第 3 章

Template Method 樣式

3.1 複習：「變與不變之分離」原則

變與不變的分離(Separate code that changes from the code that doesn't)是設計卡榫(Hook)函數及應用框架之基本原則和手藝。大文豪蘇東坡在其赤壁賦中寫道：「蓋將自其變者而觀之，則天地曾不能以一瞬；自其不變者而觀之，則物與我皆無盡也，而又何羨乎?」。其說明了，人們可兼具多種觀點，可同時看出同一個系統中的變與不變之相貌。大科學家 愛因斯坦 在其相對論裡也告訴我們：表面上看來相對的外貌下，可能蘊藏著不變的特性。例如，物質與能量從外貌看來是相對的(即變的)，但其背後蘊含著某種不變。

雖然蘇東坡和愛因斯坦所觀察的對象都是自然物，而不是像軟體、桌子、車子等人造物；但是在人們心靈深處，其心智的運用是一致的，當我們觀察人造物而能區分出變與不變的部份時，就能將之分離開來，而獲得優越之設計。例如，人們觀察車子的車體和輪胎兩者，自其變者而觀之，輪胎每年都需要換新；自其不變者而觀之，車體引擎十多年都仍然依舊。於是，汽車設計師就將輪胎與車體分離開來，因而出現「輪盤」來銜接輪胎與車體兩個分離的東西。這個輪盤就是兩者的接口處，扮演卡榫的角色。同樣地，在軟體系統上，藉由「變與不變的分離」之原則和手藝，能設計出優越的 Hook 函數，並支持各種設計樣式。

就Android框架而言，所謂「不變」的部份，就是它屬於各種應用程式間的共同部分，所以不隨著應用的改變而改變，因此稱之爲不變。並不意味著它在本質上是不變的。例如，『畫海鷗』與『畫蝴蝶』是兩個不同的應用程式，它們兩者都含有一段相同的『畫天空背景』程式片段。所以這個『畫天空背景』程式片段是兩者間之一致部分，就稱之爲兩個應用程式的「不變」部分。Android框架含有眾多這種不變(即不因個別『應用』程式而變)的部分，所以稱爲『應用框架』(Application Framework)。

3.2 複習：「變與不變之分離」手藝

分離出變(Variant)與不變(Invariant)部份之後，就可以將不變部份寫在父類別(Super class)裡，而變的部份就寫在子類別(Subclass)裡，然後藉由Java、C++等電腦語言的類別繼承(Class Inheritance)機制組織起來。現在，就讓我們從簡單的

Java範例程式談起吧!

<<撰寫程式>>

Step-1. 建立一個 Java 應用程式專案：Ex03-01。

Step-2. 撰寫 AA 類別。

```
// AA.java
public class AA {
   private String x;
      AA(String str){    x = str;    }
   public void print(){    System.out.println(x);    }
}
```

Step-3. 撰寫 BB 類別。

```
// BB.java
public class BB {
     private int x;
        BB(int k){    x = k;    }
        public void print(){    System.out.println(x);      }
}
```

Step-4. 撰寫 JMain 類別。

```
// JMain.java
public class JMain {
        public static void main(String[] args) {
        AA a = new AA("hello");
        a.print();
}}
```

現在就來練習變與不變的分離手藝了，記得，前面提過，這裡的變與不

變，並無關於本質(Essence)，只是基於我們的觀點(Point of View)而已。其步驟如下：

<<步驟 1：兼具變與不變兩個觀點>>

把 AA 和 BB 視爲兩個不同應用程式裡的類別，並且觀察其變與不變：

AA 類別

```
private String x;
public void print() {
    System.out.println(x);
}
.....
```

BB 類別

```
private int x;
public void print() {
    System.out.println(x);
}
.....
```

首先，從資料項目型態之差異，就能看出它們之間的變化點(Variant)：

String x; <- - - - - 相異 - - - - - -> int x;

隨之也能看出函數內容的「會變」之部分：

```
void print() {
   System.out.println( x );
}
```

```
void print() {
   System.out.println( x );
}
```

相異

此時，卡榫函數就派上用場了。

<<步驟 2：將會變部份寫入卡榫函數>>

將變與不變部份分離開來，然後將會變部份寫入卡榫函數裡，如下述的 hook_getData()函數：

AA 類別

```
private String x;
public final void template_print() {
     System.out.println( hook_getData() );
}
public String hook_getData() {
   return x;
}
.....
```

BB 類別

```
private int x;
public final template_print() {
     System.out.println( hook_getData() );
}
public String hook_getData() {
     return String.valueOf(x);
}
.....
```

分離之後，template_print()函數含有不變的部份；而 hook_getData()含有會變的部份。

<<步驟 3：將不變部份寫入抽象類別>>

接著，搭配父、子類別的繼承關係，將不變部分的 template_print()函數移入新定義的 SuperAB 父類別裡，並且定義抽象的 hook_getData()卡榫函數，如下圖 3-1 所示。如此就設計出卡榫函數了。

在 SuperAB 裡的 template_print()函數呼叫到抽象的 hook_getData()函數，此時藉由繼承機制而反向呼叫到子類別的 hook_getData()函數，實現了父、子類別之溝通與合作了。

當我們將之對應到 Android 框架時，因爲 AF 的用意就是要吸收不變的部份，所以像 SuperAB 這樣的抽象父類別將會被納入框架裡，而像 AA 這樣的子類別將會被納入應用程式裡。

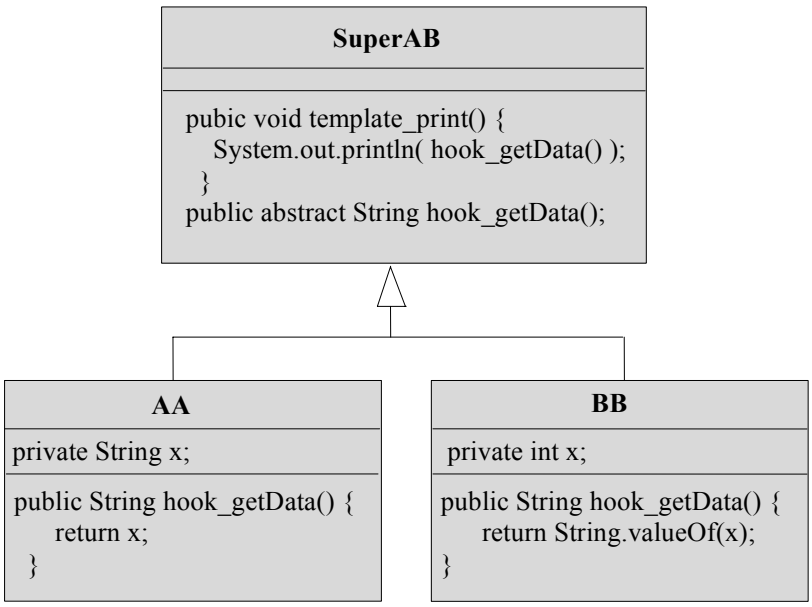


圖 3-1　抽象：抽出變或不變部分

於是，這父、子類別之溝通也就是框架與應用程式之溝通了。則上述的 hook_getData()就扮演著框架與應用程式之間的卡榫角色了。於是，可撰寫程式來實現上圖。

<<撰寫程式>>

Step-1. 建立一個 Java 應用程式專案：Ex03-02。

Step-2.　撰寫 SuperAB 類別。

```
// SuperAB.java
public abstract class SuperAB {
        public void template_print(){
                System.out.println(hook_getData());
         }
        protected abstract String hook_getData();
}
```

Step-3.　撰寫 AA 類別。

// AA.java

```
public class AA extends SuperAB{
    private String x;
    AA(String str){    x = str;    }

    @Override protected String hook_getData() {
        return x;
}}
```

Step-4.　撰寫 BB 類別。

// BB.java

```
public class BB extends SuperAB{
        private int x;
        BB(int k){    x = k;    }

        @Override protected String hook_getData() {
            return String.valueOf(x);
}}
```

Step-5.　撰寫 JMain 類別。

// JMain.java

```
public class JMain {
        public static void main(String[] args) {
        AA a = new AA("hello");
        a.template_print();
}}
```

此時，你可以將 SuperAB 納入 AF 裡，於是 JMain 類別裡的指令：

```
a.template_print();
```

呼叫了 AF 裡的 template_print()函數。其內之指令：

```
public void template_print(){
    System.out.println(hook_getData());
}
```

就呼叫到 AF(的 SuperAB 類別)裡的 hook_getData()函數，進而反向呼叫到 AP(的 AA 子類別)裡的 hook_getData()函數。

3.3 複習：框架的反向控制

在上一章(第 2 章)裡介紹過，反向控制(Inversion of Control)是應用框架魅力的泉源。其常見的實現機制有二：

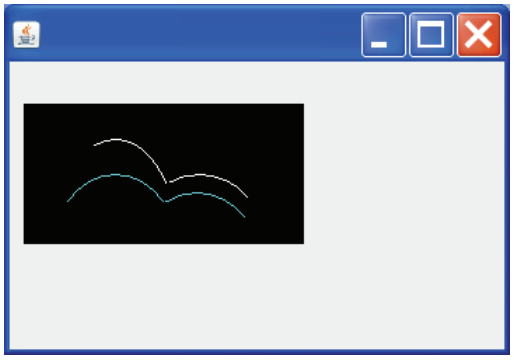
1) 繼承(Inheritance) + 卡榫函數
   ---- 在 Template Method 樣式裡，可看到其典型的用法。
2) 委託(Delegation) + 卡榫函數
   ---- 在 Observer 樣式裡，可看到其典型的用法。

Android 是個完整的應用框架，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。在本章裡，將先介紹「繼承 + 卡榫函數」實現機制的概念和基本用法，並且介紹 Template Method 樣式，藉由樣式的專業手藝來讓你深刻體會這些機制的精緻用法。如此，除了更能活用 Android 之外，也能逐漸提升你自己設計新應用框架的信心和能力。至於「委託 + 卡榫函數」實現機制則留待第 5 章介紹 Observer 樣式時，再詳加說明了。

在上一節(第 3.1 節)裡，介紹過「變與不變之分離」的原則和方法。在本節裡，將稍做一些複習，然後搭配 Template Method 樣式，以樣式所含的專家技藝來提升我們的設計品質。首先，從「繼承(Inheritance) + Hook 函數」的實現機制談起，如下述之範例 Ex03-03。

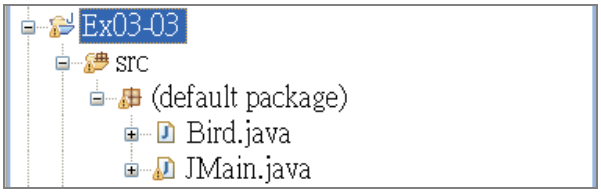
<<操作情境>>

此範例程式執行時，繪畫出兩隻海鷗，如下圖：



<<撰寫程式>>

Step-1. 建立一個 Java 應用程式專案：Ex03-03。



Step-2.　撰寫 Bird 類別。

```
// Bird.java
import java.awt.*;
public class Bird {
        public void paint(Graphics gr) {
                // 畫背景指令
                gr.setColor(Color.black);       gr.fillRect(10,30, 200,100);
                // 畫圖(海鷗)指令
                gr.setColor(Color.cyan);
                gr.drawArc(30,80,90,110,40,100);    gr.drawArc(88,93,90,100,40,80);
                gr.setColor(Color.white);
                gr.drawArc(30,55,90,150,35,75);     gr.drawArc(90,80,90,90,40,80);
        }}
```

Step-4. 撰寫 JMain 類別。

```
// JMain.java
import java.awt.*;
import javax.swing.*;
class JP extends JPanel {
        public void paintComponent(Graphics graph){
                super.paintComponents(graph);
                Bird bird = new Bird();
                bird.paint(graph);
        }}
public class JMain extends JFrame {
    public JMain(){    setTitle("");    setSize(350, 250);    }
    public static void main(String[] args) {
            JMain frm = new JMain();    JP panel = new JP();
            frm.add(panel);
            frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frm.setVisible(true);
```

```
}}
```

<<說明>>

由於框架的核心設計原則就是：「變與不變的分離(Separate code that changes from the code that doesn't)。針對Bird類別的paint()函數之內容，把不會隨著應用程式之不同而改變的部分(即Invariant)移到新的Shape父類別裡。目前這是一個畫海鷗的AP，我們還可以撰寫畫河馬、貓熊、蝴蝶等不同的AP，那麼，那些是會隨著AP不同而變化的部份呢? 而那些又是不變的部份呢? 於此，可發現其不變部份爲：

```
public void paint(Graphics gr){
        // 畫天空背景
        gr.setColor(Color.black);
        gr.fillRect(10,30, 200,100);
    }
```

是無論畫海鷗或畫蝴蝶之應用，這個天空背景的指令都是相同的，具有這樣特性的部份，就稱爲不變的部份，就將它擺在父類別(例如 Shape 類別)裡。如下圖：

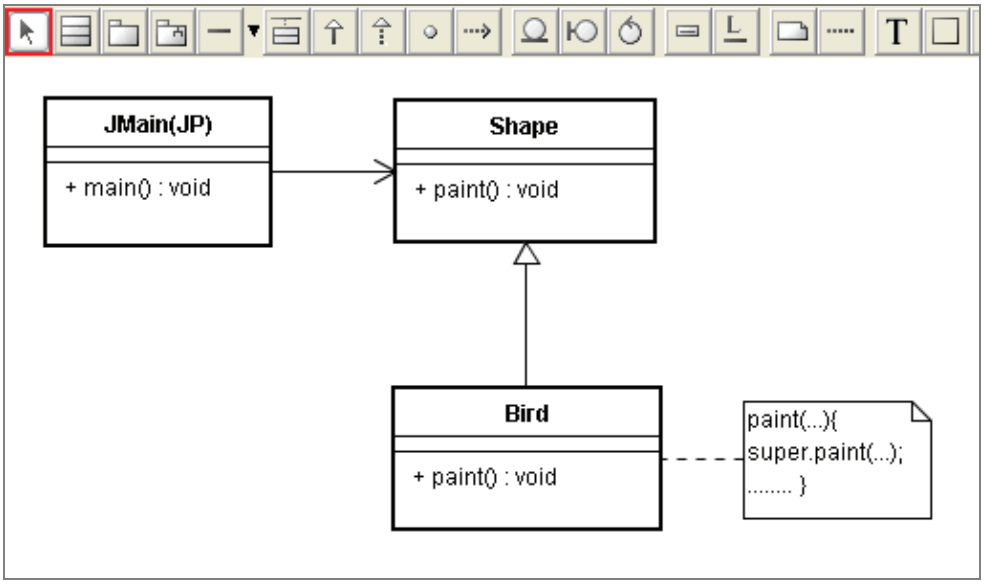


圖3-2　變與不變分離了

於是，AF 就含有眾多這種不變(即不因個別「應用」程式而變)的父類別部分，所以稱爲「應用框架」。至於 paint()函數的其他內涵：

```
public void paint(Graphics gr){
        // 畫圖(海鷗)指令
        .....
    }
```

其內部指令只用來畫海鷗。如果想改畫蝴蝶時，這些指令都必須重新改寫。這種會隨著個別應用(例如畫海鷗與畫蝴蝶是不同的應用)而不同的部份，就通稱爲「會變」或「善變」(Variant)部分，就將它擺在子類別(例如 Bird 類別)裡。於是，上述範例程式可改寫爲：

<<撰寫程式>>

Step-1. 建立一個 Java 應用程式專案：Ex03-04。

Step-2.　撰寫 Shape 類別。

```
// Shape.java
import java.awt.*;
public abstract class Shape{
        public void paint(Graphics gr){
        // 畫天空背景
          gr.setColor(Color.black);      gr.fillRect(10,30, 200,100);
        }
   }
```

Step-3.　撰寫 Bird 類別。

```
// Bird.java
import java.awt.*;
public class Bird extends Shape {
```

```
    @Override public void paint(Graphics gr){
        super.paint(gr);
        // 畫圖(海鷗)指令
        gr.setColor(Color.cyan);
        gr.drawArc(30,80,90,110,40,100);     gr.drawArc(88,93,90,100,40,80);
        gr.setColor(Color.white);
        gr.drawArc(30,55,90,150,35,75);      gr.drawArc(90,80,90,90,40,80);
  }}
```

Step-4. 撰寫 JMain 類別。

```
// JMain.java
import java.awt.*;
import javax.swing.*;
class JP extends JPanel {
      public void paintComponent(Graphics gr){
            super.paintComponents(gr);
            Shape sp = new Bird();
            sp.paint(gr);
      }}
public class JMain extends JFrame {
   public JMain(){   setTitle("");   setSize(350, 250);   }
   public static void main(String[] args) {
      JMain frm = new JMain();     JP panel = new JP();
      frm.add(panel);
      frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frm.setVisible(true);
}}
```

<<說明>>

至此，變與不變的部份已經分離了，也分別納入父、子類別裡，並且定義了一個名叫 paint()的 Hook 函數，來銜接這兩個類別。其銜接方法是，由子類別的 paint()函數呼叫父類別的 paint()函數先去執行不變部份，然後才執行子類別內的會變部份。這是一種常見的銜接方法。不過，還有其他的銜接方法，例如下一小節裡所要介紹的 Template Method 樣式，將呈現另一種銜接方法。

3.4 介紹 Template Method 樣式

在上一節的圖 3-2 裡，其主角是 Shape 抽象類別，它提供了兩個介面：

1) 與 JMain(或 JP)類別的溝通介面：

這是對類別體系之外的類別的服務介面。目前此介面含有 paint()函數給外界使用。於是，JP 類別裡的指令：

```
Shape sp = new Bird();
sp.paint(gr);
```

就使用了這個對外的介面。

2) 與子類別的溝通介面：

這是提供給子類別進行客製化的介面。目前此介面含有一個 paint()函數來與子類別銜接。例如，Bird 子類別的指令：

```
public class Bird extends Shape {
   @Override public void paint(Graphics gr){
         ……
    }
}
```

就使用了 Shape 所提供的繼承介面來進行客製化動作。其中，你可發現：上述的對內和對外的兩個介面都內涵同一個 paint()函數，則這兩個介面之間會產生高度的相依性(Dependency)，它會降低應用框架的彈性，所以 Template Method 樣式就可用來化解這樣的問題，以便降低相依性，增加應用框架的彈性。於是應用了 Template Method 樣式來改善上述 Ex03-04 範例，修改爲如下之範例程式 Ex03-05：

<<撰寫程式>>

Step-1. 建立一個 Java 應用程式專案：Ex03-05。

Step-2. 撰寫 Shape 類別。

```
// Shape.java
import java.awt.*;
public abstract class Shape{
        public void template_paint(Graphics gr){
                // 畫天空背景
                gr.setColor(Color.black);      gr.fillRect(10,30, 200,100);
                // 畫前景
                hook_paint(gr);
       }
     protected abstract void hook_paint(Graphics gr);
}
```

Step-3. 撰寫 Bird 類別。

```
// Bird.java
import java.awt.*;
public class Bird extends Shape {
        @Override
        public void hook_paint(Graphics gr){
            // 畫圖(海鷗)指令
            gr.setColor(Color.cyan);
            gr.drawArc(30,80,90,110,40,100);      gr.drawArc(88,93,90,100,40,80);
            gr.setColor(Color.white);
            gr.drawArc(30,55,90,150,35,75);       gr.drawArc(90,80,90,90,40,80);
   }}
```

Step-4. 撰寫 JMain 類別。

```
// JMain.java
import java.awt.*;
import javax.swing.*;
class JP extends JPanel {
        public void paintComponent(Graphics gr){
                super.paintComponents(gr);
                Shape sp = new Bird();
```

```
            sp.template_paint(gr);
        }}
public class JMain extends JFrame {
        public JMain(){    setTitle("");    setSize(350, 250);    }
        public static void main(String[] args) {
        JMain frm = new JMain();      JP panel = new JP();
        frm.add(panel);
        frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frm.setVisible(true);
}}
```

於是，對內與對外的介面分開了。Shape 類別提供 template_paint()函數給外(類別體系之外)界使用。例如，JP 類別裡的指令：

```
Shape sp = new Bird();
sp.template_paint(gr);
```

就呼叫了 template_paint()介面函數。而其 hook_paint()函數則留給子孫類別使用。如下圖：

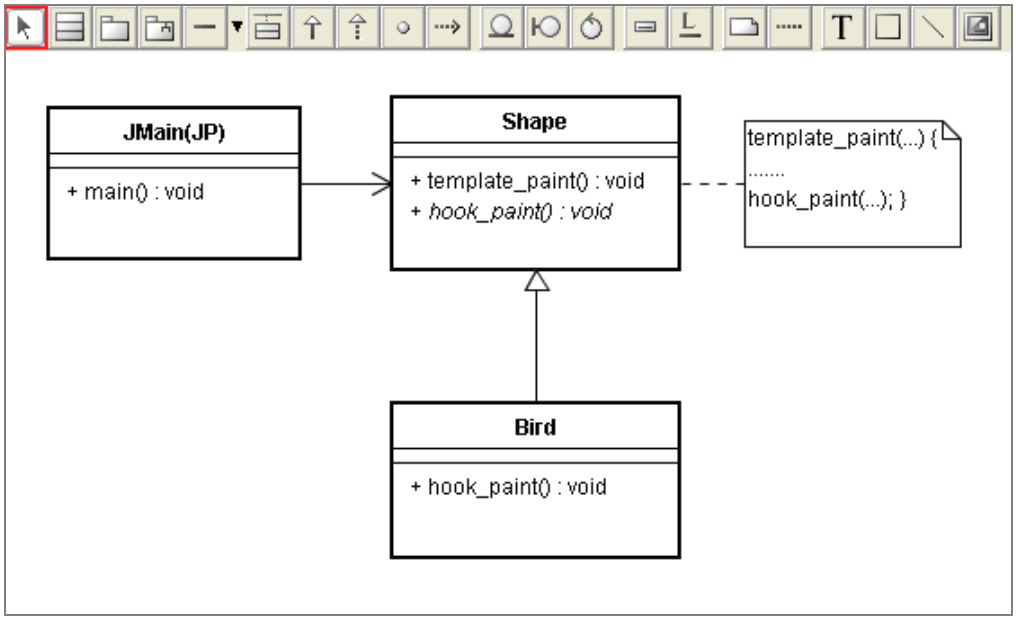


圖 3-3　抽象類別與其子類別的介面

其實，這已經運用了大家熟知的 Template Method 樣式了。在 GoF 的<<Design Patterns>>一書裡，就介紹了這個常用的樣式，如下圖：

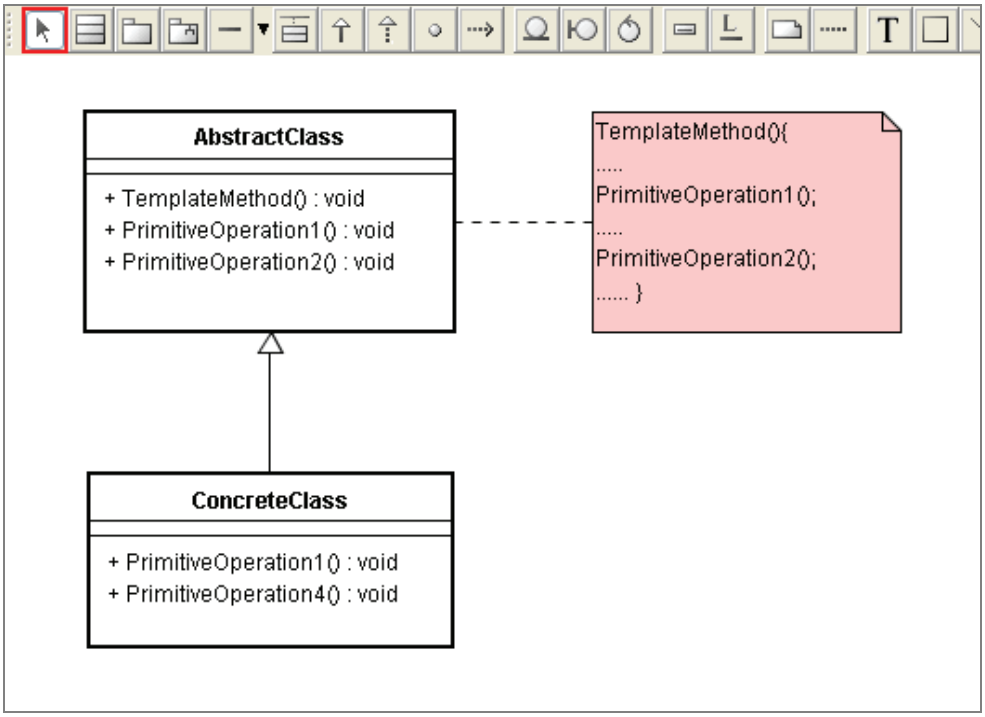


圖 3-4　GoF 的 Template Method 樣式[GoF]

圖 3-3 裡的 template_paint()函數就是 Template Method 樣式(即圖 3-4)裡的 TemplateMethod()。而圖 3-3 裡的 hook_paint()函數則是 Template Method 樣式(圖 3-4)裡的 PrimotiveOperation()函數。

由於樣式是專家們從過去經驗中焠鍊出來的，用來引導人們的思維，可促進人們「依樣畫葫蘆」，進而「舉一反三」以便能更有效化解目前或未來所面臨的問題。例如，我們能「依樣畫葫蘆」並加以修正，發揮 Java 的介面定義機制，如下圖：

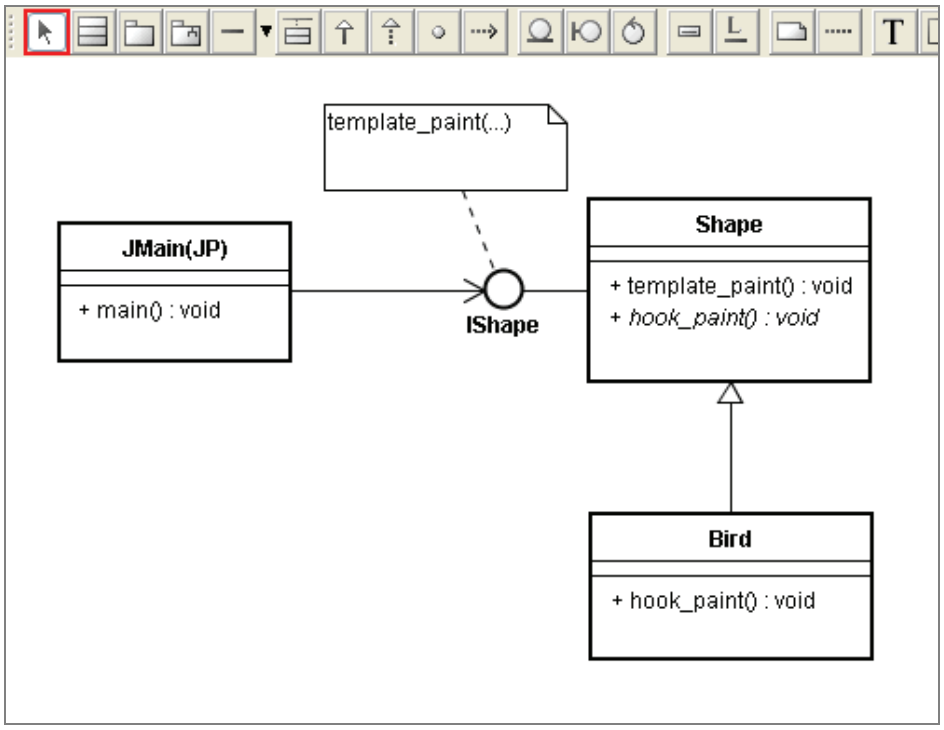
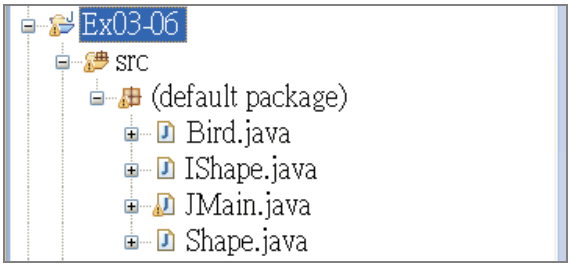


圖 3-5　抽象類別與 Cleint 的介面

於是，可以依據此圖而將上述的 Ex03-05 範例改寫如下：

<<撰寫程式>>

Step-1. 建立一個 Java 應用程式專案：Ex03-06。



Step-2.　定義 IShape 介面。

```
// IShape.java
import java.awt.Graphics;
interface IShape {
        void template_paint(Graphics gr);
}
```

Step-3. 撰寫 Shape 類別。

```
// Shape.java
import java.awt.*;
public abstract class Shape implements IShape {
        public void template_paint(Graphics gr){
                // 畫背景
                invariant_paint(gr);
                // 畫前景
                hook_paint(gr);
        }
        private void invariant_paint(Graphics gr){
                // 畫天空背景
                gr.setColor(Color.black);
                gr.fillRect(10,30, 200,100);
                // 畫前景
        }
    protected void hook_paint(Graphics gr){}
}
```

Step-3. 撰寫 Bird 類別。

```
// Bird.java
import java.awt.*;
public class Bird extends Shape {
        @Override public void hook_paint(Graphics gr){
            // 畫圖(海鷗)指令
            gr.setColor(Color.cyan);
            gr.drawArc(30,80,90,110,40,100);      gr.drawArc(88,93,90,100,40,80);
            gr.setColor(Color.white);
            gr.drawArc(30,55,90,150,35,75);       gr.drawArc(90,80,90,90,40,80);
  }}
```

Step-4. 撰寫 JMain 類別。

```
// JMain.java
import java.awt.*;
import javax.swing.*;
```

```
class JP extends JPanel {
    public void paintComponent(Graphics gr){
            super.paintComponents(gr);
            IShape isp = new Bird();
            isp.template_paint(gr);
      }}
public class JMain extends JFrame {
    public JMain(){    setTitle("");    setSize(350, 250);    }
    public static void main(String[] args) {
            JMain frm = new JMain();
            JP panel = new JP();
            frm.add(panel);
            frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frm.setVisible(true);
}}
```

<<說明>>

在本章裡，我們運用「變與不變分離」手藝來設計出 Shape 與 Bird 父、子類別，再參考 Template Method 樣式，而得到更優越的設計，如圖 3-5 的 IShape 介面和 Shape 抽象類別。由於這 IShape 介面和 Shape 抽象類別都是「不變」的部份，就能納入應用框架裡，成爲 AF 的內容。之後，想畫河馬或蝴蝶的人，就能重用 (Reuse)應用框架裡的 IShape 介面和 Shape 抽象類別，而加速應用程式的開發了。

3.5 Android 的 Template Method 樣式

在 Android 裡，處處可見 Template Method 樣式之應用。然而，其 Template Method 大多深藏於上層的父類別裡，在 Android 的應用程式只能看到子類別的卡榫函數而已。也就是說，在 Android 的應用程式裡，通常看不到 Template Method 樣式的頭：Template Method；而只能看到它的尾巴：卡榫函數。

3.5.1 Template Method 樣式範例之一

在 Android 的 View 類別體系裡，有個多形(Polymorphic)的 onDraw()函數，它是一個卡榫函數，也就是 Template Method 樣式的尾巴。此樣式的頭(即 template 函數)是定義於父類別 View 裡的 draw()函數。由於 Template Method 樣式總是見尾不見首，所以必須去察看 View 類別的原始碼才能看到 draw()函數。茲寫個範例程式(Ex03-07)來展示這個 onDraw()卡榫函數。

<<操作情境>>

此程式執行時，畫面上顯示出一個 2D 繪圖區，如下圖：

<<撰寫程式>>

Step-1. 建立一個 Android 程式專案：Ex03-07。

Step-2.　撰寫 View 的子類別：myView 類別。

```
// myView.java
package com.misoo.pkcc;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.view.View;

public class myView extends View{
        private Paint paint= new Paint();

        myView(Context ctx) {    super(ctx);    }
        @Override protected void onDraw(Canvas canvas) {
                int line_x = 10;
                int line_y = 50;
                canvas.drawColor(Color.WHITE);
                paint.setColor(Color.GRAY);
                paint.setStrokeWidth(3);
                canvas.drawLine(line_x, line_y, line_x+120, line_y, paint);
                paint.setColor(Color.BLACK);
                paint.setStrokeWidth(2);
                canvas.drawText("這是GraphicView繪圖區", line_x, line_y + 50, paint);
                int pos = 70;
                paint.setColor(Color.RED);
                canvas.drawRect(pos-5, line_y - 5, pos+5, line_y + 5, paint);
                paint.setColor(Color.YELLOW);
                canvas.drawRect(pos-3, line_y - 3, pos+3, line_y + 3, paint);
}}
```

Step-3.　撰寫 Activity 的子類別：myActivity。

```
// myActivity.java
package com.misoo.pkcc;
import android.app.Activity;
import android.os.Bundle;

public class myActivity extends Activity {
        private myView gv = null;

        @Override public void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              gv = new myView(this);
              setContentView(gv);
}}
```

<<說明>>

當程式執行時，框架反向呼叫到 myActivity 類別的 onCreate()函數，進而執行到其內部的指令：

```
setContentView(gv);
```

欲顯示出 gv 的畫圖區時，框架就呼叫 View 類別的 draw()函數(即 template 函數)。接著，此函數呼叫 onDraw()卡榫函數，就在繪圖區畫出圖形了。

3.5.2 Template Method 樣式範例之二

Template Method 樣式好像一隻烏龜，有些看不到牠的頭，例如上一小節的 Ex03-07 範例程式裡，烏龜的頭(即 template 函數)隱藏於 View 父類別裡，在應用程式碼裡，看不到它。

然而，有些 Template Method 樣式烏龜會伸出頭來，就與前面的圖 3-5 一樣，其父類別會提供一個介面，就像烏龜伸出頭來。此時就能看到 Template Method 樣式的頭和尾巴了。現在，茲舉 Android 的 Binder 父類別爲例，讓你來欣賞他幕後的那隻 Template Method 樣式烏龜，看看牠的長相。在 Android 的 Binder 父類別裡就定義了一個 transact()函數(即 template 函數)，而且它會呼叫 onTransact()卡榫函數，如下圖：

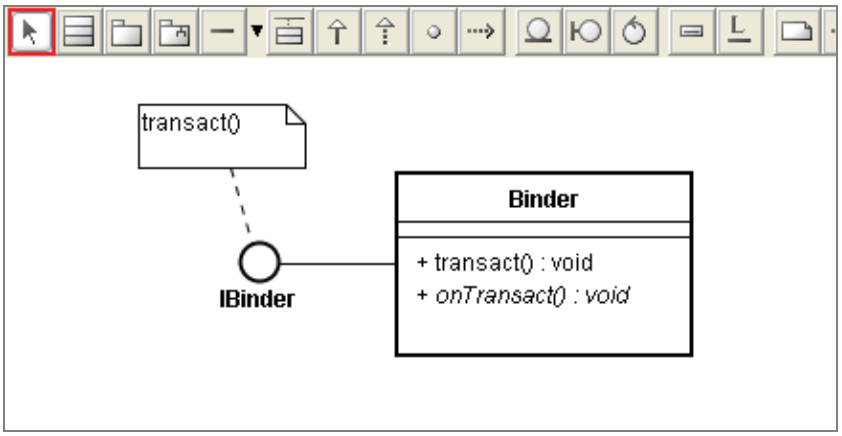


圖 3-6　Android 裡的 Binder 抽象類別

基於上圖裡的Binder抽象類別，就能撰寫mp3PlayerBinder應用類別，並藉由卡榫函數來銜接Binder父類別，如下圖：

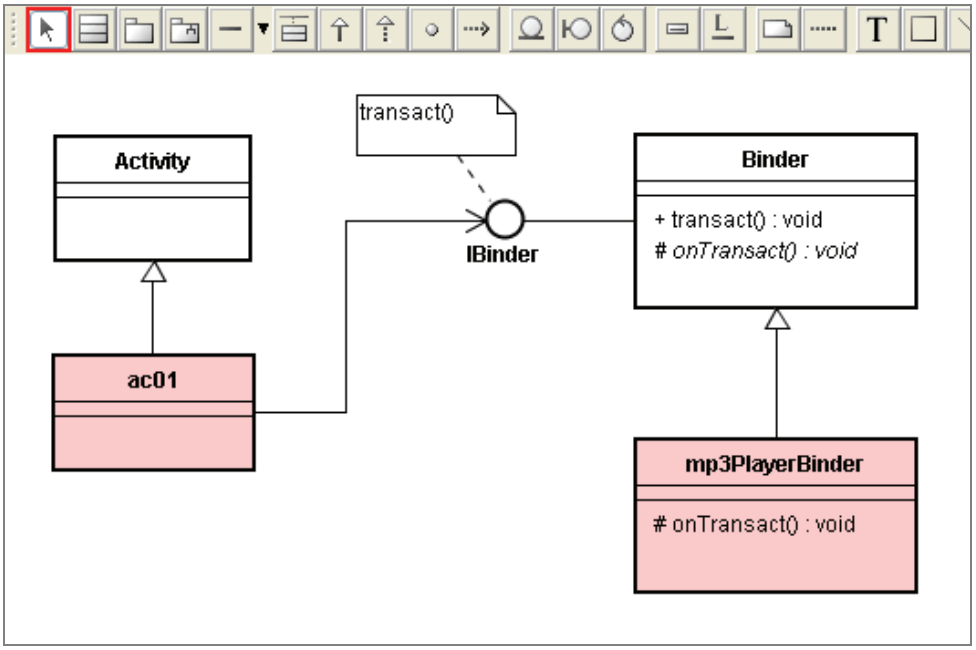


圖 3-7　Android 裡典型的 Template Method 樣式

從上圖範例，你可以欣賞到Template Method樣式的實用之美了。基於這個樣式，可以進一步開發出更複雜的應用類別。雖然表面上是複雜的，但是其中的Template Method樣式之美，會讓我們覺得該應用程式其實是簡單有序的，而不是繁雜無章的。其展現了設計樣式之美，也創造了Android框架之美。例如，可以撰寫一個看來複雜的 (其實是簡而美的)範例程式，如下圖：

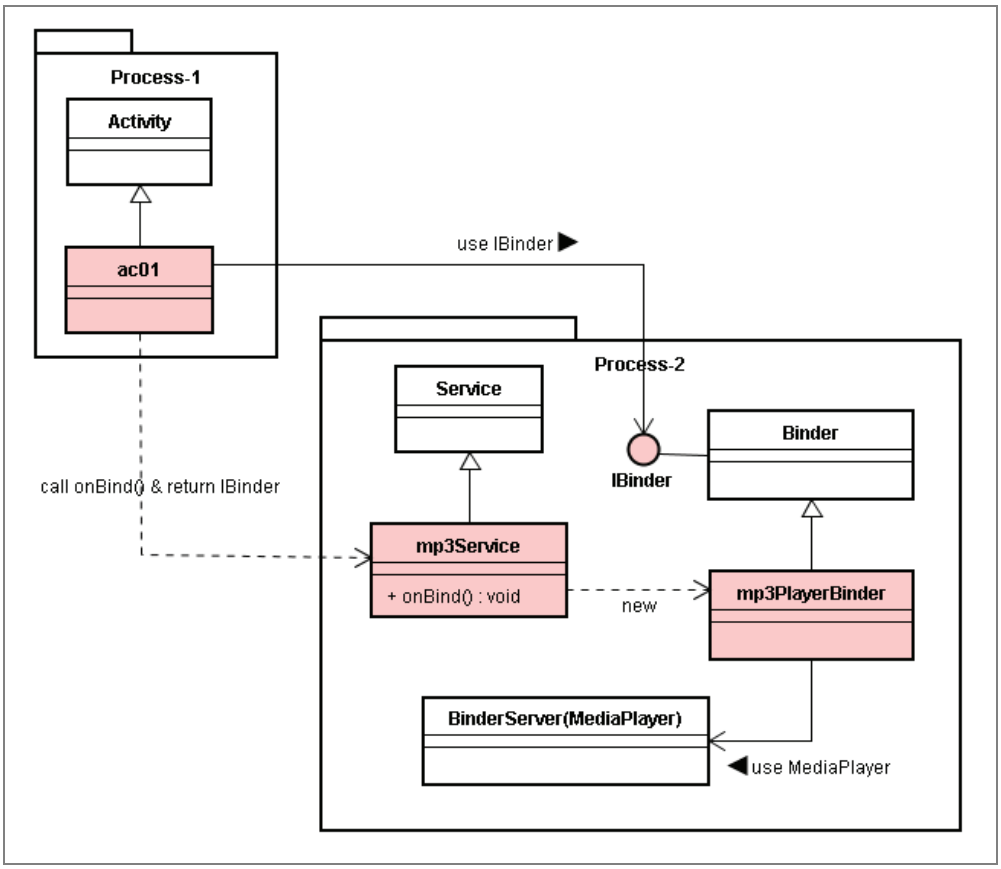
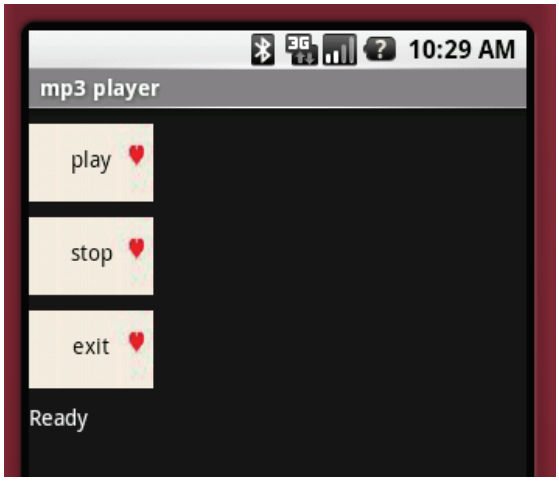


圖 3-8　複雜中有樣式

樣式本身並無美與醜，但是它常常會提升人們接受複雜的能力，因而讓人們不害怕複雜，這就是樣式之美的由來。現在，茲撰寫個 Android 程式來實現上圖，如下：

<<操作情境>>

此 AP 執行時，畫面上顯示出 mp3 播放的畫面：



<<撰寫程式>>

Step-1: 建立 Android 專案：Ex03-08。

Step-2: 撰寫 Activity 的子類別：ac01，其程式碼如下：

```
// ac01.java
package com.misoo.pkzz;
import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.graphics.Color;
```

```
import android.os.Bundle;
import android.os.IBinder;
import android.os.RemoteException;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.LinearLayout;
import android.widget.TextView;

public class ac01 extends Activity implements OnClickListener {
    private final int WC = LinearLayout.LayoutParams.WRAP_CONTENT;
    private final int FP = LinearLayout.LayoutParams.FILL_PARENT;
    private static ac01 appRef = null;
    private myButton btn, btn2, btn3;
    public TextView tv;
    private IBinder ib;

    public static ac01 getApp()
            {   return appRef;   }
    public void btEvent(String data)
            {  setTitle(data);     }
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        appRef = this;

        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        btn = new myButton(this);    btn.setId(101);
        btn.setText("play");         btn.setOnClickListener(this);
        LinearLayout.LayoutParams param =
            new LinearLayout.LayoutParams(btn.get_width(), btn.get_height());
        param.topMargin = 10;
        layout.addView(btn, param);

        btn2 = new myButton(this);  btn2.setId(102);
        btn2.setText("stop");       btn2.setOnClickListener(this);
        layout.addView(btn2, param);
        btn3 = new myButton(this);  btn3.setId(103);
        btn3.setText("exit");       btn3.setOnClickListener(this);
        layout.addView(btn3, param);

        tv = new TextView(this);     tv.setTextColor(Color.WHITE);
        tv.setText("Ready");
        LinearLayout.LayoutParams param2
            = new LinearLayout.LayoutParams(FP, WC);
        param2.topMargin = 10;
        layout.addView(tv, param2);
```

```
            setContentView(layout);
            bindService(new Intent(ac01.this,
                mp3Service.class), mConnection, Context.BIND_AUTO_CREATE);
    }

    private ServiceConnection mConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className, IBinder ibinder)
            {   ib = ibinder;   }
        public void onServiceDisconnected(ComponentName className) { }
    };
    public void onClick(View v) {
        switch (v.getId()) {
        case 101:
            tv.setText("Playing audio...");       setTitle("MP3 Music");
            try { ib.transact(1, null, null, 0);
            } catch (RemoteException e) {   e.printStackTrace();        }
            break;
        case 102:
            tv.setText("Stop");
            try { ib.transact(2, null, null, 0);
            } catch (RemoteException e) {    e.printStackTrace();   }
            break;
        case 103:   finish();   break;
        }
  }}
```

Step-3: 撰寫 Service 的子類別：mp3Service，其程式碼如下：

```
// mp3Service.java
package com.misoo.pkzz;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class mp3Service extends Service {
    private IBinder mBinder = null;
    @Override public void onCreate() {
        mBinder = new mp3PlayerBinder(getApplicationContext());
    }
    @Override public IBinder onBind(Intent intent) {    return mBinder;   }
}
```

Step-4: 撰寫 Binder 的子類別：mp3PlayerBinder，其程式碼如下：

// mp3PlayerBinder.java

```
package com.misoo.pkzz;
import android.content.Context;
import android.media.MediaPlayer;
import android.os.Binder;
import android.os.Parcel;
import android.util.Log;

public class mp3PlayerBinder extends Binder{
        private MediaPlayer mPlayer = null;
        private Context ctx;

        public mp3PlayerBinder(Context cx){     ctx = cx;     }
        @Override
        public boolean onTransact(int code, Parcel data, Parcel reply, int flags)
                        throws android.os.RemoteException    {
                if(code == 1)    this.play();
                else if(code == 2)    this.stop();
                return true;    }
        public void play(){
                if(mPlayer != null) return;
                mPlayer = MediaPlayer.create(ctx, R.raw.test_cbr);
                try {    mPlayer.start();
                } catch (Exception e) {
                        Log.e("StartPlay", "error: " + e.getMessage(), e);
        }}
        public void stop(){
                if (mPlayer != null) {
                        mPlayer.stop();    mPlayer.release();    mPlayer = null;    }
   }}
```

Step-5: 撰寫 Button 的子類別：myButton，其程式碼如下：

// myButton.java

```
package com.misoo.pkzz;
import android.content.Context;
import android.widget.Button;

public class myButton extends Button {
        public myButton(Context ctx) {
```

```
            super(ctx);
            super.setBackgroundResource(R.drawable.heart);
        }
        public int get_width() {    return 80;      }
        public int get_height() {    return 50;     }
}
```

<<說明>>

表面上看來，這程式是蠻複雜的，但是只要看看 ac01 裡的 onClick()函數，其指令：

```
 public void onClick(View v) {
        switch (v.getId()) {
        case 101:
                ……
                ib.transact(1, null, null, 0);
                ……
        case 102:
                ……
                ib.transact(2, null, null, 0);
                ……
 }}
```

所呼叫的 transact()函數就是定義於 Template Method 樣式裡的 template 函數。這 transact()函數反向呼叫 mp3PlayerBinder 子類別的 onTransact()函數，如下：

```
public class mp3PlayerBinder extends Binder{
        ……
      @Override public boolean onTransact(int code, Parcel data, Parcel reply,
                int flags) throws android.os.RemoteException {
            if(code == 1)    this.play();
            else if(code == 2)    this.stop();
            return true;
} }
```

在 Template Method 樣式的指引下，我們對各條指令的角色都清晰有致，因而會覺得上述程式是簡單有序的，這是 Template Method 樣式的完美示範。◆

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