物件與類別

緣由

- 真實世界均由物件所構成，維持世界的運作是由各個物件之間互動來產生。為了模擬真實世界，解決真實世界的問題，利用「物件」的概念來架構所有的軟體，並把物件視為軟體的基本單位。

物件導向程式設計簡介 (OOP, object-oriented programming)

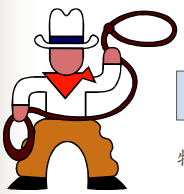
- 一套應用物件導向的方法與原則來開發程式。
- 又包含物件導向分析與設計(OOA/D)兩個過程
  - 物件導向分析(object-oriented analysis):焦點放在調查問題與需求上(問題分析與需求分析)
  - 物件導向設計(object-oriented design):焦點放在滿足需求的概念性解決方案，定義滿足需求的軟體物件，並實作(design pattern)
- 統一模型語言(UML，unified modeling language),它不是OOA/D的方法論，它只是表示法而已(UML的CASE工具，可繪製UML圖)

物件(object)是什麼?

- object---物件(東西)、概念
  宇宙間任何具體的東西或抽象的事物
- 三個主要特徵來描述物件(Object):
  - 狀態(State)：指物件各種特性的現況　屬性(property)
  - 行為(Behavior)：指物件的功能。　方法(method)
  - 身份(Identity)[Grady Booch, 1991]：身份用來標示一個物件，可能是一個名稱或是號碼
  - Examples:
    - 具體的東西
      - 車子:
        - State:速度，排氣量，顏色，重量，幾人座
        - Behavior : 油門,方向盤,煞車
        - Identity: A先生擁有的汽車
    - 抽象的事物:
      - 會議:
        - State :開會日期,地點,議題,出席人員
        - Behavior :決策
        - Identity :A公司的每月例行會議

物件(object)是什麼?

- 人



物件化

屬性：
性別、年齡、身高、體重、嗜好……
行為：
走路、吃飯、睡覺、說話、雜耍……

視窗物件

屬性：
位置：X=100，Y=120
外框粗細：3點
外框顏色：灰
檢視：大圖示
……
行為：
縮小、放大、關閉、存檔、開檔、複製、剪下、貼上……

物件(object)是什麼?

- 視窗上的按鈕：
  - 屬性：物件的描述資料，例如按鈕的外觀、顏色、大小、形狀等，就是這一個按鈕物件的屬性。
  - 行為：可以對物件做的處理。例如按下按鈕、放開按鈕等動作，就是這一個按鈕物件的行為。
  - Identity:視窗工具列第2個按鈕

物件導向設計(OOP) 的優點

- 物件導向設計可以強化程式碼的擴充性(Extensible)與重用性(Reusable)。
- 符合人性思維模式
- 易於剖析問題
- 獨立且合作、維護容易

物件導向程式設計的基本特性

- 封裝性（Encapsulation）:將資料與處理資料的方法集中在一個類別中，欲取得類別內的資料，必須透過方法來取得，因此資料對外是隱藏的
- 繼承性（Inheritance）:利用繼承的方式來遺傳上層的功能及依需要增減其函式。可以簡化重覆撰寫程式，以及減少出錯的可能。
- 多型性（Polymorphism）:用同樣的表示方式而能處理不同類別資料的方式，就稱之為多型

C++利用類別(class)來實現物件的概念

- **類別（class）**是一種使用者自定的資料型態
  - 類別可像結構一樣，可以在類別中定義多種基本資料型態(如int、char、float)的變數，這些資料變數稱為類別的**資料成員（data member）**。
  - 類別中還可以定義所需的功能函數稱為**成員函數（member function）**。
- 加上行為(函式)的結構—類別（class）
- 把「資料」與處理資料的程式「函式」整合起來

類別（class）

- 一個類別包含：
  - 資料成員（Data member）
  - 成員函數（Member functions）
- 將資料和函數放在一起的動作就稱為封裝（Encapsu-lation）
- 由類別所宣告的變數叫做物件(object)

```
struct Cuboid
{
    int length;
    int width;
    int height;
};

int area(Cuboid r)
{
    return 2 * (r.length * r.width
            + r.width * r.height
            + r.height * r.length);
}

int volumn(Cuboid r)
{
    return r.length * r.width * r.height;
}
```

結構與函數

```
class Cuboid
{
public:
    int length;
    int width;
    int height;

    int area()
    {
        return 2 * (length * width
            + width * height
            + height * length);
    }
    int volumn()
    {
        return length * width * height;
    }
};
```

類別與函數

- **C++物件導向程式設計（Object-Oriented Programming）**是以類別物件為主的程式設計。

宣告類別名稱

- **class 類別名稱**
  {
  private:
  //定義私用成員
  public:
  //定義公用成員
  };

宣告類別名稱

- **範例一**

```
class Employee
{
private:
    //類別私用成員;
public:
    //類別公用成員;
};
```

- **範例二**

```
class Employee
{
    //類別私用成員;
public:
    //類別公用成員;
};
```

類別成員變數及函數

- **範例**

```
class Employee {
    int EmpId;                  //定義private資料成員
    char name[20];              //定義private資料成員
public:
    void inputEmp();            //宣告public成員函數原型
    void outputEmp();           //宣告public成員函數原型
};
```

類別成員函數

- **範例續**

```
void Employee::inputEmp() {  //定義inputEmp成員函數
    cout << "EmpId:" << endl;
    cin >> EmpId;
    cout << "EmpName:" << endl;
    cin >> name;
}
void Employee::outputEmp() {    //定義outputEmp成員函數
    cout << "EmpId:" << EmpId << endl;
    cout << "EmpName:" << name << endl;
}
```

- **傳回型態 類別名稱::函數名稱(參數列)**
  {
  //敘述區
  }

建構子（Constructor）與解構子（Destructor）

建構子的基本概念

- 意義
  - 建構子就像是一個類別中用來製造物件的函數，而製造物件的工作包括宣告一個物件所使用的空間，以及初始值的設定

```
class ACCOUNT {
    ACCOUNT(unsigned long, unsigned int, char *, double); //建構子
    RESULT deposit(unsigned long, unsigned int, double);  //存款
    RESULT drawing(unsigned long, unsigned int, double);  //提款
    ..........
};
```

建構子的特性

- 函數的名稱和類別名稱相同。
- 沒有傳回值。
- 宣告物件時，建構子會被自動呼叫，且其主要內容就是替資料成員設定適當的初始值。

設定物件的初始值

建構子函數中的內容

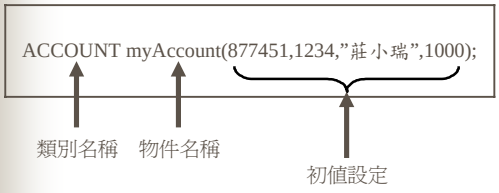
```
ACCOUNT::ACCOUNT(unsigned long id,unsigned
int pw, char *n, double m)
{
   ID_No = id;
   password = pw;
   strcpy ( name , n );
   balance = m;
}
```

設定物件的初始值（續）

精簡的寫法

```
ACCOUNT::ACCOUNT(unsigned long id, unsigned int pw,
char *n, double m):ID_No(id),password(pw),balance(m)
{
   strcpy(name,n); //非一對一資料指定，仍需寫在函數主體內
}
```

宣告物件並給初值的方法



建構子多載

- 一個類別可建立多個建構子，形成建構子多載的情形
- 何時需要建構子多載
  - 需要有不同初始值設定個數的情形下

```
Class ACCOUNT {
Public:
    ACCOUNT();  //無參數的建構子
    ACCOUNT(unsigned long, unsigned int, char *, double);
                                        //有參數的建構子
.....................................
};
```

解構子（Destructor）

- 「解構子」函數的作用和建構子恰好相反，當程式中不再使用某物件時，可以呼叫解構子，將此物件所佔用的記憶體空間，歸還給系統

```
~ACCOUNT();
```

解構子的特性

- 名稱與類別名稱相同
- 沒有傳回值
- 當物件離開其作用範圍時，解構子會被呼叫，其主要目的是釋放物件所佔用的記憶體空間

- 底下是解構子在使用上的特性：
  - 一個類別只能有一個解構子。
  - 解構子的第一個字必須是~，其餘則與該類別的名稱相同。
  - 解構子不含任何參數也不能回傳值。
  - 解構子不可以多載(overload)，也就是說每一個類別只能有一個解構子。
  - 當物件的生命期結束時，或是我們以delete敘述將new敘述配置的物件釋放時，編譯器就會自動呼叫解構子。另外在程式結束前，所有在程式中曾經宣告的物件，都會依照先建構者後解構的順序執行(first-construct-last-destructor)。

建構子(Constructor)與解構子(Destructor)

- 所謂建構式（constructor），就是物件誕生後第一個執行（並且是自動執行）的函式，它
- 的函式名稱必定要與類別名稱相同。
- 相對於建構式，自然就有個解構式（destructor），也就是在物件行將毀滅但未毀滅之前一刻，最後執行（並且是自動執行）的函式，它的函式名稱必定要與類別名稱相同，再在最前面加一個~ 符號。

建構子(Constructor)與解構子(Destructor)

- 當使用者沒有特別為類別宣告建構函數和解構函數時，則編譯器會為每一個類別製作一對沒有引數的建構函數和解構函數
- 如：

```
class account{
    account( );
    …
    ~account( );
};
```

建構子(Constructor)與解構子(Destructor)

- 於全域物件（如本例之*GlobalObject*），程式一開始，其建構式就先被執行（比程式進入點更早）；程式即將結束前其解構式被執行。MFC 程式就有這樣一個全域物件，通常以application object 稱呼之。
- 對於區域物件，當物件誕生時，其建構式被執行；當程式流程將離開該物件的存活範圍（以至於物件將毀滅），其解構式被執行。
- 對於靜態（static）物件，當物件誕生時其建構式被執行；當程式將結束時（此物件因而將遭致毀滅）其解構式才被執行，但比全域物件的解構式早一步執行。
- 對於以*new* 方式產生出來的區域物件，當物件誕生時其建構式被執行。解構式則在物件被*delete* 時執行。

物件的宣告與操作

物件的宣告

- 宣告物件的方法和結構型態的宣告的方法相同，直接使用類別的名稱來宣告

```
ACCOUNT MyAccount;
ACCOUNT Jack, Mary, Diana;     //宣告物件
ACCOUNT AccSet[10];            //宣告物件陣列
……
```

物件的指定

- 將一個物件指定給另一個物件，則會將所有資料成員的內容複製到被指定的物件中，相對應的資料成員

```
呼叫成員函數……
void main()
{
   ACCOUNT JACK(8816001,1111,"JACK",1000);
   ACCOUNT MARY(8816002,2222,"MARY",2000);
   ACCOUNT SAMS=JACK;              //指定JACK物件的資料給SAMS

   cout << "指定前...";
   cout << "\nJACK balance=" << JACK.chk_balance(8816001,1111);
   cout << "\nMARY balance=" << MARY.chk_balance(8816002,2222);
   cout << "\nSAMS balance=" << SAMS.chk_balance(8816001,1111);
   JACK=MARY;              //指定MARY物件的資料給JACK
   cout << "\nJACK=MARY後...";
   cout << "\nJACK balance=" << JACK.chk_balance(8816002,2222);
   cout << "\nMARY balance=" << MARY.chk_balance(8816002,2222);
   cout << "\nSAMS balance=" << SAMS.chk_balance(8816001,1111);
}
```

動態配置物件

- 當物件需要在程式執行階段才能決定要不要宣告時，可以運用動態配置的方法，機動性的宣告物件

```
ACCOUNT *AccPointer = new ACCOUNT;
```

動態物件的初始化

```
ACCOUNT *AccPointer = new ACCOUNT(8816001,1234,"JACK",1000);
```

動態物件的存取

```
AccPointer->deposit(8816001,1234,1000);
```

四種不同的物件生存方式（in stack、in heap、global、local static）

```
把所有可能的物件生存方式及其建構式呼叫時機做個整理,在C++ 中，有四種方法可以產生一個物件。第一種方法是在堆疊（stack）之中產生它：
void MyFunc()
{
CFoo foo; // 在堆疊（stack）中產生foo 物件
...
}
第二種方法是在堆積（heap）之中產生它：
void MyFunc()
{
...
CFoo* pFoo = new CFoo(); // 在堆積（heap）中產生物件
}
第三種方法是產生一個全域物件（同時也必然是個靜態物件）：
CFoo foo; // 在任何函式範圍之外做此動作
第四種方法是產生一個區域靜態物件：
void MyFunc()
{
...
}
```

- 不論任何一種作法，C++ 都會產生一個針對*CFoo* 建構式的呼叫動作。前兩種情況，C++在配置記憶體--來自堆疊（stack）或堆積（heap）-- 之後立刻產生一個隱藏的（你的原始碼中看不出來的）建構式呼叫。第三種情況，由於物件實現於任何「函式活動範圍（function scope）」之外，顯然沒有地方來安置這樣一個建構式呼叫動作。
- 是的，第三種情況（靜態全域物件）的建構式呼叫動作必須靠startup 碼幫忙。startup 碼是什麼？是更早於程式進入點（*main* 或*WinMain*）執行起來的碼，由C++ 編譯器提供，被聯結到你的程式中。startup 碼可能做些像函式庫初始化、行程資訊設立、I/O stream 產生等等動作，以及對static 物件的初始化動作（也就是呼叫其建構式）。

- 當編譯器編譯你的程式，發現一個靜態物件，它會把這個物件加到一個串列之中。更精確地說則是，編譯器不只是加上此靜態物件，它還加上一個指標，指向物件之建構式及其參數（如果有的話）。把控制權交給程式進入點（*main* 或*WinMain*）之前，startup 碼會快速在該串列上移動，呼叫所有登記有案的建構式並使用登記有案的參數，於是就初始化了你的靜態物件。
- 第四種情況（區域靜態物件）相當類似C 語言中的靜態區域變數，只會有一個實體（instance）產生，而且在固定的記憶體上（既不是stack 也不是heap）。它的建構式在控制權第一次移轉到其宣告處（也就是在*MyFunc* 第一次被呼叫）時被呼叫。