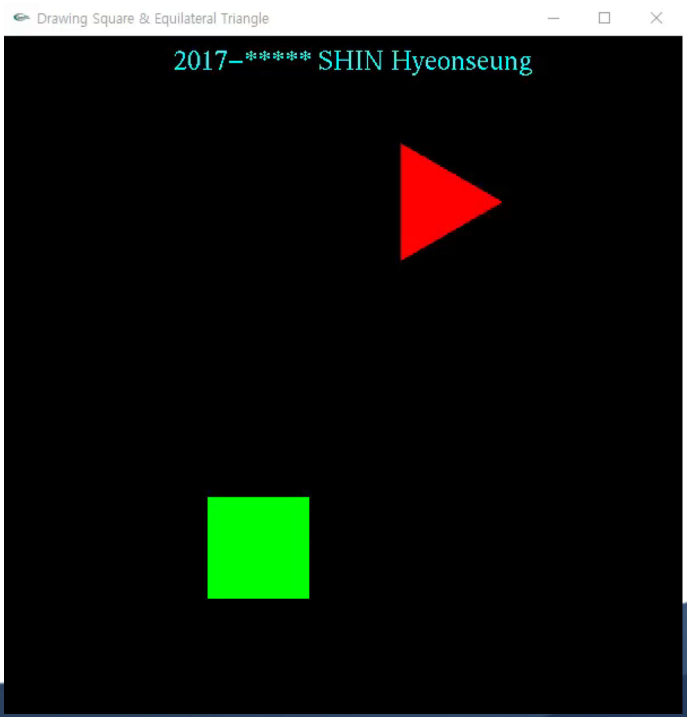
LAB I
Week 09

Seoul National University
Graphics & Media Lab
HyeonSeung Shin

Today's Mission

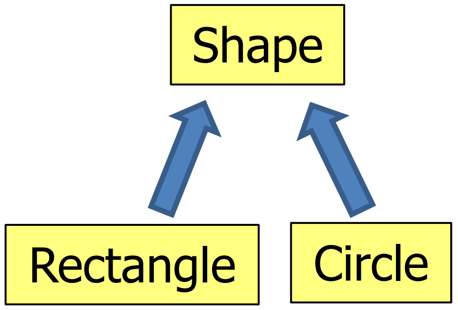
- Draw a moving square and a triangle using
  - Inheritance
  - Abstract class
  - Pure virtual function

- Display characters in the OpenGL window


Drawing Square & Equilateral Triangle
2017-****** SHIN Hyeonseung

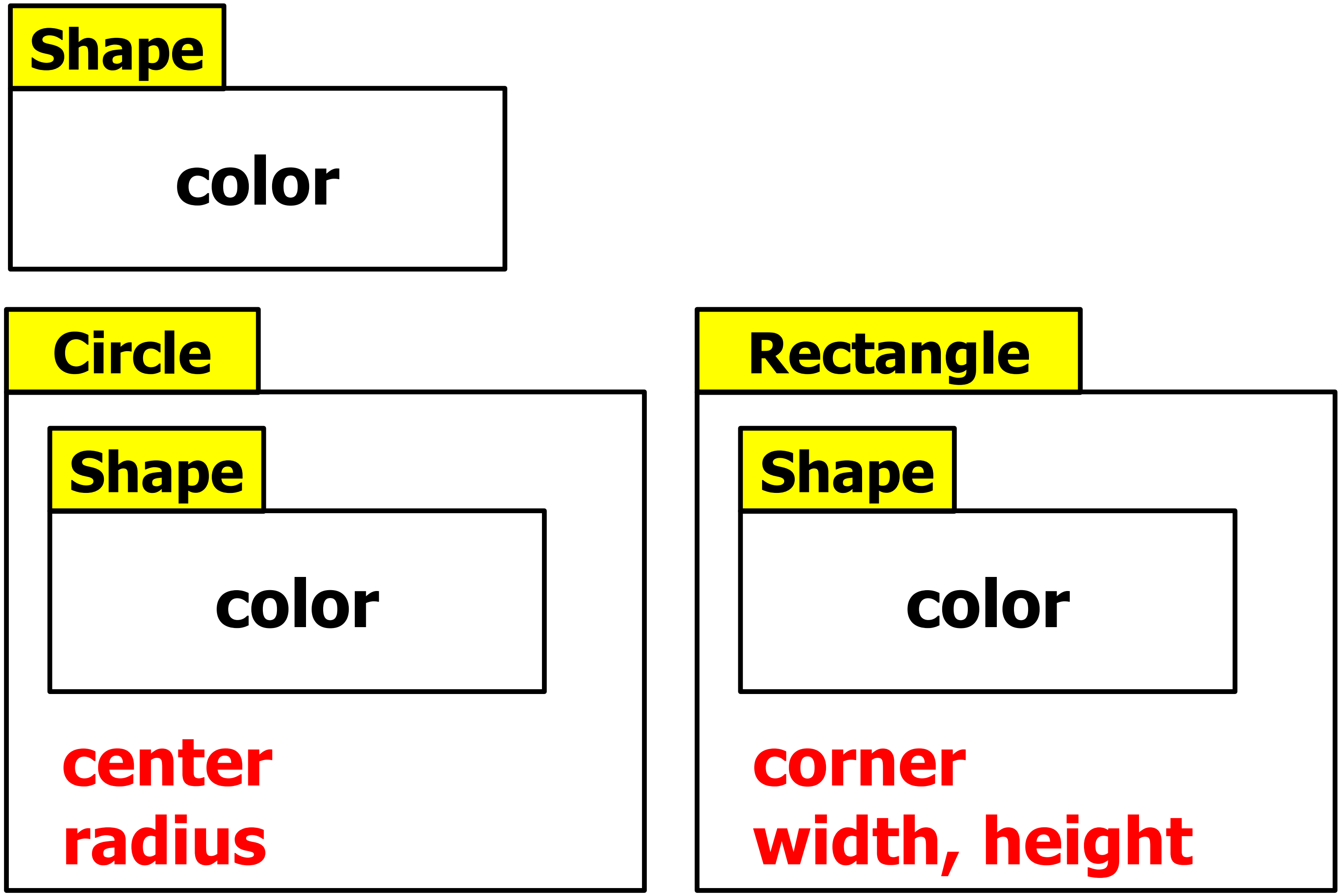
An Example

- Think about Shape, Rectangle, Circle
  - Rectangle or Circle is a Shape.
  - Each Shape has its own color and needs to be drawn.



```
class Shape {
public:
   Shape();
   virtual void draw(Image& img) const = 0;

protected:
   unsigned char color[3];
};

class Rectangle : public Shape {
public:
   void draw(Image& img) const;
   Vec2   corner;
   float  width, height;
};

class Circle : public Shape {
public:
   void draw(Image& img) const;
   Vec2   center;
   float  radius;
};
```

Inherited Datafields



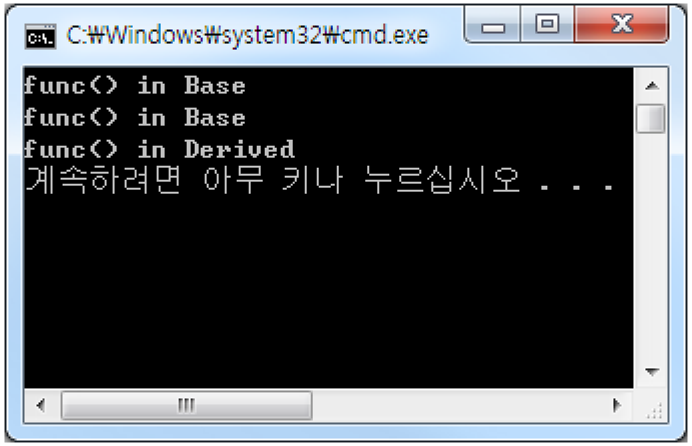
Public, Protected, Private Inheritance

- Example

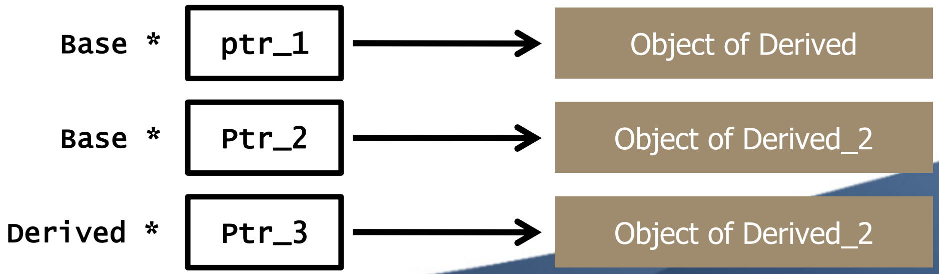
```
class Base {
public :
   int public_var;
protected :
   int protected_var;
private :
   int private_var;
};

class Public_Derived : public Base {
   // public_var is public
   // protected_var is protected
   // private_var is not accessible
};

class Protected_Derived : protected Base {
   // public_var, protected_var are protected
   // private_var is not accessible
};

class Private_Derived : private Base {
   // public_var, protected_var are private
   // private_var is not accessible
};
```

Non-Virtual Function

```
class Base {
public :
   void func() {
      std::cout << "func() in Base" << std::endl;
   }
};

class Derived : public Base {
public :
   void func() {
      std::cout << "func() in Derived" << std::endl;
   }
};

class Derived_2 : public Derived {
public :
   void func() {
      std::cout << "func() in Derived_2" << std::endl;
   }
};

void main() {
   Base * ptr_1 = new Derived();
   Base * ptr_2 = new Derived_2();
   Derived * ptr_3 = new Derived_2();
   ptr_1->func();
   ptr_2->func();
   ptr_3->func();
}
```

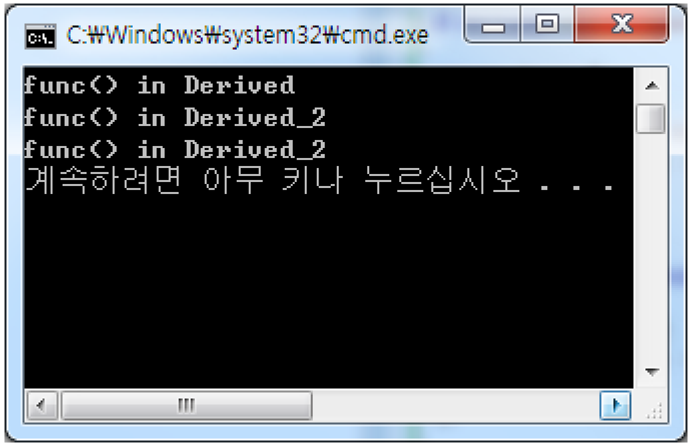


```
func() in Base
func() in Base
func() in Derived
계속하려면 아무 키나 누르십시오 . . .
```

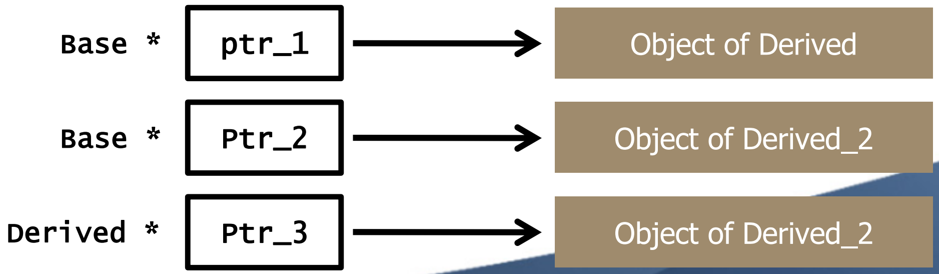


Virtual Function

```
class Base {
public :
    virtual void func() {
        std::cout << "func() in Base" << std::endl;
    }
};

class Derived : public Base {
public :
    void func() {
        std::cout << "func() in Derived" << std::endl;
    }
};

class Derived_2 : public Derived {
public :
    void func() {
        std::cout << "func() in Derived_2" << std::endl;
    }
};

void main() {
    Base * ptr_1 = new Derived();
    Base * ptr_2 = new Derived_2();
    Derived * ptr_3 = new Derived_2();
    ptr_1->func();
    ptr_2->func();
    ptr_3->func();
}
```



```
func() in Derived
func() in Derived_2
func() in Derived_2
계속하려면 아무 키나 누르십시오 . . .
```



Pure Virtual Functions

- A pure virtual function is defined by writing =0 after the function parameter list.

- Defining a function as pure virtual indicates that the function provides only the interface so that the derived classes must override the null definition.
  - The pure virtual function must be implemented by the derived class. Otherwise, it creates a compilation error.

```
class Base {
public :
   virtual void func() = 0;   // pure virtual function
};
```

Abstract Class

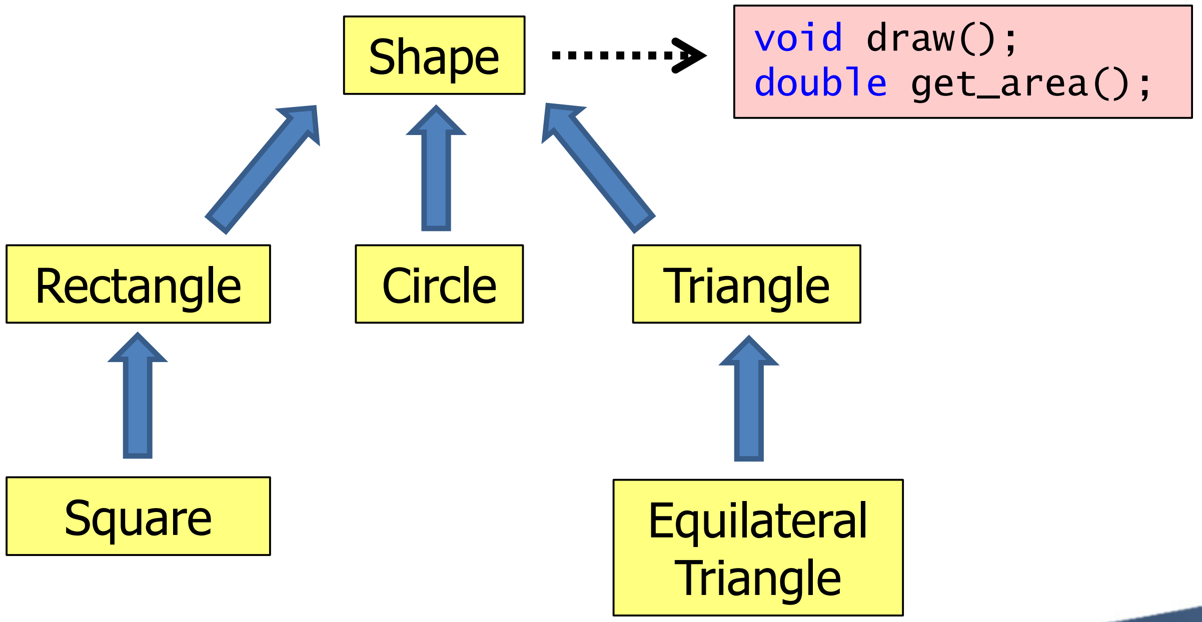
- A class containing one or more pure virtual functions.

```
class Base {                                  // abstract class
public :
   virtual void func() = 0;
};

class Derived : public Base {                 // abstract class
};

class Derived_2 : public Derived {            // not abstract class
public :
   void func() {
      std::cout << "func() in Derived_2" << std::endl;
   }
};
```

Example of an Abstract Class (Shape)

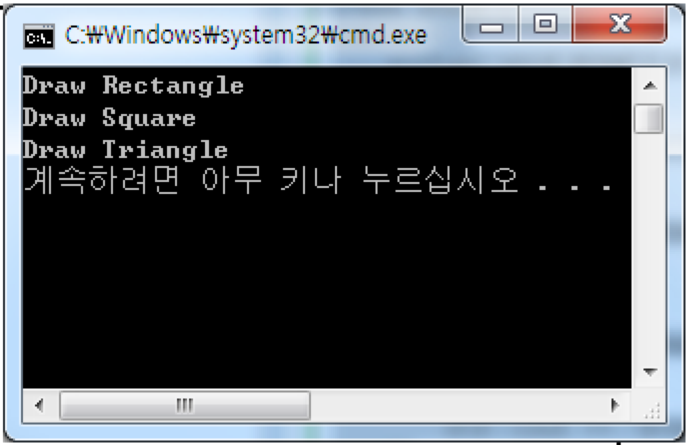
- A pure virtual function provides an **interface** for the derived classes to override.
- Actual implementations should be made by the derived classes

```
class Shape {                                   // abstract base class
public :
   virtual void draw() = 0;
   virtual double get_area() = 0;
};
```



Container Using Abstract Class

```
class Shape {                          // abstract base class
public :
   virtual void draw() = 0;
};

class Rectangle : public Shape {
public :
   void draw() { std::cout << "Draw Rectangle" << std::endl; }
};

class Square : public Shape {
public :
   void draw() { std::cout << "Draw Square" << std::endl;     }
};

class Triangle : public Shape {
public :
   void draw() { std::cout << "Draw Triangle" << std::endl;  }
};

void main() {
   std::vector<Shape*> shapes;
   shapes.push_back(new Rectangle());
   shapes.push_back(new Square());
   shapes.push_back(new Triangle());

   for(std::vector<Shape*>::iterator it=shapes.begin();it!=shapes.end();++it)
      (*it)->draw();
}
```



```
Draw Rectangle
Draw Square
Draw Triangle
계속하려면 아무 키나 누르십시오 . . .
```

Displaying characters

- glRasterPos

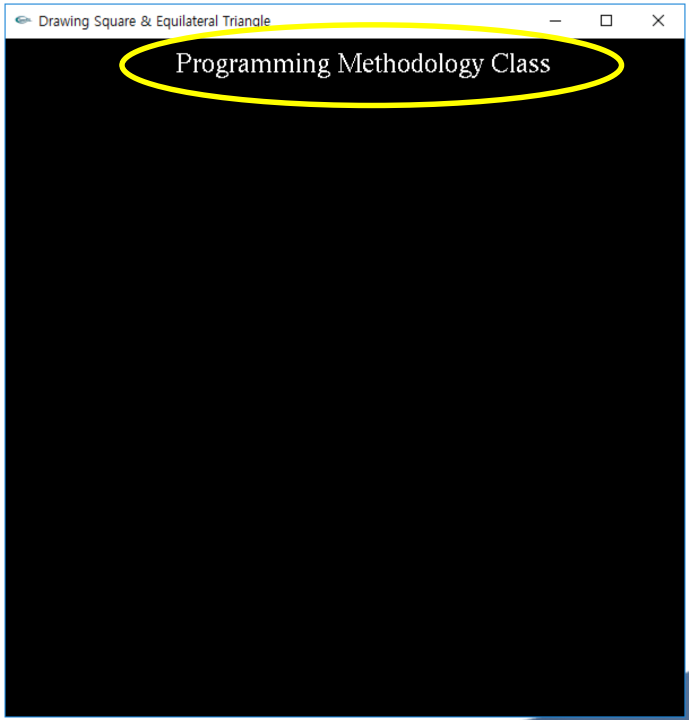
```
void draw_string(void * font, const char* str, int x, int y) {
        glRasterPos2i(x, y);
        for (int  i = 0; i < strlen(str); i++)
                glutBitmapCharacter(font, str[i]);
}
```

- glutBitmapCharater(font, character)
  - font
    - GLUT_BITMAP_8_BY_13
    - GLUT_BITMAP_9_BY_15
    - GLUT_BITMAP_TIMES_ROMAN_10
    - GLUT_BITMAP_TIMES_ROMAN_24
    - GLUT_BITMAP_HELVETICA_10
    - GLUT_BITMAP_HELVETICA_12
    - GLUT_BITMAP_HELVETICA_18

Displaying characters

```
void draw_string(void * font, const char* str, float x, float y) {
	glRasterPos2f(x, y);
	for (int i = 0; i < strlen(str); i++)
		glutBitmapCharacter(font, str[i]);
}
```

```
void renderScene() {
	// Clear Color and Depth Buffers
	glClear(GL_COLOR_BUFFER_BIT |
	GL_DEPTH_BUFFER_BIT);

	glColor3f(1, 1, 1);
	draw_string(GLUT_BITMAP_TIMES_ROMAN_24,
	"Programming Methodology Class", -0.5, 0.9);

	glutSwapBuffers();
}
```

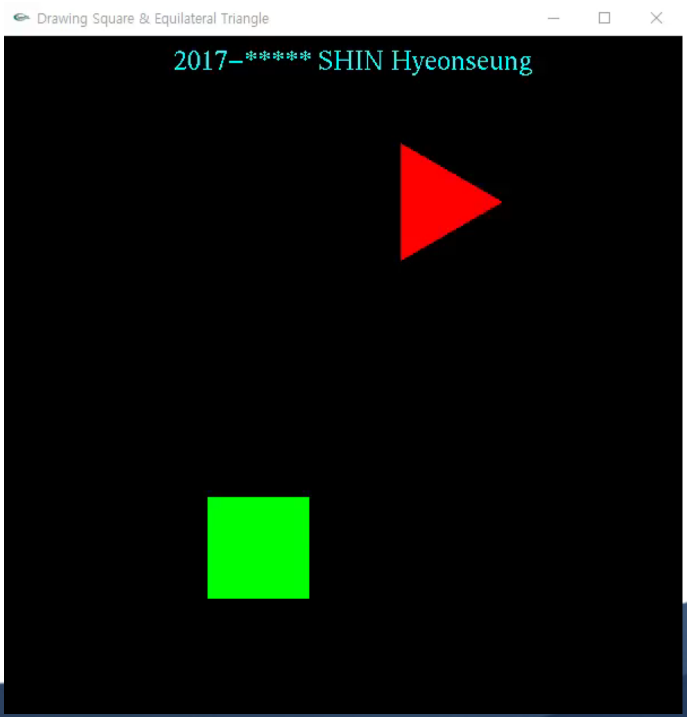


Programming Methodology Class

Today's Mission

- Draw a moving square and a triangle using
  - Inheritance
  - Abstract class
  - Pure virtual function

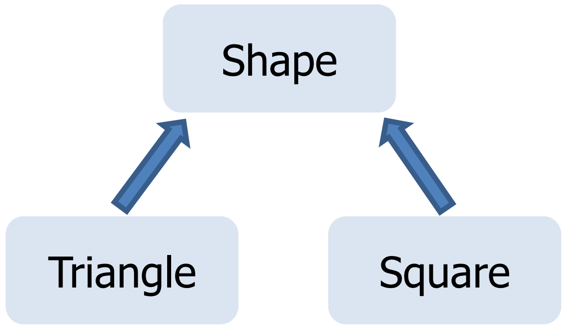
- Display characters in the OpenGL window


Drawing Square & Equilateral Triangle
2017-***** SHIN Hyeonseung

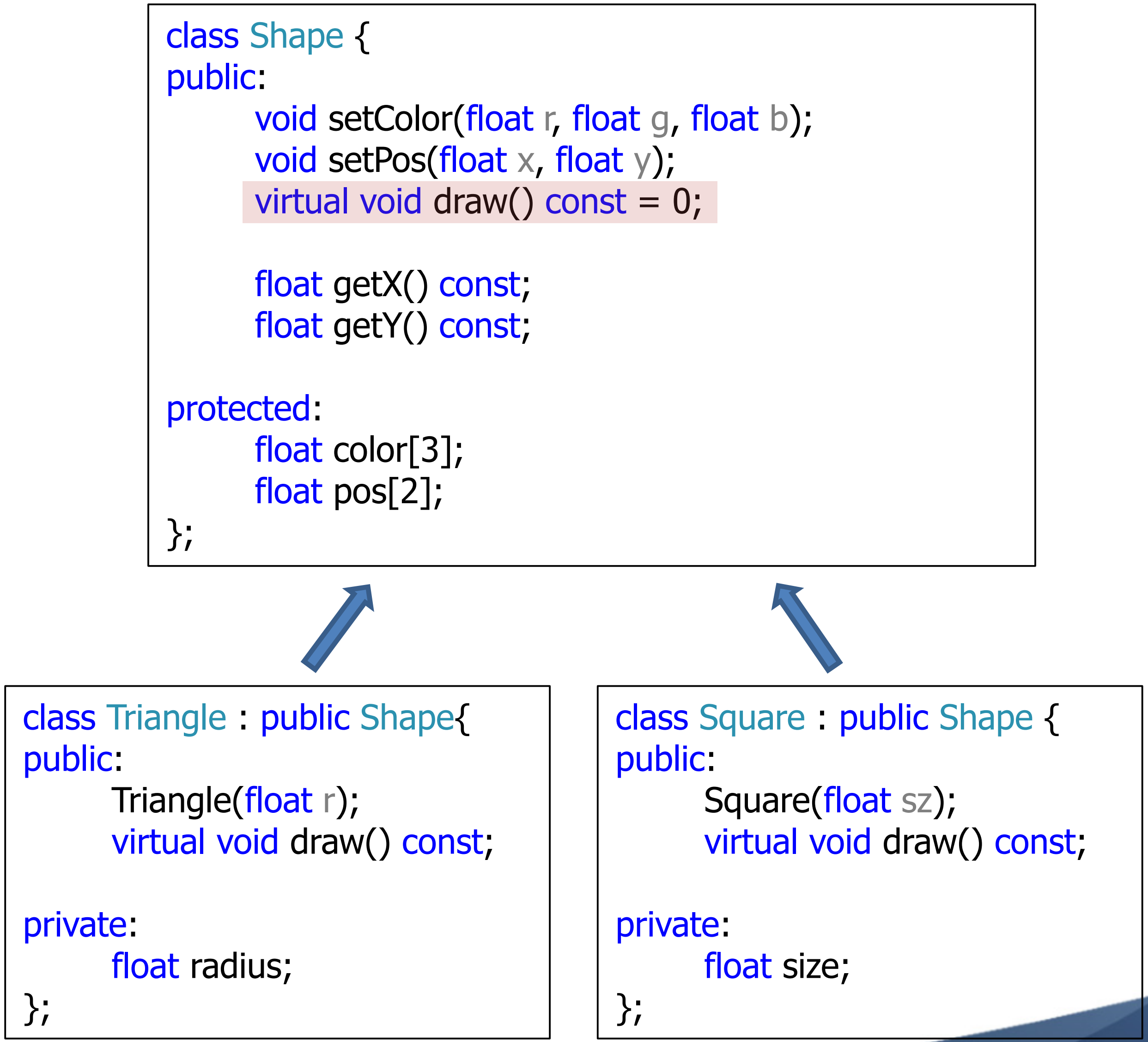
Given & To Do

- Given
  - Definition of the base class Shape

- To Do
  - Define subclasses Square and Triangle
    - Define pure virtual function Draw() of Shape
  - Display a string (your name and id) in the OpenGL window.



Class Diagram



```
class Shape {
public:
        void setColor(float r, float g, float b);
        void setPos(float x, float y);
        virtual void draw() const = 0;

        float getX() const;
        float getY() const;

protected:
        float color[3];
        float pos[2];
};
```

```
class Triangle : public Shape{
public:
        Triangle(float r);
        virtual void draw() const;

private:
        float radius;
};
```

```
class Square : public Shape {
public:
        Square(float sz);
        virtual void draw() const;

private:
        float size;
};
```

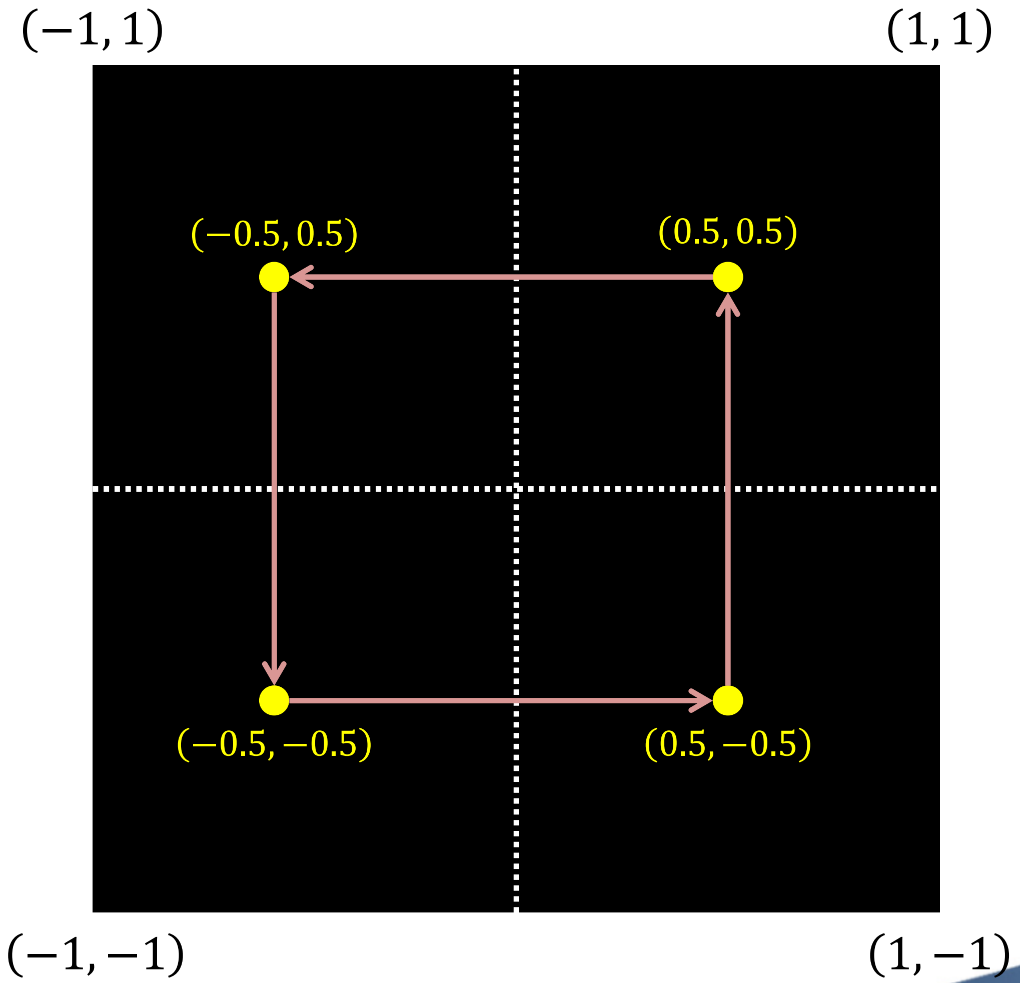
Introduction to New Classes

- Specification
  - Member variables of each class
    - Shape: pos, color
    - Square: size (= side length)
    - Triangle: radius (= distance between center and each vertex)

  - Position and color
    - Square
      - Position: (0.5, 0.5)
      - Color: green (0, 1, 0)
    - Triangle
      - Position: (-0.5, -0.5)
      - Color: red (1, 0, 0)

Introduction to New Classes

- Specification
  - Speed: 0.01

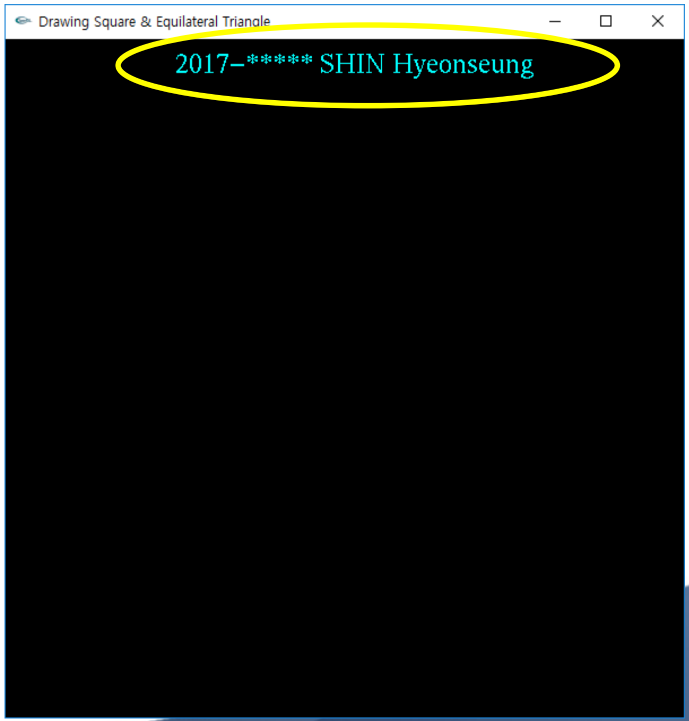


How to Display Char in OpenGL?

- You can use any font, color and position
- In this lab, display your student ID & name

```
void draw_string(void * font, const char* str, float x, float y) {
	glRasterPos2f(x, y);
	for (int i = 0; i < strlen(str); i++)
		glutBitmapCharacter(font, str[i]);
}
```

```
void renderScene() {
	// Clear Color and Depth Buffers
	glClear(GL_COLOR_BUFFER_BIT |
	GL_DEPTH_BUFFER_BIT);

	glColor3f(0, 1, 1);
	draw_string(GLUT_BITMAP_TIMES_ROMAN_24,
	“2017-***** SHIN Hyeonseung", -0.5, 0.9);

	glutSwapBuffers();
}
```



2017–***** SHIN Hyeonseung