Design Pattern

Lecture 3

回顧

- 抽象工廠-「解耦」
- **合成/聚合複用原則 (CARP)**
  - 少用繼承，多用組合。
- 策略模式 (Strategy)
- 裝飾模式 (Decorator)
- 觀察者模式 (Observer)
  - 事件與委托 event & delegate
- 面向對象的編程「規範化」的重要性

設計原則

- 單一責任原則 (Single Responsibility Principle, SRP)
- 依賴倒轉原則 (Dependency Inversion Principle, DIP)
  - 好萊塢原則 (Hollywood Principle)
- 合成/聚合複用原則 (Composite/Aggregate Reuse Principle, CARP)
- 開放關閉原則 (Open-Closed Principle, OCP)
- 最少知識原則/迪米特原則 (Least Knowledge Principle, LKP/ Law of Demeter, LOD)
- 接口隔離原則 (Interface Segregation Principle, ISP)
- 里氏替換原則 (Liskov Substitution Principle, LSP)

設計模式

- 策略模式 (Strategy)
- 單例模式 (Singleton)
- 觀察者模式 (Observer)
- 工廠模式 (Factory)
  - 簡單工廠 (Simple Factory)
  - 抽象方法 (Abstract Method)
  - 抽象工廠 (Abstract Factory)
- 裝飾模式 (Decorator)
- 適配器模式
- 外觀模式
- 命令模式
- 代理模式
- 橋接模式
- 訪問者模式
- 模板模式
- 原型模式
- 備忘錄模式
- 中介者模式
- 迭代器模式
- 解釋器模式
- 蠅量/享元模式
- 組合模式
- 复合模式
- 責任/職責鏈模式
- 建造者/生成器模式

本回

- Singleton的多種實現

- 開放關閉原則 (Open-Closed Principle, OCP)
- 最少知識原則/迪米特原則 (Least Knowledge Principle, LKP/Law of Demeter, LOD)
- 好萊塢原則 (Hollywood principle)

- 模板模式 (Template)
- 外觀模式 (Facade)
- 適配器模式 (Adapter)
- 命令模式 (Command)
- 代理模式 (Proxy)
- 橋接模式 (Bridge)

內部靜態類

- 前提: 非靜態的類的變量及函數，只能通過”實例”才能被訪問。

- 內部類: 可以訪問外部類的任意變量與函數 (public/private/protected)。
  - 當內部類被標記成private時，意味著這一內部類並不提供給外部類以外的類使用。
- 外部類: 只能訪問內部類的public變量與函數。

- 內部類相當於一個「代碼塊」。在「代碼塊」中可以任意訪問塊外成員，而塊外只能訪問塊內對外開放的成員。

```
private void Method () {
  int i = 0;
    {
        private string name = “abc”;
         public int age = 10;
        i = 10;
    }
  age = 20;
}
```

Singleton的多種實現

- 類的初始化，有可能不是在調用的那一刻初始化的。所以如果在運行過程中不一定會用到的類被初始化時，那麼就會造成性能浪費。所以才需要懶加載。
- 但是懶加載在多線程中會有重複創建的問題，因此引入了雙重鎖的概念。
- 但是「上鎖」和「解鎖」的過程依然會對性能造成影響。因此引入了內部靜態類的實現。
- 此外，在.NET 4.0之後，C#加入了Lazy<T>來實現對象的延遲初始化。

- 在Unity中，因為MonoBehaviour是單線程的，所以不用太擔心多線程的問題。又因為MonoBehaviour在創建Scene的時候就會被加載，所以可以直接使用Lazy Loading或是在Awake中初始化。

- 開放關閉原則 (Open-Closed Principle, OCP)
- 最少知識原則/迪米特原則 (Least Knowledge Principle, LKP/Law of Demeter, LOD)
- 好萊塢原則 (Hollywood principle)

- 模板模式 (Template)
- 外觀模式 (Facade)
- 適配器模式 (Adapter)
- 命令模式 (Command)
- 代理模式 (Proxy)
- 橋接模式 (Bridge)

原則

- 依賴倒轉原則 (Dependency Inversion Principle, DIP)
  - 針對接口編程，不針對實現編程
  - 好萊塢原則 (Hollywood principle) : don’t call us, we’ll call you.
    - 模板方法模式 (Template)

- 開放關閉原則 (Open-Closed Principle, OCP)
  - 類應該對擴展開放，對修改關閉。
  - 裝飾者模式 (Decorator)

- 最少知識原則/迪米特原則 (Least Knowledge Principle, LKP/Law of Demeter, LOD)
  - 如果兩個類不必彼此直接通信，那麼這兩個類就不應當發生直接的相互作用。如果其中一個類需要調用另一個類的某一個方法的話，可以通過第三者轉發這個調用。
  - 外觀模式 (Facade)、中介者/調停者模式 (Mediator)

- 開放關閉原則 (Open-Closed Principle, OCP)
- 最少知識原則/迪米特原則 (Least Knowledge Principle, LKP/Law of Demeter, LOD)
- 好萊塢原則 (Hollywood principle)

- 模板模式 (Template)
- 外觀模式 (Facade)
- 適配器模式 (Adapter)
- 命令模式 (Command)
- 代理模式 (Proxy)
- 橋接模式 (Bridge)

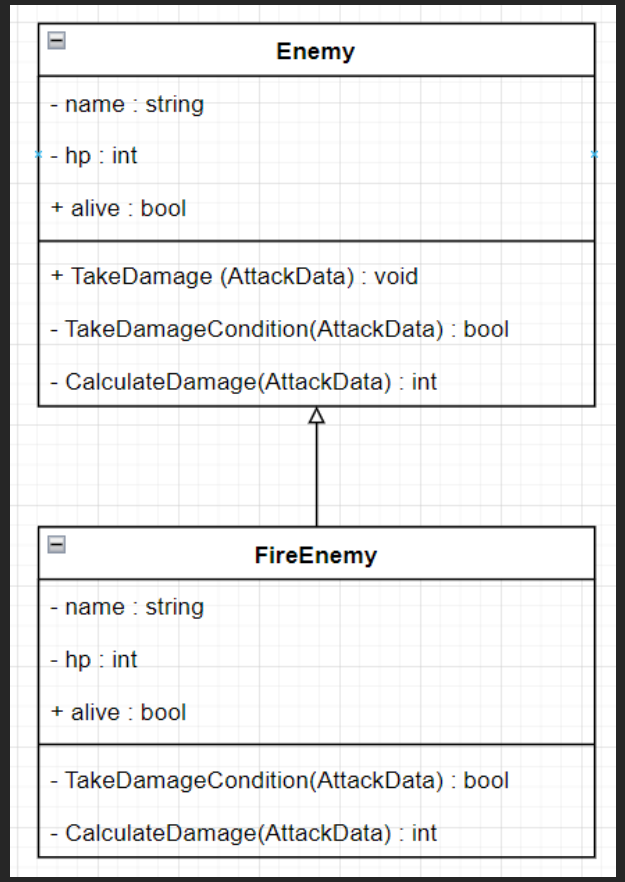
模板模式 (Template)

- 在一個方式中定義一個算法的骨架，而將一些步驟延遲到了子類中。模板方法使得子類可以在不改變算法結構的情況下，重新定義算法中的某些步驟。

- 在父類中，有一套算法執行的流程，而具體的算法實現，由子類實現。

- 好處
  - 減少代碼的重复性
  - 一種處理，有多種不同的版本

模板模式 (Template)



好萊塢原則 (Hollywood principle)

- 讓我們調用你們，而不是讓你們調用我們(don’t call us, we’ll call you）。

- 低層組件不要調用高層組件，由高層組件調用。
- 可以防止環形依賴。

- **Framework中不會調用「用戶的方法」，而是像一個黑箱那樣運作的。**

Q&A

- 開放關閉原則 (Open-Closed Principle, OCP)
- 最少知識原則/迪米特原則 (Least Knowledge Principle, LKP/Law of Demeter, LOD)
- 好萊塢原則 (Hollywood principle)

- 模板模式 (Template)
- 外觀模式 (Facade)
- 適配器模式 (Adapter)
- 命令模式 (Command)
- 代理模式 (Proxy)
- 橋接模式 (Bridge)

外觀模式 (Facade)

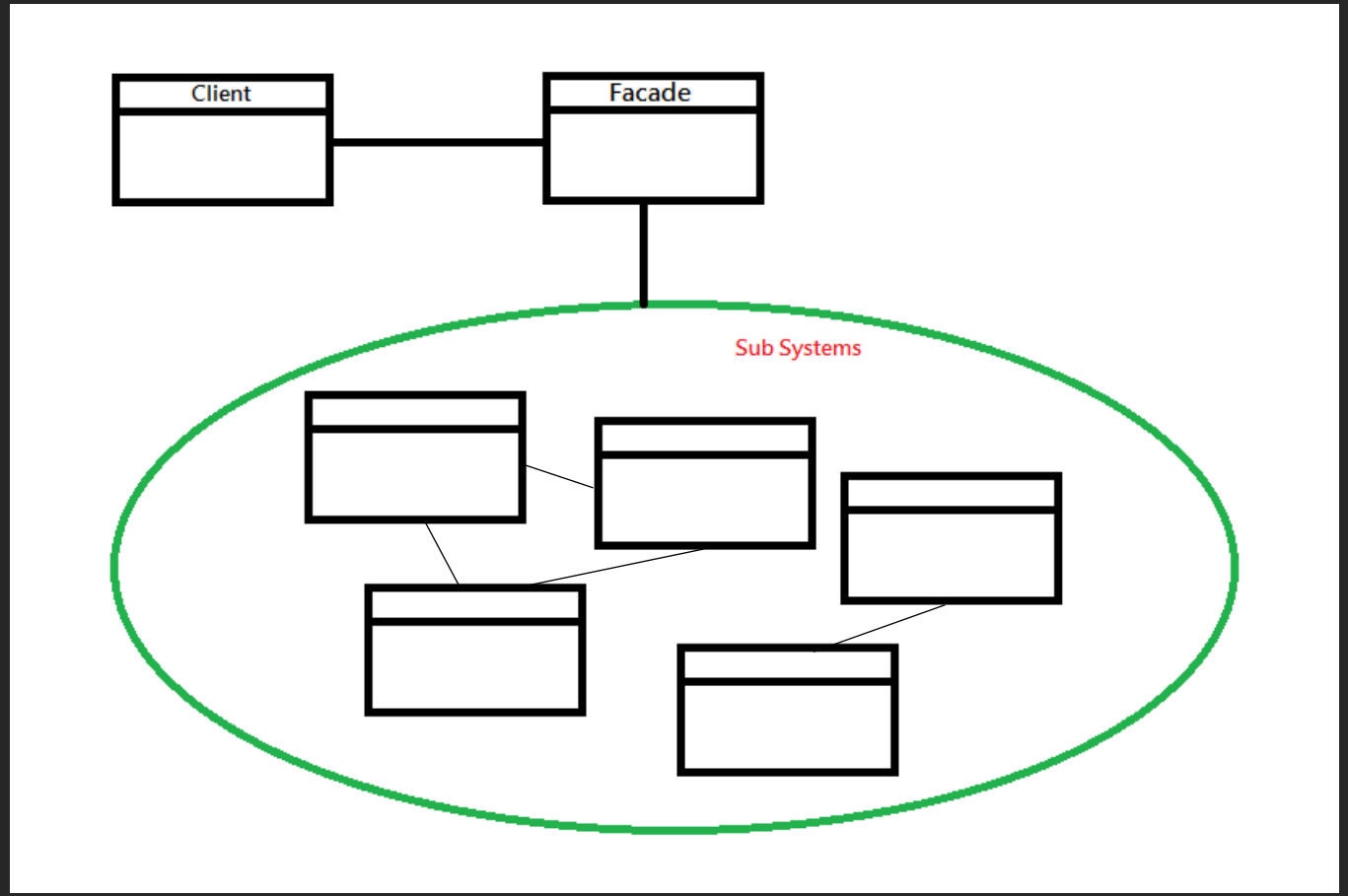
- **提供了一個統一的接口，用來訪問子系統中的一群接口。外觀定義了一個高層接口，讓子系統更容易使用。**

- 外觀模式**不封裝子系統**而是一個**接口集合**。

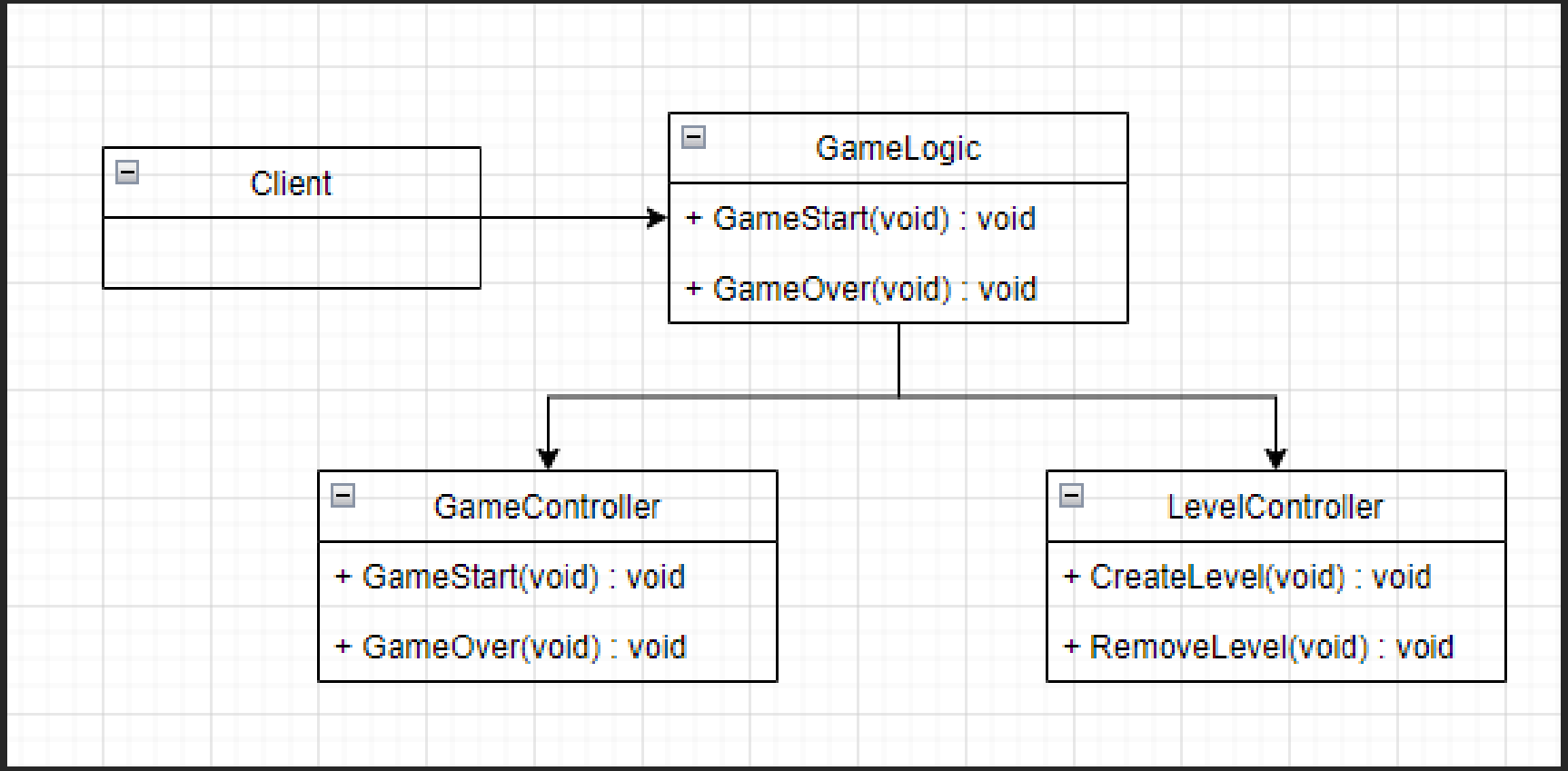
- 例子:
  - 在智能家具中，控制器可能會有「早上、中午、晚上」的幾個默認設定。通過按下「早上」來啟用各種不同的家具，例如咖啡機、燈、電腦等等。那麼這個「早上」就是一種外觀方式。
  - 它本身並不屬於「家具」的集合裏，而是一個第三者把「家具」(接口)統合在了一起。

- 能將複雜的**子系統溝通交給單一的一個類負責**，並提供單一界面給客戶端使用，減少系統間的耦合度。
- 由於將所有子類集中在外觀模式接口類中，最終會導致外觀模式接口類過於龐大且難以維護。

外觀模式 (Facade)



外觀模式 (Facade)


Client
GameLogic
+ GameStart(void) : void
+ GameOver(void) : void
GameController
+ GameStart(void) : void
+ GameOver(void) : void
LevelController
+ CreateLevel(void) : void
+ RemoveLevel(void) : void

最少知識原則/迪米特原則
Least Knowledge Principle, LKP/Law of Demeter, LOD

- 利用外觀類這個第三者，令「使用者」與「具體多個對象」**解耦**。

- 開放關閉原則 (Open-Closed Principle, OCP)
- 最少知識原則/迪米特原則 (Least Knowledge Principle, LKP/Law of Demeter, LOD)
- 好萊塢原則 (Hollywood principle)

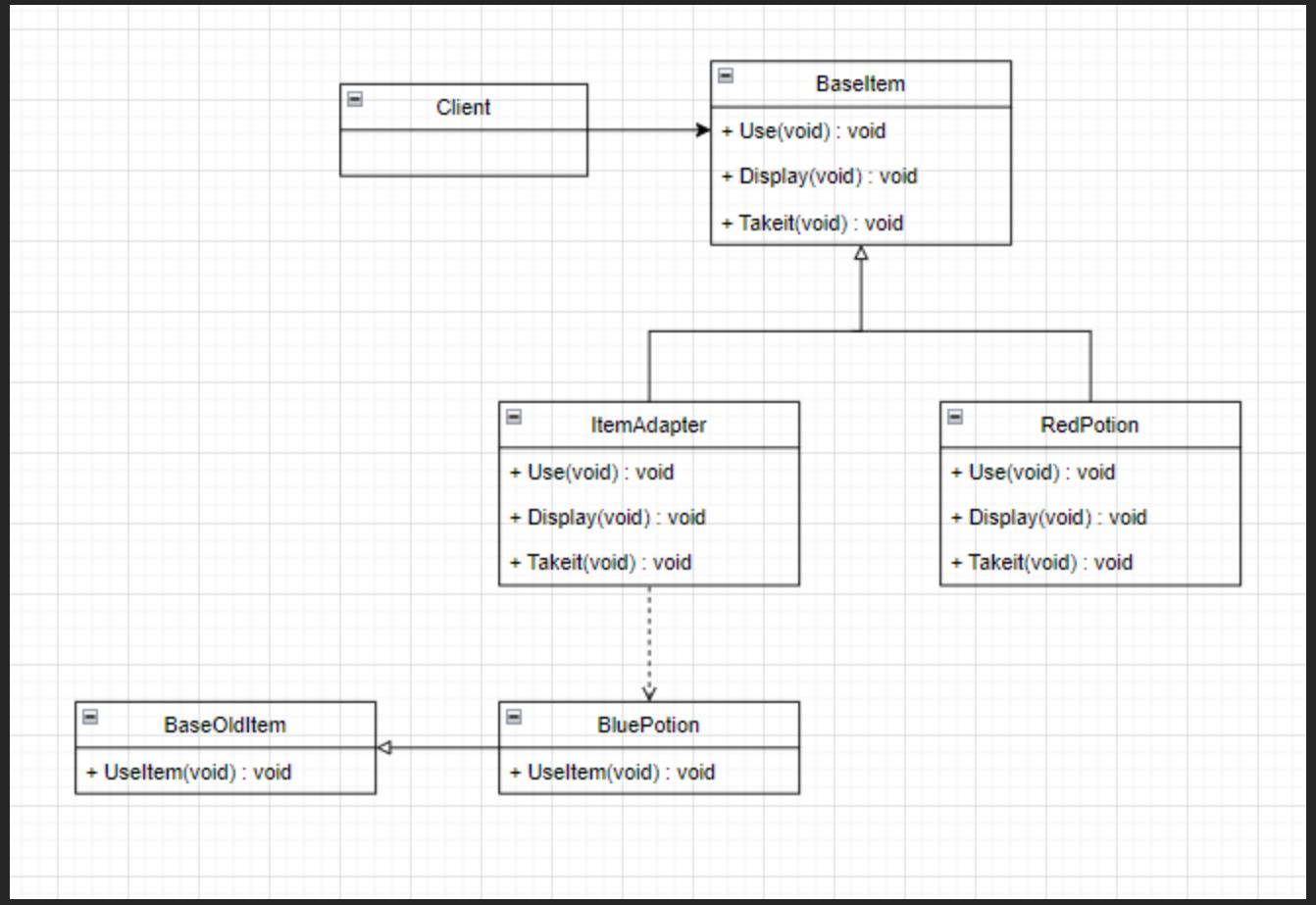
- 模板模式 (Template)
- 外觀模式 (Facade)
- 適配器模式 (Adapter)
- 命令模式 (Command)
- 代理模式 (Proxy)
- 橋接模式 (Bridge)

適配器模式 (Adapter)

- 將一個類的接口，轉換成客戶的期望的另一個接口。適配器讓原本接口不兼容的類可以合作無間。

- 作為中間件被使用。

適配器模式 (Adapter)



Q&A

- 開放關閉原則 (Open-Closed Principle, OCP)
- 最少知識原則/迪米特原則 (Least Knowledge Principle, LKP/Law of Demeter, LOD)
- 好萊塢原則 (Hollywood principle)

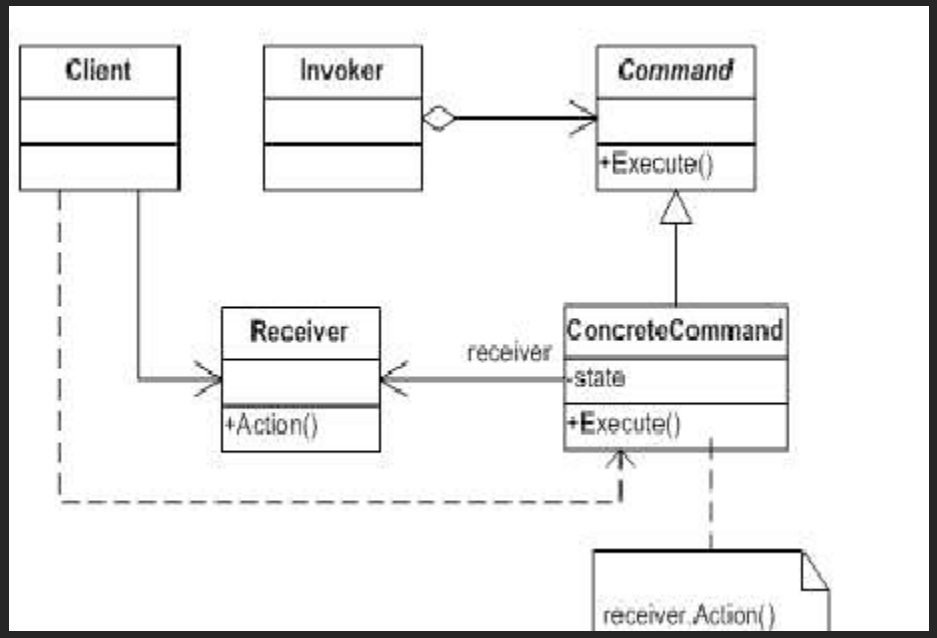
- 模板模式 (Template)
- 外觀模式 (Facade)
- 適配器模式 (Adapter)
- 命令模式 (Command)
- 代理模式 (Proxy)
- 橋接模式 (Bridge)

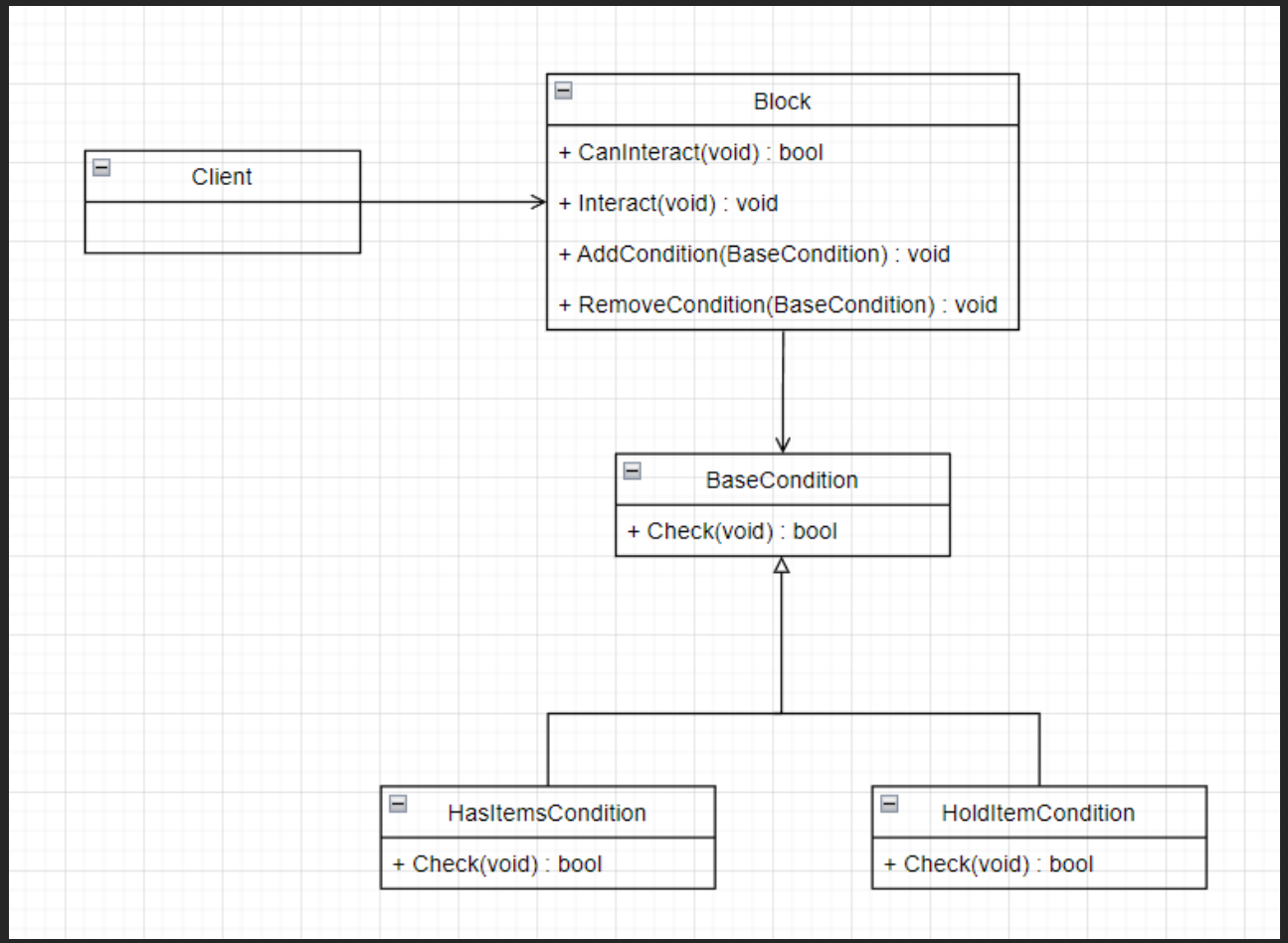
命令模式 (Command)

- 將”請求”封裝成對象，以便使用不同的請求，隊列或者日志來參數化變化對象。命令模式支持可撤銷的操作。

- 與策略模式很相似。但是策略模式封裝的是「行為」。而命令模式封裝的是「請求(命令)」。命令模式是將「請求(命令)」封裝成一個新的類族。

命令模式 (Command)





Q&A

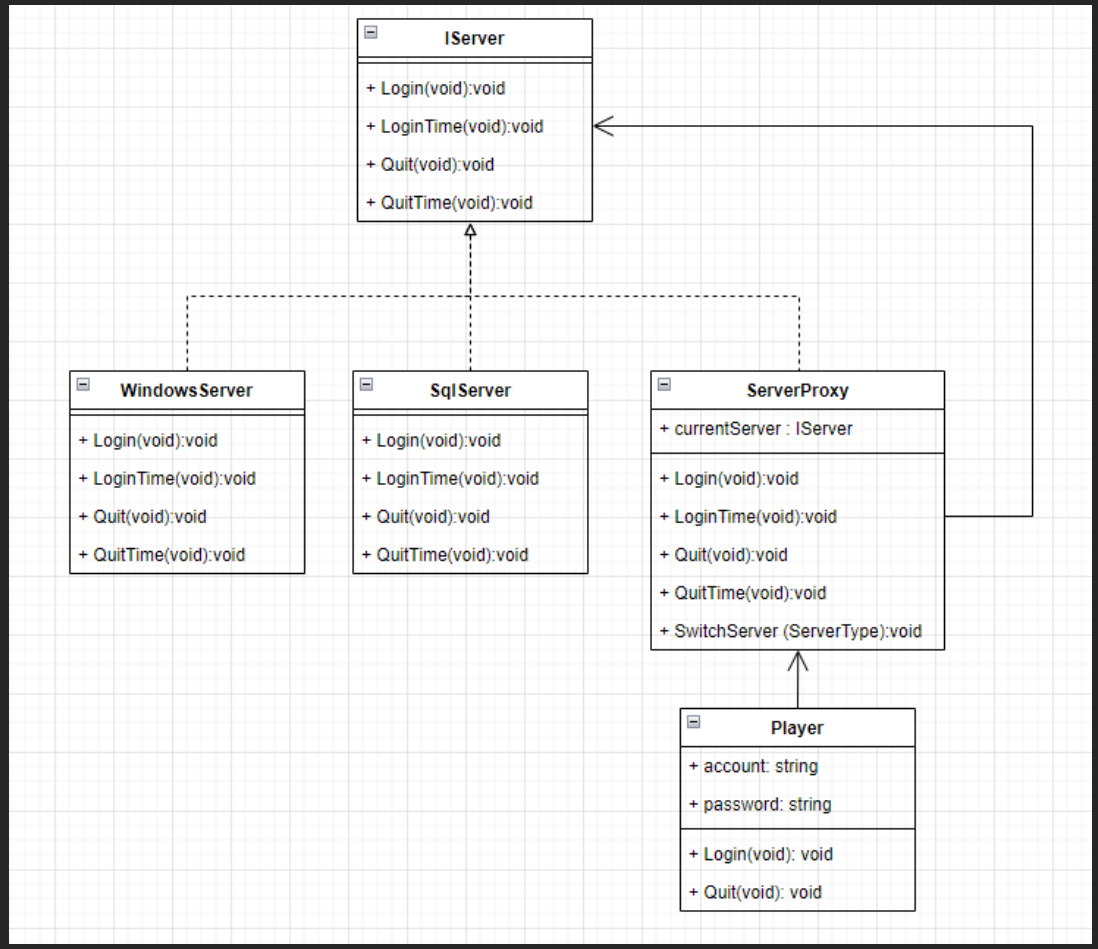
- 開放關閉原則 (Open-Closed Principle, OCP)
- 最少知識原則/迪米特原則 (Least Knowledge Principle, LKP/Law of Demeter, LOD)
- 好萊塢原則 (Hollywood principle)

- 模板模式 (Template)
- 外觀模式 (Facade)
- 適配器模式 (Adapter)
- 命令模式 (Command)
- 代理模式 (Proxy)
- 橋接模式 (Bridge)

代理模式 (Proxy)

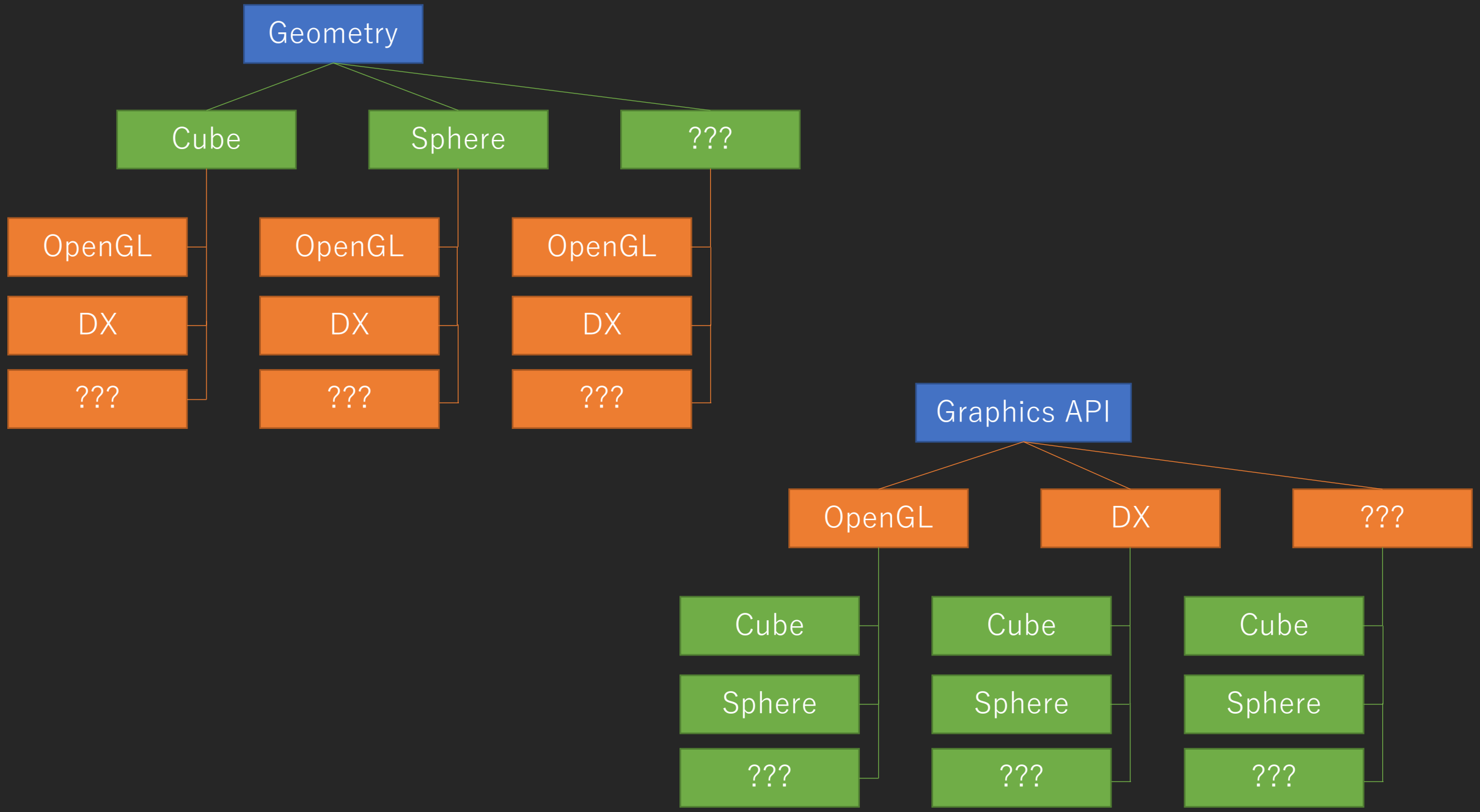
- 為另一個對象提供一個替身或占位符以控制對這個對象的認知。
- “ServerProxy”引入了一種間接性。令「使用者」與「實例」解耦。

代理模式 (Proxy)


IServer
+ Login(void):void
+ LoginTime(void):void
+ Quit(void):void
+ QuitTime(void):void
WindowsServer
+ Login(void):void
+ LoginTime(void):void
+ Quit(void):void
+ QuitTime(void):void
SqlServer
+ Login(void):void
+ LoginTime(void):void
+ Quit(void):void
+ QuitTime(void):void
ServerProxy
+ currentServer : IServer
+ Login(void):void
+ LoginTime(void):void
+ Quit(void):void
+ QuitTime(void):void
+ SwitchServer (ServerType):void
Player
+ account: string
+ password: string
+ Login(void): void
+ Quit(void): void

- 開放關閉原則 (Open-Closed Principle, OCP)
- 最少知識原則/迪米特原則 (Least Knowledge Principle, LKP/Law of Demeter, LOD)
- 好萊塢原則 (Hollywood principle)

- 模板模式 (Template)
- 外觀模式 (Facade)
- 適配器模式 (Adapter)
- 命令模式 (Command)
- 代理模式 (Proxy)
- 橋接模式 (Bridge)


Geometry
Cube
Sphere
???
OpenGL
DX
???
OpenGL
DX
???
OpenGL
DX
???
Graphics API
OpenGL
DX
???
Cube
Sphere
???
Cube
Sphere
???
Cube
Sphere
???

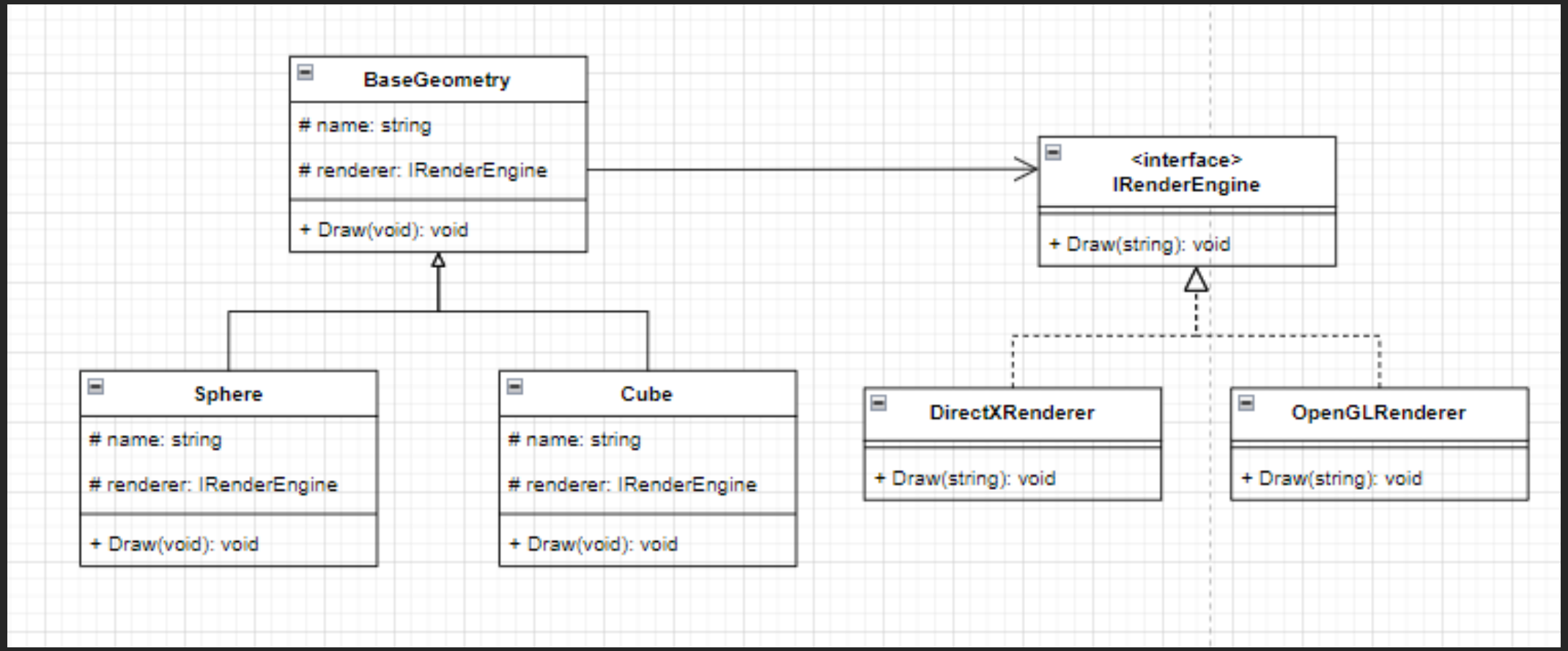
繼承的問題

擴展性問題: 如果新增圖形類，就要新增各圖形API版本。同理，新增圖形API也是如此。

橋接模式 (Bridge)

- 將抽象部分與它的實現部分分離，使它們都可以獨立地變化。
- 少用繼承，多用組合

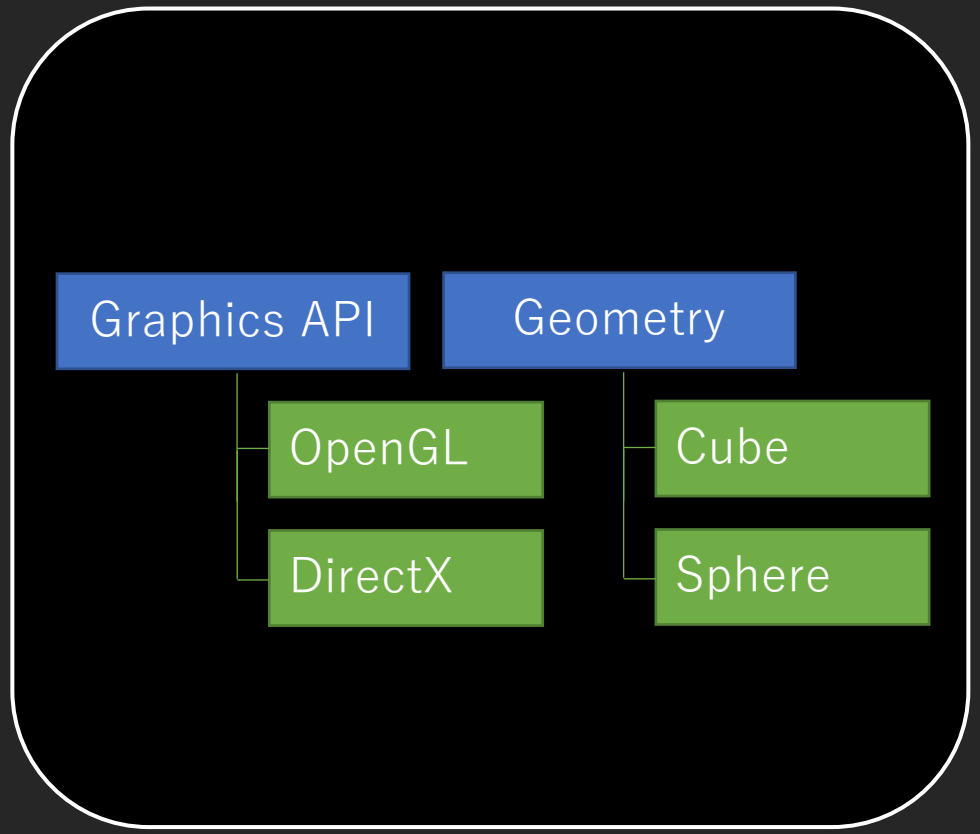
橋接模式 (Bridge)

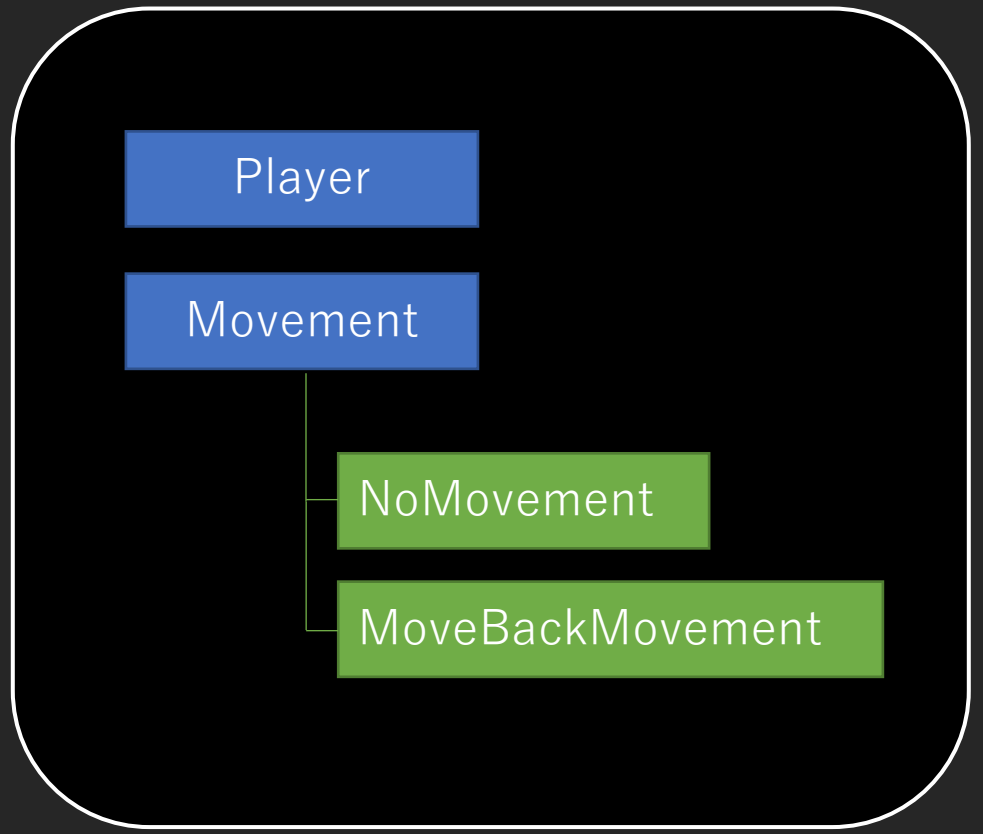

BaseGeometry
name: string
renderer: IRenderEngine
+ Draw(void): void
<interface>
IRenderEngine
+ Draw(string): void
Sphere
name: string
renderer: IRenderEngine
+ Draw(void): void
Cube
name: string
renderer: IRenderEngine
+ Draw(void): void
DirectXRenderer
+ Draw(string): void
OpenGLRenderer
+ Draw(string): void

橋接模式 vs 策略模式

- 兩者都是以「聚合/組合」的形式，將兩者相結合，減少繼承。

- 策略模式: 考慮的是「算法家族」。
- 橋接模式: 簡單地來說是每一層都使用了策略模式。


Graphics API
Geometry
OpenGL
DirectX
Cube
Sphere


Player
Movement
NoMovement
MoveBackMovement

來敲代碼吧!

Q&A

總結

- 三個原則
  - 開放關閉原則 (Open-Closed Principle, OCP)
  - 最少知識原則/迪米特原則 (Least Knowledge Principle, LKP/Law of Demeter, LOD)
  - 好萊塢原則 (Hollywood principle)

- 六個模式
  - 模板模式 (Template)
  - 外觀模式 (Facade)
  - 適配器模式 (Adapter)
  - 命令模式 (Command)
  - 代理模式 (Proxy)
  - 橋接模式 (Bridge)

下回

- 策略模式 (Strategy)
- 單例模式 (Singleton)
- 觀察者模式 (Observer)
- 工廠模式 (Factory)
  - 簡單工廠 (Simple Factory)
  - 抽象方法 (Abstract Method)
  - 抽象工廠 (Abstract Factory)
- 裝飾模式 (Decorator)
- 適配器模式 (Adapter)
- 外觀模式 (Facade)
- 命令模式 (Command)
- 代理模式 (Proxy)
- 橋接模式 (Bridge)
- 模板模式 (Template)
- 迭代器模式
- 組合模式
- 中介者模式
- 原型模式
- 訪問者模式
- 解釋器模式
- 備忘錄模式
- 蠅量/享元模式
- 複合模式
- 責任/職責鏈模式
- 建造者/生成器模式
- 單一責任原則
- 依賴倒轉原則
  - 好萊塢原則
- 合成/聚合複用原則
- 開放關閉原則
- 最少知識原則/迪米特原則
- 接口隔離原則
- 里氏替換原則

課程安排

- Lecture 04
  - 迭代器模式
  - 組合模式
  - 中介者模式
  - 原型模式
  - 訪問者模式
  - 接口隔離原則
  - 里氏替換原則
- Lecture 05
  - 解釋器模式
  - 備忘錄模式
  - 蠅量/享元模式
  - 複合模式
  - 責任/職責鏈模式
  - 建造者/生成器模式
- Lecture 06
  - 總結
  - 複習
- Lecture 07
  - PureMVC

結語

- 設計模式是將「抽象」、「模塊化」、「規範性」變成可能的一種方法。
- OOP編程是工程師們協作的一種編程思維。應該在編程時多思考如何將代碼變得
  - 易讀
  - 易擴展
  - 健壯性
- **原則**或是**模式**的定義雖然很短，但都匯聚了前人的經驗，要完全理解並應用並不是一件簡單的事。

Q&A