搞笑談軟工

敏捷開發，設計模式，精實開發，Scrum，軟體設計，軟體架構

2012 年 1 月 6 日 星期五

亂談軟體設計（6）：Single Choice Principle

今天先把課本換成 Object-Oriented Software Construction, 2nd 來看一下 Single Choice Principle。原本以為這個設計原則比較容易寫，但是為了寫這一篇 Teddy 又把課本的 61-63 頁看了一次，覺的還是有點虛虛的（還好本系列主題是「亂談」軟體設計...XD）。後來用 google 找了一下，看到兩篇有趣的討論：

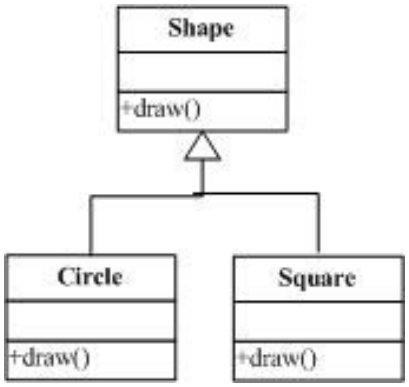
- http://c2.com/cgi/wiki?SingleChoicePrinciple
- http://www.geocities.com/tablizer/meyer1.htm#singlechoice

這兩篇討論的內容有興趣的鄉民請自行參閱，Teddy 分享一下自己對於此原則的簡單認知。先看一下課本 63 的定義：

> **Single Choice Principle**
>
> > *Whenever a software system must support a set of alternatives, one and only one module in the system should know their exhaustive list.*

（當一個軟體系統必須要支援一組替代方案的時候，系統中應該只有一個模組知道這些替代方案的所有清單。）

依照往例，看完定義之後還是不知道這個原則是什麼東東，看例子比較快。還記得之前在「亂談軟體設計（4）：Liskov Substitution Principle」提過的繪圖系統範例嗎？假設鄉民們要設計一個繪圖系統，可以畫圓（Circle）與正方形（Square）等形狀，於是鄉民們設計了一個父類別叫做 Shape（形狀），然後讓 Circle 與 Square 都繼承自 Shape。



這些畫在螢幕上的圖，最後被存成文字檔案。文字檔內容長成類似下面這樣子：

```
define shape {
    type=circle
    location=25,6
    ...
}

define shape {
    type=square
    location=36,10
    ...
}
```

現在鄉民們要寫程式把這段文字檔的內容讀出來，然後把檔案中所紀錄的圖形畫在畫面上，其中有一段程式可能長成下面這樣：

```
List shapes = new ArrayList();
if (type=="circle") then {
   shapes.add(new Circle(...));
}
else if (type=="square") then {
   shapes.add(new Square(...));
}
...
else if (type=="XXX") then   {
   shapes.add(new XXX(...));
}
```

為了把所有的圖形物件都讀出來，上面這段程式必須要「詳盡列舉（know their exhaustive list）」出所有 Shape 類別的子類別（a set of alternatives）。看到這邊有鄉民會說，這種 if then else 的

寫法，不是 OO（物件導向）的程式。說得很對，Teddy 曾經在某本書（又忘了是那一本）看到一種說法：「物件導向程式設計就是要把程式中的 if-then-else（或是 switch-case）全部拿掉」。這可能嗎？至少上面這段程式碼是可能的，要如何做？先看一個不花腦經的範例：

- 首先，幫產生每個物件的這段程式（例如 new Circle(...)）用一個 Command pattern 包起來，例如：

```
interface IShapeFactory{
      Shape newInstance(argumens...);
}

class SquareFactory implements IShapeFactory{
     public Shape newInstance(argumens...) {
          return new Square(argumens);
     }
}
```

- 然後宣告一個 HashMap<string, ishapefactory> ，HashMap 的 key 存放 Shape type name，value 存放 IShapeFactory 的實做（implementation，例如 SquareFactory）。
</string, ishapefactory>

- 接著初始化這個 HashMap。

```
HashMap<string, ishapefactory> factoryMap = new
       LinkedHashMap<string, ishapefactory>();</string, ishapefactory></string, ishapefactory>
factoryMap.add("square", new SquareFactory() );
factoryMap.add("circle", new CircleFactory() );
  ...
factoryMap.add("XXX", new XXXFactory() );
```

- 然後就可以用一個迴圈（loop）來取代原本的 if-then-else

```
List shapes = new ArrayList();
for (String shapesFromFile: shape) {
     if   factoryMap.containsKey(shape) {
          shapes.add(factoryMap.get(shape).newInstance(...));
     }
     else {
          throw new RuntimeException("Unsupported shape: " + shape);
     }
}
```

不管是那一種方法，程式中都有一個地方需要去「詳盡列舉（know their exhaustive list）出所有 Shape 類別的子類別（a set of alternatives）」（範例一的 if-then-else 以及範例二標成藍色的那幾行字）。就算是鄉民們把範例二標成藍色的那幾行字從程式碼中抽離出來放到設定檔中，還是需要在設定檔中列舉出所有 Shape 類別的子類別。

結論就是，Single Choice Principle 告訴我們，當有這樣的情況出現的時候，one and only one module in the system should know their exhaustive list，為什麼？因為這個列表可能會隨著時間而改變，例如，假設原本繪圖系統不支援多邊形（Polygon），但是後來因為客戶要求而加入了 Polygon 這個繪圖類別。如果程式中有一個以上的地方都必需要列舉出所有 Shape 類別的子類別，那麼當新增了 Polygon 類別之後，就要記得去改到每一個需要修改地方，這樣的程式是很容易出現 bugs 的。

又是扯了一堆，其實 Single Choice Principle 可以簡單看成是一種對於「避免重複程式碼（duplicated code）」的特殊要求。因為如果程式可以做到 Single Choice Principle，除了可以避免 duplicated code 以外，還可以支援之前介紹過的「Open-Closed Principle」。為什麼，請鄉民們花一分鐘想一下。

其實答案很簡單，就在 Teddy 前面舉的那兩個程式範例中。Teddy 之前提過 Open-Closed Principle 的最高境界：

藉由增加新的程式碼來擴充系統的功能，而不是藉由修改原本已經存在的程式碼來擴充系統的功能

> *(If the OCP is applied well, then further changes of that kind are achieved by adding new code, not by changing old code that already works.)*

對啊，在繪圖系統的例子中，理想上鄉民們可以藉由新增一個 Polygon 類別，然後把這個 Polygon 類別「外掛」到系統中，不用修改系統的程式就以擴充新的功能。但是，實際上還是要有人去負責處理這些「外掛」，不管這些處理外掛的邏輯是直接寫成 if-then-else，或是用比較物件導向的方法用迴圈取代，或是寫在外部設定檔中（ini 檔或是 xml 檔案），如果能做到 Single Choice Principle，只有一個人（程式模組）需要知道處理這些所有外掛的邏輯，那麼這樣的設計就算是有支援到 Open-Closed Principle。

講完收工，吃中飯去。

友藏內心獨白：以前都是寫完睡覺，現在變成寫完吃中飯。
張貼者： Teddy Chen 於 下午 12:29