基礎程式設計（Java）　講義

類別與物件深入：this、static、組合、enum、final

Deitel & Deitel《Java How to Program: Early Objects》11e, Ch.8

§ 1　一、名詞速查表

中文	English	一句話定義	科目	章節
this 參照	this reference	指向「目前這個物件」，方法內用來存取自己的成員	Java	CH8
多載建構子	overloaded constructor	同名建構子但參數列不同，依引數自動選一個	Java	CH8
無引數建構子	no-argument constructor	不收參數的建構子，常用來給預設值	Java	CH8
預設建構子	default constructor	類別沒寫任何建構子時，Java 自動補的無參數建構子	Java	CH8
組合	composition	一個類別的實例變數是「另一個類別的物件」（has-a）	Java	CH8
列舉型別	enum	一組固定具名常數的型別，`enum Day { MON, TUE }`	Java	CH8
static 變數	static variable / class variable	屬於「類別」而非物件，所有物件共享同一份	Java	CH8
static 方法	static method / class method	屬於類別、用類名呼叫，不需先建物件、不能用 `this`	Java	CH8
實例變數	instance variable	屬於「物件」，每個物件各有獨立一份	Java	CH8
static import	static import	`import static` 後可直接寫 `PI`、`sqrt` 不加類名	Java	CH8
final 變數	final variable	只能賦值一次的常數，賦值後不可再改	Java	CH8
套件	package	把相關類別歸到同一命名空間，用 `package` 宣告	Java	CH8
垃圾回收	garbage collection	JVM 自動回收沒有任何參照指向的物件記憶體	Java	CH8

中文	English	一句話定義	科目	章節
精確小數	BigDecimal	`java.math` 的類別，做精確小數運算，避開 `double` 的浮點誤差（多用於金額）	Java	CH8

§ 2 二、核心概念

CH3 學會寫一個類別、用 `private` 封裝、用建構子初始化。CH8 把「類別與物件」往深處補滿：同一物件內部怎麼自我參照（`this`）、一個類別怎麼提供多種建立方式（多載建構子）、一個類別怎麼「內含」另一個類別的物件（組合）、怎麼定義一組固定常數（`enum`），以及哪些成員屬於「整個類別共享」而非「每個物件各一份」（`static`）。

> 全章最關鍵的分界是 **static（類別層級）vs instance（物件層級）**。實例成員每個物件各一份，`new` 幾個物件就有幾份；`static` 成員整個類別只有一份，所有物件共享。這條界線決定了「物件計數器要用 static」「常數 `PI` 用 static final」「`main` 是 static 所以不需先有物件」這些考點。

第二條主線是 **組合（has-a）**：類別的欄位可以是另一個物件（如 `Employee` 內含一個 `Date birthDate`），用點號層層存取（`emp.getBirthDate()`）。這是 early-objects 建構大型程式的基本拼法。

§ 3 三、主要內容

3.1 1. this 參照：指向目前這個物件

方法被呼叫時，`this` 自動指向「呼叫該方法的那個物件」。最常見用途是**區分同名的參數與實例變數**：

```
public class Time {
    private int hour;
    public void setHour(int hour) {    // 參數 hour 遮蔽了實例變數 hour
        this.hour = hour;              // 左邊 this.hour 是實例變數，右邊是參數
    }
}
```

- 沒有遮蔽時 `this` 可省略：方法內直接寫 `hour` 等同 `this.hour`。
- `this` 也能用來**在建構子裡呼叫另一個建構子**：`this(0, 0, 0);`（必須是建構子的第一行）。

3.2 2. 多載建構子與無引數建構子

同一類別可寫多個建構子，只要**參數列不同**（型別或個數），`new` 時依引數自動選對應的那個。這叫**多載（overloading）**：

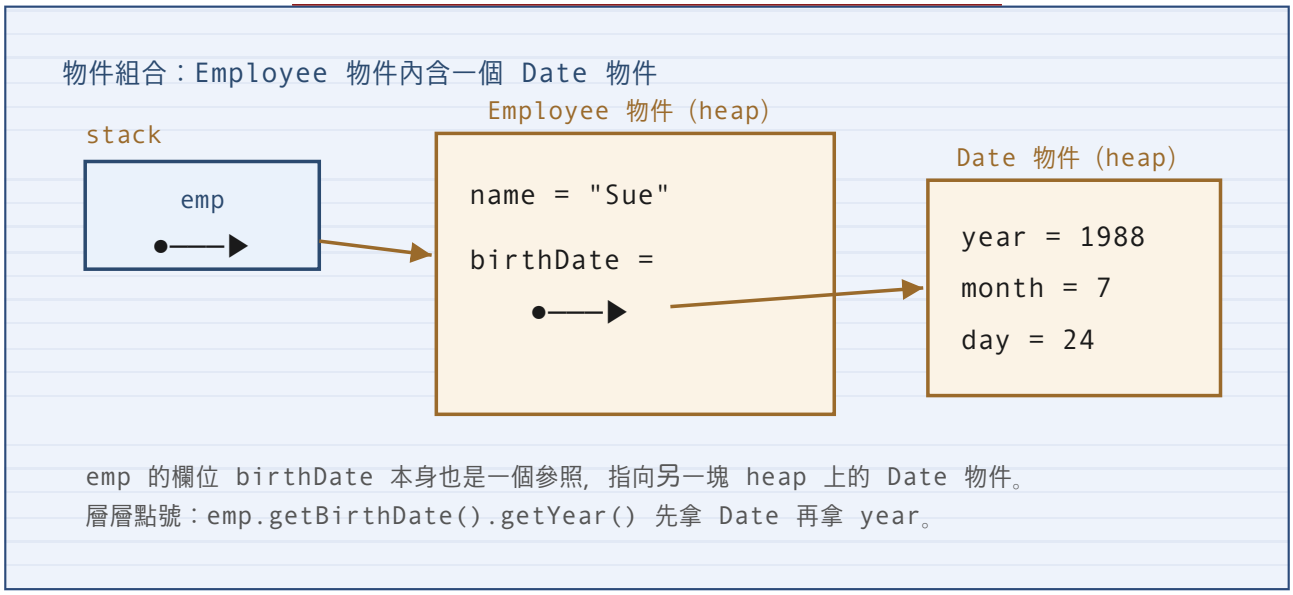
```
public class Time {
    private int hour, minute, second;
    public Time() { this(0, 0, 0); }                  // 無引數：委派給三參數版
    public Time(int h) { this(h, 0, 0); }             // 一參數
    public Time(int h, int m, int s) {                // 三參數：真正設值的
        this.hour = h; this.minute = m; this.second = s;
    }
}
// new Time();          → 00:00:00
// new Time(9);         → 09:00:00
// new Time(9, 30, 15); → 09:30:15
```

- **預設建構子**：若整個類別**一個建構子都沒寫**，Java 自動補一個無參數、空 body 的建構子。一旦你寫了**任何**建構子，這個自動補的就消失；想要無引數版要自己補。
- 用 `this(...)` 委派可避免重複的初始化程式碼。

3.3 3. 組合：類別內含另一個物件（has-a）

實例變數不必是基本型別，可以是**另一個類別的物件**。這叫**組合（composition）**，描述「A 有一個 B」的 has-a 關係：

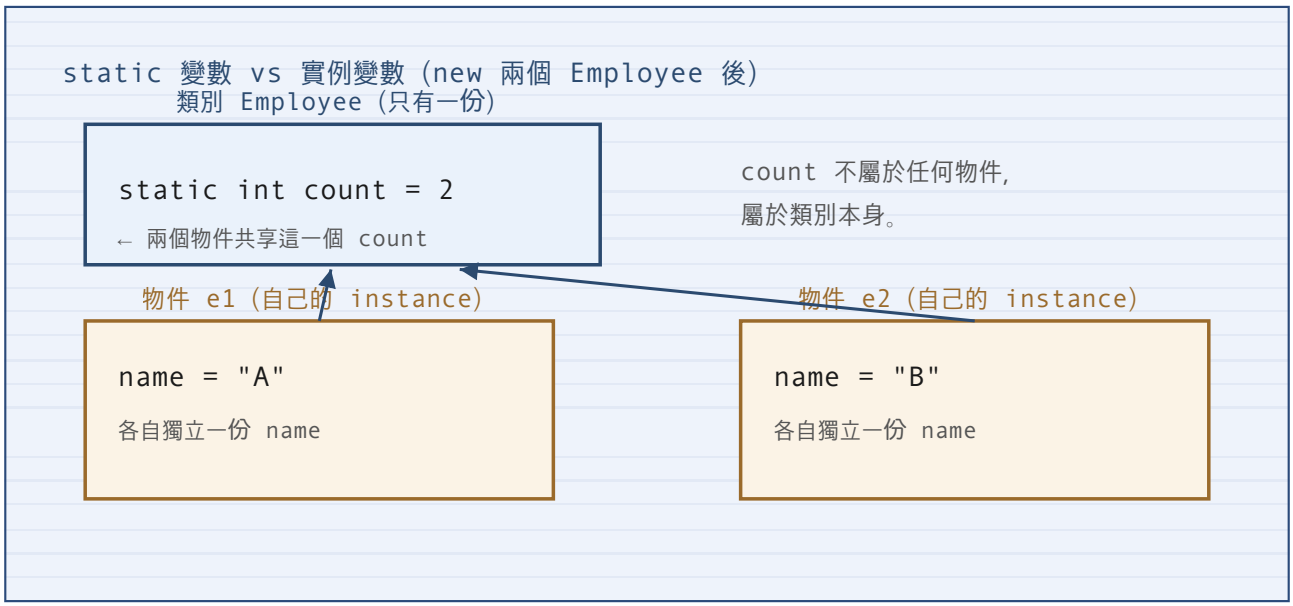
```
public class Employee {
    private String name;
    private Date birthDate;                  // 組合：Employee 內含一個 Date
    public Employee(String name, Date birthDate) {
        this.name = name;
        this.birthDate = birthDate;
    }
    public Date getBirthDate() { return birthDate; }
}
// emp.getBirthDate().getYear() — 用點號層層存取被內含物件的方法
```



3.4 4. static 類別成員：所有物件共享一份

實例變數每個物件各一份；`static` **變數整個類別只有一份**，所有物件共享。經典用途是**計數「目前建立了幾個物件」**：

```
public class Employee {
    private String name;
    private static int count = 0;          // 類別變數：所有 Employee 共享一份
    public Employee(String name) {
        this.name = name;
        count++;                           // 每 new 一個就 +1
    }
    public static int getCount() { return count; }   // static 方法用類名呼叫
}
// new Employee("A"); new Employee("B");
// Employee.getCount() → 2    (用「類名.方法」呼叫，不需物件)
```



何時用 static：① 計數所有物件共用的數值（`count`）；② 與特定物件無關的工具方法（`Math.sqrt`、`main`）；③ 共用常數（`Math.PI`）。`static` 方法**不能用 `this`、不能直接存取實例變數**（它執行時可能根本還沒有任何物件）。

3.5 5. final 實例變數（常數）

`final` 變數**只能賦值一次**，之後不可再改，用來表達常數。常與 `static` 合用成「類別共享的常數」：

```
public class Circle {
    private final double radius;                  // 物件常數：建構子設一次後固定
    public static final double PI = 3.14159;      // 類別常數：全大寫慣例
    public Circle(double radius) { this.radius = radius; }   // final 必須在此設妥
}
```

- `final` 實例變數必須在**宣告時**或**每個建構子裡**完成賦值，否則編譯錯誤。
- `static final` 是「類別層級的常數」，慣例全大寫加底線（`MAX_SIZE`）。

3.6 6. enum 列舉型別

`enum` 定義一組**固定的具名常數**，比用 `int` 當代號更安全可讀：

```
public enum Day { MON, TUE, WED, THU, FRI, SAT, SUN }   // 七個常數

Day today = Day.WED;
for (Day d : Day.values())              // values() 回傳所有常數的陣列
    System.out.println(d + " 序號 " + d.ordinal());   // ordinal() 從 0 起
```

- `values()` 回所有常數、`ordinal()` 回宣告順序（0 起）、`valueOf("MON")` 由字串轉常數。
- enum 可放進 `switch`，且常數名直接寫（`case MON:` 不必寫 `Day.MON`）。

3.7 7. static import、套件、垃圾回收與 BigDecimal

- **static import**：`import static java.lang.Math.*;` 之後可直接寫 `sqrt(x)`、`PI`，省去 `Math.` 前綴。
- **套件與存取**：`package` 把類別歸入命名空間。存取層級由寬到窄：`public`（任何類別）→ 無修飾子（同套件可見，稱 package-private）→ `private`（只有自己類別）。
- **垃圾回收**：物件失去所有參照後成為垃圾，JVM 的 garbage collector 自動回收其記憶體，**程式不需手動釋放**（與 C 的 `free` 不同）。`finalize` 已過時、不保證被呼叫，不應依賴它做清理。
- **BigDecimal 精確小數**：`double`／`float` 是二進位浮點，表示 `0.1`、金額這類十進位小數會有微小誤差（如 `0.1 + 0.2` 不等於 `0.3`），不可用於金錢計算。`java.math.BigDecimal` 做精確

的十進位運算，用 `add`／`subtract`／`multiply` 等方法（不是 `+ - *` 運算子），並用 `RoundingMode` 控制四捨五入。

```
import java.math.BigDecimal;
import java.math.RoundingMode;

BigDecimal price = new BigDecimal("19.99");    // 用「字串」建構才精確，別用 double
BigDecimal qty   = new BigDecimal("3");
BigDecimal total = price.multiply(qty);         // 用方法運算，不能寫 price * qty
total = total.setScale(2, RoundingMode.HALF_UP);    // 取兩位小數、四捨五入
// total → 59.97
```

- 配 `NumberFormat.getCurrencyInstance()` 可把金額格式化成在地貨幣字串（如 `$59.97`）。

§ 4 四、語法與 API 速查

```
this.欄位 = 參數;                          // this 區分同名參數與實例變數
this(引數);                                 // 建構子第一行：呼叫本類別另一建構子

public 類名() { ... }                       // 無引數建構子
public 類名(型別 a) { ... }                 // 多載：參數列不同即可

private OtherClass field;                   // 組合：欄位是另一物件（has-a）
emp.getDate().getYear();                    // 層層點號存取被內含物件

private static int count;                   // 類別變數：所有物件共享一份
public static 型別 方法() { ... }           // 類別方法：類名.方法() 呼叫，無 this
類名.靜態方法();                            // 用類名（非物件）呼叫

private final 型別 常數;                    // 物件常數：賦值一次（宣告或建構子）
public static final 型別 PI = 值;           // 類別常數：全大寫慣例

public enum 名 { A, B, C }                  // 列舉：固定具名常數
列舉.values();  列舉值.ordinal();  列舉.valueOf("A");

import static java.lang.Math.*;             // static import：直接寫 sqrt、PI

new BigDecimal("19.99");                    // 精確小數：用字串建構（金額計算）
a.add(b);  a.multiply(b);                   // BigDecimal 用方法運算，不用 + - *
total.setScale(2, RoundingMode.HALF_UP);// 取兩位小數、四捨五入
```

- **存取層級（寬→窄）**：`public` → package-private（無修飾子）→ `private`
- **靜態 vs 實例**：`static` 屬類別（共享、用類名呼叫）；無 `static` 屬物件（各一份、用物件呼叫）

§ 5 五、常見錯誤

- **static 方法存取實例變數**：`static` 方法執行時可能還沒有物件，直接用實例變數或 `this` 會編譯錯（`non-static ... cannot be referenced from a static context`）。
- **以為每個物件各有一份 static**：`static` 整個類別只有一份；某物件改了它，所有物件都看到新值。
- **寫了建構子卻還呼叫 `new 類名()`**：一旦自寫帶參數建構子、又沒補無引數版，`new 類名()` 會編譯錯（預設建構子已消失）。
- **`this(...)` 不在第一行**：建構子委派 `this(...)` 必須是建構子的第一個敘述，否則編譯錯。
- **final 變數沒賦值或想二次賦值**：`final` 漏在建構子賦值會編譯錯；已賦值後再 `=` 也編譯錯。
- **組合物件忘了先建**：內含的物件欄位是 `null` 時直接呼叫其方法 → `NullPointerException`。
- **用 `==` 比 enum 內容外的字串**：enum 常數可安全用 `==` 比（同一份），但別把 enum 與字串混比。
- **依賴 `finalize` 做清理**：`finalize` 不保證何時、是否被呼叫，已過時；資源釋放用 try-with-resources 或明確 close。
- **用 `double` 算金額**：`double` 有浮點誤差，金額計算要用 `BigDecimal`；且 `BigDecimal` 要用「字串」建構（`new BigDecimal("0.1")`），用 `double` 建構（`new BigDecimal(0.1)`）會把誤差一起帶進來。
- **對 `BigDecimal` 用 `+ - *`**：`BigDecimal` 是物件，運算要呼叫 `add`／`subtract`／`multiply` 方法，直接寫 `a + b` 不能編譯（也不是字串相接）。

§ 6 六、練習題

例題 1：多載建構子 + this

設計 `Time` 類別，含 `private int hour, minute`。提供三個建構子：無引數（設 0:0）、只給時（分設 0）、給時與分。用 `this(...)` 委派，並提供 `String toString()` 回傳 `HH:MM`。

> 1. 三參數建構子用 `this.hour = h; this.minute = m;` 真正設值 2. 無引數版 `this(0, 0);`、單參數版 `this(h, 0);` 委派 3. `toString()` 用 `String.format("%02d:%02d", hour, minute)`

```
public class Time {
    private int hour, minute;
    public Time() { this(0, 0); }
    public Time(int h) { this(h, 0); }
    public Time(int h, int m) { this.hour = h; this.minute = m; }
    public String toString() { return String.format("%02d:%02d", hour,
minute); }
}
// new Time() → 00:00；new Time(9) → 09:00；new Time(9,30) → 09:30
```

> 易錯：`this(...)` 沒放第一行；寫了帶參數建構子卻又呼叫 `new Time()` 而沒補無引數版。

例題 2：static 物件計數器

設計 `Widget` 類別，每 `new` 一個就讓共享計數 `count` 加一，提供 `static int getCount()` 回傳目前共有幾個物件。在 `main` 建 3 個後印出計數。

> 1. `private static int count = 0;`（類別共享一份） 2. 建構子內 `count++;` 3. `Widget.getCount()` 用類名呼叫

```
public class Widget {
    private static int count = 0;
    public Widget() { count++; }
    public static int getCount() { return count; }
    public static void main(String[] args) {
        new Widget(); new Widget(); new Widget();
        System.out.println("count = " + Widget.getCount());   // 3
    }
}
```

易錯：把 count 宣告成實例變數（每物件各一份，永遠是 1）；在 static 方法裡用 this。

例題 3：組合（類別內含另一物件）

設計 `Date`（year/month/day）與 `Employee`（name + 一個 `Date birthDate`）。在 `main` 建一個 Employee 並印出 `name 出生於 year`。

> 1. `Date` 含三 int 與 getter 2. `Employee` 的欄位 `private Date birthDate;`（組合） 3. `emp.getBirthDate().getYear()` 層層點號

```
public class Employee {
    private String name;
    private Date birthDate;
    public Employee(String name, Date birthDate) { this.name = name;
this.birthDate = birthDate; }
    public Date getBirthDate() { return birthDate; }
    public String getName() { return name; }
    public static void main(String[] args) {
        Employee e = new Employee("Sue", new Date(1988, 7, 24));
        System.out.println(e.getName() + " 出生於 " +
e.getBirthDate().getYear());
    }
}
class Date {
    private int year, month, day;
    public Date(int y, int m, int d) { year = y; month = m; day = d; }
    public int getYear() { return year; }
}
```

易錯：birthDate 沒先 new 就用 → NullPointerException；以為組合是繼承（這是 has-a 不是 is-a）。

例題 4：enum 列舉走訪

定義 `enum Day { MON, TUE, WED, THU, FRI, SAT, SUN }`，用增強 for 走訪 `Day.values()`，每個印出「名稱 + ordinal()」。

> 1. `enum Day { ... }` 2. `for (Day d : Day.values())` 3. 印 `d + "" + d.ordinal()`（序號 0 起）

```
public class DayDemo {
    enum Day { MON, TUE, WED, THU, FRI, SAT, SUN }
    public static void main(String[] args) {
        for (Day d : Day.values())
            System.out.println(d + " 序號 " + d.ordinal());
    }
}
// MON 序號 0、TUE 序號 1 …… SUN 序號 6
```

易錯：以為 ordinal 從 1 起（其實 0 起）；在 enum 常數間加逗號以外的標點。

例題 5：final 常數 + this

設計 `Circle`，含 `private final double radius`（建構子設一次）與 `public static final double PI = 3.14159`，提供 `area()` 回 `PI * radius * radius`。在 `main` 建半徑 2 的圓並印面積。

> 1. `final double radius;` 在建構子用 `this.radius = radius;` 設妥 2. `static final double PI`（類別共享常數） 3. `area()` 回 `PI * radius * radius`

```
public class Circle {
    private final double radius;
    public static final double PI = 3.14159;
    public Circle(double radius) { this.radius = radius; }
    public double area() { return PI * radius * radius; }
    public static void main(String[] args) {
        Circle c = new Circle(2.0);
        System.out.printf("area = %.4f%n", c.area());   // 12.5664
    }
}
```

> 易錯：final radius 漏在建構子賦值（編譯錯）；想在 area() 裡改 radius（final 不可二次賦值）。

§ 7 七、自我檢核

- [ ] 能說出 `this` 的用途，並用它區分同名的參數與實例變數。
- [ ] 會寫多載建構子並用 `this(...)` 委派，知道預設建構子何時消失。
- [ ] 能用組合讓一個類別內含另一個物件，並用層層點號存取。
- [ ] 講得出 static 成員與實例成員的差別（共享一份 vs 各一份），知道何時用 static。
- [ ] 會用 `static final` 定義類別常數、`final` 定義物件常數。
- [ ] 會定義 enum 並用 `values()`／`ordinal()` 走訪。
- [ ] 知道垃圾回收自動回收無參照物件，不依賴 `finalize`。
- [ ] 知道金額計算要用 `BigDecimal`（字串建構、用方法運算），不用會有浮點誤差的 `double`。