

# **O**bject-**O**riented **A**nalysis and **D**esign

## **U-02 Object Structure Modeling**

Neter Chao Ph.D.

# 內容大綱

- 導論
- 類別圖與物件圖
- 物件結構塑模
- 類別圖與物件圖建構案例
- 一般化的找尋
- 類別封裝
- 類別關聯性
- 結論

# 導論

- 在物件導向系統發展過程中，完成了使用個案塑模後，接下來便可進行物件互動行為塑模、物件結構塑模等活動，以開始架構系統。
- 類別圖是用來描述系統中物件的類型(Type)、類型間以及與子類型間之靜態關係等。
- 物件圖是用予描述一系統於某一時間點的靜態結構，該圖則是用來表達一個系統之複雜的資料結構，或藉由一時間序列的系統影像來表達系統的行為。

# 類別圖與物件圖(1/4)

- 類別圖(Class Diagram)有兩個主要元件：類別與類別間之關係。
- 物件圖也有兩個主要元件：物件與連結線。

# 類別圖與物件圖(2/4)

- 一群相關物件的定義、描述或樣版 (Definition, Description or Template) 稱為一個類別。

# 類別之表示符號

名稱
屬性
操作

# 類別圖與物件圖 (3/4)

- 類別種類
  - 執行觀點
    - 永存類別 (Persist Class)
    - 暫存類別 (Transient Class)
  - 實作觀點(B-C-E Model)
    - 實體類別 (Entity Class)
    - 邊界類別 (Boundary Class)
    - 控制類別 (Control Class)

# 類別圖與物件圖 (4/4)

- 屬性與操作種類
  - 類別的屬性（或操作）可以定義有哪些物件可以改變（或看到）該屬性（或操作），這就是類別屬性（或操作）的**可視性**（Visibility）。



# 可視性表示符號

分類	UML 表示法	Rose 表示法	權限
公開的	+		可被系統中所有類別存取
私有的	-		在巢狀類別或同一個類別內可被存取
保護的	#		在巢狀類別或同一個類別內可被存取
套件式或實作式	沒有圖示		只能被同一個套件內的其他類別存取





# 類別圖 - 類別間之關係(1/2)

- 類別間之關係指的是類別間的連結，在物件導向塑模中，類別間最重要的關係有相依(Dependency)、一般化(Generalization)、關聯(Association)與實現化(Realization)等四種關係。

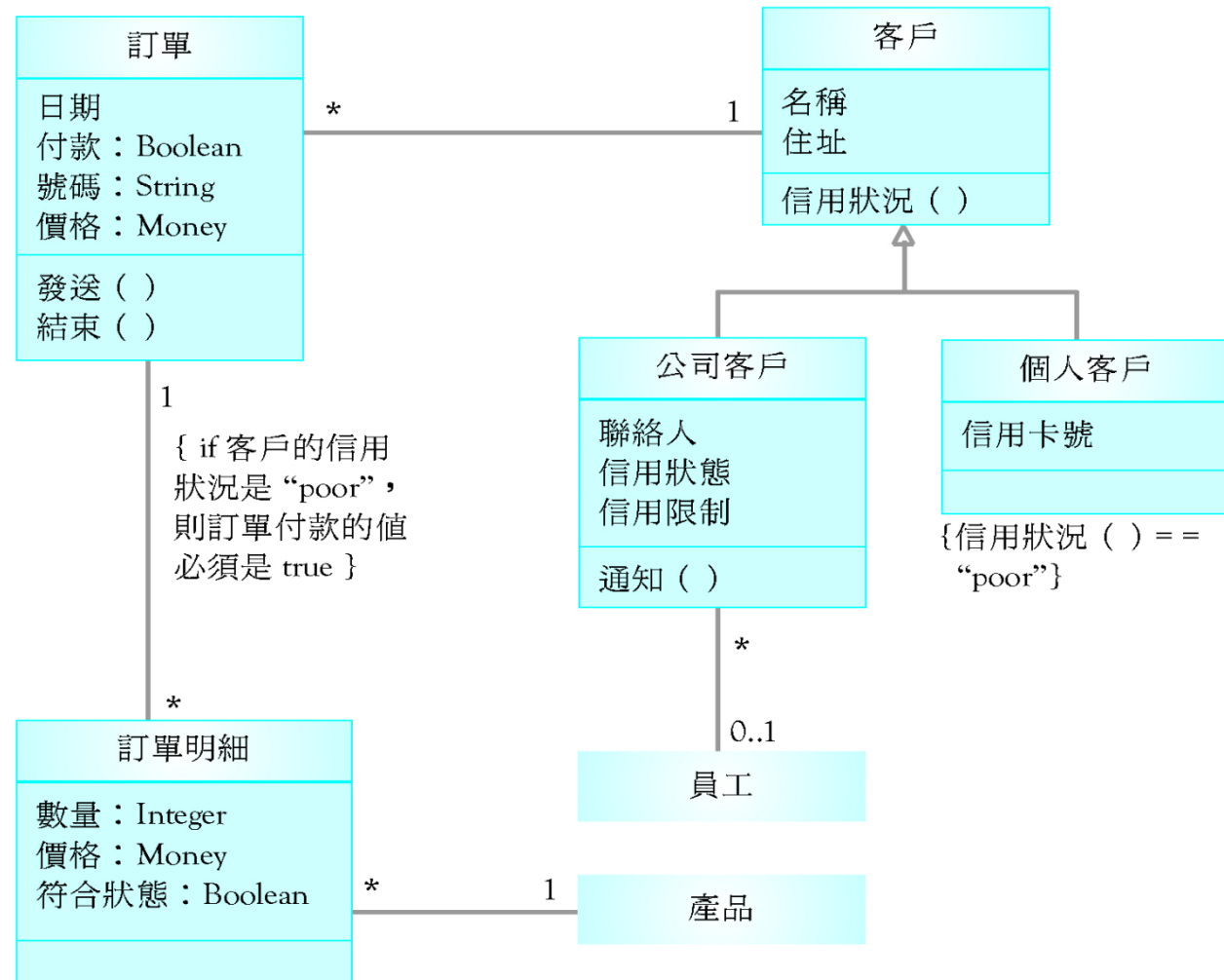
# 類別圖 - 類別間之關係(2/2)

- 相依關係
- 一般化關係
- 關聯關係
- 實現化

# 類別間之關係與表示符號

類別間之關係	符號
相依關係	
一般化關係	
關聯關係	
實現化關係	

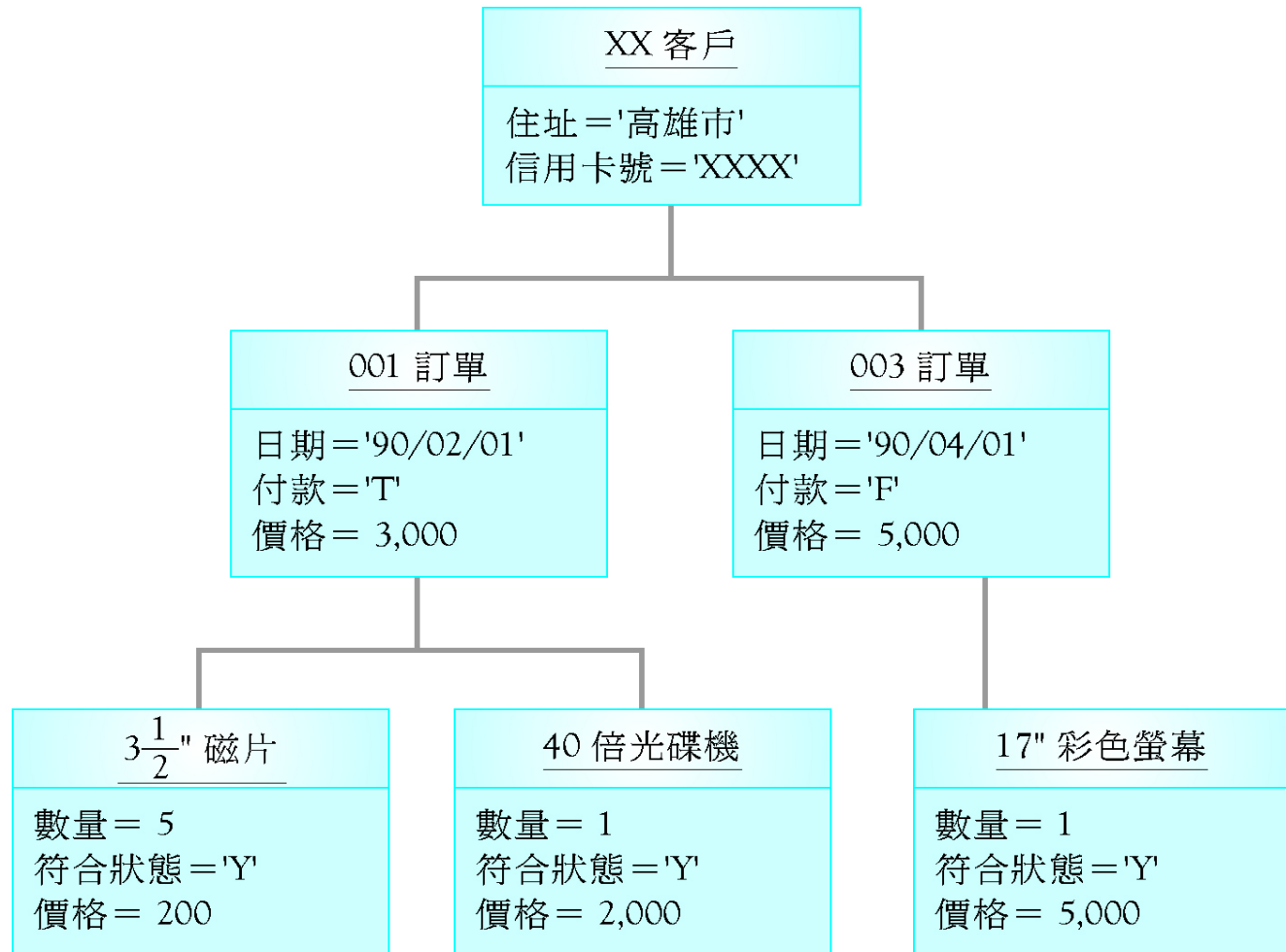
# 類別圖範例



# 物件圖

- 物件圖是用以描述一系統於某一時間點的物件靜態結構，或藉由一時間序列的系統物件影像來表達系統的行為。
  - 物件圖中有兩個主要元件：物件與連結線。

# 物件圖範例



# 物件結構塑模(1/2)

- 物件結構塑模之主要活動包括：找出類別（或物件）、類別之屬性、類別之操作、類別間之關係及建構類別圖與物件圖等，這些活動常以某順序反覆地被執行。
- 原則上，一個使用個案須建立一個類別圖。



# 物件結構塑模(2/2)

- 建構類別圖之步驟
  - 確認類別與操作
  - 確認屬性
  - 確認類別間的關係等
  - 彙製類別圖。

# 確認類別、屬性與操作(1/2)

- 確認類別與操作之準則
  - 循序圖中的邊界物件、控制物件以及實體物件分別對應到類別圖中的邊界類別、控制類別以及實體類別。
  - 循序圖中的每個物件間傳遞的訊息對應至訊息接受類別之操作。

# 確認類別、屬性與操作(2/2)

- 確認屬性之準則
  - 使用個案描述中的形容子句與所有格子句。
  - 一般如何描述這個類別？
  - 一般描述中那一個部份可以應用在這個問題領域中？
  - 介面類別中的屬性可從活動圖中的註記找出。

# 確認類別間之關係

- 假設有兩類別之物件A與B，若存在下列情況，則物件A與B間可能有關係。
  - 某一個類別會用到其他類別，且被使用的類別的改變可能會影響到使用它的類別
  - 某一類別僅具有另一類別之某些特定性質，例如屬性與操作
  - 某一類別之物件知道另一類別之物件的存在，或某一類別之物件使用到另一類別之物件的服務，但不是擁有此服務或訊息
  - 在關聯關係中，若某一方由另一方聚集而成，也就是有整體與其組件之關係
  - 由兩類別中找出需由兩類別之資料共同唯一決定的屬性作為關聯類別的屬性
  - 若某一類別之行為是由另一類別來描述

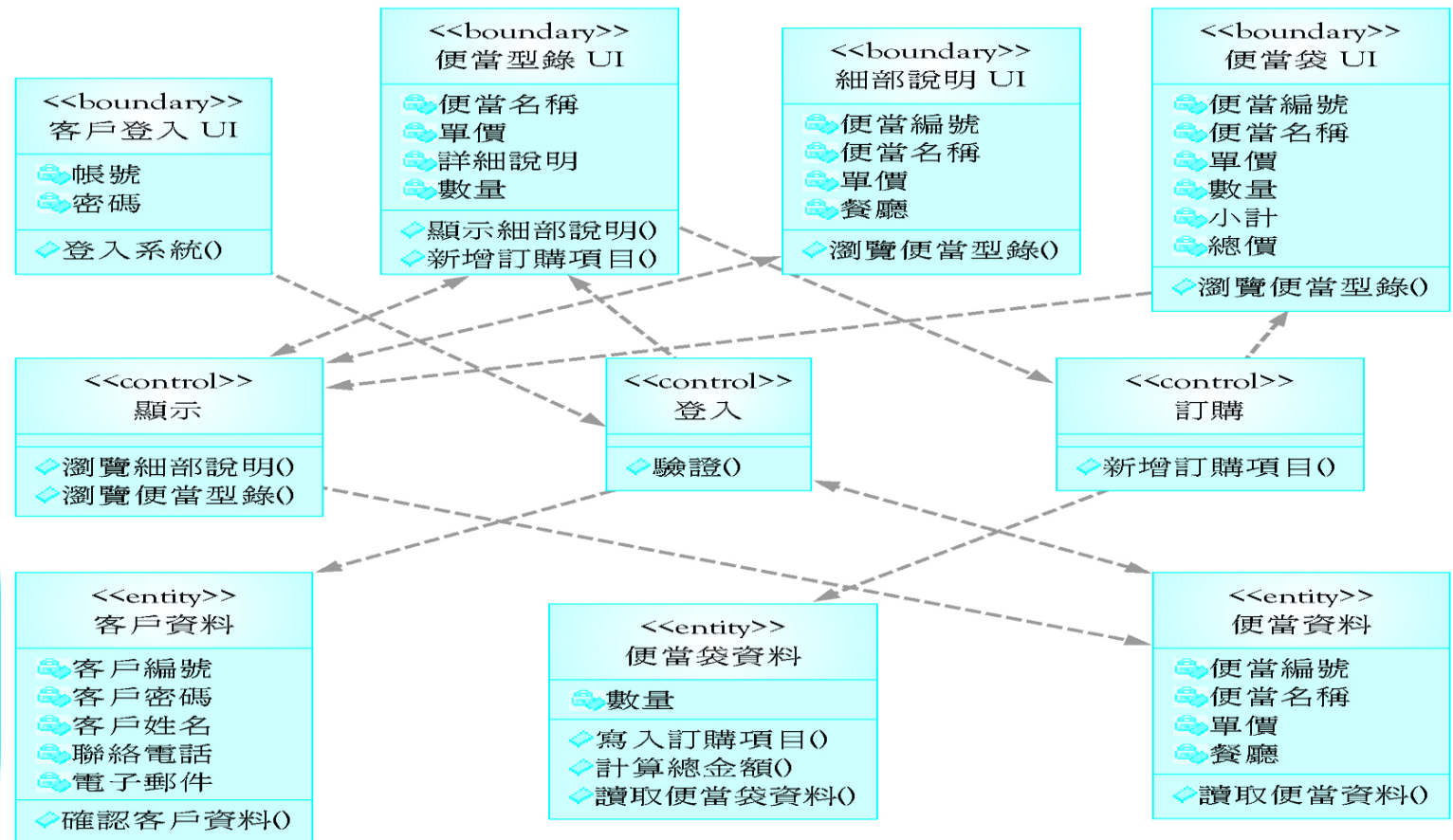
# 類別圖與物件圖建構案例(1/2)

- 以便當王公司之網路便當線上訂購系統為例，說明如何以類別圖進行物件靜態結構塑模。

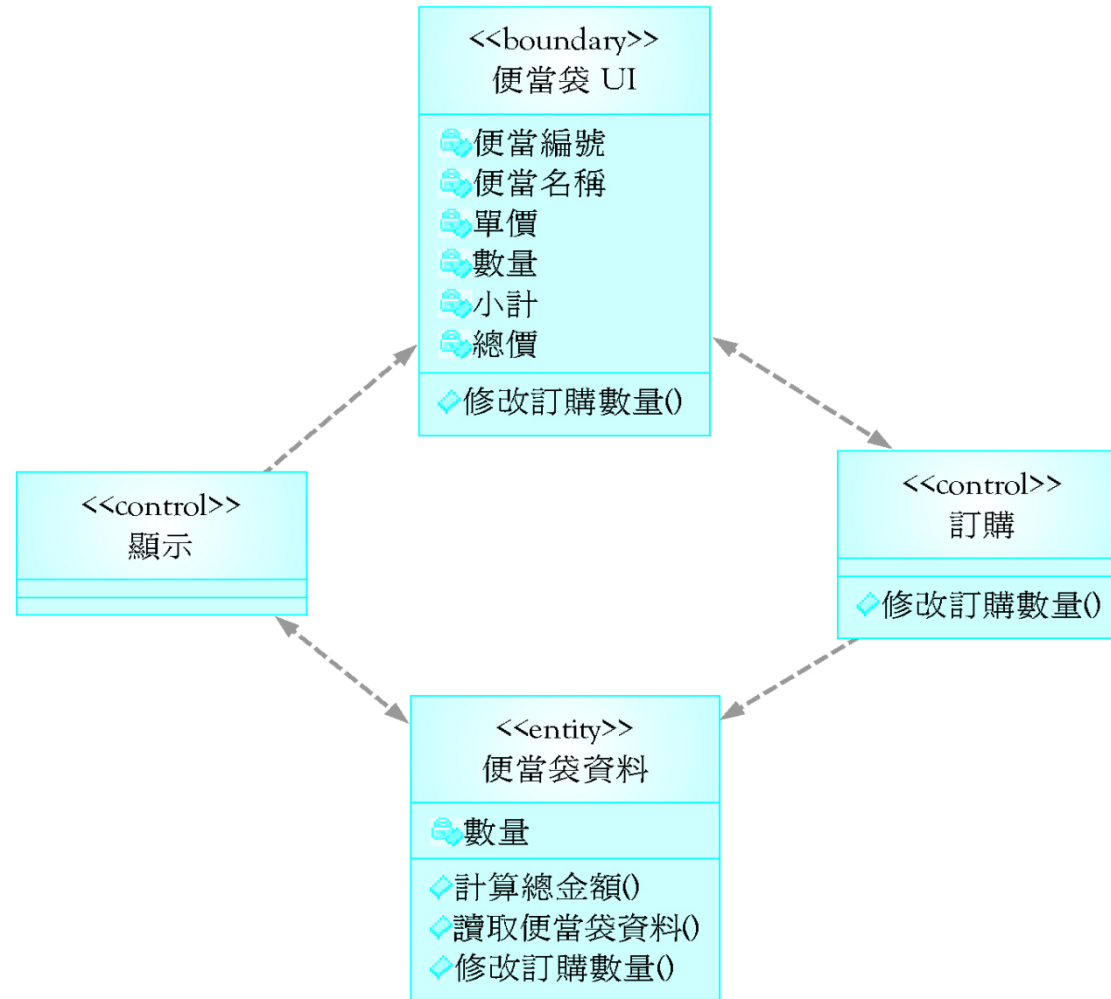
# 類別圖與物件圖建構案例(2/2)

- 新增訂購項目使用個案
- 修改訂購數量使用個案
- 刪除訂購項目使用個案
- 取消採購訂單使用個案
- 確認採購訂單使用個案

# 新增訂購項目使用個案之類別圖

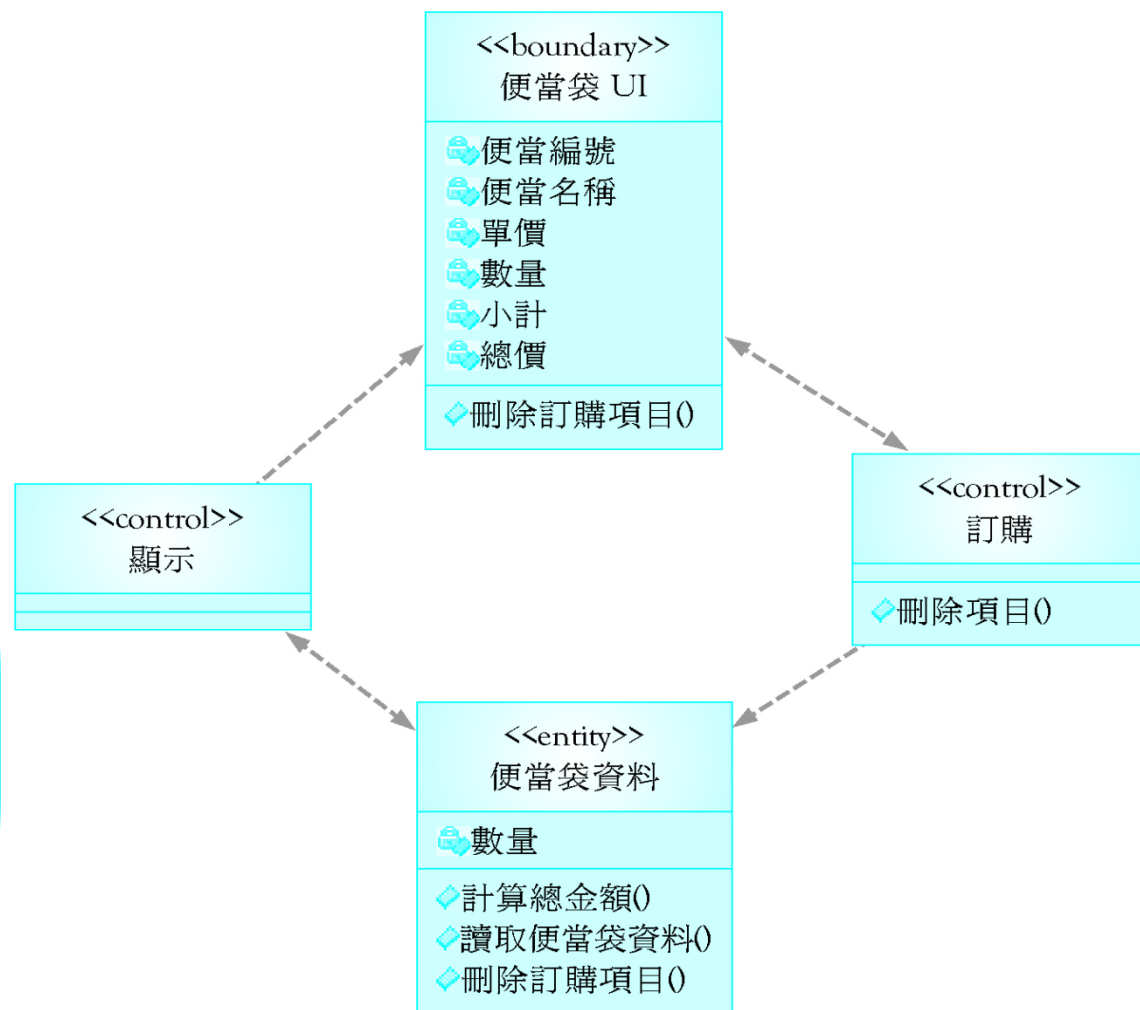


# 修改訂購數量使用個案之類別圖

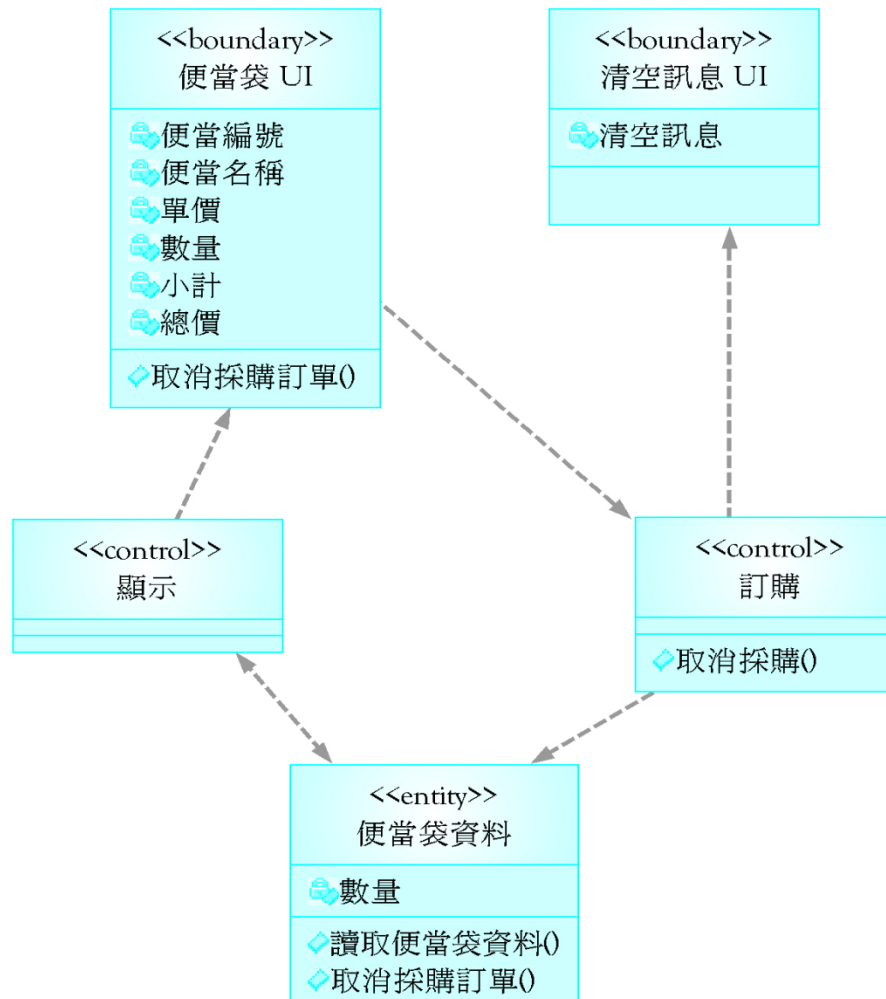




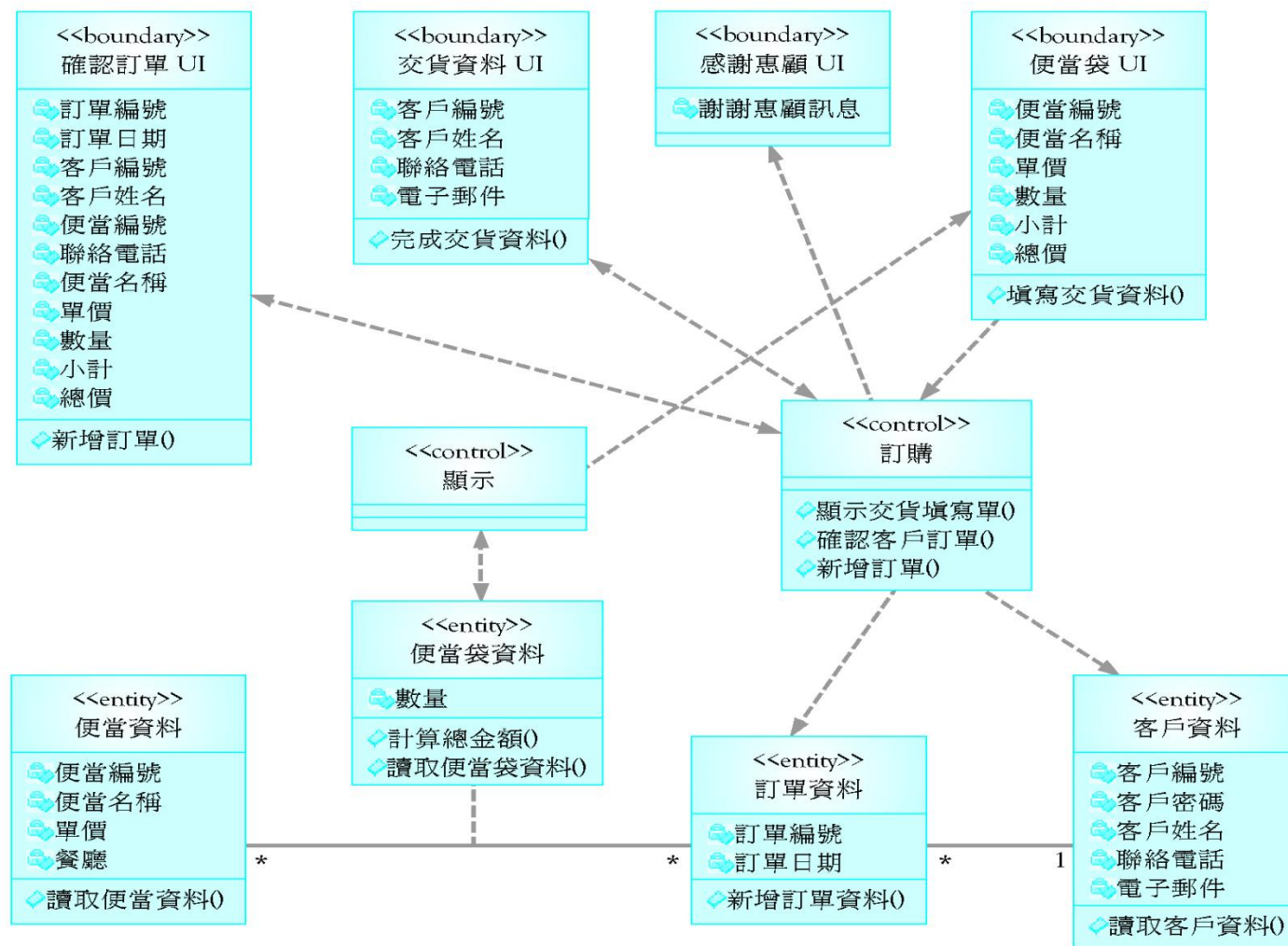
# 刪除訂購項目使用個案之類別圖



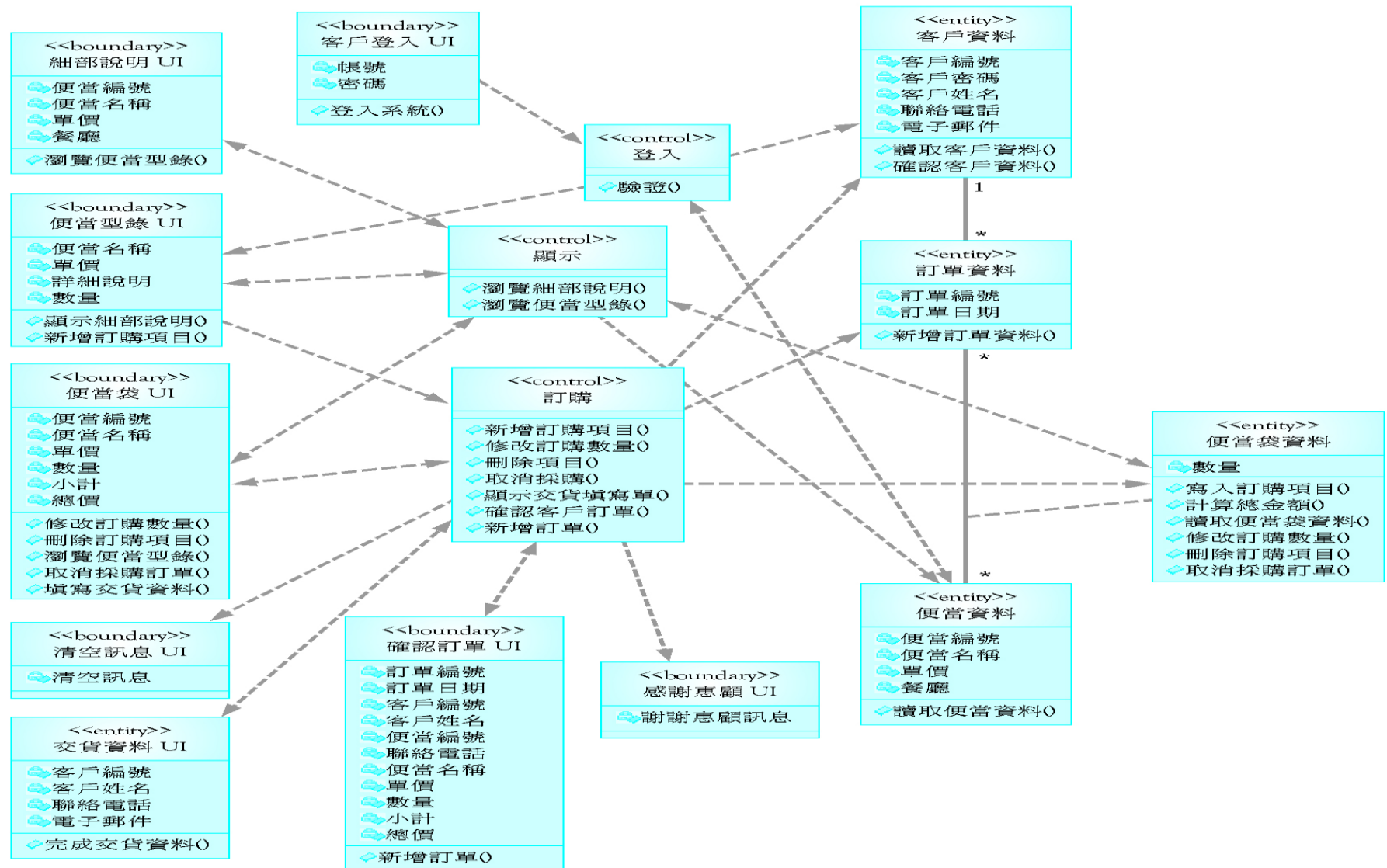
# 取消採購訂單使用個案之類別圖



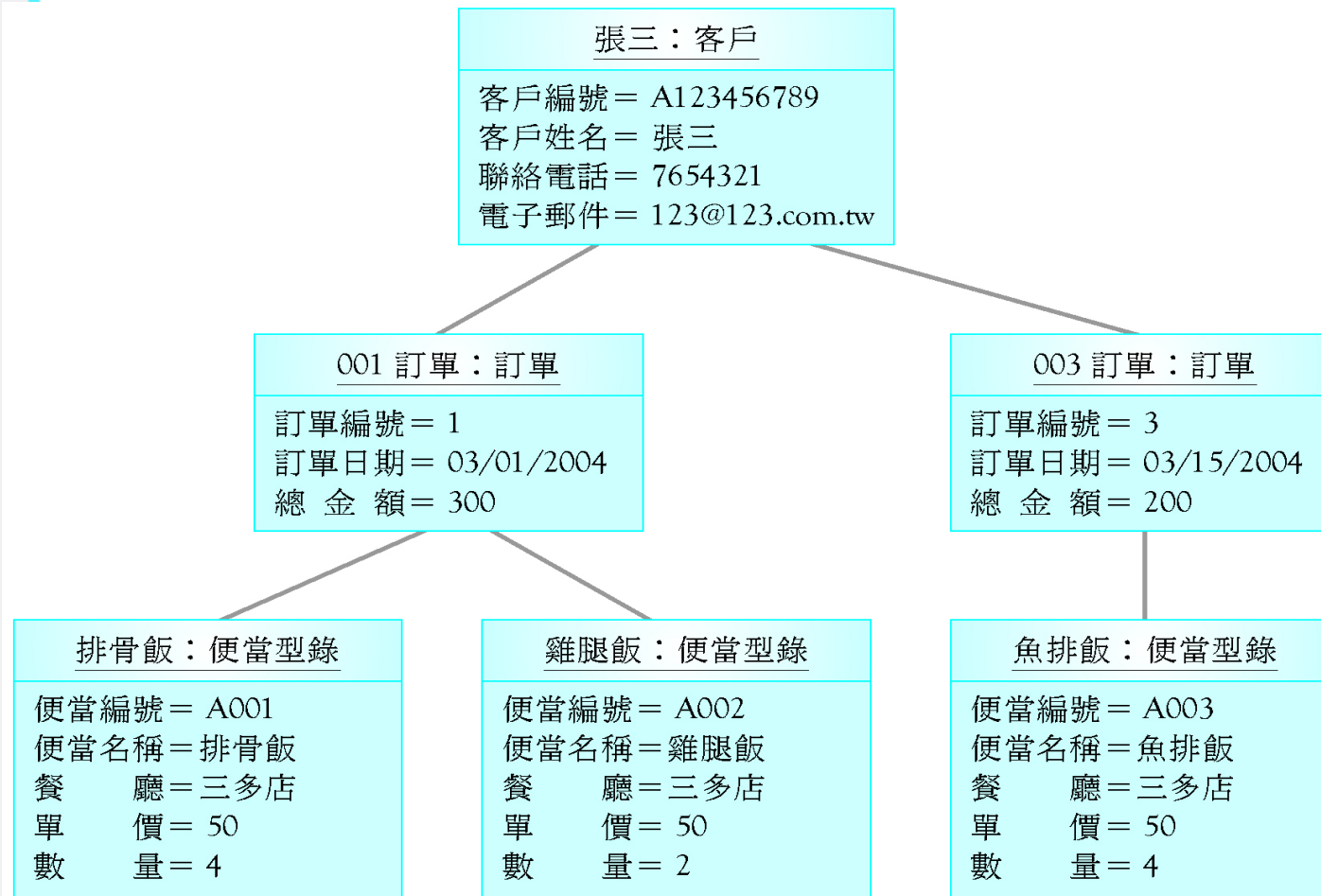
# 確認採購訂單使用個案之類別圖



# 便當王公司網路訂購系統之匯總類別圖



## 9.4.4 繪製物件圖



# 一般化的找尋

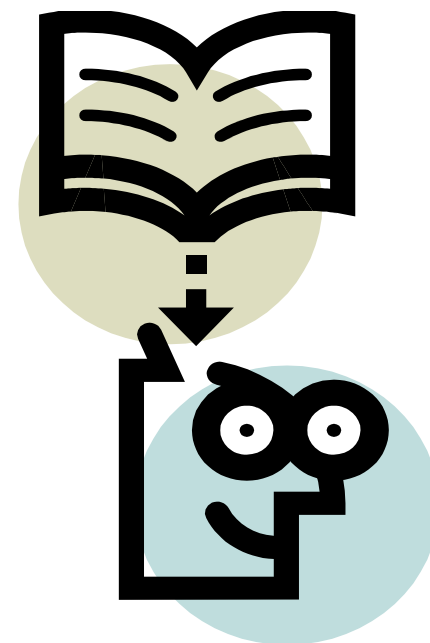
- 一般化的找尋是指有些物件具有某些共通特徵，可將這些共通特徵抽象化成另一個類別。

# 類別封裝

- 所謂類別封裝，就是找出類別的屬性與操作，並將之封裝在一起，以達到抽象資料類型之目的。
- 封裝類別時，必須將該類別分散在不同使用個案的屬性與操作彙整在一起。

# 類別的關聯性

- 類別的結合關係
- 可導覽的結合關係
- 一對多的類別關係
- 類別的相依關係





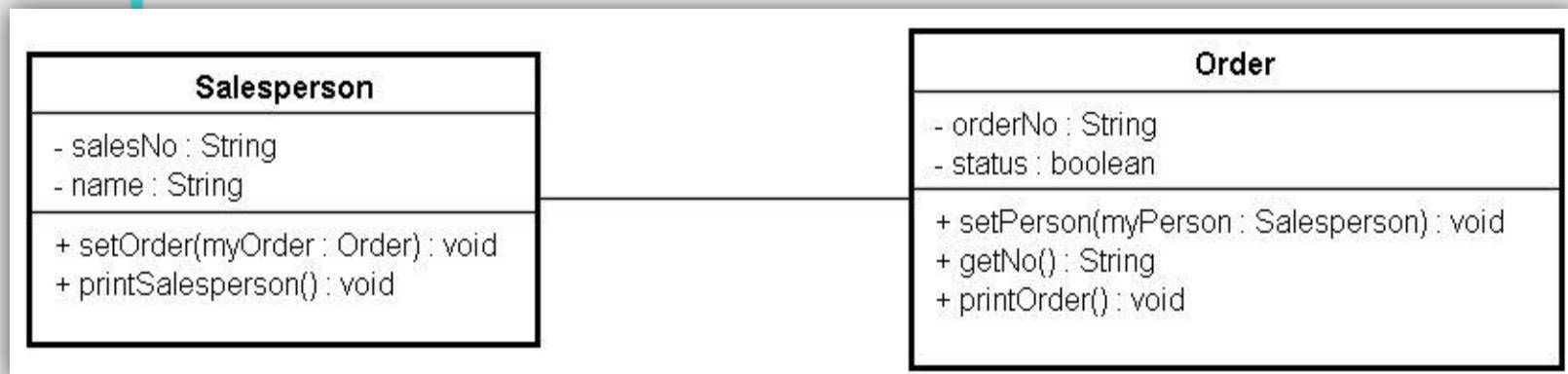
# 類別的關聯性

- 類別的**關聯性**（ Relationships ）是指類別間擁有的合作關係。在這一節筆者主要說明「二元結合關係」（ Binary Association ）即兩個類別間的關聯性和相依關係（ Dependency ）。
- 在Java類別宣告中，成員變數除了**基本資料型態**的變數外，也可以使用參考其他物件的**物件變數**，即**成員物件**。而類別的關聯性，就是使用成員物件來建立與其他類別的關係。

# 類別的結合關係-說明

- **結合關係** ( Associations ) 是地位相等的兩個類別，相互知道另一個類別存在的關係。我們可以將兩個類別視為扮演著不同的角色。
- 例如：Salesperson業務員拿到Order訂單，所以業務員知道接到哪一張訂單；Order訂單需要知道是屬於哪一位業務員的業績。

# 類別的結合關係-UML類別圖



# 類別的結合關係-Java實作

```
class Salesperson {  
    .....  
    private Order itsOrder;  
    .....  
}  
class Order {  
    .....  
    private Salesperson itsPerson;  
    .....  
}
```

# 類別的結合關係-另一種UML類別圖

- 在UML類別圖可以指明類別扮演的角色名稱（ Role Name ），如下圖所示：

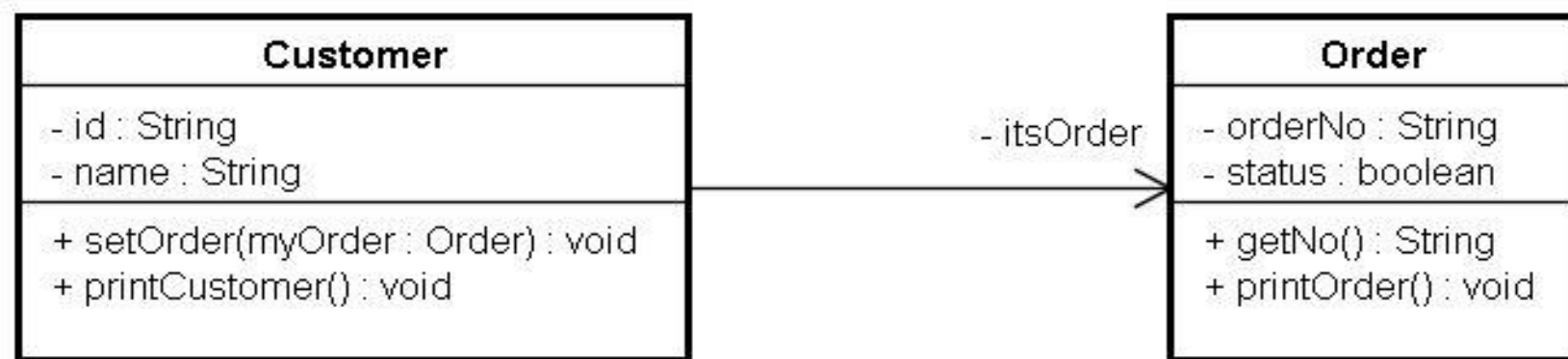


# 可導覽的結合關係-說明

- 可導覽的結合關係 ( Navigable Associations ) 是一種擁有方向性的結合關係。
- 若將結合關係視為雙向結合關係，可導覽結合關係是一種單方向的結合關係。在兩個類別中，只有其中一個類別可以取得另一個類別，反過來並不無法存取。

# 可導覽的結合關係-UML類別圖

- 例如：我們只允許從Customer客戶類別查詢Order訂單資料，就是一種可導覽的結合關係，如下圖所示：



# 可導覽的結合關係-Java實作

- 在類別宣告使用物件變數itsOrder參考其他物件，其類別宣告如下所示：

```
class Customer {  
    .....  
    private Order itsOrder;  
    .....  
}  
class Order { ..... }
```

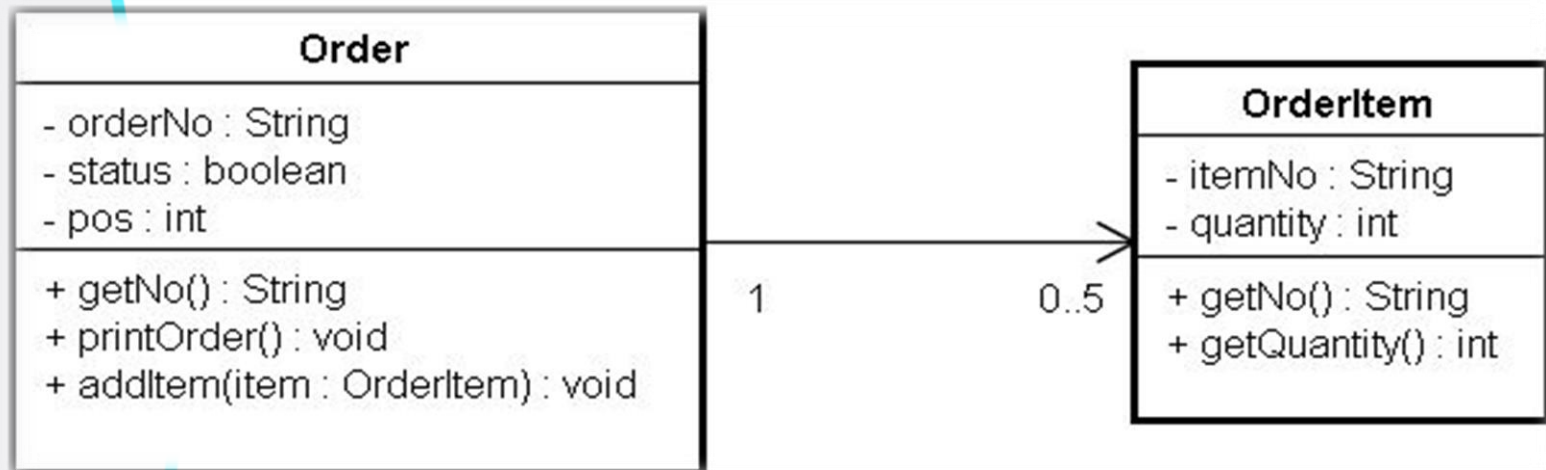


# 一對多的類別關係-說明

- 一對多的類別關係是在結合關係上指定一邊的**多重性** ( Multiplicity ) 為**1**，另一邊為**1..n**或**0..n**，此時的結合關係是1個類別對多個類別。
- 常見的範例有：公司擁有多位員工、訂單擁有多個項目和老師擁有多位學生等類別關係。

# 一對多的類別關係-UML類別圖

- 例如：Order訂單擁有0~5個OrderItem訂單項目的購買商品，此種類別關係就是一種一對多的類別關係，如下圖所示：



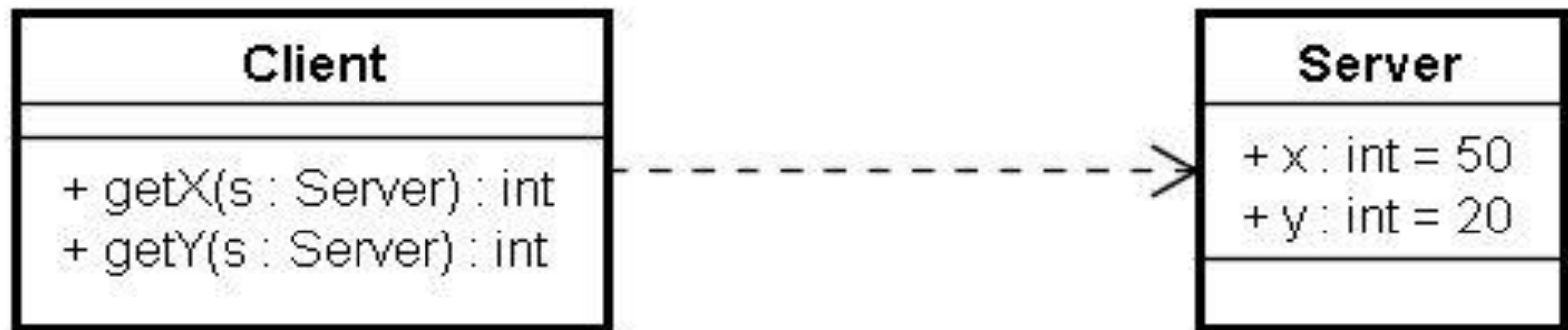
# 一對多的類別關係-Java實作

- Java類別實作一對多關係是建議使用集合物件，在此是以物件陣列實作，其類別宣告如下所示：

```
class Order {  
    .....  
    private OrderItem[] itsItem = new OrderItem[5];  
    public void addItem(OrderItem item) { }  
    .....  
}  
class OrderItem { ..... }
```

# 類別的相依關係-說明1

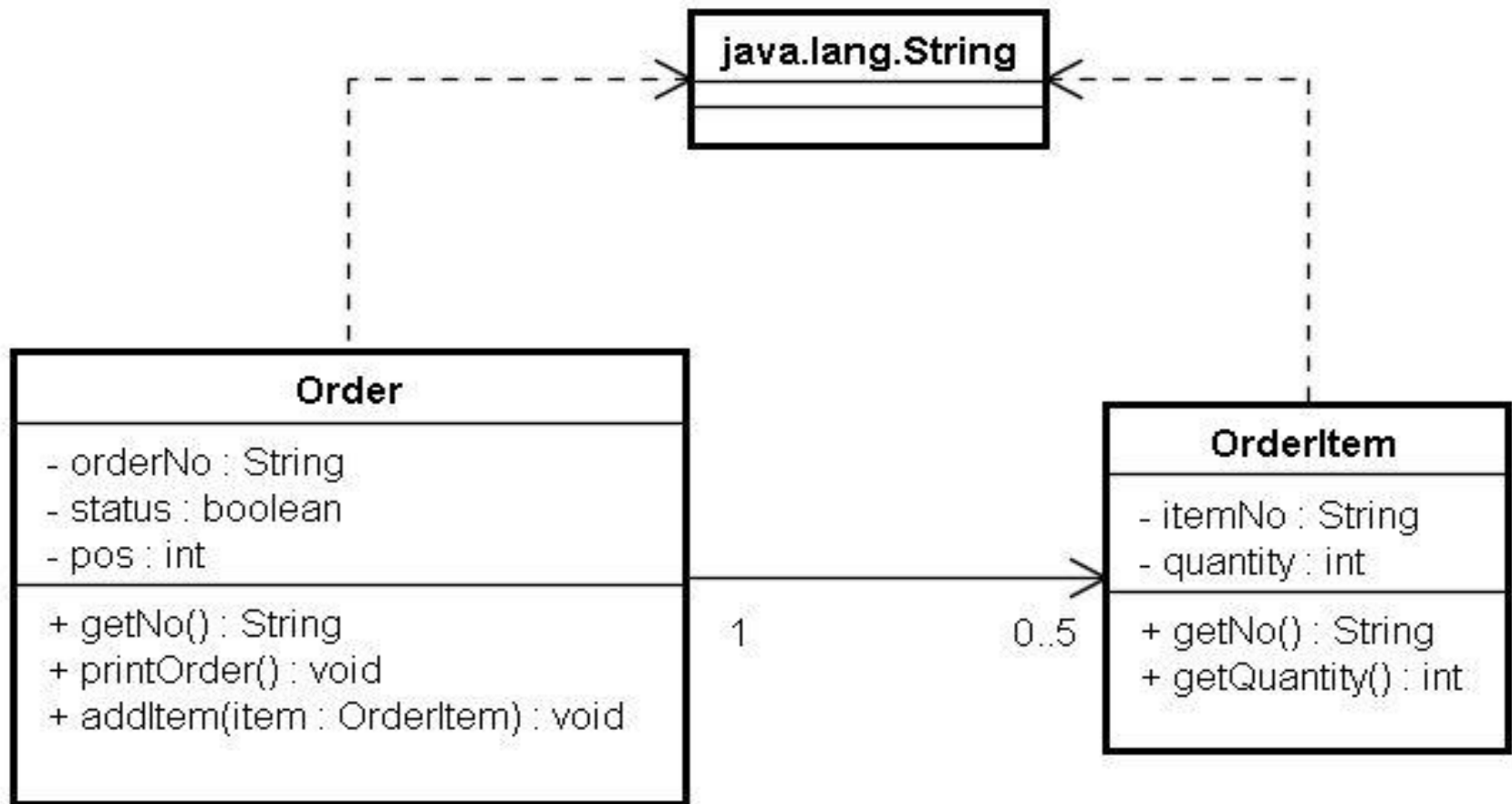
- 類別的相依關係 ( Dependency ) 是指類別的設計與實作需要依賴其他類別，屬於一種非常弱的結合關係。當類別是相依其他類別時，其他類別的更改將會影響到第1個類別。
- 例如：Client類別的方法參數或傳回值是使用Server類別的物件，如下圖所示：



# 類別的相依關係-說明2

- 換句話說，當Server類別更改時，Client類別也需要跟著修改，Server類別稱為「獨立類別」(Independent Class)，Client類別稱為「相依類別」(Dependent Class)。
- 事實上，在Java程式使用的String物件就是java.lang.String類別的物件，所以類別使用String物件作為方法參數或傳回值時，類別與String物件就擁有相依關係。

# 類別的相依關係-UML類別圖



# 巢狀類別-說明

- 巢狀類別強調類別間的關聯性，強調內層類別一定需要外層類別，如果外層類別的物件不存在，內層類別物件也不會存在，內層的成員類別稱為「內層類別」（Inner Classes）。

# 巢狀類別-巢狀類別的宣告

- Order巢狀類別的宣告，如下所示：

```
class Order { // Order外層類別
```

```
.....
```

```
class OrderStatus { // OrderStatus內層  
類別
```

```
.....
```

```
}
```

```
.....
```

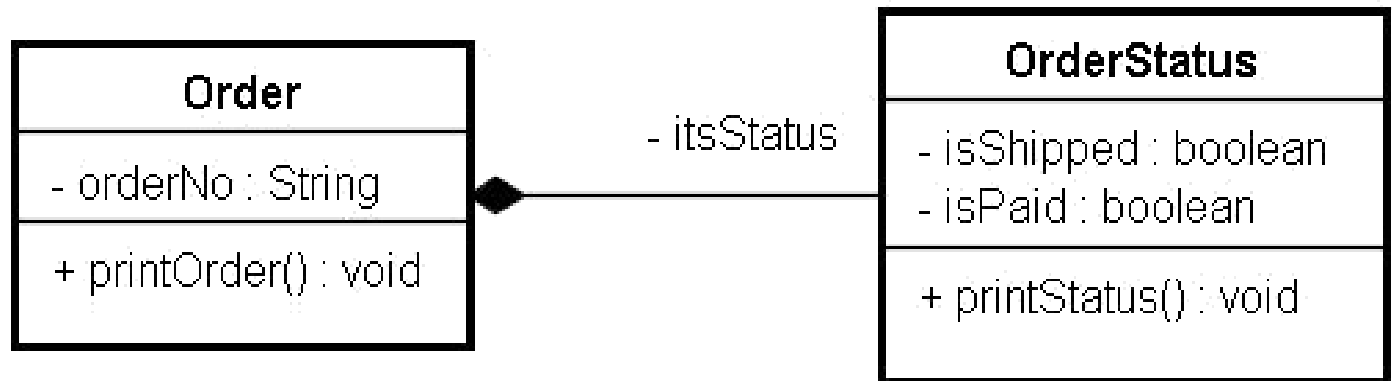
```
}
```

- Order類別擁有成員類別OrderStatus的內層類別，Order是巢狀類別的「外層類別」（Enclosing Class）。



# 巢狀類別-UML類別圖

- UML類別圖的組成關係是一種成品和零件 ( Whole-Part ) 的類別關係，強調是成品的專屬零件，如下圖所示：

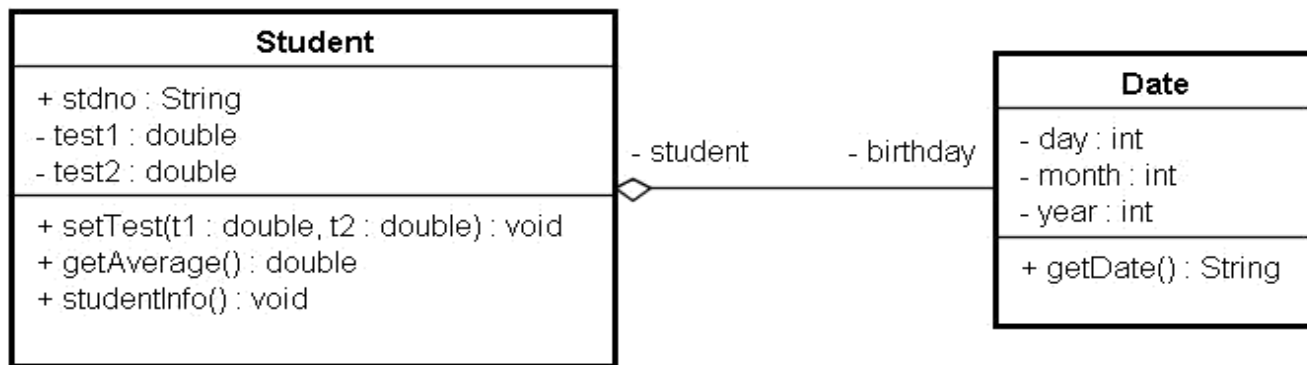


# 類別的聚合關係

- 在Java程式實作聚合關係和結合關係相同，其差異在於聚合關係的兩個類別擁有成品和零件（Whole-Part）的類別關係，並不是地位對等的兩個類別。
- 聚合關係和上一節組成關係的差異，在於組成關係的零件是專屬零件，所以組成關係的零件並不能單獨存在。聚合關係的零件可以共用，而且零件的物件可以單獨存在。

# 一對一的聚合關係-UML類別圖

- 一對一的聚合關係是指類別中擁有一個物件變數參考到其他類別的物件，此類別是成品（Whole），其他類別是零件（Part）。
- 例如：Student類別擁有Date類別的生日，生日是學生的零件。UML類別圖，如下圖所示：



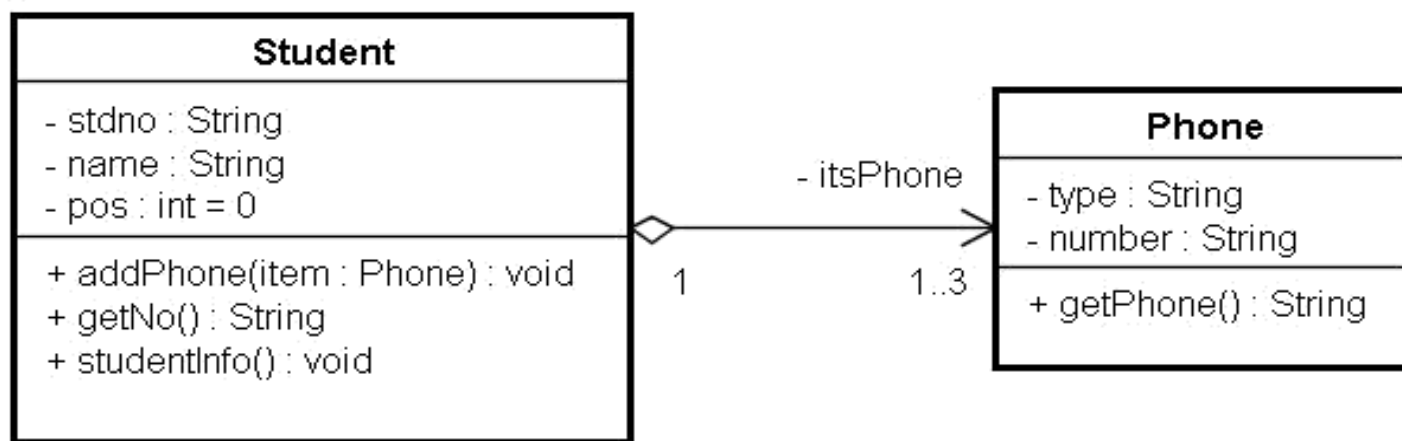
# 一對一的聚合關係-類別宣告

- Java程式碼也是使用物件變數參考其他物件，其類別宣告如下所示：

```
class Student {  
    private Date birthday;  
    .....  
}  
class Date {  
    private Student student;  
    .....  
}
```

# 一對多的聚合關係-UML類別圖

- 一對多的聚合關係是指**1個類別**對**多個類別**，也就是**成品**需要同樣的**多個零件**。例如：  
：一輛車有4個輪胎，Student學生擁有住家電話、宿舍電話和手機等多個Phone電話物件。UML類別圖，如下圖所示：



# 一對多的聚合關係-類別宣告

- 使用物件陣列來實作一對多的聚合關係，如下所示：

```
class Student {  
    .....  
    private int pos = 0;  
    private Phone[] itsPhone = new  
        Phone[3];  
    .....  
}  
class Phone { ... }
```

# 結論

- 物件結構塑模是物件導向系統分析與設計過程中很重要的一環，主要應用類別圖與物件圖來表達一個系統類別間之靜態結構關係及類別內部之屬性與操作資訊等。
- 當每個使用個案之類別圖完成後，需將各類別圖整合成一個完整的類別圖。
- 整合後的類別圖需經過正規化之檢驗與修飾，以利下一階段之資料庫設計。