Java 使用封裝和建構子

封裝的觀念與做法

1. 封裝的目的

- 封裝的目的 : 達到資訊隱藏，亦即當物件的屬性或方法只提供類別內部使用，不希望給其他物件使用時，將之隱藏
- 封裝 (Encapsulation) 是物件導向程式設計的重要一環
- 封裝可以藉由將物件的屬性 / 欄位設定為 private ，來達到隱藏的效果，使除了物件自己之外，其他物件均無法存取
- 若物件欄位設為 private，且沒有其他配套，則欄位將「完全」無法和外界物件互動，這並不是封裝的目的。封裝目的在提高安全性

  因此可以針對該欄位 (field) 特別建立 :

  1. 取得欄位值的方法 ，又稱 getter() 方法
  2. 設定欄位值的方法，又稱 setter() 方法

2. 欄位封裝的目的與做法

首先建立具備public欄位age的Person()類別

```
public class Person0{
    public int age;  // public的欄位表示完全不設限，物件可以隨意設定年齡，即便是一個不合理的年齡
}

public class PersonTest {
    public static void testPerson(){
        Person0 p = new Person0();
        p.age = 200;
        System.out.println(p.age);
    }

    public static void main(String[] args){
        testPerson();
    }
}
```

將欄位封裝，改為private；並提供public 的 getter()、setter()方法讓物件的欄位可以被存取，其中setter()方法規定設定的年齡僅能界於1-120歲間

```
public class Person1{
    private int age;
    public void setAge(int age){
        if(age >= 1 && age <= 120)  // 年齡設定只能介於 1-120 間外
```

```
            this.age = age;
    }
    public int getAge(){
        return this.age;
    }
}

public class PersonTest{
    public static void testPerson1(){
        Person1 p = new Person1();
        p.age = 200;   // compile error ，因為age是private
        p.setAge(200);
        System.out.println(p.getAge());
        System.out.println(p.age);   // compile error ，因為age是private
    }

    public static void main(String[] args){
        testPerson1();
    }
}
```

3. 方法封裝的目的與做法

4. 方法封裝的進階目的

封裝方法 (method)的要點 :

1. 類別的商業邏輯實作細節，應該盡可能 private。類別設計只留必要的 public 方法供外部物件呼叫，並轉呼叫內部商業邏輯方法。
2. public 方法不具商業邏輯，因此變動機率相對較低；這類方法因為專供其他物件呼叫，也應該盡量減少方法宣告的更動 (如修改參數、回傳型態或方法名稱等) ，避免造成其他類別的修改
3. 因為這類 public 方法是和物件互動唯一的窗口，也稱這些方法是類別的「介面 (interface)」上的 public 方法
4. 未來程式擴充時，僅需修改 private 方法的商業邏輯實作內容，不須異動介面 (interface) 上的 public 方法

- 秘訣 :

  a. 介面 (interface) 概念抽象，不容易理解。可想成「人機介面」:

  一支手機可以有很多的功能，有些適合讓消費者直接使用，有些不適合。對於不適合顯示的功能，手機設計者會使用「人機介面」，如機殼按鍵，或軟體操作等方式來巧妙隱藏。

  b. 對比到類別設計裡「封裝方法」的意義 :

  (1) 把商業邏輯 / 機密封裝，不讓類別外部的方法接觸

  (2) 維持介面上的方法不變，避免呼叫者程式碼頻繁更動 (維持介面上的東西不變，介面下的軟硬體設計方式，是可以完全不同)

使用建構子

1. 使用建構子的時機

```
public class Shirt0{
    char colorCode;
    String description;
    double price;
    int size;

    public char getColorCode(){
        return colorCode;
    }
    public void setColorCode(char colorCode){
        this.colorCode = colorCode;
    }
    public String getDescription (){
        return description;
    }
    public void setDescriptoin(String Description){
        this.description = description;
    }
    public double getPrice(){
        return price;
    }
    public void setPrice(double price){
        this.price = price;
    }
    public int getSize(){
        return size;
    }
    public void setSize(int size){
        this.size = size;
    }
    public void show(){
        System.out.println("price=" + price+ ", size=" + size);
    }
}

public class Shirt0Test {
    public static void main(String[]args){
        Shirt0 s0 = new Shirt0();
        s0.setColorCode('R');
        s0.setDescription("Outdoors Function");
        s0.setPrice(45.12);
        s0.setSize(20);
        s0.show();
    }
}
```

- 這裡有一個隱藏的危機：「物件建立後可以使用setters()方法設定屬性欄位。可是一旦欄位變多時，依賴眾多setters()方法逐欄位設定，會有遺漏的可能」。
- 這種危機，可以使用建構子(constructors)解決問題。

2. 建立建構子

「建構子」和類別成員「方法」的宣告和實作方式都很相似，但有關鍵性的不同：

(1) 建構子名稱必須和類別一樣

(2) 沒有回傳，也不是void

(3) 可以使用多載 (overload)：因為建構子名稱必須和類別名稱相同，所以只能有一種名稱；若有多個建構子同時存在，只能依賴「不同參

數」來辨識

(4) 會在物件建構過程中被 Java 呼叫，故名建構子 (constructor)。主要用來初始化物件的屬性欄位

(5) 語法：

[modifiers]　class　ClassName {　　// 第一個 ClassName 是類別名稱

[modifiers]　ClassName　([arguments]) {　// 第二個 ClassName 是建構子名稱

code_block

}

}

範例：

```
public class Shirt {
    char colorCode ;
    String description ;
    double price ;
    int size ;
    public Shirt (int size, double price) {  //定義建構子
        this.setSize(size);
        this.setPrice(price);
    }
    public void setPrice (double price){
        this.price = price ;
    }
    public void setSize(int size){
        this.size = size;
    }
    public void show(){
        System.out.println("price=" + price+ ",size=" + size);
    }
    // 其他實作內容
}
```

3. 使用建構子建立新物件

語法：　new　ClassName ( ) ;

- 在「new」關鍵字後面的「類別名稱 ()」，其實就是建構子 (constructor)！

- 所謂「實例化 (instantation)」，就是使用 new 關鍵字來呼叫建構子，完成記憶體空間 heap 裡的實例 (instance)建立

  範例 :

```
public class Shirt1Test {
    public static void main(String args[]){
        Shirt1 s11 = new Shirt1();   // error
        Shirt s1 = new Shirt1(20,45.12);  //new 關鍵字呼叫新建構子，必須強制傳入size和price兩個屬性欄位
        s1.show();
        s0.setColorCode('R');
        s0.setDescription("Outdoors Function");
    }
}
```

- 了解 Java 對於建構子的使用 :

  (1) 若程式開發者未在類別中建立建構子，則 Java 將自動提供「預設建構子 (default constructor)」，該建構子為「無參數」，而且

  我們看不到。

  (2) 若開發者已建立類別的新建構子，則 Java將「不」提供「預設建構子」。

  (3) 若已經建立其他建構子，但仍需要使用「無參數」的建構子，就必須自己建立，此時稱為「無參數建構子 (no-args

  constructor)」。

  (4) 建構子允許多個，因此可以藉由多載 (overloading)建立其他參數不同的建構子。建構子之間若要互相呼叫，必須使用「this」關

  鍵字加上參數。因為建構子串聯呼叫，像鍊子(chain)般連結，可以稱為「chaining constructors」。

  範例 :

```
class Employee{
    String name;
    static int age = 25;
    public Employee(String name) {
         this(name,age);   // this 關鍵字
         setName(name);
    }
    public Employee (String name, int age) {
        setName(name);
        setAge(age);
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String show() {
```

```
        return name + " " + age;
    }

    public static void main(String[] args) {
        Employee p1 = new Employee("rrrr");
        Employee p2 = new Employee("kkk",50);
        System.out.println(p1.show());
        System.out.println(p2.show());
    }

}
```