

<<會計資訊系統課程講義>>

系統描述工具(完整參考版)：

**DFD(資料流程圖)、SF(系統流程圖)、
PM(程序圖)、UML(統一塑模語言)**

周國華

國立屏東大學會計學系

初版：2007/09/23

本次修訂：2024/02/17

目錄

主題	頁次
智慧財產權聲明	3
第一部份：DFD (資料流程圖)	4~22
第二部分：System Flowchart (系統流程圖)	23~33
第三部分：Process Map (程序圖)	34~40
第四部份：UML (統一塑模語言)	41~69
系統描述工具在會計上的應用	70~71

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。

第一部份

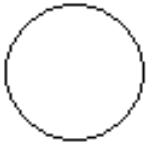
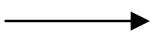
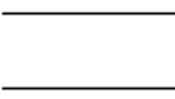
資料流程圖(DFD)

DFD：功能

- 資料流程圖(Data Flow Diagram, DFD)描述資料在系統內的子系統之間、系統與外部之間、組織內各部門之間、或組織與外部之間的流動情形，以及資料來源(source)、去向(destination)及儲存處(data store)。
- DFD是結構化系統分析及設計(SSAD)所使用的標準描述工具之一。在系統改用物件導向(OO)技術發展後，DFD仍是了解使用者需求的重要工具。
- DFD的F是 flow，通常翻譯成資料流程圖，或者資料流向圖，或更簡潔的資料流圖。

DFD：符號

- DFD用以下四種符號描述資料的流動：

	通稱為 bubble ，代表一個個體或程序。流入資料經此個體或程序處理後，轉換成流出資料。此圖形在 DFD 中代表正在描述之系統的全部或其中一部份。
	表示資料的流通路徑(資料流)，資料名稱會標示在邊上。
	表示資料的來源或去向，在 DFD 中代表正在描述之系統以外的其他系統或外部個體。
	代表儲存資料的地方(檔案或資料庫)。

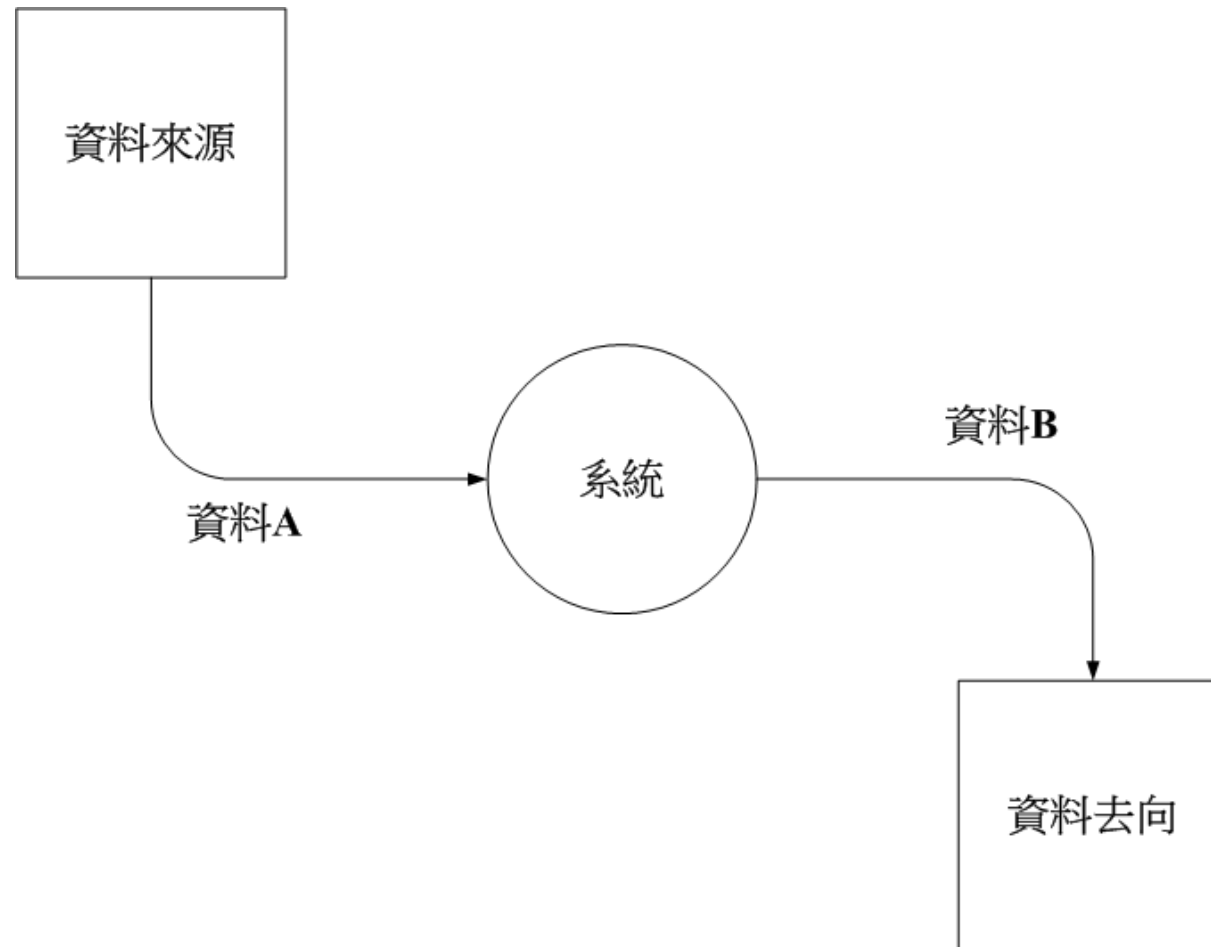
- 以上四種符號，在不同教科書或應用領域中常有不同的變異(例如: **bubble**用方形、矩形、六邊形或八邊形)，但基本概念則相同。

DFD：層次

- 按照描述的繁簡程度，**DFD**可分為以下幾個層級：
 - 背景圖(context diagram)：是**DFD**中最簡單、最上層、最抽象的圖，通常用一個**bubble**代表所描述的系統，再加上兩個方形符號表示系統之外的資料來源及去向。
 - 第0階(level 0)**DFD**：將上述單一**bubble**分解成1.0、2.0、3.0 ...等數個子系統。
 - 再細分：將上述子系統進一步分解，例如系統1.0再細分為1.1、1.2、1.3...等子系統。

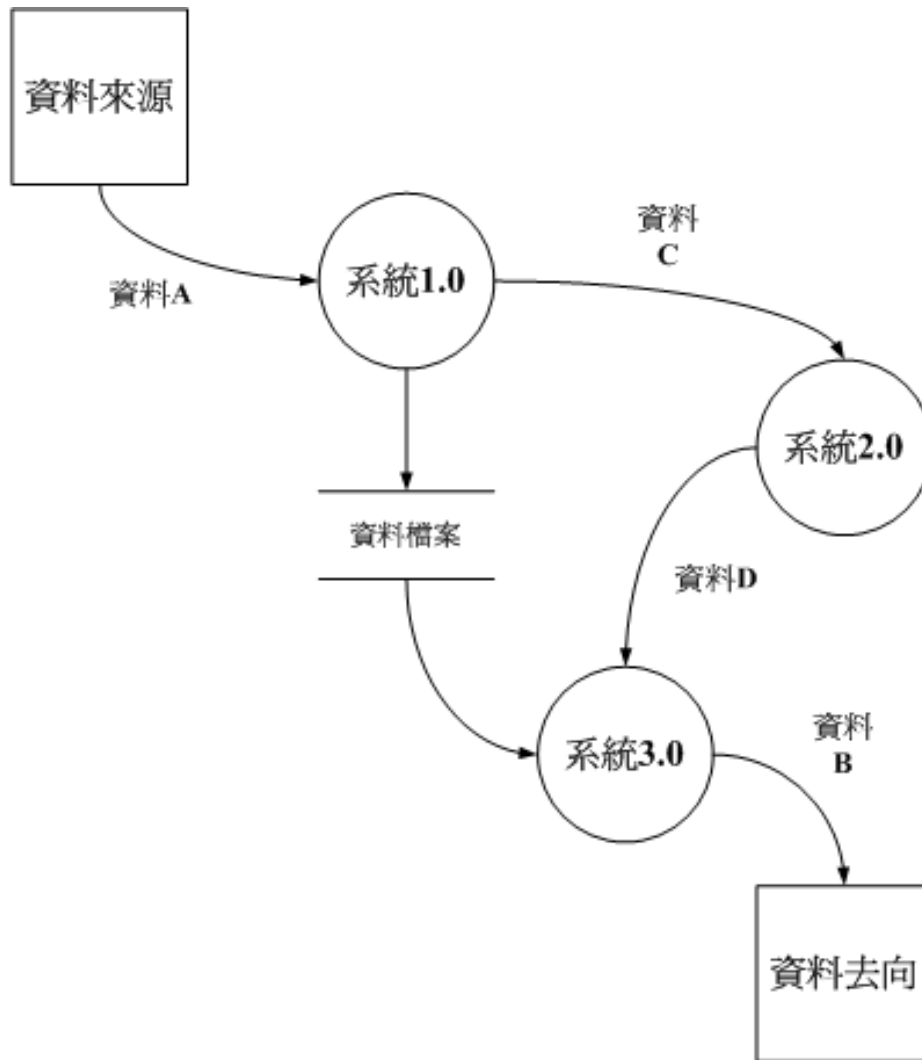
背景圖(Context Diagram)

- 背景圖的通用樣式如下：



第0階DFD圖

- 第0階DFD圖的通用樣式如下：



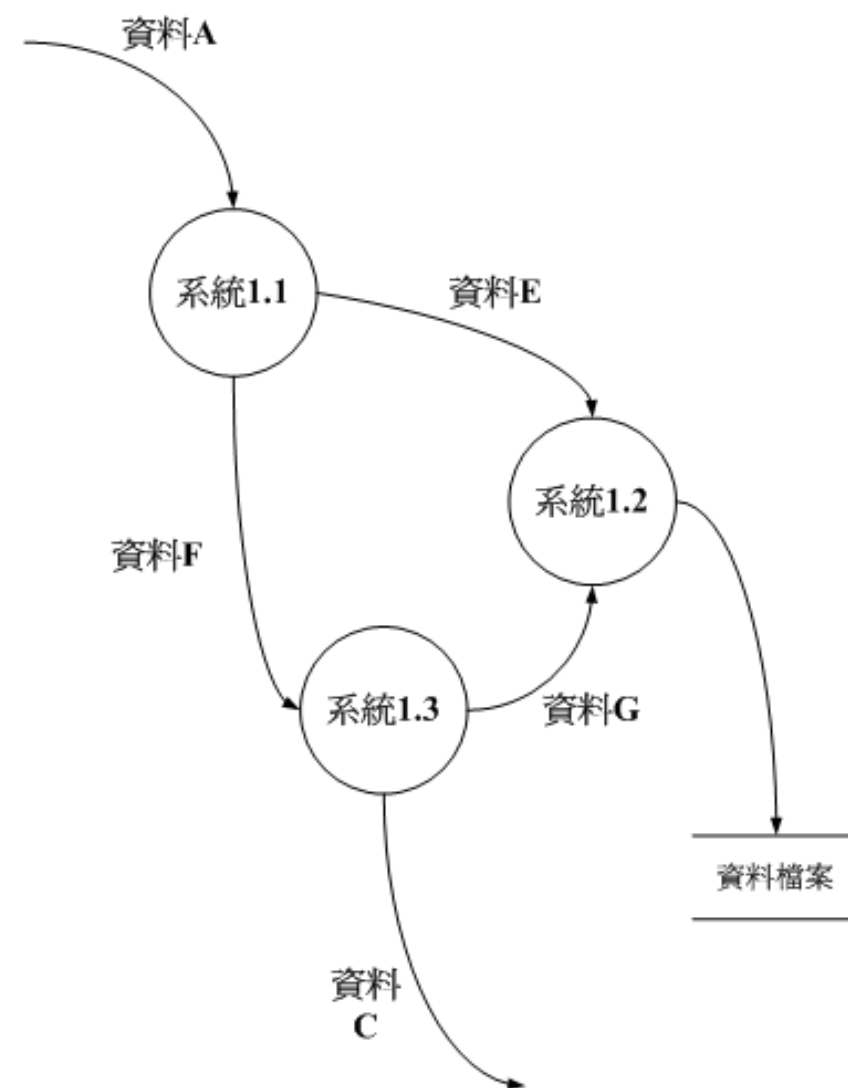
請注意：左圖的內容僅為例示，實際上的系統內容可能千變萬化。

再細分準則：流入流出一致

- 無論是將背景圖分解成第0階圖、或是將第0階圖做進一步細分，必須遵守「流入與流出上下層一致」原則。英文稱之為**a set of balanced DFDs**。
- 以前述二圖為例，背景圖所描述的系統有資料**A**流入、資料**B**流出；第0階圖也必須遵守資料**A**流入（圖例流入系統1.0）、資料**B**流出（圖例從系統3.0流出）。
- 若要進一步細分，則（依照圖例）系統**1.0**的子系統必須「一進二出」、系統**2.0**的子系統必須「一進一出」、系統**3.0**的子系統必須「二進一出」，且流入流出的資料名稱必須與上一層相同。

再細分範例

- 右圖為前述系統1.0再細分後之樣式：



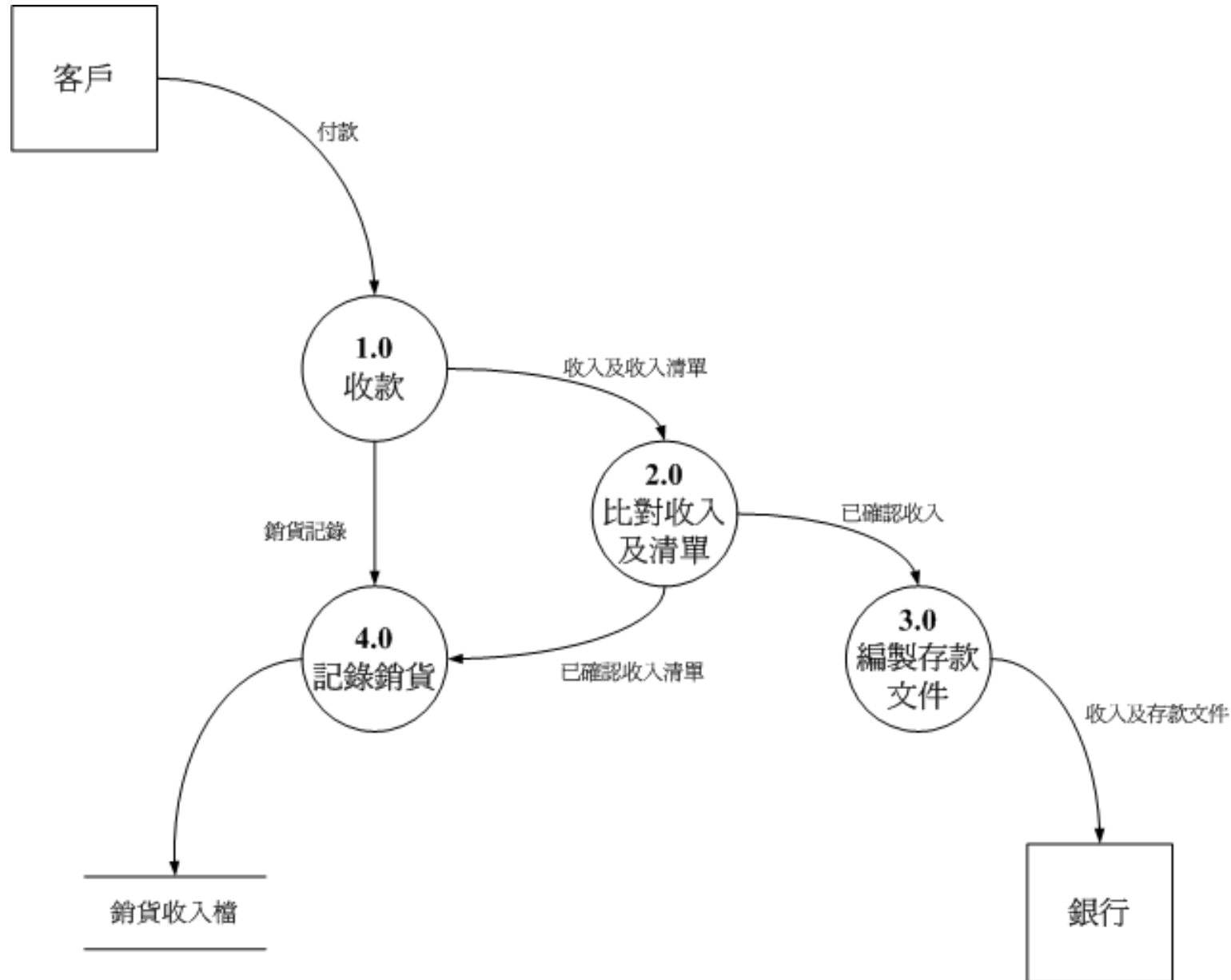
DFD：類型

- DFD可按資料及程序的描述方式分成兩種類型：
 - 實體資料流程圖(physical DFD)：此圖中，資料有具體的名稱；**bubble**是處理資料的人、地、物等個體(entity)，以名詞表示。
 - 實體DFD描述系統的基礎架構，可回答如何做(how, 物)、在哪做(when, 地)、誰來做(by whom, 人)等問題。
 - 邏輯資料流程圖(logical DFD)：此圖中，資料是泛稱；**bubble**代表處理資料的程序(process)，以動詞表示。
 - 邏輯DFD描述系統的各項作業，可回答做什麼(what)這項問題。

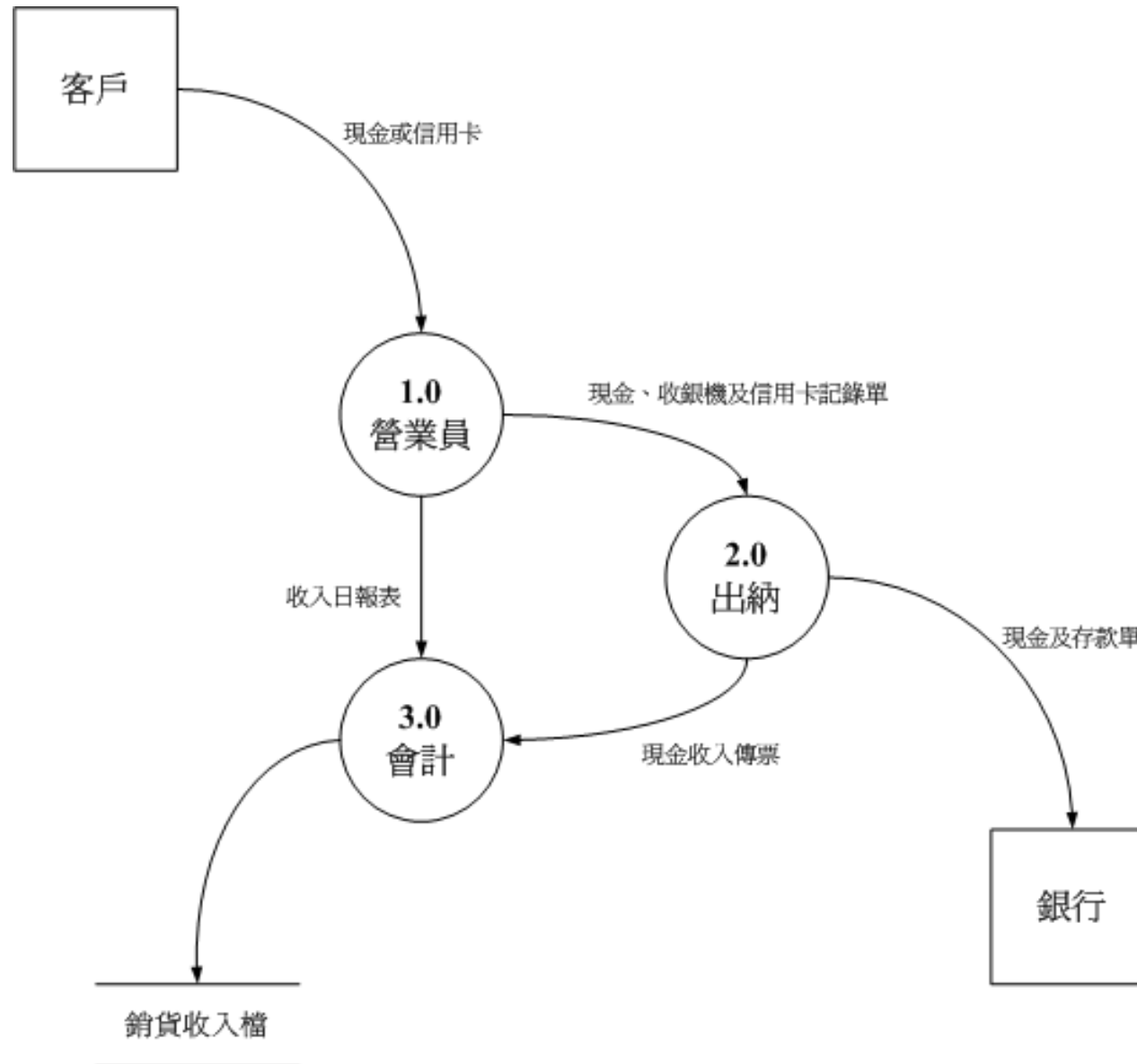
DFD : Logical vs. physical

- 長期而言，系統做什麼(**what**)的答案比較穩定，但系統如何做(**how**)、在哪做(**where**)、誰來做(**by whom**)的答案則會隨著時間及技術而改變。
- 在建置新系統時，通常會先繪製現有系統及新系統的**logical DFD**，以提供使用者新舊系統的比較資訊；然後再根據新系統的**logical DFD**，繪製**physical DFD**。

Logical DFD 範例：現銷系統



Physical DFD 範例：現銷系統



編製DFD的前置作業

- **DFD**可提供開發者及使用者瞭解特定系統的資料流程，但**DFD**的繪製者必須先瞭解特定系統的現行或修正後流程，才能繪製出正確的**DFD**。
- 欲瞭解特定系統的現行及修正後流程，通常必須由系統開發小組與使用單位人員進行**密集訪談**，取得使用單位對現行系統之流程及擬修正之流程的完整系統敘述(system narrative，詳細的文字說明)。
- 從系統敘述中找出個體(entity，包含人及機器)，以及每個個體負責的作業(activities，通常包含動作敘述)，並在表格中以個體vs.作業項目的對照方式呈現。

系統敘述範例

屏大食品公司會計總帳流程的系統敘述(部分內容)：

段	行	敘述
一	1	屏大食品公司會計部門人員依據證期局公布之「一般行業 IFRSs 會計項目及代碼」，並參考該公司會計主管所延伸設 定的其它會計項目，透過系統編製會計項目表；在營業期間 如有需要，仍可經由授權的程序進行會計項目表的增刪修。 針對營業活動中的經常性交易，可透過系統的自動拋轉機制 於事件處理的流程中，設定自動拋出那些包含適當借貸方會 計項目與金額之日記帳分錄。該公司主要針對銷貨收款、進 貨付款、生產與存貨、薪工等交易循環設定經常性交易的日 記帳拋轉功能，其它非經常營業活動、投融資活動等交易事 件，則以人工方式輸入日記帳。以上設定均需由會計部門主 管進行複核。
	2	
	3	
	4	
	5	
	6	
	7	
	8	
	9	
	10	
	11	

個體作業表範例

屏大食品公司會計總帳流程個體作業表(部分內容)：

個體	段落	作業編號	作業項目
會計人員	一	1	取得「一般行業IFRSs會計項目及代碼」與會計主管延伸設定的其它會計項目
		2	透過系統編製會計項目表（輸入）
系統		3	儲存會計項目表
會計人員		4	設定經常性交易的日記帳拋轉功能
系統		5	讀取會計項目表與交易基本資料表
		6	儲存經常性日記帳拋轉資料表
會計人員		7	設定連續性定期調整項目的日記帳拋轉功能
系統		8	讀取會計項目表
		9	儲存定期調整日記帳拋轉資料表
會計主管		10	透過系統複核會計人員的設定結果

DFD的編製原則 ³⁻¹

- Dull, Gelinas & Wheeler (AIS, 9th ed.)列舉13項編製DFD的原則：
 1. 每一個有負責到資訊處理作業的個體(entity)都應成為DFD的一個bubble或包含在其內。
 - 資訊處理作業包含擷取、轉換及儲存資料，包含人工及電腦作業，但不包含資料在個體之間的移轉。
 - 資料在個體之間的移轉，並非資訊處理作業，而是DFD內的資料流。
 - 有些作業是營運程序作業，也不屬於資訊處理作業。
 - 沒有負責任何資訊處理作業的個體則是外部個體。
 2. 一開始繪製DFD時，只要放進正常程序即可，不必包含例外或錯誤處理程序。

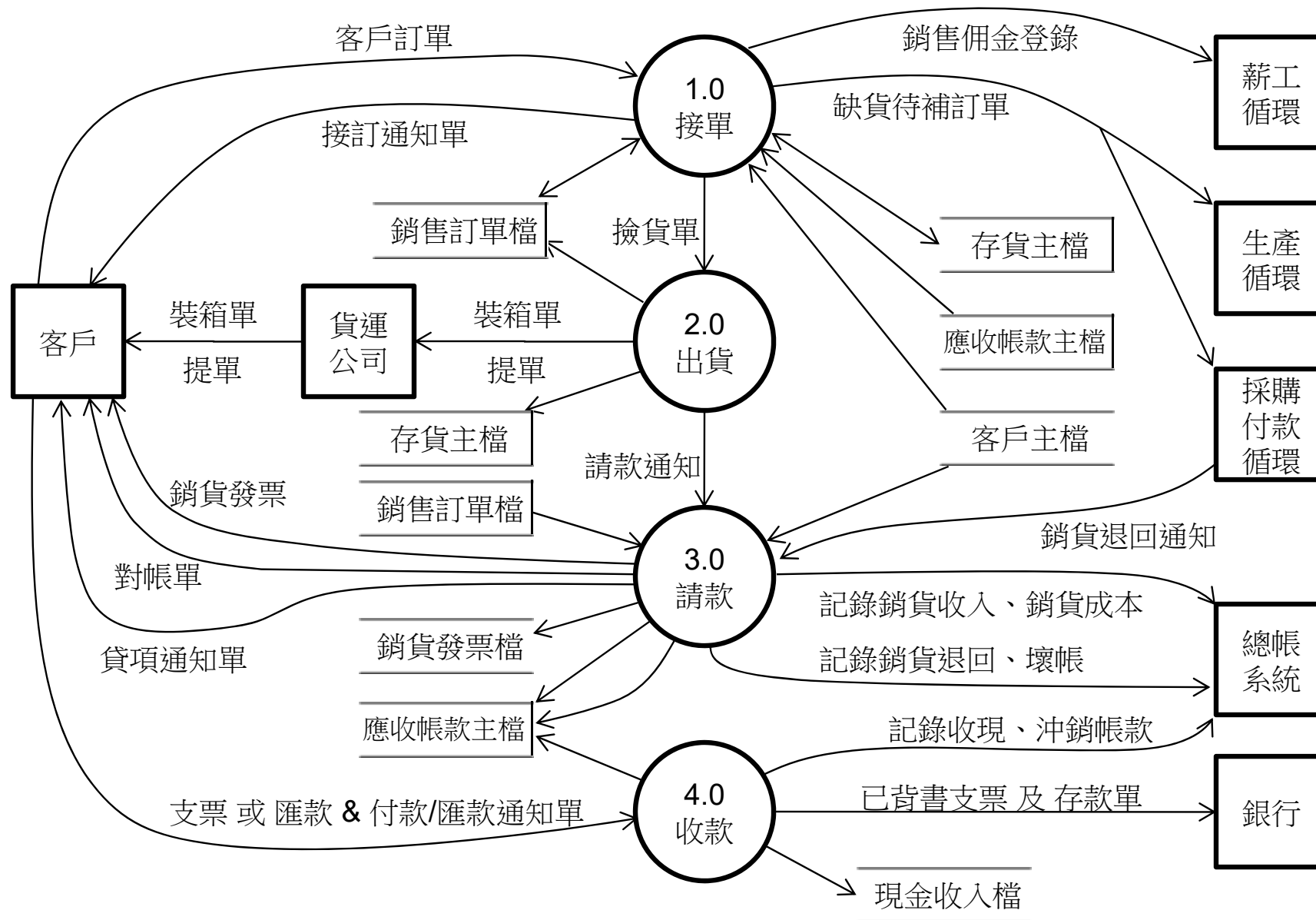
DFD的編製原則 ³⁻²

3. 只處理系統敘述中所描述的內容，不多、不少。
4. 若多個個體負責同樣的作業，只描述一個即可。
5. 每一次進、出任何一個資料儲存處，都應該畫一個流入或流出的箭頭。
6. 如果資料儲存處在邏輯上是必要的(例如：兩個處理程序之間有時間差)，即使系統敘述中未提及，也應該包含在圖中。
7. 把在同一時間、同一地點處理的多項作業包含在第0階的同一個**bubble**中。
8. 把在同一時間、不同地點處理的多項作業包含在第0階的同一個**bubble**中。

DFD的編製原則 ³⁻³

9. 把在邏輯上相互關連的多項作業包含在第0階的同一個**bubble**中。
10. 為了有較佳的可讀性，每一份**DFD**圖最好只包含**5**至**7**個**bubble**。
11. 在資料流向一個營運程序個體時，若該個體只有營運程序的功能，應該以正方形表達；若該個體要執行資訊處理作業，就應該以**bubble**表達。
12. 在實體**DFD**內，讀寫電腦內的資料，應該通過一個代表電腦的**bubble**來進行。
13. 在邏輯**DFD**內，資料流不能從編號較後的**bubble**流入編號較前的**bubble**。

DFD範例：銷售及收款循環(第0階邏輯DFD)



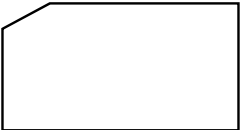
第二部分

系統流程圖

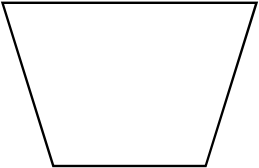
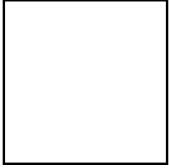
Flowchart：三種類型

- 系統流程圖(system flowchart)：將特定企業程序(business process)的流程作完整表達，包含該程序的資訊程序(information process)，即 資料輸入、處理、儲存及輸出)及相關的營運程序(operations process)，即 人員、設備、組織及活動)。
- 程式流程圖(program flowchart)：表達資訊程序中的電腦程式邏輯。
- 文件流程圖(document flowchart)：將文件在人工化系統(manual system)內的流轉程序作完整表達。

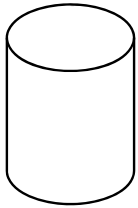
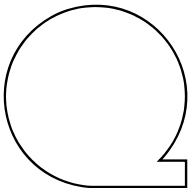
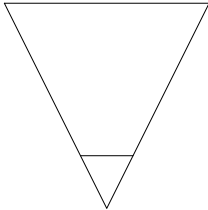
Flowchart 符號：輸入輸出

	表示輸入或輸出的文件或報告。
	人工輸入。
	打孔卡片，由讀卡機讀取。考試用答案卡是常見實例。
	通用輸入輸出符號，亦可代表會計帳簿。
	電腦螢幕，通常做為輸出符號；但具有觸控式功能的螢幕亦做為輸入符號。

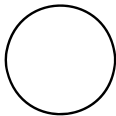
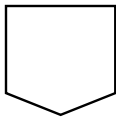
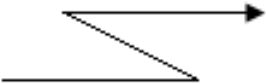
Flowchart 符號：處理

	代表電腦處理，包含資料查詢及檔案更新。
	代表人工處理。例如 編製文件、在文件上簽章等。
	透過電腦周邊設備所做的附帶性處理，例如 文件掃描、條碼及 RFID 讀取等。

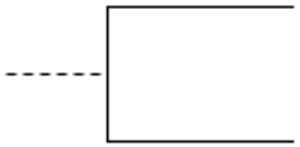
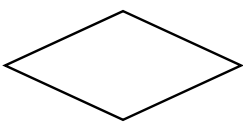
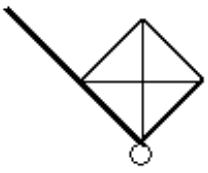
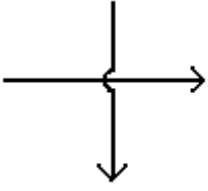
Flowchart 符號：儲存

	表示資料儲存在可直接存取資料(direct access)的媒體上，包含磁碟、光碟等。
	表示資料儲存在須循序(sequential)存取資料的媒體上，例如 磁帶。
	書面文件歸檔符號，其內上方註明文件或檔案名稱，下方三角形可書寫不同字母： N 表示按編號歸檔， A 表示按字母順序歸檔， C 或 D 表示按日期歸檔。

Flowchart 符號：連結

	代表流程的起點或終點，亦可代表外部個體。
	同頁連結，其內可書寫A、B、C...等字母。
	跨頁連結，其內可書寫頁碼及A、B、C...等字母。
	代表文件、實物或處理程序的流向。通常流向為向右、向下。
	通訊連結，代表以電話或電腦網路進行資料或訊息傳輸。

Flowchart 符號：其他

	文字註解，以虛線連結。
	批次總數、控制總數。
	決策點。
	貨物。
	流程穿越。

系統流程圖編製原則 ³⁻¹

- **Dull, Gelinas & Wheeler (AIS, 9th ed.)**列舉13項編製系統流程圖的原則：
 1. 將流程圖劃分成多個欄位，每個欄位代表一個內部個體；如有必要，亦可以特定欄位代表外部個體。欄位間可用虛線或實線間隔。
 2. 各欄位所代表的個體，應經過審慎排序，以便讓流程盡可能由左至右進行。
 3. 流程邏輯應該遵循由上往下、由左至右的原則，並以箭頭標示流向。
 4. 盡可能將單一系統的流程限縮在單一頁面內。如超過單一頁面，應使用跨頁連結符號作連接。

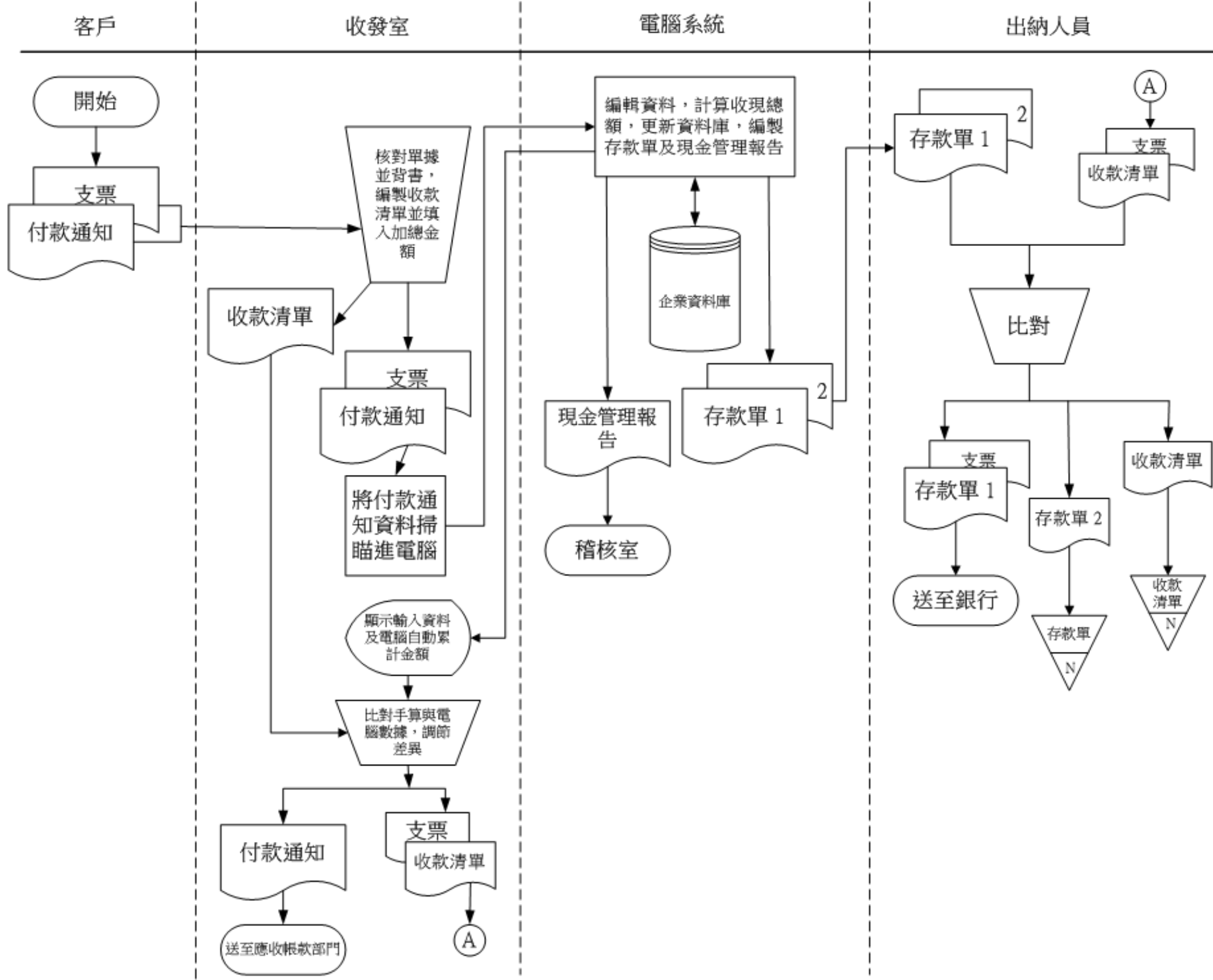
系統流程圖編製原則³⁻²

5. 每個欄位內，在兩份文件之間，至少必須有人工處理、人工輸入或儲存符號之一居間。
6. 當流程跨欄位時，流程線兩端應各連結一份文件。若連結極短且意圖明確時，亦可權宜處理。
7. 由特定電腦設備列印出的文件或報告，應先出現在該設備所在欄位內，再流轉至其他欄位。
8. 文件或報告如係由集中式系統或遠端設備列印，就不應出現在下達列印指令的電腦設備所在欄位內。
9. 資料處理如係在特定部門的電腦設備上進行，此程序應顯示在該部門所在的欄位內，或以接臨的獨立欄位顯示此程序，而不應表達在集中式系統所在的欄位內。

系統流程圖編製原則 3-3

- 10. 數個無間斷的循序處理步驟，可用單一程序或數項程序表達。
- 11. 要從電腦儲存媒體存取資料，必須經由電腦處理；故磁碟或磁帶符號必須與電腦處理符號連結。(人工輸入符號不能直接與電腦儲存媒體連結)。
- 12. 不必用人工處理符號來表達書面文件的傳遞。
- 13. 不必用人工處理符號來表達書面文件的歸檔。

系統流程圖範例：收款程序



第三部分

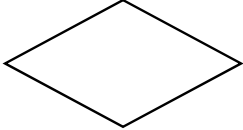
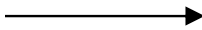
程序圖

程序圖：功能

- 程序圖(**process map**)能以相對簡單的符號完整表達企業程序(**business process**)的內涵。
- 在企業程序改變前後，最適合用程序圖以對照方式表達企業程序的現狀(**as is**)及改變後(**could be**)的新狀態。
- 常見應用領域：
 - 企業程序設計與分析(e.g., 會計師分析企業的內控程序)。
 - **ERP**系統導入前的企業流程再造(**BPR**)。
 - 六標準差(**six sigma**)管理模式的施行。

程序圖：符號

- 標準的程序圖，僅使用以下三種符號：

	代表特定程序。
	代表決策點。
	代表流程方向。

- 除了上述三種符號，許多使用單位會另外增添幾項常用符號，因此程序圖在實務上有相當大的變異性。

程序圖繪製原則 ²⁻¹

- **Damelio (1996)**列舉7項繪製程序圖的原則：
 1. 程序圖以水平方式繪製，程序所流經的各個功能領域(**functional area**)列於左邊。
 2. 各功能領域之間以實線分隔。
 3. 若功能領域內包含有子領域，各子領域之間以虛線分隔。
 4. 程序圖以矩型符號代表程序、菱形符號代表決策。
 5. 帶有箭頭的流向線(——→)，應在其旁邊標示文件名稱，代表將流入矩型程序、流入菱形決策點或流出矩型的特定文件。文件可為紙本文件或電腦檔案。

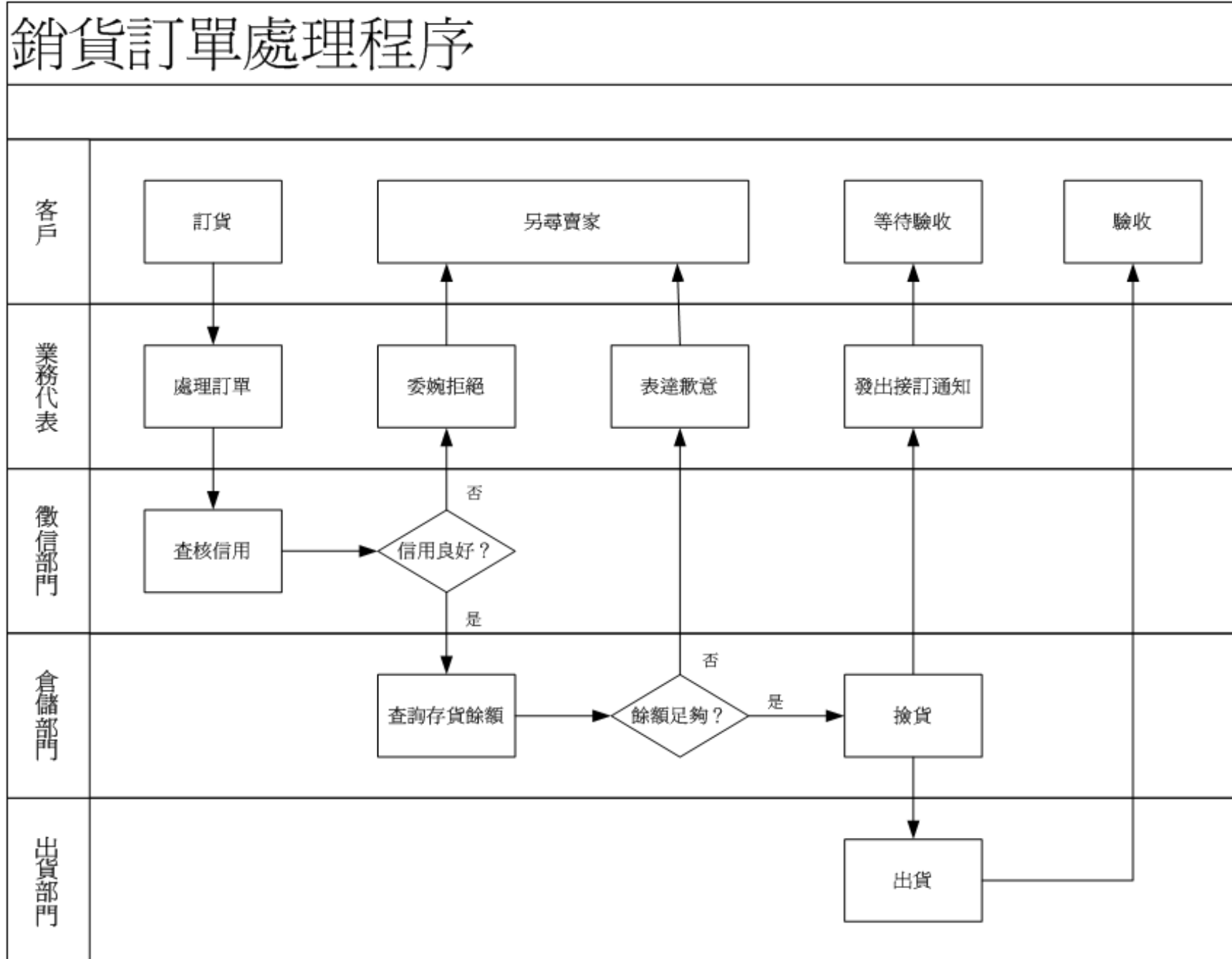
程序圖繪製原則 ²⁻²

- 6. 矩型符號只能代表程序，不能代表文件。
- 7. 文件流向應遵循由左至右、由上往下的原則。
- 以上是繪製程序圖的通用標準原則。實務上存在許多變異，例如：
 - **PWC**的程序圖有開始及結束符號，且不在流向線旁加註文件名稱，而是以一個六邊型符號代表流入或流出的物件(可能是文件或其他實物)。
 - 為了讓畫面更簡潔，許多機構所繪製的程序圖只有程序、決策符號及流向線，完全不顯示流入或流出的物件。此外，有些程序圖也不繪製分隔線。

從全局至細節

- 程序圖也可以和其他系統描述工具一樣，先繪製涵蓋面較廣、但不含細節的系統鳥瞰圖，再逐步往下延伸至特定的系統細節。
 - 如果系統內涵不複雜，各個子系統的程序可在同一頁面上描述，再以虛線隔開即可。
 - 較為複雜的系統，可在上層程序圖內標示各個程序的編號，再以其他頁面分別描述特定程序的各個子系統程序。

程序圖範例：接訂程序



第四部分

UML (統一塑模語言)

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：Java，Python，C++，C#，VB.NET，Ruby
 - iOS及Mac OS平台使用的OO語言：Swift，Objective-C
 - 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

- OO系統開發：資訊與行為並重，和傳統(結構化)系統開發相比，OO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。
 - 例如：傳統結構化系統遇上千禧蟲(Y2K)，得耗費鉅資做調整。在OO下僅需調整處理時間的類別(class)即可。
- 描述資訊與行為，不同語言有不同構念名稱：
 - Python：variable(變數)、function(函數)及method(方法)
 - Java：variable及method
 - C++：variable及function
 - VB.VET：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(**object**)就是抽象化的表徵。
 - ※ 在編程時，先定義類別(**class**)，再實作特定類別的物件。
- 例如，在學校管理系統中，有「學生」這個類別(**class**)：
 - 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

- 在OO中，把提供特定功能的變數及方法放在一個類別及其物件內，稱為封裝(encapsulation)。
 - 例如：銀行系統的「帳戶」類別及其物件
 - 變數：編號，客戶名稱，餘額，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.....
- 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類別的物件溝通，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！
 - ✓ 例如，A類別繼承B類別，一個A類別物件也是一個B類別物件；但一個B類別物件不一定是一個A類別物件。
 - 在C++及Python中，一個子類別可以繼承多個父類別。在Java及Swift中，一個子類別只能繼承一個父類別，但Java的類別可透過實作interface(介面)、而Swift的類別可透過實作protocol(協定)的方式，實質上取得多重繼承的優點。(介面或協定內的方法均為抽象方法)
 - ✓ 以Java為例，A類別實作C介面，一個A類別物件也是一個C介面。
 - ✓ Swift另外提供Protocol Extension的功能，讓類別在實做protocol時不必撰寫抽象方法的程式碼，而可直接引用Extension內的現成內容。

OO概念：多型

- 在OO中，多個方法名稱相同，卻容許有不同的運作內涵，稱為多型(**polymorphism**)，型態如下：
 - 子類別繼承自父類別的方法並加以覆寫。
 - 一個類別內有多個方法名稱相同，但各方法的參數不同(數量不同或型態不同)。此情況通稱為**overloading**(方法超載)。
 - 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或協定，並各自定義介面或協定內的抽象方法之實質內涵。

多型範例

- 例：在JAVA SE8的API中，JTextField這個類別有以下五種JTextField()建構子方法(constructor)：

Constructor Summary

Constructors

Constructor and Description

JTextField()

Constructs a new TextField.

JTextField(Document doc, String text, int columns)

Constructs a new JTextField that uses the given text storage model and the given number of columns.

JTextField(int columns)

Constructs a new empty TextField with the specified number of columns.

JTextField(String text)

Constructs a new TextField initialized with the specified text.

JTextField(String text, int columns)

Constructs a new TextField initialized with the specified text and columns.

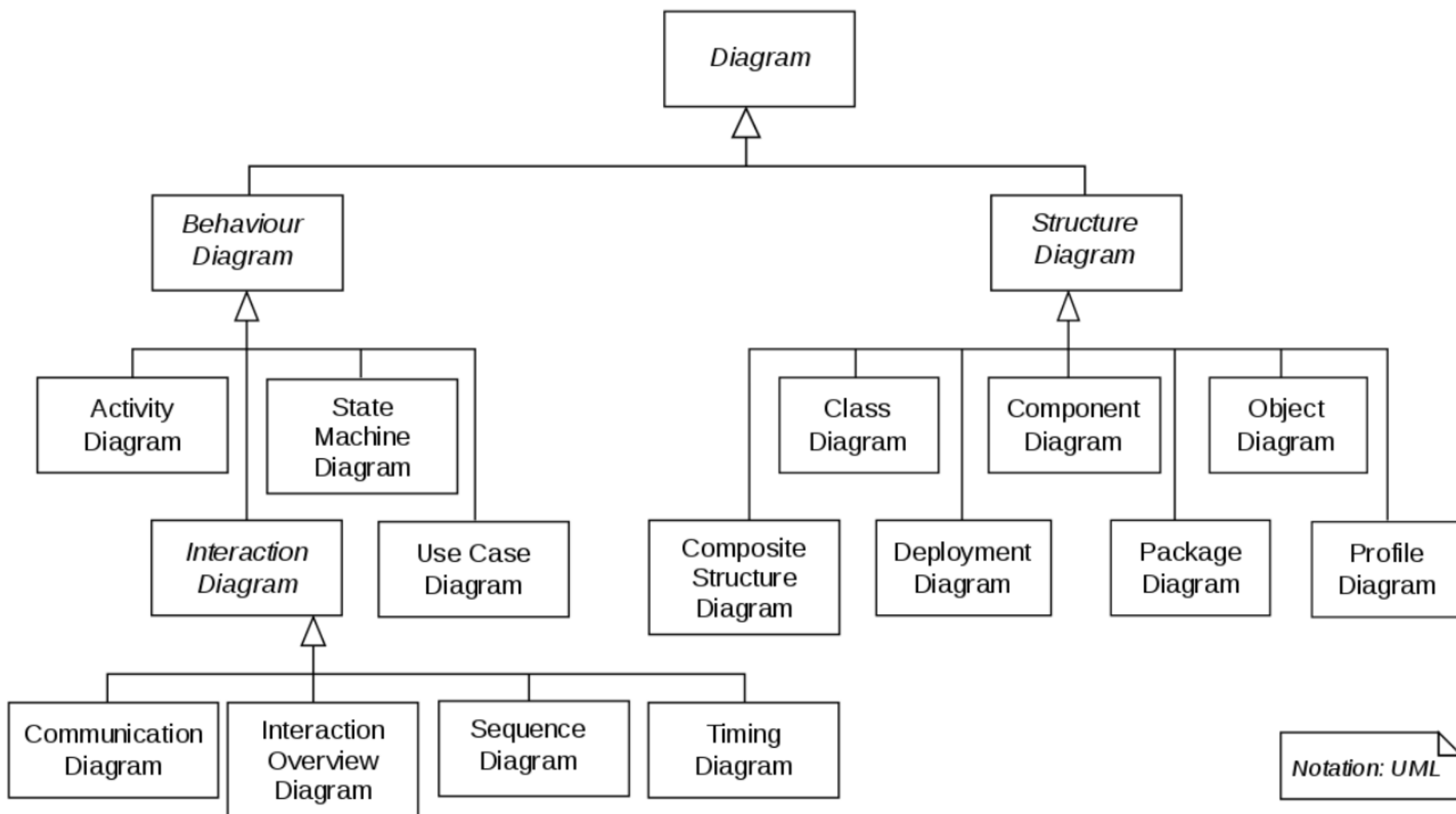
UML與OO

- **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，或稱建模)**：意指開發資訊系統時，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化模型(visual model)，以便有效傳達給程式設計師。

UML 2.0的圖形名稱 ²⁻¹

- UML 2.0將圖形分成三大類，共**14**種圖：
 - 結構圖形：**類別圖**，物件圖，元件圖，剖析圖，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的圖形名稱 ²⁻²

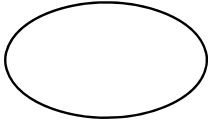
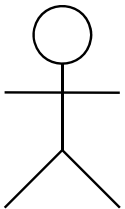


Notation: UML

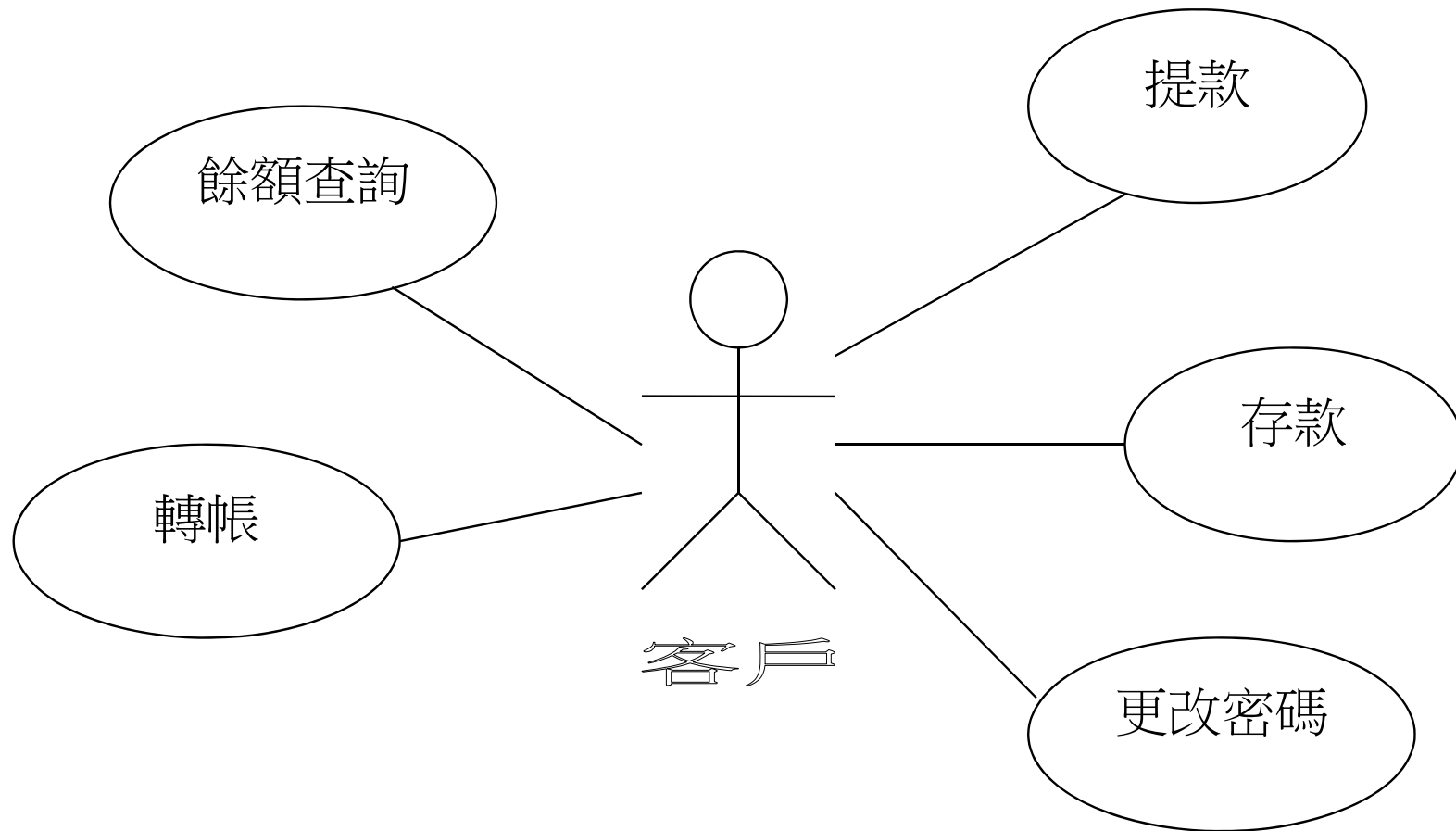
系統開發程序

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(OOA)：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(OOD)
 - 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(OOP)

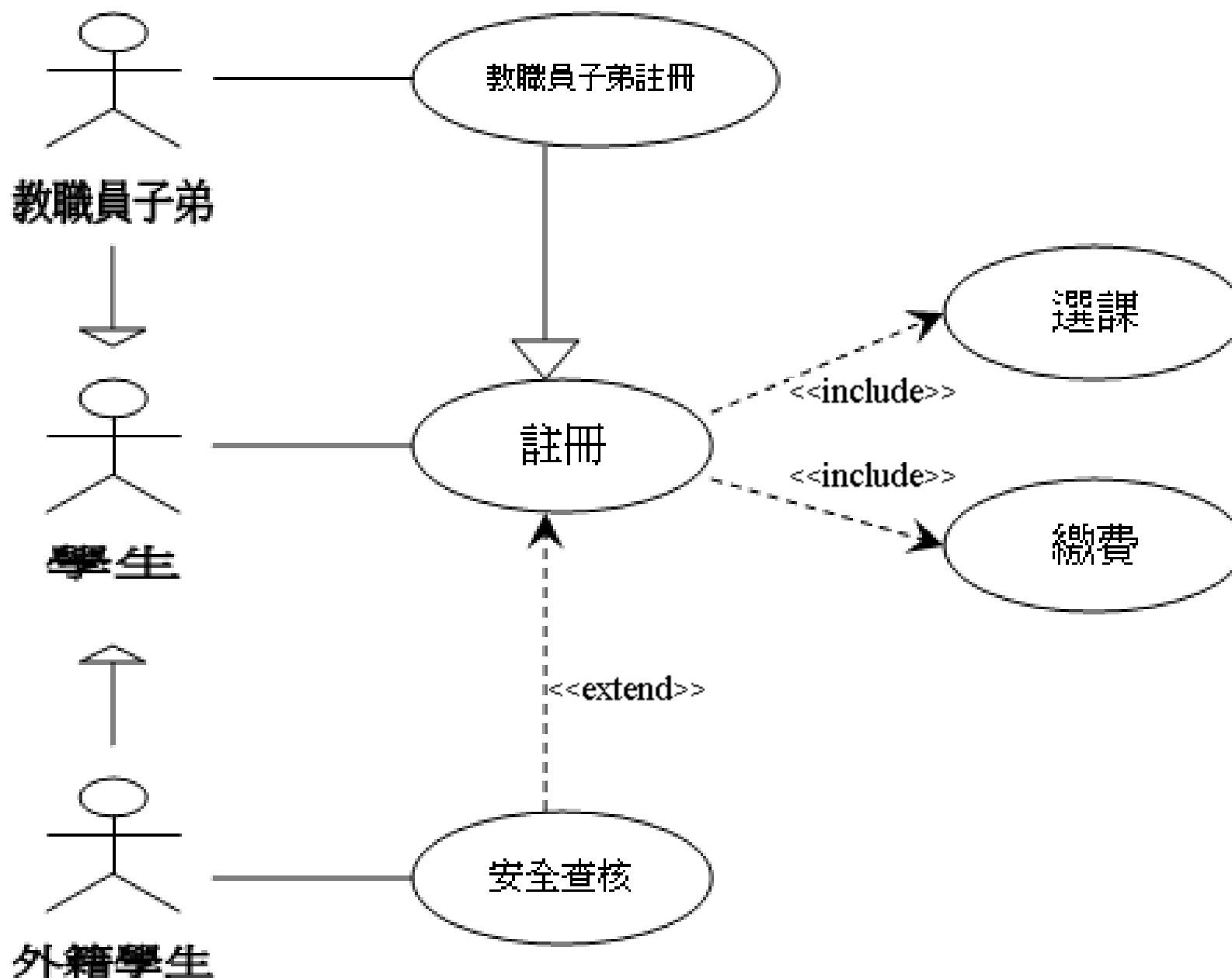
使用案例圖

- 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



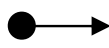
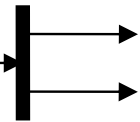
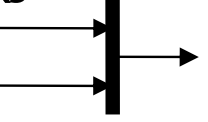
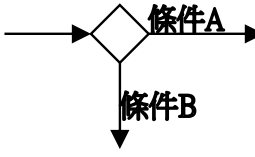
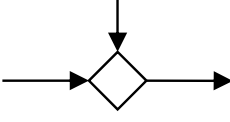
使用案例圖範例：註冊



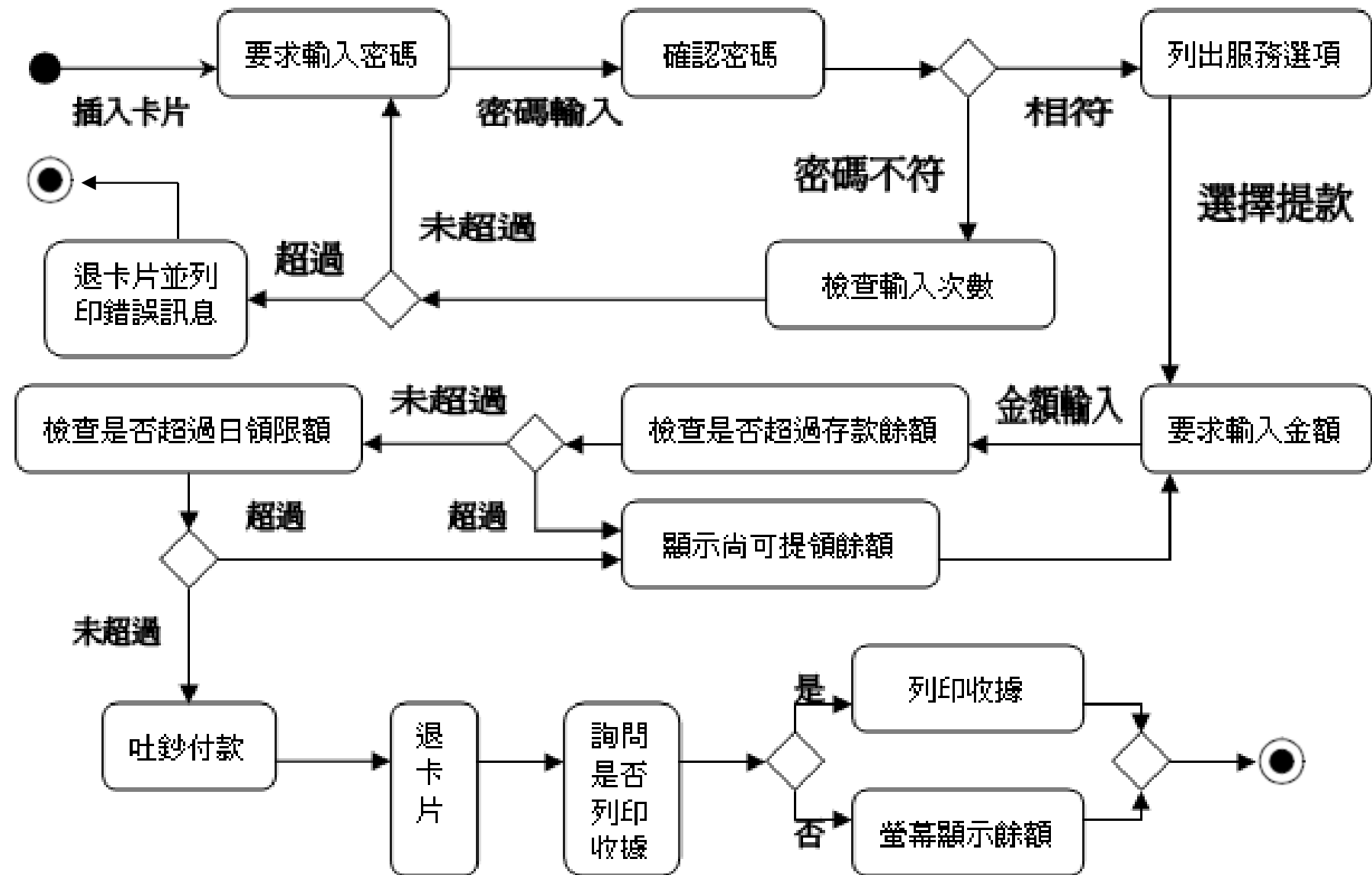
活動圖

- 活動圖(**activity diagram**，或稱作業圖)：此圖可用來描述
 - 一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



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

- 類別型態及符號：

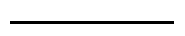
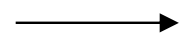
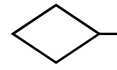
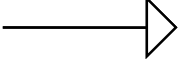
一 類別：	類別名稱	抽象類別：	類別名稱	介面：	<<Interface>> 介面名稱
	屬性		屬性		屬性
	方法		方法		方法

➤ 抽象類別意指其多個方法中至少有一個是抽象方法，介面的方法則都是抽象方法。(抽象方法：只有方法名稱而無實作內涵。)

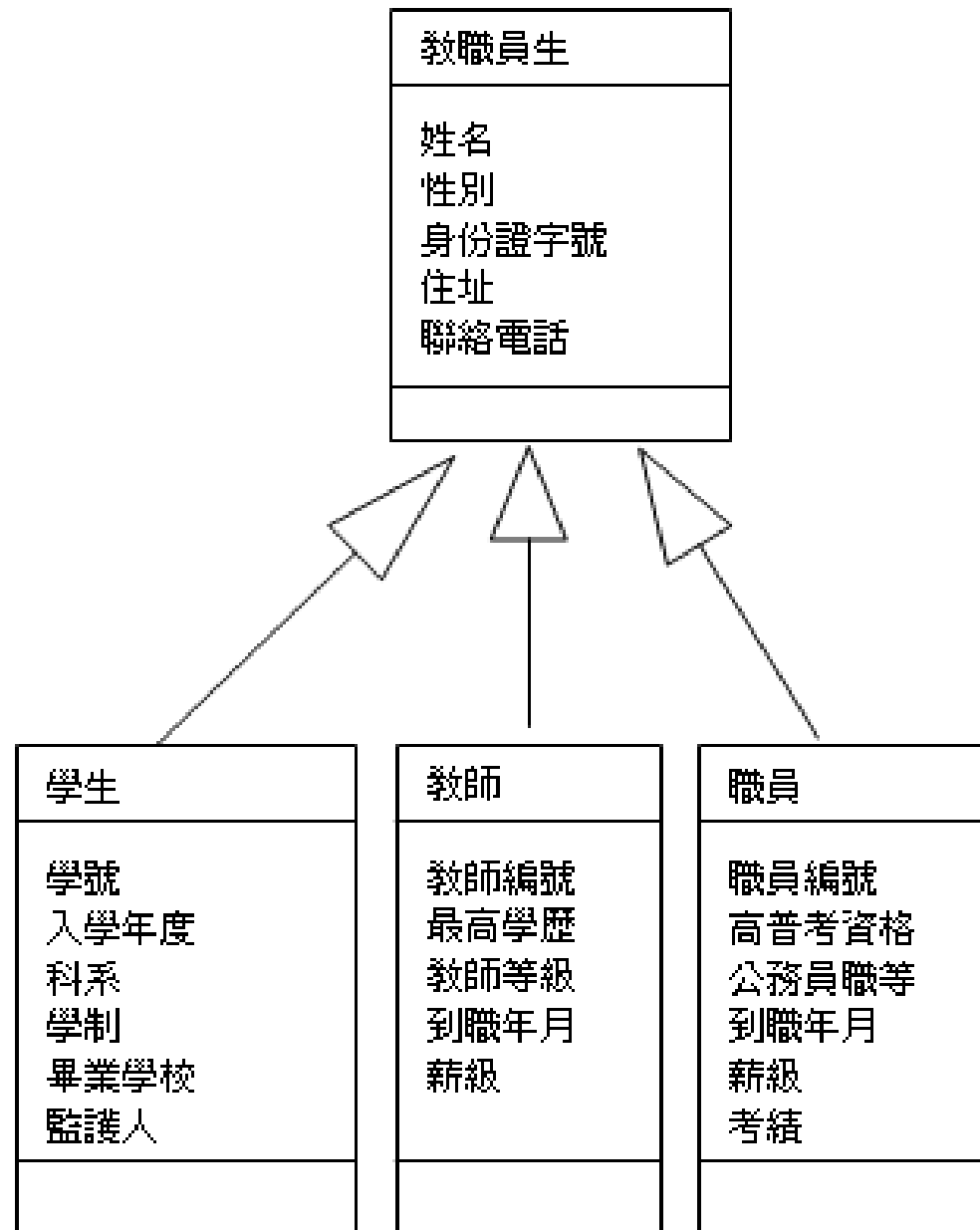
- 有時為了讓概念模型更簡潔，可將類別的屬性或方法省略。
- 屬性及方法的透明度(visibility)：
 - 公開(public)：+
 - 私有(private)：-
 - 保護(protected)：#
 - 套裝(package)：~

類別圖：符號 ²⁻²

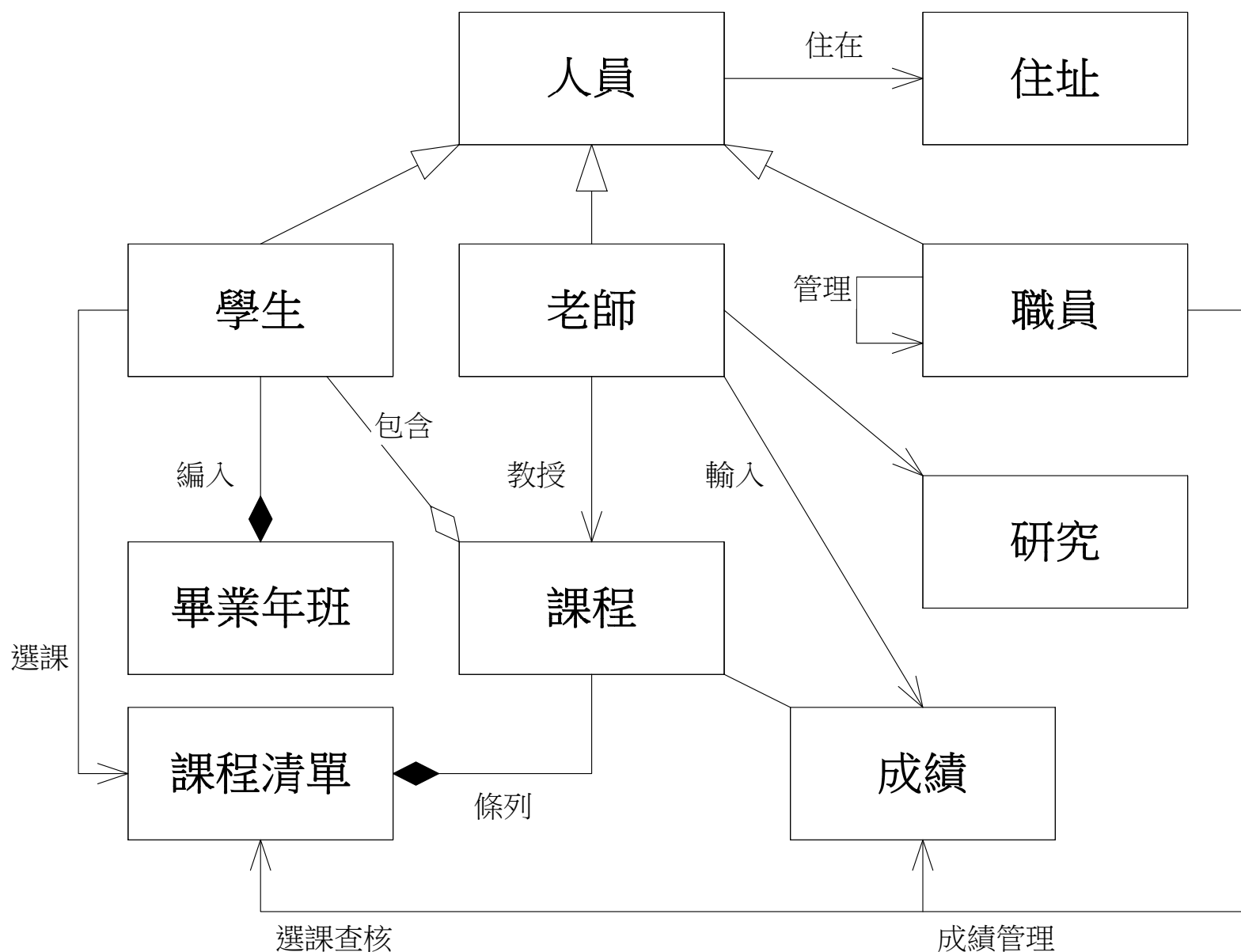
－ 類別間之關係類型及符號：

- 聯合(association)：雙向 ，單向 
 - 聯合關係的兩端，可標示多重性(multiplicity)。
 - 標示方式：*, 0, 1, 0..*, 1..*, etc.
 - 範例： 表示一個航班最多可指定一架飛機飛，亦可暫時無飛機飛；一架飛機可飛多個航班，亦可暫時無指派航班。
- 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 ----->

類別圖範例：繼承



類別圖範例：教學管理



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。
 - * 本頁內容可與p.68之MVC pattern內容相互參照。
- 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(OOA)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(OOD)階段，著重在物件之間的訊息傳遞，通稱為物件順序圖。

順序圖：符號

- 符號：

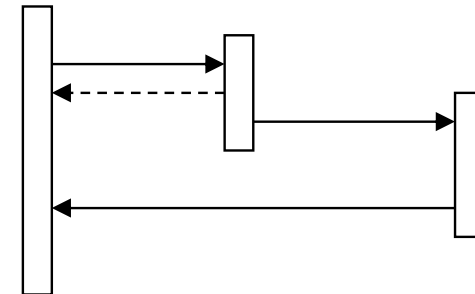
- 物件：

物件名稱：類別名稱

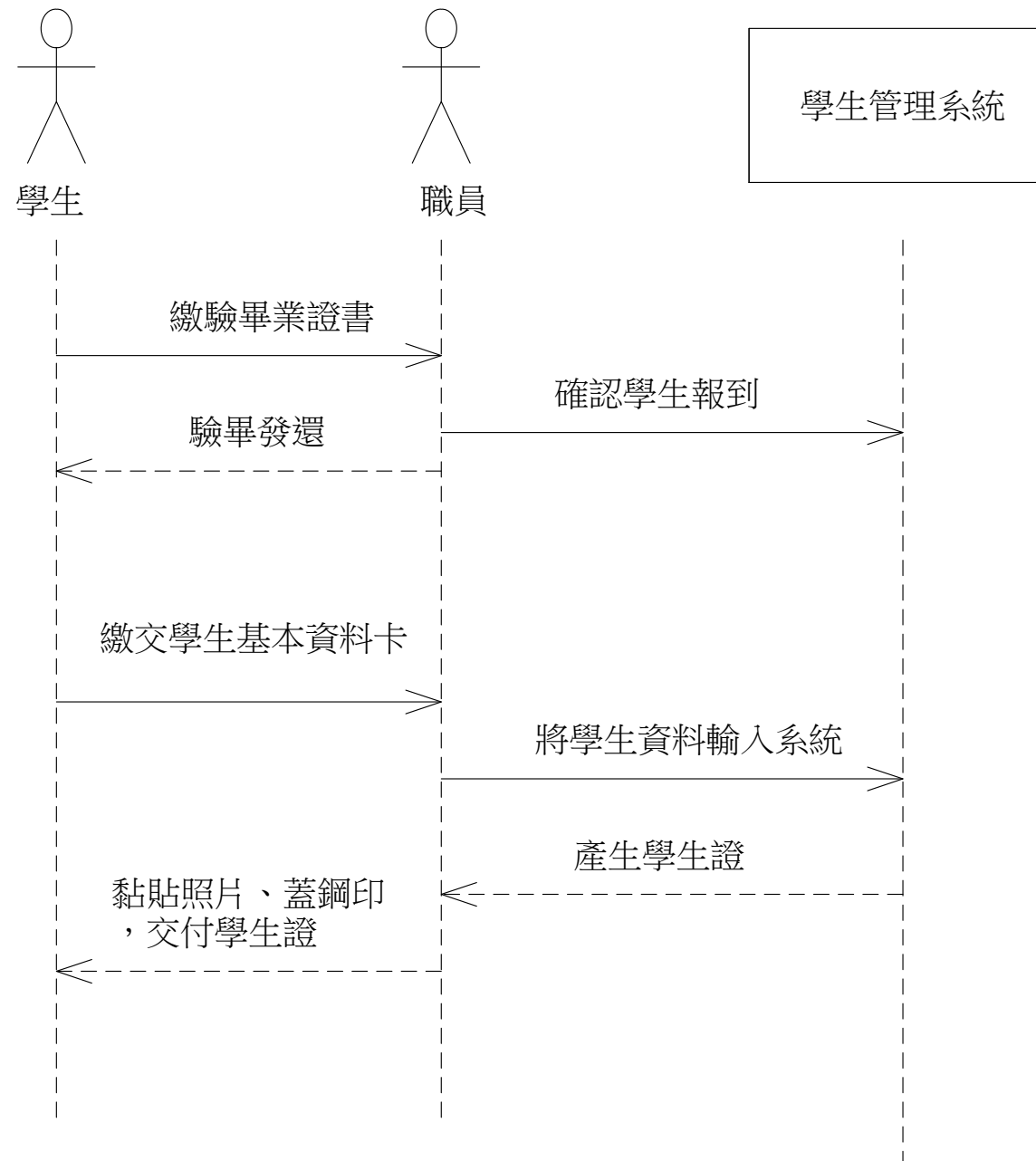
，或

:類別名稱

- 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

- **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。
- **Model-view-controller (MVC) pattern**：MVC樣式是從Smalltalk時代即存在至今的最通用樣式，它把OO應用程式的物件分成Model (模型)、View (檢視畫面)及Controller (控制器)等三種類型，一方面可以把複雜的系統架構分解成較易處理的片段，另一方面可讓系統開發團隊分頭同時進行不同類型物件的編程，提高OOP的效率。
 - **Model**：負責管理資料及系統的核心行為。Model會回應來自View對系統狀態的檢視請求，也會接收來自Controller要求改變系統狀態的指令。
 - **View**：負責把Model的狀態顯示為能與使用者互動的GUI介面。一個Model通常會對應多個Views。
 - **Controller**：把使用者下達的指令轉達給Model及View做對應的處理。

API、資料結構、演算法

- 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，大部分時間是在學習它的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。

系統描述工具在會計上的應用

- 本講義介紹的前三種描述工具在實務上應用的十分廣泛，因此在**AIS**教科書中也廣獲採用。例如：
 - **Gelinas, Dull, Wheeler and Hill (2017, 11th ed.)**：使用邏輯DFD及結構嚴謹的系統流程圖詳細介紹各個交易循環的內容。
 - **Romney, Steinbart, Summers and Wood (2020, 15th ed.)**：使用邏輯DFD及結構較為鬆散的系統流程圖詳細介紹各個交易循環的內容。某些章節亦使用程序圖做補充說明。
 - **Hall (2015, 9th ed.)**：使用邏輯DFD、文件流程圖及系統流程圖詳細介紹各個交易循環的內容。
 - **Turner and Weickgenannt (2016, 3rd ed.)**：使用程序圖、文件流程圖及邏輯DFD詳細介紹各個交易循環的內容。

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，較少提及**UML**及**OOP**的概念。
 - **Jones & Rama (2006)** 是少數以**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。
 - **Dunn (2012 eBook)** 以**UML**的**class diagram**描述各個交易循環的內容，是目前技術含量最高的**AIS**教科書。
 - 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>