# 第七章　Java 的物件導向程式設計 - 進階篇

# 本章內容

- 7-1　使用介面 (interface)
- 7-2　建立巢狀類別
- 7-3　記憶體管理與資源回收
- 7-4　Java 的 Object 類別

# 7-1 使用介面 (interface)

- 「介面 (interface)」是用來定義一套標準、規範，它可以讓程式的撰寫有一定的規則可以遵循。
- 介面也可以被視為類別的一種，但不同於一般類別的是：介面中只可以包含常數和抽象方法。
- 介面另一個很重要的特點就是：它可以當作是類別的收集器，以彌補 Java 中不允許多重繼承的性質。
- 介面只提供方法的定義而不實作該方法 。

# 7-1-1 宣告介面

- 宣告介面的方式和宣告類別很相似，但介面中只有屬性或是方法，而沒有建構子。介面的宣告方式如下：

```
[ 存取修飾字元 ] inteface 介面的名稱 [extends 介面名稱 ]{
    [ 存取修飾字元 ] 型別 屬性名稱 = 值 ;
    [ 存取修飾字元 ] 回傳型別 方法名稱 ( 參數 );
}
```

- 例如：

```
inteface Actions {
    public String name = "Some Actions";
    public void canFly();
    public void canRun();
}
```

# 7-1-2 實作介面

- 實作介面時，需要使用「 implements 」關鍵字

```
class Bird extends Animal implements Actions {}
```

- 雖然 Java 中不允許同時繼承多個類別，但卻可以同時實作多個介面。如果需要實作多個介面，您可以在介面的名稱後再使用逗號「 , 」連結第 2 個介面的名稱

```
class Bird implements Actions, Action2, Action3 {}
```

- 當類別實作了介面後，該類別擁有介面中所有的方法和屬性。類別中必需負責實作介面中的「所有」的方法。

# 7-1-3 利用介面模擬多重繼承

- 參考以下的程式碼:

```
interface Action1 {
    public void canRun();
}
interface Action2 {
    public void canFly();
}
interface Action3 {
    public void canSwim();
}
interface AllAction extends Action1, Action2, Action3
    {}
abstract class SuperAnimal implements AllAction{}
class SuperDuck extends SuperAnimal{}
```

# 7-2 建立巢狀類別

- 「**巢狀類別 (Nested Class)**」是指類別中還有其他的類別存在。架構中，外部的類別稱為「**外圍類別 (Enclosing Class)**」，內部的類別稱為「**巢狀類別 (Nested Class)**」。
- 如果外界要使用內部類別，程式中要使用「**外部類別的名稱．內部類別的名稱**」。
- 如果依照巢狀類別的存取特性，巢狀類別可以區分為:
  - 內部類別 **(Inner Class)**
  - 靜態內部類別 **(Static Inner Class)**
  - 方法類別 **(Method Class)**
  - 匿名類別 **(Anonymous Class)**

# 7-2-1 內部類別 (Inner Class)

- Inner Class 可以視為在外部類別中直接宣告的次類別，它的宣告方式和一般類別的宣告方式相同，這是最常使用到的巢狀類別。宣告方式如:

```
class OuterClass2{
    //Other Code Here
    public class InnerClass2{ }
```

- 您可以用 private 、 (default) 、 protected 、 public 等修飾字來定義 Inner Class 。
  - 如果 Inner Class 定義成 private，外圍類別以外的其他類別就無法使用該 Inner Class 了。

# 7-2-1 內部類別 (Inner Class)...

- **資源的存取**
  - 當外圍類別被實體化時，並不會自動的將 Inner Class 也同時實體化。

```
OuterClass2.InnerClass2 o2 = (new OuterClass2()).new InnerClass2();
```

  - Inner Class 中是可以使用外圍類別中的資料成員。
  - 當您建立了內部類別後，內部類別可以直接存取外圍類別的成員。而在外圍類別中，除非是先將內部類別實體化，否則是無法使用內部類別中的成員的。

# 7-2-2 靜態內部類別 (Static Inner Class)

- 在宣告內部類別時，如果使用 static 修飾字來修飾內部類別，則該內部類別稱為「**靜態內部類別 (Static Inner Class)**」。
- 靜態內部類別的記憶體位置是獨立配置的，您不需要先將外圍類別實體化就可以使用該靜態內部類別了。
- 因為如此的特性，靜態內部類別的宣告通常是為了供其他的類別使用。您可以使用以下的方式來實體化靜態的內部類別:

```
外圍類別.內部類別 物件名稱 = new 外圍類別.內部類別
```

# 7-2-2 靜態內部類別 (Static Inner Class)...

- 靜態內部類別無法直接存取外圍類別中的非 static 成員。
  - 如果需要存取外圍類別的非 static 成員，必需先將外圍類別實體化，再透過實體化後的物件來存取。
- 靜態內部類別中可以有 static 成員和非 static 成員。
  - static 成員在載入時同樣也配置了記憶體空間，外界可以直接引用。
  - 非 static 成員則必需要先將靜態內部成員實體化後才能使用。

# 7-2-2 靜態內部類別 (Static Inner Class)...

- 參考下表以了解 static 巢狀類別中的成員是否可以直接存取外圍類別中的成員，及是否可以讓外界直接使用。

| | 可否直接使用外圍類別的 static 成員 | 可否直接使用外圍類別的非 static 成員 | 可否直接讓外部直接存取 |
|---|---|---|---|
| static 巢狀類別的 static 成員 | 可 | 否 | 可 |
| static 巢狀類別的非 static 成員 | 可 | 否 | 否 |

# 7-2-3 方法類別 (Method Class)

- 在方法中宣告的類別稱為「**方法類別 (Method Class)**」。
- 方法類別只能在宣告該類別的方法中使用，使用的概念如同區域變數只能在宣告該變數的區域中使用。

```
public Object showInner(){
    //Other Code Here
    class InnerClass{  //Other Code Here}
```

# 7-2-4 匿名類別 (Anonymous Class)

- 沒有宣告名稱的類別稱為「**匿名類別 (Anonymous Class)**」。
- 宣告的方式是直接在程式中以 new 關鍵字來建立類別實體。
- 由於該類別並沒有名稱，因此它只能使用一次。宣告匿名類別時也可以定義成員及方法，但是，你不可以定義 static 成員，也不能定義類別的建構子。
- 匿名類別的宣告方式如下

```
父類別名稱 物件名稱 = new 父類別名稱(參數){
        //匿名類別中的成員與方法;
};
```

# 7-3 記憶體管理與資源回收

- JVM 採用「**垃圾收集 (Garbage Collection)**」機制不定時的自動回收不再使用的資源，我們不需要自行摧毀物件。
- 「垃圾收集」在程式執行時只會偶爾出現，JVM 並不會只為了摧毀一、二個物件而立即執行「垃圾收集」。
- 我們也可以直接將物件的參考設定成「null」，該物件的資源就有可能會被回收。例如：

```
Vehicle newCar1 = new Vehicle();  // 實體化一個類別
newCar1 = null;
```

# 7-3-1 避開循環參考

- 「**循環參照**」是指兩個物件互相參考。
  - 例如： A 物件是 B 物件的參考，而 B 物件又是 A 物件的參考。
- 如果產生循環參照時，物件佔用的資源就不會被回收，它們會一直佔用著記憶體直到程式結束為止。
- 避免循環參照常用的方法是：覆寫 finalize 方法，並在該方法中將指向其他物件的參考都設定為 null 。

# 7-3-2 finalize() 方法

- 當物件被回收時，該物件的「 finalize() 」方法就會被呼叫，我們可以將清除參考的程式碼寫在此處。「 finalize 」方法的使用方式如下:

```
protected void finalize() {
        // 清除資源的程式碼
}
```

- 使用「 protected 」修飾字是為了避免該類別之外的物件來呼叫「 finalize() 」方法 。

# 7-3-3 自行呼叫 Garbage Collector

- 您可以自行呼叫「System.gc()」方法來啓動 GC，該方法會再呼叫「Rumtime.getRuntime().gc()」方法。
  - 因此，您可以使用上述的任何一個方法來進行資源回收的工作。
- 程式中即使用呼叫了 System.gc() 方法，這項動作只是建議 JVM 盡最大的努力來進行回收不再使用到的資源的工作，JVM 仍然還是不保證垃圾收集機制會立即啓動。
  - GC 啓動時，我們無法判斷物件被回收的順序

# 7-4 Java 的 Object 類別

- Object 類別中所提供的方法如下表:

| 方法 | 作用 |
| --- | --- |
| protected Object clone() | 將物件複製另一個副本 |
| boolean equals(Object obj) | 判斷兩個物件是否相同 |
| protected void finalize() | 為 Garbage Collection 所呼叫，並銷毀該物件 |
| Class getClass() | 產生物件的 runtime class |
| int hashCode() | 傳回物件的 hashCode |
| void notify() | 喚醒物件的 wait pool 中的執行緒 |
| void notifyAll() | 喚醒物件的 wait pool 中的所有的執行緒 |
| String toString() | 傳回描述此物件的字串 |
| void wait() | 讓現行的執行緒被丟到 wait pool 中 |
| void wait(long timeout) | 讓現行的執行緒被丟到 wait pool 中，但最多只在 wait pool 中等待 timeout 毫秒 |

# 7-4-1 clone 方法

- Object 類別提供「clone」方法讓物件可以進行複製的動作。　　但「clone」方法的原始定義為「protected」，您必需要實作您自已的「clone」方法。
- 我們還是可以直接呼叫預設的「clone」方法來進行物件的複製動作。但呼叫，該類別必需要實作「Cloneable」介面，並呼叫父類別的「clone()」方法。
  - 但是，當程式中使用「super.clone()」敘述時會產生「CloneNotSupportedException」的例外物件，您必需要自行捕捉這個例外物件。

# 7-4-2 equals 方法

- Object 類別中提供了「 equals 」方法讓兩個物件可以進行比較的動作。
  - 如果要比較物件的內容是否相同，您要使用「 equals 」方法，如果要比較物件是否是同一個，則要使用「 == 」邏輯運算子。
- Object 類別提供的「 equals 」方法並不會判斷自行定義的類別物件的內容。將 Object 中的 equals 方法用於自定的類別物件時，只會得到 false 的結果。因此，您必需要在您自定的類別中改寫 Object 類別提供的 equals 方法。

# 7-4-3 toString 方法

- Object 類別提供「 toString 」方法來傳回代表這個物件的字串。
- 如果您未曾改寫而是直接使用原先定義在 Object 類別中的「 toString 」方法，則執行後，傳回的字串為:

類別名稱 @ 系統代碼