

Encapsulation & Constructor

封裝 & 建構式

Java Fundamental



Content

- ◆ 類別與物件
- ◆ 封裝
- ◆ 建構子
- ◆ 類別成員

Content

- ◆ 類別與物件
- ◆ 封裝
- ◆ 建構子
- ◆ 類別成員

Class & Object

- ◆ 在物件導向的程式語言中，類別是一個不可或缺的機制，它擔負著架構物件的重責大任，它的主要用途是在規劃物件的內部構造，表示物件與物件間的關係，聯絡管道及運作方式。
- ◆ 物件是以類別來建立的。也就是類別定義了物件的架構，然後系統才能根據類別中所定義的架構產生實際物件。
- ◆ 物件導向程式都是以類別為基本單位所組成的。
- ◆ 物件導向程式在執行時的主體是物件，而非類別。

Example : Class & Object(1)

```
1. class C {  
2.     int i;  
3. }  
  
4. public class D {  
5.     public static void main(String[] args) {  
6.         C c1;           // 宣告一個物件 c1  
7.         c1 = new C(); // 產生一個物件 c1  
8.     }  
9. }
```

Example : Class & Object(1)

◆ `int i`

- 類別 `C` 宣告了一整數變數 `i`。
- 註：資料型態包含 `int`, `float`, `double`, `char` 以及 `boolean` 等。

◆ `C c1;`

- 類別 `D` 以類別 `C` 為範本，宣告了一個物件，它的名字為 `c1`。

◆ `c1 = new C();`

- 等號右邊，`new C()` 的意義是：以類別 `C` 為範本產生一個物件。
- 經過指定運算 (等號) 後，這個以類別 `C` 為範本所產生的新物件，它的名字就叫做 `c1`。

◆ 兩者可合寫為 `C c1 = new C();`

◆ 在 `Java` 中，使用物件中的變數和方法方式是：

- 物件名稱.變數名稱
- 物件名稱.方法名稱

Example : Class & Object(2)

```
1. class C {  
2.     int i;  
3. }  
  
4. public class D {  
5.     public static void main(String[] args) {  
6.         C c1 = new C();  
7.         c1.i = 10;  
8.         System.out.println("c1.i = "+c1.i);  
  
9.         c1.i = 20;  
10.        System.out.println("c1.i = "+c1.i);  
11.    }  
12. }
```

輸出結果：

c1.i = 10

c1.i = 20

Example : Class & Object(3)

```
1. class Account {
2.     private int balance;                                // 宣告一帳戶餘額

3.     void clearAccount() { balance = 0; }                // 清空帳戶餘額的 "方法"
4.     void deposit(int m) { balance = balance + m; }      // 存錢的 "方法"
5.     int getBalance() { return balance; }                // 顯示目前餘額的 "方法"
6. }

7. public class E {
8.     public static void main(String[] args) {
9.         Account joe = new Account();                    // 產生一個帳戶 : joe
10.        Account wason = new Account();                   // 產生一個帳戶 : wason
11.        joe.clearAccount();                               // 將 Joe 的帳戶餘額清空
12.        wason.clearAccount();                             // 將 Wason 的帳戶餘額清空
13.        joe.deposit(300);                                  // Joe 存了 300 元
14.        wason.deposit(500);                               // Wason 存了 500 元

15.        System.out.println("Joe has " + joe.getBalance() + " dollars."); // 顯示 Joe 目前的帳戶餘額
16.        System.out.println("Wason has " + wason.getBalance() + " dollars."); // 顯示 Wason 目前的帳戶餘額
17.    }
18. }
```

執行結果：
Joe has 300 dollars.
Wason has 500 dollars.

Example : Class Array

◆ 陣列不僅可以儲存大量基本類型，也可以儲存大量物件。

範例 `ObjectArray.java`

```
1. class Book{
2.     String name;
3.     double price;
4.     String author;
5.     Book(String n, double p, String a){
6.         name = n;
7.         price = p;
8.         author = a;
9.     }
10.    void show(){
11.        System.out.println("書名：" + name);
12.        System.out.println("定價：" + price);
13.        System.out.println("作者：" + author);
14.    }
15. }
16.
17. class ObjectArray{
18.     public static void main(String[] args){
19.         Book[] books = new Book[2];
20.         books[0] = new Book("Java 程式設計", 580.0, "張振風");
21.         books[1] = new Book("JSP 程式設計", 650.0, "黃會紅");
22.         for(Book book : books)
23.             book.show();
24.     }
25. }
```

-----輸出-----

書名：Java 程式設計
定價：580.0
作者：張振風
書名：JSP 程式設計
定價：650.0
作者：黃會紅

Content

- ◆ 類別與物件

- ◆ 封裝

- 封裝的概念
- **Java**語言實作封裝
- 存取權限修飾字

- ◆ 建構子

- ◆ 類別成員

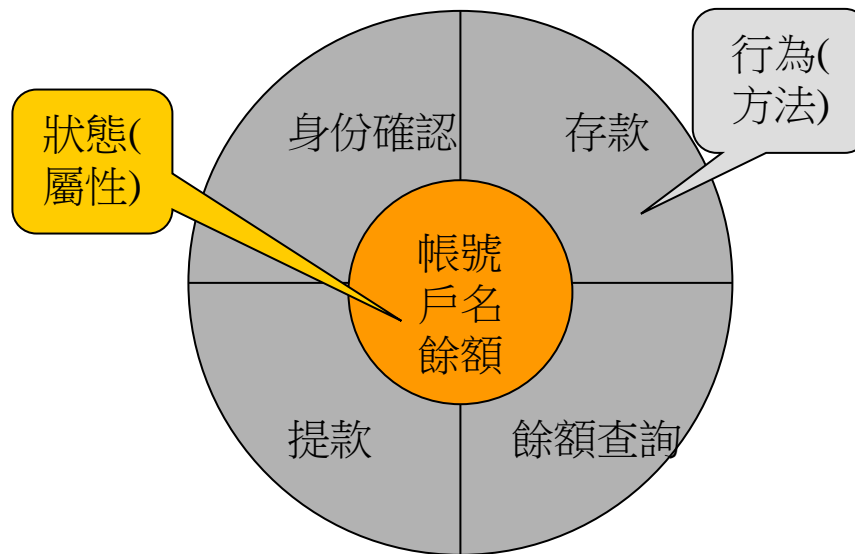
Encapsulation

- ◆ 將資料及使用此資料的所有方法包裝成一個物件。
- ◆ 封裝之特性使物件導向的系統較容易維護。
- ◆ **Java**的封裝是將物件中資源(資料或方法)的存取分為幾個等級，以便來管理如何將物件中某些資源隱藏在物件中，某些資源應該開放給外界使用。

封裝 Encapsulation

◆ 封裝 encapsulation

- **保護**類別中的資料,不讓資料被誤用或破壞
- 隱藏實作的細節,**增加應用程式可維護性**



Encapsulation

◆ 資源(資料或方法)的存取分為四個等級：

➤ **Public** (公開)

- 將類別內部的資源開放給外界使用。

➤ **Private** (私有)

- 資源的所有權已完全屬於該類別所有，**只有在類別內部**才可對其作存取動作。
- 任何外部的存取均會導致錯誤發生。

➤ **Default Access** (預設存取)

- 在所屬的 `package` 中是被視為 `public` 資源；
- 但是在其他的 `package` 中，**則會被視為是 `private` 資源而無法被使用**。

➤ **Protected** (保護)

- 在所屬的 `package` 中是被視為 `public` 資源；
- 但是在其他的 `package` 中，**則只被繼承的子類別使用**。

封裝的方法

◆ 封裝的方法

- 更改屬性為private
- 存取此類別之資料,需使用類別所提供的方法來存取
- 提供存取屬性的方法
 - getxxx()
 - setxxx()

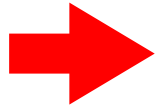
Java 語言實作封裝

```
public class MyDate {  
    public int day;  
    public int month;  
    public int year;  
}
```

```
public class TestMyDate {  
    public static void main(String args[]) {  
        MyDate d = new MyDate();  
  
        d.day = 30;  
        d.month = 2;  
        d.year = 2003; } Invalid date  
  
        System.out.println(d.day + "/" + d.month + "/" + d.year);  
    }  
}
```

Java 語言實作封裝

```
public class MyDate {  
    public int day;  
    public int month;  
    public int year;  
}
```



```
public class MyDate {  
    private int day;  
    private int month;  
    private int year;  
  
    public void setDate(int d, int m, int y) {  
        .....  
        .....  
    }  
  
    public String getDate() {  
        return day + "/" + month + "/" + year;  
    }  
}
```

setter method
setXXX()

getter method
getXXX()

Java 語言實作封裝

```
public class MyDate {  
    private int day;  
    private int month;  
    private int year;  
  
    public void setDate(int d, int m, int y) {  
        .....  
        .....  
    }  
  
    public String getDate() {  
        return day + "/" + month + "/" + year;  
    }  
}
```

```
public class TestMyDate {  
    public static void main(String args[]) {  
        MyDate d = new MyDate();  
  
        d.day = 30;  
        d.month = 2;  
        d.year = 2003; } compile error!  
  
        System.out.println(d.day + "/" + d.month +  
                           "/" + d.year);  
  
        d.setDate(28,2,2003);  
  
        System.out.println("Date: " + d.getDate());  
    }  
}
```

封裝 Shirt 類別

```
01 public class Shirt {  
02     private int shirtID = 0;  
03     private char colorCode = 'G';  
04     private String size = "XL" ;  
05     private double price = 299.00;  
06  
07     public void setColorCode(char c) {  
08         if(c=='R' || c=='G' || c=='Y')  
09             colorCode = c;  
10     }  
11     public double getColorCode ( ) {  
12         return colorCode;  
13     }  
14     public void setSize(String s) {  
15         if(s.equals("S") || s.equals("M") ||  
16             s.equals("L") || s.equals("XL") )  
17             size = s;  
18     }  
19     public String getSize( ) {  
20         return size;  
21     }  
22     public void setPrice(double p) {  
23         if(p>=0.0)  
24             price = p;  
25     }  
26     public double getPrice( ) {  
27         return price;  
28     }  
29 }  
30
```

Example : Encapsulation

```
1.  class C {
2.      private int i;           // 私有資料
3.      public int j;           // 公開資料
4.      int k;                   // 預設存取資料
5.  }
6.  public class D {
7.      public static void main(String[] args) {
8.          C c = new C();
9.          c.i = 5;              // Error! c.i 是 private 資料，禁止存取！
10.         c.j = 10;           // OK!
11.         c.k = 15;           // OK!
12.     }
13. }
```

Example : Encapsulation

```
1. class Account {
2.     private int balance;                // 私有資料
3.     public void clearAccount() { balance = 0; }    // 公開方法
4.     void deposit(int m) { balance = balance + m; } // 預設存取方法
5.     private int getBalance() { return balance; }  // 私有方法
6. }
7. class D {
8.     public static void main(String[] args) {
9.         Account joe = new Account();
10.        joe.clearAccount();                // OK!
11.        joe.deposit(300);                  // OK!
12.        System.out.println("Joe has " + joe.getBalance() + "dollars. "); // ERROR! 使用私有方法
13.    }
14. }
```

Content

- ◆ 類別與物件

- ◆ 封裝

- ◆ 建構子

- 物件屬性欄位初始化

- 建構子

- 預設建構子

- 建構子多載

- ◆ 類別成員

Java 建構子

Shirt
+shirtID: int +colorCode: char +size: String +price: double +description : String
+Shirt (c: char, s: String, p: double, d: String)
+setPrice(p: double) +getPrice () : double +displayInformation ()

```
01 public class Shirt {  
02  
03     public int shirtID = 0;  
04     public char colorCode = 'G';  
05     public String size = "XL" ;  
06     public double price = 299.00;  
07     public String description = "Polo Shirt";  
08
```

```
09     public Shirt(char c, String s, double p, String d) {  
10         colorCode = c;  
11         size = s;  
12         price = p;  
13         description = d;  
14     }  
15
```

建構子

```
16     public void setPrice(double p) {  
17         price = p;  
18     }  
19     public double getPrice ( ) {  
20         return price;  
21     }  
22     public void displayInformation() {  
23         System.out.println("Shirt ID:" + shirtID);  
24         System.out.println("Color:" + colorCode);  
25         System.out.println("Size:" + size);  
26         System.out.println("Price:" + price);  
27     }  
28
```

```
29 }  
30
```

Constructor

- ◆ 在定義類別時，可以使用「建構式」(Constructor)來進行物件的初始化。
- ◆ 「建構式」就是將類別實體化建立成物件時，所執行的方法。會在物件產生之後自動被呼叫，建構物件初始的狀態，因此稱為建構式(建構方法)。
- ◆ 建構式名稱必須與類別名稱相同，建構式不得指定傳回值。
- ◆ 例如：

```
public class SafeArray {  
    // ..  
    public SafeArray() { // 建構方法  
        // ....  
    }  
    public SafeArray(參數列) { //  
        // ....  
    }  
}
```

如果沒有定義任何的建構方法，則編譯器會自動配置一個無參數且沒有陳述內容的建構式。

程式在運行時，會根據配置物件時所指定的引數資料型態等，來決定該使用哪一個建構式新建物件。

建構子 constructor 語法

```
[modifiers] constructor_name([arguments]) {  
    ↪ Accessibility ↪ Same as class_name ↪ Arguments list  
    code_blocks  
}
```

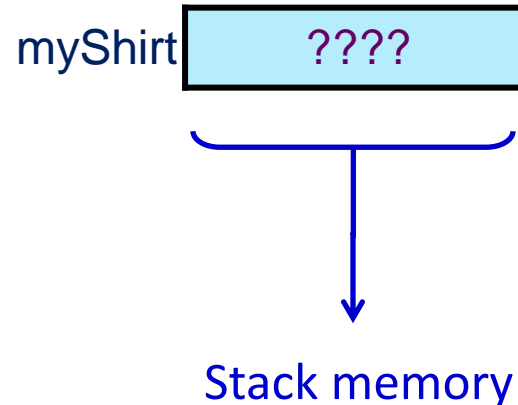
- ◆ 與類別名稱一樣
- ◆ 沒有回傳型態
- ◆ 預設建構子
- ◆ 可以多載(Overloading)

物件建構流程 – 宣告

```
public class Shirt {  
    private int shirtID = 101;  
    private char colorCode = 'R';  
    private double price = 299.0;  
    private String description = "Polo Shirt";  
  
    public Shirt(char c, double p, String d){  
        colorCode = c;  
        price = p;  
        description = d;  
    }  
}
```

Shirt myShirt;

myShirt = new Shirt('G', 199.0 , "T-Shirt");



物件建構流程 – 實體化記憶體配置

```
public class Shirt {  
    private int shirtID = 101;  
    private char colorCode = 'R';  
    private double price = 299.0;  
    private String description = "Polo Shirt";  
  
    public Shirt(char c, double p, String d){  
        colorCode = c;  
        price = p;  
        description = d;  
    }  
}
```

```
Shirt myShirt;  
myShirt = new Shirt ('G', 199.0 , "T-Shirt" );
```

myShirt

????

0	shirtID
' '	colorCode
0.0	price
NULL	description

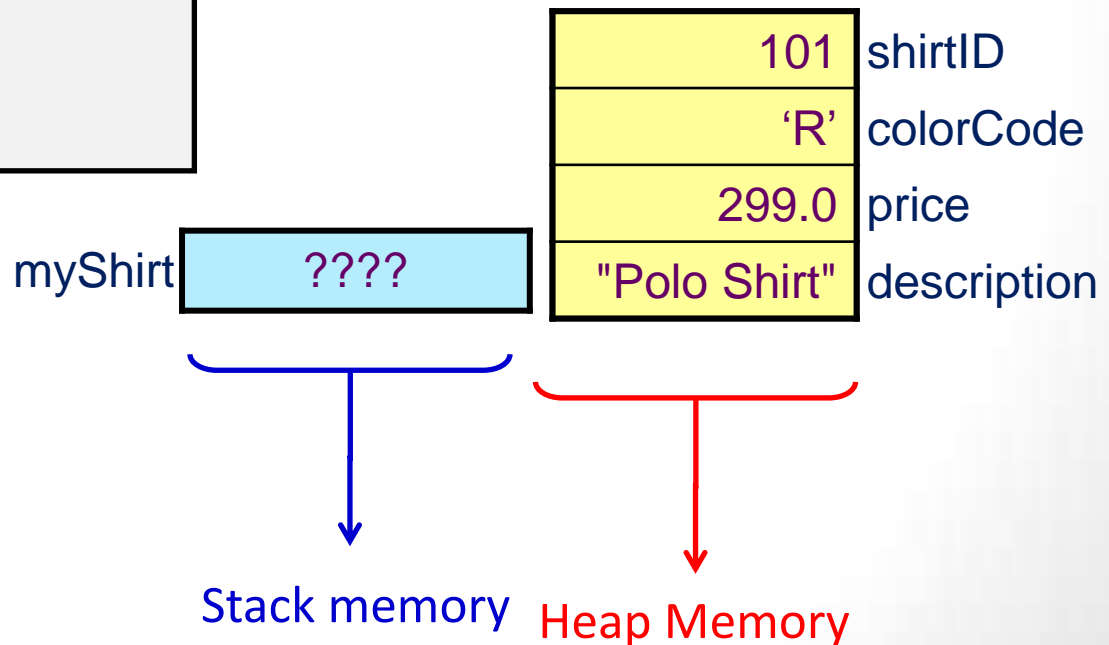
Stack memory

Heap Memory

物件建構流程 – 初始化初始值賦值

```
public class Shirt {  
    private int shirtID = 101;  
    private char colorCode = 'R';  
    private double price = 299.0;  
    private String description = "Polo Shirt";  
  
    public Shirt(char c, double p, String d){  
        colorCode = c;  
        price = p;  
        description = d;  
    }  
}
```

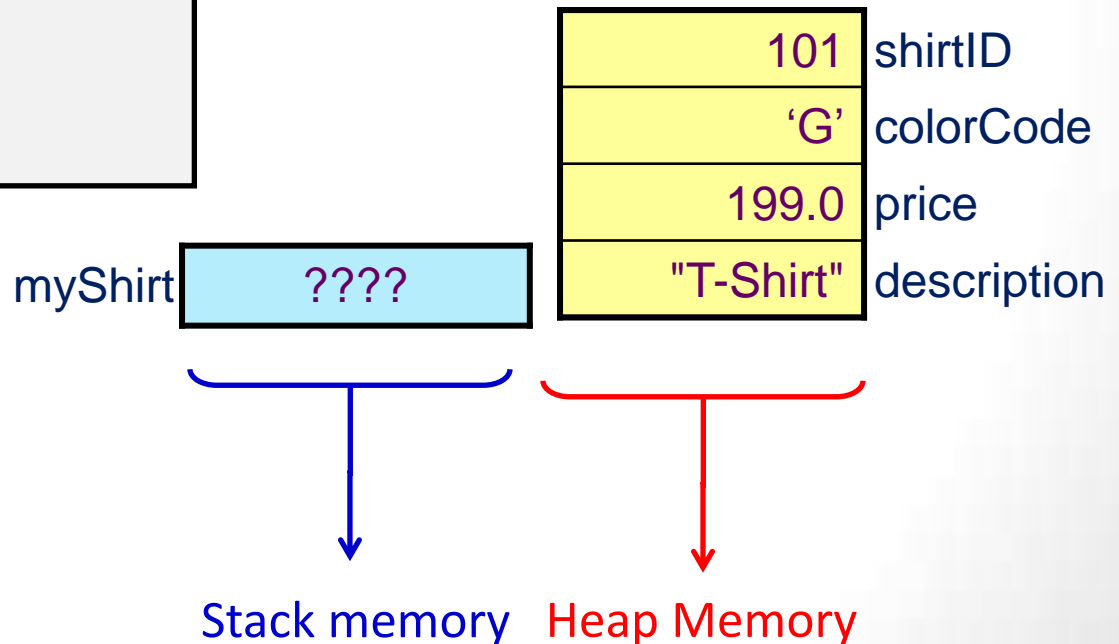
```
Shirt myShirt;  
myShirt = new Shirt ('G', 199.0 , "T-Shirt" );
```



物件建構流程 – 初始化執行建構式

```
public class Shirt {  
    private int shirtID = 101;  
    private char colorCode = 'R';  
    private double price = 299.0;  
    private String description = "Polo Shirt";  
  
    public Shirt(char c, double p, String d){  
        colorCode = c;  
        price = p;  
        description = d;  
    }  
}
```

```
Shirt myShirt;  
myShirt = new Shirt ('G', 199.0 , "T-Shirt" );
```

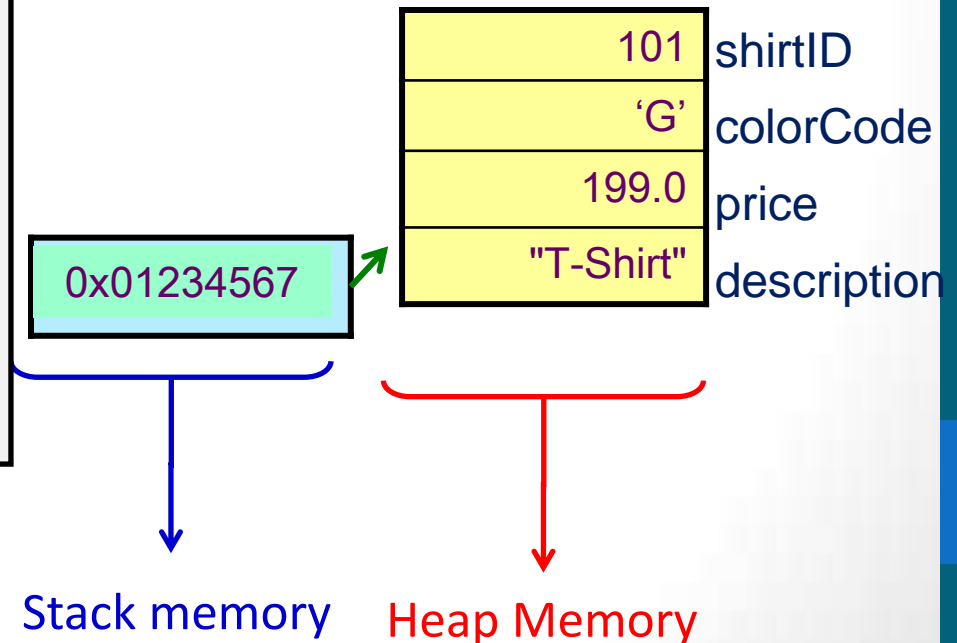


物件建構流程 – 儲存物件參考

```
public class Shirt {  
    private int shirtID = 101;  
    private char colorCode = 'R';  
    private double price = 299.0;  
    private String description = "Polo Shirt";  
  
    public Shirt(char c, double p, String d){  
        colorCode = c;  
        price = p;  
        description = d;  
    }  
}
```

Shirt myShirt;

myShirt = new Shirt('G', 199.0, "T-Shirt");



Constructor

◆ 建構式雖然很像方法，但是有3個不同點：

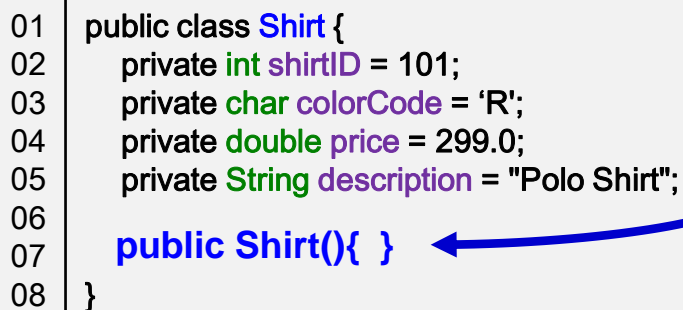
- 呼叫時機不同
- 建構式無回傳值
- 建構式名稱與類別相同

◆ 建構式可分成：

- 預設建構式
- 非預設建構式

預設建構子 Default Constructor

- ◆ 物件裡面一定要有建構子，所以在撰寫class時必須定義該物件的建構子。
- ◆ 程式中若沒有定義建構子，在編譯時期會自動加入，所加入的就稱之為預設建構子；
 - 預設建構子沒有參數列(no arguments)。
 - 除了初始物件變數或繼承時super()的定義外，預設建構子沒有其他的程式敘述(no body statement)。
 - 自行建立後預設建構子即失效。



```
01 public class Shirt {  
02     private int shirtID = 101;  
03     private char colorCode = 'R';  
04     private double price = 299.0;  
05     private String description = "Polo Shirt";  
06  
07     public Shirt(){ }  
08 }
```

```
01 public class TestShirt {  
02     public static void main(String[] args) {  
03         Shirt s = new Shirt();  
04     }  
05 }
```

javac Shirt.java

Constructor

範例 Constructor.java

```
1. class Book{
2.     String name;
3.     double price;
4.     String author;
5.     Book(){ //預設建構式
6.         name = "不詳";
7.         price = 0.0;
8.         author = "不詳";
9.     }
10.    Book(String n, double p, String a){ //非預設建構式
11.        name = n;
12.        price = p;
13.        author = a;
14.    }
15.    void show(){
16.        System.out.println("書名：" + name);
17.        System.out.println("定價：" + price);
18.        System.out.println("作者：" + author);
19.    }
20. }
21.
22. class Constructor{
23.     public static void main(String[] args){
24.         Book book1 = new Book("Java 程式設計", 580.0, "張振風");
25.         book1.show();
26.         Book book2 = new Book();
27.         book2.show();
28.     }
29. }
```

-----輸出-----

書名：Java 程式設計

定價：580.0

作者：張振風

書名：不詳

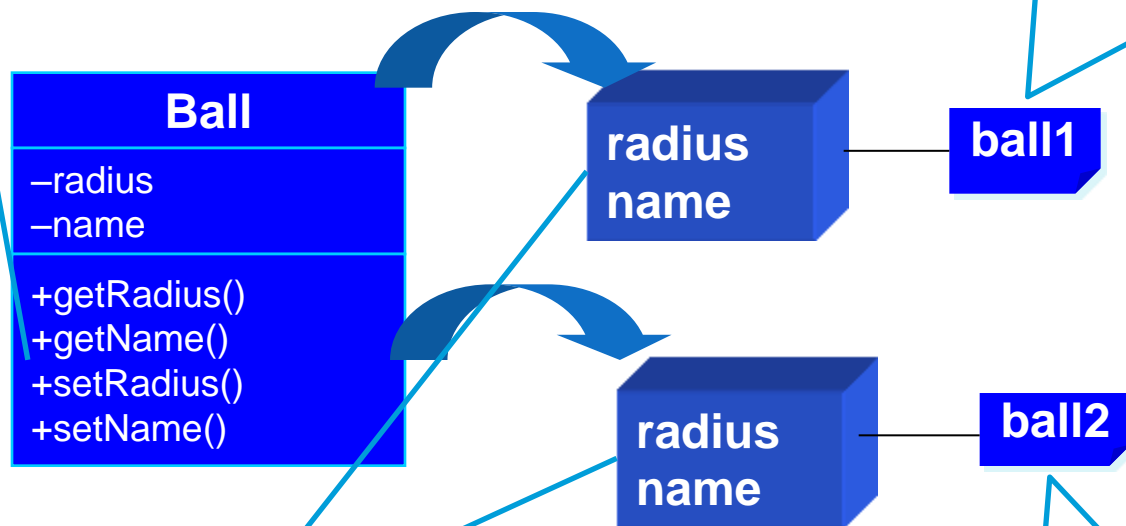
定價：0.0

作者：不詳

About this

方法成員在記憶體中只有一份

新建實體，並以ball1名稱參考



各自擁有Field成員

新建實體，並以ball2名稱參考

About this

- ◆ 方法中所撰寫的每一個資料成員其實會隱含一個**this**參考名稱，這個**this**名稱參考至**呼叫方法的物件**，當呼叫**getName()**方法時，其實相當於執行：

```
public double getName()  
{  
    return name;  
}
```



```
public double getName()  
{  
    return this.name;  
}
```

- ◆ 當使用**ball1**並呼叫**getRadius()**方法時，**this**所參考的就是**ball1**所參考的物件：



About this

- ◆ 當在方法中使用資料成員時，都會隱含使用**this**名稱，當然也可明確的指定，例如在方法定義時使用：

```
public Ball(double radius, String name) {  
  
    this.radius = radius;  
  
    this.name = name;  
  
}
```

- ◆ 參數名稱與資料成員名稱相同時，為了避免參數的作用範圍覆蓋了資料成員的作用範圍，必須明確的使用**this**名稱來指定，但如果參數名稱與資料成員名稱不相同則不用特別指定：

```
public Ball(double r, String n) {  
    radius = r;        // 實際等於this.radius = r;  
    name = n;          // 實際等於this.name = n;  
  
}
```

Constructor & this

```
public class SafeArray {  
    private int[] arr;  
  
    public SafeArray() {  
        this(10); // 預設 10 個元素  
    }  
  
    public SafeArray(int length) {  
        arr = new int[length];  
    }  
  
    ....  
}
```

使用**this(10)**，
這會呼叫另一個有參數的建構方法。

Content

- 類別與物件
- 封裝
- 建構子
- **類別成員 Static**
 - 類別屬性 static attribute
 - 類別方法 static method

類別成員及實體成員

◆ 實體成員 non-static (instance) member

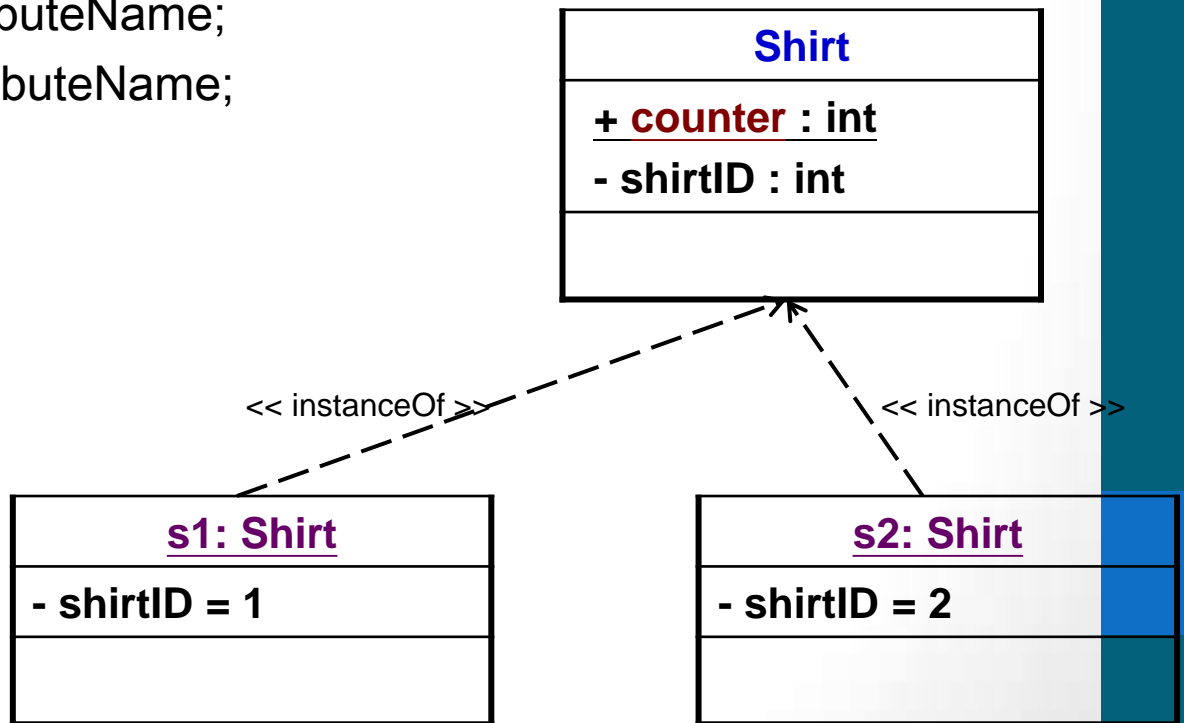
- 實體屬性：每個物件各自擁有一份資料
- 實體方法：需透過特定物件來操作
- 物件實體化之後,物件屬性才存在,物件方法才可使用

◆ 類別成員 static (class) member

- 類別屬性：不伴隨物件,由同類別所有物件共享
- 類別方法：不需特定物件,可由類別來操作
- Java用 **static** 修飾字來宣告類別屬性及類別方法

類別屬性

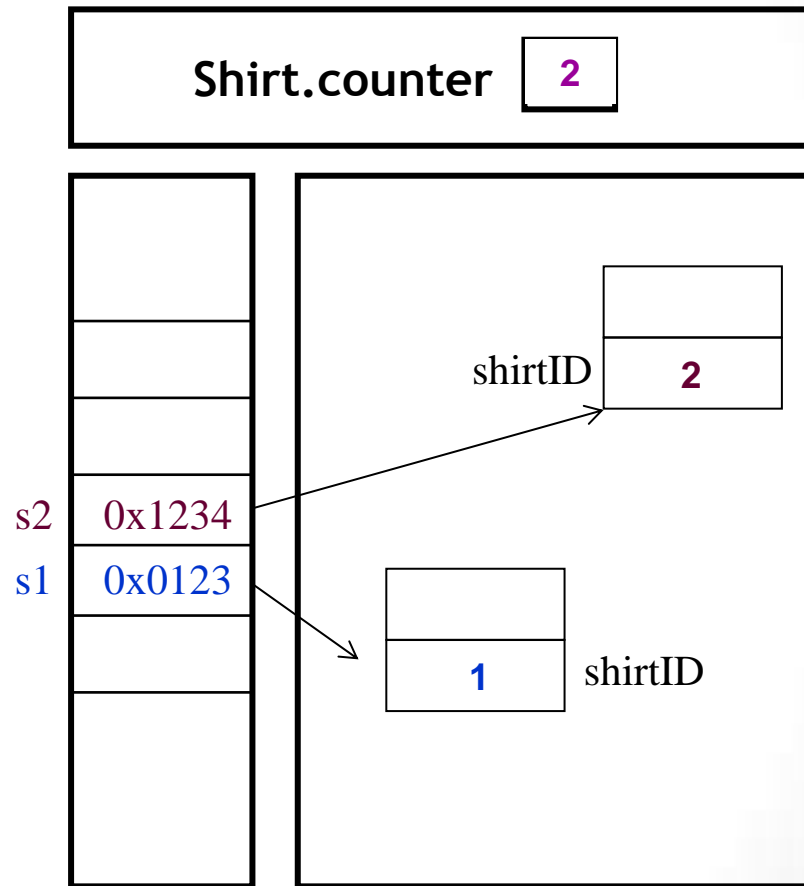
- **static** 類別屬性
 - 用來表示同一類別所有物件共用的屬性欄位 (類似全域變數 Global Variable)
 - 可用兩種方法存取
 - `ClassName.attributeName;`
 - `objectName.attributeName;`



類別屬性 static attribute

```
01 public class Shirt {  
02     public static int counter = 0;  
03     public int ShirtID;  
04  
05     public Shirt() {  
06         counter++;  
07         shirtID = counter;  
08     }  
09 }
```

```
01 public class TestShirt {  
02     public static void main(String [] args) {  
03         Shirt s1 = new Shirt();  
04         Shirt s2 = new Shirt();  
05     }  
06 }
```



類別方法 static method

◆ 用途

- 存取static variable
- 提供utility

◆ 語法

[modifiers] **static** return_type name([argu_list]) { }

◆ 呼叫用法

- `ClassName.methodName();`
- `objectName.methodName();`

類別方法 static method

```
01 public class Shirt {  
02     public static int counter = 0;  
03     public int shirtID;  
04  
05     public Shirt() {  
06         counter++;  
07         shirtID = counter;  
08     }  
09  
10     public static int getTotalCount() {  
11         return counter;  
12     }  
13 }
```

Shirt.counter

2

shirtID

2

1

shirtID

```
01 public class TestShirt {  
02     public static void main(String [] args) {  
03  
04         System.out.println("number of Shirt is "  
05             + Shirt.getTotalCount());  
06         Shirt s1 = new Shirt();  
07         System.out.println("number of Shirt is "  
08             + s1.getTotalCount());  
09         Shirt s2 = new Shirt();  
10         System.out.println("number of Shirt is "  
11             + s2.getTotalCount());  
12     }  
13 }
```

s2

0x1234

s1

0x0123



```
c:\JavaClass>javac TestShirt.java  
c:\JavaClass>java TestShirt  
number of Shirt is 0  
number of Shirt is 1  
number of Shirt is 2  
c:\JavaClass>
```

類別方法 static method

```
01 public class Shirt {
02     public static int counter = 0;
03     public int shirtID;
04
05     public Shirt() {
06         counter++;
07         shirtID = counter;
08     }
09
10     public static int getTotalCount() {
11         return counter;
12     }
13
14     public static String convertShirtSize( int numericalSize) {
15         if(numericalSize < 10){
16             return "S" ;
17         } else if(numericalSize < 14){
18             return "M" ;
19         } else if(numericalSize < 18){
20             return "L" ;
21         } else {
22             return "XL" ;
23         }
24     }
25
26 }
```

```
01 public class TestShirt {
02     public static void main(String [] args) {
03         .....
04         System.out.println("size 14 equals
05             to " + Shirt.convertShirtSize(14);
06     }
07 }
08 }
```

Q & A