MFC Windows 프로그래밍 기본편

핵심만 골라 배우는 MFC 프로그래밍

널널한 개발자 최호성 (cx8537@naver.com)
YouTube: 널널한 개발자 TV

문서 개정이력

버전	최종수정일	내용
1.0	2024-02-25	최초 작성

이미 알고 있어야 하는 것들

여기서 언급한 내용을 모르면 수업을 할 수 없음!

C언어 활용 능력

- 연결 리스트 기반 선형 자료구조 직접 구현
- 함수 포인터
- Call back 구조에 대한 이해
- 메모리에 대한 이해
  - 정적 메모리 영역
  - Stack, Heap

C++ 활용 능력

- 연산자 함수 재정의
- 추상 자료형
- 가상 함수
  - v-table
  - Called by framework
- 객체지향 주소록 2단계 이상
  - Event loop 적용
  - MVC 구조 적용

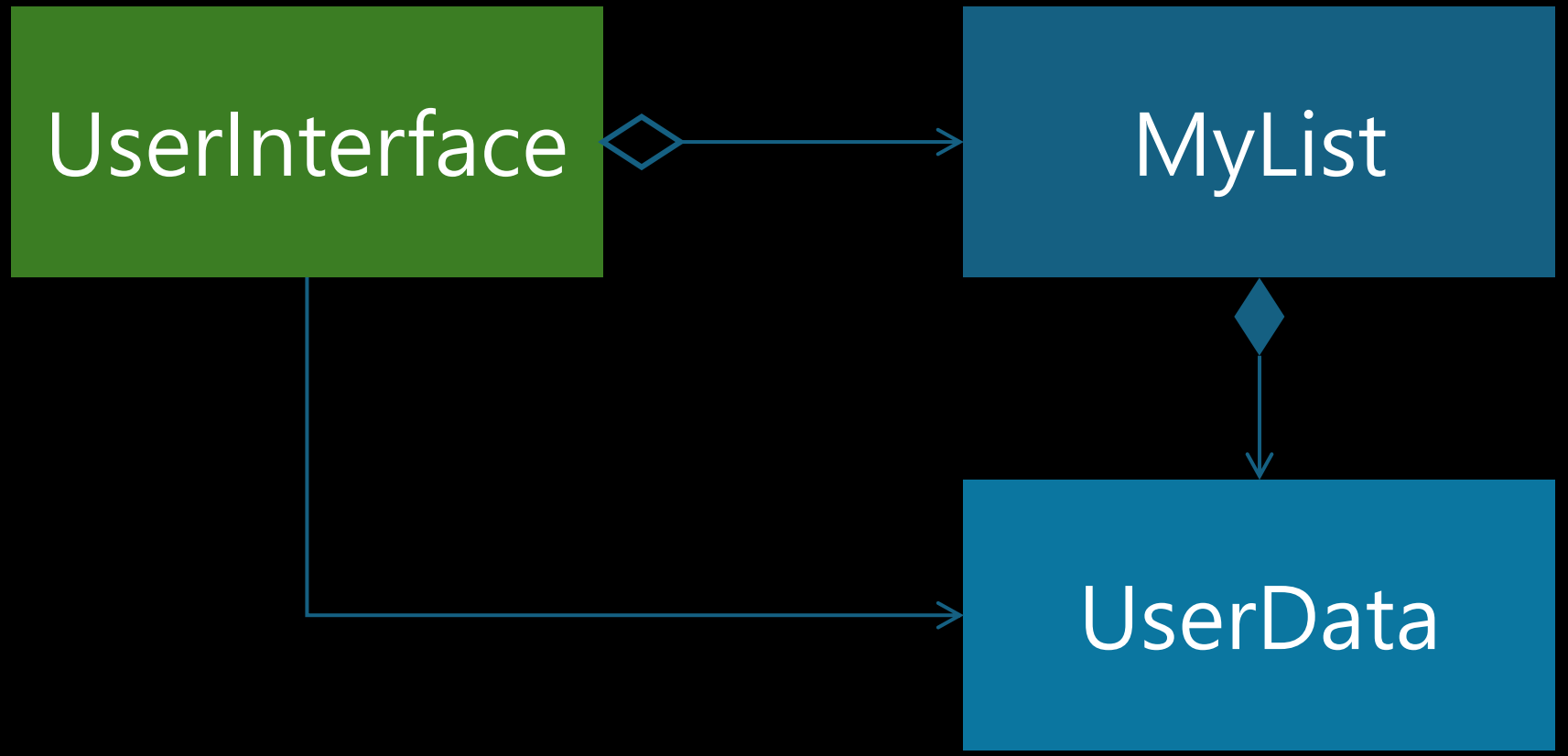
1단계: 기초적인 객체화

- 상속 관계 없이 기능 혹은 요소 단위로 코드를 분리
- 관리 대상 자료, 자료구조, UI 등 세 가지 요소를 클래스로 정의해 분리
- main() 함수에서는 세 클래스 인스턴스에 대한 관계를 규정하는 정도로 코드를 최소화

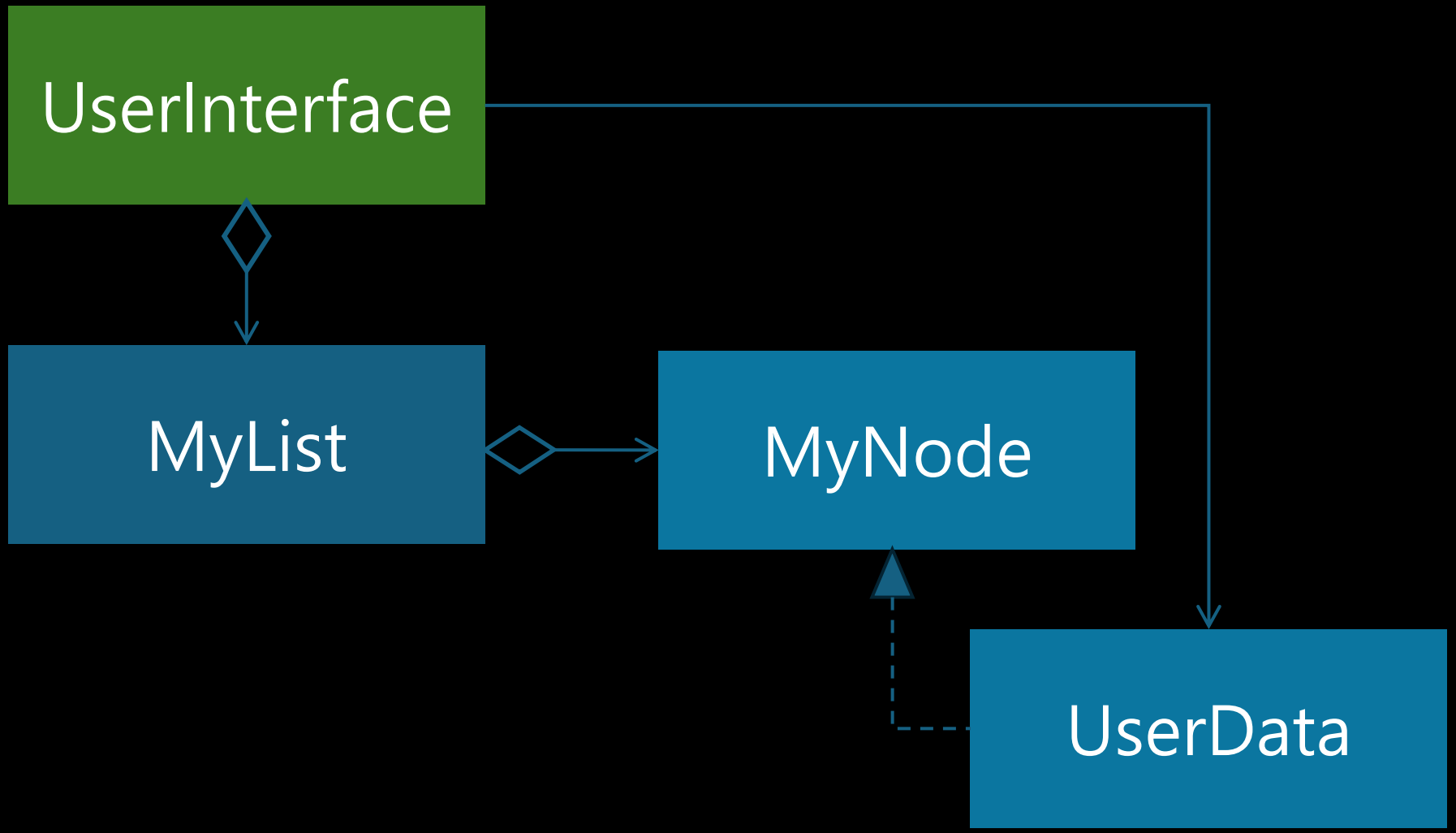
근본 없는 객체화 요령

- 자료구조, UI, 그 외 나머지 기능(제어체계)은 무조건 분리해 별도 클래스로 선언
- 입/출력 단위에 해당하는 자료가 존재한다면 그 단위 자체로 객체화
- 입/출력 단위 객체가 존재한다면 자료구조와 관리 대상 자료를 분리

1단계: 기초적인 객체화



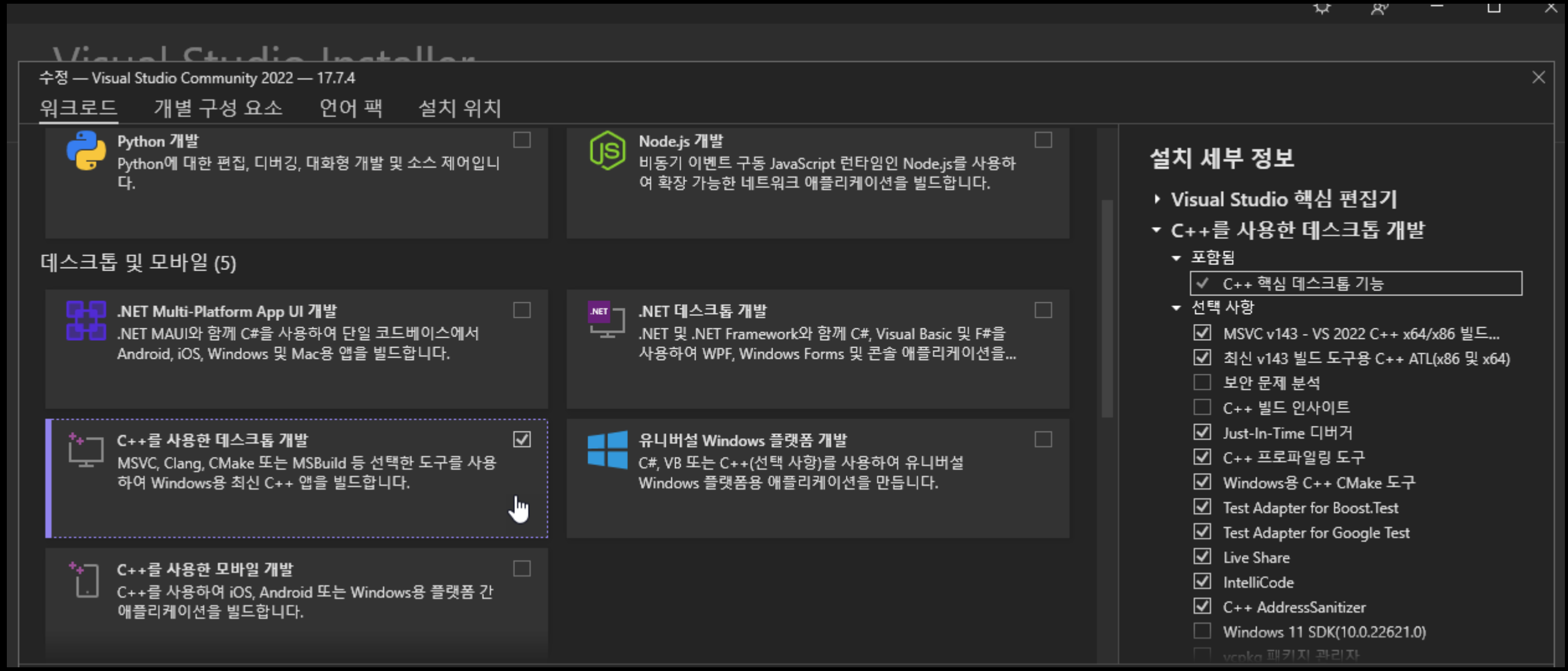
2단계: 컨테이너 구현



알고 있으면 도움 되는 것들

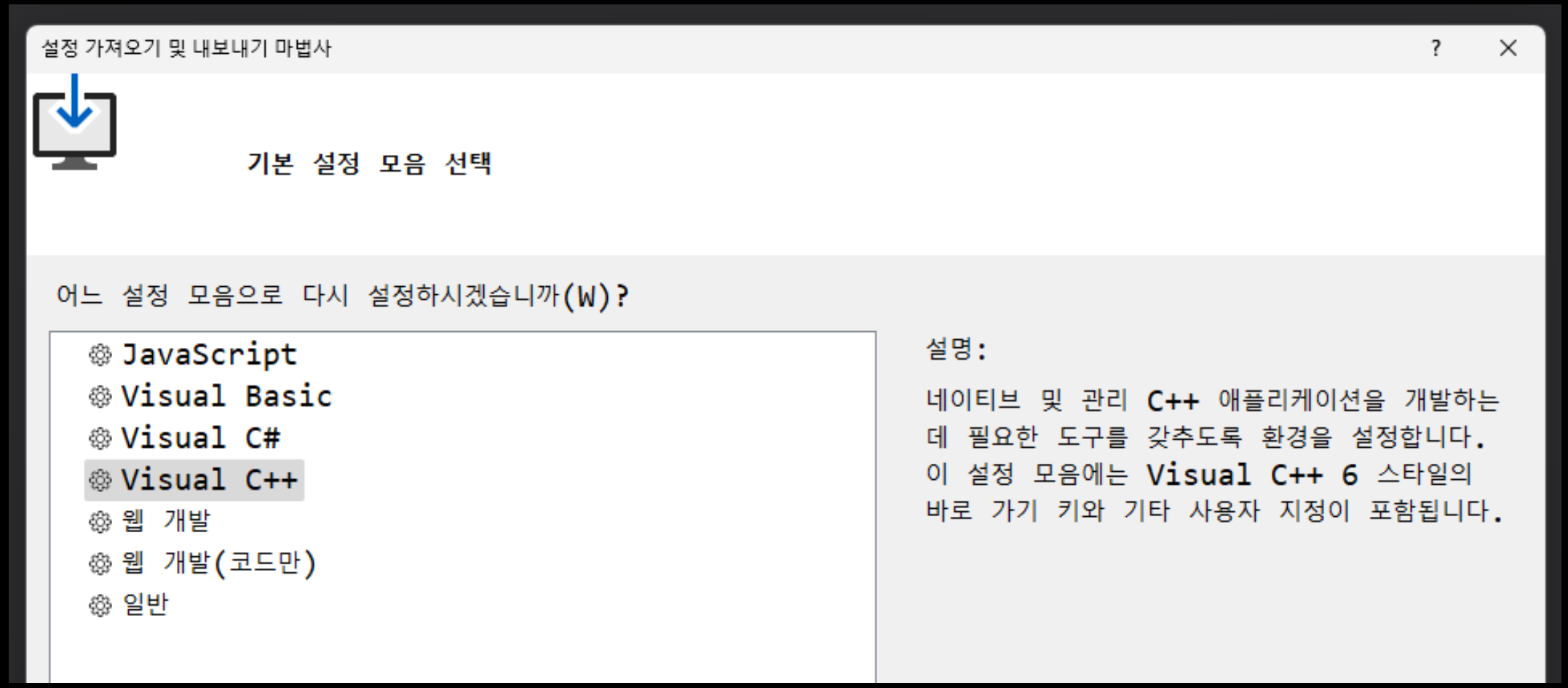
- Visual Studio 디버거 사용 방법 (C++ 기준)
  - Position/Data break point
  - Call stack
  - Memory window
  - Symbol file
- OOP 디자인 패턴
  - MVC(Model View Controller)
  - Singleton
  - Observer

C++를 사용한 데스크톱 개발 선택



Visual Studio 개발환경 설정

도구 > 설정 가져오기 및 내보내기



윈도우 프로그래밍 모델

Win32 API 기반 프로그램 구조 및 메시지 드라이븐 구조에 대해 학습합니다.

윈도우 메시지

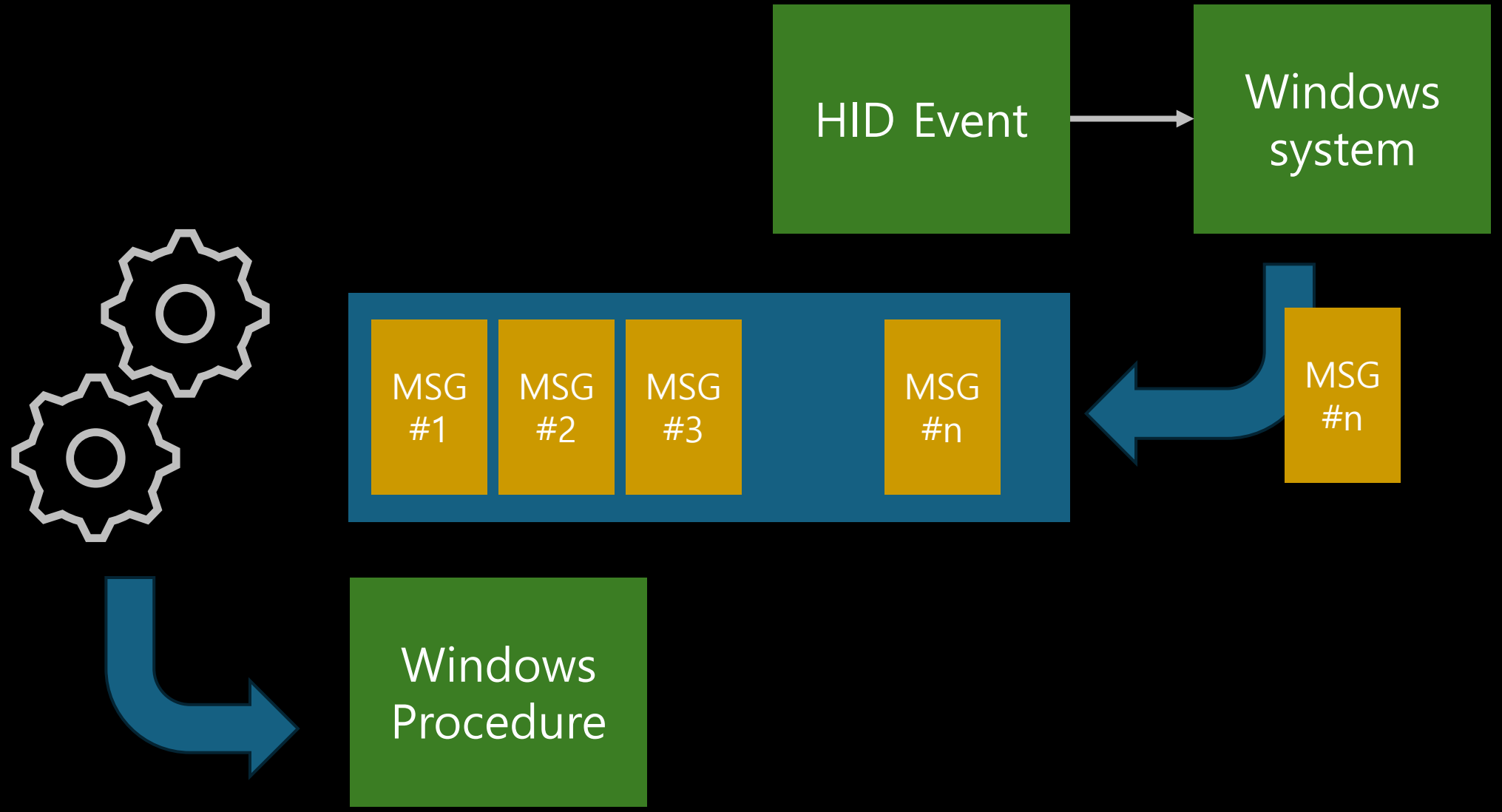
- WM_XXX형태로 정의된 부호가 없는 정수
- 모든 윈도우 메시지는 처리 함수로 전달될 때 메시지 자체와 관련된 매개변수(최대 2개)와 함께 전달 됨
  - MSG 구조체

```
/*
 * Message structure
 */
typedef struct tagMSG {
    HWND        hwnd;
    UINT        message;
    WPARAM      wParam;
    LPARAM      lParam;
    DWORD       time;
    POINT       pt;
#ifdef _MAC
    DWORD       lPrivate;
#endif
} MSG, *PMSG, NEAR *NPMSG, FAR *LPMSG;
```

메시지 드라이븐 방식

- 사용자 입력(키보드, 마우스 등) 이벤트를 OS가 감지한 후 이를 메시지로 변환해 해당 응용 프로그램(응용 프로그램 메시지 큐)에게 전달
- 메시지를 수신한 응용 프로그램은 메시지 큐에서 메시지를 하나씩 꺼낸 후 처리(함수 호출)하는 이벤트 루프 구조
- 응용 프로그램(Process)은 최소 1개 이상의 스레드로 구성되며 GUI를 갖는 응용 프로그램의 메인 스레드는 메시지 큐를 가짐

메시지 드라이븐 방식


HID Event
Windows system
MSG #1
MSG #2
MSG #3
MSG #n
MSG #n
Windows Procedure

메시지 핸들러

- **윈도우 프로시저 함수**는 메시지 큐에서 메시지를 꺼내 1차 처리하는 함수
  - 수백 종의 메시지 중 필요한 것만 식별(switch-case) 후 처리하며 등록하지 않은 경우 시스템 기본처리규칙 적용
- **메시지 마다 개별적인 처리 함수를 구현**하는 것이 보통이며 이를 **메시지 핸들러(처리기)**라 지칭
- 사용자 메뉴나 버튼 클릭 시 발생하는 메시지는 구체적인 메뉴나 버튼 식별을 위해 추가적으로 **ID 값**을 전달

Win32 API 예제 분석

```
// 전역 변수:
HINSTANCE hInst;                                // 현재 인스턴스입니다.
WCHAR szTitle[MAX_LOADSTRING];                  // 제목 표시줄 텍스트입니다.
WCHAR szWindowClass[MAX_LOADSTRING];            // 기본 창 클래스 이름입니다.

// 이 코드 모듈에 포함된 함수의 선언을 전달합니다:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                      _In_opt_ HINSTANCE hPrevInstance,
                      _In_ LPWSTR    lpCmdLine,
                      _In_ int       nCmdShow)
{
```

Win32 API 예제 분석

```
UNREFERENCED_PARAMETER(hPrevInstance);
UNREFERENCED_PARAMETER(lpCmdLine);

// TODO: 여기에 코드를 입력합니다.

// 전역 문자열을 초기화합니다.
LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
LoadStringW(hInstance, IDC_HELLOWINDOWS, szWindowClass, MAX_LOADSTRING);
MyRegisterClass(hInstance);

// 애플리케이션 초기화를 수행합니다:
if (!InitInstance (hInstance, nCmdShow))
{
    return FALSE;
}
```

Win32 API 예제 분석

```
HACCEL hAccelTable = LoadAccelerators(hInstance,
    MAKEINTRESOURCE(IDC_HELLOWINDOWS));

MSG msg;

// 기본 메시지 루프입니다:
while (GetMessage(&msg, nullptr, 0, 0))
{
    if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}
```

Win32 API 예제 분석 – MyRegisterClass()

```
WNDCLASSEXW 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_HELLOWINDOWS));
wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_HELLOWINDOWS);
wcex.lpszClassName  = szWindowClass;
wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));
```

Win32 API 예제 분석

```
//  함수: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  용도: 주 창의 메시지를 처리합니다.
//
//  WM_COMMAND  - 애플리케이션 메뉴를 처리합니다.
//  WM_PAINT    - 주 창을 그립니다.
//  WM_DESTROY  - 종료 메시지를 게시하고 반환합니다.
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_COMMAND:
        {
```

Win32 API 예제 분석

```
case WM_PAINT:
    {
        PAINTSTRUCT ps;
        HDC hdc = BeginPaint(hWnd, &ps);
        // TODO: 여기에 hdc를 사용하는 그리기 코드를 추가합니다...
        EndPaint(hWnd, &ps);
    }
    break;
case WM_DESTROY:
    PostQuitMessage(0);
    break;
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
```

Win32 API 예제 분석 – InitInstance()

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   hInst = hInstance; // 인스턴스 핸들을 전역 변수에 저장합니다.

   HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);
```

MFC 소개 및 콜렉션 클래스

MFC가 제공하는 자료구조 클래스, 문자열 클래스에 대해 알아봅니다.

MFC(Microsoft Foundation Class)

- **Win32 API를 쉽게 활용 할 수 있는 지원체계**
  - Win32 API 직접 호출 가능
  - Win32 API와 혼용해 개발 가능
- **C/C++ 기반 (GUI) 응용 프로그램 프레임워크**
  - GUI개발에 최적화
  - Windows XP 수준(Pentium III CPU + 256MB RAM) 환경에서도 구동 가능
- **문자열 처리, 컬렉션, 시스템 입/출력 클래스 제공**
- **AFX(windows Application FrameworKS)**

개발에 앞선 준비

- Visual Studio 2022
  - 2024년 현재 아직까지 Visual C++ 6.0 컴파일러를 사용하는 환경이 있음
- Windows 11 OS
  - Window 9x/NT 계열 지원 가능
- ProcessExplorer & DebugView (선택)
- 마음가짐
  - main() 함수가 없으며 흐름이 보이지 않음
  - 처음 보는 자료형의 등장

MFC collection classes

- The template-based classes
  - CArray
  - CList
  - CMap
- The collection classes not based on templates

Arrays

- CObArray
- CByteArray, CDWordArray, CWordArray, CUintArray
- CPtrArray
- CStringArray

Lists

- CObList
- CPtrList
- CStringList

Maps

- CMapPtrToWord, CMapPtrToPtr
- CMapStringToOb, CMapStringToPtr
- CMapWordToOb
- CMapWordToPtr

MFC collection classes 헤더 파일

```
#define _AFXDLL
#include <afx.h>
#include <Afxtempl.h>

int main()
{
    ...
}
```

CPtrList 활용 예

```
CPtrList list;
    list.AddTail((void*)"test");
    list.AddTail((void*)"string");
    list.AddTail((void*)"data");

    POSITION pos = list.GetHeadPosition();
    while (pos != NULL) {
        std::cout << (char*)list.GetAt(pos) << std::endl;
        list.GetNext(pos);
    }
```

CMapStringToPtr 활용 예

```
CMapStringToPtr map;
    map.SetAt(_T("test1"), (void*)"test");
    map.SetAt(_T("test2"), (void*)“string");
    map.SetAt(_T("test3"), (void*)“data");

    void* pResult = nullptr;
    map.Lookup(_T("test2"), pResult);
    if (pResult != nullptr)
        std::cout << (char*)pResult << std::endl;
```

MBCS와 유니코드

- MBCS(Multi Byte Character Set)
  - char형
  - 영문 한 글자는 1바이트, 한글 한 글자는 2바이트
  - 버전이 낮은 VS 환경의 기본 설정
- Unicode
  - wchar_t형
  - 영문, 한글 모두 한 글자에 2바이트 사용
  - 문자열 앞에 'L'을 붙여서 구분
  - TEXT(), _T() 매크로 사용

CString

- MFC 기반 응용 프로그램에서 문자열 처리를 지원하는 클래스로 MBCS, Unicode에 대한 지원과 추상성 제공
- MFC 응용 프로그램에서는 std::string보다는 가급적 CString 사용을 권장
- 문자열 사용에 대한 매우 높은 호환성 제공
- 내부적으로 효율을 높이기 위한 메모리 운영기능을 제공 하지만 간혹 메모리 사용량 증가의 원인으로 작용
- 함수 매개변수로 사용 시 주의 (빌드 모드 이슈)

CString 핵심 멤버함수

- =, !=, !=, +=, 〈, 〉, 〉=, 〈= 연산자
- CompateNoCase(), MakeUpper/Lower()
- const TCHAR* 형변환
- Trim(), TrimLeft/Right()
- Find()
- Format()
- LoadString()

CFile

- 파일 입/출력에 대한 추상성 제공
- MFC 직렬화(Serialization) 지원
  - 자동 압축
  - 네트워크 직렬화 지원
- 응용 프로그램 내부에서 C언어 표준 입/출력 함수나 Win32 API를 직접 사용하는 경우 불필요

윈도우 기초 이론

윈도우 그 자체에 대해 알고 있어야 할 필수 이론과 MFC 개발 환경에 대해 다룹니다.

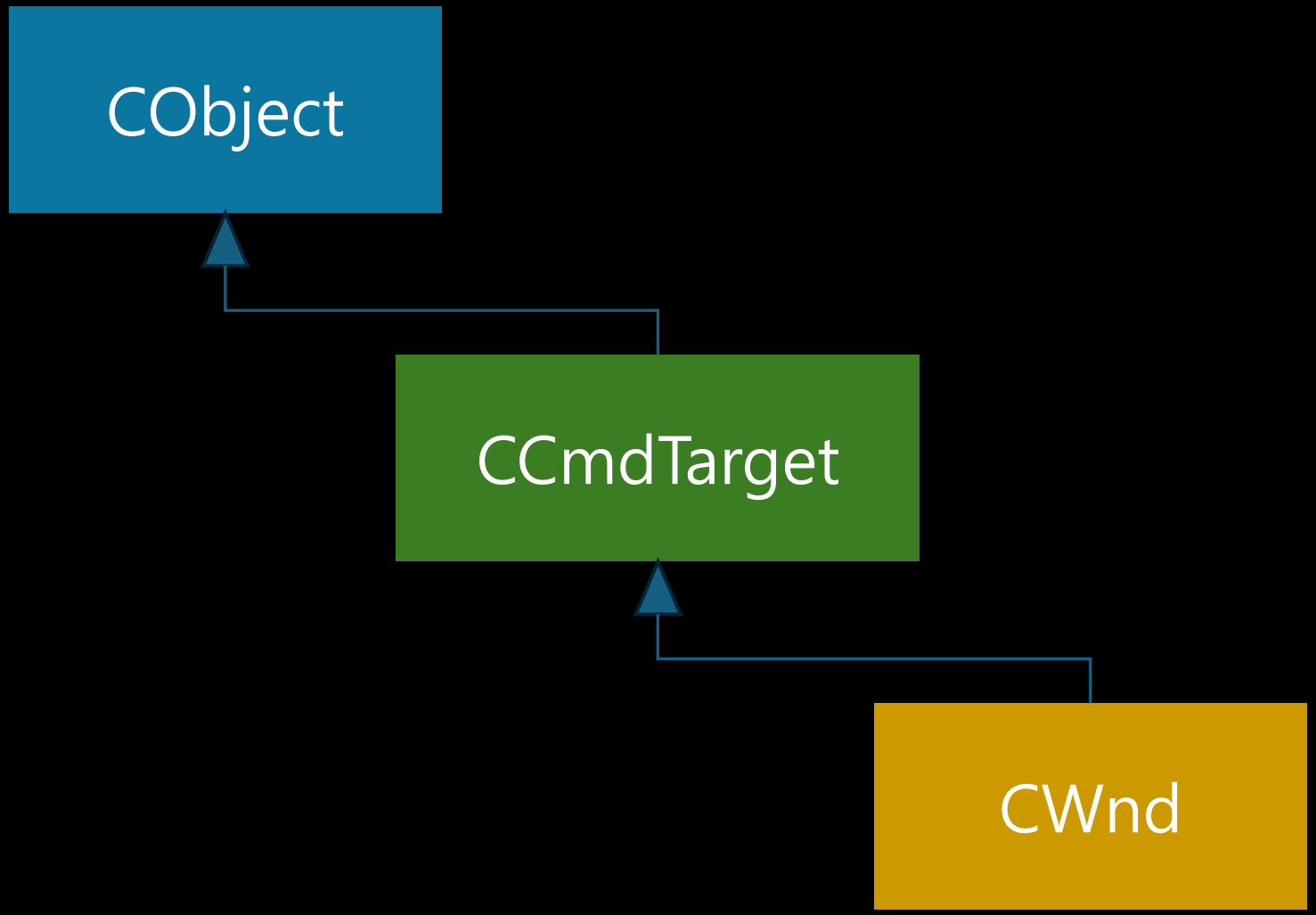
솔루션과 프로젝트

- **솔루션은 최상위 개념**
  - 한 솔루션은 최소 1개 이상의 프로젝트로 구성
- **프로젝트는 솔루션에 속하는 단위**
  - 모듈(.exe, .dll)단위로 한 프로젝트 존재
  - 각 프로젝트는 상호 의존성이 존재 할 수 있음
- **프로젝트에 속하는 요소**
  - 소스코드(.c, .cpp, .h)
  - 리소스(.rc, .ico, .bmp 등)

최초 예제 생성

1. 새 프로젝트 만들기
2. C++, Windows, 데스크톱 템플릿 선택
3. Windows 데스크톱 애플리케이션 선택
4. 프로젝트명: HelloWindows
5. 빌드(F7)
6. 실행(Ctrl + F5)
7. VS가 생성한 각종 파일과 폴더 내용 확인

CWnd 클래스 상속구조



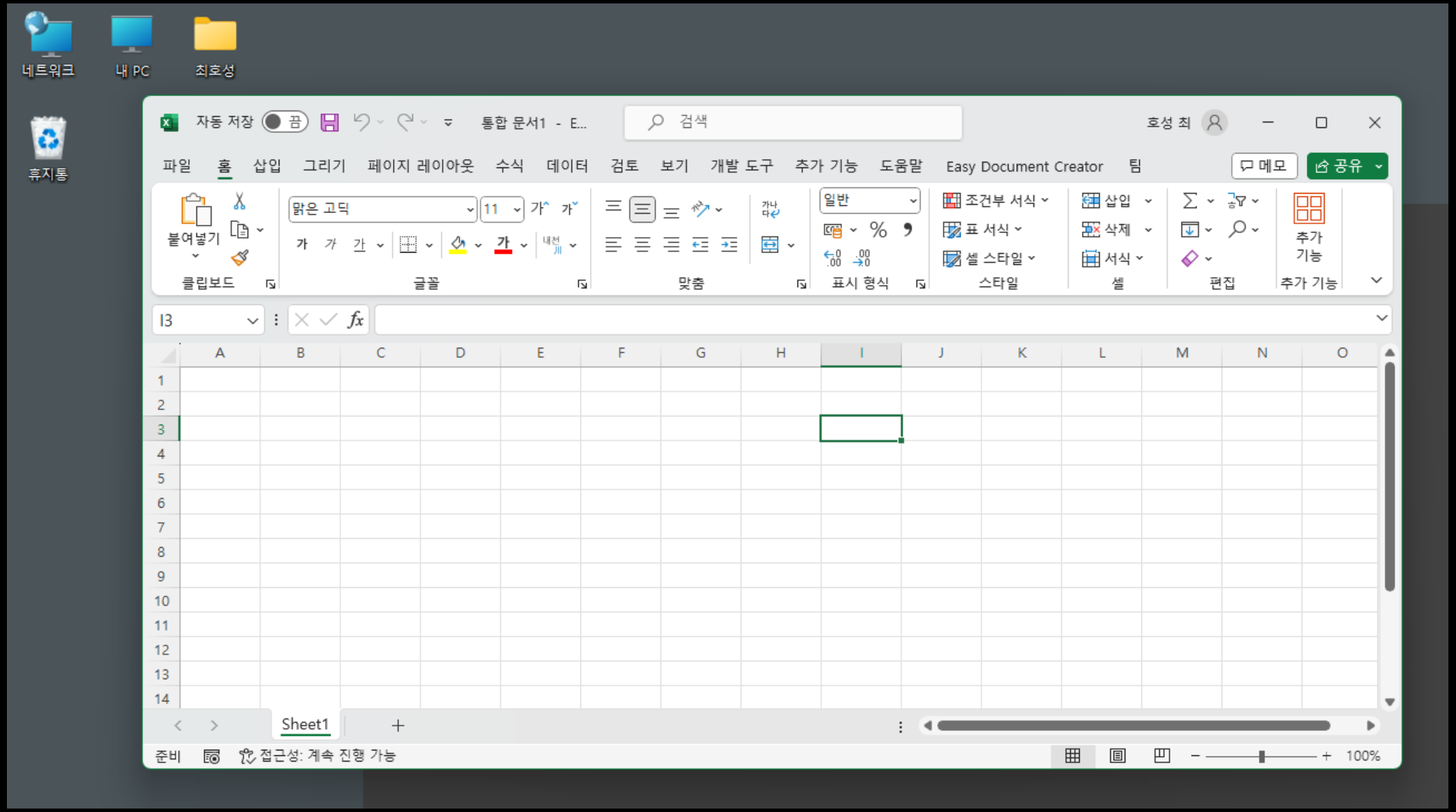
CWnd 클래스와 윈도우 속성

- 식별자(핸들, ID)
- 좌상단 좌표(CPoint)와 폭과 높이(CSize)
- 윈도우 스타일 (상수정의)
  - 프레임, 차일드
- 윈도우 상태
  - 보이기 여부
- 윈도우 프로시저
  - 최상위 프레임 윈도우마다 개별 스레드와 메시지큐 존재

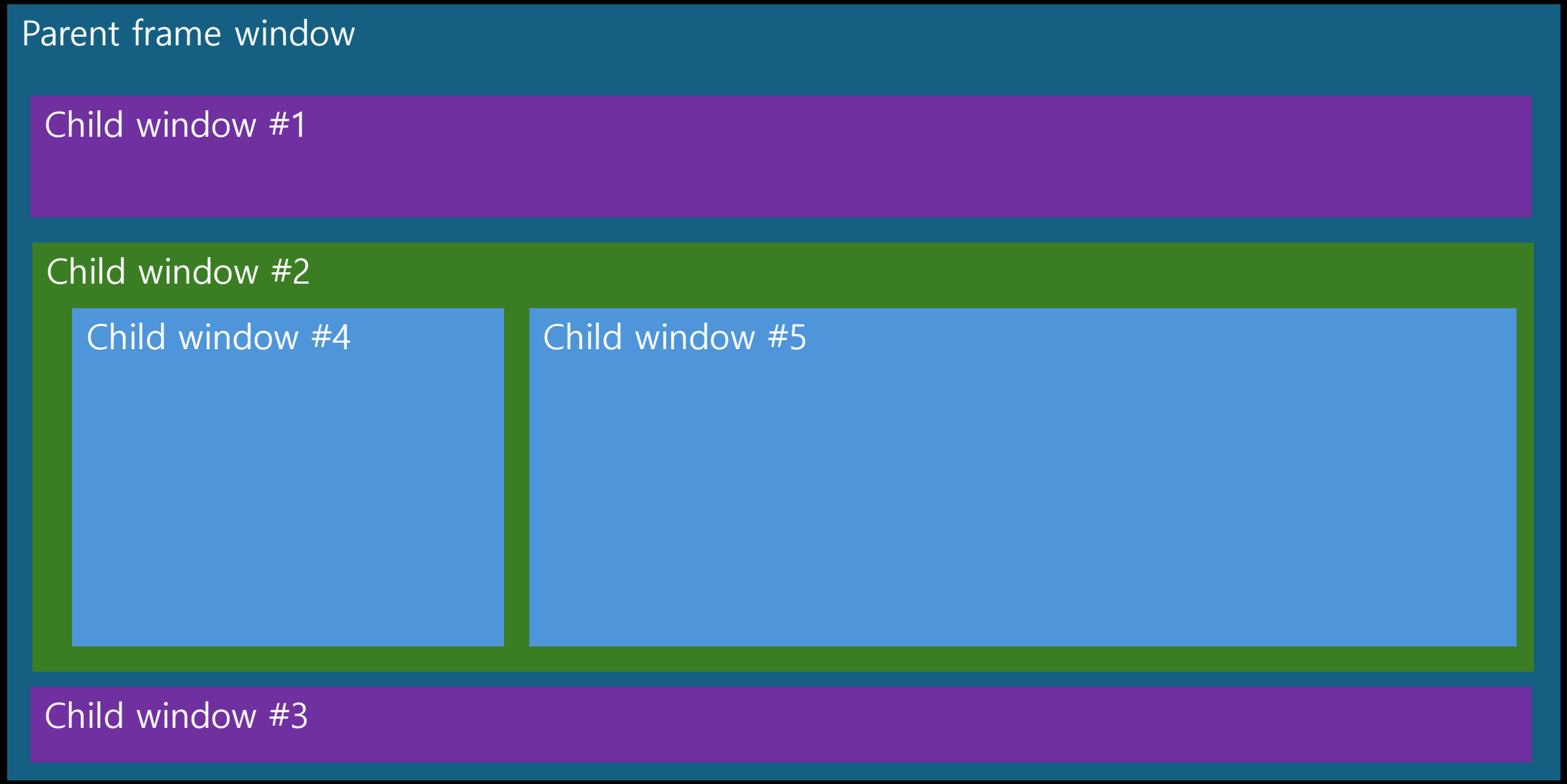
윈도우 관계

- **부모, 자식, 형제 관계**
  - 최상위 부모 윈도우는 바탕 화면(Desktop)
  - 모든 프레임 윈도우의 부모는 바탕 화면
  - 바탕 화면의 핸들은 null
- **자식 윈도우는 부모 윈도우의 상태를 공유**
  - 보임 여부 및 위치
- **자식 윈도우의 위치는 부모 윈도우를 기준으로 계산**
  - 부모 윈도우 이동 시 자식 윈도우도 함께 이동

윈도우 관계


네트워크
내 PC
최호성
휴지통
자동 저장
통합 문서1 - E...
검색
호성 최
파일 홈 삽입 그리기 페이지 레이아웃 수식 데이터 검토 보기 개발 도구 추가 기능 도움말 Easy Document Creator 팀
메모
공유
붙여넣기
맑은 고딕
11
일반
조건부 서식
표 서식
셀 스타일
삽입
삭제
서식
추가 기능
클립보드
글꼴
맞춤
표시 형식
스타일
셀
편집
추가 기능
I3
Sheet1
준비
접근성: 계속 진행 가능
100%

윈도우 관계


Parent frame window
Child window #1
Child window #2
Child window #4
Child window #5
Child window #3

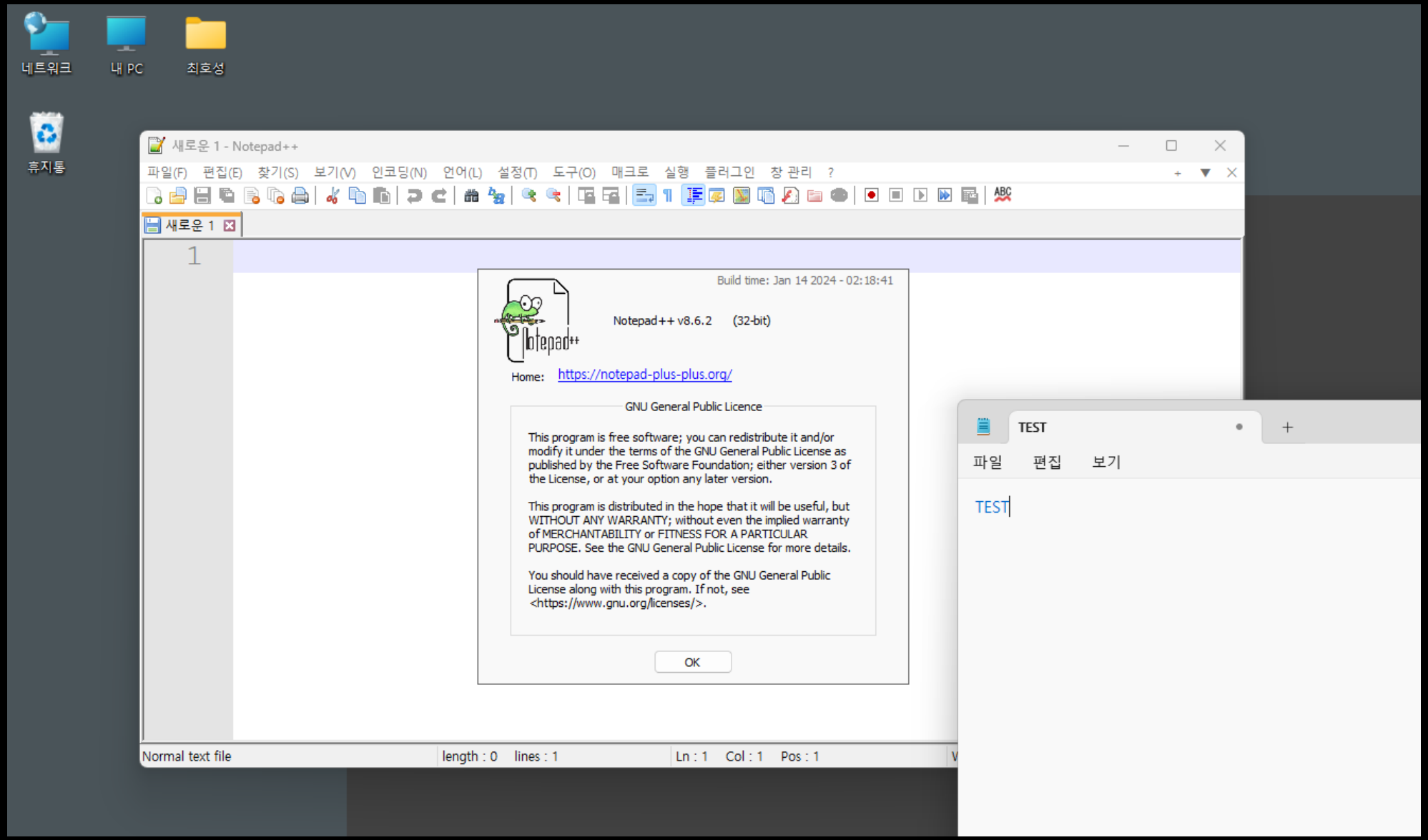
기본 화면 좌표계

- 바탕 화면(모니터)으로 표현되는 2차원 좌표계
- 바탕 화면 좌측 상단을 기준(x:0, y:0)으로 설정
  - 가로 X축은 왼쪽에서 오른쪽으로 증가
  - 세로 Y축은 위에서 아래쪽으로 증가
  - 좌표 단위는 기본적으로 화소 수
  - 모니터 해상도에 따라 바탕화면 크기가 결정

Z-order

- 화면에서 겹쳐 보이는 윈도우들 간의 순서
  - 최상위(대표적으로 작업 관리자 윈도우, TOPMOST)
  - 응용 프로그램 일반 (Non-topmost)
  - 프로그램 코드로 조절 가능
- 윈도우를 마우스 클릭 시 OS가 Z-order를 변경해 화면에 모두 표시되도록 조정하고 입력 포커스 부여
- 기존 입력 포커스를 가지고 있었던 윈도우는 그 다음 순서로 내려가고 화면에서 가려짐

기본 화면 좌표계와 Z-order

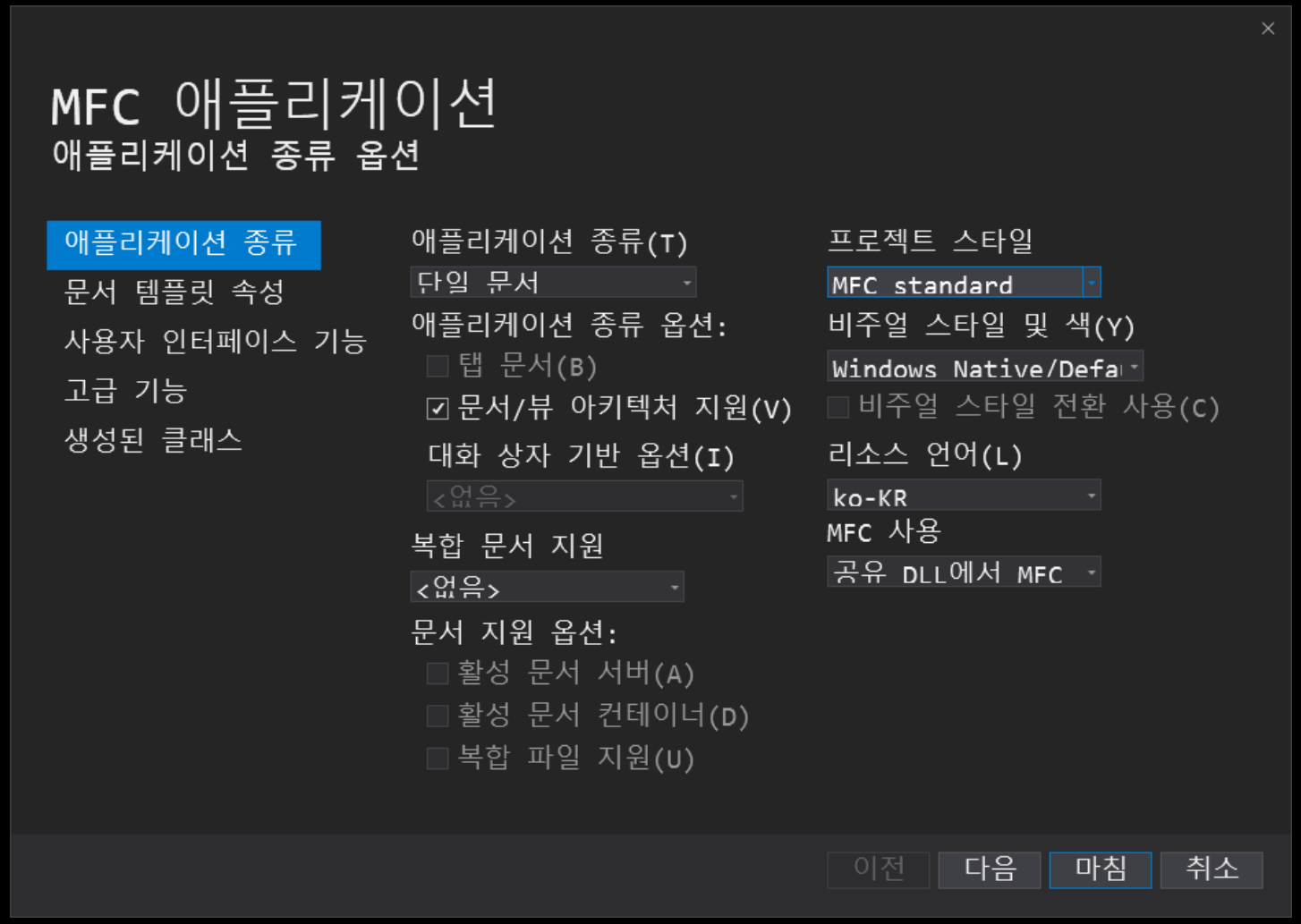


SDI 응용 프로그램

MFC 프레임워크가 적용된 가장 보편적인 구조의 응용 프로그램을 분석합니다.

SDI 예제 생성

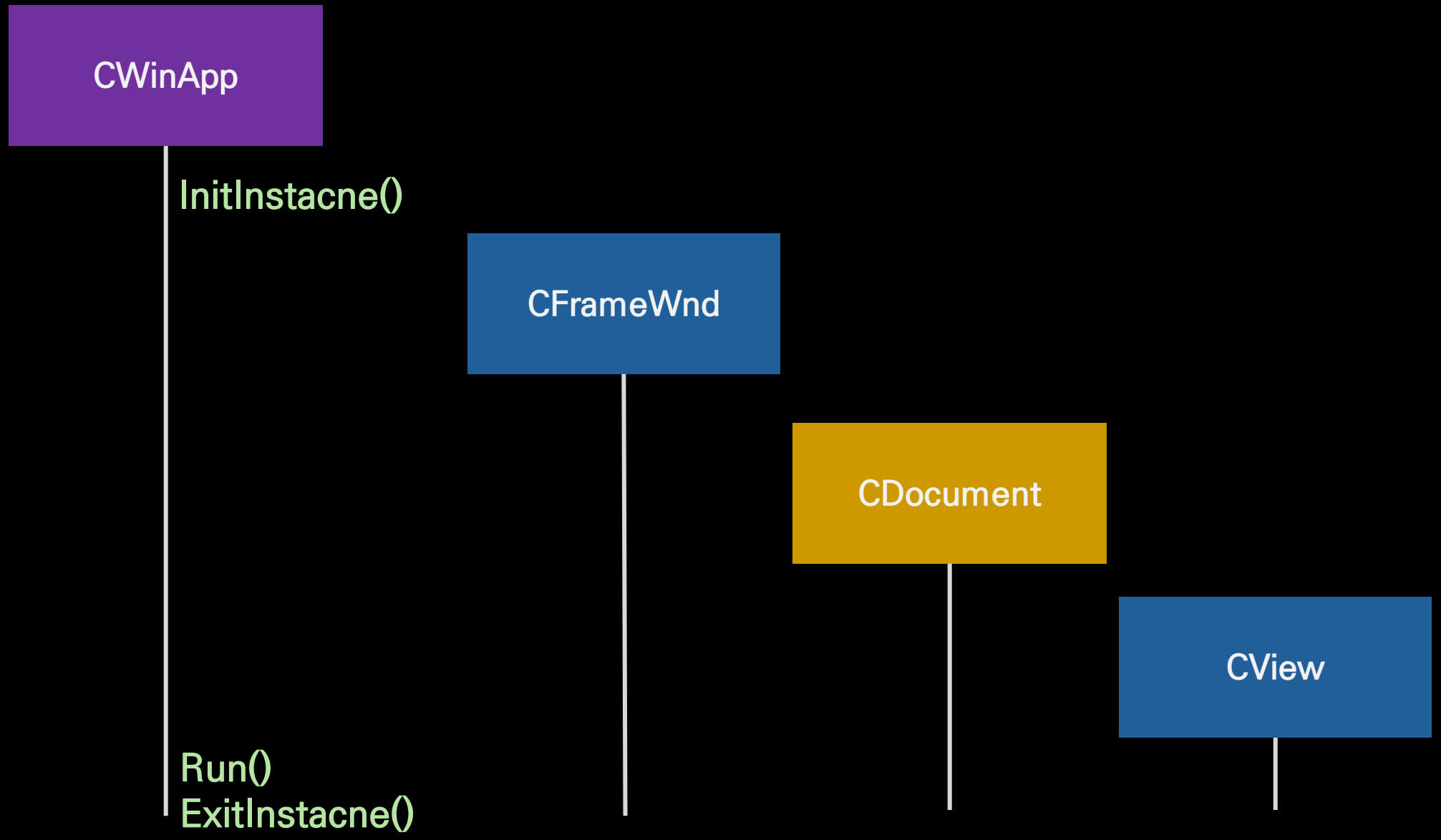
MFC앱 선택



기본 구조 및 객체 생성 순서

1. AfxWinMain()
2. CWinApp (Process, Thread)
   - Init/ExitInstance(), Run()
   - theApp (전역변수)
3. CDocument (Data)
   - 데이터(혹은 파일)
4. CFrameWnd (GUI – Frame)
5. CView (GUI – View)

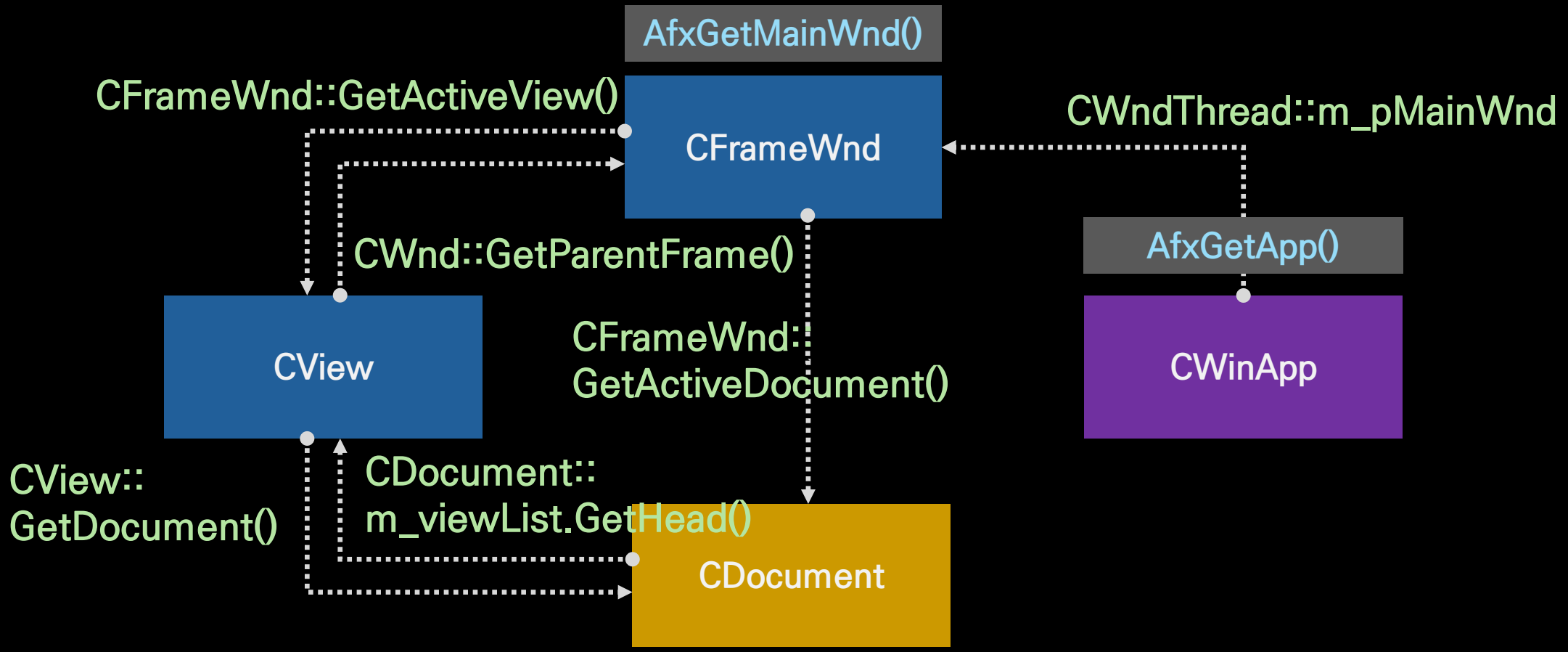
SDI 주요 클래스 인스턴스 생성



각 클래스 접근 방법

- CWinApp
  - theApp, AfxGetApp()
- CDocument
  - CFrameWnd::GetActiveDocument()
- CFrameWnd
  - AfxGetMainWnd()
- CView
  - CFrameWnd::GetActiveView();

SDI 주요 객체간 접근 방법



프레임워크 흐름 분석 – 초기화

- 01: CWinApp::CWinApp()
- 02: CWinApp::InitInstance()
- 03: CDocument::CDocument()
- 04: CFrameWnd:: CFrameWnd()
- 05: CFrameWnd::LoadFrame()
- 06: CFrameWnd::PreCreateWindow()
- 07: CFrameWnd::PreCreateWindow()
- 08: CFrameWnd::OnCreate()
- 09: CFrameWnd::OnCreateClient()
- 10: CView::CView()
- 11: CView::Create()
- 12: CView::PreCreateWindow()
- 13: CView::OnCreate()
- 14: CView::OnShowWindow()
- 15: CFrameWnd::OnCreateClient() - Return
- 16: CFrameWnd::OnCreate() - Return

프레임워크 흐름 분석 – 초기화 (이벤트 루프)

- 17: CDocument::OnNewDocument()
- 18: CView::OnInitialUpdate()
- 19: CMainFrame::OnActivateApp()
- 20: CMainFrame::OnActivate()
- 21: CMainFrame::OnShowWindow()
- 22: CView::GetDocument()
- 23: CWinApp::Run()

프레임워크 흐름 분석 – 종료

```
24:          CMainFrame::OnClose()
25:                  Function: CMainFrame::OnShowWindow()
26:                  Function: CMainFrame::OnActivate()
27:                  Function: CMainFrame::OnActivateApp()
28:                  Function: CMainFrame::DestroyWindow()
29:                  Function: CMainFrame::OnDestroy()
30:                  Function: CView::OnDestroy()
31:                  Function: CView::PostNcDestroy()
32:                  Function: CView::~CView()
33:                  Function: CMainFrame::OnNcDestroy()
34:                          Function: CMainFrame::PostNcDestroy()
35:                          Function: CMainFrame::~CMainFrame()
36:                  CMainFrame::OnNcDestroy() - Return
37:                  Function: CDocument::~CDocument()
38:          CMainFrame::OnClose() – Return
39:          CWinApp::ExitInstance()
40:   CWinApp::Run() - Return
```

메시지 맵과 핸들러

- 메시지 맵(Message map)Win32 API 방식의 윈도우 프로시저 구조를 개선해 만든 것
- 기존 switch-case 구조를 한 메시지(혹은 명령)에 대해 한 함수(핸들러)를 1:1로 매핑
- Visual Studio에서 자동으로 메시지 맵 코드를 관리 (코드 추가 및 제거(주석처리방식))

메뉴

사용자 정의 메뉴를 다루는 방법을 배웁니다.

메뉴 리소스 편집

- Visual Studio가 제공하는 가장 비주얼한 기능
- ‘리소스 뷰’가 보이도록 VS UI 설정변경
- 리소스 목록은 .rc 파일을 시각화 한 것
  - rc파일은 고유 문법을 갖는 스크립트 파일
- 리소스 데이터는 프로젝트 파일 아래 res 폴더에 위치
  - .ico, .bmp

핸들러 등록

- 여러 방법이 있으며 어떤 것을 사용하더라도 무방
- 핸들러를 등록 할 클래스 선택 후 속성에서 추가
  - 메뉴, 메시지 핸들러, 각종 가상 함수
- 수작업 가능
- 메뉴 리소스 편집 및 핸들러 등록 실습
  - 기본적으로 메뉴는 View에 등록
  - 다중 View 환경에서 핸들러가 없는 메뉴는 자동으로 Disable

등록된 핸들러 삭제

- 실수로 잘못 등록한 핸들러를 삭제 할 때는 가급적 수작업으로 삭제를 완료
  - 함수선언 삭제 (.h)
  - 메시지 맵 수정 (.cpp)
  - 함수정의 삭제 (.cpp)
- 핸들러를 그대로 두고 리소스만 삭제 하는 것도 가능하지만 바람직하지 않음

다른 메뉴 실행하기

- WM_COMMAND 메시지를 코드로 전달하는 방법으로 구현
  - 메뉴의 실체는 WM_COMMAND 메시지의 매개변수 값
  - 모든 것이 메시지라는 사실을 재확인
- PostMessage() 활용
  - SendMessage() 함수는 메시지 큐를 거치지 않고 핸들러 함수를 직접 호출
  - SendMessage() 함수는 가급적 사용하지 않는 것을 권장

바로 가기 키

- 특정 메뉴 하나에 대해 단축키를 1:1로 적용
- 리소스를 편집하는 방식으로 적용
  - 코드로도 가능하지만 잘 사용하지 않음
- 결과적으로 특정 메뉴가 선택된 것
  - 키 입력 메시지에 대해 윈도우 시스템이 전처리를 실시하고 조합된 결과를 메뉴 선택으로 인식

토글 메뉴

- **메뉴 항목을 Boolean 형식으로 활용하는 방법**
  - 확인 표시 여부를 업데이트
  - true/false 값을 멤버에 저장하는 방식으로 구현
  - 응용 프로그램 수준 메뉴는 응용 프로그램 전체에 사용된다는 점을 반드시 고려
- **사용자가 메뉴 선택 시 발생하는 Update 메시지를 처리하는 방식으로 구현**
  - 메뉴 중 어떤 것 하나라도 선택되면 Update 메시지 발생
  - 등록된 모든 업데이트 핸들러가 호출

팝업 메뉴

- 마우스 오른쪽 버튼 클릭 시 화면에 즉시 표시되는 메뉴
  - 특정 환경(혹은 상황)에 특화된 메뉴
  - Context 메뉴
- WM_RBUTTONDOWN 메시지를 처리하는 방식을 권장

툴바와 상태바

메뉴를 보다 편리하게 사용 할 수 있도록 제공되는 부가 요소에 대해 알아 봅니다.

툴바 리소스 편집

- **기본 16컬러 이미지를 툴바 버튼 이미지로 제공**
  - 최대 256 컬러 비트맵까지 편집가능
  - 트루컬러 이미지를 사용하고 싶다면 확장 형식 활용
- **특정 메뉴를 선택(실행)하기 쉽도록 제공**
  - 특정 메뉴에 대한 툴바 버튼(1:1) 제공
- **프레임 윈도우에 의존적**
  - 프레임 윈도우 안쪽(상하좌우)에 도킹
  - 별도 윈도우로 분리가능

툴바 도킹 위치 제어

- 도킹 윈도우는 도킹 대상 프레임 윈도우와 관련 속성이 일치해야 적절히 작동
  - 프레임 윈도우에서 도킹 허용 위치(상하좌우면) 설정
  - 도킹 윈도우에서 도킹 할 위치 설정
- BCGSoft사 라이브러리가 MFC에 통합된 시점부터는 구형 툴바 활용도가 줄어듦
  - 확장 툴바는 트루컬러 버튼 이미지를 기본 제공
  - 단, 리소스 편집기에서 직접 편집 불가

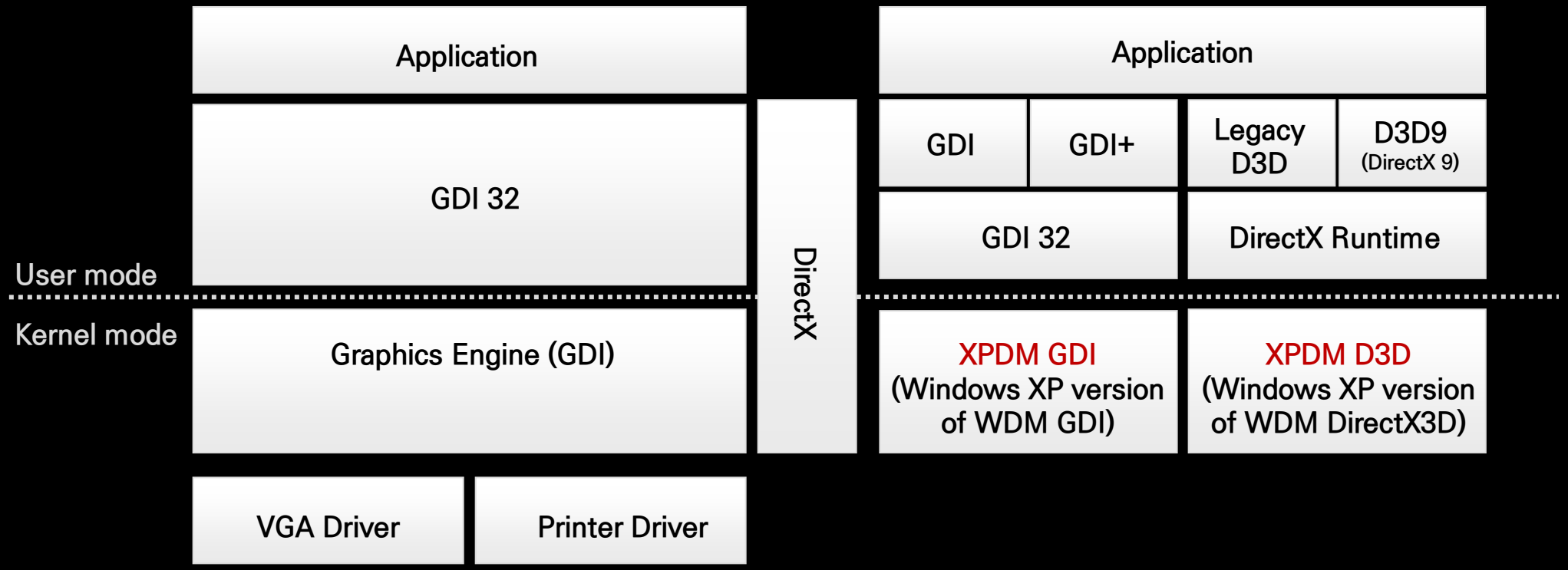
상태 표시줄 연결

- 상태 표시줄은 메뉴에 대한 설명이나 토글 키 정보를 출력하는 기능이 기본 제공
  - 보통 메뉴에 대한 설명이나 진행 상황을 출력
  - 추가 기능을 원할 경우 서브 클래싱을 통해 확장
- 리소스 편집으로 기본적인 기능 활용가능

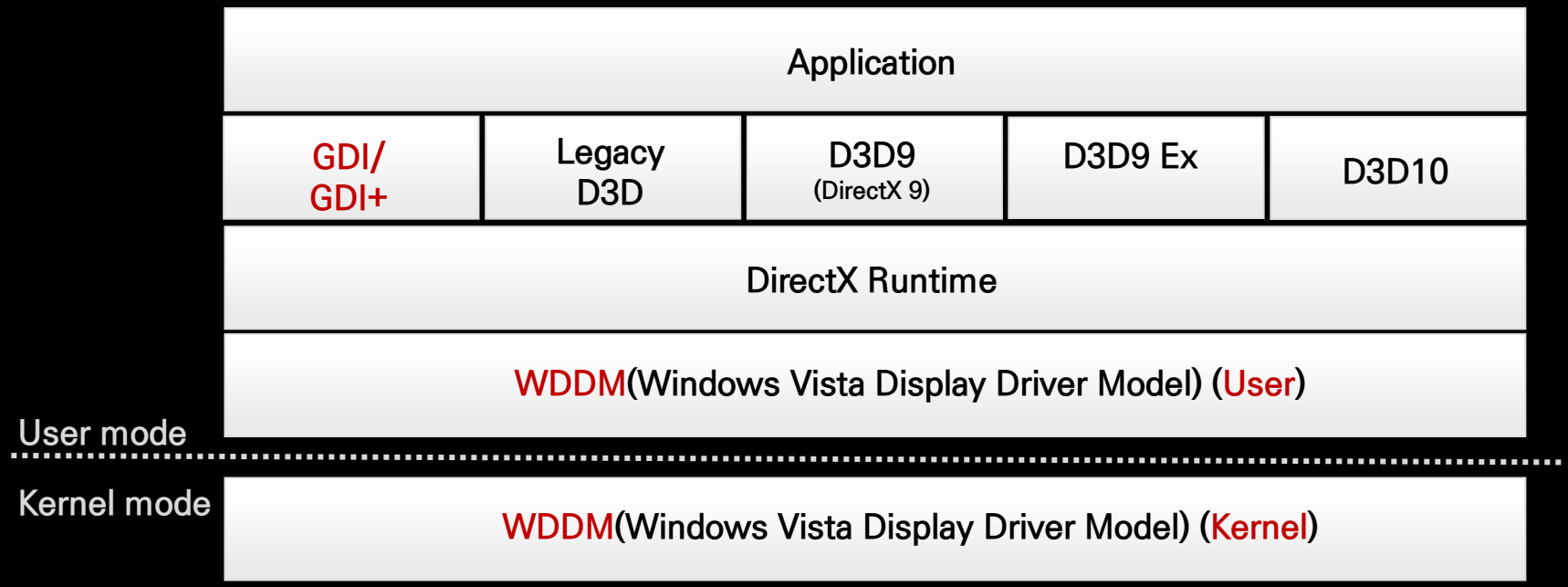
GDI 기본

구형 그래픽 처리 방법인 GDI 프로그래밍 기법 중 기초적인 내용을 학습니다.

윈도우 그래픽 시스템 구조 (2000, XP)



윈도우 그래픽 시스템 구조 (Vista)



렌더링 시점

- 윈도우 화면은 기본적으로 2회 렌더링
  - WM_PAINT
  - WM_ERASEBKGND
- WM_PAINT 메시지 처리시에는 반드시 CPaintDC를 사용 (렌더링 완료 통지)

DC에 대한 이해

- **윈도우에 대한 DC를 생성한 후 렌더링 인터페이스로 활용**
  - 생성한 DC(GetDC())는 수동으로 삭제(ReleaseDC())
- **렌더링 도구 객체를 DC에 적용(SelectObject())**
  - CPen, CBrush, CFont 등 다수
  - 시스템이 제공하는 객체 존재 (Stock object)
- **구형 GDI와 신형 GDI+**
- **DC 종류**
  - CDC, CClientDC, CPaintDC, CMetaFileDC

DC에 대한 이해

- **DC가 비트맵을 갖지 않으면 출력 불가**
  - 특정 윈도우에 대한 DC는 비트맵을 가짐
  - 비트맵은 대상 윈도우 화면과 동일
- **여러 DC를 생성해 이미지를 합성하는 것이 가능**
  - 모니터 화면이 아닌 비트맵에 그리기 가능
  - 여러 비트맵에 그린 후 한 화면에 합성
  - 화면에 출력은 비트맵 출력보다 느림

DC에 대한 이해

- GDI의 한계
  - 24비트 컬러까지 사용가능
  - Win32 API를 적절히 활용하더라도 Anti-aliasing 처리는 불가능
- GDI+의 장점
  - 다수의 고급 그리기 기능 대거 제공
  - 구형 매핑 모드를 대체 할 수 있는 전환기능 제공
  - Anti-aliasing 및 32비트 트루컬러 제공(Alpha channel)
  - 주요 코드가 C#과 호환

User mode와 Kernel mode

User mode

Kernel mode

S/W

H/W

RGB 컬러

- Red, Green, Blue 등 빛의 3원색을 각각 부호가 없는 8비트 정수(0 ~ 255)로 기술
- RGB() 매크로를 이용해 색상값 정의

펜과 선

- **펜(CPen)객체를 DC에 적용한 후 그리기 시작**
  - 별도 펜을 지정하지 않을 경우 기본 펜(1픽셀, 검정색)적용
  - 외곽선을 그리지 않으려면 NULL펜 설정
- **선은 펜 속성에 따라 모양 결정**
  - 색상 및 굵기
  - 실선(Solid)
  - 점선
  - 사용자 정의 패턴

도형과 브러시

- 기본 도형은 네모와 원 두 가지
  - 결과적으로 2차원 화면상의 모든 도형은 네모
  - 원은 네모 속에 존재
  - CRect, CPoint, CSize
- 도형 외각은 DC가 선택한 Pen으로 그려지며 그 내부는 브러시로 채워짐
  - CPen, CBrush

글꼴과 문자열 출력

- 문자열은 그려지는 것으로 이해 할 수 있음
- MBCS와 Unicode 환경을 반드시 고려
- 응용 프로그램 수준에서 글꼴을 일치시키려면 레더링 전에 미리 전역 객체로 생성해두는 것이 바람직
- 핵심 함수
  - TextOut()
  - TabbedTextOut()
  - DrawText()

비트맵 다루기

GDI 환경에서 비트맵을 포함한 이미지를 다루는 방법에 대해 배웁니다.

비트맵 리소스 다루기

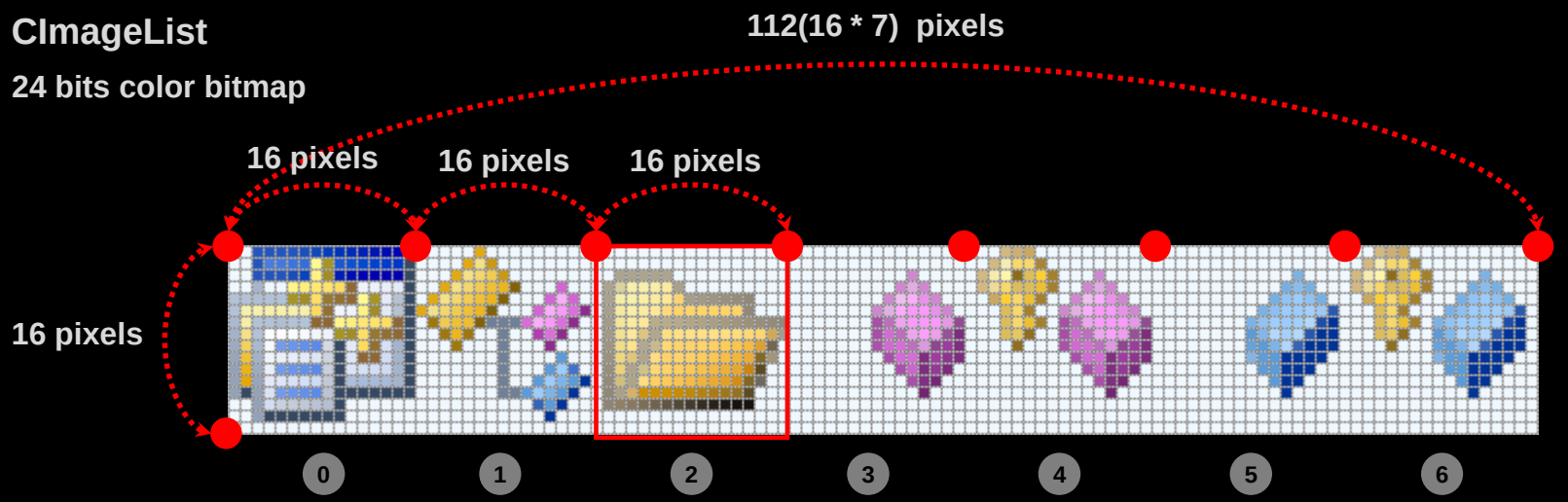
- **응용 프로그램 리소스로 비트맵을 포함 시킨 후 활용**
  - 24비트 비트맵(DIB)을 활용하는 가장 일반적인 방법
  - 실행파일에 비트맵이 포함되어 빌드
- **리소스에서 비트맵을 로드 한 후 출력하는 방식**
  - 리소스 비트맵을 메모리 DC로 로드
  - 메모리 DC를 화면 DC로 출력
  - 메모리 DC와 화면 DC를 합성해 출력하는 것도 가능 (합성모드 존재)

이미지 출력 API

- BitBlt()
  - 대상 DC로 원본 DC에 세트된 비트맵을 비트 블록 단위로 전송하는 함수
- StratchBlt()
  - BitBlt() 함수와 유사하지만 이미지를 확대/축소 할 수 있음
- TransparentBlt()
  - 특정 색상을 투명처리해 출력
- AlphaBlend()
  - 대상 이미지와 원본을 반투명하게 합성

CImage와 CImageList

- CImage 클래스를 이용해 리소스가 아닌 외부 파일 이미지를 쉽게 다룰 수 있음
- CImageList 클래스를 이용해 아이콘 이미지 다수를 가져와 배열처럼 묶어 사용가능



GDI 고급

그리기 모드나 매핑 모드 등 그리기 이상의 영상처리 기법을 배웁니다.

그리기 모드

- 결과적으로 두 영상을 합성하는 방법
  - BitBlt() 함수 참고
- 다양한 속성 존재

매핑 모드

- 쉽게 생각해 화면의 폭과 높이를 변경해 출력되도록 해주는 것
  - 화면 확대 축소 구현 가능
  - 축소 시 렌더링 결과가 소위 깨져 보일 수 있음
- 매핑 모드에 따라 화면 좌표계가 달라짐
- 기준 좌표의 위치를 조정해 변경할 수 있음
  - 마이너스 위치에 렌더링 가능
  - 스크롤 윈도우와 조합해 활용

윈도우 영역

- DC의 기본 설정은 윈도우 크기와 영역 크기가 일치
- 영역을 벗어난 그리기는 모두 무시
  - 이 특징을 이용해 고급 UI개발 가능
  - Syntax coloring editor 개발 시 유용
- 최상위 프레임 윈도우의 영역을 변경할 경우 윈도우 모양이 달라질 수 있음

깜빡임 방지

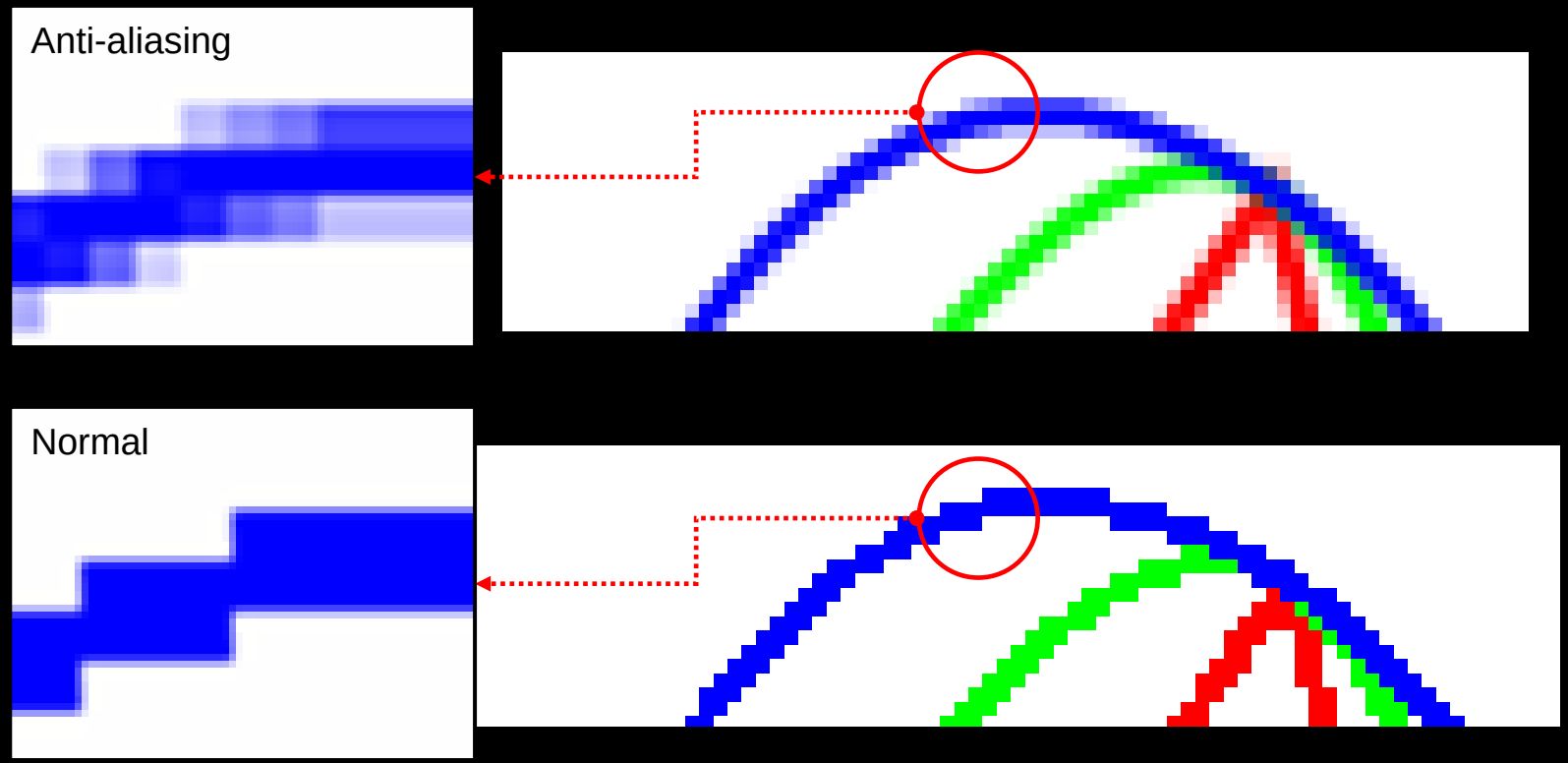
- WM_ERASEBKGND 메시지 발생 시 배경을 일단 덮어쓰며 이어지는 WM_PAINT 메시지 처리 시 새로 덧입혀지는 구조
  - WM_ERASEBKGND 메시지 발생 시 기본 렌더링을 하지 않도록 변경해 개선
- **더블 버퍼링** 기법을 적용해 손쉽게 해결 가능
  - CBufferDC를 개발해 기존 CPaintDC를 대체

GDI+ 소개

GPU를 이용할 수 있는 보다 개선된 렌더링 기법인 GDI+에 대해 학습합니다.

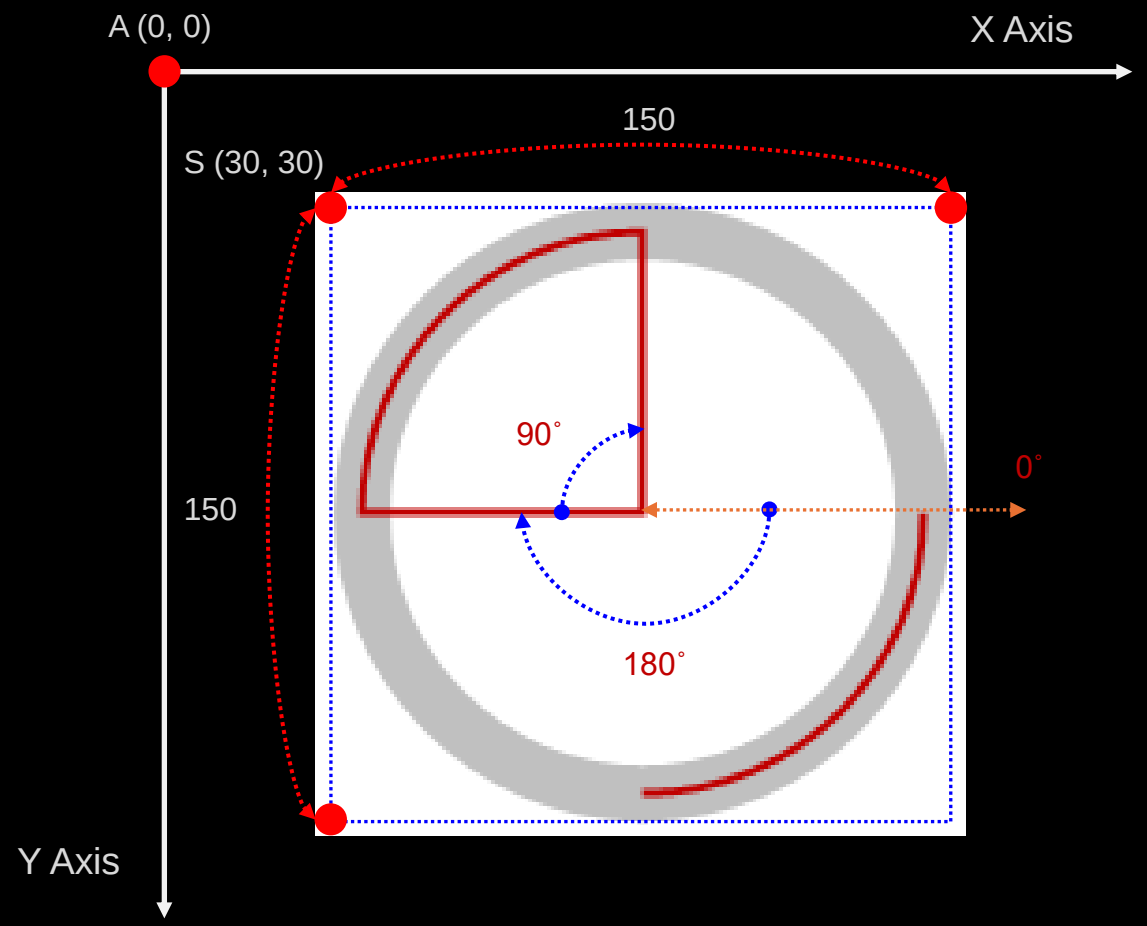
곡선 안티 에일리어싱

```
graphics.SetSmoothingMode(SmoothingModeHighQuality);
```

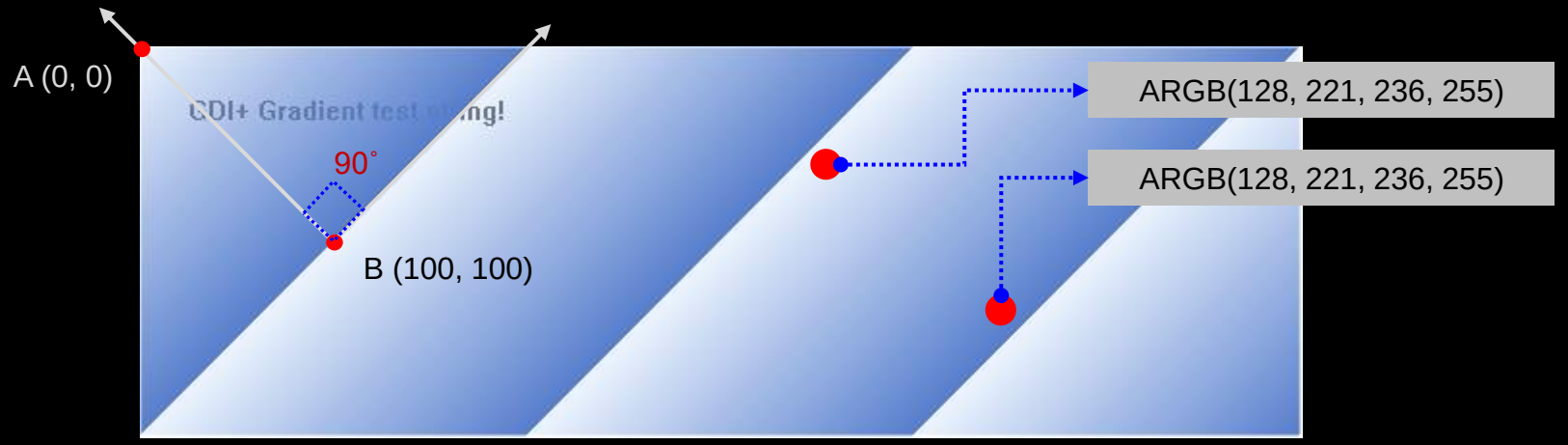


도형과 브러시

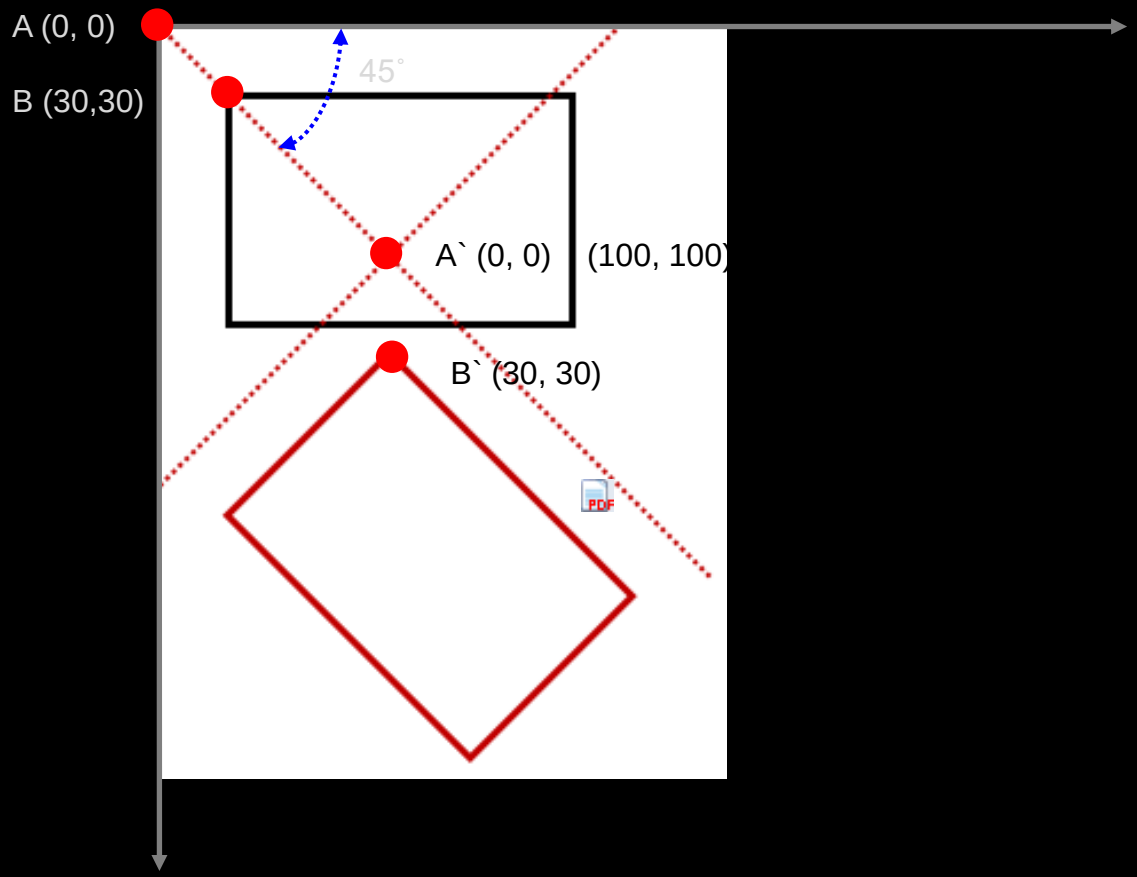
```
graphics.DrawArc(&RedPen, 30, 30, 150, 150, 0.0f, 90.0f);
graphics.DrawPie(&RedPen, 30, 30, 150, 150, 180.0f, 90.0f);
```



그라데이션과 텍스처



좌표계 변환



사용자 입력 이벤트 처리

키보드와 마우스 등 HID 입력을 처리하는 방법에 대해 배웁니다.

키보드 입력

- 키보드와 마우스(HID)는 각각 유일한 입력장치
  - 한 프로그램이 독점하지 말아야 함
- 모든 키보드 입력은 입력 포커스를 가진 응용 프로그램에 전달되는 것이 원칙
- 입력한 키 값은 Virtual key code로 변환해 전달 됨
- 키를 계속 누르고 있을 경우 지속적으로 메시지는 계속 발생
  - 눌린 회수가 매개변수로 함께 전달되며 1을 초과할 수 있음

마우스 이동

- 마우스 이동 시 WM_MOUSEMOVE 메시지가 발생하며 마우스 포인터 좌표값을 매개변수로 함께 전달
- 윈도우 영역을 벗어날 경우 메시지를 수신하지 못함
- 윈도우 속성에 따라 마우스 메시지를 수신하지 않을 수 있음
- 터치스크린이나 태블릿 역시 마우스와 동일하게 처리

버튼 클릭

- 윈도우 환경에서 마우스 버튼은 기본적으로 3개
  - 왼쪽, 중앙 휠, 오른쪽
  - 제어판 설정에 따라 좌우가 바뀔 수 있음(왼손)
- 모든 버튼은 Down, Up 메시지를 가짐
  - 클릭 이벤트에 대한 처리를 원한다면 Down이 아닌 Up 메시지 활용을 권장
- 더블 클릭은 마우스 버튼 Down + Up의 연속 발생 시 추가되는 메시지
- Drag and drop은 마우스 버튼 Down, Move, Up 세 메시지의 연속

마우스 이벤트 추적

- **윈도우 영역을 벗어나더라도 마우스 메시지를 수신하기 위한 방법**
  - 시스템에 하나 뿐인 마우스를 특정 응용 프로그램이 독점하는 결과
- **주의사항**
  - 좌표관련 이슈 발생(윈도우 기준, 스크린 기준)
  - 마우스 캡쳐 해제에 대해 반드시 고려해야 함

휠 버튼 처리하기

- **휠 버튼은 화면 스크롤을 목적으로 사용하는 것이 일반적**
  - 앞으로 밀면 스크롤 업
  - 손 안쪽으로 감아 당기면 스크롤 다운
- **매개변수로 스크롤 범위를 전달 받음**
  - 휠 조작에 따른 스크롤 폭은 윈도우 설정에 따름

컨트롤 윈도우와 대화상자

대화상자 기반 응용 프로그램과 컨트롤 윈도우를 조합한 GUI 개발 방법을 배웁니다.

힘들게 GDI를 배운 이유

- 윈도우 프로그래밍의 대부분은 컨트롤 윈도우를 사용해 GUI를 개발하는 것
- 이미 만들어진 컨트롤 윈도우를 활용하는 것이 일반적이나 필요하다면 커스텀 컨트롤을 새로 제작
- 기존 컨트롤 윈도우를 기반으로 확장하는 것도 가능
  - 서브 클래싱
  - 파생 클래스 제작 (예: CEditEx)

대화상자 리소스 편집

- 리소스 편집기로 파워포인트처럼 GUI화면을 구성
- 적용되는 글꼴 크기에 따라 대화상자, 컨트롤 윈도우 크기 및 배치가 달라질 수 있음
  - 다국어 지원 시 문제가 될 수 있음
  - 사용자가 임의로 글꼴 변경 시 UI 구성이 손상될 수 있음
  - 크기 및 배치 자동조정 기능 없음
- 모든 컨트롤 윈도우는 독립 팝업이 불가능한 차일드 윈도우
  - 윈도우 핸들 외에 ID로도 식별

대화상자 리소스 편집

- **탭 오더를 고려해 적절한 순서를 부여**
  - 탭 오더 입력 단축키는 Ctrl + D
  - 탭 오더는 설정 화면에서 마우스로 직접 클릭해 순서 설정
- **데이터 입력이 많은 화면 개발 시 적절한 탭 오더 반영은 필수**
  - 단, 입력 데이터가 범위를 벗어나거나 누락되는 오류 발생에 대한 편의 기능 제공은 별도로 구현

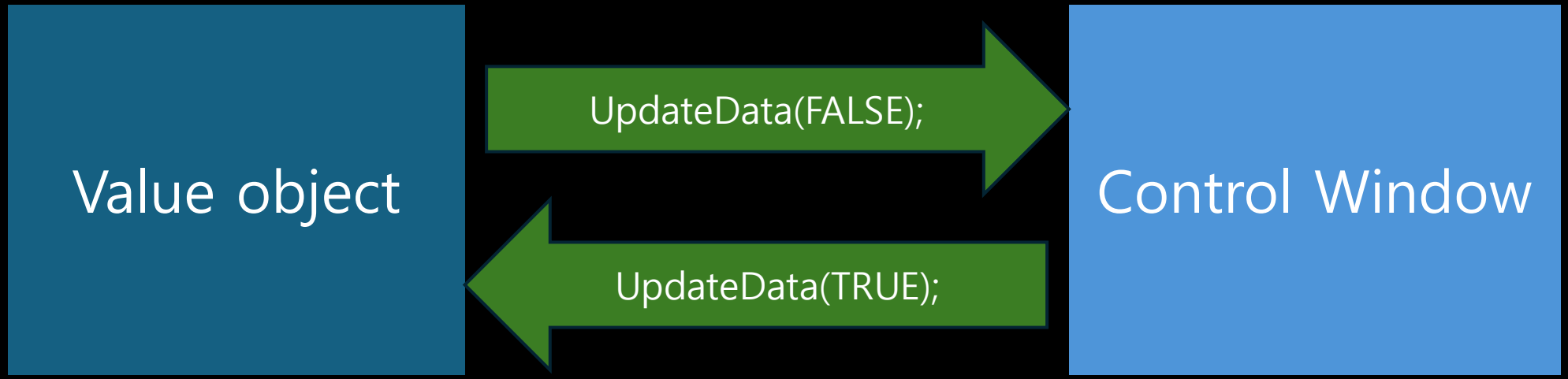
스태틱, 에디트 컨트롤

- **스태틱(Static) 컨트롤 윈도우**
  - 문자열을 표시하는 것에 특수화된 컨트롤
  - 메시지를 받지 않는 것이 기본속성
- **에디트(Edit) 컨트롤 윈도우**
  - 문자열 입력을 입력 받는 기능에 특수화
  - 복사하기와 붙여넣기 기능 제공
  - 속성 설정에 따라 여러 행을 입력 받을 수 있음
  - 스태틱 컨트롤 처럼 보이도록 할 수 있음

DDX/DDV 매커니즘

- **컨트롤 윈도우에 대한 멤버 등록 시 윈도우 객체 혹은 데이터 객체로 등록**
  - 윈도우 객체 사용 시 데이터 동기화 코드 추가
- **컨트롤 윈도우와 데이터 객체 연동 자동화**
  - 컨트롤과 멤버 데이터 간 매핑 설정에 따라 동기화
  - UpdateData() 함수를 호출해 동기화
  - 자료에 대한 유효성 검사 기능 제공

DDX/DDV 매커니즘


Value object
UpdateData(FALSE);
UpdateData(TRUE);
Control Window

통지 메시지

- **컨트롤 윈도우가 부모 윈도우에게 전달하는 메시지**
  - 컨트롤 윈도우가 수신하는 메시지나 변화를 부모 윈도우에게 알리기 위한 메시지
  - 내부적으로 SendMessage() 함수 사용
  - 결과적으로 불필요한 서브 클래싱 차단
- **컨트롤 윈도우 마다 다양한 메시지가 존재**
  - MSDN을 참고해 개발
  - 암기 할 필요 없음

모덜리스(Modeless) 대화상자

- 기존 Modal 방식과 달리 독립적인 프레임 윈도우로 작동
  - 별도 메시지 큐를 가진 것처럼 따로 작동
  - 대화상자가 화면에 표시되더라도 메인 윈도우에 입력 포커스를 주는 것은 물론 완전히 개발적으로 동작
- 정보를 입력 받는 경우보다 알림정보를 표시할 목적으로 자주 사용
  - 확인, 취소 버튼 클릭 시 대화상자 자체가 사라지는 것이 아니라 화면에서만 사라짐

공통 대화상자

- 파일 선택, 색상 선택(폴더 선택은 없음) 등 일반적인 응용 프로그램이 자주 사용하는 대화상자를 제공
- 모달 방식으로 작동
- 파일 선택 대화상자는 윈도우 탐색기와 연동되어 있는 것으로 볼 수 있음

기본 컨트롤 윈도우

다양한 종류의 컨트롤 윈도우를 어떻게 다루는지 학습합니다.

일반 버튼

- 버튼의 형태는 크게 세 가지
  - Push (확장 형식에 해당하는 명령 버튼)
  - Checkbox
  - Radio
- 내부적으로는 메뉴처럼 WM_COMMAND 메시지 발생시키며 구체적인 항목은 ID값으로 식별
- 확장 형식인 명령버튼 제공
- 디폴트 버튼 속성 적용 시 Enter키 입력에 반응

체크 박스

- 토글 기능을 제공하는 버튼 형태
- TRUE/FALSE 값에 대응하며 Value 멤버로 관리 할 수 있음

라디오 버튼

- 여러 개 중 하나를 선택하는 형식의 버튼
- int형식에 Value 멤버 추가 가능
- 여러 라디오 버튼을 한 덩어리로 묶어주기 위해서는 반드시 탭 오더가 연속되어야 하며 그 중 첫 번째 라디오 버튼은 반드시 그룹 속성을 가져야 함

리스트와 콤보 박스

- 리스트 박스는 여러 문자열을 한 목록으로 보여주기 위한 컨트롤 윈도우
  - 여러 정보를 나열해 볼 수 있도록 시각화
  - 기본 속성으로 '정렬'이 적용되어 있음에 주의
- 콤보 박스는 에디트 컨트롤과 리스트 박스를 하나로 합쳐 만들어진 컨트롤 윈도우이며 브라우저 주소 입력창이 가장 대표적
  - 리스트 중 하나를 선택하는 방식으로 입력 편의성 제공

리스트 항목 추가/삭제

- AddString()
  - 인덱스를 고려하지 않고 리스트에 문자열을 추가
  - 리스트 속성에 따라 정렬된 위치에 추가될 수 있음
  - 정렬 속성이 없을 경우 마지막에 추가
- InsertString()
  - 특정 인덱스(Zero-based)에 문자열 추가
  - 정렬 속성을 고려하지 않음

통지 메시지 처리

- 새로운 항목 추가/삭제
- 특정 항목 선택
- 목록 드롭 (콤보 박스)

윈도우 크기 변경에 대응하기

- 윈도우 크기 변경 시 발생하는 WM_SIZE 메시지에 대한 핸들러를 등록해 처리
- 윈도우 크기 조정은 보통 부모 윈도우가 자식 윈도우를 조정하는 것이 일반적
- 자식 윈도우 간 화면비나 위치는 모두 수작업 코딩해야 하며 자동으로 처리해주는 것은 없음
  - 하이브리드 앱이 필요한 이유 (HTML + CSS)

미니 프로젝트

지금까지 공부한 내용을 바탕으로 간단한 PPT 프로그램을 개발합니다.

기능 정의

- **파워포인트처럼 뷰 화면 영역에 도형을 그릴 수 있는 편집기를 제작**
  - 네모, 원 등 기본적인 도형 편집
  - 각 도형 간 Z-order 적용
  - 도형 외곽선 및 브러시 옵션 제공
  - 파일 저장하기, 불러오기 기능 제공
- **개인의 판단에 따라 추가기능 개발**
  - 도형 선택 표시하기 (다양한 방식이 존재할 수 있음)
  - 네트워크 편집 지원

파일 형식 규정

- 도형의 종류는 열거형 상수로 규정
- 도형의 위치 및 크기는 좌상단 좌표 및 폭, 높이를 저장 (CPoint, CSize)
- Z-order는 파일에 저장된 순서를 그대로 적용
  - 가장 먼저 저장된 도형이 화면의 맨 위에 나오는 구조
- MFC 직렬화 기능은 사용하지 않음

도형 그리기 및 이동

- 최초 도형을 그려 넣을 때는 툴바에서 선택한 도형 형식 및 색상을 반영해 추가
- 도형이 추가될 위치 및 크기는 사용자의 마우스 입력으로 결정
- 이미 그려져 있는 도형 중 하나를 선택하면 Z-order를 최상위로 변경하고 이동

더블 버퍼링 적용 및 도형 속성 변경

- 화면 렌더링 후 윈도우 크기 조절 시 깜빡이는 현상이 나타나지 않도록 반드시 **더블 버퍼링 기능 적용**
- 특정 도형을 선택한 후 **속성을 변경했을 때 즉시 반영**
  - 색상, 크기, 폭과 높이 등

파일로 저장하기/불러오기

- **고정길이 바이너리 형식**
  - 파일 헤더, 각종 도형요소
  - 모두 고정길이 구조체로 정의
  - 문자열 포함 시 문자열의 길이를 우선 저장하는 방식적용
- **가변길이 문자열 (XML)**
  - 현재 대부분의 오피스 제품류가 선택하고 있는 방식
  - 파일을 zip 형식으로 압축해 하나의 파일처럼 보임

감사합니다!