

# MFC 프로그램 구조

(7주차)

# 학습개요

- 학습 목표

- MFC 발전 과정과 주요 특징을 개괄적으로 이해한다.
- MFC 최상위 클래스인 CObject가 제공하는 서비스를 이해한다.
- MFC가 제공하는 전역 함수 사용법을 익힌다.
- 응용 프로그램 마법사가 자동으로 생성하는 코드 구조를 이해한다.

- 학습 내용

- MFC 개요
- MFC 구조
- MFC 응용 프로그램 구조
- 실습

# MFC 발전 과정 (1)

연도	개발 도구	MFC 버전	주요 특징
1992	MS C/C++ 7.0	1.0	16비트 윈도우 API를 클래스화, OLE 1.0 지원
1993	비주얼 C++ 1.0	2.0	도큐먼트/뷰 구조 도입, DDX/DDV 지원 AppWizard/ClassWizard를 이용한 코드 생성 지원
1993	비주얼 C++ 1.5x	2.5x	OLE 2.0 지원, ODBC 클래스 추가 속성 페이지/시트 추가
1994	비주얼 C++ 2.x	3.x	32비트 윈도우 API로 전환, 멀티스레드 지원 소켓/MAPI 클래스 추가
1995	비주얼 C++ 4.x (mfc40.dll)	4.x	컨트롤 클래스 추가, DAO와 32비트 ODBC 지원 인터넷 관련 클래스 추가
1997	비주얼 C++ 5.0 (mfc42.dll)	4.21	ATL 추가, 액티브 도큐먼트 지원 DAO 3.5/ODBC 3.0 지원, 인터넷 프로그래밍 지원 향상
1998	비주얼 C++ 6.0 (mfc42.dll)	6.0	ATL 업그레이드, 동적 HTML 지원(CHtmlView) OLE DB 클래스 추가, ADO 지원, 자잘한 클래스 추가/갱신

# MFC 발전 과정 (2)

연도	개발 도구	MFC 버전	주요 특징
2002	비주얼 C++ .NET (mfc70.dll)	7.0	MFC와 ATL의 통합 강화 CException 클래스를 추상 클래스로 변경 CString과 BSTR 사이의 변환 방법 변경 CFile 클래스 변경(64비트 오픈 지원) CTime 클래스 변경(64비트 시간 지원) 부울(Boolean) 표현식 의미 변경(BOOL → bool) ON_MESSAGE 매크로의 매개변수 타입 변경 OLE DB 템플릿 변경
2003	비주얼 C++ .NET 2003 (mfc71.dll)	7.1	ATL 일부 변경
2005	비주얼 C++ 2005 (mfc80.dll)	8.0	ATL 업그레이드 Windows Forms 지원 CTime 클래스 변경(1970년 이후 지원 → 1900년 이후 지원)
2007	비주얼 C++ 2008 (mfc90.dll)	9.0	윈도우 비스타 공용 컨트롤 지원
2008	비주얼 C++ 2008 Feature Pack	9.0	현대적 UI 지원을 위한 다수의 클래스 추가

# MFC 발전 과정 (3)

연도	개발 도구	MFC 버전	주요 특징
2010	비주얼 C++ 2010 (mfc100.dll)	10.0	윈도우 7에 새로 도입된 UI 지원을 위한 클래스 추가 멀티터치와 고해상도 인식 기능 지원 다시 시작 관리자(Restart Manager) 추가 AfxMessageBox()를 대체할 CTaskDialog 클래스 추가 애니메이션과 Direct2D 그래픽스 지원
2012	비주얼 C++ .2012 (mfc110.dll)	11.0	MFC 라이브러리와 정적 링크 시 실행 파일 크기 축소 각종 버그 수정, 윈도우 XP/서버 2003 지원 중단 (SP1 설치 시 윈도우 XP/서버 2003용 코드 생성 지원)
2013	비주얼 C++ 2013 (mfc120.dll)	12.0	각종 버그 수정, 윈도우 XP/서버 2003용 코드 생성 지원

# MFC 주요 특징 (1)

- 윈도우 응용 프로그램 작성에 드는 수고를 줄임
  - 검증된 라이브러리 재사용
  - 응용 프로그램 마법사, 클래스 마법사 같은 자동화된 생성도구의 지원
- API 기반인 SDK 프로그램과 대등한 속도
  - 내부적으로 인라인 함수를 많이 사용함
- 코드 크기 증가 최소화
  - MFC DLL

# MFC 주요 특징 (2)

- API 함수 직접 호출 가능
  - (예) `::ReleaseCapture();`
- C 언어로 작성된 윈도우 응용 프로그램을 쉽게 C++ 언어로 변경 가능
- SDK 프로그래밍에 대한 기반 지식 재활용

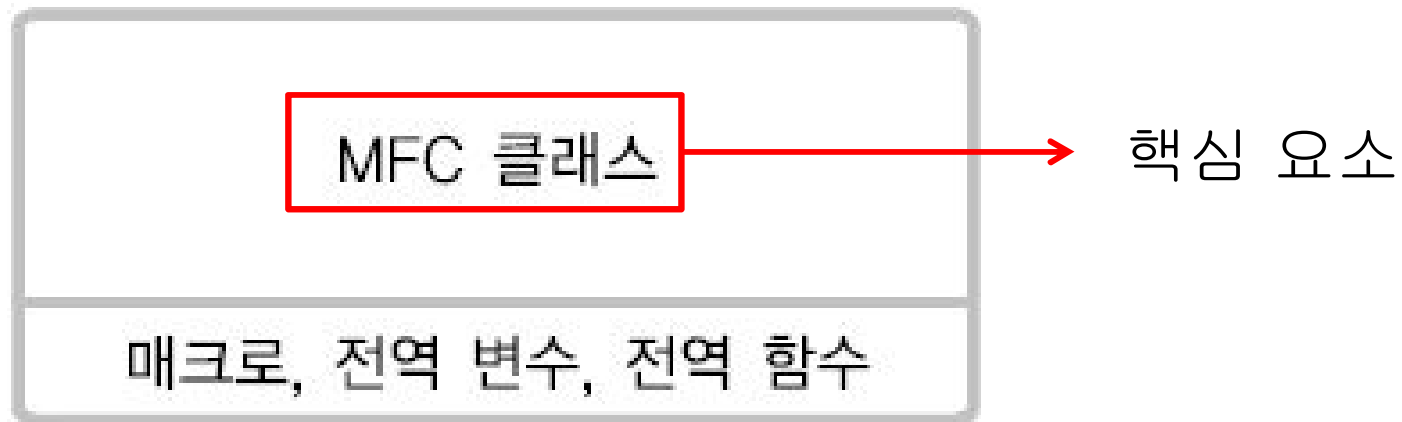
# MFC 주요 특징 (3)

- C++ 언어 이용, C 언어에 비해 API를 다루기 쉬움
  - (예) 디폴트 인자
- API를 직접 사용해서 구현할 경우, 복잡도가 높은 부분은 MFC를 이용해 쉽게 구현
  - 인쇄 기능 지원, 툴바와 상태바 처리, 데이터베이스 지원, OLE, ...



# MFC 구성 요소 (1)

- MFC 구성 요소



# MFC 구성 요소 (2)

- MFC 클래스 계층도

- CObject 클래스를 상속하는 클래스
- CObject 클래스를 상속하지 않는 독립적인 클래스



# CObject 클래스

- CObject 서비스

서비스 이름	기능
실행 시간 클래스 정보	프로그램 실행 중 객체 정보를 알아낸다.
동적 객체 생성	객체를 동적으로 생성한다.
직렬화	객체를 저장하거나 읽어들인다.
타당성 점검	객체 상태를 점검한다.
집합 클래스와의 호환성	서로 다른 클래스 객체를 집합 클래스에 저장할 수 있게 한다.

# 실행 시간 클래스 정보 (1)

- 실행 시간 클래스 정보 기능 추가

```
/* MyClass.h */  
class CMyClass : public CObject  
{  
    DECLARE_DYNAMIC(CMyClass)  
    ...  
};  
  
/* MyClass.cpp */  
#include "MyClass.h"  
IMPLEMENT_DYNAMIC(CMyClass, CObject)  
...
```

# 실행 시간 클래스 정보 (2)

- 실행 시간 클래스 정보 사용 예

```
BOOL IsMyClass(CObject *pObj)
{
    // pObj가 가리키는 객체가 CMyClass 타입인지 확인한다.
    if(pObj->IsKindOf(RUNTIME_CLASS(CMyClass))){
        ...
    }
    else{
        ...
    }
}
```

# 동적 객체 생성 (1)

- 동적 객체 생성 기능 추가

```
/* MyClass.h */
class CMyClass : public CObject
{
    DECLARE_DYNCREATE(CMyClass)
public:
    CMyClass(); // 기본 생성자가 반드시 있어야 한다.
    ...
};

/* MyClass.cpp */
#include "MyClass.h"

IMPLEMENT_DYNCREATE(CMyClass, CObject)
...
```

# 동적 객체 생성 (2)

- 동적 객체 생성 사용 예

```
// 객체를 동적으로 생성한다.  
CRuntimeClass* pRuntimeClass = RUNTIME_CLASS(CMyClass);  
CObject pObject = pRuntimeClass->CreateObject();  
  
// 객체 생성 여부를 확인한다.  
ASSERT(pObject->IsKindOf(RUNTIME_CLASS(CMyClass)));
```

# 직렬화 (1)

- 직렬화 기능 추가 (1/2)

```
/* MyClass.h */
class CMyClass : public CObject
{
    DECLARE_SERIAL(CMyClass)
public:
    CMyClass(); // 기본 생성자가 반드시 있어야 한다.
    virtual void Serialize (CArchive& ar); //가상 함수를 재정의한다.
    ...
};
```



# 직렬화 (2)

- 직렬화 기능 추가 (2/2)

```
/* MyClass.cpp */  
#include "MyClass.h"
```

```
IMPLEMENT_SERIAL(CMyClass, CObject, 1)
```

```
void CMyClass::Serialize(CArchive& ar)  
{  
    // CObject가 제공하는 가상 함수인 Serialize()를 재정의한다.  
}  
...
```

# 주요 매크로

사용 목적	매크로 이름	사용 위치
실행 시간 클래스 정보	DECLARE_DYNAMIC	클래스 선언부(*.H)
	IMPLEMENT_DYNAMIC	클래스 정의부(*.CPP)
실행 시간 클래스 정보, 동적 객체 생성	DECLARE_DYNCREATE	클래스 선언부(*.H)
	IMPLEMENT_DYNCREATE	클래스 정의부(*.CPP)
실행 시간 클래스 정보, 동적 객체 생성, 직렬 화	DECLARE_SERIAL	클래스 선언부(*.H)
	IMPLEMENT_SERIAL	클래스 정의부(*.CPP)

# 타당성 점검 (1)

- 타당성 점검 기능 추가 (1/2)

```
/* MyClass.h */  
class CMyClass : public CObject  
{  
    // 멤버 변수  
    int m_start, m_end;  
public:  
    virtual void AssertValid( ) const; // 가상 함수를 재정의한다.  
    ...  
};
```

# 타당성 점검 (2)

- 타당성 점검 기능 추가 (2/2)

```
/* MyClass.cpp */
#include "MyClass.h"

virtual void CMyClass::AssertValid( ) const
{
    // 베이스 클래스의 AssertValid() 함수를 먼저 호출한다.
    CObject::AssertValid();

    // 멤버 변수 m_start 값이 0보다 큰지 검사한다.
    ASSERT(m_start > 0);

    // 멤버 변수 m_end 값이 100보다 작은지 검사한다.
    ASSERT(m_end < 100);}

```

# 집합 클래스와의 호환성

- CObject 포인터를 저장할 수 있는 집합 클래스

종류	클래스 이름
배열	CObArray, CArray(템플릿 클래스)
리스트	CObList, CList(템플릿 클래스)
맵	CMapWordToOb, CMapStringToOb, CMap(템플릿 클래스)

# MFC 전역 함수

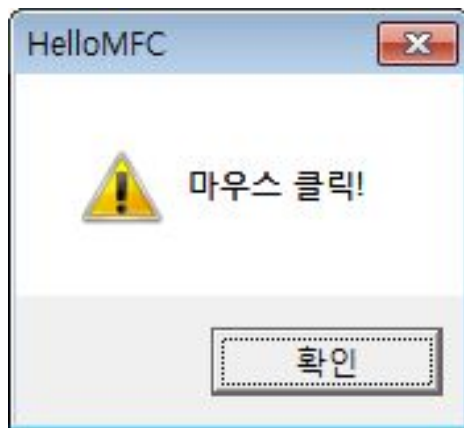
함수 이름	기능
AfxMessageBox()	메시지 상자를 표시한다.
AfxGetApp()	응용 프로그램 객체의 주소를 리턴한다.
AfxGetMainWnd()	메인 윈도우 객체의 주소를 리턴한다.
AfxGetAppName()	응용 프로그램의 이름을 리턴한다.
AfxGetInstanceHandle()	인스턴스 핸들을 리턴한다.
AfxRegisterWndClass()	윈도우 클래스를 등록한다.
AfxBeginThread()	스레드를 시작한다.
AfxEndThread()	스레드를 종료한다.

# AfxMessageBox( ) 함수

- 사용 예

```
void CMainFrame::OnLButtonDown(UINT nFlags, CPoint point)
{
    AfxMessageBox(_T("마우스 클릭!"));
}
```

- 결과



# AfxGetApp( ), AfxGetMainWnd( ), AfxGetAppName( ) 함수 (1)

## • 사용 예

```
#include <afxwin.h>
#include <locale.h>

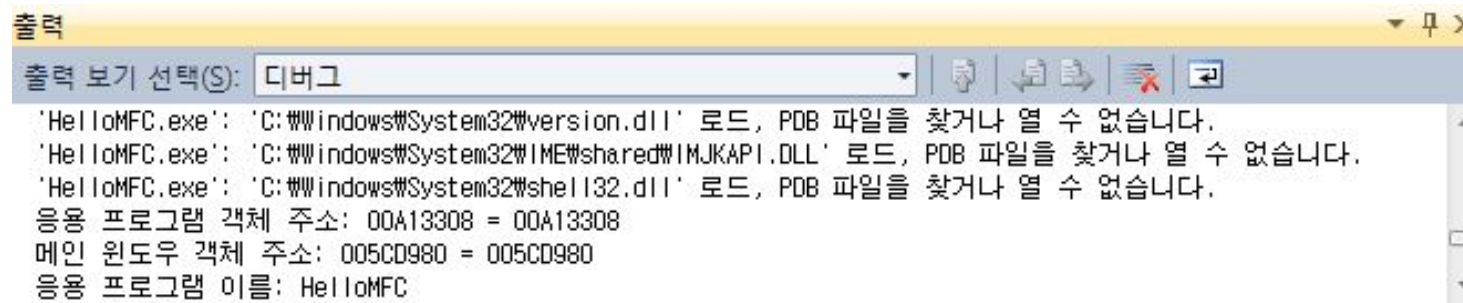
...
BOOL CHelloApp::InitInstance()
{
    _tsetlocale(LC_ALL, _T("")); // TRACE() 매크로에서 한글(유니코드) 출력에 필요!
    m_pMainWnd = new CMainFrame;
    m_pMainWnd->ShowWindow(m_nCmdShow);
    return TRUE;
}

...
void CMainFrame::OnLButtonDown(UINT nFlags, CPoint point)
{
    TRACE(_T("응용 프로그램 객체 주소: %p = %pWn"), AfxGetApp(), &theApp);
    TRACE(_T("메인 윈도우 객체 주소: %p = %pWn"), AfxGetMainWnd(), theApp.m_pMainWnd);
    TRACE(_T("응용 프로그램 이름: %sWn"), AfxGetAppName());
}
```



# AfxGetApp( ), AfxGetMainWnd( ), AfxGetAppName( ) 함수 (2)

## • 결과



```
출력
출력 보기 선택(S): 디버그
'HelloMFC.exe': 'C:\Windows\System32\version.dll' 로드, PDB 파일을 찾거나 열 수 없습니다.
'HelloMFC.exe': 'C:\Windows\System32\IME\shared\IMJAPI.DLL' 로드, PDB 파일을 찾거나 열 수 없습니다.
'HelloMFC.exe': 'C:\Windows\System32\shell32.dll' 로드, PDB 파일을 찾거나 열 수 없습니다.
응용 프로그램 객체 주소: 00A13308 = 00A13308
메인 윈도우 객체 주소: 005CD980 = 005CD980
응용 프로그램 이름: HelloMFC
```

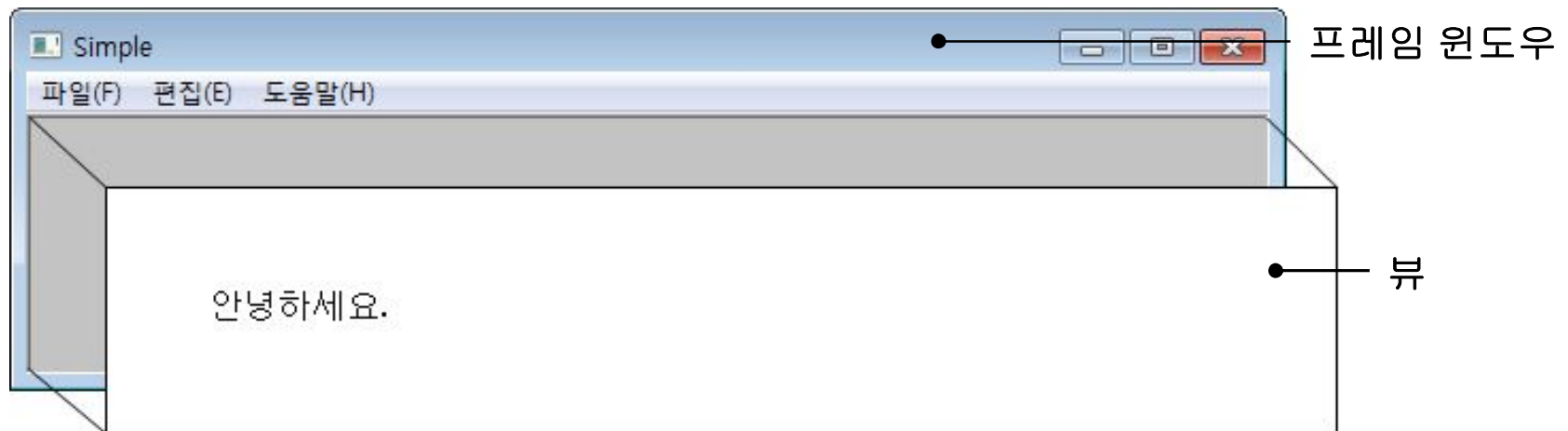
# AfxGetInstanceHandle( ) 함수

- 사용 예

```
void CMainFrame::OnLButtonDown(UINT nFlags, CPoint point)
{
    // 인스턴스 핸들값은 실행 파일이 로드된 가상 메모리 주소다.
    TRACE("인스턴스 핸들: %p\n", AfxGetInstanceHandle());
}
```

# MFC 응용 프로그램 구조 (1)

- 두 개의 윈도우로 구성된 프로그램
  - 프레임 윈도우(=메인 윈도우) + 뷰

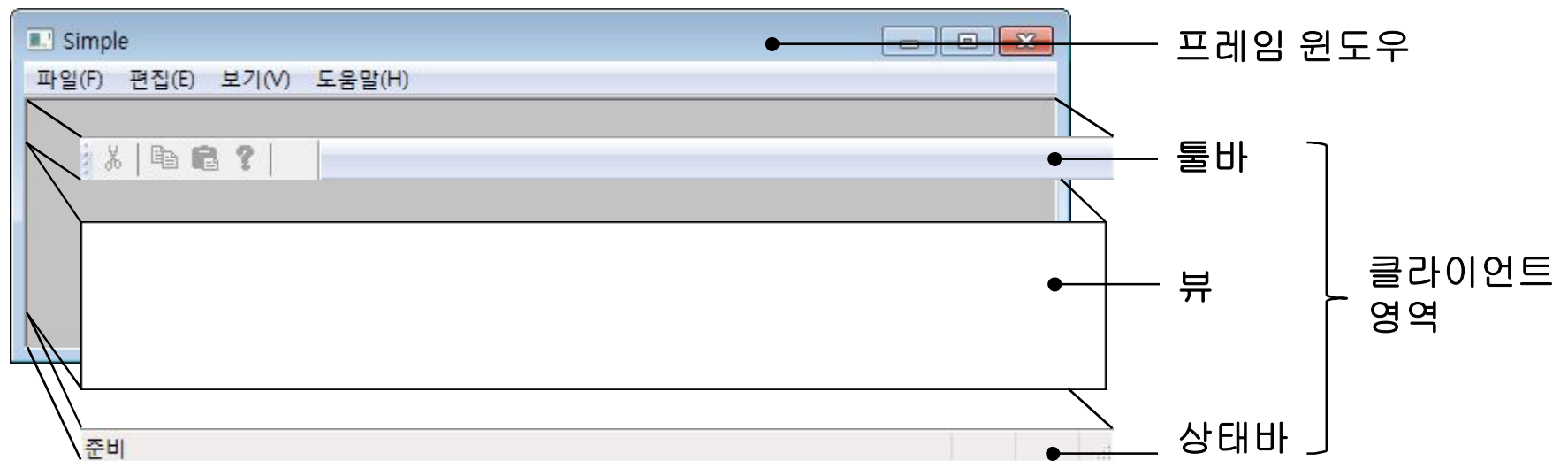


# MFC 응용 프로그램 구조 (2)

- 프레임 윈도우와 뷰의 관계 : 부모-자식 관계
- 자식 윈도우의 특징
  - 항상 부모 윈도우의 클라이언트 영역 내에 위치한다.
  - 부모 윈도우가 움직이면 같이 이동한다(상대 위치 고정).
  - 부모 윈도우가 최소화되면 자식 윈도우는 숨겨진다.
  - 부모 윈도우가 없어지면 자식 윈도우도 없어진다.

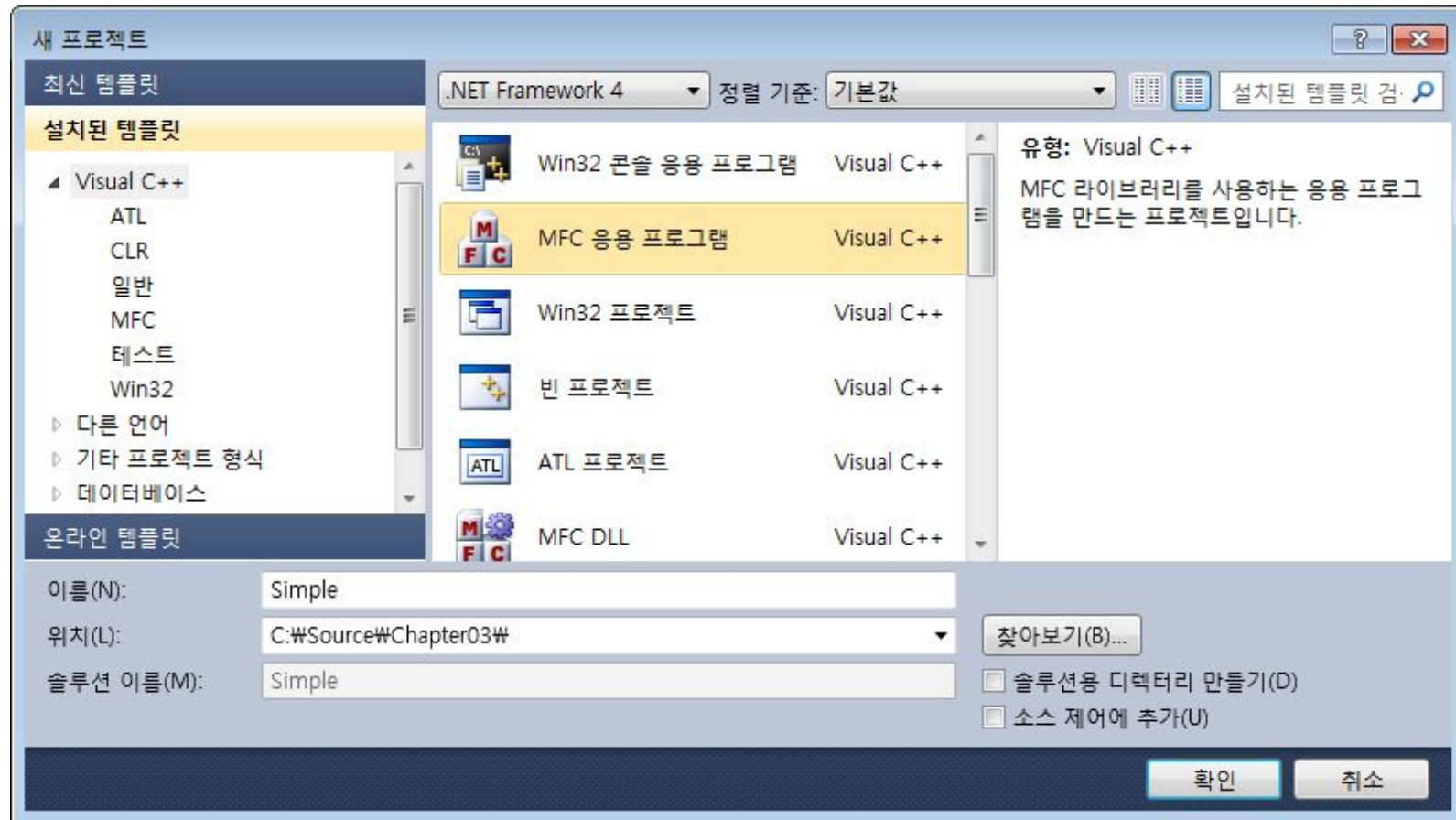
# MFC 응용 프로그램 구조 (3)

- 좀더 일반적인 MFC 프로그램의 구조



# 응용 프로그램 마법사를 이용한 MFC 응용 프로그램 생성 (1)

- 프로젝트 종류 선택



# 응용 프로그램 마법사를 이용한 MFC 응용 프로그램 생성 (2)

## • MFC 응용 프로그램 마법사 - 응용 프로그램 종류



# 응용 프로그램 마법사를 이용한 MFC 응용 프로그램 생성 (3)

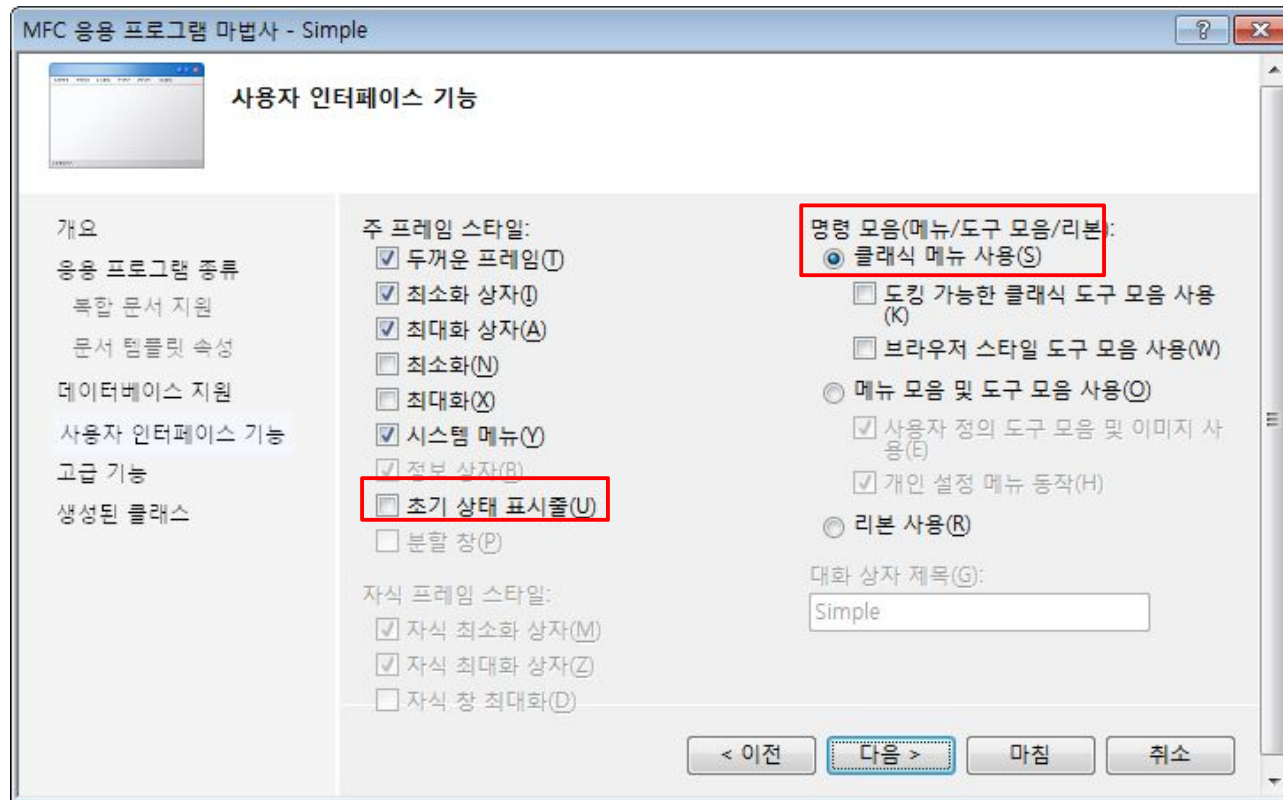
- MFC 응용 프로그램 마법사 – 데이터베이스 지원





# 응용 프로그램 마법사를 이용한 MFC 응용 프로그램 생성 (4)

- MFC 응용 프로그램 마법사 – 사용자 인터페이스 기능



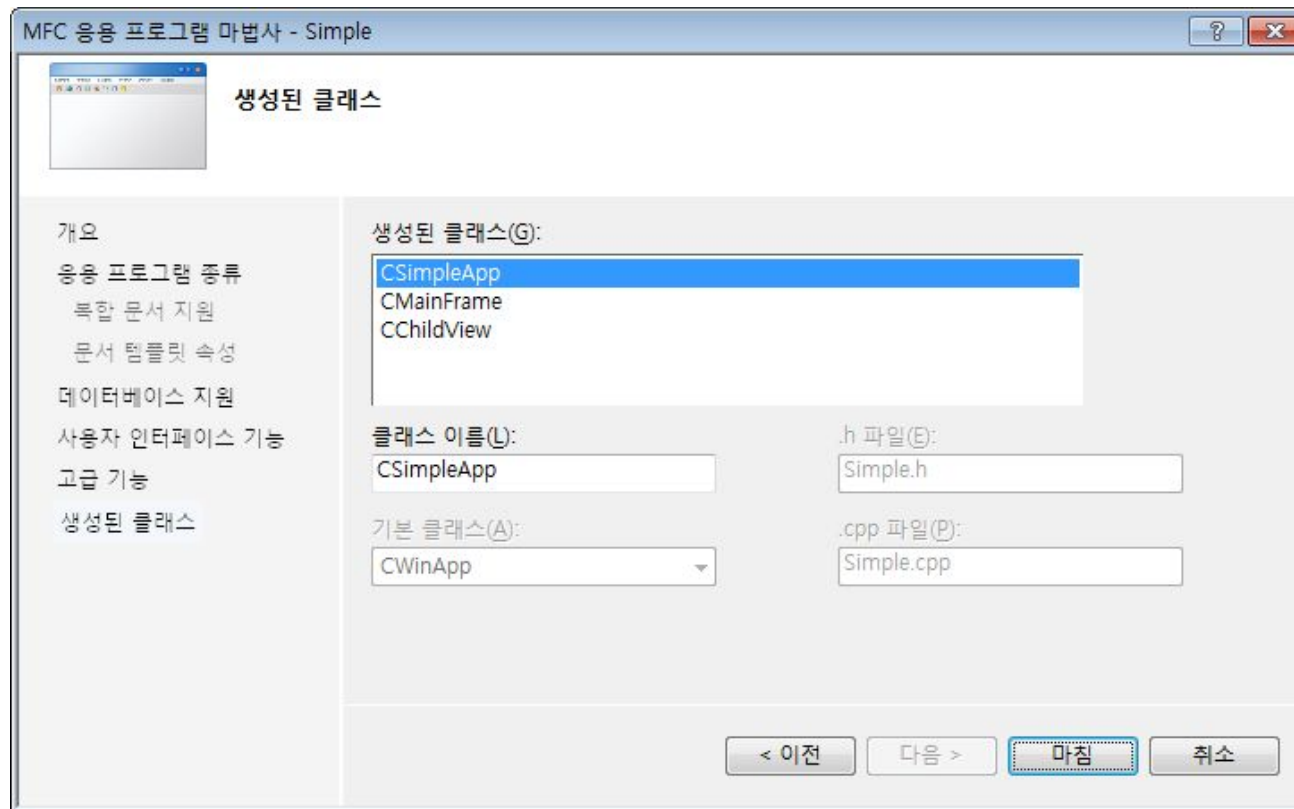
# 응용 프로그램 마법사를 이용한 MFC 응용 프로그램 생성 (5)

## • MFC 응용 프로그램 마법사 - 고급 기능



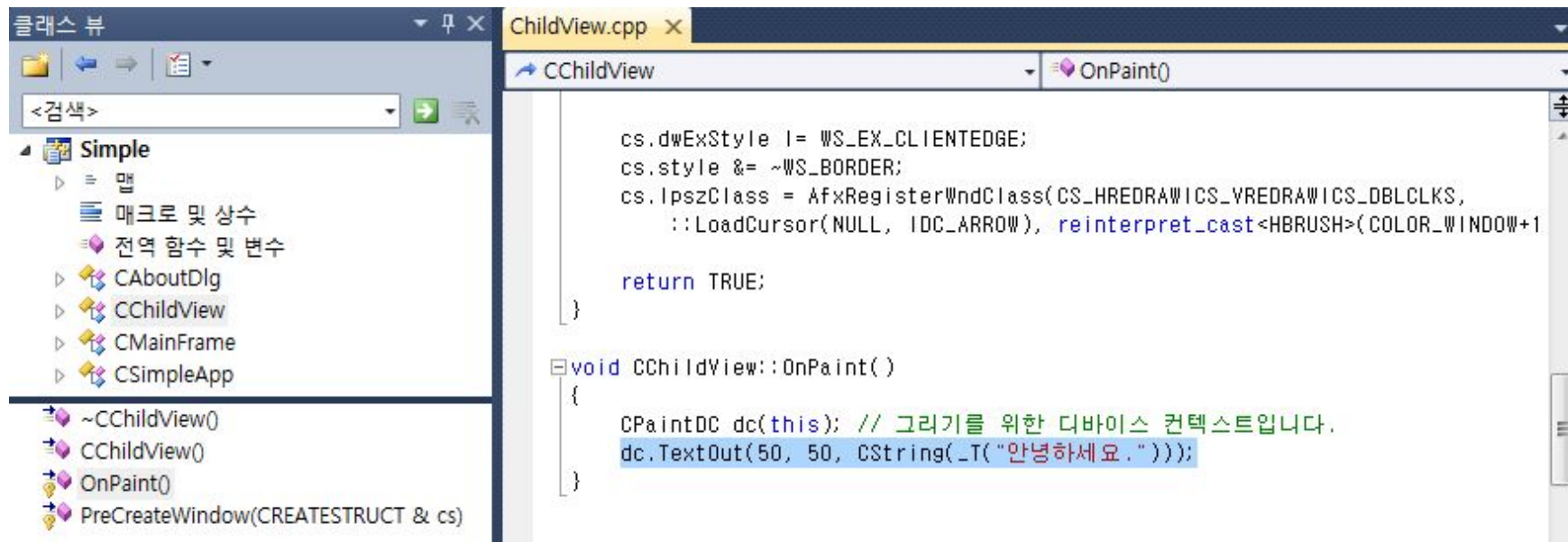
# 응용 프로그램 마법사를 이용한 MFC 응용 프로그램 생성 (6)

- MFC 응용 프로그램 마법사 - 생성된 클래스



# 응용 프로그램 마법사를 이용한 MFC 응용 프로그램 생성 (7)

- 코드 추가



실습 - MFC 응용 프로그램 마법사

# 학습정리

- 검증된 라이브러리 재사용과 응용 프로그램 마법사, 클래스 마법사 같은 자동화된 생성도구의 지원을 통해 윈도우 응용 프로그램 작성에 드는 수고를 줄일 수 있습니다.
- 내부적으로 인라인 함수를 많이 사용하므로 API 기반인 SDK 프로그램과 대등한 속도를 보입니다.
- 코드 크기 증가 최소화하기 위해 MFC 라이브러리를 dll로 제공합니다.
- MFC 응용 프로그램 개발 시 API 함수를 직접 호출해 사용할 수 있습니다.
- CObject 클래스는 실행 시간 클래스 정보, 동적 객체 생성, 직렬화, 타당성 점검, 집합 클래스와의 호환성 서비스를 제공합니다.
- 윈도우 프로그램은 최소한 두 개의 윈도우, 메인 윈도우에 해당하는 프레임 윈도우와 클라이언트 영역의 뷰 윈도우로 구성됩니다.