ATM 案例研討，第一部分：使用 UML 進行物件導向設計

25

Action speaks louder than words but not nearly as often.
—Mark Twain
Always design a thing by considering it in its next larger context.
—Eliel Saarinen
Oh, life is a glorious cycle of song.
—Dorothy Parker
The Wright brothers' design … allowed them to survive long enough to learn how to fly.
—Michael Potts

學習目標

在本章中，你將學到：

- 簡單的物件導向設計方法。
- 什麼是需求文件。
- 辨識出需求文件中的類別與類別屬性。
- 辨識出需求文件中的物件狀態、活動和操作。
- 判斷系統中的物件合作關係。
- 利用 UML 的使用案例圖、類別示意圖、狀態示意圖、活動示意圖、溝通示意圖以及順序示意圖來塑造物件導向系統的模型。

本章綱要

25.1 簡介

現在開始我們將介紹物件導向設計與實作的案例研討。本章和第 26 章中，你會設計和實作一個簡單的物件導向自動提款機 (ATM) 軟體系統。本範例爲你提供一個簡明、按部就班、完整的設計與實作體驗。我們會執行物件導向設計 (OOD) 過程裡的各種步驟，一面使用 UML，一面將這 2-13 章中所討論的物件導向概念加以結合。在本章中，你會用到六種常用的 UML 示意圖，以圖像方式來表示程式設計。在第 26 章中，我們會使用繼承進一步改良設計，然後完整地實作 850 行的 C++ 程式 (26.4 節)。

這不是一個練習，而是一個完整的學習經驗，最後將以我們設計實作出來的詳細 C++ 程式碼作爲總結。藉此讓讀者瞭解在業界中會遭遇到的形形色色問題。

當你結束第 2-13 章的物件導向程式設計課程之後，本章可視爲一個連續性的單元。你也可以在第 2-7 章以及第 9 和 13 章結束之後，以漸進的方式學習這些章節，每一節的前面都會告訴你這一節應該與哪一章相互配合。

25.2 探討 ATM 需求文件

[請注意：這一節可以接在第 2 章之後閱讀。]

我們的設計程序從**需求文件 (requirements document)** 開始，需求文件指定整個 ATM 系統的目的，以及本系統該做什麼。在本案例研討中，我們參考需求文件，判斷本系統應具備哪些功能。

需求文件

有一家地方性行庫想裝設一台新的自動櫃員機 (ATM)，讓使用者 (即行庫的客戶) 能夠進行基本的金融交易 (如圖 25.1 所示)。每位使用者只有一個該行的帳戶。ATM 的使用者必須能檢視帳戶餘額、提領款項 (就是從帳戶提款) 以及存款 (就是把錢存入帳戶)。

自動櫃員機的使用者介面包含下列硬體元件：

- 將訊息呈現給使用者看的螢幕
- 讓使用者鍵入數值資料的小鍵盤
- 發鈔票給使用者的吐鈔機
- 一個存款投入槽，接受使用者投入的存款信封

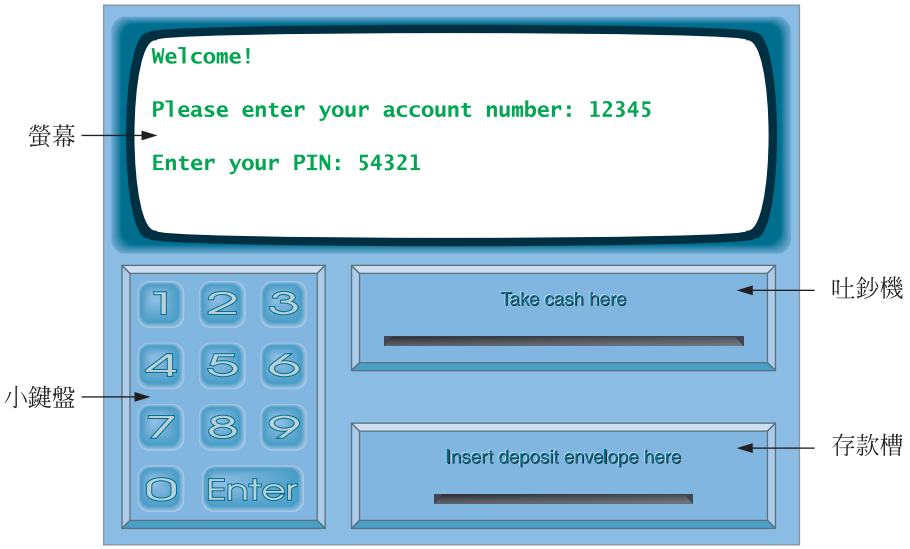

圖 25.1　自動櫃員機的使用者介面

吐鈔機每天會先放入 500 張 20 元面額的美鈔。[請注意：因爲本案例所研討的範圍有限，在此描述的某些 ATM 元件不會和眞的 ATM 一樣。例如眞正的 ATM 有讀取客戶金融卡帳號的裝置，然而我們的 ATM 卻要客戶以小鍵盤輸入帳號。還有，眞正的 ATM 在交易結束會列印明細表，但我們的 ATM 卻將所有訊息輸出到螢幕上。]

銀行想要您爲他們開發一個金融交易用的軟體，讓客戶可經由 ATM 自行使用。稍後，銀行還要將此軟體與 ATM 的硬體整合。硬體裝置的功能 (如吐鈔機、存款槽) 應該要封裝到軟體元件中，但是軟體本身不用管這些硬體裝置的運作方式。ATM 硬體還沒開發完成，因此，我們不必一開始就寫出在 ATM 上執行的程式，反而應該先開發可在個人電腦上執行的軟體作爲基本版本。這個程式版本必須利用電腦螢幕去模擬 ATM 螢幕，利用電腦鍵盤模擬 ATM 的小鍵盤。

ATM 的動作包含以帳號和個人識別碼 (PIN) 確認使用者身份 (即證明是否爲存戶本人)，然後才是新增並執行財務交易。ATM 必須和銀行的帳戶資訊資料庫互動，以驗證使用者並執行交易。[請注意：資料庫是一組存放在電腦上、有組織的資料。] 資料庫中存有每個銀行帳戶的帳號、PIN，以及表示帳戶存款總數的存款餘額。[請注意：爲了簡化，我們假設銀行僅計畫裝設一台 ATM，所以不用擔心在同一時間會有好幾台 ATM 進入資料庫存取資料的問題。再者，我們假設在 ATM 進行資料存取時，銀行內部不會對資料庫中所存的資訊進行任何的更動。此外，ATM 和所有的商務系統一樣，都要面臨相當複雜的安全性問題，這個議題遠超出 1、2 個學期的電腦科學課程所能涵蓋的範疇。無論如何，我們先簡化狀況，假設在沒有有效的安全防護下，銀行仍然非常信任 ATM 的運作，以執行資料庫的存取操作。]

當使用者開始使用 ATM 時，會經歷下列一連串的事件 (如圖 25.1 中所示)。

1. 螢幕顯示歡迎訊息，並且請使用者輸入帳號。
2. 使用者以小鍵盤輸入 5 碼帳號。
3. 螢幕顯示請使用者輸入此帳號的 PIN 碼 (個人識別碼)。
4. 使用者以小鍵盤輸入 5 碼的 PIN。
5. 如果使用者輸入有效帳號以及該帳戶的正確 PIN 碼，那麼螢幕便會顯示出主選單 (如圖 25.2 所示)。如果使用者輸入了無效的帳號或者不正確的 PIN，那麼螢幕會顯示適當訊息，然後 ATM 回到步驟 1，重新開始驗證程序。

ATM 驗證過使用者之後，主選單 (圖 25.2) 會顯示一組經過編號的選項，其中有三種交易型態：餘額查詢 (選項 1)、提款 (選項 2) 以及存款 (選項 3)。主選單亦顯示一個可讓使用者離開系統的選項 (選項 4)。接著使用者便可以選擇所需的交易 (輸入 1、2 或 3)，或者離開系統 (輸入 4)。若使用者輸入無效的選項，螢幕會顯示錯誤訊息，並重新顯示主選單。

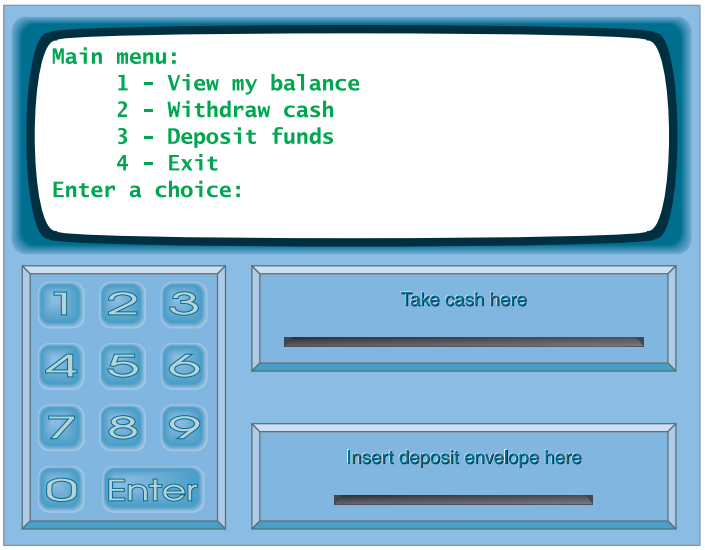

圖 25.2　　ATM 主選單

如果使用者輸入 1 進行餘額查詢，螢幕便會顯示出使用者的帳戶餘額。要完成這項工作，ATM 必須從銀行資料庫中讀取餘額資料。

當使用者輸入 2 進行提款時，會發生以下動作：

1. 螢幕顯示標準的提款金額選單 (如圖 25.3 所示)：20 美元 (選項 1)、40 美元 (選項 2)、60 美元 (選項 3)、100 美元 (選項 4) 以及 200 美元 (選項 5)。主選單也有一個選項，可讓使用者取消交易 (選項 6)。
2. 使用者以小鍵盤輸入選單選項 (1-6)。
3. 如果提款總額高於使用者的帳戶餘額，螢幕會顯示餘額不足的訊息，並且要求使用者重選一個較小的數額。若提款金額小於或等於使用者帳戶餘額 (也就是可接受的提款金額)，ATM 會進行至步驟 4。若使用者選擇取消交易 (選項 6)，ATM 會顯示主選單 (圖 25.2) 並等待使用者輸入。
4. 若吐鈔機內的鈔票足夠，ATM 會進行到步驟 5。否則螢幕會顯示訊息告知此問題，並要求使用者選擇較低的提款金額。然後會回到步驟 1。
5. ATM 從銀行資料庫的使用者帳戶餘額中扣除提款金額。
6. 吐鈔機發給使用者想要的金額。
7. 螢幕顯示取款訊息提醒使用者。

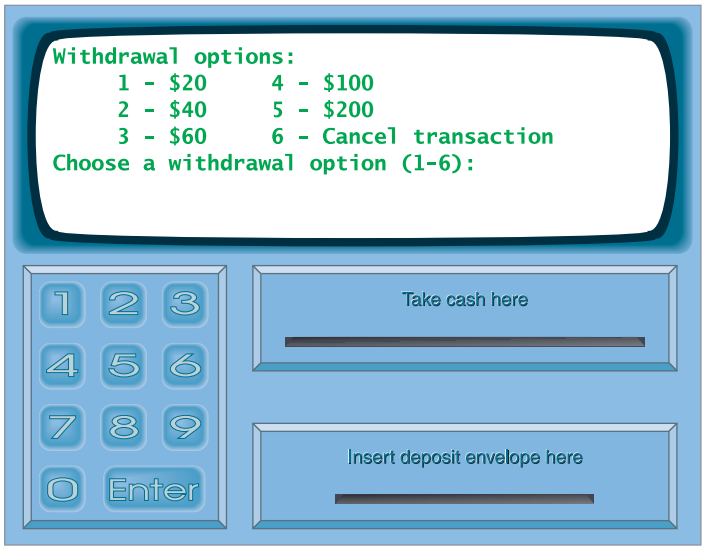

圖 25.3　ATM 提款選單

當主顯單顯示，且使用者輸入 3 進行存款時，會發生以下動作：

1. 螢幕上提醒使用者輸入存款金額，或按 0 (零) 取消此項交易。
2. 使用者以小鍵盤輸入存款數額或 0。[請注意：小鍵盤沒有小數點或者金額符號，所以使用者無法輸入帶小數的金額 (例如$1.25)。而必須以美分爲單位輸入存款金額 (好比說是 125)。然後 ATM 會用 100 去除這個數，而得到以美元表示的金額 (好比說是 125÷100＝1.25)。
3. 若使用者指定了存款金額，ATM 會進行到步驟 4。若使用者選擇取消交易 (輸入 0)，ATM 會顯示主選單 (圖 25.2) 並等待輸入。
4. 螢幕顯示出訊息，告訴使用者可將存款信封投入存款槽。
5. 若存款槽在兩分鐘之內收到存款信封，ATM 會在銀行資料庫的使用者帳戶餘額中加上提款金額。[請注意：這筆存款不可以立即提領。銀行得先實際清點存款信封內的現鈔金額，而信封內的支票也必須清算完成 (也就是錢必須從開票人的帳戶轉到收票人的帳戶中)。上述這些事件中一旦發生其中任何一件，銀行便會更新使用者在資料庫中的存款餘額。這項更新使用者資料庫存款餘額的作業與 ATM 系統無關。] 如果存款槽在時限之內沒有接收到存款信封，螢幕會顯示因爲沒有動作而取消交易的訊息。然後 ATM 顯示主選單畫面，並等待使用者的輸入。

交易成功後，系統應重新顯示主選單 (圖 25.2)，讓使用者進行其它交易。如果使用者選擇離開系統 (選項 4)，螢幕會顯示謝謝您的訊息，然後再換成歡迎訊息等待下一位使用者。

分析 ATM 系統

前面是需求文件的一個簡化案例。通常，像這樣的需求文件就是在**需求蒐集 (requirements gathering)** 的詳細過程所產生的典型結果，該過程包括與系統使用者或特定領域專家的面談。例如，負責撰寫銀行軟體 (如本 ATM 系統) 需求文件的系統分析師可能會與財務專家進行面談，確實了解系統究竟要做什麼。分析師會使用取得的資訊編出一份**系統需求 (system requirement)** 以指導系統設計師。

需求蒐集是軟體生命週期初步階段的重要工作。**軟體生命週期 (software life cycle)** 詳細指明一個軟體從最初構思到終止使用的各階段歷程。這些階段通常包括：分析、設計、實作、測試與除錯、部署、維護、與終止使用。現有幾種軟體生命週期的模型，每一種模型都有其愛用者，每一種模型也都有其規格，告訴軟體工程師何時該進行何種階段，以及多久要進行一次。**瀑布型模型 (waterfall model)** 連續地將各階段執行一次，而**反覆型模型 (iterative model)** 則會在整個產品的生命週期裡重複執行一或多個階段。

軟體生命週期的分析階段著重於釐清待解決的問題。設計任何系統時，正確的解決問題 (slove the problem right) 很重要，但解決正確的問題 (slove the right problem) 也同等重要。系統分析人員蒐集需求，該需求指出待解決的特定問題。我們的需求文件已將 ATM 系統描述得夠仔細了，您不須再進行龐雜的分析程序，我們都幫您做好了。

爲了確實掌握系統的功能，系統開發人員經常會運用**使用案例塑模 (use case modeling)** 的技術。此程序中會找出系統的**使用案例 (use cases)**，每個使用案例都代表一種系統提供給客戶的不同功能。例如，ATM 通常有數個使用案例，如「檢視帳戶餘額」、「提款」、「存款」、「轉帳」與「買郵票」。我們在此案例研討中所建構的簡化 ATM 系統，只有前面三種使用案例 (圖 25.4)。

每個使用案例描述使用者使用系統時的典型狀況。您已在需求文件中看過 ATM 系統的使用案例說明；每個交易型態 (即餘額查詢、提款與存款) 的執行步驟其實描述了三種 ATM 使用案例，也就是「餘額查詢」、「提款」與「存款」。

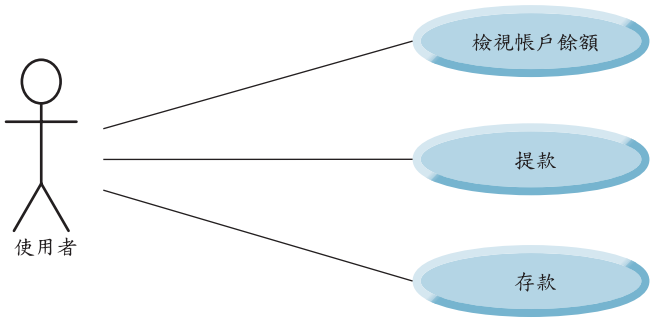

圖 25.4　使用者觀點中的 ATM 系統使用案例示意圖

使用案例示意圖

現在我們來介紹前幾種 ATM 案例研討的 UML 示意圖。我們建立**使用案例示意圖 (use case diagram)**，將系統的使用客戶 (在本案例研討中指的是銀行的客戶) 與系統間的互動建立成模型。其目標是顯示使用者與系統的互動類型，但不用顯示細節—細節會由其它 UML 示意圖提供 (我們在案例研討中會逐步介紹這些示意圖)。使用案例示意圖常常會附加一些非正式的說明文字，就像需求文件裡的文字一樣。使用案例示意圖是在軟體生命週期的分析階段產生的。在較大的系統中，使用案例示意圖是簡單但不可或缺的工具，它能夠幫助系統設計師著重在滿足使用者的需求上。

圖 25.4 是 ATM 系統的使用案例示意圖。人形圖代表**行為者 (actor)**，我們定義行爲者爲外部的實體，好比說是一個人或是另一個系統，扮演與系統互動的角色。就我們的自動櫃員機而言，行爲者就是可以從 ATM 上進行餘額查詢、提款、與存款作業的使用者。「使用者」(User) 並非是一個眞正的人，而是由眞人在與 ATM 互動時所扮演的角色所組成的。注意，一張使用案例示意圖可以包含多個行爲者。例如，一張銀行 ATM 系統的使用案例示意圖上，或許會包括一位名爲管理者的行爲者，它每天會將吐鈔機裝滿。

我們從需求文件中的這一句：「ATM 使用者可檢視帳戶餘額、提款與存款」找出系統中的行爲者。因此，上述三項使用案例的每個行爲者就是與 ATM 互動的使用者。外部實體 (眞人) 扮演著執行金融交易的使用者角色。圖 25.4 顯示一個行爲者，而他的名字「使用者」三個字則出現在圖中行爲者的下方。UML 建立每個使用案例模型的方法是將橢圓形以一條實線連到行爲者。

軟體工程師 (更精確的說，是系統分析師) 必須分析需求文件或者全套的使用案例，並且要在程式設計人員實作程式之前，先行進行系統設計。在分析階段，系統分析師會專注在理解需求文件上，以產生一份高階規格，說明系統要做什麼。設計階段的輸出，也就是**設計規格 (design specification)** 應指明系統要如何建構，以滿足這些需求。在後續幾個章節中，我們會針對 ATM 系統執行簡單的物件導向設計 (OOD) 程序，以產生含有各種 UML 示意圖與說明文字的設計規格。還記得嗎？UML 適用於任何 OOD 程序。目前業界有許多程序，最知名的是 Rational Unified Process (RUP)，由 Rational Software Corporation (現在是 IBM 的一個部門) 所開發。RUP 是一整套繁複的程序，適用於設計產業級的應用程式，本案例研討則使用我們自己簡化過的設計程序。

設計 ATM 系統

現在開始 ATM 系統的設計階段。所謂**系統 (system)**，就是一組可彼此互動以解決問題的元件。例如，爲了要執行 ATM 系統的設計工作，我們的 ATM 系統必須要有使用者介面 (如圖 25.1 所示)，這個介面含有能夠執行金融交易，並且能與銀行的帳戶資訊資料庫互動的軟體。**系統結構 (System structure)** 描述系統物件以及物件之間的關係。**系統行為 (System behavior)** 描述物件間的互動如何改變系統。 每個系統皆有結構和行爲，設計人員必須具體指定這兩者。系統結構和行爲有數種獨特形態。例如，物件間的互動與使用者和系統間的互動是不同的，但兩者共同組成系統的行爲。

UML 2 訂出 13 種示意圖型態，作爲提供系統模型文件之用。每一種示意圖分別將系統結構或行爲的不同特性加以模型化，其中六種示意圖和系統結構有關，其餘七種則與系統行爲有關。這裡只列出案例研討所用的六種示意圖，其中一種 (就是類別示意圖) 將系統結構加以模型化，其餘五種則將系統行爲加以模型化。其餘七種 UML 示意圖則會於附錄 G「UML 2：其它示意圖類型」加以簡單介紹。

1. **使用案例示意圖 (use case diagram)**：例如圖 25.4，這種圖是以使用案例 (系統性能，例如「餘額查詢」、「提款」以及「存款」) 將系統與系統外部實體 (行爲者) 之間的互動建立爲模型。
2. **類別示意圖 (Class diagram)**：在第 25.3 節中我們會解釋這種圖。類別示意圖是將系統所使用的類別或所謂的「建構區塊」(building block) 加以模型化。需求文件所描述的每一項名詞或事物，都有可能成爲系統中的類別 (例如「帳戶」、「小鍵盤」)。類別示意圖有助於我們訂定系統部件之間的結構關係。例如，ATM 系統類別示意圖會訂定出 ATM 是螢幕、小鍵盤、吐鈔機、以及存款槽的實體組合。

3. **狀態機示意圖 (state machine diagram)**：我們將會在第 25.5 節探討。這種示意圖會將物件改變狀態的各種方法加以模型化。物件的**狀態 (state)** 便是在某特定時刻該物件所有屬性的值。當物件改變狀態，該物件在系統中就可能有不同的行爲。例如，驗明某位使用者的身份證字號有效之後，ATM 會從「使用者驗證無效」的狀態，轉變到「使用者驗證有效」的狀態，此時 ATM 會允許使用者進行金融交易作業 (如餘額查詢、提款、存款)。
4. **活動示意圖 (Activity diagram)**：我們也會在第 25.5 節中介紹。它可將物件的**活動 (activity)**，也就是程式執行時的物件工作流程 (一連串事件) 加以模型化。活動示意圖會將物件所執行的活動加以模型化，並指明物件執行這些活動時的先後順序。例如，活動示意圖會指出，在螢幕顯示使用者的存款餘額畫面之前，ATM 必須先取得使用者帳戶中的存款餘額資訊 (從銀行的帳戶資訊資料庫裡面)。
5. **溝通示意圖** (**communication diagram**，早期的 UML 版本中又稱爲**「合作示意圖，collaboration diagram」**：強調互動時會發生**什麼**現象，進而將系統裡面物件之間的互動模型化。這些示意圖會顯示執行一項 ATM 交易處理時，哪些物件必須彼此互動。第 25.7 節會學到這一點。例如，ATM 必須與銀行的帳戶資訊資料庫溝通，才能取得帳戶餘額資料。
6. **順序示意圖 (Sequence diagram)**：這種示意圖也會將系統物件之間的互動加以模型化，但與溝通示意圖不一樣的地方在於，它們強調的是何時才會發生互動。在執行一項金融交易時，這些示意圖有助於顯示互動發生的先後順序。第 25.7 節可學到這一點。例如，螢幕畫面會在款項送出之前，先提醒使用者輸入提領數額。

在第 25.3 節中，我們從需求文件裡面找出相關的類別，繼續設計我們的 ATM 系統。我們會從需求文件中粹取重要的名詞或名詞片語，以找出相關類別。並運用這些類別開發出基本的類別示意圖，此類別示意圖可模型化我們的 ATM 系統結構。

資源網站

我們建立了豐富的 UML 資源中心 (`www.deitel.com/UML/`) ，你可以在這裡找到許多其他資訊的連結，包括 UML 簡介、教學文件、部落格、書籍、認證、研討會、開發工具、說明文件、電子書、FAQ、論壇、群組、C++ 的 UML、podcasts、安全性、工具、下載、訓練課程、視訊等等。

25.2 節的自我測驗

25.1 假設我們 ATM 系統中的一名使用者，能夠在二家不同的銀行帳戶之間進行轉帳。請修改圖 25.4 中的使用案例示意圖，以反映這項改變。

25.2 _____以強調**「何時」**會發生何種互動的角度，將系統物件之間的互動加以模型化。

a) 類別示意圖
b) 順序示意圖
c) 溝通示意圖
d) 活動示意圖

25.3 下列那一個選項是典型的循序型軟體生命週期階段？

a) 設計、分析、實作、測試
b) 設計、分析、測試、實作
c) 分析、設計、測試、實作
d) 分析、設計、實作、測試

25.3　找出 ATM 需求文件中的類別

[請注意：這一節可以接在第 3 章之後閱讀。]

現在我們開始設計 25.2 節所描述的 ATM 系統。本節會分析需求文件中的名詞和名詞片語，以找出建構 ATM 系統所需的類別。我們先介紹 UML 類別示意圖，以塑造這些類別之間的關係模型。爲了定義我們的系統結構，這是相當重要的起步。

辨識系統中的類別

我們藉由找出建構 ATM 系統所需的類別，開始進行 OOD。最後會用 UML 類別示意圖描述這些類別，並以 C++ 實作這些類別。首先，我們回顧第 25.2 節的需求文件，確認其中的關鍵名詞與名詞片語，幫助我們找出組成 ATM 系統的類別。我們可能會發現某些關鍵名詞與名詞片語，其實是系統中其它類別的屬性。我們也可能發現有些名詞跟系統無關，因此不該納入模型裡。而在設計過程當中，可能還會有其他的類別出現。

圖 25.5 列出需求文件中的名詞和名詞片語。我們根據它們在需求文件中出現的順序，從左到右列出。列表中一律以單數形表示。

只有對 ATM 系統有一定重要程度的名詞和名詞片語，我們才會爲它們建立類別。我們不需要建構「銀行」類別，因爲銀行並不是 ATM 系統的一部份，銀行只是委託我們建構 ATM 系統。「客戶」(Customer) 與「使用者」(user) 也是系統外部的實體，他們會與 ATM 系統互動，算是相當重要，但我們不需要在 ATM 軟體中，以類別來模擬他們。回想一下，我們曾在圖 25.4 的使用案例圖中，以行爲者 (actor) 模擬 ATM 使用者 (也就是銀行客戶)。

另外，我們也不需建立 20 元美鈔 ($ 20 bill) 跟存款信封 (deposit envelop) 類別。它們是眞實世界中的實體物件，但並非我們自動化的對象。我們可以使用一個類別來模擬吐鈔機，並用此類別中的屬性來表示系統中的鈔票。(第 25.4 節將指派各類別的屬性)。例如，吐鈔機會記錄它擁有的鈔票張數。規格中並未說明在收到存款信封後，系統該怎麼處理它們。我們可以將這個問題簡化：使用一個類別來模擬存款槽，而此類別會執行一個「已收到存款信封」的操作，這樣就足以表示信封已經存入系統中了。(在第 25.6 節將指派各類別的操作)。

需求文件中的名詞和名詞片語		
bank	money / fund	account number
ATM	screen	PIN
user	keypad	bank database
customer	cash dispenser	balance inquiry
transaction	$20 bill / cash	withdrawal
account	deposit slot	deposit
balance	deposit envelope	

圖 25.5　需求文件中的名詞和名詞片語

在這個簡單的 ATM 系統中，將各種不同的「貨幣」金額 (包括帳戶中的「存款餘額」) 表示成其它類別的屬性，似乎是最恰當的。同樣地，有一些名詞，例如「帳號」(account number)、「密碼」(PIN)，都是 ATM 系統中相當重要的資訊。它們都是銀行帳戶的重要屬性，但它們不會表現出行爲。因此將它們當作帳戶類別的屬性，是最恰當的做法。

儘管需求文件經常以一般概念來描述「交易」(transaction)，但我們不打算在此模擬廣義的金融交易。反之，我們將這三種交易 (「餘額查詢」、「提款」與「存款」) 建立成獨立的類別。這些類別擁有某些特定屬性，以執行它們所代表的交易。例如，提款需要知道使用者想提領的金額。但餘額查詢就不需額外的資料。而且，這三種交易類別各有其獨特行爲。提款類別會吐鈔給使用者，存款類別則會從使用者收取存款信封。[請注意：在第 26.3 節中，我們會採用物件導向的抽象類別與繼承觀念，抽出所有交易的共同特徵，建立成一般性的「交易」(transaction) 類別。]

我們根據圖 25.5 其餘的名詞和名詞片語，找出系統的類別，分別代表以下各項：

- ATM

- 螢幕 (screen)
- 小鍵盤 (keypad)
- 吐鈔機 (cash dispenser)
- 存款槽 (deposit slot)
- 帳戶 (account)
- 銀行資料庫 (bank database)
- 餘額查詢 (balance inquiry)
- 提款 (withdrawal)
- 存款 (deposit)

以上所列出的名詞，可能就是實作系統時所需要的類別。

現在我們可按照此名單，建立系統中的類別。依據 UML 慣例，我們在設計過程中，將類別名稱的第一個字母大寫，就跟我們撰寫 C++ 程式碼時一樣。如果類別名稱超過一個字，我們會將這些字合併在一起，並將每個字的第一個字母大寫 (如 `MultipleWordName`)。藉由此慣例，我們建立 `ATM`、`Screen`、`Keypad`、`CashDispenser`、`DepositSlot`、`Account`、`BankDatabase`、`BalanceInquiry`、`Withdrawal` 和 `Deposit` 類別。我們使用這些類別當作建構區塊，架構我們的系統。但開始建立系統前，我們必須更進一步了解類別之間的關係。

建立類別模型

UML 可讓我們用**類別示意圖 (class diagram)** 模擬 ATM 系統中類別和它們之間的關係。圖 25.6 是 ATM 類別。每個類別都以長方形表示，裡頭分三個部分。最上層是類別名稱，此名稱水平置中並採用粗體字。中間部份是類別屬性。(第 25.4 和 25.5 節會討論屬性)。最底下則是類別操作 (於第 25.6 節討論)。在圖 25.6 中，中間和底層都是空的，因爲我們尚未決定該類別的屬性與操作。

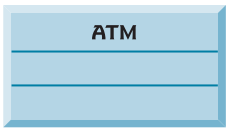

圖 25.6　使用類別示意圖表示 UML 類別

類別示意圖亦能顯示系統內各類別之間的關係。圖 25.7 顯示 ATM 類別跟 Withdrawal 類別之間的關聯。目前，我們只建立部份類別的模型以簡化說明，本節稍後會提出更完整的類別示意圖。請注意，在此圖中表示類別的長方形不再分成三層。UML 允許您在適當時候隱藏類別的屬性和操作，以提升類別示意圖的可讀性。這種圖稱爲**略圖 (elided diagram)**，在此圖中部分資訊 (譬如在中間層和底層所含的內容) 並不顯示於模型中。我們會在第 25.4 和 25.6 節將資料放入此二層。

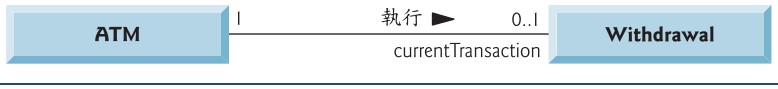

圖 25.7　類別示意圖顯示出類別間的聯繫

在圖 25.7 中，連接兩個類別的實線表示**聯繫 (association)**，是一種類別間的關係。標示在直線末端的數字就是**重數 (multiplicity)**，代表各類別有多少個物件參與這個聯繫。在本例中，沿著線從一端到另一端，可看出在任一時刻中，一個 ATM 物件都會與 0 或 1 個 `Withdrawal` 物件相聯繫。若目前使用者沒有進行交易或是進行其它交易，那麼就有零個 `Withdrawal` 物件參與，如果目前使用者要求進行提款交易，那麼就有一個 `Withdrawal` 物件參與。UML 可以模擬各種重數。圖 25.8 會列出並說明各種重數。

符號	意義
0	無
1	一個
m	整數值
0..1	零或多個
m, n	m or n
m..n	最少是 m，不超過 n
*	任何非負整數 (零或多個)
0..*	零或多個 (跟 * 意義相同)
1..*	一或多個

圖 25.8　重數種類

聯繫也可加以命名。例如，圖 25.7 中連接 ATM 類別與 `Withdrawal` 類別的線上有一個字 `Executes`，這個字就是聯繫的名稱。這部份的圖，讀作「一個 ATM 類別物件，

執行 0 或 1 個 Withdrawal 類別物件」。聯繫的名字有方向性，其方向以實心箭頭表示。例如，若將前述聯繫從右到左讀成「0 或 1 個 Withdrawal 類別物件，執行一個 ATM 類別物件」便是錯的。

在圖 25.7 中，有一個字 currentTransaction 出現在聯繫的 Withdrawal 端，這個字就是**角色名稱 (role name)**，它表示在 Withdrawal 與 ATM 的關係當中，Withdrawal 物件所扮演的角色。角色名稱確定了類別在聯繫中扮演的角色，使得類別間的聯繫產生了意義。一個類別能在系統中扮演好幾種角色。例如，在一個校園人員系統中，一個人在他與學生的關係中，扮演著「教授」的角色。而在他與另一位教授的關係中，扮演著「同事」的角色，當他指導運動選手時，便扮演起「教練」的角色了。在圖 25.7 中，角色名稱 currentTransaction 表示在 Withdrawal 物件與 ATM 類別物件的 Executes 聯繫中，Withdrawal 物件代表 ATM 目前正在處理的交易。但在其它狀況下，Withdrawal 物件可能就扮演其它角色 (例如上一筆交易)。請注意，我們在 Executes 聯繫的 ATM 端並沒有指定角色名稱在類別示意圖中，若不需角色名稱就能了解聯繫的意義，則通常會省略它。

聯繫除了可以指出簡單的關係之外，還能夠指定比較複雜的關係，譬如說某個類別物件是由其它類別物件組合而成的。想想看眞實世界裡的自動櫃員機。製造商是將哪些「零件」拼成一部可用的 ATM？需求文件告訴我們，ATM 是由一台螢幕、一具小鍵盤、一部吐鈔機、還有一個存款槽組合成的。

在圖 25.9 中，ATM 類別聯繫線末端的**實心菱形 (solid diamonds)**，表示 ATM 類別與 Screen、Keypad、CashDispenser 以及 DepositSlot 等類別之間具有**組合 (composition)** 關係。組合表示「整體/部件」(whole/part) 的關係。若組合符號 (實心菱形) 出現在聯繫線某一端，則該端的類別代表「整體」(如本例的 ATM)，而聯繫線另一端的類別就代表「部件」，例如本例的 Screen、Keypad、CashDispenser 以及 DepositSlot 等類別皆是。圖 25.9 的組合關係表示一個 ATM 類別物件，是由一個 Screen 類別的物件、一個 Keypad 類別的物件、一個 CashDispenser 類別的物件以及一個 DepositSlot 類別的物件所組合而成。所以 ATM「有」螢幕、小鍵盤、吐鈔機、還有存款槽。組合關係便是**有 (has-a)** 的關係。[第 26.3 節將會介紹，繼承是一種「是」(is-a) 的關係]。

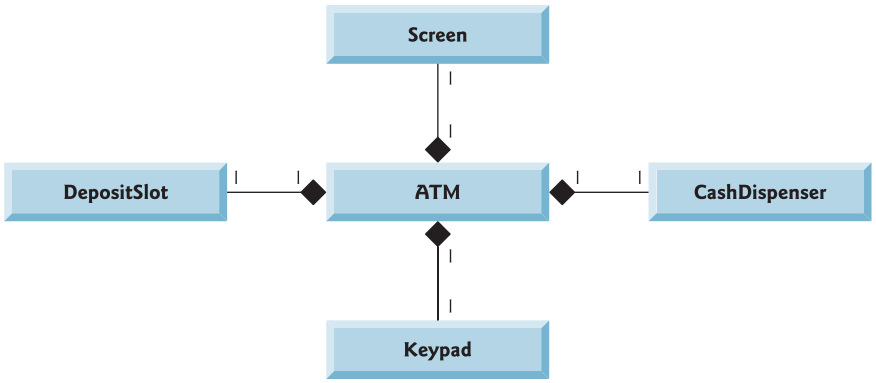

圖 25.9　顯示組合關係的類別示意圖

根據 UML 規格，組合關係具備以下特性：

1. 在組合關係中，只能有一個類別代表整體 (換句話說，菱形只能出現在聯繫線的某一端)。例如，要不就是螢幕爲 ATM 的一部分，要不就是 ATM 爲螢幕的一部分，二者只有一項能成立，絕不可能螢幕跟 ATM 二者同時代表關係中的整體。
2. 在組合關係中，只有整體存在，各部件才會存在，並由整體負責建立和移除它的部件。例如，建構一部 ATM 的行爲，就包含製造它的零件。而且，一旦移除這部 ATM，那麼它的螢幕、小鍵盤、吐鈔機、還有存款槽也都一併移除。
3. 「部件」一次只能屬於某個整體，不過「部件」能夠搬移、再掛載到另一個整體上頭，然後由此新的整體爲這個部件負責。

在我們的類別示意圖中，實心菱形代表滿足上述三項特性的組合關係。如果某個「有」的關係，無法滿足上述特性中的一或多項，依 UML 規定，在聯繫線的端末加上空心菱形表示**聚合 (aggregation)**，這是一種較弱的組合關係。舉例來說，一台個人電腦跟一台電腦螢幕便是聚合關係，因爲電腦雖然可以「有一個」螢幕，但這兩個零件能夠獨立存在，一台螢幕也可以同時連到多部電腦，因此違反前述第 2 條及第 3 條組合特性。

圖 25.10 是 ATM 系統的類別示意圖。此圖模擬了我們在本節稍早提到的大部分類別，以及由需求文件得知的各類別間聯繫。[請注意：`BalanceInquiry` 類別跟 `Deposit` 類別所參與的聯繫與 `Withdrawal` 類別相似，因此我們在圖中省略之，以保持簡潔。在第 26.3 章中，我們會將類別示意圖擴大，以涵蓋整個 ATM 系統中的類別。]

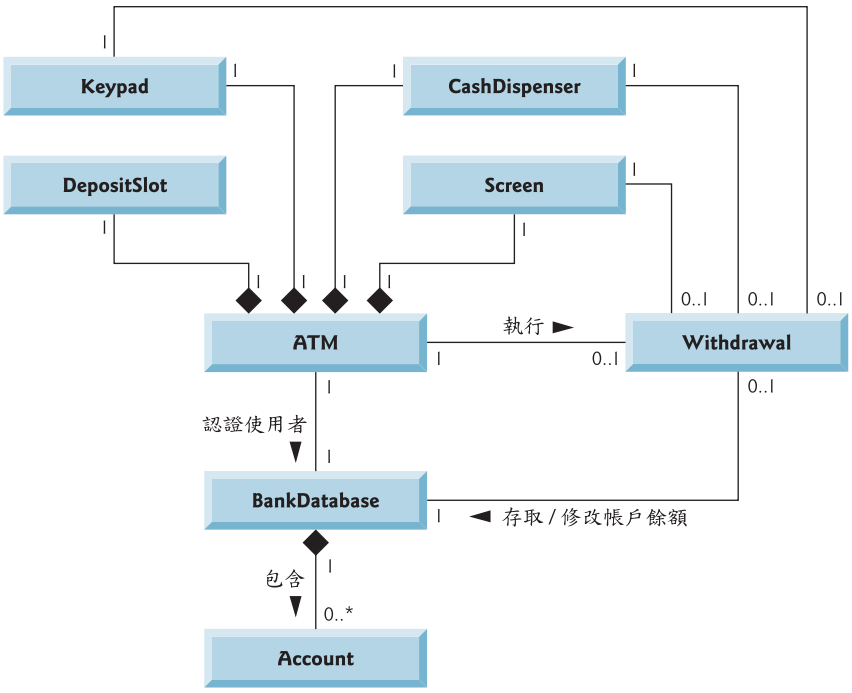

圖 25.10　ATM 系統模型的類別示意圖

圖 25.10 呈現出 ATM 系統結構的圖形模型。此類別示意圖包含 `BankDatabase` 與 `Account` 類別，以及數個未在圖 25.7 或圖 25.9 顯示的聯繫。此類別示意圖顯示，ATM 類別與 `BankDatabase` 類別間有**一對一關係 (one-to-one relationship)**，一個 ATM 物件憑藉一個 `BankDatabase` 物件來驗證使用者。在圖 25.10 中，我們也模擬了銀行資料庫中存放著許多帳戶的資訊，所以，一個 `BankDatabase` 類別物件與零至多個 `Account` 類別物件形成組合關係。請回想在圖 25.8 當中，在 `BankDatabase` 類別與 `Account` 類別間聯繫的 `Account` 那端有個重數 `0..*`，它代表有零至多個 Account 類別物件參與此組合關係。`BankDatabase` 類別跟 `Account` 類別間有**一對多關係 (one-to-many relationship)**，也就是 `BankDatabase` 包含了許多個 `Account`。同樣地，`Account` 類別跟 `BankDatabase` 類別之間有**多對一的關係 (many-to-one relationship)**，也就是多個 `Account` 位在 `BankDatabase` 裡頭。[請注意：請回想圖 25.8 中，重數 `*` 跟 `0..*` 是相同的。我們把 `0..*` 納入類別示意圖以表達得更明確。]

圖 25.10 也指出，當使用者進行提款時，「一個 Withdrawal 類別物件會透過一個 BankDatabase 類別物件，來存取或變更某個帳戶裡面的存款餘額」。我們可以在 Withdrawal 類別和 Account 類別之間直接建立一個聯繫。不過，需求文件寫的是「ATM 必須跟銀行的帳戶資訊資料庫互動」才能進行交易。銀行帳戶裡頭包含許多機密資料，因此設計系統時，系統工程人員一定要時時考慮到個人資料的安全性。因此，只有 BankDatabase 類別能夠直接存取並操作運用帳戶，系統的所有其它部份，都必須透過與資料庫的互動，才能存取或更新最新的帳戶資訊 (例如帳戶餘額)。

圖 25.10 的類別示意圖也建立了 Withdrawal 與 Screen、CashDispenser 和 Keypad 類別之間的聯繫。一項提款交易中，包括提示使用者選擇提款金額以及接收使用者輸入的數字。這些動作分別會用到螢幕和小鍵盤。吐鈔給使用者這個動作則會用到吐鈔機。

此外，圖 25.10 裡頭沒有顯示出來的 BalanceInquiry 類別和 Deposit 類別，也參與了這個 ATM 系統中與其它類別的聯繫。跟 Withdrawal 類別相似，這些類別都各與 ATM 類別及 BankDatabase 類別有聯繫。爲了顯示使用者帳戶的存款餘額，BalanceInquire 類別物件也與 Screen 類別物件有聯繫。Deposit 類別則是跟 Screen、Keypad 以及 DepositSlot 等類別有聯繫。與提款交易類似，存款交易也用到螢幕與小鍵盤，以顯示提示語以及接收使用者輸入的資料。Deposit 類別物件能存取存款槽，以收取存款信封。

現在，我們已找出 ATM 系統的類別 (不過在設計與實作過程中，可能還會找到更多類別)。在第 25.4 節中，我們會找出每個類別的屬性，接著在第 25.5 節裡頭，我們利用這些屬性來檢視這個系統如何隨時間而改變。在第 25.6 節中，我們會決定系統類別的操作。

25.3 節的自我測驗

25.4 假設我們有一個代表汽車的 Car 類別。請想想看，製造廠商會用哪些不同零件組合一輛完整的汽車。請繪出一張類別示意圖 (仿照圖 25.9) 模擬與 Car 類別相關的組合關係。

25.5 假設我們有一個代表電子文件的 File 類別，這個電子文件儲存在一部獨立、未連接至網路的電腦中，電腦以 Computer 類別表示。請問 Computer 類別與 File 類別之間是哪一種聯繫？

a) Computer 類別對於 File 類別有一對一的關係。

b) Computer 類別對於 File 類別有多對一的關係。

c) Computer 類別對於 File 類別有一對多的關係。

d) Computer 類別對於 File 類別有多對多的關係。

25.6 說說看下面的敘述是否正確，如果不正確，請解釋其理由：在一個 UML 圖中，若各類別的第二層與第三層均未顯示於模型中，則此圖稱爲略圖。

25.7 修改圖 25.10 的類別示意圖，以 `Deposit` 類別取代 `Withdrawal` 類別。

25.4　找出類別屬性

[請注意：這一節可以接在第 4 章之後閱讀。]

在第 25.3 節中，我們開始爲 ATM 系統進行物件導向設計 (OOD) 的第一階段，也就是分析需求文件，並確認該系統實作時所需的類別。我們列出需求文件中的名詞與名詞片語，並爲那些在 ATM 系統有明顯重要性的名詞跟名詞片語，各制定出一個類別。然後我們以 UML 類別圖，建立類別以及類別間關係的模型 (圖 25.10)。

類別都擁有屬性 (資料) 與操作 (行爲)。類別屬性會實作成 C++程式的資料成員，而類別操作會實作爲成員函式。本節會找出 ATM 系統所需的諸多屬性。第 25.5 章將介紹如何以這些屬性表示物件狀態。我們會在第 25.6 章決定類別的操作。

找出屬性

請想想眞實世界裡的物件屬性：人的屬性包括身高、體重，以及這個人是左撇子、右撇子、還是左右開弓。收音機的屬性包括電台設定、音量設定以及 AM、FM 設定。汽車屬性包括時速表與里程表讀數、油量以及打哪一檔。個人電腦的屬性包括製造商 (如 Dell、Sun、Apple 或 IBM)、螢幕類型 (LCD 或 CRT)、主記憶體大小與硬碟容量。

我們可從需求文件中找出描述性的字詞和片語，辨識出系統裡各類別的屬性，若覺得它對 ATM 系統有明顯重要性，便將它定爲屬性，並指定給第 25.3 節找出的某一個或數個類別。我們也可建立新屬性以表示類別所需的其它資料，隨著設計的進行，需要性也更加清楚。

圖 25.11 列出需求文件中描述各類別的字詞及片語。我們閱讀需求文件、找出所有涉及系統中類別特性的字詞和片語，整理成這張表。例如，需求文件描述提取某一「提款金額」(withdrawal amount) 的所需步驟，所以我們將「amount」列在 `Withdrawal` 類別旁邊。

圖 25.11 引導我們制定出 ATM 類別的一項屬性。`ATM` 類別掌管 `ATM` 的狀態資訊。片語「使用者通過驗證」是一種 `ATM` 狀態的描述 (我們將在 25.5 節介紹「狀態」)，因此我們在 `ATM` 類別加入 `userAuthenticated` 屬性，型別爲 **`Boolean` 屬性** (**Boolean attribute**，此屬性的值不是 `ture` 就是 `false`)。UML 的 `Boolean` 型別就等於 C++ 的

bool 型別。此屬性表示 ATM 是否已驗證目前這位使用者，對於系統而言，userAuthenticated 屬性必須爲 true，使用者才能執行交易和存取帳戶資訊。這個屬性有助於確保系統資料安全。

類別	描述性文字和片語
ATM	使用者通過驗證
BalanceInquiry	帳號
Withdrawal	帳號 金額
Deposit	帳號 金額
BankDatabase	[無描述性文字或片語]
Account	帳號 密碼 餘額
Screen	[無描述性文字或片語]
Keypad	[無描述性文字或片語]
CashDispenser	每天一開始放入 500 張 20 元鈔票
DepositSlot	[無描述性文字或片語]

圖 25.11 ATM 需求中的描述性文字和片語

BalanceInquiry、Withdrawal 以及 Deposit 等類別共享一個屬性。所有交易都會用到該使用者的「帳號」(account number)。我們將一個整數屬性 accountNumber 指定給所有的交易類別，代表類別物件所用的帳號。

由需求文件中的描述性字詞和片語，也可看出各交易類別所需屬性的差異。需求文件指出，不論是提款或存款，使用者都必須先輸入提款或存款的金額 (amount)。因此我們將 amount 屬性指定給 Withdrawal 類別與 Deposit 類別，以儲存使用者提供的數值。提款與存款的金額，都是系統執行該交易時所需要的交易特性。不過 BalanceInquiry 類別執行作業時就不需額外的資料，它只需要帳號以分辨要查詢的是哪個帳戶的餘額。

Account 類別含數個屬性。根據需求文件可知，每個帳戶都有帳號 (account number) 跟密碼 (PIN)，系統會用它們識別帳戶並驗證使用者。我們將二個整數屬性指定給

Account 類別：accountNumber 屬性跟 pin 屬性。需求文件也說，帳戶中記錄著該帳戶的餘額 (balance)，還有，在銀行確認過存款信封內的現金總數、或是支票兌現前，使用者不能提領存入的款項。但帳戶仍須記錄使用者存入的該筆款項。因此，我們決定帳戶應使用兩個 UML Double 型別屬性來代表餘額。也就是 availableBalance 屬性跟 totalBalance 屬性。availableBalance 屬性記錄使用者可從帳戶提領的金額。totalBalance 屬性則是使用者「存入」的金額 (也就是可提領的總金額，加上待確認或結清的款項總額)。例如有一位 ATM 使用者在一文不剩的帳戶裡存了 50 美元，則 totalBalance 屬性就會增加到 50 美元以記錄這次存款，不過 availableBalance 屬性依舊是 0 美元。[請注意：我們假設 ATM 交易發生後，銀行會很快地更新 Account 的 availableBalance 屬性，以反應「確定在存款信封收到 50 美元的現金或支票」這件事。我們假設此更新動作，是由某位銀行行員利用 ATM 以外的其它銀行軟體完成的。因此本案例研討不討論這個動作。]

CashDispenser 類別有一個屬性。需求文件中說，吐鈔機「每天會先放入 500 張 20 元面額的美鈔。」。吐鈔機必須記錄內含鈔票的張數，以判斷是否有足夠現金因應提款需求。我們將整數屬性 count 指派給 CashDispenser 類別，並且將其初值設定成 500。

在業界所遇到的實際問題中，需求規格書並不一定得夠詳細、精確，好讓物件導向系統設計者能據此確定所有的屬性、甚至類別。隨著設計進行，會找到更多需要加入的類別、屬性、以及行爲。我們在案例研討過程中，也會持續加入、修改和刪除系統中類別的相關資訊。

建立屬性的模型

圖 25.12 類別示意圖中，列出系統中各類別的一些屬性，圖 25.11 的描述性字詞及片語，可幫助我們確認這些屬性。爲了簡化，圖 25.12 並未顯示出類別之間的聯繫，這些聯繫請見圖 25.10。在開發設計的過程中，系統設計者常會採取這種做法。第 25.3 節講過，在 UML 當中，類別的屬性放在類別方框的中間層。我們列出每個屬性的名稱跟型別，並在它們之間以冒號 (:) 分隔，有時後面也會跟著一個等號 (=) 與初始值。

看看 ATM 類別中的 userAuthenticated 屬性：

```
    userAuthenticated : Boolean = false
```

此屬性宣告裡包含了有關這項屬性的三份資料。**屬性名稱 (attribute name)** 是 userAuthenticated。**屬性型別 (attribute type)** 是 Boolean。在 C++ 中，屬性可

用基本型別表示，如 `bool`、`int` 或 `double`，也可用類別表示。我們選擇只模型化圖 25.12 中的基本型別屬性—稍後會解釋原因。[請注意：圖 25.12 列出屬性的 UML 資料型別。實作系統時，我們會將 UML 型別 `Boolean`、`Integer` 與 `Double` 分別換成 C++ 的基本型別 `bool`、`int` 和 `double`。]

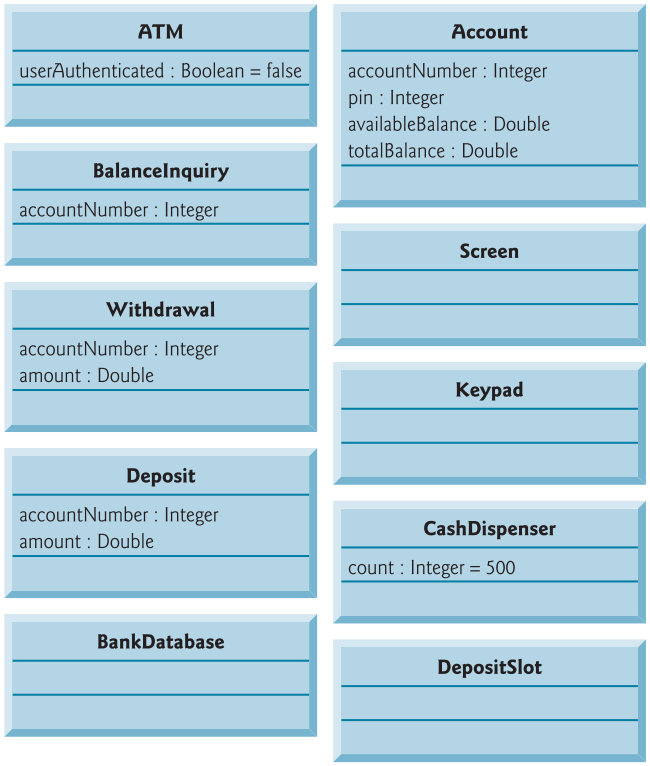

圖 25.12　擁有各種屬性的類別

我們也可以爲屬性指定初始值。`ATM` 類別的 `userAuthenticated` 屬性初始值爲 `false`，這表示系統一開始並不將使用者視爲有權限。如果某一屬性並未給定初始值，那麼圖中就只會顯示這個屬性的名稱和型別 (以冒號隔開)。例如 `BalanceInquiry` 類別的 `accountNumber` 屬性是 `Integer`。在此未顯示出初始值，因爲我們還不知道這個屬性的值是多少。這個值會在執行時期根據 ATM 使用者輸入的帳號決定。

圖 25.12 並不包括 `Screen`、`Keypad` 以及 `DepositSlot` 等類別的屬性。這些類別都是系統中的重要元件，但目前爲止，我們的設計程序還沒發現這些類別該有哪些屬

性。不過在後續的設計階段或以 C++ 實作這些類別的時候，可能還會發現某些屬性。這對軟體工程的反覆程序而言是十分正常的。

軟體工程的觀點 25.1

在設計過程的初期階段，類別通常沒有屬性 (與操作)。但我們不能省略這些類別，因爲在後續設計與實作階段中，這些類別的屬性 (與操作) 可能就會明朗化。

圖 25.12 也沒有 `BankDatabae` 類別的屬性。之前曾提過，屬性可用基本型別或類別代表。圖 25.12 的類別示意圖 (以及本案例研討的其它類似類別示意圖中) 只加入基本型別的屬性。至於類別形態的屬性，會用一條聯繫 (association，特別是組合，composition) 將「擁有此屬性的類別」以及「代表此屬性的類別」連接起來，以清楚的將此關係模型化。舉例來說，圖 25.10 的類別示意圖表示 `BankDatabase` 類別與 0 個到多個 `Account` 類別擁有組合關係。根據此組合關係，我們便知道用 C++ 撰寫 ATM 系統時，需要爲 `BankDatabase` 類別建立屬性，以存放 0 到多個 `Account` 物件。同樣的，我們會指派屬性給 ATM 類別，此類別對應到與 `Screen`、`Keypad`、`CashDispenser` 和 `DepositSlot` 的組合關係。如果在圖 25.12 的模型中也加入這些組合關係的屬性，就是多餘的了，因爲在圖 25.10 模型中的組合關係，就已表達出資料庫中包含 0 到多個帳戶，以及 ATM 由銀幕、小鍵盤、吐鈔機、存款槽組合而成的事實。軟體開發者通常會將整體/部份 (whole/part) 關係視爲組合關係，而非實作此關係所需的屬性。

圖 25.12 的類別示意圖爲我們的模型結構提供良好基礎，但這個圖還不夠完整。在 25.5 節中，我們將確認出模型中物件的狀態與活動，並在 25.6 節確認這些物件執行的操作。隨著 UML 與物件導向設計的進一步介紹，我們也會持續強化此模型結構。

25.4 節的自我測驗

25.8 想要確認出系統當中的類別屬性，典型的作法就是分析需求文件裡頭的_____。

a) 名詞與名詞片語。
b) 描述性的字詞及片語。
c) 動詞與動詞片語。
d) 以上皆是。

25.9 下列何者不是飛機的屬性？

a) 長度。
b) 翼展。
c) 飛行。
d) 座位數。

25.10 以下爲圖 25.12 類別圖中 `CashDispenser` 類別的屬性宣告，試描述其意義。

```
count : Integer = 500
```

25.5　找出物件狀態和活動

[請注意：這一節可以接在第 5 章之後閱讀。]

在第 25.4 節中，我們找出了許多實作 ATM 系統時所需要的屬性，並將這些屬性加進圖 25.12 的類別示意圖當中。本節說明這些屬性如何表現出物件的狀態。我們爲物件找出一些可能的關鍵狀態，並且描述物件如何隨系統中所發生的各種事件而改變其狀態。我們還會討論 ATM 系統中各種物件所進行的工作流程，或稱之爲**活動 (activities)**。本節介紹 `BalanceInquiry` 和 `Withdrawal` 交易物件的活動，它們負責 ATM 系統的兩項重要活動。

狀態機示意圖 (State Machine Diagram)

系統中各物件都會歷經一系列的狀態。物件的狀態 (state) 便是在某特定時刻該物件屬性的值。**狀態機示意圖** (**State machine diagrams**；一般稱之爲**狀態示意圖**，**state diagram**) 可用來建立某個物件各種狀態的模型，並且顯示出在何種情況下，物件會變成何種狀態。狀態示意圖與前幾節實例研究中所介紹的類別示意圖不同，後者著重於系統的結構，而狀態示意圖表現的是系統行爲的模型。

圖 25.13 是一張簡單的狀態示意圖，它是 ATM 類別物件的某些狀態的模型。UML 在狀態示意圖中，將狀態表示成一個**圓角矩形 (rounded rectangle)**，並且在矩形內填入狀態名稱。而連著一個箭頭的**實心圓 (solid circle)** 則代表**初始狀態 (initial state)**。請回想一下，我們在圖 25.12 的類別示意圖中，以 `Boolean` 屬性 `userAuthenticated` 模擬此狀態資訊。根據狀態示意圖得知，該屬性初始化成 `false`，亦即「使用者無權限」狀態。

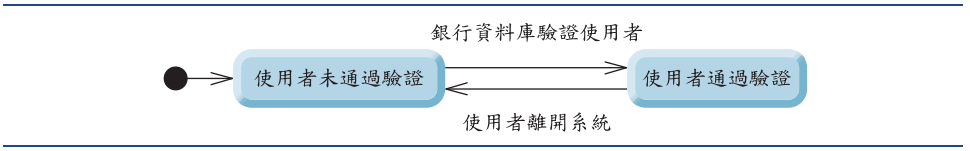

圖 25.13　ATM 物件的狀態示意圖

圖中的箭號表示狀態間的**轉換 (transition)**。一個物件可隨著系統所發生的種種事件，從某一種狀態轉換到另外一種狀態。導致轉換的事件，其名稱或描述便放在代表該轉換的流程線附近。例如，在資料庫確認使用者有權限之後，ATM 物件會從「使用者無權限」狀態變成「使用者有權限」狀態。請回想在需求文件中所述，資料庫會將使用者輸入的帳號與密碼，與資料庫中帳戶的相關資料進行比對，以判斷該使用者有無權限。假如資料庫表示使用者所輸入的帳號有效、密碼也正確，那麼 ATM 物件便會轉換成「使用者有權限」狀態，並將 `userAuthenticated` 屬性值改變爲 `true`。當使用者從主選單選擇「exit」離開系統時，ATM 物件就回到「使用者無權限」狀態，準備給下一位 ATM 使用者操作。

軟體工程的觀點 25.2

軟體開發人員通常不會將所有可能狀態、以及所有屬性的狀態轉換畫成狀態示意圖，因爲數量太多了。狀態示意圖通常只顯示最重要或最複雜的狀態與狀態轉換。

活動示意圖 (Activity Diagram)

與狀態示意圖類似，活動示意圖 (activity diagram) 可用來建立系統行爲的模型。但跟狀態示意圖不同的地方在於，活動示意圖表現的是程式執行期間，物件的工作流程 (workflow，事件的順序) 模型。活動示意圖描述的是物件執行何種活動，以及執行這些活動的先後順序。回想一下，我們曾用 UML 活動示意圖介紹第 4 章和本章中控制敘述的控制流程。

圖 25.14 建立了執行 `BalanceInquiry` 交易的動作模型。我們先假設 `BalanceInquiry` 物件已經初始化，並且已指派一個有效的帳號 (目前使用者的帳號)，因此 `BalanceInquiry` 物件會知道要檢索哪一個帳戶的餘額。圖中所示活動，包括使用者從主選單上選擇存款餘額查詢之後、ATM 回到主選單畫面之前的活動，其中 `BalanceInquiry` 物件並不會執行或初始化這些活動，因而在此並不會包含於模型中。此圖一開始會從資料庫取得使用者帳號的可用餘額。接著 `BalanceInquiry` 會取得此帳號的總餘額。最後，此交易在螢幕上顯示餘額。在這個活動之後，交易就完成了。

UML 在活動示意圖中的動作，是以一個左右兩側爲向外彎曲弧形的矩形來表示動作狀態。每個動作狀態包含一個動作運算式，此運算式會指定要執行的動作，例如「從使用者帳戶取得可用餘額」。箭頭符號連接著二個動作，顯示出動作狀態所代表的動作的發生順序。實心圓 (在圖 25.14 的最上面) 表示活動的初始狀態，也就是工作流程的開頭，此時物件尚未開始執行模型所示之各樣活動。在這裡，此交易會先執行「從資料庫取得使用者帳戶的可用餘額」動作運算式。接著，此交易取得總餘額。最後，此交易在

螢幕上顯示兩種餘額。一個空心圓包著一個實心圓 (在圖 25.14 的最下面) 的符號表示結束狀態，亦即工作流程的結束，此時物件已將模型所示活動執行完畢。

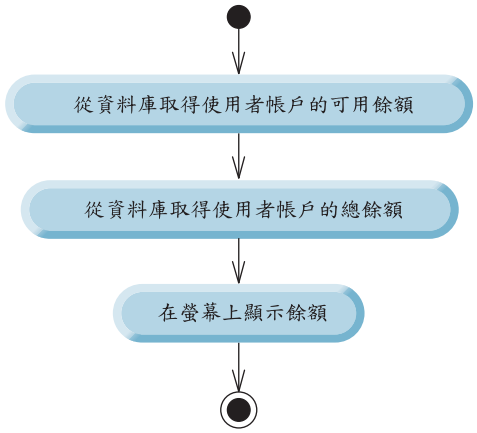

圖 25.14　`BalanceInquiry` 交易的活動示意圖

圖 25.15 爲 `Withdrawal` 交易的活動示意圖。我們假設已指定一個有效帳號給 `Withdrawal` 物件。在此模型中，不包括使用者從主選單上選取提款交易，或是 ATM 回到主選單，因爲這些動作都不是由 `Withdrawal` 物件所執行的。此交易會先顯示一個標準的提款金額選單 (圖 25.3) 及一個取消交易選項。接著，此交易會接收使用者選擇的選項。動作流程現在碰到判斷符號了。此點會根據相關的檢查條件決定下一個動作。若使用者取消交易，系統會顯示適當訊息。接著，取消流程會碰到一個合併符號，在這裡，此流程會和本交易中其它可能的活動流程 (稍後會討論) 進行合併。一個合併符號可有任意數量的轉移箭號指進來，但只有一個指出去的轉移箭號。此圖底部的判斷可決定交易是否要從頭再來一次。當使用者取消交易，檢查條件「已吐鈔或使用者取消交易」就是 true，所以控制轉移到活動的最終狀態。

如果使用者從選單上選了一項提領金額，`Withdrawal` 物件會將 `amount` 變數 (原本在圖 25.12 模型中的一項屬性) 設定成使用者所選定的金額。接著，此交易會從資料庫取得使用者帳戶的可用餘額 (就是使用者的 `Account` 物件的 `availableBalance` 屬性)。接下來，活動流程碰到另一個判斷符號。若提款金額大於使用者帳戶餘額，系統會顯示適當錯誤訊息，將錯誤告知使用者。接著，控制會合併到一個活動流程上，再流進本圖最底下的判斷符號。檢查條件「未吐鈔且使用者未取消交易」爲 `true`，所以活動流程回到本圖最上方，交易會提示使用者輸入新的提款金額。

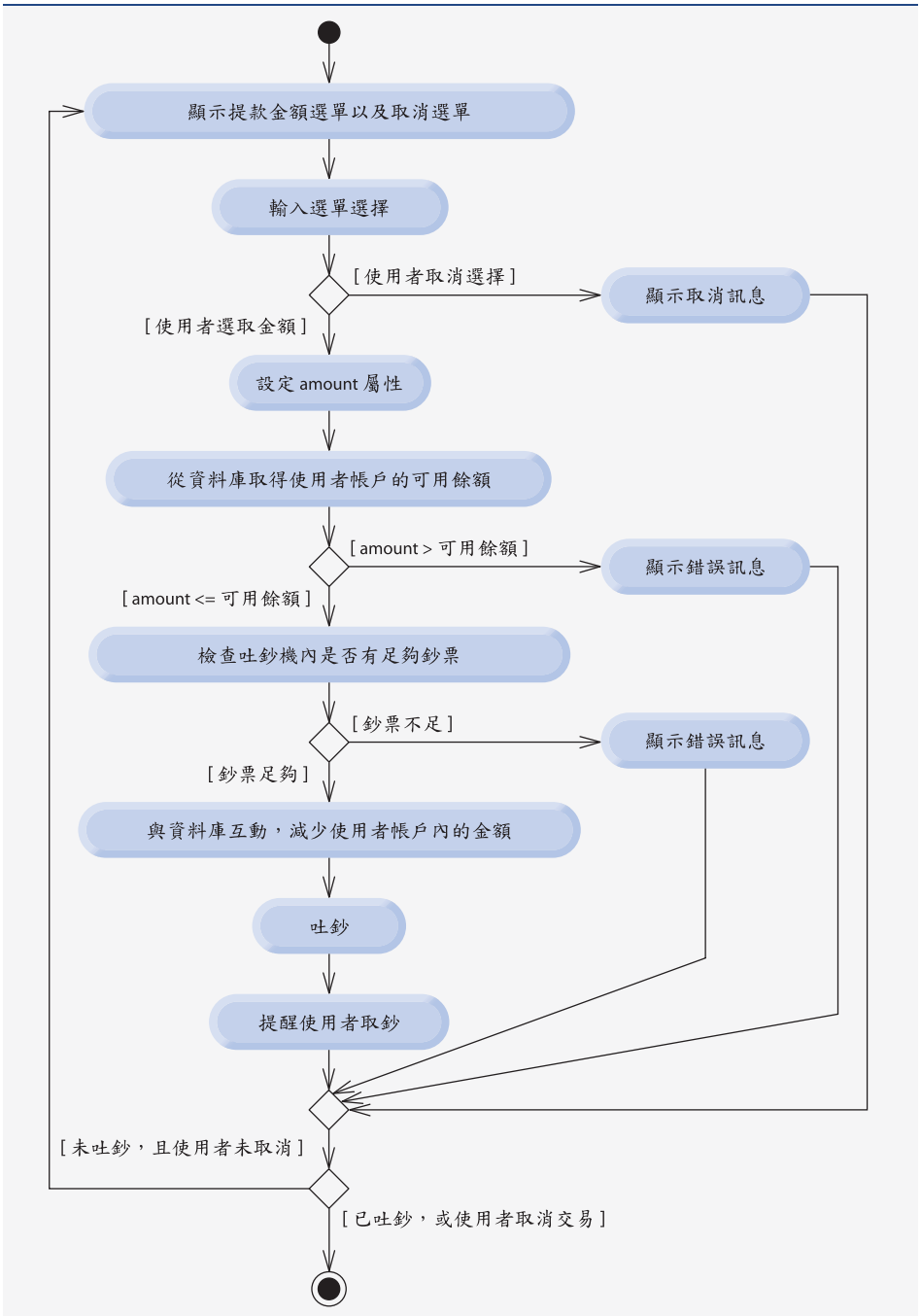

圖 25.15　`Withdrawal` 交易的活動示意圖

若提款金額小於或等於使用者帳戶餘額，交易會檢查吐鈔機內是否有足夠鈔票，以滿足提款需求。若鈔票不足，此交易會顯示適當錯誤訊息，並直接跳到最終判斷符號之前的那個合併符號。由於未吐鈔，因此活動流程回到此活動示意圖起點，交易會請使用者輸入新的提款金額。如果有足夠的現金，此交易便會跟資料庫互動，在使用者帳戶內記錄提領金額 (也就是將 `Account` 物件的 `availableBalance` 屬性與 `totalBalance` 屬性皆減去提領金額)。接下來，交易便送出所欲提領的現金，並且指示使用者取款。活動主流程接著會和兩條錯誤流程與一條取消流程合併。由於此時已吐鈔，所以活動流程會走到最終狀態。

我們已踏出建立 ATM 系統行爲模型的第一步，並說明了物件屬性如何參與物件活動的進行。在 25.6 節，我們會探究類別中的操作，以建立更完整的系統行爲模型。

25.5 節的自我測驗

25.11 說說看下面的敘述是否正確，如果不正確，請解釋其理由：利用狀態示意圖可建立系統結構方面的模型。

25.12 利用活動示意圖可建立物件執行_____的模型，還有物件執行它們的先後順序。

a) 動作
b) 屬性
c) 狀態
d) 狀態轉移

25.13 請根據需求文件，建立存款交易的活動示意圖。

25.6 找出類別的操作

[請注意：這一節可以接在第 6 章之後閱讀。]

在 25.3-25.5 節中，我們以物件導向的設計方法，完成了 ATM 系統設計的前幾個步驟。在 25.3 節裡，我們判定了稍後必須實作的類別，並且畫出第一個類別示意圖。在 25.4 節裡，我們描述類別的屬性。在 25.5 節裡，我們檢視物件的狀態，並且爲物件的狀態、狀態的轉變及其各式活動建立模型。在這一節，我們將判定實作 ATM 系統所需的類別操作 (或行爲)。

找出操作

操作是類別物件提供給客戶 (使用者) 的服務。以眞實世界的物件爲例，收音機的操作是設定電台和音量 (由使用者調整收音機控制鈕)。汽車的操作包括加速 (駕駛踩下加速

踏板)、減速 (駕駛踩下煞車或釋放油門)、轉彎及排檔。軟體物件也可以提供操作，例如一個軟體圖形物件可能提供畫圓、畫直線、畫方形等操作。試算表軟體物件提供列印試算表、加總整行或整列的數字、用長條圖或圓餅圖畫出試算表資訊之類的操作。

藉由檢視需求文件中的關鍵動詞或動詞片語，我們可以從各類別找出許多的操作。接下來，就可將這些操作一指定給系統中的類別 (圖 25.16)。圖 25.16 列出的動詞片語，可幫助我們判定各類別的操作。

類別	動詞與動詞片語
ATM	執行金融交易
BalanceInquiry	[未於需求規格書出現]
Withdrawal	[未於需求規格書出現]
Deposit	[未於需求規格書出現]
BankDatabase	認證使用者，取得帳戶餘額，在帳戶中貸記存款金額，在帳戶中借記提款金額
Account	查詢帳戶餘額，將存款金額存入帳戶，從帳戶中扣除提款金額。
Screen	顯示訊息給使用者
Keypad	取得使用者輸入的數字
CashDispenser	吐出現鈔，指出機內現金是否能滿足欲提款項
DepositSlot	接受存款信封

圖 25.16　ATM 系統中各類別的動詞和動詞片語

建立操作的模型

我們檢視圖 25.16 列出的動詞片語，找出各類別的操作。ATM 類別的「執行金融交易」(execute financial transactions)，意思是 ATM 類別需要執行 (execute) 各樣交易，因此，`BalanceInquiry`、`Withdrawal`、`Deposit` 類別各需要一項操作，以便爲 ATM 提供這項服務。我們把這個操作 (命名爲 execute) 放在更新後的圖 25.17 中，三個交易類別的第三層。在 ATM 的服務交談期 (session) 中，ATM 物件會呼叫每個交易物件的 execute 操作。

UML 表示這些操作 (在 C++中實作爲成員函式) 的方式是，先列出操作名稱，其後跟著以小括號括住、以逗號隔開的參數列表，然後再加上冒號和傳回值型別，如下所示：

operationName(*parameter1*, *parameter2*, …, *parameterN*) : *return type*

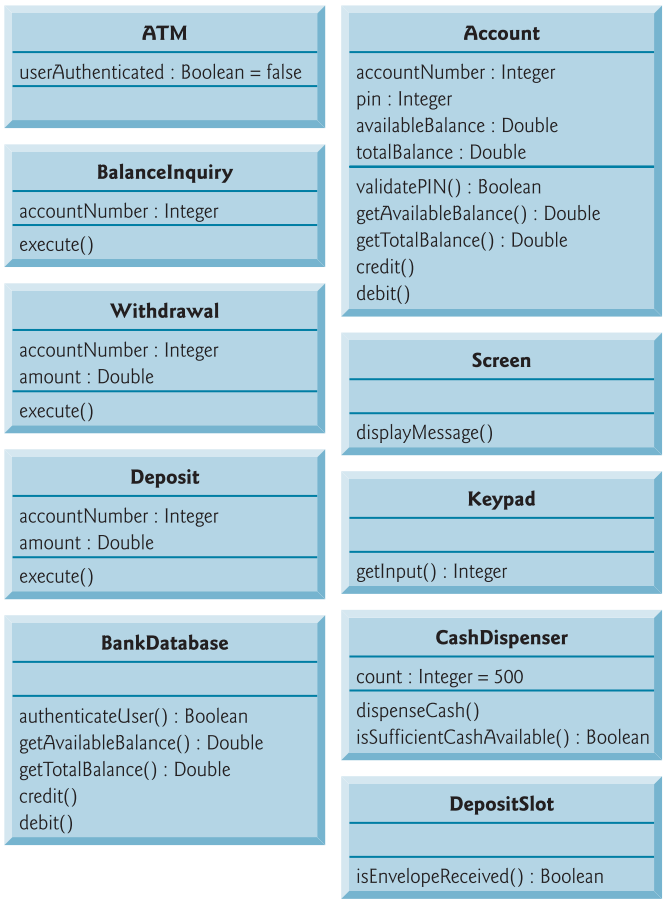

圖 25.17　ATM 系統的類別與其屬性和操作

參數列中以逗點隔開的各個參數，都有參數名稱接著冒號和參數型別，如下所示：

```
parameterName : parameterType
```

此處我們還沒有列出操作所需的參數。我們會在稍後把它們加入模型中。有些方法的傳回值型別暫時還不知道，因此在圖中暫時省略，在此階段，這是很正常的。在稍後的設計和實作過程中，我們會加入剩下的傳回值型別。

BankDatabase 與 Account 類別的操作

圖 25.16 的 BankDatabase 類別旁邊列出「認證使用者」，代表它是包含帳戶資訊的物件，並且需要判斷使用者輸入的帳號和密碼是否與銀行的帳戶資訊相符。因此，BankDatabase 類別需要一個操作，以便爲 ATM 提供認證服務。我們把 authenticateUser 操作放在 BankDatabase 類別的第三層，如圖 25.17 所示。然而，儲存使用者認證所需之帳號密碼的，並不是 BankDatabase 類別物件，而是 Account 類別，所以 Account 類別必須提供服務，以便根據 Account 物件儲存的密碼判別使用者輸入的密碼是否有效。因此，我們爲 Account 類別增加一個 validatePIN 操作。authenticateUser 和 validatePIN 操作的傳回型別爲 Boolean，代表這個操作是成功完成任務 (傳回 true)，或是失敗 (傳回 false)。

圖 25.16 列出更多與 BankDatabase 類別有關的動詞片語，如：「讀取帳戶餘額」、「在帳戶中貸記存款金額」及「在帳戶中借記提款金額」等。與「使用者認證」類似，這些動詞片語代表 BankDatabase 必須提供給 ATM 的服務，因爲它存有使用者認證及執行 ATM 交易所需的所有帳戶資料。不過，實際執行這些操作的是 Account 類別。所以，我們將各片語相對應的每個操作，同時指派給 BankDatabase 類別和 Account 類別。我們在第 25.3 節討論過，銀行帳戶是敏感資訊，故不允許 ATM 直接存取帳戶。因此，我們讓資料庫扮演 ATM 和帳戶資料之間的中介者，防止未經授權的存取。我們在第 25.7 節會討論到，ATM 類別會呼叫 BankDatabase 類別的操作，而這些操作又各自呼叫 Account 類別中與之同名的操作。

由「讀取帳戶餘額」這個片語可知，BankDatabase 類別和 Account 類別都需要 getBalance 操作。不過，我們已經在 Account 類別建立了兩個屬性 availableBalance 和 totalBalance 來代表餘額。餘額查詢時這兩個餘額屬性都要存取，才能顯示給使用者看，可是提款只需要檢查 availableBalance 的值。爲了讓系統中的物件能夠各別取得餘額屬性，我們在 BankDatabase 和 Account 類別的第三層增加 getAvailableBalance 和 getTotalBalance 操作 (圖 25.17)。我們指定這些操作的傳回值型別爲 Double，因爲取得的餘額屬性是 Double 型別。

根據「在帳戶中貸記存款金額」和「在帳戶中借記提款金額」這兩個片語，BankDatabase 和 Account 類別必須分別在存款和提款時，執行更新帳戶的操作。因此，我們把 credit 和 debit 操作指派給 BankDatabase 和 Account 類別。讀者可能記得，(存款時) 在帳戶貸記只會增加 totalBalance 屬性的值。另一方面，(提款時)

在帳戶借記則兩個餘額屬性都會減少。我們將這些實作細節隱藏在 Account 類別中，這是封裝和資料隱藏的良好範例。

如果這是眞實的 ATM 系統，BankDatabase 和 Account 類別還必須提供一組操作，讓另一個銀行系統可以在確認或是拒絕存款的全部或一部分之後，更新帳戶餘額。例如，confirmDepositAmount 操作可增加 availableBalance 屬性的值，讓存入的基金可供提領。rejectDepositAmount 操作會將 totalBalance 屬性減少，減少的金額就是最近透過 ATM 存款並且已加入 totalBalance，可是在存款信封裡卻沒有看到的金額。銀行發現使用者沒有放入正確的現金數量或是支票無法兌現 (即跳票)時，就會叫用這個操作。這些操作會讓我們的系統更完整，可是我們在類別示意圖或實作中並沒有加入，因爲這已超出我們的案例研究範圍。

類別 Screen 的操作

在 ATM 的服務過程中，Screen 類別會在不同場合透過 ATM 的螢幕，「顯示訊息給使用者看」。需求文件描述了許多使用者在螢幕上會看到的訊息，如歡迎訊息、錯誤訊息、致謝訊息等等。需求文件還要求，螢幕要能顯示提示訊息和選單給使用者。其實，提示訊息只是一段訊息，告訴使用者下一步要輸入什麼，而選單就本質上來說也是一種提示訊息，由一連串訊息 (即選項) 所組成。因此，我們沒有爲各種訊息、提示訊息、選單的顯示，一一指派操作給 Screen 類別，而是建立一個單一操作，依參數顯示各種不同的訊息。我們把這項操作 (displayMessage) 放入類別示意圖 Screen 類別的第三層，如圖 25.17 所示。現在先不必擔心這個操作的參數爲何，我們會到本節稍後再建立這個參數的模型。

類別 Keypad 的操作

依據圖 25.16，Keypad 類別旁邊「讀取使用者輸入的數字」這段描述，Keypad 類別應執行 getInput 這項操作。ATM 的小鍵盤與一般電腦鍵盤不同，它只有數字 0 到 9，因此我們指定這個操作會傳回一整數。依需求文件所述，在不同情況下，使用者會輸入各種數字，如帳號、密碼、選單選項號碼、以分 (cent，0.01 元) 爲單位的存款金額等。Keypad 類別只負責由使用該類別的客戶取得數值，並不判斷這個值是否符合特定的規範。使用這項操作的類別必須自行判斷輸入的數字是否合法，若否，則透過類別 Screen 顯示錯誤訊息。[請注意：我們實作這個系統時，是用電腦鍵盤模擬 ATM 的小鍵盤，而

且爲了簡化問題，我們假設使用者不輸入非數字鍵，因爲 ATM 的小鍵盤上並沒有這些鍵。我們會在本書稍後討論如何檢查輸入是否符合特定的型別。]

類別 CashDispenser 和 DepositSlot 的操作

在圖 25.16 中，類別 CashDispenser 的動作是「吐出現鈔」。爲此，我們建立 dispenseCash 這項操作，並將其列在圖 25.17 的 CashDispenser 類別之下。CashDispenser 類別還必須「指出機內現金是否能滿足欲提款項」，所以，我們在 CashDispenser 類別加入 isSufficientCashAvailable，這個操作會傳回 UML 的 Boolean 型別。繼續看圖 25.16，DepositSlot 類別有「接受存款信封」這項需求。存款槽必須表示它有沒有收到信封，所以我們把 isEnvelopeReceived 操作放在 DepositSlot 類別的第三層，這個操作會傳回 Boolean 值。[請注意：眞正的硬體存款槽會送信號給 ATM，表示已經收到信封。爲模擬這個行爲，我們在 DepositSlot 類別中設置一個操作，讓 ATM 類別呼叫，以判斷存款槽有沒有收到信封。]

類別 ATM 的操作

到目前爲止，我們沒有列出任何 ATM 類別的操作，因爲我們還不曉得 ATM 類別要提供系統中其他類別什麼服務。等到我們進一步進行實作時，該類別和其他類別的更多操作會一一浮現。

找出操作的參數並建立模型

到目前爲止，我們還沒考慮到操作的參數；我們只把焦點放在每個類別的基本操作爲何。現在，讓我們更進一步看看操作的參數。要找出操作的參數，必須先知道操作完成工作所需的資料爲何。

首先考慮 BankDatabase 類別的 authenticateUser 操作。該操作的工作內容是，利用使用者提供的帳號和密碼，進行身份認證。爲此，我們讓 authenticateUser 操作接收整數參數 userAccountNumber 和 userPIN，以便和資料庫中 Account 物件的帳號密碼做比對。我們在參數名稱的字首加上「user」，避免操作的參數名稱和 Account 類別的屬性名稱混淆。我們在圖 25.18 的類別示意圖中加入這些參數，作爲 BankDatabase 類別的模型。[請注意：在類別示意圖中只爲一個類別建立模型是很正常的。在本例中，我們關心的是這個類別的參數，所以暫時忽略其他類別。在稍後案例研討的類別示意圖中，參數不再是我們關心的重點，到時我們會忽略參數以節省空間。可是請記住，這些操作仍然是有參數的。]

在 UML 中，參數的表示法是：在參數列裡以逗點隔開，而且在參數名稱後面接寫冒號和參數型別。所以圖 25.18 的意思是，`authenticateUser` 操作接收兩個參數，`userAccountNumber` 和 `userPIN`，而它們都是是整數型別。以 C++ 進行實作時，會以 `int` 表示這些參數。

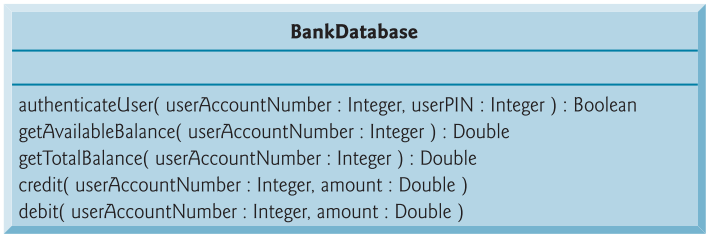

圖 25.18　BankDatabase 類別和操作參數

`BankDatabase` 類別的 `getAvailableBalance`、`getTotalBalance`、`credit`、`debit` 等操作，也都需要 `userAccountNumber` 參數，以替資料庫提供帳號資訊。爲此，我們在圖 25.18 的類別示意圖中加入這些參數。此外，`credit` 和 `debit` 操作還需要一個型別爲 `Double` 的參數 `amount`，分別代表貸記和借記的金額。

圖 25.19 的類別示意圖顯示類別 `Account` 的操作參數模型。`validatePIN` 這項操作只需要一個 `userPIN` 參數，該參數包含使用者輸入的密碼，用來和帳戶的密碼做比對。和類別 `BankDatabase` 一樣，類別 `Account` 的 `credit` 及 `debit` 操作也需要一個型別爲 `Double` 的參數 `amount`，用來代表交易金額。`Account` 類別的 `getAvailableBalance` 和 `getTotalBalance` 操作不需額外資料，就可以完成工作。類別 `Account` 的操作不需帳戶號碼作爲參數，這是因爲這些操作只會被特定的 `Account` 物件所呼叫。

圖 25.20 的模型是 Screen 類別以及有一個參數的 displayMessage 操作。這個操作只需一個 String 參數 message，代表要顯示的文字。請回憶在類別示意圖中，參數型別均以 UML 語法表示，所以圖 25.20 中的 String 型別是指 UML 型別。等到以 C++進行實作時，我們會用 C++的 string 物件代表這個參數。

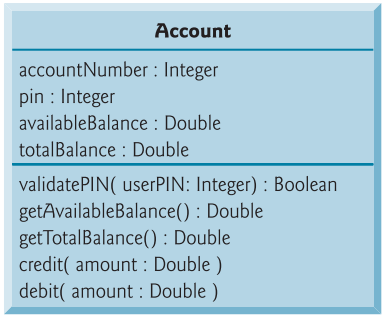

圖 25.19　Account 類別和操作參數

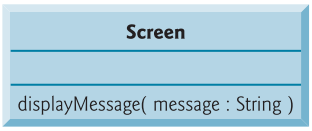

圖 25.20　Screen 類別和操作參數

圖 25.21 的類別示意圖指出 CashDispenser 類別的 dispenseCash 操作接收一個 Double 參數 amount，代表要吐出的現金金額（以元爲單位）。isSufficientCashAvailable 操作也接收一個 Double 參數 amount，代表查詢的現金金額。

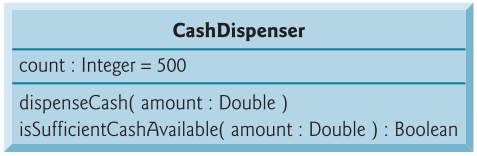

圖 25.21　CashDispenser 類別和操作參數

我們暫時先不討論類別 BalanceInquiry、Withdrawal 及 Deposit 的 execute 操作，類別 Keypad 的 getInput 操作，以及類別 DepositSlot 的 isEnvelopReceived 操作的參數。設計流程進行到目前爲止，我們還不能確定這些操作是否還需要額外的資料以完成工作，所以就先空著。我們會在後面的章節繼續進行案例研討，到時就會陸續爲這些操作增加參數。

在這一節裡，我們定義了這個 ATM 系統中、許多類別的操作，以及其中一些操作的參數和傳回型別。接下來，我們也許會更動每個類別的操作數目：可能發現需要加入新的操作，或某些操作不再需要；或者某些類別的操作需要加入更多參數和改變回傳型別。

25.6 節的自我測驗

25.14 下列哪個不是行爲？

a) 從檔案讀取資料
b) 列印輸出
c) 文字訊息
d) 取得使用者輸入資料

25.15 如果你想爲 ATM 系統增加一個操作，該操作傳回 Withdrawal 類別的 amount 屬性，如何在圖 25.17 的類別示意圖表示出這個操作？應該表示在什麼地方？

25.16 下面是一個計算機的物件導向類別示意圖操作敘述，試述其意義爲何。

```
add( x : Integer, y : Integer ) : Integer
```

25.7　找出物件間的合作關係

[請注意：這一節可以接在第 7 章之後閱讀。]

在本節中，我們把焦點放在 ATM 系統中物件的合作與互動。當兩個物件彼此溝通以完成一件工作時，稱之爲**合作 (collaborate)**；這藉由彼此呼叫操作而達成。在**合作關係 (collaboration)** 中，類別會傳遞**訊息 (message)** 給另一個類別的物件；在 C++裡訊息傳遞以成員函式呼叫的方式呈現。

我們在第 25.6 節決定了系統中各種類別的許多操作；接下來，我們把焦點放在使用這些操作的訊息。要找出合作關係，讓我們回顧第 25.2 節的需求文件。該文件描述 ATM 系統在每次交談期 (session) 中需進行哪些工作，如使用者認證，執行交易等等。這些步驟描述系統該如何進行各項工作，由這些描述，我們可以初步找出系統內容的合作關係。在本節接下來的部分，我們會帶領讀者逐步找出更多的合作關係。

判定系統中的合作關係

需求文件規定 ATM 在使用者認證和完成每種交易時所需進行的工作，因此只要仔細研讀，便可找出系統中的合作關係。依據需求規格書描述的每個動作或步驟，我們可決定系統中哪些物件必須彼此互動，以達到所需的結果。我們首先判定哪些物件是發送物件

(sending object，送出訊息的物件)，哪些是接收物件 (receiving object，爲類別用戶提供操作的物件)。接下來，選擇接收物件的一個操作 (請見第 25.6 節)，讓發送物件呼叫該操作，以便產生適當的行爲。舉例來說，ATM 於閒置時會顯示歡迎訊息。我們知道，類別 `Screen` 的物件可透過其操作 `displayMessage` 顯示訊息。因此，我們讓 `Screen` 和 ATM 合作，讓 ATM 透過呼叫類別 `Screen` 的 `displyMessage` 操作，來顯示歡迎訊息。[請注意：從現在開始，我們會直接稱呼類別名稱來代表「某類別的物件」，以免一再重複。

圖 25.22 列出需求文件指出的合作關係；我們按需求文件提到的順序，列出每個發送物件的合作關係。針對每個合作關係，雖然此合作關係在單一 `ATM` 交談期可能會發生許多次。發送者、訊息、及接收者只會列出一次，例如，圖 25.22 的第一列指出，每當 `ATM` 需要對使用者顯示訊息時，`ATM` 會與 `Screen` 合作。

類別物件	傳送訊息…	至類別物件 …
ATM	displayMessage getInput authenticateUser execute execute execute	Screen Keypad BankDatabase BalanceInquiry Withdrawal Deposit
BalanceInquiry	getAvailableBalance getTotalBalance displayMessage	BankDatabase BankDatabase Screen
Withdrawal	displayMessage getInput getAvailableBalance isSufficientCashAvailable debit dispenseCash	Screen Keypad BankDatabase CashDispenser BankDatabase CashDispenser
Deposit	displayMessage getInput isEnvelopeReceived credit	Screen Keypad DepositSlot BankDatabase
BankDatabase	validatePIN getAvailableBalance getTotalBalance debit credit	Account Account Account Account Account

圖 25.22　ATM 系統中的合作關係

考慮圖 25.22 中的合作關係。ATM 在允許使用者操作任何交易之前，要求使用者輸入帳號，然後輸入密碼。爲了要完成這些動作，它會傳送 displayMessage 訊息給 Screen。這兩個動作代表 ATM 和 Screen 之間的同一種合作，如圖 25.22 所示。ATM 藉傳送 getInput 訊息給 Keypad 取得使用者對提示訊息的回應。接下來，ATM 必須判斷使用者輸入的帳號和密碼是否與資料庫中的記錄相符；這可藉傳送 authenticateUser 訊息給 BankDatabase 達成。請記得，BankDatabase 不能直接進行使用者認證：只有使用者的 Account (即含有使用者所提供帳號的 Account 物件) 才能存取密碼，以進行使用者認證。圖 25.22 列出 BankDatabase 傳送 validatePIN 訊息給 Account 時的合作關係。

使用者認證完畢後，ATM 會送出一串 displayMessage 訊息給 Screen 以顯示主選單，並送出 getInput 訊息讓 Keypad 讀取使用者輸入。我們已經討論過這二個合作關係了。使用者選擇欲進行的交易種類之後，ATM 會把 execute 訊息送給相對應的交易類別，如 BalanceInquiry、Withdrawal 或 Deposit。例如，若使用者要查詢餘額，則 ATM 會傳送 execute 訊息給 BalanceInquiry。

繼續閱讀需求文件，我們會發現更多各種交易當中的合作關係。BalanceInquiry 藉由送 getAvailableBalance 訊息給 BankDatabase，取得使用者帳戶的可用餘額；而 BankDatabase 也藉傳送 getAvailableBalance 訊息給 Account 取得該資訊。同樣的，BalanceInquiry 藉由傳送 getTotalBalance 訊息給 BankDataBase，取得帳戶總額，而 BankDataBase 也傳送相同的訊息給使用者的 Account。爲顯示這兩個數字，BalanceInquiry 傳送 displayMessage 訊息給 Screen。

Withdrawal 傳送一連串 displayMessage 訊息給 Screen，來顯示預設提款金額選單 ($20、$40、$60、$80、$100 和$200)。Withdrawal 傳送 getInput 訊息給 Keypad，取得使用者選擇的項目。接著 Withdrawal 判斷提取金額是否小於等於使用者帳戶的餘額；Withdrawal 可藉傳送 getAvailableBalance 訊息給 BankDatabase，查詢使用者帳戶中的可用餘額。接著 Withdrawal 傳送 isSufficientCashAvailable 訊息給 CashDispenser，檢查吐鈔機中的現金是否足夠。Withdrawal 傳送 debit 訊息給 BankDAtabase，減少使用者的帳戶餘額。BankDatabase 接下來傳送相同的訊息給對應的 Account 物件。請注意，對 Account 進行借記 (debiting) 時會同時減少 totalBalance 和 availableBalance 的值。Withdrawal 傳送 dispenseCash 訊息給 CashDispenser 以吐出現金。最後，Withdrawal 傳送 displayMessage 訊息給 Screen，提示使用者拿取現金。

`Deposit` 回應 `execute` 訊息時，首先傳送 `displayMessage` 訊息給 `Screen`，提示使用者輸入存款金額。然後，`Deposit` 傳送 `getInput` 訊息給 `Keypad` 讀入使用者輸入的資料。`Deposit` 接著傳送 `displayMessage` 訊息給 `Screen`，提示使用者放入存款信封。要判斷存款槽是否收到信封，`Deposit` 傳送 `isEnvelopeReceived` 訊息給 `DepositSlot`。`Deposit` 接下來再傳送 `credit` 訊息給 `BankDatabase`，更新使用者的帳戶資訊，而 `BankDatabase` 隨後也會傳送同樣的訊息給對應的 `Account` 物件。請注意，貸記 `Account` 只會增加 `totalBalance` 的值，不會增加 `availableBalance` 的值。

互動示意圖

到目前爲止，我們已經在 ATM 系統中，判定一群物件之間可能進行的合作關係。這些互動關係可以用 UML 來畫出模型。UML 提供了數種**互動示意圖 (interaction diagram)**，可用來畫出物件彼此間的互動，建立系統行爲的模型。**溝通示意圖 (communication diagram)** 主要強調有哪些物件參與合作關係。[請注意：在 UML 早期的版本中，溝通示意圖稱爲合作示意圖 (collaboration diagram)] 與溝通示意圖類似的**順序示意圖 (sequence diagram)** 也呈現物件之間的合作關係，但主要強調的重點在於，隨著時間推移，訊息在物件之間傳送的時間點爲何。

溝通示意圖

圖 25.23 的溝通示意圖顯示 ATM 執行 `BalanceInquiry` 時的樣子。在 UML 中，物件以矩形表示，中間填有「物件名：類別名」。本例中，每個型別只有一個物件，所以我們省略物件名稱，只列出冒號和類別名。[請注意：溝通示意圖中出現多個型別相同的物件時，最好替每個物件取不同的名稱。]互相合作的物件以實心線相連；而訊息順著箭號顯示的方向，在物件之間沿實心線傳遞。箭號旁邊的訊息名稱，是接收物件的某項操作 (即成員函式) 名稱；你可以把它想成接收物件提供給發送物件 (其「客戶」) 的「服務」。

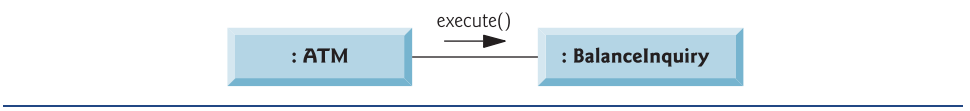

圖 25.23　ATM 執行餘額查詢的溝通示意圖

圖 25.23 裡的實心箭號代表訊息，在 UML 中稱爲**同步呼叫 (synchronous call)**，其實就是 C++ 的函式呼叫。該箭號代表控制方向是由發送物件 (ATM) 往接收物件 (BalanceInquiry)。同步呼叫的意思是，在接收物件處理完該訊息並將控制權傳回發送物件之前，發送物件不能再傳送其他的訊息或做其它事情，只能等待。例如在圖 25.23 中，ATM 呼叫 `BalanceInquiry` 的成員函式 `execute` 之後，在 `execute` 完成並且將控制權交回給 ATM 之前，不能傳送其他訊息。[請注意：在**非同步呼叫**中 (**asynchronous call**，以細箭頭表示)，發送物件不用等待接收物件交回控制權；發送非同步呼叫後便可立刻傳送其他訊息。在 C++中，非同步呼叫可利用由編譯器所提供、與平台相關 (platform-specific) 的函式庫實作之。相關技巧超出本書的範圍]。

溝通示意圖中的訊息傳送順序

圖 25.24 的溝通示意圖顯示執行 `BalanceInquiry` 的物件時，系統中的物件互動模型。這裡假設物件的 `accountNumber` 屬性已設定爲目前使用者的帳號。圖 25.24 中的合作關係始於 ATM 傳送 `execute` 訊息給 `BalanceInquiry` 之後 (和圖 25.23 一樣)。訊息名稱左邊的數字表示訊息的傳遞順序；溝通示意圖中的**訊息傳送順序 (sequence of messages)** 是依數字順序，從小到大依次傳遞。本圖中，最小的號碼是 1，最大的是 3。首先 `BalanceInquiry` 傳送 `getAvailableBalance` 訊息給 `BankDatabase` (訊息 1)，然後傳送 `getTotalBalance` 訊息給 `BankDatabase` (訊息 2)。我們可以把以逗號分隔的參數名稱，放在訊息名稱後面的括號裡，這些參數會跟著訊息一起被傳送 (即 C++ 函式呼叫中的引數)。`BalanceInquiry` 將 `accountNumber` 屬性與訊息一併傳送給 `BankDatabase`，以判斷要取得哪個 `Account` 的餘額資訊。請回想在圖 25.18 中，`BankDatabase` 類別的操作 `getAvailableBalance` 和 `getTotalBalance` 都需要一個參數以辨別帳戶。`BalanceInquiry` 接著顯示 `availableBalance` (可用餘額) 和 `totalBalance` (帳號總額)的值給使用者，這可藉傳送 `displayMessage` 訊息給 `Screen` (訊息 3，參數代表要顯示的訊息) 而達成。

請注意圖 25.24 模型中從 `BankDatabase` 傳送到 `Account` 的兩個額外訊息 (message `1.1` 和 `2.1`)。依 message `1` 和 message `2` 的要求，爲提供 ATM 二種使用者餘額，`BankDatabase` 必須傳送 `getAvailableBalance` 與 `getTotalBalance` 訊息給對應的使用者 `Account`。這種爲處理訊息而傳遞的訊息，稱爲**巢狀訊息 (nested message)**。UML 建議在巢狀訊息上使用小數編號。例如，message `1.1` 是 message `1` 的第一個巢狀訊息，即 `BankDatabase` 處理 `getAvailableBalance` 訊息時，再度送出的同名訊息。[請注意：若 `BankDatabase` 需要在處理 message `1` 時傳遞第二個巢

狀訊息，第二個訊息會被編號爲 1.2。] 訊息必須於所有巢狀訊息傳送完成之後，才可再傳送下一個訊息。例如，`BalanceInquiry` 按照順序傳送 message 2 和 message `2.1` 後，才會傳送 message 3。

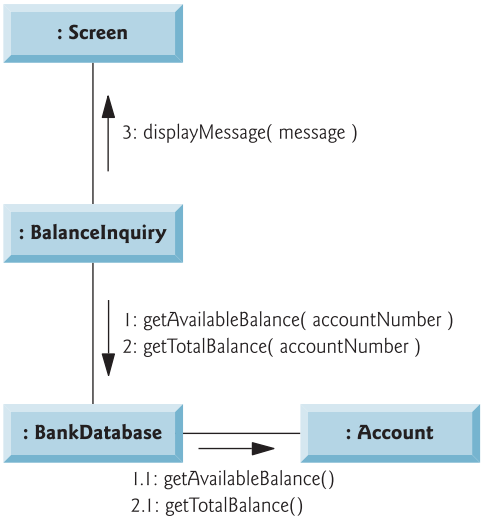

圖 25.24　執行餘額查詢的溝通示意圖

在溝通示意圖中使用巢狀編號，可幫助看圖者分辨訊息會在何時、以及在什麼狀況下被傳送。舉例來說，若我們在圖 25.24 中不使用巢狀編號，而將這些訊息編號爲 1 至 5，看圖者也許就不太容易看出 `BankDatabase` 是在處理 message `1` 的**中間**傳送 `getAvailableBalance` 訊息 (message `1.1`) 給 `Account`，而不是在處理完 message `1` **之後**。巢狀編號的小數點可幫助澄清訊息傳送的先後順序。

順序示意圖

溝通示意圖強調合作關係中的參與者，但表示時間點的方法卻不太好。這時就需要順序示意圖 (sequence diagram) 的幫助，爲合作關係的時間點建立更清楚的模型。圖 25.25 顯示的順序示意圖，描述了 `Withdrawal` 執行時互動關係的發生順序。從物件的矩形向下延伸的虛線是該物件的**生命線 (lifeline)**，代表時間的進行。動作會由上往下隨著時間的順序在物件的生命線上發生；上方的動作會在下方動作之前發生。

在順序示意圖中，訊息的傳遞和與溝通示意圖很相似。實心箭號從發送物件往接收物件延伸，代表著兩個物件之間的訊息，而箭頭指向接收物件生命線上的啓動區。**啓動區 (activation)** 由垂直的細矩形表示，代表有一個物件正在執行中。當物件歸還控制權時，便以一虛線箭頭代表傳回訊息，從歸還控制權物件的啓動區延伸到原本發送訊息的物件啓動區。爲便於閱讀，我們省略傳回訊息的箭號，在 UML 中這是合法的。順序示意圖與溝通示意圖類似，也可以將訊息參數放入訊息名稱後面的括號之中。

圖 25.25 的訊息順序始於 `Withdrawal` 傳送 `displayMessage` 訊息給 `Screen`，提示使用者選擇提款金額。接著 `Withdrawal` 傳送 `getInput` 訊息給 `Keypad`，從使用者那裡取得輸入資料。我們已在圖 25.15 的活動示意圖中，建立了 Withdrawal 裡的控制邏輯模型，因此圖 25.25 的順序示意圖中不再繪出此程式邏輯。我們繪出的是最佳狀況，即使用者帳戶的餘額大於或等於所欲提款的金額，而且吐鈔機中也有足夠的現金來滿足提款需求的情形。讀者若想深入了解如何在順序示意圖中畫出控制邏輯，請參考第 25.2 節最後的網路資源。

取得提款金額後，`Withdrawal` 會傳送訊息 `getAvailableBalance` 給 `BankDatabase`，後者接下來會傳送同名訊息給使用者的 `Account`。如果使用者的帳戶中有足夠的錢滿足此交易，`Withdrawal` 接著會傳送 `inSufficientCashAvailable` 訊息給 `CashDispenser`。機器中若有足夠的現金，`Withdrawal` 會傳送 `debit` 訊息給 `BankDatabase`，減少帳戶餘額 (`totalBalance` 和 availableBalance)，而 `BankDatabase` 會傳送 `debit` 訊息給使用者的 `Account`。最後，`Withdrawal` 傳送 `dispenseCash` 訊息給 `CashDispenser`，吐出現金，再傳送 `displayMessage` 訊息給 `Screen`，提示使用者取走現金。

至此，我們已經找出 ATM 系統中物件間的合作關係，並透過 UML 的溝通示意圖和順序示意圖建立模型。在第 26.2 節中，我們會強化模型結構，完成初步的物件導向設計，接著開始實作 ATM 系統。

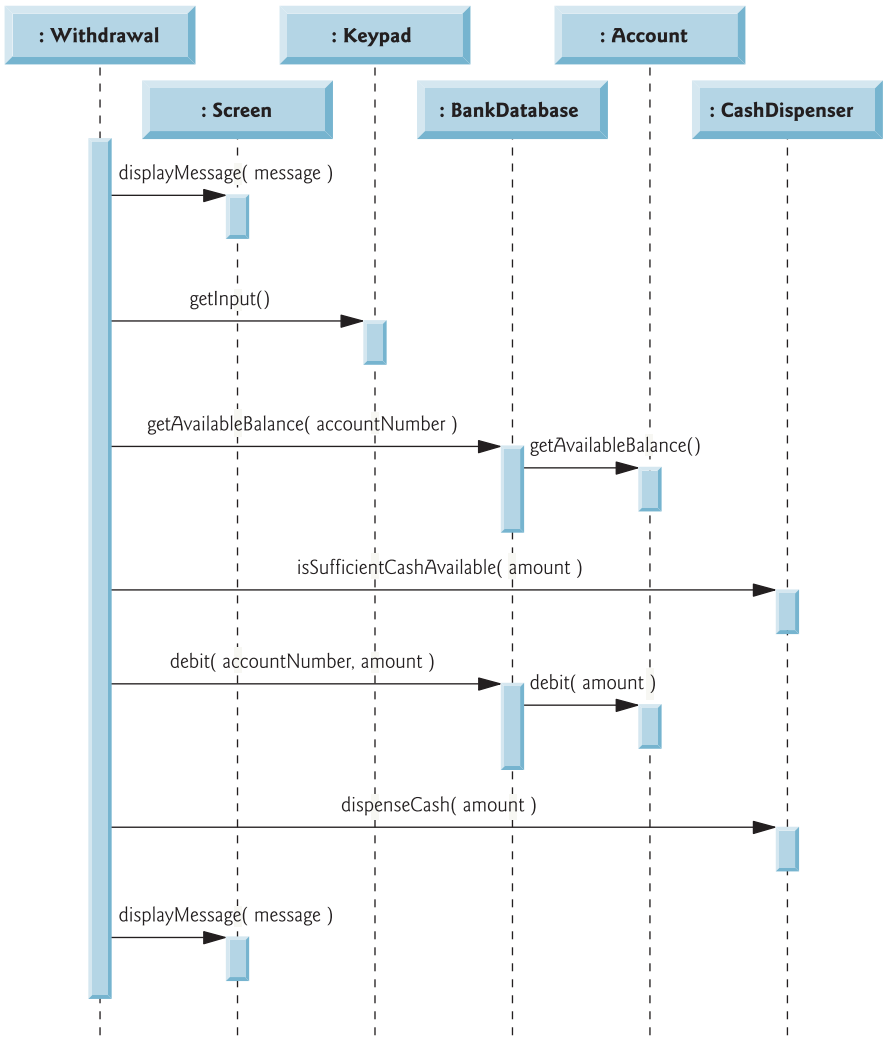

圖 25.25　以順序示意圖表示 `Withdrawal` 執行的模型

25.7 節的自我測驗

25.17 _____的意思是某類別的物件將訊息傳遞給另一個類別的物件。

a) 聯繫
b) 聚合
c) 合作
d) 組合

25.18 哪一種互動示意圖強調的是發生「何種」合作關係？哪一種強調的是「何時」發生合作關係？

25.19 建立一個順序示意圖，表示當 `Deposit` 成功執行之後，ATM 系統中物件之間互動模型，並且解釋模型中的訊息順序。

25.8　總結

在本章中，你學到了如何從詳細的需求文件開始，開發一個物件導向的設計。你使用了六種常用的 UML 示意圖，建立物件導向 ATM 軟體系統的模型。在第 26.3 章中，我們會使用繼承進一步改良設計，然後完整地實作 850 行的 C++程式 (26.4 節)。

自我測驗習題解答

25.1 圖 25.26 的使用案例示意圖代表我們 ATM 系統的修改版本，允許使用者在帳戶之間進行轉帳。

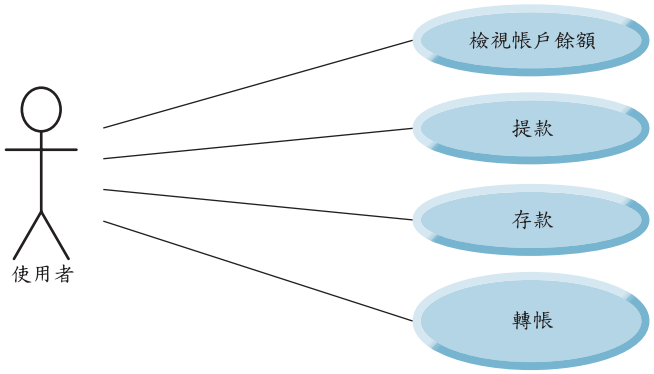

圖 25.26　ATM 系統修改版的使用案例圖，允許使用者在帳戶之間進行轉帳

25.2 b.

25.3 d.

25.4 [請注意：可能有其它答案。] 圖 25.27 之類別示意圖顯示 `Car` 類別的一部分組合關係。

25.5 c。[請注意：在電腦網路中，此關係可能是多對多的。]

25.6 眞。

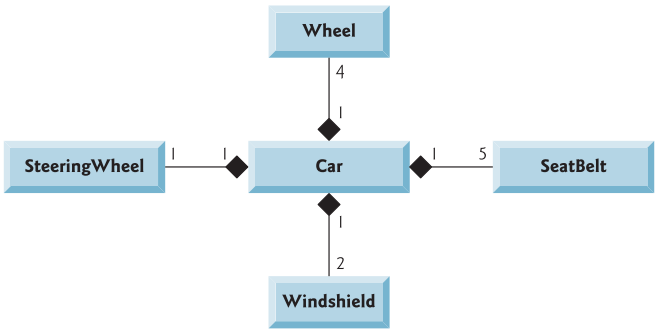

圖 25.27　顯示 Car 類別複合關係的類別示意圖

25.7 圖 25.28 爲一個 ATM 系統類別示意圖，其中是 `Deposit` 類別，而非 `Withdrawal` 類別。請注意，`Deposit` 類別不會存取 `CashDispenser` 類別，但會存取 `DepositSlot` 類別。

25.8 b.

25.9 c。飛行是飛機的操作或行爲，而不是屬性。

25.10 這表示 `count` 是個 `Integer`，初始值爲 500。此屬性隨時會追蹤 `CashDispenser` 內的鈔票數。

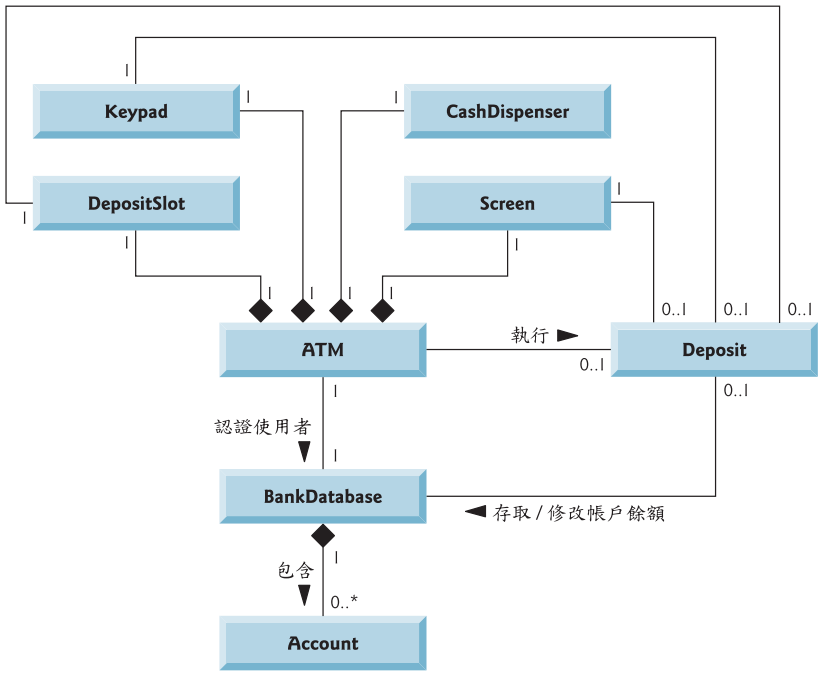

圖 25.28　ATM 系統模型中包含 `Deposit` 類別的類別示意圖

25.11 錯。狀態示意圖是建立系統行爲的模型用的。

25.12 a.

25.13 圖 25.29 所示爲存款交易的活動示意圖。這張圖建立的模型，是使用者從主選單上選定存款選項之後，以及 ATM 回到主選單之前的行爲。請回想，在接收使用者存款金額時，包括將分的整數值轉換成元的動作。還有，將存款金額記入帳戶貸方時，只增加使用者 `Account` 物件的 `totalBalance` 屬性值。銀行只有在確認過存款信封的金額以及支票兌現之後，才會更新使用者 `Account` 物件的 `availableBalance` 屬性，這個動作是獨立於 ATM 系統之外發生的。

25.14 c。

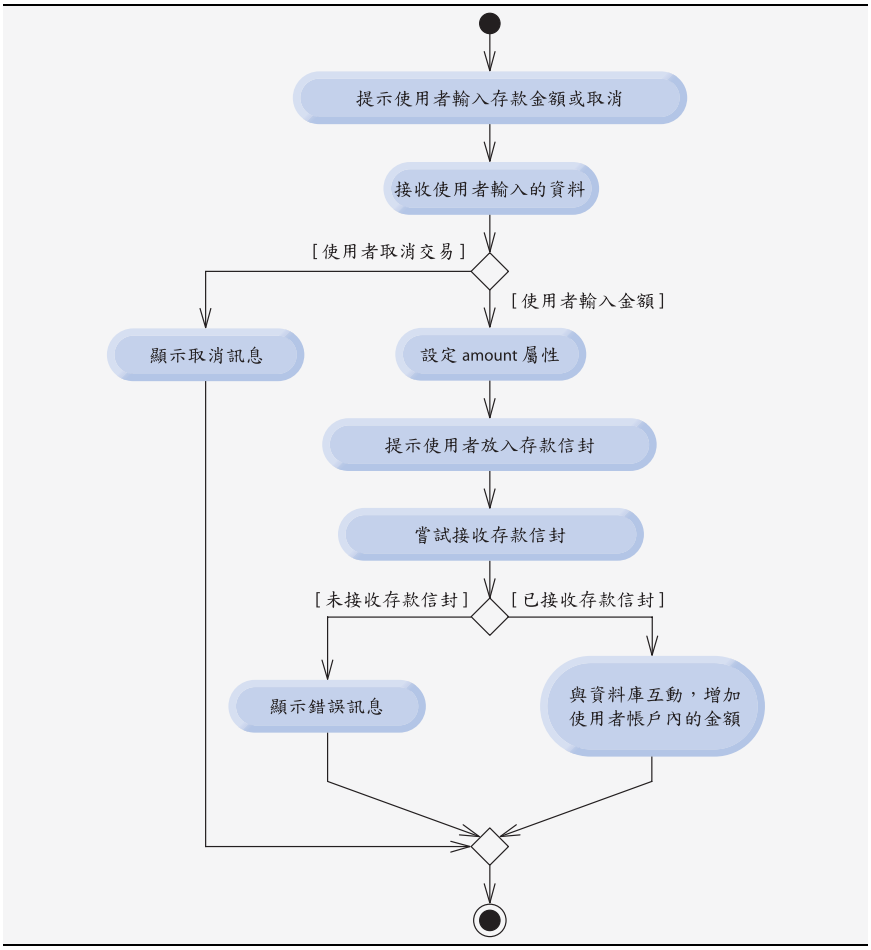

圖 25.29　`Deposit` 交易的活動示意圖

25.15 要表示取得 `Withdrawal` 類別之 `amount` 屬性的操作，可將以下操作敘述放在 `Withdrawal` 類別的操作欄位 (即第三層)：

```
getAmount( ) : Double
```

25.16 這表示名爲 `add` 的操作，其參數爲 `x` 和 `y` 二個整數，並傳回一整數值。

25.17 c。

25.18 溝通示意圖強調發生「何種」合作關係。順序示意圖強調「何時」發生合作關係。

25.19 圖 25.30 的順序示意圖表示 `Deposit` 成功執行之後，ATM 系統中所發生物件之間互動的模型。圖 25.30 指出 `Deposit` 先送 `displayMessage` 訊息給 `Screen` 來要求使用者輸入存款金額，接著 送 `getInput` 訊息給 `Keypad` 接收使用者的輸入資訊。然後，`Deposit` 藉由送 `displayMessage` 訊息給 `Screen`，指示使用者放進存款信封。接著 `Deposit` 送 `inEnvelopeReceived` 訊息給 `DepositSlot` 來確認自動櫃員機已經收到信封。最後，`Deposit` 傳送 `credit` 訊息給 `BankDatabase`，增加使用者 `Account` 的 `totalBalance` 屬性 (availableBalance 不變)；`BankDatabase` 則以送出相同的訊息給使用者 `Account` 來回應。

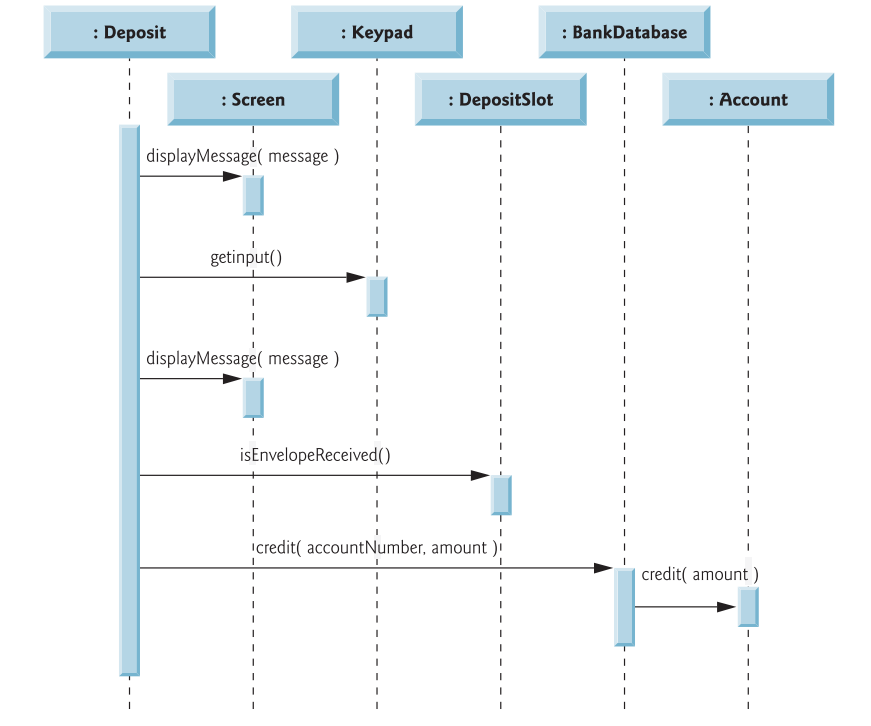

圖 25.30　以順序示意圖表示 `Deposit` 執行的模型