第 5 章

Observer 樣式

5.1 Observer 樣式美何在?

框架裡的抽象類別與應用類別之結合不只追求一時的連結，更要追求持久的和諧與成長。Observer 樣式帶來優越互換性、順暢新陳代謝，乃是締造持久和諧與成長之妙方。此 Observer 樣式之美，也帶來系統整體和諧之美。

5.1.1 互換性之意義

爲了整體和諧之美，系統的新陳代謝必須順暢，也就是，應用類別必須能隨時迅速汰舊換新、分合自如。此汰舊換新之效果通稱爲 PnP(Plug and Play)，即系統內之類別容易抽換與修理。隨著軟硬體業的發展，愈來愈重視其組件之抽換性(又稱爲互換性)，這是軟硬體產業發展必經之路。就如同，造船業自從十五世紀以來，就致力追求「標準化」和「抽換性」，成爲造船工業製程與品質管制的基礎要求。例如：黃克東教授在其文章----文藝復興與現代文明裡提到：

> 義大利威尼斯(Venice)成為海上強權後，為維持海上貿易船隊的秩序與安全，要擁有自己的武裝艦隊，西元 1436 年威尼斯政府出面經營造船廠，兼造武器，故改名為「兵工廠」(Arsenal)。..... 兵工廠的設計委員會，認為「標準化」與「互換性」是造船工業製程與品質管制的基礎要求，明訂以下標準化與互換性的要求，例如：
>
> 1. 所製造的弓，必需適合發射任何箭矢之用(即弓與箭規格介面統一，能互換使用)。
> 2. 所有的舵柱，必需照同一尺寸與樣式製造，以便每隻舵的成品，可拿來在別港換裝修理。
> 3. 所有船具包括桅桿、索具等規格介面均需一致，列為船載預備品，可在別港甚至航行中更換修理。
>
> 當時威尼斯人，已經重視標準化與互換性，有益於快速製造與裝配。在使用方面，「標準化」將使操作船舶航行技術一致，速度快捷，及調運靈活。「互換性」可使維修更為簡易便捷，使所有服役航行的船艦如同一支船(艦)隊。」

5.1.2　提升互換性之途徑：降低相依性

提升「抽換性」的有效方法是：降低相依性(Dependency)。無論是汽車、音響、電腦硬體或是軟體，其零組件之間的相依性，常決定了整個系統的穩定性以及品質，也決定了產品的製程與上市時間。因而，必須要細膩、精緻地設計零件之間的相依性，使能夠品質與效率兼具地管理系統整體，並且可以依照相關需求，隨時作出適切的供給，創造出供需雙方兼贏的高效益。

如果相依性能夠被設計得既巧妙又適當，呈現出美好的佈局，則零件之間的合作，就會如同俗語所說的：「三個臭皮匠，勝過一個諸葛亮」。由於零件之間能夠彈性合作，所以獲得和諧的整體。反之，如果相依性設計得拙劣的話，就會如同古代三個和尚沒水喝的故事，三位和尚無法溝通合作，寺廟失去整體和諧，大家都沒水喝了。重視零件的相依性管理，既追求創新速度，也兼顧穩定品質。

請您看看哈佛企管雜誌(Harvard Business Review)裡，Lee Fleming 和 Olav Sorenson 兩位專家，關於零件模組化對產品開發創新的影響所提出之研究，其中特別強調零件之間的相依性是個成功的關鍵因素。他提到[參 1]：

> “The process of innovation is, virtually by definition, filled with uncertainty; The interdependence of components has an enormous effect on the pace and complexity of the innovation process.”
> (開發創新的過程充滿著不確定性，….元件之間的相依性大大影響到開發創新的速度以及複雜度。)

他舉新力(SONY)公司的隨身聽產品爲例，說明因美好的相依性設計和管理而得到的美妙效果。所以他又繼續說到：

> “In modular designs, changing one component has little influence of others or on the system as a whole.”
> (在美好的模組化設計裡，更換單一零件時，對其它零件或系統整體的影響都是很小的。)

新力公司的工程師採取高度「標準化」及「抽換性」之零件，而迅速設計出市場上極受歡迎的隨身聽音響。另一方面，Lee Fleming 和 Olav Sorenson 也舉了拙劣管理零件相依性的例子，他說：

> “Consider the ink-jet printer. First proposed by Loard Kelvin in 1867, it took more than a hundred years to become commercially viable, even after

millions of dollars of investment by …. ”
(以噴墨印表機爲例，自從 1867 年由 Loard Kelvin 提出以來，儘管投入大量金錢，仍然在 100 多年後才開始商業化，…)

他說明其主要原因：

“The culprit: severe interdependence of the components, including the chemistry of the ink, the physical layout and composition of the resistors, and son on.”
(其癥結在於：零件間的高度相依性。包括墨水的化學性質，零件的安排方式、電阻體的組合等，彼此之間都密切相依。)

最後，他歸納說：

“Our research indicates that intermediate levels of interdependence produce the most useful inventions.”
(研究顯示，適度的相依性，能產出最有用的新產品。)

精緻而巧妙地管理相依性，就能達到如同 SONY 的隨身聽產品一般，更換單一組件時，幾乎不對其他零件或整個系統產生影響，如此即能促進系統之新陳代謝，維持系統的持久和諧。

5.1.3 Observer 樣式之美：締造互換性

隨著應用框架的普及，其背後的反向控制(IoC, Inversion of Control)觀念和機制也流行起來。像 Android 這種幕後含有 IoC 機制的框架能大幅降低系統類別之間的相依性；也就大幅提升系統整合的彈性和自由度。此 Android 框架之美來自何方呢？在第 2 章裡曾經介紹過，反向控制是應用框架魅力的泉源。其常見的實現機制有二：

1) 繼承(Inheritance) + 卡榫函數
   --- *在 Template Method 樣式裡，可看到其典型的用法。*
2) 委託(Delegation) + 卡榫函數
   --- *在 Observer 樣式裡，可看到其典型的用法。*

Android 是個完整的框架，處處可見反向控制的結構，而且都依賴上述的兩種實現機制。在前面第 3 章裡已經介紹過「繼承 + 卡榫函數」實現機制：Template Method 樣式。在本章裡將介紹「委託 + 卡榫函數」實現機制：Observer 樣式，並說明它如何締造出高度的組件互換性。

5.2　介紹 Observer 樣式

5.2.1　委託式的反向控制(IoC)

在前面兩章所介紹的 Template Method 和 Factory Method 樣式都是將「會變」的部份委託給子類別，當有所變化時，就抽換掉子類別，換上新的子類別就行了。由於該子類別與其抽象類別之間具有「繼承」關係，所以就通稱爲：繼承方式的反向控制，其結構如下圖所示：

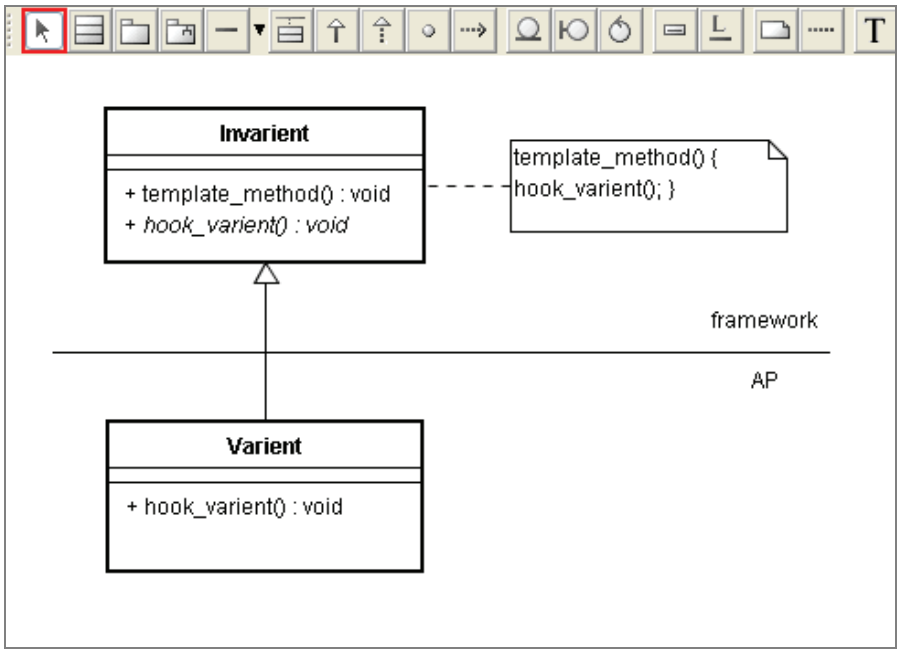


圖 5-1　繼承方式的 IoC(即 Template Method 樣式)

此圖裡的 Invarient 父類別含有「不變」的部份，而 Varient 子類別則含有「會變」的部份，而達到「變與不變分離」之目的。在第 3 章裡，已經寫過多個像上圖 5-1 的範例程式了。

現在要介紹另一種形式的反向控制，則是某一個類別將「會變」的部份委託另一個類別，這兩個類別不必具有繼承關係，而只須具結合(Association)或包含(Containment)關係。我們稱此爲：委託方式的反向控制。其目的也是一樣的，就是達到「變與不變分離」之目的。例如兩類別具有結合關係，如下圖：

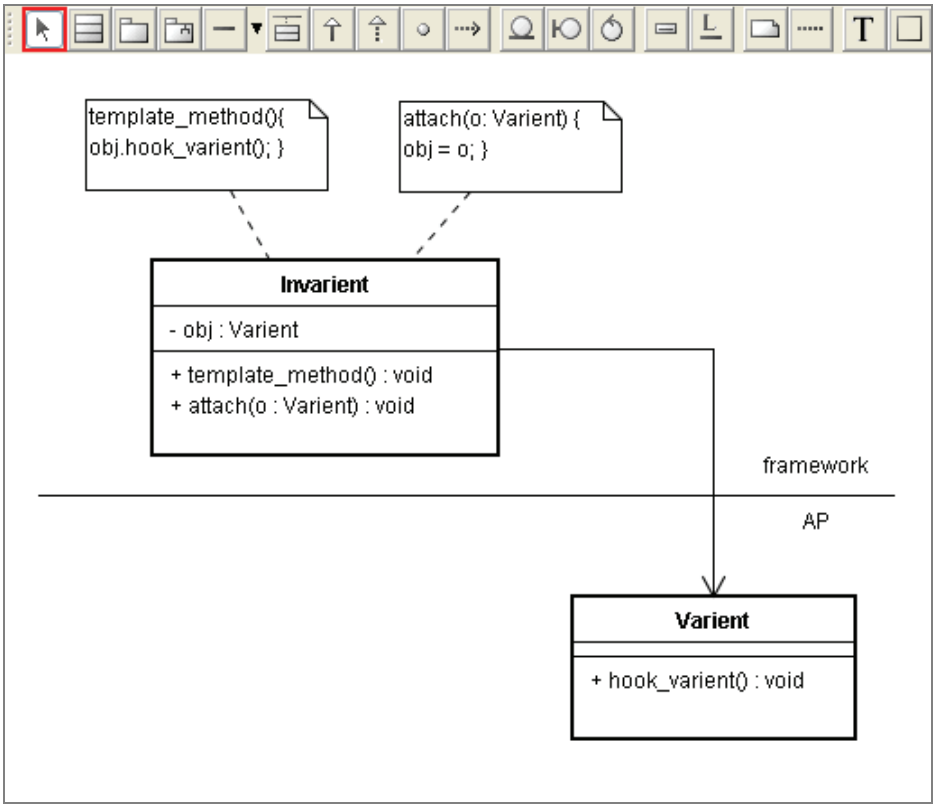


圖 5-2　變與不變之分離

此圖裡的 Invarient 類別含有「不變」的部份，而 Varient 類別則含有「會變」的部份，但是兩者之間不是類別繼承關係，而是結合關係。茲以 Java 程式來實現

上圖 5-2，如下：

<<撰寫程式>>

Step-1. 建立一個 Java 應用程式專案：Ex05-01。

Step-2.　撰寫 Invarient 類別。

```
// Invarient.java
public class Invarient {
        private Varient obj;
        public void template_method(){    obj.hook_varient();      }
        public void attach(Varient o){      obj = o;      }
}
```

Step-3.　撰寫 Varient 類別。

```
// Varient.java
public class Varient {
        public void hook_varient(){
            System.out.println("hook_varient...");
    }}
```

Step-4.　撰寫 JMain 類別。

```
public class JMain {
   public static void main(String[] args) {
   Invarient iv = new Invarient();
   iv.attach(new Varient());
   iv.template_method();
}}
```

<<說明>>

由於 Varient 包含著「會變」的部份，包括其名稱："Varient"本身都是「本身都是「會變」的。因而 Invarient 類別的 attach()函數：

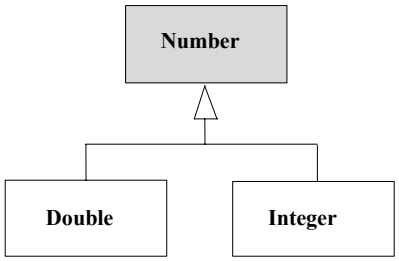
```
public class Invarient {
        .......
        public void attach(Varient o){
                ......
          }
    }
```

其參數："Varient"是「會變」的了。這表示此 Invarient 類別並未純粹只含有不變部分，我們必須進一步移除會變的部分。其基本做法是：建立參數之抽象類別。爲了演練這個基本做法，請個下述的簡單程式碼：

```
public class JMain {
    private void print(double x, int y)
                {System.out.println(x+y);      }
    private void print(int k, int y)
                {System.out.println(k*y);      }

    public static void main(String[] args) {
          JMain mObj = new JMain();
          mObj.print(3.6, 6);
          mObj.print(2, 60);
    }
}
```

這兩個 print()函數裡參數型態並不相同，屬於會變的部份。這跟上述的 attch(Varient o)函數一樣是會變的部份。此時，常見的解決方法是：建立參數之抽象類別，如下：



則兩個 print()函數之參數就一致了，如下：

```
public void print( Number numb,    int y ) {
        ……
    }
```

於是，print()函數就移除掉會變部份了。於是上述的程式碼可改寫爲：

```
// Number.java
public abstract class Number {
        public abstract String hook_method(int y);
}

// Integer.java
public class Integer extends Number{
       private int x;
       public Integer(int i)    {    x = i;    }
        public String hook_method(int y)
                {    return String.valueOf( x * y );    }
   }

// Float.java
public class Double extends Number {
        private double x;
        public Double(double a)        {    x = a;    }
        public String hook_method(int y)
                {    return String.valueOf(x + y);    }
}

// JMain.java
public class JMain {
        private void print(Number numb, int y)
                {      System.out.println(numb.hook_method(y));      }

        public static void main(String[] args) {
                JMain jm = new JMain();
                Double a = new Double(3.6f);
                Integer b = new Integer(2);
                jm.print(a, 6);    jm.print(b, 60);
   }
}
```

依據上述的做法，就可爲 Varient 類別建立一個抽象的 Observer 類別，如下圖：

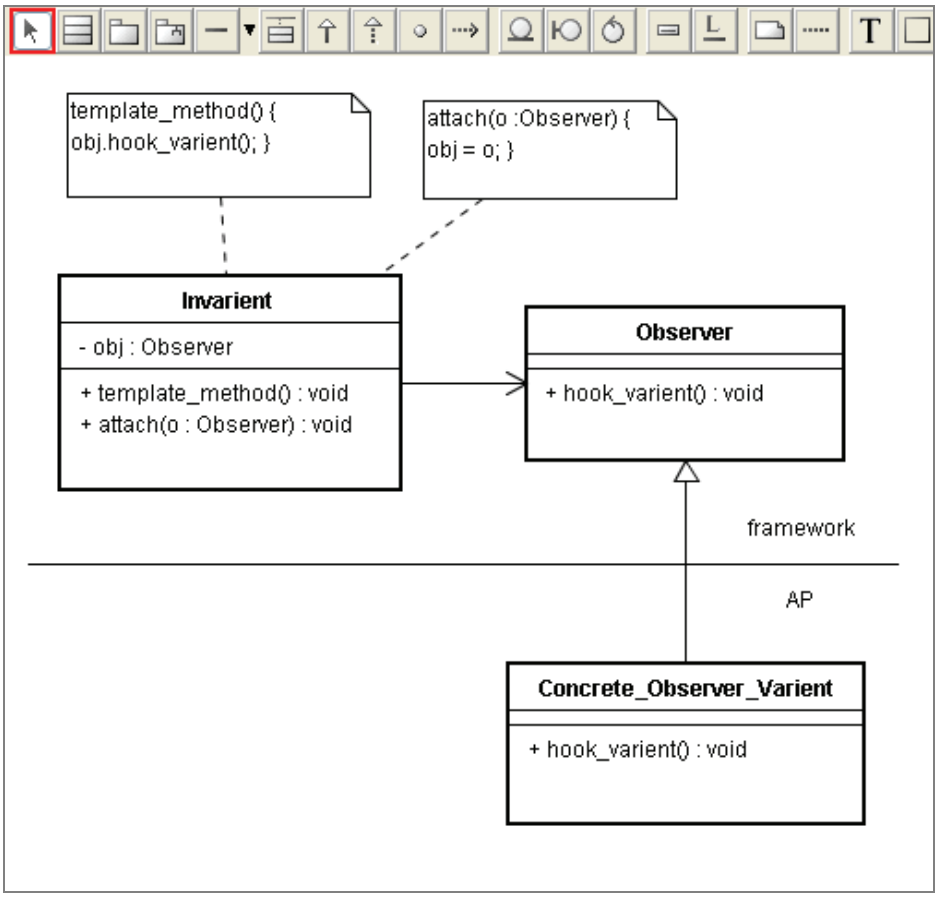
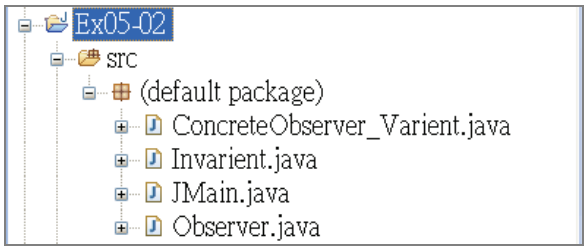


圖 5-3　建立抽象類別

於是，Invarient 類別就純粹翠只含有「不變」部份了。茲以 Java 程式來實現上圖 5-3，如下：

<<撰寫程式>>

Step-1. 建立一個 Java 應用程式專案：Ex05-02。



Step-2.　撰寫 Invarient 類別。

```
// Invarient.java
public class Invarient {
        private Observer obj;
        public void template_method()
                {     obj.hook_varient();       }
        protected void attach(Observer o)
                {    obj = o;     }
}
```

Step-3.　撰寫 Observer 類別。

```
// Observer.java
public abstract class Observer {
        public abstract void hook_varient();
}
```

Step-3.　撰寫 ConcreteObserver_Varient 類別。

```
// ConcreteObserver_Varient.java
public class ConcreteObserver_Varient extends Observer {
        public void hook_varient(){
            System.out.println("hook_varient...");
    }}
```

Step-4.　撰寫 JMain 類別。

```
// JMain.java
public class JMain {
   public static void main(String[] args) {
            Invarient iv = new Invarient();
            iv.attach(new ConcreteObserver_Varient());
            iv.template_method();

}}
```

<<說明>>

此 Invarient 類別只剩下不變的部分了，應納入框架裡。讓應用類別開發者可以從它衍生出子類別，以便包含其它會變的部份，或者誕生物件等，如下圖：

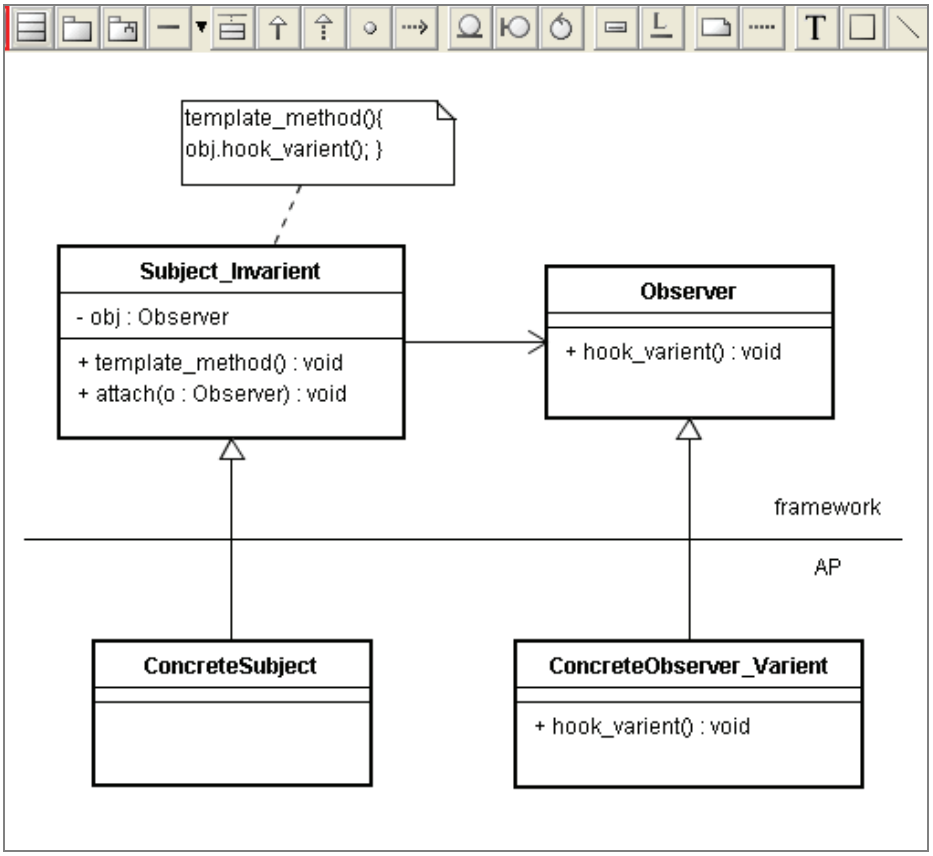


圖 5-4　從 Invarient 衍生出子類別

此 Subject_Invarient 與 Observer 類別之間具有結合關係，因而 ConcreteSubject 與 ConcreteObserver_Varient 兩類別也繼承此項關係。

5.2.2　GoF 的 Observer 樣式圖

在上圖 5-4 裡，ConcreteSubject 類別之函數隨時可以呼叫父類別的 template_method()函數，進而呼叫 Observer 的 hook_varient()函數，然後反向呼叫到 ConcreteObserver_Varient 子類別的 hook_varient()函數。所以當 ConcreteSubject 之物件的內容或狀態有所變化時，皆能夠及時通知(透過 AF 裡的父類別)ConcreteObserver_Varient 子類別之物件。換句話說，ConcreteObserver_Varient 之物件一直觀察著 ConcreteSubject 之物件的狀態變化。因之，在 GoF 的<<Design Patterns>>一書裡，稱之爲「Observer 樣式」，如下圖所示：

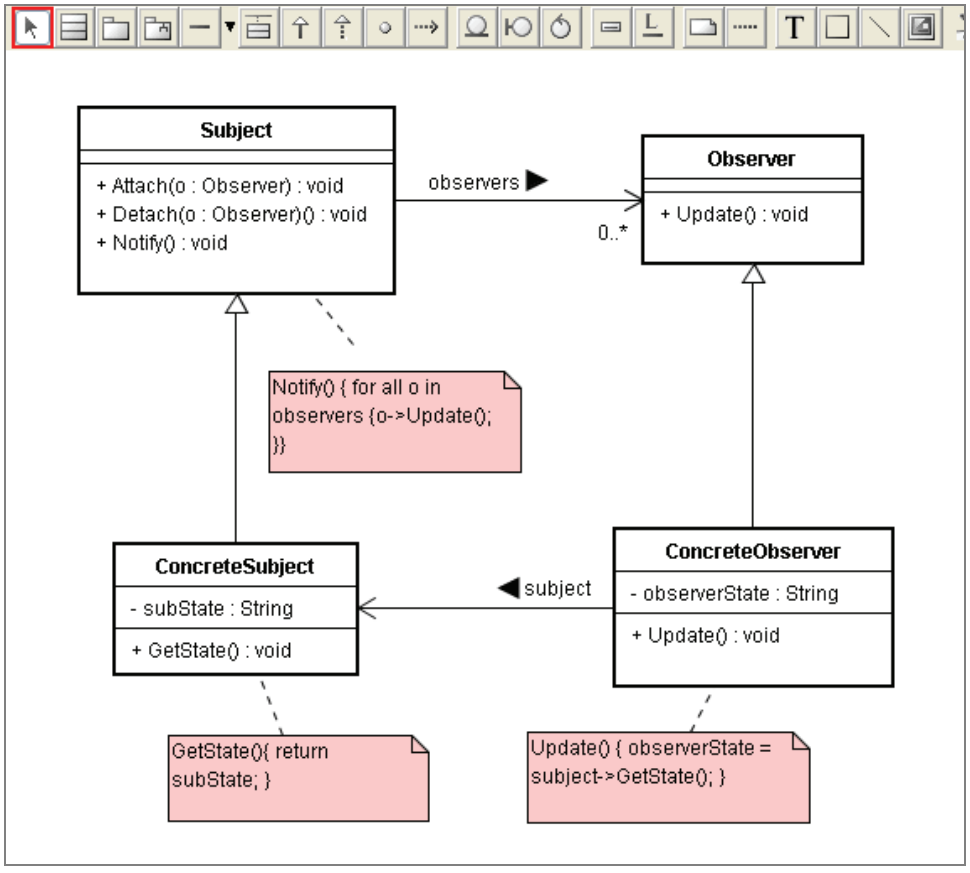


圖 5-5　GoF 的 Observer 樣式圖

5.2.3 Observer 樣式的延伸

在上面圖 5-4 和 5-5 裡，如果 Observer 是個純粹抽象類別(Pure Abstract Class)，它扮演介面角色，就相當於 Java 語言的 interface 機制，如下圖：

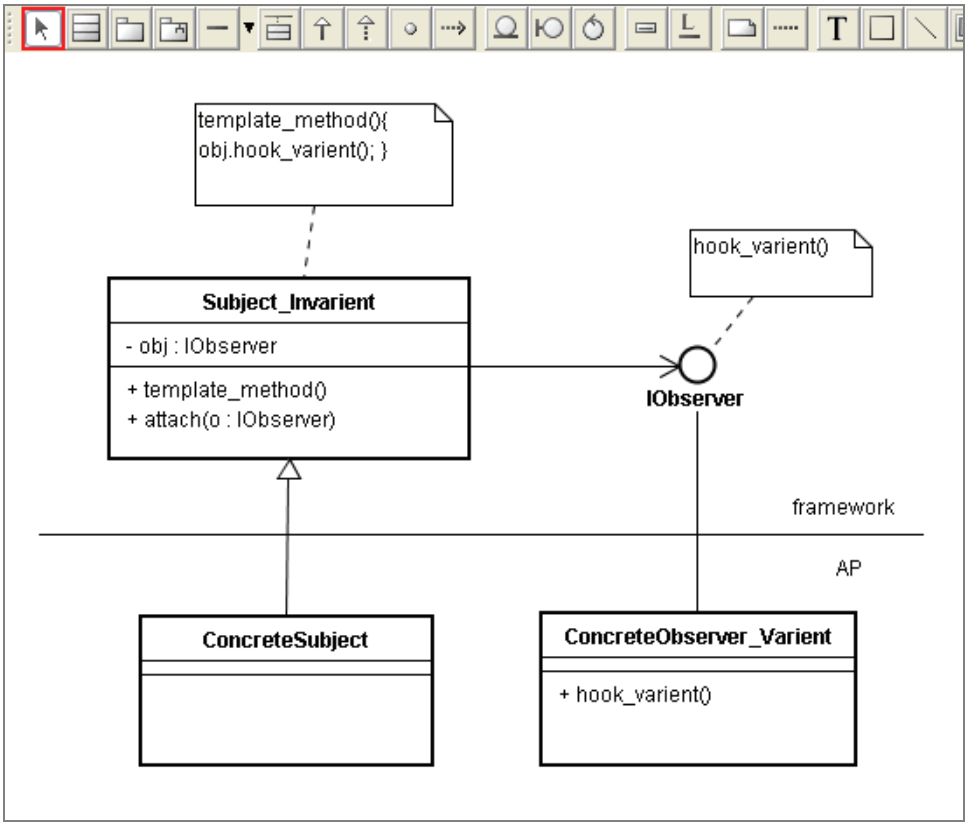
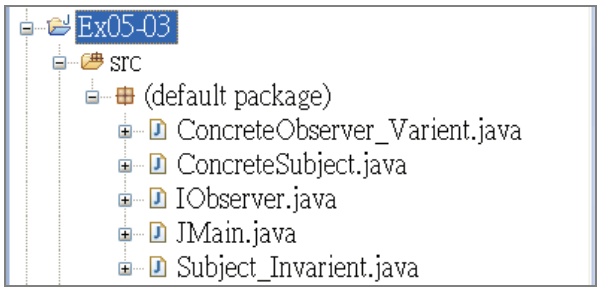


圖 5-6 善用 interface 機制

雖然父類別Observer已經變爲IObserver介面了，其卡榫函數還是存在那裡，只是形式有些變化而已。茲寫個 Java 程式來實現上圖 5-6，如下：

<<撰寫程式>>

Step-1. 建立一個 Java 應用程式專案：Ex05-03。



Step-2.　撰寫 Subject_Invarient 類別。

```
// Subject_Invarient.java
public class Subject_Invarient {
        private IObserver obj;
        public void template_method()
                        {    obj.hook_varient();    }
    protected void attach(IObserver o)
                        {    obj = o;    }
}
```

Step-3.　定義 IObserver 介面。

```
// IObserver.java
public interface IObserver {
        void hook_varient();
}
```

Step-4.　撰寫 ConcreteObserver_Varient 類別。

```
// ConcreteObserver_Varient.java
public class ConcreteObserver_Varient implements IObserver {
        public void hook_varient(){
            System.out.println("hook_varient...");
  }}
```

Step-5.　撰寫 ConcreteSubject 類別。

```
// ConcreteSubject.java
public class ConcreteSubject extends Subject_Invarient{
}
```

Step-6.　撰寫 JMain 類別。

```
public class JMain {
```

```
  public static void main(String[] args) {
        ConcreteSubject cs = new ConcreteSubject();
        cs.attach(new ConcreteObserver_Varient());
        cs.template_method();
}}
```

<<說明>>

接著，可以繼續做更進一步的設計：將 ConcreteObserver_Varient 子類別定義於 JMain 類別之內，如下圖：

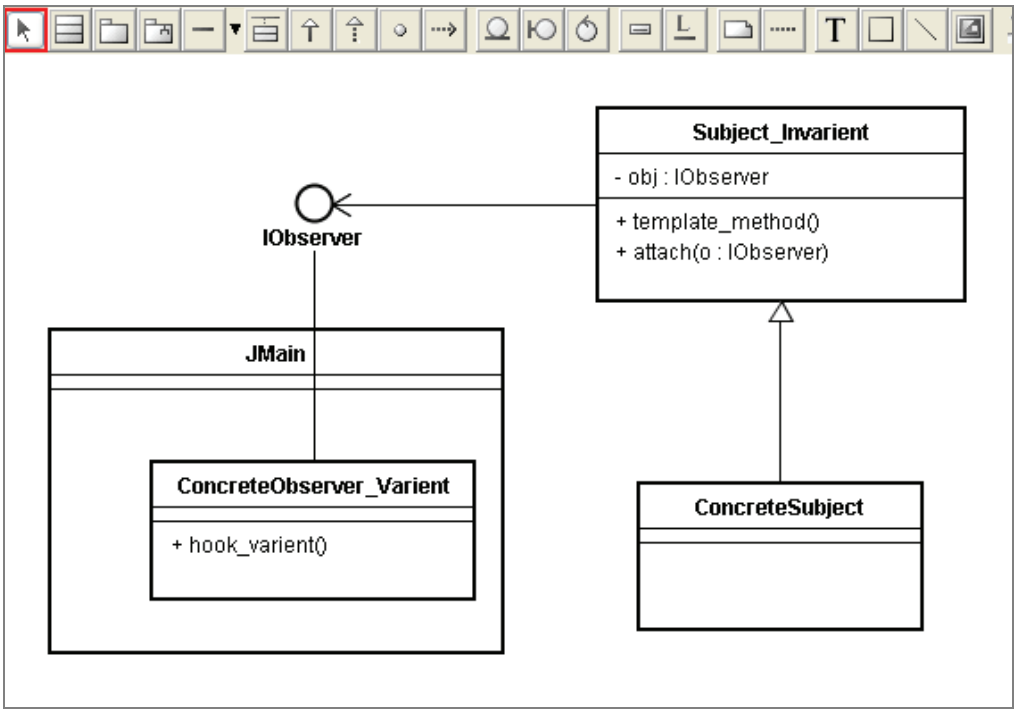


圖 5-7　更進一步的設計

當然，還可以視需要而想出更多的設計。例如，以 Client 父、子類別來替代上圖的 JMain 類別，如下圖：

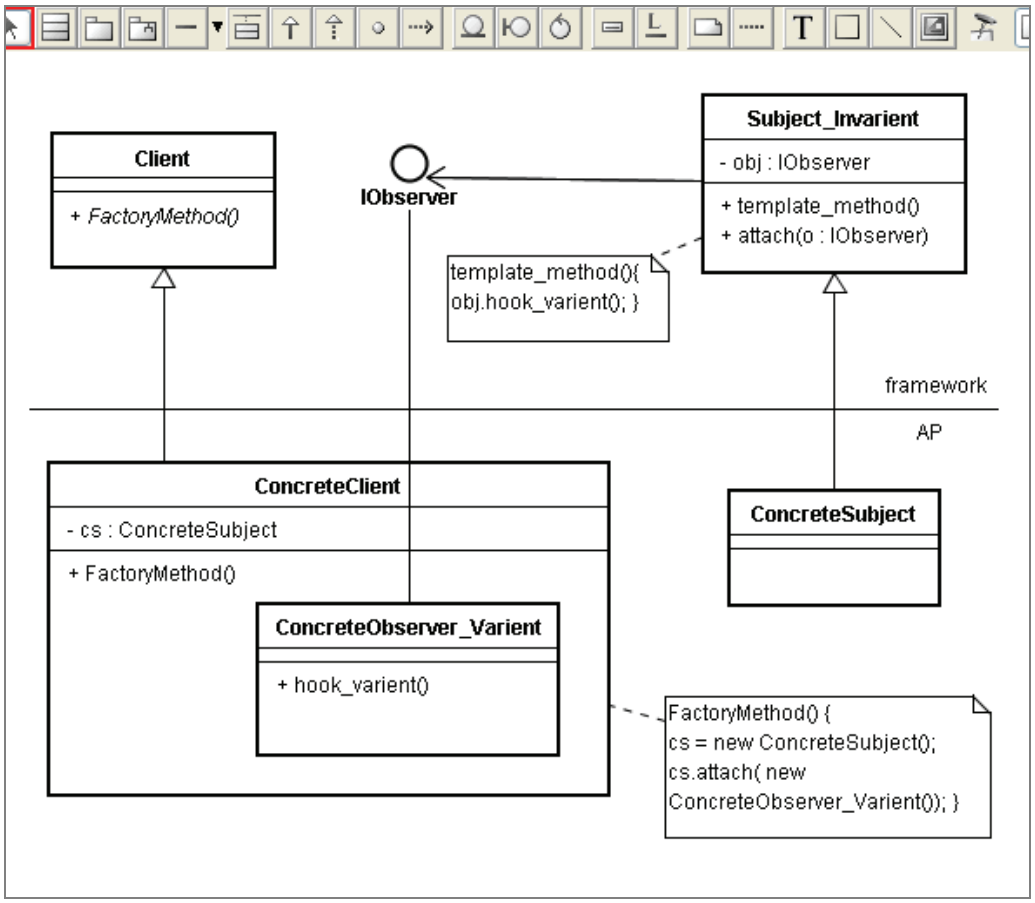
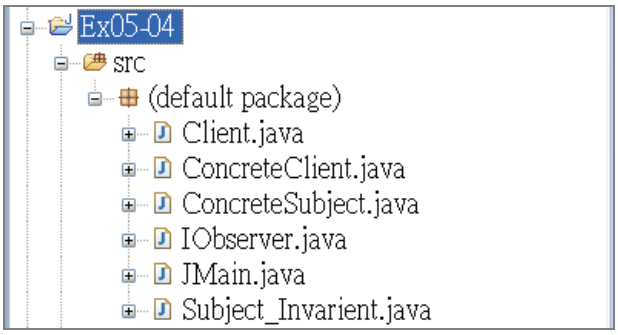


圖 5-8　設計出更多花樣

也許你已經發現到，上圖其實是 Observer 樣式與 Factory Method 樣式的一種優越組合。表面上看來很複雜，但是當你領悟到那只是上述兩種樣式的組合而已，就會覺得它是簡單有序的結構了，這就是設計樣式組合之美了。茲以 Java 程式來實現上圖 5-8，如下：

<<撰寫程式>>

Step-1. 建立一個 Java 應用程式專案：Ex05-04。



Step-2. 撰寫 Subject_Invarient 類別。

```
// Subject_Invarient.java
public class Subject_Invarient {
        private IObserver obj;
        public void template_method(){
              obj.hook_varient();
          }
     protected void attach(IObserver o){
             obj = o;
          }
}
```

Step-3. 定義 IObserver 介面。

```
// IObserver.java
public interface IObserver {
        void hook_varient();

}
```

Step-4. 定義 Client 介面。

```
// Client.java
public abstract class Client {
       public abstract void FactoryMethod();
}
```

Step-5. 撰寫 ConcreteClient 類別。

```
// ConcreteClient.java
public class ConcreteClient extends Client {
        public void FactoryMethod(){
```

```
            ConcreteSubject cs = new ConcreteSubject();
            cs.attach(new ConcreteObserver_Varient());
            cs.template_method();
    }
    private static class ConcreteObserver_Varient implements IObserver {
        public void hook_varient(){
            System.out.println("ConcreteClient::hook_varient....");
        }
}}
```

Step-5.　撰寫 ConcreteSubject 類別。

```
// ConcreteSubject.java
public class ConcreteSubject extends Subject_Invarient{
}
```

Step-4.　撰寫 JMain 類別。

```
public class JMain {
   public static void main(String[] args) {
        Client c = new ConcreteClient();
        c.FactoryMethod();
}}
```

<<說明>>

剛才提到，這是 Observer 與 Factory Method 兩種樣式的組合運用。這 ConcreteObserver_Varient 子類別通稱爲 Listener 類別。這在 Android 框架裡扮演非常亮麗的角色。再下一節裡，你將可欣賞到 Android 框架裡的 Lintener 類別及其運用了。

5.2 Android 框架與 Observer 樣式

剛才提到，Android 框架與 Listener 類別的搭配，就是 Observer 樣式的運用。它在 Android 框架裡扮演非常亮麗的角色。例如，下述的簡單 Android 程式，就可欣賞到 Observer 樣式面貌了。

<<撰寫程式>>

Step-1. 建立一個 Android 應用程式專案：Ex04-05。

Step-2. 撰寫 myActivity 類別。

```
// myActivity.java
package com.misoo.pkrr;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class myActivity extends Activity {
        @Override public void onCreate(Bundle icicle) {
                super.onCreate(icicle);
                setContentView(R.layout.main);
                Button button = new Button(this);
                button.setText("OK");    button.setBackgroundResource(R.drawable.heart);
                button.setOnClickListener(clickListener);
                setContentView(button);    }
        OnClickListener clickListener = new OnClickListener() {
            @Override public void onClick(View v) {
                        String name = ((Button)v).getText().toString();
                        setTitle( name + " button clicked");
    }};}
```

<<說明>>

表面上看來，這 ac01 應用類別是蠻簡單的，其實 Android 框架已經運用設計

樣式將複雜包裝起來，而呈現出簡潔有致的 ac01 應用類別了。茲以圖形來表達上述的小範例程式，如下：

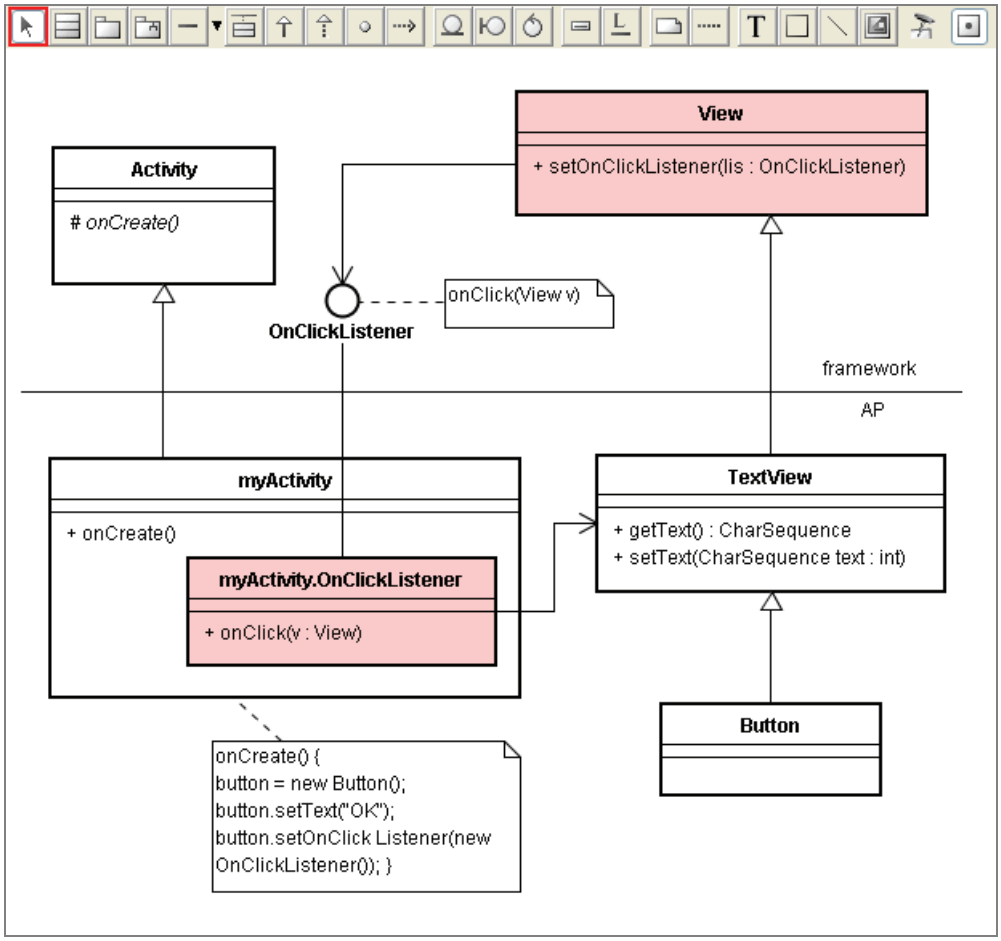


圖 5-9　複雜背後蘊藏著簡單樣式

只要你能懷著所介紹過的 Template Method、Factory Method 和 Observer 樣式來欣賞上圖，就能一眼看出其複雜表象的背後，其實蘊藏著簡單的樣式，隱約透露出白裡透紅之美。◆

有關本書作者消息更新 ….

※ 高煥堂 Android 書 2009 年新版(繁體)上市

#1. <<Google Android 應用框架原理與程式設計 36 技>> 2009 第四版
#2. <<Google Android 應用軟體架構設計>> 2009 第二版
#3. <<Google Android 與物件導向技術>> 2009 第二版
#4. <<Google Android 設計招式之美>> 2009 初版

※ 高煥堂於 2009/2/20-26

在上海開 Android 技術與產業策略講座

※ 高煥堂於 2009/4/10-16

在北京 & 上海開 Android 技術與產業策略講座

連絡 E-mail： misoo.tw@gmail.com 高煥堂 老師 收

歡迎光臨: www.android1.net 及其「Android 大舞台」版

2 月 14-15(週六&周日)

新春 台北 Android 特訓班 高煥堂 主講

主題：**軟硬整合技術與行動應用開發**(共 11 小時)

時間：2 月 14　10:00am – 5:00pm (6hr) &

　　2 月 15　1:00pm – 6:00pm(5hr)

大綱：這是 Android 技術的精華課程，內容完整，包括三個層面：

1. 軟硬整合技術
   - ARM & Linux 核心 & VM 介紹
   - 交叉編譯 Toolchains 介紹及使用
   - JAVA 與 C++程式庫連結和安裝
   - C 程式庫開發&與 Driver 的連結
   - Android 的移植問題與經驗分享
2. 框架設計招式
   - 認識 Android 的 Binder 架構設計
   - 熟悉 Android 的 Security 安全架構
   - 演練 JNI(即 Java 與 C++整合架構)
   - 介紹 Android 其他各種設計招式
3. 應用軟體開發
   - 複習 Java 語言:如類別繼承等等
   - 模擬器及 Eclipse 開發環境安裝
   - 將可執行程式安裝到 G1 手機裡
   - AP 開發:從商業流程到通訊架構
   - 資料庫技術與 SQLite 應用實務
   - Location-based 行動軟體開發實務
   - 其他 Android 行動軟體開發技術

地點：台北市，捷運後山埤站旁(步行約 4 分鐘)

費用：NT$ **9,500** 元

報名：misoo.tw@gmail.com 高煥堂老師收

詢問：(02)2739-8367 找高煥堂老師
講師：高煥堂
學歷： 美國科羅拉多大學 資管研究所
淡江大學 管理科學研究所
中興大學 法商學院
專長： 32 年資深電腦軟、硬體系統開發經驗
出版： 4 本有關 Android 技術書籍...
#1. <<Google Android 應用框架原理與程式設計 36 技>>
#2. <<Google Android 應用軟體架構設計>>
#3. <<Google Android 與物件導向技術>>
#4. <<Google Android 設計招式之美>>

--- END ---