

物件導向程式設計

大綱

- 前言
- 物件導向概述
- 物件導向理論簡介
- 物件導向方法
- 物件導向實作

前言

- 在開始進行實作前，先說明主要用到的參考書籍：
 - 如何用物件導向實作複雜的業務需求
 - **API Design for C++**
- 我們常常使用別人設計好的 API 來使用，這符合現代軟體設計的概念，利用軟體元件拼湊組合出新的功能，不做重覆的事。
- 在實務上，為了求快，工程師們「常知其然不知所以然」，反正呼叫 API 能動或能給出想要的結果就好。
- 但這樣的想法到底是讓軟體設計變的更輕鬆，還是入門容易熟練難呢？

前言(續)

- 當學校在教「理論課程」時，實作部份是直接呼叫別人已經設計好的函式庫，這中間會有個很大的「學用落差」，那就是學這麼多幹嘛？會呼叫函式庫就好。
- 但真的這樣就夠了嗎？中間跳過了甚麼？
 - 這個軟體元件為何要這樣設計？它隱含的分析及設計流程全部跳過。
 - 如果一味地只會用別人設計好的東西，那當遇到的非典型情況時，自己又該怎麼處理呢？等待別人出手救援？
- 並不是說甚麼東西都要自己重新設計，這樣太沒效率也不符合現代軟體設計原則，但最起碼還有機會當學生上課或自己下班進修時，能多一點「耐心」去了解常用到的工具軟體從何而來。
- 當你邊上理論課程，然後自己又有能力設計出基於理論的實用軟體工具，並分享給其他人使用時，豈不是更滿足也更有「成就感」！

物件導向概述

- ❑ 常見的 C 語言是屬於程序導向中結構化程式設計的概念，採取「自上而下、逐步細化、模組化」的方法，將軟體的複雜度控制在一個範圍內。
- ❑ 從而降低軟體發展的複雜度，因此 C 語言成為 20 世紀 70 年代軟體發展的潮流。
- ❑ 隨著硬體的快速發展，業務要求越來越複雜，而且程式設計應用領域越來越廣泛，結構化程式設計的「軟體生產力」遠跟不上硬體和業務的發展。
- ❑ 因為結構化程式設計的方式無法滿足軟體「可擴充性」和「可維護性」的需求，因此「物件導向」的概念才開始普及。
- ❑ 從 C++ 到後來的 Java、C# 把「物件導向」推向巔峰。
- ❑ 和「程序導向」相比，「物件導向」的概念更加貼近人類思維的特點，這也是「軟體設計」的一次重大突破。

物件導向概述(續)

- ❑ 程序導向是一種以「程序」為中心的程式流水線設計概念，其中最重要的就是「完成一件事情的步驟」。
- ❑ 這是一種機械式的邏輯，每個階段都有自己的輸入資料、處理單元及輸出資料。
- ❑ 在程序導向中，我們需要將程式分成不同的處理單元，然後設計不同單元該如何銜接，並定義每個單元的輸入及輸出資料類型。
- ❑ 程式設計人員應該都聽過「程式設計 = 演算法 + 資料結構」，這其實就是程序導向的概念。
- ❑ 而這種特徵其實和電腦的本質相關，其核心 CPU 處理指令的方式就如同流水線一樣，所以電腦是以「程序導向」為基石來進行處理。
- ❑ 即使我們使用物件導向語言來進程式開發，但它最後還是要轉換成 CPU 能執行的指令，依舊屬於程序導向。

物件導向概述(續)

- 物件導向是一種以「物件」作為中心的程式設計概念，著重在對現實世界的模擬。
- 跟程序導向不同，物件導向的方法中沒有主要控制的角色，也不需要指定嚴謹的操作順序，而是以物件為主體，指定這些物件完成任務，以及這些物件如何對外界的刺激做出反應。
- 人們大多是按照物件導向的方式進行思考，所以物件導向更加符合人類的思維習慣。
- 在物件導向的程式設計中，「程式設計 = 物件 + 互動」

物件導向概述(續)

□ 為什麼需要物件導向？

- 由於程序導向是電腦運行的基石，因此目前 C 語言、作業系統、協定堆疊、驅動程式，依然還是程序導向的天下。
- 但程序導向有著先天性的不足，那就是流程和結構相對固定，雖有效率但其擴充比較麻煩。每次需求的變更，都要對流程的每個步驟、中間的進出的資料結構進行修改。
- 物件導向是為了解決程序導向的「擴充性」問題而誕生的，因此物件導向的最大特徵就是「可擴充性」；要將變化帶來的影響控制在有限的範圍內，避免產生全流程或大範圍的影響，降低開發上的風險。
- 「經常發生變化的地方」就是物件導向的發威之處，常見及可變的主要集中在「客戶需求」部份，不變的一般都屬於電腦系統的基礎。

物件導向概述(續)

□ 為什麼需要物件導向？

- 作業系統、資料庫及協定等，相對穩定而且要求高效率，因此較適合「程序導向」。
- 而企業應用、網際網路或遊戲等應用，需求經常變更且功能不斷擴展，適用「物件導向」的概念。
- 對效能要求很高的系統軟體，基本上都是利用 C 語言寫的，例如作業系統、驅動程式、嵌入式軟體及網路設備等。

物件導向概述(續)

□ 物件導向語言不等於物件導向程式設計：

- C 語言是純粹的「程序導向」的程式語言，但不代表用 C 語言撰寫程式就是程式導向程式設計，利用 C 語言一樣可以寫出物件導向的程式。
- 同理，Java 是純粹的物件導向程式語言，但也可以用 Java 寫出程序導向的程式。
- 程序導向和物件導向都是一種「思維方式」，它是一種思考問題的方法，和具體的語言沒有必然的關係。
- 舉例來說，在 Java 中寫程序導向的程式碼，最簡單的方式是撰寫一個大類別，內含很多方法，然後在 main 函式裡面按照程式導向的方式呼叫即可。

物件導向理論簡介

□ 類別：

- 類別是「物件導向」領域裡最基礎的概念，也是物件導向分析和設計的基石。
- 所謂類別，就是站在個人的觀察角度，具有「相似點」的事物，就是同一類。

□ 物件導向類別：

- 物件導向的類別由兩部份組成：屬性和方法。
- 屬性：類別具有的特性。
- 方法：類別具有的功能。
- 實際設計程式時，可以用「屬性是名詞，方法是動詞」來判斷。

物件導向理論簡介(續)

□ 舉例來說：

- 這張數位影像的長度為 100 像素，寬度為 50 像素，色彩深度為 24 位元，這裡的長寬及色彩深度就是「數位影像」的屬性。
- 想獲取點位置 X 在 50 像素，Y 在 20 像素的 RGB 彩色值，這裡的「獲取彩色值」就是「數位影像」的方法。

□ 設計方法的基本原則：方法單一化原則，即「一個方法只做一件事」。

□ 設計屬性的基本原則：屬性最小化原則，即「屬性不可再分」。

物件導向理論簡介(續)

□ 物件：

- 物件就是一個真實存在的類別，物件導向是對現實世界的模擬，那物件就是現實世界存在的「物體」。
- 真正在軟體運行過程的是「物件」，而不是「類別」。
- 「類別」是程式撰寫時，由程式人員分析歸納而來。
- 軟體類別：軟體設計過程中歸納總結出來的分類。
- 軟體物件：軟體實際運行過程中存在的物件。
- 軟體類別是對現實類別的模擬，但不是簡單的等同；除了實作現實類別相對應的功能，還會創造出許多現實中不存在的類別。
- 這個創造過程正是各種「設計方法」、「設計模式」、「設計原則」大顯身手的地方。

物件導向理論簡介(續)

□ 物件導向的三大核心特徵：

- 封裝、繼承、多型是物件導向的三大核心特徵，判斷一種程式語言是否為物件導向的程式語言，就看其是否支援這三大核心特徵。
- 封裝資料的主要原因是「保護隱私」；在程序導向的設計中，資料結構是公開的，任何能取得資料的人都可以隨意修改，也允許以不同的方式修改。
- 如果資料被改錯了，其他依賴此資料的函式會受到影響，甚至造成「程式崩潰」。
- 物件導向的類別封裝屬性後，對屬性的修改只能透過類別的方法進行，一來不會曝露內部的具體屬性；二來對屬性都是統一的操作，不會出現亂修改的情況。
- 封裝方法的主要原因是「隔離複雜度」；每個類別只需要關注自己負責的功能如何完成即可，如果需要其他類別配合，僅需呼叫類別的方法，而不用瞭解其內部的具體實作。
- 程序導向 = 演算法 + 資料結構；這裡的資料結構是公開的，每個地方都能看到和引用，否則程序導向中的各個處理流程就沒辦法處理。

物件導向理論簡介(續)

□ 物件導向的封裝有三種方式：

- public: 這種方式就是不封裝，直接對外公開。
- protected: 這種封裝方式對外不公開，對朋友 (friend) 或子類別公開。
- private: 這種封裝方式對誰都不公開。

□ 繼承：

- 繼承是物件導向語言最基本的特徵，如果一種語言沒有繼承機制，就說不上是真正的物件導向語言。
- 在物件導向中，繼承的實際意義可以用「遺傳」來形容，程式設計師決定父類別遺傳什麼給子類別。

物件導向方法：分析和設計流程概述

- 專案管理的流程主要指導專案經理如何管理專案，但對於指導開發人員如何開發專案，並無多大用處。
- 剛畢業的學生或許對「軟體工程」很熟悉，對各個開發階段應該做什麼，不同的開發流程有什麼優缺點，說的頭頭是道；可是一旦參與專案開發後，便會有一種無從下手的感覺。
- 舉例來說：
 - 需求分析階段要分析需求，但具體怎麼分析？
 - 客戶的需求是描述語句，例如「我們需要一個影像處理的軟體」，而程式碼則是一個具體的類別和函式。那怎麼從描述語句轉化成具體的類別和函式呢？
 - 具體的語言特性，例如 C++ 的 private, protected, public 等屬性是來自哪裡？該如何設計？
 - 物件導向的類別、屬性、方法等，是怎麼設計出來的？

物件導向方法：分析和設計流程概述（續）

- ❑ 相信專案開發的問題都曾困擾著你我，但軟體工程並未給出答案，導致我們在實際開發過程中，只能在別人的設計與指導下工作，或是亂湊出能滿足需求的想法，至於效果，就得靠上天眷顧了。
- ❑ 有慧根的人，經過一段時間的磨練後，可能就漸漸掌握門道，但大多數的人可能就一直原地踏步，不斷地執行別人分配的工作內容。
- ❑ 「專案開發」也有一套完整的程序。
- ❑ 對於物件導向來說，整個開發流程實際上非常清晰，底下我們將「瀑布模型」、「敏捷開發」等稱為『管理流程』；物件導向流程開發稱為『技術流程』。

物件導向方法：分析和設計流程概述(續)

- 物件導向的技術流程可以概括如下：需求模型→領域模型→設計模型→實作模型。
 - 需求模型：透過和客戶溝通，結合產業經驗和知識，明確地闡述客戶的需求。
 - 領域模型：根據需求模型，擷取出領域相關概念，為後面的物件導向設計打下基礎。
 - 設計模型：以領域模型為基礎，綜合物件導向的各種設計技巧，完成類別的設計。
 - 實作模型：以設計模型為基礎，將設計轉譯為具體的程式語言實作，完成程式碼撰寫。
 - 技術流程環環相扣，上一步流程的輸入就是下一步流程的輸入。藉由這種 step by step 的方式，可以完成從需求到最後實作的相關工作。

物件導向方法：設計模型

- 物件導向領域經過幾十年的演進，已經發展出很多成熟的指導方針和方法，以利於評價和規範物件導向設計；其中最具代表性的就是「設計原則」和「設計模式」。
- 設計原則：
 - 當提及物件導向領域的設計原則時，其實都是在談論 Robert C Martin 的「SOLID 原則」。
 - 設計原則是一個判斷標準，應用設計原則的根本目的是要「保證可擴充性」。
- 設計模式：
 - 相較於設計原則，設計模式更加普及與流行；一般談到設計方法時，都會先想到設計模式。
 - 通常談論設計模式時，其實都是指 GoF (Gang of Four) 所講的設計模式。
 - 設計原則和設計模式是互補關係：設計原則主要規範「類別的定義」，而設計模式主要規範「類別的行為」。

物件導向方法：設計模型(續)

□ 拆分輔助類別：

- 拆分類別的主要目的，是為了使撰寫類別時能夠滿足一些框架或規範的要求；例如常見的 MVC 模式，拆分成 Model, View, Control 三個元素等。
- 只要將設計出來的類別，按照規範要求，對應拆分即可。
- 這是為了滿足框架或者規範的要求，本身並不是設計，而是實作的一個步驟，所以一般都不需要將拆分的輔助類別體現於類別模型中，只在設計程式時拆分即可。

物件導向實作

- 以物件導向的方式進行設計，只是進攻的前奏，撰寫程式才是最終的目標。
- 物件導向的理論、方法、技巧經過多年的發展後，業界已經形成基本統一的認知，但並未出現一種統一的「物件導向程式語言」。
- 程式語言的差異性：
 - 每種語言都會根據自己的需求、特徵，選擇支援或不支援物件導向，或者只支援部份物件導向的特性。
 - 各種語言對物件導向支援的程度和方式，導致將其設計轉換為程式碼時，還需要做一次翻譯或轉換的過程。
 - 若想充分應用物件導向，除了掌握物件導向的概念、方法、技巧外，還得要精通程式語言與物件導向相關的特性。

物件導向實作(續)

- 本課程以 C/C++ 為主要程式語言，並描述它們與物件導向相關的特性；在 GUI 的設計上，我們會採用 C# 及 Python 來說明。
- 先介紹 C++ 跟物件導向相關的資料，後續我們會再參考「API Design for C++」一書來了解更多的細節。
- C++
 - C++ 是第一個廣泛流行的物件導向程式語言，至今依然佔據程式語言榜單的前幾名。
 - C++ 促進了物件導向程式設計的發展，但它的複雜性也伴隨著較大的爭議；尤其是與物件導向相關的特性，光是各式各樣的術語，就能把剛入門的程式開發者搞得頭昏眼花。
 - C++ 相容於 C 語言的特性，導致在程式中，一不留神就會出現「物件導向」和「程序導向」風格混雜的情況。

物件導向實作(續)

□ 類別：

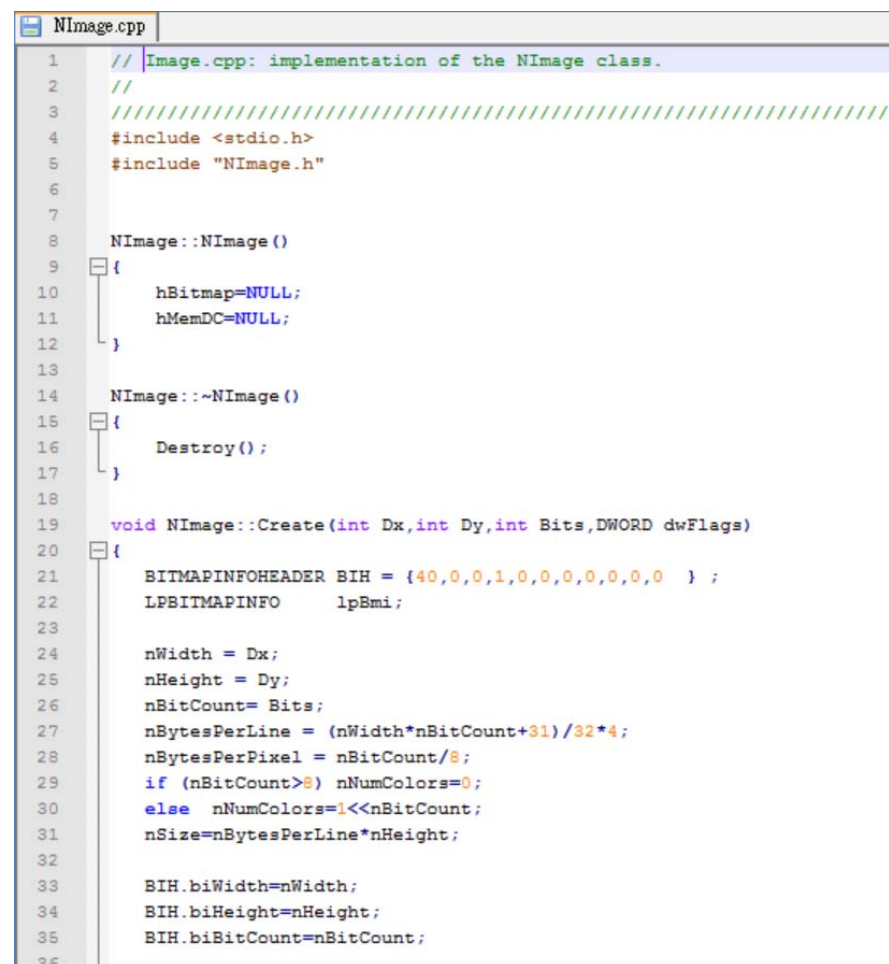
- C++ 類別透過關鍵字 `class` 來標示。
- 類別的定義包含兩部分：類別標頭(header, .h)，由 `class` 和後面的類別名稱組成。
- C++ 是分段的存取控制方式，每段的開始透過存取限定詞加上一個冒號 (`:`)，直到另一個分段開始時結束，或者到類別定義結束。
- 一個類別允許包含多個存取控制段；如果沒有指定存取限定詞，則預設為 `private`。

```
NImage.h
4
5
6 #include <windows.h>
7
8 #ifndef _NIMAGE_H
9 #define _NIMAGE_H
10
11 #define WIDTHBYTES(bits) (((bits) + 31) / 32 * 4)
12
13 class NImage
14 {
15 private:
16     HBITMAP      hBitmap;
17     LPBYTE       lpBits;
18
19     int          nWidth;
20     int          nHeight;
21     int          nBitCount;
22     int          nBytesPerLine;
23     int          nBytesPerPixel;
24     int          nNumColors;
25     int          nSize;
26
27     HDC          hMemDC;
28
29 public:
30     NImage();
31     ~NImage();
32
33 public:
34     // Overrides
35     virtual BOOL BitBlt(HDC, int, int, int, int, int, int, DWORD);
36     virtual BOOL MaskBlt(HDC, int, int, HBITMAP, DWORD);
37     virtual HDC  GetDC();
38     virtual void ReleaseDC();
```


物件導向實作(續)

□ 類別：

- 在 C++ 中，宣告和定義是分開的，其方法需要在定義檔 (source, .cpp) 實作。
- 類別實作 (source, .cpp)，由一對大括弧包圍起來，裡面包含屬性和方法的宣告。



```
1 // Image.cpp: implementation of the NImage class.
2 //
3 ////////////////////////////////////////////////////////////////////////////////
4 #include <stdio.h>
5 #include "NImage.h"
6
7
8 NImage::NImage()
9 {
10     hBitmap=NULL;
11     hMemDC=NULL;
12 }
13
14 NImage::~NImage()
15 {
16     Destroy();
17 }
18
19 void NImage::Create(int Dx,int Dy,int Bits,DWORD dwFlags)
20 {
21     BITMAPINFOHEADER BIH = {40,0,0,1,0,0,0,0,0,0 };
22     LPBITMAPINFO lpBmi;
23
24     nWidth = Dx;
25     nHeight = Dy;
26     nBitCount= Bits;
27     nBytesPerLine = (nWidth*nBitCount+31)/32*4;
28     nBytesPerPixel = nBitCount/8;
29     if (nBitCount>8) nNumColors=0;
30     else nNumColors=1<<nBitCount;
31     nSize=nBytesPerLine*nHeight;
32
33     BIH.biWidth=nWidth;
34     BIH.biHeight=nHeight;
35     BIH.biBitCount=nBitCount;
36 }
```