**POSD20171117期中考**

[兩大原則 3](#_Toc498651701)

[Program to an interface, not an implementation. 3](#_Toc498651702)

[Favor object composition over class inheritance. 3](#_Toc498651703)

[Composite 4](#_Toc498651704)

[Intent目的 5](#_Toc498651705)



[Motivation動機 5](#_Toc498651706)



[Applicability時機 5](#_Toc498651707)



[Consequences效果 6](#_Toc498651708)



[Solution解決方法 6](#_Toc498651709)



[Interpret 7](#_Toc498651710)

[[next: 
Motivation](http://www.cs.unc.edu/~stotts/GOF/hires/pat5c.htm#motivation) Intent 7](#_Toc498651711)

[[next: Applicability](http://www.cs.unc.edu/~stotts/GOF/hires/pat5c.htm#applicability) Motivation 7](#_Toc498651712)

[[next: 
Structure](http://www.cs.unc.edu/~stotts/GOF/hires/pat5c.htm#structure) Applicability 7](#_Toc498651713)

[Consequences 8](#_Toc498651714)



[Proxy 9](#_Toc498651715)

[[next: 
Also Known As](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#alsoknownas) Intent 9](#_Toc498651716)

[[next: Applicability](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#applicability) Motivation 9](#_Toc498651717)

[[next: 
Structure](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#structure) Applicability 10](#_Toc498651718)

[[next: Implementation](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#implementation) Consequences 10](#_Toc498651719)

[Builder 11](#_Toc498651720)

[[next: 
Also Known As](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#alsoknownas) Intent 11](#_Toc498651721)

[[next: Applicability](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#applicability) Motivation 11](#_Toc498651722)

[[next: 
Structure](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#structure) Applicability 12](#_Toc498651723)

[[next: Implementation](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#implementation) Consequences 12](#_Toc498651724)

[CH1:Design Pattern如何解決設計問題 13](#_Toc498651725)

[尋找適當的物件 13](#_Toc498651726)

[決定物件的規模大小 13](#_Toc498651727)

[制定物件介面 13](#_Toc498651728)

[制定物件實作 14](#_Toc498651729)

[再利用機制上陣 14](#_Toc498651730)

[執行期與編譯期結構之別 15](#_Toc498651731)

[補充 16](#_Toc498651732)

# 兩大原則

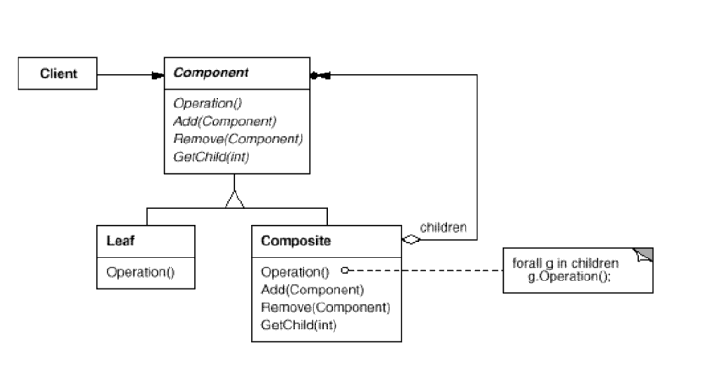
## Program to an interface, not an implementation.

使用者不需知道所用物件的型別，只需要知道物件符合預期的介面即可 使用者不需知道所用物件的類別，只需要知道介面是由哪個抽象類別定義 表示程式不需要在 compile 的時候決定物件的型態,根據子類別回傳的物件再決定型態

## Favor object composition over class inheritance.

繼承是 OOP 裡最強烈的耦合(Coupling)關係,當你動到父類別時,將影響底下所有的子類別。 繼承的關係越長、越複雜時,所造成的影響就會越大、越難維護。

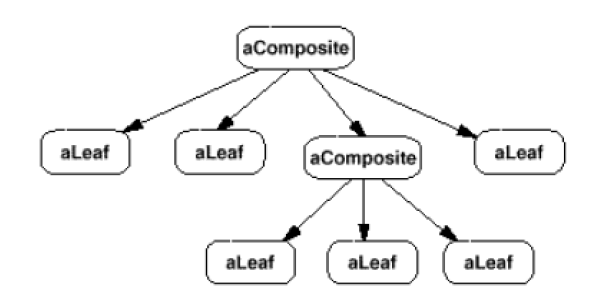
# Composite



* 表達『部分-全體』關係，讓外界以一致性的方式對待個別物件和整體物件。
* 使用遞迴複合技巧使client不必區分這些物件。
* Composite關鍵在於用一個抽象類別同時代表個別物件與物件容器。
* 無需考慮基本物件與複合物件的差異，以一致的方式處理複合結構裡的物件。
* 容易新增新的composite或leaf類別而不需修改client程式。(Open-Closed)
* Component介面極大化，為了不讓client知道面對的是leaf或composite，Component會盡可能涵蓋兩個都可能有的操作。
* Add/Remove是否該擺在component上？

1. 若選擇在 Component 中,對所有的 Component(包含 leaf node 和 composite)可以一視同仁提升通透性，但是卻犧牲了安全性，因為對於 leaf node 而言無法避免被惡意撰寫「新增、刪除」的惡搞程式，所以會在leaf上讓它們的預設版本失敗或者 拋出例外。
2. 若選擇在 Composite 中,可以在編譯時期找出「新增、刪除」leaf node 程式的錯誤，但卻失去通透性。

複合物件結構圖



## Intent目的



Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

## Motivation動機



The Composite pattern describes how to use recursive composition so that clients don't have to make this distinction.

The key to the Composite pattern is an abstract class that represents both primitives and their containers.

## Applicability時機



Use the Composite pattern when

* you want to represent part-whole hierarchies of objects.

想表達部分-全體的物件關係時。

* you want clients to be able to ignore the difference between compositions of objects and individual objects. Clients will treat all objects in the composite structure uniformly.

想讓客戶可以忽略複合物件和基本物件之間的差異，能以一致的方式處理複合結構裡的物件時。

## Consequences效果



The Composite pattern

* defines class hierarchies consisting of primitive objects and composite objects.

定義包含基本物件和複合物件的類別階層。

* makes the client simple.

使客戶的程式碼簡單化。

* makes it easier to add new kinds of components.

加入新的元件變得容易，且原有物件不受影響。

* can make your design overly general.

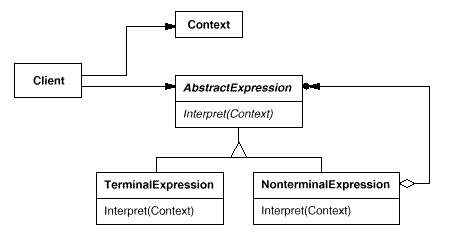
可能讓你的設計過於一般化。(缺點)

## Solution解決方法



定義一個元件介面(interface)其包含物件之共同功能，以及用來維持part-whole階層式結構的功能。所有物件必須實作此介面並維持其遞迴(recursive)的組成關係。

# Interpret



## [next: Motivation](http://www.cs.unc.edu/~stotts/GOF/hires/pat5c.htm#motivation) Intent

Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

## [next: Applicability](http://www.cs.unc.edu/~stotts/GOF/hires/pat5c.htm#applicability) Motivation

The Interpreter pattern describes how to define a grammar for simple languages, represent sentences in the language, and interpret these sentences. Interpreter旨在如何定義簡單的語言語法、語句表達方式、以及語句的解讀方式。

The Interpreter pattern uses a class to represent each grammar rule.針對每一條文法規則個定義一個類別。

## [next: Structure](http://www.cs.unc.edu/~stotts/GOF/hires/pat5c.htm#structure) Applicability

* the grammar is simple.

文法簡單

* efficiency is not a critical concern.

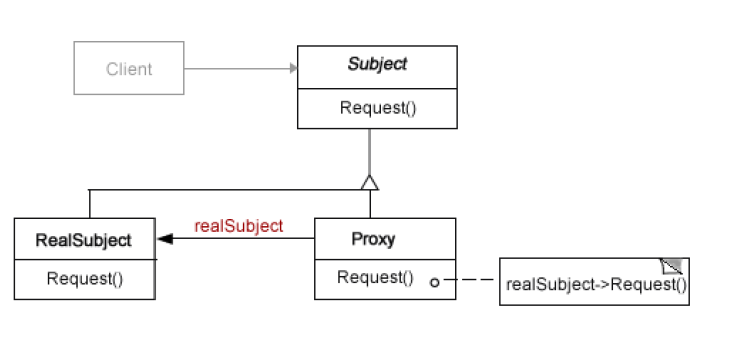
效率不重要

## Consequences



1. *It's easy to change and extend the grammar.*
2. *Implementing the grammar is easy, too.*
3. *Complex grammars are hard to maintain.*
4. *Adding new ways to interpret expressions.*

# Proxy



* 替其他物件預留代理者空位，藉此控制存取其他物件。
* 延緩生成及初始化物件所需付出的成本，等到真正需要時才將耗資源的物件建立起來。(Virtual Proxy)

## [next: Also Known As](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#alsoknownas) Intent

Provide a surrogate or placeholder for another object to control access to it.

## [next: Applicability](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#applicability) Motivation

One reason for controlling access to an object is to defer the full cost of its creation and initialization until we actually need to use it.延緩生成及初始化物件所需付出的成本，直到真正要用時才去承擔。

These constraints would suggest creating each expensive object *on demand*, which in this case occurs when an image becomes visible.等到需要時才將耗資源的物件建立起來。

The solution is to use another object, an image **proxy**, that acts as a stand-in for the real image.

## [next: Structure](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#structure) Applicability

1. A **remote proxy** provides a local representative for an object in a different address space.

遠端代理者用來代理位於不同定址空間的其他物件。

1. A **virtual proxy** creates expensive objects on demand.

虛擬代理者會在需要時才建立消耗資源的物件。

1. A **protection proxy** controls access to the original object.

設限代理者控制原始物件的存取權限、用來對同一物件設定各種不同的存取權限。

1. A **smart reference** is a replacement for a bare pointer that performs additional actions when an object is accessed.

智慧型指涉者彌補指標的不足，可在存取物件的同時進行額外的動作。

## [next: Implementation](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#implementation) Consequences

1. A remote proxy can hide the fact that an object resides in a different address space.

遠端代理者可隱瞞物件位不同定址空間一事。

1. A virtual proxy can perform optimizations such as creating an object on demand.

虛擬代理者做些最佳化措施，像是需要時才建立物件。

1. Both protection proxies and smart references allow additional housekeeping tasks when an object is accessed.

設限代理者和智慧型指涉者都可在存取物件時做些額外的家務事。

# Builder



* 從複雜物件中取出生成程序,以便在同一個生成程序製造各種不同的物件。
* 若要將『建造複雜物件的演算法』和物件零件與組裝方式保持獨立時使用Builder。
* 若要讓同一個物件生成程序能產生數種不同形式物件時使用Builder。
* 將生成程序與內部布局的程式碼隔離，builder將物件生成及佈局方式封裝起來，提高模組化程度。
* 對生成程序的掌控更細膩，成品物件是在Director的監控下一步步建出來，等到完全建好後，Director才向builder索取成品物件，與其他一個步驟建出成品的生成樣式不一樣。
* 預設情況下，Builder操作只是空殼子，不將builder系列操作宣告成pure virtual，只定義成空殼子操作，只需複寫需要的操作即可，無須全部複寫。

## [next: Also Known As](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#alsoknownas) Intent

Separate the construction of a complex object from its representation so that the same construction process can create different representations.

## [next: Applicability](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#applicability) Motivation

A reader for the RTF (Rich Text Format) document exchange format should be able to convert RTF to many text formats. The reader might convert RTF documents into plain ASCII text or into a text widget that can be edited interactively. The problem, however, is that the number of possible conversions is open-ended. So it should be easy to add a new conversion without modifying the reader.將RTF轉換成其他文件格式，可能轉換成ASCII文字檔，也可能轉換成文件視窗元件給用戶編輯。但是文件格式和轉換動作都是無窮無盡的多，要想個一勞永逸的方法，即使日後有新的文件格式出現也不必動手修改。

Each converter class is called a **builder** in the pattern, and the reader is called the **director**. Applied to this example, the Builder pattern separates the algorithm for interpreting a textual format (that is, the parser for RTF documents) from how a converted format gets created and represented.每個轉換者類別的腳色為builder，閱讀程式的角色為director。

## [next: Structure](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#structure) Applicability

Use the Builder pattern when

* the algorithm for creating a complex object should be independent of the parts that make up the object and how they're assembled.

欲將「建造複雜物件的演算法」與物件的零件及組裝方式保持獨立時。

* the construction process must allow different representations for the object that's constructed.

想讓同一個物件生成程序能夠產生數種不同布局形式的物件時。

## [next: Implementation](http://www.cs.unc.edu/~stotts/GOF/hires/pat4g.htm#implementation) Consequences

Here are key consequences of the Builder pattern:

1. *It lets you vary a product's internal representation.*

能夠改變成品物件的內部布局形式。

1. *It isolates code for construction and representation.*

將生成程序與內部布局的程式碼分開來。

1. *It gives you finer control over the construction process.*

對生成程序掌控得更加細緻。

# CH1:Design Pattern如何解決設計問題

## 尋找適當的物件

物件將資料和可施於其上的程序包裝再一起，此處的程序通常叫做method或operation。當物件收到client送來的request或message就會執行對應的操作。

Message是唯一可令物件執行操作的管道，operation則是唯一可改變物件內部狀態的管道。有了這些限制，物件內部狀態可以說是被妥為封裝(encapsulation)起來，無法接觸及自外界窺伺。

## 決定物件的規模大小

該如何決定是切的物件規模？設計模式也能幫你解決此一問題。(各種pattern可以幫你決定)

## 制定物件介面

物件所宣告的每一個操作，都得訂出該操作的名字、參數、回傳值，也是該操作的**外貌簽名**(signature)。所有已訂定的物件操作之外貌簽名稱之為此物件的**介面**(interface)

物件的介面，界定出能送給此物件的訊息類型，只有符合物件介面的外貌簽名，才能送給此物件。

介面也可包含其他介面，當型別A是型別B的**子型別**(subtype)時，A的介面會包含A的**父型別**(supertype)B所具備的介面，通常我們會說A**繼承**(inherit)B的介面。

若不知道介面，便無從得知物件的資訊，也無法進行操作。

執行期決定該將此一要求導引到哪個物件哪個操作的機制就叫做**動態繫結**(dynamic binding，你所送出的訊息直到執行期才會知道送到哪一個實作體上)。動態繫結更讓你能夠在執行期以同一介面的多個物件交互替換，這個替換性質就叫做**多型**(polymorphism)，是物件導向系統的重要觀念。

## 制定物件實作

**類別**定義出物件的內部資料和布局、物件所能執行的操作。

**抽象類別**(abstract class)主要目的是替多個子類別訂出共通介面。抽象類別會將部分或全部的實作細節交付給子類別的操作去實現，因此抽象類別無法具現出物件個體。抽象類別有宣告但未實作的操作叫做**抽象操作**(abstract operation)，與抽象類別相反叫做**具象類別**(concrete class)。

子類別可**覆寫**(override)親代類別所賦予的操作。

**針對介面而寫，不要針對實作。**

**Program to an interface, not an implementation**

定義一整族具備同一介面的物件(通常是以繼承某一抽象類別的方式實作)也是相當重要的。為什麼？因為多型需要它。

## 再利用機制上陣

**類別繼承**是某一物件來定義另一物件的實作細節，通常稱為**白箱式再利用**(white-box reuse)，白箱指的是能見度，在繼承機制下子類別常可透視親代類別的內部。

**物件複合**是將既有物件組合、複合起來，以拼出更複雜的功能。被選來組裝的物件，必須有明確定義的介面。這種再利用方式稱為**黑箱式再利用**(block-box reuse)，各物件內部細節全都隱匿看不見。

類別繼承缺點：

1. 無法在執行期才變更繼承父類別的實作細節，因為繼承是在編譯時期就定義好。
2. 複類別通常指定一子類別所需之實體布局當中的一部分而已，但是繼承會將赴類別的內部細節暴露給子類別看，所以繼承會破壞封裝性質。

解決方法是指繼承抽象類別，因為通常抽象類別幾乎不含實作部分。

物件複合還有一個作用，他封裝的比類別繼承更好。

**多用物件複合技術，少用類別繼承。**

**Favor object composition over class inheritance**

**委託**(delegation)讓複核技術和繼承依樣強大有力。在此技術下有兩個物件協同處理訊息要求，訊息接收者物件會將訊息操作委託給被委託者(delegate)去全權處理。

優點：更容易在執行其拼裝及改變行為。

缺點：委託和複合依樣有著動態、高度參數化的軟體，比靜態的軟體更難理解，執行的也慢。

## 執行期與編譯期結構之別

**聚合**(aggregation)指物件A只有或擔負另一物件B，通常說A**擁有**(have)B，或是B是A的**一部分**(part of)。※聚合也意味著被聚合B和他的擁有A的生命週期完全一樣，UML把這種嚴格的生命週期性質叫做composition：aggregation的特例。

**熟識**(acquaintance)指物件A只**知悉**(know of)另一物件B，有時稱為**連繫**關係(association)或**使用**關係(using)，可以說是比聚合關係更薄弱、物件耦合性更低。

# 補充

所謂「物件導向的特色」，指的是物件導向的三個機制，三者缺一不可，此三個機制分別為封裝（encapsulation）、繼承（inheritance）、多型（polymorphism）。

1. 封裝 (Encapsulation) : 隱藏某一方法的具體執行步驟，把過程和資料包起來，對資料的操作只能通過已定義的界面。
   * 介面 (Interface)：用以定義物件之外觀行為。
   * 實作 (Implementation)：用以存放抽象化結果及描述如何達成外觀行為。
2. 繼承 (inheritance) : 子類別會繼承父類的方法和屬性。子類別比父類別要更加具體化，如鸚鵡之於鳥。
3. 多型 (Polymorphism) : 定義具有不同功能但名稱完全相同之方法或屬性的類別。
   * 靜態多型 (Static polymorphism/Overloading) : An invocation can be operated on arguments of more than one type.
   * 動態多型 (Dynamic polymorphism/Overriding) : A same operation may behave differently on different classes.