

Module IX

使用封裝與建構子

1. 封裝基本概念
2. 存取修飾子 (Modifier)
 3. 封裝範例
4. 建構子 (Constructor)
 5. 建構子使用
 6. 覆載建構子
 7. Static修飾子

封裝基本概念

- 程式設計師設計類別時，即可指定是否讓其它類別可以存取此類別的資料成員或是方法成員
- Java資料封裝的基本就是**類別**
- 封裝做到資料隱藏與存取限制，當我們將設計好的資料與方法包在物件裡，都必須透過該物件的成員方法來對資料進行存取動作，其它程式無法直接對此物件的資料存取
- Java使用了**private, default, protected**與**public**四種存取修飾子做為封裝權限的等級



封裝做得好，快遞送到老！

存取修飾子 (1/3)

- **public**
 - Visible to the world
 - 所有類別皆能存取
- **protected**
 - Visible to the package and **all subclasses**
 - 同套件底下類別或所有其子類別都可以存取
- **default (預設)**
 - Visible to the package
 - 同套件底下的類別皆可存取
- **private**
 - Visible to the class only
 - 只有該類別內部才可存取

存取修飾子 (2/3)

- 用表格來看存取修飾子

	the Same Class	the Same Package	Subclass	Universe
public	✓	✓	✓	✓
protected	✓	✓	✓	
default	✓	✓		
private	✓			

- 存取修飾子開放等級由大至小：
public → protected → default → private

存取修飾子 (3/3)

- 存取修飾子適用場合：

	類別	實體變數	方法	建構子
public	✓	✓	✓	✓
protected	-	✓	✓	✓
default	✓	✓	✓	✓
private	-	✓	✓	✓

- 注意1. 類別只有public與default兩種修飾子可使用
- 注意2. 方法變數(區域變數)不能使用存取修飾子，因為沒有意義

封裝範例 (1/4)

- 一般而言，設計類別時，建議實體變數(instance variable)設定為private權限，再透過方法來存取資料 (如getter/setter方法)
- 想想看：筆的資料成員 – price，若是沒做好存取限制，會有什麼後果？

```
1 public class Pen {
2     private String brand;
3     private double price;
4
5     public String getBrand() {
6         return brand;
7     }
8     public void setBrand(String brand) {
9         this.brand = brand;
10    }
11    public double getPrice() {
12        return price;
13    }
14    public void setPrice(double price) {
15        this.price = price;
16    }
17 }
18 }
```

```
1 public class PenTest {
2
3     public static void main(String[] args) {
4         Pen p = new Pen();
5         p.setBrand("Mont Blanc");
6         p.setPrice(14800);
7
8         System.out.println(p.getBrand());
9         //輸出結果為Mont Blanc
10        System.out.println(p.getPrice());
11        //輸出結果為14800.0
12
13        // 以下編譯失敗，brand與price都設定為private
14        // 故出現Pen.brand is not visible
15        // 與Pen.price is not visible訊息
16        System.out.println(p.brand);
17        System.out.println(p.price);
18    }
19 }
```

封裝範例 (2/4)

- 未使用封裝有什麼情況？

```
1 public class Pen {
2     //預設牌子為"無牌"
3     public String brand = "No brand";
4     //預設售價為0
5     public double price = 0.0;
6     //使用show方法顯示出牌子與售價
7     public void show() {
8         System.out.println("Brand is:" + brand);
9         System.out.println("Price is:" + price);
10    }
11 }

1 public class PenTest {
2
3     public static void main(String[] args) {
4         Pen p = new Pen();
5         //設定牌子為SKB
6         p.brand = "SKB";
7         //設定售價為100塊(但你沒注意到多了個負號!!!)
8         p.price = -100;
9         //顯示此鋼筆的資訊
10        p.show();
11    }
12 }
```

買一隻鋼筆
老闆要倒貼
100塊！
賣得越多
賠得越多

完

你了!!!!!!

封裝範例 (3/4)

- 解決之道三部曲：

1. private → 2. public getXXX → 3. public setXXX

```
1 public class Pen {  
2     private double price = 0.0;  
3  
4     public double getPrice() {  
5         return price;  
6     }  
7     public void setPrice(double price) {  
8         if (price > 0) {  
9             this.price = price;  
10        } else {  
11            System.out.println("請確認售價設定");  
12        }  
13    }  
14    public void show() {  
15        System.out.println("Price is:" + price);  
16    }  
17 }
```

加入此判斷
問題即可解決

範例：PenGood.java +
PenTestGood.java

封裝範例 (4/4)

- 對於理想的程式碼來說，類別中絕大部份甚至是全部的變數 (Variable) 都會是用 **private** 修飾子
- 這表示它們無法直接被自己所屬以外的類別修改或查詢，只能藉由自己類別中的方法來修改或是查看
- 這些方法應該包含程式碼和運作邏輯以確保變數不會被設定成不適當的值

建構子

- 建構子名稱需與類別名稱相同
- 建構子宣告：

[modifier] constructor_name ([arguments]) {...}

- 一個類別可以有多個建構子
- 一個建構子可以傳入零至多個參數
- 建構子類似方法，可以有存取修飾子
- 建構子**沒有回傳值**，加了回傳值即成為一般方法
- 必須使用 **new** 關鍵字呼叫建構子而產生物件，並同時初始化物件的成員變數 (使用 **new** 關鍵字創建新物件時，將會分配記憶體空間給此物件)
- 注意：**Java**會自動給一個不帶參數的建構子，一旦宣告其他建構子，則**Java**會自動將此預設建構子移除

建構子使用

- **PenConstructor.java**使用建構子範例

```
1 public class Pen {  
2     public String brand;  
3     public double price;  
4  
5     public Pen (String brandXXX, double priceXXX) {  
6         brand = brandXXX;  
7         price = priceXXX;  
8     }  
9  
10    public static void main(String[] args) {  
11        Pen p = new Pen("SKB", 10);  
12        System.out.println(p.brand);  
13        System.out.println(p.price);  
14    }  
15 }
```

建構子使用 (this)

- PenConstructorThis.java 使用建構子範例

```
1 public class Pen {  
2     public String brand;  
3     public double price;  
4  
5     public Pen (String brand, double price) {  
6         this.brand = brand;  
7         this.price = price;  
8     }  
9  
10    public static void main(String[] args) {  
11        Pen p = new Pen("SKB", 10);  
12        System.out.println(p.brand);  
13        System.out.println(p.price);  
14    }  
15 }
```

this 關鍵字

1. 用來存取當前物件
2. 用來呼叫所在類別的建構式

課堂練習

- 產生一個class，名為Animal.java
- 此類別有兩個成員變數分別為age(年紀 - 幾歲 - 型別int)、weight(體重 - 公斤重 - 型別float)
- 有一成員方法名為speak()，用以列印上述兩個值
- 在main()裡透過建構子產生一個Animal，年紀和體重分別為2歲、5.0公斤，並列印此Animal的成員變數值

覆載建構子使用

- 可以藉由 `this` 關鍵字呼叫同類別底下的另一個建構子
- 建構子第一行只要有 `this(...)`，則進行呼叫其它建構子

```
2      public String brand;  
3      public double price;  
4  
5      public Pen (String brand, double price) {  
6          this.brand = brand;  
7          this.price = price;  
8      }  
9  
10     public Pen (double price) {  
11         this("SKB", price);  
12     }  
13  
14     public Pen (String brand) {  
15         this(brand, 10);  
16     }  
17  
18     public Pen () {  
19         this("SKB", 10);  
20     }
```

範例：
PenConstOverload.java

static修飾子 (1/4)

- 實體變數和方法若是宣告為 **static**，則此變數和方法即成為**類別變數** (或稱**靜態變數**)和**類別方法**(或稱**靜態方法**)
- 宣告為**static**的變數和方法，不是由任何此類別的物件單獨擁有，而是由屬於**此類別的所有物件共同擁有**
 - 補充1：實體變數由物件各自獨立維護，彼此不受干擾
 - 補充2：**static**類別變數是屬於類別的變數，但卻可以由該類別所創造(**new**)出來的物件共享共用
 - 補充3：儲存類別變數和方法的記憶體空間為**global**，與儲存物件的記憶體空間是分開的
 - 補充4：使用**static**變數和**static**方法的方式有兩種：
 - **經由類別的任何實體來呼叫 (不好也不鼓勵使用)**
 - **經由類別的名稱來呼叫 (較好的方式)**

static修飾子 (2/4)

- 當類別第一次被載入JVM時，在任何實體被建構之前，靜態的變數與方法就會先被載入，所以：
 - static方法不可使用this
 - static方法不可被覆寫(override)為非static方法
 - 宣告為靜態static方法，**不可以**存取該類別中non-static的變數和方法，**只可以**存取該類別中static的變數和方法
 - 宣告為non-static的方法，**可以**存取該類別中non-static的變數和方法，**也可以**存取該類別中static的變數和方法

```
1 public class Count {  
2     //產品序號  
3     private int serialNumber;  
4     public static int getSerialNumber() {  
5         return serialNumber; //編譯失敗，static方法裡不得存取non-static變數  
6     }  
7 }
```


static修飾子與存取控制 (3/4)


```
1 public class Count {
2     //產品序號
3     private int serialNumber;
4     public int getSerialNumber() {
5         return serialNumber;
6     }
7     //產品數量
8     private static int counter;
9     public static int getTotalCount() {
10         return counter;
11     }
12     //建構元
13     public Count() {
14         counter++;
15         serialNumber = 1000+ counter;
16     }
17 }
```

```
1 public class TestCounter {
2     public static void main(String[] args) {
3         System.out.println("起始數量:"+Count.getTotalCount());
4
5         Count count1 = new Count();
6         System.out.println("累計數量:"+Count.getTotalCount());
7         System.out.println("序號:"+count1.getSerialNumber());
8
9         Count count2 = new Count();
10        System.out.println("累計數量:"+Count.getTotalCount());
11        System.out.println("序號:"+count2.getSerialNumber());
12    }
13 }
```

static修飾子與存取控制 (4/4)

```
1 public class Count2 {
2     //產品序號
3     private int serialNumber;
4     public int getSerialNumber() {
5         return serialNumber;
6     }
7     //產品數量
8     private static int counter;
9     static {
10         counter = 0;
11         System.out.println(
12             "起始數量:" + counter + "\n");
13     }
14     public static int getTotalCount() {
15         return counter;
16     }
17     //建構元
18     public Count2() {
19         counter++;
20         serialNumber = 1000 + counter;
21     }
22 }
```

```
1 public class TestCounter2 {
2     public static void main(String[] args) {
3
4         Count2 count1 = new Count2();
5         System.out.println("累計數量:"+Count2.getTotalCount());
6         System.out.println("序號:"+count1.getSerialNumber());
7
8         Count2 count2 = new Count2();
9         System.out.println("累計數量:"+Count2.getTotalCount());
10        System.out.println("序號:"+count2.getSerialNumber());
11    }
12 }
13 }
```



static程式區塊裡的程式在載入類別時
會先執行一次

static方法簡易範例

- static方法範例：

```
public class TestStaticMethod {  
  
    public static void main(String[] args) {  
        System.out.println("請畫三角形");  
        int count = 9;  
        drawTriangle(count);  
        System.out.println("畫得不錯喔！");  
    }  
  
    public static void drawTriangle(int count) {  
        int i, j;  
        for (i = 1; i <= count; i++) {  
            for (j = 1; j <= i; j++)  
                System.out.println("*");  
            System.out.println();  
        }  
    }  
}
```



請畫三角形

*

**

畫得不錯喔！

章節整理

- 封裝目的在於提昇資料存取安全性與隱藏資料
- 建議實體變數用**private**修飾子，然後再用**getter/setter**方法存取
- 存取修飾子開發等級由大至小：**public**→**protected**→**default**→**private**
- 建構子可視為一種特殊的方法，其目的為初始化物件時使用
- 建構子也可以**Overloading**
- 我們可以使用**this**關鍵字存取當前物件資料與呼叫建構子
- **static**變數或方法(也稱之為靜態變數/靜態方法 或 類別變數/類別方法)
- 瞭解**static**變數、實體變數與方法變數之間的範圍關係