第六章　Java 的物件導向程式設計 - 基礎篇

本章內容

- 6-1　物件導向程式設計
- 6-2　實作物件導向的程式
- 6-3　建構子的使用
- 6-4　方法多載 (Method Overloading)
- 6-5　類別的繼承 (inheritance)
- 6-6　修飾字

6-1 物件導向程式設計

- 物件導向的設計方式是 Java 的核心，即使是最簡單的 Java 程式，它的寫作方式仍然是物件導向的概念。
- 以往的程式寫作模式是屬於「**程序導向 (Process-Oriented)**」的寫作方式，我們可以用「C」語言為代表。
- 另一種改進的方式是「**物件導向 (Object-Oriented)**」的設計概念，它的特徵是以資料 (或稱為物件 (Object)) 為程式的核心。

6-1-1 抽象化

- 我們會透過「抽象化」來處理或是描述日常生活中複雜的事物。
  - 例如：電視
    - 物件的狀態
    - 物件的行為
- 程式設計中，我們會以「變數」來表示物件的狀態，而以「方法 (method)」來表達物件的行為。
- 以程式設計的角度而言，我們可以說：程式，就是一堆物件 互相送出訊息 (message) ，並告訴彼此要做什麼。而所謂的「送出訊息」就是指呼叫方法。

6-1-2 類別與物件

- 設計物件導向的程式時，我們習慣以「物件」來統稱系統中的資料，而「物件」是來自於「**類別 (Class)**」。
  - 兩者之間的關係如同「依照設計藍圖 (Class) 建構房屋 (Object)」。
- 類別中定義了物件的「**成員 (member)**」。
  - 成員可以包含了物件的屬性（物件資料）和方法（物件行為）。
- 利用「類別」定義的規範產生「物件」的動作稱為「**實體化 (instantiate)**」，每一個由類別產生的物件稱為該類別的「**實體 (instance)**」。
- 在 Java 中，定義出一個新的類別也就是定義了一個新的資料型態。

6-1-3 物件導向的三大特性

- **封裝**
  - 封裝是將類別中的程式碼或是資料保護起來，避免受到外界不當的干擾或是使用。
- **繼承**
  - 繼承的概念來自於階層式的分類。主要的目的是在定義新類別時，不需要重覆的定義相同的成員變數或是成員方法。
- **多型**
  - 「一個介面，多種使用方法」。

6-2 實作物件導向的程式

- Java 的程式離不開類別，每一個 Java 的檔案中至少存在著一個類別。
- 在 Java 中定義類別
  - 類別的定義必需使用關鍵字「**class**」來完成，定義的一般格式如下：

```
class classname{
    type insatnce-variable1;
    type instance-variable2;
    ..........................
    type method1{
    }
    ..........................
}
```

6-2-1 定義類別 …

- 類別的名稱採用「Pascal」的命名方式。
  - 如果類別的名稱是多個單字組合而成時，那每個單字的第一個字母會採用大寫，其餘的字元則是小寫。
- 定義在類別中的變數或是資料稱為「**實體變數(instance variable)**」或稱為「**資料成員(data member)**」，定義在類別中的函式稱為「**方法**」，或是「**成員方法**」。
  - 實體變數和方法的名稱則是採用「Camel」的命名方式。

6-2-2 「is a」和「has a」

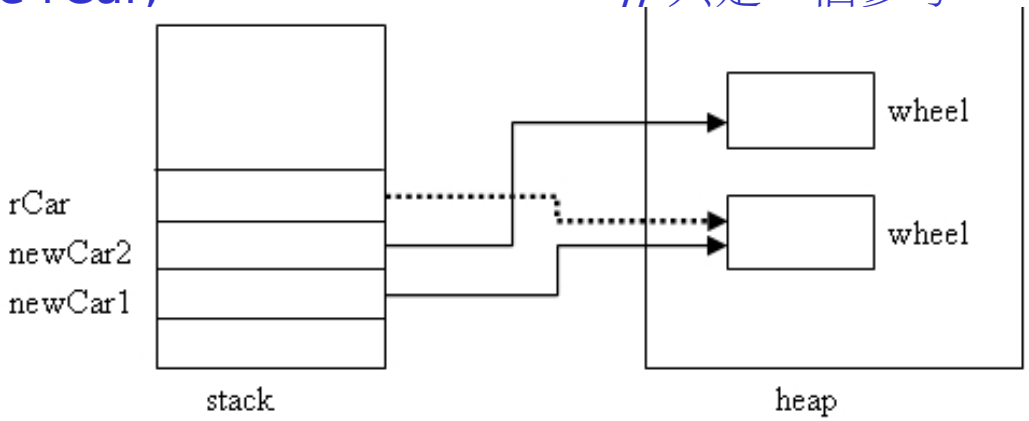
- 在描述類別時，我們時常會以「is a」和「has a」來描述類別的父類別和類別中的資料成員。「is a」描述句通常會導出類別的父類別，而「has a」則是會導出類別中的資料成員。
  - A man **is a** human that **has a** name, **a** father, and **a** mother.
    - 我們可以由以上的敘述中推導出以下的類別設計方式:

```
public class Man extends Human {
   Name theName;
   Father theFather;
   Mother theMother;
}
```

6-2-3 物件的空間配置

- 將類別實體化出多個物件時，每個物件都會有獨立的儲存空間。例如以下的程式碼:

```
class Vehicle{                          // 定義的類別
    int wheel; }                        // 定義一個實體變數
Vehicle newCar1 = new Vehicle();// 實體化一個類別
Vehicle newCar2 = new Vehicle();// 再實體化一個類別
Vehicle rCar;                           // 只是一個參考
```



6-2-4 建立資料存取的方法

- 除了特殊的情況之外，類別中的資料成員不太可能讓外界直接的存取。否則，使用者可能在未經核對的情況下，任意的修改資料的內容。
- 存取資料成員一般是透過類別中設計的方法來達成，我們也可以在方法中加上資料存取的限制。
- 設定資料成員的值

```
public void setWheel(int n){          // 定義方法，設定 wheel 的值
    wheel = n;
}
```

- 取出資料成員的值

```
public int getWheel(){        // 定義方法，傳回 wheel 的值
    return wheel;
}
```

6-2-5 設定資料的存取性

- 類別定義時，還可以使用「**存取修飾字 (access modifier)**」來限制類別或是其成員的存取資格，修飾字元的使用是資料「封裝」的具實體現。使用修飾字元的定義格式如下：

```
[modifier]* class classname{
        [modifier]* type insatnce-variable1;
        [modifier]* type instance-variable2;
        ..........................
        [modifier]* type method1{
        }
        ..........................
}
```

6-2-5 設定資料的存取性…

位置	private	無修飾字元	protected	public
同一類別	是	■	■	■
同一套件中的子類別		■	■	■
同一套件，但不是子類別		■	■	■
不同套件的子類別			■	■
不同套件，也不是子類別				■

6-3 建構子的使用

- 建構子在物件建立時立即會執行，所以，我們可以在建構子中初始化相關的成員。
- 建構子的名稱必需和類別的名稱相同，建構子可以有傳入的參數，但建構子不能自訂回傳的型態 。
  - 因為建構子的回傳型態就是物件本身。
- 定義建構子:

```
class Vehicle{                          // 定義的類別
        private int wheel;                      // 定義一個實體變數
        public Vehicle(){                       // 類別的建構子
                wheel = 4;
        }
}
```

6-3-1 參數化的建構子

- 類別中可以同時定義多個建構子，而您也可以將參數傳入建構子中，自行決定資料成員的值。例如:

```
class Vehicle{                          // 定義的類別
        private int wheel;                      // 定義一個實體變數
        Vehicle(){                      // 類別的建構子
                wheel = 4;}
        Vehicle(int n){                 // 定義有參數的建構子
                wheel = n;}
}
```

-

```
Vehicle newCar1 = new Vehicle();        // 實體化一個類別
Vehicle newCar2 = new Vehicle(6);       // 實體化一個類別，並傳入參數
```

6-3-2 預設建構子的使用

- 如果類別未曾定義建構子，則該類別在實體化時 JVM 會自動建立一個不做任何事的建構子，該建構子如下：

```
Vehicle {}
```

- 但如果類別中自行建立了建構子，JVM 就「不會」再幫類別建立建構子。

6-4 方法多載 (Method Overloading)

- 方法多載，又可以稱為「重載」、「覆載」，主要目的在於實現物件導向中的「多形」的精神。
- 物件導向程式中允許同一個類別中可以定義相同名稱的方法，這種現象就是「**同名異式**」。例如:

```
public void sayHello()
public int sayHello(int n)
```

6-4 方法多載 …

- 當方法多載時，各方法的「**簽名 (signature)** 」不可以相同。
- 所謂的「簽名」是指方法中參數的個數或是型態，但不包含方法回傳值的型態或是方法的封裝型態。
- 一旦類別中發生了此種現象，我們稱該方法被「**多載 (overloaded)** 」。

```
public int addInt (int, i1, int i2) {}
public float addFloat(float f1, float f2)
{}
public long addLonf(long l1, long l2) {}
```

6-4-1 傳值與傳參考呼叫

- 傳值參數是將參數的值直接傳入方法中，參數在方法中的任何運算都不會影響到原來的變數內容。
  - 例如：將基本資料型態的資料傳遞給方法
- 傳參考參數是將變數指向的位址傳遞給方法，因此，該參數在方法中的運算結果會影響到原來的變數。
  - 例如：將物件傳遞給方法

6-5 類別的繼承 (inheritance)

- 繼承的主要目的是在於當我們定義新類別時，可以不需要重覆的定義成員變數或是成員方法，其精神在於可以「重覆」的使用程式碼。
- Java 在繼承的機制上是採用「**單一繼承**」的方式，也就是說，一個類別最多只能直接繼承自另一個類別。
- 如果類別沒有明確的指定其父類別為何，在編譯時，類別都會自動繼承自 Object 類別。
- 在 Java 中，如果某個類別要繼承自另一個類別，在宣告衍生類別時，需要使用「 extends 」關鍵字，語法如下：

[modifier]* class *ClassName* extends *SuperClassName*{ }

6-5-1 類別成員的覆寫 (Override)

- 「**覆寫 (Override)**」又可以被稱為「改寫」、「覆蓋」、「重載」。
- 使用的概念在於子類別中可以改寫父類別中所提供的方法，子類別的物件在執行時，會使用子類別中重新改寫的方法。
- 子類別中也可以定義和父類別相同的資料成員。在子類別中定義和父類別中相同的資料成員時，子類別中的資料成員會「**隱藏 (hide)**」父類別中的資料成員的內容。

6-5-2 物件的多型

- 我們有時候希望某個物件當做是 A 類別來使用，而在另一種場合則是當做是 B 類別來使用，這就是物件的「多形」的概念。
- 物件的角色只能在父類別或是子類別中轉換，而不能轉換成沒有繼承關係的其他類別。 分辨以下的關係:

```
class A{ int I = 10}
class B extends A{ int I = 20}
A a1 = new A();      //a1.I = ?
A a2 = new B();      //a2.I = ?
B b1 = (B) a2;       //b1.I = ?
B b2 = (B) a1;       // 結果 ???
```

6-5-3 子類別中的成員方法會覆寫父類別中的成員方法

- 覆寫的基本原則:
  - 子類別中，方法的名稱、引數的型態、個數必需和父類別中的方法相同。
  - 資料回傳的型態必需相同。
  - final 的方法無法被覆寫。
  - 覆寫時，子類別中的方法的存取層級不能窄於父類別中的方法。
  - 覆寫時，子類別中的方法不能丟出比父類別中的方法更廣泛或是新的例外。

6-5-4 多載與覆寫

	多載	覆寫
同一類別中	方法的名稱相同，但簽名式不同	X
類別產生繼承關係時	X	父類別和子類別中具有相同回傳型態、名稱、簽名式的方法

6-5-5 this 、 super 與遮蔽效應

- 「遮蔽效應」是指在方法中，區域變數會取代全域變數的使用
  - 例如:

```
class Car{
    public int wheel;
    Car(int wheel){
     wheel = wheel; }
}
```

- 使用「 this 」來解決問題

```
class Car{
    public int wheel;
    Car(int wheel){
     this.wheel = wheel;     }
}
```

6-5-5 this、super與遮蔽效應

- **this的使用**
  - 「this」只能在類別中使用，它是用來參考物件實體本身，也就是說：「this」儲存的是物件本身的位址。
  - 因為「this」代表的是物件本身，您也可以在類別中使用「this()」來表示要執行該類別的預設建構子。
  - 「this」常被使用於方法之中，以便區別資料成員和參數 。
  - 「this」也會被用來在建構子中呼叫另一個建構子，以避免重複的撰寫初始資料成員的程式碼 。
    - 使用「this()」的方式呼叫另一個建構子的語法只能使用於建構子中
    - 您不能在一般的方法中使用「this()」
    - 我們不可以在建構子中重複的呼叫「this()」

6-5-5 this、super與遮蔽效應

- **super的使用**
  - 「super」會參考到目前類別的父類別，這裡所謂的父類別是指宣告類別時，「extends」關鍵字之後的類別。使用的語法如下：

```
super.成員;
```

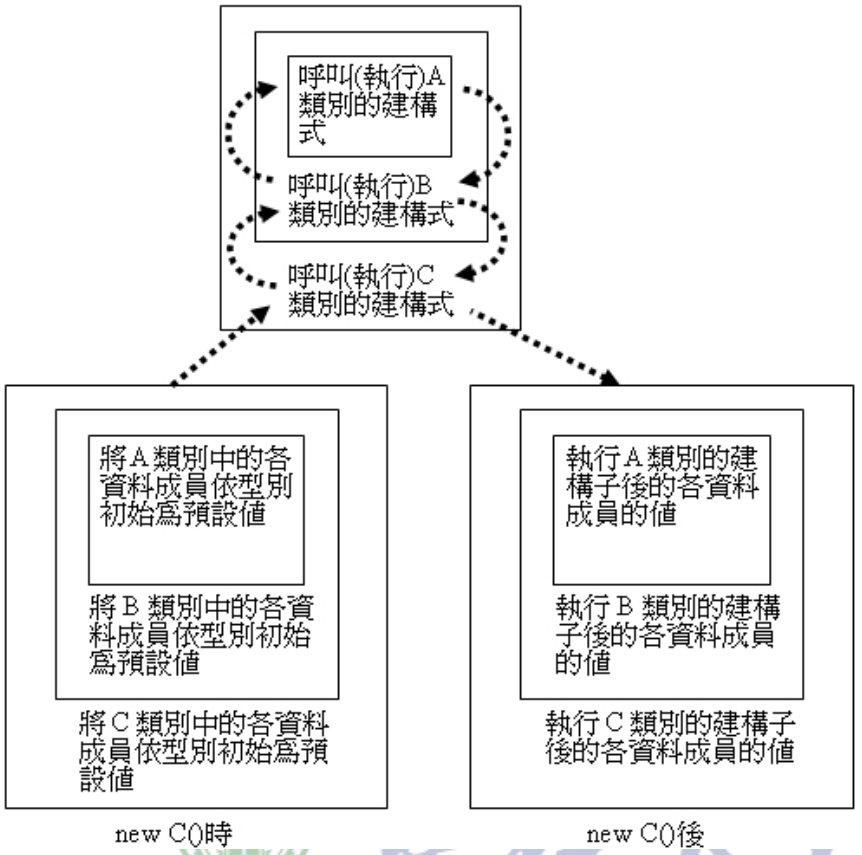
  - 呼叫父類別建構子的敘述寫在子類別建構子的第一行，否則，編譯時會產生錯誤。

6-5-6 建構子的執行

- 如果有以下的繼承關係:

```
class A {...}
class B extends A
{...}
class C extends B
{...}
```

- 參考建構子執行的順序
- 使用 super() 呼叫父類別的建構子的方式



6-6 修飾字

- **final 修飾子**
  - final 修飾子可以套用於類別、資料成員或是成員方法上。
  - 如果方法之中的變數中設定為「final」，程式可以在宣告時同時指定「final」變數的值，或是在第一次使用到「final」變數時指定變數的值。
  - final 變數的值一旦設定之後，變數的值就不可以再更改了。
  - 如果將物件變數設定為「final」，程式中也不能再改變該物件變數的參考，但您可以改變該物件變數所指到的資料內容。
  - 如果是類別中的資料成員設定為「final」，則該資料成員必需在宣告時立即指定該資料成員的值，而且無法再變更。
  - 如果類別使用了 final 修飾子，則該類無法再被繼承。

6-6 修飾字…

- **abstract 修飾子**
  - abstract 修飾子只可以套用於類別及函式上。
  - 宣告為「 abstract 」的類別只能用來被繼承，而不能夠被實體化。
  - 如果類別中的成員方法使用了「 abstract 」修飾子，該成員方法只有名稱、參數內容和回傳型別的定義，不能有實作的部份。
  - 在類別中，只要有一個成員方法使用了「 abstract 」修飾子，則該類別也必需使用「 abstract 」修飾子。但該類別中並不需要將所有的成員方法都宣告成「 abstract 」， abstract 類別中也可以宣告資料成員。

6-6 修飾字…

- **static 修飾子**
  - static 修飾子可以套用於變數、資料成員或是函式上。
  - 宣告為「 static 」的變數或是函式在所屬的類別被載入 時, 即被配置空間。
  - 不管該類別有幾個實體, 「 static 」變數或函式都只有一個 。

6-6 修飾字…

- 使用 **static** 和 **final** 設計列舉資料
  - 程式設計時常會用到「列舉 (Enumeration)」式的資料表示方式。列舉可以視為是一整組相關資料的一覽表 。設計方式如下:

```
lass Season {
        public static final int SPRING = 0;
        public static final int SUMMER = 1;
        public static final int AUTUMN = 2;
        public static final int WINTER = 3;
ublic static final String[] showSeason = {" 春天 ", " 夏天 ", " 秋天 ", " 冬天 "
```

6-6 修飾字…

- 「static」的函式
  - 函式和資料成員都可以被宣告成「static」，您也可以不需要實體化類別就可以使用「static」函式。
    - 「static」函式中可以執行其他的「static」函式，也可以在函式中使用「static」變數。
    - 您不能在「static」函式中直接呼叫非「static」函式，也不能直接使用非「static」變數。
  - Static Initializer

```
static {
    System.out.println("Static 中， age 的值是： " + age);
    age += 1;
}
```

6-6 修飾字…

- **native 修飾子**
  - native 修飾子只能用來修飾函式。Java 中允許您使用 Java 之外的程式語言寫成的函式，該方法統稱為「**原生 (native)** 」函式。native 函式的主體是位於 JVM 之外，因此，使用 native 函式時必需注意到當需要執行 native 函式時，包含 native 函式的類別庫就必需被載入。您可以使用「System.loadLibtary(“函式庫名稱”)」函式來載入類別庫。
- **synchronized 修飾子**
  - synchronized 修飾子可用來控制多執行緒程式中的程式碼。