Design Pattern

Lecture 2

回顧

- 甚麼是設計模式? 為甚麼用它?
- 甚麼是抽象?
- 原則(Principles)
  - SRP (Single Responsibility Principle, 單一責任原則)
  - DIP (Dependency Inversion Principle, 依賴倒轉原則)
- 模式(Patterns)
  - 工廠模式 (Factory)
    - 簡單工廠 (Simple Factory)
    - 工廠方法 (Factory Method)
    - 抽象方法 (Abstract Factory)
  - 單例模式 (Singleton)
    - 靜態類 vs. 單例類
    - 多線程時的鎖 & 雙重檢測鎖
- 懶加載 (Lazy Loading)

抽象工廠

- 「使用者」不需要知道具體的工廠類。即與具體的工廠類**解耦**。

其它

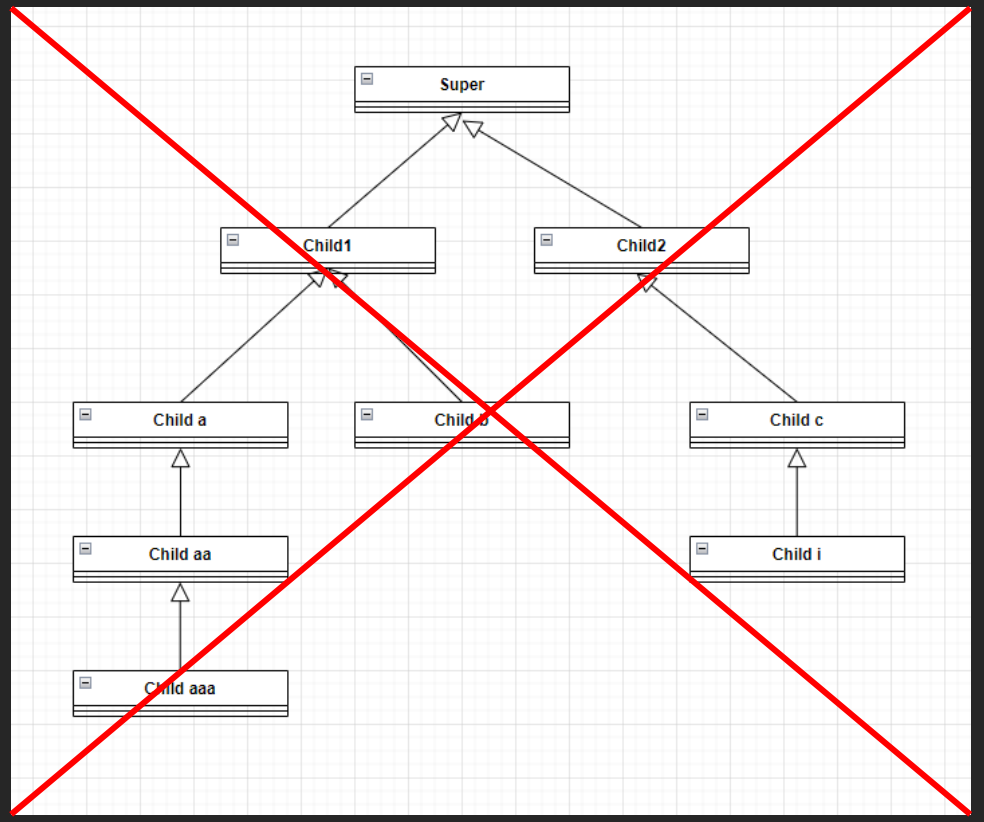

siki学院
首页
全部课程
独立游戏
Unity
虚幻
JavaEE
Python人工智能
微信小程序
神秘
移动端
Game Programming Patterns
Robert Nystrom
659 人加入学习
(24人评价)
游戏开发中的设计模式（Unity 5.6）
价格
¥ 280.00
学习有效期
永久有效
扫一扫
分享
收藏
去学习

目錄

原則

- CARP (Composition/Aggregate Reuse Principle, 合成/聚合複用原則)

- 少用繼承，多用組合。



策略模式 (Strategy)

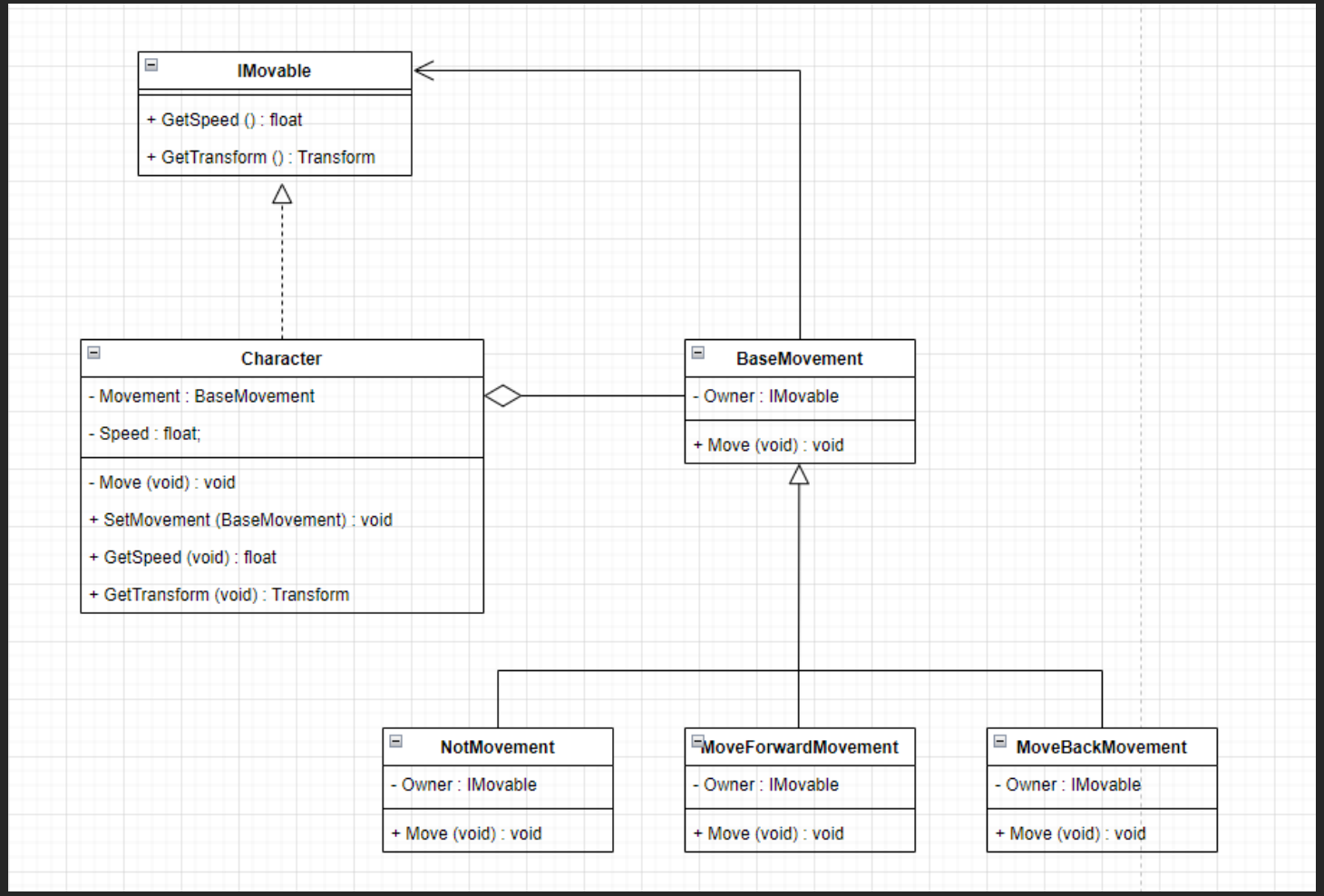
- 定義了算法族，分別封裝起來，使它們之間可以相互替換，此模式使算法的變化獨立於使用算法的用戶。

好處

- 當需要增加「算法」時，直接添加一個類
- 當需要修改「算法」時，不需要修改用戶

- 與命令模式(Command)非常像

策略模式 (Strategy)



少用繼承，多用組合。

- 「Character」中的移動行為，並不是透過繼承得來的，而是透過與「BaseMovement」組合而成的。

Q&A

裝飾模式 (Decorator)

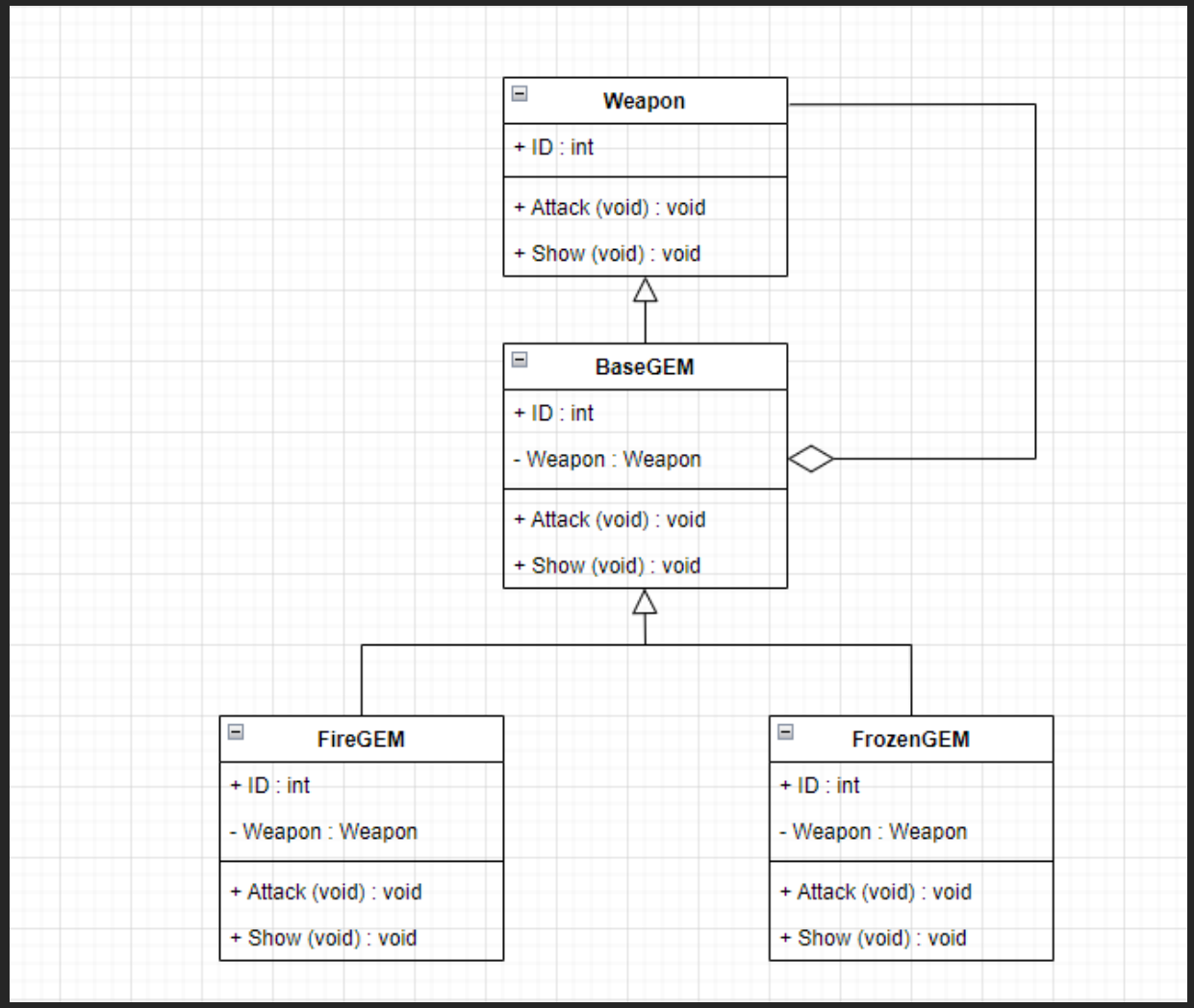
- **動態地將責任附加到對象上。若要擴展功能，裝飾者提供了比繼承更彈性的替代方案。**

- 一層一層地把對象包裝起來
- 執行時有點像遞歸那樣，一層一層地執行

- 注意:
  - 包裝順序可能會對結果產生影響
  - 會產生大量的”小類”
  - 不好刪除中間類 (需要額外保存上一個裝飾者)

- 武器 (被裝飾者)
- 寶石 (裝飾者)

裝飾模式 (Decorator)



合成/聚合複用原則 (CARP)

- 少用繼承，多用組合。

- 裝飾者模式與策略模式都是遵循了CARP
- 用組合取代了繼承

Q&A

觀察者模式 (Observer)

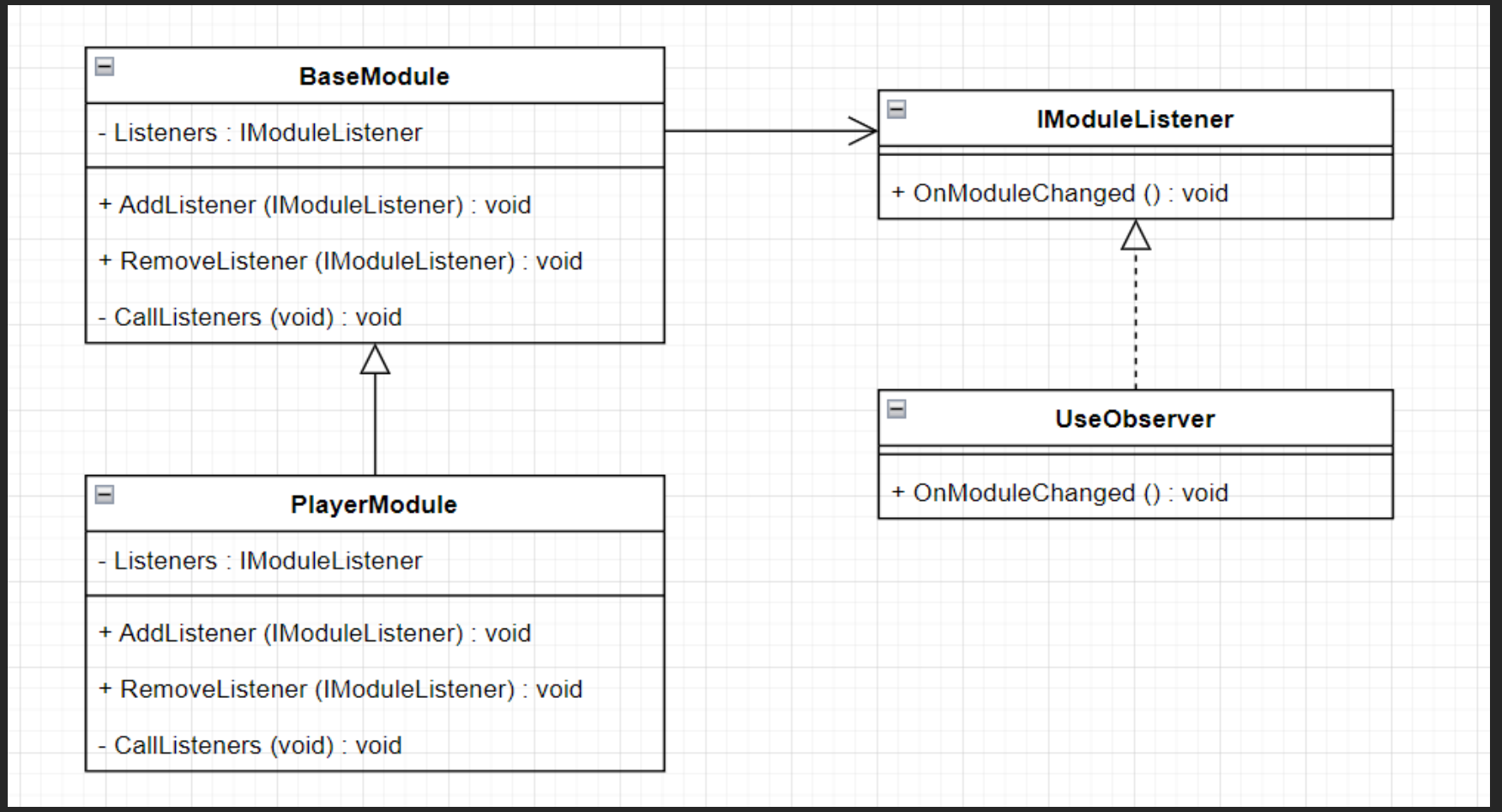
- **定義了對象之間的一對多依賴，這樣一來，當一個對象改變狀態時，它的所有依賴者都會收到通知並自動更新。**

- 相當於網站的”訂閱”，當有新消息時會自動發送到每一個訂閱者的郵箱
  - 網站: 被觀察者 (一個)
  - 訂閱者: 觀察者 (多個)

- 注意
  - 觀察者的順序不應該被依賴。即先後次序不應該影響到結果

- 數據的傳送方式
  - 推 (push) : 由被觀察者將數據傳送給觀察者 (一般，可以用struct來代替具體的參數)
  - 拉 (pull) ：由觀察者找被觀察者要

觀察者模式 (Observer)



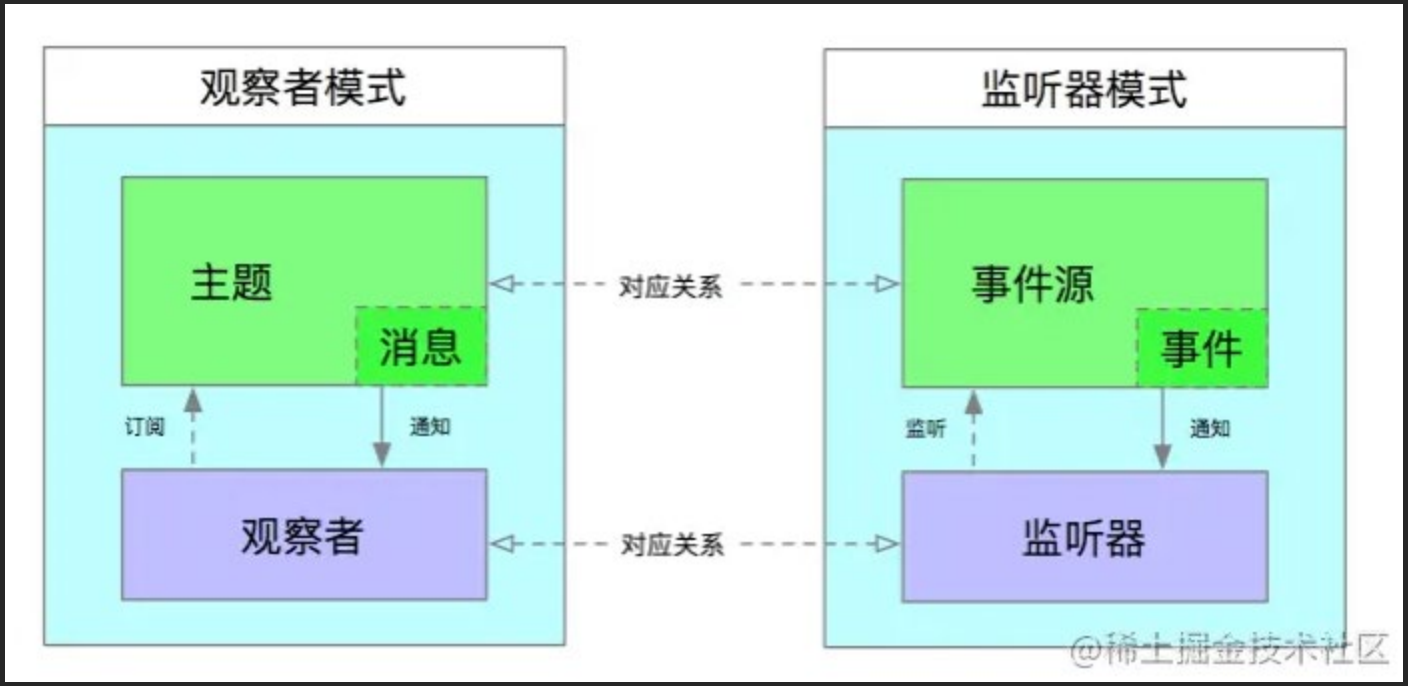
事件與委托(event & delegate)

- 是觀察者模式在特定場景下的一種改造和應用
- 增加了“事件”的概念

- 與Observer的比較
  - 觀察者 -> 事件監听器
  - 被觀察者 -> 事件管理器
  - 事件

- 例如UI中的按鈕，在點擊後會發生點擊事件，由事件管理器和事件監听器說。

觀察者模式 vs. 事件與委托



更多的是概念上的分別，在程序員之間溝通時不會產生誤會。

Q&A

結語

- 游戲中，需要與「策劃」和「美術」交互的部分，例如技能、動畫等的部分，可能比較難用得上設計模式。這些都是系統中處於「表層」位置的。
- 但是在不需要與他們交互的部分，特別是框架、底層，就會很常用到設計模式。例如Unity推出的ECS和很常用在UI上的MVC。
- 應用設計模式可以使項目規範化。

- 設計模式是非常靈活的，如何更好地運用，只能通過大量的項目來練習。