《大話設計模式》筆記

一、概念

1.物件導向的程式設計，不是類別越多越好，類別的劃分是為了「封裝」，但分類的基礎是「抽象」，具有相同屬性和功能之物件的抽象集合才是「類別」。

2.以「抽象」來隔離有可能發生的同類變化。

3.在初始化資訊不會發生變化的情況下，複製（**Clone**）是最好的辦法。

4.「委託」是一種參考方法的類型。一旦為委託分派了方法，委託將與該方法具有完全相同的行為。一個委託可以搭載多個方法。委託可以使得委託物件所搭載的方法並不需要屬於同一個類別。

5.委託物件所搭載的所有方法，必須擁有相同的參數列表和返回值類型。

6.不要為程式碼加上基於猜測的、實際不需要的功能。

二、設計原則

1.單一職責原則：就一個「類別」而言，應只會有一個引起它變化的原因。

2.開放-封閉原則：對於「擴展」是開放的，對於「更改」是封閉的。

3.依賴倒轉原則：

- 高階模組不應依賴低階模組。兩個都應依賴「抽象」。
- 細節依賴抽象，而非抽象依賴細節（亦即針對介面設計，而非針對要實現的程式）。

4.**Liskov** 替換原則：子類型必須能夠替換掉它們的父類型。

5.迪米特原則（最少知識原則）：若兩個類別不必彼此直接通信，則這兩個類別就不應當發生直接的相互作用。若其中一個類別需要調用另一個類別的某一個方法，則應透過第三者轉發這個調用。

//此原則的前提是在類別的結構設計上，每一個類別都應當降低成員的使用許可權。

6.合成/聚合複用原則：盡量使用合成/聚合，少用類別繼承。

- 聚合表示的是 A 物件可以包含 B 物件，但 **B** 物件不是 **A** 物件的一部份。
- 合成表示的是 A 物件可以包含 B 物件，且 **B** 物件是 **A** 物件的一部份。

二、設計模式簡介

1.簡單工廠模式：針對很容易變化的地方，以一個單獨的「類別」來處理它。

//想增加功能時，更改工廠類別。

2.工廠模式：定義一個用於建立物件的介面，讓子類別決定實體化哪一個類別。

//想增加功能時，修改用戶端程式碼。

3.抽象工廠模式：提供一個建立一系列相關或互相依賴物件的介面，而無須指定它們具體的類別。

4.策略模式：封裝一系列的演算法，讓它們之間可以互相替換。

5.裝飾模式：可動態地給一個物件加入功能。

/* 若只有一個 **ConcreteComponent** 類別而沒有抽象的 **Component** 類別，則 **Decorator** 類別可以是 **ConcreteComponent** 的一個子類別。

同理，若只有一個 **ConcreteDecorator** 類別，則不必建立一個單獨的 **Decorator** 類別，而可以把 **Decorator** 和 **ConcreteDecorator** 的責任合併成一個類別。*/

6.代理模式：為其他物件提供一種代理，以控制對這個物件的存取。

/* 用於遠端代理（隱藏一個物件存在於不同位址空間的事實）、虛擬代理（存放實體化需要長時間的真實物件）、安全代理（控制真實物件存取時的許可權）、智慧參考（當調用真實物件時，代理處理另外一些事）。*/

7.原型模式：用原型實例指定建立物件的種類，並且透過拷貝（**Clone**）這些原型來建立新的物件。

8.範本方法模式：定義一個操作中的演算法骨架，將一些步驟延遲到子類別中。

//用於當你要完成在某一細節層次一致的過程或一系列的步驟，但其個別步驟在更詳細的層次上的實現可能不同時。

9.外觀模式：為子系統的一組介面提供一個一致的介面。

10.建造者模式：將一個複雜物件的構建與它的表示分離，使得同樣的構建過程可以建立不同的表示。

//若需要改變一個產品的內部表示，只需要再定義一個具體的建造者即可。

11.觀察者模式：定義一對多的依賴關係，讓多個觀察者物件同時監聽某一個主題物件。這個主題物件在狀態發生變化時，會通知所有觀察者物件，使它們能夠自動更新自己。

//用於當一個物件的改變需要同時改變其他物件，且它不知道到底有多少物件有待改變時。

12.狀態模式：當一個物件的內在狀態改變時允許改變其行為。

//用於當一個物件的行為取決於它的狀態，且必須在執行時根據狀態改變它的行為。

13.轉接器模式：將一個類別的介面轉換成客戶希望的另一個介面。

//用於應用希望複用一些既有的類別，但介面又與複用環境要求不一致的狀況，且雙方都不容易修改。

14.備忘錄模式：在不破壞封裝性的前提下，捕獲一個物件的內部狀態，並在該物件之外保存這個狀態。

//用於功能較複雜的，且需要維護或記錄屬性歷史的類別，或者需要保存的屬性只是眾多屬性中的一小部份時。

15.組合模式：將物件組合成樹狀結構以表示「部分-整體」的層次結構。能使得用戶對單個物件和組合物件的使用俱有一致性。

//用於當你發現需求中是表現部分與整體層次的結構時，以及你希望用戶可以忽略組合物件與單個物件的不同，統一地使用組和結構中的所有物件時。

16.迭代器模式：提供一種方法依序存取一個聚合物件中各個元素，而又不暴露該物件的內部表示。

//用於當你需要存取一個聚集物件，且不管這些物件是什麼都需要走遍時，或是你需要對聚集有多種方式走遍時。

17.獨體模式：保證一個類別僅有一個實體，並提供一個存取它的全域訪問點。

18.橋接模式：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。

//實現指的是抽象類別和它的衍生類別用來實現自己的物件。

19.命令模式：將一個請求封裝為一個物件，讓你可用不同的請求對客戶進行參數化。

20.職責鏈模式：使多個物件都有機會處理請求，從而避免請求的發送者和接收者之間的耦合關係。將這個物件連成一條鏈，並沿著這條鏈傳遞該請求，直到有一個物件處理它為止。

21.仲介者模式：用一個仲介物件來封裝一系列的物件互動。

//用於一組已定義良好但複雜的方式進行通訊的場合，或想訂製一個分佈在多個類別中的行為，而又不想產生太多子類別時。

22.享元模式：運用共用技術有效地支援大量細粒度的物件。

//用於物件會產生大量的記憶體消耗時。

23.解譯器模式：給定一個語言，定義它的文法的一種表示，並定義一個解譯器，這個解譯器使用該表示來解譯語句。

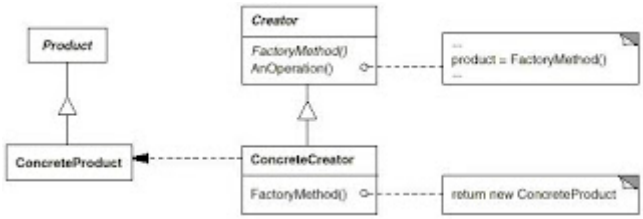
//用於當一個語言需要解譯執行，且你可將該語言中的句子表示為一個抽象語法樹時。

24.訪問者模式：表示一個作用於某物件結構中的各元素之操作。

//可使你在不改變各元素之類別的前提下，定義作用於這些元素的新操作。

三、設計模式結構

1.工廠模式：

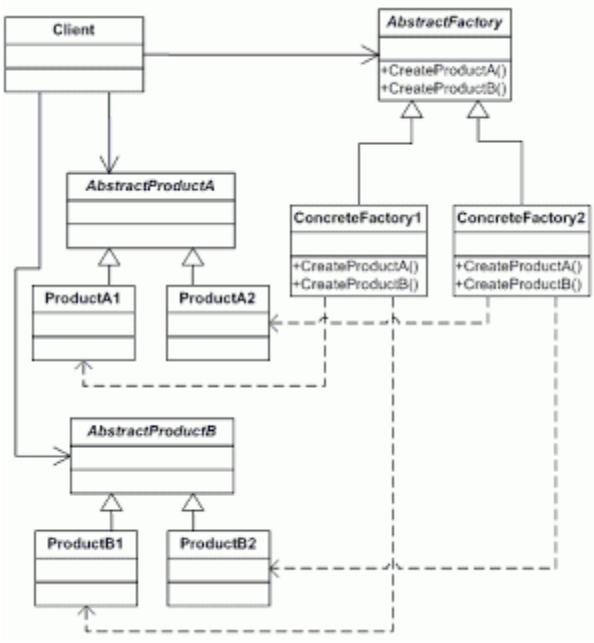


- **Product**：定義工廠方法所建立的物件介面。
- **ConcreteProduct**：具體的產品，實現 **Product** 介面。
- **Creator**：宣告工廠方法，該方法返回一個 **Product** 類型的物件。

- **ConcreteCreator**：重新定義工廠方法，以返回一個 **ConcreteProduct** 實例。

Factory Method Pattern Code

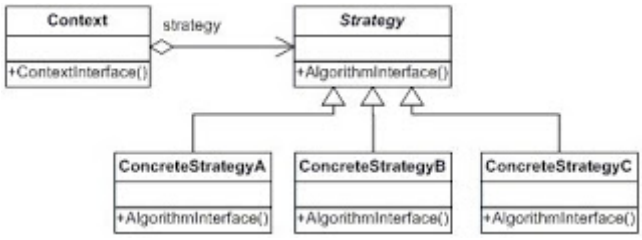
2.抽象工廠模式：



- **AbstractFactory**：抽象工廠介面，包含所有產品建立的抽象方法。
- **ConcreteFactory**：具體的工廠，建立具有特定實現的產品物件。
- **AbstractProduct**：抽象產品。
- **Product**：對抽象產品具體分類的實現。

Abstract Factory Pattern Code

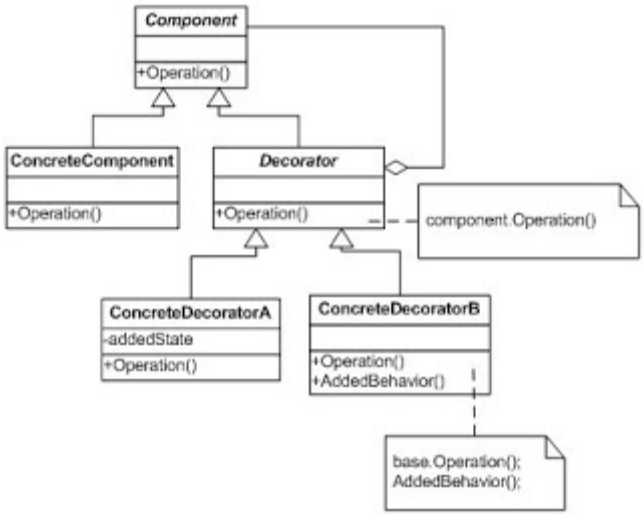
3.策略模式：



- **Strategy**：策略類別，定義所有支援之演算法的公共介面。
- **ConcreteStrategy**：具體策略類別，包裝了具體的演算法或行為，繼承於 **Strategy**。
- **Context**：代表需要改變演算法的那個對象，維護一個對 **Strategy** 物件的引用。

Strategy Pattern Code

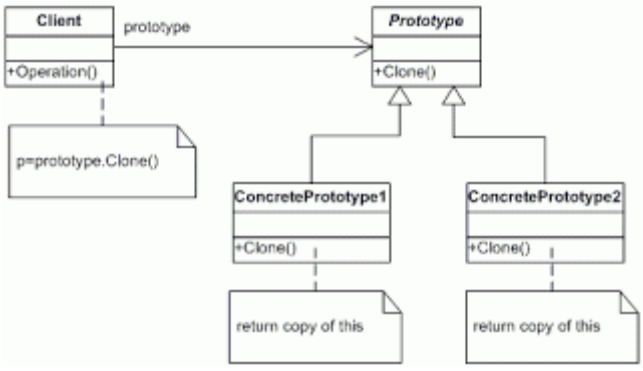
4.裝飾模式：



- **Component**：定義一個物件的介面，可以給這些物件動態地添加功能。
- **ConcreteComponent**：定義一個具體的物件，也可以給這個物件添加功能。
- **Decorator**：裝飾抽象類別，繼承 **Component**，從外類別來擴展 Component 類別的功能。
- **ConcreteDecorator**：具體的裝飾物件，負責為 **Component** 添加功能。

Decorator Pattern Code

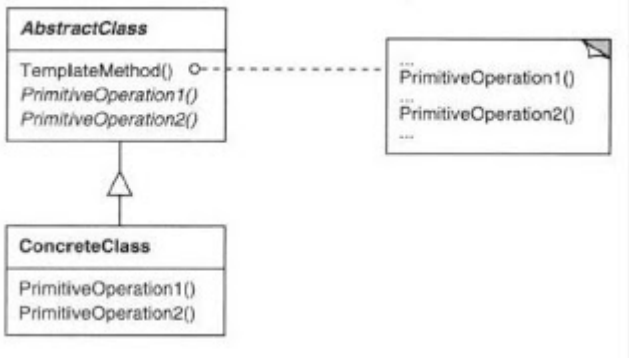
5.原型模式：



- **Prototype**：原型類別，聲明一個複製自身的介面。
- **ConcretePrototype**：具體原型類別，實現一個複製自身的操作。
- **Client**：讓一個原型複製自身，從而建立一個新的物件。

Prototype Pattern Code

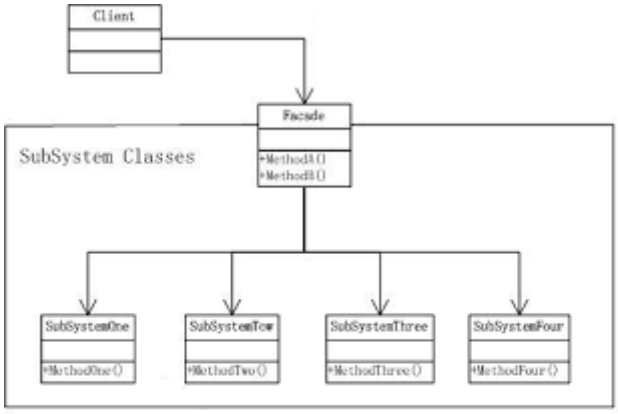
6.範本方法模式：



- **AbstractClass**：實現範本方法，定義演算法的骨架，具體子類別將重新定義 PrimitiveOperation，以實現一個演算法步驟。
- **ConcreteClass**：實現 PrimitiveOperation 以完成演算法中與特定子類別相關的步驟。

Template Method Code

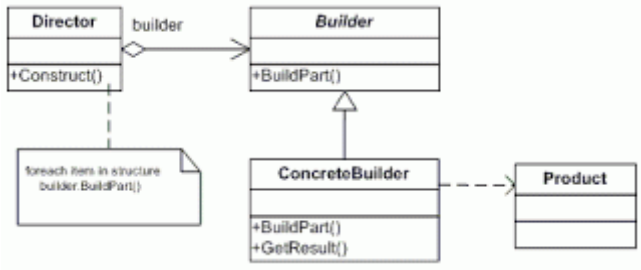
7.外觀模式：



- **Facade**：將請求代理給適當的子系統物件。
- **SubSystem Classes**：實現子系統的功能，處理 **Facade** 物件指派的任務。

Facade Pattern Code

8.建造者模式：

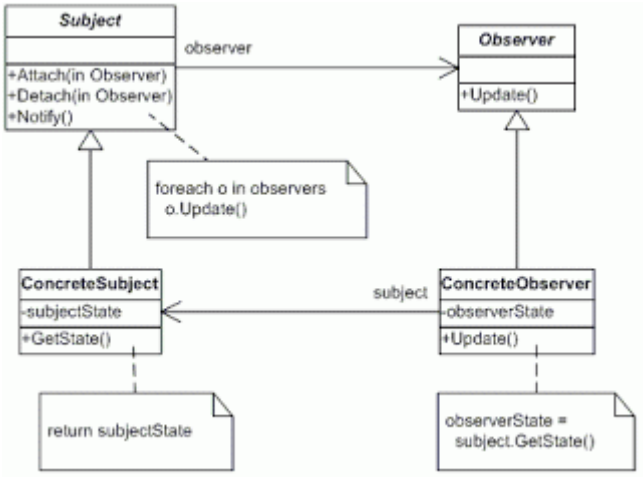


- **Builder**：為建立一個 **Product** 物件的各個零件指定的抽象介面。

- **ConcreteBuilder**：具體建造者，實現 **Builder** 介面，構造和裝配各個零件。
- **Director**：構建一個使用 **Builder** 介面的物件。
- **Product**：具體產品。

Builder Pattern Code

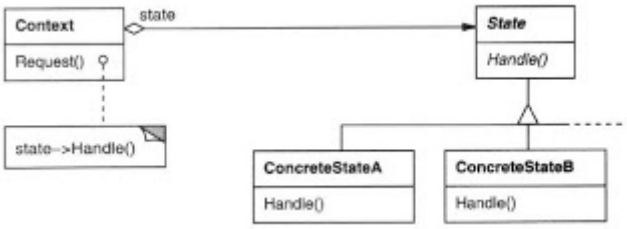
9.觀察者模式：



- **Observer**：抽象觀察者，為所有的具體觀察者定義一個介面，在得到主題的通知時更新自己。
- **ConcreteObserver**：具體觀察者，實現抽象觀察者角色所要求的更新介面。
- **Subject**：把所有對觀察者物件的引用保存在一個聚集裡，每個主題都可以有任意數量的觀察者。抽象主題提供一個介面，可以增加和刪除觀察者物件。
- **ConcreteSubject**：具體主題，將有關狀態存入具體觀察者物件。在具體主題的內部狀態改變時，發出通知給所有登記過的觀察者。

Observer Pattern Code

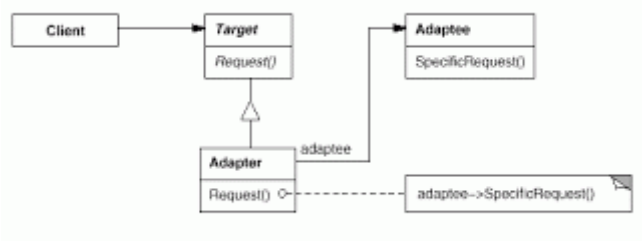
10.狀態模式：



- **State**：抽象狀態類別，定義一個介面以封裝與 **Context** 的一個特定狀態相關的行為。
- **ConcreteState**：具體狀態，每一個子類別實現一個與 **Context** 的一個狀態相關的行為。
- **Context**：維護一個 **ConcreteState** 子類別的實例，這個實例定義現在的狀態。

State Pattern Code

11.轉接器模式：



- **Target**：客戶期待的介面，也可以是具體或抽象的類別。
- **Adapter**：透過在內部包裝一個 **Adaptee** 物件，把原始介面轉換成目標介面。
- **Adaptee**：需要被轉換的類別。

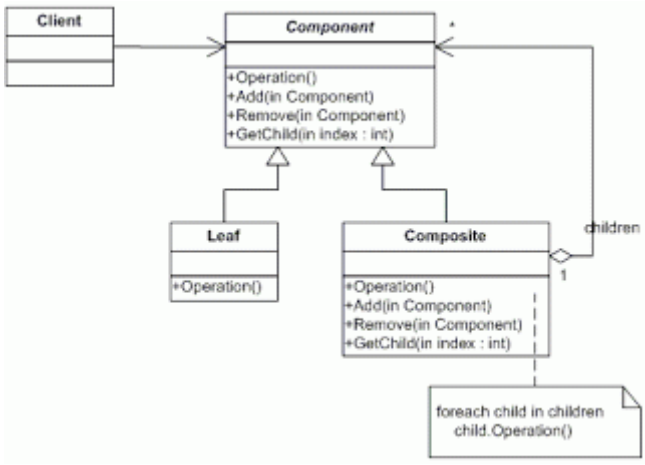
Adapter Pattern Code

12.備忘錄模式：

- **Memento**：負責儲存 **Originator** 物件的內部狀態，並可防止 Originator 以外的其他物件使用 Memento。
- **Originator**：負責建立一個 **Memento**，用以記錄目前它的內部狀態，並可使用備忘錄恢復內部狀態。
- **Caretaker**：負責保存 **Memento**。

Memento Pattern Code

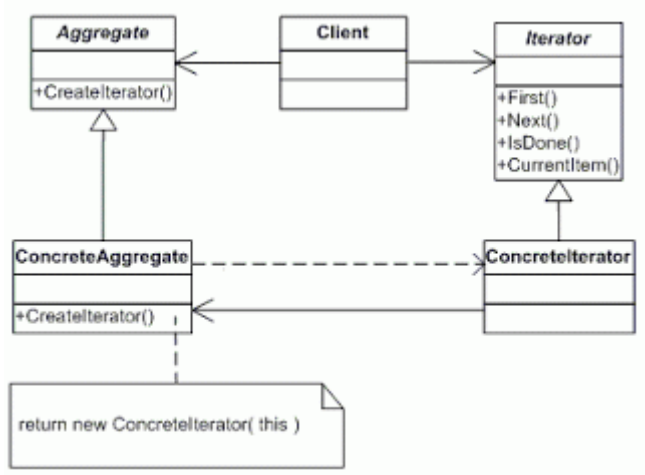
13.組合模式：



- **Component**：物件宣告介面，用於使用和管理 **Component** 的子部分。
- **Composite**：定義分枝節點的行為，用來儲存子部分。
- **Leaf**：葉節點。

Composite Pattern Code

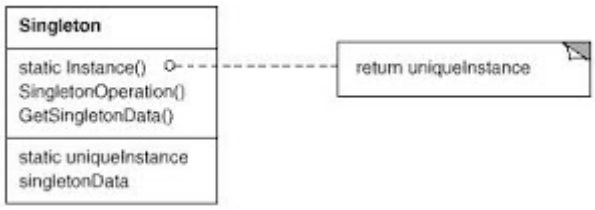
14.迭代器模式：



- **Iterator**：迭代運算抽象類別，用於定義走訪聚合物件的介面。
- **ConcreteIterator**：具體迭代運算類別，實現走訪的方法。
- **Aggregate**：聚集抽象類別。
- **ConcreteAggregate**：具體聚集類別。

Iterator Pattern Code

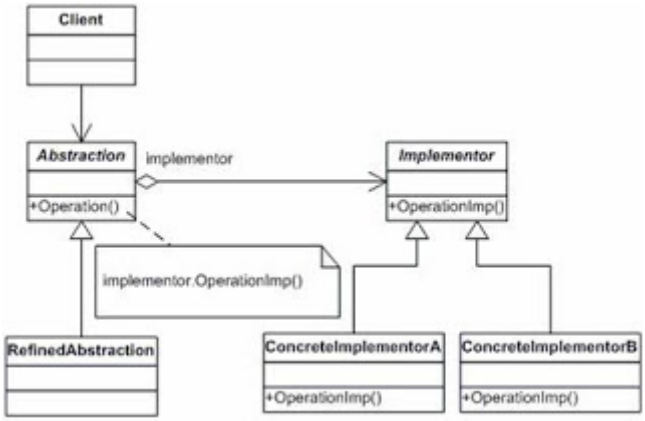
15.獨體模式：



- **Singleton**：定義一個 GetInstance 操作，允許客戶使用它的唯一實例。GetInstance 為一個靜態方法，負責建立自己的唯一實例。

Singleton Pattern Code

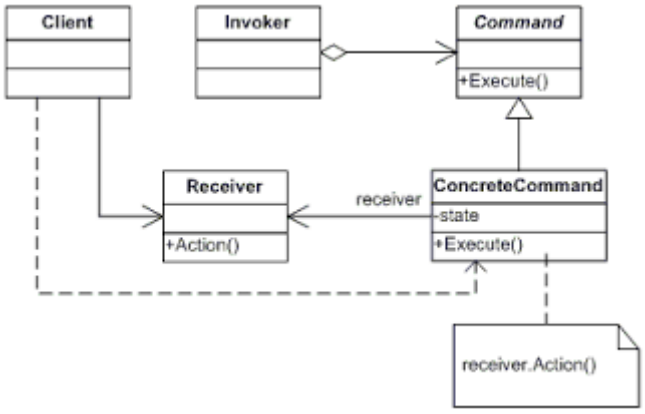
16.橋接模式：



- **Implementor**：實現。
- **ConcreteImplementor**：具體實現。
- **Abstraction**：抽象。
- **RefinedAbstraction**：被提煉的抽象。

Bridge Pattern Code

17.命令模式：

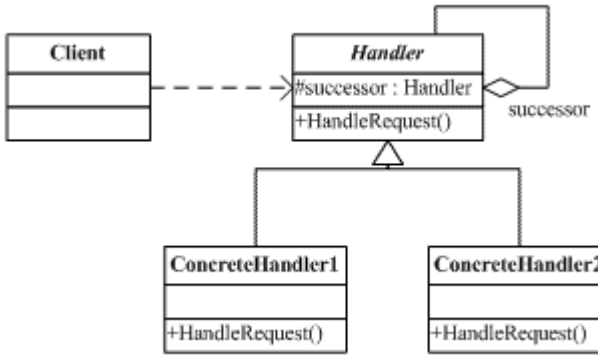


- **Command**：宣告執行操作的介面。

- **ConcreteCommand**：將一個接收者物件綁定於一個動作，呼叫接收者相應的操作，以實現 Execute。
- **Invoker**：要求該命令執行這個請求。
- **Receiver**：實施與執行一個請求相關的操作。

Command Pattern Code

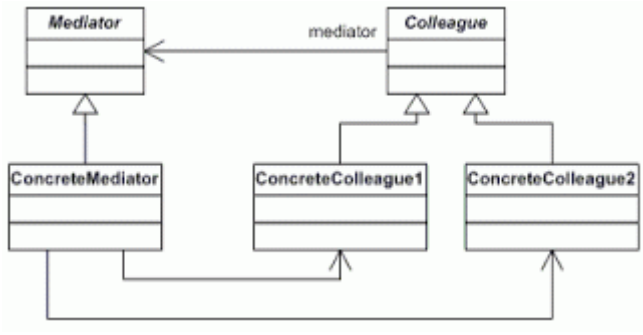
18.職責鏈模式：



- **Handler**：定義一個處理請求的介面。
- **ConcreteHandler**：具體處理者類別，處理它所負責的請求，可使用它的後繼者，若可處理該請求就處理之，否則就將該請求轉發給它的後繼者。

Chain Of Responsibility Pattern Code

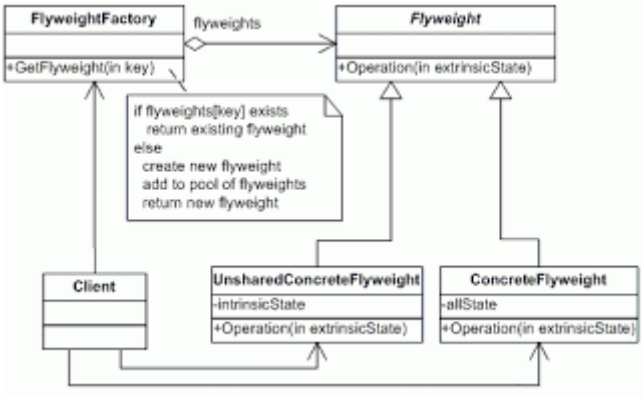
19.仲介者模式：



- **Mediator**：抽象仲介者，定義同事物件到仲介者物件的介面。
- **ConcreteMediator**：具體仲介者物件，實現抽象類別的方法，它需要知道所有具體同事類別，並從具體同事接收消息，向具體同事物件發出命令。
- **Colleague**：抽象同事類別。
- **ConcreteColleague**：具體同事類別，每個具體同事只知道自己的行為，而不了解其他同事類別的情況，但它們卻都認識仲介者物件。

Mediator Pattern Code

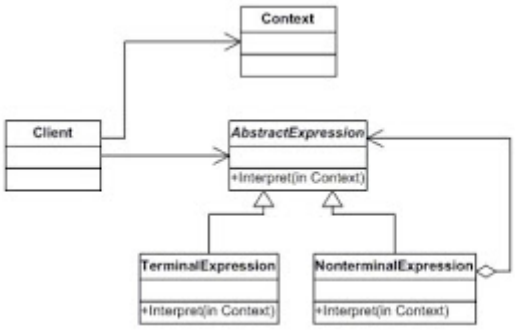
20.享元模式：



- **Flyweight**：所有具體 **Flyweight** 類別的超類別或介面，透過此介面，**Flyweight** 可接受並作用於外部狀態。
- **ConcreteFlyweight**：繼承 **Flyweight** 超類別或實現 **Flyweight** 介面，並為內部狀態增加儲存空間。
- **UnsharedConcreteFlyweight**：不需要共用的 **Flyweight** 子類別。
- **FlyweightFactory**：**Flyweight** 工廠，用來建立並管理 **Flyweight** 物件。

Flyweight Pattern Code

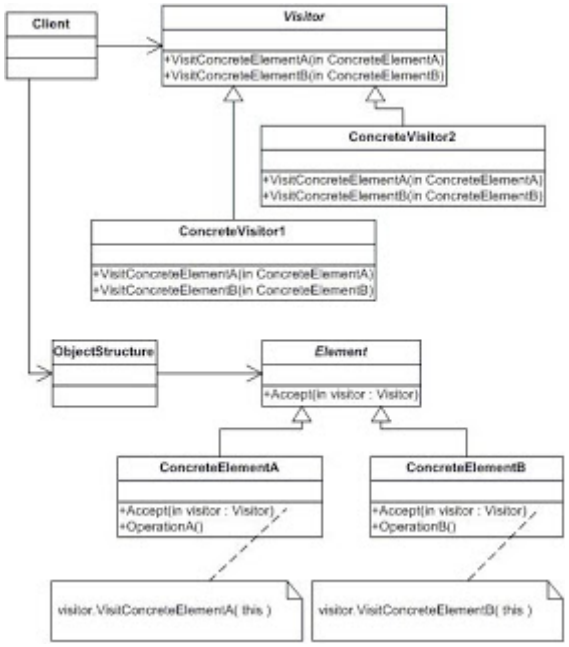
21.解譯器模式：



- **AbstractExpression**：抽象運算式，宣告一個抽象的解釋操作。
- **TerminalExpression**：終結符運算式，實現與文法中的終結符相關聯的解釋操作。
- **NonterminalExpression**：非終結符運算式，為文法中的非終結符實現解釋操作。
- **Context**：包含解譯器之外的一些全域資訊。

Interpreter Pattern Code

22.訪問者模式：



- **Visitor**：為該物件結構中 **ConcreteElement** 的每一個類別宣告一個 Visit 操作。
- **ConcreteVisitor**：具體訪問者，實現每個由 **Visitor** 宣告的操作。
- **Element**：定義一個 Accept 操作，它以一個訪問者為參數。
- **ConcreteElement**：具體元素，實現 Accept 操作。
- **ObjectStructure**：能枚舉它的元素，可以提供一個高層的介面以允許訪問者使用它的元素。

Visitor Pattern Code

四.UML 類別圖畫法：

（1）類別分三層：

- 第一層顯示類別的名稱，若是抽象類別，則以斜體顯示。
- 第二層是類別的特性，通常是欄位和屬性。
- 第三層是類別的操作，通常是方法或行為。
- 「+」表示 public；「-」表示 private；「#」表示 protected。

（2）介面以 <<interface>> 顯示：

- 第一層是介面名稱。
- 第二層是介面方法。
- 實現介面用空心三角形+虛線表示。

（3）關聯表示一個類別「知道」另一個類別。關聯關係以實線箭頭來表示。

（4）聚合關係以空心的菱形+箭頭表示。

（5）合成關係以實心的菱形+箭頭表示。

（6）依賴關係以虛線表示。

PS：《笑談軟體工程》作者 Teddy 有在其「搞笑談軟工」部落格提及過他建議的學習順序如下：

一：Singleton、Factory Method、Adapter、Composite、Command、Observer。

二：Builder、Abstract Factory、Facade、Proxy、State、Template Method。

三：Bridge、Strategy、Visitor、Mediator、Iterator、Prototype。

四：Decorator、Flyweight、Chain of Responsibility、Interpreter、Memento。

提供給各位作參考。