## □ 예제 프로그램 3: LabProject02(게임 프로그램의 골격 작성하기)

이제 앞에서 만든 LabProject01을 기반으로 게임 프로그램의 골격을 만들어 보자. 게임 프로그램의 골격을 만들기 위해 일반적인 3D 게임을 살펴보자. 다음 <그림 1>의 3D 게임의 실행 예를 보면 3D 게임은 기본적으로 사용자 입력을 받아 게임 객체들을 이동하거나 애니메이션 시키고 매번 하나의 장면을 생성하여 연속적으로 그래픽 디바이스에 출력하는 과정을 포함한다.

그림 1. 게임의 실행 예

## ■ 3D 게임 프로그램의 기본 요소

먼저 게임 프로그램의 기본 골격의 요소를 생각해보자. 게임 프로그램이 가져야 하는 기본적인 구성 요소는 다음과 같다.

- 윈도우 메시지 처리
  윈도우 메뉴와 프로그램 종료 그리고 윈도우 메시지(Message)를 처리하는 부분이다. 기존의 윈도우 프로그래밍 방법을 그대로 적용한다.
- Direct3D 디바이스 생성
  Direct3D 그래픽 디바이스, 스왑체인, 렌더 타겟 등을 생성하여 그래픽 출력을 준비하는 부분이다.
- Direct3D 디바이스 소멸
  프로그램이 종료될 때 생성하였던 Direct3D 그래픽 디바이스, 스왑체인, 렌더 타겟 등을 소멸(Release)시키는 부분이다.
- 사용자 입력 처리
  사용자 입력(마우스, 키보드)을 처리하여 게임 캐릭터의 이동 또는 게임 객체의 움직임을 제어하거나 게임에 필요한 입력을 처리하는 부분이다. DirectInput을 사용하거나 윈도우 API 함수를 사용하여 처리한다.

- 게임 객체의 생성
  게임에서 사용되는 게임 객체(Game Object)를 생성하는 부분이다.
- 게임 객체의 소멸
  게임에서 더 이상 사용되지 않는 게임 객체를 소멸시키는 부분이다.
- 게임 객체의 애니메이션 처리
  사용자 입력을 기반으로 게임 객체의 움직임 또는 애니메이션을 처리하는 부분이다.
- 게임 상태 정보 설정
  게임의 진행과 그래픽 렌더링의 관리를 위한 상태 정보를 설정하는 부분이다.
- 장면(Scene) 생성 및 렌더링
  게임의 한 장면을 생성하여 그래픽 디바이스에 출력하는 부분이다.

이제 Direct3D 그래픽 라이브러리를 사용하여 가장 기본적인 3D 그래픽 프로그램을 작성해 보자. 추가되는 코드는 다음과 같이 구분할 수 있다.

- Direct3D 라이브러리 헤더 포함하기
- Direct3D 객체 변수 선언하기
- 함수 선언하기
- 메시지 루프 변경하기
- 윈도우 크기 변경하기
- 렌더 타겟 지우기와 그리기(Render) 추가하기
- Direct3D 디바이스와 스왑 체인(Swap Chain)을 생성하는 함수를 추가하기

① 새로운 프로젝트 추가하기

앞에서 작성한 “LabProjects" 솔루션에 새로운 프로젝트 ”LabProject02“를 추가(응용 프로그램 마법사 실행)하자.

② Direct3D 라이브러리 헤더 포함하기

“stdafx.h" 파일의 마지막에 다음과 같이 Direct3D와 관련있는 헤더 파일들을 추가한다.

```
#include <string>
#include <wrl.h>
#include <shellapi.h>

#include <d3d12.h>
#include <dxgi1_4.h>

#include <D3Dcompiler.h>

#include <DirectXMath.h>
#include <DirectXPackedVector.h>
#include <DirectXColors.h>
```

```
#include <DirectXCollision.h>

#include <DXGIDebug.h>

using namespace DirectX;
using namespace DirectX::PackedVector;

using Microsoft::WRL::ComPtr;
```

<d3d12.h> 헤더 파일은 Direct3D 12 API 함수를 사용하기 위해 반드시 포함시켜야 하는 헤더 파일이다. “stdafx.h" 파일에 헤더 파일들을 추가하는 이유는 Visual Studio에서는 PCH(Precompiled Header)를 통하여 자주 변경되지 않는 헤더 파일에 대한 컴파일을 매번 하지 않아도 되도록 하는 기능을 제공하기 때문이다. 그리고 프로젝트 마법사가 만들어준 소스코드에는 “stdafx.h" 파일이 포함(#include)되도록 되어 있다.

③ 임포트(Import) 라이브러리 추가

임포트(Import) 라이브러리를 링커 속성에서 입력하지 않아도 되도록 다음을 “stdafx.h" 파일에 추가한다.

```
#pragma comment(lib, "d3dcompiler.lib")
#pragma comment(lib, "d3d12.lib")
#pragma comment(lib, "dxgi.lib")

#pragma comment(lib, "dxguid.lib")
```

④ 주 윈도우의 클라이언트 영역의 크기가 640x480이 되도록 수정한다.

클라이언트 영역의 크기 또는 후면 버퍼의 크기를 나타내는 상수를 “stdafx.h" 파일에 다음과 같이 정의한다.

```
#define FRAME_BUFFER_WIDTH    800
#define FRAME_BUFFER_HEIGHT   600
```

⑤ “CGameFramework” 클래스 생성하기

LabProject02에 게임 프로그램의 기본적 요소를 추가하기 위해 하나의 클래스(Class)를 만들도록 하자. 클래스의 이름은 “CGameFramework”이다. 이 클래스는 게임 프로그램이 가져야 될 기본적인 내용(형태)을 표현한다. <그림 2>와 같이 솔루션 탐색기에서 오른쪽 마우스 버튼으로 ”LabProject02“를 선택하고 『추가』, 『클래스』를 차례로 선택한다.
<그림 3>과 같이 클래스 추가 대화상자가 나타나면 ”설치된 템플릿“에서 『C++』을 선택하고 『C++ 클래스』를 선택한 후 『추가』 버튼을 누른다.

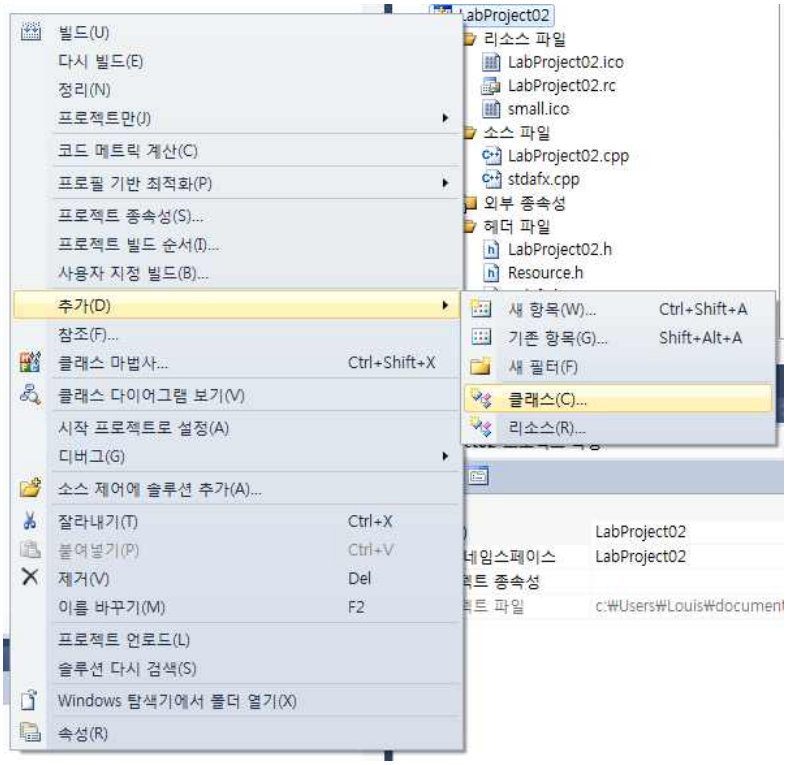


그림 2. 클래스 추가하기 메뉴

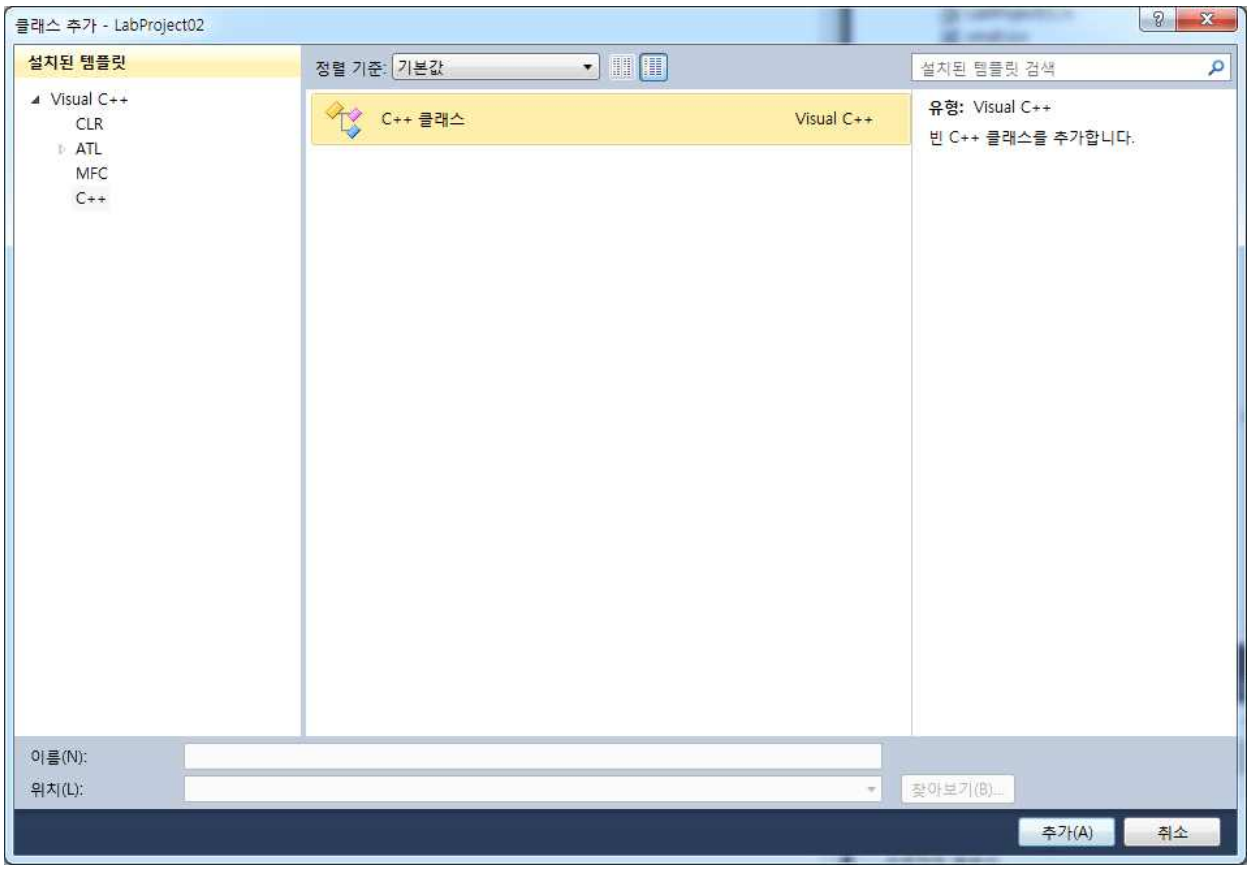


그림 3. 클래스 추가 대화상자

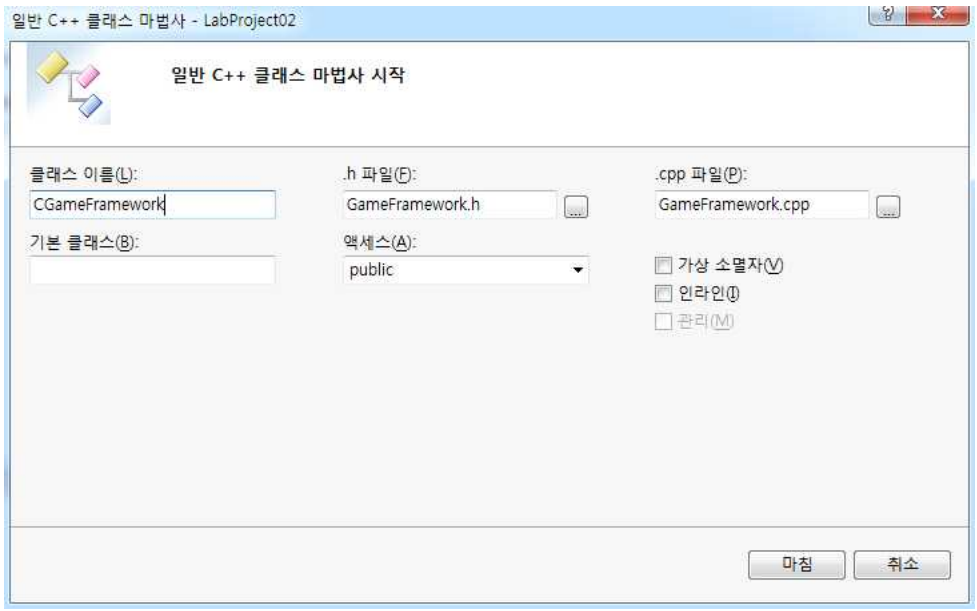


그림 4. 클래스 추가 마법사

<그림 4>와 같이 "일반 C++ 클래스 마법사"가 나타나면 『클래스 이름』에 "CGameFramework"를 입력한다. 그러면 ".h 파일"의 이름이 "GameFramework.h" 그리고 ".cpp 파일"의 이름이 "GameFramework.cpp"가 된다. 『마침』을 선택하면 클래스 마법사가 두 개의 파일을 프로젝트에 추가하여 준다(<그림 5>).

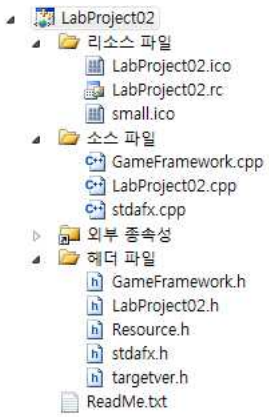


그림 5. 추가된 클래스

이 "CGameFramework" 클래스 객체는 게임 프로그램의 뼈대를 나타낸다(게임 프레임워크). 이 클래스는 Direct3D 디바이스를 생성하고 관리하며 화면 출력을 위한 여러 가지 처리(게임 객체의 생성과 관리, 사용자 입력, 애니메이션 등)를 담당한다.

다음은 LabProject 프로젝트에 추가된 "GameFramework.h"과 "GameFramework.cpp" 파일의 내용이다.

❶ "GameFramework.h" 파일

```
class CGameFramework
{
public:
    CGameFramework();
    ~CGameFramework();
};
```

❷ "GameFramework.cpp" 파일

```
#include "stdafx.h"
#include "GameFramework.h“

CGameFramework::CGameFramework()
{
}

CGameFramework::~CGameFramework()
{
}
```

앞으로 새로운 클래스를 추가하거나 파일을 추가할 때 이 방법을 사용하도록 한다.

⑥ “CGameFramework” 클래스 선언하기

이제 이 두 개의 파일을 수정하여 Direct3D 그래픽 프로그램의 기본 골격을 구현하도록 하자. 먼저, “GameFramework.h” 파일을 다음과 같이 수정한다.

```
class CGameFramework
{
private:
    HINSTANCE m_hInstance;
    HWND m_hWnd;

    int m_nWndClientWidth;
    int m_nWndClientHeight;

    IDXGIFactory4 *m_pdxgiFactory;
//DXGI 팩토리 인터페이스에 대한 포인터이다.
    IDXGISwapChain3 *m_pdxgiSwapChain;
//스왑 체인 인터페이스에 대한 포인터이다. 주로 디스플레이를 제어하기 위하여 필요하다.
    ID3D12Device *m_pd3dDevice;
//Direct3D 디바이스 인터페이스에 대한 포인터이다. 주로 리소스를 생성하기 위하여 필요하다.

    bool m_bMsaa4xEnable = false;
```

```
    UINT m_nMsaa4xQualityLevels = 0;
//MSAA 다중 샘플링을 활성화하고 다중 샘플링 레벨을 설정한다.

    static const UINT m_nSwapChainBuffers = 2;
//스왑 체인의 후면 버퍼의 개수이다.
    UINT m_nSwapChainBufferIndex;
//현재 스왑 체인의 후면 버퍼 인덱스이다.

    ID3D12Resource *m_ppd3dRenderTargetBuffers[m_nSwapChainBuffers];
    ID3D12DescriptorHeap *m_pd3dRtvDescriptorHeap;
    UINT m_nRtvDescriptorIncrementSize;
//렌더 타겟 버퍼, 서술자 힙 인터페이스 포인터, 렌더 타겟 서술자 원소의 크기이다.

    ID3D12Resource *m_pd3dDepthStencilBuffer;
    ID3D12DescriptorHeap *m_pd3dDsvDescriptorHeap;
    UINT m_nDsvDescriptorIncrementSize;
//깊이-스텐실 버퍼, 서술자 힙 인터페이스 포인터, 깊이-스텐실 서술자 원소의 크기이다.

    ID3D12CommandQueue *m_pd3dCommandQueue;
    ID3D12CommandAllocator *m_pd3dCommandAllocator;
    ID3D12GraphicsCommandList  *m_pd3dCommandList;
//명령 큐, 명령 할당자, 명령 리스트 인터페이스 포인터이다.

    ID3D12PipelineState *m_pd3dPipelineState;
//그래픽스 파이프라인 상태 객체에 대한 인터페이스 포인터이다.

    ID3D12Fence *m_pd3dFence;
    UINT64 m_nFenceValue;
    HANDLE m_hFenceEvent;
//펜스 인터페이스 포인터, 펜스의 값, 이벤트 핸들이다.

    D3D12_VIEWPORT m_d3dViewport;
    D3D12_RECT m_d3dScissorRect;
//뷰포트와 씨저 사각형이다.

public:
    CGameFramework();
    ~CGameFramework();

    bool OnCreate(HINSTANCE hInstance, HWND hMainWnd);
//프레임워크를 초기화하는 함수이다(주 윈도우가 생성되면 호출된다).
    void OnDestroy();

    void CreateSwapChain();
    void CreateRtvAndDsvDescriptorHeaps();
    void CreateDirect3DDevice();
    void CreateCommandQueueAndList();
//스왑 체인, 디바이스, 서술자 힙, 명령 큐/할당자/리스트를 생성하는 함수이다.

    void CreateRenderTargetViews();
    void CreateDepthStencilView();
//렌더 타겟 뷰와 깊이-스텐실 뷰를 생성하는 함수이다.
```

```
    void BuildObjects();
    void ReleaseObjects();
//렌더링할 메쉬와 게임 객체를 생성하고 소멸하는 함수이다.

//프레임워크의 핵심(사용자 입력, 애니메이션, 렌더링)을 구성하는 함수이다.
    void ProcessInput();
    void AnimateObjects();
    void FrameAdvance();

    void WaitForGpuComplete();
//CPU와 GPU를 동기화하는 함수이다.

    void OnProcessingMouseMessage(HWND hWnd, UINT nMessageID, WPARAM wParam, LPARAM
lParam);
    void OnProcessingKeyboardMessage(HWND hWnd, UINT nMessageID, WPARAM wParam, LPARAM
lParam);
    LRESULT CALLBACK OnProcessingWindowMessage(HWND hWnd, UINT nMessageID, WPARAM wParam,
LPARAM lParam);
//윈도우의 메시지(키보드, 마우스 입력)를 처리하는 함수이다.
};
```

⑦ “CGameFramework” 클래스 구현하기

“GameFramework.cpp” 파일을 다음과 같이 수정한다.

❶ 생성자를 다음과 같이 수정한다.

```
CGameFramework::CGameFramework()
{
    m_pdxgiFactory = NULL;
    m_pdxgiSwapChain = NULL;
    m_pd3dDevice = NULL;

    m_pd3dCommandAllocator = NULL;
    m_pd3dCommandQueue = NULL;
    m_pd3dPipelineState = NULL;
    m_pd3dCommandList = NULL;

    for (int i = 0; i < m_nSwapChainBuffers; i++) m_ppd3dRenderTargetBuffers[i] = NULL;
    m_pd3dRtvDescriptorHeap = NULL;
    m_nRtvDescriptorIncrementSize = 0;

    m_pd3dDepthStencilBuffer = NULL;
    m_pd3dDsvDescriptorHeap = NULL;
    m_nDsvDescriptorIncrementSize = 0;

    m_nSwapChainBufferIndex = 0;

    m_hFenceEvent = NULL;
    m_pd3dFence = NULL;
    m_nFenceValue = 0;

    m_nWndClientWidth = FRAME_BUFFER_WIDTH;
    m_nWndClientHeight = FRAME_BUFFER_HEIGHT;
```

```
}
```

❷ OnCreate() 함수와 OnDestroy() 함수를 다음과 같이 정의한다.

```
//다음 함수는 응용 프로그램이 실행되어 주 윈도우가 생성되면 호출된다는 것에 유의하라.
bool CGameFramework::OnCreate(HINSTANCE hInstance, HWND hMainWnd)
{
   m_hInstance = hInstance;
   m_hWnd = hMainWnd;

//Direct3D 디바이스, 명령 큐와 명령 리스트, 스왑 체인 등을 생성하는 함수를 호출한다.
   CreateDirect3DDevice();
   CreateCommandQueueAndList();
   CreateSwapChain();
   CreateRtvAndDsvDescriptorHeaps();
   CreateRenderTargetViews();
   CreateDepthStencilView();

   BuildObjects();
//렌더링할 게임 객체를 생성한다.

   return(true);
}

void CGameFramework::OnDestroy()
{
   WaitForGpuComplete();
//GPU가 모든 명령 리스트를 실행할 때 까지 기다린다.

   ReleaseObjects();
//게임 객체(게임 월드 객체)를 소멸한다.

   ::CloseHandle(m_hFenceEvent);

   for (int i = 0; i < m_nSwapChainBuffers; i++) if (m_ppd3dRenderTargetBuffers[i])
m_ppd3dRenderTargetBuffers[i]->Release();
   if (m_pd3dRtvDescriptorHeap) m_pd3dRtvDescriptorHeap->Release();

   if (m_pd3dDepthStencilBuffer) m_pd3dDepthStencilBuffer->Release();
   if (m_pd3dDsvDescriptorHeap) m_pd3dDsvDescriptorHeap->Release();

   if (m_pd3dCommandAllocator) m_pd3dCommandAllocator->Release();
   if (m_pd3dCommandQueue) m_pd3dCommandQueue->Release();
   if (m_pd3dPipelineState) m_pd3dPipelineState->Release();
   if (m_pd3dCommandList) m_pd3dCommandList->Release();

   if (m_pd3dFence) m_pd3dFence->Release();

   m_pdxgiSwapChain->SetFullscreenState(FALSE, NULL);
   if (m_pdxgiSwapChain) m_pdxgiSwapChain->Release();
   if (m_pd3dDevice) m_pd3dDevice->Release();
   if (m_pdxgiFactory) m_pdxgiFactory->Release();

#if defined(_DEBUG)
```

```
   IDXGIDebug1 *pdxgiDebug = NULL;
   DXGIGetDebugInterface1(0, __uuidof(IDXGIDebug1), (void **)&pdxgiDebug);
   HRESULT hResult = pdxgiDebug->ReportLiveObjects(DXGI_DEBUG_ALL,
DXGI_DEBUG_RLO_DETAIL);
   pdxgiDebug->Release();
#endif
}
```

❸ CreateSwapChain() 함수를 다음과 같이 정의한다.

```
void CGameFramework::CreateSwapChain()
{
   RECT rcClient;
   ::GetClientRect(m_hWnd, &rcClient);
   m_nWndClientWidth = rcClient.right - rcClient.left;
   m_nWndClientHeight = rcClient.bottom - rcClient.top;

   DXGI_SWAP_CHAIN_DESC1 dxgiSwapChainDesc;
   ::ZeroMemory(&dxgiSwapChainDesc, sizeof(DXGI_SWAP_CHAIN_DESC1));
   dxgiSwapChainDesc.Width = m_nWndClientWidth;
   dxgiSwapChainDesc.Height = m_nWndClientHeight;
   dxgiSwapChainDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
   dxgiSwapChainDesc.SampleDesc.Count = (m_bMsaa4xEnable) ? 4 : 1;
   dxgiSwapChainDesc.SampleDesc.Quality = (m_bMsaa4xEnable) ? (m_nMsaa4xQualityLevels -
1) : 0;
   dxgiSwapChainDesc.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
   dxgiSwapChainDesc.BufferCount = m_nSwapChainBuffers;
   dxgiSwapChainDesc.Scaling = DXGI_SCALING_NONE;
   dxgiSwapChainDesc.SwapEffect = DXGI_SWAP_EFFECT_FLIP_DISCARD;
   dxgiSwapChainDesc.AlphaMode = DXGI_ALPHA_MODE_UNSPECIFIED;
   dxgiSwapChainDesc.Flags = 0;

   DXGI_SWAP_CHAIN_FULLSCREEN_DESC dxgiSwapChainFullScreenDesc;
   ::ZeroMemory(&dxgiSwapChainFullScreenDesc, sizeof(DXGI_SWAP_CHAIN_FULLSCREEN_DESC));
   dxgiSwapChainFullScreenDesc.RefreshRate.Numerator = 60;
   dxgiSwapChainFullScreenDesc.RefreshRate.Denominator = 1;
   dxgiSwapChainFullScreenDesc.ScanlineOrdering = DXGI_MODE_SCANLINE_ORDER_UNSPECIFIED;
   dxgiSwapChainFullScreenDesc.Scaling = DXGI_MODE_SCALING_UNSPECIFIED;
   dxgiSwapChainFullScreenDesc.Windowed = TRUE;

   m_pdxgiFactory->CreateSwapChainForHwnd(m_pd3dCommandQueue, m_hWnd,
&dxgiSwapChainDesc, &dxgiSwapChainFullScreenDesc, NULL, (IDXGISwapChain1
**)&m_pdxgiSwapChain);
//스왑체인을 생성한다.

   m_pdxgiFactory->MakeWindowAssociation(m_hWnd, DXGI_MWA_NO_ALT_ENTER);
//"Alt+Enter" 키의 동작을 비활성화한다.
   m_nSwapChainBufferIndex = m_pdxgiSwapChain->GetCurrentBackBufferIndex();
//스왑체인의 현재 후면버퍼 인덱스를 저장한다.
}
```

❹ CreateDirect3DDisplay() 함수를 다음과 같이 정의한다.

```
void CGameFramework::CreateDirect3DDevice()
```

```
{
   HRESULT hResult;

   UINT nDXGIFactoryFlags = 0;
#if defined(_DEBUG)
   ID3D12Debug *pd3dDebugController = NULL;
   hResult = D3D12GetDebugInterface(__uuidof(ID3D12Debug), (void
**)&pd3dDebugController);
   if (pd3dDebugController)
   {
      pd3dDebugController->EnableDebugLayer();
      pd3dDebugController->Release();
   }
   nDXGIFactoryFlags |= DXGI_CREATE_FACTORY_DEBUG;
#endif

   hResult = ::CreateDXGIFactory2(nDXGIFactoryFlags, __uuidof(IDXGIFactory4), (void
**)&m_pdxgiFactory);

   IDXGIAdapter1 *pd3dAdapter = NULL;
   for (UINT i = 0; DXGI_ERROR_NOT_FOUND != m_pdxgiFactory->EnumAdapters1(i,
&pd3dAdapter); i++)
   {
      DXGI_ADAPTER_DESC1 dxgiAdapterDesc;
      pd3dAdapter->GetDesc1(&dxgiAdapterDesc);
      if (dxgiAdapterDesc.Flags & DXGI_ADAPTER_FLAG_SOFTWARE) continue;
      if (SUCCEEDED(D3D12CreateDevice(pd3dAdapter, D3D_FEATURE_LEVEL_12_0,
_uuidof(ID3D12Device), (void **)&m_pd3dDevice))) break;
   }
//모든 하드웨어 어댑터 대하여 특성 레벨 12.0을 지원하는 하드웨어 디바이스를 생성한다.

   if (!pd3dAdapter)
   {
      m_pdxgiFactory->EnumWarpAdapter(_uuidof(IDXGIFactory4), (void **)&pd3dAdapter);
      D3D12CreateDevice(pd3dAdapter, D3D_FEATURE_LEVEL_11_0, _uuidof(ID3D12Device), (void
**)&m_pd3dDevice);
   }
//특성 레벨 12.0을 지원하는 하드웨어 디바이스를 생성할 수 없으면 WARP 디바이스를 생성한다.

   D3D12_FEATURE_DATA_MULTISAMPLE_QUALITY_LEVELS d3dMsaaQualityLevels;
   d3dMsaaQualityLevels.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
   d3dMsaaQualityLevels.SampleCount = 4; //Msaa4x 다중 샘플링
   d3dMsaaQualityLevels.Flags = D3D12_MULTISAMPLE_QUALITY_LEVELS_FLAG_NONE;
   d3dMsaaQualityLevels.NumQualityLevels = 0;
   m_pd3dDevice->CheckFeatureSupport(D3D12_FEATURE_MULTISAMPLE_QUALITY_LEVELS,
&d3dMsaaQualityLevels, sizeof(D3D12_FEATURE_DATA_MULTISAMPLE_QUALITY_LEVELS));
   m_nMsaa4xQualityLevels = d3dMsaaQualityLevels.NumQualityLevels;
//디바이스가 지원하는 다중 샘플의 품질 수준을 확인한다.
   m_bMsaa4xEnable = (m_nMsaa4xQualityLevels > 1) ? true : false;
//다중 샘플의 품질 수준이 1보다 크면 다중 샘플링을 활성화한다.

   hResult = m_pd3dDevice->CreateFence(0, D3D12_FENCE_FLAG_NONE, __uuidof(ID3D12Fence),
(void **)&m_pd3dFence);
   m_nFenceValue = 0;
```

```
//펜스를 생성하고 펜스 값을 0으로 설정한다.

   m_hFenceEvent = ::CreateEvent(NULL, FALSE, FALSE, NULL);
/*펜스와 동기화를 위한 이벤트 객체를 생성한다(이벤트 객체의 초기값을 FALSE이다). 이벤트가 실행되면(Signal) 이
벤트의 값을 자동적으로 FALSE가 되도록 생성한다.*/

   m_d3dViewport.TopLeftX = 0;
   m_d3dViewport.TopLeftY = 0;
   m_d3dViewport.Width = static_cast<float>(m_nWndClientWidth);
   m_d3dViewport.Height = static_cast<float>(m_nWndClientHeight);
   m_d3dViewport.MinDepth = 0.0f;
   m_d3dViewport.MaxDepth = 1.0f;
//뷰포트를 주 윈도우의 클라이언트 영역 전체로 설정한다.

   m_d3dScissorRect = { 0, 0, m_nWndClientWidth, m_nWndClientHeight };
//씨저 사각형을 주 윈도우의 클라이언트 영역 전체로 설정한다.

   if (pd3dAdapter) pd3dAdapter->Release();
}
```

❺ CreateCommandQueueAndList() 함수를 다음과 같이 정의한다.

```
void CGameFramework::CreateCommandQueueAndList()
{
   D3D12_COMMAND_QUEUE_DESC d3dCommandQueueDesc;
   ::ZeroMemory(&d3dCommandQueueDesc, sizeof(D3D12_COMMAND_QUEUE_DESC));
   d3dCommandQueueDesc.Flags = D3D12_COMMAND_QUEUE_FLAG_NONE;
   d3dCommandQueueDesc.Type = D3D12_COMMAND_LIST_TYPE_DIRECT;
   HRESULT hResult = m_pd3dDevice->CreateCommandQueue(&d3dCommandQueueDesc,
_uuidof(ID3D12CommandQueue), (void **)&m_pd3dCommandQueue);
//직접(Direct) 명령 큐를 생성한다.

   hResult = m_pd3dDevice->CreateCommandAllocator(D3D12_COMMAND_LIST_TYPE_DIRECT,
__uuidof(ID3D12CommandAllocator), (void **)&m_pd3dCommandAllocator);
//직접(Direct) 명령 할당자를 생성한다.

   hResult = m_pd3dDevice->CreateCommandList(0, D3D12_COMMAND_LIST_TYPE_DIRECT,
m_pd3dCommandAllocator, NULL, __uuidof(ID3D12GraphicsCommandList), (void
**)&m_pd3dCommandList);
//직접(Direct) 명령 리스트를 생성한다.

   hResult = m_pd3dCommandList->Close();
//명령 리스트는 생성되면 열린(Open) 상태이므로 닫힌(Closed) 상태로 만든다.
}
```

❻ CreateRtvAndDsvDescriptorHeaps() 함수를 다음과 같이 정의한다.

```
void CGameFramework::CreateRtvAndDsvDescriptorHeaps()
{
   D3D12_DESCRIPTOR_HEAP_DESC d3dDescriptorHeapDesc;
   ::ZeroMemory(&d3dDescriptorHeapDesc, sizeof(D3D12_DESCRIPTOR_HEAP_DESC));
   d3dDescriptorHeapDesc.NumDescriptors = m_nSwapChainBuffers;
   d3dDescriptorHeapDesc.Type = D3D12_DESCRIPTOR_HEAP_TYPE_RTV;
```

```
   d3dDescriptorHeapDesc.Flags = D3D12_DESCRIPTOR_HEAP_FLAG_NONE;
   d3dDescriptorHeapDesc.NodeMask = 0;
   HRESULT hResult = m_pd3dDevice->CreateDescriptorHeap(&d3dDescriptorHeapDesc,
__uuidof(ID3D12DescriptorHeap), (void **)&m_pd3dRtvDescriptorHeap);
//렌더 타겟 서술자 힙(서술자의 개수는 스왑체인 버퍼의 개수)을 생성한다.
   m_nRtvDescriptorIncrementSize =
m_pd3dDevice->GetDescriptorHandleIncrementSize(D3D12_DESCRIPTOR_HEAP_TYPE_RTV);
//렌더 타겟 서술자 힙의 원소의 크기를 저장한다.

   d3dDescriptorHeapDesc.NumDescriptors = 1;
   d3dDescriptorHeapDesc.Type = D3D12_DESCRIPTOR_HEAP_TYPE_DSV;
   hResult = m_pd3dDevice->CreateDescriptorHeap(&d3dDescriptorHeapDesc,
__uuidof(ID3D12DescriptorHeap), (void **)&m_pd3dDsvDescriptorHeap);
//깊이-스텐실 서술자 힙(서술자의 개수는 1)을 생성한다.
   m_nDsvDescriptorIncrementSize =
m_pd3dDevice->GetDescriptorHandleIncrementSize(D3D12_DESCRIPTOR_HEAP_TYPE_DSV);
//깊이-스텐실 서술자 힙의 원소의 크기를 저장한다.
}
```

❼ CreateRenderTargetView()와 CreateDepthStencilView() 함수를 다음과 같이 정의한다.

```
//스왑체인의 각 후면 버퍼에 대한 렌더 타겟 뷰를 생성한다.
void CGameFramework::CreateRenderTargetView()
{
   D3D12_CPU_DESCRIPTOR_HANDLE d3dRtvCPUDescriptorHandle =
m_pd3dRtvDescriptorHeap->GetCPUDescriptorHandleForHeapStart();
   for (UINT i = 0; i < m_nSwapChainBuffers; i++)
   {
      m_pdxgiSwapChain->GetBuffer(i, __uuidof(ID3D12Resource), (void
**)&m_ppd3dRenderTargetBuffers[i]);
      m_pd3dDevice->CreateRenderTargetView(m_ppd3dRenderTargetBuffers[i], NULL,
d3dRtvCPUDescriptorHandle);
      d3dRtvCPUDescriptorHandle.ptr += m_nRtvDescriptorIncrementSize;
   }
}

void CGameFramework::CreateDepthStencilView()
{
   D3D12_RESOURCE_DESC d3dResourceDesc;
   d3dResourceDesc.Dimension = D3D12_RESOURCE_DIMENSION_TEXTURE2D;
   d3dResourceDesc.Alignment = 0;
   d3dResourceDesc.Width = m_nWndClientWidth;
   d3dResourceDesc.Height = m_nWndClientHeight;
   d3dResourceDesc.DepthOrArraySize = 1;
   d3dResourceDesc.MipLevels = 1;
   d3dResourceDesc.Format = DXGI_FORMAT_D24_UNORM_S8_UINT;
   d3dResourceDesc.SampleDesc.Count = (m_bMsaa4xEnable) ? 4 : 1;
   d3dResourceDesc.SampleDesc.Quality = (m_bMsaa4xEnable) ? (m_nMsaa4xQualityLevels - 1)
: 0;
   d3dResourceDesc.Layout = D3D12_TEXTURE_LAYOUT_UNKNOWN;
   d3dResourceDesc.Flags = D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL;

   D3D12_HEAP_PROPERTIES d3dHeapProperties;
   ::ZeroMemory(&d3dHeapProperties, sizeof(D3D12_HEAP_PROPERTIES));
```

```
   d3dHeapProperties.Type = D3D12_HEAP_TYPE_DEFAULT;
   d3dHeapProperties.CPUPageProperty = D3D12_CPU_PAGE_PROPERTY_UNKNOWN;
   d3dHeapProperties.MemoryPoolPreference = D3D12_MEMORY_POOL_UNKNOWN;
   d3dHeapProperties.CreationNodeMask = 1;
   d3dHeapProperties.VisibleNodeMask = 1;

   D3D12_CLEAR_VALUE d3dClearValue;
   d3dClearValue.Format = DXGI_FORMAT_D24_UNORM_S8_UINT;
   d3dClearValue.DepthStencil.Depth = 1.0f;
   d3dClearValue.DepthStencil.Stencil = 0;
   m_pd3dDevice->CreateCommittedResource(&d3dHeapProperties, D3D12_HEAP_FLAG_NONE,
&d3dResourceDesc, D3D12_RESOURCE_STATE_DEPTH_WRITE, &d3dClearValue,
__uuidof(ID3D12Resource), (void **)&m_pd3dDepthStencilBuffer);
//깊이-스텐실 버퍼를 생성한다.

   D3D12_CPU_DESCRIPTOR_HANDLE d3dDsvCPUDescriptorHandle =
m_pd3dDsvDescriptorHeap->GetCPUDescriptorHandleForHeapStart();
   m_pd3dDevice->CreateDepthStencilView(m_pd3dDepthStencilBuffer, NULL,
d3dDsvCPUDescriptorHandle);
//깊이-스텐실 버퍼 뷰를 생성한다.
}
```

❽ BuildObjects(), ReleaseObjects(), OnProcessingWindowMessage(), ProcessInput(), AnimateObjects(), OnProcessingMouseMessage(), OnProcessingKeyboardMessage() 함수를 다음과 같이 정의한다.

```
void CGameFramework::BuildObjects()
{
}

void CGameFramework::ReleaseObjects()
{
}

void CGameFramework::OnProcessingMouseMessage(HWND hWnd, UINT nMessageID, WPARAM wParam,
LPARAM lParam)
{
   switch (nMessageID)
   {
      case WM_LBUTTONDOWN:
      case WM_RBUTTONDOWN:
         break;
      case WM_LBUTTONUP:
      case WM_RBUTTONUP:
         break;
      case WM_MOUSEMOVE:
         break;
      default:
         break;
   }
}

void CGameFramework::OnProcessingKeyboardMessage(HWND hWnd, UINT nMessageID, WPARAM
wParam, LPARAM lParam)
```

```
{
   switch (nMessageID)
   {
      case WM_KEYUP:
         switch (wParam)
         {
            case VK_ESCAPE:
               ::PostQuitMessage(0);
               break;
            case VK_RETURN:
               break;
            case VK_F8:
               break;
            case VK_F9:
               break;
            default:
               break;
         }
         break;
      default:
         break;
   }
}

LRESULT CALLBACK CGameFramework::OnProcessingWindowMessage(HWND hWnd, UINT nMessageID,
WPARAM wParam, LPARAM lParam)
{
   switch (nMessageID)
   {
      case WM_SIZE:
      {
         m_nWndClientWidth = LOWORD(lParam);
         m_nWndClientHeight = HIWORD(lParam);
         break;
      }
      case WM_LBUTTONDOWN:
      case WM_RBUTTONDOWN:
      case WM_LBUTTONUP:
      case WM_RBUTTONUP:
      case WM_MOUSEMOVE:
         OnProcessingMouseMessage(hWnd, nMessageID, wParam, lParam);
         break;
      case WM_KEYDOWN:
      case WM_KEYUP:
         OnProcessingKeyboardMessage(hWnd, nMessageID, wParam, lParam);
         break;
   }
   return(0);
}

void CGameFramework::ProcessInput()
{
}

void CGameFramework::AnimateObjects()
{
```

```
}
```

❾ WaitForGpuComplete() 함수를 다음과 같이 정의한다.

```
void CGameFramework::WaitForGpuComplete()
{
   m_nFenceValue++;
//CPU 펜스의 값을 증가한다.
   const UINT64 nFence = m_nFenceValue;
   HRESULT hResult = m_pd3dCommandQueue->Signal(m_pd3dFence, nFence);
//GPU가 펜스의 값을 설정하는 명령을 명령 큐에 추가한다.
   if (m_pd3dFence->GetCompletedValue() < nFence)
   {
//펜스의 현재 값이 설정한 값보다 작으면 펜스의 현재 값이 설정한 값이 될 때까지 기다린다.
      hResult = m_pd3dFence->SetEventOnCompletion(nFence, m_hFenceEvent);
      ::WaitForSingleObject(m_hFenceEvent, INFINITE);
   }
}
```

❿ FrameAdvance() 함수를 다음과 같이 정의한다.

```
void CGameFramework::FrameAdvance()
{
   ProcessInput();

   AnimateObjects();

   HRESULT hResult = m_pd3dCommandAllocator->Reset();
   hResult = m_pd3dCommandList->Reset(m_pd3dCommandAllocator, NULL);
//명령 할당자와 명령 리스트를 리셋한다.

   D3D12_RESOURCE_BARRIER d3dResourceBarrier;
   ::ZeroMemory(&d3dResourceBarrier, sizeof(D3D12_RESOURCE_BARRIER));
   d3dResourceBarrier.Type = D3D12_RESOURCE_BARRIER_TYPE_TRANSITION;
   d3dResourceBarrier.Flags = D3D12_RESOURCE_BARRIER_FLAG_NONE;
   d3dResourceBarrier.Transition.pResource =
m_ppd3dRenderTargetBuffers[m_nSwapChainBufferIndex];
   d3dResourceBarrier.Transition.StateBefore = D3D12_RESOURCE_STATE_PRESENT;
   d3dResourceBarrier.Transition.StateAfter = D3D12_RESOURCE_STATE_RENDER_TARGET;
   d3dResourceBarrier.Transition.Subresource = D3D12_RESOURCE_BARRIER_ALL_SUBRESOURCES;
   m_pd3dCommandList->ResourceBarrier(1, &d3dResourceBarrier);
/*현재 렌더 타겟에 대한 프리젠트가 끝나기를 기다린다. 프리젠트가 끝나면 렌더 타겟 버퍼의 상태는 프리젠트 상태
(D3D12_RESOURCE_STATE_PRESENT)에서 렌더 타겟 상태(D3D12_RESOURCE_STATE_RENDER_TARGET)로 바
뀔 것이다.*/

   m_pd3dCommandList->RSSetViewports(1, &m_d3dViewport);
   m_pd3dCommandList->RSSetScissorRects(1, &m_d3dScissorRect);
//뷰포트와 씨저 사각형을 설정한다.

   D3D12_CPU_DESCRIPTOR_HANDLE d3dRtvCPUDescriptorHandle =
m_pd3dRtvDescriptorHeap->GetCPUDescriptorHandleForHeapStart();
   d3dRtvCPUDescriptorHandle.ptr += (m_nSwapChainBufferIndex *
m_nRtvDescriptorIncrementSize);
```

```
//현재의 렌더 타겟에 해당하는 서술자의 CPU 주소(핸들)를 계산한다.

   float pfClearColor[4] = { 0.0f, 0.125f, 0.3f, 1.0f };
   m_pd3dCommandList->ClearRenderTargetView(d3dRtvCPUDescriptorHandle,
pfClearColor/*Colors::Azure*/, 0, NULL);
//원하는 색상으로 렌더 타겟(뷰)을 지운다.

   D3D12_CPU_DESCRIPTOR_HANDLE d3dDsvCPUDescriptorHandle =
m_pd3dDsvDescriptorHeap->GetCPUDescriptorHandleForHeapStart();
//깊이-스텐실 서술자의 CPU 주소를 계산한다.
   m_pd3dCommandList->ClearDepthStencilView(d3dDsvCPUDescriptorHandle,
D3D12_CLEAR_FLAG_DEPTH | D3D12_CLEAR_FLAG_STENCIL, 1.0f, 0, 0, NULL);
//원하는 값으로 깊이-스텐실(뷰)을 지운다.

   m_pd3dCommandList->OMSetRenderTargets(1, &d3dRtvCPUDescriptorHandle, TRUE,
&d3dDsvCPUDescriptorHandle);
//렌더 타겟 뷰(서술자)와 깊이-스텐실 뷰(서술자)를 출력-병합 단계(OM)에 연결한다.

//렌더링 코드는 여기에 추가될 것이다.

   d3dResourceBarrier.Transition.StateBefore = D3D12_RESOURCE_STATE_RENDER_TARGET;
   d3dResourceBarrier.Transition.StateAfter = D3D12_RESOURCE_STATE_PRESENT;
   d3dResourceBarrier.Transition.Subresource = D3D12_RESOURCE_BARRIER_ALL_SUBRESOURCES;
   m_pd3dCommandList->ResourceBarrier(1, &d3dResourceBarrier);
/*현재 렌더 타겟에 대한 렌더링이 끝나기를 기다린다. GPU가 렌더 타겟(버퍼)을 더 이상 사용하지 않으면 렌더 타겟
의 상태는 프리젠트 상태(D3D12_RESOURCE_STATE_PRESENT)로 바뀔 것이다.*/

   hResult = m_pd3dCommandList->Close();
//명령 리스트를 닫힌 상태로 만든다.

   ID3D12CommandList *ppd3dCommandLists[] = { m_pd3dCommandList };
   m_pd3dCommandQueue->ExecuteCommandLists(1, ppd3dCommandLists);
//명령 리스트를 명령 큐에 추가하여 실행한다.

   WaitForGpuComplete();
//GPU가 모든 명령 리스트를 실행할 때 까지 기다린다.

   DXGI_PRESENT_PARAMETERS dxgiPresentParameters;
   dxgiPresentParameters.DirtyRectsCount = 0;
   dxgiPresentParameters.pDirtyRects = NULL;
   dxgiPresentParameters.pScrollRect = NULL;
   dxgiPresentParameters.pScrollOffset = NULL;
   m_pdxgiSwapChain->Present1(1, 0, &dxgiPresentParameters);
/*스왑체인을 프리젠트한다. 프리젠트를 하면 현재 렌더 타겟(후면버퍼)의 내용이 전면버퍼로 옮겨지고 렌더 타겟 인
덱스가 바뀔 것이다.*/

   m_nSwapChainBufferIndex = m_pdxgiSwapChain->GetCurrentBackBufferIndex();
}
```

## □ LabProject02.cpp 파일 변경

① "LabProject02.cpp" 파일에서 헤더 파일의 포함 부분 마지막에 다음을 추가한다.

❶ CGameFramework 클래스의 헤더파일을 포함시킨다.

```
#include "LabProject02.h"
#include "GameFramework.h"
```

❷ 다음 줄에 CGameFramework 클래스 객체를 전역변수로 선언한다. 이 객체는 게임 프로그램의 골격을 나타내는 객체이다.

```
CGameFramework gGameFramework;
```

② "LabProject02.cpp" 파일에서 WinMain() 함수의 메시지 루프(Message Loop) 부분을 다음과 같이 변경한다.

```
while (1)
{
   if (::PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
   {
      if (msg.message == WM_QUIT) break;
      if (!::TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
      {
         ::TranslateMessage(&msg);
         ::DispatchMessage(&msg);
      }
   }
   else
   {
      gGameFramework.FrameAdvance();
   }
}
gGameFramework.OnDestroy();
```

응용 프로그램 마법사가 생성한 코드의 메시지 루프는 응용 프로그램이 처리해야 할 윈도우 메시지가 메시지 큐에 있으면 꺼내와서 처리를 하고 메시지가 없으면 CPU를 운영체제로 반납하도록 되어있다(GetMessage() API 함수). 그러나 게임 프로그램은 프로그램이 처리할 메시지가 없더라도 화면 렌더링, 사용자 입력처리, 길찾기 등의 작업이 계속 진행되어야 한다. 그러므로 만약 처리할 메시지가 없더라도 CPU를 반납하지 않고 게임이 계속 진행되도록 해야 한다. 이를 위해서 윈도우 메시지 루프를 PeekMessage() API 함수를 사용하여 변경한다. PeekMessage() API 함수는 메시지 큐를 살펴보고 메시지가 있으면 메시지를 꺼내고 TRUE를 반환한다. 만약 메시지 큐에 메시지가 없으면 FALSE를 반환한다. 그러므로 PeekMessage() 함수가 TRUE를 반환하는 경우(응용 프로그램이 처리해야 할 윈도우 메시지가 메시지 큐에 있으면) 정상적인 윈도우 메시지 처리 과정을 수행해야 한다. 그러나 FALSE를 반환하는 경우(메시지 큐가 비어있으면) gGameFramework.FrameAdvance() 함수를 호출하여 게임 프로그램이 CPU를 사용할 수 있도록 해야 한다. 그리고 이 과정은 사용자가 프로그램을 종료할 때까지 계속 반복되도록 한다.

그리고 메시지 루프가 종료되면 gGameFramework.OnDestroy() 함수를 호출하여 프레임워크 객체를 소멸하도록 한다.

③ "LabProject02.cpp" 파일에서 MyRegisterClass() 함수를 다음과 같이 변경한다.

```
ATOM MyRegisterClass(HINSTANCE hInstance)
{
   WNDCLASSEX wcex;

   wcex.cbSize = sizeof(WNDCLASSEX);

   wcex.style = CS_HREDRAW | CS_VREDRAW;
   wcex.lpfnWndProc = WndProc;
   wcex.cbClsExtra = 0;
   wcex.cbWndExtra = 0;
   wcex.hInstance = hInstance;
   wcex.hIcon = ::LoadIcon(hInstance, MAKEINTRESOURCE(IDI_LABPROJECT02));
   wcex.hCursor = ::LoadCursor(NULL, IDC_ARROW);
   wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
//주 윈도우의 메뉴가 나타나지 않도록 한다.
   wcex.lpszMenuName = NULL;
   wcex.lpszClassName = szWindowClass;
   wcex.hIconSm = ::LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

   return ::RegisterClassEx(&wcex);
}
```

④ "LabProject02.cpp" 파일에서 InitInstance() 함수를 다음과 같이 변경한다.

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   ...
   RECT rc = { 0, 0, FRAME_BUFFER_WIDTH, FRAME_BUFFER_HEIGHT };
   DWORD dwStyle = WS_OVERLAPPED | WS_CAPTION | WS_MINIMIZEBOX | WS_SYSMENU | WS_BORDER;
   AdjustWindowRect(&rc, dwStyle, FALSE);
   HWND hMainWnd = CreateWindow(szWindowClass, szTitle, dwStyle, CW_USEDEFAULT,
CW_USEDEFAULT, rc.right - rc.left, rc.bottom - rc.top, NULL, NULL, hInstance, NULL);
   if (!hMainWnd) return(FALSE);

   gGameFramework.OnCreate(hInstance, hMainWnd);

   ::ShowWindow(hMainWnd, nCmdShow);
   ::UpdateWindow(hMainWnd);

   return(TRUE);
}
```

프로그램의 주 윈도우가 생성되면 CGameFramework 클래스의 OnCreate() 함수를 호출하여 프레임워크 객체를 초기화하도록 한다.

⑤ “LabProject02.cpp“ 파일에서 WndProc() 함수의 메시지 처리 부분을 다음과 같이 변경한다. 게임 프레임워크에서 처리할 메시지가 있으면 case 문장으로 추가한다. 현재는 WM_SIZE 메시지, 마우스 메시지, 키보드 메시지를 게임 프레임워크에서 처리하도록 한다.

```
...
   switch (message)
   {
      case WM_SIZE:
      case WM_LBUTTONDOWN:
      case WM_LBUTTONUP:
      case WM_RBUTTONDOWN:
      case WM_RBUTTONUP:
      case WM_MOUSEMOVE:
      case WM_KEYDOWN:
      case WM_KEYUP:
         gGameFramework.OnProcessingWindowMessage(hWnd, message, wParam, lParam);
         break;
      case WM_DESTROY:
         ::PostQuitMessage(0);
         break;
      default:
         return(::DefWindowProc(hWnd, message, wParam, lParam));
   }
   return 0;
}
```

* 앞으로 새로운 프로젝트를 생성하더라도 위에서 변경한 “LabProject02.cpp“ 파일의 내용은 거의 바뀌지 않을 것이다. 그러므로 앞으로 생성하는 새로운 프로젝트의 “LabProject**xx**.cpp“ 파일의 내용은 위의 내용을 그대로 복사해서 사용할 것이다.

## □ LabProject02 프로그램의 빌드 및 실행

이제 게임 프로그램의 프레임워크가 완성되었다. 앞으로는 이 프레임워크를 바탕으로 Direct3D의 새로운 요소들을 추가할 것이다.
프로젝트를 빌드하자(F7 키를 누른다).
별 문제없이 프로젝트를 빌드하고 실행할 수 있을 것이다(F5 키를 누른다).