Head First Design Pattern Note

OO 設計守則

- OO基本三大原則：可擴充、可維護、再利用。
- 找出程式中可能需要更動之處，把它們獨立出來，不需要和那些不需要更動的程式碼碼混在一起。
- 寫程式是針對介面而寫，而不是針對實踐方式而寫。
- 多用合成，少用繼承。
- 設計時，盡量讓需要互動的物件之間關係鬆綁。
- 類別應該開放，以便擴充；應該關閉，禁止修改。
- Dependency Inversion Principle 顛覆依賴守則。依賴抽象類別，不要依賴具象類別。
- 極少化守則：只和你的密友談話。
- 好萊屋守則：別呼叫我們，我們會呼叫你。
- 單一責任守則：一個類別應該只具有一個改變的理由。盡量使類別保持單一責任。

OO 設計模式

- **策略模式**

定義了演算法家族，個別封裝起來，讓它們之間可以互相替換，此模式讓演算法的變動，不會影響到使用演算法的程式。

- **觀察者模式**

定義了物件之間的一對多關係，如此一來，當一個物件改變狀態，其他相依者都會收到通知並自動被更新。

- **裝飾者模式**

動態地將責任加諸於物件上。若要擴充功能，裝飾者提供了比繼承更有彈性的選擇。

- **工廠模式**

定義了一個建立物件的介面，但由次類別決定要實體化的類別為何。工廠方法讓類別把實體化的動作，交由次類別決定。

- **抽象工廠模式**

提供了一個介面，建立相關或相依物件之家族，而不需要明確指定具象類別。

- **獨體模式**

確保一個類別只有一個實體，並給它一個存取的全域點。

- **命令模式**

將請求封裝成物件，以便使用不同的請求、佇列、或是日誌，參數化其他物件。命令模式也支援可復原的作業。

- **轉接器模式**

將一個類別的介面，轉換成另一個介面以供客戶使用。轉接器讓原本介面不相容的類別可以合作無間。

- **表象模式**

提供了一個統一的介面，用來存取次系統中的一群介面。表象定義了一個較高層次的介面，讓次系統共容易使用。

- **樣板模式**

將一個演算法的骨架定義在一個方法中，而演算法本身會用到的一些方法，則是定義在次類別中。樣板方法讓次類別在不改變演算法架構的情況下，重新定義演算法中的某些步驟（hook）。

- **反覆器模式**

讓我們能夠取得一個聚合內的每個元素，而不需要此聚合將其實踐方式暴露出來。

- **合成模式**

允許你將物件合成樹狀的結構，用來呈現「部分/整體」的階層關係。利用合成，客戶程式碼能夠一視同仁的對待個別物件以及物件合成後的結果。

- **狀態模式**

允許物件隨著內在的狀態改變而改變行為，就好像物件的類別改變了一樣。

- **代理人模式**

讓某個物件具有一個替身，藉以控制外界對此物件的接觸。

Example Code : https://github.com/xul4m4d93/DesignPattenLearn