# 繼承：superclass、subclass、is-a、extends、super、覆寫

Deitel & Deitel《Java How to Program: Early Objects》11e, Ch.9

> **本章為自學延伸，非課程考試範圍**：繼承（Inheritance）老師課程未教、不列入期中／期末考。這份是想多學一點再讀的補充，學起來對之後物件導向設計有幫助；趕考試時可先跳過、優先複習有列考的章。

## § 1　一、名詞速查表

| 中文 | English | 一句話定義 | 科目 | 章節 |
|---|---|---|---|---|
| 繼承 | inheritance | 由既有類別衍生新類別，新類別自動取得父類別的欄位與方法 | Java | CH9 |
| 父類別 | superclass / base class | 被繼承的類別，定義共通的欄位與方法 | Java | CH9 |
| 子類別 | subclass / derived class | 繼承父類別、可再加新成員或改寫的類別 | Java | CH9 |
| is-a 關係 | is-a relationship | 「子類別是一種父類別」（Dog is-a Animal），繼承表達的關係 | Java | CH9 |
| has-a 關係 | has-a relationship | 「一個物件內含另一個物件」（Car has-a Engine），組合表達的關係 | Java | CH9 |
| extends | extends | 宣告繼承的關鍵字，`class 子 extends 父` | Java | CH9 |
| protected 成員 | protected member | 存取層級介於 public 與 private：自己、子類別、同套件可見 | Java | CH9 |
| 方法覆寫 | method overriding | 子類別重新定義父類別已有的方法（同簽名），取代父版本 | Java | CH9 |
| @Override | @Override annotation | 標註此方法是覆寫父方法；簽名打錯時編譯器會報錯 | Java | CH9 |

| 中文 | English | 一句話定義 | 科目 | 章節 |
|---|---|---|---|---|
| super | super reference | 指向父類別部分，用 `super.方法()` 呼叫父版本、`super(...)` 呼叫父建構子 | Java | CH9 |
| Object 類別 | Object class | 所有類別的最終父類別，提供 `toString`、`equals` 等方法 | Java | CH9 |
| 直接父類別 | direct superclass | 子類別 `extends` 的那一個父類別（隔一層的是間接父類別） | Java | CH9 |

## § 2 二、核心概念

CH3 學會寫一個類別、CH8 把類別與物件補滿（this、static、組合）。繼承是物件導向第三根支柱：當多個類別有**共通的欄位與方法**時，把共通部分抽到一個**父類別（superclass）**，其他類別用 `extends` 繼承它，就自動取得那些成員，不必重抄。子類別只需寫「自己多出來的」或「要改寫的」部分。

> 全章最關鍵的一句話是 **is-a vs has-a**。繼承表達 **is-a**（子類別「是一種」父類別：`BasePlusCommissionEmployee` is-a `CommissionEmployee`），用 `extends`；組合（CH8）表達 **has-a**（一個物件「內含」另一個物件：`Employee` has-a `Date`），用欄位。判斷該用哪個就問一句：「A 是一種 B」還是「A 有一個 B」。

第二條主線是 **覆寫（overriding）與 `super`**：子類別可以重新定義父類別已有的方法（如 `toString`、`earnings`），呼叫時走子版本；若想在子版本裡「沿用父版本再加工」，用 `super.方法()` 呼叫父版本。建構子也一樣：子類別建構子第一行用 `super(...)` 把父類別那部分先初始化好。

第三條是 **所有類別都繼承自 `Object`**：就算沒寫 `extends`，每個類別都隱含 `extends Object`，因此都有 `toString()`、`equals()`、`getClass()` 等方法；常覆寫 `toString` 讓物件印出來好讀。

## § 3 三、主要內容

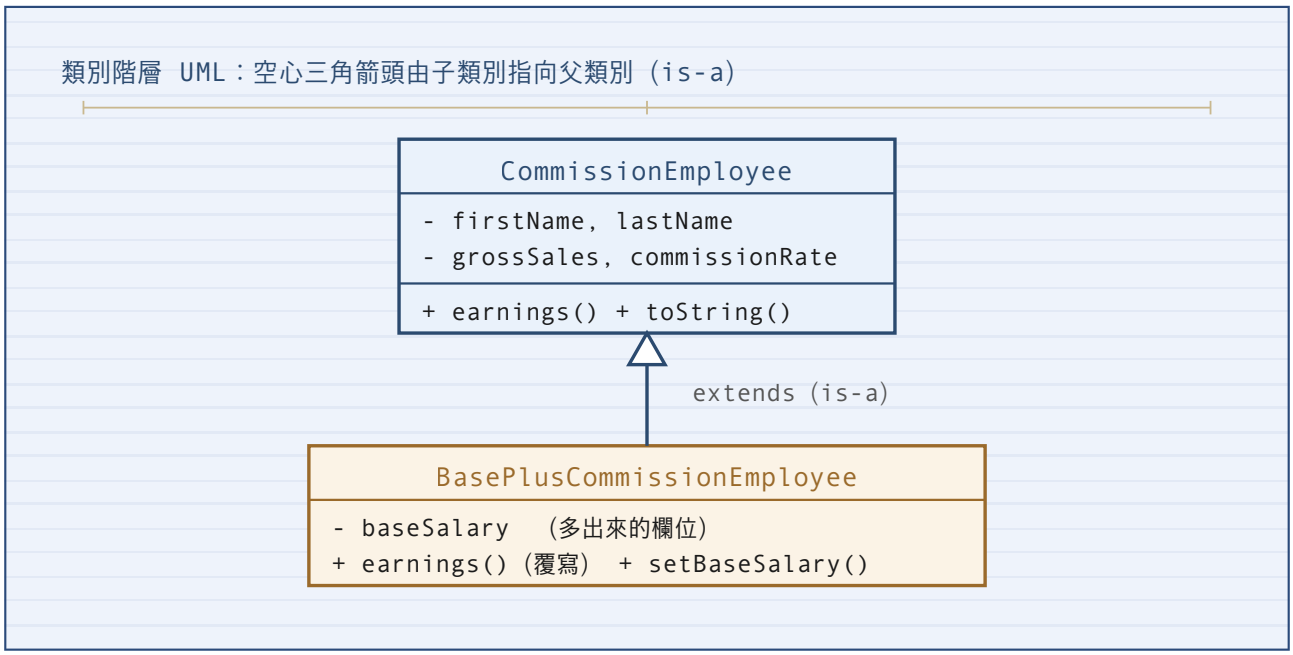
### 3.1 1. is-a 關係與 extends：由父類別衍生子類別

繼承讓子類別自動取得父類別的 `public`／`protected` 成員。用 `extends` 宣告：

```java
public class Animal {                // 父類別：共通的部分
    private String name;
    public Animal(String name) { this.name = name; }
    public String getName() { return name; }
    public String speak() { return "（某種聲音）"; }
}

public class Dog extends Animal {  // 子類別：Dog is-a Animal
    public Dog(String name) { super(name); }    // 呼叫父建構子初始化 name
    // 自動擁有 getName()；不必重寫
}
// new Dog("旺財").getName() → "旺財"（繼承來的方法）
```

- `Dog` 沒寫 `getName()`，卻能呼叫它——這就是繼承：**父類別的可見成員，子類別直接擁有**。
- 一個子類別只能 `extends` **一個**直接父類別（Java 是單一繼承）；但可以層層繼承（A→B→C）。




## 3.2 2. protected 成員：讓子類別存取得到

`private` 成員子類別**看不到**（仍被繼承，但只能透過父類別的 public 方法存取）。若想讓子類別**直接**存取父類別欄位，可把它宣告為 `protected`：

```
public class CommissionEmployee {
    protected double grossSales;        // protected：自己 + 子類別 + 同套件可見
    private double commissionRate;      // private：只有 CommissionEmployee 自己
}

public class BasePlusCommissionEmployee extends CommissionEmployee {
    public double earnings() {
        return baseSalary + grossSales * 0.1;   // 直接用 protected 的 grossSales
    }
}
```

- 存取層級由寬到窄：`public` →（同套件）→ `protected` →（同套件＋子類別）→ 無修飾子（package-private）→ `private`。
- `protected` 方便，但破壞封裝（子類別綁死了父欄位名）。**Deitel 建議優先用 `private` + public/protected 的 getter／setter**，少直接開放 `protected` 欄位。本章兩種寫法都示範，理解差別即可。

## 3.3 3. 方法覆寫與 @Override

子類別可以**重新定義**父類別已有的方法（同名、同參數列、同回傳型別），叫**覆寫（overriding）**。呼叫時，用哪個版本看**物件實際是哪個類別**：

```
public class Animal {
    public String speak() { return "（某種聲音）"; }
}
public class Dog extends Animal {
    @Override                                       // 標註：這是覆寫父方法
    public String speak() { return "汪汪"; }        // 取代父版本
}
public class Cat extends Animal {
    @Override
    public String speak() { return "喵"; }
}
// new Dog().speak() → "汪汪"；new Cat().speak() → "喵"
```

- `@Override` **不是必須，但強烈建議加**：它讓編譯器檢查「你真的有覆寫到父方法」。若簽名打錯（如 `speak(int)`、`Speak()`），編譯器會報錯，否則那會變成一個沒人呼叫的新方法、debug 很痛。
- **覆寫 vs 多載（CH6/8）別搞混**：覆寫是「子類別改寫父類別**同簽名**方法」；多載是「**同類別**內同名、**不同參數列**」。

## 3.4 4. super：呼叫父版本方法、呼叫父建構子

覆寫後若想「沿用父版本再加東西」，用 `super.方法()` 呼叫被覆寫掉的父版本：

```
public class BasePlusCommissionEmployee extends CommissionEmployee {
    @Override
    public String toString() {
        return "底薪型 — " + super.toString()      // 先拿父版本字串，再前綴
             + String.format("%n底薪：%.2f", baseSalary);
    }
}
```

建構子也用 `super(...)`，但規則更嚴：

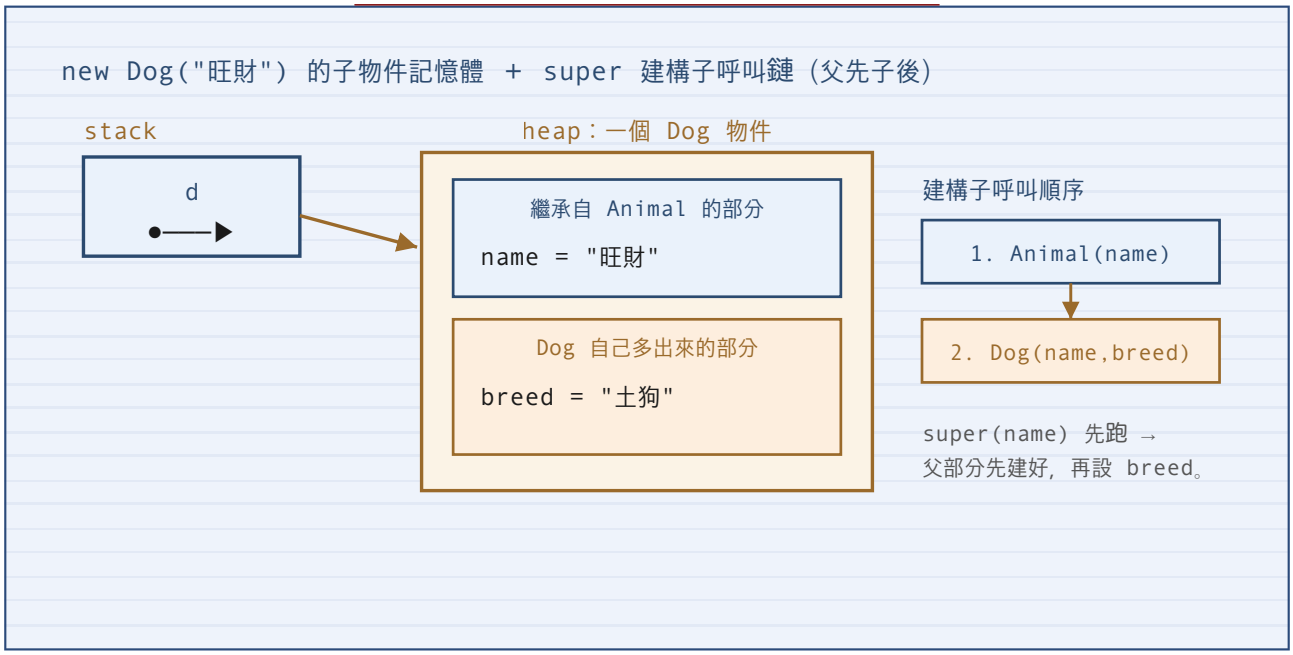
```
public class BasePlusCommissionEmployee extends CommissionEmployee {
    private double baseSalary;
    public BasePlusCommissionEmployee(String first, String last,
                                      double sales, double rate, double base) {
        super(first, last, sales, rate);     // 必須是建構子第一行：先初始化父類別部分
        this.baseSalary = base;              // 再初始化自己多出來的欄位
    }
}
```

- `super(...)` **必須是子類別建構子的第一個敘述**（和 `this(...)` 一樣只能放第一行，兩者不能同時是第一行）。
- 若子類別建構子**沒寫** `super(...)`，Java 自動插入 `super()`（呼叫父類別**無引數**建構子）。**因此若父類別只有帶參數建構子、沒有無引數版，子類別就必須明確寫** `super(...)`，否則編譯錯。

## 3.5 5. 繼承下的建構子呼叫順序

建立子類別物件時，建構子**由上而下（父先子後）**執行：先把父類別部分初始化好，子類別才在那個基礎上加自己的。

```
class A {            A() { System.out.println("A 建構子"); } }
class B extends A {  B() { System.out.println("B 建構子"); } }   // 隱含 super() 在第一行
class C extends B {  C() { System.out.println("C 建構子"); } }
// new C(); 的輸出順序：
// A 建構子
// B 建構子
// C 建構子
```




## 3.6 6. Object 類別：所有類別共同的父類別

任何類別**沒寫** `extends` **時都隱含** `extends Object`。`Object` 提供每個物件都有的方法，最常覆寫的是 `toString` 與 `equals`：

```
public class Point {
    private int x, y;
    public Point(int x, int y) { this.x = x; this.y = y; }

    @Override
    public String toString() { return "(" + x + ", " + y + ")"; }    // 印物件時好讀

    @Override
    public boolean equals(Object o) {                  // 比「內容」是否相同
        if (this == o) return true;
        if (!(o instanceof Point)) return false;
        Point p = (Point) o;
        return x == p.x && y == p.y;
    }
}
```

- `toString()`：`Object` 預設回 `類名@雜湊碼`（很難讀）；覆寫後 `System.out.println(物件)` 與字串相接會自動呼叫你的版本。
- `equals()`：`Object` 預設等同 `==`（比是否同一物件）；要比**內容相等**就覆寫它。這正是 CH8「`==` 比物件、`equals` 比內容」之所以成立的原因——`String` 等類別都覆寫了 `equals`。
- 覆寫 `equals` 慣例上也要一起覆寫 `hashCode`（放進 `HashMap`／`HashSet` 才正確）；本章重點先放 `toString`／`equals`。

### 3.7 7. 何時用繼承、何時用組合

繼承強大但容易被誤用。判準回到那句話：

- **is-a → 繼承**：「子類別**是一種**父類別」，且子類別能合理替換父類別出現的場合（`Dog` 是 `Animal`，`SavingsAccount` 是 `Account`）。
- **has-a → 組合（CH8）**：「一個物件**內含**另一個物件」（`Car` 內含 `Engine`，`Employee` 內含 `Date`）。誤把 has-a 寫成繼承（`Car extends Engine`）邏輯就亂了。
- 不確定時**傾向組合**：組合耦合較鬆、較好維護；繼承讓子類別綁死父類別實作，父類別一改可能波及全部子類別。
- 這章是自學延伸，先建立「看到共通成員 → 想想能不能抽父類別」「is-a 才繼承」的直覺即可，不必背所有細節。

## § 4 四、語法與 API 速查

```
class 子類別 extends 父類別 { ... }      // 宣告繼承（is-a），只能單一直接父類別

protected 型別 欄位;                      // protected：自己 + 子類別 + 同套件可見
                                          // 存取寬→窄：public → protected → (package)
→ private

@Override                                 // 標註覆寫（建議加，編譯器幫檢查簽名）
public 回傳型別 方法(...) { ... }         // 覆寫：同名、同參數列、同回傳型別

super.方法(...);                          // 呼叫被覆寫掉的父類別版本
super(引數);                              // 子類別建構子第一行：呼叫父建構子
                                          // 沒寫時 Java 自動補 super()（呼叫父無引數建構
子）

@Override
public String toString() { ... }          // 覆寫 Object.toString，讓物件印出好讀
@Override
public boolean equals(Object o) { ... }   // 覆寫 Object.equals，比內容相等

o instanceof 型別                         // 判斷 o 是否為某型別（覆寫 equals 常用）
(型別) o                                  // 向下轉型：把父型別參照轉回子型別
```

- **覆寫 vs 多載**：覆寫＝子改父、**同簽名**；多載＝同類別、**不同參數列**。
- **建構子順序**：父先子後；`super(...)` / `this(...)` 只能放建構子第一行（且二選一）。
- **單一繼承**：Java 一個類別只能 `extends` 一個父類別（介面才能多個，本章不涉及）。

## § 5 五、常見錯誤

- **把 has-a 寫成繼承**：`Car extends Engine`（車「是一種」引擎？不合理）。內含關係用組合（欄位），不是繼承。

- **子類別存取父類別的 `private` 欄位**：`private` 子類別看不到（編譯錯）；要嘛改 `protected`，要嘛透過父類別 public getter／setter。
- **覆寫時簽名打錯卻沒 `@Override`**：寫成 `public String Speak()`（大寫）或參數列不同，會變成「新方法」而非覆寫，父版本仍被呼叫，行為與預期不符。加 `@Override` 編譯器會立刻報錯。
- **`super(...)` 沒放第一行**：子類別建構子裡 `super(...)` 必須是第一個敘述，放後面編譯錯。
- **父類別只有帶參數建構子、子類別沒寫 `super(...)`**：Java 自動插入的 `super()` 找不到父類別無引數建構子 → 編譯錯。子類別要明確 `super(對應引數)`。
- **覆寫 `equals` 的參數型別寫錯**：`Object.equals` 的參數是 `Object o`；若寫成 `equals(Point p)` 那是多載、不是覆寫（`@Override` 會報錯）。先 `instanceof` 檢查再向下轉型。
- **以為覆寫掉父方法就拿不回父版本**：用 `super.方法()` 仍可呼叫被覆寫掉的父版本。
- **濫用繼承做程式碼重用**：只為「省去重抄」而繼承一個 is-a 不成立的類別，會讓設計變脆。不確定時用組合。

## § 6 六、練習題

### 例題 1：extends + 覆寫

設計父類別 `Animal`（含 `name`、`getName()`、`speak()` 回「（某種聲音）」）與子類別 `Dog`（覆寫 `speak()` 回「汪汪」）。在 `main` 建一個 `Dog`，印出名字與叫聲。

> 1. `Animal` 含 `private String name`、建構子、`getName()`、`speak()` 2. `Dog extends Animal`，建構子用 `super(name)`，覆寫 `speak()` 加 `@Override` 3. `main` 建 `Dog("旺財")`，印 `d.getName()`（繼承來）與 `d.speak()`（覆寫）

```
public class Animal {
    private String name;
    public Animal(String name) { this.name = name; }
    public String getName() { return name; }
    public String speak() { return "（某種聲音）"; }
}
class Dog extends Animal {
    public Dog(String name) { super(name); }
    @Override public String speak() { return "汪汪"; }
}
// 在 main：Dog d = new Dog("旺財");
// System.out.println(d.getName() + " 說 " + d.speak());  → 旺財 說 汪汪
```

易錯：忘了 `super(name)` 而父類別沒無引數建構子（編譯錯）；覆寫漏 `@Override` 且方法名打錯成新方法。

### 例題 2：super 呼叫父建構子

設計父類別 `Person`（`name`，`toString()` 回 `name`）與子類別 `Student`（多一個 `school`）。`Student` 建構子用 `super(name)` 初始化父部分，再設 `school`。在 `main` 印出 `Student` 物件。

> 1. `Person` 含 `protected String name`、建構子、`toString()` 回 `name` 2. `Student extends Person`，建構子第一行 `super(name);`，再 `this.school = school;` 3. 覆寫 `toString()` 用 `super.toString()` 串接 school

```
public class Person {
    protected String name;
    public Person(String name) { this.name = name; }
    @Override public String toString() { return name; }
}
class Student extends Person {
    private String school;
    public Student(String name, String school) {
        super(name);              // 必須第一行：先初始化父部分
        this.school = school;
    }
    @Override public String toString() { return super.toString() + " (" +
school + ") "; }
}
// new Student("小明", "淡江") → 小明 (淡江)
```

易錯：`super(name)` 沒放第一行；以為覆寫 `toString` 後拿不回父版本（其實 `super.toString()` 可）。

### 例題 3：三層繼承與建構子順序

定義三層 `A`←`B`←`C`，每個建構子印一行自己的名字。在 `main` `new C()`，預測並寫出輸出順序。

> 1. `class A { A() { 印 "A" } }` 2. `class B extends A { B() { 印 "B" } }`（隱含 `super()` 在第一行） 3. `class C extends B { C() { 印 "C" } }`；`new C()` 輸出 A→B→C

```
public class ChainDemo {
    public static void main(String[] args) { new C(); }   // 輸出：A 建構子 / B 建構子 / C 建構子
}
class A { A() { System.out.println("A 建構子"); } }
class B extends A { B() { System.out.println("B 建構子"); } }
class C extends B { C() { System.out.println("C 建構子"); } }
```

易錯：以為從子到父（C→B→A）；其實是父先子後（A→B→C），因為每個建構子第一行先跑 `super()`。

### 例題 4：protected 欄位給子類別用

設計父類別 `Shape`（`protected String name`、`area()` 回 `0`）與子類別 `Rectangle`（`width`、`height`，覆寫 `area()`）。`Rectangle` 的描述用到父類別的 `protected name`。在 `main` 印出面積。

> 1. `Shape` 含 `protected String name`、建構子、`area()` 回 `0` 2. `Rectangle extends Shape`，覆寫 `area()` 回 `width * height` 3. 子類別方法直接用繼承來的 `protected name`

```
public class Shape {
    protected String name;
    public Shape(String name) { this.name = name; }
    public double area() { return 0; }
}
class Rectangle extends Shape {
    private double width, height;
    public Rectangle(double w, double h) { super("矩形"); width = w; height =
h; }
    @Override public double area() { return width * height; }
    public String describe() { return name + " 面積 = " + area(); }   // 直接用
protected name
}
// new Rectangle(3,4).describe() → 矩形 面積 = 12.0
```

> 易錯：把 `name` 設 `private` 後子類別存取不到（編譯錯）；忘了 `super("矩形")` 初始化父欄位。

### 例題 5：覆寫 toString 與 equals

設計 `Point`（`x`、`y`），覆寫 `toString()` 回 `(x, y)`、覆寫 `equals(Object)` 比兩點座標是否相同。在 `main` 建兩個座標相同的點，印出並比較 `==` 與 `equals` 的差別。

> 1. 覆寫 `toString()` 回 `"(" + x + ", " + y + ")"` 2. 覆寫 `equals(Object o)`：先 `instanceof` 檢查、再向下轉型、比 `x`／`y` 3. `main` 建 `p1`、`p2` 同座標，印 `p1==p2`（false，不同物件）與 `p1.equals(p2)`（true，內容同）

```
public class Point {
    private int x, y;
    public Point(int x, int y) { this.x = x; this.y = y; }
    @Override public String toString() { return "(" + x + ", " + y + ")"; }
    @Override public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Point)) return false;
        Point p = (Point) o;
        return x == p.x && y == p.y;
    }
    public static void main(String[] args) {
        Point p1 = new Point(1, 2), p2 = new Point(1, 2);
        System.out.println(p1 + " vs " + p2);          // (1, 2) vs (1, 2)
        System.out.println("== : " + (p1 == p2));       // false（不同物件）
        System.out.println("equals : " + p1.equals(p2)); // true（內容相同）
    }
}
```

易錯：`equals` 參數寫成 `Point p`（變多載非覆寫，`@Override` 報錯）；漏 `instanceof` 直接轉型遇到非 Point 物件會 `ClassCastException`。

## § 7 七、自我檢核

> 提醒：本章為自學延伸、非考試範圍，以下檢核供想多學的人自評。

- [ ] 能分辨 is-a（繼承）與 has-a（組合），並說出該用哪個。
- [ ] 會用 `extends` 寫子類別，知道子類別自動擁有父類別可見成員。
- [ ] 知道 `protected` 的可見範圍，也理解 Deitel 為何建議優先用 `private` + getter／setter。
- [ ] 會用 `@Override` 覆寫方法，並說得出覆寫與多載的差別。
- [ ] 會用 `super.方法()` 呼叫父版本、`super(...)` 呼叫父建構子（必在第一行）。
- [ ] 講得出繼承下建構子的執行順序（父先子後）與原因。
- [ ] 知道所有類別繼承自 `Object`，會覆寫 `toString` 與 `equals`（含 `instanceof` 檢查）。