Java Tutorial

Chapter 2　物件導向程式設計

2.1 程序導向 VS 物件導向 (Procedural vs Object-oriented)

在正式介紹物件導向程式設計之前，首先我們要先比較以往程序式程式設計與物件導向的差別，並且說明程序式的設計方法的缺點，如此我們才能知道為何要學習了解物件導向的設計方法。

首先我們先思考，要完成一個程式需要完成的功能是什麼；而要完成這樣的功能需要哪些動作；這些動作需要遵循怎樣的程序來進行。程序導向設計比較偏向一條龍式的邏輯思考，程式的執行靠著 Main 函式主攬大權，執行到需要的運算再呼叫適當的函式進來主程式執行。然而，這種方式雖然有函式可以重複使用，但是其函式卻無法擴充延伸。再者，如此的設計方法在程式規模越來越大之後要進行改動會容易牽一髮動全身而導致難以維護，並且會為了要達到某些功能而讓程式變得太長進而影響到執行速度。因此我們就要借助物件導向的概念來進行程式設計了。物件導向設計可以看作一種程式中包含各種獨立又互相呼叫的”物件”。不若程序式程式設計就像一系列接連的指令，物件導向中的每一個物件都應該能夠接收、處理並且傳達資料給其他物件，因此每個物件都可以被看做一個小型的運作程式。如此一來，在大型的專案設計中，物件導向才能夠簡化程式碼並且更易於維護。

2.2 物件(Object)與類別(Class)

在現實世界中，每樣東西都可以被視為一種物件(Object)，而每個物件都一定會有自己的特徵與型態。更進一步來說，每種東西都有自己所屬的類別(Class)。以狗為例，”狗”可以視為一個類別，而針對每隻狗都會有自己的特徵諸如毛色、品種以及性別等等；而每隻狗更會有自己的行為如吠叫、搖尾巴或是跑來跑去。而程式世界中的物件，一如現實世界中的物體，也都會有自己的特徵與狀態。每個被創造的物件都作為程式的基本單元，將程式和資料封裝其中，以提高功能性以及擴充性。而物件裡的程式可以存取並修改物件相關聯的資料。

類別可以當作同一種物件的總稱，又或可以當作某種物件的藍圖，它定義了一件事物的特徵或行為；而在程式世界中，類別定義包含了資料的形式以及對資料的操作。值得注意的是，類別只會定進行定義，也就是只宣告該類別底下會有什麼特徵以及行為，並不會讓類別去進行任何動作。以下持續以狗的例子來介紹類別：

```
Class Dog
start
    breed
    hairColor
    gender

    barking
    tail_wagging
end
```

*此為虛擬碼(Pseudocode)

從上面的例子可以看到，我們宣告了一個叫做 dog 的類別，而裡面定義了三種特徵：breed, hair color, gender，以及兩種行為：barking, tail wagging，而這些特徵在類別裡我們稱作屬性(field)；而這些行為我們稱為方法(method)；屬性與方法則統稱成員(member)。在宣告完類別之後，我們就要創建屬於這類別的物件。物件身為類別的例項，將會實際體現類別內成員的狀態，而舉例如下：

```
Define Summer as Dog

Summer.breed = Labrador
Summer.hairColor = Gold
Summer.gender = male
Summer.barking(yes)
Summer.tail_wagging(no)
```

*此為虛擬碼(Pseudocode)

我們已經在 dog 的類別裡定義了世界上所有的狗都會有上述的特徵與行為，而 Summer 就是其中的一個實例，這隻叫做 Summer 的狗是屬於拉不拉多品種、男生並有著金色的皮毛，而且正在吠叫但沒有在搖尾巴。宣告物件之後才能實際去定義其類別中各成員的實際情況，並且成為該物件獨一無二的狀態。我們無法讓狗這個"類別"去吠叫，但是可以讓 Summer 這個實際"物件"來吠叫。

了解物件與類別之後，JAVA 提供一個整合類別的方法，稱為套件(Package)。在想要放在同一個套件底下的類別檔案裡的開頭都加上

```
package packName;
```

並且將這些類別檔案都放在同一個資料夾底下即可。而在需要調用同一個套件底下的不同類別，只需要在開頭指定套件就可以自由調用該套件底下的所有類別。

2.3 物件導向的三大特色

在了解物件與類別之後，接下來我們要來介紹物件導向的三大特色，同時也是在建立類別物件時的準則以及規則。

(1) 封裝 (Encapsulation)

第一個重要特色就是封裝性。試想如果程式允許進行以下表示：

```
summer.age = 5;
```

這就表示程式在任何地方都可以直接對此物件的方法進行存取或設定物件資料，因此若是不小心變成這樣：

```
summer.age = -5;
```

程式執行就會出現奇怪結果，更甚者有可能會直接讓程式無法執行。

為了避免以上錯誤出現，類別中的屬性都應該進行所謂的封裝。封裝就是利用存取權限的設定來指定屬性及方法的存取權限，並且利用特定的存取方法來進行讀寫。在 JAVA 中，指定存取權限的關鍵字共有四種：Public, Private, Protected, (no modifier)。

a. Public
   在一個類別當中，當一個成員被設定為 public 之後，就意味著該成員會向所有人公開，每個外部的人都可以直接透過物件呼叫使用該成員。

b. Private
   若一個成員被設定為 private，則只有該類別物件中可以被使用操作，外部的任何呼叫都無法接觸到該成員。

c. Protected
   此種成員較為特殊，當成員被宣告為 protected，對外部的任何呼叫來說都會將此成員看作 private，也就是說此成員無法被直接呼叫操作；然而，若是某個類別物件是繼承此類別，則 protected 的成員將會被視為 public 可以直接被使用。然而繼承的概念在下一節才會提到，因此在這邊只需要記得有這種成員可以使用即可。

d. (no modifier)
   當一個成員前面沒有加任何關鍵字的時候，只有自己的類別以及同套件的類別可以看到這個成員。

```
public class Dog{
  private:
    String breed;
    String hairColor;
    boolean gender;

  public:
    String name;
    int age;
}
```

在這個例子中，我們設定了三個 private 成員以及兩個 public 成員。其中的三個 private 成員，在不增加任何 public 函式的情況下要從外部去調用這三個成

員是不可能的，因此我們會增加一些 public 方法來進行調用，例如：

```
public String getBreed(){ return breed };
public String getHairColor(){ return hairColor };
```

然而若是外部的程式需要調用到該物件底下的成員的話，則我們會用到”.”來往下調用：

```
int age;
String breed;
Dog summer = new Dog();
age = summer.age;
breed = summer.getBreed();
```

如此一來我們就可以將 summer 的 age 指派給外部的”age”；另一方面，由於 Summer 的 breed 是 private 成員，所以我們必須調用上面增加的函式來獲得。

從上面的例子可以看到另一件重要的事情，也就是物件的宣告。當我們創建完一個類別之後，要進一步宣告實例的物件就必須要用到前一章也有提到過的”new”關鍵字；如同宣告陣列一樣，我們直接宣告 summer 是 Dog 類別並且以 Dog 類別建立一個物件。

修飾詞	類別	套件	子類別	所有地方
public	Y	Y	Y	Y
protected	Y	Y	Y	N
(no modifier)	Y	Y	N	N
private	Y	N	N	N

(2) 繼承 (Inheritance)

第二項重要特色則是繼承。繼承的機制可以把共通的屬性及方法從一個類別拿出來讓另一個類別獲得並使用。在這邊我們將提供屬性方法的類別稱作父類別(superclass)，而繼承接受父類別屬性方法的稱作子類別(subclass)。

子類別可以繼承所有為 public 及 protected 的成員，至於父類別裡為 private 的成員，則必須要有 public 可存取 private 屬性的方法才可以被使用。子類別也可以定義專屬的屬性及方法；然而若是定義與父類別相同名稱的屬性，會隱藏從父類別繼承下來的屬性，如此的動作並不建議這麼做；而定義與父類別相同名稱的方法，則會改寫(override)父類別的方法。改寫的機制我們會在後面做些許整理介紹。

當一個類別要從別的類別做繼承的時候，我們會使用”extends”的關鍵字來進行操作，舉例如下：

```
public Shiba extends Dog{
  void speak(){
    println("This is a Shiba dog");
  }
}

Shiba summer = new Shiba();
summer.speak();
```

我們做出了一個繼承 Dog 類別的 Shiba 類別，而在裡面增加了一個專屬 Shiba 的新方法，因此除了依然可以調用 Dog 類別中的 name, age 等屬性之外，由 Shiba 類別建立的物件也可以使用 speak 這個我們新建立的方法。

值得注意的是 Java 的子類別只能繼承自單一父類別，但可以實作多個介面(Interface)，就像是共通的規格，而我們也會在往後做介紹。

(3) 多型 (Polymorphism)

最後一個則是多型，我們知道子類別可以建立自己的屬性方法並同時分享某些父類別的功能。多型操作指的是使用同一個操作介面以操作不同的物件。所謂操作介面以 JAVA 而言通常指的就是類別上定義的 public 方法，透過這些方法可以對物件加以操作。我們可以讓不同的兩個子類別繼承同一個父類別，並且在子類別中使用父類別的一個方法但是設定做不同的事，如此一來兩個子類別同樣要做的東西都在父類別裡定義好了，只需要個別在子類別裡面增加自己各別要做的事情，以簡化程式碼的設計並且在更改的時候可以簡化不少步驟。

要舉例解釋多型最好的方法首先要先知道什麼是抽象類別(abstract class)。在 JAVA 中定義類別的時候，我們可以僅宣告方法名稱而不實作當中的邏輯，這樣的方法稱為抽象方法(abstract method)。而當英個類別包含了抽象方法，則該類別稱為抽象類別。由於抽象類別有未實作的方法，所以它不能被用來生成物件，只能被繼承，並在繼承之後實作未完成的方法。當要宣告一個抽象方法與抽象類別的時候，必須使用”abstract”關鍵字來進行宣告，以下將使用抽象類別的例子來介紹並解釋何謂多型：

```
public abstract class Human{
  public void goEat(){ println("I eat everyday") };
  public void goSleep(){println("I sleep 8 hours")};
  public abstract void goToilet();
}

public class Male extends Human{
  public void goToilet(){
   println("Stand Up");
  }
}

public class Female extends Human{
  public void goToilet(){
   println("Sit Down");
  }
}
```

從上面的例子可以知道如何創造一個抽象類別與抽象方法，更多的是，我們可以不用重複兩個類別中重複已實作的方法，而是只需要去實作不一樣動作的方法而已。而這個抽象方法，對於兩個繼承的類別會有不同的實作方式，這種概念就稱為多型，也就是前面說說的用同樣的操作方法但操作不同的物件。

2.4 物件覆寫(Overriding)與多載(Overloading)

(1)物件覆寫

在多型的章節中，我們說到了可以在繼承的子類別裡面去實作同樣方法但不同執行模式的概念，而在這邊我們就要提到其中一個常用的重要手段：物件覆寫(Overriding)。

我們在繼承的章節中有提到物件覆寫；當我們繼承的時候，若是子類別裡想要對父類別的方法重新定義，當我們在子類別裡寫下與父類別同樣的方法但是內部執行的方法不一樣，則往後在呼叫子類別的物件的時候，根據多型的規則，使用到被改動的方法只會是子類別的方法而不會執行父類別所記錄的方法。然而，若是我們改寫過的方法中繼續使用父類別原本的功能，可利用關鍵字"super"對父類別的方法進行呼叫。值得注意的是，若是要使用 super，必須放在方法定義中的第一行。

```
public class Animal{
 void move(){ println("Walking"); };
}

public class Fish extends Animal{
 void move(){ println("Swimming");};
}

public class Bird extends Animal{
 void move(){ println("Flying");};
}

public class Penguin extends Animal{
 void move(){
   super.move();
   println("Flying");
 };
}
```

從上面的例子可以看到我們在 Animal 的類別中定義了 move()的方法：動物都是用走路來進行移動的；然而我們接下來在繼承 Animal 類別的 Fish 與 Bird 中改寫了 move()的方法，讓魚跟鳥在呼叫 move()這個方法的時候顯示的不是走路而是游泳及飛行。而 Penguin 類別中我們雖然改寫了 move()方法，但同時也使用了 super 來呼叫父類別原本使用的功能。

值得一提的是，我們在多型的章節中提到的抽象類別的例子，那也是一種物件覆寫的型態；然而多型並不等於物件覆寫，多型應該被認為一種概念，而物件複寫只是體現了多型精神的其中一種手段而已。

(2)物件多載

物件多載常常與物件覆寫搞混，然而物件多載是在同一個類別中，用同一個名字定義多個方法，並且均使用不同的參數。

```
public int add(int a, int b) {
    //...
}

public double add(double a, int b) {
    //...
}

public double add(int a, double b) {
    //...
}

public double add(double a, double b) {
    //...
}
```

從上面的例子可以看到，方法名稱都是 add，但各自所需的參數都不相同，所謂不相同包括參數的個數、型態以及順序。

多載的意義在於同一個方法可以運用不同的參數列，如此可以讓同樣的工作較有彈性，並不用拘泥於參數的型態以及個數。

2.5 介面(Interface)

我們已經知道一個類別裡面的方法定義了外界如何跟物件互動，因此我們可以說方法創造了一個物件與外界的”介面”。因此，我們使用這個概念來建立介面(Interface)，它可以定義一個類別必須要有什麼方法。簡單來說，在 JAVA 裡，介面就是好幾個沒有寫內容的方法。要創建一個介面的時候，會使用關鍵字”interface”；而當一個類別要使用介面的時侯，就會使用關鍵字”implement”。

```
interface Bicycle{
 void changeGear(int newValue);
 void accelerate(int increment);
 void brake(int decrement);
}
```

```
class MountainBike implements Bicycle{
  int gear;
  int speed;

  void changeGear(int newValue){
     gear = newValue;
  };
  void accelerate(int increment){
     speed += increment;
  };
  void brake(int decrement){
     speed -= decrement;
  };

}
```

然而，介面乍看之下與先前提到的抽象類別有點像，但實際上兩者在應用上是有差別的。繼承某抽象類別的類別必定是該抽象類別的子類別，由於屬於同一個類別，只要父類別中有定義同名方法，就可以透過父類別進行多型操作。但實作某介面的類別並不屬於哪一個類別，一個物件上可以實作多個介面。介面的目的在定義一組可操作的方法，實作某介面的類別必須實作該介面所定義的所有方法。

2.6 其他關鍵字

從這一節開始會介紹幾個較為特殊的關鍵字，這些關鍵字在宣告的時候會讓這些被宣告的變數加上一些不同的特性：

(1) final

在宣告任何物件的時候，只要在前面加上 final 關鍵字，就會使該宣告的物件僅能被設定一次。

a. final + 類別

若在類別前面加上 final 關鍵字則會讓該類別變成無法繼承，這樣可以給予一些安全上的好處。

```
public final class FinalInt{
  int a;
  FinalInt(){}
  void set(int a){
   this.a = a;
  }

}

public class FinalNum extends FinalInt{
  FinalNum(){}
}
```

The type FinalNum cannot subclass the final class

a. final + 方法

而在方法前面加上 final 則是會讓該方法無法被改寫

```
public class FinalTest{
 public void m1(){println("m1 OK");}
 public final void m2(){println("m2 OK");}
}

public class FinalTest2 extends FinalTest{
 public void m2(){println("Try to Override");}
}
```

```
Cannot override the final method from sketch_160710a.FinalTest
```

b. final + 變數

當在變數前面加上 final 關鍵字之後，該變數的值將會不可變，往後呼叫該變數若是要賦予新的值將會造成 error。

```
void setup(){
  final int a = 1;
  a = 3;
  println(a);
}
```

```
The final local variable a cannot be assigned. It must be blank and not using a compound assignment
```

上圖的範例可以看到我們嘗試對一個 final 變數賦予新的值，而在實際跑的時候會在編譯階段就產生 error。另一方面，若 final 變數在宣告時不給予值，而是在後面才要追加賦予，一般來說是可行的，但是若該變數是藉由別的方法或是外部進行賦予，此時就會產生 error 訊息。

```
void setup(){
 final int a;
 a = 4;
 println(a);
}
```

```
4
```

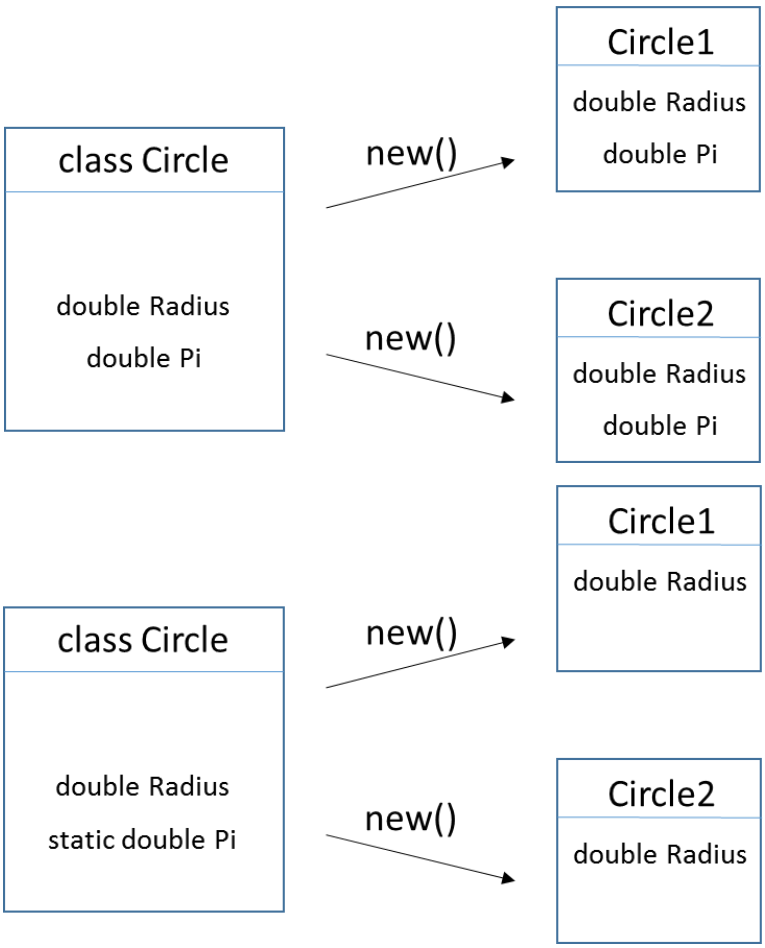
```
public class FinalInt{
  public final int a;

  FinalInt(){}

  void set(int a){
   this.a = a;
  }
}
```

```
The blank final field a may not have been initialized
```

(2) static

另外一個關鍵字則是 static，當使用 static 在一個變數或方法之後，該變數或方法將會屬於類別而不會讓每個物件各自擁有。

舉例來說，一個圓最重要的參數是半徑跟 Pi，然而 Pi 在每一個圓都是固定的，因此我們可以在 Pi 上宣告 static，表示它屬於類別。



被宣告為 static 的成員，將可以把類別名稱作為名稱空間，也就是說我們可以如此獲得 static 成員：

```
public class Circle{
  double radius;
  static final double Pi = 3.14;
}

void setup(){
  println(Circle.Pi);
}
```

```
3.140000104904175
```

同理我們也可以對方法宣告為 static。

然而我們依然可以透過物件來存取 static 成員，但是並不建議如此做來存取 static 成員。

(3) this

當我們在類別中使用任何變數的時候，都是在使用該類別中的變數，除非我們從外部使用方法傳進變數，但若是此種情況發生的時候：

```
public class Tmp{
  private int x, y , z;
  public void set(int x, int y, int z){
   x = x;
   y = y;
   z = z;
```

從上圖可以看出我們宣告了 x, y, z，而我們又使用了 set()方法從外部呼叫三個參數 x, y, z，因此可以看到在 set()方法裡出現了令人混淆的情形： x = x

因此，我們這時候會使用 this 關鍵字來避免產生遮蓋混淆的問題。

```
public class Tmp{
  private int x, y , z;
  public void set(int x, int y, int z){
   this.x = x;
   this.y = y;
   this.z = z;
```

由此我們可以確定 this.x 代表的是 Tmp 類別底下的 x；而後面賦予的 x 則是由外部傳進來的 x。

原則上來說 this 所代表的就是目前使用中的類別，而平常在類別中使用直接呼叫成員其實就隱含了 this 在其中。

2.7 檔案 I/O

當處理到資料的時候，最常使用的其中一種技巧就是讀寫檔案，然而光是要選擇以什麼方式讀寫檔案就有許多種不同方法了。但首先我們要先知道什麼是相對路徑以及絕對路徑：

(1) 路徑

在讀寫檔案的時候，我們必須先給予檔案名稱甚至是路徑才能讓程式知道該去找哪個檔案，因此我們必須寫清楚究竟檔案在什麼地方。

a. 相對路徑

一般來說，我們會以程式檔案所在的資料夾為基準，當檔案與程式黨放在同一個資料夾底下的時候，我們在給予路徑就只需要檔案名稱即可，若是在其子資料夾底下，也只需要從子資料夾開始給予路徑即可。

b. 絕對路徑

然而，若要讀寫的檔案位置在別處，則這時使用絕對路徑就會比較方便：

(2) 讀寫檔案

在這邊我們先介紹兩種比較常見簡單的讀取方法：loadStrings 以及 loadBytes。而寫出檔案則是 saveStrings 以及 saveBytes，由於讀寫的運作模式大同小異，因此我們只介紹讀取檔案的部分。這兩種方法的差異在讀進來的資料型態不同，從這兩個函式可以看出一個是以 String 讀進來而另一個是以 Byte 讀進來。

```
void setup(){
 String[] line = loadStrings("C:/Users/USER/Documents/Processing/test.txt");
 println(line.length);
 println(line[0]);
 println(line[1]);

 byte[] b = loadBytes("C:/Users/USER/Documents/Processing/test.txt");
 println(b.length);
 for(int i = 0; i < b.length; i++)
  print((char)b[i]);
```

```
2
say hi
YOYOYO
14
say hi
YOYOYO
```

從這兩個不同的讀取方式我們可以觀察到幾個不同的點：

1. String 是以一行為一個單位讀取而 Byte 是以一個字元為單位讀取

我們可以看到兩組我們都有一行印出數字的片段，這就是在看讀取進來的陣列有多長，我們可以看到以 String 的方式讀進來的長度只有 2 而 Byte 的方式有 14，這就充份展現出其讀取進來的特性之不同。

2. Byte 讀進來是 ASCII code

在 loadBytes 的片段中，我們在印出資料的時候做了一件不同的事也就是將 b 進行資料型態改變： `(char)b[i]` 這不一樣的地方在於若是在讀進來之後直接進行 print，我們只會印出該字元的 ASCII code 碼而不是我們想看到的字元，因此我們必須將其改成字元的資料型態再印出才能看到我們想要的結果。

從以上結果得知，在讀寫檔案的時候我們必須要先了解我們到底要怎樣處理檔案

再來決定要以什麼樣的型態讀寫檔案才是上策。然而讀取檔案的部分並非只有這兩種方式，尚有一些不同的方式例如叫出資料夾選擇的視窗來直接選擇檔案，又或是特別來處理圖片的 function，這些在未來若是有需要會再特別挑出來講解。

在這一章我們介紹了何謂物件導向設計，並且解說了各式各樣的物件導向特色與概念，並且在最後說明了一些讀寫檔案的方法。然而物件導向並非是一種規則，它比較重要的是在著重理解其設計觀念，如此一來才能掌握較大規模的程式。而在下一章，將會開始介紹如何實作圖形介面(GUI)，我們即將脫離鍵盤輸入與終端機輸出的時代而是進入滑鼠互動與圖像顯示的世界。