C# 3.0 程式語言 物件導向程式設計

講師：林賢達
Peter.lin@imestech.com

課程大綱

- 物件導向基礎概念
- 類別(Class)
- 類別成員(Class Member)
- 封裝(Encapsulation)
- 繼承(Inheritance)

物件導向基礎概念

- 什麼是物件導向(Object Oriented)?
  - 物件導向是將問題領域中涉及的人、事、物予以物件化，透過這些物件之間的訊息傳遞，相互溝通協調，執行某項工作。
- Windows檔案管理：System.IO命名空間
  - Directory類別：建立、移動目錄和列舉子目錄
  - File類別：建立、複製、刪除、移動和開啟檔案
  - StreamReader和StreamWriter類別：以特定的編碼方式讀取檔案內容

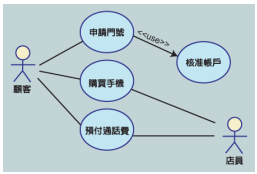
物件導向基礎概念

- 物件導向程式設計(OOP)是以物件導向分析設計(OOAD)產出的圖表為基礎
  - UML(統一建模語言) + 工具：IBM Rational Rose，Visio，SmartDraw，DynamicDraw…
  - 系統分析：功能需求 → 系統規格
    系統設計：系統規格 → 設計圖表
    程式開發：設計圖表 → 定義類別

4

物件導向基礎概念

- 使用案例(Use Case)
  - 使用案例：規劃系統功能的範圍，做什麼?不做什麼



  - 角色：參與系統運作的使用者或外部系統

5

物件導向基礎概念

- 使用案例描述
  - 申請門號描述：(1)**顧客**填寫**申請表**：**身分證號碼**、**姓名**、**出生年月日**、**住址**、**手機號碼**、**申請日期**等。(2)**店員**查核身分證明文件…
- 從案例描述中找出候選的類別
  - 名詞：物件(顧客、店員、出生年月日)或狀態(姓名、住址等)…
  - 動詞：行為，例如填寫申請表、查核身分…

6

類別

- 類別是一組具有相同資料結構和操作的物件的集合。類別對這些具有相同性質的物件的抽象，描述物件的共同特徵。
- 類別定義了一個物件的"狀態"和"行為"
  - 使用類別的私有欄位來描述與保存物件的內部狀態。
  - 使用類別中的方法去定義物件的行為。
- 類別定義了一個範本，使用此範本可以建立一個或多個物件

7

宣告類別

- 使用class關鍵字宣告類別

```
存取修飾詞 class 類別名稱 : object {//存取級別、類別名稱、父類別
        //類別成員
}       //類別主體 class{}
```

- 類別成員
  - 資料成員：常數，欄位和屬性
  - 方法成員：方法、事件，建構函式和解構函式

8

實體化(Instatiation)

- 使用類別來建立物件的過程
  - 使用new關鍵字

```
Car myCar = new Car();
```

  - 沒有可存取的建構子：使用工廠模式建立物件

```
FileStream fs = File.Create("C:\123.txt") ;
```

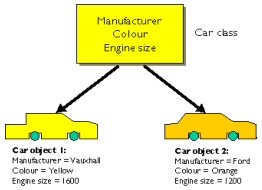
  - 使用反射(Reflection)

```
Type t= Type.GetType("Car") ;
…
```

9

類別 VS 物件

- 類別是在設計階段時完成，物件是類別在執行階段的實體，佔用記憶體空間且具有生命週期



類別是物件範本

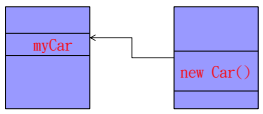
物件是類別的執行實體具有各自的狀態

物件 VS 物件變數(物件參考)

- 物件變數儲存的是物件的參考位址，而非物件本身

```
Car  myCar ;              //物件變數

myCar = new Car() ;       //物件
```



堆疊(Stack)　　堆積(Heap)

類別成員

- 常數(Constant)：代表與類別相關聯的常數值
- 欄位(Field)：類別中的變數
- 方法(Method)：負責執行類別中的計算(處理)
- 屬性(Property)：對類別中欄位值的存取邏輯
- 事件(Event)：用於說明發生了什麼事情
- 建構函式(Constructor)和解構函式(Destructor)：分別用於對類別進行實體化和摧毀物件

封裝(Encapsulation)

- 把屬性和方法封裝在類別中，然後使用類別建立物件(類別的執行實體)，接著使用物件存取屬性和叫用方法。
- 封裝的好處
  - 良好的封裝可以增加內聚性，降低耦合性
  - 類別內部的實現可以自由修改
  - 類別具有清晰的對外開放介面

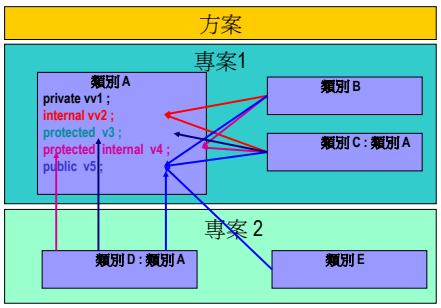
13

封裝

- 使用**存取修飾詞**來達到封裝的目的
  - private：只允許在類別中存取之
  - protected：只允許在類別或子類別中存取之
  - internal：只允許在相同專案中存取之
  - protected internal：只允許在類別、子類別或相同專案中存取之
  - public：任何地方均可存取之

14

封裝(續)



15

屬性

- 屬性提供了對物件內部狀態的存取。
- 屬性實現了物件的封裝性：不直接操作類別的資料內容，而是透過存取器來進行存取。
- 特性
  - 由get/set存取器來控制對內部狀態的存取，當取值時會叫用get函式；賦值時則會叫用set函式，可以在get/set函式提供存取邏輯
  - 內部是函式(定義)，外部則像個欄位(使用上)

16

屬性

- 使用get/set函式存取私有欄位。注意：屬性名稱後面不可以有()

```
存取修飾詞 資料類型 屬性名稱 {
    get {                           //取值
        Return 私有欄位 ;
    }

    set {                           //賦值
        if (value<0) value =0 ;
        私有欄位 = value ;          //value關鍵字表示外部傳入的值
    }
}
```

17

屬性

- 唯讀屬性(ReadOnly)

```
存取修飾詞 資料類型 屬性名稱 {
    get {
        return 私有欄位;
    }
}
```

- 唯寫屬性(WriteOnly)

```
存取修飾詞 資料類型 屬性名稱 {
    set {
        私有欄位=value ;
    }
}
```

18

索引器(Indexer)

- 索引器是一個特殊的屬性，它提供了使用索引方式方便地存取類別的集合資料的方法。
- 索引器通常在類別中表示元素集合

```
public class Sale {
    private Customer[] _customers;

    public Sale() {
        _customers = new Customer[10];
    }
}
```

19

索引器

- 索引器的定義類似屬性
  - 屬性名稱是this，表示目前物件，參數列包含在中括號[]內，不是小括號()

```
public Customer Client[ int index] {
    get {
        return _customers[index];
    }
    set {
        if (value != null) {
            _customers[index] = value;
        }
    }
}
```

20

索引器

- 使用索引器可以像操作陣列的方式來存取物件中的集合

```
Sale Peter = new Sale();

Peter[0] = new Customer("A");
Peter[1] = new Customer("B");
Peter[2] = new Customer("C");

…
```

21

方法

- 方法可用於定義物件的行為
- 無傳回值，不管有無參數，方法名稱後面一定要有小括號()

```
[存取修飾詞] void 方法名稱(參數宣告) {
        …

}
```

- 有傳回值

```
[存取修飾詞] 資料類型 方法名稱(參數宣告) {
        …
}
```

22

建構函式

- 類別的特殊方法，主要用於初始化實體的資料成員。當使用new關鍵字實體化物件時，由.NET CLR自動叫用
- 特性
  - 建構函數名稱必須與類別名稱相同，而且不可有傳回值(不使用void關鍵字)
  - 個數不定(使用重載Overload) ，可以有一個以上的建構函式

23

建構函式

- 定義建構函式。即使沒有定義它，編譯器也會自動為類別提供一個預設的建構函式

```
public class 類別名稱 {
    public 類別名稱() {
        // 初始化
        …
    }
}
```

- 當實體化物件時，由.NET CLR叫用

```
類別名稱 物件變數 = new 類別名稱();
```

24

建構函式

- 重載建構函式

```
public 類別名稱(String firstName, String lastName) : base () {
        …
}
```

```
public 類別名稱(String firstName, String lastName)
        : this (firstName, lastName, 0) {
        …
}
```

- 建構函式之間的叫用

使用this關鍵字

25

解構函式

- 建構函數名稱必須是~類別名稱，而且不可有參數和傳回值(不使用void關鍵字)
  - 用於釋放資源
  - 當物件被Garbage Collection回收時，會自動被叫用

```
~類別名稱() {
    // 釋放資源
    …
}
```

26

特性(Attribute)

- 使用中括弧([])提供特殊的資料
- 可用於類別、方法和屬性
- 例如標題、版本、Web服務、組件、安全性和自訂設定…等

```
[WebMethod()]
public void HelloWorld() {
        …
}
```

27

建立物件

- 宣告後初始化(使用new關鍵字)

```
BankAccount  acc ;
acc = new BankAccount() ;
```

- 宣告同時初始化，使用預設建構函式

```
BankAccount  acc  =new BankAccount() ;
```

- 宣告同時初始化，使用重載建構函式

```
BankAccount  acc  =new BankAccount("Peter", "Lin") ;
BankAccount  acc  =new BankAccount("Peter", "Lin",  1000) ;
```

28

摧毀物件

- GC (資源回收器，Garbage Collection)
  - 當記憶體不夠時，GC將回收不再被使用物件佔用的資源
  - 使用 *物件變數* = ***Nothing*** 標示不再被使用
  - 使用GC.Collect()強制回收資源

29

靜態(Static)成員

- 可建立全域變數、全域方法或常數(隱含的靜態成員)
- 靜態成員不屬於任何物件，可以使用類別名稱直接存取

```
double  radius = 10 ;

Double  area = Math.PI * Math.Pow(radius, 2) ;
```

- 若將類別中的某個成員宣告為static，該成員稱為靜態成員。

30

靜態成員

- 靜態成員 VS 實體成員
  - 靜態成員直接由類別存取；實體成員由物件存取。
  - 靜態成員是屬於類別：實體成員則屬於類別的實體—物件。
- 靜態方法為類別所有，因此不用建立類別的實體就可以叫用。例如System.Math類別。

31

靜態資料成員

- 允許一個類別的多個實體共用相同的欄位
- 使用static關鍵字

```
存取修飾詞 class 類別名稱 {
    public static 資料類型 欄位名稱;
    …

}
```

- 直接使用**類別名稱**來存取static欄位

```
類別名稱.欄位名稱 = 值;          //無需建立物件
```

32

靜態方法成員

- 使用static關鍵字

```
存取修飾詞 class 類別名稱 {
    public static 資料類型 方法名稱 (參數宣告) {
        …
    }
}
```

- 使用**類別名稱**來存取static方法成員

```
類別名稱.方法名稱(引數);          //無需建立物件
```

- 靜態方法成員只能存取類別中的靜態資料成員

33

命名空間(Namespace)

- 使用命名空間來組織類別(邏輯的儲存位置)
  - 增強可讀性，例如System.IO
  - 降低類別名稱的衝突

```
namespace 命名空間名稱1 {
    存取修飾詞 class 類別名稱 {
        …
    }
}
```

```
命名空間名稱1.類別名稱
命名空間名稱2.類別名稱
```

- 類別實際的儲存位置是組件(dll)，與命名空間無直接關係

34

命名空間

- 完全引用
  - 從命名空間的最外層開始引用

```
NS1.Class1 c;
```

- 匯入命名空間
  - 在程式碼檔上方使用**using**關鍵字匯入要使用的命名空間

```
using NS1 ;
using Other = NS2 ;        //命名空間的別名(Alias)
Class1 c1;
Other.Class1 c2 ;
```

35

部分類別(Partial Class)

- C# 2.0 可以將類別、結構或介面的定義拆分到一個以上的原始碼檔案中

```
public class BankAccount {
    private double _blance;
    …
}
```

- 在類別宣告前加入partial關鍵字

```
public partial class BankAccount {
    public double WithDraw(double ) {…};
    …
}
```

36

部分類別

- 什麼情況下使用部分類別？
  - 開發大型專案時，將一個類別分佈於多個獨立檔案中可以讓多位程式開發人員同時對該類別進行處理
  - 將視覺化設計器自動建立的原始碼與開發人員編寫的原始碼分開，避免誤動作造成錯誤(Visual Studio 在建立Windows 表單、ASPX 網頁時都使用此方法)

37

部分類別

- 部分類別的限制
  - 同一類別的所有部分都必須定義在同一組件
  - 各個部分必須使用相同的存取修飾詞
  - 如果將任意部分宣告為抽象，則整個類別都被視為抽象類別
  - 如果將任意部分宣告為sealed，則整個類別都被視為sealed
  - 如果將任意部分宣告時指定父類別，則整個類別都將繼承該類別

38

繼承(Inheritance)

- 繼承是使用已存在的類別為基礎建立新類別的技術
- 子類別從父類別獲取public、internal和protected成員(不包含建構函式和解構函式)，而且可以定義自己的成員
- C#只支援單一繼承，子類別有且僅有一個父類別，也就是繼承關係是一對一
- 使用繼承我們可以衍生出無數個子類別，子類別還可以建立自己的子類別

39

繼承

- 為什麼要使用繼承?
  - 原始碼重用，減少撰寫相同的程式碼
  - 設計重用，在父類別定義公開的成員，然後由其衍生新的子類別，子類別可以有自己的成員
- 通常用於表示"is a"的關係，如果A類別是從B類別中繼承而來，即A是B的子類別，則我們可以說："class A is-a class B"

40

繼承

- 使用冒號(:)宣告類別子類別繼承某個父類別

```
存取修飾詞 class 子類別名稱 : 父類別名稱 {

}
```

- 如果沒有明確指定繼承哪一個父類別，則C#預設繼承自object

```
存取修飾詞 class 類別名稱 {

}
```

41

繼承

- 使用abstract宣告的類別必須被繼承，不能被實體化，這種類別又稱為抽象類別
- 使用sealed宣告的類別不能被繼承(防止繼承)

42

繼承：*Demo NumericTextBox*

- 只允許輸入數字的文字方塊
- 步驟
  - 繼承自System.Windows.Forms.TextBox
  - 覆寫OnKeyPress方法，檢查輸入字元

```
if (! Char.IsDigit(e.KeyChar)) {
        e.Handled = true ;
}
```

43

使用base關鍵字

- 在子類別中叫用父類別的方法成員
- 可以叫用public、protected和internal方法

```
存取修飾詞 Class 子類別名稱 : 父類別名稱{
    public 資料類型 方法名稱 (參數宣告) {
        …
        base.方法名稱(參數) ;
    }
}
```

- base不是一個真正的物件

44

課程大綱

- 多形(Polymorphism)
  - 重載(Overload)
  - 覆寫(Overrides)
- 介面(Interface)
- 委派(Delegate)
- 事件(Event)

45

多形(Polymorphism)

- 同一個操作作用在不同的類別實體(物件)，不同的類別有不同的解釋，產生不同的執行結果
- 多個類別具有同名方法，但有不同的操作

樂器可以演奏歌曲，不同的樂器有不同的演奏方式，例如：

鋼琴：用手彈鍵盤
提琴：用手拉琴弓
長笛：用嘴吹口管

46

多形

- 實作方式
  - 類別：使用**重載**(Overload)來實現多形。在編譯時，根據參數類型和傳回類型等資訊來決定執行哪一個操作
  - 繼承：透過子類別**覆寫**(Overrides)父類別的Overridable方法來實作多形。在執行時，才會根據實際情況來決定執行哪一個操作
  - 介面：類別實作在介面所定義的成員。一個類別可實作多個介面

47

重載(Overload)

- 編譯時期的多形性通過重載方法來實現，使用參數類型來確認叫用哪一個方法。
- 在一個類別中的方法可以有多個不同的版本：有相同的函式名稱，但參數的個數、順序和資料類型必須不同

```
public OrderItem  GetOrderItem() {
        return  new OrderItem();
}
```

```
public  OrderItem  GetOrderItem(int  itemID) {
        return  this.OrderItems(itemID) ;
}
```

48

覆寫(Override)

- 執行時期的多形性通過子類別覆寫父類別中的虛擬方法來實現。
- 子類別為了滿足個別需求重新定義某個方法的不同實作，也就是在子類別中覆寫從父類別繼承而來的方法
- 只有虛擬方法和抽象方法才能被覆寫

49

覆寫

- 使用override關鍵字來覆寫父類別的virtual或abstract方法
  - 方法名稱，參數宣告和傳回類型都必須相同
  - 執行時期將確定叫用物件是什麼類別的實體，然後叫用適當的覆寫方法

50

虛擬(Virtual)方法

- 使用virtual關鍵字宣告虛擬方法

```
[存取修飾詞] virtual 資料類型 方法名稱 (參數宣告);
```

- 類別可以定義虛擬方法、虛擬屬性以及虛擬索引子。它的子類別能夠覆寫這些成員，實現類別的多形性。
- 虛擬方法可以有實作

51

抽象(abstract)方法

- 在父類別只定義全部子類別所共用的一般形式，讓每個子類別各自實作其細節，父類別本身必提供這些方法的實作細節。
- 使用abstract關鍵字宣告抽象方法，可以看成是沒有實作的虛擬方法

```
[存取修飾詞] abstract 資料類型 方法名稱(參數宣告);
```

- 繼承抽象類別的子類別，它必須實作抽象類別中的所有抽象方法。

52

抽象方法

- 包含一個或多個抽象方法的類別，必須宣告成抽象類別。

```
[存取修飾詞] abstract class 類別名稱(參數宣告)
{
    //抽象方法
}
```

- 因為抽象類別沒有完整的實作，所以不能建立抽象類別的物件。

53

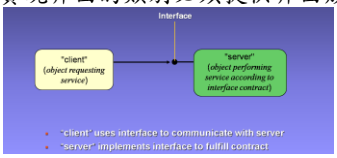
Override VS Overload

	Override	Overload
位置	存在於繼承關係的類別	同一個類別
方法名稱	相同	相同
參數宣告	相同	必須不同
傳回類型	相同	可以不同

54

介面(Interface)

- 介面為類別提供了藍圖(blueprint)，只提供定義，但不含實作部分
- 實現介面的類別必須提供介面成員的實作



- 介面本身可以從多個基底(父)介面衍生而來

55

介面

- 定義方法、屬性和事件的簽章(Signature)
- 使用interface關鍵字

```
interface 介面名稱 {

    資料類型 屬性名稱 { set; get; }

    void 方法名稱(參數宣告);
    資料類型 方法名稱(參數宣告);
}
```

- 介面也可以繼承，使用冒號(:)

56

委派(Delegate)

- 委派是函式的封裝，代表某一類函式。這些函式都具有相同的簽章：相同的參數類型和傳回值類型。
- 委派類型指定它所代表的方法之傳回類型和參數宣告，也就是它可以代表具有相同的參數列和傳回類型的任何一個方法
- 委派是一個類型，當實體化時需提供一個函式參考作為建構函數的參數，此函數必須和委派具有相同的簽章。

57

委派

- 為什麼要使用委派？提供更靈活的方法叫用
  - 用於非同步回呼(Asynchronous Callback)：由於產生委派實體是一個物件，所以可以將其作為參數來傳遞，也可以將其賦值給屬性。此時非同步方法便可以將一個委派作為參數，並且以後可以叫用該委派。
  - 多執行緒程式設計中使用委派來指定啟動一個執行緒時叫用的方法
  - 提供事件處理機制，使用委派指定某個事件的處理函式

58

委派

- 委派的宣告方式與方法的宣告方式相似
  - 使用delegate宣告一個委派物件，宣告時需指定委派所代表函式的參數宣和傳回值類型

```
存取修飾詞 delegate 資料類型 委派類型名稱(參數宣告);
```

  - 委派類型均繼承自Delegate或MulticaseDelegate
- 代表一組具有相同參數宣告與傳回值類型的函式

```
public 資料類型 方法名稱(參數宣告) {
    …
}
```

59

委派

- 使用委派：先宣告委派類型，再產生實體並指定所代表具體函式
  - 使用new關鍵字建立委派的實體，同時在括號內指定一個具體的函式

```
委派類型名稱 委派物件變數;
委派物件變數 = new 委派類型名稱(函式名稱);
```

  - 使用委派實體，並傳入給委派所代表方法的參數來叫用其代表的函式

```
委派物件變數.Invoke (引數);
```

60

委派

- 使用delegate叫用來排序陣列
  - SortAscending遞增排序： 2, 1, 3 ⇨ 1, 2, 3
  - SortDescending遞減排序： 2, 1, 3 ⇨ 3, 2, 1
- 宣告delegate類型

```
delegate void  SortMethod (int[]  intArray) ;
```

61

委派

- 定義SortAscending與SortDescending方法

```
private  void SortAscending (int[] IntArray)
{
      Array.Sort(IntArray) ;
}
```

```
private void  SortDescending (int[] IntArray)
{
      Array.Sort(IntArray) ;
     Array.Reverse(IntArray) ;
}
```

62

委派

- 建立委派實體並指定所代表的具體函式，然後叫用函式

```
private  void SortArray(bool Ascending,  int[]  IntArray)
{
      SortMethod del;
      if (Ascending == true)
         del = New SortMethod(AddressOf SortAscending) ;
      else
         del = New SortMethod(AddressOf SortDescending) ;

      del.Invoke(IntArray) ;
}
```

63

事件

- 使用委派來實作事件處理機制，稱為事件委派
- 事件委派的基本形式

```
public delegate  void  EventHandler (object  sender, object, EventArgs e) ;
```

  - object sender: 來源物件
  - EventArgs e，System.EventArgs類別的實體或衍生類別實體，包含事件的相關訊息

64

事件

- .NET Framework定義了許多的事件委派

```
public delegate void KeyEventHandler(object sender, KeyEventArgs e) ;

public delegate void MouseEventHandler(object sender, MouseEventArgs e) ;
```

- 自訂事件委派

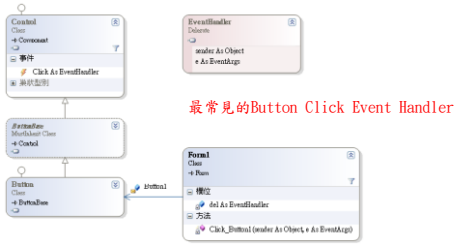
```
public delegate void NameEventHandler(object source,
      NameEventArgs e) ;
```

  - 必須定義NameEventArgs類別

65

事件委派

- .NET Framework的事件委派實例



最常見的Button Click Event Handler

66

事件

- 使用event關鍵字宣告事件名稱，它的類型為事件委派

```
public event NameEventHandler OnNameHandler ;
```

- 觸發事件

```
OnNameListEvent(this , new NameListEventArgs()) ;
```

- 建立物件

```
Protected NameList MyList = New NameList();
```

67

事件

- 事件處理函式，參數宣告必須與事件委派一致

```
private void ShowNameListInfo(object sender, NameListEventArgs e)
{
    …
}
```

- 使用+=動態建立事件與處理函式的關聯

```
NameListEventHandler EventHandlerDelegate = new
    OnNameListEventHandler(ShowNameListInfo);

myList.OnNameListEvent += EventHandlerDelegate;
```

68

事件

- 建立事件與事件處理函式的關聯(續)

```
myList.NameListEvent += new
    OnNameListEventHandler(ShowNameListInfo);
```

- 使用-=動態取消關聯

```
NameListEventHandler EventHandlerDelegate = new
    OnNameListEventHandler(ShowNameListInfo);

myList.OnNameListEvent -= EventHandlerDelegate;
```

```
myList.NameListEvent - = new
    OnNameListEventHandler(ShowNameListInfo);
```

69

Q & A