파일 입출력 I

(12주차)

학습개요

- 학습 목표
  - CFile 클래스를 이용한 파일 입출력 기법을 익힌다.
  - 도큐먼트/뷰 구조를 개괄적으로 이해한다.

- 학습 내용
  - 일반 파일 입출력
  - 도큐먼트/뷰 구조
  - 실습

파일 입출력 개요

- 파일 입출력 방법
  - 일반 파일 입출력
    - CFile (파생) 클래스
    - Read(), Write() 등의 함수 이용
  - 직렬화
    - CArchive 클래스
    - << 또는 >> 연산자 이용

파일 입출력 개요

- MFC 클래스 계층도

✓ 파일 입출력 기능 제공
✓ 파생 클래스에 공통의 인터페이스 제공

CFile 클래스

- CFile 클래스가 제공하는 핵심 입출력 연산
  - 파일을 열거나 생성한다(Open).
  - 데이터를 읽는다(Read).
  - 데이터를 쓴다(Write).
  - 입출력 위치를 변경한다(Seek).
  - 파일을 닫는다(Close).

CFile 클래스

- 열기와 생성

```
try {
    CFile file(_T("mytest.txt"), CFile::modeReadWrite);
}
catch (CFileException* e) {
    e->ReportError();
    e->Delete();
}
```

```
CFile file;
CFileException e;
if(!file.Open(_T("mytest.txt"), CFile::modeReadWrite, &e))
    e.ReportError();
```

CFile 클래스

- 파일 접근과 공유 모드

플래그	의미
CFile::modeCreate	파일을 무조건 생성한다. 이름이 같은 파일이 있다면 크기를 0으로 바꾼다.
CFile::modeNoTruncate	| 연산자로 CFile::modeCreate 플래그와 조합해서 사용하면, 이름이 같은 파일이 있을 때 크기를 0으로 바꾸지 않고 그 파일을 연다.
CFile::modeRead	파일을 읽기 전용 모드로 열거나 생성한다.
CFile::modeReadWrite	파일을 읽기 및 쓰기 모드로 열거나 생성한다.
CFile::modeWrite	파일을 쓰기 전용 모드로 열거나 생성한다.
CFile::shareDenyNone	파일에 대한 읽기와 쓰기를 다른 프로세스에 허용한다.
CFile::shareDenyRead	다른 프로세스가 파일을 읽는 것을 금한다.
CFile::shareDenyWrite	다른 프로세스가 파일에 쓰는 것을 금한다.
CFile::shareExclusive	다른 프로세스가 파일을 읽거나 파일에 쓰는 것을 금한다.

CFile 클래스

- 닫기 : 방법1

```
void CExFileView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CFile file;
    CFileException e;
    if(!file.Open(_T("mytest.txt “), CFile::modeReadWrite, &e)){
        e.ReportError();
        return;
    }
    ...
} // CFile::~CFile() 소멸자가 호출된다.
```

CFile 클래스

- 닫기 : 방법2 (1/2)

```
void CChildView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CFile file;
    CFileException e;
    if(!file.Open(_T("mytest1.txt"), CFile::modeReadWrite
       | CFile::modeCreate | CFile::modeNoTruncate, &e))
    {
      e.ReportError();
      return;
    }
    ...
    file.Close();
```

CFile 클래스

```
if(!file.Open(_T("mytest2.txt"), CFile::modeReadWrite
        | CFile::modeCreate | CFile::modeNoTruncate, &e))
    {
      e.ReportError();
      return;
    }
    ...
    file.Close();
    ...
}
```

CFile 클래스

```
UINT CFile::Read(void* lpBuf, UINT nCount);
void CFile::Write(const void* lpBuf, UINT nCount);
```

```
ULONGLONG CFile::Seek(LONGLONG lOff, UINT nFrom); // MFC 7.0 ~
LONG CFile::Seek(LONG lOff, UINT nFrom); // ~ MFC 6.0
```

nFrom	의미
CFile::begin	파일의 처음 위치에서 lOff만큼 파일 포인터 이동
CFile::current	현재 파일 포인터 위치에서 lOff만큼 파일 포인터 이동
CFile::end	파일의 끝 위치에서 lOff만큼 파일 포인터 이동

CFile 클래스

- 기타 함수
  - CFile::GetLength(), CFile::SetLength()
    - 파일의 현재 크기를 얻거나 변경한다.
  - CFile::GetPosition()
    - 파일 포인터의 현재 위치를 얻는다.
  - CFile::LockRange(), CFile::UnlockRange()
    - 파일의 일정 영역을 잠그거나 해제한다. 잠근 영역은 다른 프로세스가 접근할 수 없다.
  - CFile::GetFilePath(), CFile::GetFileName()
    - 파일의 전체 경로(Full Path)와 이름을 얻는다.

CMemFile 클래스

- 사용 예

```
void CFileView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CMemFile file;

    // 메모리 파일에 데이터 쓰기
    int a = 100;
    file.Write(&a, sizeof(a));

    // 메모리 파일에서 데이터 읽기
    file.SeekToBegin();
    int b;
    file.Read(&b, sizeof(b));

    // 읽은 데이터 출력하기
    TRACE(_T("b = %d₩n"), b);
}
```

CStdioFile 클래스

- 사용 예 (1/2)

```
void CChildView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CStdioFile file1;
    CFileException e;
    if(!file1.Open(_T("test1.txt"), CFile::modeRead, &e)) {
        e.ReportError();
        return;
    }

    CStdioFile file2;
    if(!file2.Open(_T("test2.txt"), CFile::modeWrite
    |CFile::modeCreate, &e)) {
```

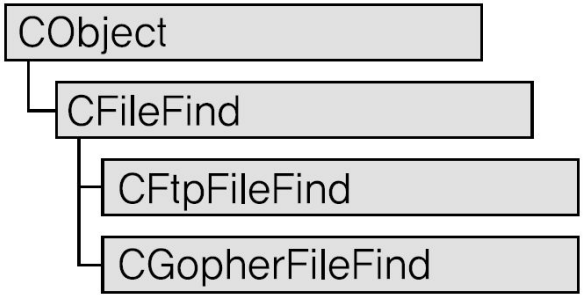
CStdioFile 클래스

- 사용 예 (2/2)

```
        e.ReportError();
        return;
    }
    CString str;
    while(file1.ReadString(str)) {
        str.MakeUpper(); // 대문자로 바꾼다.
        file2.WriteString(str + "₩n");
    }
}
```

CFileFind 클래스

- MFC 클래스 계층도



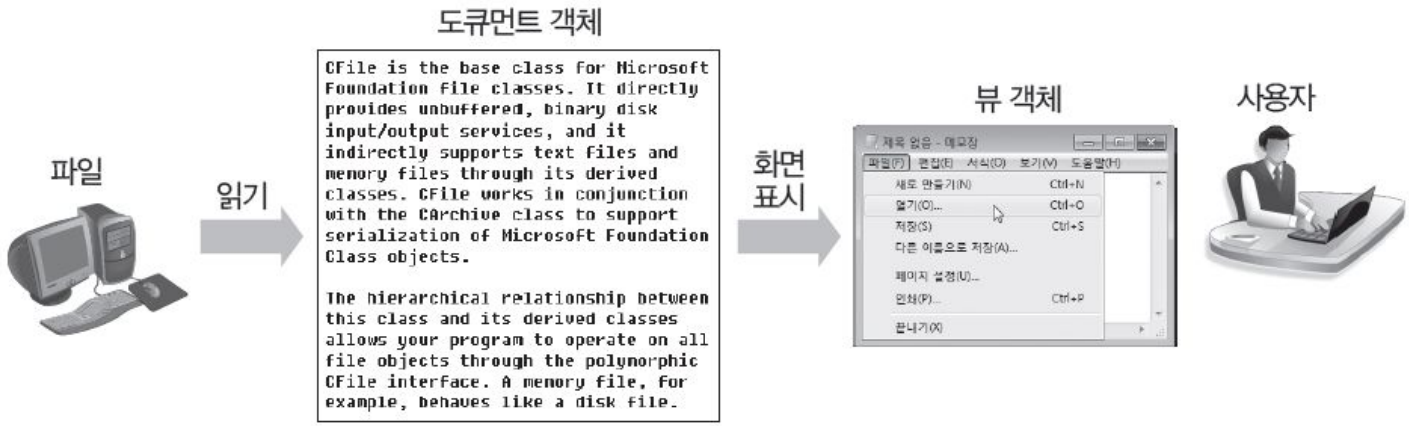
CFileFind 클래스

- 사용 예

```
void CChildView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CFileFind finder;
    BOOL bWorking = finder.FindFile(_T("*.*"));
    while(bWorking) {
        bWorking = finder.FindNextFile();
        if(finder.IsDirectory())
            TRACE(_T("[%s]₩n"), (LPCTSTR)finder.GetFileName());
        else
            TRACE(_T("%s₩n"), (LPCTSTR)finder.GetFileName());
    }
}
```

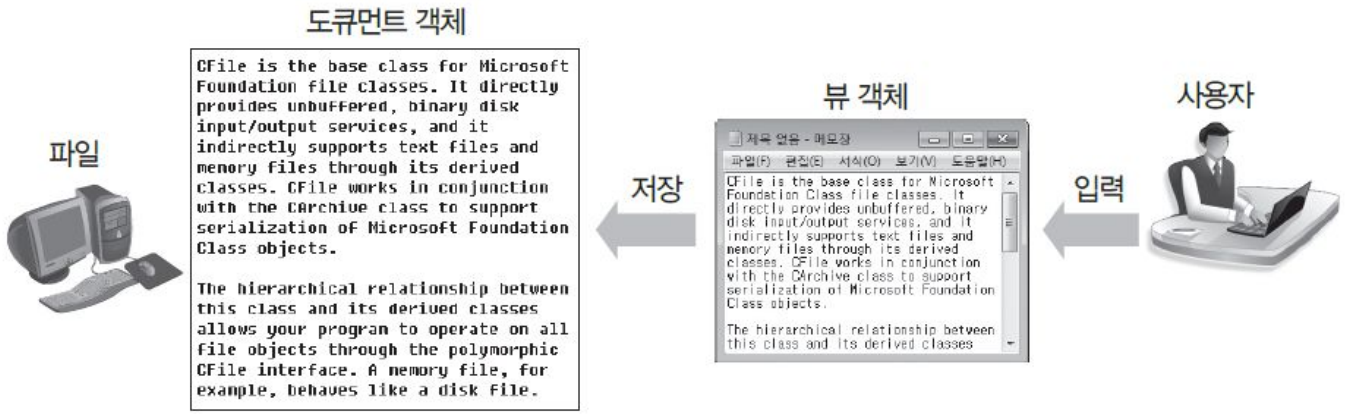
도큐먼트와 뷰

- 디스크에 저장된 파일 데이터를 읽는 경우



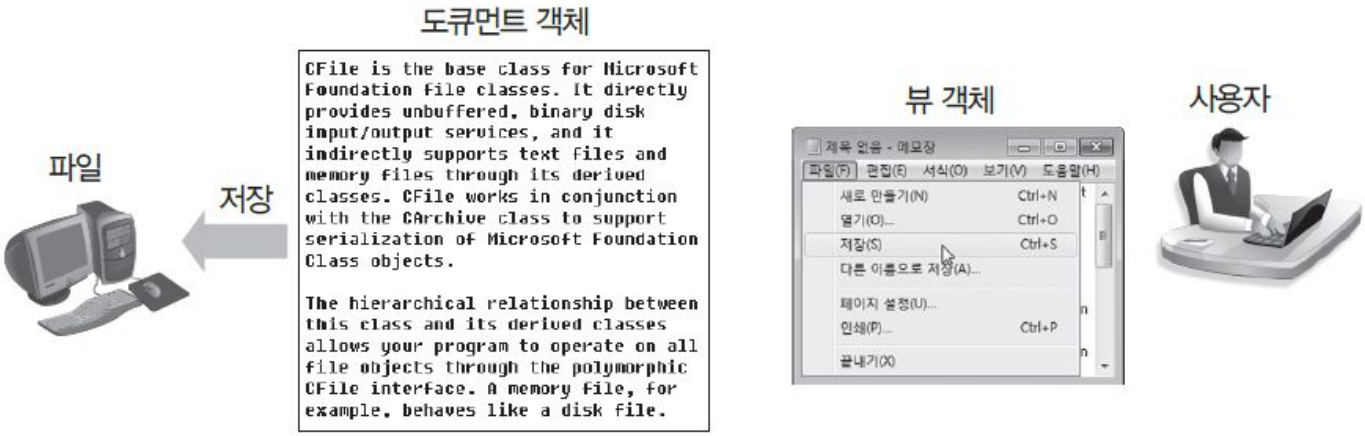
도큐먼트와 뷰

- 사용자가 데이터를 입력하는 경우



도큐먼트와 뷰

- 입력된 데이터를 디스크 파일에 저장하는 경우



도큐먼트와 뷰 클래스의 역할

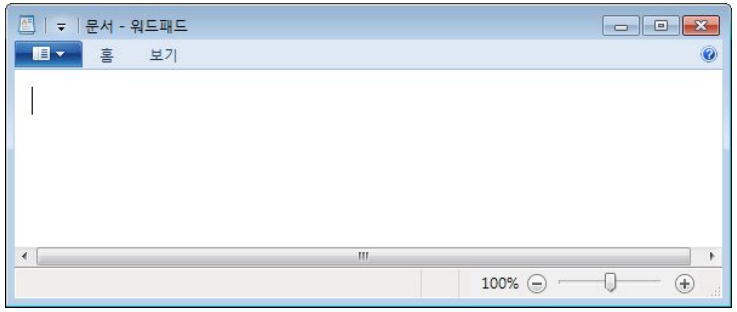
클래스	역할
도큐먼트	데이터를 저장하거나 읽어들인다. 데이터의 변경 사항이 생기면 뷰의 화면을 갱신한다.
뷰	데이터를 화면에 표시한다. 사용자와의 상호 작용을 담당한다.

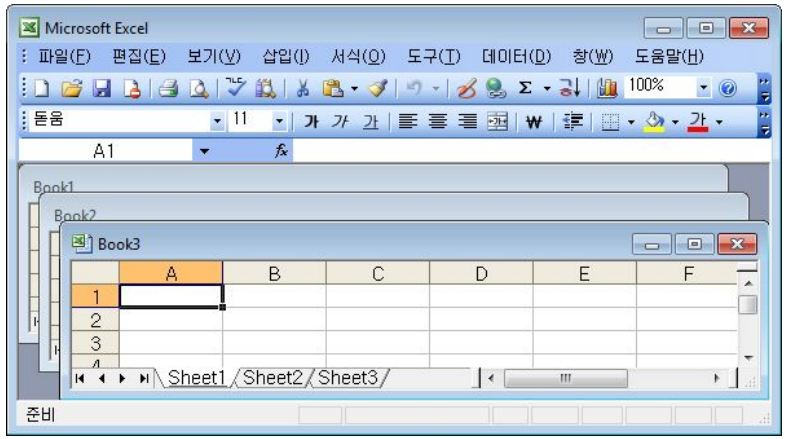
도큐먼트/뷰 구조의 장점

- 서로 다른 기능을 도큐먼트와 뷰 클래스로 분리해서 구현하므로 개념적으로 이해하기 쉽다.
- 도큐먼트 하나에 뷰가 여러 개 존재하는 모델을 구현하기 쉽다.
  - 예) 비주얼 C++ 2013 편집창
- MFC에서 도큐먼트/뷰 구조를 위해 제공하는 부가 서비스를 이용할 수 있다.
  - 예) 직렬화

SDI와 MDI

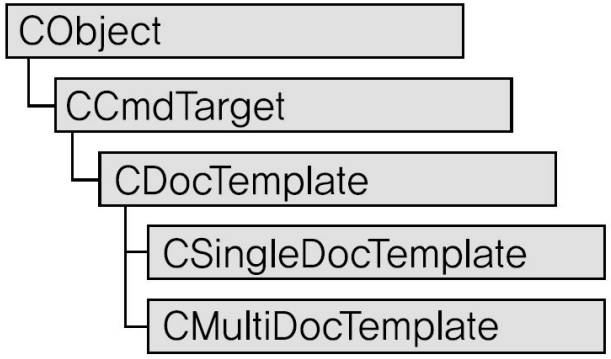
- 다룰 수 있는 문서의 개수에 따라 구분





도큐먼트 템플릿

- 도큐먼트, 프레임 윈도우, 뷰 클래스 정보를 유지
- 필요에 따라 해당 객체를 동적으로 생성
- MFC 클래스 계층도



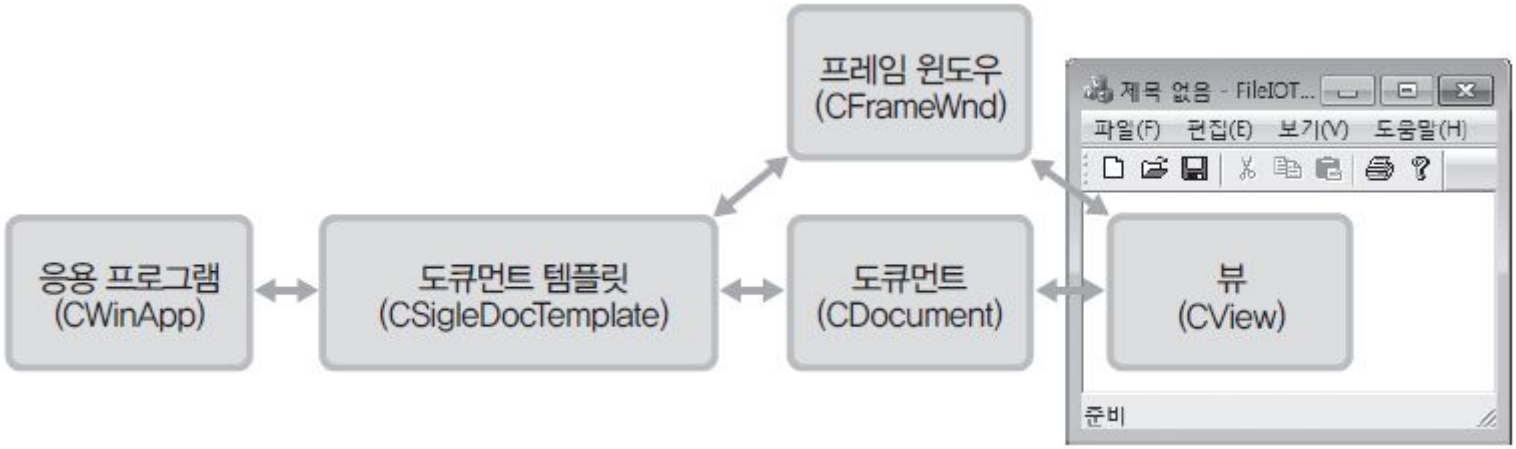
도큐먼트/뷰 템플릿 생성 및 등록

- InitInstance() 함수

```
BOOL CFileIOTestApp::InitInstance()
{
    ...
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
      IDR_MAINFRAME, // 리소스 ID
      RUNTIME_CLASS(CFileIOTestDoc), // 도큐먼트 클래스 정보
      RUNTIME_CLASS(CMainFrame), // 프레임 윈도우 클래스 정보
      RUNTIME_CLASS(CFileIOTestView)); // 뷰 클래스 정보
    if(!pDocTemplate)
      return FALSE;
    AddDocTemplate(pDocTemplate); // 응용 프로그램 객체에 도큐먼트 템플릿 등록
    ...
}
```

주요 객체의 생성 관계(SDI)

생성 주체	생성되는 것
①응용 프로그램 객체	②도큐먼트 템플릿 객체
도큐먼트 템플릿 객체	③도큐먼트 객체, ④프레임 윈도우 객체
프레임 윈도우 객체	⑤뷰 객체



실습

학습정리

- 일반 파일 입출력에서는 CFile 클래스 혹은 CMemFile, CStdioFile과 같은 파생 클래스의 Open(), Read(), Write(), Seek(), Close() 등의 함수를 이용합니다.
- CFile 객체는 소멸자 자동 호출에 의한 묵시적인 파일 닫기가 가능하며, Close() 함수 직접 호출을 통한 명시적 파일 닫기가 가능합니다.
- 객체 직렬화에서는 CArchive 클래스의 << 또는 >> 연산자를 이용합니다.
- 도큐먼트/뷰 구조에서 도큐먼트 객체는 데이터를 저장하거나 읽어들이는 역할을 수행하며, 데이터의 변경 사항이 생기면 뷰의 화면을 갱신합니다.
- 도큐먼트/뷰 구조에서 뷰 객체는 데이터를 화면에 표시하는 역할을 수행하며, 사용자와의 상호 작용을 담당합니다.
- 도큐먼트/뷰 구조는 서로 다른 기능을 도큐먼트와 뷰 클래스로 분리해서 구현하므로 개념적으로 이해하기 쉬우며, 도큐먼트 하나에 뷰가 여러 개 존재하는 모델을 구현하기 쉬우며, 객체 직렬화와 같은 MFC에서 도큐먼트/뷰 구조를 위해 제공하는 부가 서비스를 쉽게 이용할 수 있습니다.
- 응용 애플리케이션이 다루는 도큐먼트 객체의 수에 따라 SDI(Single Document Interface)와 MDI(Multiple Document Interface)로 구분하며, 이 때 도큐먼트 템플릿 클래스를 통해 도큐먼트, 프레임 윈도우, 뷰 클래스 정보를 유지합니다.