類別圖

UML的類別圖（Class Diagram）是一種可表示一組類別、物件個體和介面之間靜態關係的圖型。雖然稱為類別圖，但使用到的不只是類別而已。

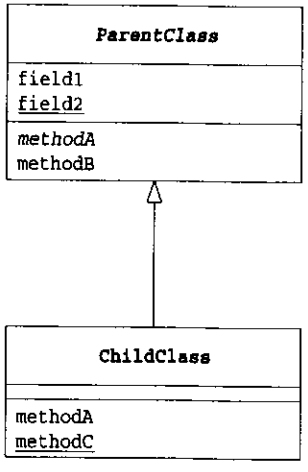
類別和階層的關係

Java程式與對應類別圖的範例即如圖0-1所示。

圖 0-1 類別階層關係的類別圖

```
abstract class ParentClass {
    int field1;
    static char field2;
    abstract void methodA();
    double methodB() {
        // ...
    }
}

class ChildClass extends ParentClass {
    void methodA() {
        // ...
    }
    static void methodC() {
        // ...
    }
}
```



上圖所表示的是ParentClass和ChildClass這2個類別之間的關係。白色箭頭△方向就是類別的階層關係。箭頭是從子類別（subclass）指向父類別（superclass）（所以這是extends的箭頭）。

ParentClass是ChildClass的父類別、ChildClass則是ParentClass的子類別。父類別也稱為基本類別（Base Class），而子類別又稱為衍生類別或擴充類別。

類別均以方框表示，方框內再以橫線分割幾個部分，分別是：

- **類別名稱**
- **欄位名稱**
- **方法名稱**

有時除了名稱之外，還會有其他附屬資訊（如存取控制、方法的引數或型別等），或者是反其道而行省略圖中不需要的項目（故不一定每次都能從類別圖還原成原始程式）。

abstract類別（抽象類別）名稱以斜體表示。如ParentClass為抽象類別，故寫成斜體。

static欄位（類別欄位）名稱加底線。如field2為類別欄位，故加上底線。

abstract方法（抽象方法）以斜體表示。如ParentClass的methodA是抽象方法，故寫成斜體。

static方法（類別方法）名稱加底線。如ChildClass的methodC為類別方法，故加上底線。

▶▶深入說明 - Java和C++的名詞術語

Java跟C++的名詞術語稍微有點不一樣。Java的欄位（field）對應到C++的成員變數（member variable），Java的方法（method）則對應到C++的成員函數（member function）。

▶▶深入說明 - 箭頭方向

在 UML 當中，箭頭是從子類別指向父類別。也許有人認為反過來應該比較容易理解，畢竟是先有父類別才能生出子類別。

其實，各位可以這麼想：定義子類別時，要用 extends 指定父類別。所以子類別一定會知道父類別是誰；但是父類別就不一定知道子類別在哪裡。必須知道對方在哪裡，才能指出來，因此箭頭方向是從子類別指向父類別。

介面與實作

圖 0-2 也是類別圖範例之一。

圖中表示一個 Printable 介面，該介面上實作了一個 PrintClass 類別。介面名稱為斜體。

虛線的白色箭頭是表示介面與實作類別的關係。箭頭從實作類別指向介面（所以這是 implements 的箭頭）。

UML 中的 Java 介面則以<<interface>>表示。

圖 0-2 介面和實作類別的類別圖

```
interface Printable {
    abstract void print();
    abstract void newPage();
}

class PrintClass implements Printable {
    void print() {
        // ...
    }
    void newPage() {
        // ...
    }
}
```

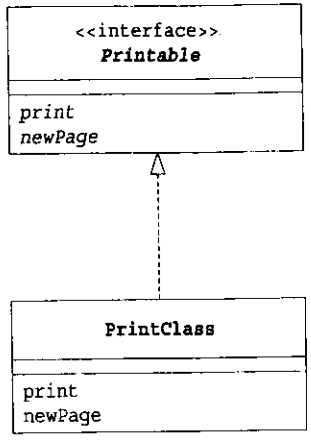
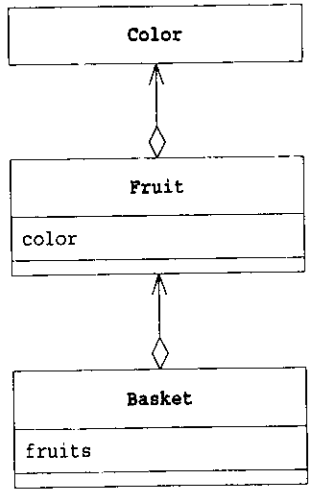


圖 0-3 聚合關係的類別圖

```
class Color {
    // ...
}

class Fruit {
    Color color;
    // ...
}

class Basket {
    Fruit[] fruits;
    // ...
}
```



聚合

圖 0-3 也是類別圖範例之一。

圖中所表示的是 Color（顏色）、Fruit（水果）、Basket（果籃）這 3 個類別的關係。Basket 類別中的 fruits 欄位是 Fruit 類別的陣列，Basket 類別的物件個體則有多個 Fruit 類別的物件個體。而 Fruit 類別的 color 欄位是 Color 類別型別，Fruit 類別的物件個體則只有 1 個 Color 類別的物件個體。從平面來看，它們的關係就好比是果籃裡面有一些水果，各種水果有不同的顏色。

這種「結合」的關係就稱為「聚合」（aggregation）。只要結合了物件個體，無論個數多寡都有聚合關係。配列、java.util.Vector 類別也好，如何實作也罷，只要有結合物件個體，就具有聚合關係。

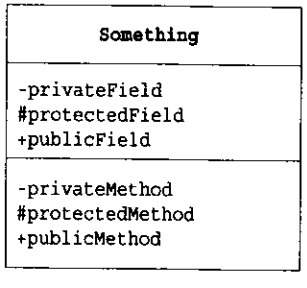
白色菱形的直線代表聚合關係，請把它想成是把東西放在菱形的容器上。

存取控制

圖 0-4 還是類別圖的範例之一。

圖 0-4 存取控制的類別圖

```
class Something {
    private   int privateField;
    protected int protectedField;
    public    int publicField;
    private   void privateMethod() {
    }
    protected void protectedMethod() {
    }
    public    void publicMethod() {
    }
}
```



圖中所表示的是方法和欄位的存取控制。UML 只要在方法或欄位的名稱前面加上符號，就可以表示存取控制。

如果是 +，則表示這是 public 的方法或欄位，從任何位置都可以存取。

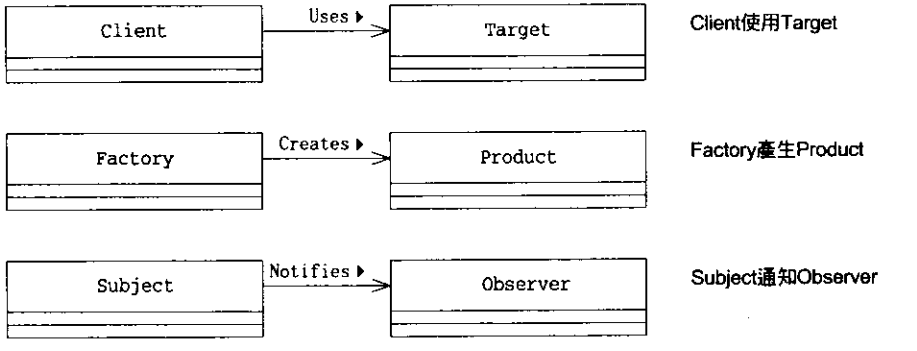
而如果是 -，則表示這是 private 的方法或欄位，此時從類別以外的位置就無法存取。

若是 #，則表示這是 protected 的方法或欄位，只有同一類別、子類別或同一 package 內的類別才能存取。

類別間的關聯性

在有相關的名稱前面加上黑色三角形（▸），就可以表示類別間的關聯性。範例即如圖 0-5 所示。

圖 0-5 類別的關聯性



順序圖

UML的順序圖（Sequence Diagram）是用來表示啟動程式時，會按照怎樣的順序執行哪些方法、哪些現象會依什麼順序發生。

類別圖表示「不因時間而變化的部分（靜態關係）」，而順序圖則表示「隨著時間變化的部分（動態行為）」。

處理流程和物件間的協調

圖 0-6 是一種順序圖的範例。

圖 0-6 順序圖範例（呼叫方法）

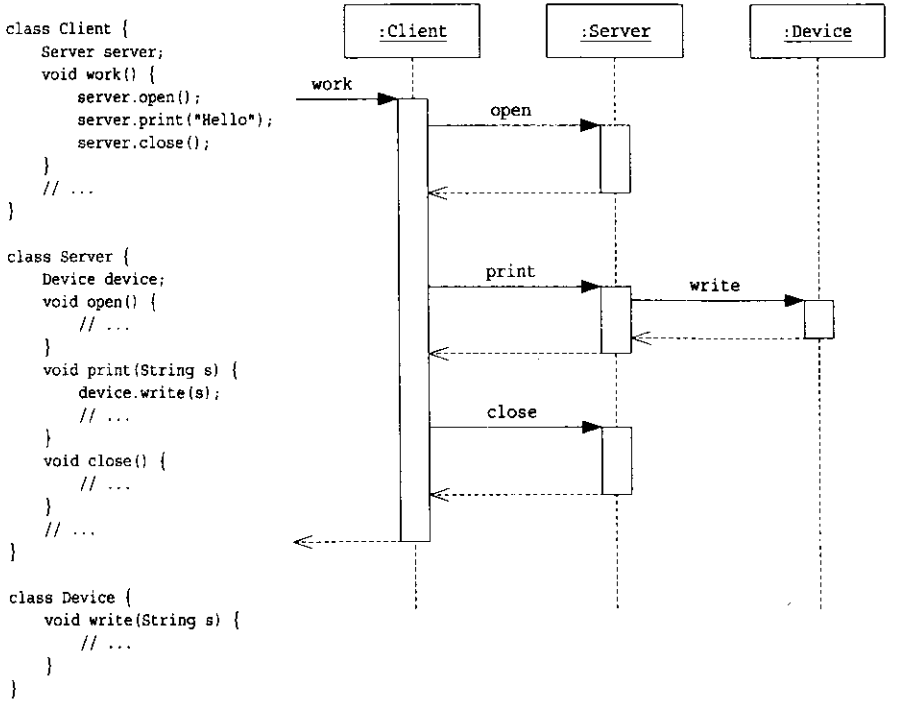


圖 0-6 的右邊是順序圖，左邊則是相對應的 Java 程式（一部分）。

在這個範例中出現了 3 個物件個體，分別對應到圖例上方的 3 個方框。在方框中，類別名稱寫在冒號（:）後面，而且還加上底線，即如：Client, :Server, :Device。意思是指 Client 類別的物件個體、Server 類別的物件個體以及 Device 類別的物件個體。

如果每個物件個體均須命名，則將其名稱寫在冒號之前，如 server: Server。

所有物件個體都有一條往下延伸的虛線，稱為 lifeline（生命線）。在此請將時間座標看成由上而下，上面是過去、下面則是未來。只在有物件個體時，才會有 lifeline。

lifeline 中間有一個長條型的方框，表示該物件正在動作中。

圖中有幾個箭頭指向左右兩邊。請先看到標示為 open 的箭頭記號。若線段為實心且箭頭全黑（——►）則表示呼叫方法。此圖是 client 呼叫 server 的 open 方法。因為已經成功呼叫出 open 方法，所以 server 物件個體是動作中的狀態，故為長條型的方框。

open 箭頭所指的 server 長條型方框下面，還有一條指向 client 的虛線箭頭（- - - - ►）。這是指 open 方法的 return（回復）。這個範例把所有方法的 return 都畫出來，不過也有人寧願省略以免麻煩。

因為控制又回到了 client，所以 server 物件個體本來動作中的長條方框就要結束。

繼續以同樣的方式呼叫 print 方法。這次是直接在 print 方法內就呼叫 device 物件個體的 write 方法。

類似這樣兩個以上的物件個體之間的動作都可以圖示出來。順序圖要跟著 lifeline、由上而下依序解讀。如果出現箭頭時，也要跟著去確認物件個體間的協調動作。