PREFACE

序言

讀者們好久不見！六年了才將第二版更新到第三版，真不好意思。

這幾年確實花了不少時間在物聯網（IoT）以及數位版權管理（DRM）影音串流的領域鑽研，相關的心得也深入淺出地整理一些到著作中。

智慧型手機到如今已經超過 10 個年頭，其上的 App 百花齊放、百家爭鳴，其中的通訊應用，舉凡社群軟體、線上串流、物聯網、各種電子商務軟體等等，隨著網路技術的升級再升級，在可見的將來，沒有最多，只有更多！

作者選擇手機 App 作為多年耕耘的教學場域，除了看準手機是拉近城鄉差距的極佳學習載具以外，也確信這個低門檻、高 CP 值的 3C 產品會是引起學習程式動機不錯的進入點。當然，其中又以 Android 為甚！

這次將「第一次學 Android 就上手」第三版濃縮成 16 章、共四篇的作法，就是希望保留最最重要的學習元素給讀者，讓讀者集中最少的精神來掌握最大的學習效益。

學習的次序應該是要先能掌握前面兩篇，然後無論是接著學習第三篇，還是先進入到第四篇學習官方版型，則由讀者自行選擇，並沒有一定的強制。

如果要我對這一版書提出看法，我會說第二章的基本動作歸納，以及第 13~14 章的官方版型介紹，應該算是本書的兩大特色，而這兩大特色也深深呼應著書名，希望協助讀者能快些上手安卓。

其次如前述，由於花了一些時間在 IoT 以及 DRM 影音串流，所以相關的應用反映在第 12 章以及第 15~16 章，其中有不少新的材料，讀過第二版的讀者應該可以感受得到。

雖然多次校搞，仍難免疏漏，作者將於確認之後，公布勘誤表於碁峰官網或是作者的網頁（aerael.com）。

感謝碁峰出版社大力支持，讓這次第三版順利問世，特此致謝。

另外，為了拋磚引玉，讓較缺乏資源的兒童或青少年受惠，「第一次學 Android 就上手」第三版從上架開始到 2020 年底為止，依照其峰資訊所統計的售出數量，作者將以定價的十分之一乘上售出數量，作為捐款數目，聊表心意。初步確認的對象會有「財團法人中華民國兒童福利聯盟文教基金會」等，捐款詳情請見 aerael.com 網頁。

2019.8 于台北

CHAPTER

04

觸控行為

4.1 前言

觸控螢幕（Touch Screen）技術其實不是近年來的新技術，早在多年前觸控介面就與 CRT 螢幕結合起來，經常出現在各大展場中，扮演導覽的角色；後來，金融 ATM 櫃員機也普遍應用。

換句話說，觸控螢幕應用在 PC 或 Notebook 筆電的情形雖然並不多見，但在個人數位助理（PDA）卻經常見到，因為往往搭配一支觸控筆，作隨身筆記等文書工作相當方便。

根據作者的觀察，這一波從 2007 年左右至今的智慧行動裝置浪潮所標準配備的「滑動式（Sliding）」觸控螢幕之所以風起雲湧，除了造成 2011 年底市場排名極大的變化，更造就後起之秀前仆後繼地發展，其濫觴要算是蘋果電腦的音樂隨身聽所謂的 ClickWheel（點擊式轉盤）介面了！

因為 ClickWheel 所帶來創新的操控潮流與風評，結合「點擊（Click）」與「轉動（Wheel）」於一身的極簡風格，逐漸為多數用戶所接受和期待使用的心理作用，確實造就後來無論是 iPod Touch、iPhone，以及目前市佔最高的 Android 系列手機與平板裝置。

智慧型手機在這十年發展以來，不外乎以下四類主要的觸控手勢行為：

1. 單點點擊（Click）：常見於各種按鈕點擊、選單選取、App 啟動等等之操作。
2. 單點滑動（稱為 Sliding 或是 Dragging）：常見於各種密碼解鎖、浮動式按鈕、滑動元件等等之操作。
3. 兩點縮放（Zooming）：常見於地圖縮放、相片縮放、網頁縮放等等之操作。
4. 多點手勢（Multi-touch gesture）：常見於特定 App 之手勢應用。

因此，本章特別在中間兩項觸控手勢，也就是滑動與縮放的行為，加以選材，說明觸控行為的用法精華。

4.2 用圖片作觸控點擊演練

在人機介面的訊息顯示內容中，文字與圖形絕對佔有多數的角色，而所謂的圖文並茂、看圖能說故事的訊息擷取概念，也貼切形容圖片絕對是一項極佳的媒體溝通管道。

在 Android，所有的視覺化元件都屬於 View 類別的「子孫」，這樣的說法源自於物件導向的觀念。而 View 類別都提供一個 Background 屬性，可以在 xml 或 Java 中設定背景圖片（Android 的圖片至少有 Drawable、Bitmap 和 Resource 三種存在的形式）。也就是說，圖片與視圖元件在此有了交集！

不僅如此，有些視圖元件還提供了前景圖片的設定方式，例如，ImageView、ImageButton 等等，即使其它元件沒有提供設定前景圖片的 API，卻能藉由覆寫 View 類別的 onDraw(Canvas)或 draw(Canvas)方法來設定圖片。

換言之，有了背景圖片和前景圖片的雙重設定方式，就能讓整個 Android App 的畫面更加豐富好看！而本小節就先以圖片結合點擊手勢做一些簡單的用法說明。

4.2.1 點擊圖片模擬翻牌動作

底下我們會展開一系列影像元件的測試動作，為要模擬 Poker 四張花色的翻牌動作。因此，我們事先準備好五張圖片，都放置在/res/drawable 資料夾內，分別是 96 x 96 的牌背 Android 圖案，以及 80 x 80 的牌面花色圖案，如圖 4-1 所示。

翻牌動作好比上一章所介紹的開關技巧，特別是 2 段式開關，所以同樣可以藉由一個 boolean 變數來記錄目前是牌面是向上或向下？

圖 4-2 顯示如何利用 ImageView 作為圖片展示的元件，以及相關的屬性設定動作。圖 4-3 則是 ImageView_1 所要達成的翻牌效果之擷圖。

最後是程式實作的部份，讀者可以看到很有趣的一件事情是，翻牌動作簡直就像開關，圖 4-4 實作的 clickToOpen 方法如同應用上一章的開關功能！

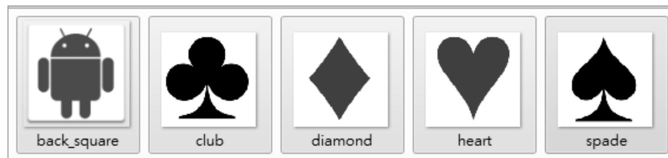
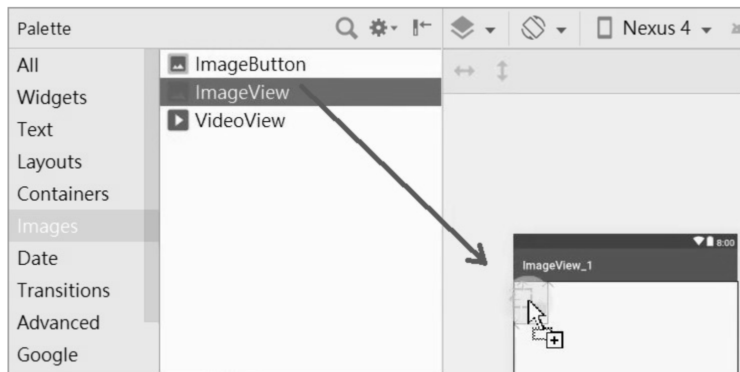
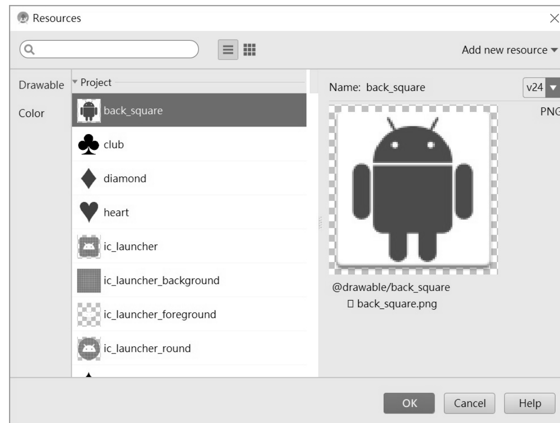


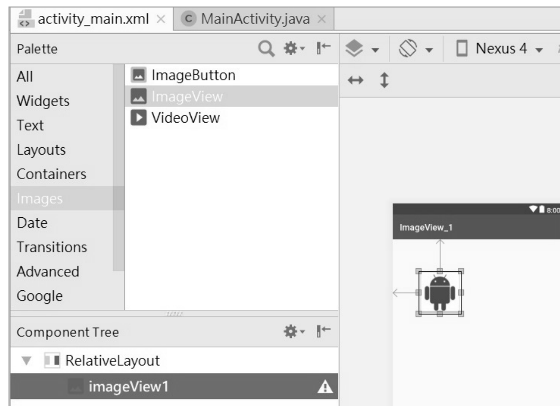
圖4-1 預先準備好的 Poker 花色圖片和牌背 Android 圖案。



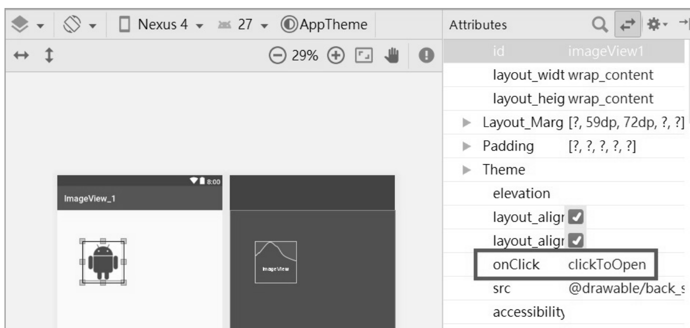
(a)



(b)



(c)



(d)

圖4-2 (a)以調色板拖曳 ImageView 元件；(b)拖曳之後出現對話框可選取初始圖片；(c)ImageView 元件上的圖片預覽；(d)以 ImageView 元件註冊 onClick 動作。

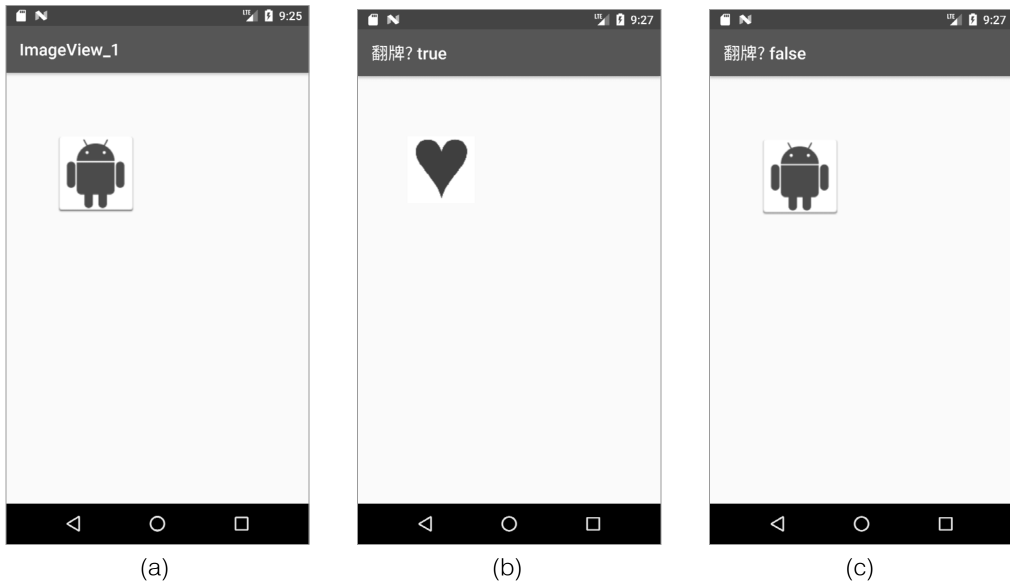


圖4-3 (a)ImageView_1 初始畫面；(b)點擊圖案奇數次，圖案變為紅心，表示翻牌；(c)點擊圖案偶數次，圖案變回 Android 機器人圖，表示蓋牌。

activity_main.xml × MainActivity.java ×

```
8   public class MainActivity extends AppCompatActivity {
9
10      ImageView iv1;
11      boolean bStatus = false;
12
13      @Override
14      protected void onCreate(Bundle savedInstanceState) {
15          super.onCreate(savedInstanceState);
16          setContentView(R.layout.activity_main);
17          //
18          iv1 = (ImageView) findViewById(R.id.imageView1);
19      }
20
21      public void clickToOpen(View v) {
22          bStatus = !bStatus;
23          if(bStatus)
24              iv1.setImageResource(R.drawable.heart);
25          else
26              iv1.setImageResource(R.drawable.back_square);
27          setTitle("翻牌? "+ bStatus);
28      }
29  }
```

圖4-4 運用 setImageResource 指令可以在 Java 程式內動態設定 ImageView 的圖片。

4.2.2 圖片縮放完善撲克顯示

圖片縮放的必要性，至少存在兩種情況：

- 圖片原稿尺寸比實際需要來得大或小
- 同一份圖片在程式內的應用，出現有兩種以上大小不同的尺寸需求

以圖 4-3 的 ImageView_1 專案為例，牌面（80 x 80）和牌背（96 x 96）的尺寸其實不同，眼尖的讀者可以發現翻牌/蓋牌時會有些晃動感！

圖片的縮放主要透過 Bitmap 類別和以下三組 API 達成，程式如圖 4-5：

1. bitmap = BitmapFactory.decodeResource((getResources(), R.drawable.OOO);
2. bitmap = Bitmap.createScaleBitmap(bitmap, dstWidth, dstHeight, filter);
3. iv1.setImageBitmap(bitmap);

其中，createScaleBitmap()指令就是其中縮放的關鍵，然後執行結果就如圖 4-6 所示，能夠以適當的比例將圖片嵌入 ImageView 中。[1]

至於 createScaleBitmap()指令中的第 4 個參數 filter 應該填 true 或 false，在這個例子影響不大，但如官網所描述，建議值是 true：

```
  Whether or not bilinear filtering should be used when scaling the bitmap.
If this is true then bilinear filtering will be used when scaling which has
better image quality at the cost of worse performance. If this is false then
nearest-neighbor scaling is used instead which will have worse image quality
but is faster. Recommended default is to set filter to 'true' as the cost of
bilinear filtering is typically minimal and the improved image quality is
significant.
```

因為既能改善縮放畫質，同時所需運算的成本也不大。

[1] https://developer.android.com/reference/android/graphics/Bitmap

```
MainActivity.java
15      Bitmap bmpHeart, bmpBack;
16
17      @Override
18      protected void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20          setContentView(R.layout.activity_main);
21          //
22          iv1 = (ImageView) findViewById(R.id.imageView1);
23
24          bmpHeart = BitmapFactory.decodeResource(getResources(), R.drawable.heart);
25          bmpBack = BitmapFactory.decodeResource(getResources(), R.drawable.back_square);
26          bmpBack = Bitmap.createScaledBitmap(bmpBack,
27                  bmpHeart.getWidth(), bmpHeart.getHeight(), filter: true);
28          iv1.setImageBitmap(bmpBack);
29      }
30
31      public void clickToOpen(View v) {
32          bStatus = !bStatus;
33          if(bStatus)
34              iv1.setImageBitmap(bmpHeart);
35          else
36              iv1.setImageBitmap(bmpBack);
37          setTitle("翻牌? "+ bStatus);
38      }
```

圖4-5 運用 ①BitmapFactory.decodeResource() 、②Bitmap.createScaleBitmap() 搭配 ③setImageBitmap()指令可以在 Java 程式內動態設定 ImageView 的圖片縮放。

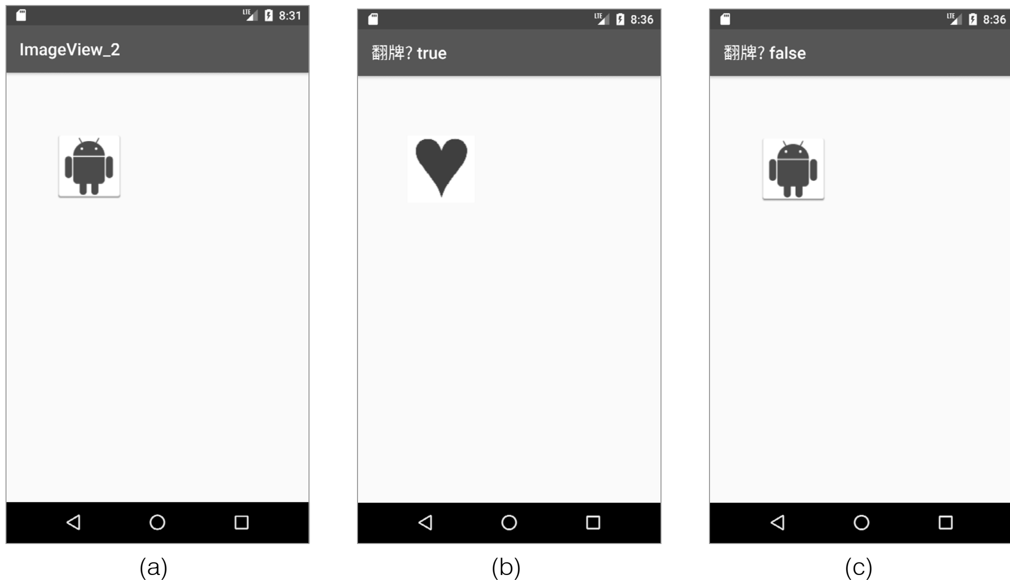


圖4-6 以 ImageView_2 專案說明圖片縮放：(a)初始畫面，牌背大小已經調整；(b)點擊圖案奇數次，圖案變為紅心，尺寸一致；(c)點擊圖案偶數次，圖案變回 Android 機器人圖，尺寸仍然一致。

4.3 自製視圖元件

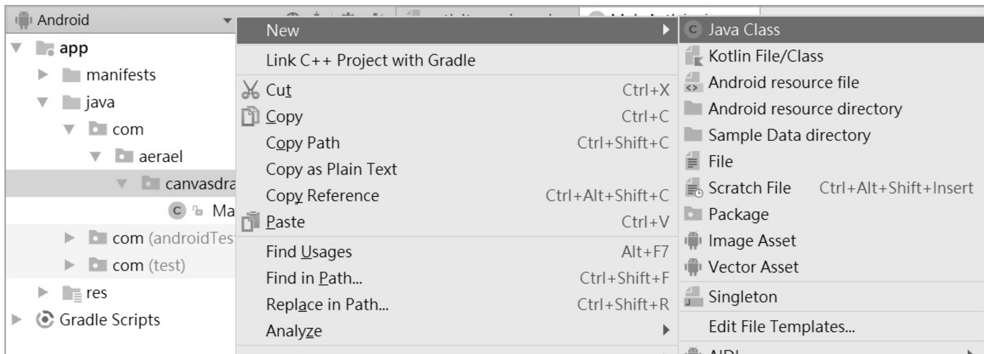
本章一開始就提到，大部份的視圖元件雖然沒有提供設定前景圖片的 API，卻能藉由覆寫 View 類別的 onDraw(Canvas)或 draw(Canvas)方法來設定圖片，甚至如果有必要，還能藉由覆寫所有的 View 子類別來改變原來的繪圖行為。

自製視圖元件之重點除了元件的繪圖（Draw）之外，通常還要針對元件的觸控事件作處理，才能達到人機介面互動的設計。因此，這一小節先分別用 CanvasDraw_1 和 CanvasDraw_2 專案介紹自製視圖的基本操作和觸控事件處理方式。然後才能進一步說明自製視圖與 ImageView 在觸控事件的應用上，分別需要如何進行。

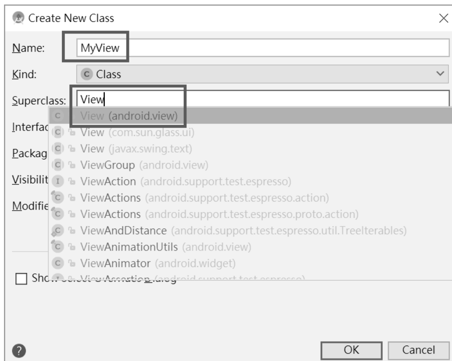
4.3.1 視圖的顯示與觸控

本節以圖 4-7 和 4-8 說明自製視圖元件的兩部份重點：一個是元件的顯示設計，一個是元件的觸控處理。

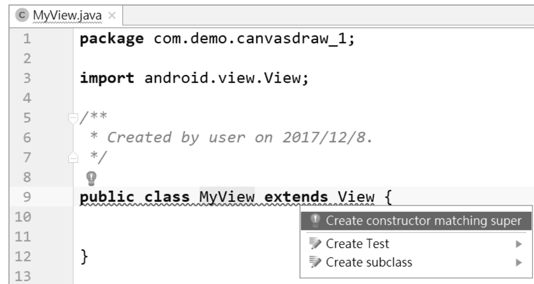
就先以一張圖片為例子，除了 ImageView 的作法之外，事實上所有的 View 都提供畫布（Canvas）的繪製功能，如此一來，就能隨設計者的心意，自行製作視圖元件，也就是所謂的客製化（Customized）。



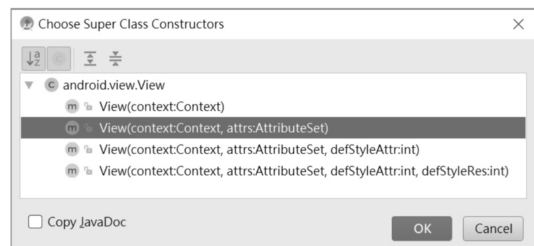
(a)



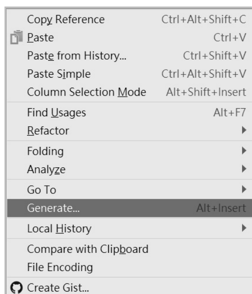
(b)



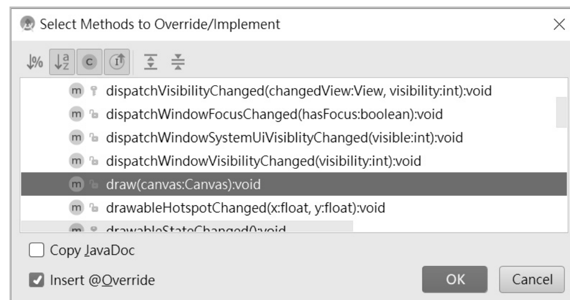
(c)



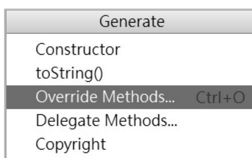
(d)



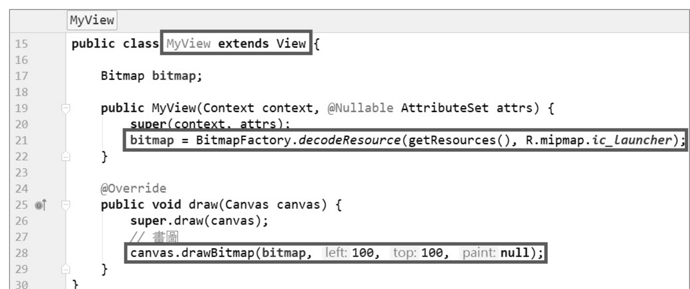
(e)



(g)



(f)



(h)

(i) (j)

(k)

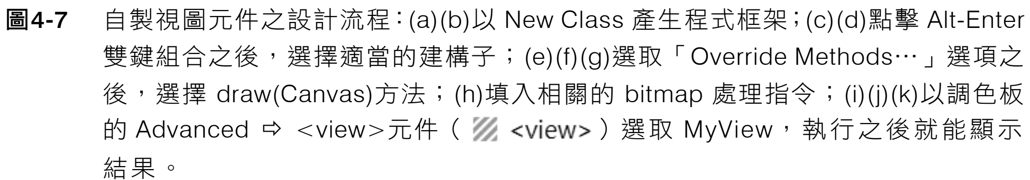
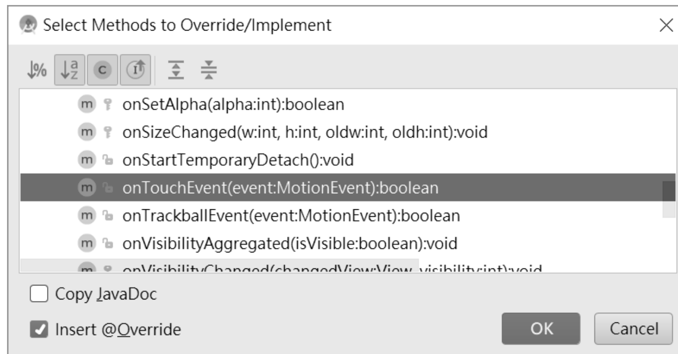


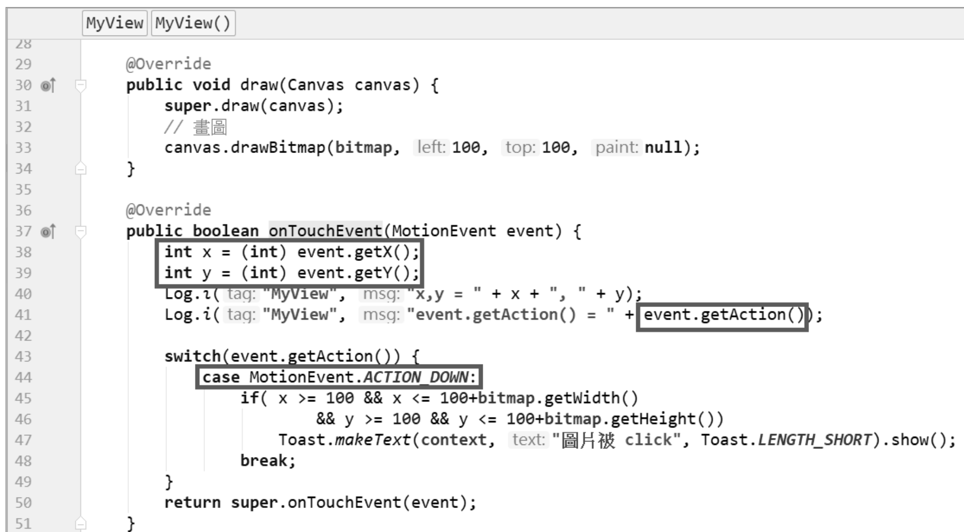
圖4-7 自製視圖元件之設計流程：(a)(b)以 New Class 產生程式框架；(c)(d)點擊 Alt-Enter 雙鍵組合之後，選擇適當的建構子；(e)(f)(g)選取「Override Methods…」選項之後，選擇 draw(Canvas)方法；(h)填入相關的 bitmap 處理指令；(i)(j)(k)以調色板的 Advanced ⇨ <view>元件（ <view>）選取 MyView，執行之後就能顯示結果。

首先，我們可以隨意新增一個專案，並在此專案以滑鼠右鍵點擊套件（package）名稱，選擇 New Class 就能新增一個 class，這時讀者可以參考如圖 4-7(a)(b)所示的內容，再按下完成鈕，就能產生一個圖 4-7(c)般的自製視圖元件框架。框架產生後還需要選擇適當的建構子才能解除錯誤，可以按下 Alt-Enter 雙鍵組合選擇建構子解除錯誤，如圖 4-7(d)所示。

接著仍以滑鼠右鍵點擊，但要注意所點擊的區域須在「類別內、眾方法外」的位置，則必然可以跳出如圖 4-7(e)的對話框，這時再游移鼠標至「Generate」並選取「Override Methods...」選項即可，如圖 4-7(f)所示。



(a)



(b)

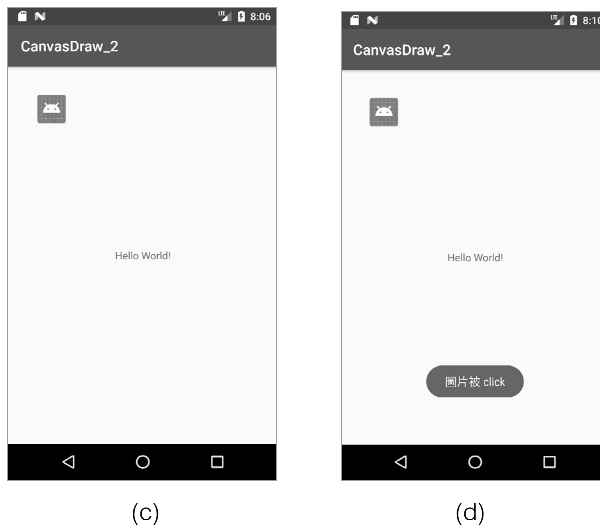


圖4-8 CanvasDraw_2 專案：(a)選取覆寫 onTouchEvent(MotionEvent)；(b)判斷是否觸控在圖片範圍內；(c)初始畫面；(d)點擊圖案之後以 Toast 顯示訊息。

從圖 4-7(g)與(h)可以看到，選取 draw(Canvas)方法之後，就會自動嵌入程式框架，此時讀者就可將前面幾節所學的相關的 Bitmap 藉由「Bitmap 工廠」加以「生產」出來，然後藉由以下指令將圖片畫在(100,100)的座標點上：

```
canvas.drawBitmap(bitmap, 100, 100, null);
```

最後善用調色板的「Advanced ⇨ <view>元件（ <view> ）」，選取 MyView 並執行如圖 4-7(i)(j)(k)。

元件的觸控處理不能再以 onClick，而是要改成覆寫 onTouchEvent 方法，理由已經在第二章的第六個基本動作和 BasicAction_6 專案介紹過。

因為 onClick 是針對一整個元件是否被按壓或釋放，並不考慮確切的 x, y 點擊位置，若是要處理元件中的某一塊區域，就要利用 onTouchEvent，因為它提供 x, y 座標可以加以判斷。

因此，如圖 4-8(a)(b)所示，須選取覆寫 onTouchEvent(MotionEvent)，並加以判斷所碰觸的 x, y 位置是否屬於圖片範圍內？接下來的動作再溫習一次。

onTouchEvent(MotionEvent)的參數型態為 MotionEvent，所包含的 API 至少有三組相當常用，如圖 4-8(b)：

- `int x = (int) event.getX();`
- `int y = (int) event.getY();`
- `event.getAction()`

其中的 event.getAction()又有三種觸控事件最為常見，只是 CanvasDraw_2 專案僅用到觸控按壓的 MotionEvent.ACTION_DOWN，模擬 onClick 事件。在此攔截觸控按壓的 MotionEvent.ACTION_DOWN 事件就能根據所按壓的座標點與圖片的座標區域作比較，看看是否有碰到圖片。執行結果如圖 4-8(c)與(d)。

值得一題的是，CanvasDraw_2 專案用到一個技巧，就是儲存自製視圖的 MyView 建構子的 Context 參數，作為後面 Toast 的一個參數之用！

4.4 用圖片作觸控滑動演練

第 3 章開宗明義提到「在 Android，所有的視覺化元件都屬於 View 類別的子孫」，這個 View 類別通常譯成「視圖」。這個視圖元件不但是在視覺輸出（Output）上有許多呈現的方式，它也內建一個重要的觸控輸入（Input）控制器，稱為 onTouchEvent()，其 API 如下：

```
public boolean onTouchEvent(MotionEvent event);
```

滑動手勢的應用非常多元，最出名的應該是滑動解鎖的手勢，因為蘋果陣營成功取得這項專利，還迫使其它智慧行動裝置改變它們的螢幕解鎖策略！

此外，利用滑動手勢控制 ListView 和 RecycleView 等多元素元件的操作介面也會在本書後續章節介紹。本節則介紹滑動手勢分別在自製視圖和 ImageView 上的用法。

4.4.1 藉由自製視圖滑動圖片

第 4.3 節 CanvasDraw_2 專案簡單點到了如何覆寫 onTouchEvent()，圖 4-9 的 MoveImage_1 專案則要開始示範如何以手指滑動圖片，從圖(a)到(c)分別是滑動過程的三張螢幕擷圖。

它的技術有一大部份其實已經在第 4.3 節的 CanvasDraw_2 專案介紹過，就是利用 Canvas 將 Bitmap 畫上去，然後運用 onTouchEvent 作觸控判斷；最大的差異在於 CanvasDraw_2 專案只用到 ACTION_DOWN 事件，但 MoveImage_1 專案則用齊了三組事件：從第一次碰觸螢幕、滑動、到離手事件。

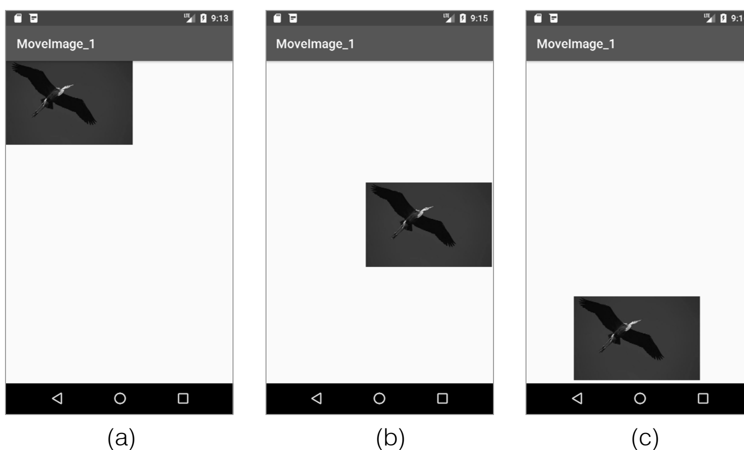
圖 4-10 是程式碼的重點標示：圖(a)先將 Bitmap 建立並縮小至 600x400 之後，就利用 onDraw(Canvas)方法先將圖片畫在螢幕左上角。

圖(b)就是最關鍵之處：因為它需要在第一次碰觸圖片時作兩件事：記錄碰觸點與圖片左上角的相對 x, y 分量，並設下碰觸旗標。

其次在手指移動圖片時，隨時更新最新的圖片 x, y 座標，並重新繪圖。最後在手指離開圖片時，再將碰觸旗標歸零。重新繪圖的指令如下：

```
invalidate();
```

如果視圖可見，invalidate()指令將在某個時間點呼叫 onDraw()。



(a) (b) (c)

圖4-9 MoveImage_1 專案的執行結果：(a)初始畫面，圖片位在左上角；(b)用手指(或是在模擬器則用滑鼠)將圖片移動到原位置的右下方；(c)再移到螢幕正下方。

```
MainActivity.java
17      @Override
18      protected void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20  //      setContentView(R.layout.activity_main);
21
22          mBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.myimage);
23          mBitmap = Bitmap.createScaledBitmap(mBitmap, dstWidth: 600, dstHeight: 400, filter: true);
24          mView = new MyView( context: this);
25          setContentView(mView);
26      }
27
28      private class MyView extends View {
29          private int imageX = 0;
30          private int imageY = 0;
31          int dx, dy;
32          boolean bTouched = false;
33          //
34          public MyView(Context context) { super(context); }
37          @Override
38          protected void onDraw(Canvas canvas) {
39              canvas.drawBitmap(mBitmap, imageX, imageY, paint: null);
40          }
```

(a)

```
MainActivity.java
37          @Override
38          protected void onDraw(Canvas canvas) { canvas.drawBitmap(mBitmap
41          @Override
42          public boolean onTouchEvent(MotionEvent event) {
43              int x = (int) event.getX();
44              int y = (int) event.getY();
45              if(event.getAction() == MotionEvent.ACTION_DOWN){
46                  if(x>imageX && x < imageX + mBitmap.getWidth() &&
47                          y>imageY && y < imageY + mBitmap.getHeight()) {
48                      dx = x - imageX;
49                      dy = y - imageY;
50                      bTouched = true;
51                  }
52              }
53              else if(event.getAction() == MotionEvent.ACTION_MOVE){
54                  if(bTouched) {
55                      imageX = x - dx;
56                      imageY = y - dy;
57                  }
58              }
59              else if(event.getAction() == MotionEvent.ACTION_UP){
60                  bTouched = false;
61              }
62              invalidate();                                   //再描繪的指示
63              return true;
64          }
MainActivity › MyView › onTouchEvent()
```

(b)

圖4-10 MoveImage_1 專案的程式重點：(a)Bitmap 的建立與繪製；(b)圖片滑動的演算方法，需用到①第一次碰觸螢幕、②滑動、以及③離手共三組觸控事件，並透過 invalidate()指令，於每次事件之後，重新讓 onDraw()程式動作。

4.4.2 藉由 ImageView 滑動圖片

手指滑動圖片的方法不只有前面所介紹的，另一種是直接以 ImageView 元件來顯示圖片並加以滑動。

然而，我們在第 3 章雖然也介紹以 onClick 對 ImageView 作翻牌、蓋牌的應用，卻未曾結合 onTouchEvent()作滑動的操作說明，事實上，我們確實可以運用 ImageView 註冊 onTouch 事件控制器，再隨著手指在圖片上的滑動動作來更新 ImageView 的位置。

圖 4-11 將它的運算原理揭露出來，首先，螢幕上的圖片放大來看，必然有一個左上角座標作為圖片（ImageView）繪製的起點。其次，手指可以在圖片上任一點接觸，假設為圖中黑色實心圓所代表的位置。

因此，理論上手指可以有八個方向進行相鄰點滑動（東、西、南、北、東北、東南、西北、西南），但實務上，MotionEvent.ACTION_MOVE 事件會隨著手指滑動的速度，決定所偵測到的「下一個點」x_1 和 y_1，這時，(x_0, y_0)和(x_1, y_1)所形成的向量 d 將會是滑動 ImageView 圖片後，最新位置的更新依據。

從圖 4-12 的 MoveImage_2 執行結果看不出來與 MoveImage_1 有何差別？但是圖 4-13 的程式就可以看出，不依靠 Canvas 作法的方法二如圖(a)所示，只需要以 RelativeLayout.LayoutParams 作一些參數初始化，並註冊 onTouch 事件監聽器，再於圖(b)所示加以實作滑動手勢的 onTouch 控制器即可。

讀者可能注意到，在此專案中，使用 setLayoutParams()的效果可以免用 invalidate()。

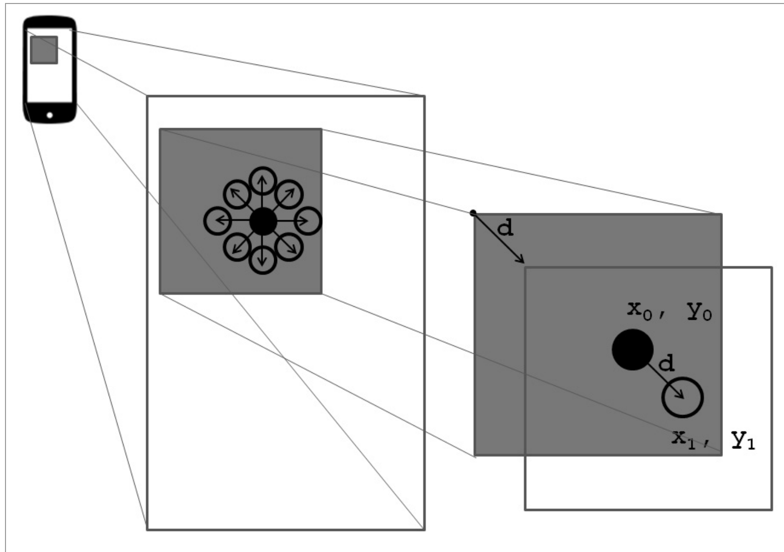
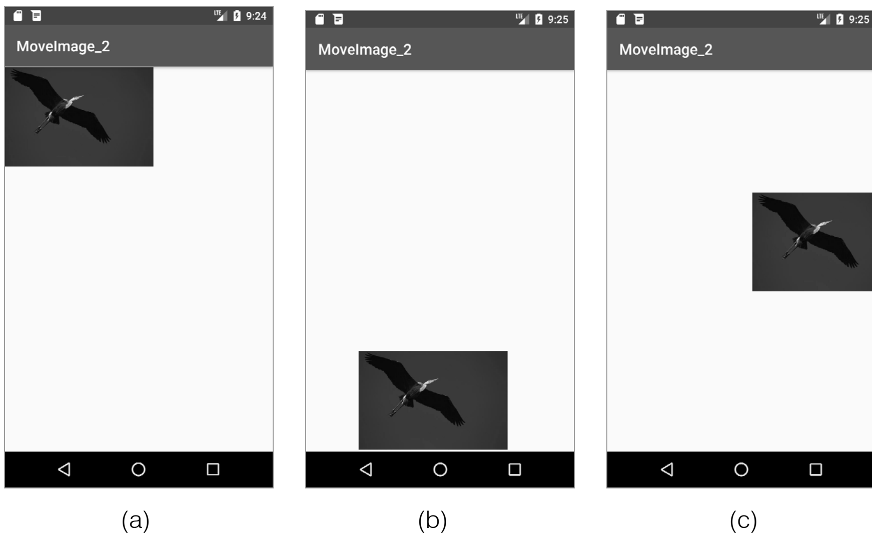


圖4-11 藉由 ImageView 滑動圖片的原理展示。



(a) (b) (c)

圖4-12 MoveImage_2 專案的執行結果：(a)初始畫面，圖片位在左上角；(b)用手指(或是在模擬器則用滑鼠)將圖片移動到螢幕正下方；(c)再移往右上方。

```java
MainActivity.java
12  public class MainActivity extends AppCompatActivity implements OnTouchListener {
13
14      RelativeLayout rl1;
15      ImageView iv1;
16      RelativeLayout.LayoutParams params;
17      int x0, y0, ivW = 600, ivH = 400;
18
19      @Override
20      protected void onCreate(Bundle savedInstanceState) {
21          super.onCreate(savedInstanceState);
22          setContentView(R.layout.activity_main);
23
24          prepareViews();
25      }
26
27      private void prepareViews() {
28          rl1 = (RelativeLayout) findViewById(R.id.relative);
29          iv1 = new ImageView( context: this);
30          iv1.setImageResource(R.drawable.myimage);
31          params = new RelativeLayout.LayoutParams(ivW, ivH);
32          params.addRule(RelativeLayout.ALIGN_LEFT);
33          params.addRule(RelativeLayout.ALIGN_TOP);
34          iv1.setLayoutParams(params);
35          iv1.setOnTouchListener(this);
36          rl1.addView(iv1);
37      }
```

(a)

```java
MainActivity.java
39      @Override
40      public boolean onTouch(View view, MotionEvent event) {
41          // TODO Auto-generated method stub
42          if(view == iv1) {
43              int x = (int) event.getX();
44              int y = (int) event.getY();
45              if(event.getAction() == MotionEvent.ACTION_DOWN){
46                  x0 = x;
47                  y0 = y;
48              }
49              else if(event.getAction() == MotionEvent.ACTION_MOVE){
50                  params = (LayoutParams) iv1.getLayoutParams();
51                  params.leftMargin += (x - x0);
52                  params.topMargin  += (y - y0);
53                  params.rightMargin += (x - x0) + ivW;
54                  params.bottomMargin  += (y - y0) + ivH;
55                  iv1.setLayoutParams(params);
56              }
57          }
58          return true;
59      }
```

(b)

圖4-13 MoveImage_2 專案的重點內容：(a)以 RelativeLayout.LayoutParams 作 ImageView 的版面參數初始化，並註冊 onTouch 事件監聽器；(b)參考圖 4-11 的原理實作滑動手勢的 onTouch 事件監聽器。

4.5 兩指縮放圖片

觸控螢幕和傳統螢幕之所以不同，除了可以觸控方式取代滑鼠成為新的人機介面方式以外，觸控螢幕優於滑鼠的地方，還有它能支援多點觸控！

「多點觸控」讓用戶有更多、更棒的人機介面經驗，像是本節所要介紹的「兩指縮放圖片」功能，僅僅兩點的觸控變化，就讓操作變得更豐富有趣。

「兩指縮放」的動作普遍在 App 見得到，特別是智慧型手機一般擁有較小尺寸的螢幕，對於資訊的「放大顯示」有著本質上的需求。因此，舉凡網頁、地圖、照片預覽等等，都是它能充分發揮的地方。也難怪智慧型手機這麼讓人欲罷不能！

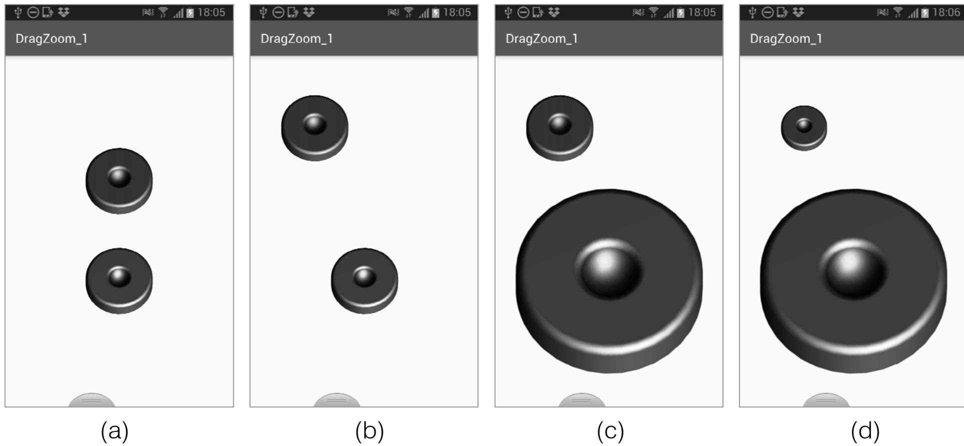
以圖 4-14 的 DragZoom_1 專案為例，初始畫面圖(a)置中放上兩顆像是鈕扣般的按鈕，這兩顆鈕扣確實都能像 4.2 節的滑動功能一般，在螢幕上滑來滑去，讀者可以嘗試看看，或是參考圖 4-14(b)。

從程式列表 4-1 可以清楚看到，它是利用 ImageView 的滑動技術所作成的，無論在模擬器或是在實體手機上，都能成功演練。

然而，接下來的兩個操作演練，就必須要在實體手機上了！因為絕大多數的 PC 螢幕都未有支援多點觸控螢幕的功能。如圖 4-14(c)和(d)所示，分別是對畫面下方和上方的鈕扣圖片，進行兩指的放大與縮小之動作，果然圖片就因此變大或縮小了！

讀者可能聽過 Zooming 這個名詞，經常用在像是相機從長鏡頭到特寫鏡頭平滑地變化動作，也常在地圖或其他顯示資訊的軟體見到 ➕ 和 ➖ 的縮放按鈕。這就是 DragZoom_1 專案名稱裏頭 Zoom 的由來，其實它主要包含一個 DragZoomListener，本質就是一個 OnTouchListener。

它的用法確實如同 OnTouchListener，讀者可以查看 DragZoom_1 專案裏頭的 MainActivity.java 程式，用法如後。



(a) (b) (c) (d)

圖4-14 DragZoom_1 專案的執行結果：(a)初始畫面，兩個 ImageView 呈現兩張圖片；(b)用手指(或是在模擬器則用滑鼠)將上方圖片往左上角移動；(c)利用兩隻手指(可能無法在模擬器用滑鼠實驗)，將下方圖片作「撐開手勢」完成放大；(d) 再利用兩隻手指，將上方圖片作「擠回手勢」完成縮小。

如前述，這兩個鈕扣圖片既然是運用 ImageView 加以呈現的，我們就可以透過 findViewById()指令先取得該物件指標：

```
ImageView iv1 = (ImageView) findViewById(R.id.imageView1);
```

然後如同 OnTouchListener 的註冊方式，將 DragZoomListener 物件初始化以後，作為 setOnTouchListener()指令的參數，就能開始使用，用法是不是很直覺呢？

```
iv1.setOnTouchListener(new DragZoomListener());
```

程式列表 4-1 有幾個值得觀察與欣賞的重點：

① 一次包含 CLICK、DRAG、與 ZOOM，是怎麼辦到的？

② spacing(MotionEvent event)和 midPoint(PointF point, MotionEvent event)兩組 API 的作用為何？

③ 為何多了一個 MotionEvent.ACTION_MASK 須和 event.getAction()一起作用？

關於上述①，簡答之，DRAG 最簡單，就是只要沒有第二個手指接觸螢幕，且又發生 ACTION_MOVE 的事件，就是為 DRAG 模式。

CLICK 模式也不難解釋，就是當 DRAG 的移動距離小於一個臨界值（程式列表 4-1 暫定為 x 與 y 的位移各自小於 2，讀者可以自訂），就視為 CLICK。

ZOOM 模式比較複雜，所以用圖 4-15 的流程圖解釋一下，它必須有兩個發生的條件：第二個觸控點事件、以及滿足兩觸控點的最小距離門檻值。

從圖 4-15 能看出上述②關於 spacing()和 midPoint()兩組 API 的應用場景。

至於上述③的 ACTION_MASK 之作用，則需要利用表格 4-1 中，各個事件背後所代表的 16 進位常數值，才能加以解釋。[2]

表4-1 程式列表 4-1 出現的六種 MotionEvent 與其所代表的 16 進位常數值對照 event.getAction()所收到的 16 進位值

編號	名稱（依照出現的順序）	16 進位常數值	event.getAction()
1	MotionEvent.ACTION_MASK	0x000000ff	0x000000ff
2	MotionEvent.ACTION_DOWN	0x00000000	0x00000000
3*	MotionEvent.ACTION_POINTER_DOWN	**0x00000005**	**0x00000105**
4	MotionEvent.ACTION_UP	0x00000001	0x00000001
5	MotionEvent.ACTION_MOVE	0x00000002	0x00000002
6*	MotionEvent.ACTION_POINTER_UP	**0x00000006**	**0x00000106**

從表格 4-1 可以清楚發現，有兩個事件值和缺少執行 ACTION_MASK()的結果是不一樣的，一個是 ACTION_POINTER_DOWN、一個是 ACTION_POINTER_UP，但是差異不大，就在第 3 位的 16 進位值多了一個 1，這個 1 就是代表第 2 個觸控點。換句話說，如果利用 ACTION_MASK 和 event.getAction()彼此作個「位元邏輯&」，就能過濾掉那個 1 了。

2 https://developer.android.com/reference/android/view/MotionEvent.html#ACTION_MASK

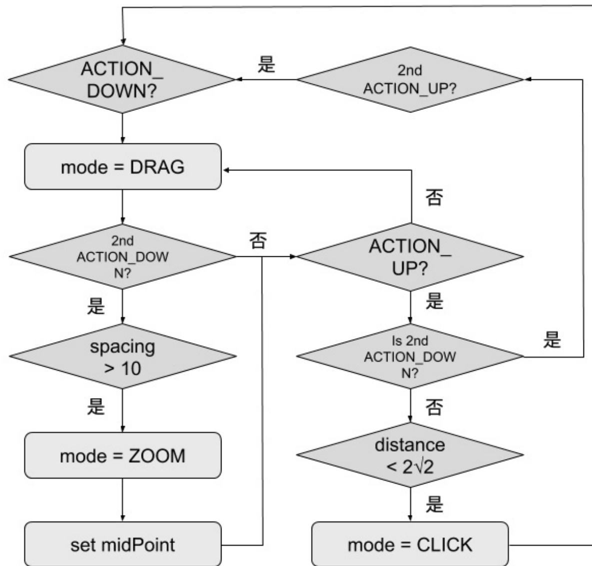


圖4-15 DragZoom_1 專案的重點流程圖，說明 mode 發生 DRAG、ZOOM、CLICK 的時機點，以及 midPoint()和 spacing()兩組 API 的使用點。

程式列表 4-1：DragZoomListener.java

```
10  public class DragZoomListener implements OnTouchListener {
11
12      private static final String TAG = "DragZoomListener";
13
14      private ImageView iv1;
15
16      // We can be in one of these 4 states
17      private static final int NONE = 0;
18      private static final int DRAG = 1;
19      private static final int ZOOM = 2;
20      private static final int CLICK = 3;
21      private int mode = NONE;
22
23      // Remember some things for zooming
24      private PointF start = new PointF();
25      private PointF mid = new PointF();
26      private float oldDist = 1f;
27      private PointF startRef = new PointF();
```

```
28
29      @Override
30      public boolean onTouch(View v, MotionEvent event) {
31          iv1 = (ImageView) v;
32          switch (event.getAction() & MotionEvent.ACTION_MASK) {
33              case MotionEvent.ACTION_DOWN:
34                  start.set(event.getX(), event.getY());
35                  startRef.set(iv1.getX()+event.getX(),
    iv1.getY()+event.getY());
36                  Log.d(TAG, "ACTION_DOWN: " + (iv1.getX()+event.getX()) + ",
    " + (iv1.getY()+event.getY()) );
37                  mode = DRAG;
38                  break;
39              case MotionEvent.ACTION_POINTER_DOWN:
40                  oldDist = spacing(event);
41                  Log.d(TAG, "oldDist=" + oldDist);
42                  if (oldDist > 10f) {
43                      midPoint(mid, event);
44                      mode = ZOOM;
45                      Log.d(TAG,  "mode=ZOOM:  "  +  iv1.getX()  +  ",  "  +
    iv1.getY());
46                  }
47                  break;
48              case MotionEvent.ACTION_UP:
49                  double dxDiff = (iv1.getX()+event.getX()) - startRef.x;
50                  int xDiff = (int) Math.abs(dxDiff);
51                  double dyDiff = (iv1.getY()+event.getY()) - startRef.y;
52                  int yDiff = (int) Math.abs(dyDiff);
53                  if (mode == DRAG && xDiff < 2 && yDiff < 2){
54                      mode = CLICK;
55                      Log.d(TAG, "mode= CLICK");
56                      iv1.performClick();
57                  }
58              case MotionEvent.ACTION_POINTER_UP:
59                  mode = NONE;
60                  Log.d(TAG, "mode=NONE");
61                  break;
62              case MotionEvent.ACTION_MOVE:
63                  if (mode == DRAG) {
64                      // 重點在於 setScaleX,Y 之後的縮放比要反應到 setX,Y 去，反
    應方式如下
65                      iv1.setX(iv1.getX() + (event.getX() - start.x)*iv1.
    getScaleX());
```

```
66                     iv1.setY(iv1.getY() + (event.getY() - start.y)*iv1.
    getScaleY());
67                 }
68                 else if (mode == ZOOM) {
69                     float newDist = spacing(event);
70                     Log.d(TAG, "newDist=" + newDist);
71                     if (newDist > 10f) {
72                         float scale = newDist / oldDist;
73                         iv1.setScaleX(iv1.getScaleX()*scale);
74                         iv1.setScaleY(iv1.getScaleY()*scale);
75                     }
76                 }
77                 break;
78         }
79         return true; // indicate event was handled
80     }
81 
82     /** Determine the space between the first two fingers */
83     private float spacing(MotionEvent event) {
84         float x = event.getX(0) - event.getX(1);
85         float y = event.getY(0) - event.getY(1);
86         return (float) Math.sqrt(x * x + y * y);
87     }
88 
89     /** Calculate the mid point of the first two fingers */
90     private void midPoint(PointF point, MotionEvent event) {
91         float x = event.getX(0) + event.getX(1);
92         float y = event.getY(0) + event.getY(1);
93         point.set(x / 2, y / 2);
94     }
95 }
```

4.6 思考與練習

讀完本章之後，可以嘗試思考與練習以下題目：

1. 試參考第二章基本動作 12 之 BasicAction_12b 專案，將 4.5 節的 DragZoom_1 專案的 DragZoomListener 作成 Android Library，再於 MainActivity 利用專案之 Project Structure 對話框，新增 Module Dependency，觀察程式的動作是否仍然正確？

   或是直接開啟隨書雲端 zip 之 DragZoom_2 專案，加以執行測試。

2. 試將 MoveImage_2 專案內的 param 的作法，改成 DragZoom_1 專案內的 setX()和 setY()的作法，觀察程式的動作是否仍然正確？

3. 試將隨書雲端 zip 之 FingerPaint_1 專案執行起來，並測試以下功能：

   - 在預設的色彩下，用手指或滑鼠寫下安卓兩個字。
   - 擦掉安卓兩個字，並更換色彩。
   - 在更換的色彩下，用手指或滑鼠寫下 Android 英文字。