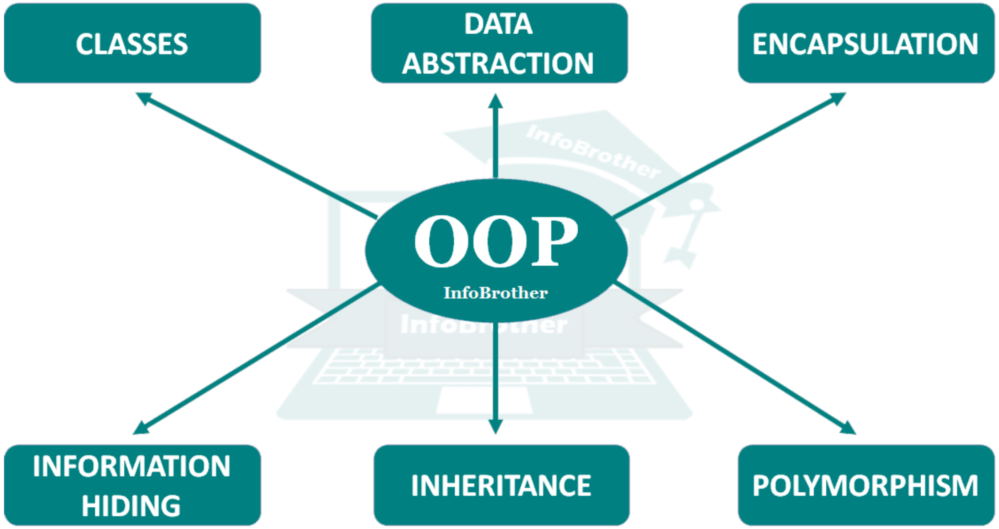
Computer Graphics Practice

Lecture 02

Dept. of Game Software
Yejin Kim

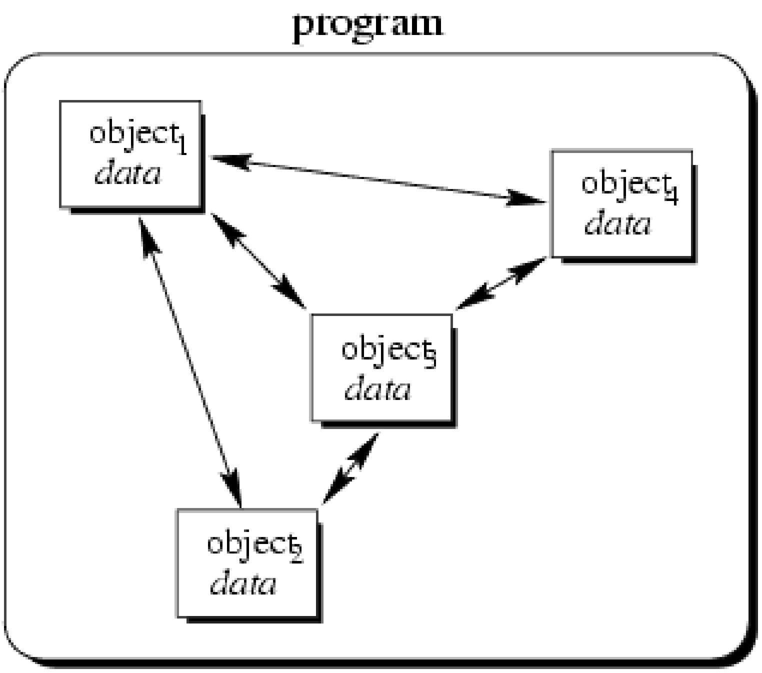
Plan

- Object-oriented (C++) programming
- Event-driven (Windows) programming
- Tutorial
  - OOP: Polymorphism
  - Creating Windows
  - Creating Framework
  - Initializing Direct3D

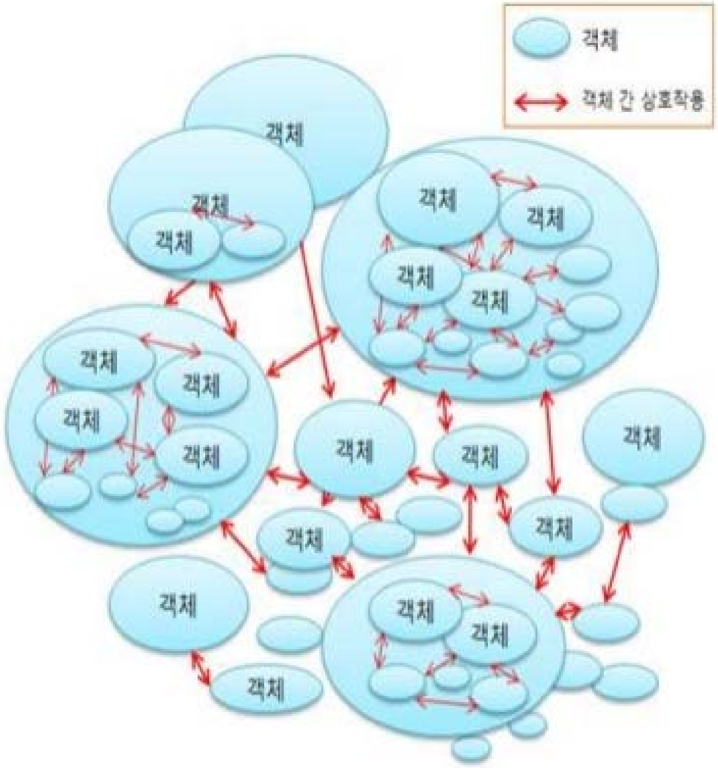


Object-Oriented Programming

- Object-oriented structures
  - Object: **attribute**(data structure) + **method**(function)
  - Each object communicates via messages

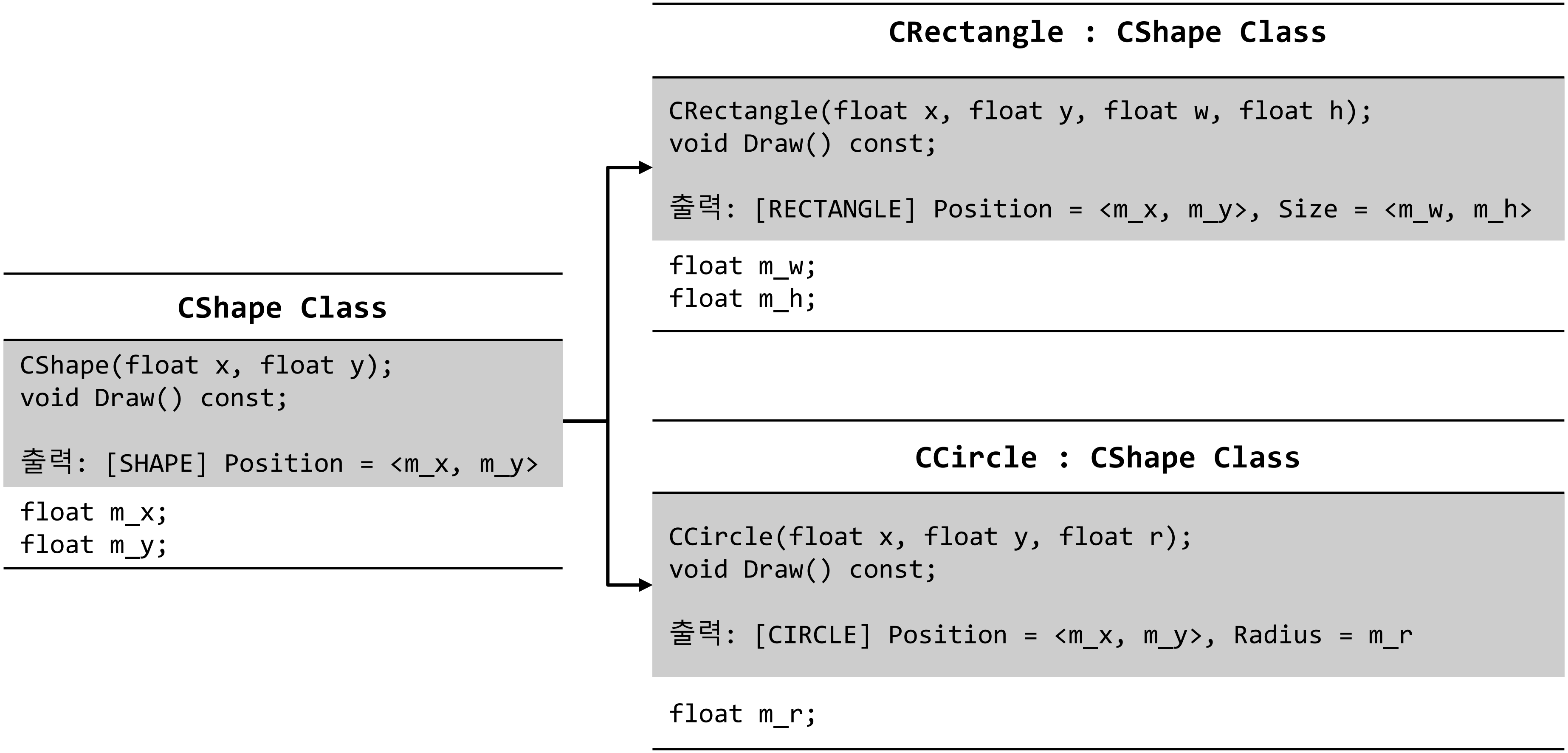


Object-oriented



Tutorial: OOP

- Creating Shape objects
  - Create child objects **inherited** from the parent object as follows:



CShape Class

```
CShape(float x, float y);
void Draw() const;

출력: [SHAPE] Position = <m_x, m_y>
```

```
float m_x;
float m_y;
```

CRectangle : CShape Class

```
CRectangle(float x, float y, float w, float h);
void Draw() const;

출력: [RECTANGLE] Position = <m_x, m_y>, Size = <m_w, m_h>
```

```
float m_w;
float m_h;
```

CCircle : CShape Class

```
CCircle(float x, float y, float r);
void Draw() const;

출력: [CIRCLE] Position = <m_x, m_y>, Radius = m_r
```

```
float m_r;
```

Tutorial: OOP

- Using the Shape objects

```
// main.cpp
int main()
{
	CShape a(100,40);
	CRectangle b(120,40,50,20);
	CCircle c(200,100,50);

	a.Draw();
	b.Draw();
	c.Draw();

	return 0;
}
```

  – what if we want to store different data types into a single variable?

  → Use the type conversion of a pointer

Tutorial: OOP

- Pointing a child object using a pointer of Parent class
  - Parent class의 pointer로 child object를 가리킬 수 **있음**

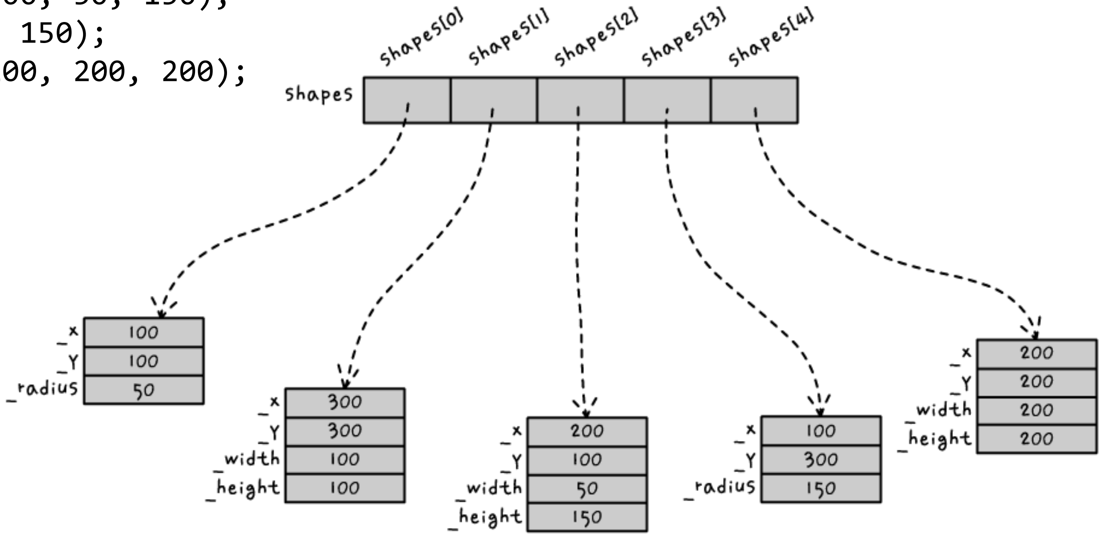
```
CCircle cir;          // Child class의 object 생성

// Parent class의 pointer 변수로 child object를 가리킴
CShape* shape = &cir;// OK
```

- Maintaining child objects using a pointer of parent class
  1. Parent class pointer의 array을 설정
  2. 필요 시 마다 **new**를 사용하여 child class 생성
  3. 생성된 child class의 주소 값을 parent class pointer에 할당
  4. 다 사용 했으면 **delete**를 사용하여 memory 해제

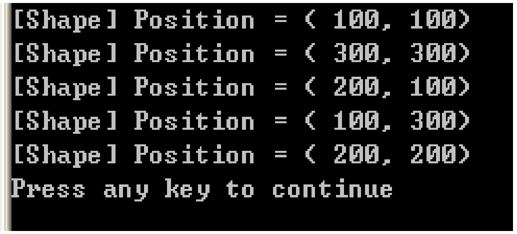
Tutorial: OOP

- Maintaining child objects using a pointer of parent class
  - CShape class의 object들을 array에 담아서 사용한 예

```
// main.cpp
int main()
{
      CShape* shapes[5] = {NULL};

      shapes[0] = new CCircle(100, 100, 50);
      shapes[1] = new CRectangle(300, 300, 100, 100);
      shapes[2] = new CRectangle(200, 100, 50, 150);
      shapes[3] = new CCircle(100, 300, 150);
      shapes[4] = new CRectangle(200, 200, 200, 200);

      for (int i = 0; i < 5; ++i)
          shapes[i]->Draw();

      for (int i = 0; i < 5; ++i)
      {
          delete shapes[i];
          shapes[i] = NULL;
      }

      return 0;
}
```



Tutorial: OOP

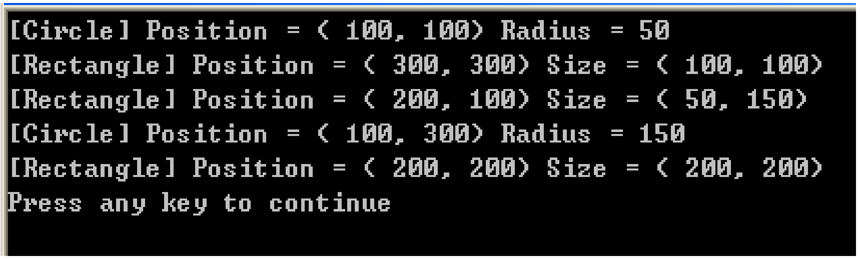


```
[Shape] Position = ( 100, 100)
[Shape] Position = ( 300, 300)
[Shape] Position = ( 200, 100)
[Shape] Position = ( 100, 300)
[Shape] Position = ( 200, 200)
Press any key to continue
```

- Executing the program: Draw()
  - CShape class의 function이 호출됨
    - Why? Parent class의 pointer로 child class를 가리킬 경우 해당 pointer type을 기준으로 function를 호출함
  - Object의 실제 type에 따라 호출할 순 없을까?
    - Use **virtual function** in Parent class
    - Member function이 compile시에 결정되지 않고, runtime시 결정

      → member function의 동적인 선택(dynamic binding)
- Draw() 문제 해결 예

```
// Shape.h
class CShape
{
        ...
        virtual void Draw() const;
};
```



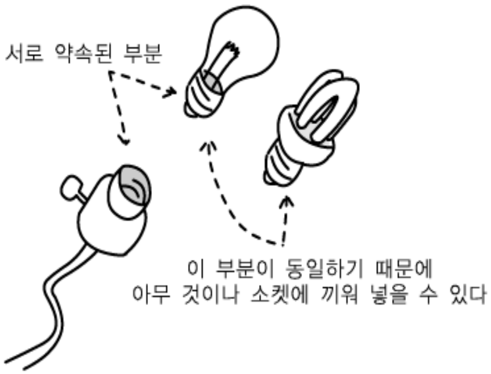
```
[Circle] Position = ( 100, 100) Radius = 50
[Rectangle] Position = ( 300, 300) Size = ( 100, 100)
[Rectangle] Position = ( 200, 100) Size = ( 50, 150)
[Circle] Position = ( 100, 300) Radius = 150
[Rectangle] Position = ( 200, 200) Size = ( 200, 200)
Press any key to continue
```

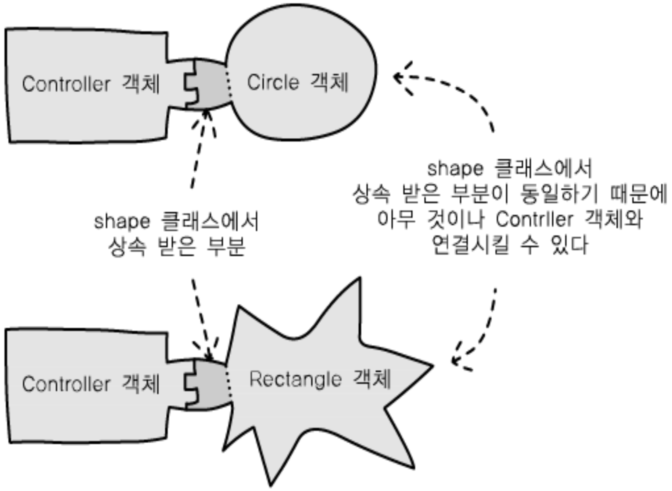
  - the same entity (function or object) behaves differently in different scenarios (*서로 다른 객체가 동일한 메시지에 대하여 서로 다른 방법으로 응답할 수 있는 기능) → **Polymorphism** in C++

Object-Oriented Programming

- Polymorphism in C++
  - Object type에 관계 없이 동일한 방법으로 다룰 수 있는 능력
    - e.g. Circle이나 Rectangle object들을 type에 상관 없이 Shape object처럼 다룰 수 있음
  - Object 간의 coupling(결합)을 약하게 만들어서, object 간의 연결을 유연하게 해 줌
    - e.g. 아래와 같이 새로운 함수를 만들 때

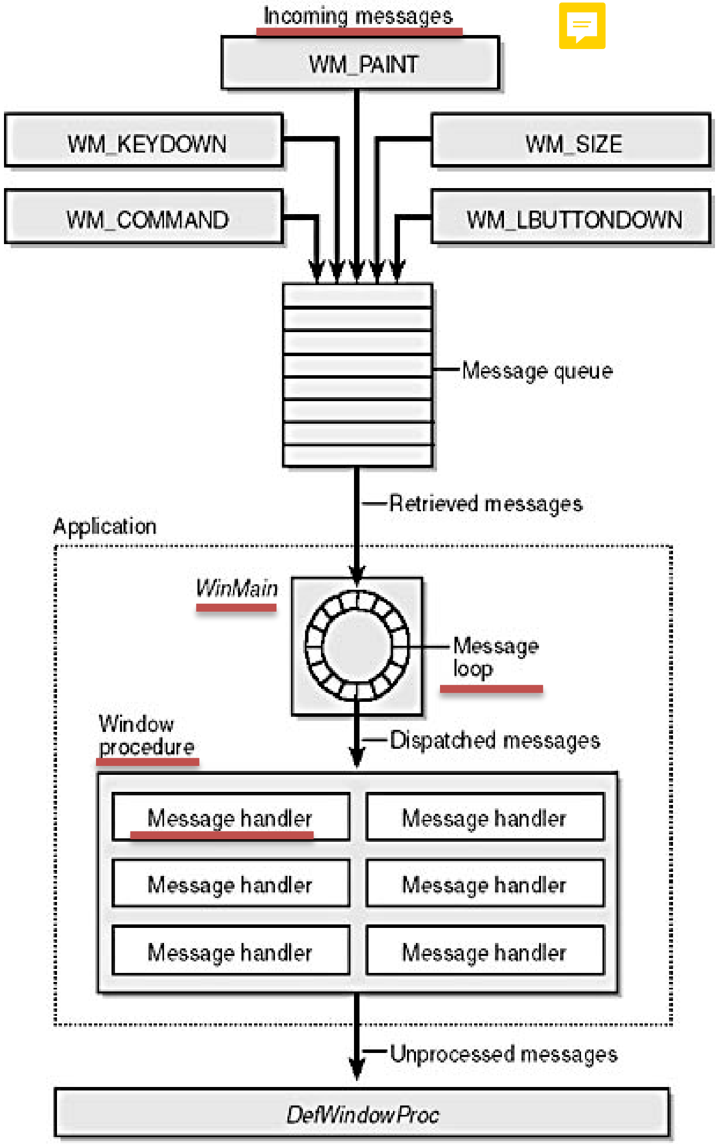
```
// 도형을 원점으로 이동하는 함수
void CController::MoveToOrigin(CShape *p)
{
        p->Move(0, 0);
        p->Draw();
}
```


서로 약속된 부분
이 부분이 동일하기 때문에
아무 것이나 소켓에 끼워 넣을 수 있다



Event-driven Programming

- Windows programing
  - **Event**: State change of input device or program inside
  - **Message**: Form of state change to the program
  - **Message handler**: Function handling event
- Win32 program structure
  - Begins with WinMain() function
  - Starts a massage loop in the WinMain() for waiting messages
  - Gets messages from operating system, a user or the program
  - Messages are processed by windows procedure
  - Ends when Quit message is given



Event-driven Programming

- Win32 Program Structure

```
WinMain(…)                           ← main function
{
       WNDCLASS …                    ← Define a new program
       CreateWindows (…)             ← Create a window

       while(GetMessage (…))         ← Message Loop
       {

              DispatchMessage(…)     ← Message Handler
       }                                (Windows Procedure)
}
```

Event-driven Programming

- Common Windows Messages

Message	**Sent When**
WM_CHAR	A character is input from the keyboard.
WM_COMMAND	The user selects an item from a menu, or a control sends a notification to its parent.
WM_CREATE	A window is created.
WM_DESTROY	A window is destroyed.
WM_LBUTTONDOWN	The left mouse button is pressed.
WM_LBUTTONUP	The left mouse button is released.
WM_MOUSEMOVE	The mouse pointer is moved.
WM_PAINT	A window needs repainting.
WM_QUIT	The application is about to terminate.
WM_SIZE	A window is resized.

Event-driven Programming

- Windows programming
  - is not making everything from nothing
  - is rather assembling existing functions and data types
- Extended Functions and data types are distributed as a form of library
  - e.g. 2D drawing functions (OpenCV, Simple2D, Direct2D, etc.)
    3D drawing functions (Direct3D, OpenGL, Vulkan, etc.)
    Physics functions (ODE, Bullet, PhysX, etc.)
    Sound functions (OpenAL, SDL, DirectShow, etc.)
    …

Event-driven Programming

- Application Programming Interface(API)
  - A set of functions for controlling and using operating system
  - Mostly C functions
- Windows(Win32) API
  - Collection of C functions for making windows programming (library)
  - e.g. Functions for

    creating new windows,
    adding a button,
    adding a new menu,
    …

Tutorial: Windows

- Create an empty Windows
  – 새 프로젝트 만들기 → 빈 프로젝트
  – Project 속성 → 고급 → 문자 집합 → 설정 안 함
  – Project 속성 → 링커 → 시스템 → 하위 시스템 → 창(/SUBSYSTEM:WINDOWS)
- Create WinMain()

```cpp
// Main.cpp
#include <windows.h>
LONG WINAPI WndProc (HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE
hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{
    WNDCLASS wc;
    HWND hwnd;
    MSG msg;

    wc.style = 0;
    wc.lpfnWndProc = (WNDPROC) WndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (NULL, IDI_WINLOGO);
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName = NULL;
    wc.lpszClassName = "MyWndClass";

    RegisterClass (&wc);
```

```cpp
    hwnd = CreateWindow (
        "MyWndClass",        // WNDCLASS name
        "SDK Application",   // Window title
        WS_OVERLAPPEDWINDOW,// Window style
        CW_USEDEFAULT,       // Horizontal position
        CW_USEDEFAULT,       // Vertical position
        300,                 // Initial width
        200,                 // Initial height
        HWND_DESKTOP,        // Handle of parent window
        NULL,                // Menu handle
        hInstance,           // Application's instance handle
        NULL                   // Window-creation data
    );

    ShowWindow (hwnd, nCmdShow);
    UpdateWindow (hwnd);

    while (GetMessage (&msg, NULL, 0, 0)) {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return msg.wParam;
}
```

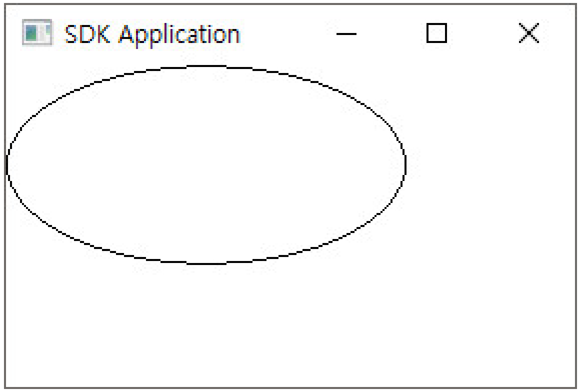
Tutorial: Windows

- Create WndProc()

```
// Main.cpp
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hdc;

    switch (message) {

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps);
        Ellipse (hdc, 0, 0, 200, 100);
        EndPaint (hwnd, &ps);

        return 0;

    case WM_DESTROY:
        PostQuitMessage (0);
        return 0;
    }
    return DefWindowProc (hwnd, message, wParam, lParam);
}
```

Tutorial: Windows

- Run the program
  - Check the main Windows and the circle

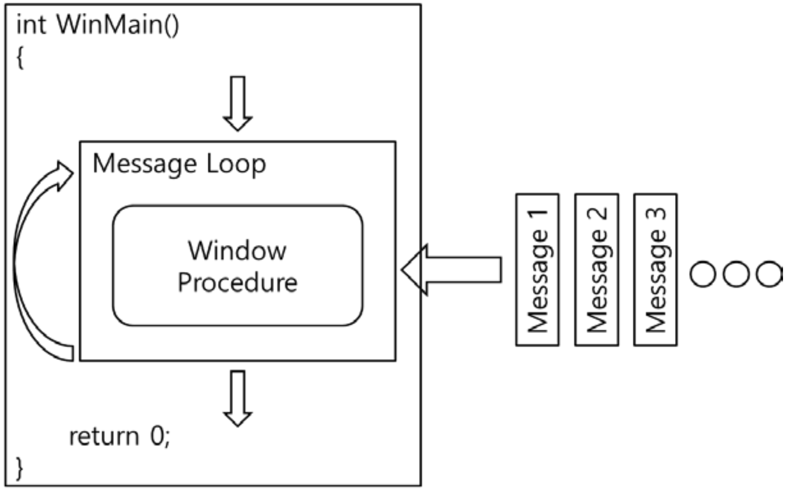


Tutorial: Windows

- Code looks complex, but same structure

```
// Main.cpp
#include <windows.h>
LONG WINAPI WndProc (HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE
hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{
      ...
      while (GetMessage (&msg, NULL, 0, 0)) {
              TranslateMessage (&msg);
              DispatchMessage (&msg);
      }
      ...
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM
wParam, LPARAM lParam)
{
      ...
    switch (message) {
          case WM_PAINT:
            ...

          case WM_DESTROY:
            ...
    }
    return DefWindowProc (hwnd, message, wParam, lParam);
}
```

=



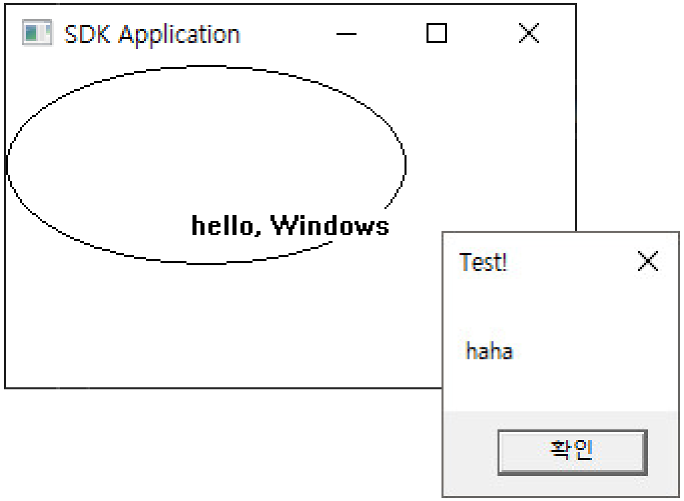
Tutorial: Windows

- Add a text output and an event handler for LMB

```
// Main.cpp
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
     ...

     case WM_PAINT:
       hdc = BeginPaint(hWnd, &ps);
       Ellipse (hdc, 0, 0, 200, 100);

       RECT rect;
       GetClientRect(hwnd, &rect);
       DrawText(hdc, "hello, Windows", -1, &rect, DT_SINGLELINE|DT_CENTER|DT_VCENTER);

       EndPaint(hWnd, &ps);
       return 0;

      case WM_LBUTTONDOWN:
            MessageBox(hwnd, "haha", "Test!", MB_OK);
            break;

     ...
}
```

Tutorial: Windows

- Run the program
  - Check the text and the pop-up box when LMB is clicked

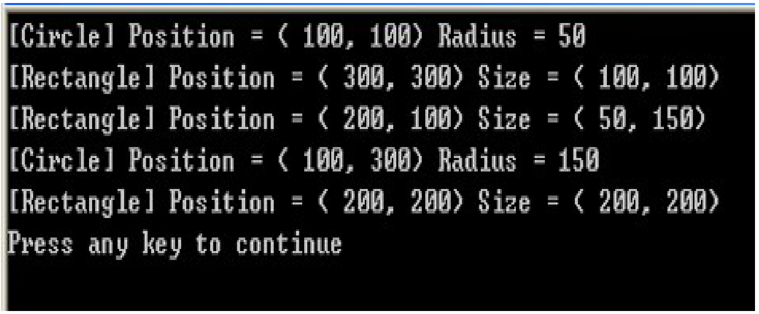


hello, Windows

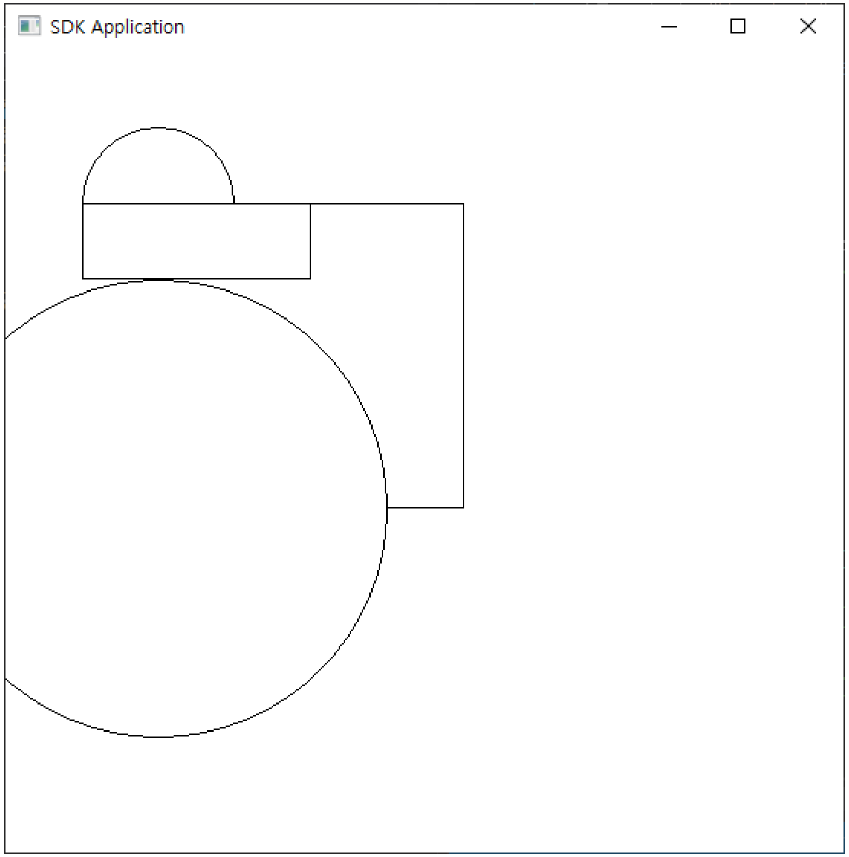
haha

Tutorial: Windows

- Draw Shape objects on Windows
  - Use Shape object codes and polymorphism

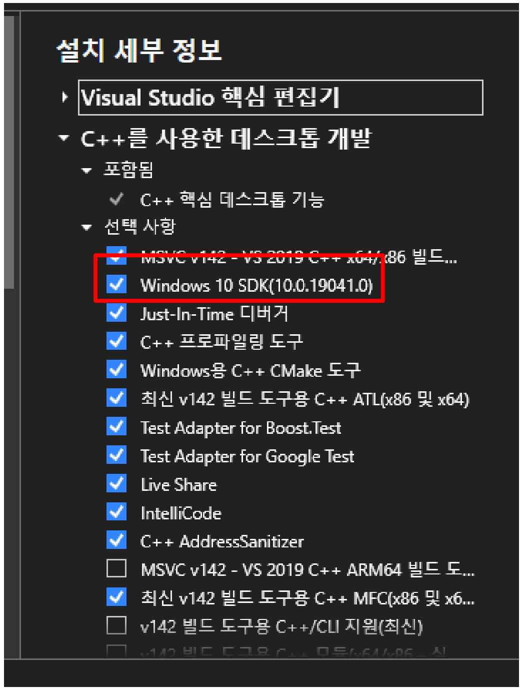


```
[Circle] Position = ( 100, 100) Radius = 50
[Rectangle] Position = ( 300, 300) Size = ( 100, 100)
[Rectangle] Position = ( 200, 100) Size = ( 50, 150)
[Circle] Position = ( 100, 300) Radius = 150
[Rectangle] Position = ( 200, 200) Size = ( 200, 200)
Press any key to continue
```



Tutorial: Framework Setup

- Setting up DirectX 11 with Visual Studio 2019
  - Install Visual Studio 2019 with Windows 10 SDK
    - DirectX SDK is a part of the Windows 10 SDK
    - Install DirectX SDK (June 2010) if old DirectX functions are used
  - Create a Win32 project: 새 프로젝트 만들기 → 빈 프로젝트
    - 속성 → 링커 → 시스템 → 하위 시스템: **창(/SUBSYSTEM:WINDOWS)**
    - 속성 → C/C++ → 고급 → 특정 경고 사용 안 함: **4005**
      - 4005: DirectX macro redefinitions
  - Build with **x86**

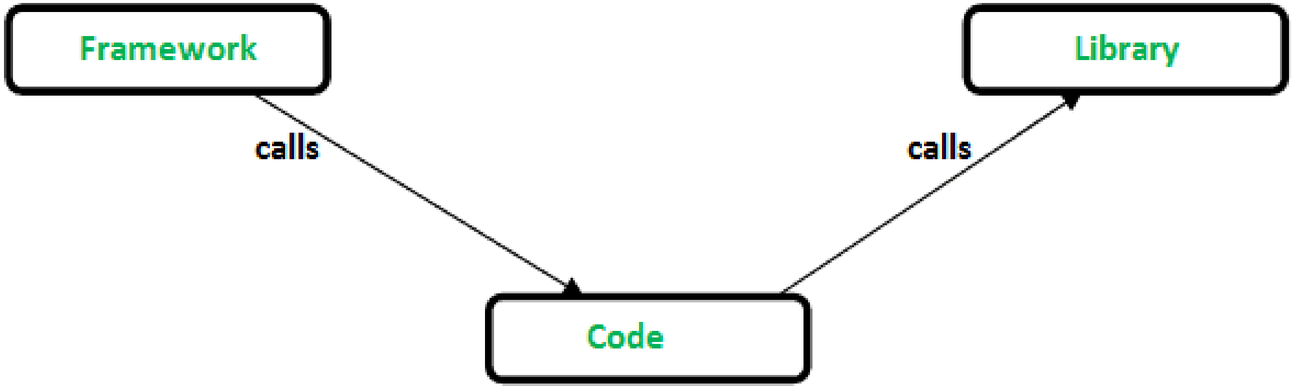


설치 세부 정보

- Visual Studio 핵심 편집기
- C++를 사용한 데스크톱 개발
  - 포함됨
    - ✓ C++ 핵심 데스크톱 기능
  - 선택 사항
    - ☑ MSVC v142 - VS 2019 C++ x64/x86 빌드...
    - ☑ Windows 10 SDK(10.0.19041.0)
    - ☑ Just-In-Time 디버거
    - ☑ C++ 프로파일링 도구
    - ☑ Windows용 C++ CMake 도구
    - ☑ 최신 v142 빌드 도구용 C++ ATL(x86 및 x64)
    - ☑ Test Adapter for Boost.Test
    - ☑ Test Adapter for Google Test
    - ☑ Live Share
    - ☑ IntelliCode
    - ☑ C++ AddressSanitizer
    - □ MSVC v142 - VS 2019 C++ ARM64 빌드 도...
    - ☑ 최신 v142 빌드 도구용 C++ MFC(x86 및 x6...
    - □ v142 빌드 도구용 C++/CLI 지원(최신)
    - □ v142 빌드 도구용 C++ 모듈(x64/x86 – 실

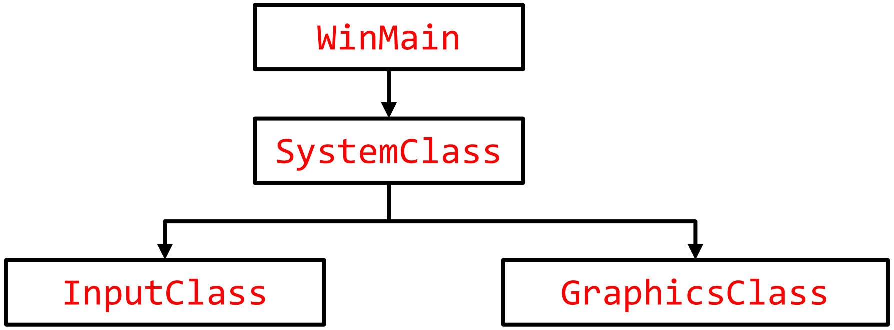
Tutorial: Framework Setup

- Creating a Framework and Windows
  - Code framework
    - Handles the basic windows functionality and provides an easy way to expand the code in an organized and readable manner.
    - Keep the framework as thin as possible.



Tutorial: Framework Setup

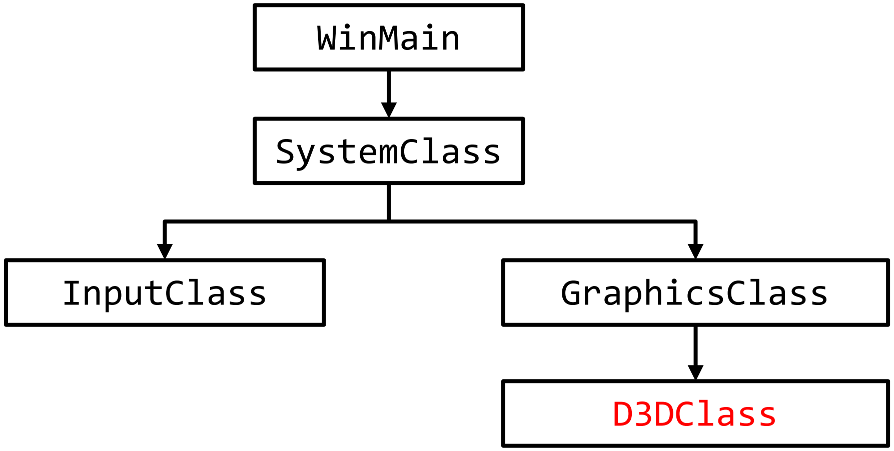
- Creating a Framework and Windows
  - **WinMain**: handle the entry point of the application
  - **SystemClass**: encapsulate the entire application that will be called from within the WinMain function
  - **InputClass**: handle user inputs
  - **GraphicsClass**: initialize and shut down D3DClass object



- Exercises
  - Run the framework in a full screen mode

Tutorial: Framework Setup

- Initializing Direct3D
  - **D3DClass**: initialize the DirectX graphics code



WinMain

SystemClass

InputClass

GraphicsClass

D3DClass

- Exercises
  - Change the clear(background) color to yellow.
  - Add a code that prints out the video card name and memory amount to a text file: "VideoInfo.txt".

Q & A