技術筆記

筆記更新紀錄

|  |  |  |
| --- | --- | --- |
| 日期 yyyy/MM/dd | 更新人員 | 更新備註 |
| 2019/07/20 | Miles | 建立 設計模式- Simple Factory Pattern 內容 |
| 2019/07/21 | Miles | 建立 設計模式- Factory Method Pattern 內容 |
| 2019/07/22 | Miles | 建立 設計模式- Abstract Factory Pattern 內容 |
| 2019/07/23 | Miles | 建立 設計模式- SOLID 內容 |

目錄

[1 SOLID 4](#_Toc14871928)

[1 Simple Factory Pattern/Static SimpleFacotry Pattern 5](#_Toc14871929)

[2 Factory Method Pattern 9](#_Toc14871930)

[3 Abstract Factory Pattern 12](#_Toc14871931)

## 1 SOLID

|  |
| --- |
| 簡介與說明 |
| 1. S = Single Responsibility Principle(SRP) 單一職責(單一責任原則)   指一個類別只負責一件事情，從這樣的講解當中，我們要注意不能將功能切得太細導致過度設計(over design)的情況。  例如一個方法(method)就只做一件事情，一個類別(class)就只做一個功能該做的事情，多於的變數(value)、方法(method)就不需要出現。   1. O = Open/Close Principle(OCP) 開放/封閉原則   物件導向設計當中最重要的開放(擴充)與封閉(修改)原則。一套軟體必須保留高延展性，方便日後的更新與修改，必須減少過度的耦合(Coupling)發生。而延展模組就是物件導向中的繼承  ，而延展的方法相當多，例如直接繼承舊有的模組並且覆蓋新功能上去(Override)、繼承抽象類別、透過抽象類別的規範來實現新功能、透過多型(Overload)來實現新功能。   1. L = Liskov Substitution Principle(LSP) Liskov替換   在一個系統中，子類別應該要可以替換掉父類別而不會受到影響程式架構。  子類別應該要可以執行父類別想做的事情。   1. I = Interface Segregation Principle(ISP) 介面隔離   把不同功能的功能從介面中分離出來。   1. D = Dependency Inversion Principle(DIP) 依賴凡轉   高階模組不應依賴低階模組，兩個都應該依賴在抽象概念上;抽象概念不依賴細節，而是細節依賴在抽象概念。意思就是說: 話不能說的太死，盡量講一些概念性的東西。  子類別肯定依賴父類別，而父類別不能去依賴子類別實作的功能。  抽象類別不應該有太多細節，而實作類別依賴抽象細節，意思是說依賴應該是單向的。因為程式設計上最好能達到低耦合，可惜光繼承就是一個依賴，因此單向的依賴是最乾淨的，雙向的依賴就違反常理的依賴會導致程式碼不容易追蹤。 |
| 適用時機 |
| 任何時候，基本功 |
| 參考範例 |
| 無 |
| 參考網站 |
|  |

## 2 Simple Factory Pattern/Static SimpleFacotry Pattern

|  |
| --- |
| 簡介 |
| 1. 稱為Static Simple Factory Pattern 與 Simple Factory Pattern。 2. 對客戶端隱藏產品產生的細節，物件如何生成，生成前是否與其他物件建立依賴關係，客戶端皆不用理會，用以將物件生成方式之變化 與客戶端程式碼隔離。 3. 根據需求，直接由工廠產生該物件出來。 |
| 適用時機 |
| 1. 統一建立物件。 2. 能夠有效率的產生、管理、操作物件。 |
| 參考範例(1) |
| 此範例使用了介面(interface)作為訂定規則的規範，並且建立建立一個獨立的工廠(factory)進行建立物件的依據，工廠負責生產特定需求的物件達到設計需求。   1. 我們設計了一個介面，這個介面專門讀取不同格式的檔案      1. 我們建立了兩個實現這個介面的檔案類型(XML、JSON)    1. JSONLoader =>專門讀取JSON格式      * 1. XMLLoader =>專門讀取XML格式      1. 建立一個Factory      1. 當我們要使用某個類別時，直接使用了工廠來產生這個物件     我們可以從範例中發現，註解的地方為 未使用SimpleFactoryPattern的設計模式，如果我要讀取XML格式內容就必須產生一個新物件。  但是這樣有一個問題就是，如果我今天把XMLLoader類別名稱改成了XALLoader類別名稱的話  那麼範例中的new XMLLoader就會出現錯誤，因為這樣產生了高依賴性。  所以我們統一集中在一個類別(class)當中，並且由這個Factory幫我們產生。  這就是SimpleFactoryPattern的基本範例。  但是以上有一個問題，如果我今天再加入CSVLoader類別(class)的話，我就要去動到Factory當中的getLoader方法(method)，這個行為其實我們違反了SOLID設計原則的OCP原則，所以比較正統的工廠模式有另一種寫法，可以參考 工廠方法模式(Factory Method Pattern)。  我們可以從程式碼中看到  C:\Users\ASUS\Desktop\擷取.JPG  如果今天我們把JSON改成CSV，那麼代表著  C:\Users\ASUS\Desktop\擷取.png  我們必須把LoaderFactory內的getLoader方法進行修改，並且在LoaderType 的enum進行添加CSV相關的程式碼。以上行為就會違法SOLID當中的O(OCP)原則。   1. UML |
| 參考網站 |
|  |

## 3 Factory Method Pattern

|  |
| --- |
| 簡介 |
| 1. 定義一個建立物件(Object)的介面(inferface)，但讓實現(implements)這個介面的類別(class)來決定實體化哪個類別。(出處:維基百科) 2. 由子類別(subClass)來實現這個方法來建立具體類別的物件。(出處:維基百科) 3. 工廠方法帶來的方便在，無須修改原本的程式碼，在符合OCP原則情況之下，當產品發生改變時，只要在新增功能上去即可。 4. 缺點是只能建立單一類型的功能。 5. 與SimpleFactory不同的是，我們將工廠也一併切分了，而不是一個工廠統一生產多種不同類型物件。 6. 將設計一個產生物件的規範(interface)，讓子類別(sub-class)來實現規範建立物件。 7. 未來進行擴充更容易。(就是在建立一個新的工廠) |
| 適用時機 |
| 1. 建立物件需要大量重複的程式碼。 2. 有效率的產生、管理、操作物件。 |
| 參考範例(1) |
| 1. 我們建立了 TV 的規範，也就是介面(interface)或者抽象(abstract)，範例中使用介面。     這個規範就是，所有的TV都具備開與關的功能。   1. 並且建立了兩個類別來實現這個TV(練習用)。        1. 建立一個工廠的介面(interface)用來規範建立的標準。      1. 並且子類別來實現這個工廠規範。     這裡要特別注意，我們將產生TV的規範切分，不同類型的TV由不同的工廠(Factory)處理。     1. 最後由呼叫端來決定要建立哪一個物件。     範例當中我們決定產生ProductTV\_B 所以我們直接呼叫專門產生ProductTV\_B的工廠(Factory)來進行產生實體動作。   1. UML   C:\Users\ASUS\Desktop\Class Diagram3.jpg |
| 參考網站 |
|  |

## 4 Abstract Factory Pattern

|  |
| --- |
| 簡介 |
| 1. 提供一個建立一系列相關或相互依賴物件的介面，而無需指定它們具體的類別。 2. 管理有關聯性的物件。 |
| 適用時機 |
|  |
| 參考範例(1) |
| 1. UML     由子類別來產生對應的實體。   1. 假設:Audi和BMW都有生產休旅車，而我的工廠專精生產休旅車，不一定需要會做某品牌全系列的商品。在我想要生產其他車種時只需要抽換具體工廠即可(更換生產的產品)。 2. 定義Audi的介面     定義BMW的介面     1. 定義產品            1. 抽象工廠可以製作某品牌的Jeep或是SUV      1. 不同工廠產生不同廠牌的車種        1. 測試的方式     可以看到，當我們要產生不同的車種，我們可以直接抽換成要產生的工廠，這樣就不會有問題了。 |
| 參考網站 |
| <https://ithelp.ithome.com.tw/articles/10208955> |

## 5 Adapter Pattern(Object Adapter/Class Adapter)

|  |
| --- |
| 簡介 |
| 1. 又稱之為 轉接器。 2. 將兩個不相容的類別結合在一起使用，屬於結構型模式，需要有Adaptee(被適配者)與Adapter(適配者)兩個身分。 |
| 適用時機 |
| 1. 當有需要將兩個沒有任何關係的類別組合使用。 |
| 參考範例(1) |
|  |
| 參考網站 |
|  |

## 6

|  |
| --- |
| 簡介 |
|  |
| 適用時機 |
|  |
| 參考範例(1) |
|  |
| 參考網站 |
|  |

## 標題

|  |
| --- |
| 簡介 |
|  |
| 適用時機 |
|  |
| 參考範例(1) |
|  |
| 參考網站 |
|  |