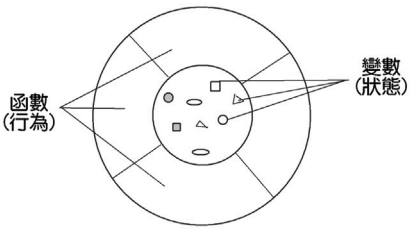
4-1 物件導向程式的設計觀念

物件導向是分析問題和解決問題的方法，是依照我們人類真實的思維來分析和解決問題。程式設計利用這種觀念使程式物件與真實世界有一個很直接的關係，不需要做任何的轉換就很容易理解。

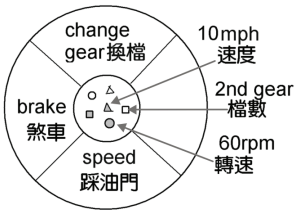
4-1-1 何謂物件(Object)

我們生活四周存在著許多的『物件』，如有形物件的汽車、微波爐等，和無形物件的銀行帳戶、學生成績等。不管哪種物件都可用狀態(State)及行為(Behavior)二種事物來描述，如汽車的車速、檔數是狀態的描述，加速(減速)、換檔是行為的描述。銀行帳戶的餘額、利息是狀態的描述，存款(提款)、結清是行為的描述。

軟體上的物件是一組變數及其相關函數(或稱方法 method)所組成的軟體單位。變數是記錄物件的狀態(或稱屬性 properties)，函數是描述物件的行為(或稱功能 functions)。



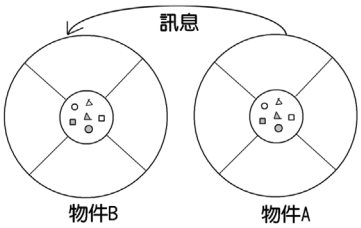
如下圖是一個描述汽車的軟體物件。它有三個記錄汽車屬性的變數及三個相關的函數。



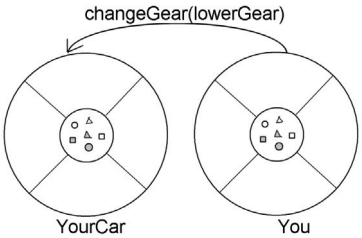
4-1-2 何謂訊息(Message)

一個男生要約班上的女孩出去吃飯，腦筋裡想要送出的資訊應該是：約會的對象(如美蘭)、約會的方式(如吃飯)及吃飯地點(如希爾頓餐廳)。三樣資訊合起來就是一個訊息，接受邀約的女孩接到這個訊息才能有所回應。

也就是說當物件 A 要執行物件 B 內的函數時，物件 A 所丟出來的資訊就是訊息。

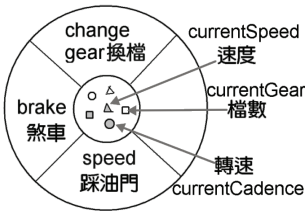


訊息內容必須包括執行者(哪個物件)、執行什麼(物件內哪個函數)以及函數執行所需要的資料。如下圖顯示訊息包括三部份：接受訊息物件的名稱(如：YourCar)、函數名稱(如：changeGear)及傳給函數的變數(如：lowerGear)。

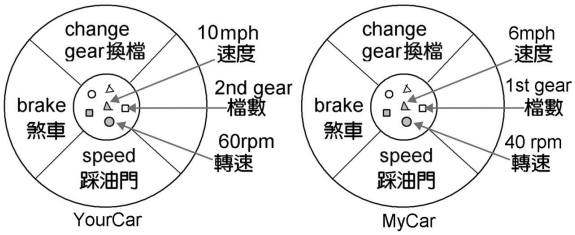


4-1-3 何謂類別(Class)

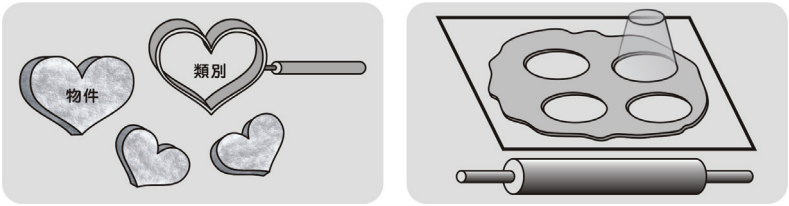
汽車是根據汽車藍圖(Blueprint)製造出來，物件也是根據它的藍圖所產生，而類別就是據以產生物件的藍圖，它描述物件內所有的變數及函數。如前述汽車的類別：



一張汽車藍圖可以製造出許多同款的汽車，一個類別可以產生任何數量的物件。



類別又如同製作餅乾的壓模，一個壓模可以製作出許多相同形狀的餅乾，類別也能產生許多的物件。



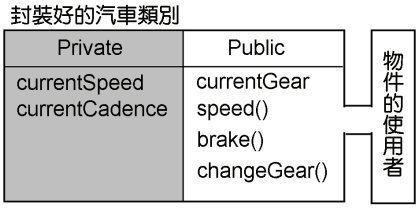
相同類別的不同物件，有著相同的函數，所以一個執行函數的訊息，要指明是哪個物件的哪個函數。如要執行 changeGear，要指明哪部車的 changeGear。

另外變數也是各有一套，分別記錄物件各自的狀態。如不同車子的使用狀態未必相同，你的車開每小時 50 公里，我的車開每小時 40 公里，所以一個存取變數資料，也要指明是哪個物件的哪個變數。

4-1-4 何謂封裝(Encapsulation)

封裝的目的是讓使用者不需知道物件內部是如何運作，就能透過操作介面，完成使用的目的，如日常生活的微波爐。

軟體上所謂的封裝是指將軟體中的變數及使用變數的函數封裝在一起成為一個物件，並依需要設定為多重的控管等級。對 C# 而言，將資料封裝為一個類別 Class，每個類別中的變數或函數可以被設定為隱密的 private 或公開的 public，以確保類別內的變數和函數，能被我們正確的存取和使用，即使我們不了解函數實際運作，只要知道如何正確使用類別的函數就能達到使用目的，而不致發生錯誤。

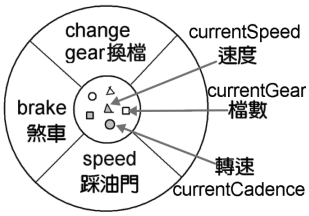


如我們想要由鍵盤輸入資料，只要知道如何使用 Scanner 類別下的函數即可，而無需知道該函數是如何寫的。

4-2 類別的設計(Class Implement)

的使用必須先有物件的存在，而要產生物件必須先有描述(定義)物件的藍圖--即類別，所以我們必須先談類別的設計，再談物件的使用。

4-2-1 類別的定義



將前面的汽車藍圖用程式指令來定義，可寫成：

```
using System;
public class Ex4211 {  // 定義類別名稱，名稱前面一定是 public class
// 以下稱實作變數 instance variables (或稱變數成員 data member)
private int currentSpeed;
private int currentCadence;
public int currentGear;
// 以下稱實作函數 instance method (或稱函數成員 member function)
public void speed(int intSpeed) {
 currentSpeed += intSpeed;
}
public void brake() {
 currentSpeed = 0;
}
public void changeGear(int GearNo) {
 currentGear = GearNo;
}
}
```

定義類別標題

```
public class 類別名稱 {    }        // 名稱前面一定是 public class
```

例如：

```
public class Car {  }
```

定義類別變數與函數

在類別標題後的 block 內定義類別變數與函數，類別的變數與函數稱為 **實作變數(instance variables)** 與 **實作函數(instance method)**，它們的特性是必須透過物件的產生才能生效，不像一般的變數與函數可以直接拿來使用。

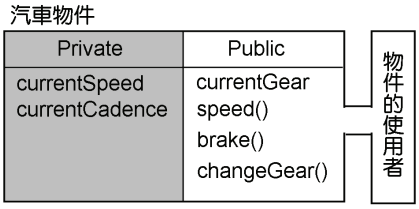
```
擷取方式定義   變數型態   變數名稱;
擷取方式定義   函數回應型態   函數名稱(函數參數);
```

```
private int currentSpeed;
public void changeGear(int GearNo) { }
```

類別中的變數與函數必須根據使用方式與限制給予適當的定義，即給予擷取方式定義(access modifiers)又稱能見度定義(visibility modifiers)或擷取等級(access level)，可分為：

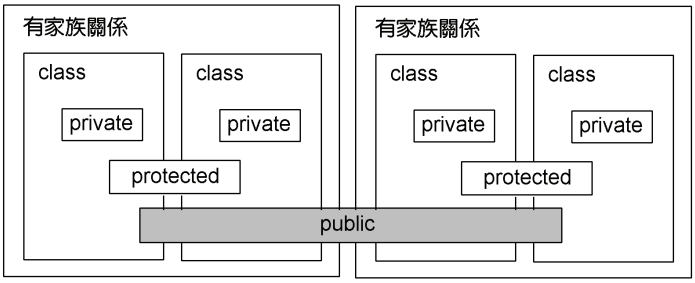
- public、private 及 protected

  『public』，意指物件的使用者可以讀取這些變數或執行這些函數。如果不要物件的使用者讀取這些變數或執行這些函數，將擷取方式定義設為『private』或『protected』。



如上例使用者程式不能直接擷取"currentSpeed"及"currentCadence"內的資料。必須透過 public 的函數來執行，原因是為了物件資料的一致，統一由物件內的函數來讀寫。

至於 protected 則配合類別的繼承使用(請參閱 4-4 類別的繼承)，當類別中的變數或函數限制於本身或繼承才能使用時，則可定義為 protected。



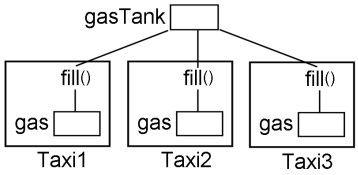
各別 instance 及共用 Class(Static)

如果同一類別的每個物件各有自己的一套變數或函數時，這些變數或函數稱為個別變數(Instance variables)或個別函數(Instance methods)。

如果同一類別的每個物件共用一個變數或函數時，這個變數或函數稱為類別(或靜態)變數或類別(或靜態)函數，它們以"static"關鍵字標註。

如各計程車物件除了個別的油料 gas 外，在車行還有個共同的油料庫 gasTank，各計程車加油時各物件的 gas 會增加，但共有的 gasTank 會減少。

```
public class Taxi {
    public static int gasTank;
    private int gas;
    public void fillGas(int qty){
       gas+=qty;      // 加油到計程車油箱
       gasTank-=qty; // 從油料庫取油
    }
}
```



提示 其他部份與基礎篇所介紹的變數定義、函數定義相同，請參閱該部份章節。

4-2-2 UML 在 C# 的使用

UML Unified Modeling Language 的首字縮寫，中譯為『統一塑模語言』。UML 並不是程式語言，而是一種軟體藍圖的表示方式，在 C#是用來描述類別的內容，由 UML 表示圖我們可以概括的了解一個類別的內容。我們經常被要求根據 UML 圖去完成類別的定義，也有機會對專案進行分析寫成 UML 圖交給程式設計師設計程式，所以 UML 圖是軟體開發同仁間溝通的一種方式。

- 如汽車類別的 UML 寫成：

Car
- currentSpeed: int +currentGear: int - currentCadence: int
+speed(intSpeed: int) +break():void +changeGear(GearNo int):void

第一格表類別名稱，第二格描述變數，第三格描述函數。

- ★『+』表該變數或函數為 public，『-』表該變數或函數為 private。
- ★『()』裡面表輸入的變數名稱與型態。
- ★『:』後面表輸出的資料型態。

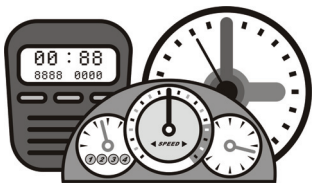
- 下圖是字串類別 String Class 的 UML，可以清楚的了解到它的內容：

String
− value − count
+ String() + String(in s : String) + length() + charAt() + indexOf() + valueOf(in n : int) : String + valueOf(in d : double) : String + charAt(in n : int) : char + equals(in o : Object) : boolean + indexOf(in ch : int) : int + indexOf(in ch : int, in start : int) : int + indexOf(in s : String) : int + indexOf(in s : String, in start : int) : int + substring(in strt : int) : String + substring(in strt : int, in end : int) : String

4-2-3 實例介紹

Time Class 的範例。

問題描述



假設我們要寫個程式記錄徑賽選手的成績，我們把目標放在設計一個新的資料型態–暫時稱它為 CTime。為了比賽需要它必須有時、分、秒的資料記錄。一旦這個資料型態設計好我們可以用在任何需要記錄時間的程式。

首先我們想：物件的使用者對這個 CTime 物件會有哪些動作要求？

- 設定時間
- 讀取時間，細分為時、分、秒的讀取
- 顯示時間

每個動作寫成程式碼就是一個函數，這些函數合起來就是 CTime 類別的公用介面：

```
bool setTime(int hours,int mins,int secs);
int getHours();
int getMins();
int getSecs();
void print();
```

設定時間有可能資料錯誤，函數要做必要的檢查並予回應，所以設定時間函數設有 bool 的回應值。另外在公用介面內必須至少有一個建構函數 Constructor 用以產生物件，由於可以有一個以上方式產生物件，所以建構函數可以不止一個。比如 CTime 物件可以不設初值產生，也可以設定初值產生，因此我們需要在公用介面增加二個建構函數：

```
CTime();
CTime(int hours,int mins,int secs);
```

建構函數是類別內的一個會員函數，當宣告類別的物件時它會自動執行。建構函數是用來設定物件的初始狀態。當建構函數沒有參數傳入時稱為預設的 default 建構函數，否則稱為參數型的建構函數。

建構函數有一些特別規定：

- 函數名稱必須與類別名稱相同。
- 它沒有回應值，也不必設定 void。
- 可有多個建構函數，當然它們的函數名稱都必須與類別名稱一樣，但它們的簽樣(Signature)不能相同。

函數名稱的命名原則雖然與變數名稱的命名原則相同，但變數名稱在有效範圍內不能重複宣告(請參考 2-1-5 變數宣告位置)，否則鬧雙包時電腦無從判斷你要的資料是那一個，而函數的識別是根據它的簽樣 Signature，所以它們的名稱可以相同，但簽樣不能相同，這種名稱相同簽樣不同的情形稱為函數的多重定義 overloading。函數的簽樣包括名稱、參數型態與參數數目，函數簽樣是電腦認定函數的識別。一個類別內可定義多個建構函數就是函數多重定義的應用。

下列幾個函數的名稱相同但簽樣不同，是容許同時存在的：

```
func();
func(int a);
func(double a);
func(int a, int b);
```

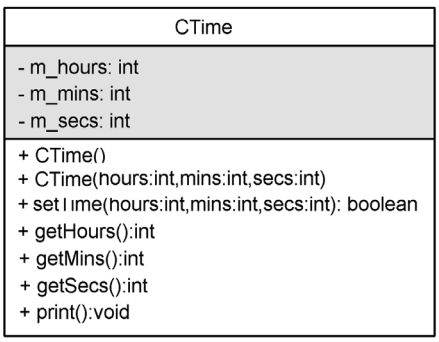
如果不是建構函數雖無必要使用多重定義，但在使用上有其方便性，如在 Convert 類別的 ToString ()函數就有好幾個版本，不同版本將不同型態的資料轉成字串：

```
Convert.ToString(Bool)
Convert.ToString(Byte)
Convert.ToString(Char)
Convert.ToString(Decimal)
Convert.ToString(DateTime)
Convert.ToString(Double)
Convert.ToString(Int32)
      :
```

再回到主題：除了公用介面的函數外，還需要與這些函數相關的資料，也就是時、分、秒的時間記錄，如下面三欄：

```
int m_hours;
int m_mins;
int m_secs;
```

這些公用介面的函數都允許使用者引用，擷取方式定義都定為 public，而物件中變數的存取已透過函數處理，使用者不能直接引用，擷取方式定義定為 private。以 UML 表示如下：



類別的程式設計

建構函數(Constructors)

預設的建構函數寫成：

```
public CTime()
{
    m_hours=0;
    m_mins=0;
    m_secs=0;
}
```

當物件使用者以下列指令產生 CTime 類別的物件時，自動執行此建構函數，執行後 myTime 物件的時間初值會設成 0：

```
CTime myTime = new CTime();
```

參數型的建構函數寫成：

```
CTime(int hours, int mins, int secs) {
   m_hours= hours;
   m_mins= mins;
   m_secs= secs;
}
```

當物件使用者以下列指令產生 CTime 類別的物件時，自動執行此建構函數，執行後 myTime 物件的時、分、秒初值會分別設成 12、45、30：

```
CTime myTime = new CTime(12,45,30);
```

成員函數(member functions)

```
public bool setTime(int hours, int mins, int secs)
{
   if (!isValidTime(hours, mins, secs))   // 請參閱本節(3)補助函數說明
        return false;
   else {
        m_hours=hours;
        m_mins=mins;
        m_secs=secs;
        return true;
```

```
    }
}
```

執行後：如果輸入的資料合理，成員變數的時、分、秒初值會分別設成 hours, mins, secs，並回應 true，否則時間初值不會改變，然後回應 false

```
public int getHours(){
      return m_hours;
}
```

執行後：回應時的設定值 m_hours

```
public int getMins(){
      return m_mins;
}
```

執行後：回應分的設定值 m_mins

```
public int getSecs(){
      return m_secs;
}
```

執行後：回應秒的設定值 m_secs

```
public void print(){
      Console.Write(m_hours+":");
      Console.Write((m_mins<10?"0":"")+m_mins+":");
      Console.WriteLine((m_secs<10?"0":"")+m_secs);
}
```

執行後：時間會顯示在螢幕，格式為 hh:mm:ss

print 函數中為了將時間資料顯示整齊(如將 4:7:1 顯示為 4:07:01)，我們使用了一個 C# 的運算元 - **?:**，其用法如下：

```
(Bool expression ? do if true : do if false)
```

當邏輯判斷式結果為『真』時，以『do if true』為回應值，否則以『do if false』為回應值。應用在這裡是當分、秒只有一位時前面多加個 0，即分、秒小於 10 時，顯示 0，否則不顯示任何字元(即"")。

補助函數(auxiliary function)

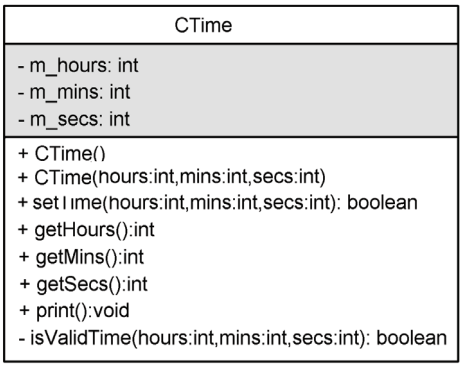
SetTime 函數在設定時間前需要先檢查使用者傳入的時間是否合理，這個檢查是由補助函數 isValidTime 執行，它不屬於類別的成員函數。

```
private bool isValidTime(int hours,int mins,int secs)
{
    return ( hours >=0 && hours < 24 && mins >= 0
           && mins < 60&& secs >= 0 && secs < 60 );
}
```

執行後：如果傳入的時間合理回應 true，否則回應 false。

因為這個函數只能被成員函數或建構函數所引用，不能為物件使用者所引用，所以將它的擷取方式定義設為 private。

增加了 isValidTime 補助函數以後，Ctime 的 UML 表示圖改為：



以上類別程式重新整理如下：

```
public class CTime
{
// data members
     private int m_hours;
     private int m_mins;
     private int m_secs;
// constructors
     public CTime(){
          m_hours=0;
          m_mins=0;
          m_secs=0;
     }

     public CTime(int hours, int mins, int secs) {
          m_hours= hours;
```

```
            m_mins= mins;
            m_secs= secs;
      }
// member methods
       public bool setTime(int hours, int mins, int secs)
      {
            if (!isValidTime(hours, mins, secs))
                  return false;
            else {
                 m_hours=hours;
                 m_mins=mins;
                 m_secs=secs;
                 return true;
            }
      }

      public int getHours(){
             return m_hours;
      }

      public int getMins(){
             return m_mins;
      }

      public int getSecs(){
             return m_secs;
      }

      public void print()
      {
             Console.Write(m_hours+":");
             Console.Write((m_mins<10?"0":"")+m_mins+":");
             Console.WriteLine((m_secs<10?"0":"")+m_secs);
      }

      private bool isValidTime(int hours,int mins,int secs)
      {
              return ( hours >=0 && hours < 24 && mins >= 0
                       && mins < 60&& secs >= 0 && secs < 60 );
      }
}
```

測試程式(Testing Driver)的設計

完成 CTime 類別程式並成功翻譯成 ByteCode 檔後。需要撰寫一個測試程式測試。測試程式是能測試類別內所有函數的腳本程式。這個腳本沒有一定的標準，它可以是任何的劇本，只要所有的函數都能上台表演，並能檢視它們的正確性。例如：

```
using System;
public class Ex4237
{
   public static void Main()
   {
          CTime myTime = new CTime();         // 1
          CTime yourTime = new CTime(1,30,2);      // 2
          yourTime.print();        // 3
          myTime.print();          // 4
          if (myTime.setTime(1,9,2))    // 5
          myTime.print();
   }
}
```

- 2 與 3 是檢視 CTime(int, int, int)與 print()的正確性，必須顯示 1:30:02
- 1 與 4 是檢視 CTime()與 print()的正確性，必須顯示 0:00:00
- 5 是檢視 setTime (int, int, int)與 print()的正確性，必須顯示 1:09:02

提示 類別程式必須經過實作才會生效，不能直接執行，而測試程式是可以直接執行的，所以它有一個執行時的切入點 Main()，這是類別程式所沒有。