設計模式 (Design Patterns)

大哉問 大局觀

侯 捷

2021/08/13 西安 / 華為 ICT (Information & Communication Technology) 大會

不易 流行

— 松尾芭蕉

技術修煉不能只是追新，對於 "基石" 必須深入掌握 。

設計模式 之 迷霧森林 (What? Why? How?)

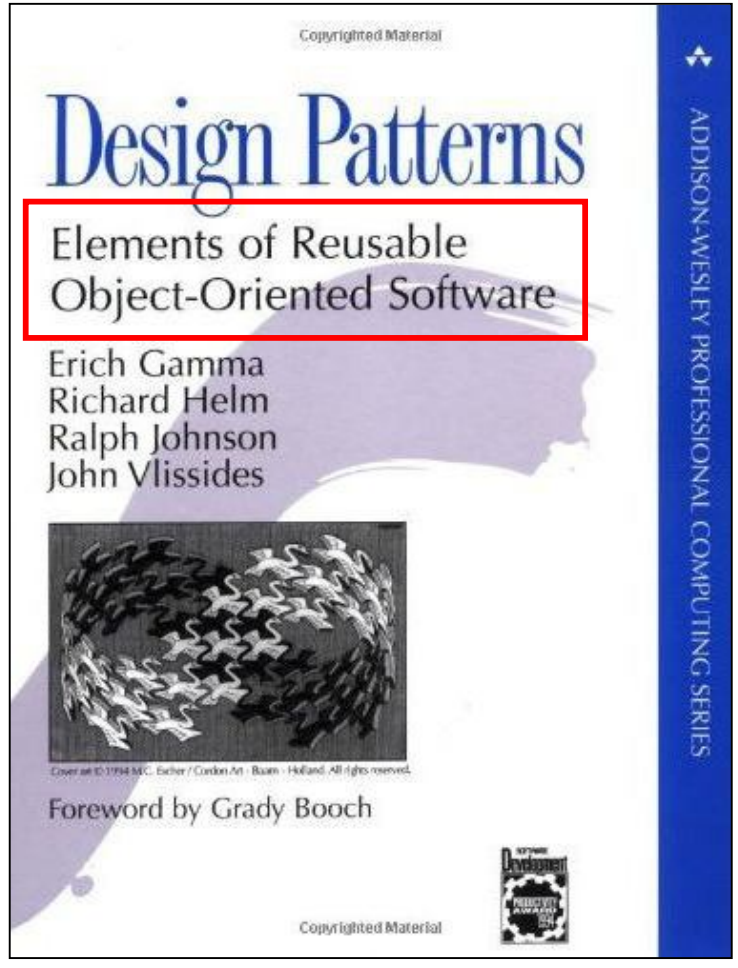
- 它是什麼?
- 它到底是什麼?
- 它和編程語言無關嗎?
- 似乎大家都在談它！
- 似乎不懂它就掉漆?
- 我需要學習它嗎?
- 我能夠學習它嗎?
- 我該怎麼學習它?

Elements of Reusable Object-Oriented Software

可復用 (之) 面向對象軟件之元素

最重要： 虛函數 (virtual functions)，
它可以 "穿越"。

就像面對多線程 (multi-threading)，
我們要時時注意把正確的觀念帶入
代碼。



牽一髮而動全局

經常地，永遠地，思考...

OCP

不是 Open Compute Project

而是 Open Closed Principle

OCP (Open-Closed Principle)，開放-封閉 原則

In object-oriented programming, the open–closed principle states "software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification"; that is, such an entity can allow its behavior to be extended without modifying its source code.

也就是說，這樣的 entities 得以在源碼不變的情況下擴展行為。

OCP (Open-Closed Principle)，開放-封閉 原則

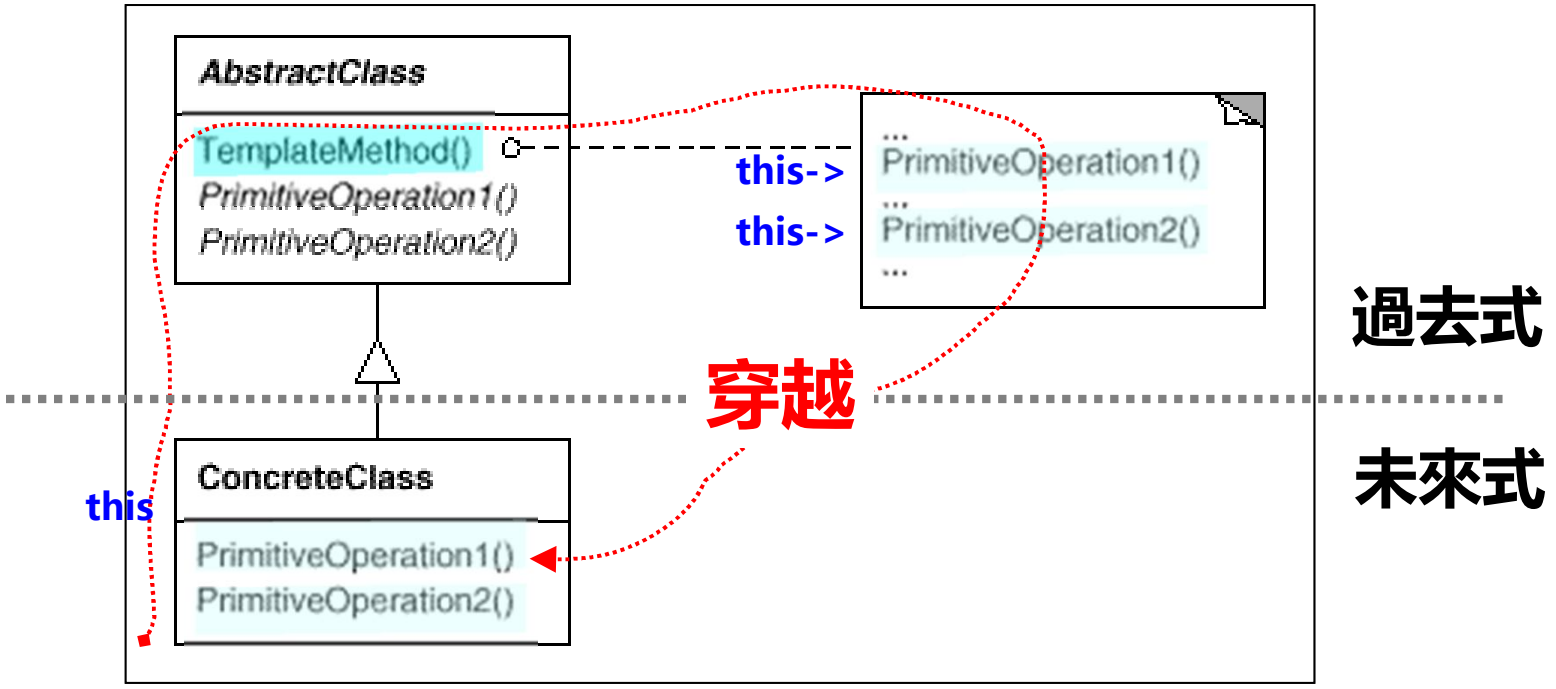
在任何 OOPL 中都可以創造出「固定且代表一組無限可能的行為」的抽象。這些抽象就是 abstract base classes，而那組無限可能的行為則以所有可能的 derived classes 來表現。

所有系統在其生命週期中都會改變。如何才能在面對變動的情況下創造穩定且可長久延續的設計呢？如果 OCP 應用得宜，未來的變動是以增添新代碼達成，而不是修改運作正常的舊代碼。

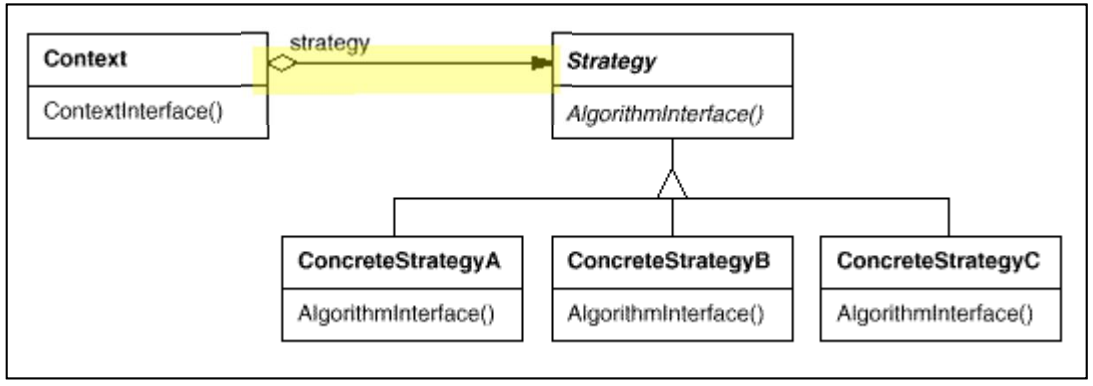
實現 OCP 的二個基礎手段：Template Method 和 Strategy

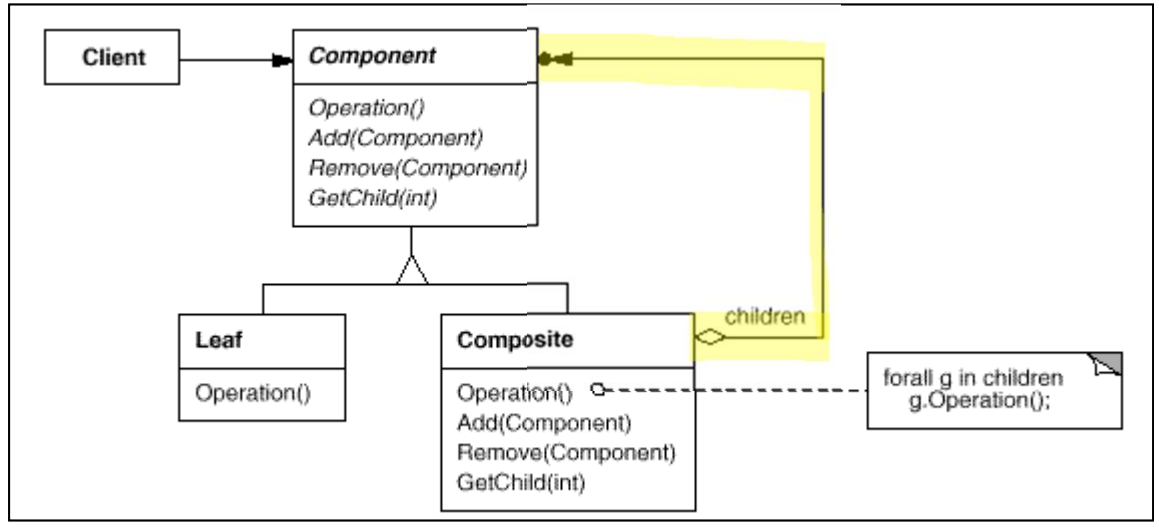
簡言之 虛函數 (virtual functions) 和多態 (polymorphism; ptrs to a base of hierarchy)

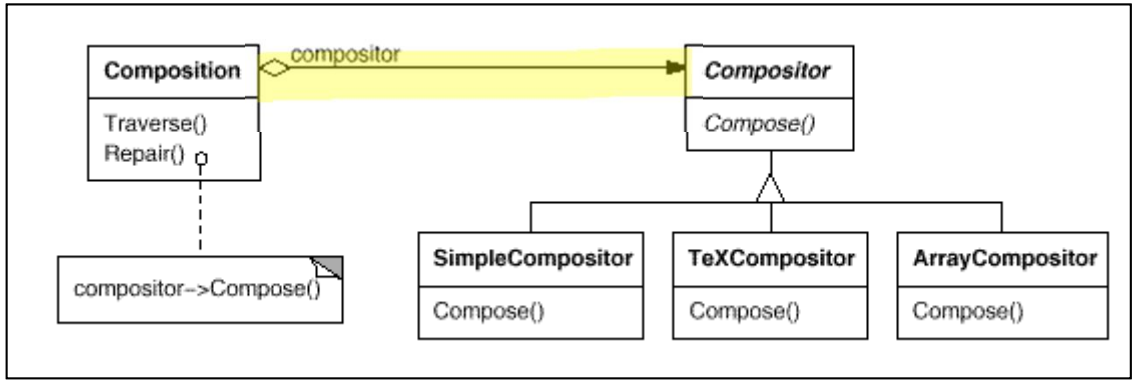
設計模式，基礎手段之一 Template Method

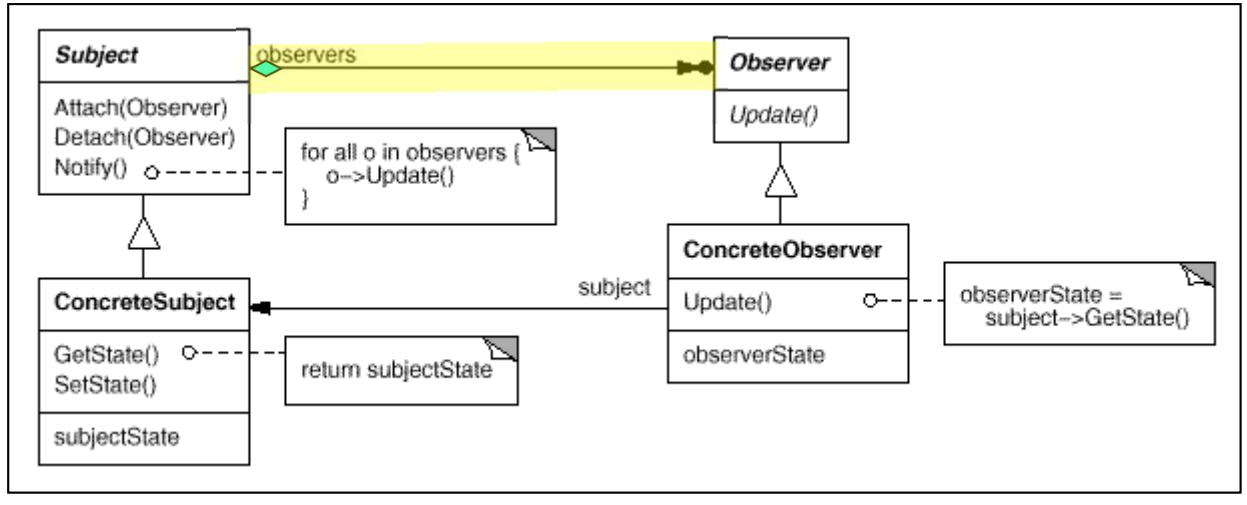


設計模式，基礎手段之二 Strategy





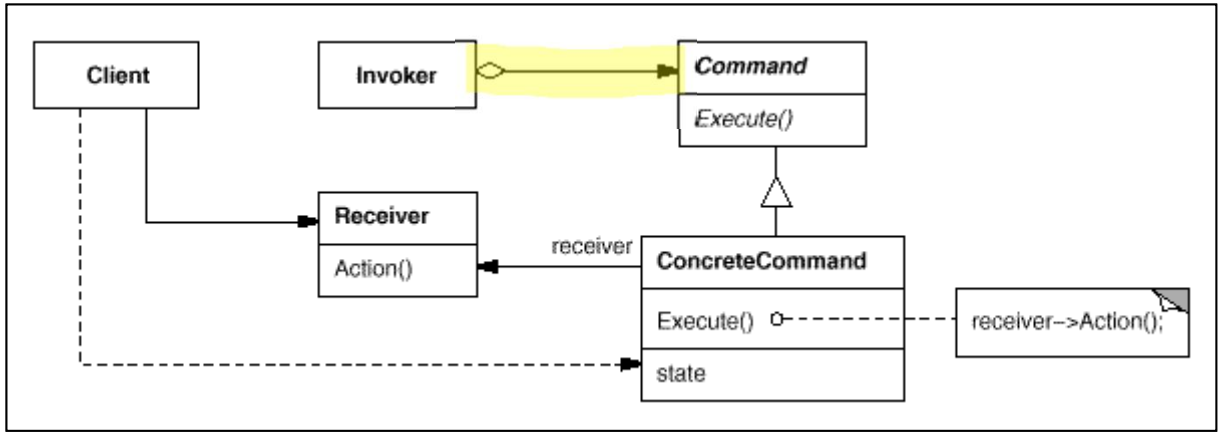


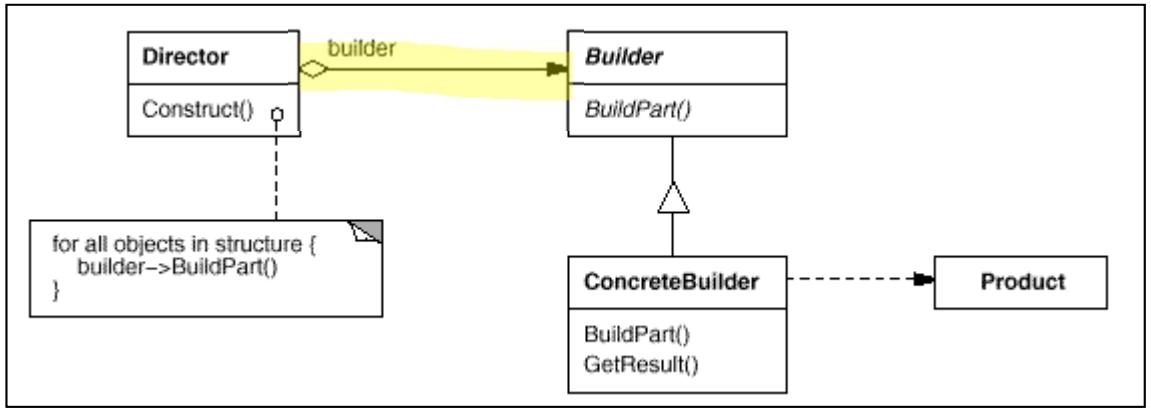


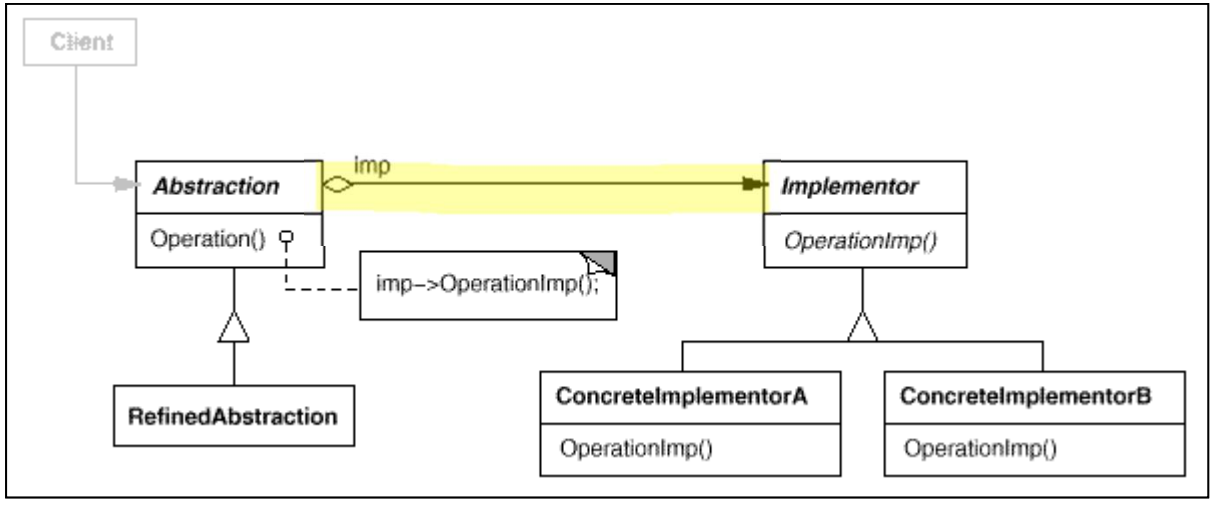
千篇一律的手段
(多態)

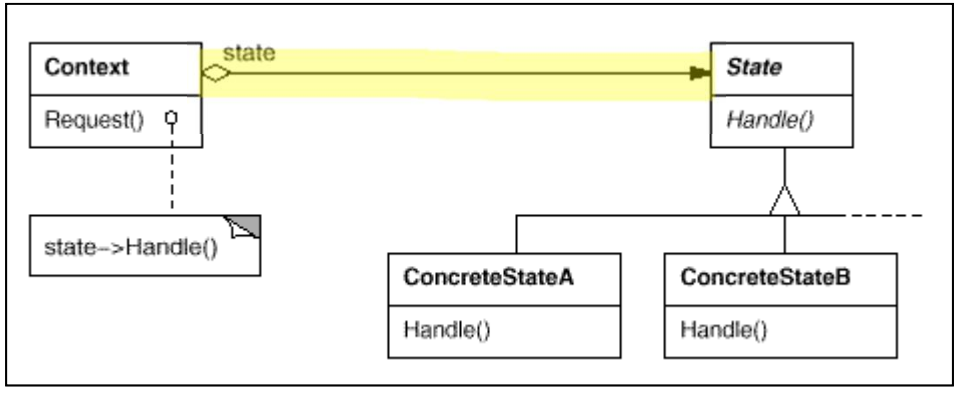
設計模式，基礎手段之二 Strategy

族繁不及備載





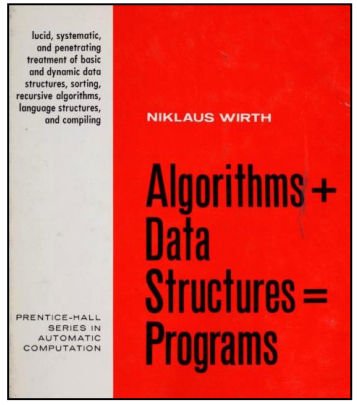




編程金句

算法 + 數據結構 = 程序

(Algorithms + Data Structures = Programs)

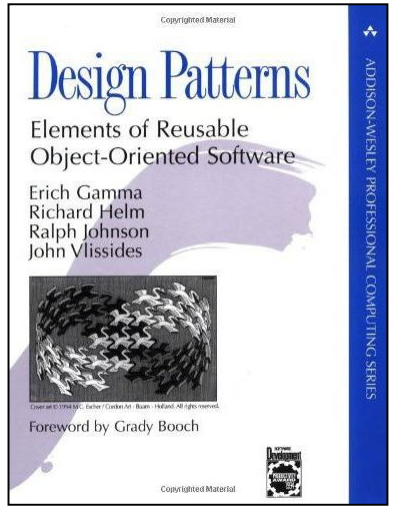


中間層 就是 銀彈/利器 (強而有力、適應各種場合的解決方案)

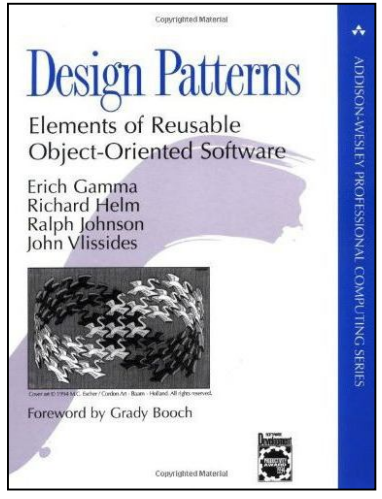
(The middle layers are the silver bullets)

面向接口編程，而非面向實現。

(Program to an interface, not an implementation)



program to an interface, not an implementation.



(p.18)

There are two benefits to manipulating objects solely in terms of the interface defined by abstract classes:

1, Clients remain unaware of the specific types of objects they use, as long as the objects adhere to the interface that clients expect.

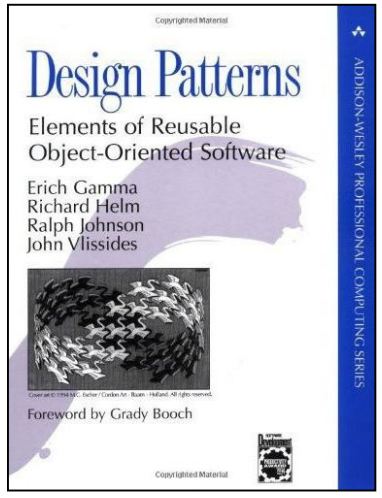
2, Clients remain unaware of the classes that implement these objects. Clients only know about the abstract class(es) defining the interface.

This so greatly reduces implementation dependencies between subsystems that it leads to the following principle of reusable object-oriented design:

Program to an interface, not an implementation.

Don't declare variables to be instances of particular concrete classes. Instead, commit only to an interface defined by an abstract class. You will find this to be a common theme of the design patterns in this book.

program to an interface, not an implementation.



(p.18)

若能只依據 abstract classes 定義出來的 interface 來操作 objects，Client 便得以保持二個未知：

1, Client 不(必)知道其所使用的 objects 的明確類型，只要這些 objects 嚴守 client 所期望的 interface 即可。

2, Client 不(必)知道這些 objects 歸屬哪些 classes，只(需)知道哪些 abstract classes 定義了該 interface。

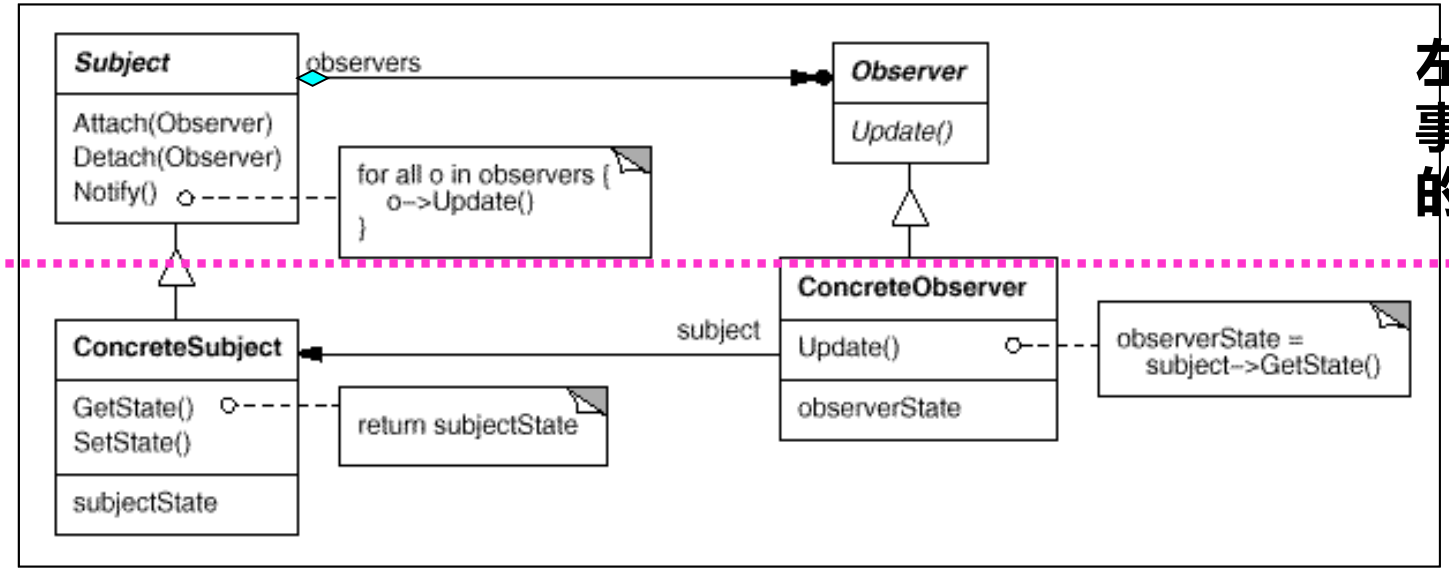
這將大幅降低 subsystems 之間的實現相依性，從而引領出下面這個守則：

面向接口編程，而不是面向實現編程。

不要將變量聲明為某個特定具象類的實體(instance)，應該只將它聲明為遵循某抽象類所定義的接口，這便是本書所說的設計模式的共同主題。

program to an interface, not an implementation，例一

Observer



左圖幾可寫入代碼庫。事實是：Java 標準庫的確如此。

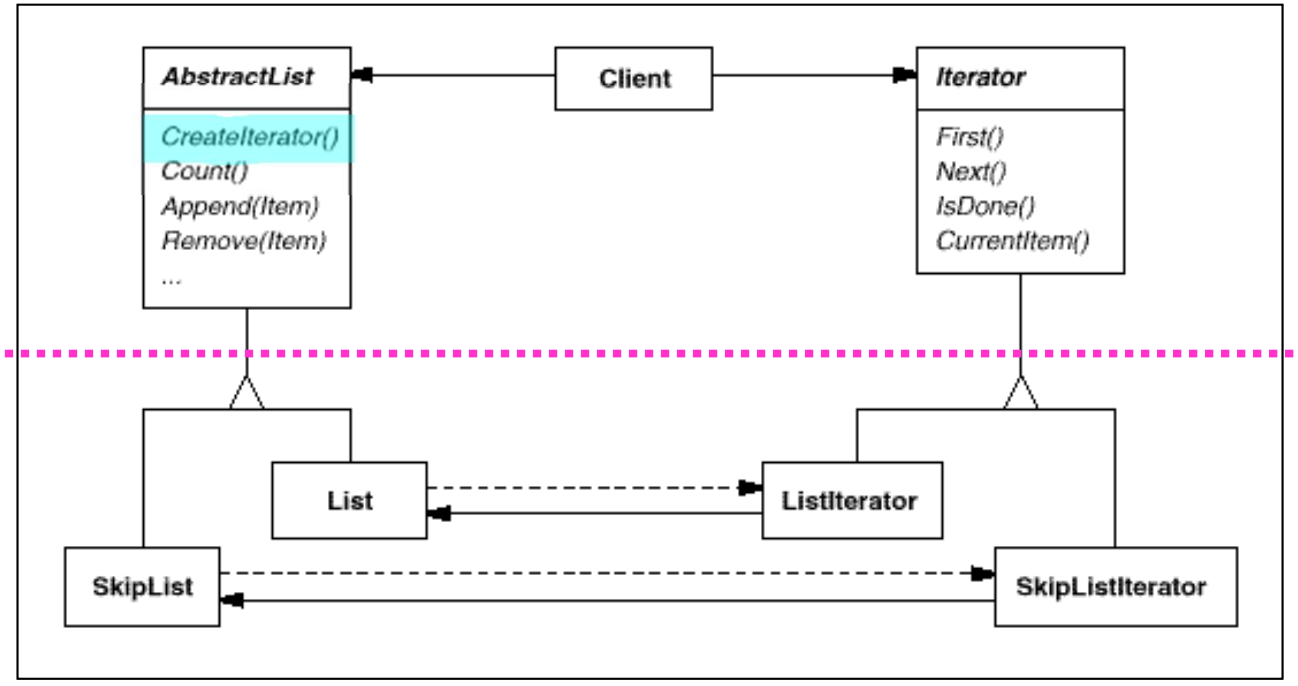
只依據 abstract classes 定義出來的 interface 來操作 objects，Client 得以保持二個未知：

1, Client 不(必)知道其所使用的 objects 的明確類型，只要這些 objects 嚴守 client 所期望的 interface 即可。

2, Client 不(必)知道這些 objects 隸屬哪些 classes，只(需)知道是哪些 abstract classes 定義了該 interface。

program to an interface, not an implementation，例二

Iterator



左圖幾可寫入代碼庫。事實是：Java 標準庫的確如此。

只依據 abstract classes 定義出來的 interface 來操作 objects，Client 得以保持二個未知：

1, Client 不(必)知道其所使用的 objects 的明確類型，只要這些 objects 嚴守 client 所期望的 interface 即可。

2, Client 不(必)知道這些 objects 隸屬哪些 classes，只(需)知道是哪些 abstract classes 定義了該 interface。

觀念整理

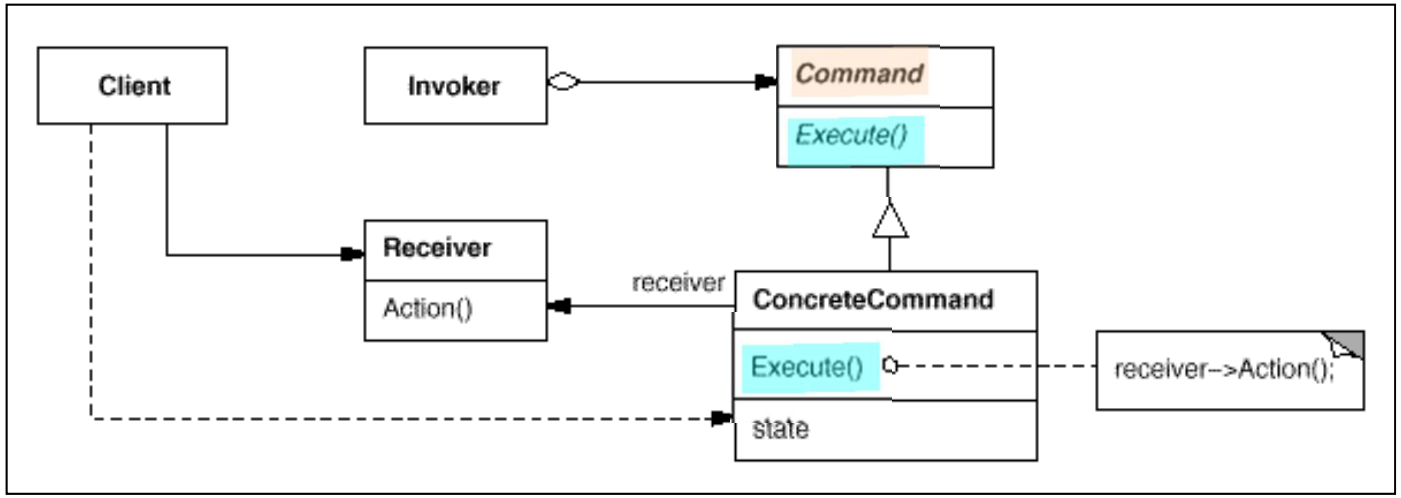
- 所謂"設計模式"，就是前人(1996)整理下來，在設計框架時 所面臨的常見問題 的共通、高效、有實用價值 的解決方案。軟件界最基本最知名有23個設計模式。
- 學習設計模式，從上層說，就是首先學習那23個設計模式。
- 從根本說，則是學習 "面向接口編程" (program to an interface)。
- 面向接口編程，為的是實現 OCP，避免牽一髮動全局；在避免修改代碼的同時，又具備擴展功能的能力。
- OCP 的兩大根本手段，一是 Template Method，一是 Strategy。
- 從面向對象語言的層面看，一是虛函數，一是多態。
- 掌握這二個根本手段，就洞析了所有設計模式的技術原理，剩下的只是這個設計模式的"題域"。

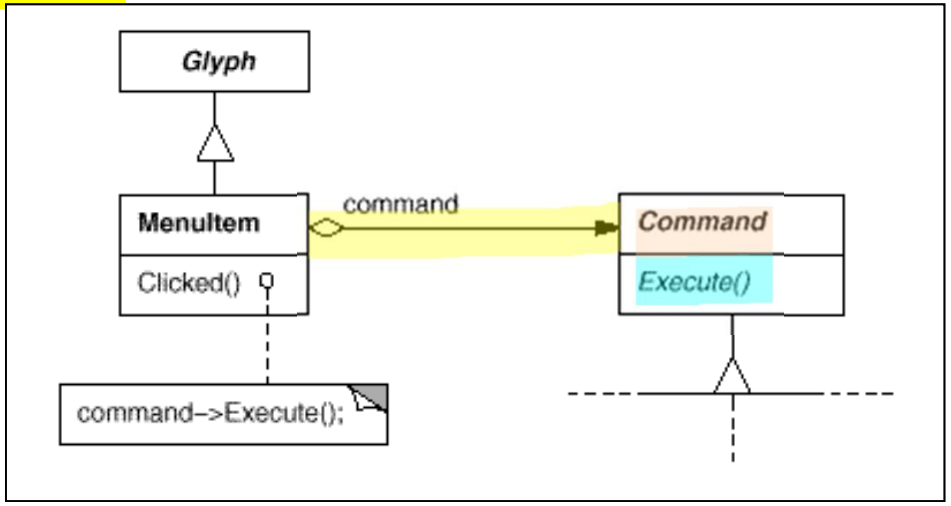
設計模式，學習曲線 與 學習困境

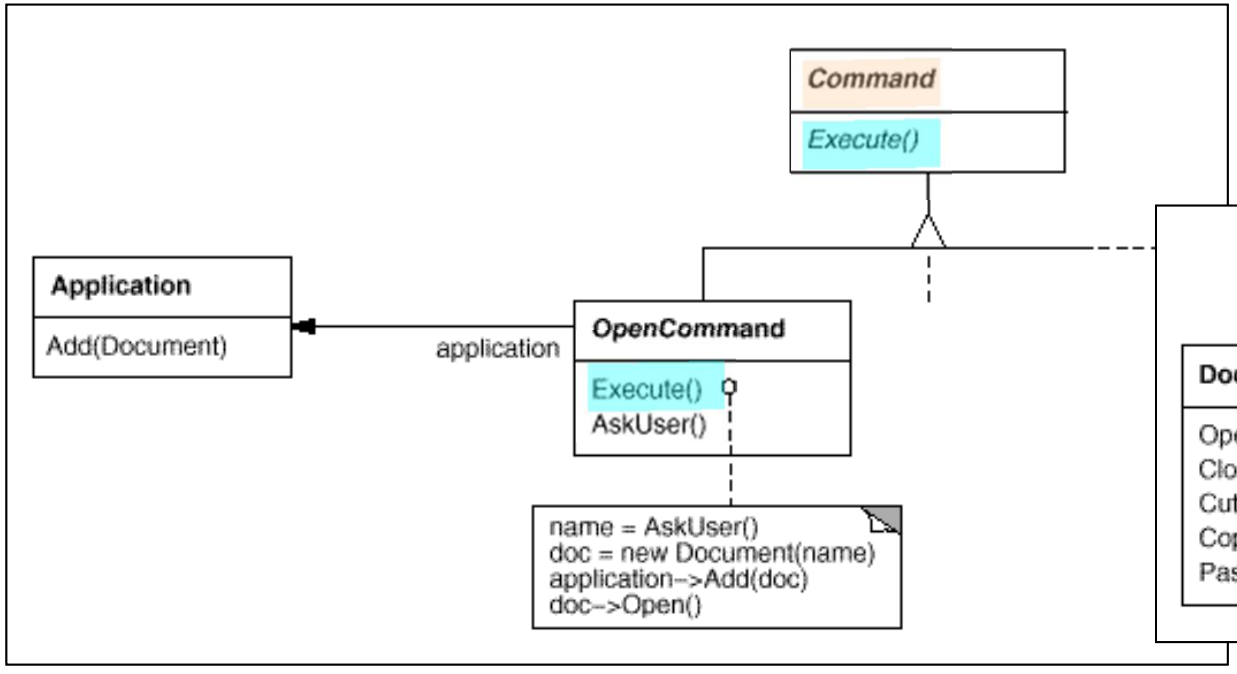
- 徹底理解 Object-Oriented 的意義。
  理解 this，理解 virtual。
- 學會看圖說故事。
- 把 OO、SOLID 等等觀念融入血液融入呼吸。

- 這個領域有很多抽象描述
- 學習路上的資源 (示例) 往往和工業級有極大距離。

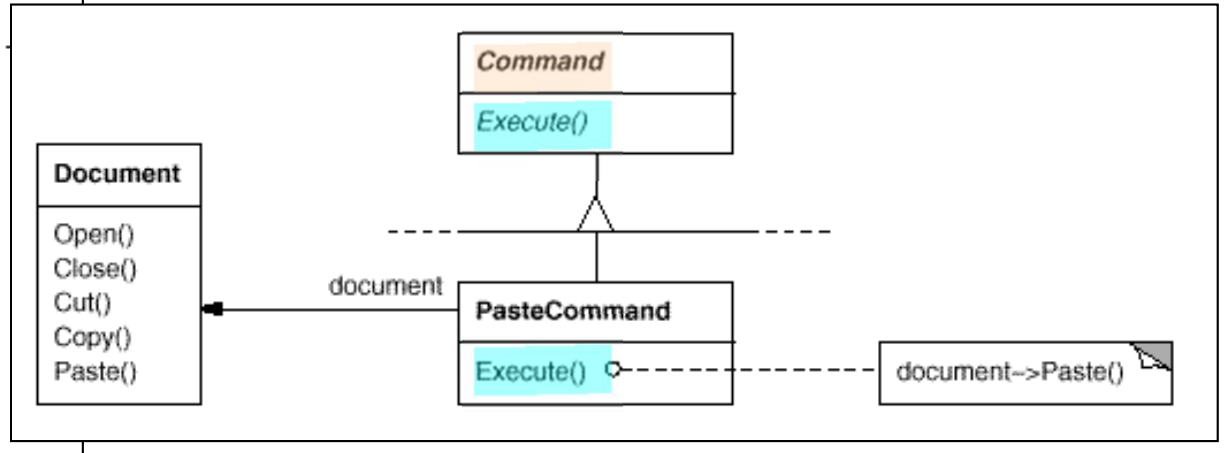
玩具示例 vs. 工業等級，例一 Command



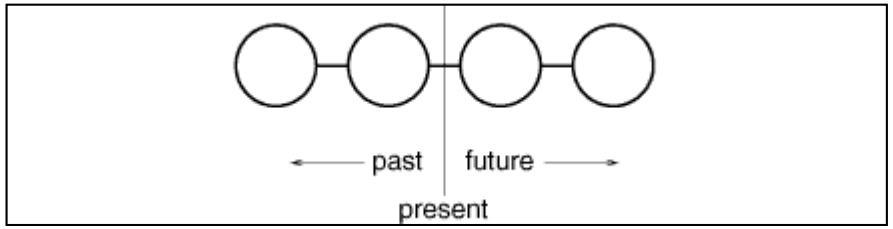


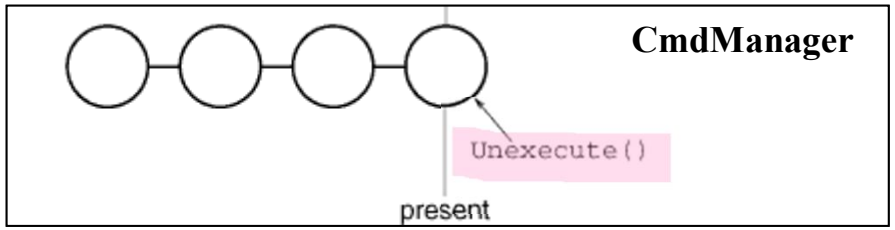


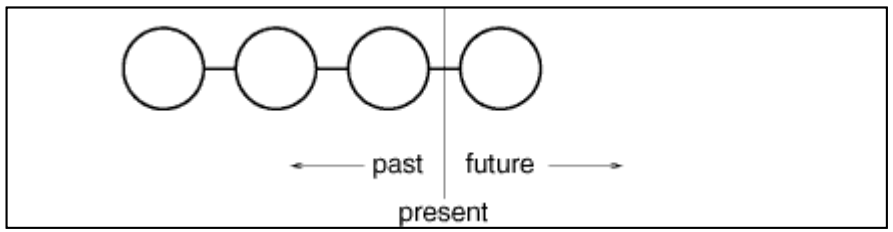
Command 大量運用在菜單 (選單; menu) 系統上。

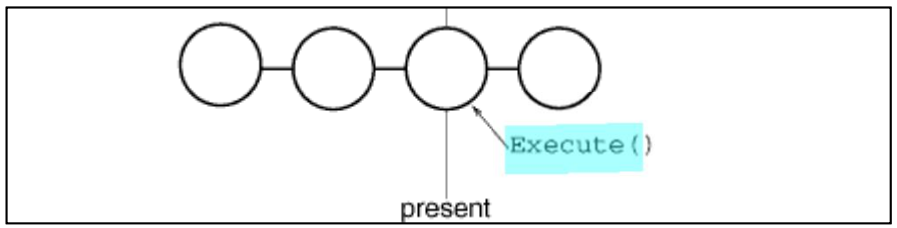


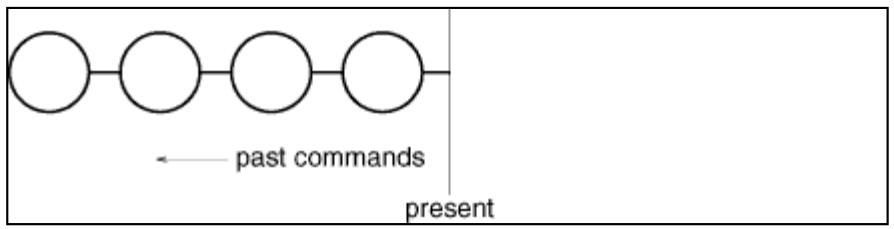
玩具示例 vs. 工業等級，例一 Command / Undo-Redo



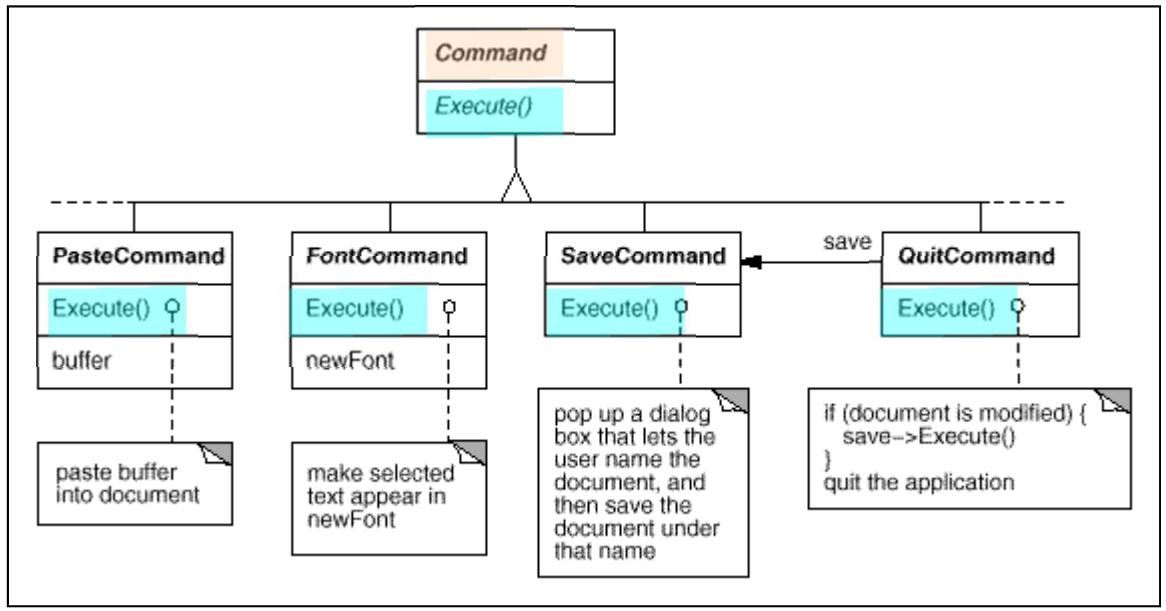




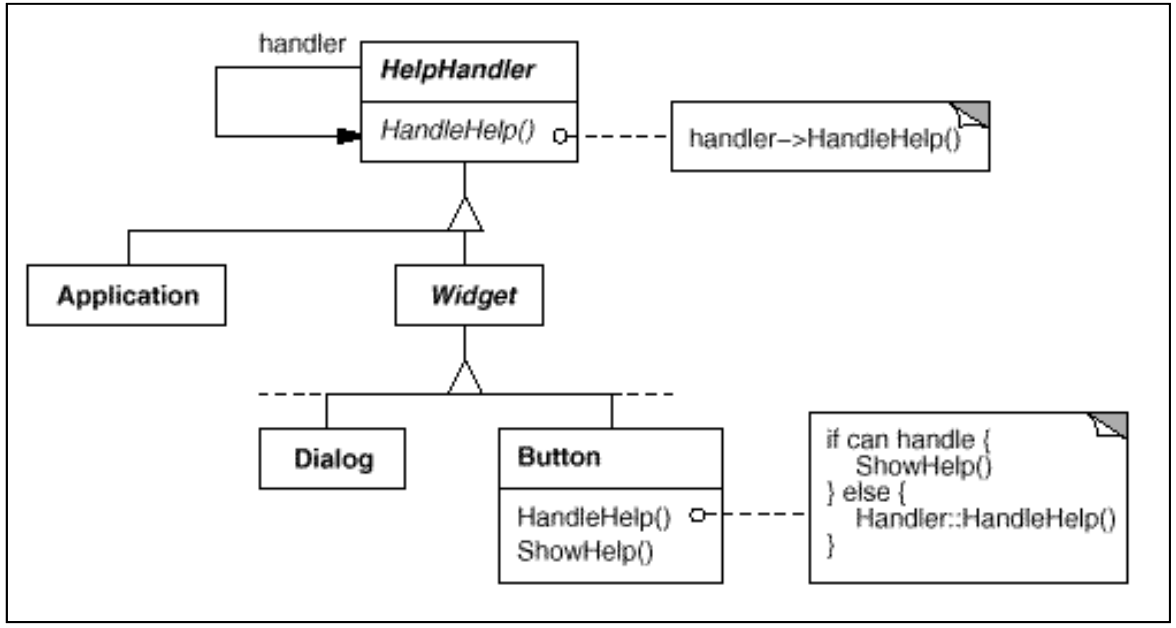




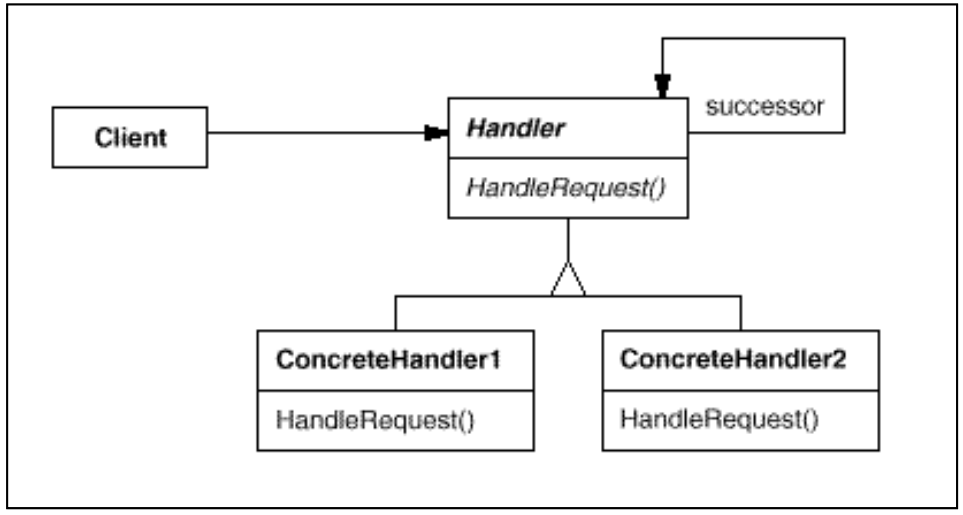
Unexecute() 就是最折磨人的地方



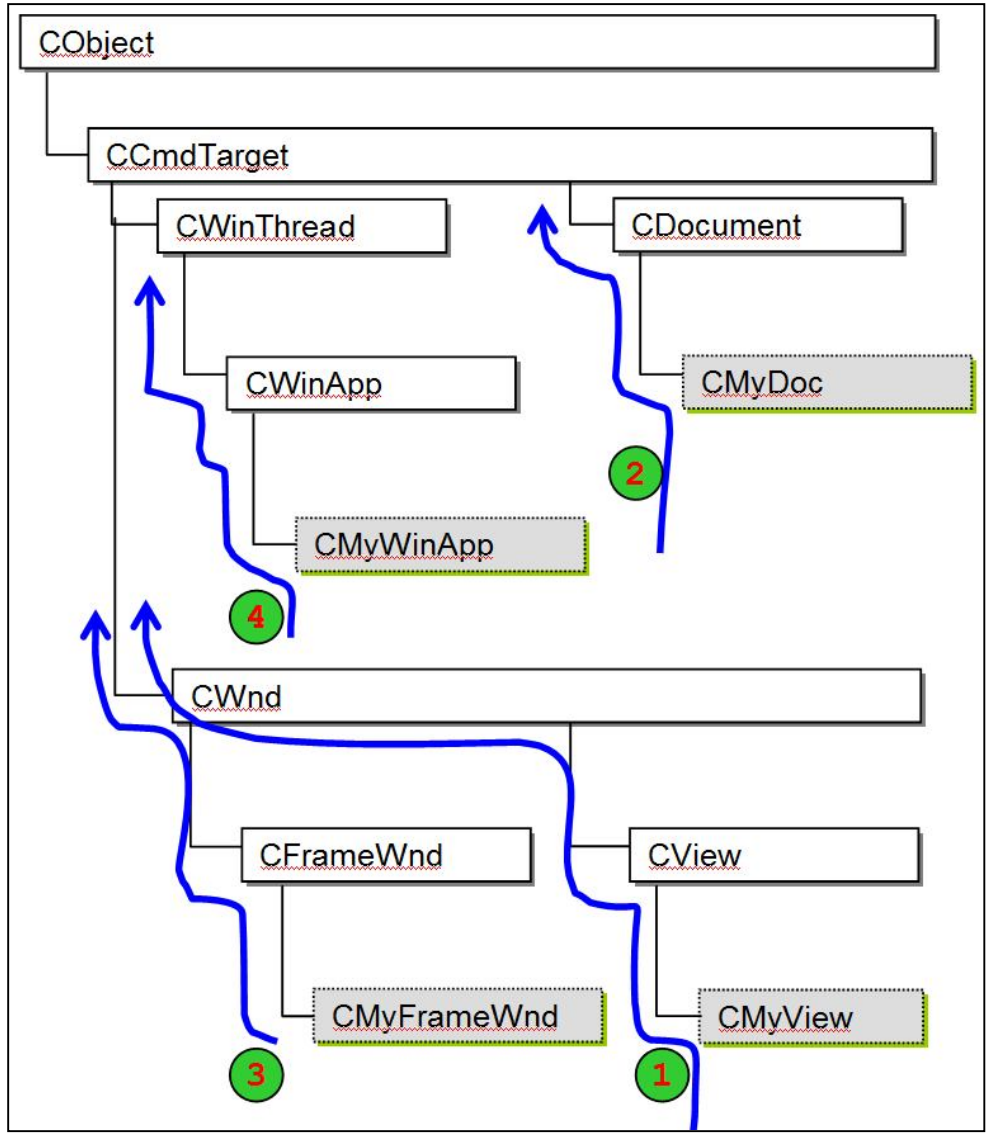
玩具示例 vs. 工業等級，例二 Chain of Responsibility



Chain of Responsibility (責任鏈) 是以訊息為基礎 (Message Based) 系統 (如 Windows) 賴以生存的技術。



玩具示例 vs. 工業等級，例二 Chain of Responsibility



命令訊息接收物的型態	處理次序
Frame 視窗	1. View 2. Frame 視窗本身 3. app object
View	1. View 本身 2. Document
Document	1. Document 本身 2. Document Template

動腦優先 or 動手優先?

None

學而不思則罔
思而不學則殆

— 孔夫子

The End