<<會計資訊系統課程講義>>

統一塑模語言(UML)語法精要

-- 物件導向概念、需求分析及系統分析

周國華

屏東商業技術學院會計系

初版：2006/8/4

本次修訂：2010/8/25

智慧財產權聲明

- 本文件係由周國華老師獨自撰寫，除引用之概念屬於原文作者外，其餘文字及圖形內容之智慧財產權當然屬於周老師獨有。
- 任何機構或個人，在未取得周老師同意前，不得直接以本文件做為學校、研究機構、企業、會計師事務所、政府機關或財團法人機構舉辦教學或進修課程之教材，否則即屬侵權行為。
- 任何機構或個人，在未取得周老師同意前，不得在自行編撰的教材中直接大量引用本文件的內容。若屬單頁內部分內容之引用，亦請註明出處。

物件導向(OO) 2-1

- Object-oriented (OO)：應用程式(application)由可重複使用的軟體物件(object)或元件(component)組合而成。
  - 軟體物件可用來描述實體物件以及抽象概念。
  - 元件是由功能相關的物件組合而成。
- OOAD：物件導向分析(analysis)與設計(design)。
- OOP：物件導向程式設計(programming)。
- OO語言：
  - 最早具有OO重要特色的語言：Simula (1967)
  - OO理論據以發展的語言：SmallTalk (1972~1980)
  - 目前主流OO語言：C++，JAVA，Ruby，Python，C#，VB.NET。
  - 極具親和力的3D教學用OO語言：Alice 2.2, 3.0

※ 中國大陸把OO翻譯為「面向對象」，OOP為「面向對象編程」。

物件導向(OO) 2-2

- 傳統(結構化)系統開發：以資料為中心(data-centric)，強調資料的蒐集、管理及表達。
  - 資料庫的設計及建立是重點。
  - 可輕易處理資料庫的變動。
  - 當企業規則或系統行為改變時，較難處理。
- OO系統開發：資訊與行為並重，所建立的系統較具彈性，能更有效處理企業規則或系統行為的變動。
- 描述資訊與行為，不同語言有不同構念名稱：
  - Java稱為variable(變數)及method(方法)，C++稱為variable及function(函數)，VB.VET稱為variable、function及sub(副程式)。

OO概念：抽象化

- 將真實世界的複雜現象以簡化的模型加以描述，稱為抽象化(abstraction)。在OO中，類別(class)及其物件就是抽象化的表徵。
- 例如，在學校管理系統中，有「學生」這個類別，此類別可能有身份證字號、學號、姓名、性別、生日、住址等變數，及註冊、選課、申請成績單、畢業離校等方法。在真實世界中，任何一位學生的特質及能力都遠超過上述「學生」類別所描述的內容，但就學校管理系統而言，上述類別的描述或已足夠。

OO概念：封裝 2-1

- 在OO中，把提供特定功能的變數及方法放在一個物件內，稱為封裝(encapsulation)。
- 例如：銀行系統的「帳戶」物件
  - 變數：編號，餘額，客戶名稱，地址，帳戶類型，利率，開戶日期。
  - 方法：開戶，結清，存款，提款，更改帳戶類型，更改客戶名稱，更改地址.....

※ 在編程時，先定義類別(class)，再實作特定類別的物件。

OO概念：封裝 2-2

- 封裝的優點：
  - 模組化(modularity)：每個類別的程式碼可以單獨撰寫，並可重複再用。
  - 資訊隱藏(information hiding)：物件透過公共介面與其他物件溝通，物件內的變數及方法不必公開。

OO概念：繼承

- 在OO中，子類別(subclass)可繼承(inherit)父類別(superclass)的所有變數及方法，再增添額外的變數及方法。子類別亦可覆寫(override)繼承自父類別的方法，提供新的詮釋方式。
- 子類別的一個物件，也是其父類別的物件。反之不然！
- 在Java中，一個子類別只能繼承一個父類別；在C++中，一個子類別可以繼承多個父類別。但Java的類別可透過實作interface的方式，實質上取得多重繼承的優點。

OO概念：多型

- 在OO中，方法名稱相同，卻容許有不同的運作內涵，稱為多型(polymorphism)：
  - 子類別繼承自父類別的方法並加以覆寫。
  - 多個子類別繼承自同一父類別的抽象方法，再各自定義實質內涵。
  - 多個類別實作同一個介面(interface)，並各自定義介面內的抽象方法之實質內涵。
  - 一個類別內有多個方法名稱相同，但各方法的參數不同。此情況通稱為overloading (方法超載)。

UML與OO

- Unified Modeling Language (UML)：統一塑模語言，是物件導向分析與設計的標準工具語言，亦可用來描述企業程序。
  - UML 2.0並未完整支援data modeling，但class diagram可提供類似ERD的資料塑模功能。
- 塑模(modeling)：開發資訊系統時，必須先確認使用者需求，並將此需求以通用的圖形及語法建立成視覺模型(visual model)，以便有效傳達給程式設計師。

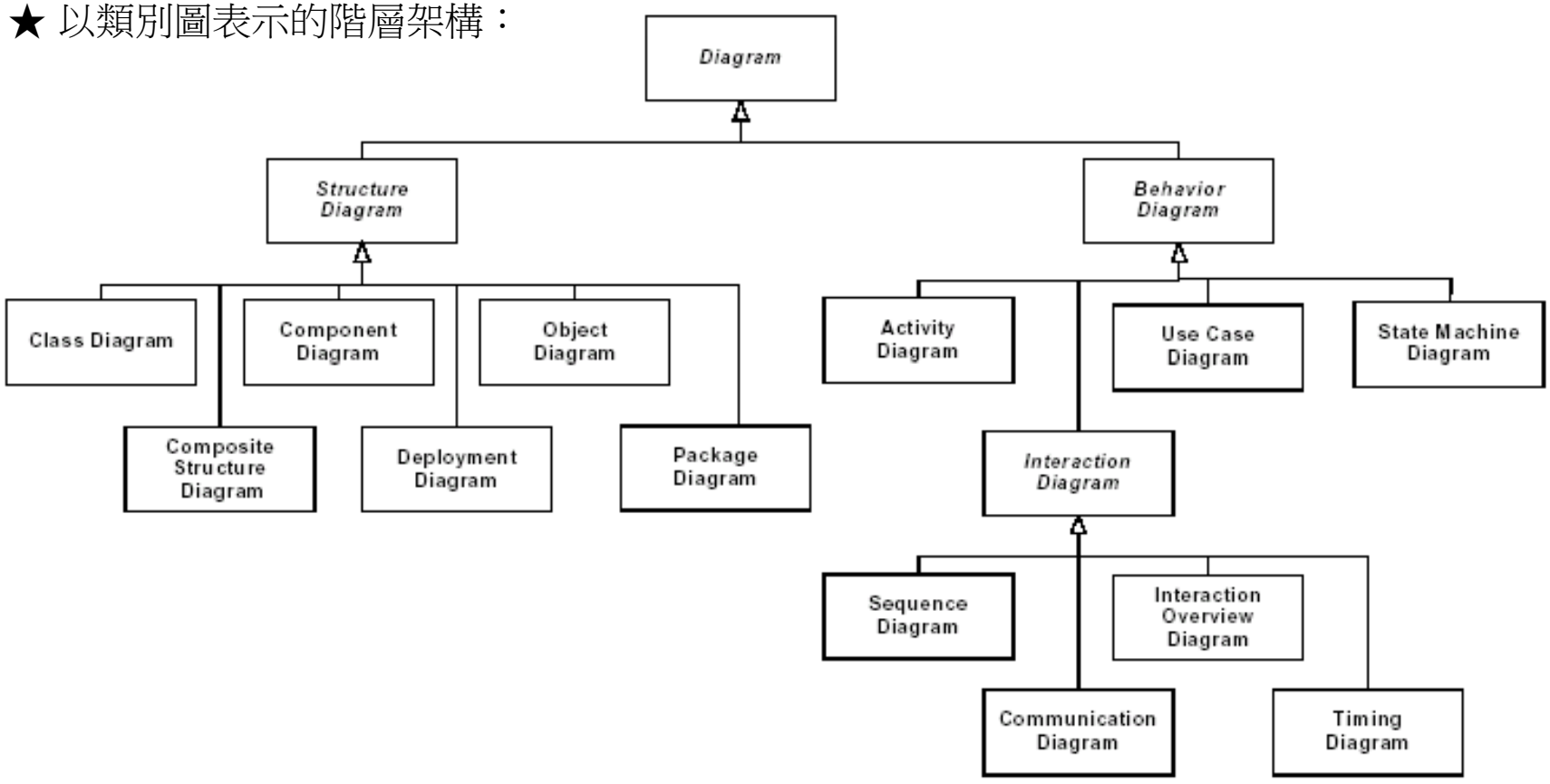
UML 2.0的圖形：分類名稱

- UML 2.0將圖形分成三大類，共13種圖：
  - 結構圖形：**類別圖**，物件圖，元件圖，複合結構圖，佈署圖，套件圖。
  - 行為圖形：**使用案例圖**，**活動圖**，狀態機器圖。
  - 互動圖形：**順序圖**，溝通圖，計時圖，互動觀點圖。

  * 本課程將介紹四種與需求分析及系統分析相關的圖形：類別圖、使用案例圖、活動圖及順序圖。

UML 2.0的圖形：階層架構

★ 以類別圖表示的階層架構：

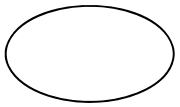
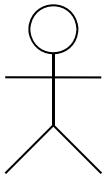


* 本圖取材自http://upload.wikimedia.org/wikipedia/commons/7/74/Uml_diagram.svg

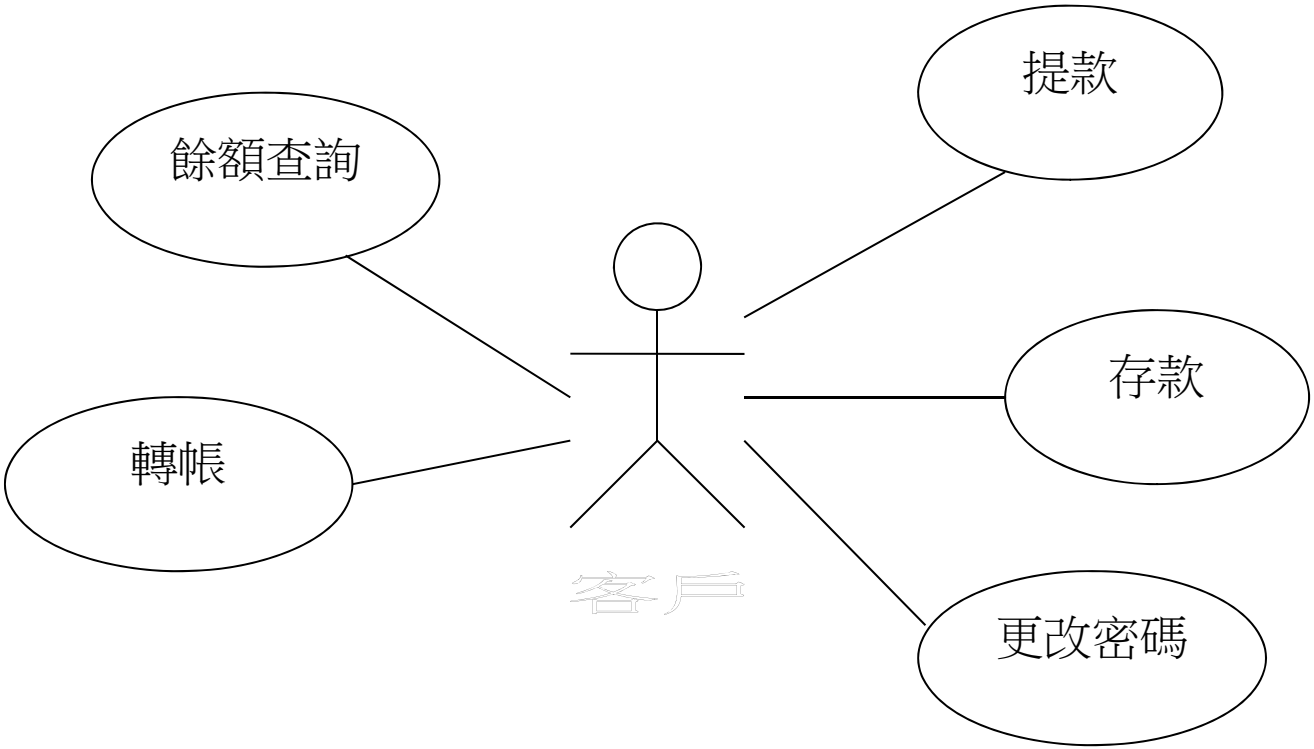
塑模程序：OOAD

1. 確認需求：Use Case Model
   - 從使用者取得完整的需求資料。
     - UML圖形工具：使用案例圖，活動圖。
2. 系統分析：Conceptual Model or Analysis Model
   - 將需求資料轉成開發者觀點。
     - UML圖形工具：概念類別圖，系統順序圖。
3. 系統設計
   - 將概念模型轉成可供特定程式語言實作的觀點。
     - UML圖形工具：設計類別圖，物件順序圖，溝通圖……
4. 程式設計

使用案例圖

- 使用案例圖(use case diagram, ucd)：此圖可表達使用者對系統功能的期待，每個use case代表使用者認定系統應提供的某項功能。與該系統互動的人(使用者、維護者)或其他系統，在ucd中稱為角色(actor)。
- 符號：
  - 使用案例： 
  - 角色： 

使用案例圖範例：ATM



問題討論

- ATM的UCD中，應不應該包含「確認密碼」及「列印報告」這兩個use case？

UC之間的關係：include

- include (包含)：uc A----<<include>>--→ uc B，表示在uc A (此為base uc)內的活動流程遇到inclusion point B時，必須轉移至uc B，在完成uc B的活動流程後，再回到uc A內完成剩下的活動流程。
- 優點：
  - 大而複雜的uc可拆解成多個小而簡單的uc。
  - 一個uc可被多個uc包含，故可將通用的程序抽離成通用uc。

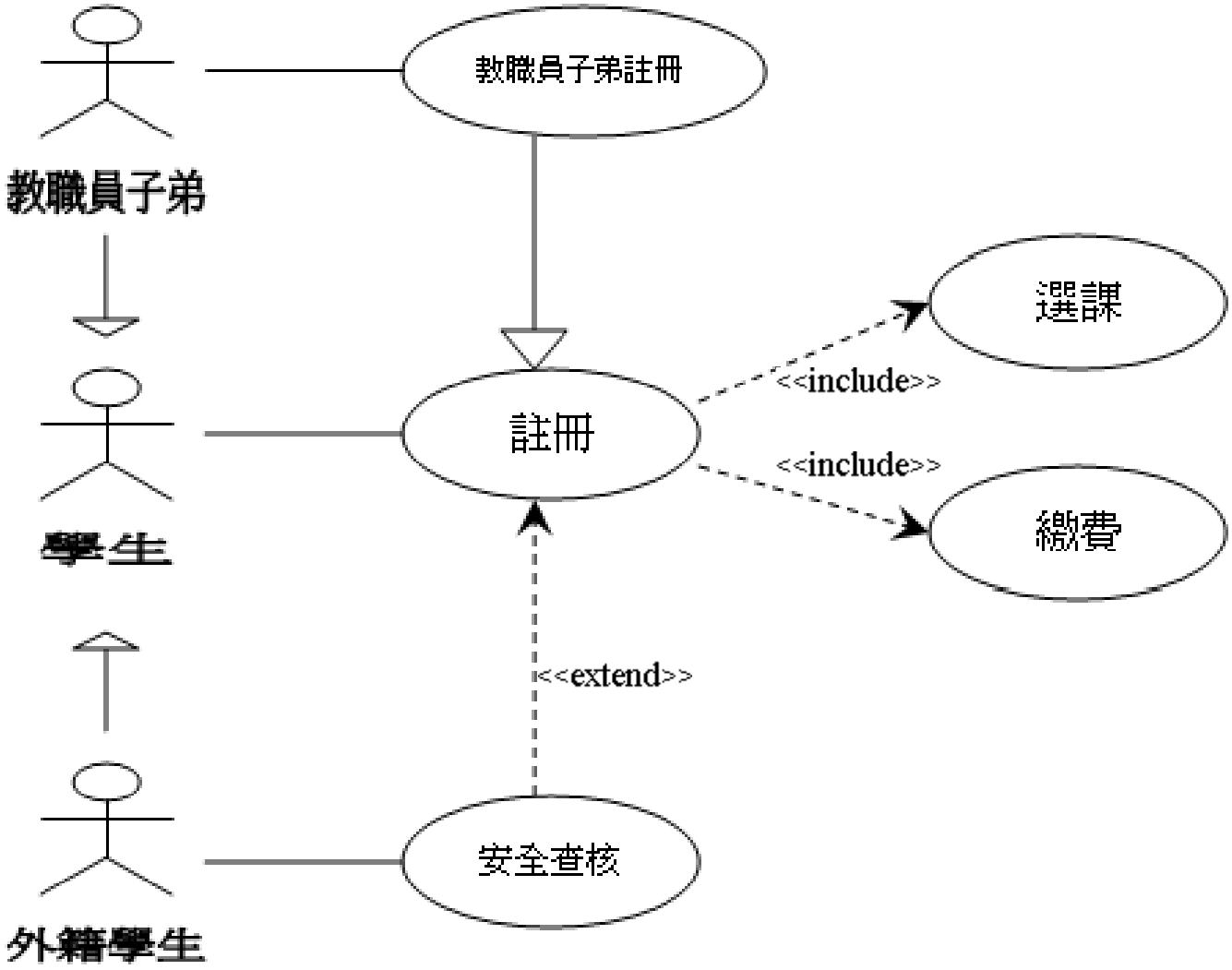
UC之間的關係：extend

- extend (延伸)： uc X----<<extend>>-⇢ uc Y，表示在uc Y (此為base uc)內的活動流程遇到extension point時，需判斷uc X是否滿足延伸條件，如為「否」，則略過uc X，繼續uc Y的後續流程；如為「是」，則轉移至uc X，在完成uc X的活動流程後，再回到uc Y內完成剩下的活動流程。
- 優點：當系統內容需與時俱進時，可將改變的部分放在extension uc內，如此可讓base uc維持穩定性。

繼承關係

- UC間的繼承關係：uc O —▷ uc P，表示uc O繼承uc P。通常uc O會改寫uc P內的部分程序。
- 角色間的繼承關係：actor A —▷ actor B，表示actor A繼承actor B。

使用案例圖範例：註冊



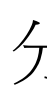
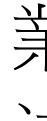
問題討論

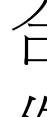
- 「uc 教職員子弟註冊」與「uc 註冊」的潛在差異為何？
- 「教職員子弟」這個角色應不應該 繼承「學生」角色？還是應該獨立？

活動圖

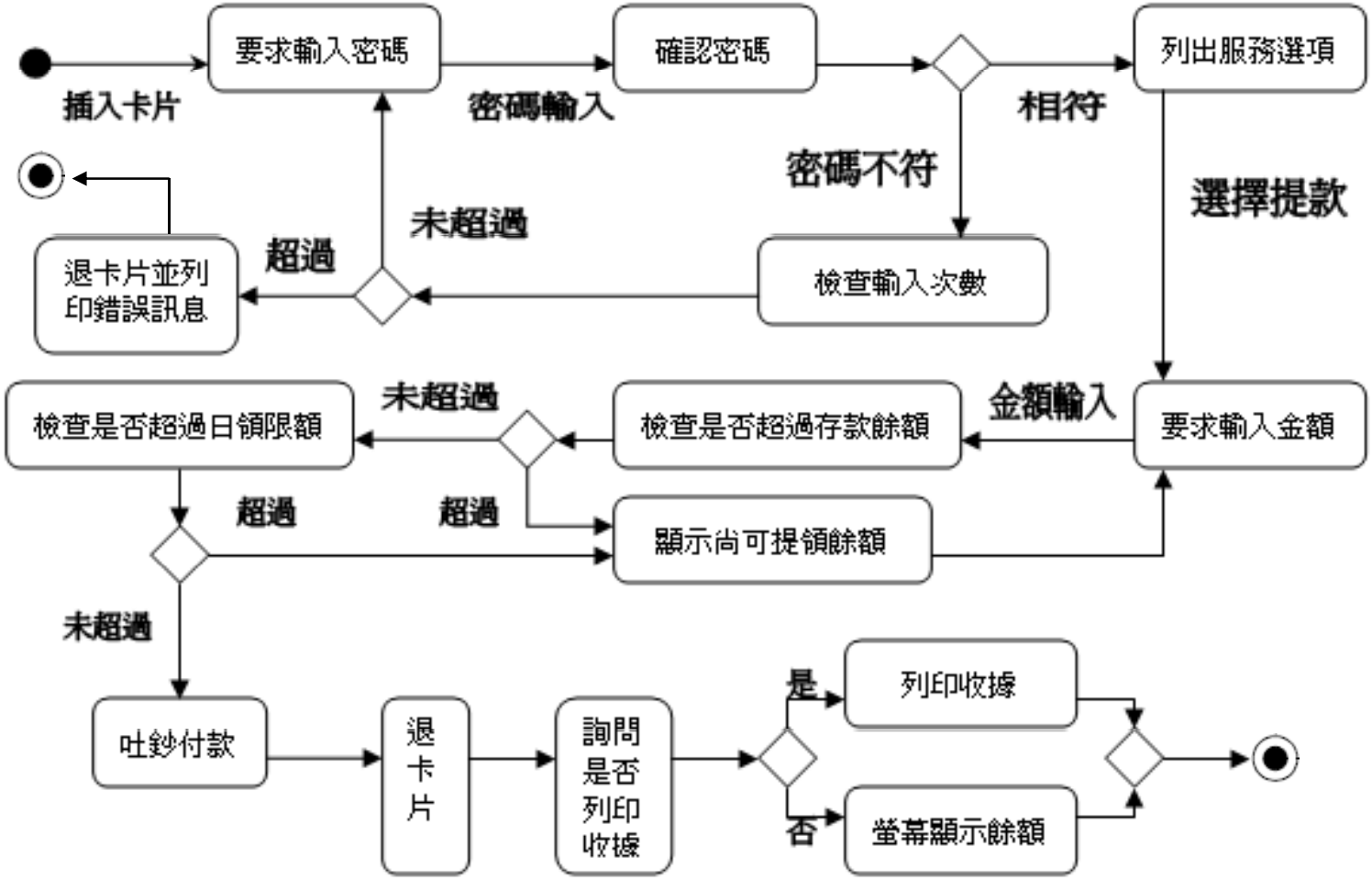
- 活動圖(activity diagram，或稱作業圖)：此圖可用來描述
  - 個別使用案例內的詳細作業流程。
  - 企業程序。
  - 企業規則的細節。
- 在傳統結構化分析中所使用的DFD及system flowchart，在OOAD中可用活動圖代替。

活動圖：符號

- 符號：
  - 作業起點： 
  - 作業終點(可有多個)： 
  - 分岔點(fork，一作業進、多作業同時出)： 
  - 會合點(join，上述多個平行作業同時進、一作業出)： 
  - 決策點(decision，一進、擇一出)： 

  - 合併點(merge，多進一出)： 
  - 作業內容： 

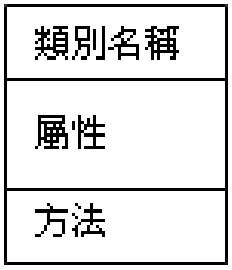
活動圖範例：提款


要求輸入密碼
確認密碼
列出服務選項
插入卡片
密碼輸入
相符
密碼不符
選擇提款
退卡片並列印錯誤訊息
超過
未超過
檢查輸入次數
檢查是否超過日領限額
未超過
檢查是否超過存款餘額
金額輸入
要求輸入金額
超過
超過
顯示尚可提領餘額
未超過
吐鈔付款
退卡片
詢問是否列印收據
是
否
列印收據
螢幕顯示餘額

類別圖

- 類別圖(class diagram)：此圖藉由描述系統內的各個類別以及類別之間的關係，以呈現系統結構。OO技術的核心是類別及其物件，故此圖是OOAD中最核心的圖形。
- OO內的每個類別包含名稱、屬性(即資料，包含變數及常數)及方法(即行為)，與傳統結構化系統將資料交由資料庫、行為交由應用程式處理的模式大不相同。
- 分類：
  - 概念類別圖：在系統分析(OOA)階段繪製的類別圖，不必考量特定技術內涵(e.g., Java or C++)，也不必考慮技術細節(可忽略屬性及方法)。此圖可用來塑模企業程序，形成概念模型(conceptual model)。
  - 設計類別圖：在系統設計(OOD)階段繪製的類別圖，必須將選定技術之細節包含在圖形內。

類別圖：符號 2-1

- 符號：
  - 類別： 

抽象類別： 

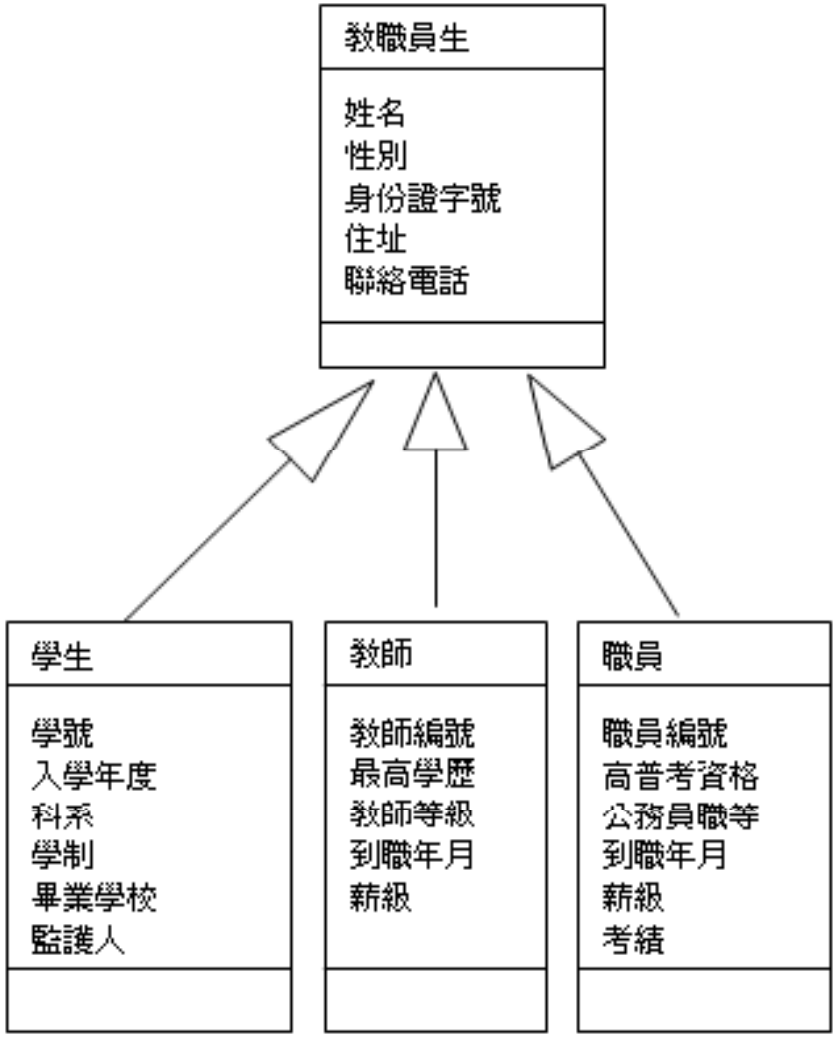
介面： 

  - 有時為了讓概念模型更簡潔，可將類別的屬性或方法省略。
  - 屬性及方法的透明度(visibility)：
    - 公開(public)：+
    - 保護(protected)：#
    - 私有(private)：-
    - 套裝(package)：~

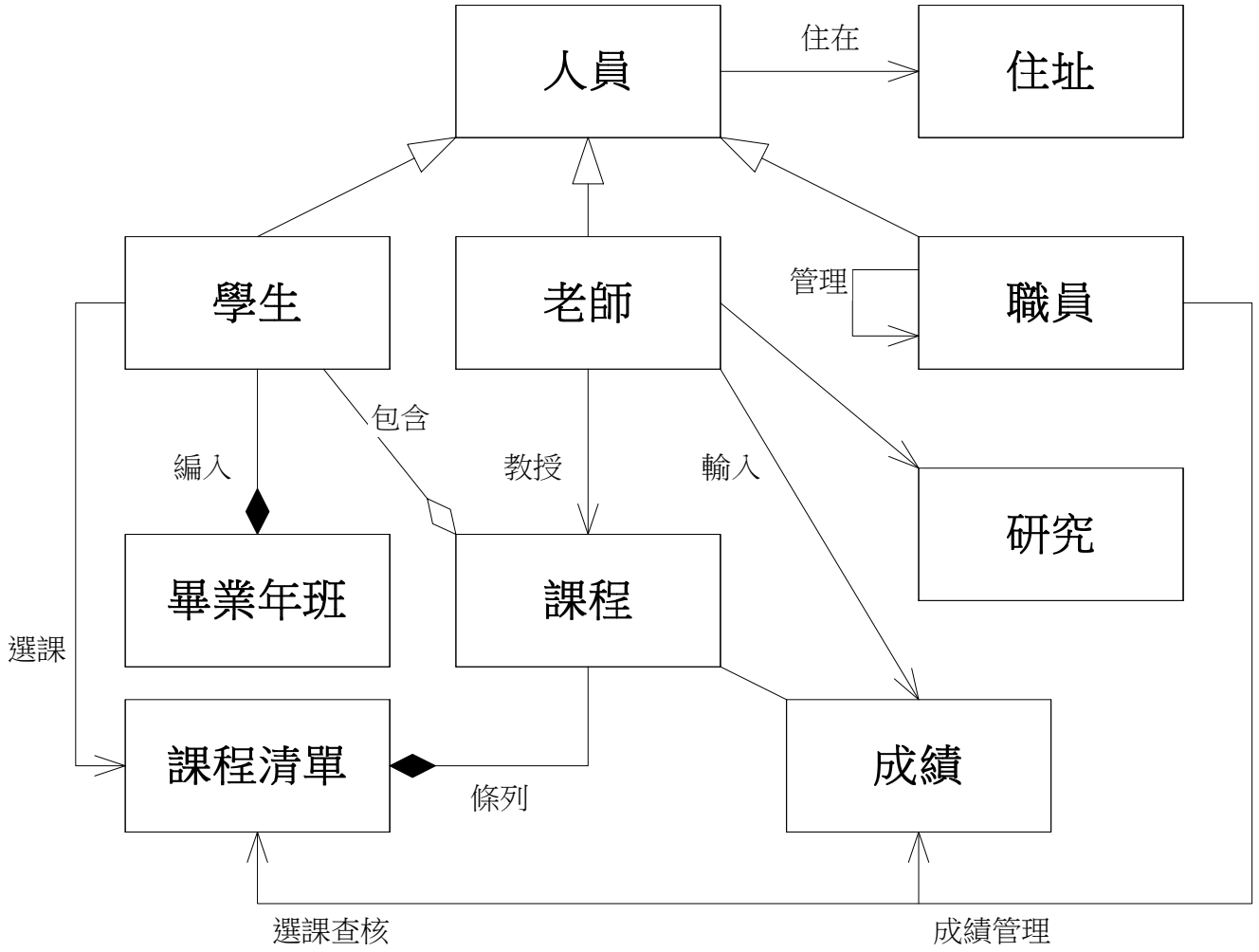
類別圖：符號 2-2

- 關係：
  - 聯合(association)：雙向 ——— ，單向 ⟶
  - 依賴(dependency)： ⇢
  - 聚合(aggregation)：
    - by ref：汽車◇——— 輪胎，表示後者是前者的一部份，並可獨立存在。
    - by value：公司◆——— 部門，表示後者是前者的一部份，但無法獨立存在。
  - 繼承(generalization)：繼承類別 ⟶▷ ，實作介面 ⇢▷
- 多重性(multiplicity)：*, 0, 1, 0..*, 1..*, etc.

類別圖範例：繼承


教職員生
姓名
性別
身份證字號
住址
聯絡電話
學生
學號
入學年度
科系
學制
畢業學校
監護人
教師
教師編號
最高學歷
教師等級
到職年月
薪級
職員
職員編號
高普考資格
公務員職等
到職年月
薪級
考績

類別圖範例：教學管理



問題討論

- 在上圖中，兩個類別間的多重性應該如何標示？
- 教師的分級(助教、講師、助理教授、副教授、教授)應該做為教師類別的屬性、還是應該獨立成為教師下的子類別？或是可以其他方法做進一步分類(教學型教師、研究型教師、行政型教師)？

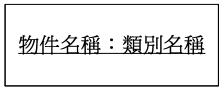
類別圖：典型

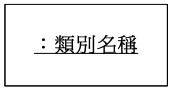
- 設計階段的三種典型類別(stereotype)
  - Boundary class：此類別做為系統與外部之間的橋樑，可再分為兩類：
    - 使用者介面：處理系統與使用者之間的互動。
    - 系統介面：處理系統與其他系統之間的互動。
  - Control class：此類別負責協調其他類別的工作，通常每個使用案例都會有一個control class。此類別接收由boundary class傳來的訊息後，再轉成一系列的訊息傳遞給entity classes。
  - Entity class：此類別封裝企業資料及企業邏輯，是類別圖的核心所在。
- 可在類別圖的類別名稱上冠上<<boundary>>、<<control>>、<<entity>>等符號。

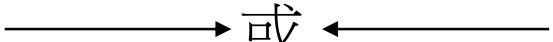
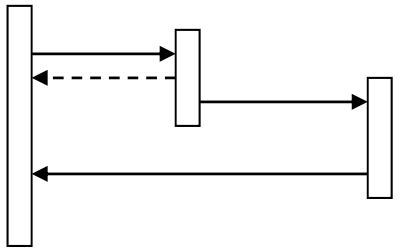
順序圖

- 順序圖(sequence diagram)：依時間順序描述系統內部各成員之間的互動，通常可分成兩大類：
  - 描述使用情境：主要用於系統分析階段，著重在角色(actor)與系統之間的互動，將系統當成一個黑盒子，通稱為系統順序圖。
  - 描述方法邏輯：主要用於系統設計階段，著重在物件之間的訊息傳遞。

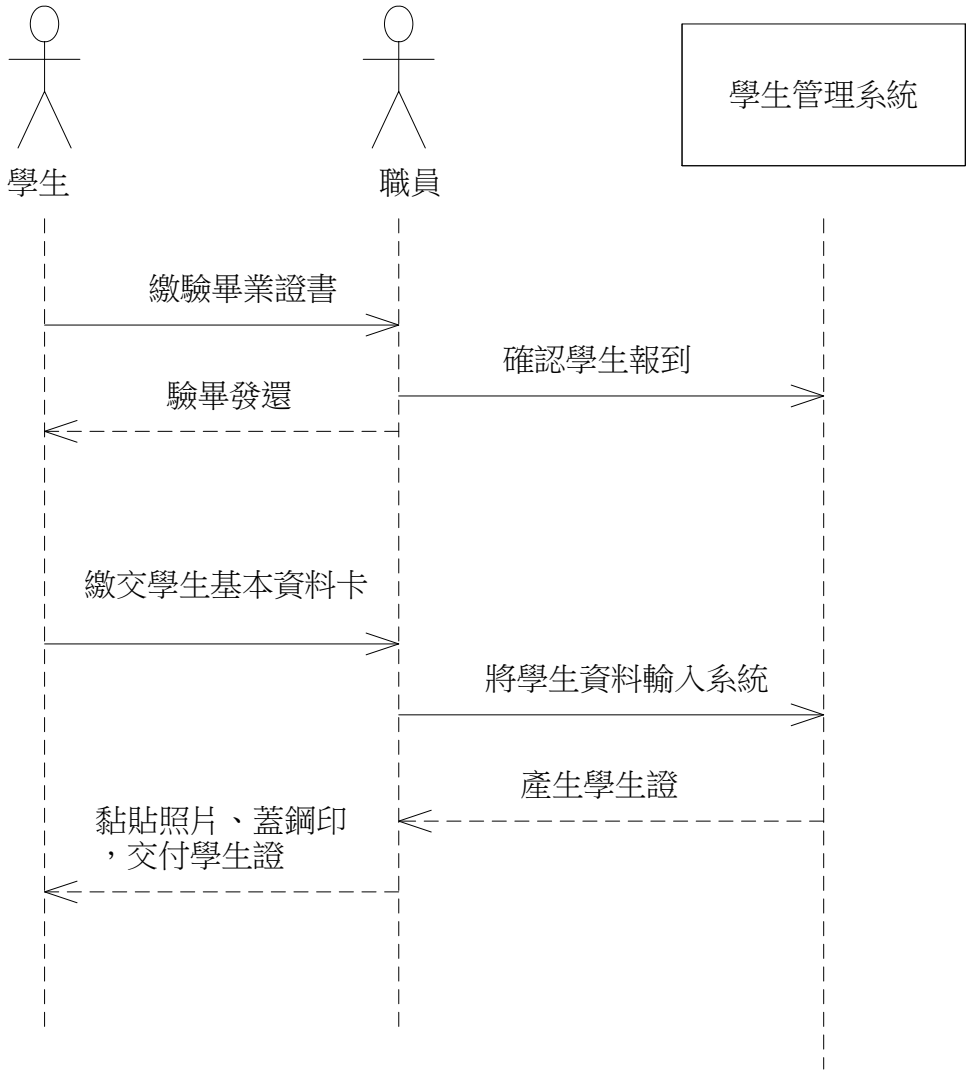
順序圖：符號

- 符號：
  - 物件： 

，或 

  - 生命線(lifeline)：由上到下的虛線，代表物件、角色或系統的存活時間。
  - 訊息線： 
或
  - 回應線：
或
  - 促動盒：位於物件生命線上的長條矩形，代表訊息已發出， 並由接收物件進行處理中。 

系統順序圖範例：新生報到



設計樣式、API、資料結構、演算法

- 設計樣式(design pattern)：樣式(pattern)是指被許多人一再重複使用的作業方式，通常代表解決類似問題的理想方法。電腦科學領域已發展出許多設計樣式，可供OOD階段參考採用，以提昇效率。
- OOAD結束後，系統發展即進入OOP階段：
  - API (application programming interface)：應用程式介面，指個別程式語言的技術內涵及寫作規範。每種程式語言的API都不相同，學習某種程式語言，大部分是在學習它的API內容。
  - 資料結構(data structure)：是探討如何把資料有效率地存放在電腦中的學問。基本的資料結構包括陣列(Array)、串列(List)、堆疊(Stack)、佇列(Queue)、樹狀(Tree)及圖形(Graph)等，不同程式語言的API規範雖然有異，但資料結構的內涵則大致相同。
  - 演算法(algorithm)：是指完成一件任務所需要的具體步驟和方法。程式設計可說是「資料結構 ＋ 演算法」的結合。演算法的優劣可從空間複雜度及時間複雜度來判斷；同樣硬體條件下，越省時、省空間通常就是越好的演算法。

UML在會計上的應用

- 實務上：新的會計資訊系統多半採用OOP做為開發工具。
- 教學上：AIS教科書雖然都有系統開發的章節，但大部分的內容都在介紹傳統的開發工具，鮮少提及UML及OOP的概念。
  - Jones & Rama (2006) 是目前唯一以UML貫穿全書內容的AIS教科書，但主要的描述工具僅限於活動圖。
  - 由於AIS課程的主要目的在幫助學生建立企業系統流程的概念，而非培養系統開發能力，因此描述工具的學習具有相當彈性。隨著UML及OO的普及，未來的AIS教科書可望增加對於UML及OO概念的介紹。
- 以UML設計會計系統之參考範例(連結至政大機構典藏)：http://nccur.lib.nccu.edu.tw/bitstream/140.119/35187/8/35601908.pdf