物件導向概念

陳建良


AU
CSIE
真理大學
Aletheia University
資訊工程學系
Department of Computer Science and Information Engineering

程式設計方法

- 非結構化程式設計
- 程序式與結構化程式設計
- 模組化程式設計
- 物件導向程式設計

程式設計方法-說明

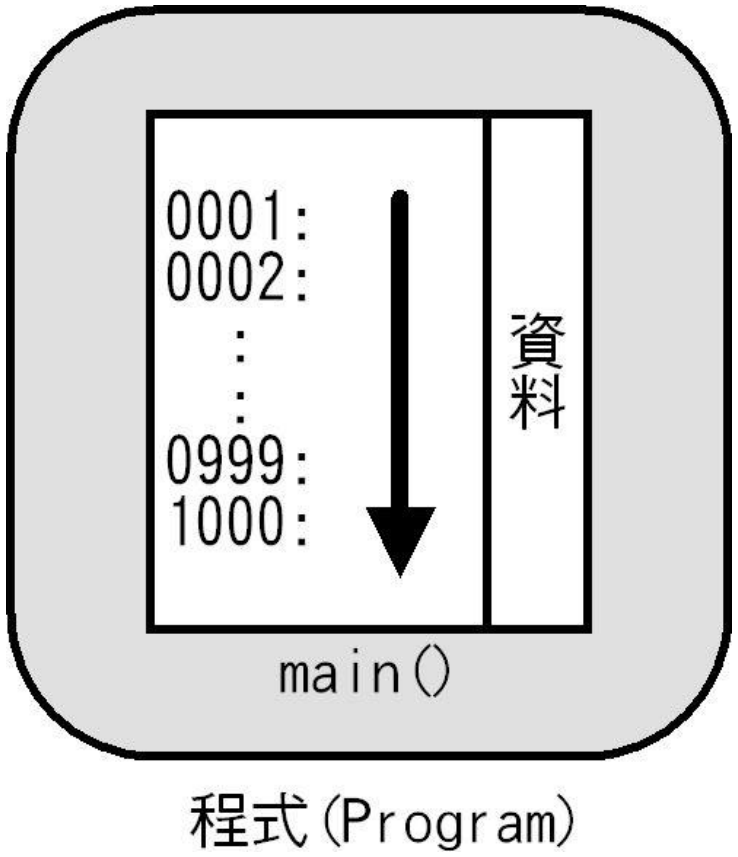
- 計算機科學的「軟體工程」（Software Engineering）專注於研究如何建立正確、可執行和良好撰寫風格的程式碼，嘗試使用一些已經驗證過且可行方法來解決程式問題。
- 「程式設計」（Programming）是使用指定的程式語言，例如：Java語言，以指定風格或技術來撰寫程式碼，在此所謂的風格或技術就是電腦解決程式問題的程式設計方法。

程式設計方法-種類

- 學習程式設計通常都會經歷數個學習過程，即四種程式設計技術（Programming Techniques），或稱為程式設計風格（Programming Styles），如下所示：
  - 非結構化程式設計（Unstructured Programming）。
  - 程序式程式設計（Procedural Programming）與結構化程式設計（Structured Programming）。
  - 模組化程式設計（Modular Programming）。
  - 物件導向程式設計（Object-Oriented Programming）。

非結構化程式設計-圖例

- 非結構化程式設計的程式碼是使用線性方式來依序的執行。



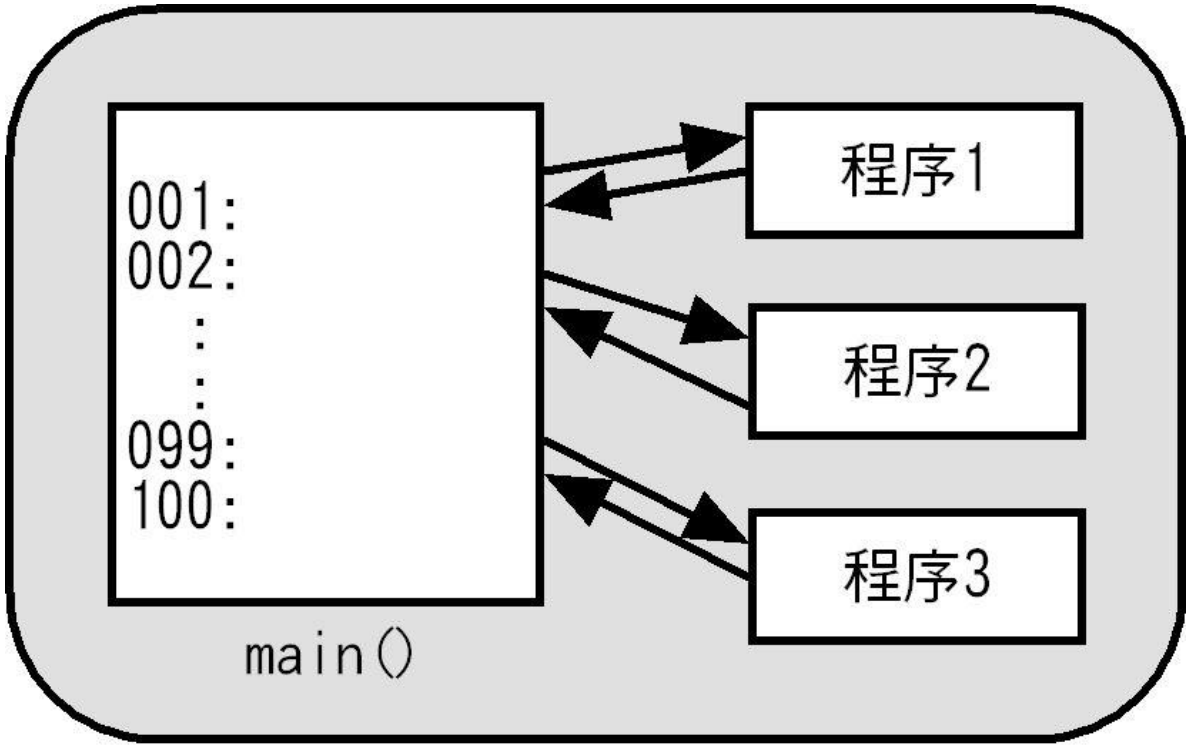
非結構化程式設計-問題

- 程式碼以線性方式執行，如果需要重複操作，例如：計算10次1加到100，就需要重複10次相同的程式碼。
- 如果沒有複製多段程式碼，可以使用GOTO指令，GOTO指令很好用，可以跳到程式中的任何位置，不過，亂跳的結果反而增加程式的複雜度，或產生一些無用的程式碼片斷，稱為「義大利麵程式碼」（Spaghetti Code），程式碼如同義大利麵一般的盤根錯節。
- 非結構化程式的所有程式碼處理的資料都屬於「全域」（Global）資料，不論第1列或第999列都可以直接存取資料，如果不小心拼字錯誤，造成資料誤存，就有可能發生在第1~999列，增加程式除錯的困難度。

程序式與結構化程式設計-說明

- 程序式程式設計是將程式中重複的程式片斷抽出成為「程序」（Procedures，或稱為Subroutine、Routine）或「函數」（Functions），即一段執行特定功能的程式碼。
- 程式因為已經分割成程序，所以在main()方法的程式碼只是依序呼叫各程序的「程序呼叫」（Procedure Call），即執行各程序。整個程式使用流程控制連接各程序，即目前程式設計最常使用的結構化程式設計，屬於程序式程式設計的子集。

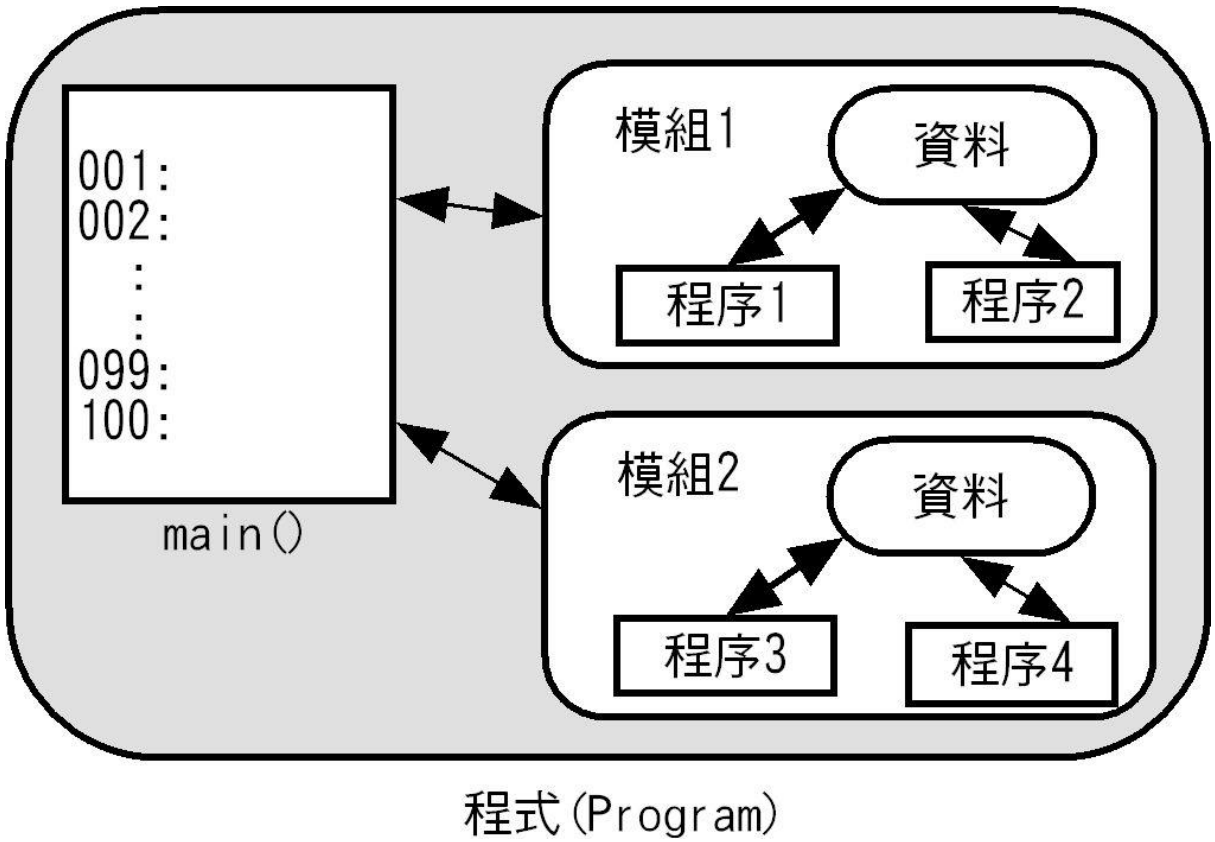
程序式與結構化程式設計-圖例



模組化程式設計-說明

- 模組化程式設計是程序式程式設計的下一個階段，為了能夠重複使用程序式程式設計分割建立的程序，我們可以將相同功能的程序或函數結合在一起成為獨立的「模組」（Modules），模組是處理指定功能的子程式。

模組化程式設計-圖例



模組化程式設計-函式庫

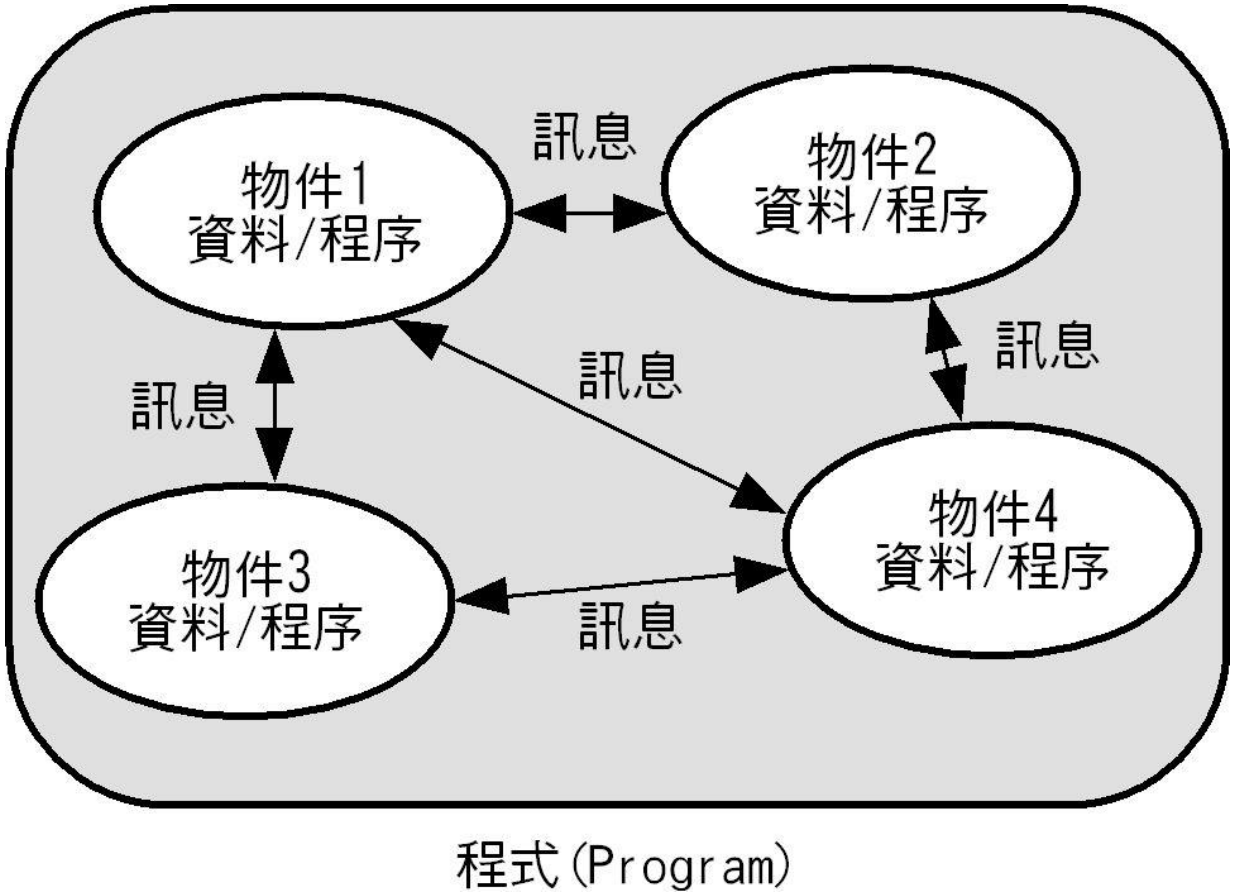
- 模組包含處理的資料和程序/函數，在main()方法呼叫各模組的函數時，可以視為呼叫函式庫（Libraries）的函數，在功能上如同是一個工具箱（Toolbox）。
- 例如：C語言本身很小，**大部分C語言的功能都是由函式庫提供，即編譯程式廠商提供的模組。**
- Java語言是物件導向程式語言，並不會使用模組化程式設計，可以類比的是類別方法的函式庫，例如：Math類別，提供各種數學常數與方法。

物件導向程式設計-說明

- 物件導向程式設計是一種更符合人性化的程式設計方法，將原來專注於問題的分解，轉換成了解問題本質參與的東西，也就是「物件」（Object）。
- 物件內含處理的資料和相關程序，在物件之間是使用訊息來進行溝通。
- 不同於模組化程式設計的模組，物件很容易擴充功能和重複使用，只需建立好物件後，由下而上就可以逐步擴充成為一個完善的物件集合來解決程式問題。

物件導向程式設計-圖例

- 什麼是《物件》
- 什麼是《資料》
- 什麼是《程序》
- 什麼是《訊息》



物件導向的思維

- 什麼是物件
- 識別物件
- 物件導向的抽象化 – 建立類別

物件導向的思維

- 物件導向技術源於1960年代的Simula程式語言，它是Simulation Language的簡稱，這是一種模擬語言，希望使用電腦程式來模擬真實世界的各種處理過程。
- 換句話說，物件導向的思維就是我們現實生活的思維方式，人類自然的思考方式，其實各位早已知道，而且一直使用它來思考問題。

什麼是物件-定義

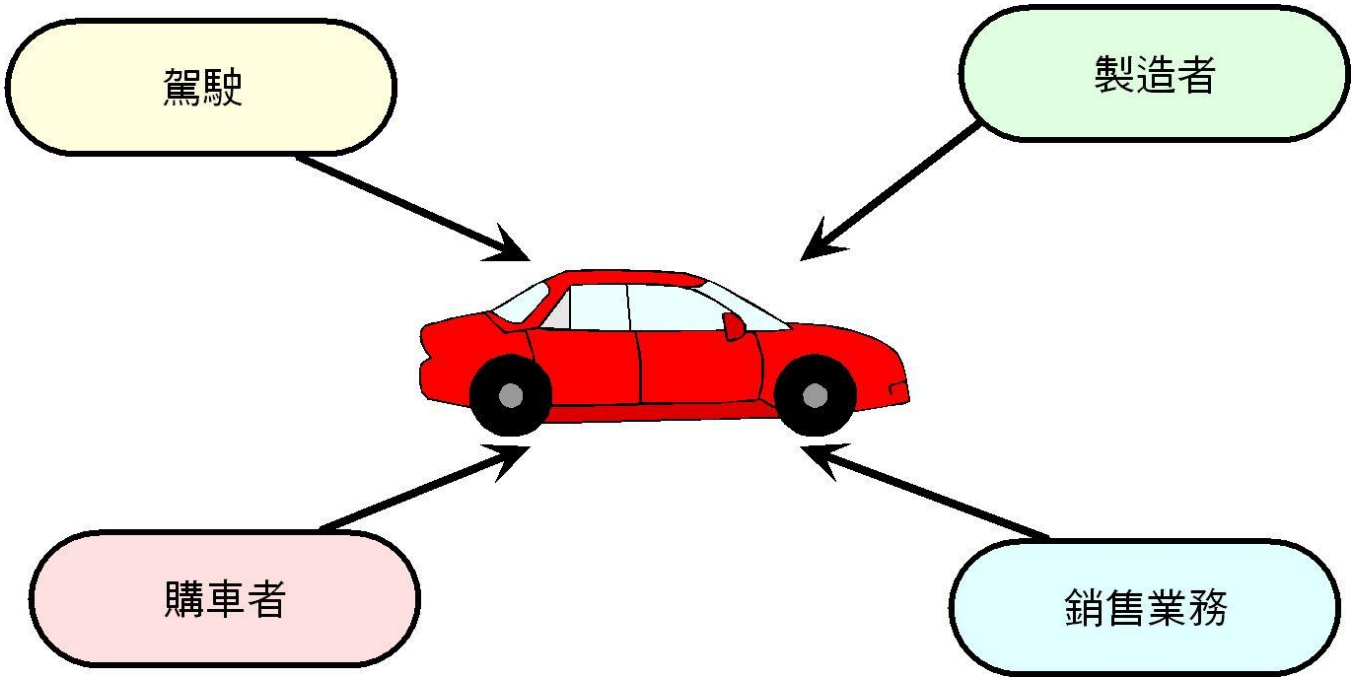
- 物件的英文是Object，在此討論的是現實生活中的物件，並不是程式中的物件，物件的英文原意有物體、東西、對象和目的。
- 換句話說，物件不見得是一種看得到或摸得到的實體，它可能只是一個概念，一種我們可以認知的東西。

什麼是物件-圖例

什麼是物件-認知

- 物件是一種可以認知的東西，例如：我們認知一輛車，是因為聯想到：
  - 車子是紅色。
  - 車子有四個門。
  - 車子有四個輪胎。
- 車輛是車的概念，人類在自然思考時，就會自動將車輛分解成整體（車輛）和部分（門和輪胎）之間的關聯。

什麼是物件-不同的認知



識別物件-可能物件

- 物件導向思維的另一個重點是如何從描述的問題中識別出物件，以便建立問題的模型，如下：
  - 問題中是否有【具體的事物】，例如：人、書、電腦、車子等。
  - 問題中是否有【事件】，例如：訂購商品、借書、參加會議、旅遊等。
  - 問題中是否有【角色】或【組織成員】，例如：員工、客戶、售貨員等。
  - 問題中是否有【位置】、【地方】或【結構】，例如：座標、圖書館、圓、三角形，長方形等。
  - 如果是使用英文描述的問題，以文法來分析，句子中的【名詞】或【名詞子句】都是可能的物件。

識別物件-物件特徵 I

- 保留資訊（**Retained Information**）：物件需要能夠保留資訊，所以物件一定擁有一些屬性，即資料。
- 需要提供服務（**Needed Service**）：物件需要能夠提供服務，例如：更改屬性的操作。
- **共通屬性（Common Attributes）**：所有出現的物件都擁有共通的屬性。

識別物件-物件特徵2

- **共通操作（Common Operators）**：所有出現的物件都擁有共通的操作。
- **本質的需求（Essential Requirements）**：擁有其它外部的實體物件，它需要取得物件的資訊，例如：Order訂單物件需要取得Customer客戶物件的【地址】屬性。
- **多重屬性（Multiple Attributes）**：物件擁有的屬性並非只屬於它，它可以是一個更大物件的屬性。

物件導向的抽象化-建立類別（現象與觀念）

- 現象（Phenomena）是真實世界中，在指定問題範圍內，你可以認知出的物件。例如：紅色汽車、白色貨車和銀色休旅車等。
- 觀念（Concepts）是描述現象的共通特性，排除其詳細部分。例如：紅色汽車、白色貨車和銀色休旅車是一種陸上交通工具的車輛。觀念基本上擁有三種特性，如下所示：
  - 名稱（Name）：用來區別其他觀念的名稱。
  - 目的（Purpose）：描述現象需要符合哪些特性，才能成為觀念的成員。
  - 成員（Member）：哪些現象屬於觀念的成員。

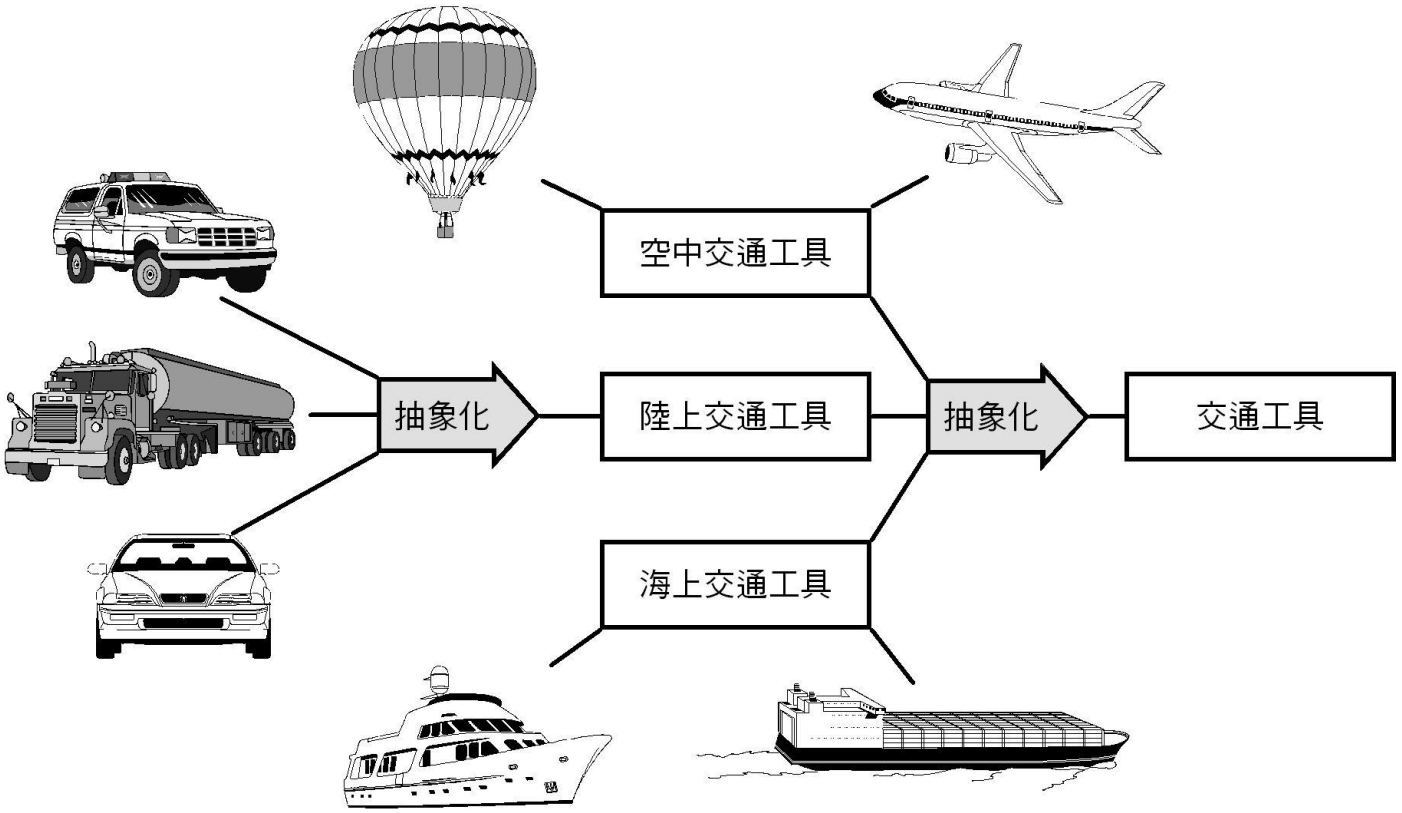
物件導向的抽象化-建立類別(現象與觀念-圖例)

名稱(Name)　　目的(Purpose)　　成員(Member)

物件導向的抽象化-建立類別(抽象化)

- 物件導向技術的抽象化（Abstraction）是從物件中抽出共通部分的特徵，排除其詳細部分。以現象和觀念來說，就是將現象分類成觀念；以物件導向技術來說，就是將物件抽出成類別（Class）。
- 當我們將問題識別出的物件抽象化成類別後，就可以找出類別的關聯性（Relationships）。事實上，找出類別關聯性並不是一件說到就可以作到是事，這也是學習物件導向程式設計遇到的最大瓶頸。

物件導向的抽象化-建立類別(抽象化圖例)



抽象資料型態

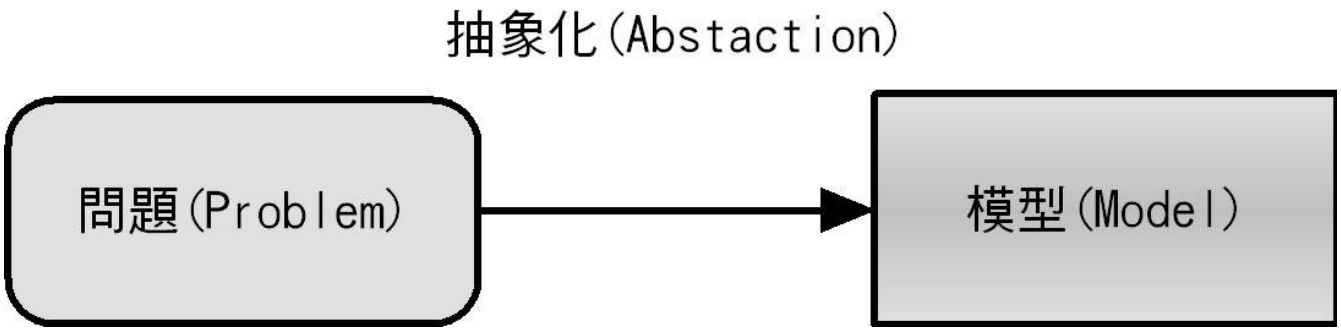
- 抽象化 – 塑模
- 程序或函數抽象化
- 資料抽象化
- 抽象資料型態（ADT）
- 抽象資料型態與物件導向

抽象資料型態

- 物件導向程式設計的精神是資料抽象化，透過抽象資料型態來建立電腦與真實世界的橋樑，描述和模擬真實世界的實體東西，簡單的說，物件導向程式設計就是一種抽象資料型態的程式設計。
- 本節將從抽象化開始，說明傳統程式設計方法將操作和資料分開思考；物件導向程式設計則是將資料和操作一起進行思考的差異。

抽象化 – 塑模(說明)

- 程式設計的目的是解決問題，也就是將現實生活中的真實問題轉換成電腦程式，讓電腦執行程式幫助我們解決問題，這個過程是找出問題的模型，稱為「塑模」（Modeling），如下圖所示：



抽象化 – 塑模(目的與範例)

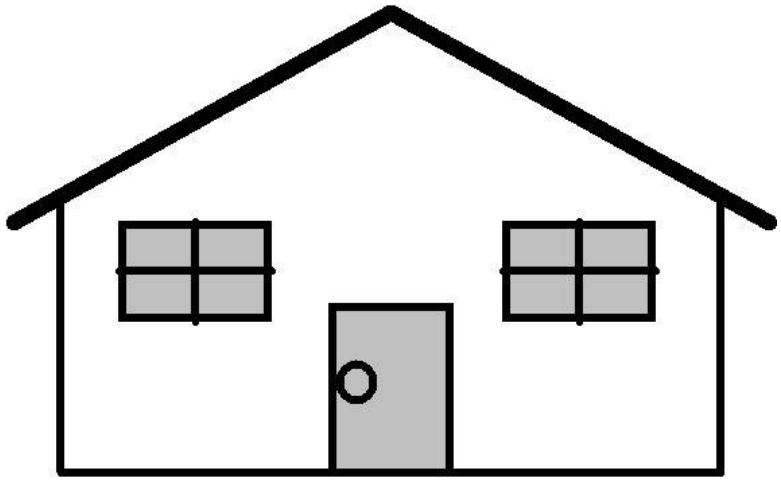
- 將問題轉換成模型的方式稱為「抽象化」（Abstraction），其主要的目的是定義問題的二個屬性，如下所示：
  - **資料（Data）**：問題影響的資料。
  - **操作（Operators）**：問題產生的操作。
- 例如：個人基本資料的問題，可以抽象化成Person模型，資料部分是：姓名、地址和電話號碼，操作部分是：指定和取得客戶資料的姓名、地址和電話號碼。

程序或函數抽象化-由上而下的設計方法

- 程序或函數抽象化（Procedure Abstraction or Function Abstraction）的針對傳統由上而下的程式設計方法，將問題分割成一個個子工作。
- 由上而下的設計方法（Top-down Design）主要是以程序為單位來切割工作，也就是所謂的「程序式程式設計」（Procedural Design）。
- 由上而下的設計方法是一種循序漸進了解問題的方法。

程序或函數抽象化-由上而下的設計方法(範例)

- 例如：目前有一個工作是繪出房屋的圖形，房屋圖形的繪圖工作並不是一筆畫可以完成，我們可以將它分割成多個小工作來處理，以由上而下的設計方法來了解整個繪圖工作的問題。

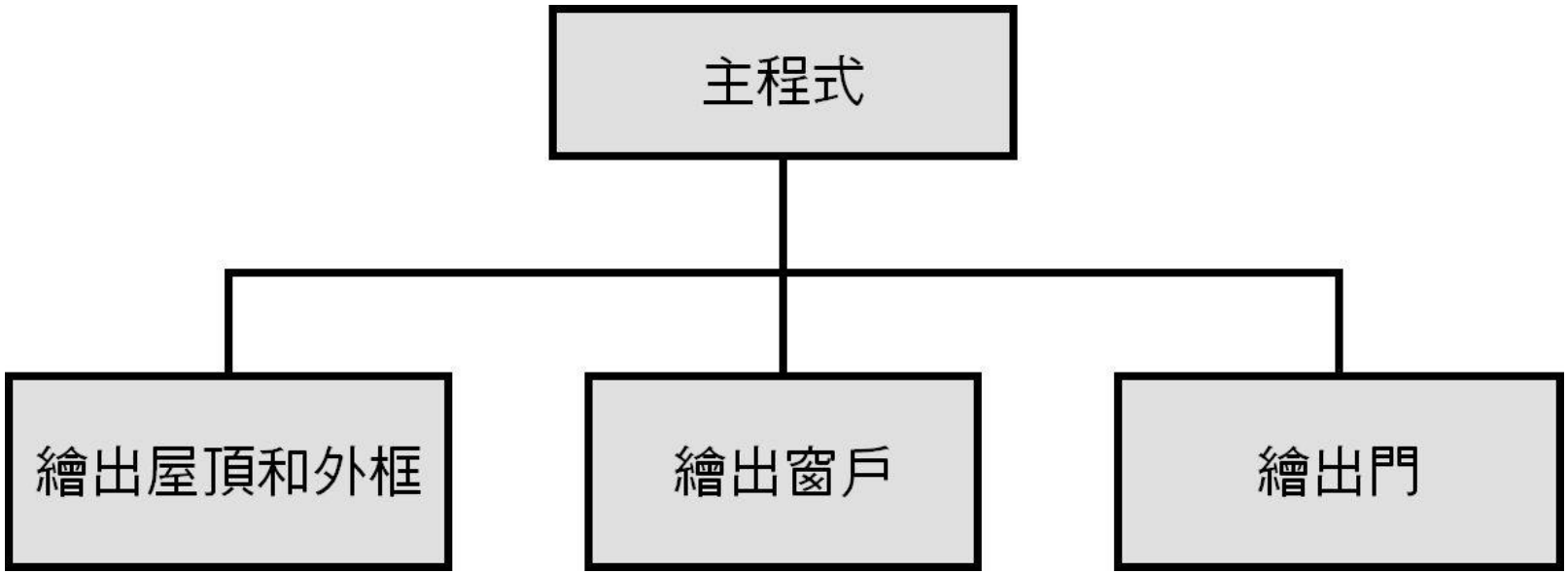


程序或函數抽象化-由上而下的設計方法(步驟一)

- 第一步驟：從房屋繪圖工作可以粗分為三個小工作，如下所示：
  - 繪出屋頂和外框。
  - 繪出窗戶。
  - 繪出門。
- 依據上述工作分割，可以建立各分割小工作之間的模組架構，主程式的虛擬碼，如下所示：

```
Call Draw Outline
Call Draw Windows
Call Draw Door
```

程序或函數抽象化-由上而下的設計方法(步驟一-圖例)



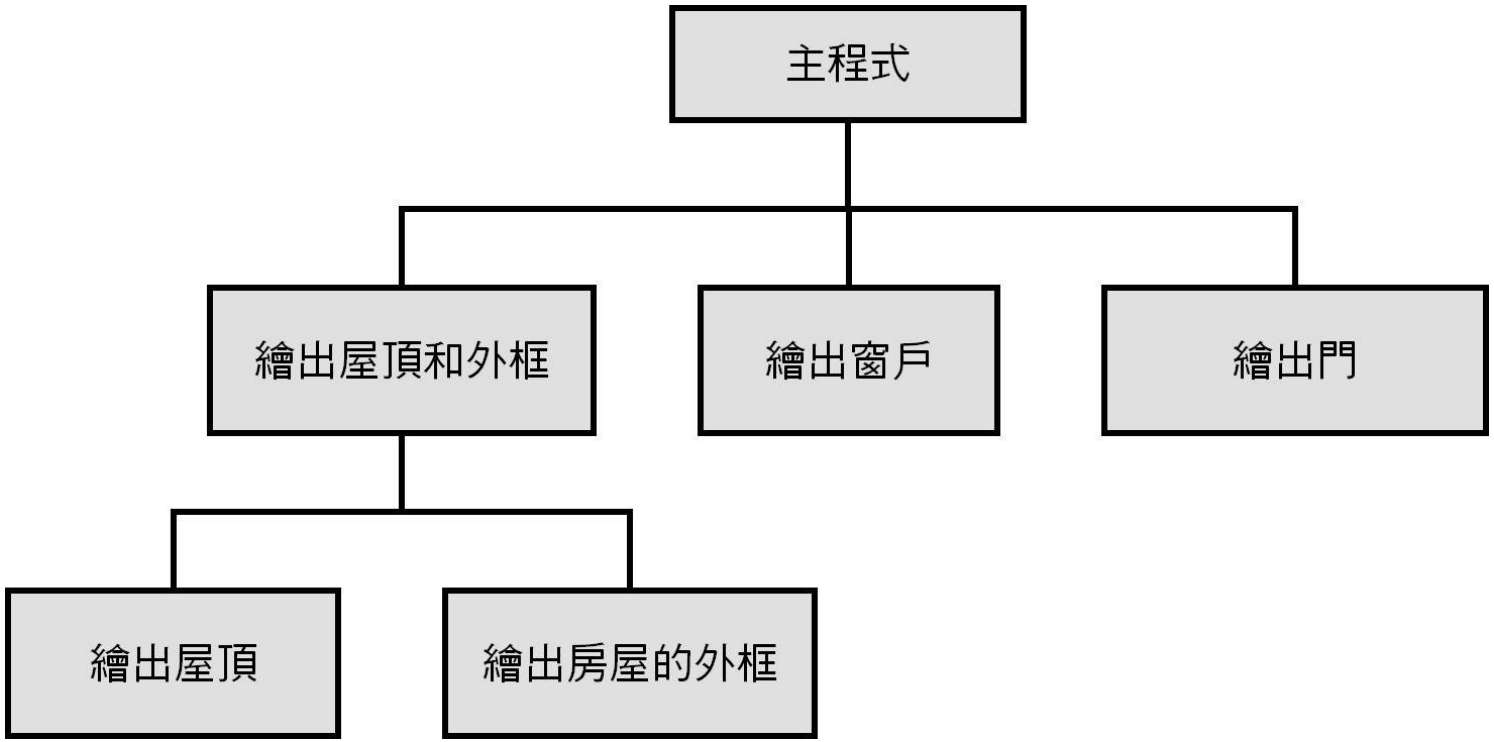
程序或函數抽象化-由上而下的設計方法(步驟二)

- 第二步驟：接著將第一個小工作【繪出屋頂和外框】（Draw Outline）再次進行分割成二個小工作，如下所示：
  - 繪出屋頂。
  - 繪出房屋的外框。
- 依據上述工作分割，我們可以建立下一層各問題間的模組架構，在【繪出屋頂和外框（Draw Outline）】模組是呼叫其下的模組，其虛擬碼如下所示：

  Call Draw Roof

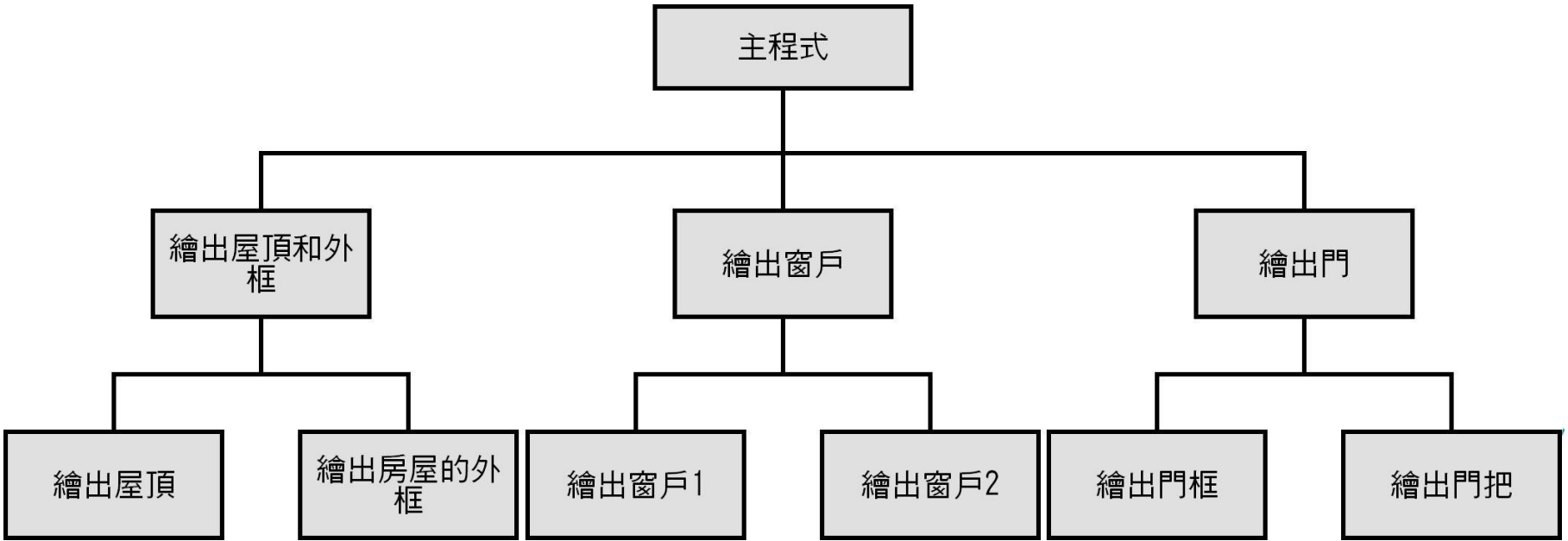
  Call Draw House Frame

程序或函數抽象化-由上而下的設計方法(步驟二-圖例)



程序或函數抽象化-由上而下的設計方法(最後結果)

- 在繪出房屋問題，我們可以將問題分解成一個個繪圖操作的程序，其最後結果的操作步驟是將問題以程序分割出來，著重於需要執行哪些處理的操作屬性。



程序或函數抽象化-程序或函數抽象化

- 在由上而下的設計方法分割的程序或函數之中，我們並不用考量實作的程式碼，只需定義好程序或函數使用介面的參數和傳回值，將它視為一個黑盒子，換句話說，程式可以使用任何符合介面的程序或函數來取代，稱為程序或函數抽象化。
- 當定義出程序的使用介面後，如果開發出另一種更佳的演算法，只需將程序的程式碼改為新的演算法來實作，並不用更改使用介面，即可增加執行的效率，稱為程序抽象化。

資料抽象化-說明

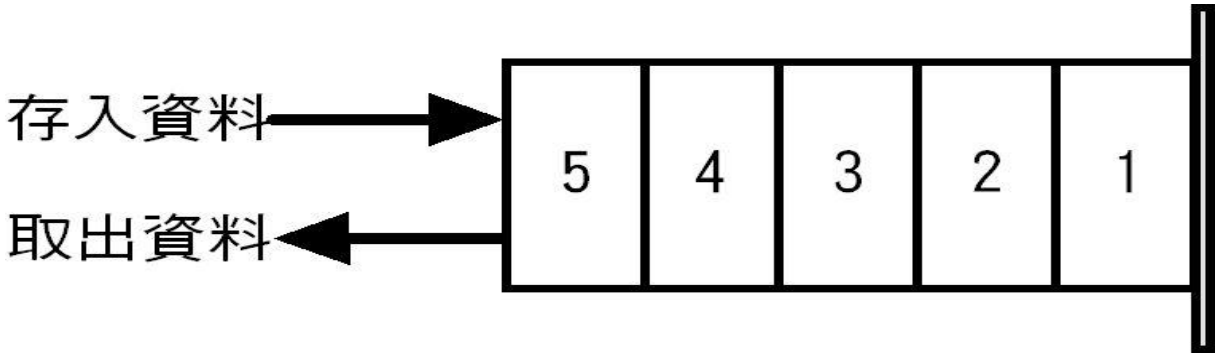
- 「資料抽象化」（Data Abstraction）是一種方法將基本資料型態的變數組合成複合資料（Compound Data），使用函數來處理複合資料，以便隱藏實際複合資料的儲存方式。

資料抽象化-資料結構

- 資料結構就是一種資料抽象化，其目的是研究程式使用的資料在電腦記憶體的儲存方式，以便撰寫程式處理問題時，能夠使用最佳的資料儲存方式，並且提供一種策略或方法來有效率的善用這些資料，以便達到下列目的，如下所示：
  - 程式執行速度快。
  - 資料佔用最少的記憶空間。
  - 更快速的存取這些資料。

資料抽象化-堆疊範例

- 使用模組化程式設計建立堆疊（Stacks）資料結構，其兩種特性，如下所示：
  - 只允許從堆疊的頂端存取資料。
  - 資料存取的順序是後出先進（Last Out, First In），也就是後存入堆疊的資料，反而先行取出。



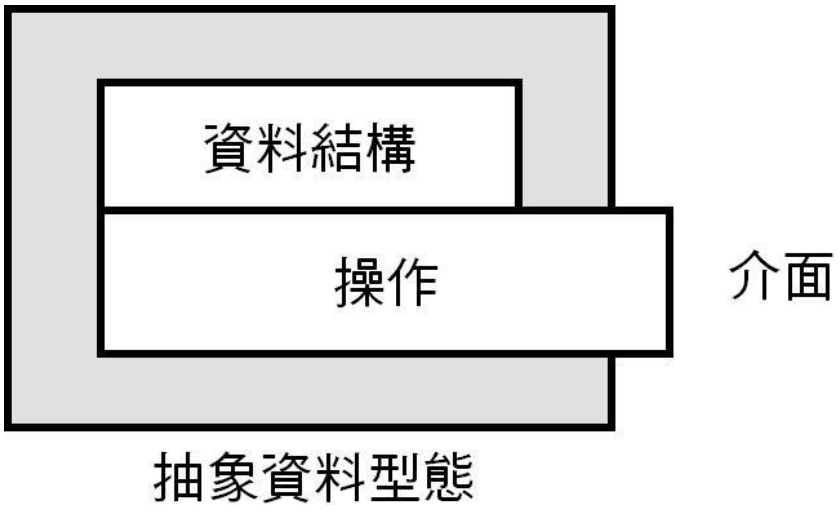
資料抽象化-堆疊抽象化

- 堆疊可以使用多種方式來儲存資料，例如：陣列或串列，不過這不重要，因為堆疊是使用pop()、push()和empty()函數存取堆疊資料，實際的資料儲存方式被隱藏在這些函數之後，如此稱為資料抽象化。
- 傳統結構化程式設計是一種以資料為中心的程式設計方法，程式是由資料結構和演算法組成，如下所示：

  程式 = 資料結構 + 演算法

抽象資料型態（ADT）-說明

- 「抽象資料型態」（Abstract Data Type）是使用資料抽象化的方法建立的自訂資料型態，抽象資料型態包含資料和相關操作，將資料和處理資料的操作一起思考，結合在一起，操作是對外的使用介面，如下圖所示：

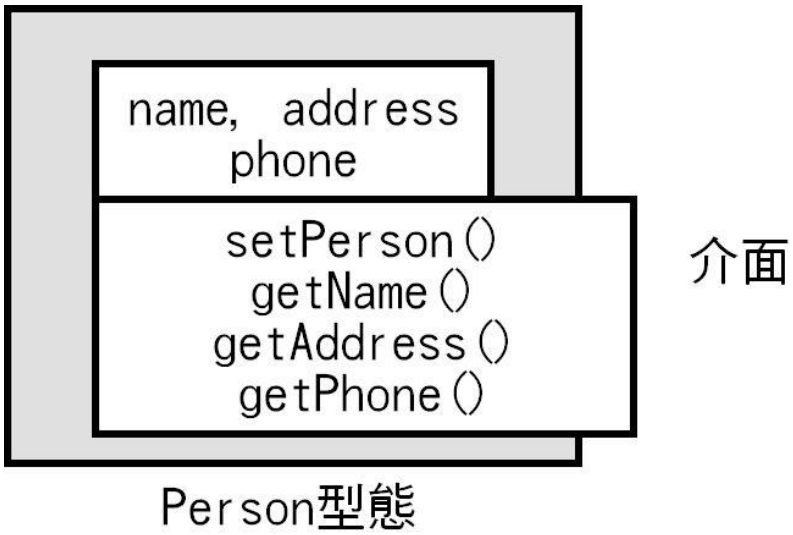


抽象資料型態（ADT）-物件導向的抽象資料型態

- 物件導向程式語言的抽象資料型態，在Java語言就是「類別」（Class），強調使用抽象資料型態描述真實世界的各種實體，簡單的說，實體是一個東西。
- 在將個人基本資料問題抽象化成Person模型，可以對應真實世界的人實體，內含姓名name、地址address和電話號碼phone等資料，setPerson()指定個人資料，getName()、getAddress()和getPhone()取出個人資料的操作，如此可以建立Person抽象資料型。

抽象資料型態（ADT）-範例

- 以Java語言來說Person型態就是Person類別，程式可以使用Person類別建立多個Person副本，用來模擬真實世界的個人，例如：朋友、同事或客戶等。



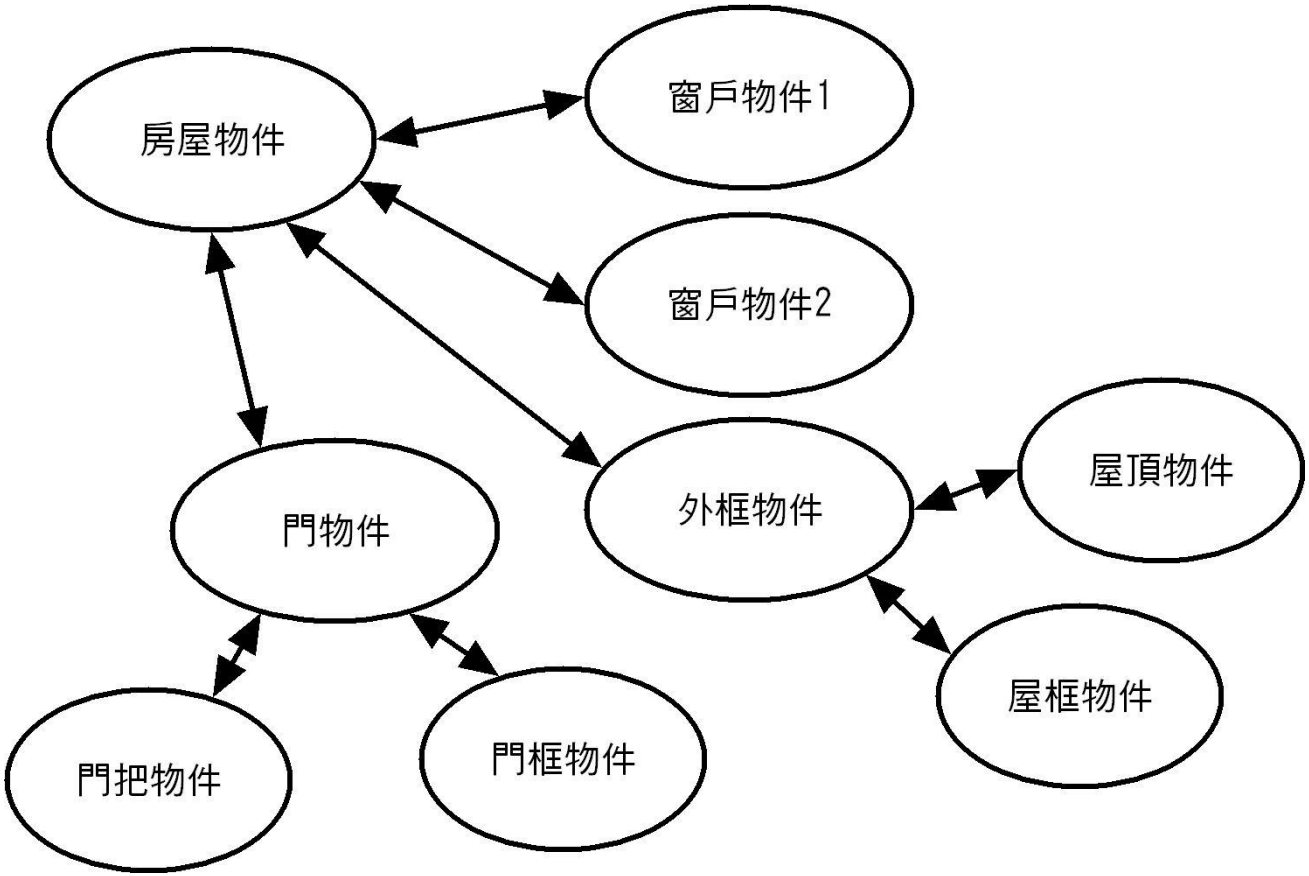
抽象資料型態與物件導向-說明

- 物件導向程式設計的精神是資料抽象化，每一個物件屬於一種抽象資料型態的類別，物件導向將問題的資料屬性和資料本身的相關操作一起思考，並不考量其它資料或不相關的操作，以便建立一個個完善定義的物件（Object）。

抽象資料型態與物件導向-範例說明

- 房屋是由一個個物件來組成，不同於傳統程式設計將資料和操作分開思考，物件中的資料需要和操作一起思考，物件包含資料和處理此資料的相關操作，例如：門把物件包含門把尺寸、色彩等資料，再加上繪出門把操作。窗戶物件包含窗戶尺寸、色彩和位置資料，繪出窗戶的操作。
- 整個房屋是由門、窗戶和外框物件組成，門是由門把和門框組成，外框是由屋頂和屋框所組成。因為問題是由各種物件組成，如同車輛是由成千上萬個零件所組裝。

抽象資料型態與物件導向-範例圖例



抽象資料型態與物件導向- 模擬真實世界

- 物件導向程式設計就是在模擬真實世界，以便找出解決問題所需的物件集合和其關聯性，物件之間使用訊息建立關係，透過物件集合之間的合作來解決程式問題，如下所示：

  程式 = 物件 + 訊息

- 如同車輛是由成千上萬個零件所組裝，物件導向程式設計可以視為是一個組裝工作，將眾多現成或改進的物件結合起來。

抽象資料型態與物件導向- -軟體IC

- 很明顯的！物件導向技術更貼近人類的思維，每一個物件是一個零件，如同「軟體IC」（Software IC），只需選擇不同的IC就可以裝配出不同規格的主機板，只需選用適當的軟體IC，我們就可以輕鬆完成應用程式設計。
- 更進一步，如果現在有另一項工作需要繪出一幢別墅，我們就可以直接利用上述範例中現成的房屋零件，在擴充各物件的功能後，即可輕鬆組合出一幢別墅，這種擴充方式就是物件導向的「繼承」（Inheritance）觀念。

物件導向的應用程式開發

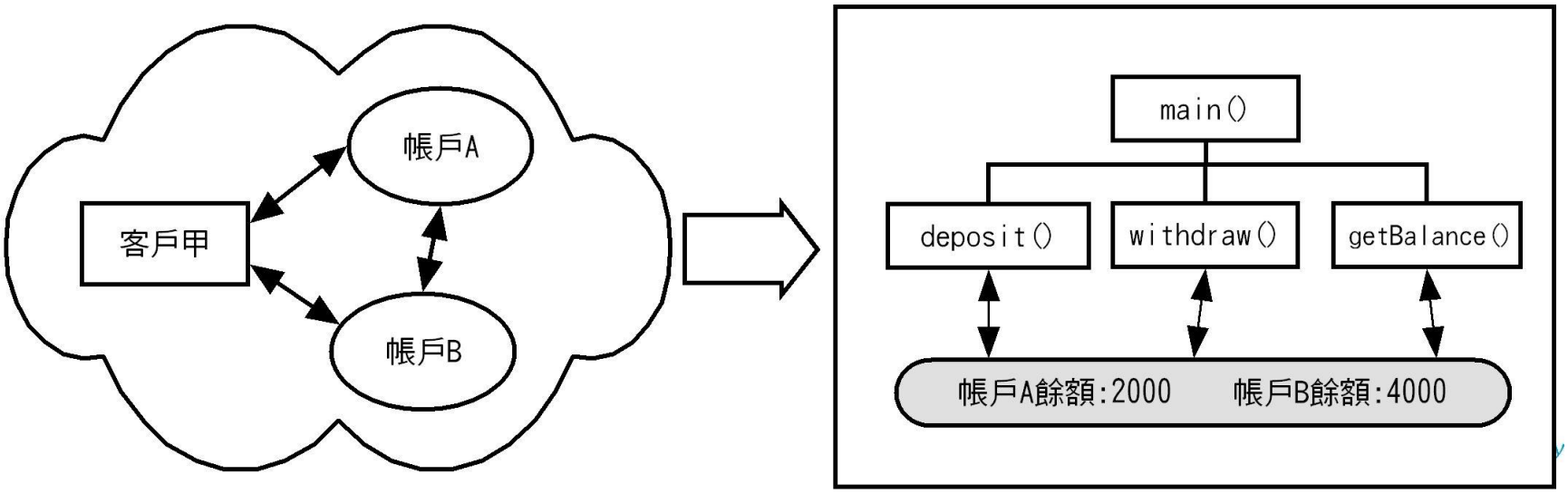
- 傳統的應用程式開發
- 物件導向的應用程式開發

傳統的應用程式開發-說明

- 傳統的應用程式開發是將資料和操作分開來思考，著重於如何找出解決問題的程序或函數。例如：一家銀行的客戶甲擁有帳戶A和B共兩個帳戶，客戶甲在查詢帳戶A的餘額後，從帳戶A提出1000元，然後將1000元存入帳戶B。
- 傳統的應用程式開發建立的應用程式模型是解決問題所需的程序與函數，包含：存款的deposit()函數、提款的withdraw()函數和查詢餘額的getBalance()函數。

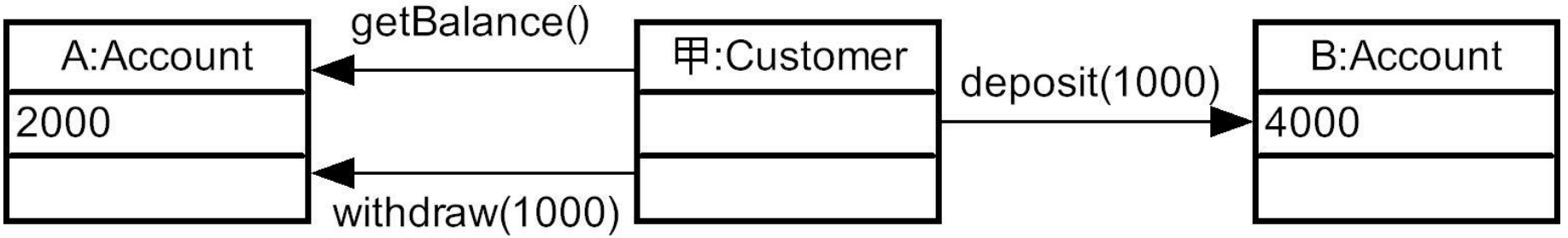
傳統的應用程式開發-圖例

- 左邊是真實世界中，參與的物件和其關聯性，右邊是經過結構化分析和設計（Structured Analysis/Design）後建立的應用程式模型。



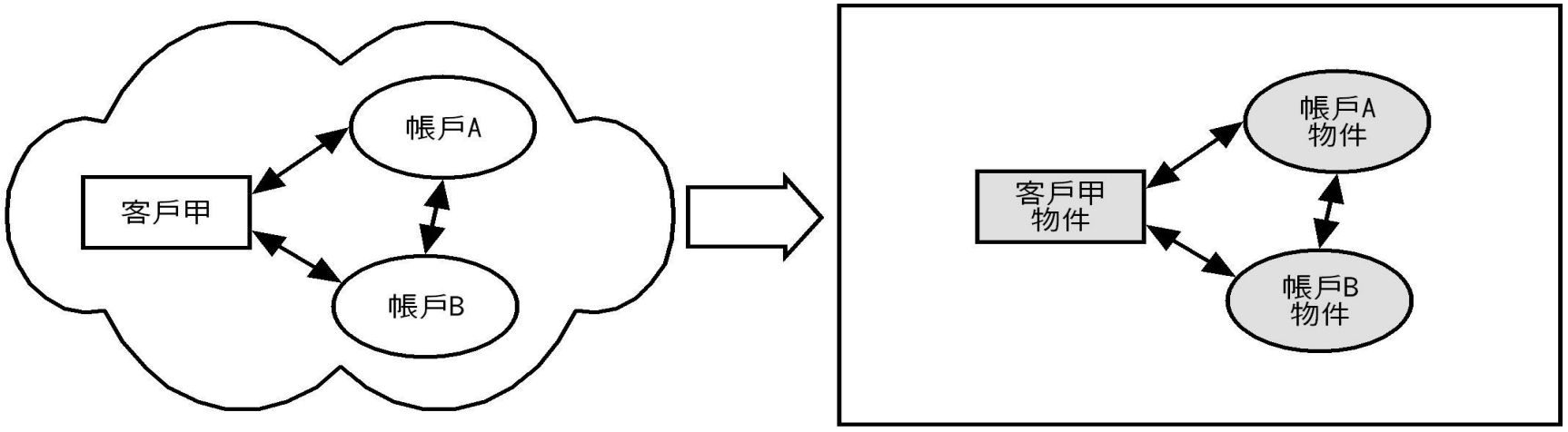
物件導向的應用程式開發-說明

- 物件導向的應用程式開發是將資料和操作一起思考，其主要的工作是找出參與物件和物件之間的關聯性，並且透過這些物件的通力合作來解決問題。
- 物件導向應用程式開發因為是將資料和操作一起思考，所以帳戶物件除了餘額資料外，還包含處理帳戶餘額的相關方法：**getBalance()**、**withdraw()**和**deposit()**方法，如下圖所示：



物件導向的應用程式開發-圖例

- 電腦系統建立一個對應真實世界物件的模型，簡單的說，這是一個模擬真實世界的物件集合，稱為物件導向模型（Object-Oriented Model）。



真實世界　　應用程式

物件導向技術的三大觀念

- 物件觀念
- 訊息觀念
- 類別觀念

物件導向的抽象化- 建立類別(UML塑模過程)

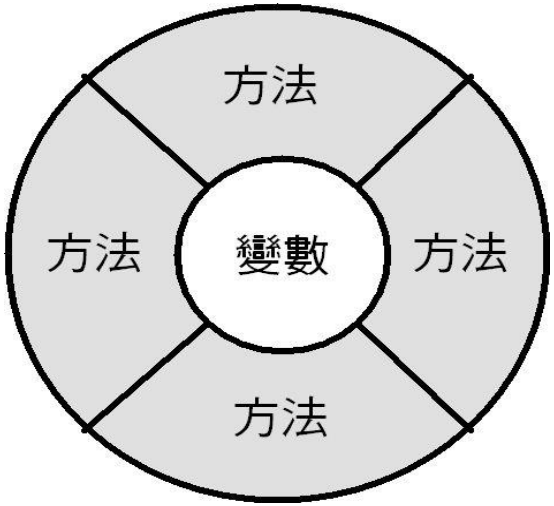
- UML的塑模過程，其基本步驟如下所示：
  - 從問題的需求分析開始，建立使用案例圖（Use Case Diagram）。
  - 從寫出的使用案例情節（Scenarios）來識別出物件，抽象化成類別。
  - 使用順序圖（Sequence Diagram）和合作圖（Collaboration Diagrams）找出類別行為和關聯性。
  - 建立物件導向模型的UML類別圖（Class Diagram）。

物件導向技術的三大觀念

- 物件導向技術有三種重要觀念：物件、訊息和類別，簡單說明如下所示：
  - **物件**：提供資料和處理資料程序（Java語言是方法）的封裝。
  - **訊息**：在物件之間的溝通方式，可以建立互動和支援多型。
  - **類別**：物件的分類，可以實作類別架構的繼承。

物件觀念-什麼是物件

- 物件是物件導向技術的關鍵，以程式的角度來說，它是電腦用來模擬現實生活的東西或事件，也是組成整個程式的元件。物件是資料與相關處理資料的程序和函數結合在一起的組合體，資料就是變數，程序和函數在Java語言稱為方法，如下圖所示：



物件觀念-封裝

- 物件的方法是對外的使用介面，資料和相關方法的實作程式碼都包裹隱藏起來，稱為「封裝」（Encapsulation）。
- 對於程式設計者來說，我們並不用考慮物件內部方法的程式碼是如何撰寫，只需要知道這個物件提供什麼介面和如何使用它即可。

物件觀念-三種特性

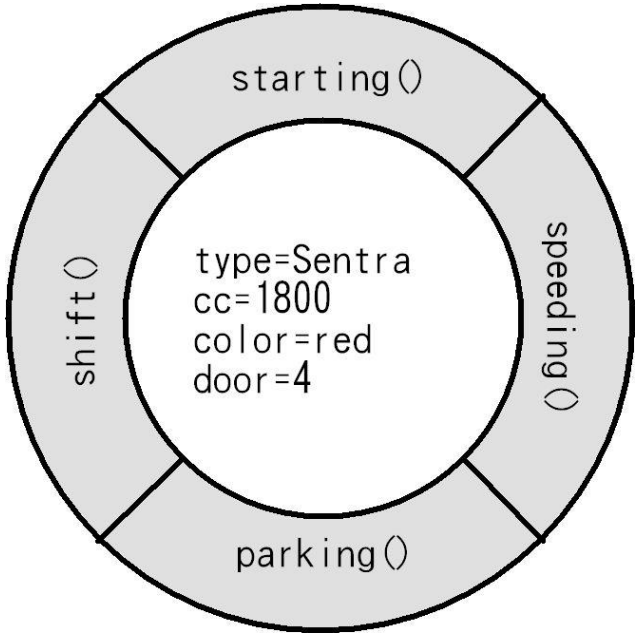
- **狀態（State）**：物件所有**屬性**（Attributes）目前的狀態值，屬性是用來儲存物件的狀態，可以是布林值變數，也可能是另一個物件。例如：車子的車型、排氣量、色彩和自排或手排等屬性。
- **行為（Behavior）**：行為是物件可見部分提供的服務，也就是塑模所抽象化的操作，可以作什麼事？Java語言是使用**方法**來實作行為。例如：車子可以發動、停車、加速和換擋等。
- **識別字（Identity）**：**識別字是用來識別不同的物件**，每一個物件都擁有獨一無二的識別字，Java語言是使用物件參考（Reference）作為物件的識別字。

物件觀念-物件的範例1

- 物件可以模擬真實生活的東西，例如：Car1物件模擬一輛1800cc紅色四門的Sentra車子。
- Car1是物件的識別字，使用Car1識別字就可以在眾多模擬其他車輛的Car2、Car3、Car4….物件中，識別出指定的車輛物件。
- Car1物件的屬性和行為，如下所示：
  - **屬性**：車型（type）、排氣量(cc)、色彩(color)、幾門(door)。
  - **行為**：發動（starting）、停車（parking）、加速（speeding）、換檔（shift）。

物件觀念-物件的範例2

- Car1物件模擬車輛時是使用變數儲存屬性目前的狀態值，並且建立方法來模擬行為，如下所示：
  - 狀態：type=Sentra、cc=1800、color=red、door=4。
  - 方法：
    - starting()
    - parking()
    - speeding()
    - shift()



物件觀念-複合物件

- 「複合物件」（Composite Object）是指物件的屬性是另一個物件，例如：上述Car1物件door是整數的車門數，如果是車門Door物件時，Car1物件就是一個複合物件。

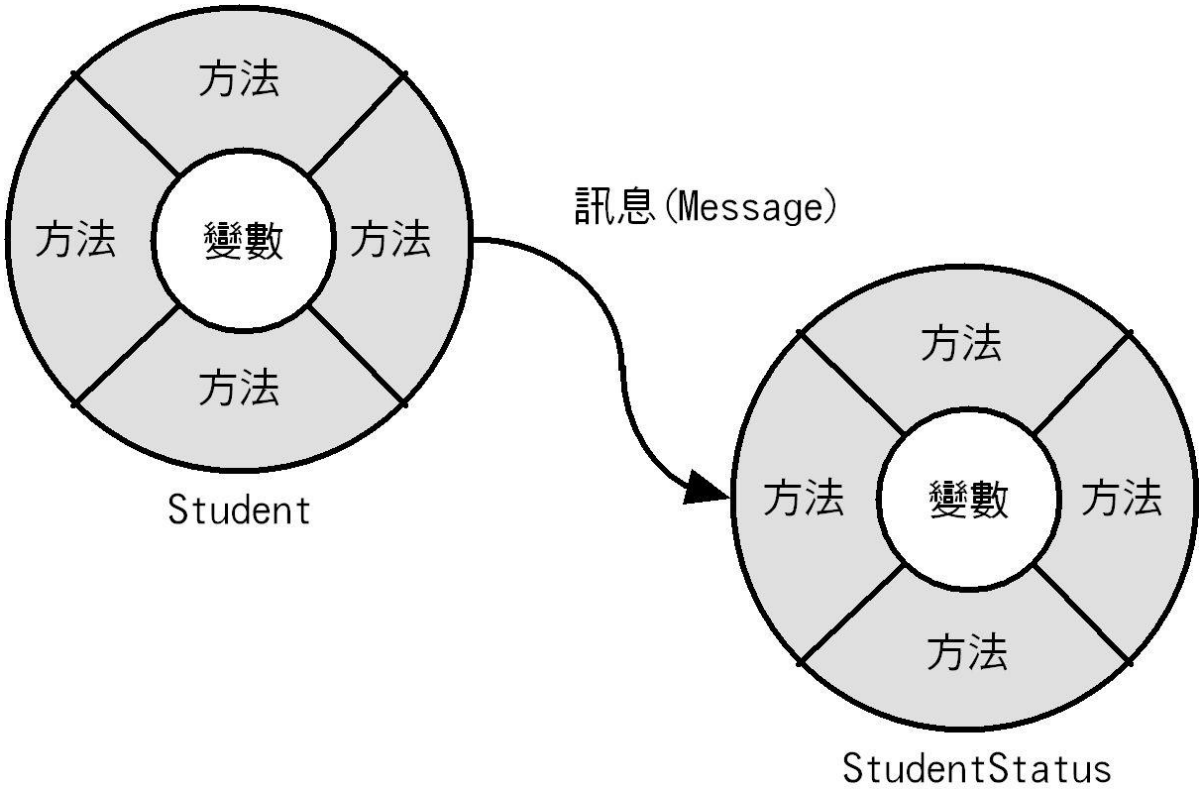
訊息觀念-說明

- 物件可以模擬現實生活的東西，但是現實生活中的東西會彼此互動。例如：學生要求成績（學生與成績物件）、約同學看電影（同學與同學物件）和學生彈鋼琴（學生與鋼琴物件）等互動。所以，我們建立的物件之間也需要互動，使用的就是訊息（Message）。

訊息觀念-什麼是訊息

- 物件是使用訊息來模擬彼此的互動，換句話說，訊息是物件之間的溝通橋樑，可以啟動另一個物件來執行指定的行為。
- 例如：Student學生物件需要查詢成績，學生成績是儲存在StudentStatus物件，此時Student物件可以送一個訊息給StudentStauts物件，告訴它需要查詢學生的成績。

訊息觀念-訊息圖例



訊息觀念-訊息圖例說明

- 訊息是從Student物件的發送物件（Sender）送到StudentStatus接收物件（Receiver），訊息內容是一個命令，要求執行一個指定的方法query()，方法可以加上參數，例如：查詢學生姓名name的成績，訊息提供3種資訊，如下：

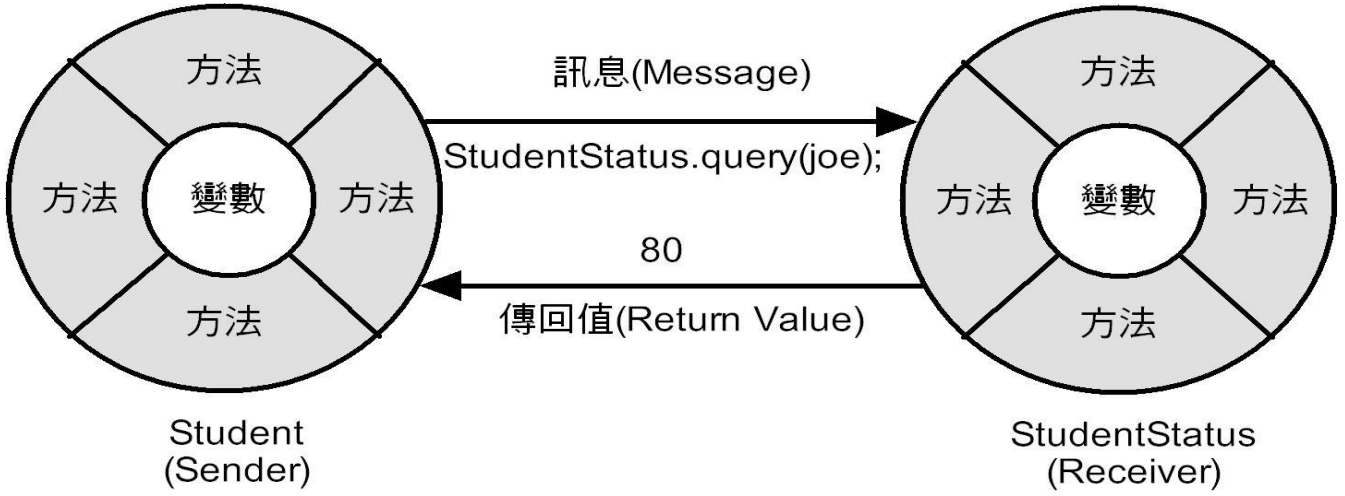
  Smalltalk：StudentStatus query:joe

  C++/Java：StudentStatus.query(joe);

- 訊息的「：」符號前是使用的程式語言，以後才是真正的訊息內容，指出接收物件是StudentStatus，要求執行的方法是query()，其參數是joe。

訊息觀念-傳回值

- 在接收物件接到訊息後，就會執行指定的方法，然後回應訊息給發送物件（也可能不回應），稱為「傳回值」（Return Value），即查詢結果的學生成績，如下圖所示：

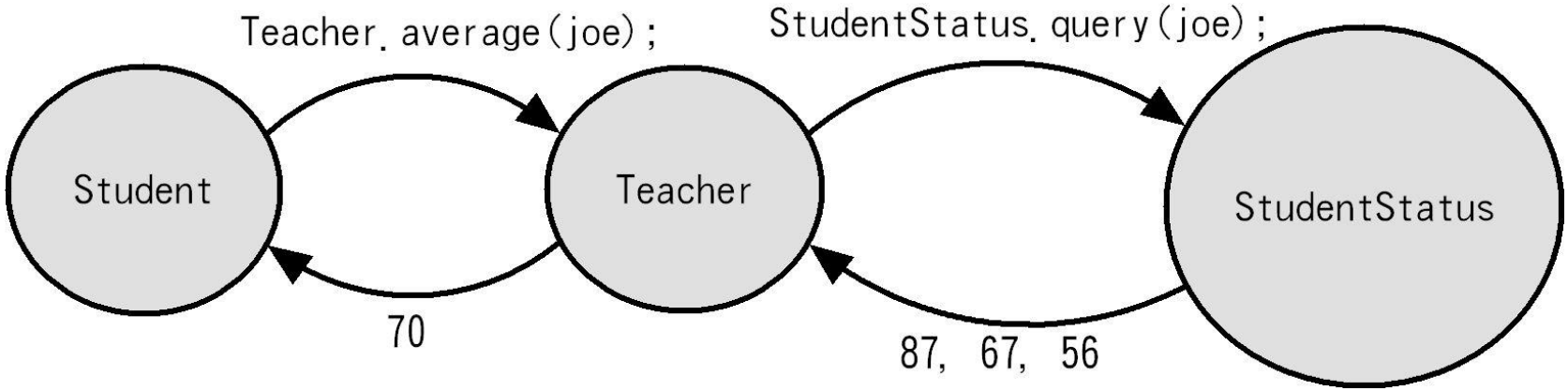


訊息觀念-循序操作(說明)

- 物件送出的訊息，有可能在接收物件執行方法後就產生回應訊息。
- 也有可能是觸發另一個訊息，操作會繼續送出一系列訊息給其他物件，以便依序執行各物件的指定方法來完成整個操作，稱為「循序操作」（Sequential Operation）。

訊息觀念-循序操作(圖例)

- 例如：學生平均成績的查詢是送訊息到Teacher物件執行average()方法，Techer物件將觸發另一個訊息到StudentStatus物件查詢學生的三科成績，如下圖所示：



訊息觀念-過載

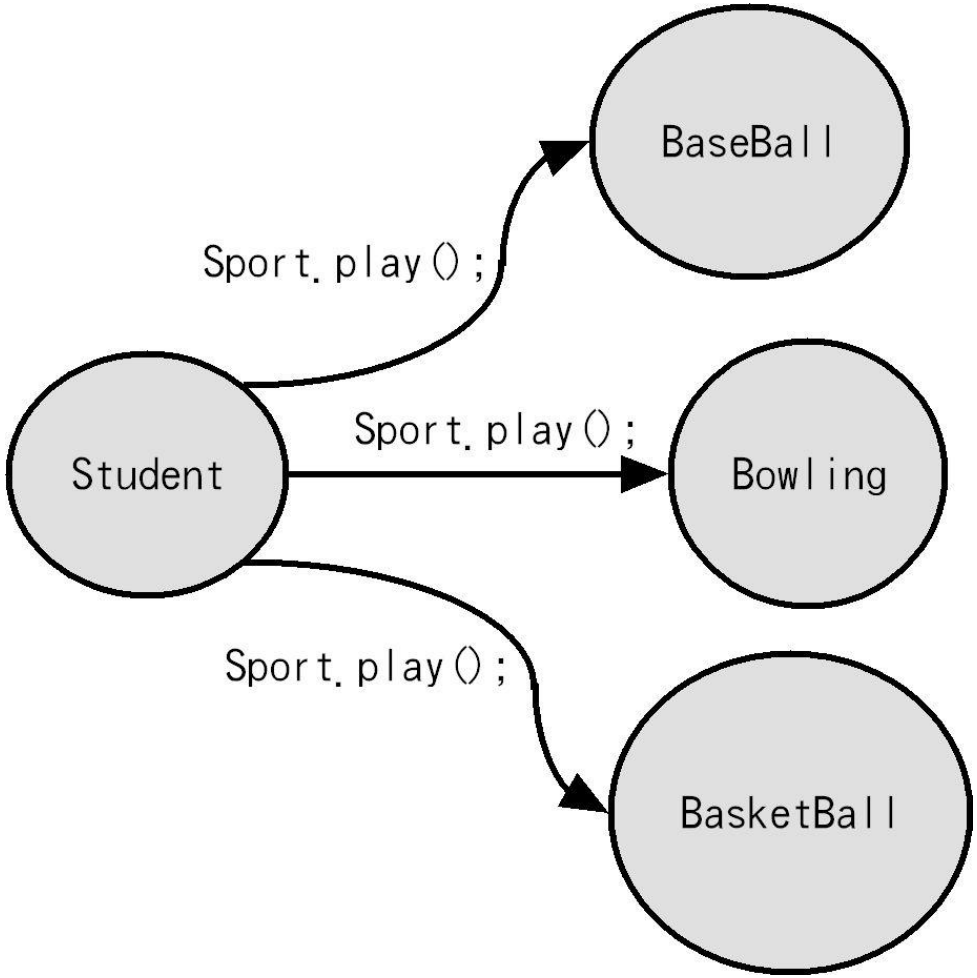
- 在物件導向程式設計的物件是依接收的訊息來執行不同的方法，換句話說，只需訊息不同足以讓物件辨識，一樣可以執行同名的方法。
- 例如：執行Utility物件的max()方法的訊息，如下所示：

```
Utility.max(23, 45);
Utility.max(23, 45, 87);
Utility.max('a', 'z');
```

訊息觀念-多形(說明)

- 「多形」（Polymorphism）是另一種名稱再用，這是指各物件針對同一個訊息擁有不同的反應，也就是同一個名稱擁有不同的操作。
- 因為在人類的思維中，對於同一種工作，就算對象不同，也會使用同名的操作。

訊息觀念-多形(圖例)



訊息觀念-多形(圖例說明1)

- 使用Java程式模擬學生打球，對於Student學生物件來說，都是送出Sport.play();的相同訊息，表示打球，因為「動態連結」（Dynamic Binding）機制，在執行階段才決定訊息真正的接收物件，以此例在執行時送出的3個訊息，如下所示：

```
BaseBall.paly();
Bowling.play();
BasketBall.play();
```

訊息觀念-多形(圖例說明2)

- 訊息是在執行時才決定Sport代表的物件分別為：BaseBall、Bowling和BasketBall，簡單的說，在程式碼送出的是相同訊息，只是執行階段送給了不同的接收物件，分別是打棒球、打保齡球和藍球。
- 因為對於人類來說都是打球Sport.play();，雖然都是play()，但是實際的接收物件不同，所以會執行不同的操作，這種觀念稱為多形，或稱為同名異式。

類別觀念-什麼是類別1

- 類別（Class）是一種分類，將擁有相同特性的物件集合歸類成同一個類別。換句話說，類別就是物件的藍圖，可以用來建立物件。
- 摸擬車輛的Car1、Car2、Car3、Car4…物件都擁有相同屬性和行為，只是狀態不同。這些物件屬於同一類物件，所以可以建立名為Car的範本來建立這些物件，如同工廠依照藍圖製造車輛，此範本就是類別，屬於同一類別的物件即該類別的「實例」（Instance），也稱為副本。

類別觀念-什麼是類別2

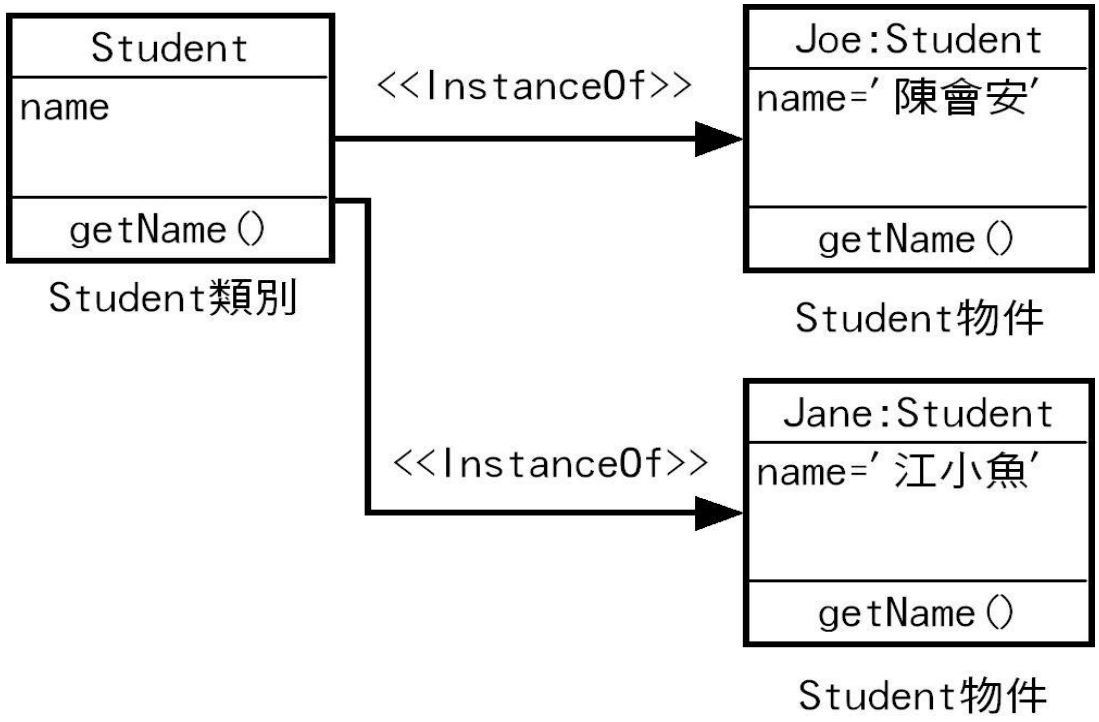
- 類別也可以想像成是扮演的角色。例如：模擬教室上課，在同一間教室擁有30人，其中1位是老師，其他是學生。
- 換句話說，如果每一個人是一個物件，30個物件可以進一步分類成屬於Teacher類別和Student類別的物件集合，也就是哪些物件扮演老師，哪些物件是學生。

類別觀念-物件的藍圖(說明)

- 類別是一種抽象資料型態，其目的是用來建立物件，使用類別建立的物件稱為類別的實例（Instance）。
- 例如：使用Student類別建立29位Student物件，這些物件和類別擁有相同的屬性和行為，只是狀態值不同，即物件的變數值不同。

類別觀念-物件的藍圖(圖例)

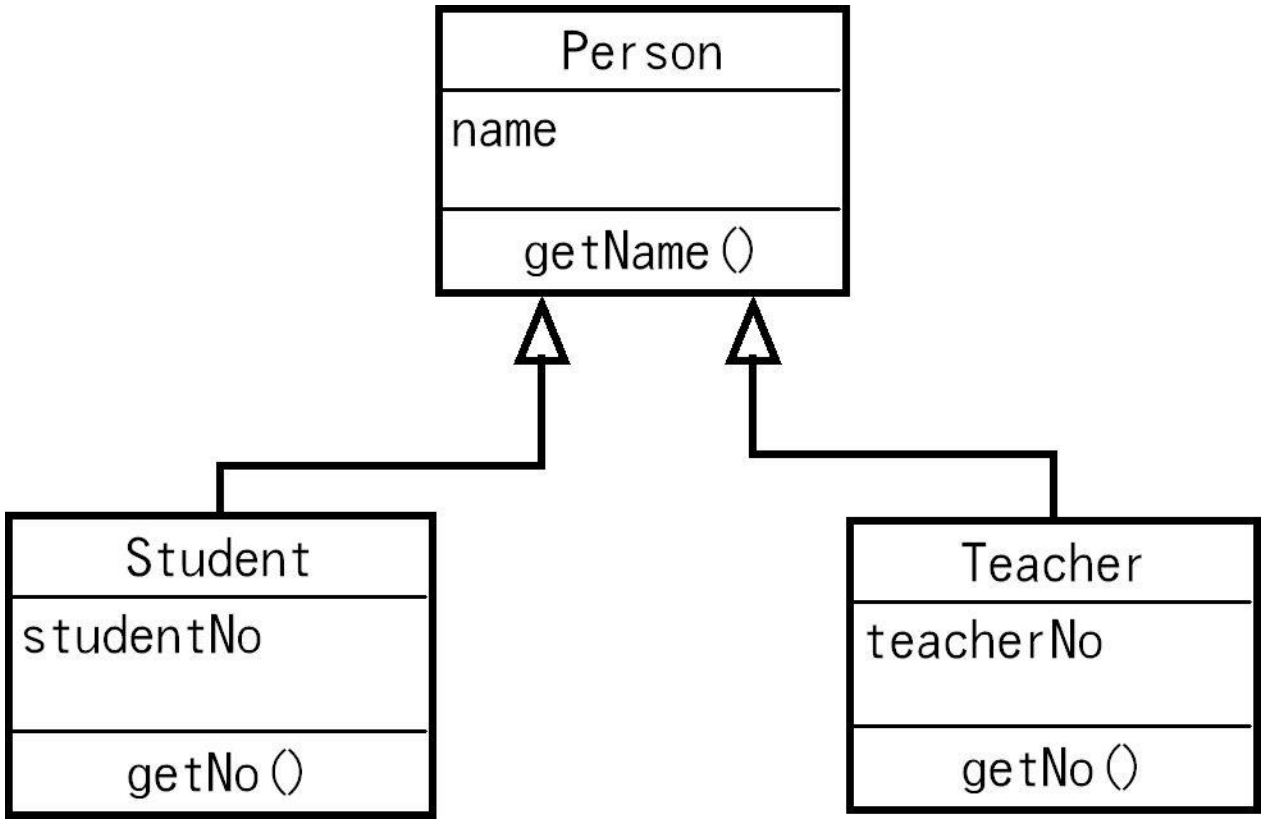
- 例如：一位學生的姓名name是【陳會安】，另一位是【江小魚】，如下圖所示：



類別觀念-繼承(說明)

- 學生和老師都是人，換言之，我們可以先定義Person類別模擬人類，然後擴充Person類別建立Student和Teacher類別模擬學生和老師，稱為「繼承」（Inheritance）。

類別觀念-繼承(圖例)



類別觀念-繼承(圖例說明)

- Student和Teacher類別是繼承自Person類別，我們稱Student和Teacher類別為繼承類別的「子類別」（Subclass）或「延伸類別」（Derived Class），繼承的Person類別稱為「父類別」（Superclass）或「基礎類別」（Base Class）。
- 如果有多個子類別繼承同一個父類別，每一個子類別稱為「兄弟類別」（Sibling Classes）。

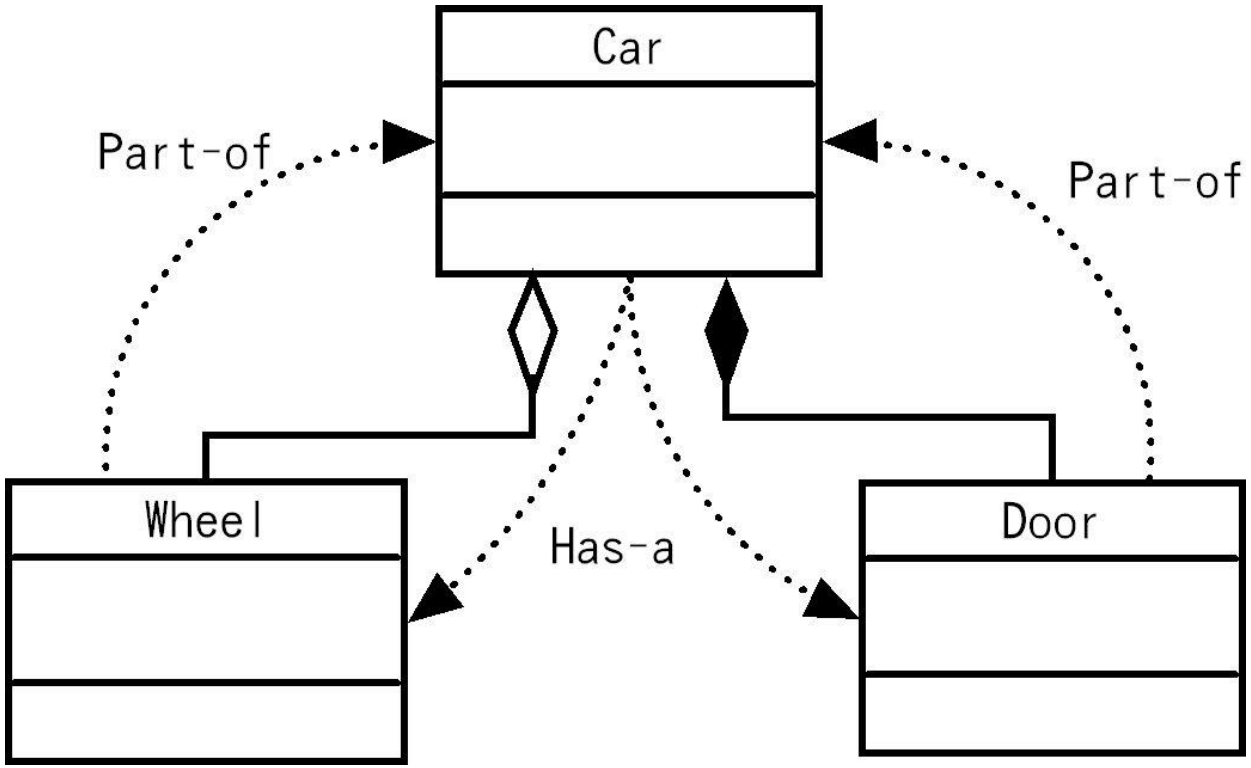
類別觀念-類別架構

- 繼承的子類別可以有多層，如果將整個類別關係的樹狀結構繪出來，就稱為「類別架構」（Class Hierarchy），如果父類別不只一個，即繼承多個類別，稱為「多重繼承」（Multiple Inheritance）。

類別觀念-類別關聯性(說明)

- 類別關聯性（Relationships）可以指不同類別間的關係。例如：繼承是一種Is-a的類別關聯性，在UML稱為「一般關係」（Gereralization）。另外還擁有一種稱為「成品和零件」（Whole-Part）的類別關聯性，即Part-of和Has-a關係。

類別觀念 – 類別關聯性(圖例)



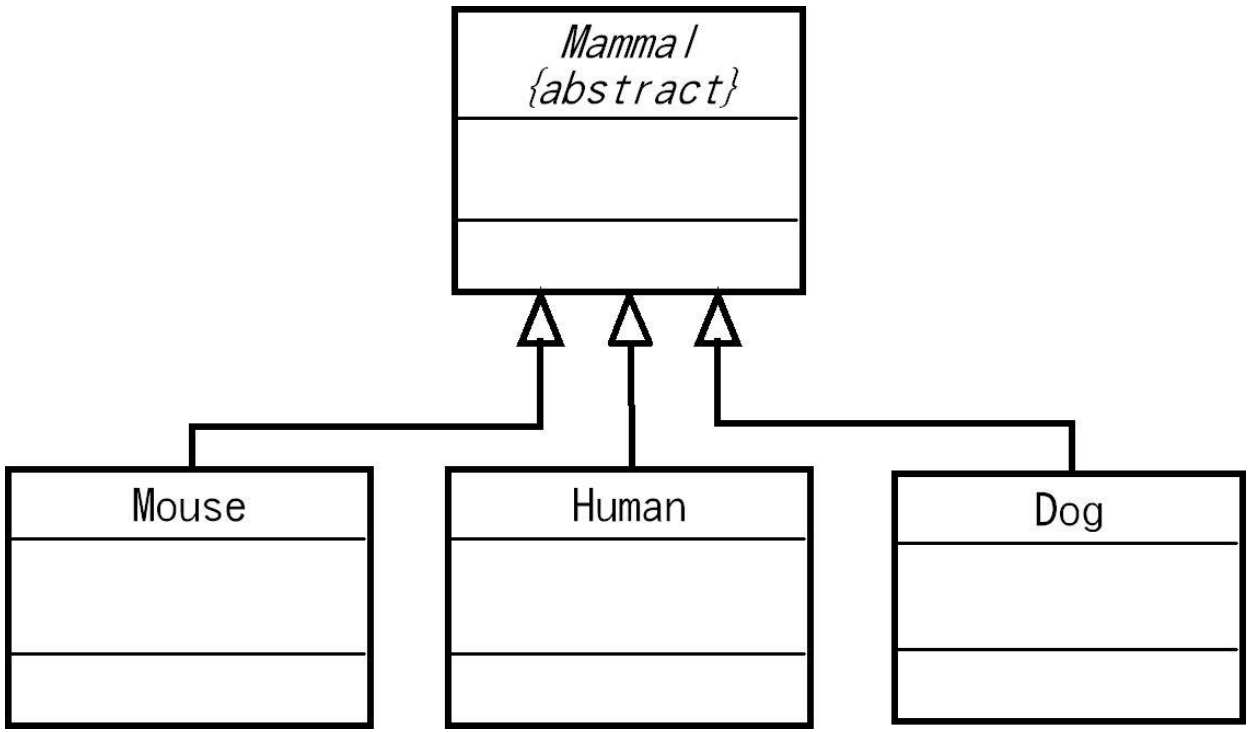
類別觀念-類別關聯性 (圖例說明)

- **Part-of關係**：指此類別是其他類別的零件，以上圖為例Wheel車輪和Door車門是Car車類別的零件。
- **Has-a關係**：相反於Part-of關係，Car類別Has-a擁有Wheel和Door類別。
  - 在UML的上述關係稱為「聚合關係」（Aggregation），或是另一種更強調Whole-part關係稱為「組成關係」（Composition）。

類別觀念-抽象類別(說明)

- 「抽象類別」（Abstract Class）是一種不能完全代表物件的類別，換句話說，它並不能用來建立物件副本，抽象類別擁有眾多類別的共同部分，主要的目是是作為其它類別的父類別，例如：哺乳類動物的分類。

類別觀念 – 抽象類別 (圖例)



類別觀念-抽象類別 (圖例說明)

- 類別架構的父類別是哺乳類（Mammal），Mouse、Human和Dog類別是繼承自Mammal類別，因為老鼠、人和狗都屬於哺乳類動物，我們可以使用Mouse、Human和Dog類別建立模擬老鼠、人和狗等物件。
- 事實上，並沒有任何動物叫哺乳類，所以並不會建立模擬哺乳類的物件，這個類別只是描述哺乳類動物的共同特徵，以便其它屬於哺乳類的動物繼承此物件，換句話說，Mammal類別就是一個抽象類別。