基礎程式設計（Java）　講義

類別與物件：自己寫 class、建構子與參照

Deitel & Deitel《Java How to Program: Early Objects》11e, Ch.3

§ 1　一、名詞速查表

中文	English	一句話定義	科目	章節
類別	class	物件的藍圖，定義有哪些資料（欄位）與行為（方法）	Java	CH3
物件	object / instance	依類別造出來的實體，每個有自己的實例變數值	Java	CH3
實例變數	instance variable	宣告在類別內、方法外的欄位，每個物件各有一份	Java	CH3
方法	method	類別裡定義的行為，物件可以呼叫來做事	Java	CH3
set 方法	set method	修改實例變數的公開方法（如 `setName`）	Java	CH3
get 方法	get method	讀取實例變數的公開方法（如 `getName`）	Java	CH3
建構子	constructor	與類別同名、`new` 時自動呼叫來初始化物件；**沒有回傳型別**	Java	CH3
物件建立運算式	class instance creation	`new Account()`，配置一個新物件並回傳其參照	Java	CH3
封裝	encapsulation / information hiding	把資料設 `private`、只透過 `public` 方法存取	Java	CH3
存取修飾子	access modifier	`public`（公開）、`private`（只有自己類別能用）	Java	CH3
UML 類別圖	UML class diagram	三格方框畫類別：類名／屬性／操作	Java	CH3
基本型別	primitive type	`int double boolean` 等，變數**直接存值**	Java	CH3
參照型別	reference type	類別型別（如 `Account`、`String`），變數**存的是指向物件的參照**	Java	CH3
this	this	指向「目前這個物件」，用來區分參數與實例變數	Java	CH3

§ 2 二、核心概念

CH2 用的是**現成的**類別（`System.out`、`Scanner`）；CH3 開始**自己寫類別**。每個類別是一張藍圖，定義「有什麼資料（實例變數）」和「能做什麼（方法）」，再用 `new` 依藍圖造出**物件**。

> 三個關鍵轉變：① 一個類別可造出**多個物件**，各自保有獨立的實例變數值；② 實例變數設 `private`、用 `public` 的 set/get 控制存取，叫**封裝**；③ 物件變數存的不是物件本身，而是**指向物件的參照**（reference）。這三點是 early-objects 的考試核心。

§ 3 三、主要內容

3.1 1. 宣告一個類別：Account

```
// Account.java
public class Account {
    private String name;              // 實例變數（每個 Account 各有一份）

    public void setName(String name) {   // set 方法
        this.name = name;             // this.name 是實例變數，name 是參數
    }
    public String getName() {             // get 方法
        return name;
    }
}
```

- 實例變數宣告在**類別內、方法外**，預設值：物件型別為 `null`、`int` 為 `0`、`double` 為 `0.0`、`boolean` 為 `false`。
- `this.name = name;`：左邊 `this.name` 是實例變數，右邊 `name` 是參數，`this` **用來區分同名的兩者**。

3.2 2. 建立並使用物件：AccountTest

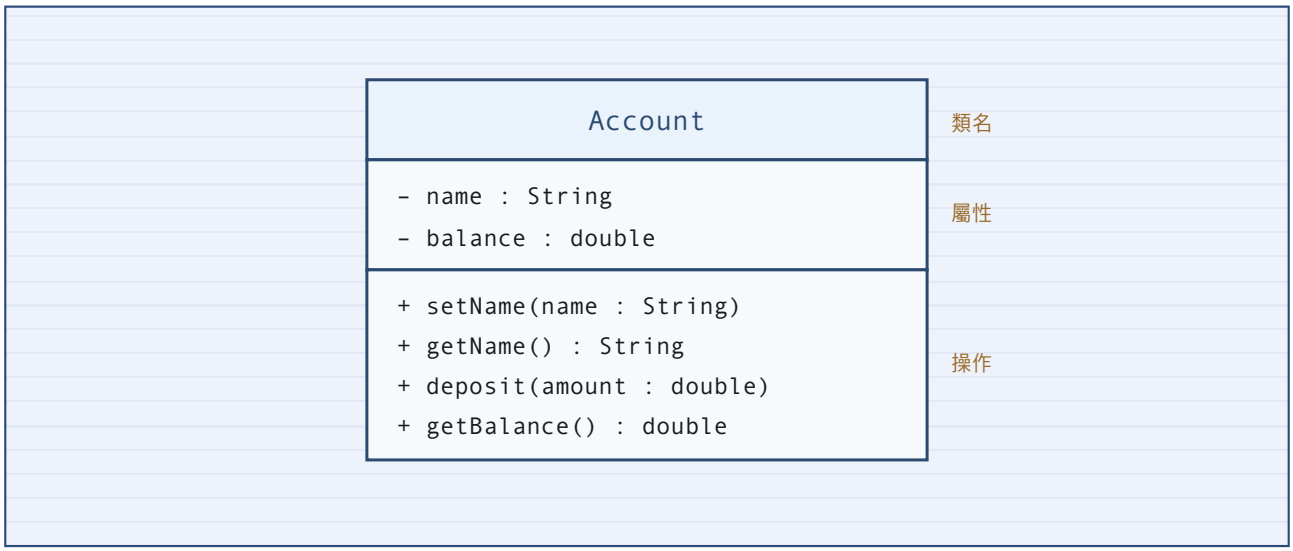
```
// AccountTest.java
public class AccountTest {
    public static void main(String[] args) {
        Account myAccount = new Account();   // 造一個 Account 物件
        myAccount.setName("Jane Green");     // 呼叫方法
        System.out.printf("name = %s%n", myAccount.getName());
    }
}
```

- `new Account()` 是**物件建立運算式**：配置新物件、回傳它的參照，存進 `myAccount`。
- `物件.方法()`：用點號呼叫該物件的方法。

- 兩個類別可一起編譯：`javac Account.java AccountTest.java`，執行含 `main` 的：`java AccountTest`。

3.3 3. UML 類別圖

UML 用三格方框描述類別（與語言無關）：



`-` 代表 `private`、`+` 代表 `public`；屬性寫「名稱 : 型別」，操作寫「名稱(參數) : 回傳型別」。

3.4 4. 封裝：private 資料 + public 方法

把實例變數設 `private`，外界**不能直接改**，只能透過 `public` 的 set/get。好處：set 方法可**驗證**資料（擋掉不合理的值），get 方法可控制呈現方式。這叫**資訊隱藏（information hiding）**。

```
public void deposit(double amount) {
    if (amount > 0.0)            // 驗證：只接受正數
        balance = balance + amount;
}
```

3.5 5. 建構子：建立物件時就初始化

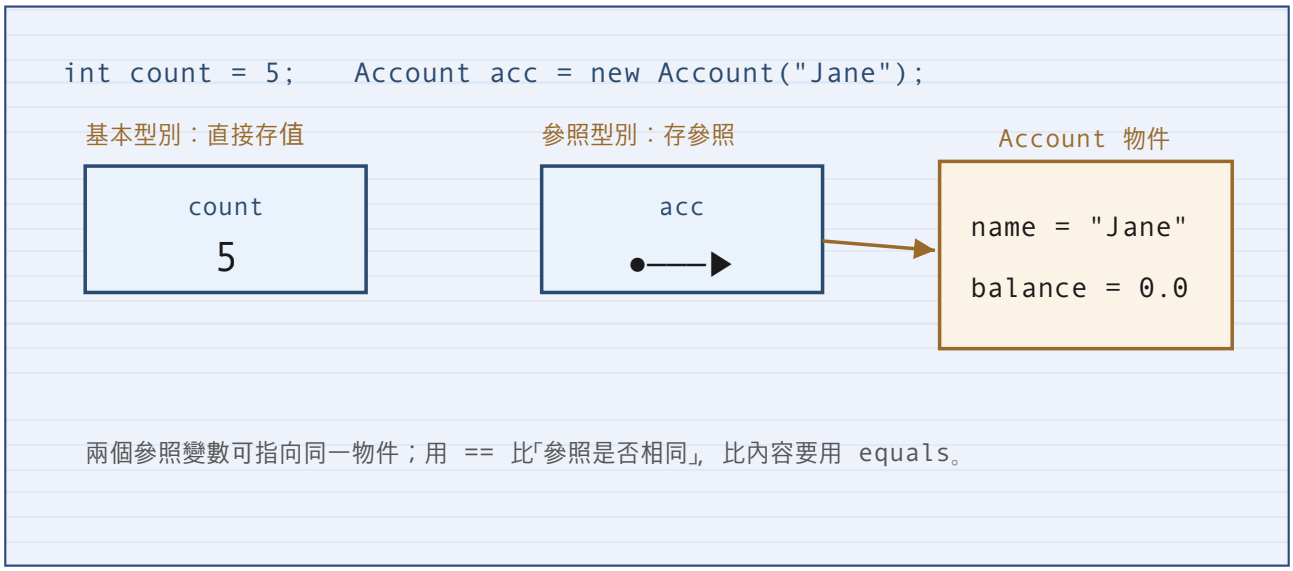
```
public class Account {
    private String name;
    public Account(String name) {      // 建構子：與類名同名、無回傳型別
        this.name = name;
    }
    public String getName() { return name; }
}
// 使用：
Account acc = new Account("Jane Green");   // 建立時就設好 name
```

- 建構子**與類別同名、沒有回傳型別**（連 `void` 都不寫）。

- 若你沒寫任何建構子，Java 給一個無參數的**預設建構子**；一旦自己寫了建構子，預設的就消失。

3.6 6. 基本型別 vs 參照型別（本章招牌考點）

- **基本型別**（`int`、`double`、`boolean`…）：變數**直接存值**。
- **參照型別**（類別型別，如 `Account`、`String`）：變數存的是**指向物件的參照**，物件本身在另一塊記憶體（heap）。



§ 4 四、語法與 API 速查

類別骨架

```
public class 類名 {
    private 型別 欄位;                          // 實例變數
    public 類名(參數) { this.欄位 = 參數; } // 建構子（無回傳型別）
    public void set欄位(型別 v) { this.欄位 = v; }   // set
    public 型別 get欄位() { return 欄位; }             // get
}
```

建立物件：`類名 變數 = new 類名(引數);` **呼叫方法**：`變數.方法(引數);` **this**：`this.欄位` 指本物件的實例變數（區分同名參數） **存取修飾子**：實例變數 `private`、方法通常 `public` **編譯多類別**：`javac A.java B.java` → `java 含main的類別`

§ 5 五、常見錯誤

- **建構子寫了回傳型別**：`public void Account(...)` 變成「剛好同名的普通方法」，不是建構子，`new` 時不會被呼叫。建構子**不寫任何回傳型別**。

- **檔名與 `public class` 不同名**：`public class Account` 必須存成 `Account.java`。
- **用 `==` 比較物件內容**：`==` 比的是「兩個參照是否指向同一物件」，比內容要用 `equals`（如字串 `s1.equals(s2)`）。
- **忘記 `this` 造成遮蔽**：參數與實例變數同名時，方法內直接寫 `name = name` 是「參數設給自己」，實例變數沒被改；要寫 `this.name = name`。
- **物件沒 `new` 就用**：參照是 `null` 時呼叫方法會 `NullPointerException`。
- **把實例變數設 `public`**：破壞封裝，外界可隨意改成不合理的值。

§ 6 六、練習題

例題 1：寫一個 Student 類別

設計 `Student` 類別，含 `private` 實例變數 `name`（String）與 `score`（int），提供 `setName/getName/setScore/getScore`。在 `StudentTest` 的 `main` 建一個物件、設值並印出。

> 1. `Student.java`：宣告兩個 private 實例變數 2. 寫四個 set/get，用 `this` 區分參數 3. `StudentTest.java`：`new Student()`、呼叫 set、再 get 印出

```
public class Student {
    private String name;
    private int score;
    public void setName(String name) { this.name = name; }
    public String getName() { return name; }
    public void setScore(int score) { this.score = score; }
    public int getScore() { return score; }
}

public class StudentTest {
    public static void main(String[] args) {
        Student s = new Student();
        s.setName("小明");
        s.setScore(90);
        System.out.printf("%s 的分數 = %d%n", s.getName(), s.getScore());
    }
}
```

易錯：兩個 public 類別要分成 Student.java 與 StudentTest.java；set 方法漏 `this` 會改不到實例變數。

例題 2：加上建構子

把例題 1 的 `Student` 改成用建構子 `Student(String name, int score)` 初始化，`new` 時直接給值。

> 1. 寫建構子，與類名同名、無回傳型別 2. `this.name = name; this.score = score;` 3. `new Student("小明", 90)` 一行建好

```
public class Student {
    private String name;
    private int score;
    public Student(String name, int score) {    // 建構子，無回傳型別
        this.name = name;
        this.score = score;
    }
    public String getName() { return name; }
    public int getScore() { return score; }
}
// 使用：Student s = new Student("小明", 90);
```

> 易錯：寫成 `public void Student(...)` 就不是建構子了；建構子不寫回傳型別。

例題 3：參照觀念預測

判斷下列輸出，並說明為什麼。

```
Account a = new Account("Jane");
Account b = a;              // b 指向同一物件
b.setName("Mary");
System.out.println(a.getName());
```

> 1. `b = a` 複製的是「參照」還是「物件」？ 2. a 與 b 指向同一物件還是兩個物件？ 3. 透過 b 改名，a 看到的是？

輸出 **`Mary`**。`b = a` 只複製參照，a 與 b 指向**同一個** Account 物件；透過 b 改 name，a 讀到的是同一物件的新值。

> 易錯：以為 `b = a` 會複製出第二個物件。參照型別的賦值只複製參照，不複製物件。

§ 7 七、自我檢核

- [ ] 能宣告一個含 `private` 實例變數與 `public` set/get 的類別。
- [ ] 會用 `new` 建立物件、用點號呼叫方法。
- [ ] 能寫建構子，並說出它與普通方法的差別（同名、無回傳型別）。
- [ ] 能讀寫 UML 類別圖三格（類名／屬性／操作、`+`／`-`）。
- [ ] 能解釋封裝與資訊隱藏，說出 set 方法驗證的好處。
- [ ] 能畫出基本型別（直接存值）與參照型別（存參照指向 heap 物件）的差別，並說明 `==` 與 `equals`。