

UML 系列之一

UML 答客問

第一輯

作者簡介

邱郁惠(271080@gmail.com)

畢業於東吳大學資訊科學系，研究 UML、OOA 十餘年，2007 年創辦 UML

Blog(<http://www.uml.tw>)推廣 UML 技術，並專職於企業內訓、專案輔導、自辦課程、專欄寫作。

擔任過 NEC、華夏、中科院、百通、MISOO 物件教室、大竝、中華汽車、HSDc(2007)、資策會(2008)、台灣大哥大(2008)、意藍科技(2008)、新鼎(2008)、博客來(2009)、網飛訊(2010)、文化大學推廣部(2010)、北區職訓局(2010)、PMI-TW 國際專案管理學會(2010)、巨鷗(2010)、三商電腦(2010)、秀傳醫療(2011)、翔生資訊(2011)、國際厚生(2011)、中華電信(2012)等公司的內訓講師及輔導顧問，也擔任過物件導向雜誌主編暨 UML/OOAD 專欄作家、RUN！PC 旗標資訊月刊(2008~2009)、以及 iThome 電腦報(2008~2012)專欄作家。

出版的繁體書有《寫給 SA 的 UML/MDA 實務手冊》(天瓏銷售第 1 名,已絕版)、《寫給 C++程式設計師的 UML 實務手冊》(天瓏銷售第 4 名,已絕版)、《UML-SystemC 晶片設計實務》(已絕版)、《OCUP/UML 初級認證攻略》(天瓏銷售第 14 名,已絕版)、《寫給 SA 的 UML/UseCase 實務手冊》(天瓏銷售第 10 名,已絕版)、《學會 UML/OOAD 這樣開始就對了》(金石堂預購第 1 名,已絕版)、《Visual Studio 2010/UML 黃金準則》、《SA 前進 UML 專案現場》、《IT 人，你如何表達？》(未出紙本版)。

同時，出版的簡體書有《系統分析員 UML 實務手冊》、《C++程式師 UML 實務手冊》、《SoC 設計實務手冊》、《UML 那些事兒》、《系統分析師 UML 用例實戰》、《UML 和 OOAD 快速入門》、《Visual Studio 2010 和 UML 黃金法則》、《系統分析師 UML 項目實戰》。

目前擁有 OCUP(OMG-Certified UML Professional)三級認證、PMP(Project Management Professional)認證、ITIL V3 Foundation 認證、IBM OOAD(Object Oriented Analysis and Design)認證、Scrum Master 認證，曾榮獲大陸頒予《優秀 IT 技術圖書原創作者》獎。

目錄



- Q1 “object association” 是？ ---- 2
- Q2 請問 UML 六種箭頭的分別？ ---- 11
- Q3 何處有 UML 分析設計範例？ ---- 26
- Q4 如何抓出系統的類別？ ---- 44
- Q5 畫 UML 圖的先後順序？ ---- 52
- Q6 請教有關活動圖的問題？ ---- 62
- Q7 請問會員系統的 use case？ ---- 69
- Q8 如何在循序圖裡畫 iteration？ ---- 82
- Q9 類別的建構元要畫在哪啊？ ---- 103
- Q10 請問 UML 工具？ ---- 108

目錄

- Q1 “object association” 是？ ---- 2
- Q2 請問 UML 六種箭頭的分別？ ---- 11
 - Q2.1 六種箭頭 ---- 13
 - Q2.2 聚合關係 ---- 15
 - Q2.3 組合關係 ---- 17
- Q3 何處有 UML 分析設計完整範例？ ---- 26
 - Q3.1 CIM-1：定義企業流程 ---- 26
 - Q3.2 CIM-1：定義企業流程 ---- 28
 - Q3.3 CIM-2：分析企業流程 ---- 29
 - Q3.4 CIM-3：定義系統範圍 ---- 30
 - Q3.5 PIM-1：分析系統流程 ---- 32
 - Q3.6 PIM-2：分析企業規則 ---- 36
 - Q3.7 PIM-3：定義靜態結構 ---- 38
 - Q3.8 PIM-4：定義操作及方法 ---- 39
- Q4 如何抓出系統的類別？ ---- 44
 - Q4.1 五種常見的物件種類 ---- 45
 - Q4.2 善用交易樣式 ---- 49
- Q5 畫 UML 圖的先後順序？ ---- 52
- Q6 請教有關活動圖的問題？ ---- 62
 - Q6.1 決策或合併 ---- 64

- Q6.2 避免無謂的決策 ---- 67
- Q6.3 條件不重疊、不遺漏 ---- 68
- Q7 請問會員系統的 use case ? ---- 69
 - Q7.1 包含關係 ---- 72
 - Q7.2 擴充關係 ---- 77
- Q8 如何在循序圖裡畫 iteration ? ---- 82
 - Q8.1 片段 ---- 85
 - Q8.2 引用 ---- 88
 - Q8.3 隨意 ---- 94
 - Q8.4 擇一 ---- 98
 - Q8.5 並行 ---- 98
 - Q8.6 迴圈 ---- 100
 - Q8.7 中斷 ---- 101
- Q9 類別的建構元要畫在哪啊 ? ---- 103
 - Q9.1 建構元與解構元 ---- 104
 - Q9.2 物件之生滅 ---- 106
- Q10 請問 UML 工具 ? ---- 108
 - Q10.1 免費的 UML 開發工具—StarUML ---- 109
 - Q10.2 新增使用案例圖 ---- 113
 - Q10.3 繪製使用案例圖 ---- 116
 - Q10.4 編寫使用案例敘述 ---- 122

UML 答客問

1

※※

- Q1 “object association”是？
- Q2 請問 UML 六種箭頭的分別？
- Q3 何處有 UML 分析設計完整範例？
- Q4 如何抓出系統的類別？
- Q5 畫 UML 圖的先後順序？
- Q6 請教有關活動圖的問題？
- Q7 請問會員系統的 use case？
- Q8 如何在循序圖裡畫 iteration？
- Q9 類別的建構元要畫在哪啊？
- Q10 請問 UML 工具？

Q1 “object association”是？

“object association”是？

(hyaloid於程式設計俱樂部之提問)

請問在程式寫作上“object association”是怎樣的觀念，或作法呢？

簡答：

What—兩物件間具有靜態且固定的連結(link)時，對應到它們所屬的類別上，兩類別之間即存有「結合關係」(association)。

Why—透過靜態且固定的連結，一物件便可以呼叫另一物件之操作，使得一群物件可以協力提供完善的服務。

How—以C++為例，可以在類別中宣告靜態的物件或物件指標，就跟宣告資料成員(data member)的作法相同。

物件鮮少獨自存在，通常都會與其他種類的物件連結(link)，成就一加一大於二的綜效。所以，我們可以在類別之間建立結合關係(association relationship)，用來表達兩類別所產物件之間的連結情況。整個系統內部的物件透過這些連結，形成點點相連的物件網絡，協力對外提供系統服務。

請看物件圖 1-1，多次申購富邦長虹這檔基金，因此一個基金物件會連結到多個申購交易物件。從某一個申購交易物件處，可以連結到申購的基金物件，即刻獲知該基金的最新淨值及手續費。

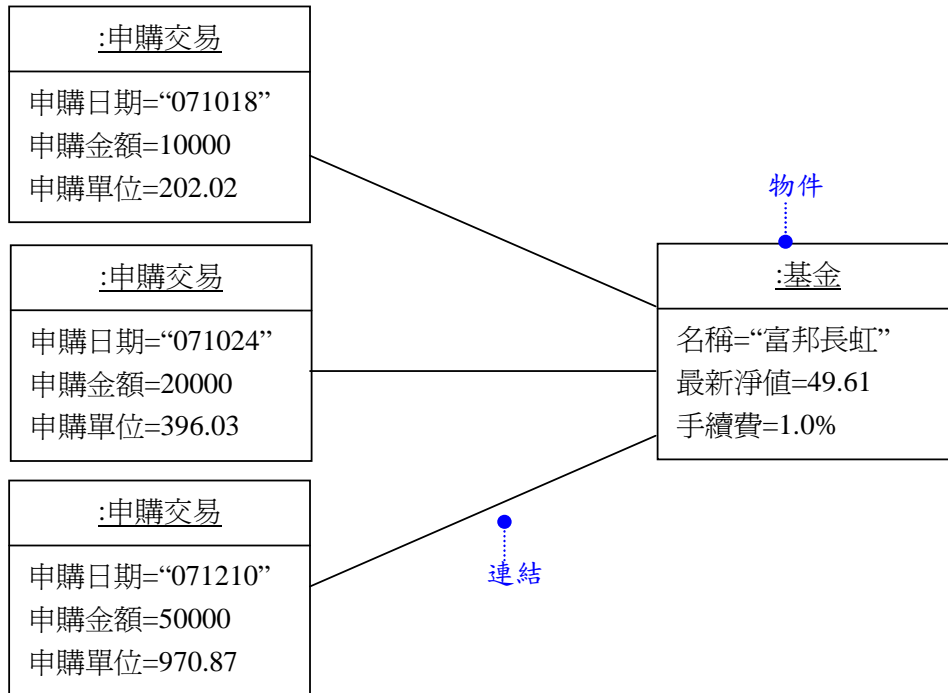


圖 1-1: 物件之間的連結

投資人可以多次申購同一檔基金，而任一申購交易成立時，只能夠指定申購一檔基金；兩種物件之間的連結，有諸如此類的細節資訊，我們一併記錄在結合關係的兩結合端(association end)處。兩物件之間的連結數目，稱為個體數目(multiplicity)。

看到圖 1-2，基金與申購交易兩類別之間有結合關係，且一個基金物件可以連結零到無限多(0..*)個申購交易物件，但一個申購交易物件至多至只能也一定要連結到一個基金物件。

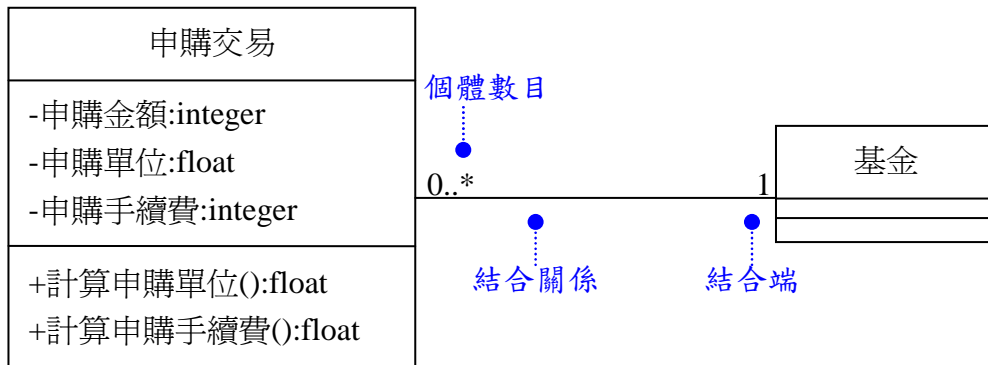


圖 1-2: 結合關係

結合關係的細節資訊，通常放置於兩結合端，像是結合端標示箭頭，代表該結合端具可航行性(navigation)，如圖 1-3 所示。因此，申購交易物件可以循著箭頭方向航行到基金物件處，且因為基金端的個體數目為 1，所以執行期間的物件圖，會如圖 1-4，兩物件之間只有一個連結。在前述看到的類別圖中，兩結合端若沒有特別標示箭頭，通常意味著還未定義結合端的可航行性。

此外，個體數目的表示法是「下限..上限」，最小數目為零(0)，最大數目為無設限(*)，若僅標示一個數目即上下限相同。最單純的狀況是「1」，至少至多只能連接一個物件。

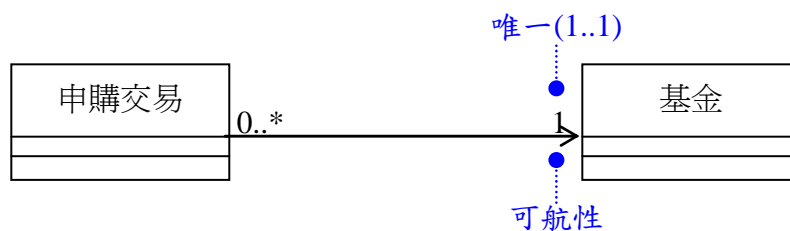


圖 1-3: 可航性

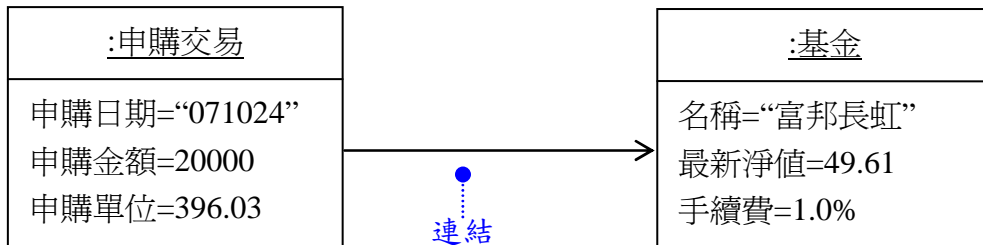


圖 1-4: 連結

最後，我們以圖 1-5 為例，對照說明 C++ 程式碼如何實作出個體數目為 1 的結合關係。此外，我們在 main.cpp 中產生了兩個物件，如圖 1-6 所示。在使用者輸入基金淨值、手續費和申購金額之後，申購交易物件會算出申購單位和手續費，執行畫面如圖 1-7 所示。

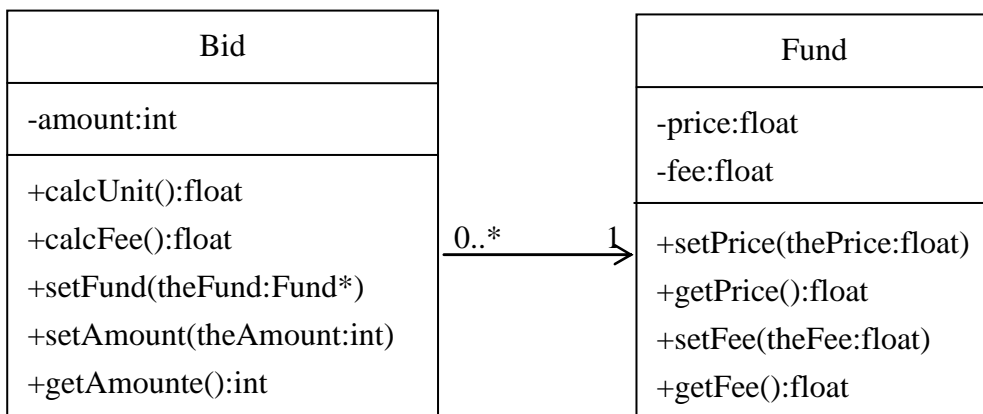


圖 1-5: 個體數目為 1



圖 1-6: 物件圖



圖 1-7: 執行畫面

下述的圖 1-8~12 為 C++的原始程式碼，特別需要注意之處，如下所述：

- **Fund.h**—特別注意到圖 1-8 的行號 6，“public”關鍵字標記著公開等級的能見度，往下從行號 6~9 都是公開等級的操作，可由外界視見並使用之。至於，行號 11 的“private”關鍵字則標記出私有等級的能見度，所以在行號 11~12 宣告皆為私有屬性。
- **Bid.h**—特別注意到圖 1-10 的行號 4，將 **Fund.h** 含入之後，才能於稍後的行號 15 中，宣告一個基金物件指標，此即為個體數目為 1 的實作方法之一。行號 10 設計了一個公開的操作，讓外界可以傳入基金物件的指標，也就建立起申購交易物件指向基金物件的連結了。

- Bid.cpp—圖 1-11 的行號 7 和 14 中，申購交易物件可以透過連結呼叫基金物件的公開操作，取得基金的最新淨值和手續費。
- main.cpp—圖 1-12 的行號 10 和 12 產生了一個名為 myBid 的申購交易物件，和另一個名為 myFund 的基金物件。隨後，在行號 23 中，將基金物件的指標傳給申購交易物件，就此建立起申購交易物件指向基金物件的連結。最後，在行號 25~26 呼叫申購交易物件的公開操作 calcUnit()、calcFee()，計算申購單位及手續費。

```
//// EX01_01
1.  // Fund.h
2.  //
3.  #pragma once
4.  class Fund
5.  {
6.  public:
7.      void setPrice(float);
8.      float getPrice();
9.      void setFee(float);
10.     float getFee();
11. private:
12.     float price;
13.     float fee;
14. };
//// EX01_01
```

圖 1-8: Fund.h

```
//// EX01_01
1.  // Fund.cpp
2.  //
3.  #include "Fund.h"
4.  void Fund::setPrice(float thePrice)
```

```
5.  {
6.      price=thePrice;
7.  }

8.  float Fund::getPrice()
9.  {
10.     return price;
11. }

12. void Fund::setFee(float theFee)
13. {
14.     fee=theFee;
15. }

16. float Fund::getFee()
17. {
18.     return fee;
19. }
///  
EX01_01
```

圖 1-9: Fund.cpp

```
///  
EX01_01
1.  // Bid.h
2.  //

3.  #pragma once
4.  #include "Fund.h"

5.  class Bid
6.  {
7.  public:
8.      float calcUnit();
9.      float calcFee();
10.     void setFund(Fund*);
11.     void setAmount(int);
12.     int getAmount();

13. private:
14.     int amount;
15.     Fund *fundObj;
16. };
```

```
//// EX01_01
```

圖 1-10: Bid.h

```
//// EX01_01
1.  // Bid.cpp
2.  //
3.  #include "Bid.h"
4.  float Bid::calcUnit()
5.  {
6.      float thePrice, theUnit;
7.      thePrice=fundObj->getPrice();
8.      theUnit=amount/thePrice;
9.      return theUnit;
10. }
11. float Bid::calcFee()
12. {
13.     float theFundFee, theFee;
14.     theFundFee=fundObj->getFee();
15.     theFee=amount*theFundFee;
16.     return theFee;
17. }
18. void Bid::setFund(Fund *theFund)
19. {
20.     fundObj=theFund;
21. }
22. void Bid::setAmount(int theAmount)
23. {
24.     amount=theAmount;
25. }
26. int Bid::getAmount()
27. {
28.     return amount;
29. }
//// EX01_01
```

圖 1-11: Bid.cpp

```
//// EX01_01
1.  // main.cpp
2.  //

3.  #include <cstdlib>
4.  #include <iostream>
5.  #include "Fund.h"
6.  #include "Bid.h"
7.  using namespace std;

8.  int main(int argc, char *argv[])
9.  {
10.     Fund myFund;
11.     float thePrice, theFee;
12.     Bid myBid;
13.     int theAmount;

14.     cout << "請輸入大華大華基金淨值： ";
15.     cin >> thePrice;
16.     myFund.setPrice(thePrice);
17.     cout << "請輸入大華大華基金手續費： ";
18.     cin >> theFee;
19.     myFund.setFee(theFee);

20.     cout << "請輸入申購金額： ";
21.     cin >> theAmount;
22.     myBid.setAmount(theAmount);
23.     myBid.setFund(&myFund);

24.     cout << "您申購的單位及手續費為： "
25.         << "(" << myBid.calcUnit() << ")"
26.         << "(" << myBid.calcFee() << ")" << endl << endl;

27.     system("PAUSE");
28.     return EXIT_SUCCESS;
29. }
//// EX01_01
```

圖 1-12: main.cpp

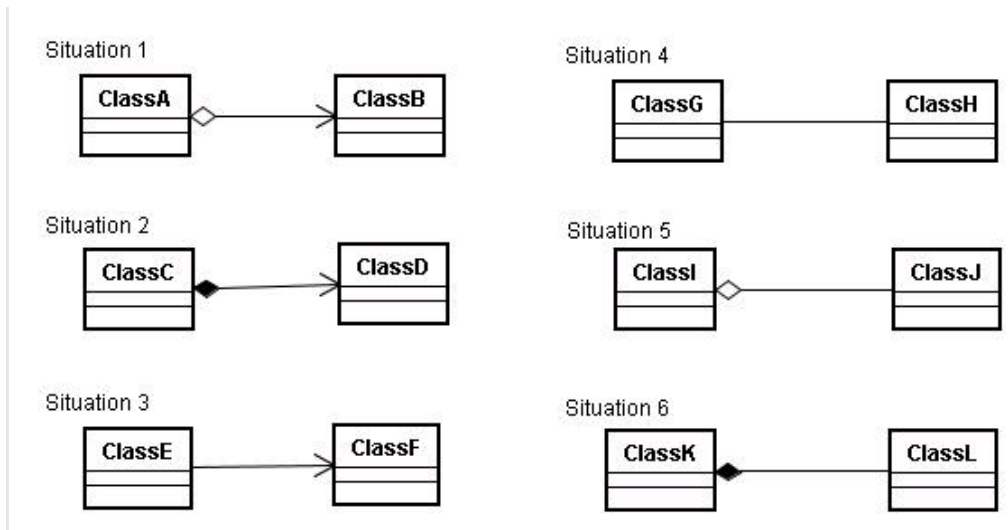
Q2 請問 UML 六種箭頭的分別？

請問UML六種箭頭的分別？

(kitsam於JavaWorld@TW之提問)

小弟一直在自修軟體設計模式，過程中需要學習UML，但小弟發覺教設計模式的書本中，它們所畫的UML有時用空心菱形箭頭，但有時卻用實心菱形箭頭，而有時用一個沒有菱形的箭頭，但我卻看不出它們的分別，而書也沒有提及，最後我卻越看越亂，當我想畫一張UML來表達一些class時，經常遇到的就是不知道何時用那條箭頭，是有菱形還是沒有菱形的？

以下的UML就包含了六種不同的箭頭，小弟就是不明白那六種箭頭的分別，我曾嘗試過在JUDE繪畫後export成Java code，但發覺那六個situation所export出來的Java code都是一樣的。請問以下六種箭頭的分別在那裡？



簡答：

結合關係(association)有兩種變形，一種為空心菱形的「聚合關係」(aggregation)，另一種為實心菱形的「組合關係」(composition)。聚合或組合關係兩端的物件，一端扮演「整體物件」(whole object，整件)，另一端扮演「部分物件」(part object，部件)。在結合關係中，兩端的物件並不具有「整體-部分」(whole-part)的特色。

一般，我們可以透過檢核下列四項要件，判斷兩類別之間採用結合、聚合還是組合較適當：

1. 在企業領域的專業概念裡，兩種物件之間有一種固定不變且需要保存的靜態關係。(結合關係的要件)
2. 在資訊化時，系統會用到這些靜態關係，而且必須將它們存到資料庫。(結合關係的要件)

(符合上述要件1~2，即可採用結合關係)

3. 在企業領域的專業概念裡，兩種物件之間有整體-部分的靜態關係。(聚合關係的要件)

(符合上述要件1~3，即可採用聚合關係)

4. 部件只能連結一個整件，且整件被註銷(destroy)時，部件必須一塊被註銷。(組合關係的要件)

(符合上述要件1~4，即可採用組合關係)

Q2.1 六種箭頭

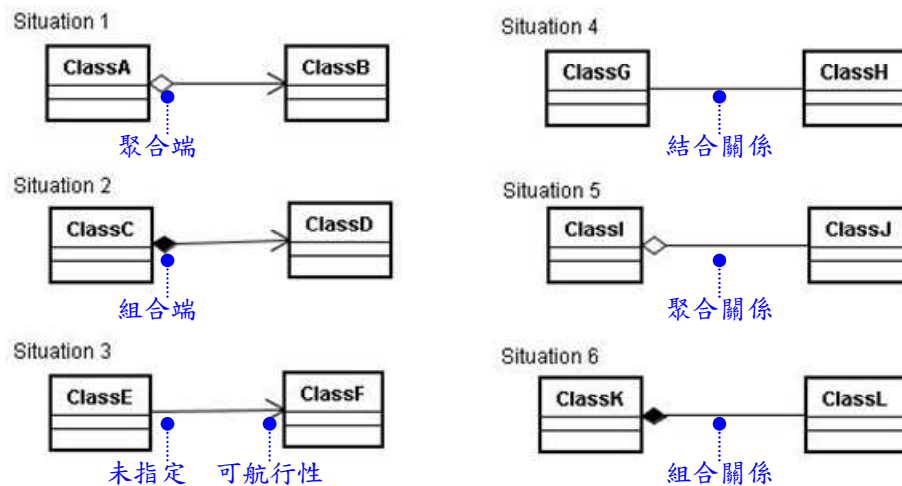


圖 1-13: 六種箭頭

首先，我們先來說明圖 1-13 中的 UML 圖示，如下：

- 實線—實線圖示稱為「結合關係」(association)，表達兩物件之間有靜態的連結(link)，如圖 1-13 的情況 3 和 4 都是結合關係。
- 箭頭—箭頭圖示稱為「可航行性」(navigation)，表達可由另一端物件連結指向具箭頭端物件。所以，在圖 1-13 的情況 3 中，類別 E 所產出的物件將具有一連結關係，可以連結到類別 F 所產出的物件。因此，我們會說，F 物件具有可航行性。
- 空心菱形—其中一端具空心菱形的圖示稱為「聚合關係」(aggregation)，它是結合關係的變形，表達兩物件具有「整體-部分」(whole-part)的特質，如圖 1-13 的情況 1 和 5 都是聚合關係。
- 實心菱形—其中一端具實心菱形的圖示稱為「組合關係」(composition)，它也是結合關係的變形，表達兩物件具有「整體-部分」的特質。此外，部件(part object)只能連結一個整件(whole object)，且整件被註銷(destroy)時，部件必須一塊被註銷。圖 1-13 的情況 2 和 6 即為組合關係。

再者，箭頭可以放置於任何一端，標示可航行性，若未放置箭頭，代表「未指定」(unspecified)，也就是還未指定可航性或不可航行。新版的 UML 提出另一個打叉的圖示，用來標記不可航行端，如圖 1-14 所示。

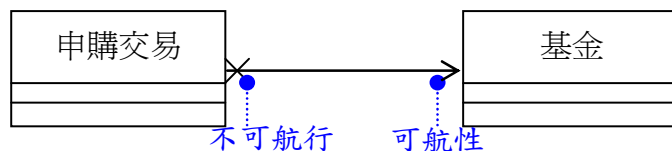


圖 1-14: 不可航行

倘若要標示雙向的可航行性，就得如圖 1-15 所示，兩端都放置箭頭。

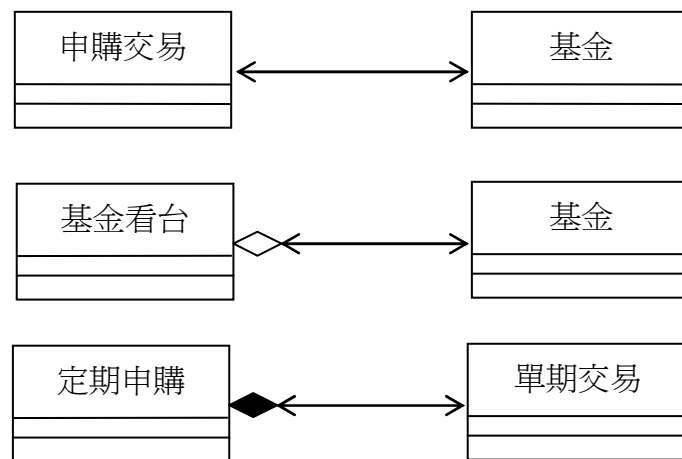


圖 1-15: 雙向的可航行性

Q2.2 聚合關係

接著，我們來看一個「基金看台」的例子，以此說明聚合關係。每一個基金看台可以設定多檔自選基金，如此一來，投資人可以同時觀察某一群組的基金報酬率，如圖 1-16 所示。假設，我們分別以短期、中期和長期這三個投資角度來設置看台，並且將日盛上選基金同時選入這三個基金看台中。

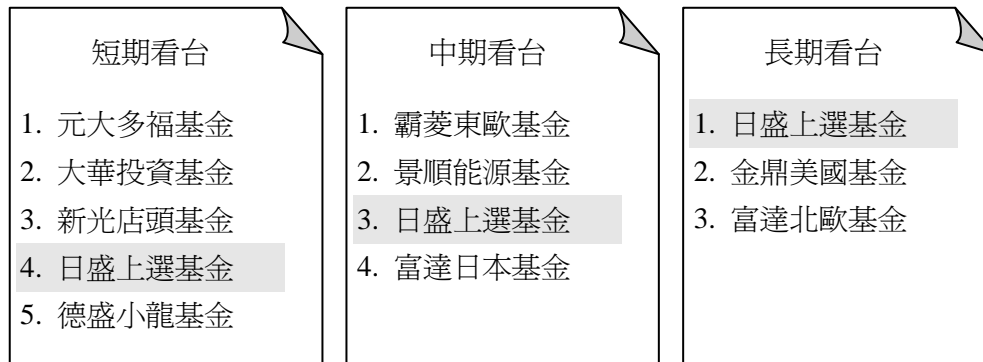


圖 1-16: 同一檔基金可以被選入多個基金看台中

如果採用結合關係的設計，可以獲得如圖 1-17 的設計圖，不過卻無法表達出基金看台與基金之間的整體-部份特色。所以，這種情況下，就可以考慮改用聚合關係，如圖 1-18 所示。

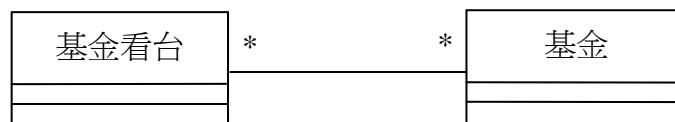


圖 1-17: 結合關係

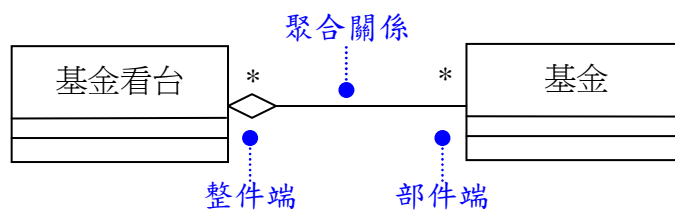


圖 1-18: 聚合關係

聚合關係跟結合關係的最主要差異，就在於整體-部份特質，除此之外，兩者就沒有什麼差異了。也因此，在 C++ 或 Java 語言的實作上，聚合關係與結合關係的實作方法相同，兩者並無區別。

像在結合關係的範例中，我們可以在外界產生某一物件，然後將該物件指標傳送給另一物件。無論是結合關係或者是聚合關係，可以採用由第三者(外界)分別產生兩物件，同時建立起兩物件之連結的實作方法。

來看圖 1-19 的示意圖，由 AP(應用程式)先分頭產生帳戶物件和申購交易物件，然後再將申購交易物件的指標傳給帳戶物件，如此建立起帳戶與申購交易兩物件之間的連結。所以，AP 可以直接呼叫申購交易物件，帳戶物件也可以透過指標間接呼叫申購交易物件。針對結合或聚合關係，實務上常用此種實作手法。

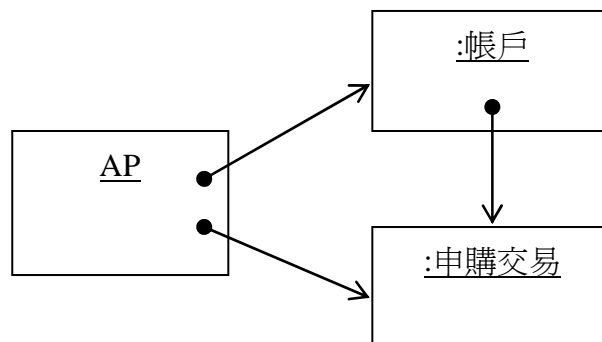


圖 1-19: 由第三者產生兩物件

Q2.3 組合關係

不過組合關係就大不同了，它除了同樣具有整體-部分的特色外，跟聚合關係最大的不同之處在於，組合關係中的整件直接掌握部件的生滅，聚合關係中的整件並不具有生滅部件的權力。一旦，組合中的整件不復存

在時，其組部件也不能單獨存在，必須同時消滅。再者，外界也不能直接與部件溝通，必須透過整件代為傳達訊息。

因此，在 C++ 或 Java 的實作上，聚合關係中的整件不能被動擁有外界傳來的部件指標，而應該直接掌握部件的生滅大權，這與結合或聚合的實作方法大不同。換言之，在實作組合關係時，不能由第三者產生部件之後，才將部件指標傳給整件，而是應該由整件直接負責部件之生滅。

先來看圖 1-20 的例子，投資人跟銀行約定定期定額申購基金之後，每月的約定日一到，就會自動產生一筆定額交易。在這樣的例子中，定期定額申購與單期交易就非常適合使用組合關係，一旦某筆定期定額申購被刪去時，底下的單期交易就不該在繼續存留。

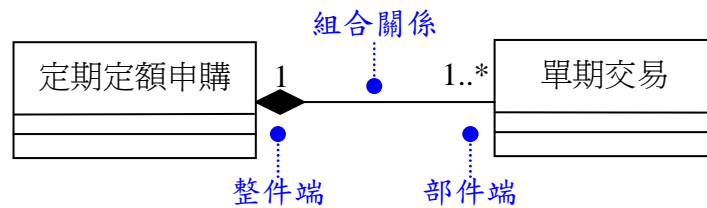


圖 1-20: 組合關係

再看到圖 1-21 的示意圖，帳戶物件與申購交易物件之間為一般性的結合關係，而定期定額申購與單期交易之間則為嚴格的組合關係。

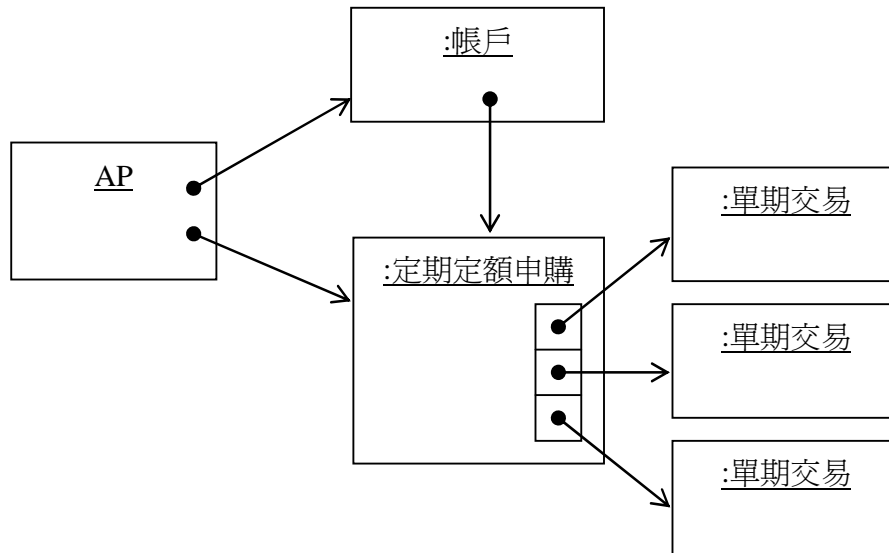


圖 1-21: 結合與組合

現在，我們以圖 1-22 為例，說明如何用 C++ 來實作組合關係。在這個範例中，定期定額申購(RegularBid)物件是整件必須負責產生它的部件，即為單期交易(BidItem)物件，兩者之間為組合關係。而每一個單期交易物件會連結到一個基金物件。最後，在 main 中，將呼叫定期定額申購物件，請它連結到部件小計資產，而後再累計算出總資產。

為了降低這個範例的複雜度，沒有讓使用者從外界輸入資料，而是直接在 main.cpp 中產出如圖 1-23 所示的物件。在 main 中，將產生定期定額申購物件和基金物件，至於單期交易物件則由定期定額申購物件負責產生。再看到圖 1-24 是執行畫面，計算得出總資產。

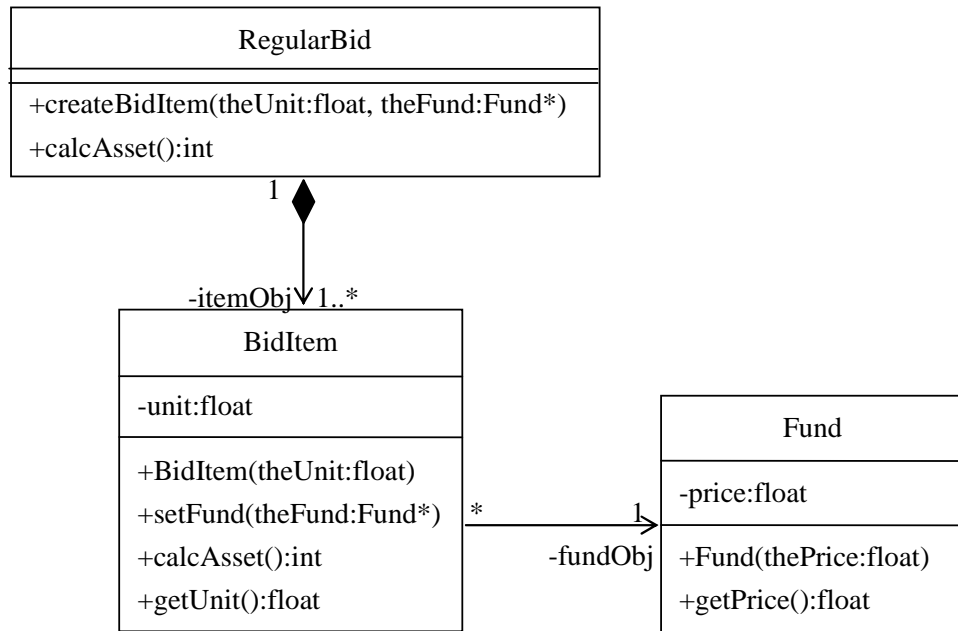


圖 1-22: 組合關係

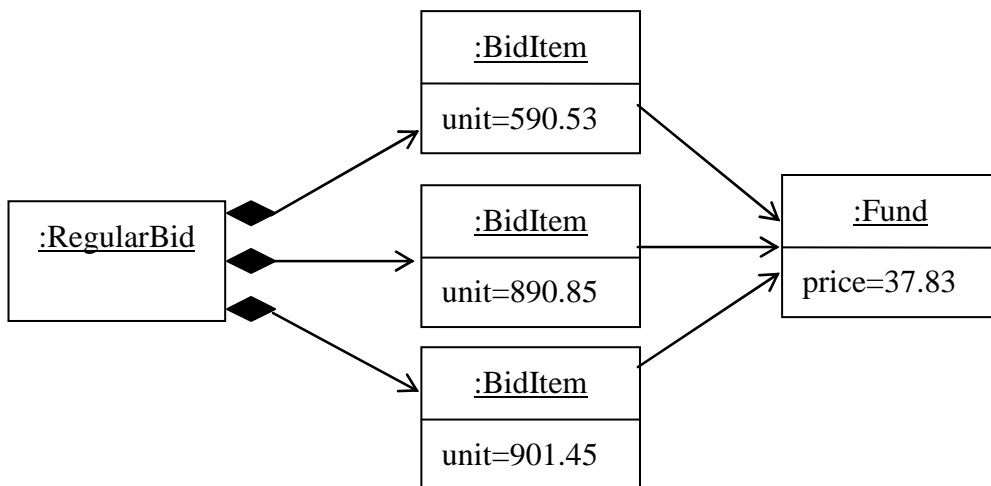


圖 1-23: 物件圖

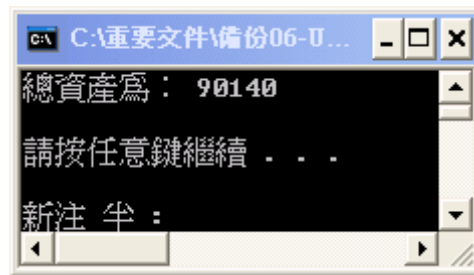


圖 1-24: 執行畫面

下述的圖 1-25~31 為 C++的原始程式碼，特別需要注意之處，如下所述：

- RegularBid.h—特別注意到圖 1-27 的行號 14，定期定額申購物件為組合中的整件，所以此處宣告了其部件。至於行號 11 的公開操作，用來產生一個新的單期交易物件。
- RegularBid.cpp—看到圖 1-28 的行號 6~8，此處產生一個新的單期交易物件，並建立起與基金物件之間的連結，隨後將單期物件指標存入向量物件中。
- BidItem.h 與 BidItem.cpp—在圖 1-29~30 中，同樣為了降低範例的複雜度，所以我們在產生單期物件之際，便透過建構元設定好單期的申購單位。
- main.cpp—看到圖 1-31 的行號 11~14 中，產生基金和單期交易物件之際，便直接設定基金淨值和申購單位數。

```
//// EX01_02
1.  // Fund.h
2.  //
```

```
3.  #pragma once
4.  class Fund
5.  {
6.  public:
7.      Fund(float);
8.      float getPrice();

9.  private:
10.     float price;
11. };
//// EX01_02
```

圖 1-25: Fund.h

```
//// EX01_02
1.  // Fund.cpp
2.  //

3.  #include "Fund.h"

4.  Fund::Fund(float thePrice)
5.  {
6.      price=thePrice;
7.  }

8.  float Fund::getPrice()
9.  {
10.     return price;
11. }
//// EX01_02
```

圖 1-26: Fund.cpp

```
//// EX01_02
1.  // RegularBid.h
2.  //

3.  #pragma once
4.  #include <cstdlib>
5.  #include <vector>
```

```
6.  #include "BidItem.h"
7.  using namespace std;

8.  class RegularBid
9.  {
10. public:
11.     void createBidItem(float,Fund*);
12.     int calcAsset();

13. private:
14.     vector<BidItem*> itemObj;
15. };
//// EX01_02
```

圖 1-27: RegularBid.h

```
//// EX01_02
1.  // RegularBid.cpp
2.  //

3.  #include "RegularBid.h"

4.  void RegularBid::createBidItem(float theUnit,Fund *theFund)
5.  {
6.      BidItem *myItem=new BidItem(theUnit);
7.      myItem->setFund(theFund);
8.      itemObj.push_back(myItem);
9.  }

10. int RegularBid::calcAsset()
11. {
12.     int size,theAsset=0;
13.     size=itemObj.size();
14.     for(int i=0;i<size;i++)
15.         theAsset=theAsset+itemObj[i]->calcAsset();
16.     return theAsset;
17. }
//// EX01_02
```

圖 1-28: RegularBid.cpp

```
//// EX01_02
1.  // BidItem.h
2.  //

3.  #pragma once
4.  #include "Fund.h"

5.  class BidItem
6.  {
7.  public:
8.      BidItem(float);
9.      void setFund(Fund*);
10.     int calcAsset();
11.     float getUnit();

12. private:
13.     float unit;
14.     Fund *fundObj;
15. };
//// EX01_02
```

圖 1-29: BidItem.h

```
//// EX01_02
1.  // BidItem.cpp
2.  //

3.  #include "BidItem.h"

4.  BidItem::BidItem(float theUnit)
5.  {
6.      unit=theUnit;
7.  }

8.  void BidItem::setFund(Fund *theFund)
9.  {
10.     fundObj=theFund;
11. }

12. int BidItem::calcAsset()
13. {
14.     return unit*fundObj->getPrice();
}
```

```
15. }  
16. float BidItem::getUnit()  
17. {  
18.     return unit;  
19. }  
//// EX01_02
```

圖 1-30: BidItem.cpp

```
//// EX01_02  
1. // main.cpp  
2. //  
  
3. #include <cstdlib>  
4. #include <iostream>  
5. #include "RegularBid.h"  
6. #include "Fund.h"  
7. using namespace std;  
  
8. int main(int argc, char *argv[])  
9. {  
10.     RegularBid myRegularBid;  
11.     Fund *myFund=new Fund(37.83);  
12.     myRegularBid.createBidItem(590.53,myFund);  
13.     myRegularBid.createBidItem(890.85,myFund);  
14.     myRegularBid.createBidItem(901.45,myFund);  
  
15.     cout << "總資產為： "  
16.         << myRegularBid.calcAsset() << endl << endl;  
  
17.     system("PAUSE");  
18.     return EXIT_SUCCESS;  
19. }  
//// EX01_02
```

圖 1-31: main.cpp

Q3 何處有 UML 分析設計完整範例？

何處有UML分析與設計完整範例圖？

(samswan於程式設計俱樂部之提問)

請教各位先進：最近在看活學活用UML與樣式第二版，感覺上觀念很凌亂，不知道各位是怎麼學習過來的？還有是否有完整的實作案例可參考？

簡答：

我寫了一個基金系統的模擬個案，請您到UML Blog(<http://www.uml.tw.com>)下載此範例之pdf檔。

這個範例中，包含了一簡單的分析步驟，使用了使用案例圖及敘述、活動圖、狀態圖、類別圖和循序圖。不過，此範例僅止於分析階段，並未涉及設計階段的UML產出，當然也未包含程式碼。

Q3.1 分析步驟

本範例採用 MDA(Model-Driven Architecture)開發程序，做為專業分工的依據，因此系統分析師的工作聚焦於 CIM 與 PIM 階段，至於 PSM 及編碼階段則交由其他的設計師負責之。MDA 主要將產出的 UML 模式，分為下列三個階段：

- CIM(Computation Independent Model) — 聚焦於系統環境及需求，但不涉及系統內部的結構與運作細節。
- PIM(Platform Independent Model) — 聚焦於系統內部細節，但不

涉及實作系統的實體平台(platform)。

- PSM(Platform Specific Model) — 聚焦於系統落實於特定實體平台的細節。例如，Spring、EJB2 或.NET 都是一種實體平台。

因之，系統分析師執行了前述的 CIM 與 PIM 步驟，並且獲得高品質的產出之後，設計師會依據實作平台進一步產出 PSM 階段的設計，並交由程式設計師按圖編碼，編寫出適用於特定實體平台的程式碼。

依據 MDA，本範例所提及的步驟及產出，歸屬於 CIM 與 PIM 階段，並未涉及 PSM 階段。如下：

- CIM-1：定義企業流程，產出企業使用案例模式。
- CIM-2：分析企業流程，產出活動圖。
- CIM-3：定義系統範圍，產出系統使用案例圖。
- PIM-1：分析系統流程，產出系統使用案例敘述。
- PIM-2：分析企業規則，產出狀態圖。
- PIM-3：定義靜態結構，產出類別圖。
- PIM-4：定義操作及方法，產出循序圖。

在 CIM 階段，系統分析師約莫花一~二週的時間，盡快產出初步的系統使用案例，以便讓相關的決策人員可以從中挑選出首期開發的系統使用案例，而這也就是首期的系統範圍。

隨後，專案正式進入 PIM 階段，也是正式進入分析階段，所以系統分析師將投入更多的時間，針對首期的系統使用案例詳述細部規格，做為正式需求文件的一部份，也做為企業人員與開發人員之間的溝通文件。

此外，系統分析師需多加注意，CIM 階段與 PIM 階段的產出方式略有不同。系統分析師在結束 CIM 階段之後，才決定出 PIM 階段的系統範圍，也同時正式進入 PIM 階段。但是，在進入到 PIM 階段之後，系統分析師將所有系統使用案例依相關性分成數組，以組別方式產出該組系統使用案例涉及的 PIM-1~4 產出，隨後交給後續的開發人員進行設計、編碼及測試。然後，逐步產出一組一組的 PIM-1~4 產出，跟 CIM 的產出方式不同。

Q3.2 CIM-1：定義企業流程

定義及分析企業流程(business process)是爲了盡快釐清系統範圍，以便估算開發成本及時程，可不是爲了要改造企業流程，系統分析師千萬別誤解了此步驟之目的。所以，系統分析師在定義及分析企業流程時，記得挑選跟系統有關的企業流程。

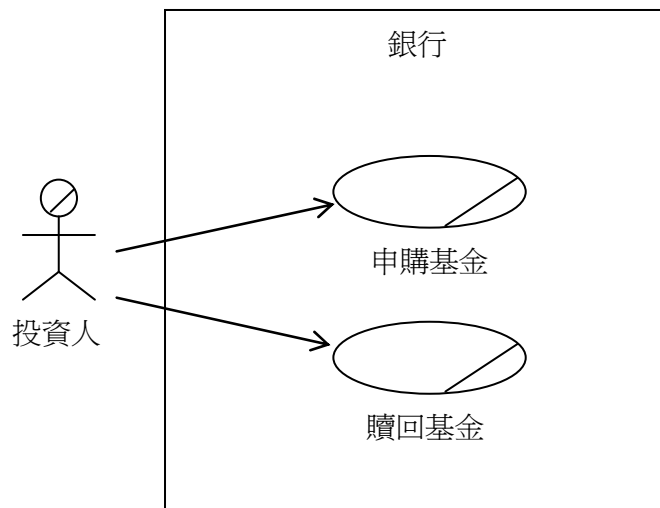


圖 1-32： 企業使用案例圖

CIM-1 定義企業流程的產出，主要有如下的企業使用案例圖和簡述。請看圖 1-32 的企業使用案例圖，圖中的每一個企業使用案例代表一條企業流程，企業參與者則代表位於企業外但會啟動或參與企業流程者。投資人到銀行臨櫃申購基金，啟動了銀行內部的一段關於申購基金的企業流程。再者，投資人也可能臨櫃辦理贖回基金，這又引發了另一條企業流程。

至於企業使用案例簡述，簡潔扼要即可，我們主要用它來記錄和區辨企業流程。

企業使用案例名稱	簡述
1. 申購基金	投資人於銀行營業時間，向銀行單筆或定期定額申購基金。
2. 贖回基金	投資人於銀行營業時間，向銀行贖回基金。

Q3.3 CIM-2：分析企業流程

經由 CIM-1 圈出了系統將參與的企業流程之後，針對每一個企業使用案例，系統分析師得開始分析它的工作流程，並且繪製活動圖(activity diagram)與企業人員取得共識。隨後到了 CIM-3 時，才能夠依此定義出系統可以協助之處，並且規劃出系統範圍。

此處，我們挑選一般的申購基金流程當示範，並繪製出如圖 1-33 所示的活動圖，展示了單筆申購基金的一般交易流程。

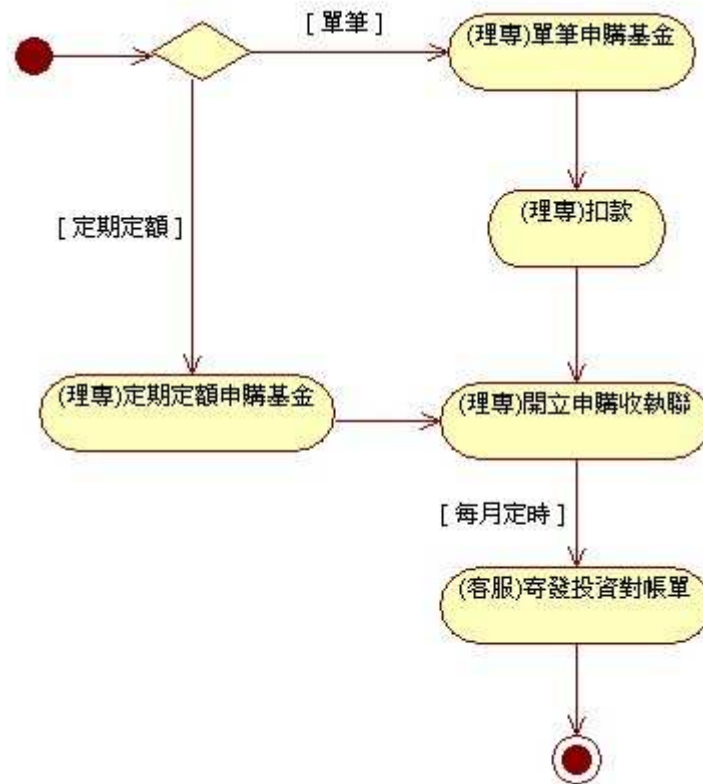


圖 1-33: 申購基金之一般流程的活動圖

Q3.4 CIM-3：定義系統範圍

經過了 CIM-1 的定義企業流程，以及 CIM-2 的分析企業流程之後，終於進入到 CIM-3 這場壓軸戲了。CIM-1 和 CIM-2 的產出文件，跟 CIM-3 的產出文件之間，有如下的關聯性：

- CIM-2 活動圖中的每一個行動，都可能成為 CIM-3 的系統使用案例。

- CIM-1 中的企業參與者，以及 CIM-2 中的行動負責人，都可能成為 CIM-3 的系統參與者(system actor)。

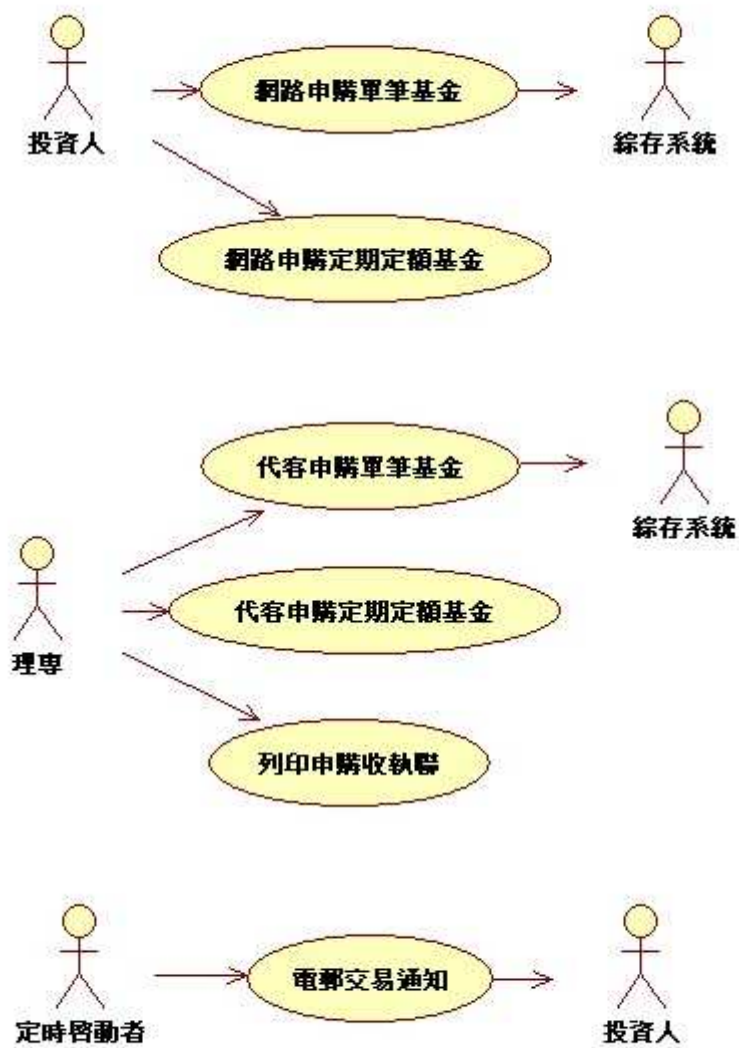


圖 1-34: 分析一般流程所定義出的系統使用案例

針對上述的圖 1-33 一般流程的活動圖，我們分析得出如圖 1-34 的系統使用案例圖，以及下述的使用案例簡述。

系統使用案例名稱	簡述
1. 網路申購單筆基金	投資人上網下單購買某檔基金。
2. 網路申購定期定額基金	投資人上網申購定期定額基金。
3. 代客申購單筆基金	投資人臨櫃申購基金，理專使用系統代客申購單筆基金。
4. 代客申購定期定額基金	投資人臨櫃申購定期定額基金，理專使用系統代客申購定期定額基金。
5. 列印申購收執聯	投資人臨櫃申購基金手續完成之後，理專將列印申購收執聯交給投資人。
6. 電郵交易通知	系統於交易完成之際，自動電郵交易通知給投資人。

Q3.5 PIM-1：分析系統流程

在 CIM 階段，系統分析師約莫花一~二週的時間，盡快產出初步的系統使用案例，以便讓相關的決策人員可以從中挑選出首期開發的系統使用案例，而這也就是首期的系統範圍。

隨後，專案正式進入 PIM 階段，也是正式進入分析階段，所以系統分析師將投入更多的時間，針對首期的系統使用案例詳述細部規格，做為正式需求文件的一部份，也做為企業人員與開發人員之間的溝通文件。


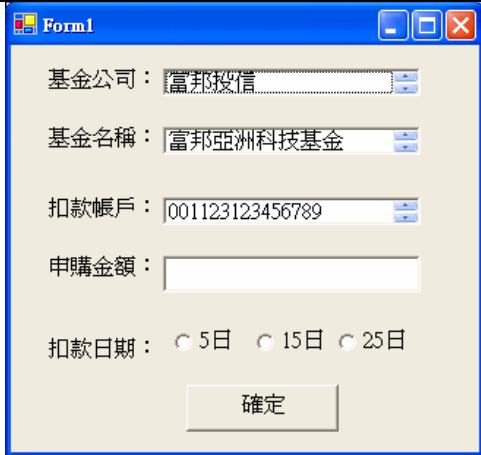
所以，系統分析師在 PIM-1 的主要工作，將針對每一個系統使用案例，分析其內部細節，並編寫詳盡的系統使用案例敘述(use case description)。UML 並未提出標準的敘述格式可供遵守，不過系統分析師可以在網路上找到許多實用的使用案例敘述格式，或者翻閱一些 UML 或使用案例相關書籍，也可以發現許多很有特色的使用案例敘述格式。

此處，我們示範編寫「網路申購單筆基金」和「網路申購定期定額基金」的系統使用案例敘述，如下：

使用案例名稱	網路申購單筆基金
使用案例編號	SUC001
使用案例簡述	投資人上網下單購買某檔基金。
使用案例圖	<pre> graph LR A[投資人] --> B(網路申購單筆基金) B --> C[綜存系統] </pre>
主要流程	<ol style="list-style-type: none"> 1. 系統列出基金公司清單及名下之基金清單，以及約定之扣款帳戶。 2. 投資人從中選定一家基金公司及其名下的某一檔基金，並且挑選某一個約定之扣款帳戶，鍵入申購金額，按下確定鍵。 3. 系統計算出手續費。 4. 系統連線綜存系統，查詢綜存帳戶餘額，確認餘額是否足夠支付交易款項。

	<ol style="list-style-type: none"> 5. 系統出現交易確認訊息，供投資人做最後確認。 6. 投資人按下最後確認鍵。 7. 系統連線綜存系統，扣交易款，交易成立。 8. 系統回傳申購收執聯，並且提供列印功能，供投資人選擇列印與否。
替代流程	<ol style="list-style-type: none"> 2a. [金額不符]系統出現申購額必須為萬元倍數之訊息，回到主要流程 2，供投資人重新輸入申購資料。 2b. [金額過低]系統出現最低申購額之訊息，回到主要流程 2，供投資人重新輸入申購資料。 2c. [金額過高]系統出現最高申購額之訊息，回到主要流程 2，供投資人重新輸入申購資料。 4a. [餘額不足]系統出現餘額不足的訊息，回到主要流程 2，供投資人重新輸入申購資料。
例外流程	<ol style="list-style-type: none"> 7a. [扣款失敗]系統出現交易失敗的訊息，該系統使用案例執行失敗。
企業規則	<ol style="list-style-type: none"> 1. 交易款項 = 申購金額 + 手續費 2. 手續費 = 申購金額 × 基金管理費 × 銀行折扣 3. 國內基金最低申購金額為一萬元，境外基金最低申購金額為三萬元。 4. 每筆交易款項(申購金額+手續費)不得超過 200 萬元。 5. 系統依照公司原有的編碼方式產出交易編號。
非 UML 文檔	基金申購書 pdf 檔、申購收執聯 pdf 檔。

其它	填了假資料的「基金申購書」和「申購收執聯」紙本。
----	--------------------------

使用案例名稱	網路申購定期定額基金
使用案例編號	SUC002
使用案例簡述	投資人上網申購定期定額基金。
使用案例圖	 <pre> graph LR A[投資人] --> B(網路申購定期定額基金) </pre>
參考畫面	
主要流程	<ol style="list-style-type: none"> 系統列出基金公司清單及名下之基金清單、約定之扣款帳戶，以及扣款日期。 投資人從中選定一家基金公司及其名下的某一檔基金，並且挑選某一個約定之扣款帳戶，鍵入申購金額，選擇一扣款日期，並且按下確定鍵。

	<ol style="list-style-type: none"> 3. 系統計算出手續費。 4. 系統出現交易資料，供投資人做最後確認。 5. 投資人按下最後確認鍵。 6. 系統回傳定期定額申購約定書，並且提供列印功能，供投資人選擇列印與否。
替代流程	<ol style="list-style-type: none"> 2a. [金額不符]系統出現申購額必須為千元倍數之訊息，回到主要流程 2，供投資人重新輸入申購資料。 2b. [金額過低]系統出現最低申購額之訊息，回到主要流程 2，供投資人重新輸入申購資料。 2c. [金額過高]系統出現最高申購額之訊息，回到主要流程 2，供投資人重新輸入申購資料。
企業規則	<ol style="list-style-type: none"> 1. 交易款項 = 申購金額 + 手續費 2. 手續費 = 申購金額 × 基金管理費 × 銀行折扣 3. 定期定額國內基金最低申購金額為三千元，定期定額境外基金最低申購金額為五千元。 4. 每筆交易款項(申購金額+手續費)不得超過 200 萬元。 5. 系統依照公司原有的編碼方式產出交易編號。

Q3.6 PIM-2：分析企業規則

企業透過一組規則(business rules)來控制整體的運作，包括人員、流程、系統、概念的運作，皆受制於企業規則。由此足見企業規則之重要，所以早從 PIM-1 的系統使用案例敘述，一直到此處的 PIM-2 狀態圖以及稍

後的 PIM-3 類別圖，我們都會要求系統分析師必需透過這些 UML 圖，記錄且呈現重要的企業規則。

譬如，在經過 PIM-1 的步驟之後，我們認為「定期定額申購」是很重要的企業物件，而且涉及許多重要的企業規則，所以決定為它繪製如圖 1-35 的狀態圖，以便組織企業規則，同時也對定期定額申購有更深入的理解。

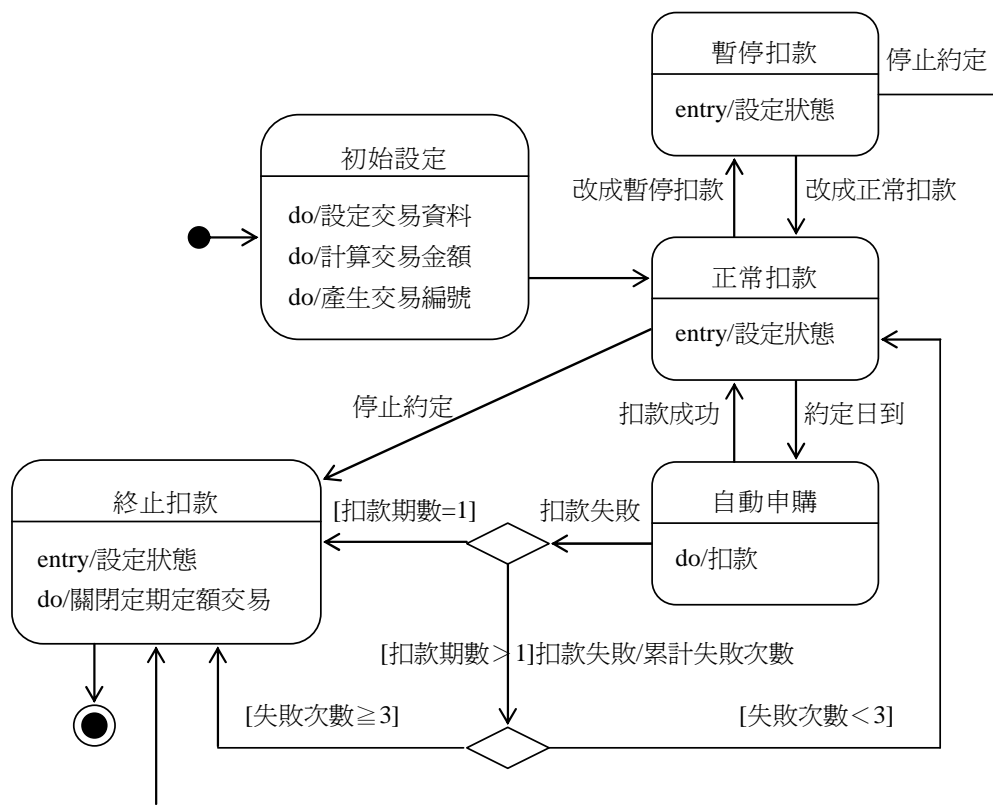


圖 1-35: 定期定額申購物件之狀態圖

Q3.7 PIM-3：定義靜態結構

在 PIM-3 中，系統分析師用類別圖來表達系統內部的靜態結構；系統具備穩定且具彈性的靜態結構，才能夠順應需求變動，迅速支撐多樣化的系統使用案例。之後，類別圖可能經由設計師之手，進行調整，並且成為程式設計師最關切的設計圖之一。程式設計師通常會依照類別圖的內容，來編寫並組織原始程式碼。

在 PIM-3 的過程中，系統分析師尋找操作絕對優先於尋找屬性。因為屬性隨處可見，特別是從 PIM-1 蒐集而來的表單，裡頭多的是物件必須保存的屬性。而尋找操作就沒這麼直接簡單了，系統分析師必需多動腦筋才能定義出操作，所以先別管屬性了，記得優先找操作。

進行 PIM-3 時，系統分析師可以經由下列步驟，建立起如圖 1-36 的類別圖：

1. 套用交易樣式，並且經過調整之後，系統分析師可以獲得初步的靜態結構。
2. 分析 PIM-2 的狀態圖之後，系統分析師可以為類別增添屬性及操作。
3. 分析 PIM-1 蒐集來的表單，系統分析師可以為類別增添更多的屬性。
4. 經過 PIM-4 的循序圖，系統分析師可以為類別增添更多的操作，並且描述操作之方法。

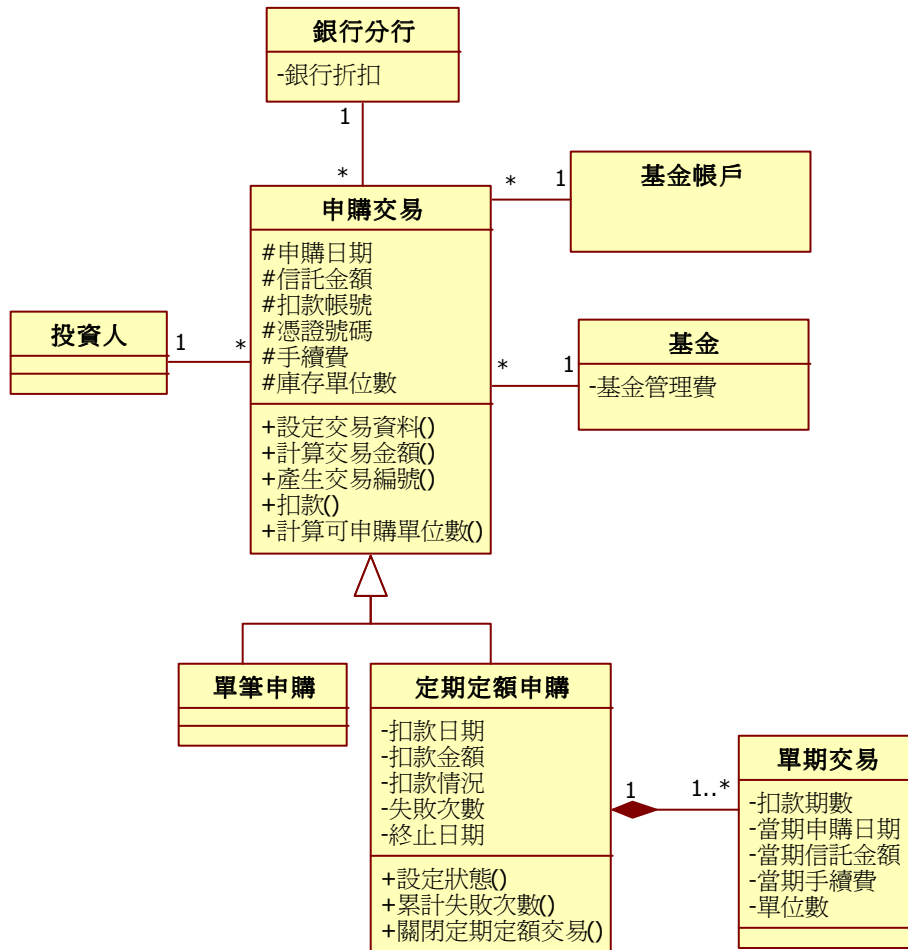


圖 1-36: 類別圖

Q3.8 PIM-4：定義操作及方法

在 PIM-4 中，系統分析師可以用循序圖來表達，系統內部一群物件合力完成某一個系統使用案例時，執行期間的互動情形。之後，循序圖可能

經由設計師之手，進行調整，並且成為程式設計師最關切的設計圖之二（另一張是類別圖）。程式設計師通常會依照循序圖的內容，編寫出方法的原始程式碼雛型。

此外，PIM-1 的系統使用案例敘述和 PIM-3 的類別圖，對 PIM-4 的循序圖，有不可或缺的貢獻。從 PIM-1 的系統使用案例敘述中，系統分析師可以分析出系統流程。而在 PIM-3 的類別圖中，系統分析師定義出系統內部的靜態結構。隨後，到了 PIM-4 的循序圖時，則結合了系統使用案例以及靜態結構兩者。

系統分析師經由循序圖的思考與表達，試圖安排依據類別們所產出的一群物件之間的互動，讓這一群物件可以合力完成某一個系統使用案例。同時，在循序圖中，一群物件互動所引發的操作，則可以回饋給類別圖，定義出更多的操作及屬性，甚至發現之前未發現的其它類別及關係。

系統分析師可參考下述步驟來繪製循序圖：

1. 扮演啟動者的參與者物件放置於循序圖最左方；扮演支援者的參與者物件放至於循序圖的最右方。
2. 針對系統使用案例敘述裡所記載每項流程步驟，判斷執行時需要使用到哪些資料，且可指派擁有該資料的物件負責該項工作。
3. 試著執行循序圖，以便調整流程，並且為操作加上參數。
4. 把繪製循序圖時所找到的操作及屬性，回饋給類別圖。

以「網路申購單筆基金」系統使用案例之主要流程為例，我們示範繪製出如圖 1-37 所示的循序圖。

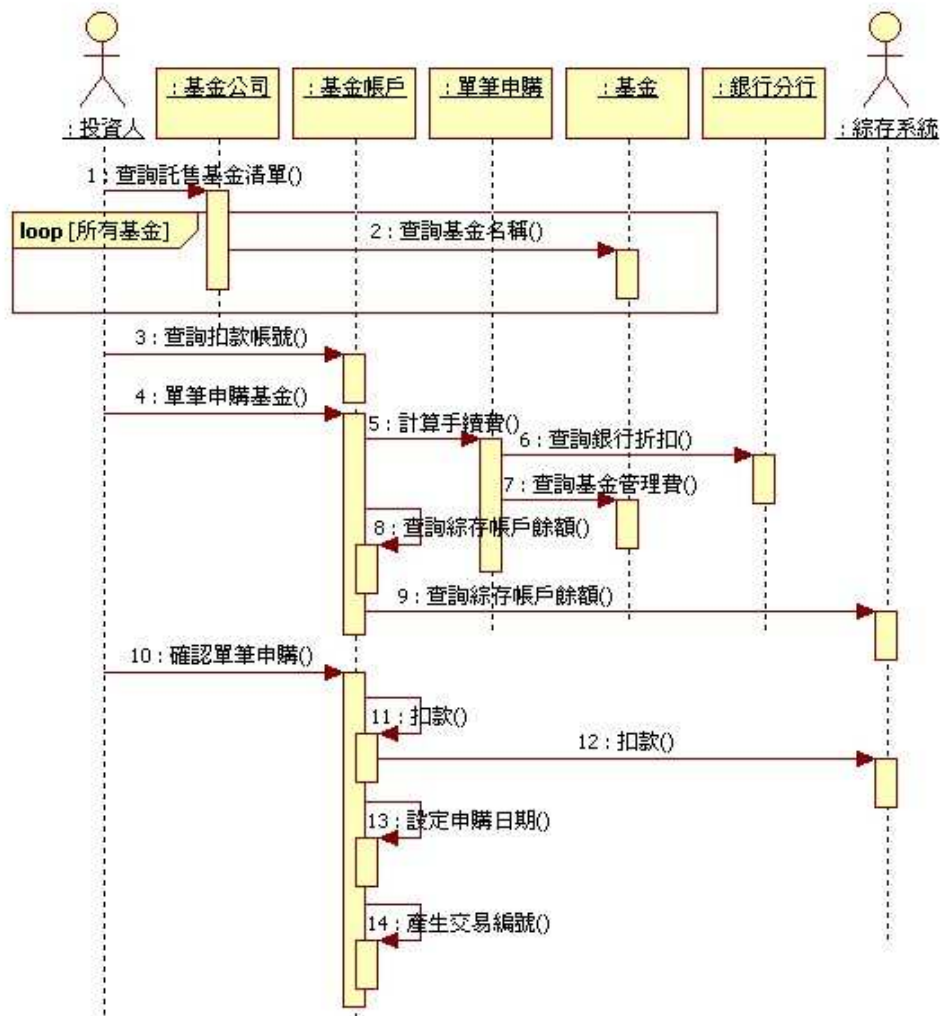


圖 1-37: 循序圖

最後，系統分析師可以試著執行一次循序圖的流程，並且為操作加上參數。增添輸入(in)及輸出(out)參數如下：

1. 查詢託售基金清單(out 基金名稱清單)
2. 查詢基金名稱(out 基金名稱, 基金代號)
3. 查詢扣款帳號(out 扣款帳號)
4. 單筆申購基金(in 基金代號, 申購金額)
5. 計算手續費(in 申購金額, out 手續費)
6. 查詢銀行折扣(out 銀行折扣)
7. 查詢基金管理費(out 基金管理費)
8. 查詢綜存帳戶餘額(out 綜存帳戶餘額)
9. 查詢綜存帳戶餘額(in 扣款帳號, out 綜存帳戶餘額)
10. 確認單筆申購(out 憑證號碼)
11. 扣款()
12. 扣款(in 交易金額)
13. 設定申購日期()
14. 產生交易編號(out 憑證號碼)

由於，單筆申購和定期定額申購計算手續費的方法相同，所以系統分析師可以將單筆申購類別裡的「計算手續費」操作移至申購交易類別，並匯總上述循序圖所新增的操作與相關屬性，更新類別圖如 2-38 所示。

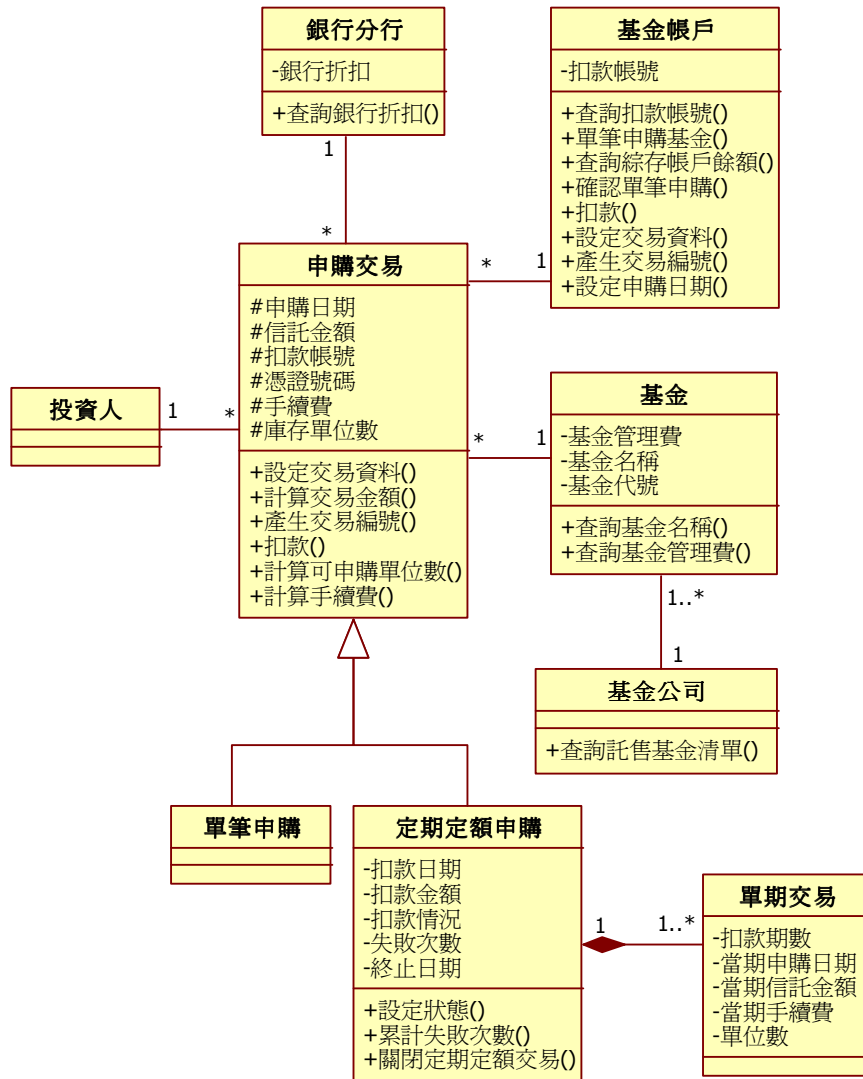


圖 1-38: 更新之後的類別圖

Q4 如何抓出系統的類別？

如何抓出系統的類別？

(znnoopp於程式設計俱樂部之提問)

請問各位先進，是以何種方法如何抓出系統的類別？從use case diagram到class diagram這邊總是銜接的很吃力。除了一般名詞、動詞...方法，和說不出理由的「經驗值」外，不知道怎麼去訂定出一個依據。從設計樣式來反推，感覺上又有點怪怪的。

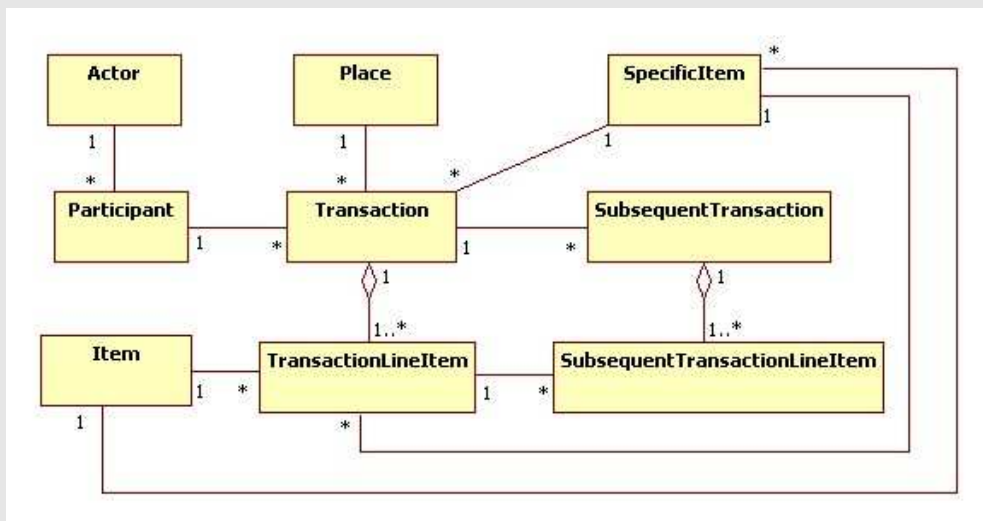
簡答：

軟體專家Sally Shlaer和Stephen J. Mellor提出下列五種常見的物件種類，供您參考：

1. 實體物件(physical object)—這是決不會錯過的物件種類，幾乎眼前所見的一切，無論是自然物或人造物，都屬於實體物件。
2. 角色物件(role object)—人類或組織所扮演的角色。
3. 事件物件(incident object)—特定條件或時刻下發生的事件，可以對應成軟體物件。
4. 互動物件(interaction object)—互動物件通常含有交易或契約的特質，並且涉及到兩個以上的物件。
5. 規格物件(specification object)—庫存系統或製造業的應用系統比較常出現規格物件的實例。

再者，交易樣式(transaction patterns)也是很實用的切入點，可以

用來找到跟交易有關的物件。在多數的企業領域中，「交易」(transaction)通常是一項很重要的企業概念，而且一旦交易發生，也經常需要保存相關的交易資料。以此為核心概念，Peter Coad在“Object Models: Strategies, Pattern, and Applications”一書中，提出一個以交易為主的靜態結構，稱之為「交易樣式」，如下圖所示。



Q4.1 五種常見的物件種類

感官能觸及的實體物件(physical object)，最常對應成軟體類別的物件種類。對於初學者而言，根本不需要額外具備什麼樣特殊的知識，就能夠輕易將日常生活裡大大小小的物品對應成軟體類別。實體物件的例子不勝枚舉，眼前所視見的一切有形有體的物品，無論是有生命、無生命、自然物、人造物、可食的、不可食的等等都是實體物件。

不過，除了隨處可見的實體物件之外，真實世界裡其實還充斥著大量

非實體物件(nonphysical objects)。我們在定義軟體類別時，容易忽略重要的非實體物件，通常需要透過學習及提醒，才懂得將它們對應成軟體類別。比方說，我們可以歸類信用卡、簽帳單為可視見的實體物件，以及歸類刷卡消費和預借現金為非實體物件。

軟體專家 Sally Shlaer 和 Stephen J. Mellor 提出下列五種常見的物件種類，供您參考：

- 實體物件(physical object)—這是決不會錯過的物件種類，幾乎眼前所見的一切，無論是自然物或人造物，都屬於實體物件。比方說，人、樹木、貓狗、建築物、檯燈、信用卡、螢幕等等，舉例不完。光從圖 1-39 裡，我們就可以找到好多種實體物件了。

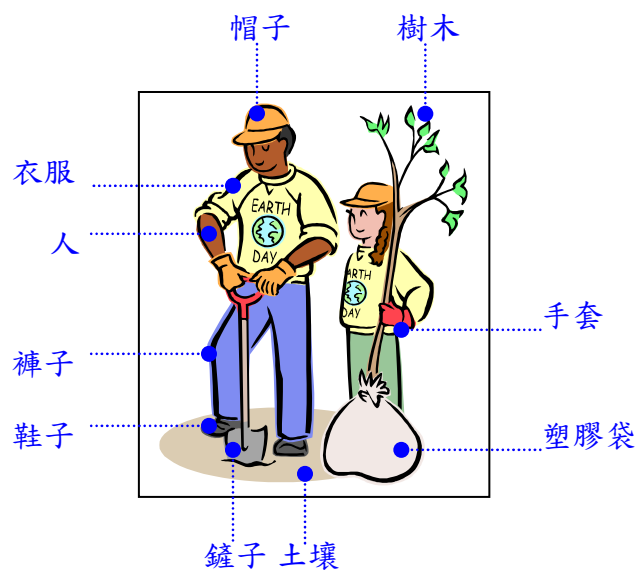


圖 1-39: 實體物件

- 角色物件(role object)—人類或組織所扮演的角色。一般而言，如果我們會定義一種角色物件，就會有第二種、第三種角色物件。這是因為我們會用各式不同的角色來對一大群的人分類，特別是用工作職稱做為角色名稱。比方說，在醫院領域裡，我們慣用醫生、護士、藥劑師、看護和病人等等的職稱來分辨人。甚至針對醫生，我們還會細分成婦產科醫生、外科醫生等等。



圖 1-40: 角色物件

- 事件物件(incident object)—特定條件或時刻下發生的事件，可以對應成軟體物件。比方說，初次啓用信用卡時，我們可能會透過語音系統，進行開卡手續，這是特定條件下會發生的事件。另外一個特定時刻下會發生的事件，像是飛機通常會有固定的航班，其中的飛機屬於實體物件，而航班屬於事件物件，如圖 1-41 所示。

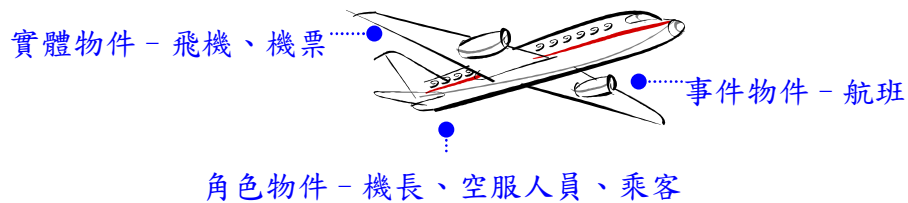


圖 1-41: 事件物件

- 互動物件(interaction object)—互動物件通常含有交易(transaction)或契約(contract)的特質，並且涉及到兩個以上的物件。交易是最常見且重要的互動物件，要是可以先定義出交易物件，通常可以順利找出一連串相關的物件。比方說，以一般的消費交易為例，我們可以以消費交易為核心，找出相關的人、地、物，如圖 1-42 所示。人是指參與交易的相關角色，如顧客、店員。地是指發生交易的地點，如商家或是進行結帳的收銀機。至於，物則是指交易物品或是相關物品，如商品或信用卡、發票、簽帳單。

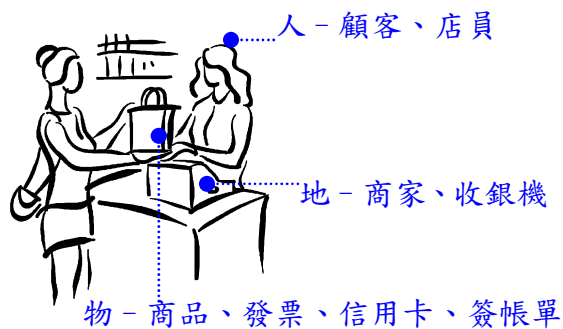


圖 1-42: 互動物件

- 規格物件(specification object)－庫存系統或製造業的應用系統比較常出現規格物件的實例。而且，如果有機型、規格的概念，通常也會有針對該規格產出的實體物件。比方說，我的印表機機型是 HP LJ1000，其中印表機是實體物件，而機型則是規格物件，如圖 1-43 所示。

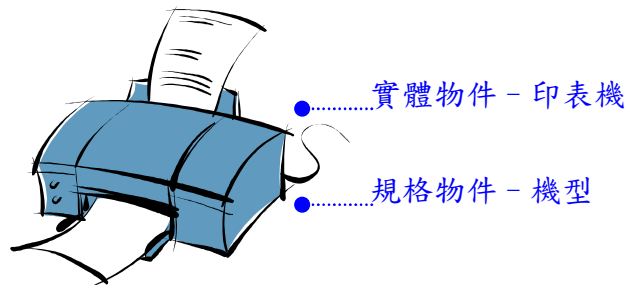


圖 1-43: 規格物件

Q4.2 善用交易樣式

在多數的企業領域中，「交易」(transaction)通常是一項很重要的企業概念，而且一旦交易發生，也經常需要保存相關的交易資料。以此為核心概念，Peter Coad 在“Object Models: Strategies, Pattern, and Applications”一書中，提出一個以交易為主的靜態結構，稱之為「交易樣式」(transaction patterns)，如圖 1-44 所示。

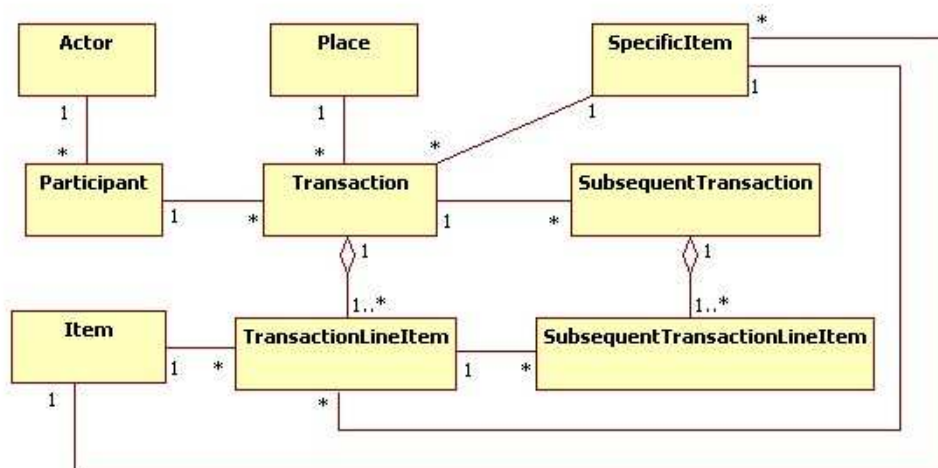


圖 1-44: 交易樣式

善用交易樣式將有助於建構以交易為主的類別圖。以基金系統為例，套用交易樣式之後，可以獲得如圖 1-45 的類別圖。對應的細節如下：

- **Participant-Transaction** 投資人-定期定額申購。
- **Place-Transaction** 銀行分行-定期定額申購。
- **SpecificItem-Transaction** 基金帳戶-定期定額申購。
- **Transaction-TransactionLineItem** 定期定額申購-單期交易。
- **Item-TransactionLineItem** 基金-單期交易。

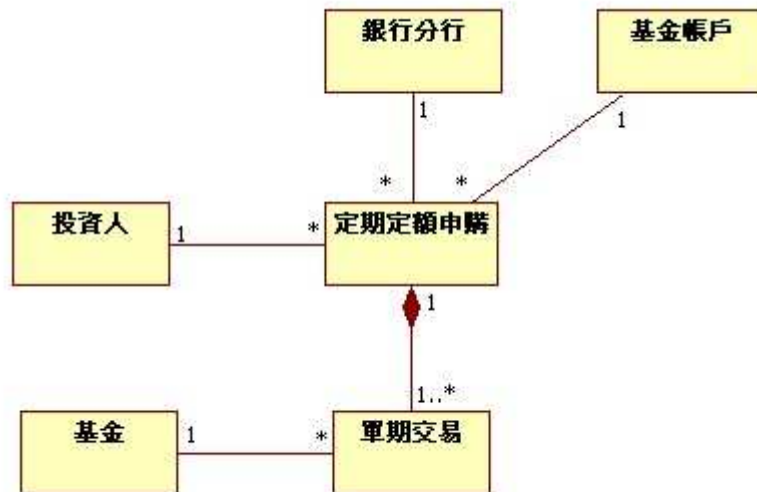


圖 1-45: 套用交易樣式

交易樣式通常不會適用於所有的情況，所以我們套用樣式之後，可能會需要調整。延續前述的基金系統，可能調整成圖 1-46，細節如下：

- 同一個定期定額申購底下，所有單期交易都是申購相同的基金，所以將原先的「基金-單期交易」改成「基金-定期定額申購」。
- 建立申購交易與單筆申購、定期定額申購之間的一般性關係。
- 投資人、銀行分行和基金帳戶與定期定額申購之間的結合關係，全部改成連接申購交易。

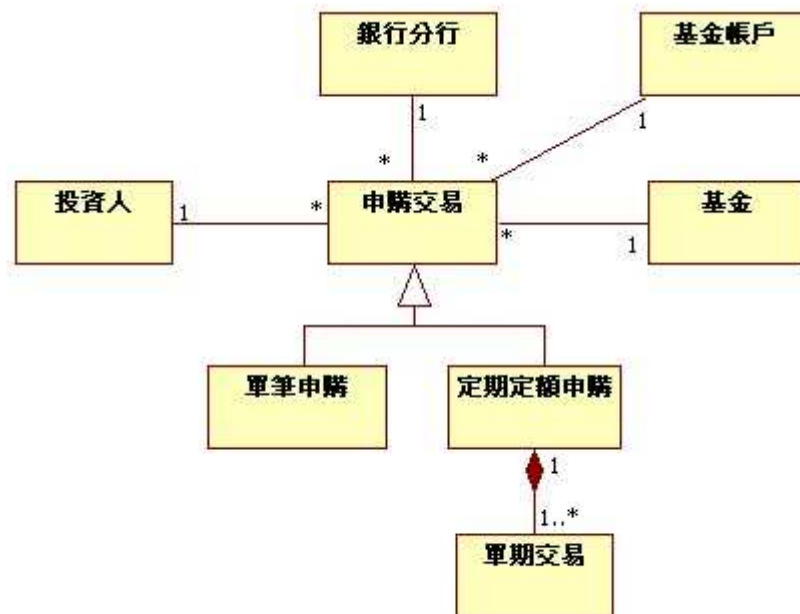


圖 1-46: 調整之後的類別圖

Q5 畫 UML 圖的先後順序？

UML的先後順序？

(chen於JavaWorld@TW之提問)

小弟目前仍是在學的學生。因課程的關係，僅初步的了解軟體工程。由於小弟目前所看到的UML相關文章或文件，大都僅是對於UML裡的如：use case diagram、class diagram、sequence diagram等作了詳細的介紹，不過，似乎沒有提到它們的先後順序，因此想請

教各位先進的是—

等use case diagram畫完後，接下來要畫什麼哪個diagram呢？說實話，小弟僅只了解class diagram為何，但仍不清楚class diagram如何實作？

簡答：

實務上，使用案例圖及敘述、類別圖與循序圖三者之搭配，幾乎是UML專案的基本型，所以在分工或外包的設計文檔中，通常少不了這三款UML圖。常見的開發程序是，並行建構使用案例圖文與類別圖，接著才建構循序圖以及按圖編碼，如下圖所示。

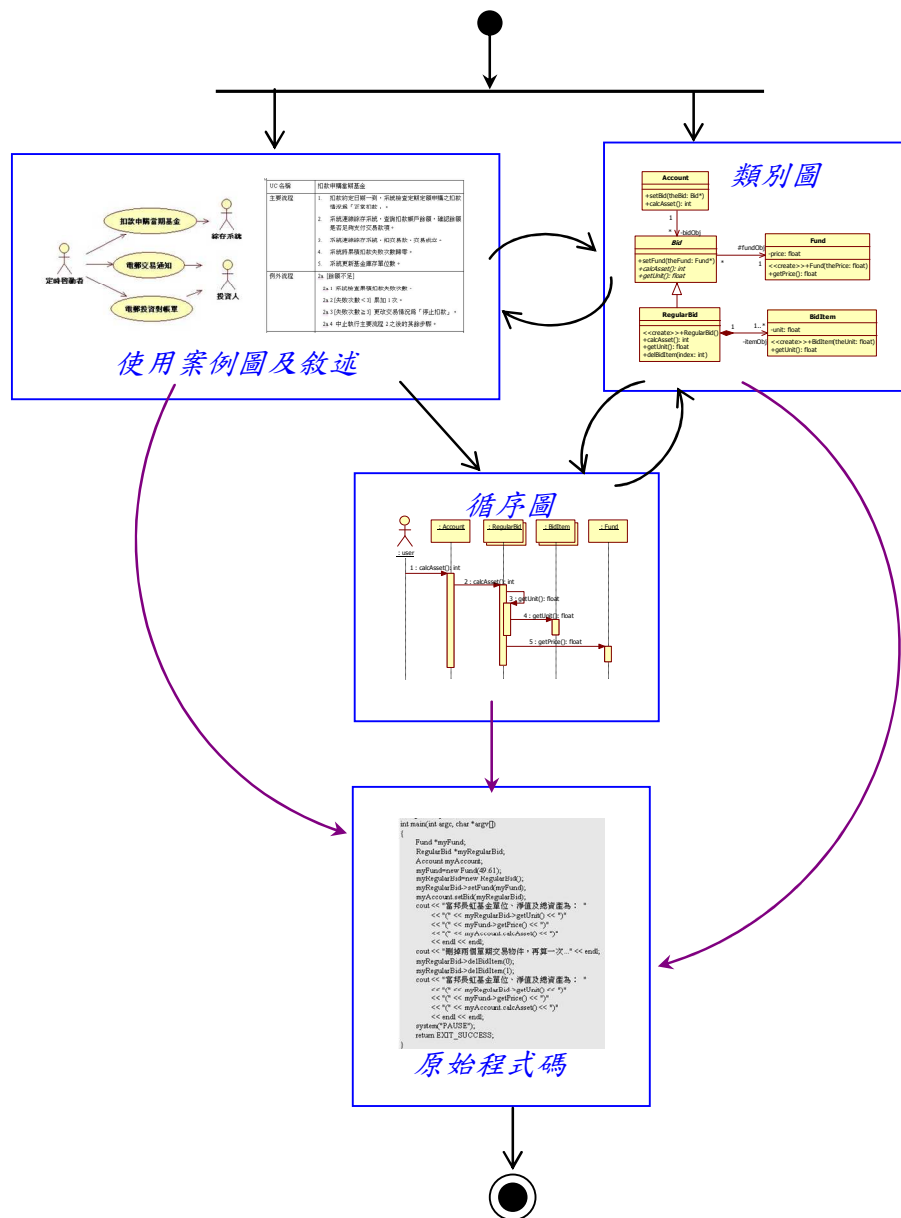


圖 1-47: 簡易的開發程序

一個系統只有一個內部結構，而且系統對外提供的所有的服務，都僅依賴這個穩定的內部結構所支撐。然而，每一項服務的運作方式皆不同，所以雖然系統僅有一個靜態結構，可以卻有很多個動態行為。

因此，透過 UML 圖來呈現系統的狀況時，一個系統僅有一張呈現系統內部結構的類別圖，而且無論使用案例圖中有多少個使用案例。但是，每一個使用案例至少對應一張循序圖，呈現出系統執行使用案例期間，其內部的一群物件互動的運作情況。

再者，類別圖通常不是一次就能夠設計完全，而是透過一個又一個的使用案例，以及一張又一張的循序圖，三者經過多次循環更新的歷程後，類別圖才逐步成形且穩定下來。

最後，我們以基金系統為範例，帶您看到三款圖之間的搭配。假想，我們是 C++ 程式設計師，參與這個 UML 專案，接到了如圖 48~50 之使用案例圖文、類別圖和循序圖，可能按圖施工寫出如圖 1-51~57 的 C++ 程式碼。

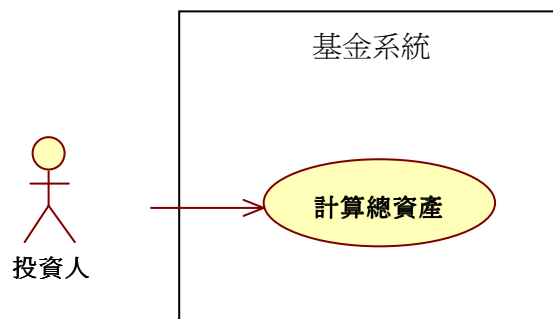



圖 1-48： 使用案例圖

使用案例名稱	計算總資產
使用案例簡述	投資人以基金目前淨值估算總資產。

參考畫面	 A screenshot of a Windows command prompt window. The title bar reads 'C:\重要文件\備份06-UML答客問\第一輯\EX01_...'. The command prompt shows the following text: '日盛上選基金單位及淨值： <2325.02><37.83>', '總資產為： 151046', and '請按任意鍵繼續 . . . '.
主要流程	<ol style="list-style-type: none"> 1. 螢幕上秀出基金單位數及淨值。 2. 計算出總資產。 3. 螢幕上秀出總資產。
企業規則	<ol style="list-style-type: none"> 1. 資產 = 基金淨值 × 單位數
參考文件	<ol style="list-style-type: none"> 1. 基金系統之使用案例圖，參考圖 1-48。 2. 基金系統之類別圖，參考圖 1-49。 3. 計算總資產之循序圖，參考圖 1-50。

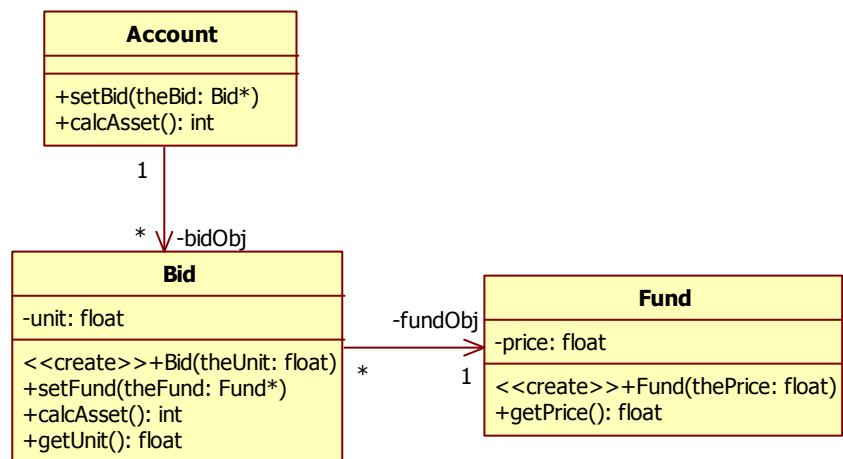


圖 1-49: 類別圖

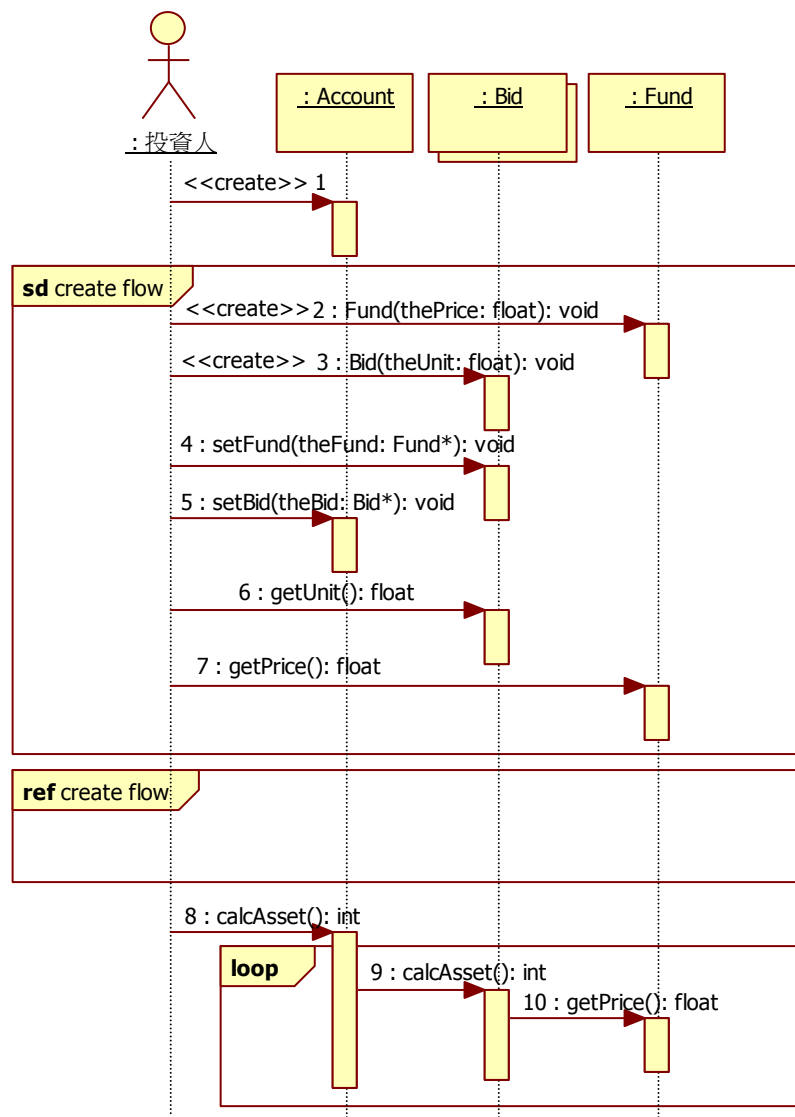


圖 1-50: 計算總資產使用案例之循序圖

```
//// EX01_03
1.  // calcAssetUC.cpp
2.  //

3.  #include <cstdlib>
4.  #include <iostream>
5.  #include "Bid.h"
6.  #include "Account.h"
7.  #include "Fund.h"
8.  using namespace std;

9.  int main(int argc, char *argv[])
10. {
11.     Fund *myFund;
12.     Account myAccount;
13.     Bid *myBid;

14.     myFund=new Fund(19.84);
15.     myBid=new Bid(3180.02);
16.     myBid->setFund(myFund);
17.     myAccount.setBid(myBid);
18.     cout << "大華大華基金單位及淨值： "
19.         << "(" << myBid->getUnit() << ")"
20.         << "(" << myFund->getPrice() << ")" << endl;

21.     myFund=new Fund(37.83);
22.     myBid=new Bid(2325.02);
23.     myBid->setFund(myFund);
24.     myAccount.setBid(myBid);
25.     cout << "日盛上選基金單位及淨值： "
26.         << "(" << myBid->getUnit() << ")"
27.         << "(" << myFund->getPrice() << ")" << endl;

28.     cout << "總資產為： "
29.         << myAccount.calcAsset() << endl << endl;

30.     system("PAUSE");
31.     return EXIT_SUCCESS;
32. }
//// EX01_03
```

圖 1-51: calcAssetUC.cpp


```
//// EX01_03
1. // Account.h
2. //

3. #pragma once
4. #include <cstdlib>
5. #include <vector>
6. #include "Bid.h"
7. using namespace std;

8. class Account
9. {
10. public:
11.     void setBid(Bid*);
12.     int calcAsset();

13. private:
14.     vector<Bid*> bidObj;
15. };
//// EX01_03
```

圖 1-52: Account.h

```
//// EX01_03
1. // Account.cpp
2. //

3. #include "Account.h"

4. void Account::setBid(Bid *theBid)
5. {
6.     bidObj.push_back(theBid);
7. }

8. int Account::calcAsset()
9. {
10.     int size,theAsset=0;
11.     size=bidObj.size();
12.     for(int i=0;i<size;i++)
13.         theAsset=theAsset+bidObj[i]->calcAsset();
14.     return theAsset;
15. }
```

```
//// EX01_03
```

圖 1-53: Account.cpp

```
//// EX01_03
1.  // Bid.h
2.  //

3.  #pragma once
4.  #include "Fund.h"

5.  class Bid
6.  {
7.  public:
8.      Bid(float);
9.      void setFund(Fund*);
10.     int calcAsset();
11.     float getUnit();

12. private:
13.     float unit;
14.     Fund *fundObj;
15. };
//// EX01_03
```

圖 1-54: Bid.h

```
//// EX01_03
1.  // Bid.cpp
2.  //

3.  #include "Bid.h"

4.  Bid::Bid(float theUnit)
5.  {
6.      unit=theUnit;
7.  }

8.  void Bid::setFund(Fund *theFund)
9.  {
10.     fundObj=theFund;
```

```
11. }  
12. int Bid::calcAsset()  
13. {  
14.     return unit*fundObj->getPrice();  
15. }  
  
16. float Bid::getUnit()  
17. {  
18.     return unit;  
19. }  
//// EX01_03
```

圖 1-55: Bid.cpp

```
//// EX01_03  
1. // Fund.h  
2. //  
  
3. #pragma once  
  
4. class Fund  
5. {  
6. public:  
7.     Fund(float);  
8.     float getPrice();  
  
9. private:  
10.    float price;  
11. };  
//// EX01_03
```

圖 1-56: Fund.h

```
//// EX01_03  
1. // Fund.cpp  
2. //  
  
3. #include "Fund.h"  
  
4. Fund::Fund(float thePrice)
```

```
5.  {  
6.      price=thePrice;  
7.  }  
  
8.  float Fund::getPrice()  
9.  {  
10.      return price;  
11. }  
//// EX01_03
```

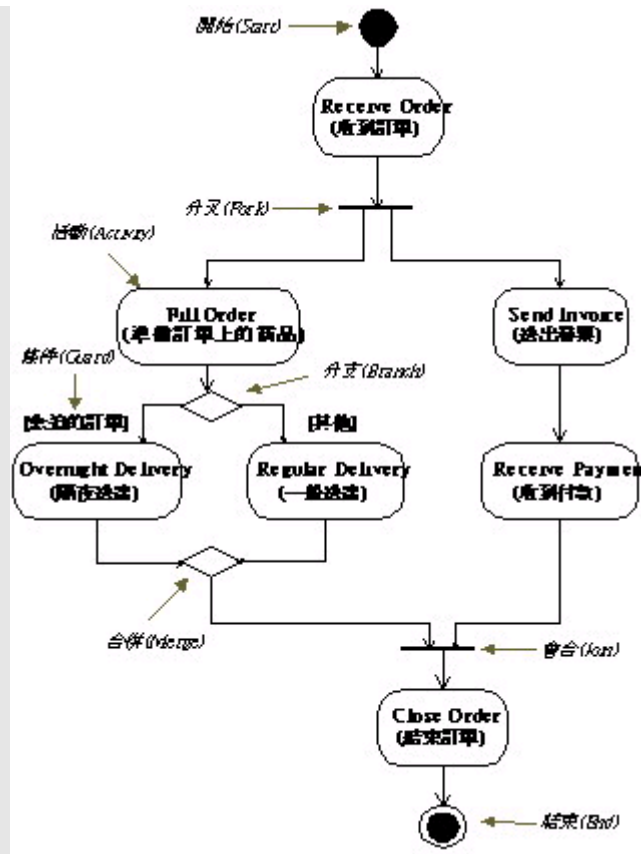
圖 1-57: Fund.cpp

Q6 請教有關活動圖的問題？

請教有關Activity Diagram的問題？

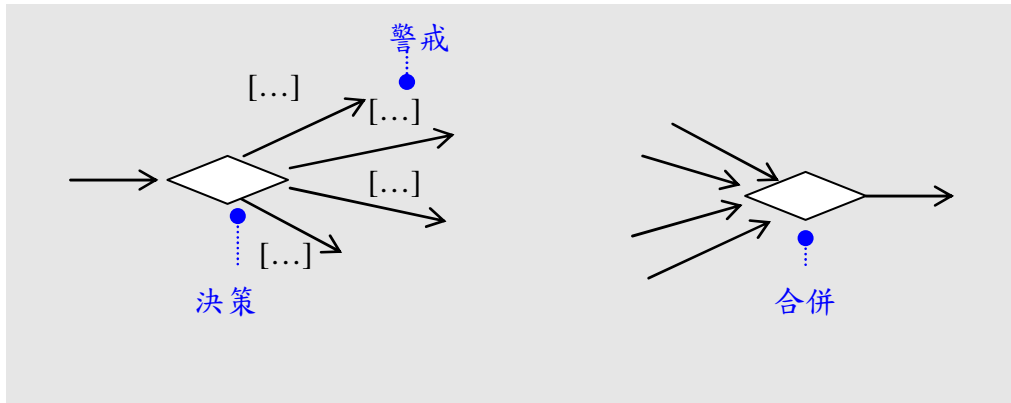
(koi於JavaWorld@TW之提問)

以下有一張圖是我在學習的時候看到的，我不太了解它為什麼可以利用「菱形」的區塊去當作「分支」與「合併」用，「菱形」不是都是拿來當decision的嗎？



簡答：

決策(decision)和合併(merge)的圖示相同，都是空心大菱形，如下圖所示。在活動圖面上，決策菱形只會有一條射入線，但是有兩條以上的射出線，且通常會配上警戒條件(guard)。至於，合併菱形則會有多條射入線，且不需要配合警戒條件，但只有一條離開的射出線。



Q6.1 決策或合併

決策是一種多擇一的情況，多條射出線配合警戒條件，最後只能有一條流程線通過警戒條件，進入並執行下一個行動。一般而言，警戒條件不會獨立存在，它通常用來配合限制流程線的射出。如果，一個行動射出多條流程線，也需要配合設置警戒條件。

請看圖 1-58 的活動圖片段，結帳之後進入一個決策點，判斷是否要託運貨品。如果，決定要自行運送，整個活動圖就結束了。否則，就進入並執行託運行動。此決策有兩條射出線，但只有一條流程線會通過警戒條件的限制，射出轉換到下一個行動。

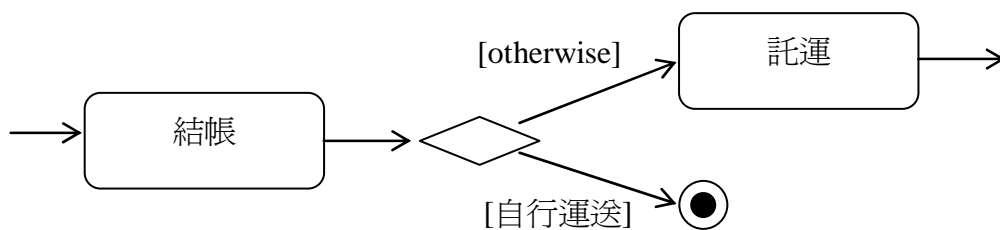


圖 1-58: 決策

至於，合併菱形則會有多條射入線，且不需要配合警戒條件，但只有一條離開的射出線。所以，無論是任何一條或多條射入線進入合併點之後，將合併成一條射出線離開合併點，並進入執行下一個行動。

請看圖 1-59 的活動圖，結帳或換貨行動之後會進入同一個合併點，意味著兩者任一個行動執行結束之後，都會進入運送行動。所以，如果鑑賞之後發現貨品有瑕疵時，會進入換貨行動，隨後會再一次執行運送行動，總共執行了兩次運送行動。

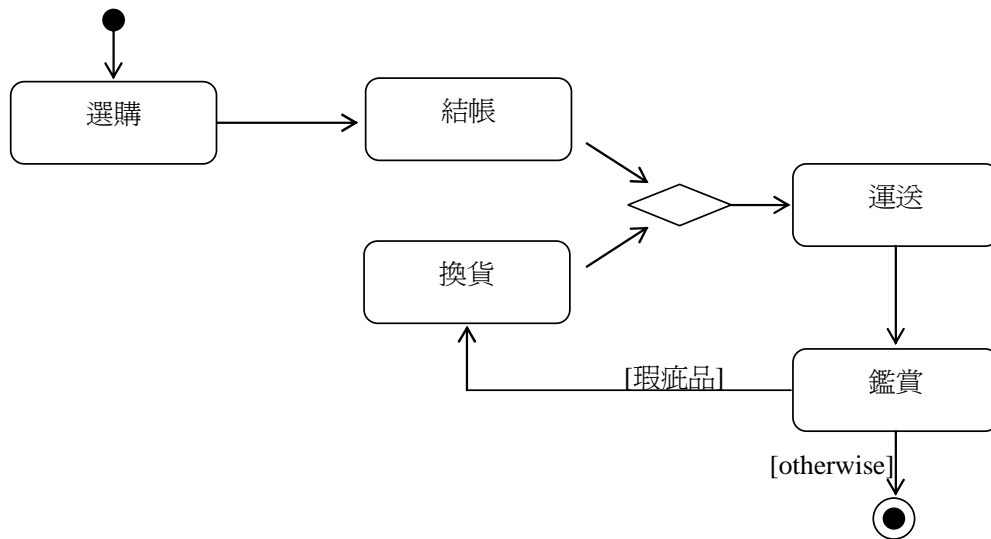


圖 1-59: 遇到瑕疵品將執行兩次運送

簡言之，正常的流程是活動起點、選購、結帳、運送、鑑賞、活動終點，如圖 1-60 所示。

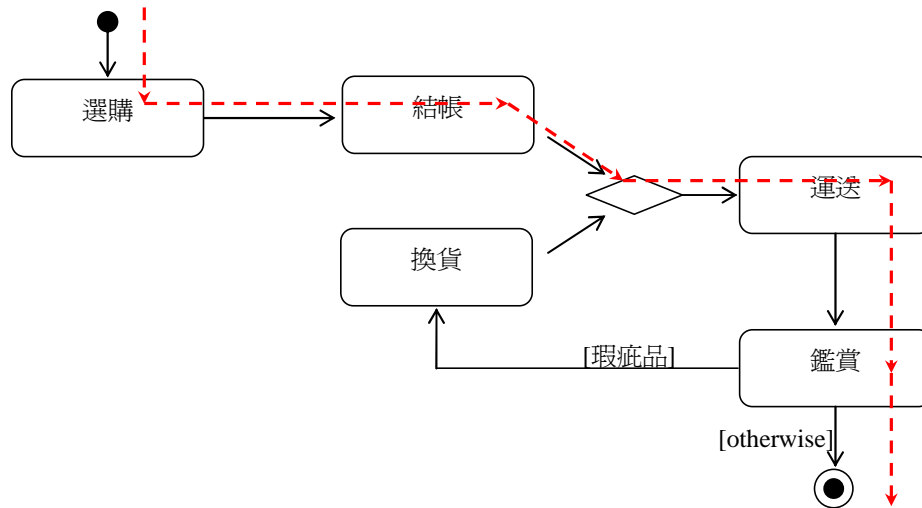


圖 1-60: 正常流程

而有瑕疵品的流程是活動起點、選購、結帳、運送、鑑賞、換貨、運送、鑑賞、活動終點，如圖 1-61 所示。

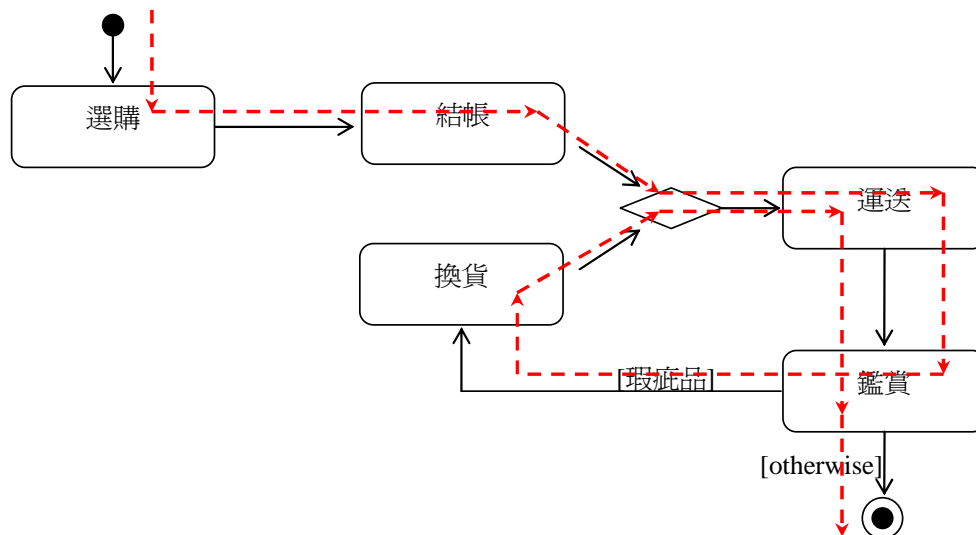


圖 1-61: 有瑕疵品的流程

Q6.2 避免無謂的決策

如果，行動本身有足夠的資訊可以進行決策，則可以不需要設置決策點，僅需要配合警戒條件來不會獨立存在，它通常用來配合限制流程線的射出。請看圖 1-62 的活動圖片段，因為在確認選課資料行動中，已經有該課程是否為付費或免費課程的資訊了，因此已經可以決定下一項該執行的行動了。

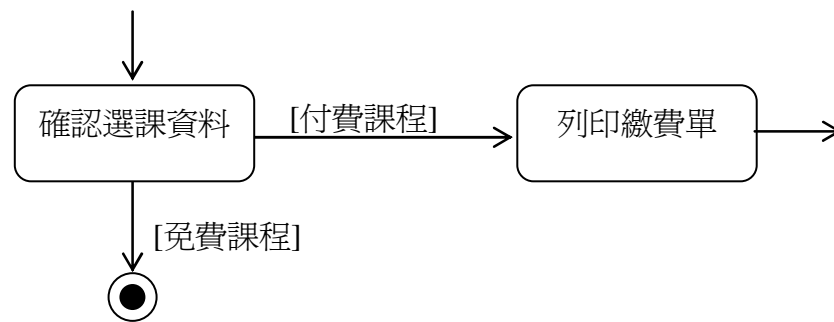


圖 1-62: 行動隱含決策

既然，行動可以隱含著決策，相較之下，我們使用決策菱形，通常是為了突顯當下需要外界給予可作決策的資訊。再看另一個例子，結帳之際我們恐怕還無法有足夠的資訊決定是否託運貨品，直到發現貨品太大、太重或太多時，這時才具足了決策資訊，因此可以決定採用託運或自行運送，如圖 1-63 所示。

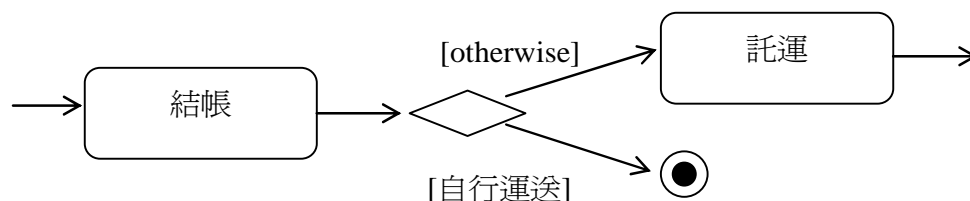


圖 1-63: 需要外界輸入資訊進行決策

或者，將圖 1-63 的設計改成圖 1-64 也行，因為到了取貨之際，知道貨品不容易自行運送，所以決定採用託運方式。

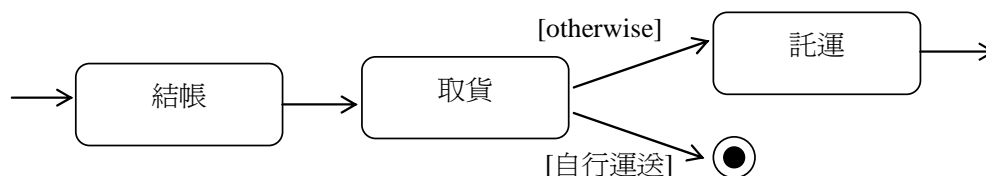


圖 1-64: 取貨時有足夠資訊決定運送方式

Q6.3 條件不重疊、不遺漏

從某一個決策或行動射出多條流程線的情況下，一定會配合警戒條件來決定一條射出線，進入下一個行動。所以，設置警戒條件時，一定要做到條件不重疊、不遺漏。如果，警戒條件有重疊的情況，將會發生同時發出多條射出線的錯誤，需多加注意。

另外，倘若警戒條件有遺漏的話，也會發生無射出線的錯誤。此時，可以善用[otherwise]的警戒條件，讓沒有明確列出的其它情況可以進入 otherwise，避免產生無法執行到下一個行動的困境。

Q7 請問會員系統的 use case ？

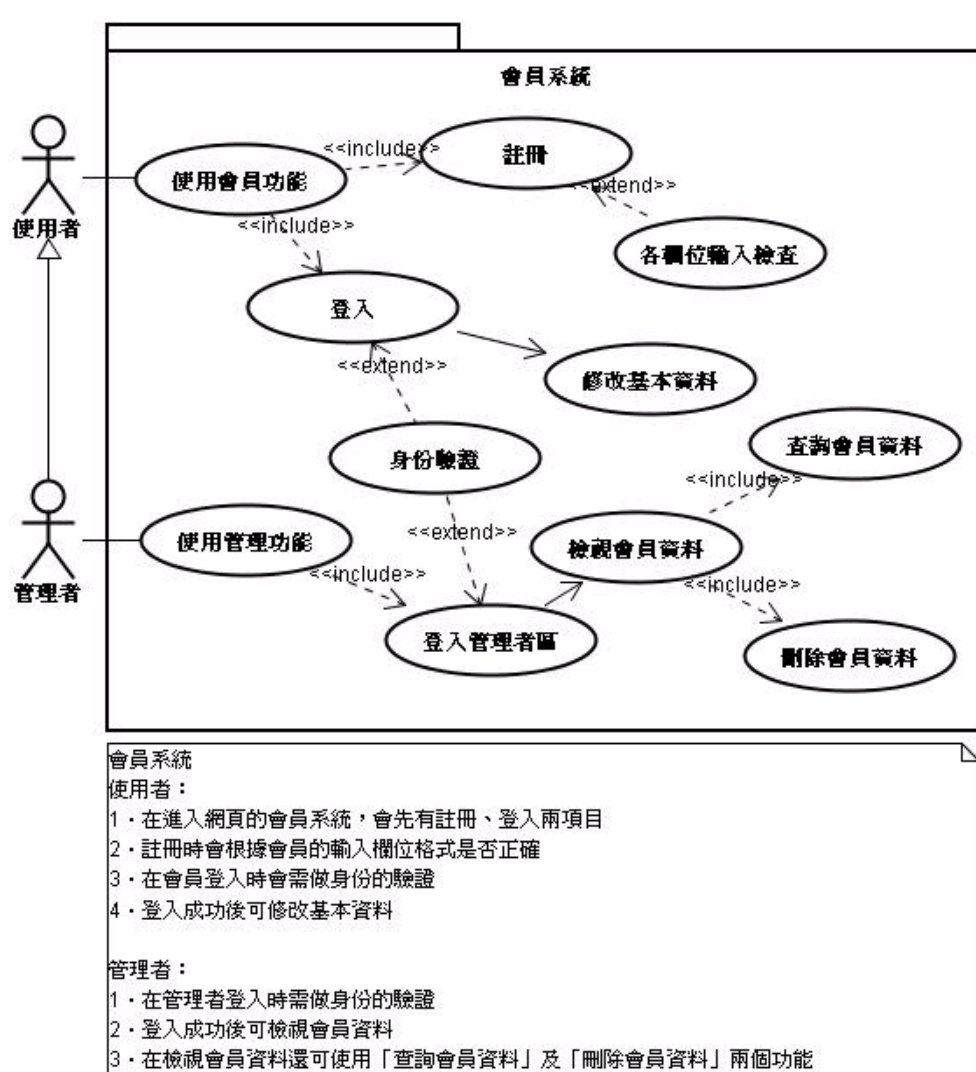
請問會員系統的 use case ？

(umpdsa於JavaWorld@TW之提問)

不好意思，請問這樣的 use case 有什麼錯誤的地方？

第一次這樣畫 use case，請各位大大覺得有問題就重重的批評一番。不然一直沒有機會學到這方面的知識。有看過書，但一直沒機會畫出來，所以從來不知道自己錯在哪？

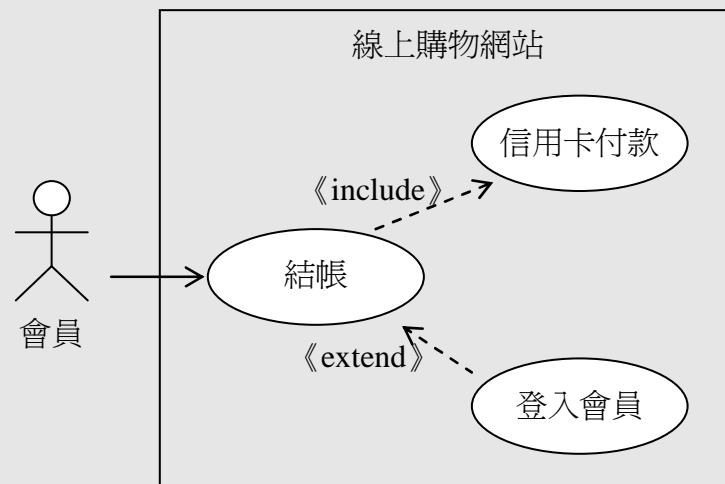
可以的話請各位大大指點，尤其是 use case 跟 use case 間的關連線的地方，常常會用錯！



簡答：

使用案例之間有兩種主要的關係，包含關係(include)與擴充關係(extend)。透過包含關係，用例可以將其它使用案例內部的流程包含進來成為自己的流程。請看下圖的例子，結帳用例包含信用卡付款

用例，意味著在結帳流程裡，將包含一段信用卡付款流程。在包含關係中，結帳使用案例稱為基礎用例(base use case)，信用卡付款使用案例則稱為包含用例(inclusion use case)。



有些時候，在發生特定條件時，才會額外執行某一段流程。這種情況，不能使用包含關係，而是要用擴充關係。因此，登入會員用例擴充結帳用例，意味著在執行結帳流程期間，如果發生會員未登入的情況時，可以透過擴充關係額外啟動登入會員的流程。在擴充關係中，結帳使用案例稱為基礎用例(base use case)，登入會員使用案例則稱為擴充用例(extension use case)。

簡言之，基礎用例執行期間，一定會連帶執行包含用例，但卻不一定會執行擴充用例。所以在上述的例子裡，執行結帳流程期間，一定會執行信用卡付款流程，但是卻在發現該會員還未進行登入手續時，才會開始執行登入會員的流程。

Q7.1 包含關係

包含關係(include)用於兩使用案例之間，它的圖示是標示《include》字眼的帶箭頭虛線，由基礎用例(base use case)指向包含用例(inclusion use case)，意謂著基礎用例含有包含用例，如圖 1-65 所示。

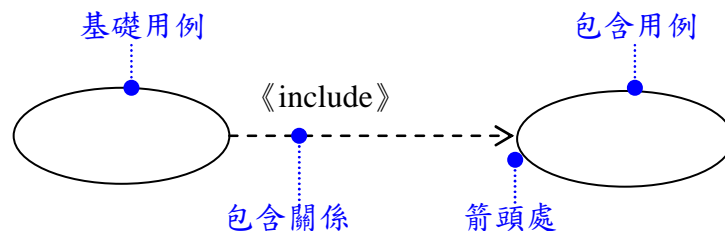


圖 1-65: 包含關係

如果，數個使用案例裡面有一段相同的流程，可以分離出另一個使用案例以便共享。在這樣的關係下，包含者稱為基礎用例，被包者稱為包含用例。比方說，如果我們打電話向信用卡的客服人員提出掛失信用卡的申請時，客服人員都會先核對身分。還有，如果去電要求更改帳單寄送地址時，客服人員也會先核對來電者身分，如圖 1-66 所示。

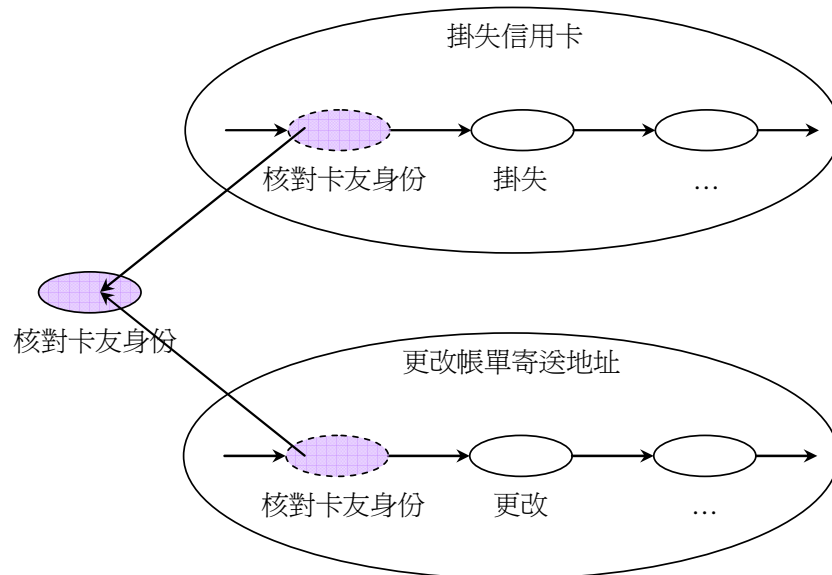


圖 1-66: 核對卡友身份

在這樣的情況下，一旦我們設計了掛失信用卡及更改帳單寄送地址這兩項使用案例時，便可以將兩者內部相同的流程分離出來，新增另一個包含用例－核對卡友身份，如圖 1-67 所示。

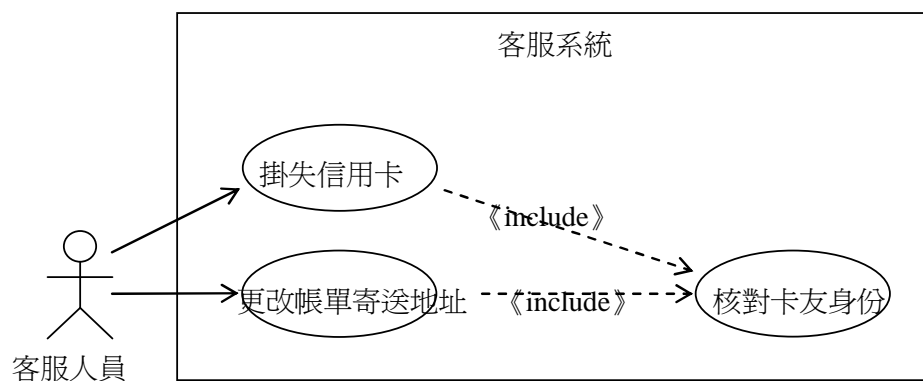


圖 1-67: 客服系統

方便維護是分離出包含用例的好處之一。因為分離出相同的流程，所以一旦需要變動使用案例時，只要變動一次即可。比方說，核對卡友身份的流程有了變動，這時只要變更一次，所有基礎用例都可以同步更新。

另一個好處是，可以迅速組裝成嶄新的服務。比方說，客服系統多了一項新服務，可以透過電話提高信用卡額度。當然，來電要求提高信用卡額度時，一樣需要先核對來電者的身分。此時，我們可以新增一個提高信用卡額度的使用案例，並且其部份流程組裝自原有的核對卡友身份，迅速提供嶄新的服務給卡友，如圖 1-68 所示。

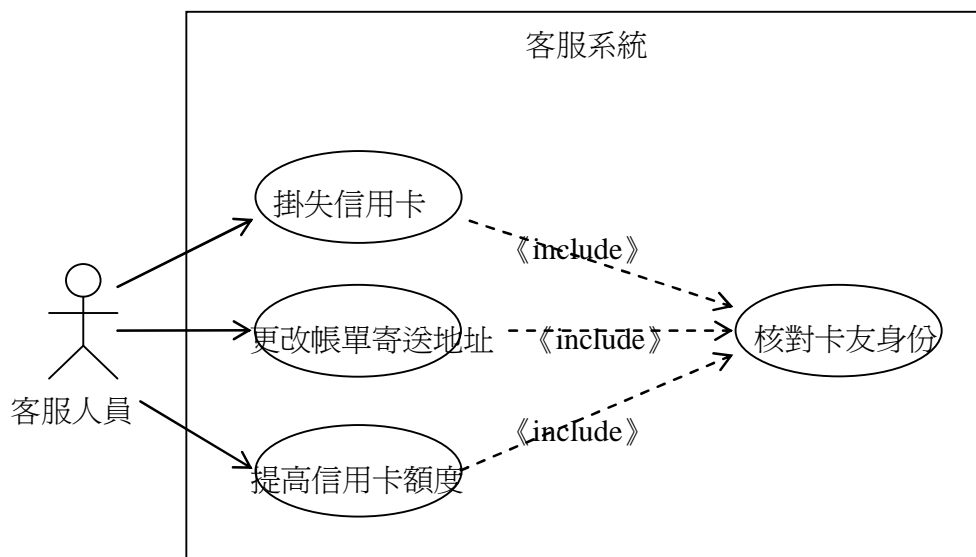


圖 1-68: 提高信用卡額度

我們再來看另一個古典迷網的例子。原本古典迷網是一個古典樂迷的社群網站，主要提供會員查詢古典 CD，以及分享心得，所以僅提供三項服務，分別為：登入會員、查詢古典 CD、貼上心得，如圖 1-69 所示。

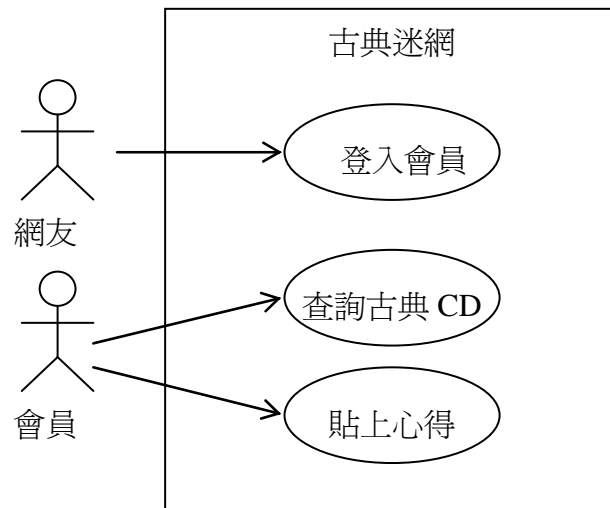


圖 1-69: 古典迷網

過了一段時間之後，因應眾多會員的要求，開始提供團購古典 CD 的服務。而團購古典 CD 服務裡，必須經過查詢古典 CD 以及會員登入的動作，所以我們使用包含關係，組裝了原有的查詢古典 CD 及登入會員使用案例，成為嶄新的團購古典 CD 使用案例，如圖 1-70 所示。

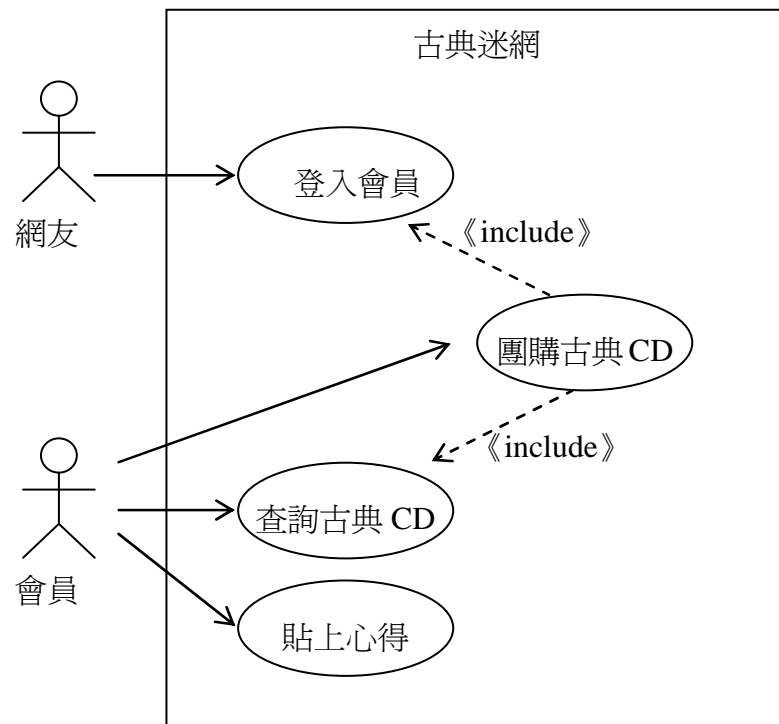


圖 1-70: 團購古典 CD

請看圖 1-71 的示意圖，古典迷網會執行四項主要流程。其中在執行團購古典 CD 使用案例期間，將包含執行查詢古典 CD 及登入會員這兩個使用案例。

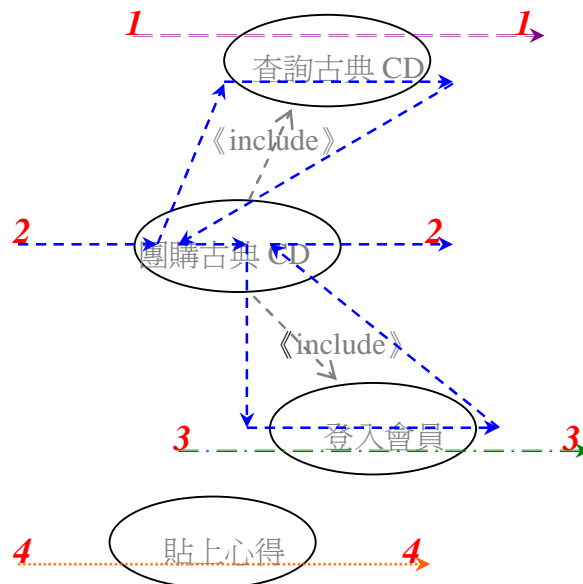


圖 1-71: 四條流程

Q7.2 擴充關係

使用案例之間的另一種關係稱為擴充關係(extend)，它的圖示是標示《extend》字眼的帶箭頭虛線，由擴充用例(extension use case)指向基礎用例(base use case)，意謂著擴充用例擴充了基礎用例的流程，如圖 1-72 所示。

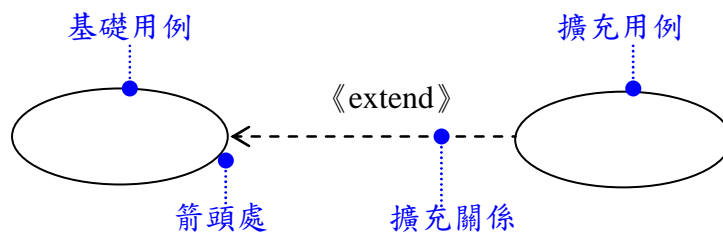


圖 1-72: 擴充關係

一個完整的使用案例裡，可以選擇擴充一小段流程，獲取另一段小服務。在這樣的關係下，原先就已經提供完整服務的使用案例稱為基礎用例，而選擇性擴充的額外服務稱為擴充用例。比方說，有些自動櫃員機在提款服務中，額外提供換百元鈔的擴充服務，如圖 1-73 所示。

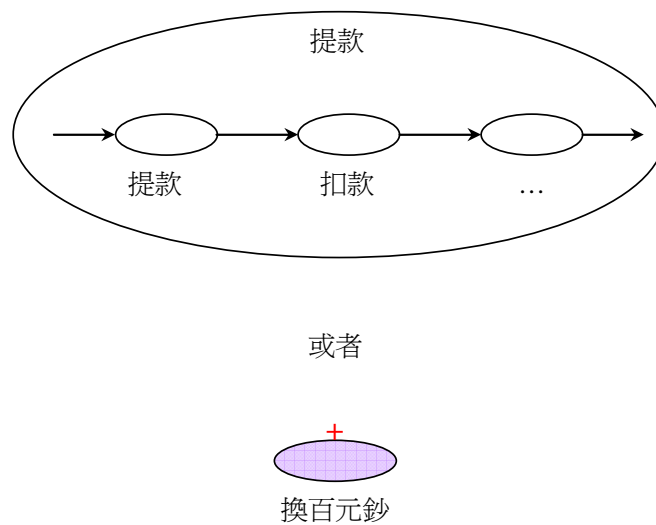


圖 1-73: 加換百元鈔

由於，換百元鈔不是獨立的服務，它僅是提款流程中的一個擴充選項，所以我們可以使用擴充關係來表達這樣的設計，如圖 1-74 所示。

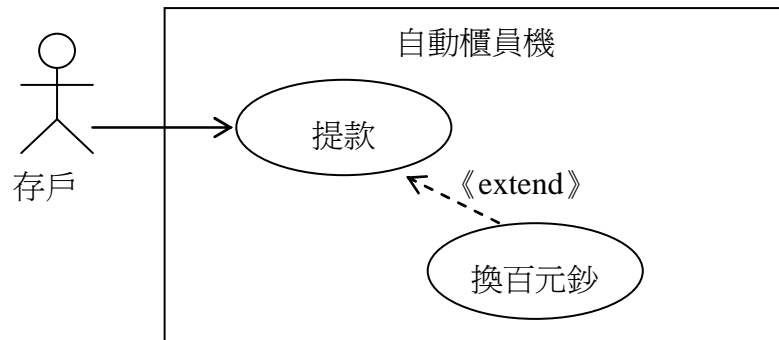


圖 1-74: 自動櫃員機

如果我們再加上包含關係的概念之後，繪製成如圖 1-75 的使用案例圖。提款使用案例與查詢餘額使用案例，含有一段相同的列印交易憑證流程。

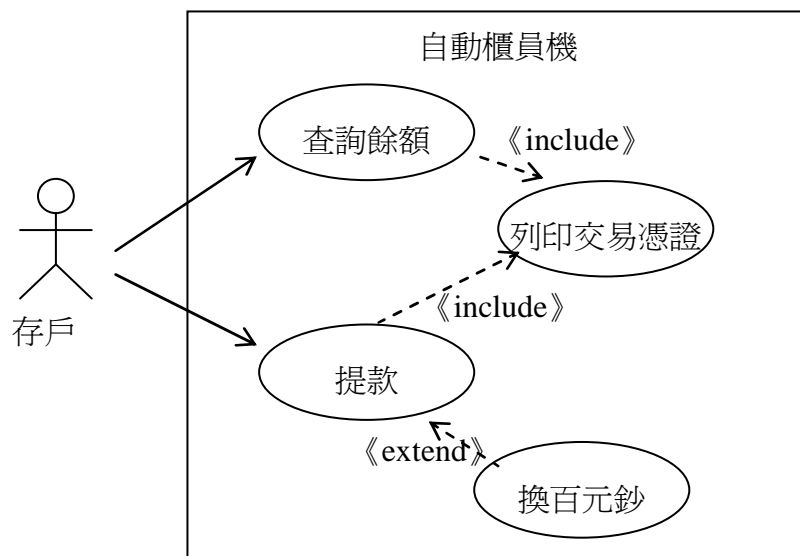


圖 1-75: 列印交易憑證

有些自動櫃員機不把列印交易憑證視為查詢餘額或提款交易裡的一段必要流程，而是視它為選擇性的擴充服務，如果是這樣的設計，可以將圖 1-75 改成圖 1-76。

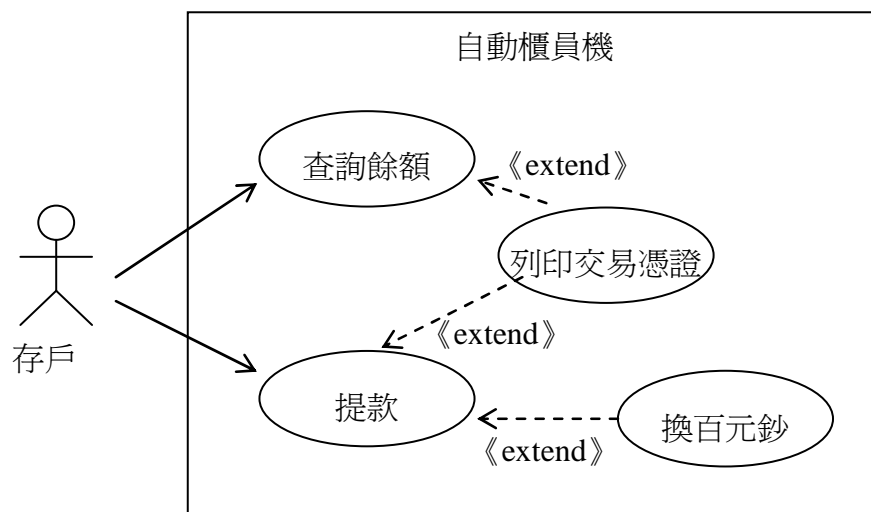


圖 1-76: 可以不列印交易憑證

系統改版時，通常不會取消原有的完整服務，而是選擇提供擴充的額外服務，這樣的情況很適合使用擴充關係。比方說，原先的信用卡系統提供刷卡服務，如圖 1-77 所示。

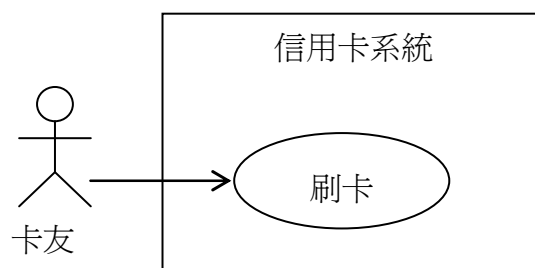


圖 1-77: 信用卡系統

後來因為盜刷情況嚴重，所以提出新的安全服務。卡友可以設定刷卡通知，當刷卡金額超過設定金額之際，系統就會傳送簡訊給卡友，提醒卡友注意。不過，卡友也可以不需理會這項貼心的小服務，維持原先的刷卡消費情況。

因此，我們提出如圖 1-78 的使用案例圖。新版的信用卡系統新增了一項設定刷卡簡訊的服務，一旦卡友設定了通知金額，往後只要刷卡超過設定金額時，系統就會執行一項額外的擴充用例—傳送簡訊。

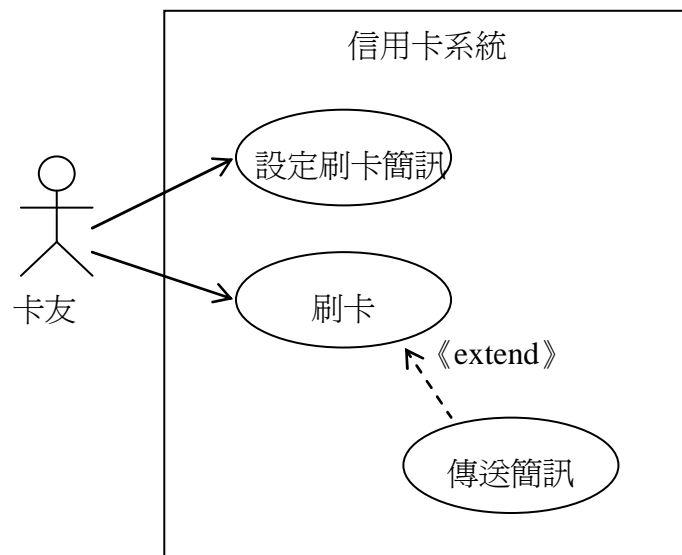


圖 1-78: 傳送簡訊

請看圖 1-79 的示意圖，信用卡系統會執行三項流程。其中在執行刷卡使用案例期間，將會檢查簡訊設定，一旦刷卡超過設定金額時，會額外執行擴充用例裡的流程，傳送簡訊通知持卡人。

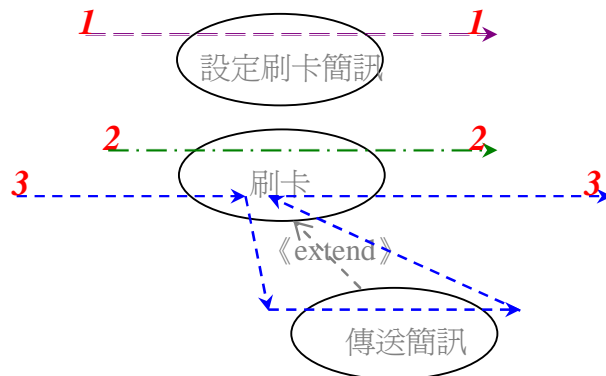


圖 1-79：三條流程

Q8 如何在循序圖裡畫 iteration ？

如何在sequence diagram裡畫 iteration ？

(worookie於JavaWorld@TW之提問)

我想好好用UML來表達一個sequence diagram裡的iteration，但是我總覺得我畫出來的東西怎麼畫都不對。希望有高人能指點一下關鍵的地方。

這裡舉個例子好了：有三個class/object：MyTest，MyMoney，and MyAccount。其中myMoney有10個myAccount，每一個myAccount都有一個數值，當用MyTest呼叫MyMoney的getTotalBalance()時，MyMoney就會去對他的10個MyAccount's分別呼叫calcBalance()，然後累加得出總值是多少。

程式如下。對此例子，不知有沒有人會畫正確的UML sequence diagram的？謝謝。

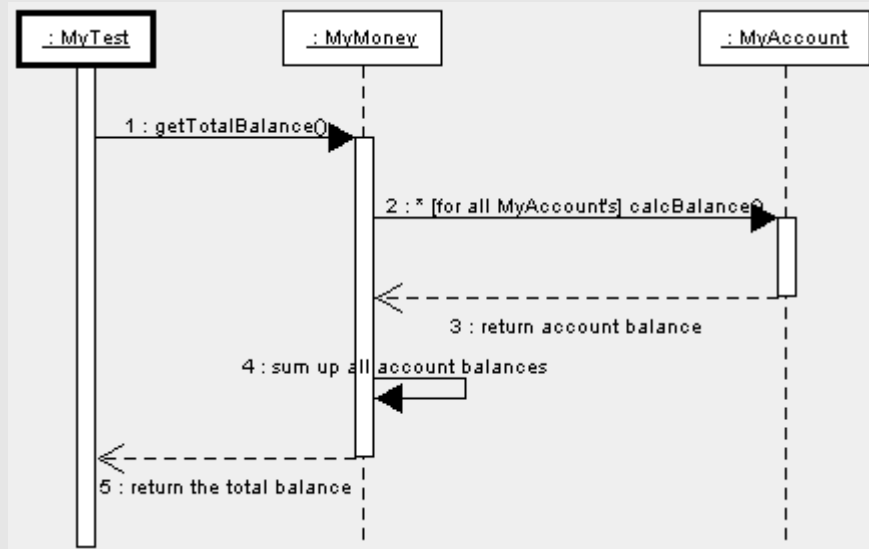

```
import java.util.*;

public class MyTest {
    public static void main(String args[]) {
        MyMoney myMoney = new MyMoney();
        int inTotalBalance = myMoney.getTotalBalance();
    }
}

class MyMoney {
    Vector veAccount;
    MyMoney() {
        veAccount = new Vector();
        for (int i=0; i<10; i++) {
            veAccount.add(new MyAccount());
        }
    }
    int getTotalBalance() {
        int totalBalance = 0;
        int totalSize = veAccount.size();
        for (int i=0; i<totalSize; i++) {
            MyAccount myAccountIterate = (MyAccount) veAccount.get(i);
            int inBalanceIterate = myAccountIterate.calcBalance();
            totalBalance += inBalanceIterate;
        }
        return totalBalance;
    }
}

class MyAccount {
    int calcBalance() {
        // 假設經過複雜的運算回傳一個值
        return 100;
    }
}
```

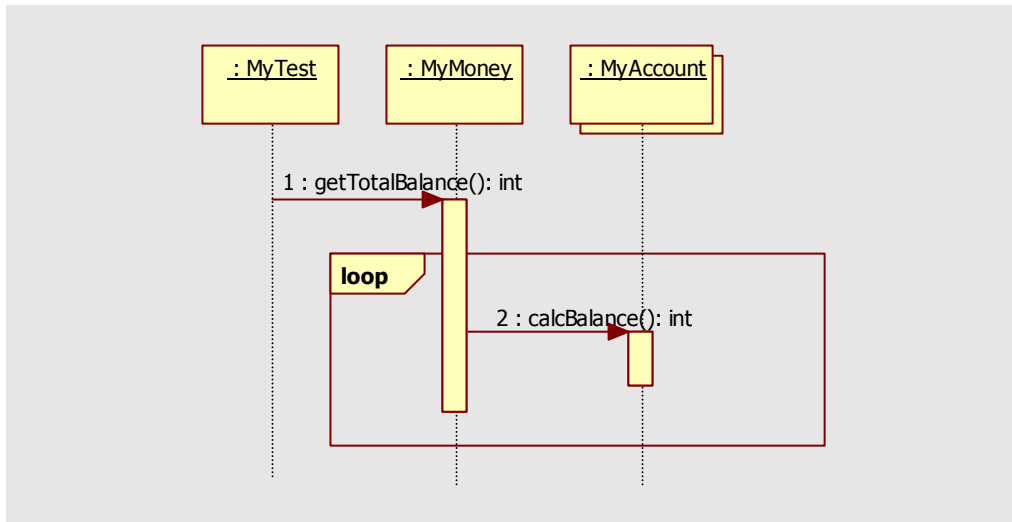
底下是我所能想到最接近的表達方式，還請諸位先進不吝指正，謝謝。



簡答：

片段(fragment)是UML 2.x版的新圖示，它改善了循序圖一直令人詬病的兩個現象。其一，循序圖無法表達過於複雜的控制流程，可是偏偏有許多設計細節，都需要藉由循序圖來傳達。其二，互動設計又多又雜，許多互動片段都可以重用，可是UML 1.x版就是沒提供重用的機制。所幸，UML 2.x版提供了片段的圖示，讓設計師可以鎖定某一互動片段，進行局部的重用或更複雜的設計。

因此，針對上述的例子，簡單繪製出如下圖所示的循序圖。



Q8.1 片段

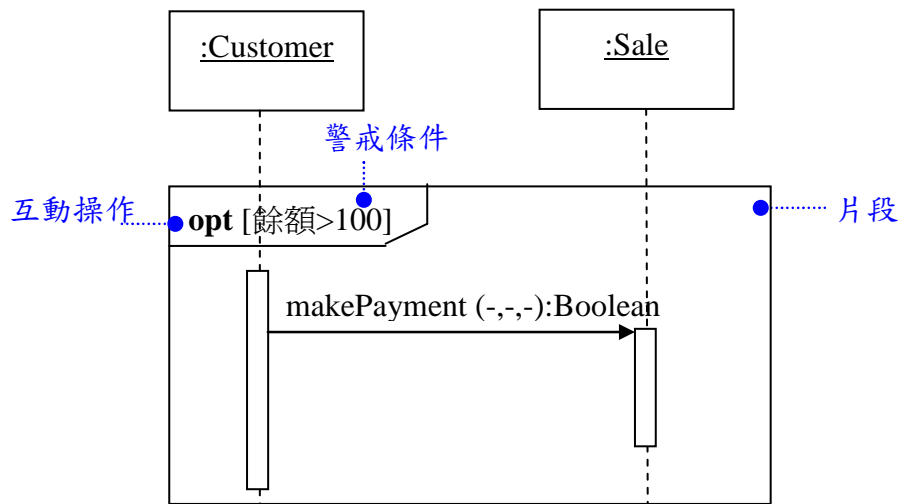


圖 1-80: opt 片段

針對循序圖裡的某一互動片段，用一個大框架(frame)圍住，並且於框內左上角以粗體字標示出互動操作(interaction operator)或互動名稱。比方說，我們以粗體字標示 **opt**，意謂著當警戒條件成立時，才會執行片段裡的互動，如圖 1-80 所示。**opt** 是隨意(option)的縮寫，也就是說，隨著警戒條件是否成立，才決定要不要執行大框架圍住的互動片段。

如果，這個互動片段並無特殊的操作，僅框住並命名某一互動片段，便於說明或重用時，就會以粗體字標示 **sd**，代表整張循序圖(sequence diagram)，或者是循序圖裡的互動片段。請看圖 1-81，循序圖的內容會循序圖名稱，框架內部則放置一群物件互動的情況。

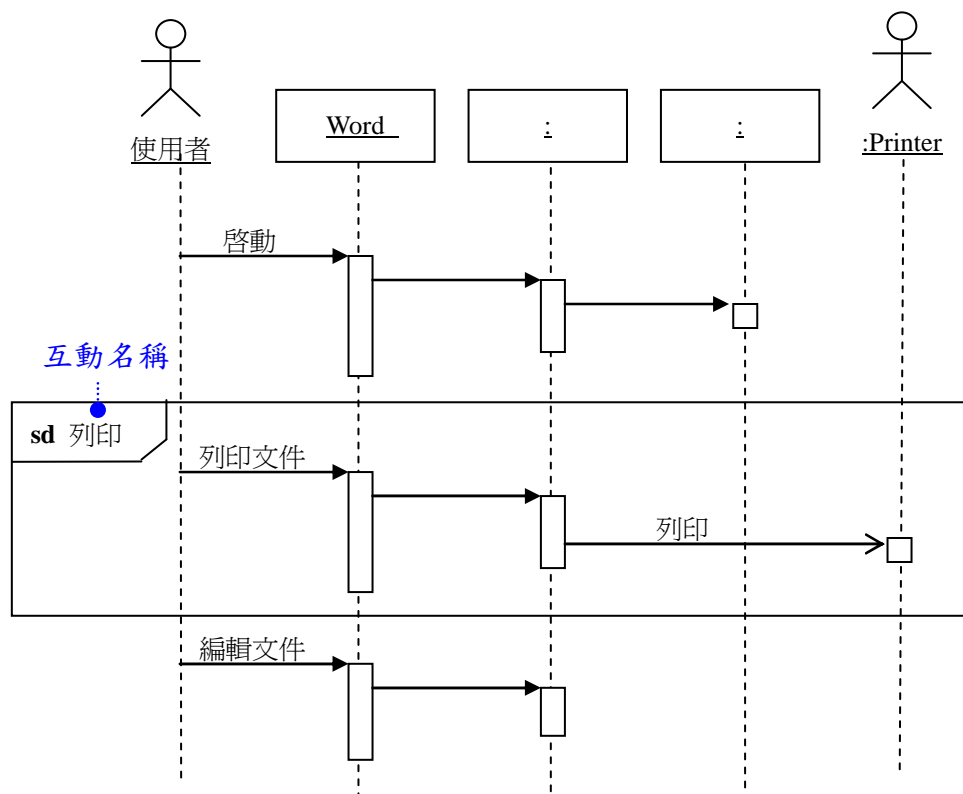


圖 1-81: 互動片段

再者，片段的框架裡頭還可以放置其它片段，構成更多樣化的互動設計。比方說在圖 1-82 裡，有一個迴圈(loop)片段，當 $x>0$ 的情況下，就會持續不斷地執行迴圈裡的互動。而迴圈片段裡，還有一個隨意(option)片段，當 $y>0$ 時，才會執行隨意片段裡的互動。

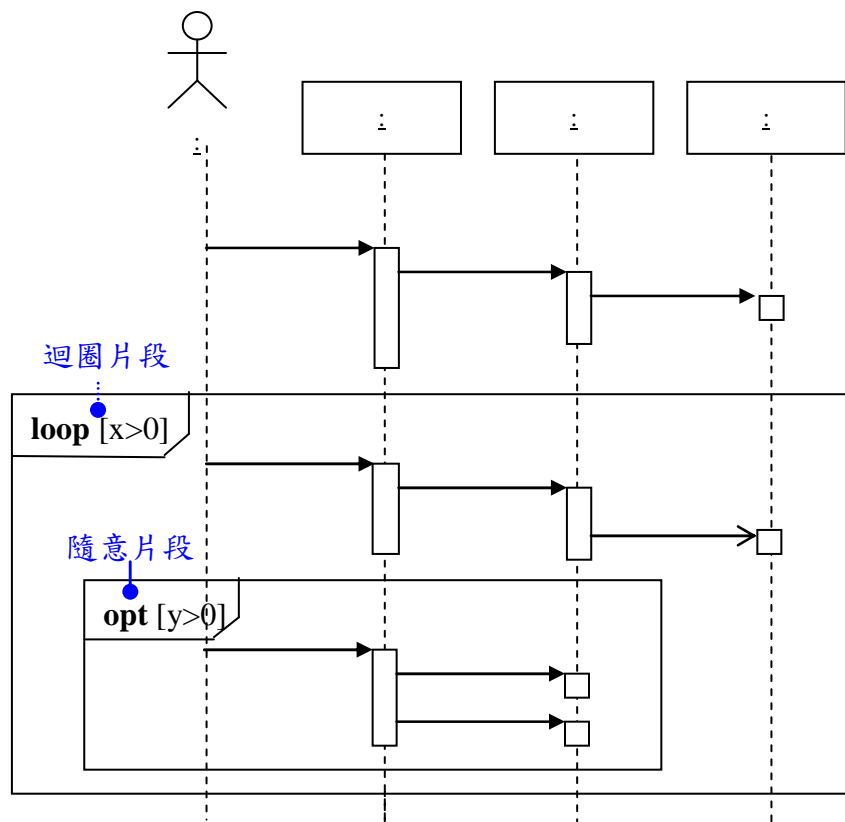


圖 1-82: 片段裡面有片段

除了 **sd** 和 **ref** 不是互動操作外，UML 2.x 提出約十個類似 **opt** 的互動操作。接下來的次小節裡，我們會談到 **sd** 和 **ref** 片段，以及五個比較常用

的互動操作，包括 `opt`、`alt`、`par`、`loop` 和 `break`。

Q8.2 引用

在任何一張循序圖中，使用標示 `ref` 的空框架，並於空框架內部標示出欲引用的片段名稱，就意謂著引用某一份已經定義好的互動片段。至於，被引用的原始設計，則使用 `sd` 標示的框架圍住一互動片段或整張循序圖，即成為可重用的設計單元。

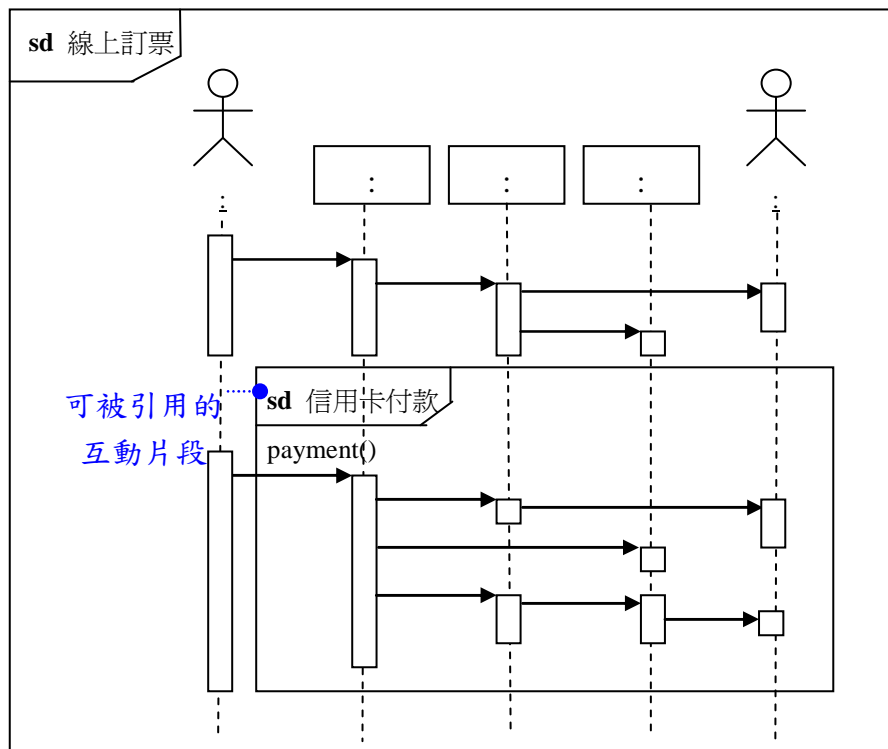


圖 1-83: 信用卡付款

比方說，在線上訂票的循序圖裡，有一段信用卡付款的互動片段，如

圖 1-83 所示。由於，在線上訂房的循序圖裡，也會用到信用卡付款的互動片段，所以可以用 sd 標示框出這段互動設計，以便線上訂房循序圖裡可以引用。

請看圖 1-84 的線上訂房循序圖，由於引用了圖 1-83 裡的信用卡付款片段，所以重用了一大段的互動設計。

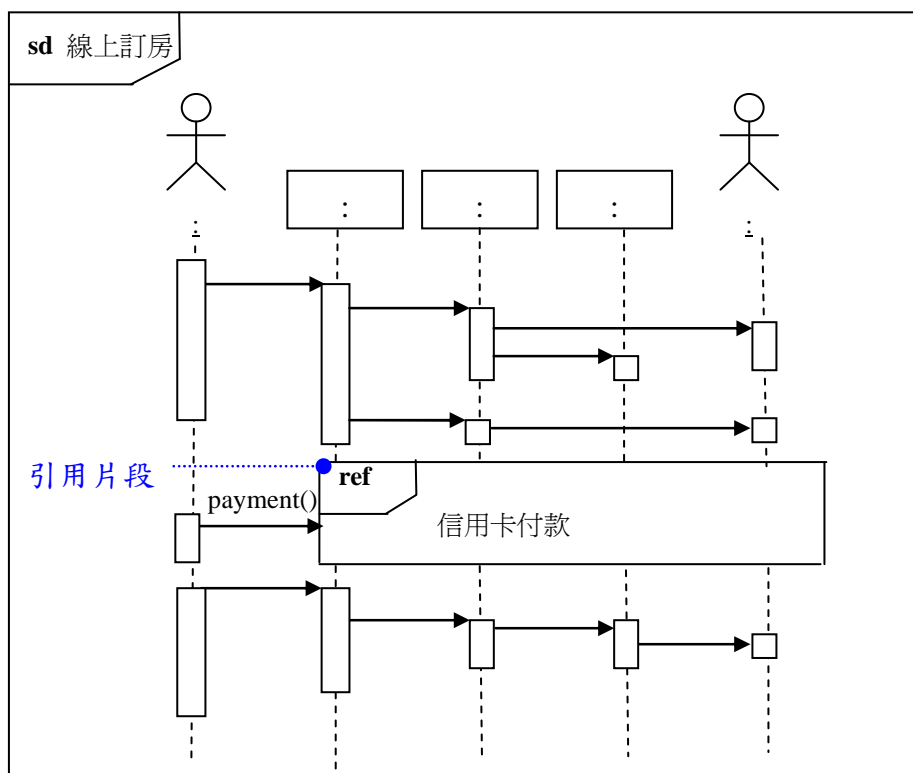


圖 1-84: 引用片段

其實，框架也很佔空間，特別是一張循序圖裡有多個片段的時候。在實務上，我們有個非 UML 標準的表示法，可以簡化圖面。請看圖 1-85 的表示法，不用框架表示引用片段，而是直接在中括號裡，標示 ref 字眼和

互動名稱，中括號外接著標示傳入互動的第一個訊息名稱。

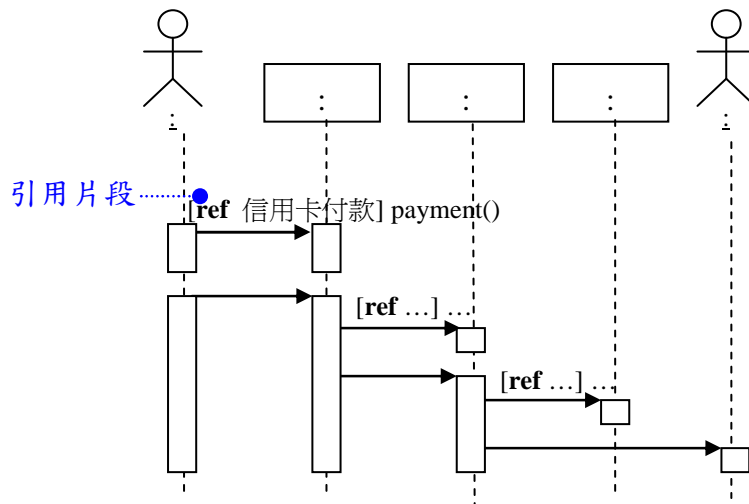


圖 1-85: 非標準表示法

通常，整個互動片段會帶有輸出入參數，可以在互動名稱後面標示出這些參數。未標示互動參數的片段，不是代表它沒有參數，只是隱藏或省略了，觀看者可能要找出原始片段，了解它的輸出入參數。當然，標示出互動參數比較佔圖面空間，不過可以節省觀看者查看原始設計的時間。

請看圖 1-86 是互動名稱的格式，跟操作名稱十分相似。特別需要注意之處，互動名稱前會標示 **sd** 表示原始的互動設計，或標示 **ref** 表示引用。再者，互動的參數不同於單一操作的輸出入參數，而是以整個互動片段為範圍的輸出入參數。

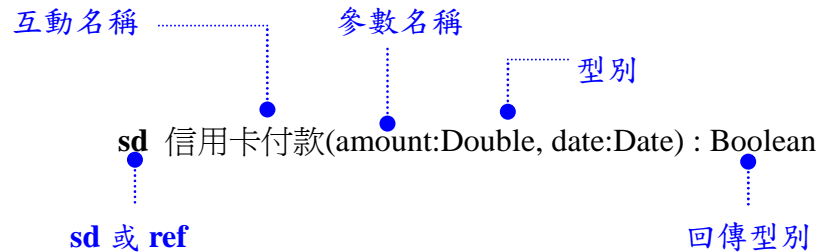


圖 1-86: 互動名稱格式

如果，互動片段有多個回傳參數時，也可以改用如圖 1-87 的表示法，標示出參數的流向。輸入參數標示 **in**；傳回參數標示 **out**；輸入參數經過操作處理之後，還會傳回給呼叫物件時，可以標示 **inout**。在此例中，信用卡付款互動片段，附有一個輸入參數，兩個回傳參數。

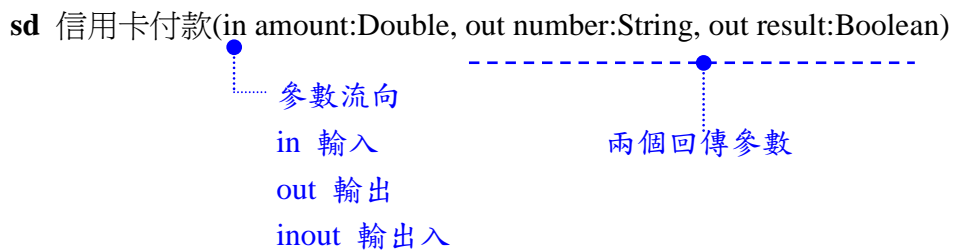


圖 1-87: 多個回傳參數

特別是單張可被引用的互動片段，最好把互動的輸出入參數標示清楚，方便引用。因為，單張互動片段不需要擔心佔用圖面空間，或者是造成圖面複雜的問題，原設計師只要稍微費點心思標示清楚，日後便於引用，如圖 1-88 所示。

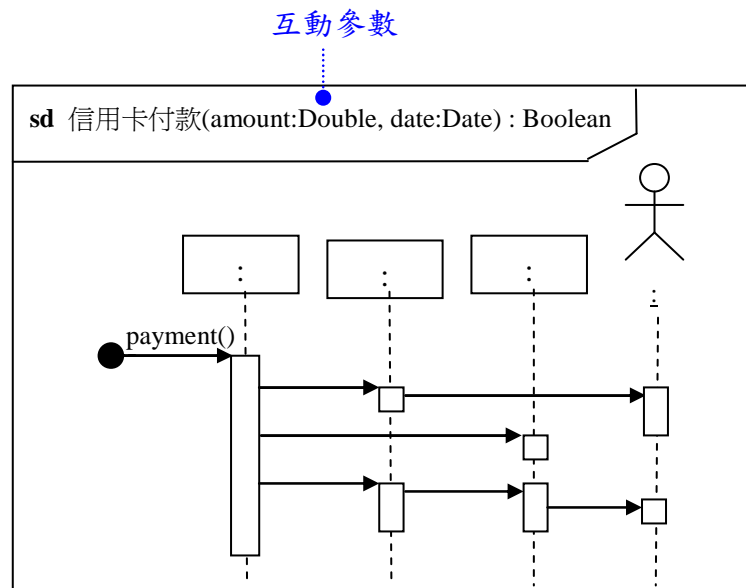


圖 1-88: 附有參數的互動

引用片段除了標示出互動參數之外，也可以填入輸入參數的數值，或是直接將回傳數值指定給某個變數或參數，以便進行接續的動作。請看圖 1-89，線上訂房循序圖中引用了信用卡付款的互動片段，並且將互動回傳的數值直接指定給 `status`。

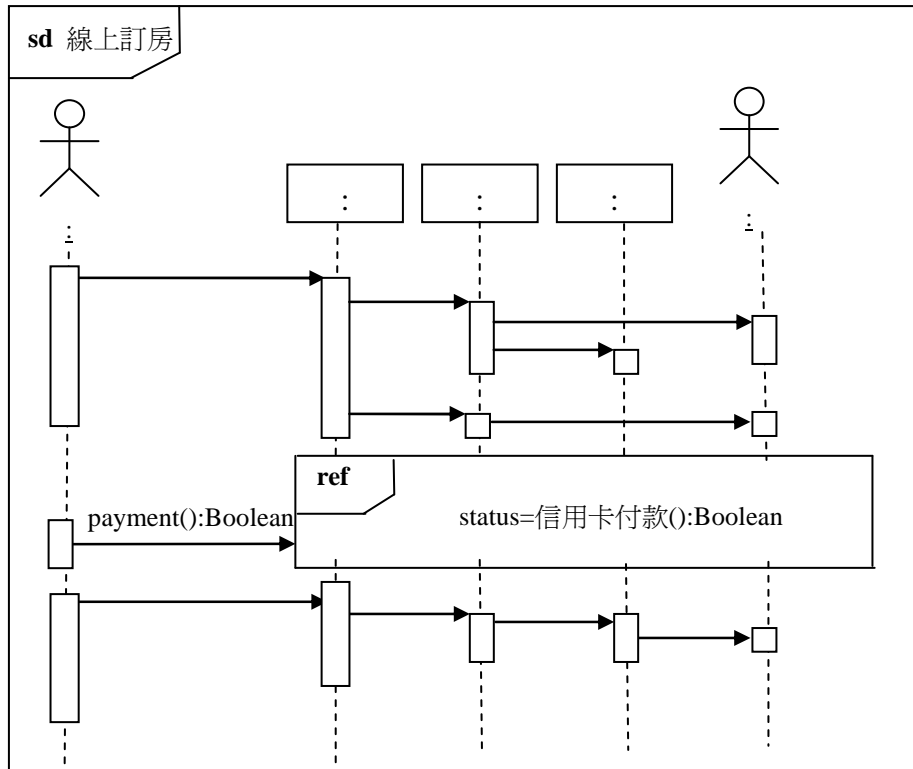


圖 1-89: 附有參數引用片段

如果，我們採用非 UML 標準的表示法的話，也是可以表達互動參數，如圖 1-90 所示。

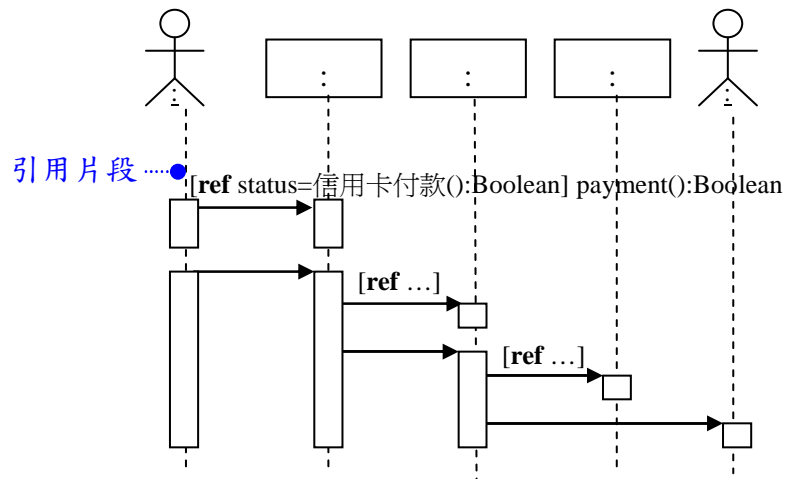


圖 1-90: 引用片段的非標準表示法

Q8.3 隨意

有些互動片段僅在符合某些條件的情況下，才會開始運作。這種帶有警戒條件的片段，稱為隨意片段(option fragment)。請看圖 1-91，標示 opt 的片段，當 YN=True 這個警戒條件成立時，才會執行片段裡的流程。

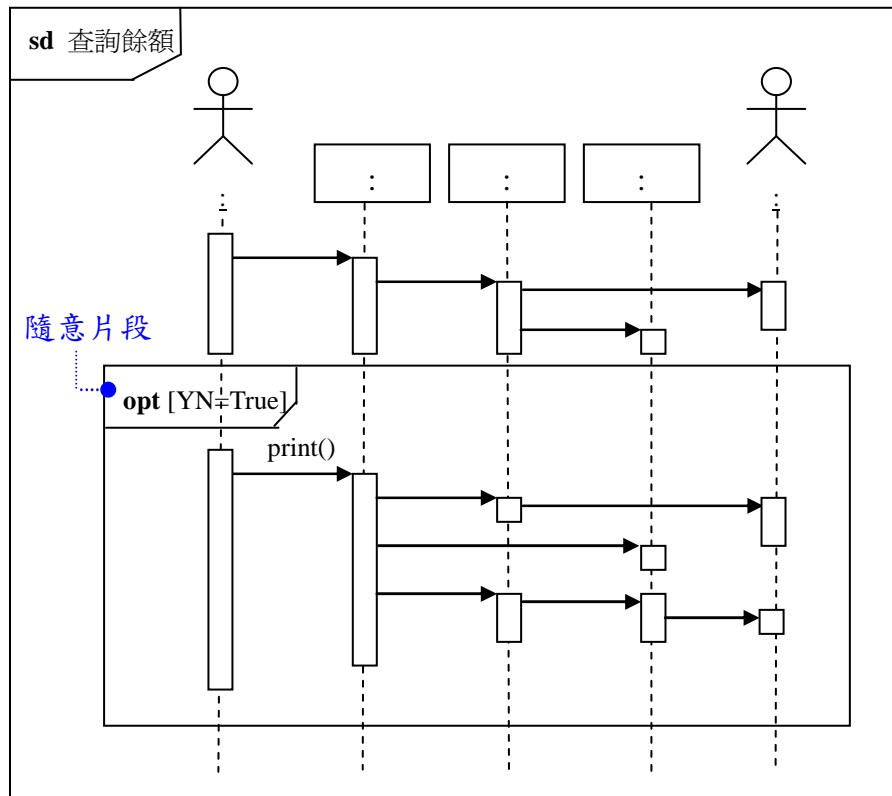


圖 1-91: 隨意片段

片段裡面還可以有其它片段。比方說，在隨意片段裡頭放置引用片段，意謂著這個引用片段裡的互動不一定會執行，如圖 1-92 所示。

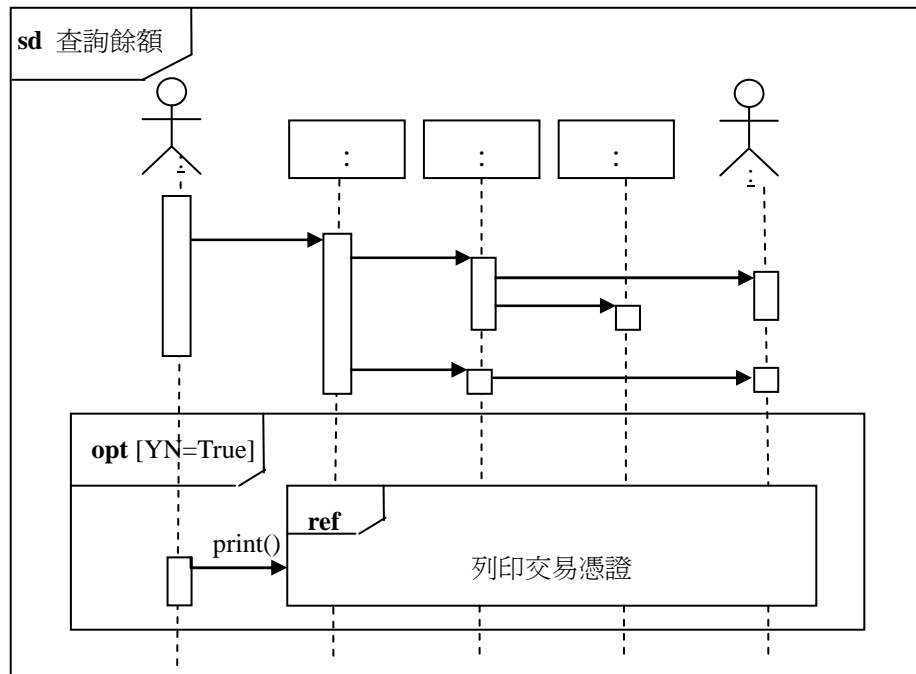


圖 1-92: 片段內的片段

如果，隨意片段裡的只有一條訊息的話，使用隨意片段就太佔圖面空間了。在實務上，我們會將警戒條件直接標示在訊息旁，而不用隨意片段，以便簡化圖面的複雜度。不過，在 UML 2.x 版中，已經去除直接把警戒條件標示在訊息旁的作法。所以，圖 1-93 的表示法不符合 UML 2.x 標準。在這個例子裡，顧客物件在餘額大於 100 時，才會發出調用 `makePayment` 操作的訊息。

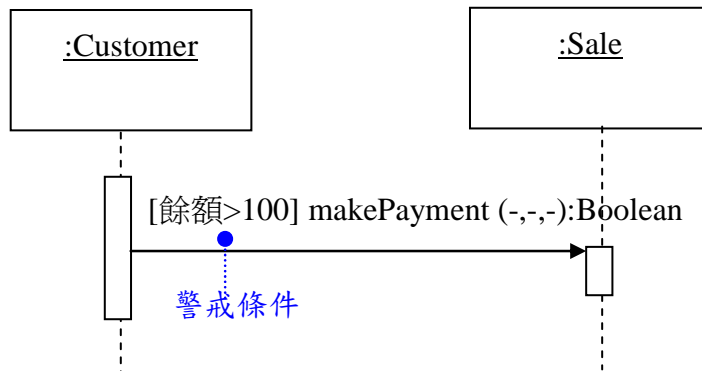


圖 1-93: 警戒條件

再者，一個隨意片段像個是非題，是就執行，非就不執行。多個隨意片段組合起來，就成了多選題，可能全部都不執行，也可能全部執行，或者僅執行某幾個片段。請看圖 1-94，一個複合片段(combined fragment)裡，使用虛線來隔成數個互動區域(interaction operand)，每個互動區域都可以帶有自己的警戒條件。

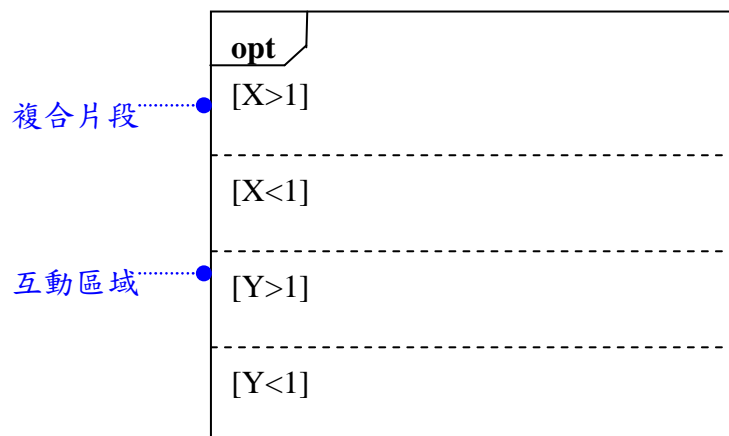


圖 1-94: 複合的隨意片段

Q8.4 擇一

如果說，隨意片段像是非題，複合的隨意片段像多選題，那麼標示 alt 的擇一片段(alterative fragment)就像是單選題。請看圖 1-95 的擇一片段，意謂著僅有一個互動片段會執行，不多不少，就執行一個。通常，我們會特別為擇一片段設計 else 互動區域，以確保一定會執行一段互動。

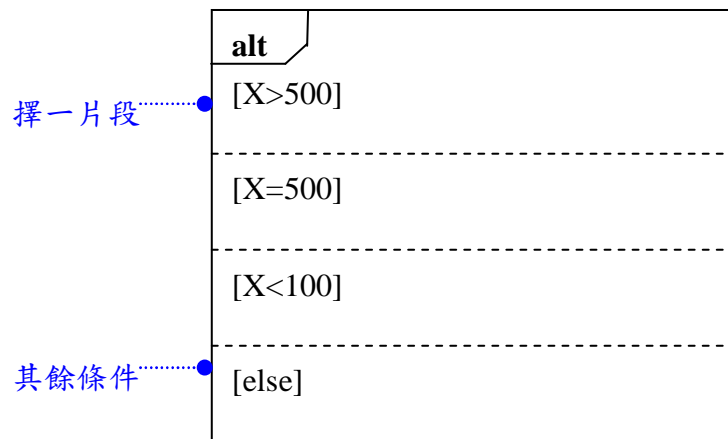


圖 1-95: 擇一片段

Q8.5 並行

無論，隨意片段或擇一片段都屬於依序執行的片段，如果要並行執行多個互動片段時，就需要使用標示 par 的並行片段(parallel fragment)了。請看圖 1-96 的並行片段，意謂著所有的互動片段會同時執行，而不是依序執行。

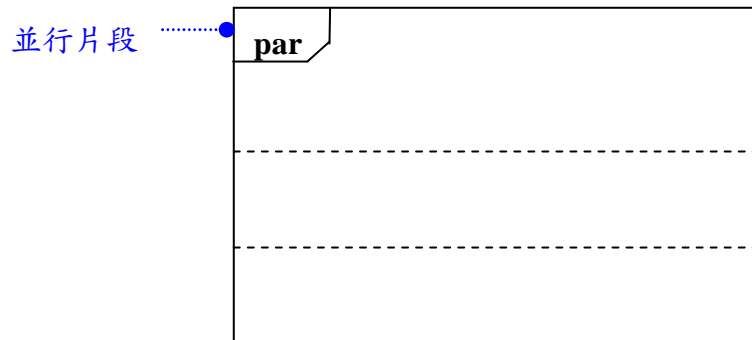


圖 1-96: 並行片段

請看圖 1-97 的例子，在執行提款流程期間，將並行執行扣款、退卡、吐鈔這三段互動片段。

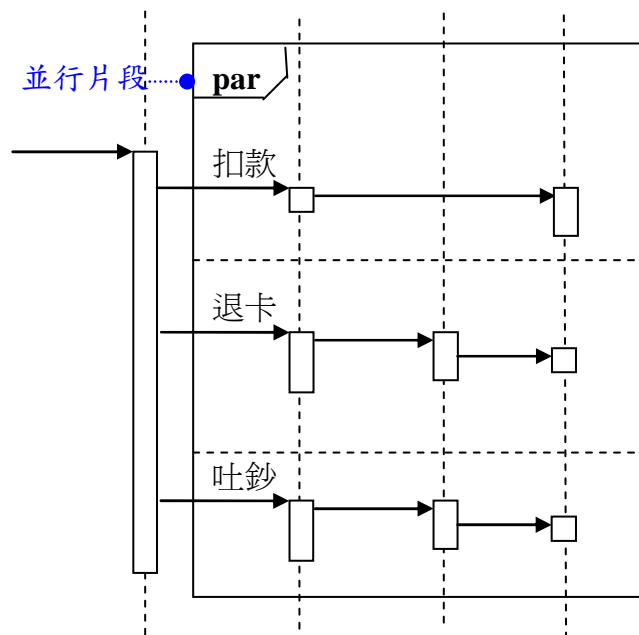


圖 1-97: 並行執行三段互動片段

Q8.6 迴圈

如果，套用程式語言的行話，隨意片段就像 if 子句，擇一片段像 case 子句。至於，此處標示 loop 的迴圈片段(loop Fragment)，當然就像是 for-loop 子句了。

請看圖 1-98，在 loop 字眼後面可以附上最少及最多的迴圈執行次數，以及是否繼續執行的警戒條件。當迴圈執行達到最少的執行次數之後，每多執行一次就會檢核一次警戒條件。只要，警戒條件成立，就會繼續執行迴圈，直到達最多執行次數才會停止迴圈。或者，在執行達到最少次數之後，只要一檢核到警戒條件不成立時，就會停止迴圈。

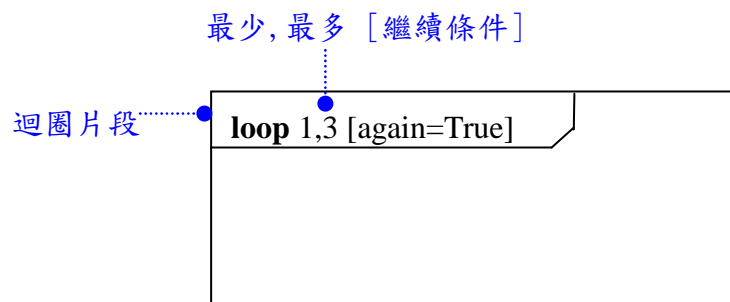


圖 1-98: 迴圈片段

比方說，我們在使用自動櫃員機提款時，通常會有三次輸入密碼的機會。所以，這個迴圈最少一次，最多三次。當然，第一次輸入密碼錯誤之後，可以選擇是否還要重新輸入密碼，所以設有 `again` 的警戒條件，可以中斷迴圈，如圖 1-99 所示。

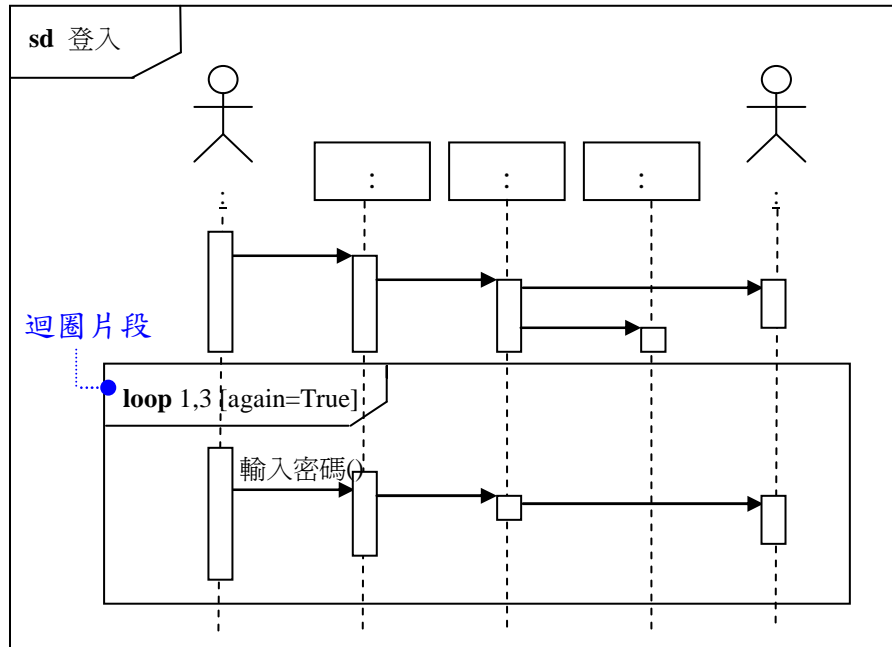


圖 1-99: 輸入密碼最多三次

Q8.7 中斷

有時候，我們希望迴圈不會停止，直到發生一個中斷事件，才中斷迴圈。請看圖 1-100，標示 **break** 的中斷片段(break fragment)，意謂著當發生了中斷片段裡的訊息時，就會中斷迴圈。

中斷片段可以出現在任何互動片段內部，中斷該互動片段。比方說，它也可以出現在 **sd** 互動片段內部，隨時中斷整張循序圖的執行，如圖 1-101 所示。

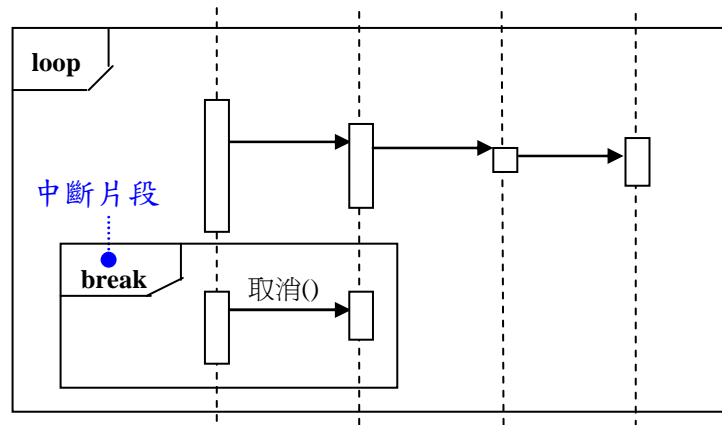


圖 1-100: 中斷迴圈

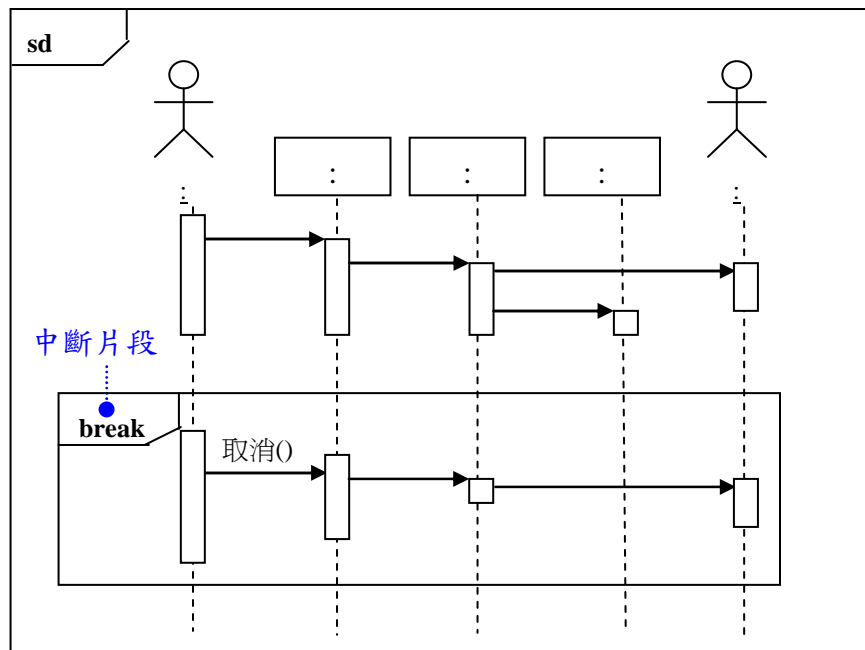


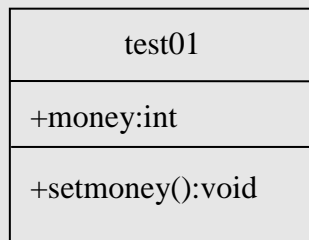
圖 1-101: 中斷循序圖

Q9 類別的建構元要畫在哪啊？

UML class diagram的constructor要畫在哪啊？

(rz600000於JavaWorld@TW之提問)

我有一個class test01。test01有一個constructor(建構元)，建構元要畫在哪啊？

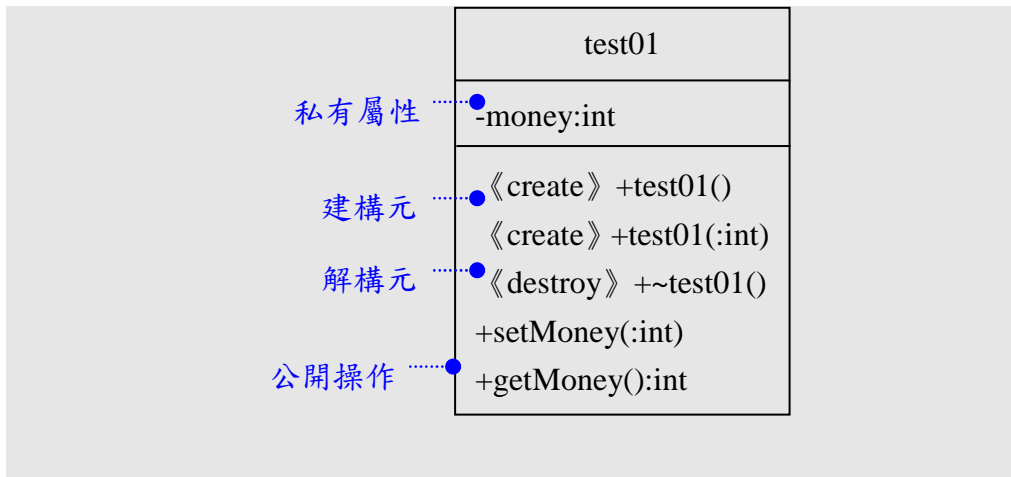


解說test01是一個class，裡面有個變數int money，有一個method setmoney:void。那test01的建構元要畫在哪啊？

簡答：

常見的操作可分為三大類，包含有：屬性數值的存取操作、計算公式、物件的建構元與解構元，這些操作都可以放置於類別圖示中的操作格中。

若是需要標記建構元或解構元的話，可於操作名稱前標記《create》代表建構元，或標記《destory》代表解構元，如下圖所示。



Q9.1 建構元與解構元

類別除了記錄共同的屬性項目之外，更重要的，它還可以用來記錄共同的操作(operation)。所以，透過一個個的類別，可以分門別類屬性與操作的程式碼，方便系統開發期間的使用，或者是系統維護期間的變動。

常見的操作可分為三大類，如下：

1. 屬性數值的存取—物件擁有自己的屬性數值，但外界不得隨意存取；外界必須透過物件對外提供的操作，才能夠存取屬性數值，此即為物件之封裝性(encapsulation)。
2. 企業規則或公式—這類操作最為重要，因為它記錄了重要的企業運作規則，或者是企業公式。舉例來說，基金的申購單位=申購金額÷基金淨值，這就是一項很重要的計算公式。所以，我們會將這個計算公式放置於某一項操作中，便於所有的基金物件計算申購單位時共用之。
3. 物件的建構與解構—在執行期間，一產生某種物件時，可能會需

要一些初始設定，常見如屬性數值的預設值設定，程式語言通常稱這種操作為「建構元」(constructor)。相對地，在執行期間，不再需要該物件而欲釋放物件所佔有的記憶體空間之際，可能需要進行某些最終設定，稱此種操作為「解構元」(destructor)。

請看圖 1-102，代表類別的三格矩形圖示中，頂格放置類別名稱，中格放置屬性，底格放置操作。申購交易類別的兩項操作，屬於企業公式類的操作，非常重要。至於，基金類別的操作則可以用來存取基金物件的屬性數值。

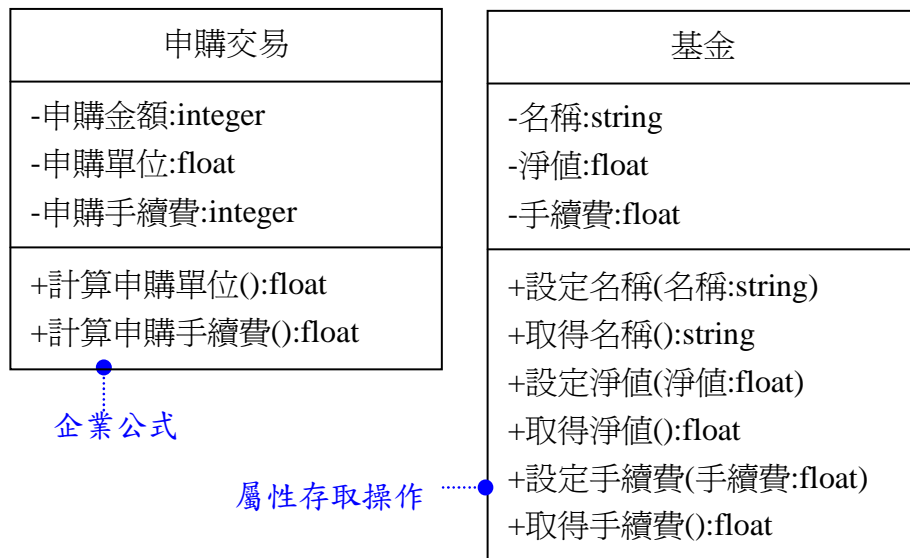


圖 1-102: 操作

物件具有封裝性，可封裝屬性數值與操作於內，除非透過物件對外公開的操作，否則外界將無法隨意使用屬性數值。實現物件封裝性的常見方

法之一，即是為每一個屬性和操作設定能見度(visibility)等級，用以決定外界能夠視見物件內部的程度，以便掌握物件的封裝性。

除非特殊情況，否則一般會預設屬性能見度為「私有等級」(private)，操作的能見度則為「公開等級」(public)。私有等級的屬性，僅物件自身可以直接使用，外界根本無法視見，當然也無法直接使用之。不過，透過公開等級的操作，外界即可間接使用屬性數值。請看圖 1-103，屬性名稱前有一個減號(-)標記，這就是私有等級的能見度；再看到操作名稱前有一個加號(+)標記，此即為公開等級的能見度。

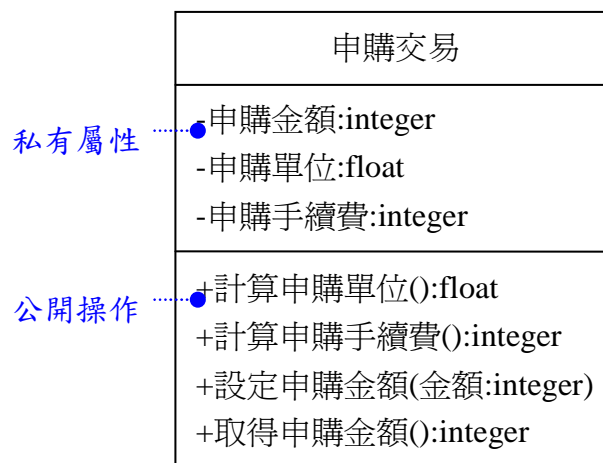


圖 1-103: 能見度

Q9.2 物件之生滅

在循序圖中，我們可以使用「誕生訊息」(creation message)與「消滅訊息」(destruction message)的圖示，表達物件執行建構元與解構元的情況。接收物件收到誕生訊息時，將執行建構元，進行物件誕生之後的初始化設定。請看圖 1-104 之左圖，誕生訊息的圖示是一條帶開放箭頭的虛

線，射向誕生物件，意味著物件是在接收到誕生訊息之際，才剛誕生。或者，也可以使用如右圖的表示法，採用呼叫訊息的帶實心箭頭實線，並於其旁標記《create》字眼。

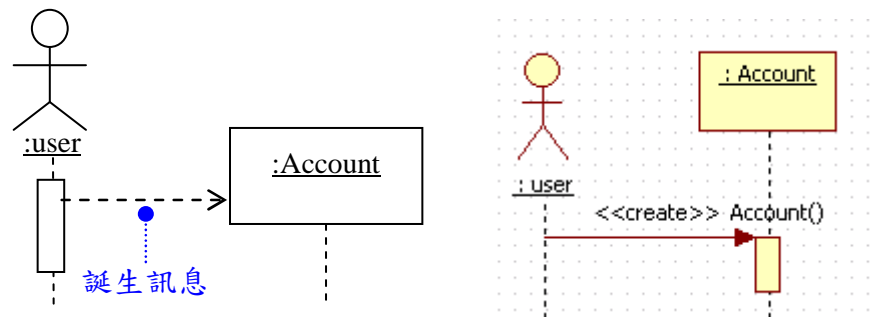


圖 1-104: 誕生訊息

至於，消滅訊息的圖示則是在訊息箭頭端打上大叉叉(x)，而且存活線終止於大叉叉處。因為物件在收到消滅訊息之後，將執行解構元，進行物件消滅之際的設定工作，然後就被消滅了、不再存活了，所以存活線也就終止了，如圖 1-105 所示。

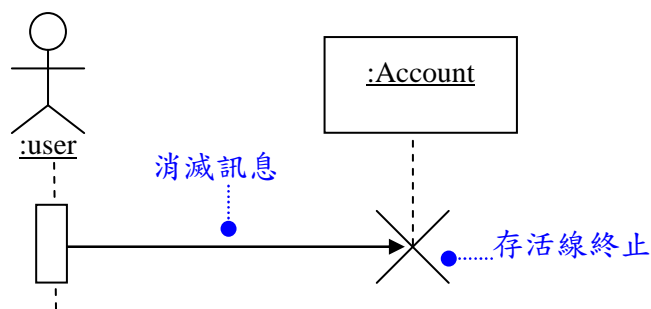


圖 1-105: 消滅訊息

Q10 請問 UML 工具？

請問UML工具？

(wanwan722於程式設計俱樂部之提問)

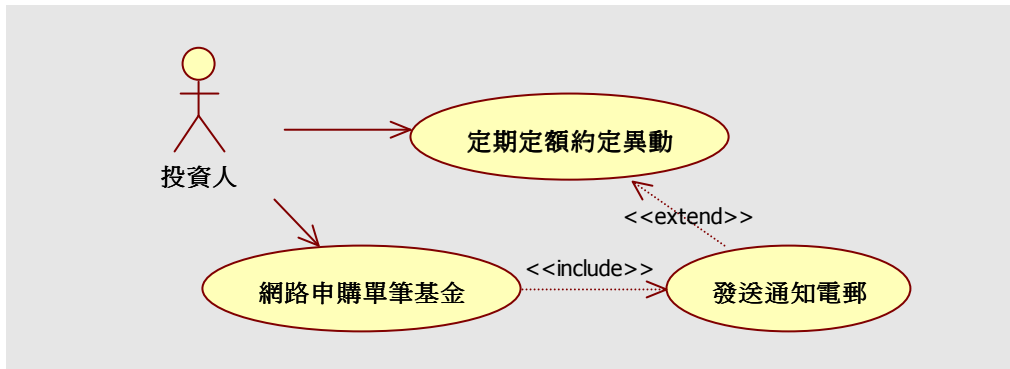
我們公司在尋找一個UML的繪圖工具，因為我們的project都不會太大。大部分都是SA人員畫完UML圖後，coding人員依照UML的diagram轉譯成code。因此我們不需要強大的diagram轉code的功能，只需要簡單但是完整的UML editor。

可以推薦一下有什麼好tools嗎？我個人有試過Together及Visual。不過都是Java的，有不是Java版的嗎？我們之前是用Delphi的Model Maker，很簡單好用，但是因為是Delphi Class Diagram沒有辦法畫出多重繼承。

簡答：

我寫了一個StarUML的介紹和操作示範，請您到UML Blog(<http://www.uml.tw.com>)下載此範例之pdf檔。

這個範例中，我們會以下述的使用案例圖為例，示範StarUML的操作，讓您可以迅速學會繪製UML圖。StarUML是一套開放原碼(open source)的UML開發工具，您可以到StarUML的官方網站(<http://www.staruml.com/>)下載。



Q10.1 免費的 UML 開發工具—StarUML



圖 1-106: StarUML 的官方網站

StarUML 是一款開放原碼的 UML 開發工具，由韓國公司主導開發出來的產品，您可以直接到 StarUML 網站(<http://www.staruml.com/>)，如圖 1-106 所示，下載約莫 22MB 的執行檔。在本小節中，我們用以示範說明的版本為 5.0.2.1570。

總歸來說，StarUML 具備下列多項特色：

- 可繪製 9 款 UML 圖—支援 UML 1.4 版裡的使用案例圖、類別圖、循序圖、狀態圖、活動圖、通訊圖、元件圖和部署圖，以及 UML 2.x 版裡的組合結構圖。
- 完全免費—StarUML 是一套開放原碼的軟體，不僅免費自由下載，連程式碼都免費開放。
- 多種格式影像檔—可匯出 JPG、JPEG、BMP、EMF 和 WMF 等等格式的影像檔。
- 正反向工程—StarUML 可以依據類別圖的內容產出 Java、C++、C# 程式碼，也能夠讀取 Java、C++、C# 程式碼反向產出類別圖。反向工程有兩個主要用途，其一是舊有的原始程式碼反轉成圖之後，可以建構 UML 模式的方式繼續將新的設計添加上去；另一項用途是，想要解析原始程式碼時，可以透過反轉的類別圖來理解，不再需要觀看一行又一行的程式碼，這將節省大量的時間和精力。
- 語法檢驗—StarUML 遵守 UML 1.4 版的語法規則，不支援違反語法的繪圖動作。
- 支援 XMI—StarUML 接受 XMI 1.1、1.2 和 1.3 版的匯入匯出。XMI 是一種以 XML 為基礎的交換格式，用以交換不同開發工具所產出的 UML 模式。
- 匯入 Rose 檔—特別的是，StarUML 可以讀取 Rational Rose 產出的檔，讓原先 Rose 的用戶可以轉而使用免費的 StarUML。早

期，Rational Rose 是市場佔有率最高的 UML 開發工具，同時也是相當昂貴的工具。由於，Rational Rose 非常聞名，後來讓 IBM 給收購了。

- 支援樣式—支援 23 種 GoF 樣式(pattern)，以及 3 種 EJB 樣式。GoF 樣式出自於 Erich Gamma 等四人合著的“Design Patterns：Elements of Reusable Object-Oriented Software”一書，其內列出了 23 種軟體樣式，可解決軟體設計上的特定問題。StarUML 也支援了 3 種常用的 EJB 樣式，分別為 EntityEJB、MessageDrivenEJB、SessionEJB。

StarUML 結合了樣式和自動產碼的功能，方便我們落實設計。請看圖 1-107 的例子，State 樣式讓物件可以依據內部狀態的變動而改變外部行為，使得物件看起來像是改變其類別。

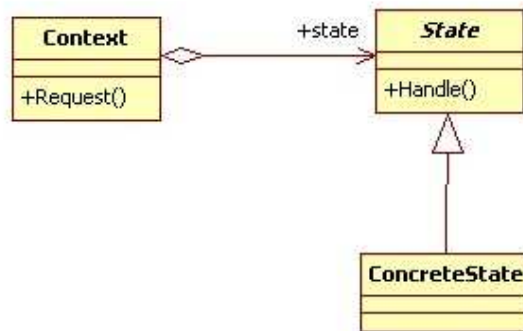


圖 1-107: State 樣式

下面圖 1-108~112 列出 StarUML 為 State 樣式自動產出的 C++的程式碼。

```
//// StarUML C++ Code
1.  #if !defined(_STATE_H)
2.  #define _STATE_H
3.  class State
4.  {
5.  public:
6.      void Handle();
7.  };
8.  #endif
//// StarUML C++ Code
```

圖 1-108: State.h

```
//// StarUML C++ Code
1.  #include "State.h"
2.  void State::Handle()
3.  {}
//// StarUML C++ Code
```

圖 1-109: State.cpp

```
//// StarUML C++ Code
1.  #if !defined(_CONCRETESTATE_H)
2.  #define _CONCRETESTATE_H
3.  #include "State.h"
4.  class ConcreteState : public State
5.  {};
6.  #endif
//// StarUML C++ Code
```

圖 1-110: ConcreteState.h

```
//// StarUML C++ Code
1.  #if !defined(_CONTEXT_H)
2.  #define _CONTEXT_H
3.  #include "State.h"
4.  class Context
5.  {
6.  public:
```

```

7.      void Request();
8.      State *state;
9.  };
10. #endif
//// StarUML C++ Code

```

圖 1-111: Context.h

```

//// StarUML C++ Code
1.  #include "Context.h"
2.  void Context::Request()
3.  {}
//// StarUML C++ Code

```

圖 1-112: Context.cpp

Q10.2 新增使用案例圖

在簡單認識 StarUML 之後，我們要來示範繪製出如使用案例圖，如下圖所示。

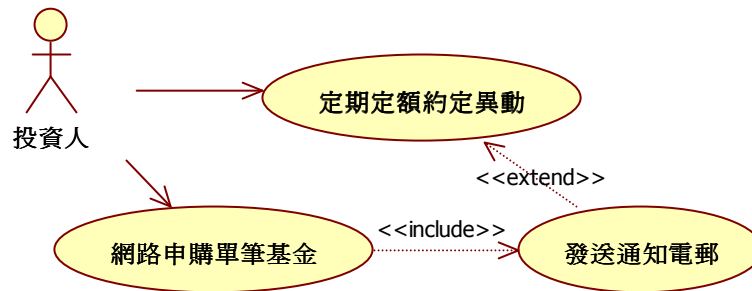


圖 1-113: 使用案例圖

一執行 StarUML 時，會出現如圖 1-114 的專案種類選擇視窗，StarUML 的預設專案是 Default Approach。我們不使用預設專案，請改選

Empty Project。



圖 1-114: New Project By Approach

接下來，您可以依照下述步驟新增使用案例圖：

1. 開啓空白專案後，請更改專案名稱爲「基金系統」，如圖 1-115 所示。



圖 1-115: 鍵入專案名稱

2. 在模式瀏覽器(Model Explorer)中，根目錄與專案同名。於專案項目處，按下滑鼠右鍵，執行選單中的【Add►Model】來新建模式，並爲新模式更名為「使用案例」，如圖 1-116 所示。



圖 1-116: 新增模式

3. 接著，點選模組設計項目，執行【Add Diagram►Use Case Diagram】來新增使用案例圖，可以更名為「主要使用案例圖」，如圖 1-117 所示。

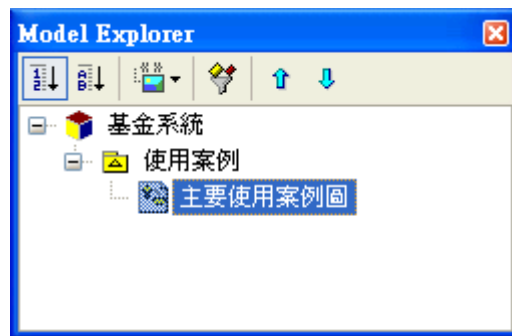


圖 1-117: 新增使用案例圖

4. 新增使用案例圖之後，StarUML 會自動備妥如圖 1-118 的使用案例圖繪製環境。

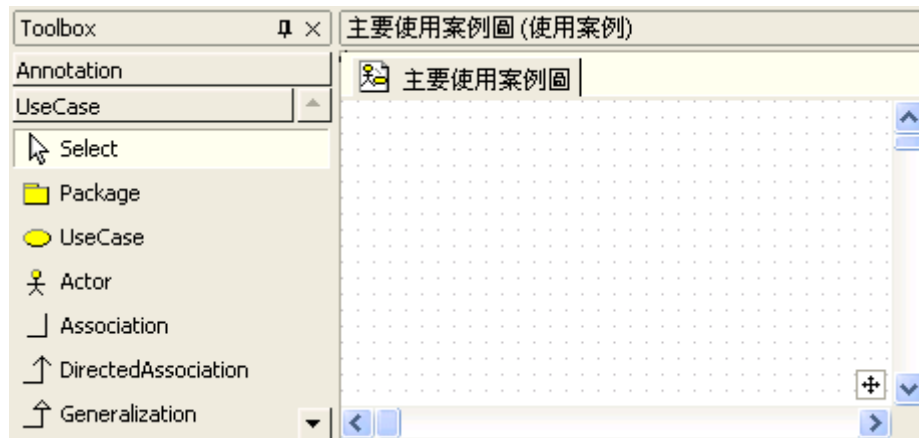


圖 1-118: 使用案例圖面及工具箱

Q10.3 繪製使用案例圖

現在，我們要來建立參與者與使用案例，步驟如下：

1. 點選工具箱裡的人型 Actor(參與者)圖示，如圖 1-119 所示。

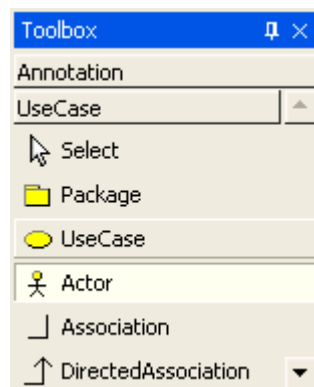


圖 1-119: 點選 Actor

2. 隨後，在圖面空白處再點一次，新增了一個代表參與者的圖示，並更名為「投資人」，如圖 1-120 所示。

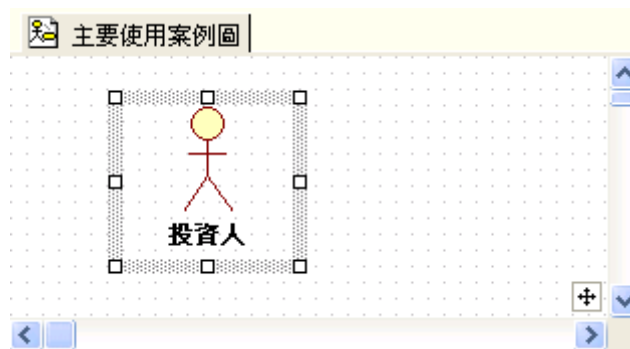


圖 1-120: 新增參與者

3. 點選工具箱裡的橢圓 UseCase(使用案例)圖示，如圖 1-121 所示。

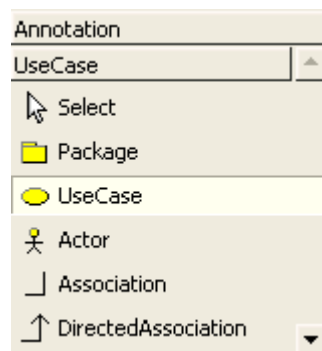


圖 1-121: 點選 UseCase

- 隨後，在圖面空白處再點一次，新增了一個代表使用案例的圖示，並更名為「定期定額約定異動」，如圖 1-122 所示。

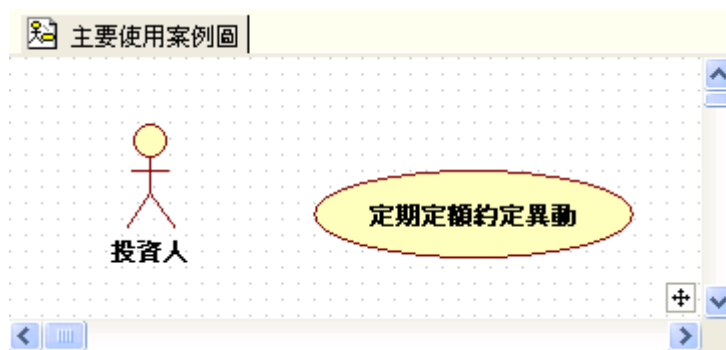


圖 1-122: 新增使用案例

- 點選工具箱裡的帶箭頭實線 DirectedAssociation(單向結合)圖示，如圖 1-123 所示。

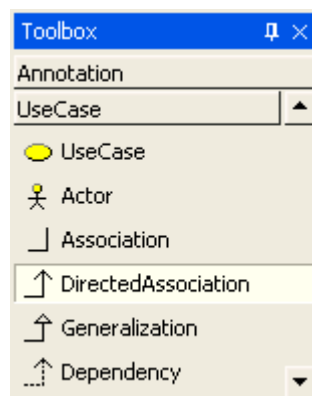


圖 1-123: 點選 DirectedAssociation

- 隨後，點選參與者並拖曳至使用案例處放開，建立起兩者之間的關係線，如圖 1-124 所示。

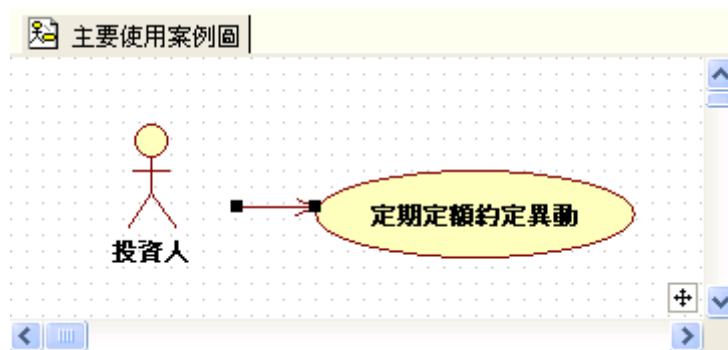


圖 1-124: 新增關係線

- 依照上述步驟，在產生另外兩個使用案例，分別命名為「網路申購單筆基金」和「發送通知電郵」，如圖 1-125 所示。

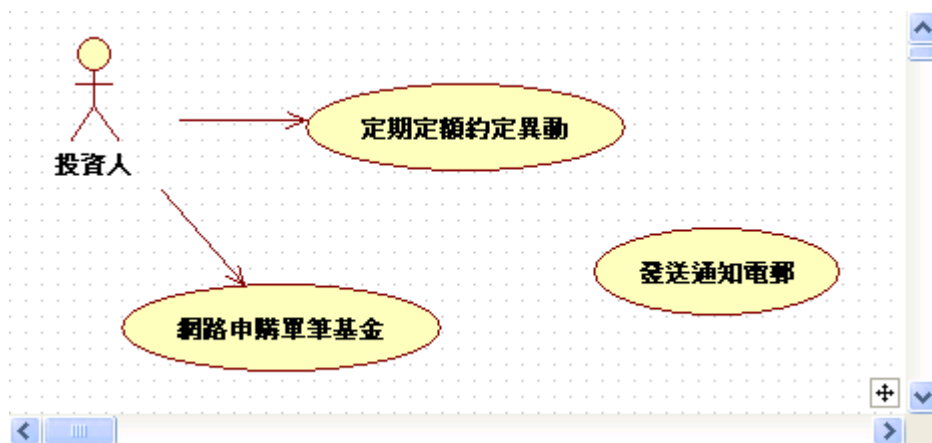


圖 1-125: 新增另兩個使用案例

8. 點選工具箱裡的帶 E 字虛線 Extend(擴充關係)圖示，如圖 1-126 所示。

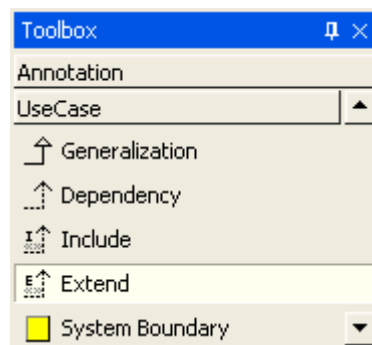


圖 1-126: 點選 Extend

9. 隨後，點選發送通知電郵並拖曳至定期定額約定異動處放開，建立出兩者之間的擴充關係線，如圖 1-127 所示。

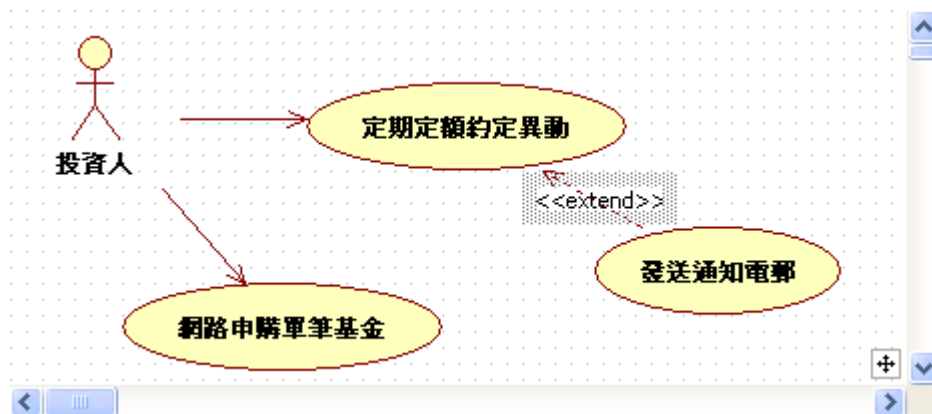


圖 1-127: 新增擴充關係

10. 點選工具箱裡的帶 I 字虛線 Include(包含關係)圖示，如圖 1-128 所示。

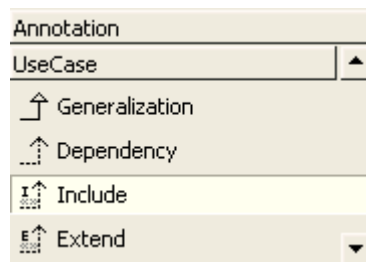


圖 1-128: 點選 Include

11. 隨後，點選網路申購單筆基金並拖曳至發送通知電郵處放開，建立出兩者之間的包含關係線，如圖 1-129 所示。

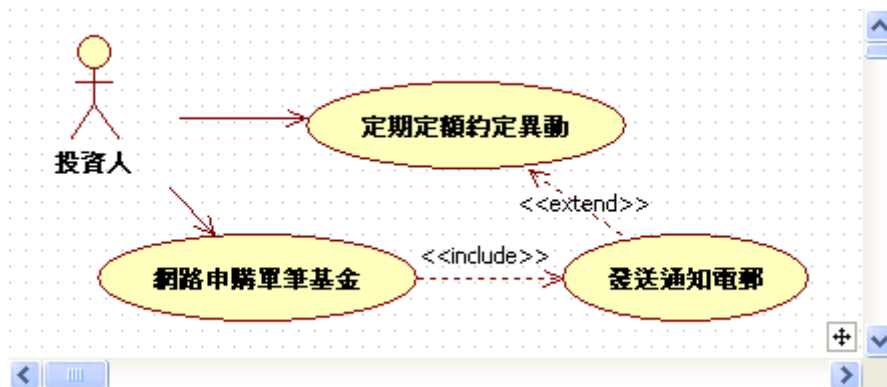


圖 1-129: 新增包含關係

Q10.4 編寫使用案例敘述

雖然，StarUML 並沒有特別為使用案例敘述設置的功能，不過只要點選使用案例圖示並打開它的文件(Documentation)頁區，還是可以於此填寫使用案例敘述，如圖 1-130 所示。若是畫面上未出現文件頁區，請勾選主選單的【View►Documentation】選項。

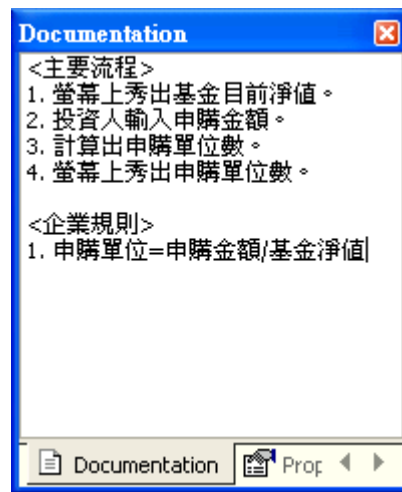


圖 1-130: 使用案例的文件頁區

除了在使用案例的文件頁區填寫使用案例敘述外，也可以在註解(note)裡填寫使用案例敘述。在 UML 裡，註解是一個通用圖示，它可以出現在 UML 所有圖款中，用來記錄備註事項。在使用案例圖裡，我們可以用它來記錄使用案例敘述。

註解圖示是一個折角矩形，像是一張便利貼，如圖 1-131 所示。而它的用途也真如便利貼，可以貼在圖面上的任何地方，以虛線連接需要註解的其它圖示。



圖 1-131: 註解圖示

新增註解的步驟，如下所述：

1. 我們之前所使用的使用案例工具箱裡，並沒有放置註解相關的工具按鍵。所以，我們要點選工具箱裡的註解(Annotation)工具箱，如圖 1-132 所示。



圖 1-132: 註解工具箱

2. 點選工具箱裡的折角矩形 Note(註解)圖示，如圖 1-133 所示。

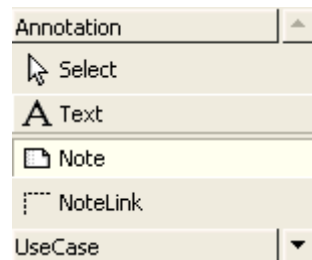


圖 1-133: 註解工具箱

3. 隨後，在圖面空白處再點一次，新增了一個註解圖示，如圖 1-134 所示。

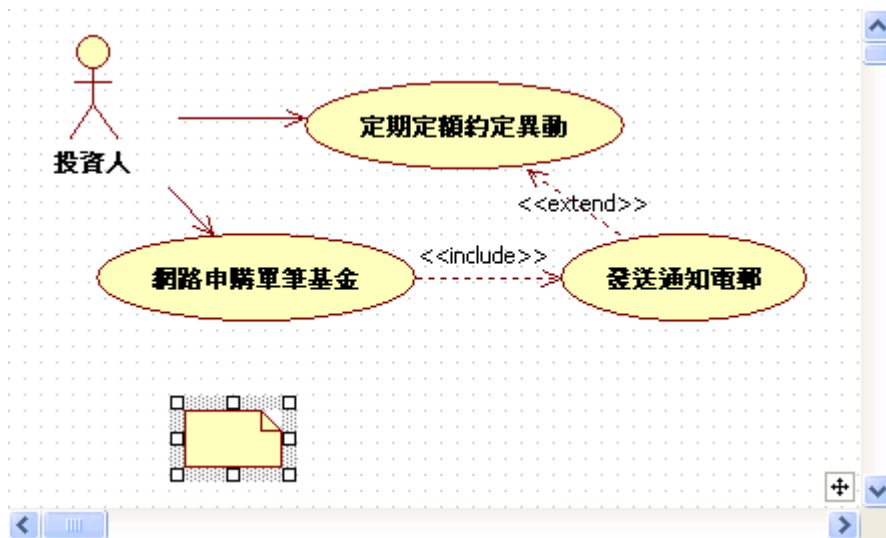


圖 1-134: 新增註解

4. 點選工具箱裡的打折虛線 NoteLink(註解接結線)圖示，如圖 1-135 所示。

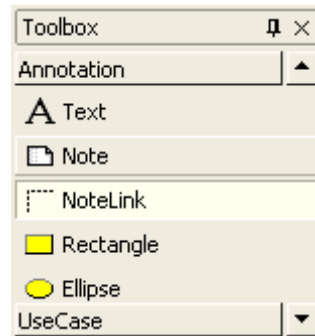


圖 1-135: 點選 NoteLink

5. 隨後，點選使用案例並拖曳至註解處放開，建立出兩者之間的關係線，如圖 1-136 所示。

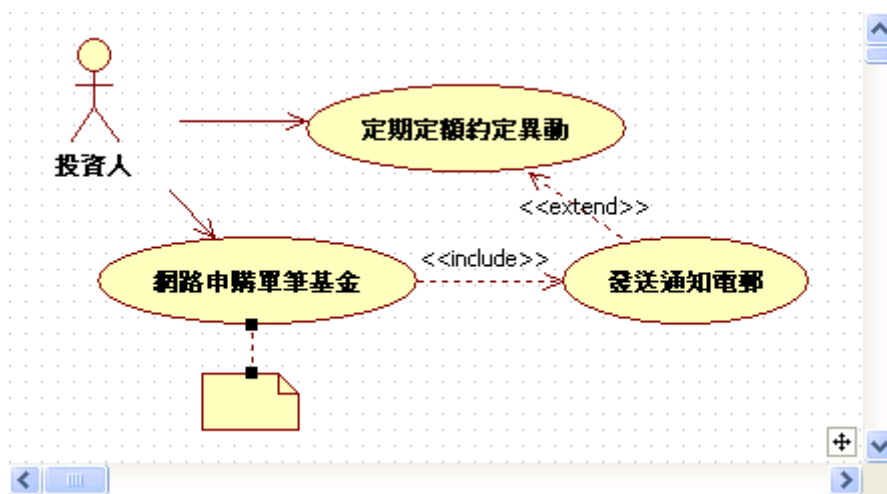


圖 1-136: 建立關係線

6. 請於註解圖示空白內部，填寫使用案例敘述，如圖 1-137 所示。

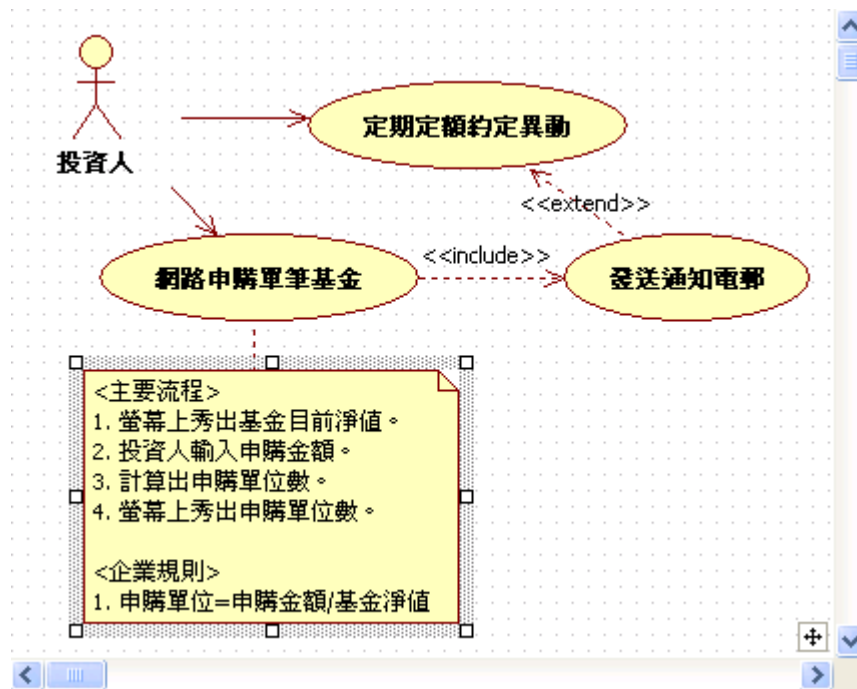


圖 1-137： 填入使用案例敘述