軟件架構 與 設計模式

Software Architecture & Design Patterns

(補充講義)

* Overview & Concepts
* Object-Oriented Principles
* Design Patterns in GoF

侯捷

2021/09/06,13,18,22,27,29,
10/11,13,18,20,25,27　同濟大學 / 上海嘉定　(周一周三 網課：19:00-20:35)

侯老师您好：我是软件设计模式的学生....。听完今晚的课程后我在实验时发现一个问题不太懂：图中的代码里，ptr是一个返回值为指针的函数，并且调用是没有问题的。但是我在main函数里定义了一个*p()，编译器并没有报错，按照我的理解在main函数中是不能定义其它函数的，根据括号和指针运算符的结合优先级，这应该也是一个返回值为指针的函数，但是我们也无法调用它，加了括号会显示报错，不加括号直接调用也不会输出任何内容，请问这个*p()在main函数下该如何理解呢？还请您不吝赐教！

```
int *ptr(int atr) {

    int *pt = NULL;
    pt = &atr;
    return pt;
}

int main() {

    int *p();
    int temp = 3;
    cout << ptr(temp) << "\n";
    return 0;

}
```

我們來看一段完整的程序➔
其中 L4~L16等同於你的代碼。

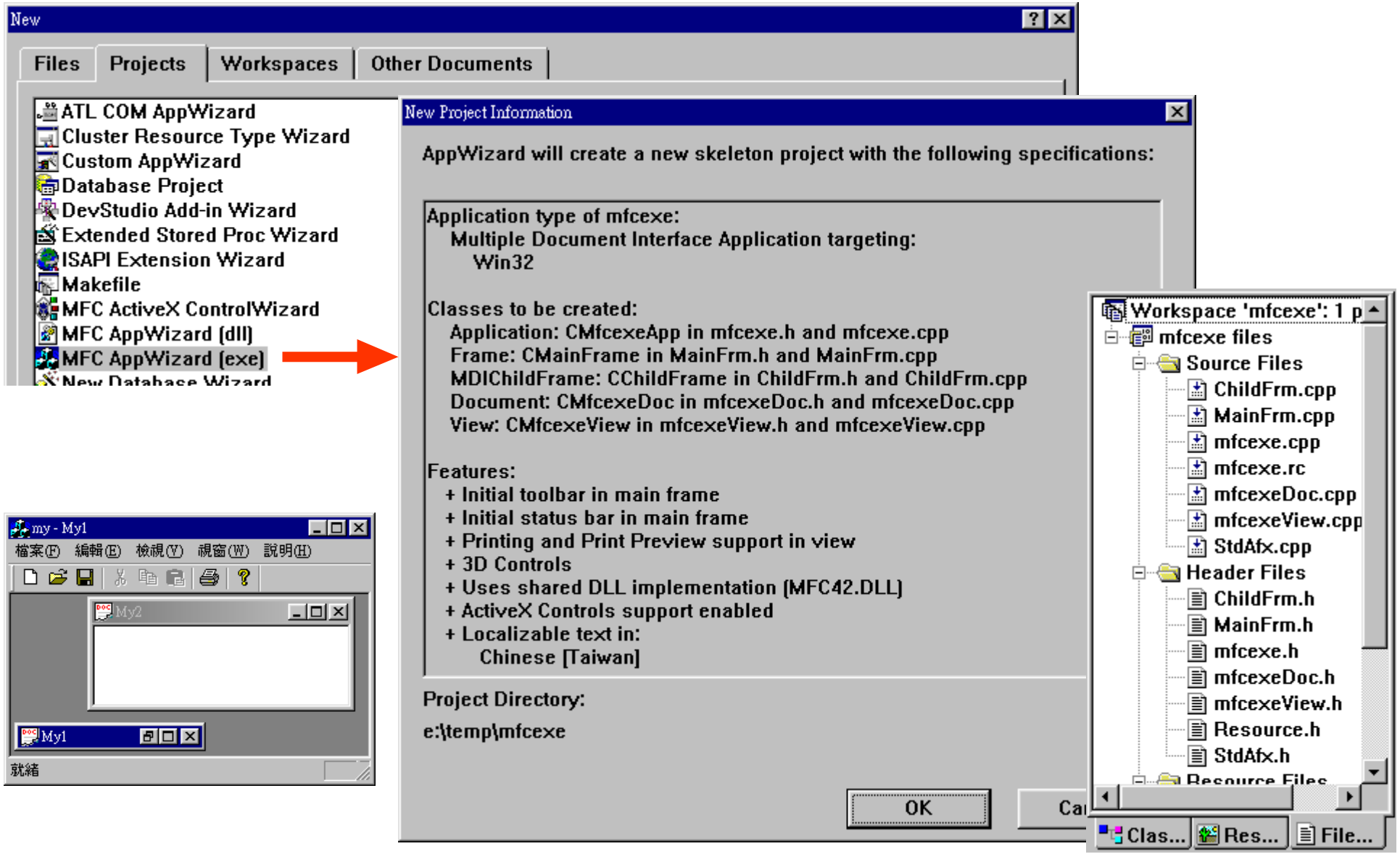
你必然注意到了，我放 * 的地方和你不相同。請以後按我的方式來放，意義才正確，有助於你理解所謂指針。

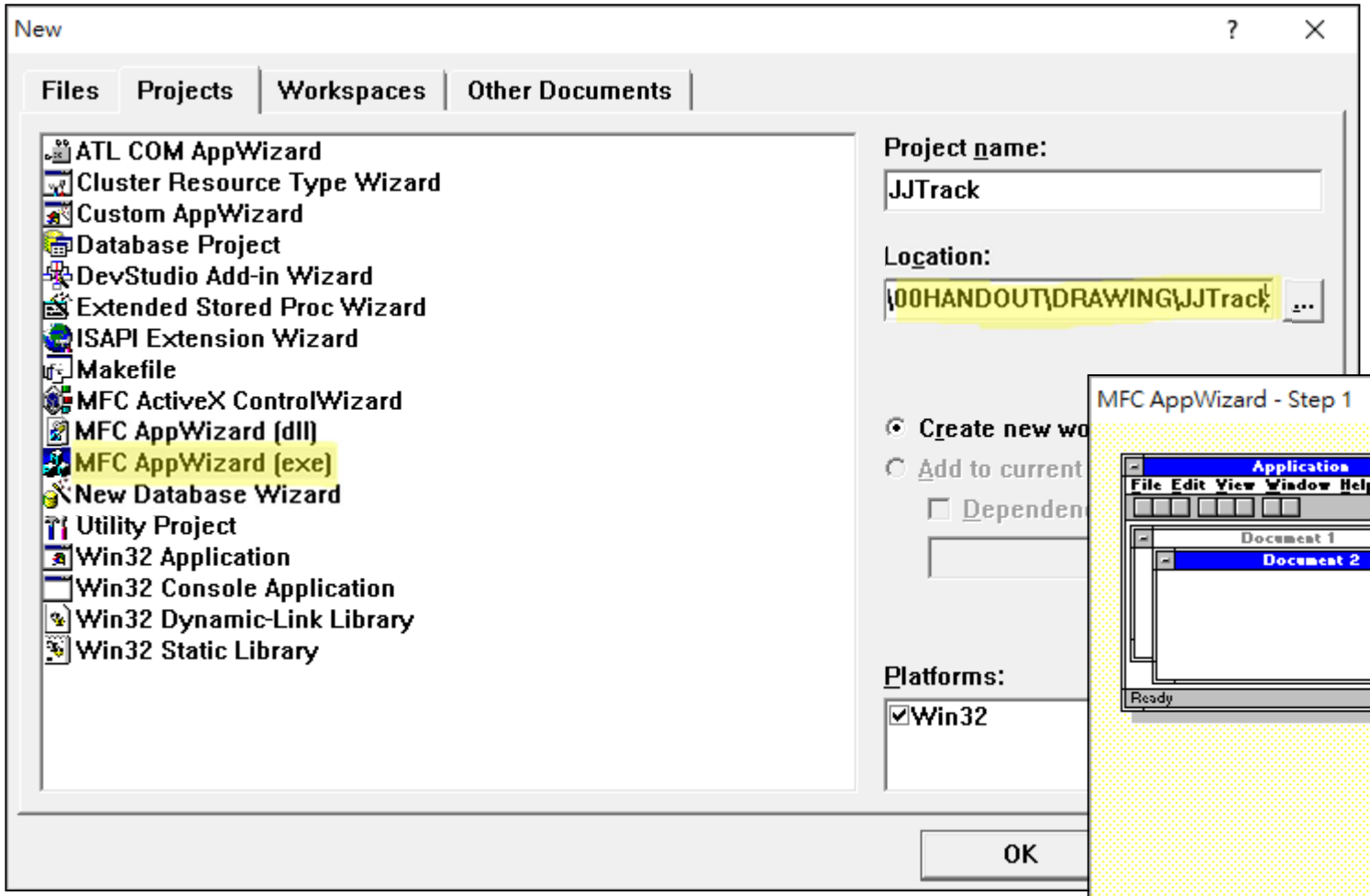
L13 的意思是：p 是個變量，其 type 是 pointer to int。p 之後的 () 用來給予初值。由於 p 的 type "pointer to int" 並不是個 class，我們看不到這個 type 的 ctor (構造函數) 是怎麼設計的，所以我們不知道在沒指定初值的情況下 p 獲得的初值是什麼。不過從 L16 結果可知，在我手上這個編譯環境中，它獲得的初值是 1 (不同的編譯器可能會有不同的結果)。
L19 我明確給了 p 初值，是 nullptr，也就是 0。
L24 我再一次明確給 p 初值為 &temp，於是 L25 和 L26 的輸出完全相同。

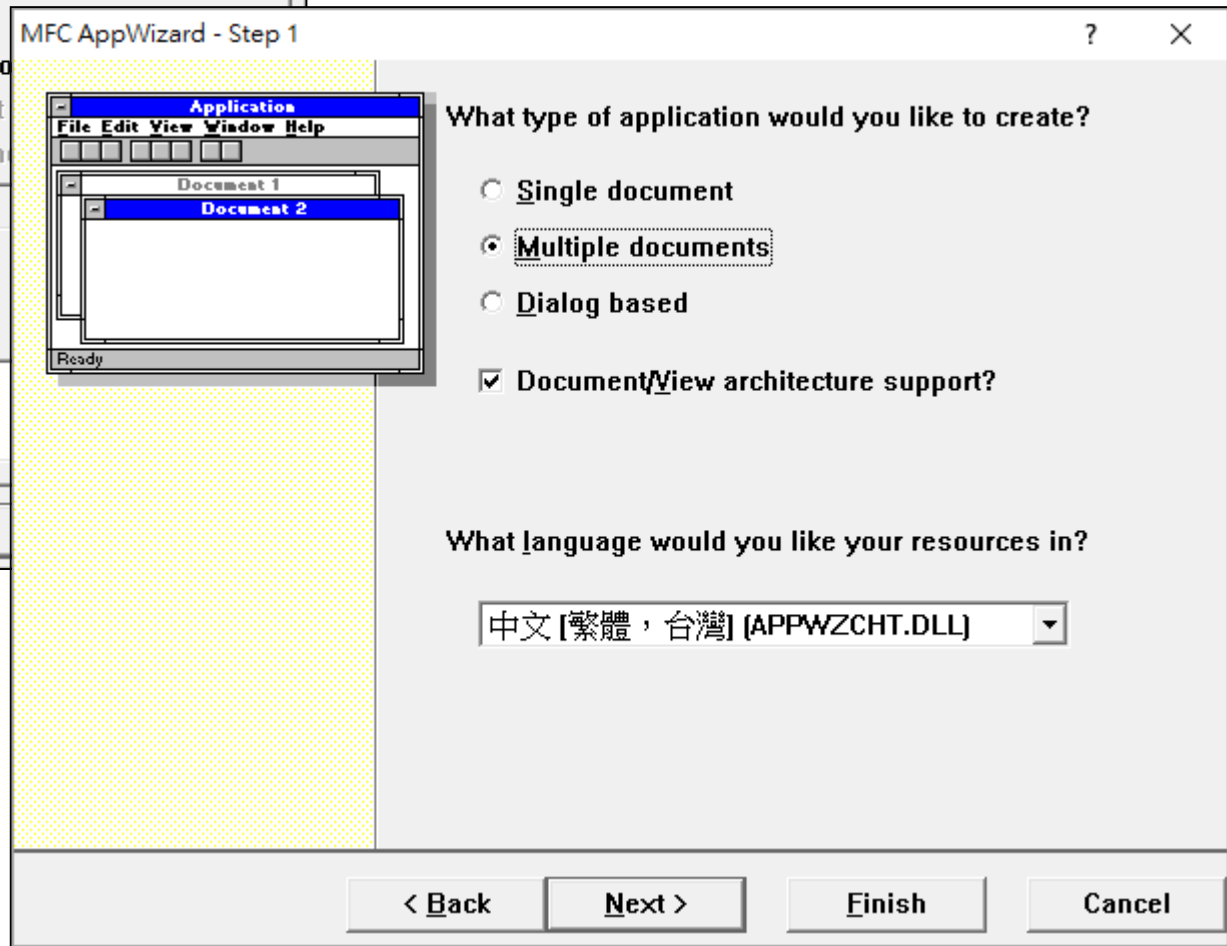
```
1   #include <iostream>
2   using namespace std;
3
4   int* ptr(int atr) {
5       int* pt = nullptr;
6       pt = &atr;
7       return pt;
8   }
9
10  int main()
11  {
12      {
13      int* p();
14      int temp = 3;
15      cout << ptr(temp) << "\n";   //0x22fe90
16      cout << p << "\n";           //1
17      }
18      {
19      int* p(nullptr);
20      cout << p << "\n";  //0
21      }
22      {
23      int temp = 3;
24      int* p(&temp);
25      cout << &temp << "\n";  //0x22fea0
26      cout << p << "\n";      //0x22fea0
27      }
28      return 0;
29  }
```

總結：
看到 ()，要特別謹慎。
它可能表示函數的參數列，如 L4。也可能表示 call operator，如 L15。也可能用來指定初值，如 L13, L19, L24。也可能表示創建一個臨時對象 (temp. object)，例如這麼寫：int(7); 或 float(3.14);

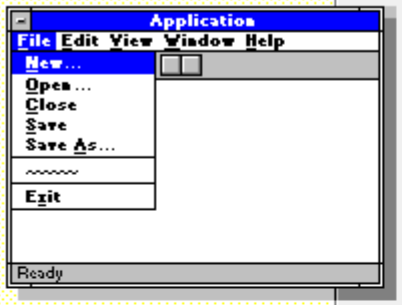
MFC Appwizard (exe)




New
Files
Projects
Workspaces
Other Documents
ATL COM AppWizard
Cluster Resource Type Wizard
Custom AppWizard
Database Project
DevStudio Add-in Wizard
Extended Stored Proc Wizard
ISAPI Extension Wizard
Makefile
MFC ActiveX ControlWizard
MFC AppWizard (dll)
MFC AppWizard (exe)
New Database Wizard
Utility Project
Win32 Application
Win32 Console Application
Win32 Dynamic-Link Library
Win32 Static Library
Project name:
JJTrack
Location:
\00HANDOUT\DRAWING\JJTrack\
...
Create new wo
Add to current
Dependen
Platforms:
Win32
OK


MFC AppWizard - Step 1
Application
File Edit View Window Help
Document 1
Document 2
Ready
What type of application would you like to create?
Single document
Multiple documents
Dialog based
Document/View architecture support?
What language would you like your resources in?
中文 [繁體，台灣] (APPWZCHT.DLL)
< Back
Next >
Finish
Cancel

MFC AppWizard - Step 2 of 6



What database support would you like to include?

- (•) None
- ( ) Header files only
- ( ) Database view without file support
- ( ) Database view with file support

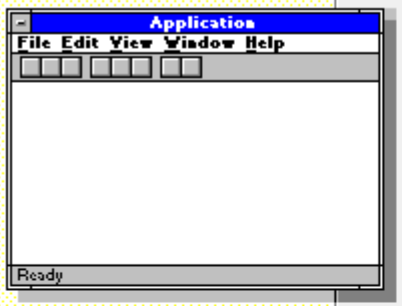
If you include a database view, you must select a data source.

Data Source...

No data source is selected.

< Back | Next > | Finish | Cancel

MFC AppWizard - Step 3 of 6



What compound document support would you like to include?

- (•) None
- ( ) Container
- ( ) Mini-server
- ( ) Full-server
- ( ) Both container and server
  - [ ] Active document server
  - [ ] Active document container

Would you like support for compound files?

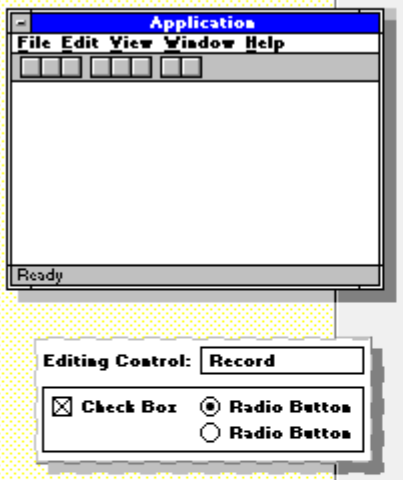
- ( ) Yes, please
- (•) No, thank you

What other support would you like to include?

- [ ] Automation
- [ ] ActiveX Controls

< Back | Next > | Finish | Cancel

MFC AppWizard - Step 4 of 6



What features would you like to include?

- [x] Docking toolbar
- [x] Initial status bar
- [ ] Printing and print preview
- [ ] Context-sensitive Help
- [ ] 3D controls
- [ ] MAPI (Messaging API)
- [ ] Windows Sockets

How do you want your toolbars to look?

- (•) Normal
- ( ) Internet Explorer ReBars

How many files would you like on your recent file list?

4

Advanced...

< Back | Next > | Finish | Cancel

MFC AppWizard - Step 5 of 6

What style of project would you like ?

- (•) MFC Standard
- ( ) Windows Explorer

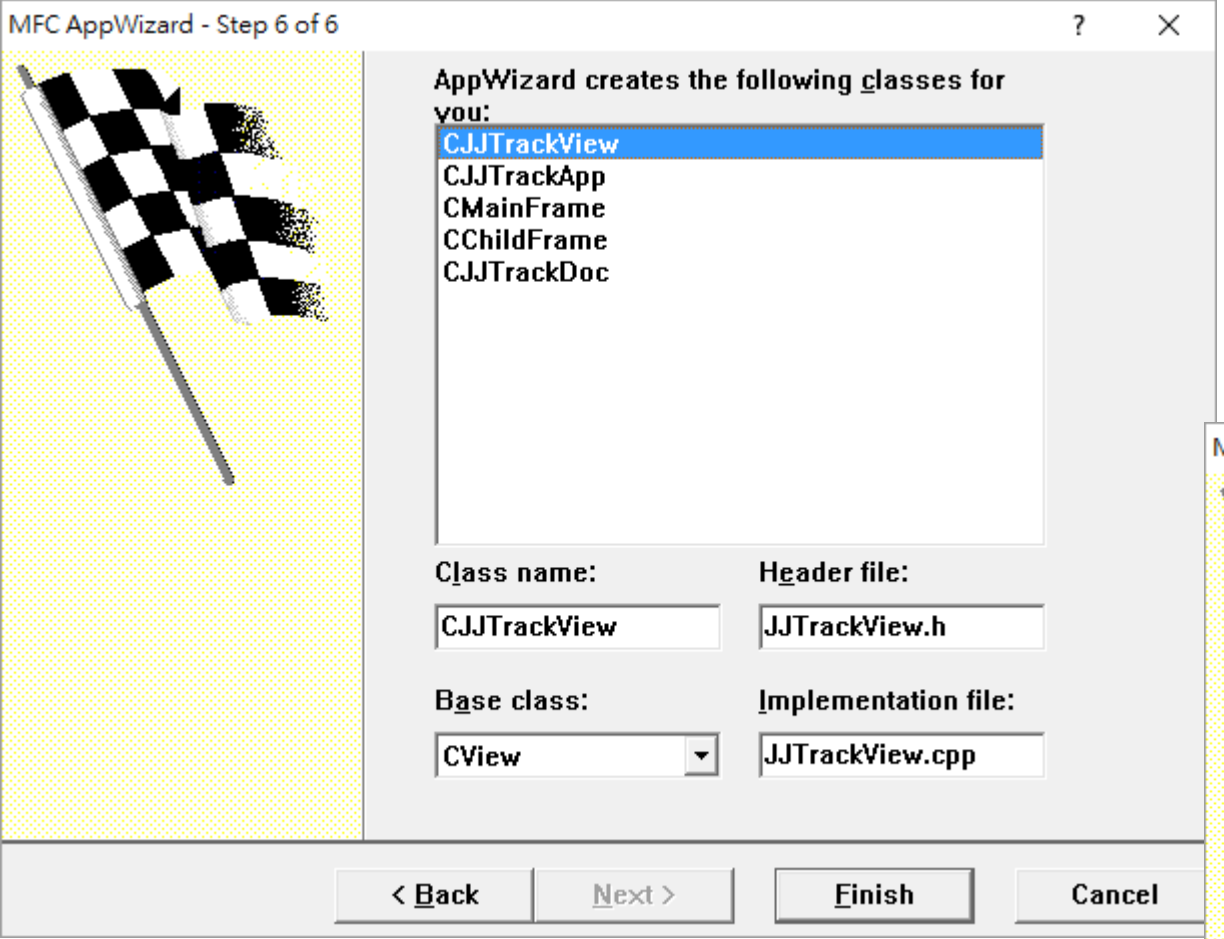
Would you like to generate source file comments?

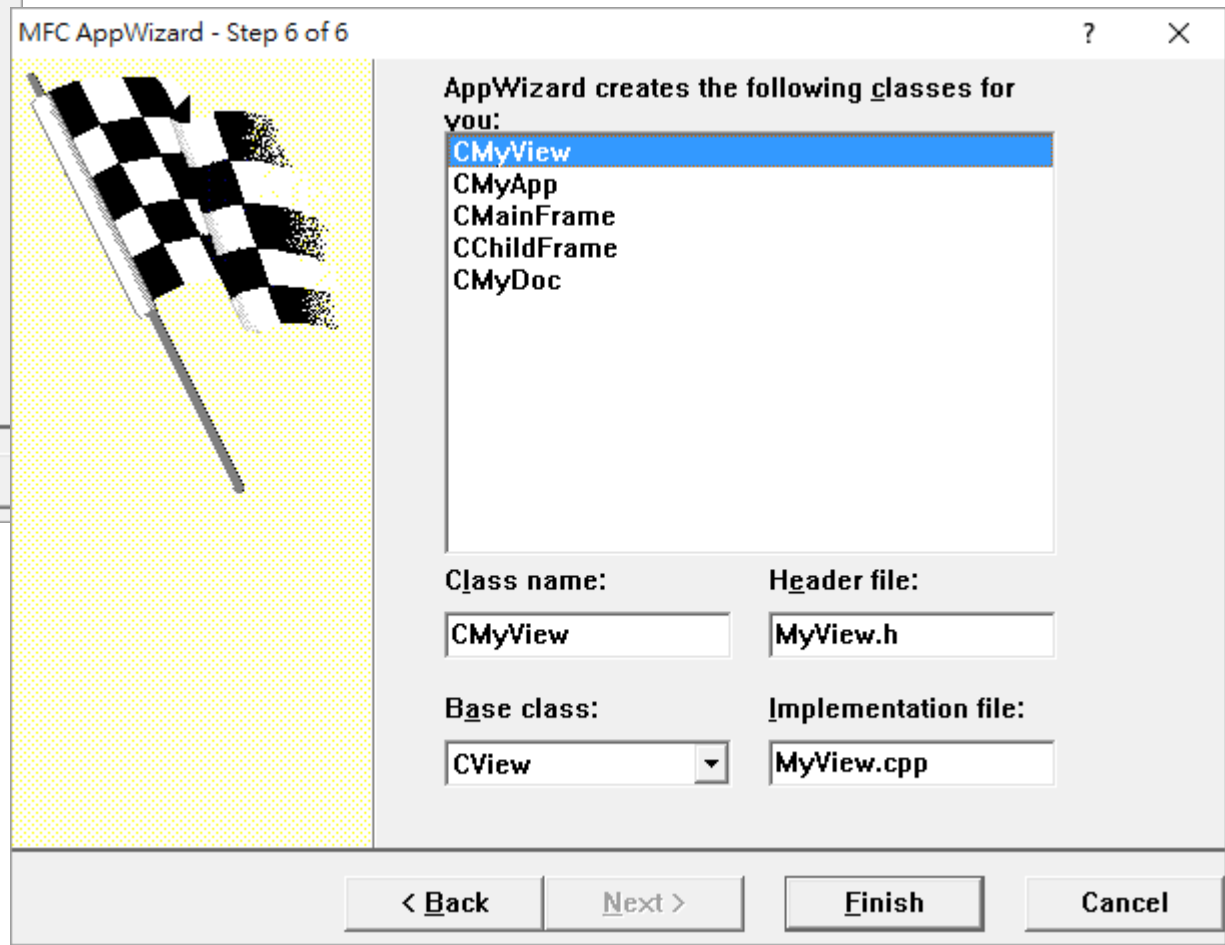
- (•) Yes, please
- ( ) No, thank you

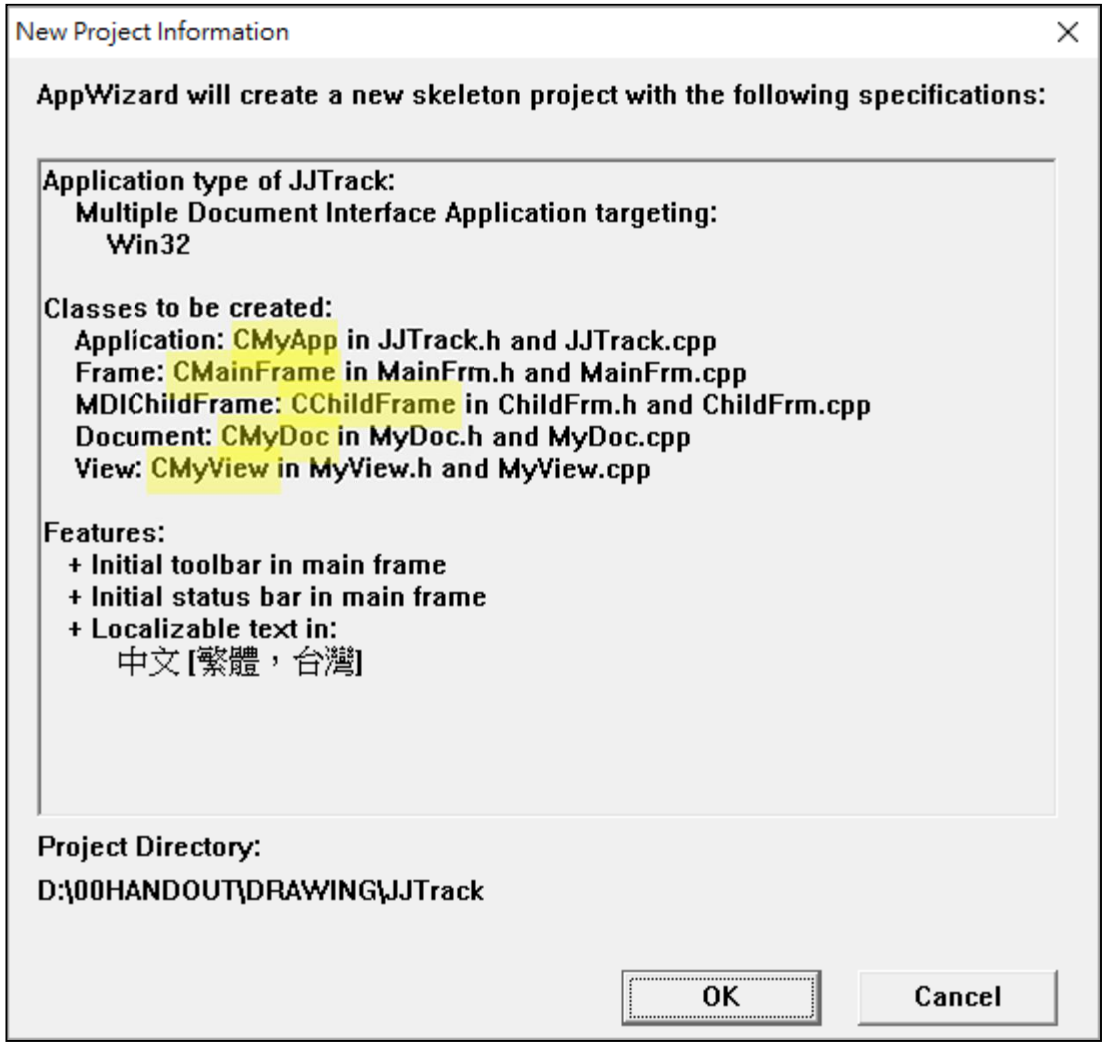
How would you like to use the MFC library?

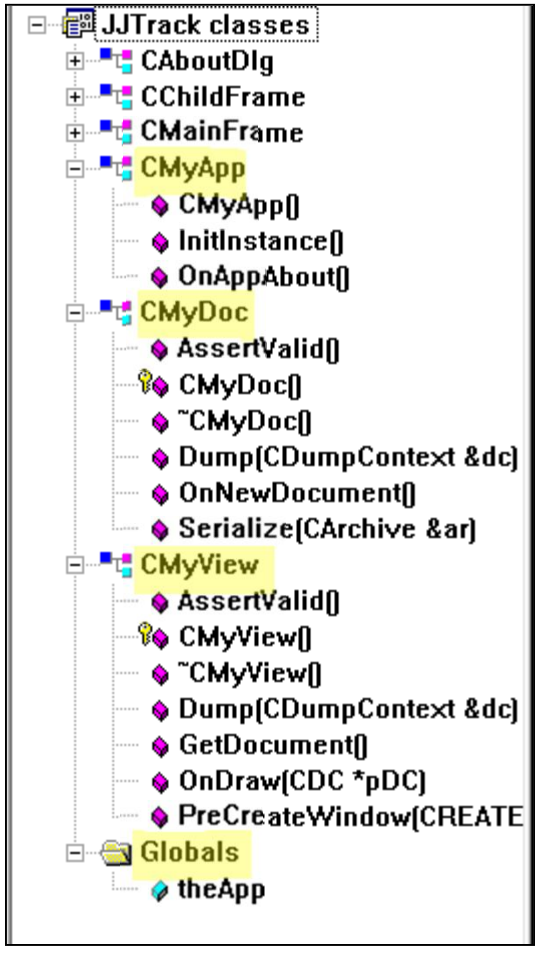
- ( ) As a shared DLL
- (•) As a statically linked library

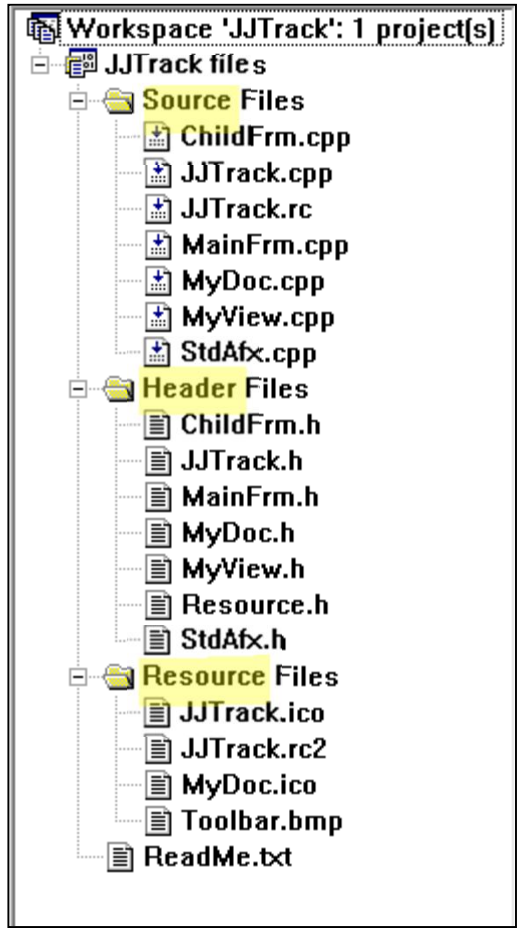
< Back | Next > | Finish | Cancel

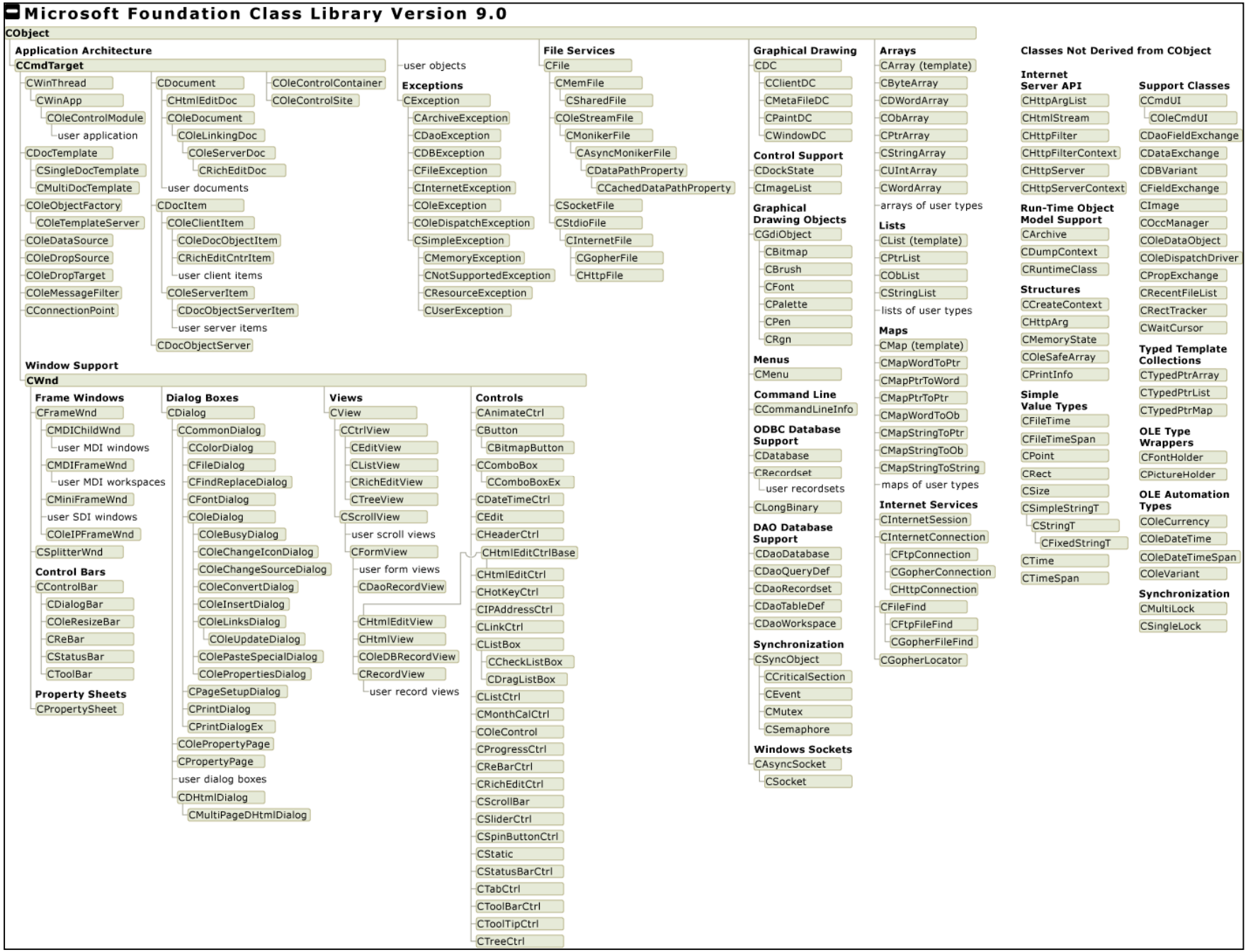

MFC AppWizard - Step 6 of 6
AppWizard creates the following classes for you:
CJJTrackView
CJJTrackApp
CMainFrame
CChildFrame
CJJTrackDoc
Class name:
CJJTrackView
Header file:
JJTrackView.h
Base class:
CView
Implementation file:
JJTrackView.cpp
< Back
Next >
Finish
Cancel

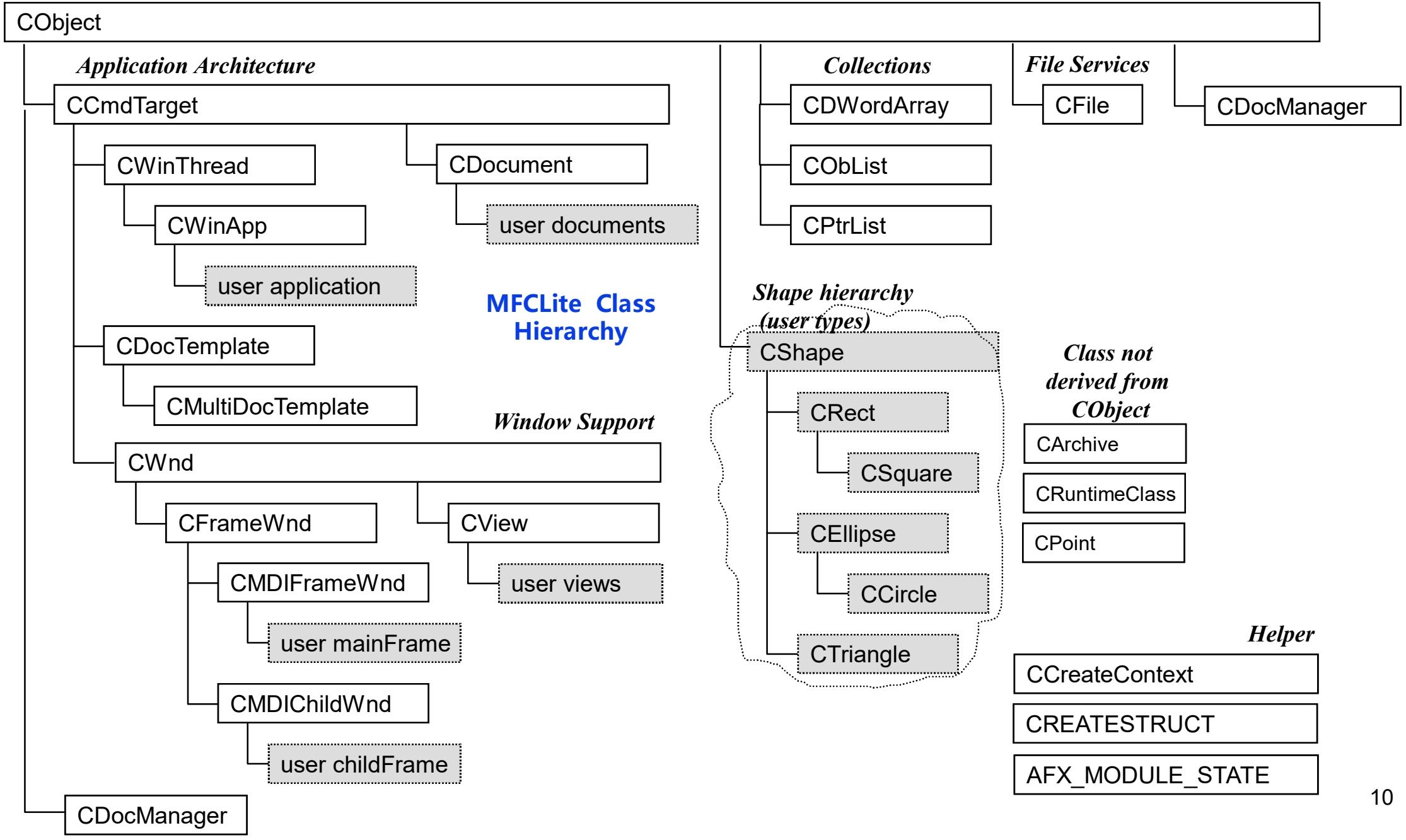

MFC AppWizard - Step 6 of 6
AppWizard creates the following classes for you:
CMyView
CMyApp
CMainFrame
CChildFrame
CMyDoc
Class name:
CMyView
Header file:
MyView.h
Base class:
CView
Implementation file:
MyView.cpp
< Back
Next >
Finish
Cancel


New Project Information
AppWizard will create a new skeleton project with the following specifications:
Application type of JJTrack:
Multiple Document Interface Application targeting:
Win32
Classes to be created:
Application: CMyApp in JJTrack.h and JJTrack.cpp
Frame: CMainFrame in MainFrm.h and MainFrm.cpp
MDIChildFrame: CChildFrame in ChildFrm.h and ChildFrm.cpp
Document: CMyDoc in MyDoc.h and MyDoc.cpp
View: CMyView in MyView.h and MyView.cpp
Features:
+ Initial toolbar in main frame
+ Initial status bar in main frame
+ Localizable text in:
中文 [繁體，台灣]
Project Directory:
D:\00HANDOUT\DRAWING\JJTrack
OK
Cancel


JJTrack classes
CAboutDlg
CChildFrame
CMainFrame
CMyApp
CMyApp()
InitInstance()
OnAppAbout()
CMyDoc
AssertValid()
CMyDoc()
~CMyDoc()
Dump(CDumpContext &dc)
OnNewDocument()
Serialize(CArchive &ar)
CMyView
AssertValid()
CMyView()
~CMyView()
Dump(CDumpContext &dc)
GetDocument()
OnDraw(CDC *pDC)
PreCreateWindow(CREATE
Globals
theApp


Workspace 'JJTrack': 1 project(s)
JJTrack files
Source Files
ChildFrm.cpp
JJTrack.cpp
JJTrack.rc
MainFrm.cpp
MyDoc.cpp
MyView.cpp
StdAfx.cpp
Header Files
ChildFrm.h
JJTrack.h
MainFrm.h
MyDoc.h
MyView.h
Resource.h
StdAfx.h
Resource Files
JJTrack.ico
JJTrack.rc2
MyDoc.ico
Toolbar.bmp
ReadMe.txt


Microsoft Foundation Class Library Version 9.0
CObject
Application Architecture
CCmdTarget
CWinThread
CWinApp
COleControlModule
user application
CDocTemplate
CSingleDocTemplate
CMultiDocTemplate
COleObjectFactory
COleTemplateServer
COleDataSource
COleDropSource
COleDropTarget
COleMessageFilter
CConnectionPoint
CDocument
CHtmlEditDoc
COleDocument
COleLinkingDoc
COleServerDoc
CRichEditDoc
user documents
CDocItem
COleClientItem
COleDocObjectItem
CRichEditCntrItem
user client items
COleServerItem
CDocObjectServerItem
user server items
CDocObjectServer
COleControlContainer
COleControlSite
user objects
Exceptions
CException
CArchiveException
CDaoException
CDBException
CFileException
CInternetException
COleException
COleDispatchException
CSimpleException
CMemoryException
CNotSupportedException
CResourceException
CUserException
File Services
CFile
CMemFile
CSharedFile
COleStreamFile
CMonikerFile
CAsyncMonikerFile
CDataPathProperty
CCachedDataPathProperty
CSocketFile
CStdioFile
CInternetFile
CGopherFile
CHttpFile
Graphical Drawing
CDC
CClientDC
CMetaFileDC
CPaintDC
CWindowDC
Control Support
CDockState
CImageList
Graphical Drawing Objects
CGdiObject
CBitmap
CBrush
CFont
CPalette
CPen
CRgn
Menus
CMenu
Command Line
CCommandLineInfo
ODBC Database Support
CDatabase
CRecordset
user recordsets
CLongBinary
DAO Database Support
CDaoDatabase
CDaoQueryDef
CDaoRecordset
CDaoTableDef
CDaoWorkspace
Synchronization
CSyncObject
CCriticalSection
CEvent
CMutex
CSemaphore
Windows Sockets
CAsyncSocket
CSocket
Arrays
CArray (template)
CByteArray
CDWordArray
CObArray
CPtrArray
CStringArray
CUIntArray
CWordArray
arrays of user types
Lists
CList (template)
CPtrList
CObList
CStringList
lists of user types
Maps
CMap (template)
CMapWordToPtr
CMapPtrToWord
CMapPtrToPtr
CMapWordToOb
CMapStringToPtr
CMapStringToOb
CMapStringToString
maps of user types
Internet Services
CInternetSession
CInternetConnection
CFtpConnection
CGopherConnection
CHttpConnection
CFileFind
CFtpFileFind
CGopherFileFind
CGopherLocator
Classes Not Derived from CObject
Internet Server API
CHttpArgList
CHtmlStream
CHttpFilter
CHttpFilterContext
CHttpServer
CHttpServerContext
Run-Time Object Model Support
CArchive
CDumpContext
CRuntimeClass
Structures
CCreateContext
CHttpArg
CMemoryState
COleSafeArray
CPrintInfo
Simple Value Types
CFileTime
CFileTimeSpan
CPoint
CRect
CSize
CSimpleStringT
CStringT
CFixedStringT
CTime
CTimeSpan
Support Classes
CCmdUI
COleCmdUI
CDaoFieldExchange
CDataExchange
CDBVariant
CFieldExchange
CImage
COccManager
COleDataObject
COleDispatchDriver
CPropExchange
CRecentFileList
CRectTracker
CWaitCursor
Typed Template Collections
CTypedPtrArray
CTypedPtrList
CTypedPtrMap
OLE Type Wrappers
CFontHolder
CPictureHolder
OLE Automation Types
COleCurrency
COleDateTime
COleDateTimeSpan
COleVariant
Synchronization
CMultiLock
CSingleLock
Window Support
CWnd
Frame Windows
CFrameWnd
CMDIChildWnd
user MDI windows
CMDIFrameWnd
user MDI workspaces
CMiniFrameWnd
user SDI windows
COleIPFrameWnd
CSplitterWnd
Control Bars
CControlBar
CDialogBar
COleResizeBar
CReBar
CStatusBar
CToolBar
Property Sheets
CPropertySheet
Dialog Boxes
CDialog
CCommonDialog
CColorDialog
CFileDialog
CFindReplaceDialog
CFontDialog
COleDialog
COleBusyDialog
COleChangeIconDialog
COleChangeSourceDialog
COleConvertDialog
COleInsertDialog
COleLinksDialog
COleUpdateDialog
COlePasteSpecialDialog
COlePropertiesDialog
CPageSetupDialog
CPrintDialog
CPrintDialogEx
COlePropertyPage
CPropertyPage
user dialog boxes
CDHtmlDialog
CMultiPageDHtmlDialog
Views
CView
CCtrlView
CEditView
CListView
CRichEditView
CTreeView
CScrollView
user scroll views
CFormView
user form views
CDaoRecordView
CHtmlEditView
CHtmlView
COleDBRecordView
CRecordView
user record views
Controls
CAnimateCtrl
CButton
CBitmapButton
CComboBox
CComboBoxEx
CDateTimeCtrl
CEdit
CHeaderCtrl
CHtmlEditCtrlBase
CHtmlEditCtrl
CHotKeyCtrl
CIPAddressCtrl
CLinkCtrl
CListBox
CCheckListBox
CDragListBox
CListCtrl
CMonthCalCtrl
COleControl
CProgressCtrl
CReBarCtrl
CRichEditCtrl
CScrollBar
CSliderCtrl
CSpinButtonCtrl
CStatic
CStatusBarCtrl
CTabCtrl
CToolBarCtrl
CToolTipCtrl
CTreeCtrl

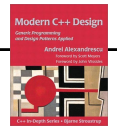

CObject
Application Architecture
CCmdTarget
CWinThread
CWinApp
user application
CDocument
user documents
CDocTemplate
CMultiDocTemplate
Window Support
CWnd
CFrameWnd
CMDIFrameWnd
user mainFrame
CMDIChildWnd
user childFrame
CView
user views
CDocManager
MFCLite Class Hierarchy
Collections
CDWordArray
CObList
CPtrList
File Services
CFile
CDocManager
Shape hierarchy (user types)
CShape
CRect
CSquare
CEllipse
CCircle
CTriangle
Class not derived from CObject
CArchive
CRuntimeClass
CPoint
Helper
CCreateContext
CREATESTRUCT
AFX_MODULE_STATE

Singleton

clients 如何取用：

Singleton::Instance()

也有設計是提供兩個 global points to access of it :
get_const_instance() 和 get_mutable_instance()



```
class Singleton {
public:
        static Singleton& Instance()  { // Unique point of access
                if (!pInstance_)
                    pInstance_ = new Singleton;
                return *pInstance_;
        }
        ... operations ...
private: //謝絕以下動作
        Singleton();
        Singleton(const Singleton&);
        ~Singleton();
        Singleton& operator=(const Singleton&);

private:
        static Singleton* pInstance_; // The one and only instance
};

// Implementation file Singleton.cpp
Singleton* Singleton::pInstance_ = 0;
```

或者不使用 private 而讓它們繼承 uncopyable. 但這些措施對於 de-serialize 能形成阻擋嗎？應該可以吧，因為 de-serialize 也需要創建 object.

Meyers Singleton

```
class Printer {
public:
  static Printer& Instance();
  ...
private:
  Printer();
  Printer(const Printer& rhs);
  ...
};

Printer& Printer::Instance()
{
  static Printer p;
  return p;
}
```

clients 如何取用：

Printer::Instance()

每當即將生成一個 object，一定有個 ctor 被喚起。「阻止創建 object」的最簡單作法就是將其 ctors 宣告為 private。

(惟一)全域訪問點

此物在函數第一次被調用時才產生。
function static 的初始化確切時機：該函數第一次被調用時之 static 定義處。

std::alloc_mt 中有多線程下的 Singleton 作法, 見 G4.8-源碼-allocator-mt.ppt
其中用到 C++11 的 std::call_once(), std::once_flag.

以一個簡單例子展示(上頁所說的)問題。

D:\DPInJava\20-Flyweight\SampleC++-static-state-in-factory\Main.cpp

non-static

```cpp
class FontDataFactory
{
private:
    std::map<char, FontData*> _pool;
    static FontDataFactory _singleton;
    FontDataFactory() { cout << "FontDat

public:
    ~FontDataFactory() {
        cout << "FontDataFactory Dtor \n
    }

static FontDataFactory& getInstance() {
        return _singleton;
    }

public:
    FontData* getFontData(char ch) {
        FontData* bc;
        auto iter = _pool.find(ch);
        if (iter == _pool.end()) {  //沒找著
            bc = new FontData(ch);  //創建一個
            _pool[ch] = bc;         //放進 map
        }
        else {  //找著了
            cout << ch << " found! using FlyWeight. \n";
            bc = (*iter).second; //使用既有的
        }
        return bc;
    }

    int pooSize() {                    兩個函數，輔助觀察用。
        return _pool.size();
    }
    void dumpPool() {
        for (auto& x : _pool)
            cout << "[" << x.first << "," << x.second << "] \n";
    }
};
FontDataFactory FontDataFactory::_singleton;    //會引發 FontDataFactory Ctor
```

```cpp
//20210915
#include <vector>
#include <map>
#include <string>
#include <iostream>
using std::cout;
using std::cin;

class FontData
{
private:
    char _ch;   //ASCII code for key

public:
    FontData(char ch) : _ch(ch) {}
};
```

與 Java 版本比較

```java
import java.util.Hashtable;
public class BigCharFactory {
    // 管理現有的BigChar的物件個體
    private Hashtable pool = new Hashtable();
    //侯捷：可改為以下兩種寫法（視後頭如何使用而定）：
    //private Hashtable<String, BigChar> pool = new Hashtable<Strin
    //private Hashtable<Character, BigChar> pool = new Hashtable<Ch

    // Singleton Pattern
    private static BigCharFactory singleton = new BigCharFactory();
    // 建構子
    private BigCharFactory() {
    }
    // 取得唯一的物件個體
    public static BigCharFactory getInstance() {
        return singleton;
    }
    // 產生（共用）BigChar的物件個體
    public synchronized BigChar getBigChar(char charname) {
        BigChar bc = (BigChar)pool.get("" + charname);
        //BigChar bc = (BigChar)pool.get(new Character(charname));
        if (bc == null) {
            bc = new BigChar(charname);      // 在此產生BigChar的物件
            pool.put("" + charname, bc);
            //pool.put(new Character(charname), bc);  //侯捷：這樣也
        }
        return bc;
    }
}
```

以一個簡單例子展示(上頁所說的)問題。

```
61  class BigString
62  {
63  private:
64      std::vector<FontData*> _bigchars;
65  public:
66      BigString(std::string str) {
67          FontDataFactory factory = FontDataFactory::getInstance();
68          //↑ 此行移至 private data member 也ok.
69
70          _bigchars.reserve(str.length()); //既然已知長度，
71          _bigchars.resize(str.length());  //就以這種方式預先持有空間。
72
73          cout << "factory.poolSize()= " << factory.pooSize() << "\n";
74          for (int i = 0; i < str.length(); i++) {
75              _bigchars[i] = factory.getFontData(str[i]); //向 factory "申請" 一個 FontData.
76          }
77          cout << "factory.poolSize()= " << factory.pooSize() << "\n";
78          factory.dumpPool();
79      }
80  };
```

L67 取得的 factory，是 FontDataFactory 的 copy ctor 被喚起後所得(本例並未撰寫該 copy ctor，因此用的是 default copy ctor)；該 ctor 的 right hand side 是 getInstance() 的返回值，那是 a reference to FontDataFactory (L30)，該返回值被 copy 至 local variable factory (L67)。至此，變量 factory 的意義就是：它 "代表" L31 _singleton。

由於 factory 是個 local variable，因此當 L79 函數結束，factory 的 dtor 會被喚起。而由於 factory "就是 / 就等同於" _singleton，後者內含 _pool (L21)，因此 _pool 的 dtor 會被喚起 (C++ 語言機制使然)，導致 _pool 清空。也因此每次進入 L66 都是一個全新的 _pool。這不是我們要的。

若要阻止 factory 的 dtor 喚起 _pool 的 dtor，惟一辦法是讓 _pool 成為 **static**。

與 Java 版本比較

```
1   public class BigString {
2       // 「大型文字」的陣列
3       private BigChar[] bigchars;
4       // 建構子
5       public BigString(String string) {
6           bigchars = new BigChar[string.length()];
7           BigCharFactory factory = BigCharFactory.getInstance();
8           for (int i = 0; i < bigchars.length; i++) {
9               bigchars[i] = factory.getBigChar(string.charAt(i));
10          }
11      }
12      // 顯示
13      public void print() {
14          for (int i = 0; i < bigchars.length; i++) {
15              bigchars[i].print();
16          }
17      }
18  }
```

```
82  int main()
83  {
84  std::string str;
85  BigString* bs[4];
86
87      cout << "main()---------- \n";
88      for(int i=0; i<3; ++i) {
89          cout << "input #" << i << " string (using digits 0~9): ";
90          cin >> str;
91          bs[i] = new BigString(str);
92      }
93      cout << "main() exit------ \n";
94  }
```

←3次取得 factory。每次通過 factory 取得 flyweights。

```
D:\DPInJava\20-Flyweight\SampleC++-static-state...
FontDataFactory Ctor
main()----------
input #0 string (using digits 0~9): 13463
factory.poolSize()= 0
3 found! using FlyWeight.
factory.poolSize()= 4
[1,0x5c6400]
[3,0x5c6410]
[4,0x5c6420]
[6,0x5c6430]
FontDataFactory Dtor
input #1 string (using digits 0~9): 4638
factory.poolSize()= 0
factory.poolSize()= 4
[3,0x5c6460]
[4,0x5c6440]
[6,0x5c6450]
[8,0x5c6470]
FontDataFactory Dtor
input #2 string (using digits 0~9): 15363
factory.poolSize()= 0
3 found! using FlyWeight.
factory.poolSize()= 4
[1,0x5c6480]
[3,0x5c64a0]
[5,0x5c6490]
[6,0x5c64b0]
FontDataFactory Dtor
main() exit------
FontDataFactory Dtor
```

永遠無法從 factory 中取得上次 string 所儲存的既有的 flyweights.(這太糟了) → 原因見上頁左下說明。

試著合理解釋，為什麼 ctor 執行1次 而 dtor 執行4次

Q : 右圖 FontDataFactory 的 ctor 被呼叫一次，dtor 被呼叫四次，為什麼沒有帶來壞影響?

A : 該 dtor 前三次被呼叫都是因為 L67 的 factory 離開 scope，不是因為某個東西被 delete 而被喚起。也就是說那東西 (a FontDataFactory object，i.e. factory, i.e. _singleton) 依然存在。這時候把 dtor 視為一般函數是 ok 的，不必認為它是哪個東西被 deleted 而導致。由於例中這個 dtor (L26) 並沒有做任何會招致損壞的動作，所以沒有影響那生命一直存在的 _singleton。

修改兩個地方，就正確了：

```
18  class FontDataFactory
19  {
20  private:
21      static std::map<char, FontData*> _pool;
22      static FontDataFactory _singleton;
```

```
56  };
57  FontDataFactory FontDataFactory::_singleton;    //會引發 FontDataFactory Ctor
58  std::map<char, FontData*> FontDataFactory::_pool;   //會引發 std::map Ctor
59
```

```
82  int main()
83  {
84  std::string str;
85  BigString* bs[4];
86
87      cout << "main()---------- \n";
88      for(int i=0; i<3; ++i) {
89          cout << "input #" << i << " string (using digits 0~9): ";
90          cin >> str;
91          bs[i] = new BigString(str);
92      }
93      cout << "main() exit------ \n";
94  }
```

←3次取得 factory。
每次通過 factory
取得 flyweights。

```
D:\DPInJava\20-Flyweight\SampleC++-static-state-in...
FontDataFactory Ctor
main()----------
input #0 string (using digits 0~9): 13463
factory.poolSize()= 0
3 found! using FlyWeight.
factory.poolSize()= 4
[1,0x696400]
[3,0x696410]
[4,0x696420]
[6,0x696430]
FontDataFactory Dtor
input #1 string (using digits 0~9): 4638
factory.poolSize()= 4
4 found! using FlyWeight.
6 found! using FlyWeight.
3 found! using FlyWeight.
factory.poolSize()= 5
[1,0x696400]
[3,0x696410]
[4,0x696420]
[6,0x696430]
[8,0x696440]
FontDataFactory Dtor
input #2 string (using digits 0~9): 15363
factory.poolSize()= 5
1 found! using FlyWeight.
3 found! using FlyWeight.
6 found! using FlyWeight.
3 found! using FlyWeight.
factory.poolSize()= 6
[1,0x696400]
[3,0x696410]
[4,0x696420]
[5,0x696450]
[6,0x696430]
[8,0x696440]
FontDataFactory Dtor
main() exit------
FontDataFactory Dtor

--------------------------------
```

試著合理解釋，為什麼
ctor 執行1次 而
dtor 執行4次

或是這麼修改，也正確：

```
18  class FontDataFactory
19  {
20  private:
21      std::map<char, FontData*> _pool;   ← non-static
22      static FontDataFactory _singleton;
23      FontDataFactory() { cout << "FontDataFactory Ctor \n"; }
24      FontDataFactory(const FontDataFactory&);   ← 不允許被 copy construction
```

```
61  class BigString
62  {
63  private:
64      std::vector<FontData*> _bigchars;
65  public:
66      BigString(std::string str) {
67
68
69
70          _bigchars.reserve(str.length()); //既然已知長度，
71          _bigchars.resize(str.length());  //就以這種方式預先持有空間。
72
73          cout << "factory.poolSize()= " << FontDataFactory::getInstance().pooSize() << "\n";
74          for (int i = 0; i < str.length(); i++) {
75              _bigchars[i] = FontDataFactory::getInstance().getFontData(str[i]);  //向 factory "申請" 一個 Font
76          }
77          cout << "factory.poolSize()= " << FontDataFactory::getInstance().pooSize() << "\n";
78          FontDataFactory::getInstance().dumpPool();
79      }
80  };
```

```
65  public:
66      BigString(std::string str) {
67          FontDataFactory factory = FontDataFactory::getInstance();
```

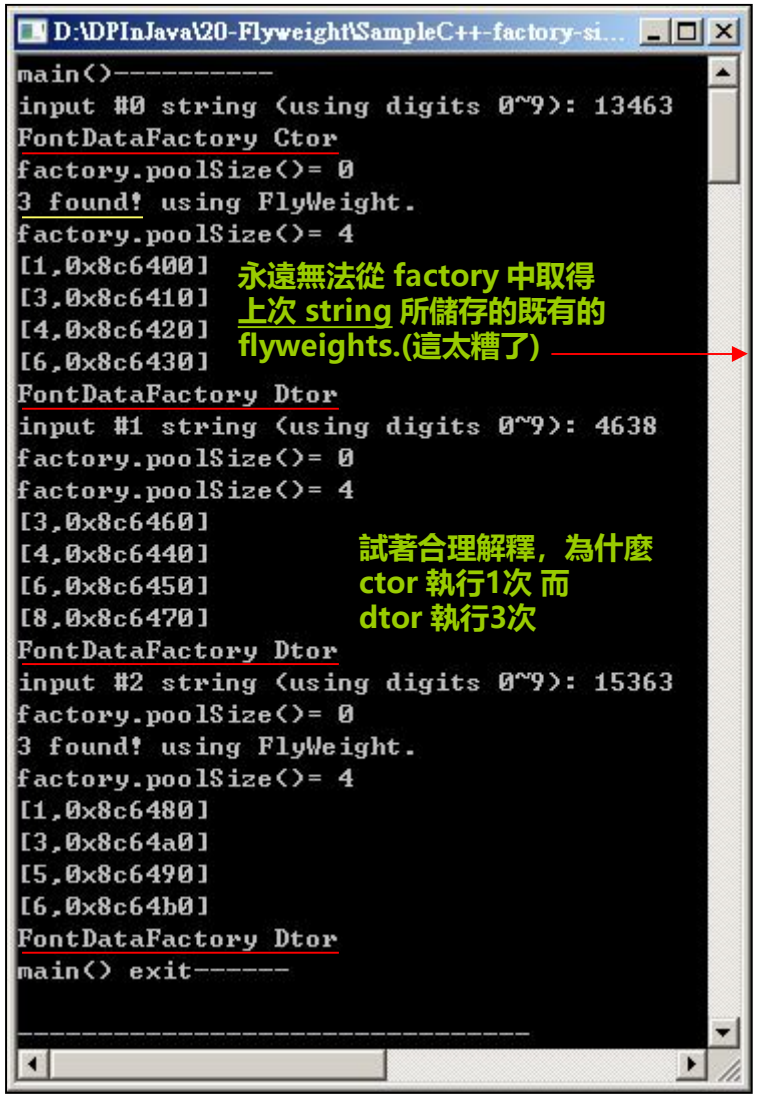
先前這麼寫，在此版本行不通，因為

```
D:\DPInJava\20-Flyweight\SampleC++-static-state...
FontDataFactory Ctor
main()----------
input #0 string (using digits 0~9): 13463
factory.poolSize()= 0
3 found! using FlyWeight.
factory.poolSize()= 4
[1,0x726400]
[3,0x726410]
[4,0x726420]
[6,0x726430]
input #1 string (using digits 0~9): 4638
factory.poolSize()= 4
4 found! using FlyWeight.
6 found! using FlyWeight.
3 found! using FlyWeight.
factory.poolSize()= 5
[1,0x726400]
[3,0x726410]
[4,0x726420]
[6,0x726430]
[8,0x726440]
input #2 string (using digits 0~9): 15363
factory.poolSize()= 5
1 found! using FlyWeight.
3 found! using FlyWeight.
6 found! using FlyWeight.
3 found! using FlyWeight.
factory.poolSize()= 6
[1,0x726400]
[3,0x726410]
[4,0x726420]
[5,0x726450]
[6,0x726430]
[8,0x726440]
main() exit------
FontDataFactory Dtor
```

這個情況是 最好/最容易解釋的情況。

若這麼修改，會如何?

```
18  class FontDataFactory
19  {
20  private:
21      std::map<char, FontData*> _pool;   ← non-static
22      static FontDataFactory* _pSingleton;
23      FontDataFactory() { cout << "FontDataFactory Ctor \n"; }
24  //  FontDataFactory(const FontDataFactory&);
25  //  FontDataFactory& operator=(const FontDataFactory&);
26  public:
27      ~FontDataFactory() {
28          cout << "FontDataFactory Dtor \n";
29      }
30  static FontDataFactory& getInstance() {
31          if (!_pSingleton) _pSingleton = new FontDataFactory;
32          return *_pSingleton;
33      }
34  public:
35      FontData* getFontData(char ch) {
36          FontData* bc;
37          auto iter = _pool.find(ch);
38          if (iter == _pool.end()) {  //沒找著
39              bc = new FontData(ch);  //創建一個
40              _pool[ch] = bc;         //放進 map
41          }
42          else {  //找著了
43              cout << ch << " found! using FlyWeight. \n";
44              bc = (*iter).second; //使用既有的
45          }
46          return bc;
47      }
48
49      int pooSize() {
50          return _pool.size();
51      }
52      void dumpPool() {
53          for (auto& x : _pool)
54              cout << "[" << x.first << "," << x.second << "] \n";
55      }
56  };
57  FontDataFactory* FontDataFactory::_pSingleton = nullptr;
```



依然有此潛在問題。

所以，問題的癥結不在於 singleton 是以 static object 完成 (本頁之前) 或是以 static pointer (本頁L22) 完成，癥結是 client 是否總是以 getInstance() 取得 singleton 本體，抑或 client 是把 getInstance() 所得的 referene to singleton 儲存起來使用。

若這麼修改，又會如何?

```
18  class FontDataFactory
19  {
20  private:
21      std::map<char, FontData*> _pool;
22      static FontDataFactory* _pSingleton;
23      FontDataFactory() { cout << "FontDataFactory Ctor \n"; }
24      FontDataFactory(const FontDataFactory&);
25      FontDataFactory& operator=(const FontDataFactory&);
26  public:
27      ~FontDataFactory() {
28          cout << "FontDataFactory Dtor \n";
29      }
30  static FontDataFactory& getInstance() {
```

← non-static

← 不允許被 copy construction

(以下同前)

```
60  class BigString
61  {
62  private:
63      std::vector<FontData*> _bigchars;
64  public:
65      BigString(std::string str) {
66  //      FontDataFactory factory = FontDataFactory::getInstance();
67
68          _bigchars.reserve(str.length()); //既然已知長度，
69          _bigchars.resize(str.length());  //就以這種方式預先持有空間。
70
71          cout << "factory.poolSize()= " << FontDataFactory::getInstance().pooSize() << "\n";
72          for (int i = 0; i < str.length(); i++) {
73              _bigchars[i] = FontDataFactory::getInstance().getFontData(str[i]);
74          }
75          cout << "factory.poolSize()= " << FontDataFactory::getInstance().pooSize() << "\n";
76          FontDataFactory::getInstance().dumpPool();
77      }
78  };
```

↓先前這麼寫，在此版本**行不通，因為**

```
D:\DPInJava\20-Flyweight\SampleC++-factory-sin...
main()----------
input #0 string (using digits 0~9): 13463
FontDataFactory Ctor
factory.poolSize()= 0
3 found! using FlyWeight.
factory.poolSize()= 4
[1,0x9a6400]
[3,0x9a6410]
[4,0x9a6420]
[6,0x9a6430]
input #1 string (using digits 0~9): 4638
factory.poolSize()= 4
4 found! using FlyWeight.
6 found! using FlyWeight.
3 found! using FlyWeight.
factory.poolSize()= 5
[1,0x9a6400]
[3,0x9a6410]
[4,0x9a6420]
[6,0x9a6430]
[8,0x9a6440]
input #2 string (using digits 0~9): 15363
factory.poolSize()= 5
1 found! using FlyWeight.
3 found! using FlyWeight.
6 found! using FlyWeight.
3 found! using FlyWeight.
factory.poolSize()= 6
[1,0x9a6400]
[3,0x9a6410]
[4,0x9a6420]
[5,0x9a6450]
[6,0x9a6430]
[8,0x9a6440]
main() exit------
```

這個情況是 最好/最容易解釋的情況。

2021年9月29日 21:52
老师好
下课后我复制了一份代码调试，您在课上
说 **FontDataFactory factory = FontDataFactory::getInstance();** 中**factory**是一个引用，我认为有些问题。在此赋值语句的右边调用的函数的确返回了一个引用，即 **FontDataFactory&** 类型，而 **factory** 变量我们一开始就声明为了**FontDataFactory**类的一个普通的实例，因此这里调用了**FontDataFactory::FontDatafactory(FontDataFactory&)** 这一隐性给出的构造函数，将右边得到的引用（左值）当作右值使用赋值给了**factory**，而我们并没有为这一构造函数写过调用它时要打印东西的代码，因此自然不能得知创建了新的**FontDataFactory**类型的变量，在该变量离开作用域时也自然会调用析构函数将新变量的所有数据清除。

← **錯誤描述** (見下頁 L75 注釋)

在您给出的解决方案中，讲义的**16**页和**18**页中只调用了一次析构函数的代码都将别处涉及到**singleton**的代码改为了**FontDataFactory::getInstance()**，可我如果觉得每次都要这样写太长，或者每次都要调用函数可能会造成多余开销该怎么办呢？我试了试将该句改为**FontDataFactory& factory = FontDataFactory::getInstance();** 将**factory**声明为引用，这样也就不会创建新的变量。即使不将**map<char,FontData*> _pool**声明为**static**，发现程序也能正常运行。望老师验证一下我的想法是否正确。谢谢！

← **錯誤描述** (inline function 即可避免 overhead)

← **非常好, 見下頁**

若這麼修改，又會如何?

D:\DPInJava\20-Flyweight\SampleC++-non-static-map-with-private-copy-ctor-and-reference-factory
它和以下(先前)版本的唯一區別在於 L74 是個 reference
D:\DPInJava\20-Flyweight\SampleC++-non-static-map-with-private-copy-ctor

```
25  class FontDataFactory
26  {
27  private:
28      std::map<char, FontData*> _pool;
29      static FontDataFactory _singleton;
30      FontDataFactory() { cout << "FontDataFactory Ctor \n"; }
31      FontDataFactory(const FontDataFactory&); //copy ctor
32      FontDataFactory& operator=(const FontDataFactory&); //copy assignment
33  public:
```

↓盡所有力氣阻止 "外界" 生成/創建 FontDataFactory
↓(因為我們要它是個 single)

```
68  class BigString
69  {
70  private:
71      std::vector<FontData*> _bigchars;
72  public:
73      BigString(std::string str) {
74      FontDataFactory& factory = FontDataFactory::getInstance();
75      //注意注意：以上不會引發 copy ctor. 它就像 ptr assignment 一樣並非 class level.
76
77          _bigchars.reserve(str.length()); //既然已知長度，
78          _bigchars.resize(str.length());  //就以這種方式預先持有空間?
79
80          cout << "factory.poolSize()= " << factory.pooSize() << "\n";
81          for (int i = 0; i < str.length(); i++) {
82              _bigchars[i] = factory.getFontData(str[i]); //向 factory "申請" 一個 FontData.
83          }
84          cout << "factory.poolSize()= " << factory.pooSize() << "\n";
85          factory.dumpPool();
86      }
87  };
```

最終結論：如果要將 singleton 取出來用，
像 L74 (使用 referene) 那麼做，會是最好最省心的作法。

```
D:\DPInJava\20-Flyweight\SampleC++-non-static-m...
FontDataFactory Ctor
main()----------
input #0 string (using digits 0~9): 13463
factory.poolSize()= 0
3 found! using FlyWeight.
factory.poolSize()= 4
[1,0x326400]
[3,0x326410]
[4,0x326420]
[6,0x326430]
input #1 string (using digits 0~9): 4638
factory.poolSize()= 4
4 found! using FlyWeight.
6 found! using FlyWeight.
3 found! using FlyWeight.
factory.poolSize()= 5
[1,0x326400]
[3,0x326410]
[4,0x326420]
[6,0x326430]
[8,0x326440]
input #2 string (using digits 0~9): 15363
factory.poolSize()= 5
1 found! using FlyWeight.
3 found! using FlyWeight.
6 found! using FlyWeight.
3 found! using FlyWeight.
factory.poolSize()= 6
[1,0x326400]
[3,0x326410]
[4,0x326420]
[5,0x326450]
[6,0x326430]
[8,0x326440]
main() exit------
FontDataFactory Dtor
```

這個情況非常好

new expression

```
Complex* pc = new Complex(1,2);
```

編譯器
轉為

```
Complex *pc;
try {
  1 void* mem  =  operator new( sizeof(Complex) );  //allocate
  2 pc = static_cast<Complex*>(mem);                 //cast
  3 pc->Complex::Complex(1,2);                       //construct
    //注意：只有編譯器才可以像上面那樣直接呼叫 ctor
}
catch( std::bad_alloc ) {
      //若allocation失敗就不執行constructor
}
```

App. 欲直接調用 ctor, 可運用 placement new :

```
new(p) Complex(1,2);
```

```
void *operator new(size_t size, const std::nothrow_t&)
                                _THROW0()
{  // try to allocate size bytes
void *p;
while ((p = malloc(size)) == 0)
 { // buy more memory or return null pointer
   _TRY_BEGIN
       if (_callnewh(size) == 0)  break;
   _CATCH(std::bad_alloc)    return (0);
   _CATCH_END
 }
return (p);
}
```

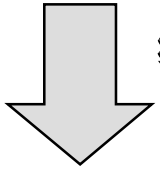
...\vc98\crt\src\newop2.cpp

The struct is used as a function parameter to operator new to indicate that the function should return a null pointer to report an allocation failure, rather than throw an exception.

struct **std::nothrow_t** {};

delete expression

```
Complex* pc = new Complex(1,2);
...
delete pc;
```



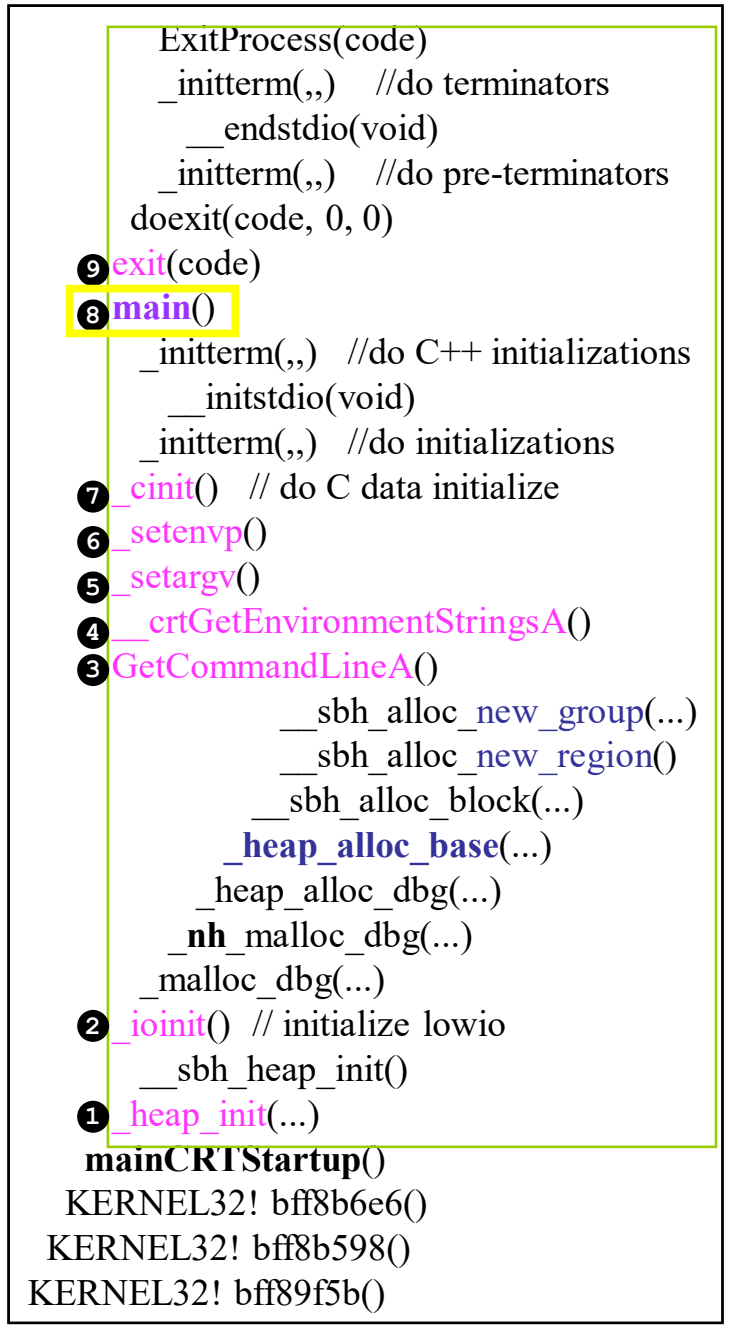
編譯器轉為...

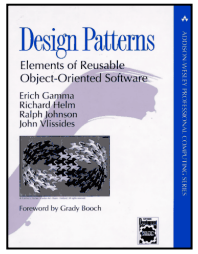
```
pc->~Complex();              //先析構
operator delete(pc);         //然後釋放內存
```

```
void __cdecl operator delete(void *p) _THROW0()
{  // free an allocated object
   free(p);
}
```

...\vc98\crt\src\delop.cpp

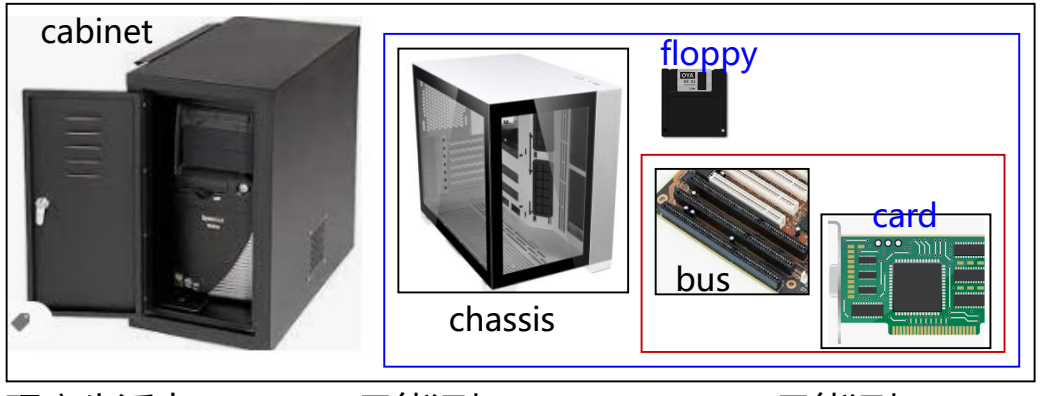
call stack


ExitProcess(code)
_initterm(,,) //do terminators
__endstdio(void)
_initterm(,,) //do pre-terminators
doexit(code, 0, 0)
9 exit(code)
8 main()
_initterm(,,) //do C++ initializations
__initstdio(void)
_initterm(,,) //do initializations
7 _cinit() // do C data initialize
6 _setenvp()
5 _setargv()
4 __crtGetEnvironmentStringsA()
3 GetCommandLineA()
__sbh_alloc_new_group(...)
__sbh_alloc_new_region()
__sbh_alloc_block(...)
_heap_alloc_base(...)
_heap_alloc_dbg(...)
_nh_malloc_dbg(...)
_malloc_dbg(...)
2 _ioinit() // initialize lowio
__sbh_heap_init()
1 _heap_init(...)
mainCRTStartup()
KERNEL32! bff8b6e6()
KERNEL32! bff8b598()
KERNEL32! bff89f5b()

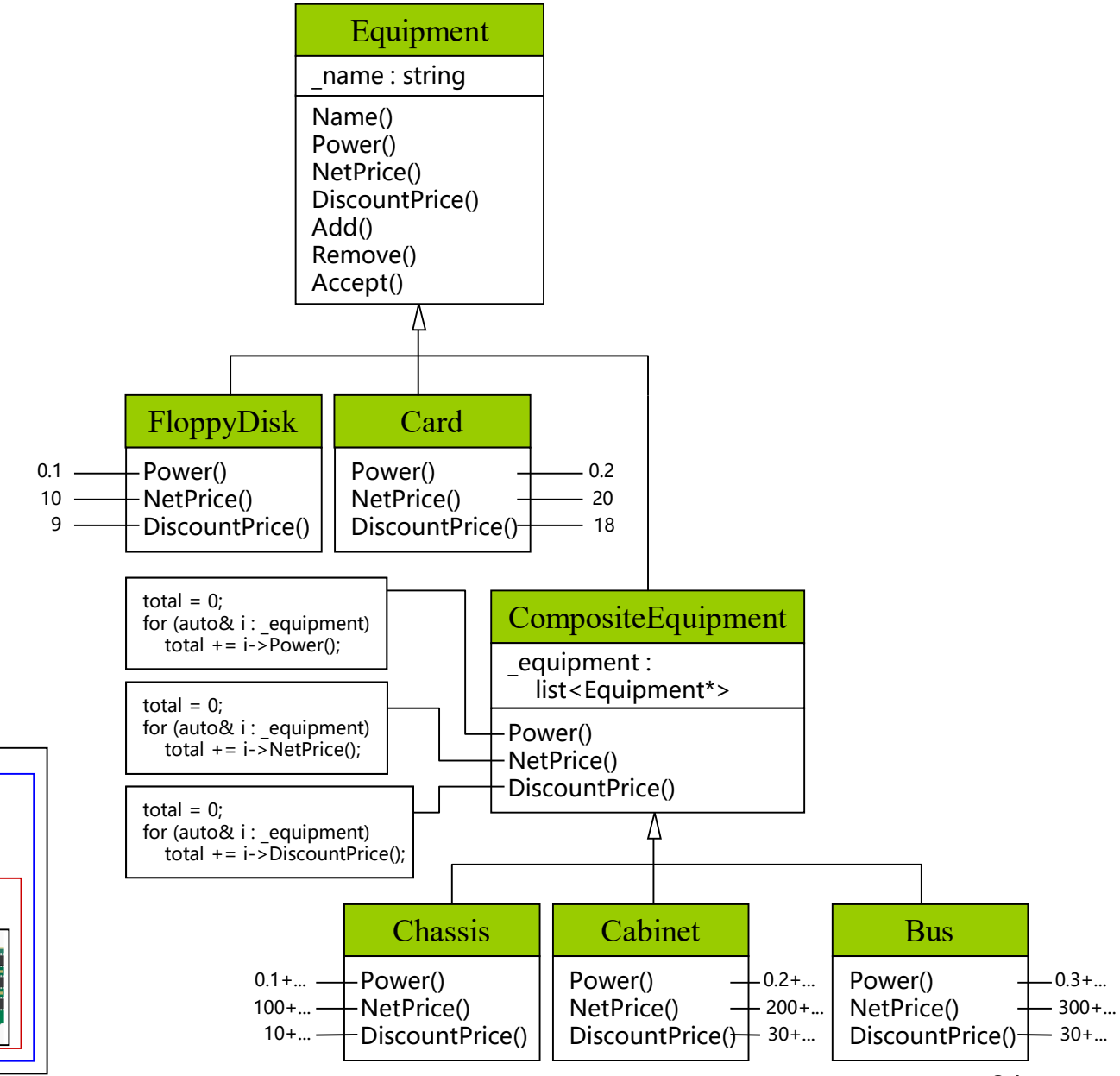


↓以下是《 Design Patterns》的例子。

```
//to assemble equipment into a (pretty simple) personal computer:
   Cabinet* cabinet = new Cabinet("PC Cabinet");
   Chassis* chassis = new Chassis("PC Chassis");

   cabinet->Add(chassis);

   Bus* bus = new Bus("MCA Bus");
   bus->Add(new Card("16Mbs Token Ring"));

   chassis->Add(bus);
   chassis->Add(new FloppyDisk("3.5in Floppy"));

   cout << "The net price is " << chassis->NetPrice() << endl;     //430
   cout << "The discount price is " << chassis->DiscountPrice() << endl;  //67
   cout << "The Power(Watt) is " << chassis->Power() << endl;//0.7
```

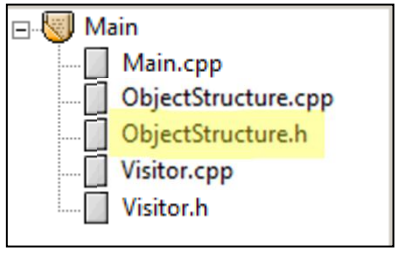


現實生活中， cabinet 只能添加 chassis； chassis 只能添加 floppy 和 bus； bus 只能添加 card。**這該如何實現?**



Main.cpp | ObjectStructure.h | Visitor.h

```
1  //20210922
2
3  #ifndef __ObjectStructure_H__
4  #define __ObjectStructure_H__
5
6  #include <list>
7  #include <string>
8  using namespace std;
9
10 #define Currency int
11 #define Watt float
12 class MyException {};
13
14 //! #include "Visitor.h"   //這樣不行。
15 class VisitorEquipment;    //這樣可以。 (上面不行而下面可以，不佳，應解決)
16
17
18 /*
19  《Design Patterns》p.334
20 Element
21  defines an Accept operation that takes a visitor as an argument.
22 */
23
24 /*
25 侯捷註：多設計一個 base abstract class Accepter;
26 class Accepter
27 {
28 public:
29     virtual void Accept(VisitorEquipment&)=0;
30 };
31 並讓所有 Equipment derived classes 都繼承自 class Accepter, 那也 ok.
32 */
```



全部寫在一個 .cpp 中，和
分散寫在不同的 .h & .cpp 中，
難度不同。應練習後一種寫法。

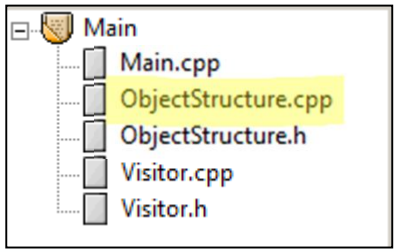
```
34 class Equipment { //原本是這樣沒繼承；當然可以。
35 //class Equipment : public Accepter { //有一種作法是讓它繼承自
36 private:
37     const std::string _name;
38 protected:
39     Equipment(const std::string& name);
40 public:
41     virtual ~Equipment();
42     const std::string& Name();
43     virtual Watt Power()=0;
44     virtual Currency NetPrice()=0;
45     virtual Currency DiscountPrice()=0;
46     virtual void Add(Equipment*);
47     virtual void Remove(Equipment*);
48     virtual void Accept(VisitorEquipment&)=0;
49 };
50
51 /*
52  《Design Patterns》p.335
53 ConcreteElement
54  implements an Accept operation that takes a visitor as an arg
55 */
56 class FloppyDisk : public Equipment
57 {
58 public:
59     FloppyDisk(const std::string& name);
60     virtual ~FloppyDisk() override;
61     virtual Watt Power() override;
62     virtual Currency NetPrice() override;
63     virtual Currency DiscountPrice() override;
64     virtual void Accept(VisitorEquipment& visitor) override;
65 };
```

```
67  class Card : public Equipment
68  {
69  public:
70      Card(const std::string& name);
71      virtual ~Card() override;
72      virtual Watt Power() override;
73      virtual Currency NetPrice() override;
74      virtual Currency DiscountPrice() override;
75      virtual void Accept(VisitorEquipment& visitor) override;
76  };
77
78  //------------------------
79  //以下是複合物 (composite)
80  //------------------------
81  /*
82   《Design Patterns》p.335
83  ObjectStructure (Program)
84    • can enumerate its elements.
85    • may provide a high-level interface to allow the visitor to visit
86      its elements.
87    • may either be a composite or a collection such as a list or a set.
88  */
89  class CompositeEquipment : public Equipment
90  {
91  protected: //20210921 原是 private，為了讓 Chassis::Accept() 能夠
92          //直接迭代，所以 protected.
93      std::list<Equipment*> _equipment;
94  protected:
95      CompositeEquipment(const std::string& name);
96  public:
97      virtual ~CompositeEquipment() override;
98      virtual Watt Power() override;
99      virtual Currency NetPrice() override;
100     virtual Currency DiscountPrice() override;
101     virtual void Add(Equipment*) override;
102     virtual void Remove(Equipment*) override;
103     virtual void Accept(VisitorEquipment& visitor) override;
104 };
```

```
106 class Chassis : public CompositeEquipment
107 {   //機殼
108 public:
109     Chassis(const std::string& name);
110     virtual ~Chassis() override;
111     virtual Watt Power() override;
112     virtual Currency NetPrice() override;
113     virtual Currency DiscountPrice() override;
114     virtual void Accept(VisitorEquipment& visitor) override;
115 };
116
117 class Cabinet : public CompositeEquipment
118 {   //櫃體
119 public:
120     Cabinet(const std::string& name);
121     virtual ~Cabinet() override;
122     virtual Watt Power() override;
123     virtual Currency NetPrice() override;
124     virtual Currency DiscountPrice() override;
125     virtual void Accept(VisitorEquipment& visitor) override;
126 };
127
128 class Bus : public CompositeEquipment
129 {   //匯流排; 總線
130 public:
131     Bus(const std::string& name);
132     virtual ~Bus() override;
133     virtual Watt Power() override;
134     virtual Currency NetPrice() override;
135     virtual Currency DiscountPrice() override;
136     virtual void Accept(VisitorEquipment& visitor) override;
137 };
138
139 #endif //__ObjectStructure_H__
```

ObjectStructure.cpp | Visitor.cpp

```cpp
//20210922

#include <list>
#include <string>
using namespace std;

#include "ObjectStructure.h"
#include "Visitor.h"

/*
 《Design Patterns》p.
*/

//.cpp 的所有 member functions 的實現，都：
//不可寫出 virtual，會導致報錯：
//  [Error] 'virtual' outside class declaration
//不可寫出 override，會導致報錯：
//  [Error] virt-specifiers in 'Add' not allowed outside a class definition

Equipment::Equipment(const std::string& name) : _name(name) {}
Equipment::~Equipment() {}
const std::string& Equipment::Name() { return _name; }
void Equipment::Add(Equipment*) { throw MyException(); }
void Equipment::Remove(Equipment*) { throw MyException(); }

FloppyDisk::FloppyDisk(const std::string& name) : Equipment(name) {}
FloppyDisk::~FloppyDisk() {}
Watt     FloppyDisk::Power() { return 0.1; }
Currency FloppyDisk::NetPrice() { return 10; }
Currency FloppyDisk::DiscountPrice() { return 9; }
void     FloppyDisk::Accept(VisitorEquipment& visitor) { visitor.Visit(this); }

Card::Card(const std::string& name) : Equipment(name) {}
Card::~Card() {}
Watt     Card::Power() { return 0.2; }
Currency Card::NetPrice() { return 20; }
Currency Card::DiscountPrice() { return 18; }
void Card::Accept(VisitorEquipment& visitor) { visitor.Visit(this); }
```

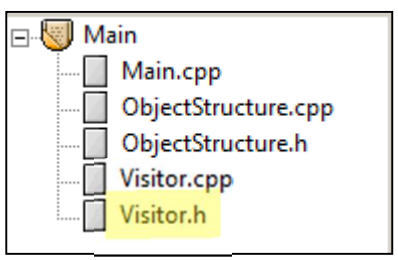



```cpp
//------------------------
//以下是複合物 (composite)
//------------------------
CompositeEquipment::CompositeEquipment(const std::string& name) : E
CompositeEquipment::~CompositeEquipment() {}
Currency CompositeEquipment::NetPrice()
{
    Currency total = 0;
    for (auto& i : _equipment) {
        total += i->NetPrice();
    }
    return total;
}
Currency CompositeEquipment::DiscountPrice()
{
    Currency total = 0;
    for (auto& i : _equipment) {
        total += i->DiscountPrice();
    }
    return total;
}
Watt CompositeEquipment::Power()
{
    Watt total = 0;
    for (auto& i : _equipment) {
        total += i->Power();
    }
    return total;
}
void CompositeEquipment::Add(Equipment* newOne)
{
    _equipment.push_back(newOne);
}
void CompositeEquipment::Remove(Equipment* oldOne)
{
    _equipment.remove(oldOne);
}
void CompositeEquipment::Accept(VisitorEquipment& visitor)
{
    visitor.Visit(this);
}
```

```
83  Chassis::Chassis(const std::string& name) : CompositeEquipment(name) {}
84  Chassis::~Chassis() {}
85  Watt     Chassis::Power() { return 0.1 + CompositeEquipment::Power(); }
86  Currency Chassis::NetPrice() { return 100 + CompositeEquipment::NetPrice(); }
87  Currency Chassis::DiscountPrice() { return 10 + CompositeEquipment::DiscountPrice(); }
88  void Chassis::Accept(VisitorEquipment& visitor) {
89      for (auto& i : _equipment) {
90          i->Accept(visitor);
91      }
92      visitor.Visit(this);
93  }
94
95
96  Cabinet::Cabinet(const std::string& name) : CompositeEquipment(name) {}
97  Cabinet::~Cabinet() {}
98  Watt     Cabinet::Power() { return 0.2 + CompositeEquipment::Power(); }
99  Currency Cabinet::NetPrice() { return 200 + CompositeEquipment::NetPrice(); }
100 Currency Cabinet::DiscountPrice() { return 20 + CompositeEquipment::DiscountPrice(); }
101 void Cabinet::Accept(VisitorEquipment& visitor) {
102     visitor.Visit(this);
103 }
104
105
106 Bus::Bus(const std::string& name) : CompositeEquipment(name) {}
107 Bus::~Bus() {}
108 Watt     Bus::Power() { return 0.3 + CompositeEquipment::Power(); }
109 Currency Bus::NetPrice() { return 300 + CompositeEquipment::NetPrice(); }
110 Currency Bus::DiscountPrice() { return 30 + CompositeEquipment::DiscountPrice(); }
111 void Bus::Accept(VisitorEquipment& visitor) {
112     visitor.Visit(this);
113 }
```

Main.cpp | ObjectStructure.h | Visitor.h

```
//20210922
/*
 《Design Patterns》p.331
Represent an operation to be performed on the elements of an object
structure. Visitor lets you define a new operation without changing
the classes of the elements on which it operates.

 《Design Patterns》p.333
Applicability
Use the Visitor pattern when
  •an object structure contains many classes of objects with differing
  interfaces, and you want to perform operations on these objects that
  depend on their concrete classes.

  •many distinct and unrelated operations need to be performed on objects
  in an object structure, and you want to avoid "polluting" their classes
  with these operations. Visitor lets you keep related operations together
  by defining them in one class. When the object structure is shared by
  many applications, use Visitor to put operations in just those applications
  that need them.

  •the classes defining the object structure rarely change, but you
  often want to define new operations over the structure. Changing the
  object structure classes requires redefining the interface to all
  visitors, which is potentially costly. If the object structure classes
  change often, then it's probably better to define the operations in
  those classes.

註：上述前二點尚未很有心得。第三點很有心得。
*/

#ifndef __Visitor_H__
#define __Visitor_H__

#include "ObjectStructure.h"
```




```
#include "ObjectStructure.h"
/*
 《Design Patterns》p.334
Visitor
declares a Visit operation for each class of ConcreteElement t
object structure. The operation's name and signature identifie
class that sends the Visit request to the visitor. That lets t
visitor determine the concrete class of the element being visi
Then the visitor can access the element directly through its p
interface.

侯捷註：由於 C++ overload 特性，這些 operation 可以重名（同名）

 《Design Patterns》p.337
Each object structure will have an associated Visitor class.
This abstract visitor class declares a VisitConcreteElement op
for each class of ConcreteElement defining the object structur
Each Visit operation on the Visitor declares its argument to b
particular ConcreteElement, allowing the Visitor to access the
interface of the ConcreteElement directly. ConcreteVisitor cla
override each Visit operation to implement visitor-specific be
for the corresponding ConcreteElement class.
*/
class VisitorEquipment { //base victor
protected:
    VisitorEquipment() {};
public:
    virtual ~VisitorEquipment() {};
    virtual void Visit(FloppyDisk*) {};
    virtual void Visit(Card*) {};
    virtual void Visit(Chassis*) {};
    virtual void Visit(Bus*) {};
    virtual void Visit(Cabinet*) {};
    virtual void Visit(CompositeEquipment*) {};
        // and so on for other concrete subclasses of Equipmen
};
```

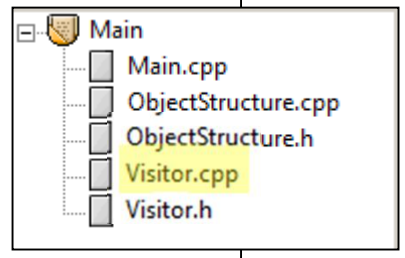
```cpp
/*
《Design Patterns》p.334
ConcreteVisitor
implements each operation declared by Visitor. Each operation
implements a fragment of the algorithm defined for the corresponding
class of object in the structure. ConcreteVisitor provides the
context for the algorithm and stores its local state. This state
often accumulates results during the traversal of the structure.

《Design Patterns》p.337
Each class of ConcreteElement implements an Accept operation that
calls the matching Visit... operation on the visitor for that
ConcreteElement. Thus the operation that ends up getting called
depends on both the class of the element and the class of the visitor.
*/
class VisitorPricing : public VisitorEquipment {
//這個 visitor 要求取得(計算)售價
private:
    Currency _total = 0;
public:
    VisitorPricing();
    ~VisitorPricing() override;
    Currency& GetTotalPrice() { return _total; };

    virtual void Visit(FloppyDisk*) override;
    //virtual void Visit(Card*);
    virtual void Visit(Chassis*) override;
    //virtual void Visit(Bus*);
    //virtual void Visit(Cabinet*);
    //virtual void Visit(CompositeEquipment*);
    // ...
};
```

```cpp
class VisitorPricing2 : public VisitorEquipment {
//這個 visitor 要求取得(計算)拋售售價 (和正常售價的差別在於 GetTotalPrice() 打八折)
private:
    Currency _total = 0;
public:
    VisitorPricing2();
    ~VisitorPricing2() override;
    Currency& GetTotalPrice() { return _total *= 0.8; };

    virtual void Visit(FloppyDisk*) override;
    //virtual void Visit(Card*);
    virtual void Visit(Chassis*) override;
    //virtual void Visit(Bus*);
    //virtual void Visit(Cabinet*);
    //virtual void Visit(CompositeEquipment*);
    // ...
};
```

```
/*
《Design Patterns》p.335
Consequences
Some of the benefits and liabilities of the Visitor pattern are as follows:
 1.Visitor makes adding new operations easy. Visitors make it easy to add
 operations that depend on the components of complex objects. You can define
 a new operation over an object structure simply by adding a new visitor.
 In contrast, if you spread functionality over many classes, then you must
 change each class to define a new operation.

 2.A visitor gathers related operations and separates unrelated ones.
 Related behavior isn't spread over the classes defining the object structure;
 it's localized in a visitor. Unrelated sets of behavior are partitioned
 in their own visitor subclasses. That simplifies both the classes defining
 the elements and the algorithms defined in the visitors. Any algorithm-specific
 data structures can be hidden in the visitor.
 (侯捷：尚未有好體會)

 《Design Patterns》p.336
 3.Adding new ConcreteElement classes is hard. The Visitor pattern makes it
 hard to add new subclasses of Element. Each new ConcreteElement gives rise
 to a new abstract operation on Visitor and a corresponding implementation
 in every ConcreteVisitor class. Sometimes a default implementation can be
 provided in Visitor that can be inherited by most of the ConcreteVisitors,
 but this is the exception rather than the rule.

 So the key consideration in applying the Visitor pattern is whether you
 are mostly likely to change the algorithm applied over an object structure
 or the classes of objects that make up the structure. The Visitor
 class hierarchy can be difficult to maintain when new ConcreteElement
 classes are added frequently. In such cases, it's probably easier just to
 define operations on the classes that make up the structure. If the
 Element class hierarchy is stable, but you are continually adding operations
 or changing algorithms, then the Visitor pattern will help you manage the changes.

 4.Visiting across class hierarchies. An iterator
 can visit the objects in a structure as it traverses them by calling
 their operations. But an iterator can't work across object structures
 with different types of elements.

 This implies that all elements the iterator can visit have
 parent class Item.

 Visitor does not have this restriction. It can visit object
 have a common parent class. You can add any type of object
 interface. For example, in
     class Visitor {
    public:
        // ...
        void Visit(MyType*);
        void Visit(YourType*);
    };
 MyType and YourType do not have to be related through inher
 (侯捷：尚未有好體會)

 5.Accumulating state. Visitors can accumulate state as they
 element in the object structure. Without a visitor, this st
 passed as extra arguments to the operations that perform th
 or they might appear as global variables.

 6.Breaking encapsulation. Visitor's approach assumes that t
 interface is powerful enough to let visitors do their job.
 the pattern often forces you to provide public operations t
 an element's internal state, which may compromise its encap
*/
class Inventory { //庫存  //《Design Patterns》書中無此段代
private:
    int sum = 0;
public:
    void Accumulate(const Equipment* e) {
        //...
    }
    operator int() { return sum; }
};
```

```
198 class VisitorInventory : public VisitorEquipment {
199 //這個 visitor 要求取得(計算)庫存
200 private:
201     Inventory _inventory;
202 public:
203     VisitorInventory();
204     ~VisitorInventory() override;
205     Inventory& GetInventory() { return _inventory; };
206
207     virtual void Visit(FloppyDisk*) override;
208     //virtual void Visit(Card*) override;
209     virtual void Visit(Chassis*) override;
210     //virtual void Visit(Bus*) override;
211     // ...
212 };
213
214 #endif //__Visitor_H__
```

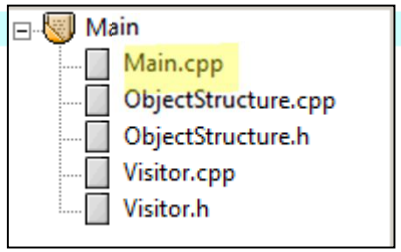
ObjectStructure.cpp | Visitor.cpp

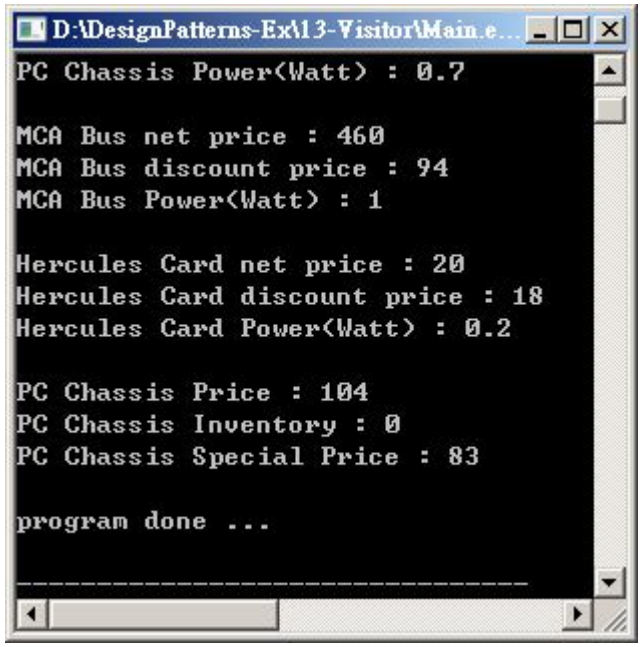



```
1  //20210922
2
3
4  #include "Visitor.h"
5
6  //售價
7  VisitorPricing::VisitorPricing() {}
8  VisitorPricing::~VisitorPricing() {}
9  void VisitorPricing::Visit(FloppyDisk* e) {
10     _total += e->NetPrice();        //個別物以定價出售
11 }
12 void VisitorPricing::Visit(Chassis* e) {
13     _total += e->DiscountPrice();   //複合物以折扣價出售
14 }
15
16 //拋售售價
17 VisitorPricing2::VisitorPricing2() {}
18 VisitorPricing2::~VisitorPricing2() {}
19 void VisitorPricing2::Visit(FloppyDisk* e) {
20     _total += e->NetPrice();        //個別物以定價出售
21 }
22 void VisitorPricing2::Visit(Chassis* e) {
23     _total += e->DiscountPrice();   //複合物以折扣價出售
24 }
25
26 //存貨; 庫存
27 VisitorInventory::VisitorInventory() {}
28 VisitorInventory::~VisitorInventory() {}
29 void VisitorInventory::Visit(FloppyDisk* e) {
30     _inventory.Accumulate(e);
31 }
32 void VisitorInventory::Visit(Chassis* e) {
33     _inventory.Accumulate(e);
34 }
```

Main.cpp | ObjectStructure.h | Visitor.h

```
1  //20210922
2
3  #include <iostream>
4  using namespace std;
5
6  #include "Visitor.h"
7  #include "ObjectStructure.h"
8
9  //------------------
10 int main()
11 {
12     {
13     //to assemble equipment into a (pretty simple) personal computer:
14     Cabinet* cabinet = new Cabinet("PC Cabinet");
15     Chassis* chassis = new Chassis("PC Chassis");
16     Bus* bus = new Bus("MCA Bus");
17     bus->Add(new Card("16Mbs Token Ring"));
18     cabinet->Add(chassis);
19     chassis->Add(bus);
20     chassis->Add(new FloppyDisk("3.5inch Floppy"));
21
22     cout << cabinet->Name() << " net price : " << cabinet->NetPrice() << endl;              //630
23     cout << cabinet->Name() << " discount price : " << cabinet->DiscountPrice() << endl;    //87
24     cout << cabinet->Name() << " Power(Watt) : " << cabinet->Power() << endl;               //0.9
25     cout << endl;
26     cout << chassis->Name() << " net price : " << chassis->NetPrice() << endl;              //430
27     cout << chassis->Name() << " discount price : " << chassis->DiscountPrice() << endl;    //67
28     cout << chassis->Name() << " Power(Watt) : " << chassis->Power() << endl;               //0.7
29     cout << endl;
30
31
32     Card card("Hercules Card");
33     FloppyDisk floppyDisk("5.25in Floppy");
34     bus->Add(&card);
35     bus->Add(&floppyDisk);
36 //! card.Add(&floppyDisk);  //terminate called after throwing an instance of 'MyException'
37                             //很好，Card 不是複合物，不能對它 Add().
```





```
39      cout << bus->Name() << " net price : " << chassis->NetPrice() << endl;              //460
40      cout << bus->Name() << " discount price : " << chassis->DiscountPrice() << endl;    //94
41      cout << bus->Name() << " Power(Watt) : " << chassis->Power() << endl;               //1
42      cout << endl;
43      cout << card.Name() << " net price : " << card.NetPrice() << endl;              //20
44      cout << card.Name() << " discount price : " << card.DiscountPrice() << endl;    //18
45      cout << card.Name() << " Power(Watt) : " << card.Power() << endl;               //0.2
46      cout << endl;
47
48
49
50      VisitorPricing   visitorPricing;
51      VisitorInventory visitorInventory;
52      chassis->Accept(visitorPricing);
53      cout << chassis->Name() << " Price : " << visitorPricing.GetTotalPrice() << endl; //104
54      chassis->Accept(visitorInventory);
55      cout << chassis->Name() << " Inventory : " << visitorInventory.GetInventory() << endl; //0
56
57      VisitorPricing2   visitorPricing2;
58      chassis->Accept(visitorPricing2);
59      cout << chassis->Name() << " Special Price : " << visitorPricing2.GetTotalPrice() << endl; //83
60
61
62      chassis->Add(&card);     //按現實生活，card 應被加入 bus 而不是直接加入 chassis。
63                               //但目前的實現無法檢測這一點。
64                               //這真像 Java Swing windowing system.
65
66      delete cabinet;
67      delete chassis;
68      delete bus;
69      //有些 memory block 未被明白清除(如上)，像是程式中 "直接將 new 所得 pointer 做為函數實參傳遞" 者。
70      //它們的地址沒有被某個 pointer 記錄下來，也就無法被明白 delete.
71      }
72
73      cout << "\nprogram done ... \n";
74      return 0;
75  }
```

← 欲解決"亂點鴛鴦譜" 的情況，
以 Cabinet 為例可以這麼做 (見下頁)

欲解決"亂點鴛鴦譜" 的情況，以 Cabinet 為例可以這麼做。

原本

```
117 class Cabinet : public CompositeEquipment
118 {   //櫃艙
119 public:
120     Cabinet(const std::string& name);
121     virtual ~Cabinet() override;
122     virtual Watt Power() override;
123     virtual Currency NetPrice() override;
124     virtual Currency DiscountPrice() override;
125     virtual void Accept(VisitorEquipment& visitor) override;
126 };
```

修改後

```
107 class Chassis;
108 class Cabinet : public CompositeEquipment
109 {   //櫃艙
110 public:
111     Cabinet(const std::string& name);
112     virtual ~Cabinet() override;
113     virtual Watt Power() override;
114     virtual Currency NetPrice() override;
115     virtual Currency DiscountPrice() override;
116     virtual void Accept(VisitorEquipment& visitor) override;
117             void Add(Chassis* chassis);
118 };
```

實作

```
104 void Cabinet::Add(Chassis* chassis) {
105     _equipment.push_back(chassis);
106 }
107
```

↑這個函數會遮掩 (hidden) 繼承而來的
virtual void Add(Equipment*)
於是以下 main() 之中 L65, L68 失敗，L67 成功。

```
62      chassis->Add(&card);    //按現實生活，card 應被加入 bus 而不該直接加入 chassis。
63                              //但目前的實現無法檢測這一點。
64                              //這真像 Java Swing windowing system.
65  //! cabinet->Add(new FloppyDisk("5.25 inch Floppy"));
66  //[Error] no matching function for call to 'Cabinet::Add(FloppyDisk*)'. 很好.
67      cabinet->Add(new Chassis("IBM Chassis"));   //ok
68  //! cabinet->Add(new Bus("MCA Bus2"));
69  //[Error] no matching function for call to 'Cabinet::Add(Bus*)' 很好.
70  //也就是說，Cabinet 就是只接受 Chassis*, 不"直接接受" Chassis 內的 Bus or Card or Floppy.
71
```

↑

Q : 奇怪，根據 LSP 不是應該可以嗎?
A : 因為 Chassis 並非 Bus or Card or Floppy 的 base.
也因此，如果想讓 Chassis::Add() 能(只能) 接受 A, B, C 三"種"東西，
就該為 class A, class B, class C 設計一個共同的 base class。

The End