 단순히 기반 클래스로만 사용되며, 실제 객체 인스탄스를 생성하지는 않음
- **만약 파생 클래스에서 기반 클래스의 순수 가상 함수를 모두 구현하지 않으면, 그 파생 클래스도 추상적 기반 클래스로만 사용됨**

상속관계 클래스 인스탄스간의 upcasting/downcasting과 가상함수의 내부 동작

포인터와 가상함수

◆ 파생 클래스 객체와 기반 클래스 객체간의 자료형 호환성

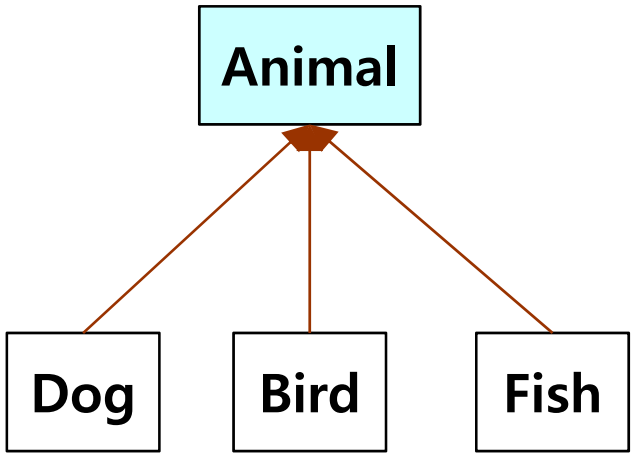
- 기반 클래스로 부터 상속받아 파생된 클래스는 "is a" 관계가 있으며, 파생클래스들은 내부에 기반 클래스의 속성을 함께 가지고 있음
- 파생 클래스 객체를 기반 클래스 객체에 대입시킬 수 있음
- 단, 기반클래스를 파생클래스에 대입시킬 수는 없음

◆ "is a" 관계에 따른 자료형 호환성

- is-a 관계가 있는 상속 체계에서 파생클래스의 객체를 기반클래스의 객체로 대입하는 것은 가능
- (예) **Dog "is a"** an **Animal** 관계가 있으므로, Dog 객체를 Animal 객체로 대입하는 것은 가능함;
  그러나 Animal 객체를 Dog 객체에 대입하는 것은 가능하지 않음
  (정의가 되지 않은 속성이 존재하게 됨)

상속관계에서의 자료형 호환성 문제

```
class Animal
{
  private:
    string type;
  public:
    virtual void getBreed() const;
};

class Dog : public Animal
{
   private:
    string breed;
   public:
    virtual void getBreed() const;
};
```



Class Animal and Class Dog

◆ **Now given declarations:**
class Dog : class Animal; // Dog inherits from Animal
Animal vAnimal;
Dog vDog;

◆ **Anything that "is a" Dog "is a" Animal:**

- vDog.name = "Tiny";
  vDog.breed = "Great Dane";
  vAnimal = vDog; // **upcasting**
- upcasting은 가능함

◆ 파생클래스 객체를 기반클래스 객체로 대입하는 **upcasting**은 가능하나, 반대 방향의 대입은 가능하지 않음

- Animal 객체 "is not always a" Dog 객체 (Animal 객체 중에서는 Dog 객체가 아닌 (예: Cat) 경우가 존재함)
- 파생된 클래스 (class **Dog**)는 기반 클래스 (class Animal)에 추가되는 속성 (예: breed)을 가질 수 있음

Upcasting과 Downcasting

- **Downcasting (기반클래스 객체를 파생 클래스 객체에 대입하는 경우)**

```
class Dog : class Animal;
Animal vAnimal;
Dog vDog;
...
vDog = static_cast<Dog>(vAnimal);  //ILLEGAL !!
```

Dog에는 Animal에 정의되어 있지 않은 정보도 포함되어 있으므로, down casting에서는 일부 정보가 정의되지 않은 채로 남아 있게 된다.
(예: Dog에는 breed 정보가 정의되어 있지만, Animal에는 breed 정보가 없으므로, 위 assignment에서는 breed 정보가 정의되어 있지 않은 상태로 남아 있게 된다.)

- **Upcasting (파생 클래스 객체를 기반 클래스 객체에 대입하는 경우)**

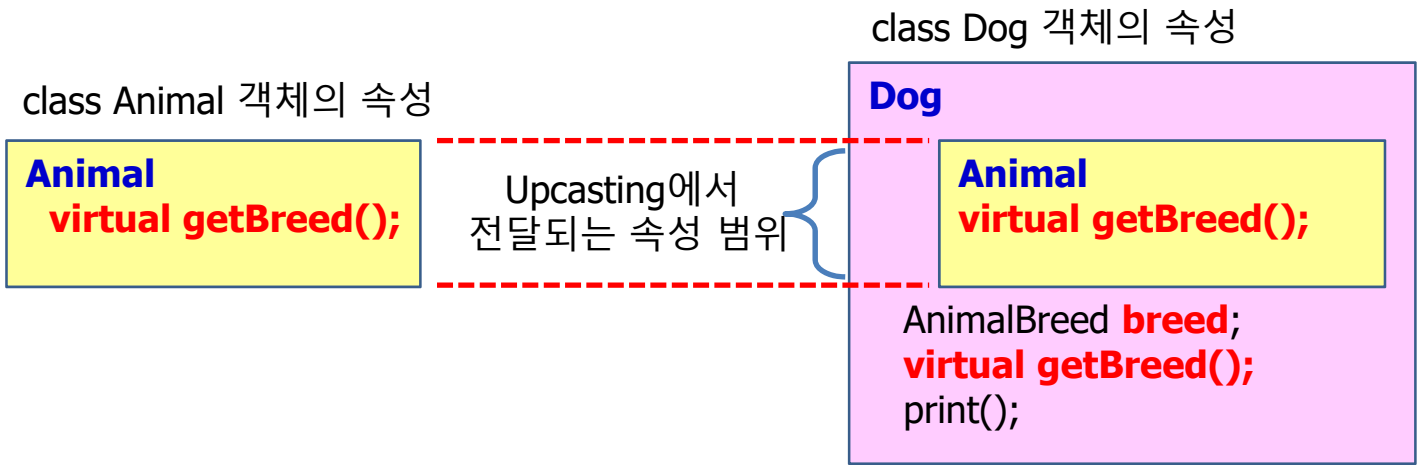
```
vAnimal = vDog;    // Legal !!
vAnimal = static_cast<Animal>(vDog);  // Also legal !!
```

Upcasting에서는 파생클래스가 기반클래스의 "is a"관계에 있고, 모든 정보를 포함하고 있다.

Upcasting에서의 슬라이싱 (Slicing) 문제 (1)

◆ Slicing 문제

- 앞의 예제에서 **vAnimal** 은 "is a" Great Dane이며, 이름이 Tiny임
- vAnimal이 Animal의 객체로 처리되고 있으나, 가상함수 getBreed()를 사용하여 breed를 가져올 수 있도록 구성



Upcastig에서의 슬라이싱 (Slicing) 문제 (2)

- **vAnimal**으로 대입할 때 **breed** 정보는 생략됨
  - **cout << vAnimal.breed; // Illegal !!**
    - ERROR가 발생됨
  - slicing problem이라고 함

◆Upcasting에서의 Slicing 문제

- Dog 객체가 Animal 객체로 대입되었기 때문에 Animal 클래스가 정의한 속성만 포함될 수 있게 되며, Animal 객체로 대입되면서 Dog 객체로서 가졌던 모든 속성을 옮겨올 수 없게 됨

Upcastig에서의 슬라이싱 (Slicing) 문제 (3)

◆ **class Dog : class Animal;**

```
Animal *pAnimal;
Dog *pDog;
pDog = new Dog;
pDog->name = "Tiny";
pDog->breed = "Great Dane";
pAnimal = pDog;
```

◆ **pAnimal** 포인터를 사용하여 ***pDog** 객체가 가지는 모든 속성을 접근할 수 없음**:**

```
cout << pAnimal->breed;      // ILLEGAL !!
cout << pAnimal->getBreed(); // Legal !!
```

//breed는 class Animal의 속성이 아님

//base class 포인터를 사용하여 파생 클래스 객체의 data member를 직접 접근/변경할 수 는 없음

//base class의 virtual member function과 late binding을 통하여 파생 클래스 객체의 data member 접근/변경 가능

Upcastig에서의 슬라이싱 (Slicing) 문제 (4)

- **기반클래스 (base class)의 포인터를 사용하여 파생 클래스의 가상 멤버 함수를 호출**
  - **virtual function**으로 구현된 기능을 **polymorphism**을 사용하여 접근
- **가상 멤버 함수를 사용하여야 함**
  - **pAnimal->getBreed();**
  - class Animal의 getBreed() 가상함수가 virtual로 선언되어 있기 때문에 pAnimal = pDog; 이 실행된 이후에는 class Dog의 **getBreed()** 함수로 연결됨!
  - C++ 프로그램 **pAnimal** 포인터는 "**late binding**"으로 어떤 객체에 연결되는지 실행 시점까지 대기

가상함수의 내부 동작

- **실제 가상함수가 어떻게 구현되어 있는지에 대하여 알 필요가 없음**
  - 정보 은닉 (**information hiding)**에 해당됨
- **가상함수 테이블 (Virtual function table)**
  - 컴파일러가 가상함수 테이블을 만들어 가상함수들의 binding을 관리
  - 각 가상함수에는 포인터가 할당되며, 해당 가상 함수가 실행될 때 실제 실행되는 함수를 가리키게 됨
- **가상 함수를 포함하는 객체는 가상함수 테이블을 가리키는 포인터를 가짐**
  - 객체의 가상함수가 실행될 때 실제 실행되는 함수를 쉽게 찾을 수 있도록 관리

다형성 구현을 위한 가상 함수 설정 및 구현 절차

절차	예

(1) 기반클래스에서 가상함수 선언	class Shape
	{
	public:
	  virtual void draw(); // class Shape의 가상 함수 선언
	}

(2) 파생 클래스에서 가상 함수 구현	class Triangle: public Shape
	{
	public:
	  virtual void draw(); // class Triangle의 draw() 함수 정의
	}
	void Triangle::draw() // class Triangle의 draw() 함수 구현
	{ . . . . }

(3) 컴파일러가 가상함수 테이블 생성	

(4) 기반클래스 포인터에 파생클래스 객체 주소 설정 → 가상 함수 테이블 update (실제 실행될 함수 지정)	Shape *pShape; // class Shape 포인터 선언
	Triangle tri(. . . .); // class Triangle 객체 생성
	pShape = &tri; // 기반 클래스 포인터 값으로
	// 파생 클래스 객체 주소 설정 (upcasting)

(5) 기반클래스 포인터를 사용하여 기반클래스 가상함수 실행 요청	pShape→draw(); // → 실제로 파생클래스의 해당 함수 실행

클래스 소멸자 (Destructor)를 가상함수로 지정

◆객체가 소멸될 때 동적으로 할당되어 사용하였던 자원을 반납하여야 함

◆클래스 소멸자를 가상함수로 지정

```
Base *pBase = new Derived;
...
delete pBase;
```

- 기반 클래스 객체의 포인터에 파생 클래스 객체 포인터를 대입시키고, 기반클래스 포인터를 통하여 파생 클래스의 소멸자를 호출할 수 있도록 구성
- 클래스 소멸자를 가상함수로 지정함으로써 가능

◆모든 클래스의 소멸자를 가상함수로 설정/구현하는 것이 좋은 정책임

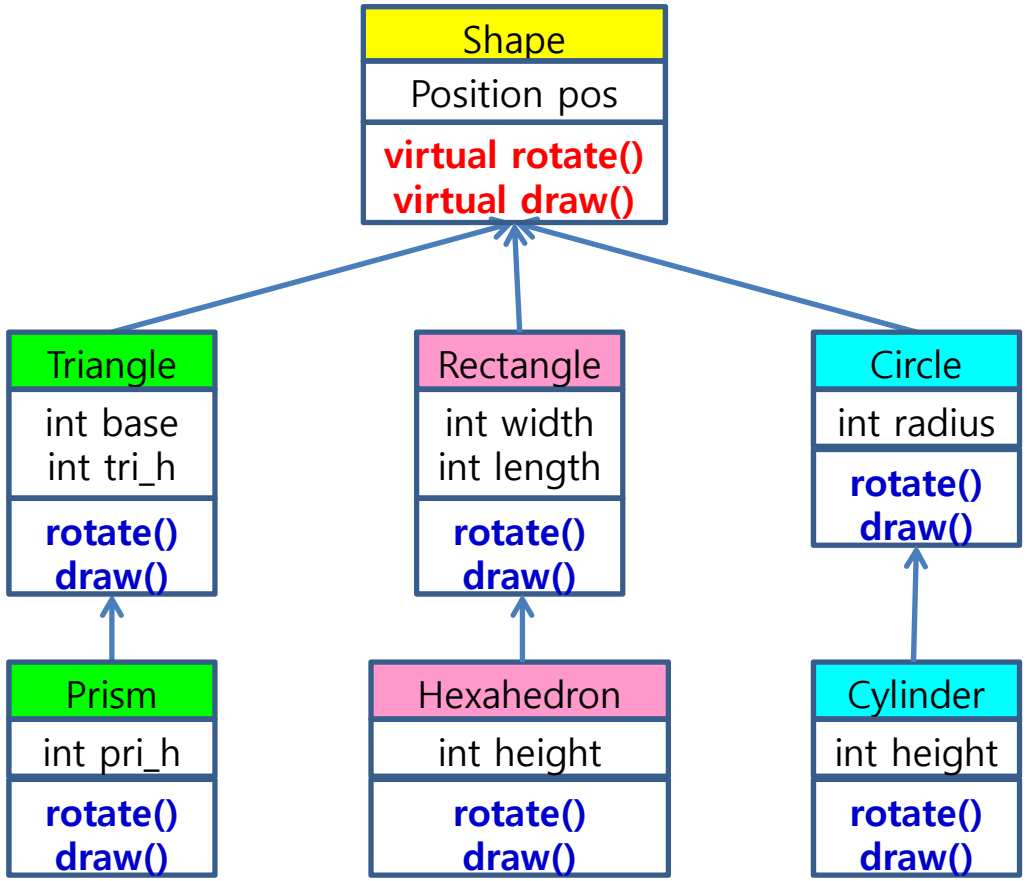
가상함수의 단점

- ◆가상함수는 다형성을 구현할 수 있도록 하는 장점이 있음
- ◆그러나, 중요한 단점 (major disadvantage)도 있음
  - 가상함수 테이블을 구성하고, 사용하여야 하므로 메모리가 추가로 사용됨
  - 가상함수가 실행될 때, 실제 실행하는 함수를 가상 함수 테이블을 사용하여 찾고, 연결시키야 하기 때문에 추가 지연이 발생됨
- ◆따라서, 반드시 필요한 경우에만 가상 함수를 사용하도록 하여야 함

다형성 (Polymorphism)과 가상함수 (Virtual Function) 기반의 그래픽 응용 프로그램

가상함수와 다형성 기반의 픽셀단위 도형 그리기

◆ 상속 체계도



가상함수와 다형성 기반의 픽셀단위 도형 그리기

◆ Windows 환경에서의 픽셀단위 도형 출력을 위한 함수

구분	함수 및 인수	기능 설명
환경 설정	GetConsoleWindow()	get Console Window
	GetDC()	get device context
	SetBkColor(hdc, color)	배경 (back ground) 색깔 설정
	SetTextColor(hdc, color)	문자 색깔 설정
픽셀단위 위치 이동	MoveToEx(hdc, x, y, NULL)	지정된 좌표 (x, y)로 이동
펜 관리	CreatePen()	펜을 생성
브러시 관리	CreateSolidBrush()	도형 채우기에 사용되는 브러시 생성
픽셀단위 색깔 설정	SetPixel(hdc, x, y, color);	지정된 좌표 (x, y)의 픽셀을 지정된 색깔로 표시
도형 그리기	LineTo(hdc, x, y)	현재 픽셀 위치로부터 지정된 좌표까지 선 그리기
	Ellipse(hdc, x1, y1, x2, y2);	타원형 그리기
	Polygon(hdc, p, n)	POINT p[num_points]로 지정된 꼭지점을 연결하는 다각형 그리기
문자 출력	TextOut()	문자열 출력

```
/** Color.h */

#ifndef COLOR_H
#define COLOR_H
#include <Windows.h>
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

// COLORREF is defined in <Windows.h>
// The COLORREF value is used to specify an RGB color,
//     in hexadecimal form of 0x00bbggrr
const COLORREF RGB_BLACK   = 0x00000000;
const COLORREF RGB_RED     = 0x000000FF;
const COLORREF RGB_GREEN   = 0x0000FF00;
const COLORREF RGB_BLUE    = 0x00FF0000;
const COLORREF RGB_ORANGE  = 0x0000A5FF;
const COLORREF RGB_YELLOW  = 0x0000FFFF;
const COLORREF RGB_MAGENTA = 0x00FF00FF;
const COLORREF RGB_WHITE   = 0x00FFFFFF;
ostream& printRGB(ostream& ostr, const COLORREF color);
// RGB color code chart: https://www.rapidtables.com/web/color/RGB_Color.html
/* Note: RGB(red, green, blue) macro also provides COLORREF data
   . RGB(FF, 00, 00) => 0x000000FF (RGB_RED)
   . RGB(00, FF, 00) => 0x0000FF00 (RGB_GREEN)
   . RGB(00, 00, FF) => 0x00FF0000 (RGB_BLUE)
*/
#endif
```

Color.cpp

```
/** Color.cpp */
#include <iostream>
#include <Windows.h>
#include "Color.h"

ostream& fprintRGB(ostream& fout, const COLORREF color)
{
    unsigned int red, green, blue;

    red = (color & 0x000000FF);
    green = (color & 0x0000FF00) >> 8;
    blue = (color & 0x00FF0000) >> 16;
    fout << "RGB(" << setw(2) << hex << uppercase << internal << setfill('0') << red;
    fout << "," << setw(2) << hex << uppercase << internal << setfill('0') << green;
    fout << "," << setw(2) << hex << uppercase << internal << setfill('0') << blue;
    fout << ")";

    return fout;
}
```

Class Shape

```
/* Shape.h (1) */
#ifndef SHAPE_H
#define SHAPE_H
#include <string>
#include <Windows.h>
#include <conio.h>
#include "ConsolePixelDrawing.h"
#include "Color.h"
using namespace std;
#define PI 3.14159

class ConsolePixelFrame;

class Shape
{
        friend ostream&
           operator<<(ostream &, const Shape &);
public:
        Shape(); // default constructor
        Shape(string name);
        Shape(int px, int py, double angle,
           COLORREF ln_clr, COLORREF br_clr,
           int pen_thick, string name);
           // constructor
        virtual ~Shape();
```

```
/** Shape.h  (2) */

        virtual void
         draw(ConsolePixelFrame cp_frame);
        virtual void draw();
          // used for testing of late binding
        void print(ostream &);
        double get_pos_x() const { return pos_x; }
        double get_pos_y() const { return pos_y; }
        void set_pos_x(int x) { pos_x = x; }
        void set_pos_y(int y) { pos_y = y; }
        string getName() { return name; }
        void setName(string n) { name = n; }
        Shape& operator=(const Shape& s);
protected:
        double pos_x; // position x
        double pos_y; // position y
        double angle;
        string name;
        int pen_thickness;
        COLORREF line_color;
        COLORREF brush_color;
};

#endif
```

```
/**   Shape.cpp   (1) */
#include <iostream>
#include "Shape.h"
#include <iomanip>

using namespace std;

Shape::Shape()  // default constructor
{
   pos_x = pos_y = 0;
   angle = 0;
   line_color = brush_color = RGB_BLACK;
   name = "no_name";

   //cout << "Shape::Default constructor (" << name
   << ").\n";
}

Shape::Shape(string n)
:name(n)
{
   pos_x = pos_y = 0;
   angle = 0;
   line_color = brush_color = RGB_BLACK;
   //cout << "Shape::Constructor (" << name << ").\n";
}
```

```
/**   Shape.cpp   (2) */
Shape::Shape(int px, int py, double ang,
COLORREF ln_clr, COLORREF br_clr, int
pen_thick, string n)
{
   pos_x = px;
   pos_y = py;
   angle = ang;
   line_color = ln_clr;
   brush_color = br_clr;
   pen_thickness = pen_thick;
   name = n;
   //cout << "Shape::Constructor (" << name << ").\n";
}

Shape::~Shape()
{
   //cout << "Shape::Destructor (" << name << ").\n";
}

void Shape::draw()
{
   cout << "draw() of “ << name << endl;
}
```

```
/**   Shape.cpp   (3) */

void Shape::draw(ConsolePixelFrame cp_frame)
{
    /* virtual function that will be late-binded to
    sub-class's draw() */
}

Shape& Shape::operator=(const Shape& s)
{
    pos_x = s.pos_x;
    pos_y = s.pos_y;
    angle = s.angle;
    name = s.name;
    line_color = s.line_color;
    brush_color = s.brush_color;

    return *this;
}
```

```
/**   Shape.cpp   (3) */

ostream& operator<<(ostream &ostr, Shape &shp)
{
    ostr << shp.name << ": pos("
    << shp.get_pos_x() << ", " << shp.get_pos_y()
    << ")";
    ostr << ", line_color(";
    printRGB(ostr, shp.line_color);
    ostr << "), brush_color(";
    printRGB(ostr, shp.brush_color);
    ostr << ")";
    return ostr;
}

void Shape::print(ostream &ostr)
{
    ostr << name << ": pos(" << get_pos_x() << ",
    " << get_pos_y() << ")";
    ostr << ", line_color(";
    printRGB(ostr, line_color);
    ostr << "), brush_color(";
    printRGB(ostr, brush_color);
    ostr << ")";
}
```

class Circle

```
/** Circle.h (1) */
#ifndef Circle_H
#define Circle_H
#include <string>
#include "Shape.h"
using namespace std;

#define PI 3.14159
class Circle : public Shape
{
    friend ostream&
      operator<<(ostream &, const Circle &);
public:
    Circle();
    Circle(string name);
    Circle(int px, int py, int r, double ang,
    COLORREF ln_clr, COLORREF br_clr, int
    pen_thick, string name);
    //Circle(Circle &tr);
    ~Circle();
    double getArea();
    virtual void draw(ConsolePixelFrame
      cp_frame);
```

```
/** Circle.h (2) */

    void print(ostream &);
    int getRadius() const { return radius; }
    void setRadius(int r) { radius = r; }
    Circle& operator=(const Circle& cir);
protected:
    int radius;
};
#endif
```

```
void Circle::draw(ConsolePixelFrame cp_frame)
{
    HPEN new_pen, old_pen;
    HBRUSH new_brush, old_brush;
    HDC hdc;
    int pos_center_x, pos_center_y;
    pos_center_x = cp_frame.get_pos_org_x() + get_pos_x();
    pos_center_y = cp_frame.get_pos_org_y() + get_pos_y();

    hdc = cp_frame.getConsole_DC();
    new_pen = CreatePen(PS_SOLID, pen_thickness, line_color);
    old_pen = (HPEN)SelectObject(hdc, new_pen);
    new_brush = CreateSolidBrush(brush_color);
    old_brush = (HBRUSH)SelectObject(hdc, new_brush);
    Ellipse(hdc, pos_center_x - radius, pos_center_y - radius,
        pos_center_x + radius, pos_center_y + radius);
    SelectObject(hdc, old_pen);
    DeleteObject(new_pen);

    SelectObject(hdc, old_brush);
    DeleteObject(new_brush);
}
```

Class Triangle

```
/** Triangle.h (1) */

#ifndef TRIANGLE_H
#define TRIANGLE_H
#include <string>
#include "ConsolePixelDrawing.h"
#include "Shape.h"
using namespace std;

class Triangle : public Shape
{
    //friend ostream& operator<<(ostream &,
    //   const Triangle &);
public:
    Triangle();
    Triangle(string name);
    Triangle(int px, int py, int b, int h, double ang,
        COLORREF ln_clr, COLORREF br_clr, int
        pen_thick, string name);
    //Triangle(Triangle &tr);
    ~Triangle();
    double getArea();
    virtual void draw(ConsolePixelFrame
        cp_frame);
```

```
/** Triangle.h (2) */

    void print(ostream &);
    int getBase() { return base; }
    int getHeight() { return tri_height; }
    Triangle& operator=(const Triangle& tri);
protected:
    int base;
    int tri_height;
};

#endif
```

```
void Triangle::draw(ConsolePixelFrame cp_frame)
{
    HDC hdc;
    HPEN new_pen, old_pen;
    HBRUSH new_brush, old_brush;
    int pos_center_x, pos_center_y;
    pos_center_x = cp_frame.get_pos_org_x() + get_pos_x();
    pos_center_y = cp_frame.get_pos_org_y() + get_pos_y();

    POINT p[3];
    p[0].x = pos_center_x - base / 2;
    p[0].y = pos_center_y + tri_height * 1.0 / 2.0;
    p[1].x = pos_center_x + base / 2;
    p[1].y = pos_center_y + tri_height * 1.0 / 2.0;
    p[2].x = pos_center_x;
    p[2].y = pos_center_y - tri_height * 1.0 / 2.0;
    hdc = cp_frame.getConsole_DC();
    new_pen = CreatePen(PS_SOLID, pen_thickness, line_color);
    old_pen = (HPEN)SelectObject(hdc, new_pen);
    new_brush = CreateSolidBrush(brush_color);
    old_brush = (HBRUSH)SelectObject(hdc, new_brush);

    Polygon(hdc, p, 3);
    SelectObject(hdc, old_pen);
    DeleteObject(new_pen);
    SelectObject(hdc, old_brush);
    DeleteObject(new_brush);
}
```

Class Rectang

```
/** Rectang.h (1) */
#ifndef Rectangle_H
#define Rectangle_H
#include <string>
#include "ConsolePixelDrawing.h"
#include "Shape.h"
using namespace std;

class Rectang : public Shape
{
   //friend ostream& operator<<(ostream &,
     // const Rectangle &);
public:
   Rectang();
   Rectang(string name);
   Rectang(int px, int py, int w, int l, double ang,
      COLORREF ln_clr, COLORREF br_clr, int
      pen_thick, string name);
   //Rectangle(Rectangle &tr);
   ~Rectang();
   double getArea();
   virtual void draw(ConsolePixelFrame
      cp_frame);
```

```
/** Rectang.h (2) */
   void print(ostream &);
   int getWidth() { return width; }
   int getLength() { return length; }
   Rectang& operator=(const Rectang& rec);

protected:
   int width;
   int length;
};

#endif
```

```
void Rectang::draw(ConsolePixelFrame cp_frame)
{
    HDC hdc;
    HPEN new_pen, old_pen;
    HBRUSH new_brush, old_brush;
    int pos_center_x, pos_center_y;
    pos_center_x = cp_frame.get_pos_org_x() + get_pos_x();
    pos_center_y = cp_frame.get_pos_org_y() + get_pos_y();

    POINT p[4];
    p[0].x = pos_center_x - width / 2;
    p[0].y = pos_center_y - length / 2;
    p[1].x = pos_center_x + width / 2;
    p[1].y = pos_center_y - length / 2.0;
    p[2].x = pos_center_x + width / 2;
    p[2].y = pos_center_y + length / 2.0;
    p[3].x = pos_center_x - width / 2;
    p[3].y = pos_center_y + length / 2.0;

    hdc = cp_frame.getConsole_DC();
    new_pen = CreatePen(PS_SOLID, pen_thickness, line_color);
    old_pen = (HPEN)SelectObject(hdc, new_pen);
    new_brush = CreateSolidBrush(brush_color);
    old_brush = (HBRUSH)SelectObject(hdc, new_brush);
    Polygon(hdc, p, 4);
    SelectObject(hdc, old_pen);
    DeleteObject(new_pen);
    SelectObject(hdc, old_brush);
    DeleteObject(new_brush);
}
```

class Polygon

```
/** Polygon.h  (1) */

#ifndef PolyGon_H
#define PolyGon_H

#include <string>
#include "ConsolePixelDrawing.h"
#include "Shape.h"
using namespace std;

class PolyGon : public Shape
{
    //friend ostream& operator<<(ostream &,
    //   const PolyGonle &);
public:
    PolyGon();
    PolyGon(string name);
    PolyGon(int px, int py, int radius, int num_poly,
        double ang, COLORREF ln_clr, COLORREF br_clr,
        int pen_thick, string name);
    //PolyGonle(PolyGonle &pg);
    ~PolyGon();
    //double getArea();
    virtual void draw(ConsolePixelFrame
    cp_frame);
```

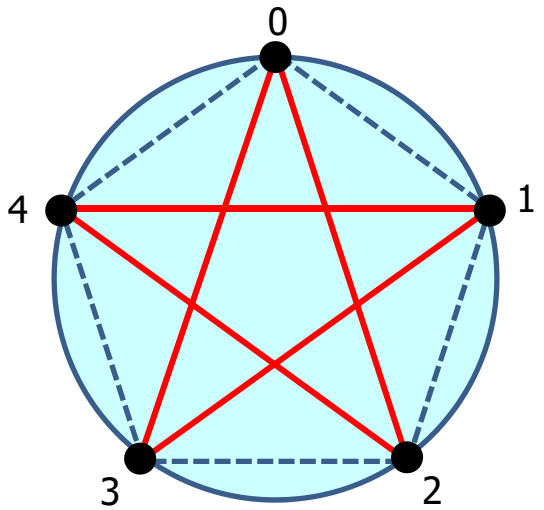
```
/** Polygon.h  (2) */

    void print(ostream &);
    int getRadius() { return radius; }
    int getNumPoly() { return num_poly; }
    PolyGon& operator=(const PolyGon& pg);

protected:
    int radius;
    int num_poly;
};

#endif
```

```
void PolyGon::draw(ConsolePixelFrame cp_frame)
{
    POINT *points = (POINT *)malloc(num_poly * sizeof(POINT));
    double rad_angle, delta_angle; // angle in radian
    int pos_center_x, pos_center_y;
    pos_center_x = cp_frame.get_pos_org_x() + get_pos_x();
    pos_center_y = cp_frame.get_pos_org_y() + get_pos_y();

    int x, y;
    HDC hdc;
    HPEN new_pen, old_pen;
    HBRUSH new_brush, old_brush;
    hdc = cp_frame.getConsole_DC();

    delta_angle = 2.0*PI / num_poly;
    rad_angle = PI / 2.0;
    for (int i = 0; i < num_poly; i++, rad_angle += delta_angle)
    {
        x = pos_center_x + radius * cos(rad_angle);
        y = pos_center_y - radius * sin(rad_angle);
        points[i].x = x;  points[i].y = y;
    }

    new_pen = CreatePen(PS_SOLID, pen_thickness, line_color);
    old_pen = (HPEN)SelectObject(hdc, new_pen);
    new_brush = CreateSolidBrush(brush_color);
    old_brush = (HBRUSH)SelectObject(hdc, new_brush);
    Polygon(hdc, points, num_poly);
    SelectObject(hdc, old_pen);
    DeleteObject(new_pen);
    SelectObject(hdc, old_brush);
    DeleteObject(new_brush);
}
```

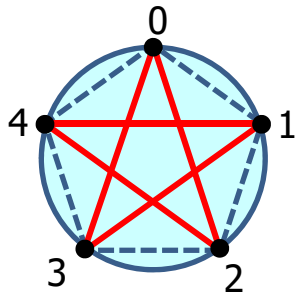
class Star

```
/* Star.h (1) */
#ifndef Star_H
#define Star_H

#include <string>
#include "ConsolePixelDrawing.h"
#include "Shape.h"
using namespace std;

class Star : public Shape
{
    //friend ostream& operator<<(ostream &,
    //   const PolyGonle &);
public:
    Star();
    Star(string name);
    Star(int px, int py, int radius, int num_vertices,
    double ang, COLORREF ln_clr, COLORREF br_clr,
    int pen_thick, string name);
    //PolyGonle(PolyGonle &pg);
    ~Star();
    //double getArea();
    virtual void draw(ConsolePixelFrame
        cp_frame);
```

```
/* Star.h (2) */

    void fprint(ostream &);
    int getRadius() { return radius; }
    int getNumPoly() { return num_vertices; }
    Star& operator=(const Star& pg);
protected:
    int radius;
    int num_vertices;
};

#endif
```



```
void Star::draw(ConsolePixelFrame cp_frame)
{
      POINT *points = (POINT *)malloc(5.0 *
         sizeof(POINT));
      double rad_angle, delta_angle;
         // angle in radian
      int pos_center_x, pos_center_y;
      pos_center_x = cp_frame.get_pos_org_x() +
         get_pos_x();
      pos_center_y = cp_frame.get_pos_org_y() +
         get_pos_y();

      int x, y;
      HDC hdc;
      HPEN new_pen, old_pen;
      HBRUSH new_brush, old_brush;
      hdc = cp_frame.getConsole_DC();

      delta_angle = 2.0*PI / 5.0;
      rad_angle = PI / 2.0;
      for (int i = 0; i < 5.0; i++, rad_angle +=
         delta_angle)
      {
            x = pos_center_x + radius * cos(rad_angle);
            y = pos_center_y - radius * sin(rad_angle);
            points[i].x = x;
            points[i].y = y;
      }
```




```
      new_pen = CreatePen(PS_SOLID,
          pen_thickness, line_color);
      old_pen = (HPEN)SelectObject(hdc,
          new_pen);
      new_brush = CreateSolidBrush(brush_color);
      old_brush = (HBRUSH)SelectObject(hdc,
          new_brush);

      //instead of Polygon(hdc, points, num_poly);
      MoveToEx(hdc, points[0].x, points[0].y,
          NULL);
      LineTo(hdc, points[2].x, points[2].y);
      LineTo(hdc, points[4].x, points[4].y);
      LineTo(hdc, points[1].x, points[1].y);
      LineTo(hdc, points[3].x, points[3].y);
      LineTo(hdc, points[0].x, points[0].y);

      SelectObject(hdc, old_pen);
      DeleteObject(new_pen);
      SelectObject(hdc, old_brush);
      DeleteObject(new_brush);
}
```

ConsolePixelFrame.h

```
/* ConsolePixelDrawing.h (1) */
#ifndef PIXEL_DRAWING_H
#define PIXEL_DRAWING_H

#include <iostream>
#include <string>
#include <Windows.h>
#include <conio.h>
#include "Shape.h"
#include "Color.h"

using namespace std;

/* PEN_Stypes */
#define PS_SOLID         0
#define PS_DASH          1       // -------
#define PS_DOT           2       // .......
#define PS_DASHDOT       3       // _._._._
#define PS_DASHDOTDOT    4       // _.._.._
#define PS_NULL          5
#define PS_INSIDEFRAME   6
#define PS_USERSTYLE     7
#define PS_ALTERNATE     8

#define MAX_NUM_SHAPES 100
```

```
/* ConsolePixelDrawing.h (2) */

class Shape;

class ConsolePixelFrame
{
public:
    ConsolePixelFrame(int org_x, int org_y);
    ~ConsolePixelFrame();
    void addShape(Shape* new_shape);
    void drawShapes();
    int get_pos_org_x() { return pos_org_x; }
    int get_pos_org_y() { return pos_org_y; }
    HDC getConsole_DC() { return console_DC; }
private:
    HWND console;
    HDC console_DC; // device context
    Shape **pShapes; // Array of Shape Pointers
    int num_shapes;
    int capacity;
    int pos_org_x;
    int pos_org_y;
    bool isValidIndex(int sub);
};
#endif
```

ConsolePixelFrame.cpp

```
/* ConsolePixelDrawing.cpp (1) */
#include "ConsolePixelDrawing.h"

ConsolePixelFrame::ConsolePixelFrame(int px, int py)
{
    console = GetConsoleWindow();
    console_DC = GetDC(console);
    pShapes = new Shape*[MAX_NUM_SHAPES];
    num_shapes = 0;
    capacity = MAX_NUM_SHAPES;

    pos_org_x = px;
    pos_org_y = py;
}

ConsolePixelFrame::~ConsolePixelFrame()
{
     //delete[] shapes;

     //ReleaseDC(console, console_DC);
}
```

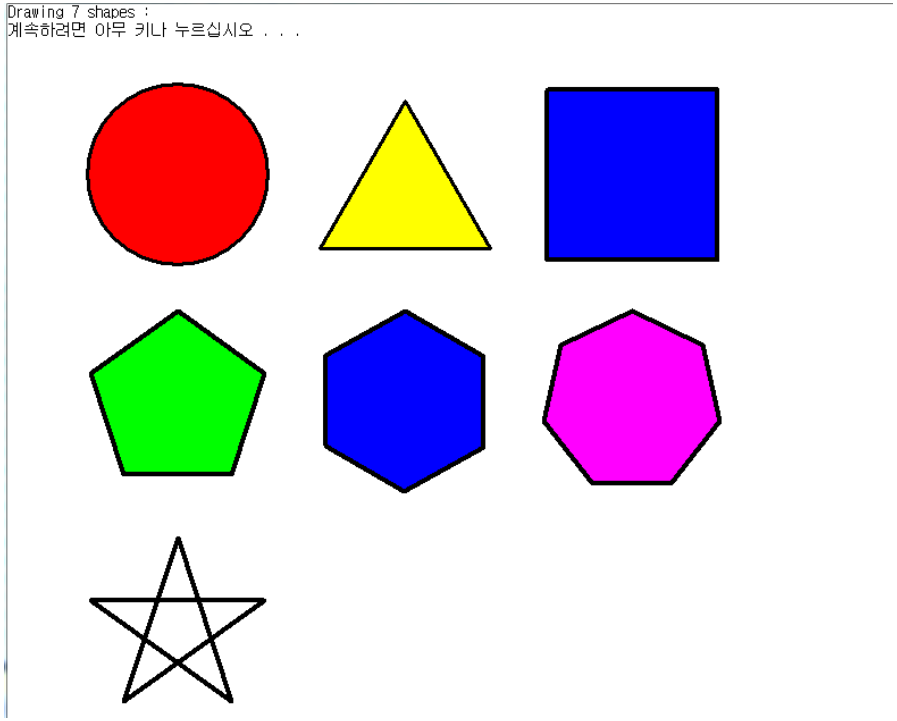
```
/* ConsolePixelDrawing.cpp (2) */

void ConsolePixelFrame::addShape(Shape *pNew_shape)
{
    if (num_shapes >= capacity)
    {
        cout << "ConsolePixelFrame::addShape ==> Expanding capacity to "
             << capacity * 2 << "shapes " << endl;
        Shape **old_shapes = pShapes;

        pShapes = new Shape*[capacity * 2];
        if (pShapes == NULL)
        {
            cout << "Error in expanding dynamic array for shapes capacity "
                 << capacity * 2 << "shapes " << endl;
            exit;
        }
        for (int i = 0; i < num_shapes; i++)
        {
            pShapes[i] = old_shapes[i];
        }
        capacity = capacity * 2;
        delete[] old_shapes;
    }
    pShapes[num_shapes] = pNew_shape;
    num_shapes++;
}
```

```
/* ConsolePixelDrawing.cpp (3) */

void ConsolePixelFrame::drawShapes()
{
    cout << "Drawing " << num_shapes << " shapes :" << endl;
    if (num_shapes > 0)
    {
        for (int i = 0; i < num_shapes; i++)
            pShapes[i]->draw(*this);
    }
}

bool ConsolePixelFrame::isValidIndex(int index)
{
     if ((index < 0) || (index >= num_shapes))
     {
        cout << "Error in ConsolePixelFrame::isValidIndex : current number of shapes ("
                << num_shapes << "), index : " << index << ") !!" << endl;
        return false;
     }
     else
        return true;
}
```

main()

```
/** main.cpp (1) */
#include <iostream>
#include <string>
#include "ConsolePixelDrawing.h"
#include "Shape.h"
#include "Triangle.h"
#include "Circle.h"
#include "Rectang.h"
#include "Polygon.h"
#include "Star.h"
using namespace std;

int main()
{
   Circle  cir(100, 100, 80, 0.0, RGB_RED, RGB_RED, 3, "Circle");
   Triangle tri(300, 100, 150, 130, 0.0, RGB_BLACK, RGB_YELLOW, 3, "Triangle");
   Rectang rec(500, 100, 150, 150, 0.0, RGB_BLACK, RGB_BLUE, 4, "Rectangle");
   PolyGon poly_5(100, 300, 80, 5, 0.0, RGB_BLACK, RGB_GREEN, 4, "Polygon_5");
   PolyGon poly_6(300, 300, 80, 6, 0.0, RGB_BLACK, RGB_BLUE, 4, "Polygon_6");
   PolyGon poly_7(500, 300, 80, 7, 0.0, RGB_BLACK, RGB_MAGENTA, 4, "Polygon_7");
   Star star_5(100, 500, 80, 5, 0.0, RGB_BLACK, RGB_GREEN, 4, "Star_5");
```

```
/** main.cpp (2) */

   ConsolePixelFrame frame(50, 50);

   frame.addShape(&cir);
   frame.addShape(&tri);
   frame.addShape(&rec);
   frame.addShape(&poly_5);
   frame.addShape(&poly_6);
   frame.addShape(&poly_7);
   frame.addShape(&star_5);

   frame.drawShapes();

   return 0;

}  // end of main()
```



Homework 6

Homework 6

6.1 가상함수와 지연 바인딩 (late-binding)을 사용하여 제공되는 다형성 (polymorphism)의 장점에 대하여 예를 들어 상세하게 설명하라.

6.2 가상함수를 사용하여 다형성 (polymorphism)을 제공하기 위한 지연 바인딩 (late-binding)이 실행되는 방법에 대하여 예를 들어 구체적으로 설명하라.

6.3 가상함수와 지연 바인딩 (late-binding) 을 사용하여 구현되는 다형성 (polymorphism)의 단점에 대하여 예를 들어 상세하게 설명하라.

6.4 상속, 다형성 및 가상함수를 사용하는 다음 C++ 클래스를 구현하라 : shape, Rectangle, Circle, Triangle, Hexahedron, Cylinder, and Prism.

Class Shape contains protected data members of struct position that contains int x and int y as coordinates. It also contains angle, name and color. The color is defined as an enumeration of COLOR that includes RED, GREEN, BLUE, BLACK, ORANGE, YELLO, CYAN, and WHITE. The class Shape contains virtual function draw() that will be used in late biding for polymorphism.

Class Rect inherits from class Shape, contains protected data member int width and int depth. It also contains virtual function draw().

Class Circle inherits from class Shape, contains protected data member int radius. It also contains virtual function draw().

Class Triangle inherits from class Shape, contains protected data members int base and int tri_height. It also contains virtual function draw().

Class Hexahedron inherits from class Rectangle, and contains height. It also contains virtual function draw().

Class Cylinder inherits from class Circle, and contains data member int height. It also contains virtual function draw().

Class Prism inherits from class Triangle, and contains data member int height. It also contains virtual function draw().

Class Square, Circle, Triangle, Hexahedron, Cylinder, and Prism have member method "double area() const;" to calculate the area of each shape, and operator over-loadings of "<<" for printout the object.

6.5 앞의 homework 6.4번에서 구현한 클래스들을 다음 응용 프로그램을 사용하여 실행시키고, 정확한 동작을 확인하라.

```
/** main.cpp  (1) */

#include <iostream>
#include <string>
#include <fstream>
#include "Position.h"
#include "Shape.h"
#include "Triangle.h"
#include "Prism.h"
#include "Rectangle.h"
#include "Hexahedron.h"
#include "Circle.h"
#include "Cylinder.h"

using namespace std;

#define NUM_SHAPES 7

int main()
{
    ofstream fout;
    Shape s1(Position(1, 1), 0, RGB_BLUE, "Blue Shape");
    Triangle t1(Position(2, 2), 3, 4, 0, RGB_RED, "Red Triangle");
    Prism p1(Position(3, 3), 6, 7, 8, 0, RGB_GREEN, "Green Prism");
    Rect r1(Position(4, 4), 1, 2, 0, RGB_YELLOW, "Yellow Rectangle");
    Hexahedron hx1(Position(5, 5), 5, 6, 7, 0, RGB_ORANGE, "Orange Hexahedtron");
    Circle cir(Position(6, 6), 6, 0, RGB_MAGENTA, "Magenta Circle");
    Cylinder cyl(Position(7, 7), 8, 6, 0, RGB_WHITE, "White Cylinder");

    Shape *shapes[NUM_SHAPES];
```

```
/** main.cpp  (2) */

    fout.open("Output.txt");
    if (fout.fail())
    {
        cout << "Fail to open an output file (Output.txt)₩n";
        exit(1);
    }
    shapes[0] = &s1;
    shapes[1] = &t1;
    shapes[2] = &p1;
    shapes[3] = &r1;
    shapes[4] = &hx1;
    shapes[5] = &cir;
    shapes[6] = &cyl;

    for (int i = 0; i < NUM_SHAPES; i++)
    {
        shapes[i]->draw(fout);
    }

    fout.close();

    return 0;

}  // end of main()
```

6.5 (cont.) 출력 결과 예 (“output.txt”)

(1) color를 6자리 16진수로 표시

```
Shape::draw() => [name(Blue Shape), pos(1, 1), color(FF0000)]
Triangle::draw() => [name(Red Triangle), pos(2, 2), base (3), tri_height (4), color(0000FF), triangle area (6)]
Prism::draw() => [name(Green Prism), pos(3, 3), base (6), tri_height (7), color(00FF00),
   prism surface area (21), prism volume (168)]
Rect::draw() => [name(Yellow Rect), pos(4, 4), width (1), length (2), color(00FFFF), rectangle area (2)]
Hexahedron::draw() => [name(Orange Hexahedron), pos(5, 5), width (5), length (6), height (7), color(00A5FF),
   hexahedron surface area (214)), hexahedron volume (210)]
Circle::draw() => [name(Magenta Circle), pos(6, 6), radius (6), color(FF00FF), circle area (113.097)]
Cylinder::draw() => [name(White Cylinder), pos(7, 7), radius (8), height (6), color(FFFFFF),
   cylinder surface area (703.716)]), cylinder volume (1206.37)]
```

(2) color를 RGB(RR, GG, BB) 형식으로 표시 (option)

```
Shape::draw() => [name(Blue Shape), pos(1, 1), color(RGB(00,00,FF))]
Triangle::draw() => [name(Red Triangle), pos(2, 2), base (3), tri_height (4), color(RGB(FF,00,00)), triangle area (6)]
Prism::draw() => [name(Green Prism), pos(3, 3), base (6), tri_height (7), color(RGB(00,FF,00)),
   prism surface area (21), prism volume (168)]
Rect::draw() => [name(Yellow Rect), pos(4, 4), width (1), length (2), color(RGB(FF,FF,00)), rectangle area (2)]
Hexahedron::draw() => [name(Orange Hexahedron), pos(5, 5), width (5), length (6), height (7), color(RGB(FF,A5,00)),
   hexahedron surface area (214)), hexahedron volume (210)]
Circle::draw() => [name(Magenta Circle), pos(6, 6), radius (6), color(FF00FF), circle area (113.097)]
Cylinder::draw() => [name(White Cylinder), pos(7, 7), radius (8), height (6), color(RGB(FF,FF,FF)),
   cylinder surface area (703.716)]), cylinder volume (1206.37)]
```