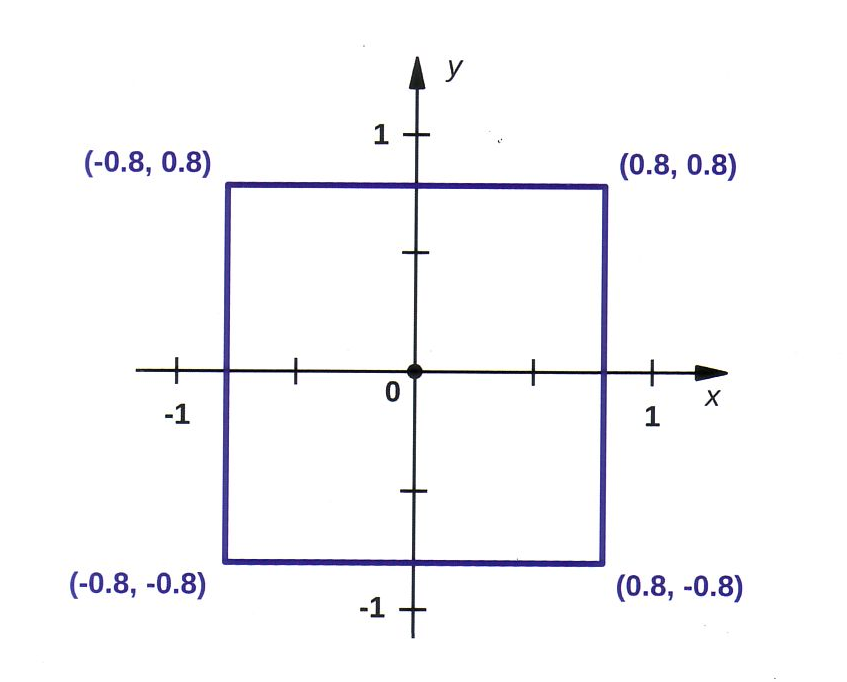
Android Canvas API

2D 繪圖



Yevgen Karpenko

Contents

[序 7](#_Toc481700893)

[第一章 Canvas API 簡介 10](#_Toc481700894)

[1.1 Canvas 10](#_Toc481700895)

[1.2 Paint 11](#_Toc481700896)

[1.3 Path 11](#_Toc481700897)

[1.4 Typeface 12](#_Toc481700898)

[1.5 Matrix 12](#_Toc481700899)

[1.6 範例: CH01\_Canvas 12](#_Toc481700900)

[1.6.1 MainActivity 12](#_Toc481700901)

[1.6.2 MyCanvas 13](#_Toc481700902)

[1.6.3 程式碼解說 : onSizeChange() 14](#_Toc481700903)

[1.6.4程式碼解說: Paint 15](#_Toc481700904)

[1.6.5程式碼解說: onDraw() 16](#_Toc481700905)

[1.6.6 螢幕截圖 17](#_Toc481700906)

[第二章 座標系統 20](#_Toc481700907)

[2.1 笛卡爾座標系統 20](#_Toc481700908)

[2.2 裝置或顯示座標系統 21](#_Toc481700909)

[2.3 邏輯座標系統 23](#_Toc481700910)

[2.4 全域與區域座標系統 23](#_Toc481700911)

[第三章 基本圖形 25](#_Toc481700912)

[3.1 Lines 25](#_Toc481700913)

[3.1.1 Example 25](#_Toc481700914)

[3.1.2 Code Review: Constructor 27](#_Toc481700915)

[3.1.3 Code Review: onSizeChanged( ) 27](#_Toc481700916)

[3.1.4 Code Review: onDraw( ) 28](#_Toc481700917)

[3.2 Colors 28](#_Toc481700918)

[3.2.1 Alpha Channel 28](#_Toc481700919)

[3.2.2 Integers 28](#_Toc481700920)

[3.2.3 Color Class 28](#_Toc481700921)

[3.3 Cap Styles 29](#_Toc481700922)

[3.4 Introduction to Paths 30](#_Toc481700923)

[3.4.1 Example 30](#_Toc481700924)

[3.4.2 Code Review : Constructor 32](#_Toc481700925)

[3.4.3 Code Review : onSizeChanged() 32](#_Toc481700926)

[3.4.4 Code Review : onDraw() 33](#_Toc481700927)

[3.5 Join Styles 33](#_Toc481700928)

[3.5.1 Example 34](#_Toc481700929)

[3.5.2 Code Review : Constructor 35](#_Toc481700930)

[3.5.3 Code Review : onSizeChanged( ) 35](#_Toc481700931)

[3.5.4 Code Review : onDraw( ) 36](#_Toc481700932)

[3.5.5 Screenshots 36](#_Toc481700933)

[3.5.6 Non-Closed Path 38](#_Toc481700934)

[3.6 Paint Styles : Stroke and Fill 39](#_Toc481700935)

[3.6.1 Example 40](#_Toc481700936)

[3.6.2 Code Review : Constructor 41](#_Toc481700937)

[3.6.3 Code Review : onSizeChanged() 41](#_Toc481700938)

[3.6.4 Code Review : onDraw() 41](#_Toc481700939)

[3.6.5 Screenshots 42](#_Toc481700940)

[3.7 Dashed Lines 42](#_Toc481700941)

[3.7.1 Example 43](#_Toc481700942)

[3.7.2 Code Review 44](#_Toc481700943)

[Chapter 04 Working with Text 46](#_Toc481700944)

[4.1 Standard Fonts 46](#_Toc481700945)

[4.1.1 Typeface 46](#_Toc481700946)

[4.1.2 Serifs 46](#_Toc481700947)

[4.1.3 Proportion 47](#_Toc481700948)

[4.1.4 Typeface Class 47](#_Toc481700949)

[4.1.5 Example 48](#_Toc481700950)

[4.2 Custom Fonts 51](#_Toc481700951)

[4.2.1 Example 51](#_Toc481700952)

[4.3 Horizontal Alignment 53](#_Toc481700953)

[4.3.1 Standard Alignment 53](#_Toc481700954)

[4.3.2 Custom Alignment 55](#_Toc481700955)

[4.4 Vertical Alignment 57](#_Toc481700956)

[4.4.1 Baseline 57](#_Toc481700957)

[4.4.2 Ascent and Descent 58](#_Toc481700958)

[4.4.3 Bottom Alignment 60](#_Toc481700959)

[4.4.4 Center Alignment 60](#_Toc481700960)

[4.4.5 Top Alignment 64](#_Toc481700961)

[4.5 Shadow 66](#_Toc481700962)

[Chapter 5 Rectangles 69](#_Toc481700963)

[5.1 Example : Bar Chart 69](#_Toc481700964)

[5.1.1 Model 70](#_Toc481700965)

[5.1.2 View 71](#_Toc481700966)

[5.1.3 Code Review : Class Variables 74](#_Toc481700967)

[5.1.4 Code Review : Constructor 75](#_Toc481700968)

[5.1.5 Code Review : onSizeChnged() 75](#_Toc481700969)

[5.1.6 Code Review : 76](#_Toc481700970)

[5.1.7 Screenshots 78](#_Toc481700971)

[Chapter 6 Circles, Ovals and Arcs 80](#_Toc481700972)

[6.1 Circles 80](#_Toc481700973)

[6.2 Ovals 82](#_Toc481700974)

[6.3 Arcs 84](#_Toc481700975)

[6.4 Examples : Circular Chart 87](#_Toc481700976)

[6.4.1 Code Review : Constructor 89](#_Toc481700977)

[6.4.2 Code Review : onSizeChnged() 90](#_Toc481700978)

[6.4.3 Code Review : onDraw() 90](#_Toc481700979)

[Chapter 07 Gradients 92](#_Toc481700980)

[7.1 Linear Gradients 92](#_Toc481700981)

[7.1.1 Example: Vertical Gradient 92](#_Toc481700982)

[7.1.2 Example : Horizontal Gradient 95](#_Toc481700983)

[7.1.3 Example : Diagonal Gradient 95](#_Toc481700984)

[7.1.4 Tile Mode 96](#_Toc481700985)

[7.2 Radial Gradients 98](#_Toc481700986)

[7.2.1 Example : CLAMP Mode 98](#_Toc481700987)

[7.2.2 Example:3D Ball 100](#_Toc481700988)

[7.2.3 Example : REPEAT Mode 102](#_Toc481700989)

[7.3 Sweep Gradients 103](#_Toc481700990)

[7.3.1 Example 103](#_Toc481700991)

[7.3.2 Custom Sweep Angle 105](#_Toc481700992)

[Chapter 08 Introduction to Coordinate Transformations 108](#_Toc481700993)

[8.1 Transformation Operations 108](#_Toc481700994)

[8.1.1 Translation 108](#_Toc481700995)

[8.1.2 Scaling 108](#_Toc481700996)

[8.2 Example : Triangle 109](#_Toc481700997)

[8.2.1 Transform 110](#_Toc481700998)

[8.2.2 Activity 111](#_Toc481700999)

[8.2.3 View 111](#_Toc481701000)

[8.2.4 Code Review : Constructor 113](#_Toc481701001)

[8.2.5 Code Review : onSizeChanged() 113](#_Toc481701002)

[8.2.6 Code Review : onDraw() 114](#_Toc481701003)

[8.2.7 Screenshots 114](#_Toc481701004)

[8.3 Example : Stock Chart 116](#_Toc481701005)

[8.3.1 Transform 118](#_Toc481701006)

[8.3.2 Chart Model 119](#_Toc481701007)

[8.3.3 Activity 120](#_Toc481701008)

[8.3.4 ChartView 120](#_Toc481701009)

[8.3.5 Code Review: Constructor 122](#_Toc481701010)

[8.3.6 Code Review: onSizeChanged() 122](#_Toc481701011)

[8.3.7 Code Review: onDraw() 123](#_Toc481701012)

[8.3.8 Screenshot 123](#_Toc481701013)

[Chapter 9 Transformation Matrices 124](#_Toc481701014)

[9.1 Transformation Operations 124](#_Toc481701015)

[9.1.1 Identity Matrix 124](#_Toc481701016)

[9.1.2 Translation 125](#_Toc481701017)

[9.1.3 Scaling 125](#_Toc481701018)

[9.1.4 Rotation 125](#_Toc481701019)

[9.2 Matrix Concatenation 125](#_Toc481701020)

[9.3 Matrix Class 126](#_Toc481701021)

[9.3.1 Constructor 126](#_Toc481701022)

[9.3.2 Concatenation Methods 126](#_Toc481701023)

[9.3.3 Coordinate Transformation 126](#_Toc481701024)

[9.4 Example : Triangle 127](#_Toc481701025)

[9.4.1 Code Review : Constructor 128](#_Toc481701026)

[9.4.2 Code Review : onSizeChange() 129](#_Toc481701027)

[9.4.3 Code Review : onDraw() 129](#_Toc481701028)

[9.4.4 Screenshots 129](#_Toc481701029)

[9.5 Example : StockChart 130](#_Toc481701030)

[9.5.1 Code Review : Constructor 132](#_Toc481701031)

[9.5.2 Code Review : onSizeChanged() 132](#_Toc481701032)

[9.5.3 Code Review : onDraw() 133](#_Toc481701033)

[9.5.4 Screenshot 133](#_Toc481701034)

[9.6 Current Transformation Matrix(CTM) 134](#_Toc481701035)

[9.6.1 Example 134](#_Toc481701036)

[9.6.2 Code Review : Class Variables 135](#_Toc481701037)

[9.6.3 Code Review : Constructor 135](#_Toc481701038)

[9.6.4 Code Review : onSizeChanged() 136](#_Toc481701039)

[9.6.5 Code Review : onDraw() 136](#_Toc481701040)

[9.6.6 Screenshot 136](#_Toc481701041)

[9.6.7 Fixed Example 137](#_Toc481701042)

[9.6.8 Code Review : Constructor 138](#_Toc481701043)

[9.6.9 Code Review : onSizeChanged() 139](#_Toc481701044)

[9.6.10 onDraw() 139](#_Toc481701045)

[9.6.11 Screenshot 139](#_Toc481701046)

[Chapter 10 Multithreading 140](#_Toc481701047)

[10.1 Processes and Threads 140](#_Toc481701048)

[10.2 Handler 141](#_Toc481701049)

[10.2.1 Example: Multithreaded Stock Chart 141](#_Toc481701050)

[10.2.2 Layout 144](#_Toc481701051)

[10.2.3 Activity 145](#_Toc481701052)

[10.2.4 ChartView 148](#_Toc481701053)

[10.3 SurfaceView 153](#_Toc481701054)

[10.3.1 Example: Progress Indicator 154](#_Toc481701055)

[10.4 Example: SurfaceView and Handler in the same Application 163](#_Toc481701056)

[10.4.1 MessageListener 164](#_Toc481701057)

[10.4.2 MessageHandler 164](#_Toc481701058)

[10.4.3 ChartSurfaceView 165](#_Toc481701059)

[10.4.4 ChartDataThread 169](#_Toc481701060)

[10.4.5 Layout 171](#_Toc481701061)

[10.4.6 Activity 171](#_Toc481701062)

[Index 174](#_Toc481701063)

[Android 2D Graphics with Canvas API 174](#_Toc481701064)

[Canvas API 174](#_Toc481701065)

[Intended Audience 175](#_Toc481701066)

# 序

雖然Android選用 Java 來當成開發語言，可是實際上也只是擷取了Java語言中有關的流程控制及資料處理等部份來使用而已，並不是全盤的採用。這主要是因為當初Android裝置實在是無法與真正的電腦比。由於硬體規格的限制，Android 避開了Java重量級的圖形操作介面及繪圖函數庫等等，而自行開發更適合行動裝置使用的套件。關於 Android 的圖形操作介面，市面上已經有太多的書籍著墨在這上面了，所以我們只會討論繪圖函數庫部分。

Android 目前提供了兩種繪圖函數庫來供開發者選用，分別是Canvas API 及OpenGL ES。這兩組 API 都有提供動畫的功能，主要的區別為對 2 D 與 3D 的支援程度不同。大致說明如下:

#### Canvas API

如果你所要開發的 App 不是對聲光效果要求極高的3D動畫手機遊戲，純粹只是要讓App有些與眾不同的美工元素，那這個較容易上手的繪圖的函數庫應個就是您的最佳選擇。其實Google對於這組繪圖函數庫並沒有一個正式的名稱，只因為其用來繪圖的主要物件名稱就叫做 Canvas，所以大部分的人就直接將其稱為Android Canvas API。其套件名稱為 **android.graphics，**所以程式撰寫時要加上 **import android.graphics. \*** 的指令。

Canvas API是個極有效率繪圖函數庫，主要應用在 2 D 繪圖及動畫。所以相較於專精3D繪圖的 OpenGL ES 來說 Canvas API算是個簡單易上手的 API。不過不要因此就小看Canvas API，這組 API 可不是那種僅僅只是提供了繪製文字、線條，方塊等等基本功能的玩具級 API，而是一組具有完整繪圖功能的的 API。此外由於Canvas API不需處理3D資料，且在Android 3.0之後又新增了硬體加速的功能，所以Canvas API有極高的執行效率，就算用來開發一款類似"超級瑪利歐"的遊戲都沒甚麼問題。

#### OpenGL ES

OpenGL ES 是Khronos集團專為嵌入式裝置所打造的 2D 及 3D 繪圖引擎，這是一組具備硬體加速的功能的高效能 API。OpenGL ES 可說是 OpenGL 的簡化版本，其名稱中的 ES 乃是 Embedded System 的縮寫。OpenGL ES 應用的範圍極廣，在多種嵌入式平台上都可發現其蹤跡，包括 Android 及 iOS 裝置等等。而在 Android 上的版本更是同時提供了 Java 及 C/C++ 兩種程式語言的 API，所以不管是用 Android SDK 或 NDK 都可用來開發具備OpenGL ES功能的 App。

**對本書讀者群的期望**

本書是專為有興趣學習如何使用 Android 繪圖功能的讀者所撰寫的。因此我們會期望讀者能夠有一些 Java 的知識及 Android SDK 的使用經驗。此外我們些希望讀者已經具備 Android 開發的基本觀念，例如Activities、Views 等元件的使用方式，並且熟悉如何測試自己所設計的 Android App。

本書中使用許多的程式範例來展示如何繪製基本圖形，解釋座標及位移的基本觀念，並對兩種常用來設計具多執行緒能力繪圖程式的開發模式加以探討。

**本書內容簡介**

**第一章 Canvas API 簡介**。本章簡單的描述 Canvas API 的基本作用，介紹這組 API 中最常用到的幾個類別，例如: Canvas、Paint、Path、Typeface 及Matrix 等等。此外也會稍微聊到邏輯座標及裝置座標之間的差異。在實作部分則是提供了一個簡單的範例來顯示Android裝置上的螢幕可供程式繪圖使用的尺寸大小。

**第二章 座標系統。**在這一章我們會認識到幾個 Android 2D 繪圖常會用到的幾個座標系統，例如笛卡爾座標(Cartesian coordinate)、裝置座標(device coordinate)、邏輯座標(logical coordinate)等等。並且會帶出全域座標(global coordinate)及區域座標(local coordinate)的觀念。

**第三章 基本圖形**。本章會探討直線(line)及路徑(path)這兩個基本圖形**(Drawing Primitives)**的屬性，例如colors、paint styles、cap、join types 以及 dash path effects 等等的作用。

**第四章 文字的處理。**這章主要是探討 Canvas API 中有關於文字的顯示方式。我們將會學到如何選用不同的字體，如何指定文字的排列與對齊格式，以及如何設定文字的 size、color、shadow 等的屬性。

**第五章 矩形。**這章會練習矩形(rectangle)的繪製，並將其實際應用在 bar chart 的製作上。

**第六章 圓形、橢圓及弧形。**在這一章中會學到有關於圓形(circle)、卵形(oval)及弧形(arc)的繪製方式。並將其實際應用在統計圖表或進度表的繪製上。

**第七章 漸層。**這一章主要探討漸層的繪製方式及其應用。

**第八章 座標轉變。**這章主要討論座標轉變 (coordinate transformation)背後所使用到的數學觀念，以及在不使用矩陣(matrix)的狀況下如何做座標轉變。我們會使用兩個簡單的圖形來展示座標如何轉變。

**第九章 轉變矩陣。**在這一章我們將學習轉變矩陣(transformation matrices)的用法。而在我們在上一章中所做了兩個範例也會修改成使用轉變矩陣來處理座標轉變的步驟。此外我們也會說明為何不建議更改目前轉變矩陣(current transformation matrix, CTM)。

**第十章 多執行緒。**在這章裡我們會討論撰寫 Android 繪圖程式時，常會用到的兩種多執行緒(multithreaded)實作方式。第一種方式採用 Handler 及 Message 這兩個類別來作為執行緒之間的溝通管道。第二種方式則是使用了 SurfaceView 這個類別來實作多執行緒繪圖。當然我們之前所做的股市圖(stock chart)範例也會修改成使用這兩種實作方式的多執行緒版本。

**聯繫資訊:**

如果你發現我們的文字描述或者是範例程式碼有任何需要修正之處，或者是有其他的建議事項。歡迎與作者聯繫。

# 第一章 Canvas API 簡介

就如同我在序文中所提過的，Google 並沒有給予**android.graphics** 套件中的類別一個正式的名稱。所以程式設計師們通常就稱之為 Canvas library 或是 Canvas API。所以在後續的內文中不管是 Canvas library 或是 Canvas API 我們指的都是同一組類別的集合。

Canvas API 是Android 內建的一組 2D 繪圖函式庫，雖然表面上看來 Canvas API 僅僅只提供了幾個繪製文字、線條、矩形、圓形等等基本圖形的方法。事實上藉由這些基本圖形的排列組合，我們就可在 Android 的顯示畫面上繪製各式各樣的複雜圖形。Canvas API 中所有的類別都集中放置在 android.graphics 套件之下，其中有幾個特別重要的類別我們必須先介紹，例如 Canvas、Paint、Path 等等。在認識了這幾個類別後會帶大家實作一個顯示 Canvas 繪圖區域尺寸的範例。

## 1.1 Canvas

記得唸小學時每次上美術課，老師第一件事就是確認我們有沒有帶圖畫紙。因為不管是筆還是顏料都可以跟其他同學借或者是共用。唯獨圖畫紙是無法共用的，沒帶那基本上那一堂美術課就是浪費了。要使用 Android Canvas API，當然也必須要有個類似圖畫紙的物件來讓你將圖形繪製上去。而 Canvas 這個類別的定位就是 Android 的圖畫紙。有了 Canvas 物件實例(instance)後我們就可以利用其內建的一些 method 來讓將文字、線條、矩形、圓形等等基本圖形繪製在上面。取得 Canvas物件實例最簡單的方法就是直接複寫 View 類別中的 onDraw() method。程式碼如下所列:

|  |
| --- |
| public class MyView extends View  {  @Override  public void onDraw(Canvas canvas)  {  //...程式碼省略  }  } |

細心的您可能已經注意到onDraw() 的參數就是個Canvas物件，所以你不用實做的步驟就可以直接取得Canvas物件實例。每當一個使用 View 來當顯示畫面的 App 被啟動時、螢幕方向改變時或者是有觸控的動作時，Android 都會重新呼叫這個 method 來更新螢幕的顯示內容。當然若您必須要在程式中強制更新螢幕內容時，也可透過呼叫invalidate() 或這是 postinvalidate() 這兩個 method 來達成目的。在 Android 系統中實際用來處理螢幕顯示物件幾乎都是繼承 View 這個類別而來。如果我們將Canvas比喻成圖畫紙的話，那 View 就相當於是放置圖畫紙的畫框了，所以我們才會將Canvas物件當參數傳入 View 類別的onDraw()中來進行實際的繪圖動作。

## 1.2 Paint

Paint 這個類別可用來指定繪圖基本元素的風格及顏色等等的資訊。Paint 內建了不少好用的 method讓我們在繪圖上更能得心應手，例如量測文字的尺寸大小等等。所有的繪圖 method都會使用 Paint類別來當參數，因此很多程式設計師們通常都會在 constructor 中先建立一些 Paint類別實例，然後在後續的 onDraw()中重複使用這些這些Paint類別實例。如下列程式碼所示:

|  |
| --- |
| public class MyClass extends View {  private Paint paint;  public MyClass(Context context) {  super(context);  paint = new Paint();  paint.setColor(Color.BLACK);  }  @Override  public void onDraw(Canvas canvas) {  canvas.drawLine(10,10,100,100, paint);  }  } |

## 1.3 Path

Path 這個類別可用來繪製直線、曲線或者是其他圖形的外框。Path幾乎可以說是繪製其他更複雜圖形的基礎。我們可以將 Path 設定成開放或者是封閉，藉以構成特定的圖形。舉例來說，由三個點所組成的 Path 如果設定成開放，那就只是個曲折的線條，可是設定成封閉後就變成一個三角形了。通常為了讓繪圖的顯示速度更為流暢，我們往往會先將所需要 Path 實例都建立好，然後在 onDraw() 直接將所有現成的 Path 實例一起繪製上去。

簡單的Path 實例建立與繪製步驟如下程式碼所示:

|  |
| --- |
| public void onDraw(Canvas canvas) {  Path path = new Path();  path.moveTo(10, 10);  path.lineTo(100, 100);  canvas.drawPath(path, paint);  } |

## 1.4 Typeface

Typeface 這個類別主要用來指定所使用的文字的外觀及字體(typeface)。以下的程式碼展示如何指定所用的字型為serif 粗體字。

|  |
| --- |
| Typeface font = Typeface.create("serif", Typeface.Bold); |

關於字型這個主題我們在第四章有更詳細的探討。

## 1.5 Matrix

Matrix 這個類別內含一個用來處理座標轉換的 3x3矩陣(matrix)。

關於座標及座標轉換這個主題，本書後續的章節會有較詳細且實用的說明。第二章就會介紹一些常用座標的基本知識。而在第八章則會開始探討座標轉換的數學觀念及其實踐的步驟，讓您在不使用現成 Matrix 類別的狀況下也能做座標轉換。第九章則是介紹如何利用 Matrix 類別來實現座標轉換。

## 1.6 範例: CH01\_Canvas

現在我們來實作一個簡單的App，這個 App 只是將 Android 裝置螢幕可供繪圖使用的View Size以高度寬度各有多少畫素的格式在螢幕的正中央顯示出來。

如果您完全沒有任何的 Android 程式開發經驗，我們建議您可考慮先花點時間去學一些基本的知識再回頭來看本書。希望您至少已經學會如何建置 Android 的開發環境及撰寫幾個 Hello World 等級的 App。因為本書雖然定位是入門書籍，不過所指的是Android繪圖入門，而不是 Android 開發入門，所以並沒有包含這些程式開發的基礎內容。若有需要，請讀者自行參考其他入門書籍。

### 1.6.1 MainActivity

請執行 Android Studio，然後建立一個名為 CH01\_Canvas 的專案。接著照著範例程式碼輸入內容後就可以進行測試。其中 package 及 activity 的命名可以依您的喜好自行變更，不見得非得要跟範例一模一樣才行。

|  |
| --- |
| **package** com.janaslee.ch01\_canvas;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle;  **public class** MainActivity **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  *//setContentView(R.layout.activity\_main);* MyCanvas view = **new** MyCanvas(**this**);  **this**.setContentView(view);  } } |

這裡我們要做的事情很簡單，只是將程式碼中的

setContentView(R.layout.activity\_main);

改成

MyCanvas view = **new** MyCanvas(**this**);

而已。

這行程式碼現在會出現錯誤訊息，因為MyCanvas 是我們接下來即將動手做的一個 Class，目前還不存在，所以 Android Studio 找不到。這行程式碼的作用是將 MainActivity 的顯示畫面改成我們自訂的 MyCanvas 實例。原本 MainActivity 的顯示畫面是activity\_main.xml。

### 1.6.2 MyCanvas

接下來請新增一個名為 MyCanvas 的 Java Class，由於要拿來當顯示畫面，所以這個 Class 必須繼承 View，且需要覆寫其中的幾個 method，如下所列:

|  |
| --- |
| **package** com.janaslee.ch01\_canvas;  **import** android.content.Context; **import** android.graphics.Canvas; **import** android.graphics.Color; **import** android.graphics.Paint; **import** android.view.View;  **public class** MyCanvas **extends** View {  **private int width**;  **private int height**;  **private** Paint **paint**;  **public** MyCanvas(Context context) {  **super**(context);  setBackgroundColor(Color.***WHITE***);  **paint** = **new** Paint();  **paint**.setColor(Color.***BLACK***);  **paint**.setTextAlign(Paint.Align.***CENTER***);  }   @Override  **protected void** onSizeChanged(**int** w, **int** h, **int** oldw, **int** oldh) {  **width** = w;  **height** =h;  }   @Override  **public void** onDraw(Canvas canvas) {  **float** textSize = Math.*min*(**width**, **height**) \* 0.12f;  **paint**.setTextSize(textSize);  **float** x = **width** \* 0.5f;  **float** y = **height** \* 0.5f;  String str = **"w="** + **width** + **", h="** + **height**;  canvas.drawText(str, x, y, **paint**);  } } |

將這個 Class 存檔後，原本MainActivity 中的錯誤訊息就會消失。

### 1.6.3 程式碼解說 : onSizeChange()

現在我們來看看這些程式碼的功用。首先是程式一開始所宣告的width 和 height兩個類別變數(class variables)，這用來存放目前螢幕上可用繪圖尺寸的寬度及高度各是多少像素(pixel)。

|  |
| --- |
| private int width; private int height; ... @Override protected void onSizeChanged(int w, int h, int oldw, int oldh) {  width = w;  height =h; } |

其後有個名為 onSizeChanged() 的 method，這是覆寫自其父類別 View 的 method，所以要加上 @Override 的宣告。當 App 中的 View 類別剛被啟動時，這個 method 的內容就先會被執行一次，讓 App 能夠根據正確的尺寸來繪製畫面。接下會被當 Android 裝置螢幕的顯示尺寸有變動時，系統就會自動再次呼叫這個 method 來讓更新顯示的內容。最常見的狀況就是旋轉螢幕的情形，由於絕大多數的 Android 裝置都不採用正方形的螢幕，所以只要螢幕一旋轉，寬高尺寸就會變動，顯示的內容難免要跟著調整。

這個 method 的四個參數分別代表螢幕可繪圖區域的高度及高度的新舊值。目前我們只會用到新的值，這個 method 中的程式碼內容就是將高度及高度的新值分別存到 width 和 height這兩個類別變數中。

其實 onSizeChanged() 是個處理與螢幕尺寸及顯示方向相關資料的好地方。許多的計算，例如物件座標轉換、位移、縮放等等都可集中在這裡先處理好。等真正繪圖時直接取用，這樣可讓繪圖過程更加流暢。不會有因為要等運算結果而出現停頓的狀況。

### 1.6.4程式碼解說: Paint

Paint 這個類別用來指定繪圖所用的顏色及筆觸型態等等資訊。你可以將其想像成我們在上真正美術課時時所用的畫筆及顏料的組合。我們可以用 new 指令來建立一個 Paint 的物件實例，然後再陸續的指定所要的顏色、字形、筆觸等等屬性。在本範例中我們僅僅只是指定了顏色、文字對齊方式及文字大小等三種屬性而已，事實上還有更多的屬性可以使用，在後面的章節會有更詳盡的介紹。

|  |
| --- |
| private Paint paint;  ...  paint = new Paint();  paint.setColor(Color.*BLACK*);  paint.setTextAlign(Paint.Align.*CENTER*);  ...  paint.setTextSize(textSize); |

為了讓執行效率更佳，我們將宣告了一個名為 paint 的 Paint 實例，並將其放在 method 的外面，使其成為 MyCanvas中每一個 method 都能存取的類別變數。然後在建構函數(constructor)中將paint 初始化而不是在 onDraw() 中初始化。之所以這樣做是因為若我們將物件的宣告及初始化的動作放在onDraw() 中，則每次畫面更新時，這些物件都要重新產生及初始化一次。這對行動裝置來說是很浪費資源的。在上面的程式碼中我們對 pain 設定了三個屬性，分別是將顏色指定成黑色，

|  |
| --- |
| **paint**.setColor(Color.***BLACK***); |

將文字對齊的方式指定為置中對齊，

|  |
| --- |
| **paint**.setTextAlign(Paint.Align.***CENTER***); |

以及設定文字的大小尺寸。

|  |
| --- |
| **paint**.setTextSize(textSize); |

### 1.6.5程式碼解說: onDraw()

就繪圖而言onDraw()可說是 View 類別中最重要 method。從 method 前面的 @Override 宣告我們就可判定onDraw()也是從 View 類別中繼承覆寫而來。通常會在一些基礎類別就定義的 method 往往是Android作業系統會主動去呼叫的。這個 method 也不是例外，每當有必要更新顯示畫面的情況發生時，例如螢幕的方向改變、使用者有觸控的動作、有內容必須加入畫面或從畫面移除時，Android作業系統就會重新呼叫onDraw()。此外細心的您可能已經發現 onDraw() 的參數恰好就是個 Canvas 實例，這就表示Android 作業系統呼叫 onDraw() 就會順便將 Canvas實例傳入，讓您可以直接進行繪圖的動作。

|  |
| --- |
| @Override **public void** onDraw(Canvas canvas) {  **float** textSize = Math.min(width, height) \* 0.12f;  paint.setTextSize(textSize);  **float** x = width \* 0.5f;  **float** y = height \* 0.5f;  String str = **"w="** + width + **", h="** + height;  canvas.drawText(str, x, y, paint); } |

我們使用 Canvas 類別的 drawText() 來將字串的內容繪製在畫面上。這個 method 的參數有四個，分別是所要顯示的字串、X軸及Y軸的座標及 paint 實例。通常我們會將所要的筆觸及顏色等等資料設定在paint 之中。

|  |
| --- |
| canvas.drawText(str, x, y, paint); |

為了讓顯示畫面在各種裝置上都能夠看起來一致，我們會動態的計算繪圖所需的一些數值，比如文字的大小尺寸、X軸及Y軸的座標等等。首先我們先將文字的大小尺寸設定為畫面寬度的 12%。

|  |
| --- |
| float textSize = width \* 0.12f; |

然後將顯示文字的X軸及Y軸的座標設為寬高的中間，也就是50%的位置。

|  |
| --- |
| **float** x = width \* 0.5f;  **float** y = height \* 0.5f; |

之所以要這樣做是因為 Android 裝置的螢幕尺寸規格不一，從畫面不到兩吋 240\*240像素的智慧型手錶到超過50吋 4K像素的 Android TV都有。若是直接使用真實像素來繪圖就有可能在某個裝置上看起來很完美的畫面換到另一個裝置上卻是慘不忍睹。

使用百分比來代替真正數值的作法其實已經是個簡單的座標轉變(coordinate transformation)實作了。將寬度與高度的範圍定為 0% 到 100%(或者是 0 到 1)就是邏輯座標系統(logical coordinate system)的基本觀念。透過這種方式我們在程式設計時，就可以不用為實際執行App的裝置所用的螢幕尺寸而傷透腦筋了。

在本例中我們將字體大小設定成寬度的12%(0.12)，當App執行時會乘以由onSizeChange()所偵測到的實際寬度來得到字體大小的尺寸了。同樣的顯示文字的邏輯座標設定為寬度與高度的50%(0.5)，當App執行時再計算裝置螢幕正中間的真實座標。這樣的動作就稱為邏輯座標對裝置座標的轉換，這個動作的名稱乍看之下挺唬人的，不過實際上也就只是將百分比(percentages)乘上像素而已。

座標轉換的內涵當然不會這麼陽春，後續的章節我們會對幾種常用座標轉換做更進一步的介紹。

### 1.6.6 螢幕截圖

現在請執行這個App，您就可以在螢幕中央看到目前所用的 Android 裝置螢幕可供程式繪圖使用的寬度及高度分別是多少像素。這組數字可能與您目前所用裝置硬體規格書上所記載的數字稍有差異，不用驚訝這是正常的現象。因為Android 裝置的螢幕除了顯示 App 的畫面外，在螢幕的上方通常也會有一些保留給作業系統顯示圖示及資訊的區域，此外目前大多數的廠商在製造Android 裝置時已經逐漸取消真實按鍵，而改用螢幕上的虛擬按鍵代替，這些虛擬按鍵也會占用螢幕上的可用區域。在 Figure 1.3 中我們可以看到原本解析度為 800 \* 1280 的螢幕因為下方已有三個虛擬按鍵，所以可供繪圖使用的區域只剩下 800\*1098。您可以旋轉螢幕方向及使用不同的裝置來檢視其寬度及高度的可用像素。

Figure 1.1 及 Figure 1.2 展現的是原本解析度為 480 \* 800 的手機螢幕在縱向與橫向時的可用像素。

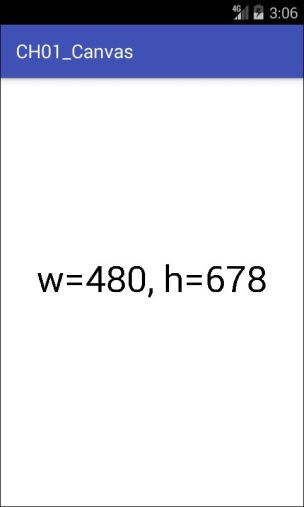


Figure 1-1.

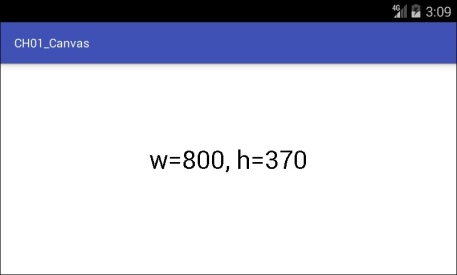


Figure 1-2.

這個手機採用的作業系統是Android 4.2而且有真實的按鈕。所以在螢幕的下端並沒有 System Bar 及虛擬按鍵，不過螢幕的上端依然有Action Bar 及程式標題欄(window title)。這些系統的顯示區多多少少會占用掉一些顯示空間。在螢幕為縱向時看起來可能還好，可是當你把螢幕轉成橫向時，這些區域所占的比率顯然就有點大了。所以這種低解析度的螢幕較不利在橫向時顯示完整資料，因為就算沒有虛擬按鍵的情況下也已經損失了超過20%的顯示空間了，高度就只剩下307像素可用了。

Figure 1-3 及 1-4分别展現真實解析度為 800 \* 1200 的平板螢幕在縱向與橫向時的可用像素。

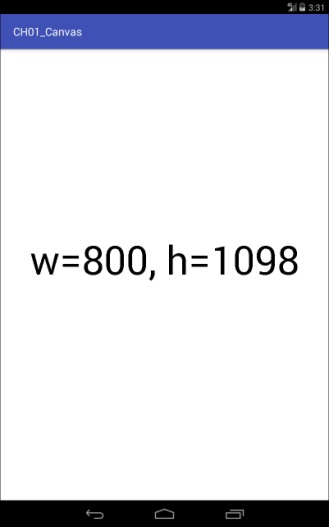


Figure 1-3.



Figure 1-4.

這個平板所採用的作業系統是 Android 4.3 除了螢幕頂端有Action Bar 及程式標題外，螢幕的下端也有放置虛擬按鍵的System Bar。所以被作業系統占用像素較多，不過由於螢幕的解析度較高，所以被系統占用的比率雖然同樣也是超過 20%。可是高度仍有 618 像素可用，對資料的顯示影響就沒那麼大了。

不過行動裝置的螢幕本來就不大，所以很多程式設計師會對顯示空間斤斤計較。在開發 APP 時往往會將螢幕頂端的Action Bar及程式標題隱藏起來，這樣就可釋放出不少可用的顯示空間，特別是有必要使用橫向的顯示畫面時。

開發 Android App 時我們根本就不可能去預測或規定使用者將來所會使用的Android裝置螢幕的尺寸及解析度。因此繪圖時千萬不要直接指定像素的真實座標，而是要改用邏輯座標轉換的方式。也就是僅量以螢幕高度及寬度的百分比來間接的指定繪圖物件的尺寸。

螢幕規格種類太過複雜可能是 Android 開發人員最令人困擾的事了，關於這一點，開發其他作業系統程式的人員可能會覺得幸福多了。PC就不用說了，就算是 iOS 系統其螢幕的規格加一加也沒幾種，所以根本就不用去為邏輯座標轉換的問題傷腦筋。接下來我們就要介紹一下常用的幾種座標了。

# 第二章 座標系統

對使用Illustrator 或 Corel Draw這類現成電腦繪圖(computer graphics)軟體的美工人員來說，座標轉換(coordinate conversion)這個議題似乎不太可能會遇到，因為這些麻煩事通常已經由程式設計師代為解決了。所以大多數的美工人員可能從來都沒有意識到電腦螢幕座標的Y軸跟我們在數學課所學的座標的Y軸方向其實是相反的。

但是對於一個程式設計師來說，座標系統(coordinate system)之間的轉換是學習電腦繪圖無法避免的一個基本議題。因為不管你要繪製什麼，只要座標不對圖形就無法正常顯示在螢幕上。因此座標轉換可說是初學電腦繪圖程式設計最先要理解的觀念，而身為 Android 程式設計師的您，除了要知道如何處裡螢幕座標與數學座標的轉換外，還要再加上與之前提過的邏輯座標之間的轉換。

簡單的說，座標系統就是一種描述一個點甚至是一個物件在二維平面或三維空間所在位置的方法。相信大家在學校的數學課應該都已經學會點、線、面的座標描述方式。後面的描述應該不至於覺得太過抽象。

## 2.1 笛卡爾座標系統

笛卡爾座標系統(Cartesian Coordinate System)可以看成是我們在學校數學課時所學的直角座標系統(rectangular coordinate system)，這可說是幾何學(geometry)及電腦繪圖(computer graphics)的基礎。笛卡爾座標系統使用兩條成直角相交的參考線(perpendicular reference lines)來做為座標軸(coordinate axes)，水平線稱為 X軸(X axis)，垂直線稱為 Y軸(Y axis)。兩條軸線的交會點稱為原點(origin)。

座標用來描述平面上某個點的所在位置，其格式為(x,y)，就是用小括號將兩個數字圍起來並以逗點分隔。座標的第一個數字稱為 X 座標，用來表示沿著 X 軸從原點移動的距離，第二個數字稱為 Y 座標，用來表示沿著 Y 軸從原點移動的距離。X 座標在原點右邊的為正數，在原點左邊的為負數；Y 座標在原點上方的為正數，在原點下方的為負數。例如某個點的座標為 (2,3) 表示位於原點右邊2單位，上方3單位之處。Figure 2-1 為笛卡爾座標中兩個點的座標及其位置，Point A 在 (-0.7, -0.5)，Point B 在 (0.5, 1)。

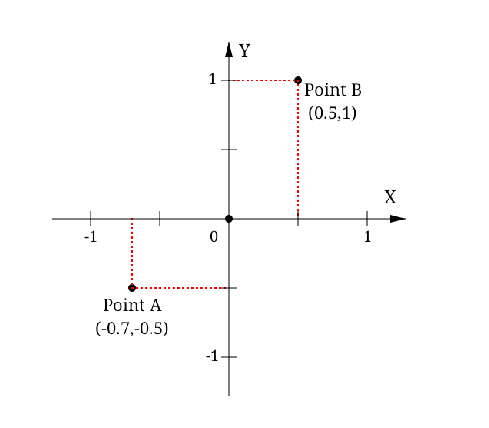


Figure 2-1.

## 2.2 裝置座標系統

電腦、手機或平板等等所用的螢幕都屬於光柵設備(raster devices)，也就是說這類的裝置的顯示畫面是由許許多多各自獨立的像素(pixel)所組成的，這也是螢幕上可以控制顏色與亮度的最小單位，pixel這個字是從 picture及element這兩個單字合併而來，用來表示”圖像元素”的概念。單位面積內的pixel越多代表解析度越高，所顯示的影像就越細緻，以目前主流的Full HD螢幕來說，其規格為 1920 \* 1080，所以這種螢幕上就有2073600各自獨立的pixel，每一個pixel的顏色與亮度都可以不同。如果將螢幕局部放大就可看出每個像素通常由三個小點或三個小方塊所組成，這三個小點分別對應到光的三原色---紅(red)、綠(green)、藍(blue)。透過RGB三原色的強弱組合讓像素可以呈現不同的顏色與亮度。而螢幕的顯示區就是由pixel以二維格柵(two-dimensional grid)的方式所組成，例如 Figure 2-2 所示就是一個 1280 x 800 的螢幕 pixel 排列方式。

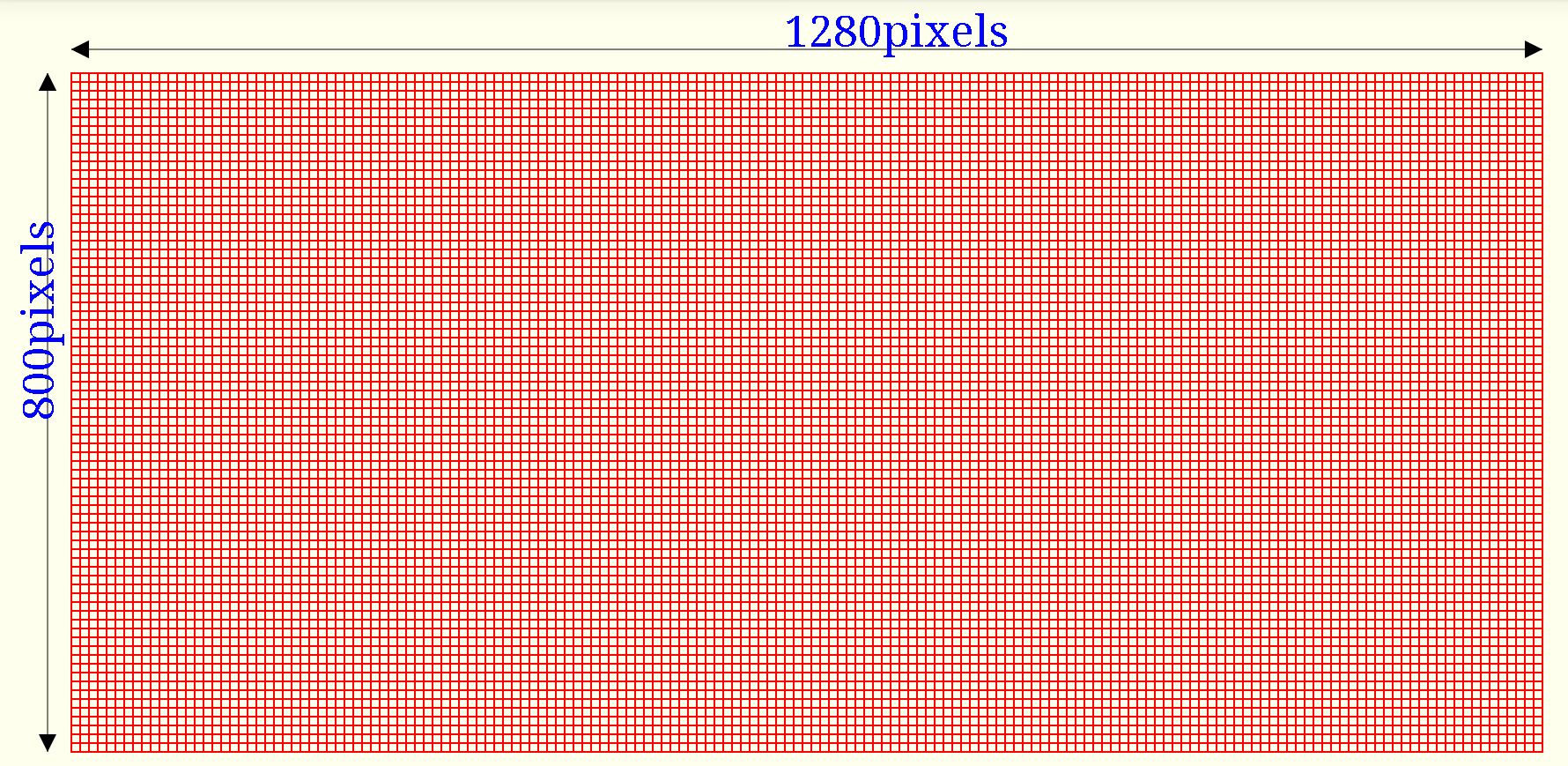


Figure 2-2.

螢幕所使用的座標又稱為裝置座標系統(**device coordinate system**)，基本上這可以當成是一個以 pixel為距離單位的笛卡爾座標，不過有三個重大的差異必須要特別注意。第一個差異是裝置座標的原點(origin)改成固定放在左上角。第二個差異是 Y 軸的正負方向反轉，改成原點向下為正數向上為負數。第三個差異是裝置座標中X 軸與Y 軸的最大值直接受限於硬體裝置的規格。目前包括Android Canvas API在內的許多繪圖函數庫都是採用裝置座標來處理繪圖資料。Figure 2-3 使用裝置座標來表示一個1280 x 800 pixel的螢幕。

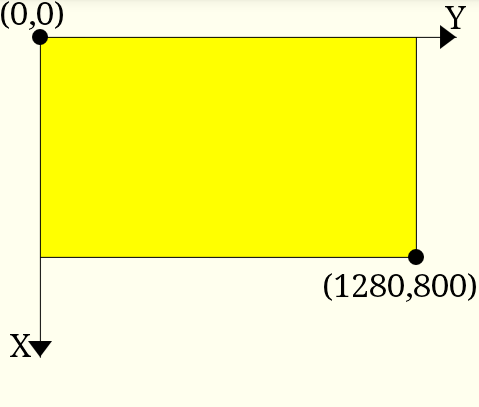


Figure 2-3

由於目前 Android 裝置的螢幕規格眾多，所以在程式中使用裝置座標系統會有個困擾，那就是無法預測將來 App 執行時所會遇到的螢幕規格。因為在某種解析度呈現完美的畫面，搬到另外一種解析度可能就慘不忍睹。例如在 Full HD 裝置上的所製作的圖像若不經過任何處理直接搬到 480x320 pixels結果可能只能看到一個角落。螢幕規格太多也是Android軟體開發過程中最令人苦惱之處。相較之下，其他的系統，例如 iOS的開發者就幸福多了，iPhone出了這麼多代，可是螢幕規格也不過十來種。

## 2.3 邏輯座標系統

邏輯座標系統( Logical coordinate system)是一種專供程式設計等等特定用途，與硬體裝置無關，採間接方式存取螢幕資訊的座標系統。基本上其底層還是笛卡爾座標系統，只不過座標的距離單位改用邏輯單位而非真實點數。邏輯座標的 X 軸與 Y 軸的邏輯單位可以是不同的。舉例來說，如果你想要繪製一個用來展現公司銷售狀況的業績圖，那水平的 X 軸可能就會以天為單位，而垂直的 Y 軸所用的單位可能就是銷售數字，如 Figure 2-4 所示。

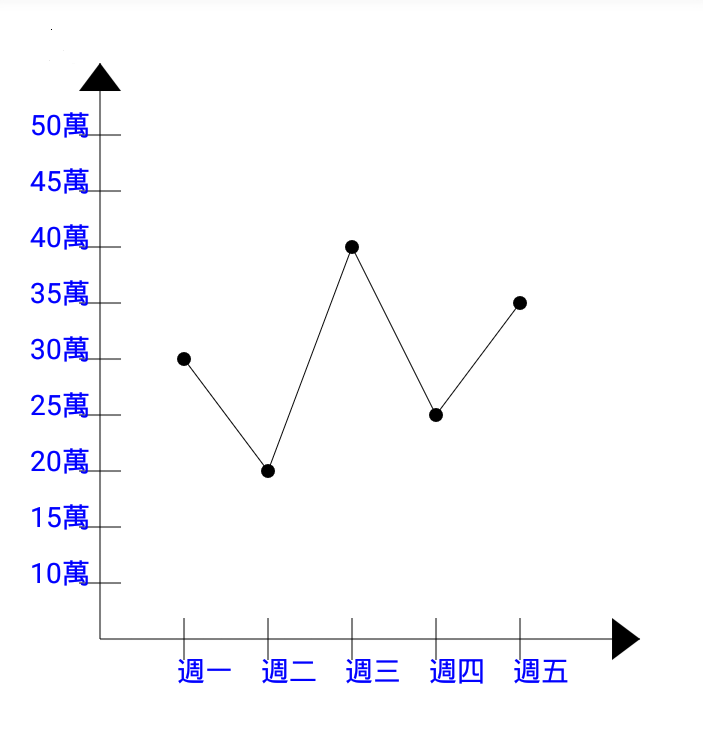


Figure 2-4.

## 2.4 全域與區域座標系統

如果你有過任何電腦繪圖軟體的使用經驗，那相信您對全域(global)與區域(local)座標系統的概念應該不會覺得太過突兀而難以接受。

大部分的作業系統，諸如Microsoft Windows、Linux、MacOS等等都可以在螢幕上同時顯示多個視窗。對作業系統來說，整個螢幕的座標就是所謂的全域座標。而每一個視窗都有自己的座標來顯示其內容，這些視窗個別使用的座標就是區域座標。假設你使用繪圖軟體同時打開兩張圖片，然後分別在這兩張圖片中距離下邊界及左邊界各 2 公分之處各畫一個小紅點。就區域座標而言放置小紅點的標誌的座標可能都同樣是(2,2)，可是就全域座標而言這可是螢幕上兩個完全不同的座標。不過使用者通常只要處理區域座標就可以，區域座標與全域座標的轉換依般來說會由作業系統代勞。所以就算使用者移動視窗，其區域座標依然可以維持不變，而作業系統會自動將其轉換成正確的全域座標。

在Android的繪圖 API中，View是用來放使用者操作介面元件的基礎類別，這個類別通常會占用螢幕上的一個矩形的(rectangular)區域來做為顯示之用，並且會使用自己的區域座標系統來進行繪圖的動作。同樣的 Android 作業系統會自動將 View 所使用的區域座標轉換成全域座標座標然後顯示在螢幕上。。

Figure 2-5 是一個全域與區域座標系統關係的示意圖。在這個例子中全域座標代表的是一個顯示規格為 1280x800 pixels 螢幕的裝置座標。

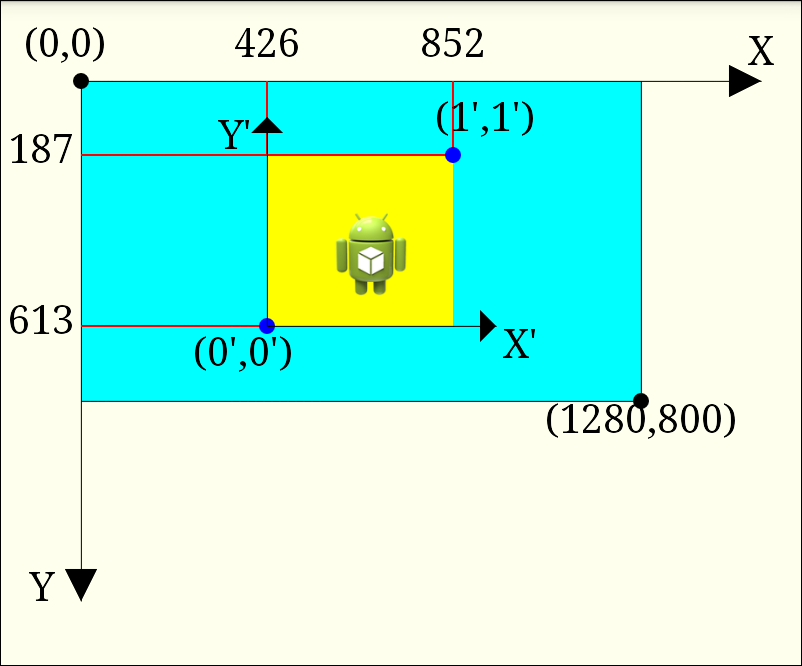


Figure 2-5.

在這個例子中全域座標對應的是一個顯示規格為 1280X800 pixels的螢幕的裝置座標。這個全域座標的原點是在螢幕的左上角，而畫面中的程式視窗則是使用自己的區域座標系統。這兩種坐標系統最大的差異就是 Y軸的方向相反，全域座標Y軸向下數字遞增，而區域座標的Y軸則是向上數字遞增。區域座標系統的原點實際上位於全域座標的(426,613)位置。此外所用的標示單位也不同，全域座標的標示單位就是螢幕像素，而區域座標的標示單位相當於裝螢幕的462個像素。所以區域座標的(1,1)實際上是位於全域座標的(852,172)的位置。

最後還有一點要澄清的是全域座標不一定就是等於裝置座標，比如現在許多的裝置都會保留螢幕的一部分來當成虛擬按鍵或訊號顯示之用，在這種狀況下全域座標可就比裝置座標小很多了。

# 第三章 基本圖形

在本書中基本圖形(Drawing primitives)指的是諸如線條(line)、矩形(rectangle)、圓形(circle)、橢圓形(ellipse)及路徑(path)。藉由這些基本圖形的組合就可以繪製出更複雜的圖形。

Canvas 物件內建了一些drawXXX() 的method來繪製這些基本圖形。每一種基本圖形都有一組用來設定其外觀樣式的屬性來搭配。舉例來說，line就可透過color 及 width 兩種屬性來設定其顏色及寬度。這些屬性是另外儲存在 Paint 物件中，因此在Android裝置上繪圖時，通常會先產生一個 Paint 的物件實例，然後設定一些屬性值，最後再將這個 Paint 物件當成參數傳入 Canvas 的drawXXX() 繪圖 method 中進行繪圖的動作。如下列程式所示:

|  |
| --- |
| // Create Paint object  Paint paint = new Paint();  paint.setColor(Color.BLUE);  paint.setStrokeWidth(80);  …  //Draw a line  canvas.drawLine(100, 100, 700, 700, paint); |

在這一章我們主要探討 line 及 path 這兩種基本圖形，並藉由調整color、paint styles、cap、join type及dash path 等屬性來改變這些基本圖形的外觀。

## 3.1 Lines

Canvas 中有不少的method 都可用來繪製線條，最常被使用的大概就是 drawLine()，其內容如下所列：

|  |
| --- |
| Void drawLine(float startX, float startY, float endX, float endY, Paint paint) |

這個 method 會在(startX, startY)與(endX, endY)這兩個點之間繪製一條指定樣式的線段。

This method draws a line segment between two points (startX, startY) and (endX, endY) with the specified paint. The Paint object provides different line attributes or styles such as line color and width.

### 3.1.1 Example

In this example we will draw a thick blue line shown in Figure 3-1.

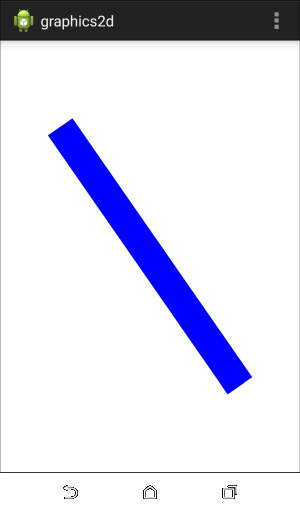


Figure 3-1.

In most example in this book we will use the same GraphicsActivity class introduced in Chapter 1 and only change the name of the content view. The content view of this example is shown below.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.view.View;  **public** **class** LineView **extends** View  {  **private** Paint paint;  **private** **int** width;  **private** **int** height;    **public** LineView(Context context) {  **super**(context);  setBackgroundColor(Color.***WHITE***);  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setColor(Color.***BLUE***);  }    @Override  **public** **void** onDraw(Canvas canvas)  {  canvas.drawLine(width\*0.2f, height\*0.2f,  width\*0.8f, height\*0.8f, paint);  }    @Override  **protected** **void** onSizeChanged(**int** w, **int** h, **int** oldw, **int** oldh)  {  width = w;  height = h;  paint.setStrokeWidth(Math.*min*(w, h)\*0.1f);  }  } |

Let’s review the code.

### 3.1.2 Code Review: Constructor

First, we set the view background color.

|  |
| --- |
| setBackgroundColor(Color.WHITE); |

Then we create a Paint object.

|  |
| --- |
| paint = new Paint(Paint.ANTI\_ALIAS\_FLAG); |

The Paint.ANTI\_ALIAS\_FLAG attribute is used to make lines look smoother. Usually it is a good idea to enable this attribute.

Finally we set the line color.

|  |
| --- |
| paint.setColor(Color.BLUE); |

### 3.1.3 Code Review: onSizeChanged( )

First we stored the view width and height. These values are used in onDraw() method to calculate line coordinates.

|  |
| --- |
| width = w;  height = h; |

Then we set the line width to 10% of the short side of the view. We used a percentage to make sure that this example looks the same on different Android devices.

|  |
| --- |
| paint.setStrokeWidth(Math.min(w, h)\*0.1f); |

### 3.1.4 Code Review: onDraw( )

In this method we drew the line with the paint created in the constructor.

|  |
| --- |
| canvas.drawLine(width\*0.2f, height\*0.2f,  width\*0.8f, height\*0.8f, paint); |

We used ratios to calculate start and end coordinates of the line.

## 3.2 Colors

Canvas library uses ARGB model to describe a color. In this model each color is represented by four components or channels: alpha, red, green and blue. Each component can have a value from 0 to 255.

### 3.2.1 Alpha Channel

The alpha channel describes opacity. The value of 0 corresponds to a fully transparent (invisible) pixel, whereas the value of 255 represents a fully opaque pixel.

### 3.2.2 Integers

Colors are represented by integers. As you may know, in Java, integers are 4 bytes long. Each byte is used to store one of four components of a color in the following order from the most significant byte to the least significant: alpha, red, green, blue.

An integer color value can be easily expressed using 8 hexadecimal digits with each pair of digits representing the values of the alpha, red, green and blue channel, respectively. For example a blue opaque color has the following hexadecimal value:

|  |
| --- |
| int blue = 0xFF0000FF; |

### 3.2.3 Color Class

The Color class provides several helper methods for creating color integers, such as

|  |
| --- |
| paint.setColor(Color.rgb(0,0,255)); |

and

|  |
| --- |
| paint.setColor(Color.argb(255,255,255,255)); |

It also defines several constants, such as WHITE and BLUE, we used in the previous example:

|  |
| --- |
| setBackgroundColor(Color.WHITE);  …  Paint.setColor(Color.BLUE); |

### 3.3 Cap Styles

As you can see from Figure 3-1, the line has flat edges and looks like a rectangle. Usually when you draw a line, you don’t want it to look like that. Fortunately, default decoration of line ends can be changed. Lines can have one of three cap styles: butt, round or square as shown in Figure 3-2.

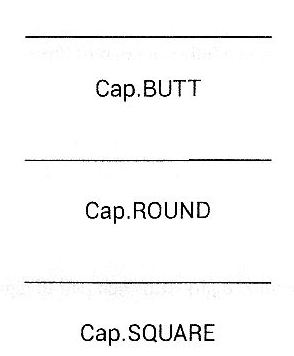


Figure 3-2.

Three cap styles are defined in Paint.Cap enum.

* **Cap.BUTT.** A flat edge is put perpendicular to each end of the line with no cap added. This is the default cap style.
* **Cap.ROUND.** A semicircle or rounded end cap is added to each end of the line. The cap diameter equals the line width value. This style adds 1/2 of the line width to each end of a line.
* **Cap.SQUARE.** A square end cap is added to each end of the line. The cap width is 1/2 of the line width.

The following method is used to set a cap style.

|  |
| --- |
| paint.setStrokeCap(Cap.ROUND); |

A cap style is applied to both ends of a line segment. It is not possible to have different cap styles for each end of a line.

Unfortunately there is a serious bug in Android Canvas API. In accelerated drawing mode, which is usually the default mode, drawLine() method always uses Paint.Cap.BUTT style. Even if we modify the previous example to set round line cap in the constructor as shown below

|  |
| --- |
| …  paint.setStrokeCap(Cap.ROUND)  … |

on most modern Android devices, the line will still have flat edges as in Figure 3-1. Fortunately there is a workaround. Instead of drawLine() method we can use Path API.

### 3.4 Introduction to Paths

The Path class represents a series of straight line segments, curves or other drawing primitives. A path can be open or closed. Creation of a path and drawing a path are two separate operations.

First, you create a path

|  |
| --- |
| Path path = new Path(); |

Then, you add some drawing primitives to it, like a line segment.

|  |
| --- |
| path.moveTo(100,100);  path.lineTo(700,700); |

And finally, you draw a path.

|  |
| --- |
| canvas.drawPath(path, paint); |

### 3.4.1 Example

In this example we will create and draw a simple path consisting of a single line, similar to the line from the previous example, but with round caps. The code of the application content view is shown below.

|  |
| --- |
| package com.example.graphics2d;  import android.content.Context;  import android.graphics.Canvas;  import android.graphics.Color;  import android.graphics.Paint;  import android.graphics.Paint.Cap;  import android.graphics.Paint.Style;  import android.graphics.Path;  import android.view.View;  public class PathView extends View{  private Paint paint;  private Path path;  public PathView(Context context) {  super(context);  setBackgroundColor(Color.WHITE);  paint = new Paint(Paint.ANTI\_ALIAS\_FLAG);  paint.setColor(Color.BLUE);  paint.setStyle(Style.STROKE);  paint.setStrokeCap(Cap.ROUND);  }  @Override  public void onDraw(Canvas canvas)  {  canvas.drawPath(path, paint);  }  @Override  protected void onSizeChanged(int width, int height, int oldw, int oldh)  {  paint.setStrokeWidth(Math.min(width, height)\*0.1f);  path = new Path();  path.moveTo(width\*0.2f, height\*0.2f);  path.lineTo(width\*0.8f, height\*0.8f);  }  } |

If you run this example you should see a line with round caps as shown in Figure 3-3.

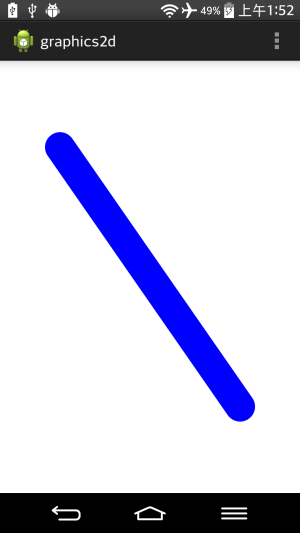


Figure 3-3.

Let’s review the code.

### 3.4.2 Code Review : Constructor

In the constructor we set the view background color and created a Path object. We also set the line color.

Then we set the paint style.

|  |
| --- |
| paint.setStyle(Style.STROKE); |

The STROKE paint style tells Canvas API to stroke the path or draw a line. There are two more paint styles: FILL and FILL\_AND\_STROKE. We will discuss different paint styles later in this chapter.

Finally we set the line cap style.

|  |
| --- |
| paint.setStrokeCap(Cap.ROUND); |

### 3.4.3 Code Review : onSizeChanged()

First we set the line width to 10% of the short side of the view.

|  |
| --- |
| paint.setStrokeWidth(Math.min(width, height)\*0.1f); |

Then we created a Path object.

|  |
| --- |
| path = new Path();  path.moveTo(width \*0.2f, height \*0.2f);  path.lineTo(width \*0.8f, height \*0.8f); |

The moveTo() and lineTo() methods use the concept of a current point or a location of the pen. The moveTo() method sets a new current point(move a pen to this point without drawing). The lineTo() method adds a line segment to the path from the current point. After that the current point is set to the end point of the line segment.

Note that creation of a path and drawing a path are two separate operations. To optimize performance of your application you should create a path in a constructor or onSizeChanged() method and then draw it in onDraw() method.

### 3.4.4 Code Review : onDraw()

In this method we drew the path created in onSizeChanged() method.

|  |
| --- |
| canvas.drawPath(path, paint); |

## 3.5 Join Styles

Another attribute which affects line appearance is a join style. Line join styles control the type of corner that is created when two lines meet. Three styles provided by Canvas API are shown in Figure 3-4.

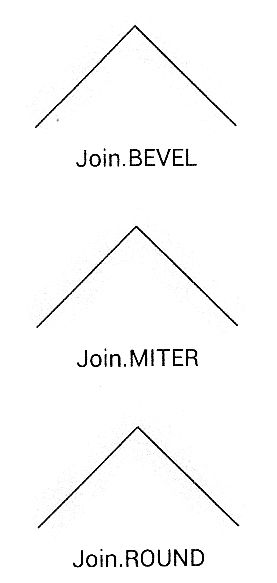


Figure 3-4.

Three join styles are defined in Paint.Join enum.

* **Join.BEVEL.** A filled triangle connects two lines, creating a beveled corner.
* **Join.MITER.** The outside edges of the lines are continued until they intersect. The resulting triangle is filled, creating a sharp or pointed corner. This is the default style.
* **Join.ROUND.** A filled arc connects two lines, creating a rounded corner.

### 3.5.1 Example

In this example we will draw a triangle and try different join styles. Let’s look at the code first and then review it.

|  |
| --- |
| package com.example.graphics2d;  import android.content.Context;  import android.graphics.Canvas;  import android.graphics.Color;  import android.graphics.Paint;  import android.graphics.Path;  import android.graphics.Paint.Cap;  import android.graphics.Paint.Join;  import android.graphics.Paint.Style;  import android.view.View;  public class TriangleView extends View  {  private Paint paint;  private Path path;  public TriangleView(Context context) {  super(context);  setBackgroundColor(Color.WHITE);  paint = new Paint(Paint.ANTI\_ALIAS\_FLAG);  paint.setColor(Color.BLUE);  paint.setStyle(Style.STROKE);  paint.setStrokeJoin(Join.MITER);  }  @Override  protected void onSizeChanged(int width, int height, int oldw, int oldh)  {  paint.setStrokeWidth(Math.min(width, height)\*0.1f);  path = new Path();  path.moveTo(w\*0.5f, h\*0.2f);  path.lineTo(w\*0.8f, h\*0.8f);  path.lineTo(w\*0.2f, h\*0.8f);  path.close();  }    @Override  public void onDraw(Canvas canvas)  {  canvas.drawPath(path, paint);  }  } |

### 3.5.2 Code Review : Constructor

In the constructor we set the view background color, created a Paint object and set its color and style. We also set the join style.

|  |
| --- |
| paint.setStrokeJoin(Join.MITER); |

### 3.5.3 Code Review : onSizeChanged( )

First we set the line width to 10% of the short side of the view.

|  |
| --- |
| paint.setStrokeWidth(Math.min(width, height)\*0.1f); |

Then we created a Path object.

|  |
| --- |
| path = new Path(); |

We used three points A(width\*0.5f, height\*0.2f), B(width\*0.8f, height\*0.8f), and C(width\*0.2f, height\*0.8f) to draw a triangle. The moveTo() method sets the start of the path at point A.

|  |
| --- |
| path.moveTo(width\*0.5f, height\*0.2f); |

The lineTo() methods and two line segments AB and BC to the path.

|  |
| --- |
| path.lineTo(width\*0.8f, height\*0.8f);  path.lineTo(width\*0.2f, height\*0.8f); |

Finally, the close() method closes the path. If the last point of a path is not equal to the first point, a line segment is automatically added. In our example, line segment CA is automatically added to the path.

|  |
| --- |
| path.close(); |

### 3.5.4 Code Review : onDraw( )

In this method we drew the path created in onSizeChanged() metod.

|  |
| --- |
| canvas.drawPath(path, paint); |

### 3.5.5 Screenshots

If you run this example you should see a triangle as shown in Figure 3-5.

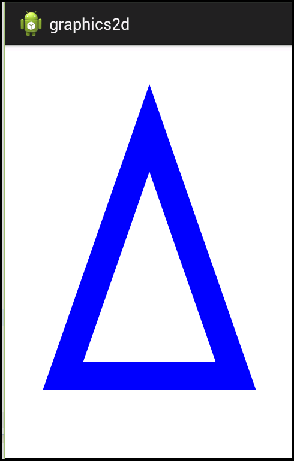


Figure 3-5.

If you change the join style to ROUND

|  |
| --- |
| Paint.setStrokeJoin(Join.ROUND); |

You should see a triangle as shown in Figure 3-6.

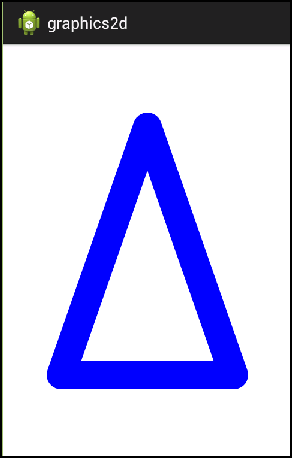


Figure 3-6.

Finally, let’s change the join style to BEVEL

|  |
| --- |
| paint.setStrokeJoin(Join.BEVEL); |

You should see a triangle as shown in Figure 3-7.

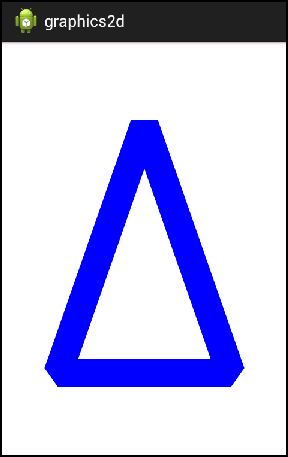


Figure 3-7.

### 3.5.6 Non-Closed Path

If you do not close a path, a join style will not be applied to the starting and end points of a path. Let’s use the ROUND line cap and MITER join

|  |
| --- |
| paint.setStrokeCap(Cap.ROUND);  paint.setStrokeJoin(Join.MITER); |

and draw a triangle without closing it.

|  |
| --- |
| path = new Path();  path.moveTo(w\*0.5f, h\*0.2f);  path.lineTo(w\*0.8f, h\*0.8f);  path.lineTo(w\*0.2f, h\*0.8f);  path.lineTo(w\*0.5f, h\*0.2f); |

You should see a triangle as shown in Figure 3-8.

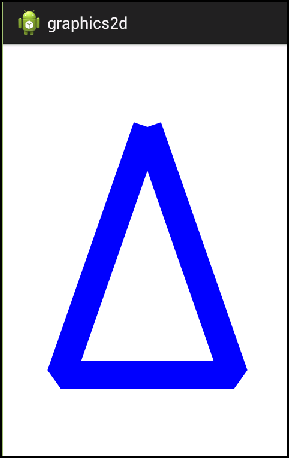


Figure 3-8.

## 3.6 Paint Styles : Stroke and Fill

There are three paint styles in Canvas API.

* Style.STROKE. draws a line with specified stroke width, color and other attributes.
* Style.Fill. fills the area contained within the closed path or another drawing primitives such as a circle or rectangle. This is the default style.
* Style.FILL\_AND\_STROKE strokes and fills with the same color. This style is almost never used, because usually you want to fill a path with one color and stroke it with another one.

The following method is used to set a paint style.

|  |
| --- |
| Paint.setStyle(Style.STROKE); |

### 3.6.1 Example

This example shows how to fill a triangle with one color and stroke it with another color. Let’s look at the code first and then review it.

|  |
| --- |
| package com.example.graphics2d;  import android.content.Context;  import android.graphics.Canvas;  import android.graphics.Color;  import android.graphics.Paint;  import android.graphics.Path;  import android.graphics.Paint.Join;  import android.graphics.Paint.Style;  import android.view.View;  public class TriangleViewFill extends View  {  private Paint paint;  private Path path;  public TriangleViewFill(Context context) {  super(context);  setBackgroundColor(Color.WHITE);  paint = new Paint(Paint.ANTI\_ALIAS\_FLAG);  paint.setStrokeJoin(Join.ROUND);  }  @Override  protected void onSizeChanged(int width, int height, int oldw, int oldh)  {  paint.setStrokeWidth(Math.min(width, h)\*0.1f);  path = new Path();  path.moveTo(width \*0.5f, height \*0.2f);  path.lineTo(width \*0.8f, height \*0.8f);  path.lineTo(width \*0.2f, height \*0.8f);  path.close();  }    @Override  public void onDraw(Canvas canvas)  {  //Fill  paint.setStyle(Style.FILL);  paint.setColor(Color.rgb(255, 153, 0));  canvas.drawPath(path, paint);  //Stroke  paint.setStyle(Style.STROKE);  paint.setColor(Color.rgb(204, 0, 0));  canvas.drawPath(path, paint);  }  } |

### 3.6.2 Code Review : Constructor

In the constructor we set the view background color, created a Paint object and set its join style.

### 3.6.3 Code Review : onSizeChanged()

In this method we set the stroke width

|  |
| --- |
| paint.setStrokeWidth(Math.min(width, height)\*0.1f); |

and created the triangle path.

|  |
| --- |
| path = new Path();  path.moveTo(width \*0.5f, height \*0.2f);  path.lineTo(width \*0.8f, height \*0.8f);  path.lineTo(width \*0.2f, height \*0.8f);  path.close(); |

### 3.6.4 Code Review : onDraw()

In this method we drew the path twice.

First, the path was filled with yellow color.

|  |
| --- |
| paint.setStyle(Style.FILL);  paint.setColor(Color.rgb(255, 153, 0));  canvas.drawPath(path, paint); |

Then it was stroked with red color.

|  |
| --- |
| paint.setStyle(Style.STROKE);  paint.setColor(Color.rgb(204, 0, 0));  canvas.drawPath(path, paint); |

### 3.6.5 Screenshots

If you run this example you should see a triangle as shown in Figure3-9.

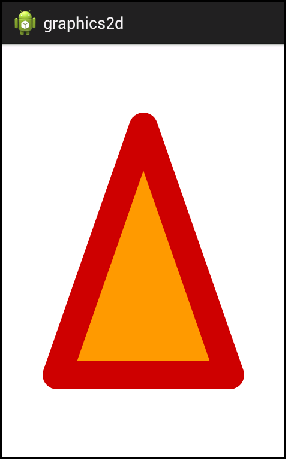


Figure3-9.

## 3.7 Dashed Lines

So far we only drew solid lines, but Canvas library also supports dashed and dotted lines. Do draw a dashed line you have to:

* Create an instance of the DashPathEffect class.
* Configure a Paint object to use this path effect.

You create a DashPathEffect object by calling its constructor.

|  |
| --- |
| DashPathEffect(float[] intervals, float phase) |

It has the following parameters:

**‧intervals** – elements of this array specify widths of dashes in pixels. Even elements specify “on” intervals, and odd elements specify “off” intervals. The array must contain an even number of wntries.

**‧phase** – horzontal offset in pixels. Usually this parameter is 0.

After a path effect is created you configure a Paint object to use this effect by calling the following method of the Paint class.

|  |
| --- |
| public PathEffect setPathEffect(PathEffect effect) |

As a convenience, the parameter passed is also returned. You can pass null to clear any previous path effect.

### 3.7.1 Example

This example shows how to use the DashPathEffect class to draw dashed and dotted lines. Let’s look at the code first and then review it.

|  |
| --- |
| package com.example.graphics2d;  import android.content.Context;  import android.graphics.Canvas;  import android.graphics.Color;  import android.graphics.DashPathEffect;  import android.graphics.Paint;  import android.graphics.Paint.Style;  import android.graphics.Path;  import android.view.View;  public class DashedLineView extends View  {  private Paint paint;  private Path path;  public DashedLineView(Context context) {  super(context);  setBackgroundColor(Color.WHITE);  paint = new Paint(Paint.ANTI\_ALIAS\_FLAG);  paint.setColor(Color.BLUE);  paint.setStrokeWidth(30);  paint.setStyle(Style.STROKE);  }  @Override  public void onDraw(Canvas canvas)  {  canvas.drawPath(path, paint);  }  @Override  protected void onSizeChanged(int width, int height, int oldw, int oldh)  {  path = new Path();  path.moveTo(50, 100);  path.lineTo(width - 50, 100);  DashPathEffect effect = new DashPathEffect(new float[]{30,30},60);  paint.setPathEffect(effect);  }  } |

### 3.7.2 Code Review

In the constructor we set the view background color and created a Paint object. In onSizeChanged() method we created a path consisting of a single horzontal line.

|  |
| --- |
| path = new Path();  path.moveTo(50, 100);  path.lineTo(width - 50, 100); |

Then we created a DashPathEffect object.

|  |
| --- |
| DashPathEffect effect = new DashPathEffect(new float[]{90,30},0); |

The intervals array in DashPathEffect constructor has two values: 90 pixels for the painted segment of the line and 30 pixels for the unpainted segment. The phase parameter is 0.

Finally, we configured the Paint object to use this path effect to stroke the path.

|  |
| --- |
| paint.setPathEffect(effect); |

In onDraw() method we drew the path.

|  |
| --- |
| canvas.drawPath(path, paint); |

If you run this example you should see a dashed line shown in Figure 3-10.



Figure 3-10.

Now, let’s see how the phase parameter in the DashPathEffect constructor affects the dash pattern. If you set the phase parameter to 60 pixels as shown below

|  |
| --- |
| DashPathEffect effect = new DashPathEffect(new float[]{90,30},60); |

The dash pattern will shift 60 pixels to the left as shown in Figure 3-11.



Figure 3-11.

You can also use round stroke cap style to create dotted lines as shown below.

|  |
| --- |
| paint.setStrokeCap(Cap.ROUND);  DashPathEffect effect = new DashPathEffect(new float[]{1,60},0); |



Figure 3-12.

We used 1 pixel width for the painted segment of a line in the DashPathEffect constructor. Round line caps were added by Canvas library.

# Chapter 04 Working with Text

As you may remember, we already used basic text API in our first program in Chapter 1 to display a view size.In this chapter we will take a closer look at Canvas API text rendering capabilities. You will learn how to use different fonts, how to measure and align text, and how to change other text attributes, such as size, color and shadow.

## 4.1 Standard Fonts

Every Android device comes with three standard fonts: one proportional sans-serif font, one proportional serif font and one monospaced font. The actual fonts installed on a device depend on the device model and the version of Android operating system. Many devices use Droid Sans, Droid Serif, and Droid Sans Mono fonts. The default sans-serif font on the Android Ice Cream Sandwich operating system is Roboto. There is no API to find names of fonts installed on a device.

Let’s discuss some common typographic terms.

### 4.1.1 Typeface

A typeface or a font family is a set of characters that share common design features. After the advent of digital typography and desktop publishing, there is no clear distinction between between terms font and typeface. Usually term typeface refers to a design, like Times New Roman or Palatino and fonts are implementations of a design, such as TrueType, OpenType, or PostScript fonts of Times New Roman typeface.

### 4.1.2 Serifs

In typography, a serif is a small decorative line added as embellishment to a character. Typefaces are often described as being serif or sans serif(without serifs).

* **Serif** typefaces use small decorative marks to embellish characters. Serif fonts include Times New Roman, Courier, New Century Schoolbook, and Palatino.
* **Sans serif** typefaces are composed of simple lines and do not use serifs or small lines at the ends of characters. Popular sans serif fonts are Helvetica, Avant Garde, Arial, and Geneva.

### 4.1.3 Proportion

Typefaces can be proportional and non-proportional.

* **Proportional** typefaces contain characters of varying widths. For examples, the W and M letters are wider than most letters, and the I is narrower.
* **Non-proportional** or **monospaced** typefaces use the same width for all characters. The first monospaced typefaces were designed for typewriters, therefore monospaced fonts are sometimes called typewriter fonts. Popular monospaced typefaces are Courier and Monaco.

### 4.1.4 Typeface Class

The Typeface class specifies the typeface and a style of a font. Every Android device supports three typefaces:

* sans-serif
* serif
* monospace

and four styles:

* NORMAL
* BOLD
* ITALIC
* BOLD\_ITALIC

The following method can be used to create an instance of the Typeface class, give a font family name and a style.

|  |
| --- |
| Typeface font = Typeface.create(“serif”,Typeface.ITALIC); |

If null is passed for the name, then the default font will be chosen. On most devices it is the sans-serif font.

The Typeface class also provides several predefined typefaces.

|  |
| --- |
| Typeface.DEFAULT = Typeface.create(null, Typeface.NORMAL);  Typeface.DEFAULT\_BOLD = Typeface.create(null, Typeface.BOLD);  Typeface.SANS\_SERIF = Typeface.create(“sans-serif”, Typeface. NORMAL);  Typeface. SERIF = Typeface.create(“serif”, Typeface. NORMAL);  Typeface. MONOSPACE = Typeface.create(“monospace”, Typeface. NORMAL); |

The following method of the Paint class is used to set a typeface.

|  |
| --- |
| public Typeface setTypeface(Typeface typeface) |

If null is passed, any previous typeface will be cleared. This method returns the parameter passed.

### 4.1.5 Example

Let’s create a simple application that will display several standard fonts available on every Android device.

#### Activity

We start with the new project and activity.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.app.Activity;  **import** android.os.Bundle;  **import** android.view.Window;  **import** android.widget.TextView;  **public** **class** GraphicsActivity **extends** Activity {  @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  **this**.requestWindowFeature(Window.***FEATURE\_NO\_TITLE***);  TextView view = **new** TextView(**this**);  setContentView(view);  }  } |

The code is similar to other examples from previous chapters. To save some space we will not show the title or the action bar which takes a lot of space in Android 4.x.

|  |
| --- |
| **this**.requestWindowFeature(Window.***FEATURE\_NO\_TITLE***); |

#### View

Next, we will create the view.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.graphics.Typeface;  **import** android.view.View;  **public** **class** TextView **extends** View{  **private** Paint paint;  **public** Typeface(Context context) {  **super**(context);  setBackgroundColor(Color.***WHITE***);  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setColor(Color.***BLACK***);  paint.setTextSize(70);  }    @Override  **public** **void** onDraw(Canvas canvas)  {  **int** x = 20;  **int** y = 90;  paint.setTypeface(Typeface.***DEFAULT***);  canvas.drawText("Default", x, y, paint);  y+=90;  paint.setTypeface(Typeface.***DEFAULT\_BOLD***);  canvas.drawText("Default Bold", x, y, paint);  y+=90;  paint.setTypeface(Typeface.***SANS\_SERIF***);  canvas.drawText("SANS SERIF", x, y, paint);  y+=90;  paint.setTypeface(Typeface.***SERIF***);  canvas.drawText("SERIF", x, y, paint);  y+=90;  paint.setTypeface(Typeface.***MONOSPACE***);  canvas.drawText("MONOSPACE", x, y, paint);  y+=90;  Typeface font = Typeface.*create*("serif", Typeface.***BOLD***);  paint.setTypeface(font);  canvas.drawText("serif BOLD", x, y, paint);  y+=90;  font = Typeface.*create*("serif", Typeface.***ITALIC***);  paint.setTypeface(font);  canvas.drawText("serif ITALIC", x, y, paint);  }  } |

#### Code Review

As you may have noticed, to make this example simpler, we hard-coded the font size and coordinates. Therefore this example will look different on different Android devices. In real applications you should always calculate the font size and device coordinates based on the view size and width-to-height ratio.

In the constructor we created a Paint object, set the background and paint colors. Also we set font size in pixels.

|  |
| --- |
| paint.setTextSize(70); |

In OnDraw() method we drew several standard fonts. We used both predefined fonts:

|  |
| --- |
| paint.setTypeface(Typeface.***DEFAULT***); |

and created several new typeface objects:

|  |
| --- |
| Typeface font = Typeface.*create*("serif", Typeface.***BOLD***); |

To select a font, we used the following method of the Paint object.

|  |
| --- |
| paint.setTypeface(font); |

#### Screenshot

A screenshot of the application is shown in Figure 4-1.



Figure 4-1.

Because we hard-coded the font size and coordinates, this example may look different on your device.

## 4.2 Custom Fonts

In addition to three standard fonts, Canvas API supports any TrueType or OpenType fonts. Any custom font you want to use has to be bundled with your application. The assets folder at the project root is a convenient location to store raw files. Any file you save in the assets folder is included in the application as-is and the original filename is preserved.

The following method of the Typeface class can be used to create a custom font.

|  |
| --- |
| Public static Typeface createFromAsset(AssetManager mgr, String path) |

This method has two parameters.

The first parameter is an instance of AssetManager class which provides access to the application raw asset files. Usually you get it from the Context by calling

|  |
| --- |
| context.getAssets(); |

The second parameter is a file path in the assets directory.

### 4.2.1 Example

The following example shows how to use custom fonts in your application.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.graphics.Typeface;  **import** android.view.View;  **public** **class** CustomFontView **extends** View{  **private** Paint paint;  **private** Typeface font1, font2, font3;  **public** CustomFontView(Context context) {  **super**(context);  setBackgroundColor(Color.***WHITE***);  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setColor(Color.***BLACK***);  paint.setTextSize(90);  font1 = Typeface.*createFromAsset*(context.getAssets(), "AGENCYB.TTF");  font2 = Typeface.*createFromAsset*(context.getAssets(), "AlexBrush-Regular.ttf");  font3 = Typeface.*createFromAsset*(context.getAssets(), "BAUHS93.TTF");  }    @Override  **public** **void** onDraw(Canvas canvas)  {  **int** x = 15;  **int** y = 100;  paint.setTypeface(font1);  canvas.drawText("AGENCYB", x, y, paint);  y+=150;  paint.setTypeface(font2);  canvas.drawText("AlexBrush", x, y, paint);  y+=150;  paint.setTypeface(font3);  canvas.drawText("BAUHS93", x, y, paint);  }  } |

#### Code Review

We downloaded three fonts from [www.fontsquirrel.com](http://www.fontsquirrel.com):

* EuphoriaScript-Regular.otf(OpenType)
* Pacifico.ttf(TrueType)
* Chantelli\_Antiqua.ttf(TrueType)

and put them in the assets folder at the project root.

In the constructor we create three Typeface objects from font files stored in the assets folder.

|  |
| --- |
| font1 = Typeface.*createFromAsset*(context.getAssets(), "AGENCYB.TTF");  font2 = Typeface.*createFromAsset*(context.getAssets(), "AlexBrush-Regular.ttf");  font3 = Typeface.*createFromAsset*(context.getAssets(), "BAUHS93.TTF"); |

To get a reference to the AssetManager we called

|  |
| --- |
| context.getAssets() |

To select a font we used the same method as in the previous example.

|  |
| --- |
| paint.setTypeface(font1); |

#### Screenshot

A screenshot of the application is shown in Figure 4-2.



Figure 4-2.

Because we hard-coded the font and coordinates, this example may look different on your devices.

## 4.3 Horizontal Alignment

To align text horizontally you can either let Canvas library do it for you or if you want more control over exact location of a text you can calculate the x-coordinate yourself.

By default, text is left aligned. It means that text is drawn to the right of the (x,y) origin passed to drawText() method of the Canvas object.

### 4.3.1 Standard Alignment

The easiest way to change the text alignment is to call the following method of the Paint class.

|  |
| --- |
| void setTextAlign(Align align) |

The align parameter of this method can have one of the following values:

* Align.LEFT
* Align.CENTER
* Align.RIGHT

#### Example

The example below shows how to use setTextAlign() method in your application.

|  |
| --- |
| package com.example.graphics2d;  import android.content.Context;  import android.graphics.Canvas;  import android.graphics.Color;  import android.graphics.Paint;  import android.graphics.Paint.Align;  import android.view.View;  public class TextAlignView extends View{  private Paint paint;  private int width;  private int height;  public TextAlignView(Context context) {  super(context);  setBackgroundColor(Color.WHITE);  paint = new Paint();  paint.setColor(Color.BLACK);  paint.setTextSize(90);  }    @Override  public void onDraw(Canvas canvas)  {  int x = width/2;  int y = 120;  canvas.drawLine(x, 0, x, height, paint);  paint.setTextAlign(Align.LEFT);  canvas.drawText("Left", x, y, paint);  y+=120;  paint.setTextAlign(Align.CENTER);  canvas.drawText("Center", x, y, paint);  y+=120;  paint.setTextAlign(Align.RIGHT);  canvas.drawText("Right", x, y, paint);  }  @Override  public void onSizeChanged(int w, int h, int oldw, int oldh){  width = w;  height = h;  }  } |

A screenshot of the application is shown in Figure 4-3.

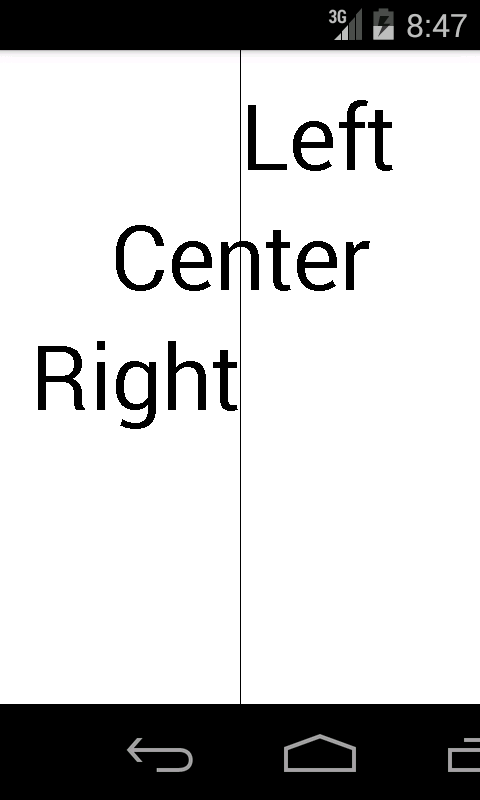


Figure 4-3.

This method of aligning a text should be enough for most applications, but if you want more control over exact text location, you can keep default text alignment and adjust the x-coordinate yourself.

### 4.3.2 Custom Alignment

Before calculating the x-coordinate, you have to measure the text. The Paint object provides a method for that.

|  |
| --- |
| Void getTextBounds(String text, int start, int end, Rect bounds) |

It has the following parameters.

* text – a string to measure.
* start – index of the first character in the string. Usually you pass 0.
* end – index of the last character in the string plus one. Usually you pass string length here.
* bounds – returns the smallest rectangle that encloses all of the characters. Left, right, top and bottom values of the rectangle are relative to the origin or x- and y-coordinates passed to drawText() method.

Let’s call getTextBounds() method and examine returned values in a debugger.

|  |
| --- |
| @Override  public void onDraw(Canvas canvas)  {  int x = width/2;  int y = 120;  canvas.drawLine(x,0,x,height,paint);  Rect rect = new Rect();  String str = “Left”;  paint.getTextBounds(str, 0, str.length(), rect);  ….  } |

The exact values depend on the screen resolution and fonts installed on a device. We got the following values.

|  |
| --- |
| rect.left = 7  rect.right = 150  rect.top = -71  rect.bottom = 1 |

To calculate the x-coordinate to adjust the horizontal text alignment we only need the left and right values. We will talk more about the top and bottom values in the next section,

As you can see from Figure 4-3, there is some space between the letter L and the vertical line. The space is 7 pixel wide in our case. But this value depends on a letter and a font. The rect.left tells you what the value is. If you don’t want this extra space you can adjust your x-coordinate as shown below to shift the text left.

|  |
| --- |
| canvas.drawText(str, x-rect.left, y, paint); |

To center the text, the following formula can be used.

|  |
| --- |
| float offset = (rect.left + rect.right) / 2.0f;  canvas.drawText(str, x-rect.right, y, paint); |

## 4.4 Vertical Alignment

Unfortunately Canvas API doesn’t provide any methods to align text vertically. We have to measure text and calculate the y-coordinate ourselves before passing it to drawText() method. Before we learn how this can be done, let’s discuss some font metrics vocabulary.

### 4.4.1 Baseline

The baseline is the imaginary horizontal line on which most letters sit. The y-coordinate in drawText() method is the y-coordinate of the baseline. Let’s create a simple view to draw a baseline in addition to some text.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.graphics.Typeface;  **import** android.graphics.Paint.Align;  **import** android.view.View;  **public** **class** TextViewBaseline **extends** View{    **private** Paint paint1;  **private** **int** height;  **private** **int** width;    **public** TextViewBaseline(Context context) {  **super**(context);  setBackgroundColor(Color.***WHITE***);  paint1 = **new** Paint();  paint1.setColor(Color.***BLACK***);  paint1.setTextSize(100);  paint1.setTypeface(Typeface.***SERIF***);  }  @Override  **public** **void** onDraw(Canvas canvas){  **int** x = width/2;  **int** y = 150;  canvas.drawLine(0, y, width, y, paint1);  paint1.setTextAlign(Align.***CENTER***);  canvas.drawText("Python", x, y, paint1);  }    @Override  **protected** **void** onSizeChanged(**int** w, **int** h, **int** oldw, **int** oldh){  width = w;  height = h;  }  } |

The screenshot of the example is shown in Figure 4-4.

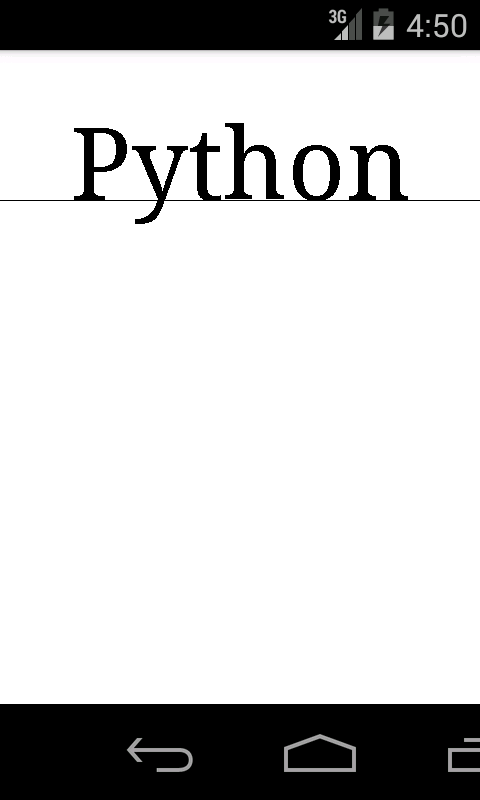


Figure 4-4.

As you can see most letters sit on the baseline and he letter y extends below the baseline.

### 4.4.2 Ascent and Descent

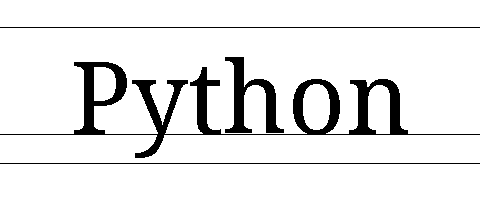
The **ascent** is the distance between the baseline and the top of the tallest character. The ascent usually includes extra space required to display letters with accents, such as an umlaut in German language.

The descent is the distance between the baseline and the lowest descending letter in a typeface, such as the y letter.

Let’s modify onDraw() method to draw both ascent and descent lines.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Align;  **import** android.graphics.Typeface;  **import** android.view.View;  **public** **class** TextAscent **extends** View{    **private** Paint paint1;  **private** **int** height;  **private** **int** width;    **public** TextAscent(Context context) {  **super**(context);  setBackgroundColor(Color.***WHITE***);  paint1 = **new** Paint();  paint1.setColor(Color.***BLACK***);  paint1.setTextSize(100);  paint1.setTypeface(Typeface.***SERIF***);  }  @Override  **public** **void** onDraw(Canvas canvas){  **int** x = width/2;  **int** y = 150;  **float** ascent = paint1.ascent();  **float** descent = paint1.descent();  canvas.drawLine(0, y, width, y, paint1);  canvas.drawLine(0, y+ascent, width, y+ascent, paint1);  canvas.drawLine(0, y+descent, width, y+descent, paint1);  paint1.setTextAlign(Align.***CENTER***);  canvas.drawText("Python", x, y, paint1);  }    @Override  **protected** **void** onSizeChanged(**int** w, **int** h, **int** oldw, **int** oldh){  width = w;  height = h;  }  } |

The screenshot of the modified example is shown in Fugure4-5.



Fugure4-5.

As you can see there is some extra space between the ascent line and the tallest letters. That space is reserved for upper case letters with accents as shown in Figure 4-6.

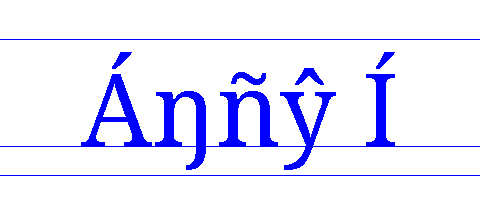


Figure 4-6.

### 4.4.3 Bottom Alignment

By default, text is bottom aligned at the font baseline as shown in Figure 4-4. You don’t have to do any extra work to align it.

### 4.4.4 Center Alignment

There are two common ways to center align text. The first method is based on the font ascent and descent values and the second on exact text height.

The following example shows how to use both methods.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.graphics.Rect;  **import** android.graphics.Typeface;  **import** android.graphics.Paint.Align;  **import** android.view.View;  **public** **class** CenterAlignment **extends** View{  **int** font\_size = 100;  Paint paint1,paint2;  **int** height, width;  String str = "Play";    **public** CenterAlignment(Context context) {  **super**(context);  setBackgroundColor(Color.***WHITE***);  paint1 = **new** Paint();  paint1.setColor(Color.***BLACK***);  paint1.setTextSize(font\_size);  paint1.setTypeface(Typeface.***SERIF***);  paint1.setTextAlign(Align.***CENTER***);  paint2 = **new** Paint();  paint2.setColor(Color.***RED***);  paint2.setStrokeWidth(2);    }  @Override  **public** **void** onDraw(Canvas canvas){  center1(canvas, str,100);  center2(canvas, str,300);  }    //centered by ascent and descent  **private** **void** center1(Canvas canvas, String str, **int** y) {  **int** x = width/2;  **float** offset = (paint1.ascent()+paint1.descent())/2;  canvas.drawText(str, x, y-offset, paint1);  **float** topY = y - font\_size/2f;  **float** bottomY = y + font\_size/2f;    canvas.drawLine(0, y, width, y, paint2);  canvas.drawLine(0, topY, width, topY, paint2);  canvas.drawLine(0, bottomY, width, bottomY, paint2);    }  //centered by bounds  **private** **void** center2(Canvas canvas, String str, **int** y) {  **int** x = width/2;  Rect bounds = **new** Rect();  paint1.getTextBounds(str, 0, str.length(), bounds);    **float** offset = bounds.exactCenterY();  canvas.drawText(str, x, y-offset, paint1);    **float** topY = y - font\_size/2f;  **float** bottomY = y + font\_size/2f;    canvas.drawLine(0, y, width, y, paint2);  canvas.drawLine(0, topY, width, topY, paint2);  canvas.drawLine(0, bottomY, width, bottomY, paint2);    }  @Override  **protected** **void** onSizeChanged(**int** w, **int** h, **int** oldw, **int** oldh){  width = w;  height = h;  }  } |

A screenshot of the example is shown in Figure 4-7. There are two lines of text. The first line was aligned based on the front ascent and descent values and the second, based on exact text height obtained from getTextBounds() method.

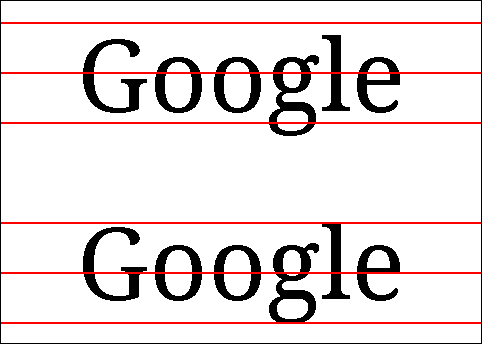


Figure 4-7.

The first method is based on the font ascent and descent values and is implemented in center1() method. First we calculated the y-coordinate offset which is the distance from the baseline to the center.

|  |
| --- |
| **float** offset = (paint1.ascent()+paint1.descent())/2; |

Then we shifted text down by the offset.

|  |
| --- |
| canvas.drawText(str, x, y-offset, paint1); |

Let’s go over math involved in the offset calculation. In Canvas API paint.ascent() value is negative and paint.descent() is positive. It makes sense because Android Canvas API uses the upper-left corner of the screen as the origin and the y-coordinate increases downward. In our example the following values were used.

|  |
| --- |
| ascent = -92.8  descent = 23.6  offset = (ascent + descent) / 2 = (-92.8 + 23.6) / 2 = -34.6 |

We could have calculated maximum text first and then added half-height to the ascent to get the same offset value.

|  |
| --- |
| height = (abs(ascent) + abs(descent)) = 116.4  offset = ascent + height / 2 = -92.8 + 58.2 = -34.6 |

We subtracted the offset from the y-coordinate, because the offset is negative and we have to shift text down, or increase the y-coordinate value.

|  |
| --- |
| canvas.drawText(str, x, y-offset, paint); |

The second method is similar to the first one. The only difference is that we measure text exactly instead of using maximum ascent and descent values. The method is more accurate. The implementation is provided by center2() method. The getTextBounds() methods is used to measure the text.

|  |
| --- |
| Rect bounds = **new** Rect();  paint1.getTextBounds(str, 0, str.length(), bounds); |

It measures both the height and width of the text. In our example we are only interested in the top and bottom fields of the Rect object. The ascent and descent values are the maximum values for a font, but bounds.top and bounds.bottom are exact values of the text being measured. Let’s compare the values.

|  |
| --- |
| ascent = -92.8  descent = 23.6  bounds.top = -76  bounds.bottom = 24 |

The Rect object provides a method to calculate vertical center of a rectangle.

|  |
| --- |
| **float** offset = bounds.exactCenterY(); |

We could have used the following formula instead.

|  |
| --- |
| **float** offset = (bounds.top + bounds.bottom)/2.0f; |

### 4.4.5 Top Alignment

Similar to center alignment, the y-coordinate of top aligned text can be adjusted based on either font ascent or the exact text height.

The example below shows how to use both methods.

|  |
| --- |
| package com.example.graphics2d;  import android.content.Context;  import android.graphics.Canvas;  import android.graphics.Color;  import android.graphics.Paint;  import android.graphics.Rect;  import android.graphics.Typeface;  import android.graphics.Paint.Align;  import android.view.View;  public class TopAlignment extends View{  private static final int FONT\_SIZE = 100;  private Paint paint;  private int height, width;    public TopAlignment(Context context) {  super(context);  setBackgroundColor(Color.WHITE);  paint = new Paint();  paint.setColor(Color.BLACK);  paint.setTextSize(FONT\_SIZE);  paint.setTypeface(Typeface.SERIF);  paint.setTextAlign(Align.CENTER);  }  @Override  public void onDraw(Canvas canvas){  top1(canvas, "Python",50);  top2(canvas, "Python",250);  }    private void top1(Canvas canvas, String str, int y) {  int x = width/2;  canvas.drawText(str, x, y-paint.ascent(), paint);  canvas.drawLine(0, y, width, y, paint);  }  //centered by bounds  private void top2(Canvas canvas, String str, int y) {  int x = width/2;  Rect bounds = new Rect();  paint.getTextBounds(str, 0, str.length(), bounds);  canvas.drawText(str, x, y-bounds.top, paint);  canvas.drawLine(0, y, width, y, paint);  }  @Override  protected void onSizeChanged(int w, int h, int oldw, int oldh){  width = w;  height = h;  }  } |

A screenshot of the example is shown in Figure 4-8. There are two lines of text. The first line was aligned based on the font ascent and the second on exact text height obtained from getTextBounds() method.

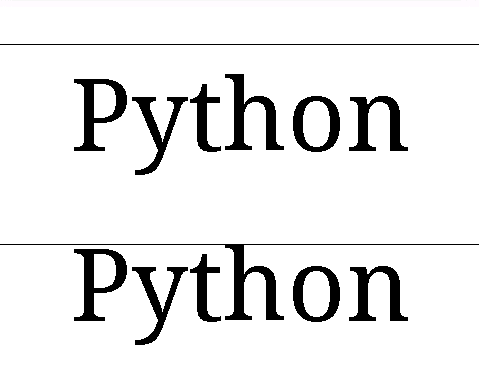


Figure 4-8.

Top alignment based on the font ascent was implemented in top1() method.

|  |
| --- |
| canvas.drawText(str, x, y-bounds.top, paint); |

The second method based on exact text height obtained from getTextBounds() method was implemented in top2 method.

|  |
| --- |
| Rect bounds = **new** Rect();  paint.getTextBounds(str, 0, str.length(), bounds);  canvas.drawText(str, x, y-bounds.top, paint); |

As you can see from Figure 4-8. the first method leaves a gap between the horizontal line the text is aligned to and the tallest letter. The second method is more accurate.

## 4.5 Shadow

A shadow is an additional layer painted below the main object such as a text or another graphics primitive. A shadow can improve the appearance of a text in some cases, but it does not mean you have to use shadows all the time,

The Paint object provides the following method to configure shadow appearance.

|  |
| --- |
| void setShadowLayer(float radius, float dx, float dy, int color) |

It has several parameters.

radius – a blur radius in pixels. If radius is 0, the shadow layer is removed. A higher radius make the image blurrier.

dx – an offset of a shadow in the horizontal direction in pixels.

dy – an offset of a shadow in the vertical direction in pixels.

color – a shadow color.

The example below shows how to use shadows in your application.

|  |
| --- |
| package com.example.graphics2d;  import android.content.Context;  import android.graphics.Canvas;  import android.graphics.Color;  import android.graphics.Paint;  import android.graphics.Typeface;  import android.graphics.Paint.Align;  import android.view.View;  public class Shadow extends View{    private Paint paint;  private int height, width;  public Shadow(Context context) {  super(context);  setBackgroundColor(Color.WHITE);  paint = new Paint(Paint.ANTI\_ALIAS\_FLAG);  paint.setColor(Color.BLACK);  paint.setTextSize(100);  paint.setTypeface(Typeface.SERIF);  paint.setTextAlign(Align.CENTER);  }  @Override  public void onDraw(Canvas canvas){  int x = width/2;  int y = 120;  paint.setShadowLayer(10, 10, -10, Color.GRAY);  canvas.drawText("Shadow", x, y, paint);  y+=140;  paint.setShadowLayer(16, 10, 10, Color.GRAY);  canvas.drawText("Shadow", x, y, paint);  }  @Override  protected void onSizeChanged(int w, int h, int oldw, int oldh){  width = w;  height = h;  }  } |

A screenshot of the application is shown in Figure 4-9.

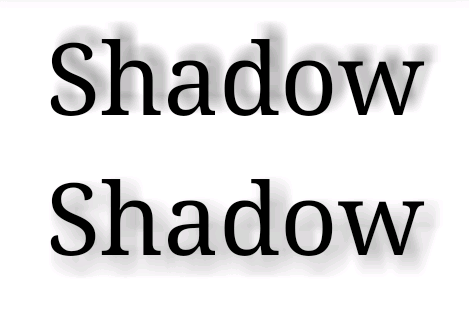


Figure 4-9.

You can experiment with different parameters, and see which values work best with a particular typeface and font size.

# Chapter 5 Rectangles

Canvas API supports two types if rectangles: regular rectangles and rectangles with rounded corners. There are several methods to draw a regular rectangles:

|  |
| --- |
| void drawRect(float left, float top, float right, float bottom, Paint paint)  void drawRect(RectF rect, Paint paint)  void drawRect(Rect rect, Paint paint) |

and one method to draw a rectangle with rounded corners

|  |
| --- |
| void drawRoundRect(RectF rect, float rx, float ry, Paint paint) |

The parameters of all four methods are listed below.

* left – the x-coordinate of the left side of the rectangle to be drawn.
* top – the y-coordinate of the top side of the rectangle to be drawn.
* right – the x-coordinate of the right side of the rectangle to be drawn.
* bottom –the y-coordinate of the bottom side of the rectangle to be drawn.
* rect – the rectangle to be drawn.
* rx – the x-radius of the oval used to round the corners.
* ry – the y-radius of the oval used to round the corners.
* paint – the paint used to draw the rectangle.

Note that you can also use Path API to draw rectangles.

## 5.1 Example : Bar Chart

In this example we will draw a bar chart shown in Figure 5-1.

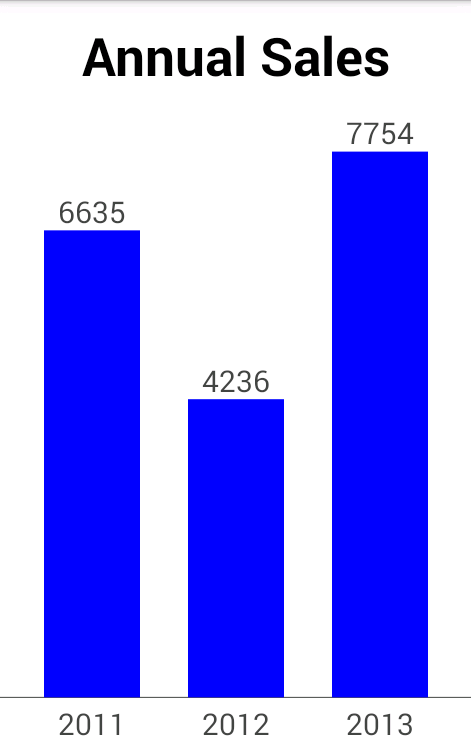


Figure 5-1.

It is a good practice to separate data and its representation. In our application the BarChartModel class provides data and the BarChartView class display it.

### 5.1.1 Model

The BarChartModel class is shown below.

|  |
| --- |
| **package** com.example.graphics2d;  **public** **class** BarChartModel {  **private** String labels[]={"2012","2013","2014"};  **private** **float** values[]={6635,4236,7754};  **private** **float** maxValue;    **public** BarChartModel(){  // Calculate max value  maxValue = values[0];  **for**(**int** i = 1; i < values.length;i++){  **if**(values[i] > maxValue){  maxValue = values[i];  }  }  }    **public** **int** size(){  **return** values.length;  }    **public** String getLabel(**int** index){  **return** labels[index];  }    **public** **float** getValue(**int** index){  **return** values[index];  }    **public** **float** maxValue(){  **return** maxValue;  }  } |

The BarChartModel class has the following methods:

* size() – returns number of elements or bars in the model.
* getLabel() – returns a label for each bar. In our example, it is a year.
* getValue() – returns annual sales value for each bar. This value is used to calculate bar height.
* maxValue() – returns the maximum value. We will need this value to calculate bar height.

### 5.1.2 View

The bar chart has several components: a title, labels and the plot area – a part of the view where we draw bars or rectangles. A mock-up of the bar chart view with all its components is shown in Figure 5-2.

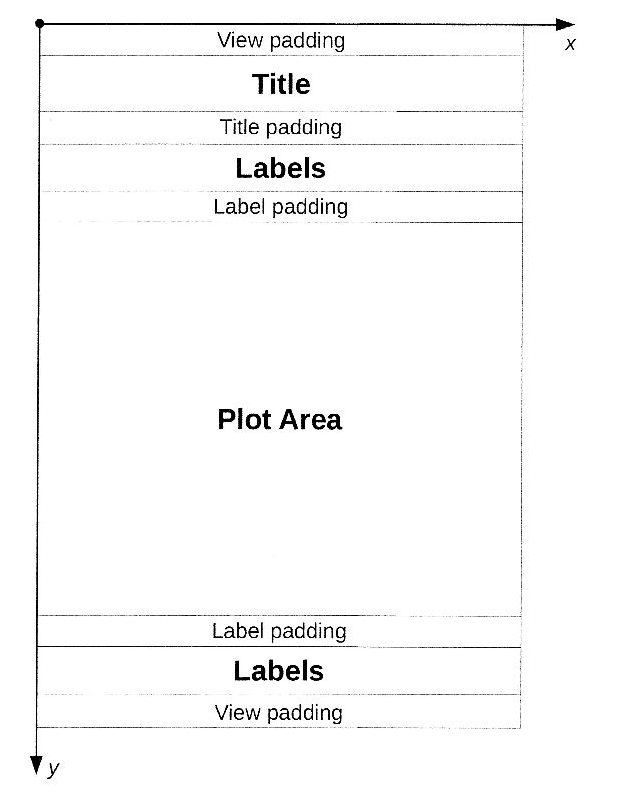


Figure 5-2.

We will calculate coordinates of all components at run-time as ratios or percentages of the view size.

Let’s look at the code first and then do a review.

|  |
| --- |
| package com.janas.ch5barchart;  import android.content.Context;  import android.graphics.Canvas;  import android.graphics.Color;  import android.graphics.Paint;  import android.graphics.Paint.Align;  import android.graphics.Paint.Style;  import android.graphics.RectF;  import android.graphics.Typeface;  import android.view.View;  public class BarChartView extends View{    private Paint barPaint;  private Paint titleTextPaint;  private Paint labelTextPaint;  float totalWidth, barWidth, barPading, labelPading, titleY, labelY, plotAreaButtomY, plotAreaHeight;  BarChartModel model;  RectF bar;    public BarChartView(Context context) {  super(context);  setBackgroundColor(Color.WHITE);    barPaint = new Paint(Paint.ANTI\_ALIAS\_FLAG);  barPaint.setColor(Color.BLUE);  barPaint.setStyle(Style.FILL);    titleTextPaint = new Paint(Paint.ANTI\_ALIAS\_FLAG);  titleTextPaint.setColor(Color.BLACK);  titleTextPaint.setTextAlign(Align.CENTER);  titleTextPaint.setTypeface(Typeface.DEFAULT\_BOLD);    labelTextPaint = new Paint(Paint.ANTI\_ALIAS\_FLAG);  labelTextPaint.setColor(Color.DKGRAY);  labelTextPaint.setTextAlign(Align.CENTER);    model = new BarChartModel();  bar = new RectF();  }    public float getBarLeftX(int barIndex){  return barPading \* (barIndex + 1) + barWidth \* barIndex;  }    @Override  public void onDraw(Canvas canvas){  // Title  canvas.drawText("Annual Sales", totalWidth/2, titleY, titleTextPaint);    // Horizontal line below bars  canvas.drawLine(0, plotAreaButtomY , totalWidth, plotAreaButtomY, labelTextPaint);    //Bars  for(int i = 0; i < model.size();i++){  //bar coordinates  float leftX = getBarLeftX(i);  float rightX = leftX + barWidth;  float topY = plotAreaButtomY - plotAreaHeight \* model.getValue(i)/model.maxValue();  //bar  bar.left = leftX;  bar.right = rightX;  bar.top = topY;  bar.bottom = plotAreaButtomY;  canvas.drawRect(bar, barPaint);  //Label  float labelX = (leftX + rightX)/2;  canvas.drawText(model.getLabel(i), labelX, labelY, labelTextPaint);  //Value Text  //String valueText = ((int) model.getValue(i)) + "";  String valueText = String.format("%.0f", model.getValue(i));  canvas.drawText(valueText, labelX, topY-labelPading, labelTextPaint);  }  }    @Override  protected void onSizeChanged(int w, int h, int oldw, int oldh){  totalWidth = w;  float viewPading = h \* 0.03f;  float titlePading = h \* 0.05f;  labelPading = h \* 0.01f;  barPading = w \* 0.1f;  float titleTextSize = h \* 0.07f;  float labelTextSize = h \* 0.04f;  titleY = viewPading + titleTextSize;  labelY = h - viewPading;    float plotAreaTopY = viewPading + titleTextSize + titlePading + labelTextSize + labelPading;  plotAreaButtomY = h - viewPading - labelTextSize - labelPading;  plotAreaHeight = plotAreaButtomY - plotAreaTopY;    //Bar width  int numBars = model.size();  barWidth = (totalWidth - (numBars + 1)\*barPading)/numBars;    //Font sizes  titleTextPaint.setTextSize(titleTextSize);  labelTextPaint.setTextSize(labelTextSize);  }  } |

### 5.1.3 Code Review : Class Variables

We have a lot of variables in this example. There are several Paint objects for bars and text.

|  |
| --- |
| private Paint barPaint;  private Paint titleTextPaint;  private Paint labelTextPaint; |

The following variables are used to store the width, padding and y-coordinates of different components of the chart. These variables are set in onSizeChanged() method and then used in onDraw() method to draw the chart.

|  |
| --- |
| **float** totalWidth;  **float** barWidth;  **float** barPading;  **float** labelPading;  **float** titleY;  **float** labelY;  **float** plotAreaButtomY;  **float** plotAreaHeight; |

This is the model for our chart. It will be created in the constructor.

|  |
| --- |
| **private** BarChartModel model; |

Remember that mobile devices have limited resources. You should avoid creating too many objects to save memory. We will reuse one RectF object for all bars.

|  |
| --- |
| **private** RectF bar; |

### 5.1.4 Code Review : Constructor

In the constructor we set view background color, created paint objects and the chart model.

### 5.1.5 Code Review : onSizeChnged()

In this method we calculated the width, padding and y-coordinates of different components of the chart. We used percentages in our calculations. For example, the horizontal bar padding is 10% of the view width.

|  |
| --- |
| barPading = w \* 0.1f; |

The label font size is 4% of the view height.

|  |
| --- |
| **float** labelTextSize = h \* 0.04f; |

In this example we used the same percentages for both portrait and landscape orientations, but you can use different values in your application.

#### Plot Area Coordinates

The plot area top coordinate is calculated relative to the view top:

|  |
| --- |
| plotAreaTopY = viewPading + titleTextSize + titlePading +  labelTextSize + labelPading; |

The plot area bottom coordinate is calculated relative to the view bottom:

|  |
| --- |
| plotAreaButtomY = h - viewPading - labelTextSize - labelPading; |

#### Bar Width

If you look at Figure 5-1 you will see that there are 3 bars and 4 horizontal paddings (spaces). Horizontal padding is fixed. It is 10% of the view width.

|  |
| --- |
| barPading = w \* 0.1f; |

The total padding is

|  |
| --- |
| totalPadding = (numBars + 1)\*barPading; |

For 3 bars, the total padding as percentage of the view width is:

|  |
| --- |
| totalPadding = (3 + 1)\* 10% = 40% |

To get the bat width we have to subtract the total padding from the view width(100%) and divide the result by the number of bars(3):

|  |
| --- |
| barWidth = (100% - 40%)/3 = 20%; |

The generic formula used in our example is shown below.

|  |
| --- |
| barWidth = (totalWidth - (numBars + 1)\*barPading)/numBars; |

### 5.1.6 Code Review :

In this method we drew all components of the chart. We started with the title.

|  |
| --- |
| canvas.drawText("Annual Sales", totalWidth/2, titleY, titleTextPaint); |

Then we drew a horizontal line below bars(see Figure 4-8).

|  |
| --- |
| canvas.drawLine(0, plotAreaButtomY , totalWidth, plotAreaButtomY, labelTextPaint); |

Finally we iterated over bar data, calculated x- and y-coordinates of each bar, drew the bar and labels.

#### Bar Coordinates

The following method is used to calculate the left coordinate of each bar.

|  |
| --- |
| **public** **float** getBarLeftX(**int** barIndex){  **return** barPading \* (barIndex + 1) + barWidth \* barIndex;  } |

The formula is very simple. We calculate number of paddings and bars in front of each bar. Note that the barindex stars from 0. For example, there are 2 paddings and 1 bar in front of the second bar (barindex=1):

|  |
| --- |
| leftX = barPading \* (1 + 1) + barWidth \* 1; |

The right coordinate is calculated as follow:

|  |
| --- |
| rightX = leftX + barWidth; |

Mow let’s talk about y-coordinates. We draw bars in the plot area of the chart view as shown in Figure 5-3.

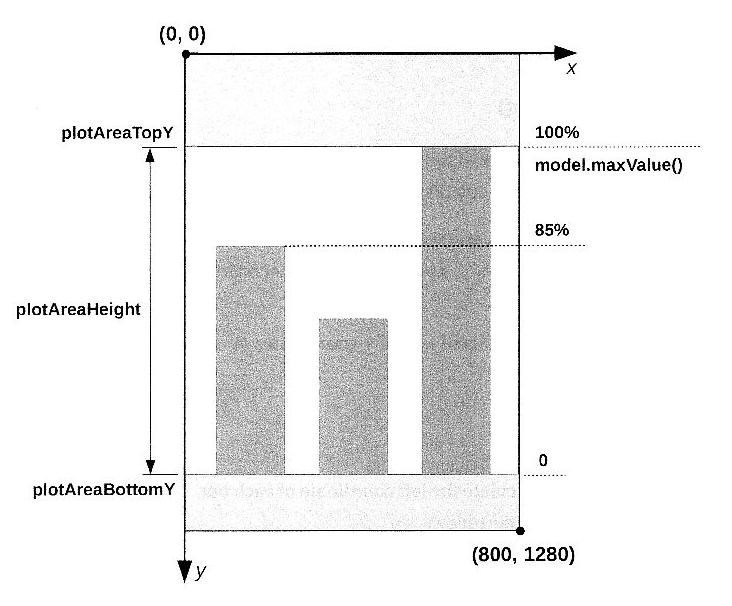


Figure 5-3.

Note that the bar chart view uses device coordinates with the origin at the upper-left corner of the view. The x-coordinate increases to the right, and the y-coordinate increases downward.

The bottom coordinate of each bar is equal to the plot area bottom coordinate. The top coordinate depends on the bar value (sales). Our bar chart model has maxValue() method which returns the value corresponding to the tallest bar on the bar chart. The height of the tallest bar is equal to the height of the plot area. The top coordinate of the tallest bar can be calculated as follow:

|  |
| --- |
| topY = plotAreaBottomY – plotAreaHeight; |

We can use ratios or percentages to find the height of a bar relative to the height of the tallest bar. For example, the height of the first bar is 85% of the tallest bar or the plot area height:

|  |
| --- |
| barHeight = plotAreaHeight \* model.getValue(0)/model.maxValue()  = plotAreaHeight \* 6535/7654  = plotAreaHeight \*0.85 |

Then the top coordinate is

|  |
| --- |
| topY = plotAreaBottomY – barHeight  = plotAreaBottomY – plotAreaHeight \* model.getValue(i)/model.maxValue() |

#### Labels

For each bar we drew two labels. A year labels below the bar

|  |
| --- |
| float labelX = (leftX + rightX)/2;  canvas.drawText(model.getLabel(i), labelY, labelTextPaint); |

and value label above the bar

|  |
| --- |
| String valueText = String.format(“%.0f”, model.getValue(i));  canvas.drawText(valueText, labelX, topY-labelPading, labelTextPaint); |

### 5.1.7 Screenshots

Screenshots of the application in the portrait and landscape modes are shown in Figure 5-1 and Figure 5-4.

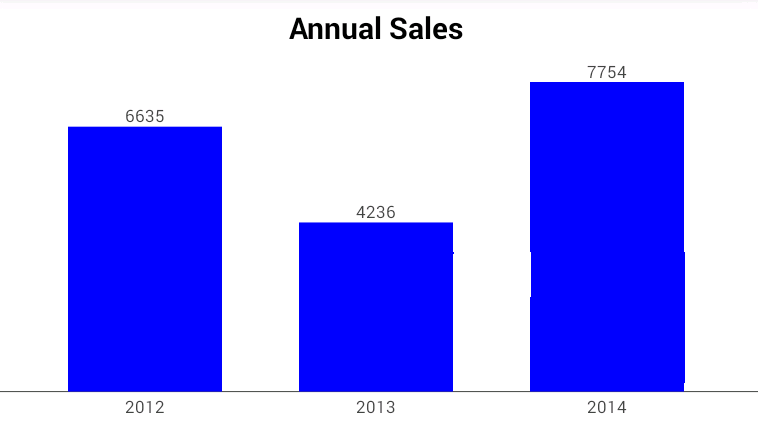
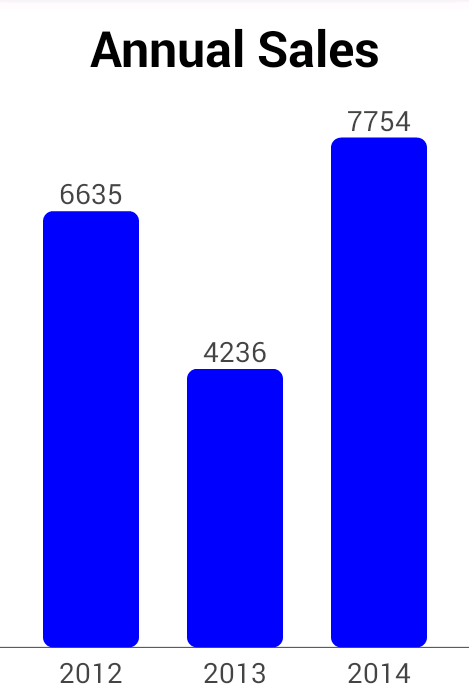


Figure 5-4.

If you replace drawRect() with the drawRoundRect() method

|  |
| --- |
| canvas.drawRoundRect(rect, 10, 10, barPaint); |

The bars will have rounded corners, as shown in Fugure 5-5.



Fugure 5-5.

# Chapter 6 Circles, Ovals and Arcs

In this chapter you will learn how to draw circles, ovals and arcs.

## 6.1 Circles

To draw a circle you can use the drawCircle method of the Canvas class. A circle will be filled or stroked based on the style of a paint.

|  |
| --- |
| void drawCircle(float cx, float cy, float radius, Paint paint) |

The drawCircle method has the following parameters:

* cx, cy – the x- and y-coordinate of the center of the circle.
* radius – the radius of the circle.
* paint – a paint object.

The example below shows how to draw a circle.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Style;  **import** android.view.View;  **public** **class** CircleView **extends** View{    **private** **int** centerX;  **private** **int** centerY;  **private** **float** radius;  **private** Paint paint;    **public** CircleView(Context context) {  **super**(context);  setBackgroundColor(Color.***WHITE***);  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setStyle(Style.***STROKE***);  paint.setColor(0xff000099);  }    @Override  **public** **void** onDraw(Canvas canvas){  paint.setStrokeWidth(radius \* 0.3f);  canvas.drawCircle(centerX, centerY, radius, paint);  }    @Override  **protected** **void** onSizeChanged(**int** w, **int** h, **int** oldw, **int** oldh)  {  centerX = w/2;  centerY = h/2;  radius = Math.*min*(w, h)\*0.7f /2;  }  } |

Let’s review the code.

In the constructor we set the view background color and created a Paint.

In onSizeChanged() method we calculated coordinated of the center of the circle

|  |
| --- |
| centerX = w/2;  centerY = h/2; |

and the circle radius

|  |
| --- |
| radius = Math.*min*(w, h)\*0.7f /2; |

We wanted the circle diameter to be 70% of the short side of the view. Therefore the radius is 70%/2.

In onDraw() method we set the line (stroke) width to 30% of the radius and drew a circle.

|  |
| --- |
| paint.setStrokeWidth(radius \* 0.3f);  canvas.drawCircle(centerX, centerY, radius, paint); |

A screenshot of the application is shown in Figure 6-1.

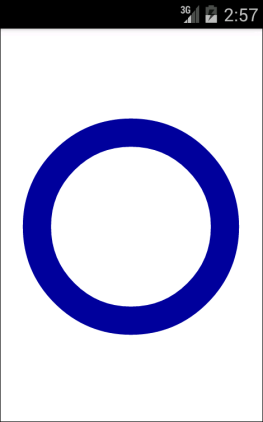


Figure 6-1.

## 6.2 Ovals

The drawOval method of the Canvas class can be used to draw both ovals and circles.

|  |
| --- |
| void drawOval(RectF rect, Paint paint) |

The drawOval method has the following parameters:

* **rect** – a rectangle that defines the bounds of the oval. If the width and height of the rectangle are equal, a circle is drawn. The rectangle is represented by the coordinates of its four edges: left, right, top, and bottom. Left and right are x-coordinates, and top and bottom are y-coordinates.
* paint – a paint object.

In the following example we will draw both an oval and a rectangle that defines the bounds of the oval.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Style;  **import** android.graphics.RectF;  **import** android.view.View;  **public** **class** OvalView **extends** View{  **private** Paint paint;  **private** **float** padding;  **private** RectF rect;  **public** OvalView(Context context) {  **super**(context);  setBackgroundColor(Color.***WHITE***);  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setStyle(Style.***STROKE***);  rect = **new** RectF();  }    @Override  **public** **void** onDraw(Canvas canvas)  {  paint.setStrokeWidth(padding);  paint.setColor(Color.***RED***);  canvas.drawOval(rect, paint);    paint.setStrokeWidth(padding/12);  paint.setColor(0xff000077);  canvas.drawRect(rect, paint);    }    @Override  **protected** **void** onSizeChanged(**int** width, **int** height, **int** oldw, **int** oldh)  {  padding = Math.*max*(width, height) \* 0.1f;  rect.left = padding;  rect.right = width - padding;  rect.top = padding;  rect.bottom = height - padding;  }  } |

Let’s review the code. In the constructor we set the view background color and created a paint and a rectangle.

In onSizeChanged() method we set coordinates of the rectangle that defines the bounds of the oval.

|  |
| --- |
| padding = Math.*max*(width, height) \* 0.1f;  rect.left = padding;  rect.right = width - padding;  rect.top = padding;  rect.bottom = height - padding; |

In onDraw() method we drew the oval first

|  |
| --- |
| paint.setStrokeWidth(padding);  paint.setColor(Color.***RED***);  canvas.drawOval(rect, paint); |

and then the rectangle.

|  |
| --- |
| paint.setStrokeWidth(padding/12);  paint.setColor(0xff000077);  canvas.drawRect(rect, paint); |

A screenshot of the application is shown in Figure 6-2.

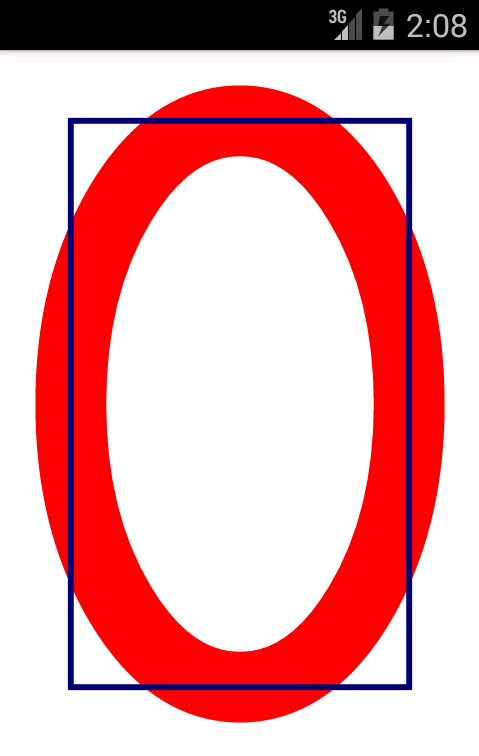


Figure 6-2.

We used different line width for the rectangle and the oval to show you how the oval is positioned relative to its bounding rectangle. As you can see from Figure 6-2, the oval goes half line width outside the bounding rectangle.

## 6.3 Arcs

Arcs are oval segments. The drawArc method of the Canvas class is used to draw arcs.

|  |
| --- |
| void drawArc(RectF rect, float startAngle, float sweepAngle, Boolean useCenter, Paint paint) |

The drawArc method has the following parameters:

* **rect** – a bounding rectangle which defines the shape and size of the arc.
* **startAngle** – an angle in degrees where the arc begins. An angle of zero degrees corresponds to 3 o’clock on a watch. Both positive or negative values can be used.
* **sweepAngle** – an angle in degrees measured clockwise from the startAngle parameter to ending point of the arc. Both positive or negative values can be used.
* **useCenter** – if true, a wedge is drawn. Based on the style of the paint the wedge can be filled or stroked.
* **paint** – a paint object.

The example below shows how to draw arcs.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Cap;  **import** android.graphics.Paint.Style;  **import** android.graphics.RectF;  **import** android.view.View;  **public** **class** ArcView **extends** View{  **private** **int** centerX;  **private** **int** centerY;  **private** **float** radius;  **private** RectF rect;  **private** Paint paint;  **public** ArcView(Context context) {  **super**(context);  setBackgroundColor(Color.***WHITE***);  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  rect = **new** RectF();  }  @Override  **public** **void** onDraw(Canvas canvas)  {  paint.setStyle(Style.***FILL***);  paint.setColor(Color.***LTGRAY***);  canvas.drawOval(rect, paint);    paint.setStyle(Style.***STROKE***);  paint.setStrokeCap(Cap.***ROUND***);  paint.setColor(0xff000099);  canvas.drawArc(rect, 0, 90, **false**, paint);    paint.setStyle(Style.***FILL\_AND\_STROKE***);  paint.setColor(0xff339933);  canvas.drawArc(rect, 180, 90, **true**, paint);  }  @Override  **protected** **void** onSizeChanged(**int** w, **int** h, **int** oldw, **int** oldh){  centerX = w/2;  centerY = h/2;  radius = Math.*min*(w, h)\*0.7f/2;  paint.setStrokeWidth(radius\*0.3f);  rect.left = centerX - radius;  rect.right = centerX + radius;  rect.top = centerY - radius;  rect.bottom = centerY + radius;  }  } |

A screenshot of the example is shown in Figure 6-3.

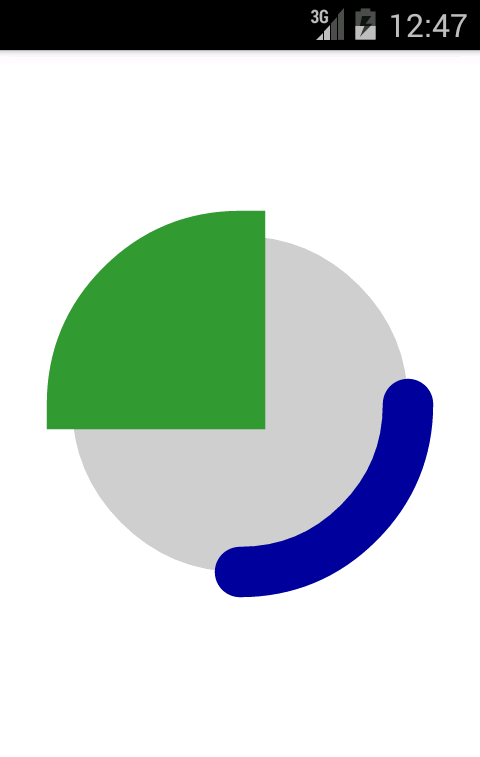


Figure 6-3.

Let’s review the code. In the constructor we set the view background color and created a paint and a rectangle.

In onSizeChange() method we set coordinates of the rectangle that defines the bounds of the circle. The circle diameter is 70%of the short side of the view and consequently the radius is 70% / 2.

|  |
| --- |
| centerX = w/2;  centerY = h/2;  radius = Math.*min*(w, h)\*0.7f/2;  paint.setStrokeWidth(radius\*0.3f);  rect.left = centerX - radius;  rect.right = centerX + radius;  rect.top = centerY - radius;  rect.bottom = centerY + radius; |

Also we set the stroke width to 30% of the circle radius.

|  |
| --- |
| paint.setStrokeWidth(radius\*0.3f); |

In onDraw() method we drew three shapes.

1. A filled circle.

|  |
| --- |
| paint.setStyle(Style.***FILL***);  paint.setColor(Color.***LTGRAY***);  canvas.drawOval(rect, paint); |

Note that we used the drawOval() method to draw the circle.

1. An arc with rounded line caps.

|  |
| --- |
| paint.setStyle(Style.***STROKE***);  paint.setStrokeCap(Cap.***ROUND***);  paint.setColor(0xff000099);  canvas.drawArc(rect, 0, 90, **false**, paint); |

Note how the arc extends half line width outside the bounding rectangle(the filled circle).

1. A wedge.

|  |
| --- |
| paint.setStyle(Style.***FILL\_AND\_STROKE***);  paint.setColor(0xff339933);  canvas.drawArc(rect, 180, 90, **true**, paint); |

## 6.4 Examples : Circular Chart

In this example we will draw a circular chart shown in Figure 6-4. It can be used to display statistical data or as a progress indicator.

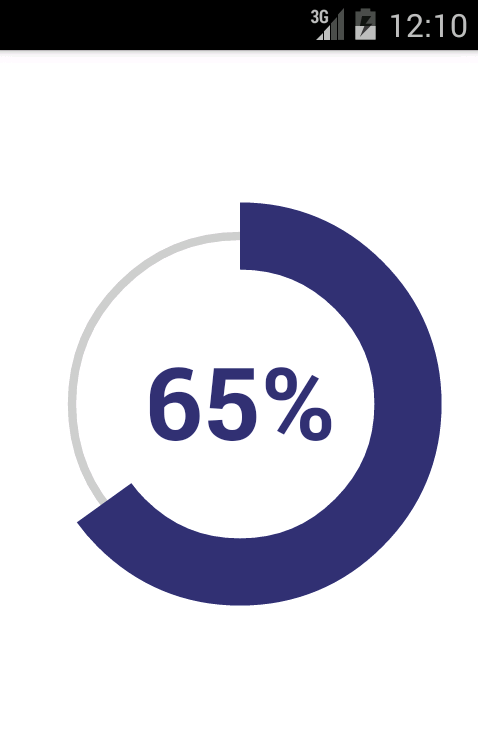


Figure 6-4.

The code of the application content view is shown below.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Style;  **import** android.graphics.Rect;  **import** android.graphics.RectF;  **import** android.graphics.Typeface;  **import** android.view.View;  **public** **class** CircularChart **extends** View{  **private** **float** value;  **private** String label;  **private** **int** centerX;  **private** **int** centerY;  **private** **float** radius;  **private** **float** textX;  **private** **float** textY;  **private** RectF arcOval;  **private** Rect textBounds;  **private** Paint paint;  **private** Paint txtPaint;  **private** **int** color1 = Color.***LTGRAY***;  **private** **int** color2 = 0xff333377;  **public** CircularChart(Context context) {  **super**(context);  setBackgroundColor(Color.***WHITE***);  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setStyle(Style.***STROKE***);  txtPaint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  txtPaint.setColor(color2);  txtPaint.setTypeface(Typeface.***DEFAULT\_BOLD***);  arcOval = **new** RectF();  textBounds = **new** Rect();  //Set value  value = 0.65f;  label = String.*format*("%.0f%%", value \* 100);  }  @Override  **public** **void** onDraw(Canvas canvas){  //Draw circle  paint.setStrokeWidth(radius\*0.05f);  paint.setColor(color1);  canvas.drawCircle(centerX, centerY, radius, paint);  //Draw arc  paint.setStrokeWidth(radius\*0.4f);  paint.setColor(color2);  canvas.drawArc(arcOval, -90, value\*360, **false**, paint);  //Draw label  canvas.drawText(label, textX, textY, txtPaint);  }  @Override  **protected** **void** onSizeChanged(**int** w, **int** h, **int** oldw, **int** oldh){  //Center  centerX = w/2;  centerY = h/2;  //Radius  radius = Math.*min*(w, h)\*0.7f/2;  //Text size  txtPaint.setTextSize(radius\*0.6f);  //Center text  txtPaint.getTextBounds(label, 0, label.length(), textBounds);  textX = centerX - (textBounds.left + textBounds.right)\*0.5f;  textY = centerY - (textBounds.top + textBounds.bottom)\*0.5f;  //Arc oval  arcOval.left = centerX-radius;  arcOval.right = centerX+radius;  arcOval.top = centerY-radius;  arcOval.bottom = centerY+radius;  }  } |

### 6.4.1 Code Review : Constructor

First, we set the view background color and created Paint objects.

Then we created two temporary rectangles to draw an arc.

|  |
| --- |
| arcOval = **new** RectF(); |

and to measure text.

|  |
| --- |
| textBounds = **new** Rect(); |

These rectangles are used in onSizeChanged() method.

Finally, we set the chart value and its corresponding label(65%).

|  |
| --- |
| value = 0.65f;  label = String.*format*("%.0f%%", value \* 100); |

### 6.4.2 Code Review : onSizeChnged()

First, we calculated the x- and y-coordinates of the center of the view

|  |
| --- |
| centerX = w/2;  centerY = h/2; |

and the radius of the circle

|  |
| --- |
| radius = Math.*min*(w, h)\*0.7f/2; |

We wanted the circle diameter to be 70% of the short side of the view. Therefore the radius is 70%/2.

Then we set a text size to 60% of the radius.

|  |
| --- |
| txtPaint.setTextSize(radius\*0.6f); |

After that we measured the labels size.

|  |
| --- |
| txtPaint.getTextBounds(label, 0, label.length(), textBounds); |

and calculated the x- and y-coordinates to center-align the label horizontally and vertically.

|  |
| --- |
| textX = centerX - (textBounds.left + textBounds.right)\*0.5f;  textY = centerY - (textBounds.top + textBounds.bottom)\*0.5f; |

We explained these formula in chapter 3 when discussed horizontal (section 3.3) and vertical (section 3.4) text alignment.

Finally we set the arc coordinates.

|  |
| --- |
| arcOval.left = centerX-radius;  arcOval.right = centerX+radius;  arcOval.top = centerY-radius;  arcOval.bottom = centerY+radius; |

### 6.4.3 Code Review : onDraw()

First we drew a circle to outline the chart.

|  |
| --- |
| paint.setStrokeWidth(radius\*0.05f);  paint.setColor(color1);  canvas.drawCircle(centerX, centerY, radius, paint); |

Then we drew an arc. The sweep angle or the arc is proportional to the value it represents. In our example, the full circle (360 degrees) represents 100%. Therefore the angle corresponding to 65% is 360\*0.65. We wanted the arc to start at 12 o’clock, therefore we used the start angle value of -90 degrees.

|  |
| --- |
| paint.setStrokeWidth(radius\*0.4f);  paint.setColor(color2);  canvas.drawArc(arcOval, -90, value\*360, **false**, paint); |

Finally we drew the label.

|  |
| --- |
| canvas.drawText(label, textX, textY, txtPaint); |

# Chapter 07 Gradients

So far, in our previous examples, we filled shapes with solid colors. In this chapter you will learn how to use gradients. A gradient fill is a fill that varies from one color to another. Canvas API supports several gradient types. Let’s discuss most common ones.

## 7.1 Linear Gradients

A linear gradient varies along the line, also called gradient axis, between two points. Figure 7-1 shows an example of a linear gradient which varies along the vertical line that goes from the top of a rectangle to its bottom.



Figure 7-1.

To fill a shape with a linear gradient you have to:

1. Create an instance of the LinearGradient class.

2. Configure a Paint object to use the gradient you created,

The next several examples shows how to use different linear gradients.

### 7.1.1 Example: Vertical Gradient

In this example we will draw a gradient shown in Figure 7-1. The code of the application content view is shown below.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.LinearGradient;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Style;  **import** android.graphics.Shader;  **import** android.view.View;  **public** **class** LinearGradientView **extends** View  {  **private** **int** color1 = 0xff00ff00;  **private** **int** color2 = 0xff005500;  **private** **int** width, height;  **private** LinearGradient gradient;  **private** Paint paint;  **public** LinearGradientView(Context context) {  **super**(context);  setBackgroundColor(Color.***WHITE***);  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setStyle(Style.***FILL***);  }  @Override  **public** **void** onDraw(Canvas canvas)  {  canvas.drawRect(0,0,width, height, paint);  }  @Override  **protected** **void** onSizeChanged(**int** width, **int** height, **int** oldw, **int** oldh)  {  **this**.width = width;  **this**.height = height;  gradient = **new** LinearGradient(0,0,0,height, color1, color2, Shader.TileMode.***CLAMP***);  paint.setShader(gradient);  }  } |

#### Code Review

First, we defined two colors we will use to create a gradient.

|  |
| --- |
| **private** **int** color1 = 0xff00ff00;  **private** **int** color2 = 0xff005500; |

In the constructor we set the background color of the view, created a paint object and set its style to Fill.

In onSizeChanged() method we stored the view width and height we will use to draw (fill) a rectangle.

|  |
| --- |
| **this**.width = width;  **this**.height = height; |

Then we created an instance of the LinearGradient class.

|  |
| --- |
| gradient = **new** LinearGradient(0,0,0,height, color1, color2, Shader.TileMode.***CLAMP***); |

We use the LinearGradient constructor shown below.

|  |
| --- |
| LinearGradient(float x1, float y1, float x2, float y2, int color1, int color2, Shader.TileMode tile); |

The constructor has the following parameters:

* x1, y1 - the x- and y-coordinate of the starting point of the gradient line.
* x2, y2 - the x- and y-coordinate of the end point of the gradient line.
* color1 – the color at the start of the gradient line.
* color2 – the color at the end of the gradient line.
* tile – the tile mode specifies how to draw a gradient when the area being filled is bigger the area defined by the gradient. In this example, both areas are the same, so this parameter does not affect the gradient. The tile mode can have one of the following values: CLAMP, MIRROR, REPEAT. We will discuss this parameter later in this section.

The gradient axis in this example is a vertical line which goes from the top of a rectangle to its bottom.

After we created the gradient, we configured the paint object to use this gradient to fill a rectangle.

|  |
| --- |
| paint.setShader(gradient); |

Note that LinearGradient class extends the basic class called Shader. The Paint class uses setShader() method to set any shader object, including LinearGradient.

Finally, in onDraw() method we filled the rectangle with the gradient created in onSizeChanged() method.

|  |
| --- |
| canvas.drawRect(0,0,width, height, paint); |

A screenshot of this application is shown in Figure 7-1.

### 7.1.2 Example : Horizontal Gradient

In this example we will draw a horizontal gradient shown in Figure 7-2.



Figure 7-2.

To create this gradient we have to change the LinearGradient constructor as shown below.

|  |
| --- |
| gradient = **new** LinearGradient(0,0,width,0, color1, color2, Shader.TileMode.***CLAMP***); |

In this example we used horizontal gradient axis which goes from the left side of the rectangle to its right side.

### 7.1.3 Example : Diagonal Gradient

To draw the gradient shown in Figure 7-3 we have to change the LinearGradient constructor again.



Figure 7-3

|  |
| --- |
| gradient = **new** LinearGradient(0,0,width,height, color1, color2, Shader.TileMode.***CLAMP***); |

In this example we used the gradient axis which goes from the upper-left corner of the rectangle to its lower-right corner.

### 7.1.4 Tile Mode

The gradient tile mode specifies how to draw a gradient when the area being filled is bigger than the area defined by the gradient. The tile mode can have one of the following values.

* CLAMP – use the terminal colors of the gradient to fill the remainder of the region.
* MIRROR – mirror the same linear gradient along the gradient axis before the starting point and after the end point.
* REPEAT – repeat the same linear gradient along the gradient axis before the starting point and after the end point.

The next several examples show how to use different gradient tile modes.

#### CLAMP Mode

In this example we will draw a vertical gradient shown in Figure 7-4.



Figure 7-4

|  |
| --- |
| gradient = **new** LinearGradient(0,0,0,height/2, color1, color2, Shader.TileMode.***CLAMP***); |

The vertical gradient axis goes from the top of a rectangle to its center (height/2). The remainder of the rectangle (from height/2 to height) is filled with the end color (color2).

#### MIRROR Mode

In this example we will drawn a vertical gradient shown in Figure 7-5.

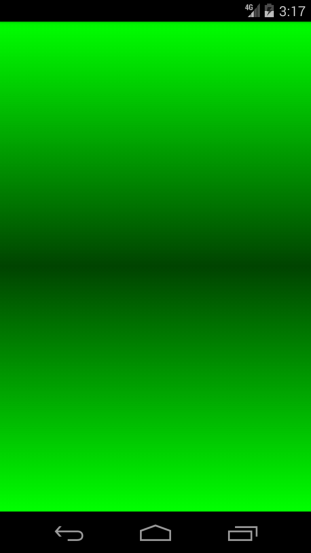


Figure 7-5.

New version of the constructor is shown below.

|  |
| --- |
| gradient = **new** LinearGradient(0,0,0,height/2, color1, color2, Shader.TileMode.***MIRROR***); |

Similar to the previous example the vertical gradient axis goes from the top of a rectangle to its center (height/2). The remainder of the rectangle (from height/2 to height) is filled with the mirrored gradient.

#### REPEAT Mode

In this example we will draw a vertical gradient shown in Figure 7-6.



Figure 7-6.

The modified version of constructor is shown below.

|  |
| --- |
| gradient = **new** LinearGradient(0,0,0,height/2, color1, color2, Shader.TileMode.***REPEAT***); |

Similar to the previous example the vertical gradient axis goes from the top of a rectangle to its center (height/2). The remainder of the rectangle(from height/2 to height) is filled with the same gradient.

## 7.2 Radial Gradients

A radial gradient is a fill that varies radically along the radius of a circle as shown in Figure 7-7. The next several examples show how to use different radial gradients.

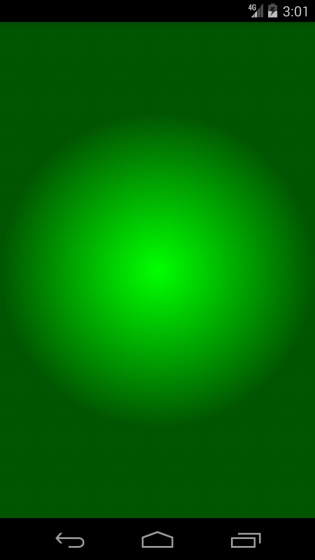


Figure 7-7.

### 7.2.1 Example : CLAMP Mode

The CLAMP tile model is the most common for radial gradients. In this example we will draw a radial gradient shown in Figure 7-7. The code of the application content view is shown below.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.R.color;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Style;  **import** android.graphics.RadialGradient;  **import** android.graphics.Shader;  **import** android.view.View;  **public** **class** RadialGradientView **extends** View{  **private** **int** color1 = 0xff00ff00;  **private** **int** color2 = 0xff005500;  **private** **int** width, height;  **private** RadialGradient gradient;  **private** Paint paint;  **public** RadialGradientView(Context context) {  **super**(context);  setBackgroundColor(color.***white***);  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setStyle(Style.***FILL***);  }  @Override  **public** **void** onDraw(Canvas canvas){  canvas.drawRect(0, 0, width, height, paint);  }    @Override  **protected** **void** onSizeChanged(**int** width, **int** height, **int** oldw, **int** oldh){  **this**.width = width;  **this**.height = height;  **float** centerX = width/2.0f;  **float** centerY = height/2.0f;  **float** radius = Math.*min*(width, height) / 2.0f;  gradient = **new** RadialGradient(centerX, centerY, radius, color1, color2, Shader.TileMode.***CLAMP***);  paint.setShader(gradient);  }  } |

#### Code Review

We used the same colors as in linear gradient examples.

|  |
| --- |
| **private** **int** color1 = 0xff00ff00;  **private** **int** color2 = 0xff005500; |

The constructor and onDraw() method are also the same.

In onSizeChanged() method we stored the view width and height

|  |
| --- |
| **this**.width = width;  **this**.height = height; |

and calculated the center coordinated and the radius of the radial gradient

|  |
| --- |
| **float** centerX = width/2.0f;  **float** centerY = height/2.0f;  **float** radius = Math.*min*(width, height) / 2.0f; |

Then we created an instance of the RadialGradient class.

|  |
| --- |
| gradient = **new** RadialGradient(centerX, centerY, radius, color1, color2, Shader.TileMode.***CLAMP***); |

We used the RadialGradient constructor shown below.

|  |
| --- |
| RadialGradient(float cx, float cy, float radius, int color1, int color2, Shader.TileMode tile) |

The constructor has the following parameters:

* cx, cy – the x- and y- coordinate of the center of the gradient circle.
* radius – the radius of the gradient circle.
* color1 – the color at the center of the circle.
* color2 – the color at the edge of the circle.
* tile – the tile mode which specifies how to draw a gradient when the area being filled is bigger than the area defined by the gradient.

After we created the gradient, we configured the paint object to use this gradient to fill a rectangle.

|  |
| --- |
| paint.setShader(gradient); |

In this example we used CLAMP tile mode, therefore the area outside the gradient circle is filled with color2.

A screenshot of this application is shown in Figure 7-7.

### 7.2.2 Example:3D Ball

Radial gradients can be used to create 3D effects. In this example we will draw a 3D ball shown in Figure 7-8.

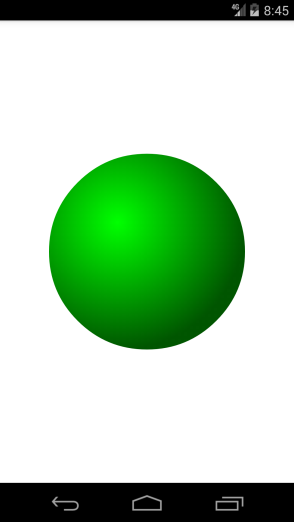


Figure 7-8.

Let’s look at the code first and then review it.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Style;  **import** android.graphics.RadialGradient;  **import** android.graphics.Shader;  **import** android.view.View;  **public** **class** RadialGradientView2 **extends** View  {  **private** **int** color1 = 0xff00ff00;  **private** **int** color2 = 0xff005500;  **private** **int** width, height;  **private** **float** centerX, centerY, radius;  **private** RadialGradient gradient;  **private** Paint paint;  **public** RadialGradientView2(Context context)  {  **super**(context);  setBackgroundColor(Color.***WHITE***);  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setStyle(Style.***FILL***);  }    @Override  **public** **void** onDraw(Canvas canvas)  {  canvas.drawCircle(centerX, centerY, radius, paint);  }    @Override  **protected** **void** onSizeChanged(**int** width, **int** height, **int** oldw, **int** oldh)  {  centerX = width /2.0f;  centerY = height /2.0f;  radius = Math.*min*(width, height) / 3.0f;  gradient = **new** RadialGradient(centerX-radius\*0.3f, centerY-radius\*0.3f,radius\*1.3f,color1, color2, Shader.TileMode.***CLAMP***);  paint.setShader(gradient);  }  } |

#### Code Review

We used the same colors as in previous examples.

|  |
| --- |
| **private** **int** color1 = 0xff00ff00;  **private** **int** color2 = 0xff005500; |

The constructor is also the same.

In onSizeChanged() method we calculated coordinates of the center of the view and the circle radius.

|  |
| --- |
| centerX = width /2.0f;  centerY = height /2.0f;  radius = Math.*min*(width, height) / 3.0f; |

Then we created an instance of RadialGradient class and configured the Paint object to use this gradient to fill the circle.

|  |
| --- |
| gradient = **new** RadialGradient(centerX-radius\*0.3f,  centerY-radius\*0.3f,radius\*1.3f,color1, color2,  Shader.TileMode.***CLAMP***);  paint.setShader(gradient); |

Note how we shifted the center of the gradient left and up 30% of the radius relative to the center of the circle.

|  |
| --- |
| gradientCx = centerX-radius\*0.3f;  gradientCy = centerY-radius\*0.3f; |

Also we increased the radius of the gradient 30% to compensate for the shift.

|  |
| --- |
| gradientRadius = radius\*1.3f; |

In onDraw() method we drew (filled) the circle.

|  |
| --- |
| canvas.drawCircle(centerX, centerY, radius, paint); |

### 7.2.3 Example : REPEAT Mode

You can also use radial gradients to create interesting effects. In this example we will draw the intricate circular pattern shown in Figure 7-9.

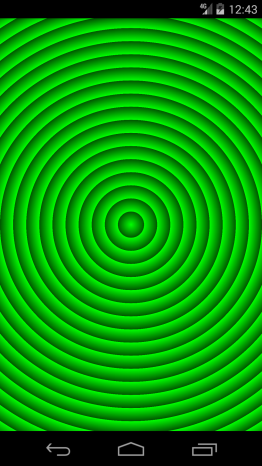


Figure 7-9.

To draw this gradient we reused the code from the first radial gradient example and modified the radius and the tile mode as shown below.

|  |
| --- |
| **float** radius = Math.*min*(width, height) / 20.0f;  gradient = **new** RadialGradient(centerX, centerY, radius, color1, color2,  Shader.TileMode.***REPEAT***); |

## 7.3 Sweep Gradients

A sweep gradient also known as angle gradient is a fill that varies in a clockwise sweep around the center as shown in Figure 7-10.



Figure 7-10.

### 7.3.1 Example

In this example we will draw a sweep gradient shown in Figure 7-10. The code of the application content view is shown below.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.R.color;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Style;  **import** android.graphics.SweepGradient;  **import** android.view.View;  **public** **class** SweepGradientView **extends** View {  **private** **int** color1 = 0xff00ff00;  **private** **int** color2 = 0xff110000;  **private** SweepGradient gradient;  **private** Paint paint;  **private** **float** centerX;  **private** **float** centerY;  **private** **float** radius;  **public** SweepGradientView(Context context){  **super**(context);  setBackgroundColor(color.***white***);  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setStyle(Style.***STROKE***);  }  @Override  **protected** **void** onSizeChanged(**int** width, **int** height, **int** oldw, **int** oldh){  centerX = width / 2.0f;  centerY = height / 2.0f;  radius = Math.*min*(width, height) \* 0.7f/2.0f;  gradient = **new** SweepGradient(centerX, centerY, color1, color2);  paint.setShader(gradient);  paint.setStrokeWidth(radius/2);  }  @Override  **public** **void** onDraw(Canvas canvas){  canvas.drawCircle(centerX, centerY, radius, paint);  }  } |

#### Code Review

We use the same colors as in linear and radial gradient examples.

|  |
| --- |
| **private** **int** color1 = 0xff00ff00;  **private** **int** color2 = 0xff005500; |

In the constructor we set the view background color, created a paint object and set its style to STROKE.

In onSizeChanged() method we calculated the center coordinates and the radius of a circle we will drive.

|  |
| --- |
| centerX = width / 2.0f;  centerY = height / 2.0f;  radius = Math.*min*(width, height) \* 0.7f/2.0f; |

Then we created a SweepGradient object.

|  |
| --- |
| gradient = **new** SweepGradient(centerX, centerY, color1, color2); |

We used the SweepGradient constructor shown below.

|  |
| --- |
| SweepGradient(float cx, float cy, int color1, int color2); |

The constructor has the following parameters:

* cx, cy – the x- and y- coordinate of the gradient center.
* color1 – the color at the start o the sweep.
* color2 – the color at the end o the sweep.

After we created the gradient, we configured the paint object to use this gradient to draw a circle.

|  |
| --- |
| paint.setShader(gradient); |

Finally we set line width to 50

|  |
| --- |
| paint.setStrokeWidth(radius/2); |

In onDraw() method we drew the circle.

|  |
| --- |
| canvas.drawCircle(centerX, centerY, radius, paint); |

A screenshot of the application is shown in Figure 7-10. As you can see from the screenshot, the sweep starts at 0 degrees (3 o’clock) and the SweepGradient constructor does not have any parameters to change this value. Fortunately there is a workaround.

### 7.3.2 Custom Sweep Angle

You can change the sweep angle of a SweepGradient by modifying its local transformation matrix. We will discuss coordinate transformations and transformation matrices in Chapter 8 and 9. If you want, you can read those chapters first to better understand this section.

In this example we will use a sweep angle of -45 degrees to draw a gradient shown in Figure 7-11.

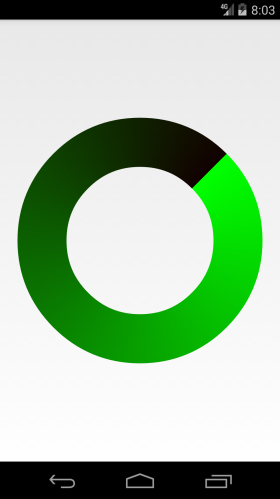


Figure 7-11.

We will reuse the code from the previous example and modify its onSizeChanged() method as shown below.

|  |
| --- |
| @Override  **protected** **void** onSizeChanged(**int** width, **int** height, **int** oldw, **int** oldh){  centerX = width / 2.0f;  centerY = height / 2.0f;  radius = Math.*min*(width, height) \* 0.7f/2.0f;  gradient = **new** SweepGradient(centerX, centerY, color1, color2);    Matrix matrix = **new** Matrix();  matrix.postTranslate(-centerX, -centerY);  matrix.postRotate(-45);  matrix.postTranslate(centerX, centerY);  gradient.setLocalMatrix(matrix);    paint.setShader(gradient);  paint.setStrokeWidth(radius/2);  } |

Let’s review the changes. After creating the gradient we changed its transformation matrix to rotate the gradient 45 degrees counterclockwise.

|  |
| --- |
| Matrix matrix = **new** Matrix();  matrix.postTranslate(-centerX, -centerY);  matrix.postRotate(-45);  matrix.postTranslate(centerX, centerY);  gradient.setLocalMatrix(matrix); |

We used 3 transformation operations:

1. Move (translate) the origin of the local coordinate system to (centerX, centerY).

|  |
| --- |
| matrix.postTranslate(-centerX, -centerY); |

2. Rotate the coordinate system 45 degrees counterclockwise(negative value) around the new orgin (centerX, centerY).

|  |
| --- |
| matrix.postRotate(-45); |

3. Move the orgin back.

|  |
| --- |
| matrix.postTranslate(centerX, centerY); |

# Chapter 08 Introduction to Coordinate Transformations

In Chapter 2 you learned that there are two types of coordinate systems used in computer graphics: device or screen coordinate systems and logical coordinate systems. Many computer graphics libraries, including Android Canvas API, use device coordinate system by default. Usually it is very difficult to work with screen coordinates directly. That is why many programmers use different logical coordinate systems for different tasks. For example, to draw a stock chart, it is more convenient to use a logical coordinate system with days on horizontal axis and stock prices on vertical axis.

In this chapter we will discuss the math behind coordinate transformations and learn how to transform coordinates without matrices. I hope that information from this chapter will help you better understand transformation matrices discussed in Chapter 9.

## 8.1 Transformation Operations

There are several different ways to change coordinates, such as translation or move, scaling, and rotation. In this chapter we will talk about translation and scaling only. Translation, scaling, and rotation are called simple transformation operations. Several simple transformations can be applied sequentially to achieve more complex transformations.

### 8.1.1 Translation

Translation also known as move, involves shifting the origin of the coordinate system horizontally and vertically by a specific amount. Transformation formulas for x and y coordinates are shown below.

|  |
| --- |
| newX = moveX + x;  newY = moveY + y; |

### 8.1.2 Scaling

Scaling lets you stretch or shrink coordinates along the x and y axes independly. Transformation formula for x and y coordinates are shown below.

|  |
| --- |
| newX = scaleX \* x;  newY = scaleY \* y; |

Now we have enough information to try several examples.

## 8.2 Example : Triangle

In this example we will create a simple Android application to draw a triangle.

First, let’s define a logical coordinate system for the triangle. We will use ratios for percentages for the width and height of a view. This type of coordinate system is called normalized coordinate system. Also, our logical coordinate system has the origin in the lower-left corner. Figure 8-1 shows an example of a device coordinate system with dimentions of 1280x800 pixels and the corresponding logical coordinate system.

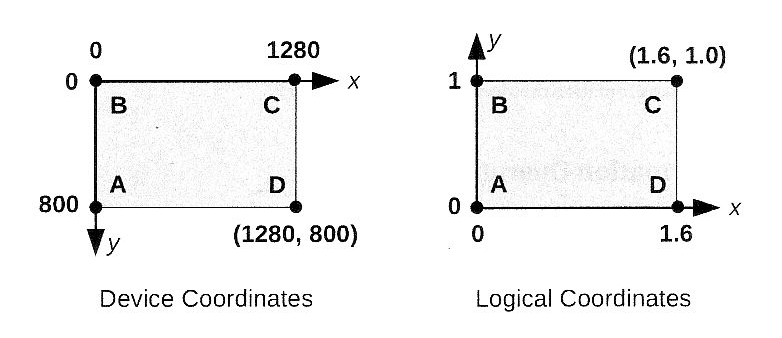


Figure 8-1.

As you can see, the normalized height is 1 or 100% and the normalized width is 1.6 (1280/800) or 160%. Figure 8-1 also shows four points A, B, C, D and its coordinates in both the device and logical coordinate systems.

By default, all drawing functions of Canvas API use device coordinates, so we have to convert all logical coordinates to device coordinates. Figure 8-2 shows how our logical coordinate system can be transformed into the device coordinate system.

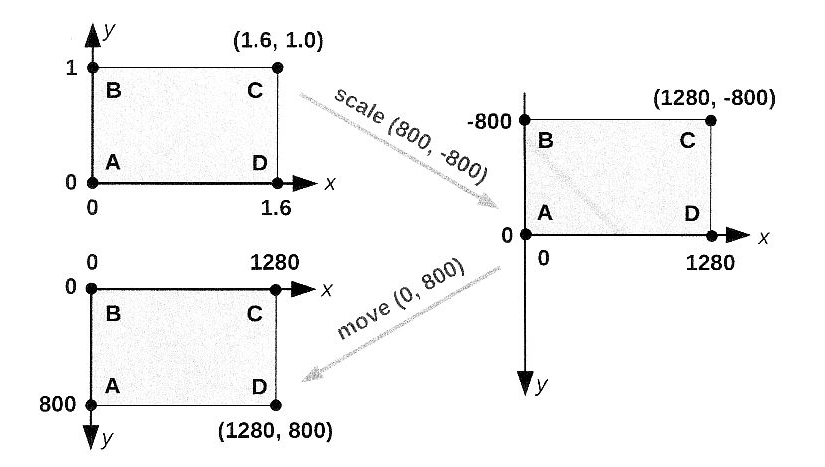


Figure 8-2.

We perform the scale first, followed by the move. Note that in order to flip our coordinate system vertically, we use a negative value (-800) for y-axis scale.

Now, let’s define a triangle in the logical coordinate system we just discussed, as shown in Figure 8-3. This is the triangle we want to draw in this example.

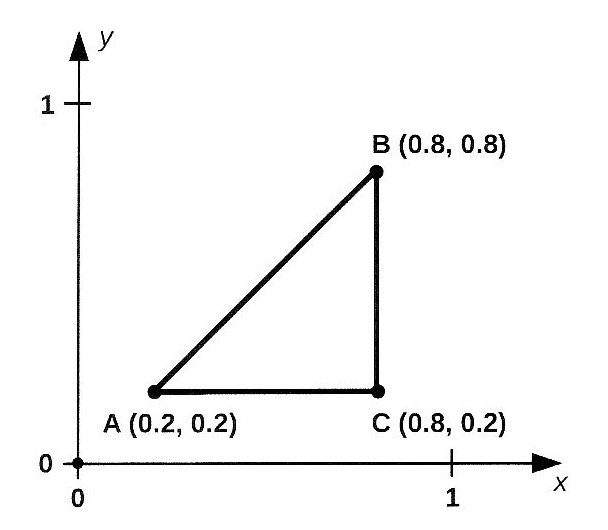


Figure 8-3.

### 8.2.1 Transform

We will use the following java class to do transformations described in Figure 8-2.

|  |
| --- |
| **package** com.example.graphics2d;  **public** **class** Transform {  **private** **float** moveX;  **private** **float** moveY;  **private** **float** scaleX;  **private** **float** scaleY;  **public** Transform(){  setMove(0,0);  setScale(1,1);  }  **public** **void** setScale(**int** x, **int** y) {  scaleX = x;  scaleY = y;  }  **public** **void** setMove(**int** x, **int** y) {  moveX = x;  moveY = y;  }  **public** **float** transformX(**float** x){  **return** moveX + scaleX \* x;  }  **public** **float** transformY(**float** y){  **return** moveY + scaleY \* y;  }  } |

The Transform class has two groups of methods.

* The setMove() and setScale() methods setup transformation operations.
* The transformX() and transformY() methods transform x and y coordinates from the logical coordinate system to the device coordinate system.

To transform coordinates you have to create an instance of the Transform class and setup transformation operations you want to do.

|  |
| --- |
| Transform transform = new Transform();  Transform.setScale(800,-800);  Transform.setMove(0,800); |

Then you can convert one or more coordinates.

|  |
| --- |
| float newX = transform.transformX(0.8f);  float newY = transform.transformY(0.2f); |

### 8.2.2 Activity

We will use the following activity with this example.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.app.Activity;  **import** android.os.Bundle;  **import** android.view.Window;  **public** **class** GraphicsActivity **extends** Activity {  @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  requestWindowFeature(Window.***FEATURE\_NO\_TITLE***);  TriangleView view = **new** TriangleView(**this**);  setContentView(view);  }  } |

You should be familiar with this code already. To save some space we will not show the title or the action bar which takes a lot of space in Android 4.x.

|  |
| --- |
| requestWindowFeature(Window.***FEATURE\_NO\_TITLE***); |

### 8.2.3 View

The TriangleView is the main view of our application. It does the drawing and coordinates transformations.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.graphics.Path;  **import** android.graphics.Paint.Style;  **import** android.view.View;  **public** **class** TriangleView1 **extends** View {  **private** Transform transform;  **private** Path path;  **private** Paint paint;  **public** TriangleView1(Context context) {  **super**(context);  setBackgroundColor(Color.***WHITE***);  transform = **new** Transform();  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setStyle(Style.***STROKE***);  paint.setColor(Color.***BLUE***);  paint.setStrokeWidth(25);  }  @Override  **public** **void** onDraw(Canvas canvas){  canvas.drawPath(path, paint);  }  @Override  **protected** **void** onSizeChanged(**int** w, **int** h, **int** oldw, **int** oldh){  //Init transform  //Landscape mode  **if**(w > h) {  transform.setScale(h, -h);  transform.setMove(0, h);  }**else**{  transform.setScale(w, -w);  transform.setMove(0, h);  }  //Convert coordinates  **float** Ax = transform.transformX(0.2f);  **float** Ay = transform.transformY(0.2f);  **float** Bx = transform.transformX(0.8f);  **float** By = transform.transformY(0.8f);  **float** Cx = transform.transformX(0.8f);  **float** Cy = transform.transformY(0.2f);  //create path  path = **new** Path();  path.moveTo(Ax, Ay);  path.lineTo(Bx, By);  path.lineTo(Cx, Cy);  path.close();  }  } |

### 8.2.4 Code Review : Constructor

In the constructor we set the view background color

|  |
| --- |
| setBackgroundColor(Color.***WHITE***); |

and created a Paint object.

|  |
| --- |
| transform = **new** Transform();  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setStyle(Style.***STROKE***);  paint.setColor(Color.***BLUE***);  paint.setStrokeWidth(25); |

Also we created an instance of the Transform class. It will be initialized later in onSizeChanged() method.

|  |
| --- |
| transform = **new** Transform(); |

### 8.2.5 Code Review : onSizeChanged()

The onSizeChanged() method is called by Android framework when TriangleView size changes, either after the view was created or when Android device orientation changes. This is the best place to initialize the transform object.

The scale transformation is a little defferent for the landscape and portrait orientation. The short side of a view (the height in landscape mode and the width in portrait mode) is used for this operation.

|  |
| --- |
| //Landscape mode  if(w > h) {  transform.setScale(h, -h);  transform.setMove(0, h);  }else{  transform.setScale(w, -w);  transform.setMove(0, h);  } |

The move transfornmation is the same for both the landscape and portrait orientation.

|  |
| --- |
| transform.setMove(0, h); |

Once the transform object is initialized we can convert logical coordinates of three points A, B, and C (see Figure 8-3) to device coordinates.

|  |
| --- |
| **float** Ax = transform.transformX(0.2f);  **float** Ay = transform.transformY(0.2f);  **float** Bx = transform.transformX(0.8f);  **float** By = transform.transformY(0.8f);  **float** Cx = transform.transformX(0.8f);  **float** Cy = transform.transformY(0.2f); |

Next, we created a path to draw three lines AB, BC, and CA.

|  |
| --- |
| path = **new** Path();  path.moveTo(Ax, Ay);  path.lineTo(Bx, By);  path.lineTo(Cx, Cy);  path.close(); |

Note that close() method adds CA line segment to the path. You don’t have to call

|  |
| --- |
| path.lineTo(Ax, Ay); |

### 8.2.6 Code Review : onDraw()

The onDraw() method is very simple. It draws the path created in onSizeChanged() method.

|  |
| --- |
| canvas.drawPath(path, paint); |

### 8.2.7 Screenshots

Figure 8-4 and 8-5 shows screenshots of the application in the portrait and landscape modes.

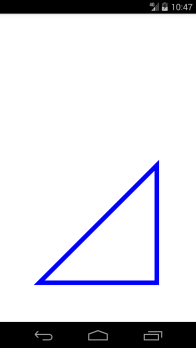


Figure 8-4.

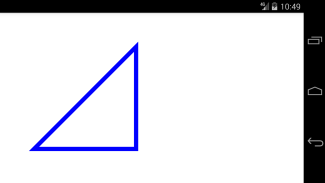


Figure 8-5.

Remember that the logical coordinate system shown in Figure 8-1. Has the origin in the lower-left corner of the view and the view height and width are different. That’s why the triangle in Figure 8-4 is bottom aligned and in the Figure 8-5 is left-aligned. If you want to center the triangle, you have to adjust the move transformation by (w – h)/2 as shown below.

|  |
| --- |
| **if**(w > h) { //Landscape mode  transform.setScale(h, -h);  transform.setMove((w-h)/2, h);  //transform.setMove(0, h);  }**else**{ //Portrait mode  transform.setScale(w, -w);  transform.setMove(0, h-(h-w)/2);  //transform.setMove(0, h);  } |

In landscape mode, the x coordinate is adjusted, because the width is greater than the height. In portrait mode, the height is greater than the width, therefore the y coordinate is adjusted.

Figure 8-6 and 8-7 show the center aligned triangle in the portrait and landscape modes.

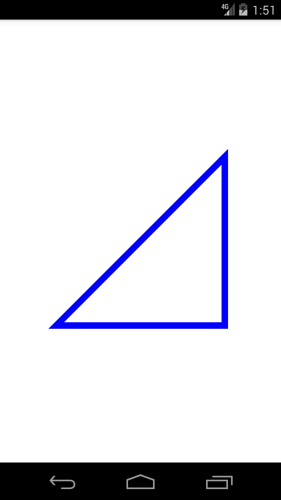


Figure 8-6.

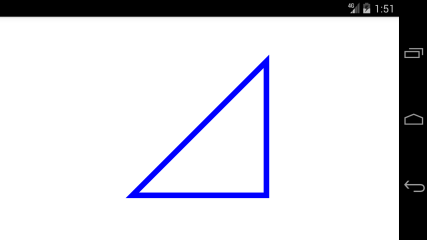


Figure 8-7.

## 8.3 Example : Stock Chart

In this example we will draw a chart of daily closing prices of S&P 500 index. The S&P 500 is one of the most commonly used analysts and economists at Standard & Poor’s. Historical stock and index prices are available for free from many web sites, such as Yahoo Finance or Google Finance. In this example we will use March 2013 daily closing prices shown below.

|  |
| --- |
| **private** **float** prices[]=  {1518.20f, 1525.20f, 1539.79f, 1541.46f, 1544.26f,  1551.18f, 1556.22f, 1542.48f, 1554.52f, 1563.23f,  1560.70f, 1552.10f, 1548.34f, 1558.71f, 1545.80f,  1556.89f, 1551.69f, 1563.77f, 1562.85f, 1569.19f  };    **private** String dates[] =  {"2005-04-01","2005-04-02","2005-04-03","2005-04-04","2005-04-05",  "2005-04-06","2005-04-07","2005-04-08","2005-04-09","2005-04-10",  "2005-04-11","2005-04-12","2005-04-13","2005-04-14","2005-04-15",  "2005-04-16","2005-04-17","2005-04-18","2005-04-19","2005-04-20"  }; |

We already saw an example of a stock chart in Figure 2-4 in Chapter 2 when we discussed logical coordinate systems. In this example we will use similar logical coordinates shown in Figure 8-8.

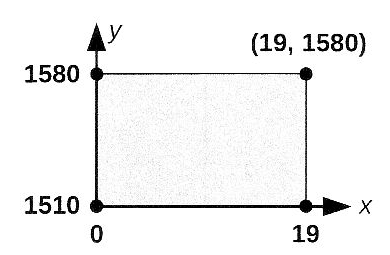


Figure 8-8.

The horizontal axis represents days, or more precisely, an index in prices[] or days[] array. The index can have values from 0 to 19. The vertical axis represents prices which range from 1510 to 1580.

By default, all drawing functions of Canvas API use device coordinates, so we have to convert all logical coordinates to device coordinates. To draw a chart we will use all the space available on the screen (view). That means that the height-to-width ratio of a chart will depend on a screen resolution of a device. Figure 8-9 shows three transformations required to convert logical coordinate system to device coordinates. Note that all numbers in Figure 8-9 are based on the device coordinate system with dimensions of 1280\*800 pixels.

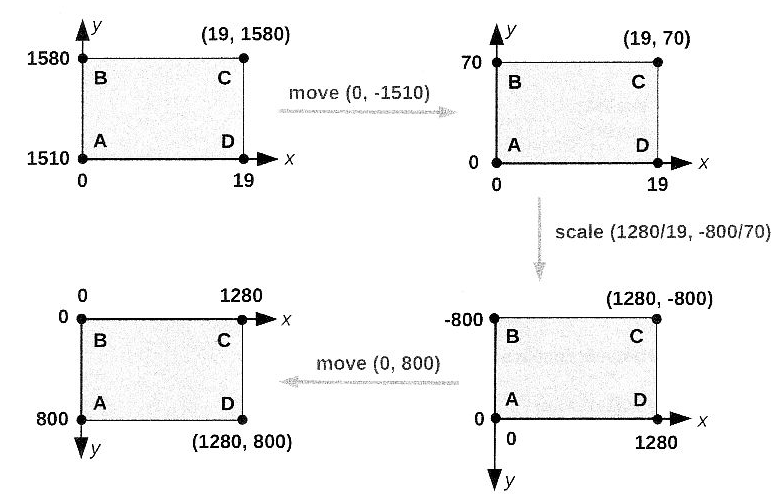


Figure 8-9.

Let’s quickly review each transformation.

* move(0, -1510) This transformation moves the origin from (0,0) to (0, 1510).
* scale(1280/19,-800/70) As in the previous example, this transformation flips coordinates vertically and changes the width of the view from 19 to 1280 and the deight from 70 to 800.
* move(0,800) This transformation moves the origin from (0,-800) to (0,0). This is the same transformation we did in the previous example

### 8.3.1 Transform

In the previous example we created the Transform class to perform a scale and move operations. We have to modify this class to support an additionalmove transformation as shown below.

|  |
| --- |
| **package** com.example.graphics2d;  **public** **class** Transform {  **private** **float** moveBeforeScaleX;  **private** **float** moveBeforeScaleY;  **private** **float** moveAfterScaleX;  **private** **float** moveAfterScaleY;  **private** **float** scaleX;  **private** **float** scaleY;  **public** Transform(){  setMoveBeforeScale(0,0);  setMoveAfterScale(0,0);  setScale(1,1);  }  **public** **void** setMoveBeforeScale(**float** x, **float** y){  moveBeforeScaleX=x;  moveBeforeScaleY=y;  }  **public** **void** setMoveAfterScale(**float** x, **float** y){  moveAfterScaleX=x;  moveAfterScaleY=y;  }  **public** **void** setScale(**float** x, **float** y){  scaleX=x;  scaleY=y;  }  **public** **float** transformX(**float** x){  **return** moveAfterScaleX + scaleX \* (moveBeforeScaleX+x);  }  **public** **float** transformY(**float** y){  **return** moveAfterScaleY + scaleY \* (moveBeforeScaleY+y);  }  } |

In addition to the scale transformation, the new version of Transform class supports two move operations: move beforescale and move after scale. New transform() and transformY() methods are shown below.

|  |
| --- |
| **public** **float** transformX(**float** x){  **return** moveAfterScaleX + scaleX \* (moveBeforeScaleX+x);  }  **public** **float** transformY(**float** y){  **return** moveAfterScaleY + scaleY \* (moveBeforeScaleY+y);  } |

### 8.3.2 Chart Model

It is common to separate data from its visual representation. In our example, the ChartView class is responsible for visual representation of the data and the ChartModel class provides the API to access the data.

|  |
| --- |
| **package** com.example.graphics2d;  **public** **class** ChartModel {  **private** **float** prices[]={  1518.20f, 1525.20f, 1539.79f, 1541.46f, 1544.26f,  1551.18f, 1556.22f, 1542.48f, 1554.52f, 1563.23f,  1560.70f, 1552.10f, 1548.34f, 1558.71f, 1545.80f,  1556.89f, 1551.69f, 1563.77f, 1562.85f, 1569.19f  };  **private** **float** minPrice = 1510f;  **private** **float** maxPrice = 1580f;  **public** ChartModel(){}  **public** **float** getMinPrice(){  **return** minPrice;  }  **public** **float** getMaxPrice(){  **return** maxPrice;  }  **public** **int** getNumberOfDays(){  **return** prices.length;  }  **public** **int** getMinDay(){  **return** 0;  }  **public** **float** getPrice(**int** index){  **return** prices[index];  }  } |

To make this example simpler, we hardcoded min and max prices instead of calculating them.

### 8.3.3 Activity

The ChartActivity is similar to the activity from the previous example.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.app.Activity;  **import** android.content.pm.ActivityInfo;  **import** android.os.Bundle;  **import** android.view.Window;  **public** **class** ChartActivity **extends** Activity {  @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  requestWindowFeature(Window.***FEATURE\_NO\_TITLE***);  setRequestedOrientation(ActivityInfo.***SCREEN\_ORIENTATION\_LANDSCAPE***);  ChartView view = **new** ChartView(**this**);  **this**.setContentView(view);  }  } |

A stock chart looks much better in the landscape mode, so we will enforce this screen orientation by calling the following method.

|  |
| --- |
| setRequestedOrientation(ActivityInfo.***SCREEN\_ORIENTATION\_LANDSCAPE***); |

Also, to have more space on the screen we will hide the action bar, which is pretty big in Android 4.x.

|  |
| --- |
| requestWindowFeature(Window.***FEATURE\_NO\_TITLE***); |

### 8.3.4 ChartView

The ChartView class is responsible for drawing the chart.

|  |
| --- |
| package com.example.graphics2d;  import android.content.Context;  import android.graphics.Canvas;  import android.graphics.Color;  import android.graphics.Paint;  import android.graphics.Paint.Cap;  import android.graphics.Paint.Join;  import android.graphics.Paint.Style;  import android.graphics.Path;  import android.view.View;  public class ChartView extends View{  private Transform transform;  private ChartModel model;  private Paint paint;  private Path path;  public ChartView(Context context){  super(context);  setBackgroundColor(Color.WHITE);  paint = new Paint(Paint.ANTI\_ALIAS\_FLAG);  paint.setStyle(Style.STROKE);  paint.setColor(Color.BLUE);  paint.setStrokeWidth(5);  paint.setStrokeCap(Cap.ROUND);  paint.setStrokeJoin(Join.ROUND);  transform = new Transform();  model = new ChartModel();  }  @Override  public void onDraw(Canvas canvas){  canvas.drawPath(path, paint);  }  @Override  protected void onSizeChanged(int w, int h, int oldw, int oldh){  //Init transform  transform.setMoveBeforeScale(0, -model.getMinPrice());  float scaleX = (float) w/(model.getMaxDay()-model.getMinDay());  float scaleY = (float) h/(model.getMaxPrice()-model.getMinPrice());  transform.setScale(scaleX, -scaleY);  transform.setMoveAfterScale(0, h);  //Create path  path = new Path();  float x = transform.transformX(0);  float y = transform.transformY(model.getPrice(0));  path.moveTo(x, y);  for(int i = 1; i < model.getNumbersOfDays();i++){  x=transform.transformX(i);  y=transform.transformY(model.getPrice(i));  path.lineTo(x, y);  }  }  } |

### 8.3.5 Code Review: Constructor

In the constractor we set the view background color and created a Paint object. We used round stroke cap and join styles and anti-alias flag to make the line look smoother.

|  |
| --- |
| setBackgroundColor(Color.WHITE);  ……  paint.setStrokeCap(Cap.ROUND);  paint.setStrokeJoin(Join.ROUND); |

Also we created the Transform and ChartModel objects.

|  |
| --- |
| transform = new Transform();  model = new ChartModel(); |

### 8.3.6 Code Review: onSizeChanged()

Most of the work is done in this method.

Before we can do any coordinate transformations, we have to initialize the transform object. There are three transformation operations we have to do(see Figure 8-9).

1. Move the origin from (0,0) to (0,minPrice).

|  |
| --- |
| transform.setMoveBeforeScale(0, -model.getMinPrice()); |

2. Scale. Negative scaleY value is used to flip coordinates vertically.

|  |
| --- |
| float scaleX = (float) w/(model.getMaxDay()-model.getMinDay());  float scaleY = (float) h/(model.getMaxPrice()-model.getMinPrice());  transform.setScale(scaleX, -scaleY); |

Move the origin back to (0,0).

|  |
| --- |
| transform.setMoveAfterScale(0, h); |

Once the transform object is initialized we can convert logical coordinates todevice coordinates. Device coordinates are used to create a Path.

|  |
| --- |
| path = new Path();  float x = transform.transformX(0);  float y = transform.transformY(model.getPrice(0));  path.moveTo(x, y);  for(int i = 1; i < model.getNumbersOfDays();i++){  x=transform.transformX(i);  y=transform.transformY(model.getPrice(i));  path.lineTo(x, y);  } |

Note that in this example we did not close the path.

### 8.3.7 Code Review: onDraw()

In onDraw() method we drew the path created in onSizeChanged().

|  |
| --- |
| canvas.drawPath(path, paint); |

### 8.3.8 Screenshot

A screenshot of the application is shown in Figure 8-10.

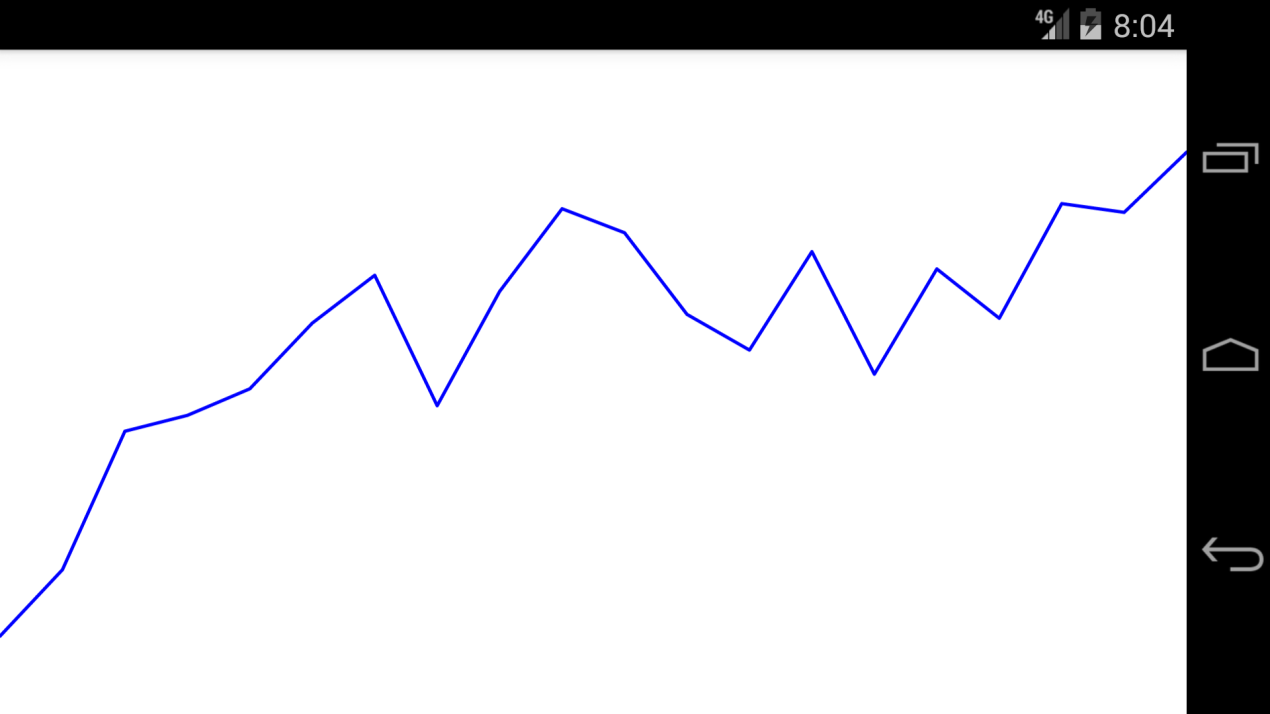


Figure 8-10.

# Chapter 9 Transformation Matrices

In the previous chapter you learned that there are several simple transformation operations, such as translation, scaling and rotation. Each transformation operation can be represented by a 3x3 matrix shown below.

The last row of a matrix always contains the constant values [0 0 1]. These values are required to perform matrix concatenation which is explained later in this chapter.

A coordinate can be represented by a column vector.

The last element of a vector always contains the constant value of 1. This extra value is required for matrix multiplication. Two matrices can be multiplied only if the number of columns in the first matrix equals the number of rows in the second matrix. Therefore 3x3 matrix can be only multiplied by 3x1 vector.

To transform source coordinates (x,y) to destination coordinates (x’,y’) we have to multiply a transformation matrix by a source coordinate vector.

The result of the multiplication is shown below.

## 9.1 Transformation Operations

Let’s review matrices for several simple transformation operations.

### 9.1.1 Identity Matrix

The identity matrix is the matrix in which all the elements in the main diagonal are equal to 1 and all other elements are equal to 0.

It does not perform any transformations.

### 9.1.2 Translation

Translation, also known as move, involves shifting the origin of the current coordinate system horizontally and vertically by a specific amount. Transformation matrix for a translation operation is shown below.

These are the transformation equations:

### 9.1.3 Scaling

Scaling lets you stretch or shrink coordinates along the x and y axes independently. Transformation matrix for a scaling operation is shown below.

These are the transformation equations:

### 9.1.4 Rotation

Rotation operation moves coordinates by the specified angle. Transformation matrix for a rotation operation is shown below.

These are the transformation equations:

## 9.2 Matrix Concatenation

One of the advantages of using matrices is that you can combine several transformations by multiplying corresponding matrices.

For example, if you want to combine a scale and translate operations, you multiply the scale and translate matrices to produce a composite matrix.

The process of combining several transformation matrices is called concatenation.

The order in which matrices are concatenated is important. Matrix multiplication is not commutative. The result of multiplying matrix A by matrix B does not necessarily equal the result of multiplying matrix B by matrix A.

## 9.3 Matrix Class

The Matrix class holds a 3x3 matrix for transforming coordinates. This class has many methods, but usually you would use just a few of them in most programs. Let’s review some common methods.

### 9.3.1 Constructor

To create a Matrix object you call its constructor which creates an identity matrix.

|  |
| --- |
| Matrix matrix = new Matrix( ); |

### 9.3.2 Concatenation Methods

One of the advantages of using matrices is that you can combine several transformations and store them in a single matrix. Usually you create a Matrix object and then concatenate it with one or more transformation operations.

There are two groups of concatenation methods: pre-and post-concatenation methods.

* Pre-concatenation methods such as preConcate(), preTranslate(), preScale(), preRotate() apply new operation before the preceding operation.
* Post-concatenation methods such as postConcat(), postTranslate(), postScale(), postRotate() apply new operation after the preceding operation.

A code snippet below shows scale and translate operations discussed in section 8.2 of the previous chapter.

|  |
| --- |
| matrix.postScale(800, -800);  matrix.postTranslate(0, 800); |

### 9.3.3 Coordinate Transformation

The Matrix class provides several methods to transform coordinates. All these methods start with “map” prefix, such as mapPoints(), mapVectors(), mapRadius, mapRec().

The following code snippet shows how to apply the matrix to the array of points specified by src, and write the transformed points into the array of points specified by dst.

|  |
| --- |
| float[] src;  float[] dst;  …  Matrix.mapPoints(dst, src); |

The points in a Path object can also be transformed as shown below.

|  |
| --- |
| Path path;  …  Path.transform(matrix); |

## 9.4 Example : Triangle

In this section we will modify the example we created in section 8.2 of the previous chapter to use transformation matrices instead of the Transformation class we wrote ourselves. Let’s look at the modified version of the TriangleView class first and then discuss the changes.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Matrix;  **import** android.graphics.Paint;  **import** android.graphics.Path;  **import** android.graphics.Paint.Style;  **import** android.view.View;  **public** **class** TriangleViewMatrix **extends** View{  **private** Matrix matrix;  **private** Path path;  **private** Paint paint;  **public** TriangleViewMatrix(Context context) {  **super**(context);  setBackgroundColor(Color.***WHITE***);  paint = **new** Paint();  paint.setStyle(Style.***STROKE***);  paint.setColor(Color.***BLUE***);  paint.setStrokeWidth(25);  matrix = **new** Matrix();  }  @Override  **public** **void** onDraw(Canvas canvas){  canvas.drawPath(path, paint);  }  @Override  **protected** **void** onSizeChanged(**int** deviceWidth, **int** deviceHeight,  **int** oldw, **int** oldh){  //Init transformation matrix  matrix.reset();  **if**(deviceWidth > deviceHeight) {  matrix.postScale(deviceHeight, -deviceHeight);  matrix.postTranslate(0, deviceHeight);  }**else**{  matrix.postScale(deviceWidth, -deviceWidth);  matrix.postTranslate(0, deviceHeight);  }  //create path  path = **new** Path();  path.moveTo(0.2f, 0.2f);  path.lineTo(0.8f, 0.8f);  path.lineTo(0.8f, 0.2f);  path.close();  //Convert coordinates  path.transform(matrix);  }  } |

### 9.4.1 Code Review : Constructor

We replaced the Transform object.

|  |
| --- |
| transform = new Transform(); |

With the Matrix object.

|  |
| --- |
| matrix = new Matrix(); |

The rest of the code is the same.

### 9.4.2 Code Review : onSizeChange()

First we initialized the transformation matrix with the identity matrix.

|  |
| --- |
| matrix.reset(); |

Then we concatenated it with scale and translate operations.

|  |
| --- |
| **if**(deviceWidth > deviceHeight) {  matrix.postScale(deviceHeight, -deviceHeight); matrix.postTranslate(0, deviceHeight);  }**else**{  matrix.postScale(deviceWidth, -deviceWidth);  matrix.postTranslate(0, deviceHeight);  } |

So far the new code is very similar to the old code from Chapter 8.

Once the transformation matrix is initialized we can convert logical coordinates to device coordinates and create a Path object.

In the old code we converted coordinates first and then created a Path object with the device coordinates.

In the new code, we used logical coordinates to create a Path

|  |
| --- |
| path = **new** Path();  path.moveTo(0.2f, 0.2f);  path.lineTo(0.8f, 0.8f);  path.lineTo(0.8f, 0.2f);  path.close(); |

and then converted the points in the Path to device coordinates.

|  |
| --- |
| path.transform(matrix); |

### 9.4.3 Code Review : onDraw()

The onDraw() method didn’t change. It draws the path created in onSizeChanged() method.

|  |
| --- |
| canvas.drawPath(path, paint); |

### 9.4.4 Screenshots

Figure 9-1 and 9-2 shows screenshots of the application in the portrait and landscape modes. Not surprisingly, the screenshots look the same as in Chapter 8.

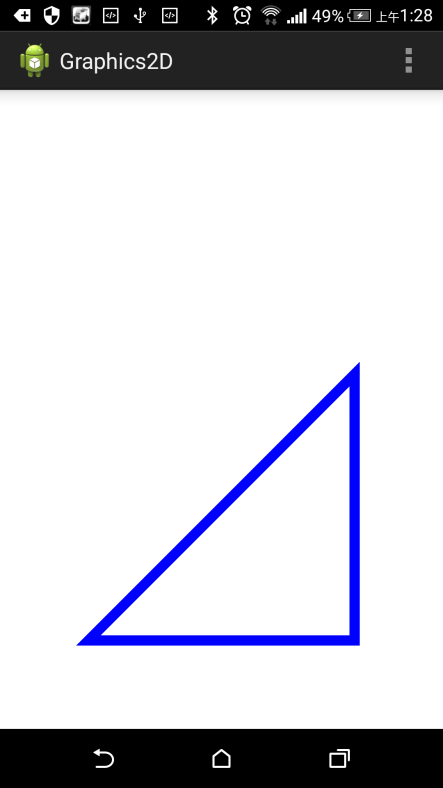


Figure 9-1.

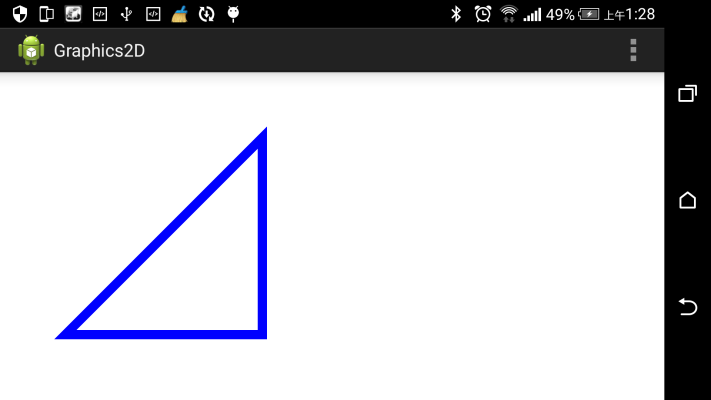


Figure 9-2.

## 9.5 Example : StockChart

In this section we will modify the example we created in section 8.3 of the previous chapter to use transformation matrices instead of the Transform class we wrote ourselves. Let’s look at the modified version of the ChartView class first and then review the changes.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Matrix;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Cap;  **import** android.graphics.Paint.Join;  **import** android.graphics.Paint.Style;  **import** android.graphics.Path;  **import** android.view.View;  **public** **class** ChartViewMatrix **extends** View{  **private** Matrix matrix;  **private** ChartModel model;  **private** Paint paint;  **private** Path path;  **public** ChartViewMatrix(Context context) {  **super**(context);  setBackgroundColor(Color.***YELLOW***);  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setStyle(Style.***STROKE***);  paint.setColor(Color.***BLUE***);  paint.setStrokeWidth(5);  paint.setStrokeCap(Cap.***ROUND***);  paint.setStrokeJoin(Join.***ROUND***);  matrix = **new** Matrix();  model = **new** ChartModel();  }  @Override  **public** **void** onDraw(Canvas canvas){  canvas.drawPath(path, paint);  }  @Override  **protected** **void** onSizeChanged(**int** w, **int** h, **int** oldw, **int** oldh){  //Init transformation matrix  matrix.reset();  matrix.postTranslate(0, -model.getMinPrice());  **float** scaleX = (**float**) w/(model.getMaxDay()-model.getMinDay());  **float** scaleY = (**float**) h/(model.getMaxPrice()-model.getMinPrice());  matrix.postScale(scaleX, -scaleY);  matrix.postTranslate(0, h);  //Create path  path = **new** Path();  path.moveTo(0, model.getPrice(0));  **for**(**int** i = 1; i < model.getNumbersOfDays();i++){  path.lineTo(i, model.getPrice(i));  }  //Convert coordinate  path.transform(matrix);  }  } |

### 9.5.1 Code Review : Constructor

We replaced the Transform object

|  |
| --- |
| transform = new Transform(); |

with the Matrix object.

|  |
| --- |
| matrix = new Matrix(); |

The rest of the code is the same.

### 9.5.2 Code Review : onSizeChanged()

Before we can do any coordinate transformations, we have to initialize the transformation matrix. First we initialized the transformation matrix with the identity matrix.

|  |
| --- |
| matrix.reset(); |

Next we concatenated the matrix with three transformation operations shown in Figure 9-3.

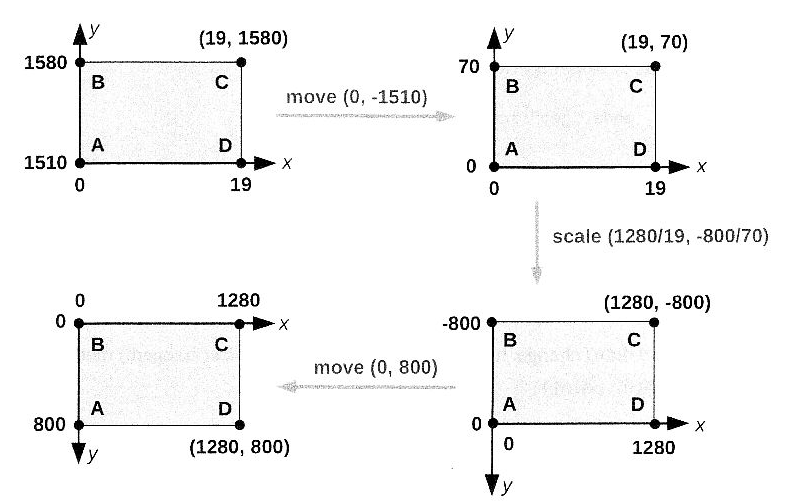


Figure 9-3.

1. Move the origin from (0,0) to (0, minPrice()).

|  |
| --- |
| matrix.postTranslate(0, -model.getMinPrice()); |

2. Scale. Negative scaleY value is used to flip coordinates vertically.

|  |
| --- |
| **float** scaleX = (**float**) w/(model.getMaxDay()-model.getMinDay());  **float** scaleY = (**float**) h/(model.getMaxPrice()-model.getMinPrice());  matrix.postScale(scaleX, -scaleY); |

3. Move the origin back to (0,0).

|  |
| --- |
| matrix.postTranslate(0, h); |

Once the transformation matrix is initialized we can convert logical coordinates first and then used device coordinates to create a Path.

In this example we used logical coordinates to create a Path

|  |
| --- |
| path = **new** Path();  path.moveTo(0, model.getPrice(0));  **for**(**int** i = 1; i < model.getNumbersOfDays();i++){  path.lineTo(i, model.getPrice(i));  } |

and then converted the points in the Path to device coordinates.

|  |
| --- |
| path.transform(matrix); |

### 9.5.3 Code Review : onDraw()

The onDraw() method didn’t change. It draws the path created in onSizeChanged() method.

|  |
| --- |
| canvas.drawPath(path, paint); |

### 9.5.4 Screenshot

A screenshot of the application is shown in Figure 9-4.

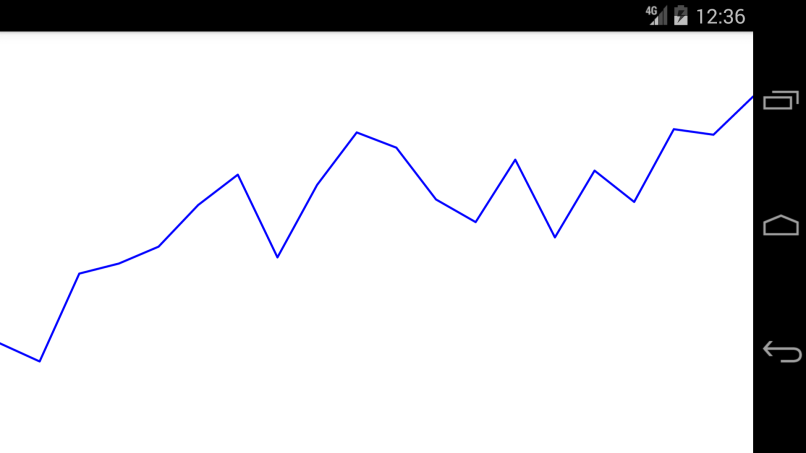


Figure 9-4.

## 9.6 Current Transformation Matrix(CTM)

Each Canvas object has a transformation matrix called the Current Transformation Matrix or CTM. The CTM is applied to any drawing performed by the Canvas object. After a Canvas Object is created, the CTM is the identity matrix and does not affect default device coordinate system.

There are several methods such as translate(), scale(), rotate() and concat() which can be used to change the Current Transformation Matrix. The following example shows how this can be done.

### 9.6.1 Example

We will use the same logical coordinate system as in stock chart example(see Figure 9-3) and try to draw a rectangle.

|  |
| --- |
| package com.example.graphics2d;  import android.content.Context;  import android.graphics.Canvas;  import android.graphics.Color;  import android.graphics.Paint;  import android.graphics.Paint.Style;  import android.graphics.RectF;  import android.view.View;  public class CTMView extends View{  private int minX = 0;  private int maxX = 19;  private int minY = 1510;  private int maxY = 1580;  private Paint paint;  private RectF rect;  private int deviceHeight;  private int deviceWidth;  public CTMView(Context context) {  super(context);  setBackgroundColor(Color.WHITE);  paint = new Paint(Paint.ANTI\_ALIAS\_FLAG);  paint.setStyle(Style.STROKE);  paint.setColor(Color.BLUE);  paint.setStrokeWidth(5);  rect = new RectF(5,1520,14,1570);  }  @Override  protected void onSizeChanged(int W, int H, int olcw, int oldh){  this.deviceWidth = W;  this.deviceHeight = H;  }  @Override  public void onDraw(Canvas canvas){  //Transform 3  canvas.translate(0, deviceHeight);  //Transform 2  float scaleX = (float) deviceWidth / (maxX-minX);  float scaleY = (float) deviceHeight / (maxY-minY);  canvas.scale(scaleX, -scaleY);  //Transform 1  canvas.translate(0, -minY);  //Draw  canvas.drawRect(rect, paint);  }  } |

### 9.6.2 Code Review : Class Variables

First we defined min and max values of x and y coordinates in the logical coordinate system. In stock chart example we got those values from the chart model.

|  |
| --- |
| private int minX = 0;  private int maxX = 19;  private int minY = 1510;  private int maxY = 1580; |

### 9.6.3 Code Review : Constructor

In the constructor we set the background color and created a Paint object. Also we created a rectangle we want to draw. Note that we used logical coordinates.

|  |
| --- |
| rect = new RectF(5,1520,14,1570); |

### 9.6.4 Code Review : onSizeChanged()

In the onSizeChanged() ethod we just stored the view width and height in class variables.

|  |
| --- |
| this.deviceWidth = W;  this.deviceHeight = H; |

### 9.6.5 Code Review : onDraw()

In section 9.3 you learned that the Matrix class has pre- and postconcatenation methods. In all our examples so far we used post-concatenation methods which we called in the same order we wanted to apply transformation operations.

The translate(), scale() and rotate() methods which can be used to change the Current Transformation Matrix are pre-concatenation methods. Therefore we have to apply transformation operations in the reverse order.

|  |
| --- |
| //Transform 3  canvas.translate(0, deviceHeight);  //Transform 2  float scaleX = (float) deviceWidth / (maxX-minX);  float scaleY = (float) deviceHeight / (maxY-minY);  canvas.scale(scaleX, -scaleY);  //Transform 1  canvas.translate(0, -minY); |

After changing the CTM, all subsequent drawing methods such as drawRect(), drawLine(), drawPath() will use our local coordinate system. So, now we can draw the rectangle.

|  |
| --- |
| canvas.drawRect(rect, paint); |

### 9.6.6 Screenshot

If you run this example application you should see an unusual rectangular shape as shown in Figure 9-5.



Figure 9-5.

So, what just happened?

One of the main reasons, you should never change the Current Transformation Matrix is that it not only transforms the coordinates of vertices or points of drawing primitives, but also the line width.

In our example, the horizontal and vertical scale factors are different, therefore horizontal and vertical lines have different width. When we called

|  |
| --- |
| paint.setStrokeWidth(5); |

we expected the line to be 5 pixels wide, but instead, this value was scaled. If we assume that the view width is 1280 pixels and the height is 800 pixels, we will get the following values of the vertical and horizontal widths:

|  |
| --- |
| vertical = 1280/19\*5=337  horizontal = 800/70\*5=57 |

This explains why vertical lines are much wider than horizontal.

So, how can we fix it? The only way to fix the line width is to convert logical coordinates to device coordinates, like we did in previous examples, without changing the CTM. The following example showns how to do it.

### 9.6.7 Fixed Example

Below is the modified version of the previous example which does not use the CTM.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Matrix;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Style;  **import** android.graphics.RectF;  **import** android.view.View;  **public** **class** CTMViewFixed **extends** View{  **private** **int** minX = 0;  **private** **int** maxX = 19;  **private** **int** minY = 1510;  **private** **int** maxY = 1580;  **private** Matrix matrix;  **private** Paint paint;  **private** RectF rect;    **public** CTMViewFixed(Context context) {  **super**(context);  setBackgroundColor(Color.***WHITE***);  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setStyle(Style.***STROKE***);  paint.setColor(Color.***BLUE***);  matrix = **new** Matrix();  }    @Override  **protected** **void** onSizeChanged(**int** deviceWidth, **int** deviceHeight, **int** olcw, **int** oldh){  matrix.reset();  //Transform 1  matrix.postTranslate(0, -minY);  //Transform 2  **float** scaleX = (**float**) deviceWidth / (maxX-minX);  **float** scaleY = (**float**) deviceHeight / (maxY-minY);  matrix.postScale(scaleX, -scaleY);  //Transform 3  matrix.postTranslate(0, deviceHeight);  //Convert coordinates  rect = **new** RectF(5,1520,14,1570);  matrix.mapRect(rect);  //Set Line width  paint.setStrokeWidth(deviceHeight\*0.05f);  }  @Override  **public** **void** onDraw(Canvas canvas){  canvas.drawRect(rect, paint);  }  } |

### 9.6.8 Code Review : Constructor

Instead of using CTM we created a Matrix object in the constructor.

|  |
| --- |
| matrix = **new** Matrix(); |

### 9.6.9 Code Review : onSizeChanged()

We initialized the transformation matrix the way we did it in the stock chart example.

|  |
| --- |
| matrix.reset();  //Transform 1  matrix.postTranslate(0, -minY);  //Transform 2  **float** scaleX = (**float**) deviceWidth / (maxX-minX);  **float** scaleY = (**float**) deviceHeight / (maxY-minY);  matrix.postScale(scaleX, -scaleY);  //Transform 3  matrix.postTranslate(0, deviceHeight);  //Convert coordinates  rect = **new** RectF(5,1520,14,1570);  matrix.mapRect(rect);  //Set Line width  paint.setStrokeWidth(deviceHeight\*0.05f); |

### 9.6.10 onDraw()

In onDraw() method we drew the rectangle created in onSizeChanged() method.

|  |
| --- |
| canvas.drawRect(rect, paint); |

### 9.6.11 Screenshot

A screenshot of the application is shown in Figure 9-6.

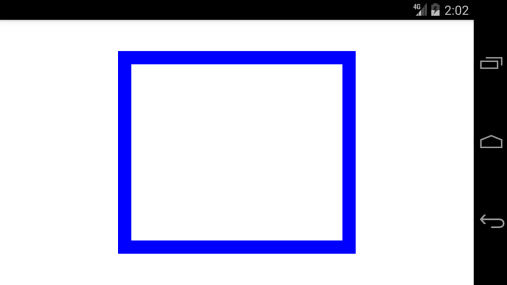


Figure 9-6.

# Chapter 10 Multithreading

In this chapter we will discuss two common approaches to writing multithreaded graphics applications in Android. The first approach uses the Handler and Message classes to communicate between threads. The second approach utilizes the SurfaceView class.

## 10.1 Processes and Threads

Every Android application runs in its own process. A process consists of the executing program code, a set of resources such as open files, memory, and one or more threads of execution.

A thread of execution is the smallest sequence of instructions to which an operating system allocates processor time. Threads exist within a processor and share its resources, imcluding memory and open files.

An application starts with just one thread, called the main thread or UI thread. This thread is responsible for dispatching events, such as drawing and touch events to user interface widgets. This is where all of the updates to the UI are made.

If your application performs long operations such as network access or database queries in the main thread, the whole UI will wait until those operations are completed, because the UI thread is blocked. Android operating system can display the “application not responding” dialog shown in Figure 10-1.

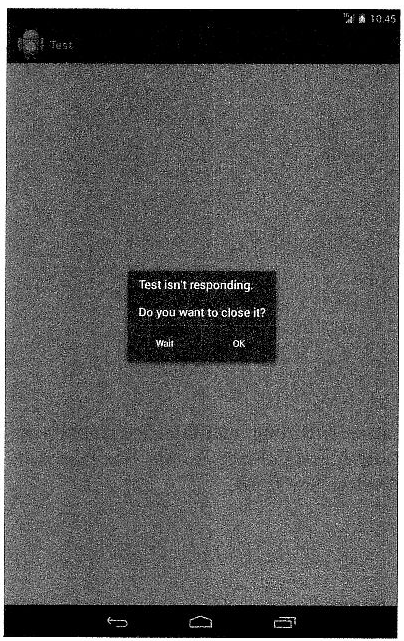


Figure 10-1.

You should execute long running tasks in separate threads also called worker or background threads. Unfortunately the Android UI toolkit is not thread-safe. Therefore, you must update user interface only from the UI thread. Worker threads can communicate with the main UI thread through a Handler class.

## 10.2 Handler

The Handler class in android.os package can be used to send messages from worker threads to the main UI thread. When new Handler object is created it is associated with the message queue of the current thread. All messages sent to this Handler from different threads are delivered to the thread in which the Handler was created.

The Handler class takes care of thread synchronization.

### 10.2.1 Example: Multithreaded Stock Chart

In this example we will add a worker thread to the stock chart application from the previous chapter. The worker thread is responsible for fetching market data and creating a stock chart model. Once a model object is created it is sent to the main UI thread to be displayed.

We will also change the UI. We will add a text field and Go button at the top of the screen as shown in Figure 10-2.

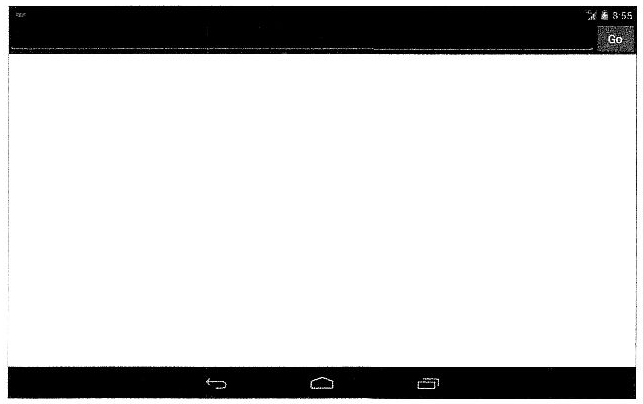


Figure 10-2.

After you enter a stock ticker, such as IBM or AAPL and it Go button, the text field and Go button are disabled and “Loading…” message is displayed as shown in Figure 10-3.

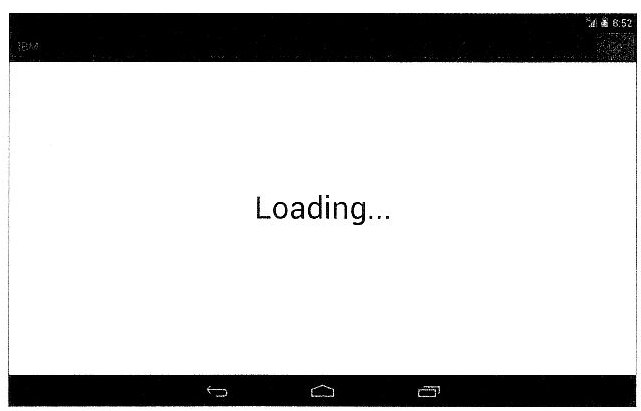


Figure 10-3.

After several seconds the stock chart is displayed and the text field and Go button are enabled again as shown in Figure 10-4.

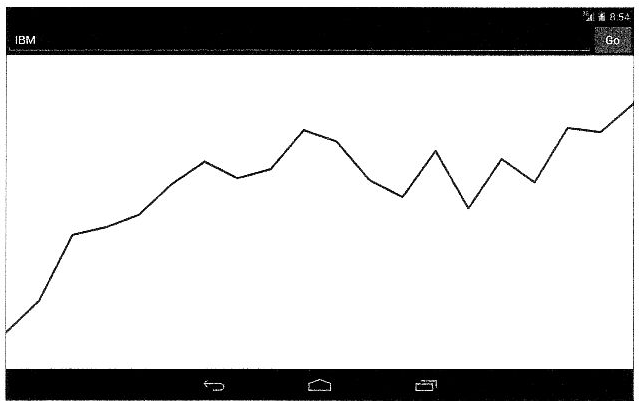


Figure 10-4.

Let’s review classes and interfaces we used in this example.

#### MessageListener

This interface defines a method which will be called by a Handler when a ChartModel object is delivered from the worker thread to the main UI thread.

|  |
| --- |
| **package** com.example.graphics2d;  **public** **interface** MessageListener {  **public** **void** onChartData(ChartModel model);  } |

#### MessageHandler

The MessageHandler class extends the Handler class and overrides its handleMessage() method.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.os.Handler;  **import** android.os.Message;  **public** **class** MessageHandler **extends** Handler{  **public** **static** **int** *MSG\_CHART\_DATA* = 1;  **private** MessageListener listener;  **public** MessageHandler(){  }  **public** **void** setMessageListener(MessageListener l){  **this**.listener = l;  }  @Override  **public** **void** handleMessage(Message msg){  **if**(listener == **null**) **return**;  **if**(msg.what == *MSG\_CHART\_DATA*){  ChartModel model = (ChartModel) msg.obj;  listener.onChartData(model);  }  }  } |

The handleMessage() method is called when a message is delivered to the Handler object. Subclasses of the Handler class must implement this method to receive messages.

The Message is a simple data structure in android.os package which is used to send data to a Handler. It has several public variables and methods to store data such as Message.what to store a message code and Message.obj to store an arbitrary object.

Messages are sent by a worker thread by calling sendMessage() method of the Handler class.

It is common to define a code for each message and assign it to the Message.what variable. It helps the recipient to identify the message. We defined MSG\_CHART\_DATA MessageHandler class.

|  |
| --- |
| **public** **static** **int** *MSG\_CHART\_DATA* = 1; |

#### ChartDataThread

The ChartDataThread class is a worker thread responsible for fetching market data.

|  |
| --- |
| package com.example.graphics2d;  import android.os.Message;  public class ChartDataThread extends Thread {  private MessageHandler handler;  private String ticker;  public ChartDataThread(MessageHandler handler, String ticker) {  this.handler = handler;  this.ticker = ticker;  }    public void run() {  try {  Thread.sleep(5000);  }catch(Exception ex) {  }  ChartModel data = new ChartModel();  Message msg = Message.obtain();  msg.what = MessageHandler.MSG\_CHART\_DATA;  msg.obj = data;  handler.sendMessage(msg);  }  } |

The constructor has two parameters: MessageHandler to send messages to the main thread and a stock ticker. To simplify the example we will use the same hard-coded S&P 500 market data for all queries. Therefore a stock ticker is not used. Also we added a 5 seconds delay to simulate a network communication delay as shown below.

|  |
| --- |
| Thread.sleep(5000); |

You can modify this example to get real market data from Yahoo! Finance of another source.

After the delay, a ChartModel is created. This is the same class we used in the previous chapter. It has hard-coded historical S&P 500 market data. If you decide to fetch market data from Yahoo! Finance or another source, you will have to modify the ChartModel to store the data and to calculate minimum and maximum prices.

Next, we got a Message object.

|  |
| --- |
| Message msg = Message.obtain(); |

Although the Message class has a public constructor, the preferred way to get a Message is to call Message.obtain() method which returns an object from the global pool. If you send a lot of messages in your application it will reduce the load on a garbage collector and improve the application performance.

Next, we set a message type. This step is optional, but usually it helps the recipient to identify what this message is about.

|  |
| --- |
| msg.what = MessageHandler.MSG\_CHART\_DATA; |

Then we stored a ChartModel in the Message.

|  |
| --- |
| msg.obj = data; |

Finally, the message is sent to the handler.

|  |
| --- |
| handler.sendMessage(msg); |

### 10.2.2 Layout

The following layout is used by the application. It is stored in **res/layout/chart\_mt.xml** file.

|  |
| --- |
| <?xml version=”1.0” encoding=”utf-8”?>  <LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:layout\_width=*"match\_parent"*  android:layout\_height=*"match\_parent"*  android:orientation=*"vertical"* >  <LinearLayout  android:layout\_width=*"match\_parent"*  android:layout\_height=*"wrap\_content"*  android:orientation=*"horizontal"* >  <EditText  android:id=*"@+id/txtTicker"*  android:layout\_weight=*"1"*  android:layout\_width=*"wrap\_content"*  android:layout\_height=*"wrap\_content"* />  <Button  android:id=*"@+id/btnGo"*  android:layout\_width=*"wrap\_content"*  android:layout\_height=*"wrap\_content"*  android:text=*"Go"* />  </LinearLayout>  <com.example.graphics2d.ChartViewMT  android:id=*"@+id/chartView"*  android:layout\_width=*"match\_parent"*  android:layout\_height=*"wrap\_content"*  android:layout\_weight=*"1"* />  </LinearLayout> |

It has a text view, Go button and a chart view.

### 10.2.3 Activity

The ChartActivity class implements two interfaces: the OnClickListener to handle clicks on “Go” button and the MessageListener to display a chart when a ChartModel object is delivered from the worker thread to the main UI thread.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.app.Activity;  **import** android.content.pm.ActivityInfo;  **import** android.os.Bundle;  **import** android.view.View;  **import** android.view.View.OnClickListener;  **import** android.view.Window;  **import** android.widget.Button;  **import** android.widget.EditText;  **public** **class** ChartActivity **extends** Activity **implements** OnClickListener, MessageListener{  **private** Button btnGo;  **private** EditText txtTicker;  **private** ChartViewMatrix chartView;  **private** MessageHandler handler;    @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  **this**.setRequestedOrientation(  ActivityInfo.***SCREEN\_ORIENTATION\_LANDSCAPE***);  **this**.requestWindowFeature(Window.***FEATURE\_NO\_TITLE***);  **this**.setContentView(R.layout.***activity\_chart***);  txtTicker = (EditText) findViewById(R.id.***txtTicker***);  btnGo = (Button) findViewById(R.id.***btnGo***);  btnGo.setOnClickListener(**this**);  chartView = (ChartViewMatrix) findViewById(R.id.***chartView***);  //Handler  handler = **new** MessageHandler();  handler.setMessageListener(**this**);  }  @Override  **public** **void** onClick(View v) {  btnGo.setEnabled(**false**);  txtTicker.setEnabled(**false**);  chartView.setLoading(**true**);  String ticker = txtTicker.getText().toString();  ChartDataThread thread = **new** ChartDataThread(handler, ticker);  thread.start();  }  @Override  **public** **void** onChartData(ChartModel model) {  chartView.setModel(model);  btnGo.setEnabled(**true**);  txtTicker.setEnabled(**true**);  }  } |

In the constructor we set the preferred screen orientation and removed the title.

|  |
| --- |
| **this**.setRequestedOrientation(  ActivityInfo.***SCREEN\_ORIENTATION\_LANDSCAPE***);  **this**.requestWindowFeature(Window.***FEATURE\_NO\_TITLE***); |

In previous examples we created a content view programmatically, but in this example we loaded the view from the layout XML.

|  |
| --- |
| **this**.setContentView(R.layout.***activity\_chart***); |

After the content view is created, we stored references to the text field, Go button and chart view in class variables.

|  |
| --- |
| txtTicker = (EditText) findViewById(R.id.***txtTicker***);  btnGo = (Button) findViewById(R.id.***btnGo***);  chartView = (ChartViewMatrix) findViewById(R.id.***chartView***); |

We also set OnClickListener of the Go button.

|  |
| --- |
| btnGo.setOnClickListener(**this**); |

Finally we created the MessageHandler and set its message listener.

|  |
| --- |
| handler = **new** MessageHandler();  handler.setMessageListener(**this**); |

The MessageHandler is created in the main UI thread, therefore all message sent from a worker thread will be delivered to the main UI thread.

The onClick() method is called when Go button is clicked. This method is defined in OnClickListener interface.

|  |
| --- |
| @Override  **public** **void** onClick(View v) {  btnGo.setEnabled(**false**);  txtTicker.setEnabled(**false**);  chartView.setLoading(**true**);  String ticker = txtTicker.getText().toString();  ChartDataThread thread = **new** ChartDataThread(handler, ticker);  thread.start();  } |

In this method we disabled the text field and Go button to prevent a user from submitting a new request before the previous request is completed.

|  |
| --- |
| btnGo.setEnabled(**false**);  txtTicker.setEnabled(**false**); |

Also we requested the chart view to show the “Loading…” message.

|  |
| --- |
| chartView.setLoading(**true**); |

Next we got a stock ticker from the EditText control. To simplify this example we don’t validate the ticker.

|  |
| --- |
| String ticker = txtTicker.getText().toString(); |

Finally, we created and started a worker thread.

|  |
| --- |
| ChartDataThread thread = **new** ChartDataThread(handler, ticker);  thread.start(); |

We passed the handler and the stock ticker to the thread. The ChartDataThread fetches market data from the ticker, creates a ChartModel and sends a message with the ChartModel to the handler. The handler processes the message and calls the onChartData() method with the ChartModel as a parameter.

|  |
| --- |
| @Override  **public** **void** onChartData(ChartModel model) {  chartView.setModel(model);  btnGo.setEnabled(**true**);  txtTicker.setEnabled(**true**);  } |

The onChartData() method is defined in the MessageListener interface and implemented by the ChartActivity. It runs in the main UI thread. In this method we requested the chart view to display a chart

|  |
| --- |
| chartView.setModel(model); |

and then enabled the text field and Go button.

|  |
| --- |
| btnGo.setEnabled(**true**);  txtTicker.setEnabled(**true**); |

### 10.2.4 ChartView

This class is responsible for displaying a stock chart and the “Loading…” message. It is similar to the chart view from the previous chapter.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Matrix;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Align;  **import** android.graphics.Paint.Cap;  **import** android.graphics.Paint.Join;  **import** android.graphics.Paint.Style;  **import** android.graphics.Path;  **import** android.util.AttributeSet;  **import** android.view.View;  **public** **class** ChartViewMatrix **extends** View{  **private** Matrix matrix;  **private** ChartModel model;  **private** Paint paint, textPaint;  **private** Path path;  **private** **int** deviceWidth, deviceHeight;  **private** **boolean** loading = **false**;    **public** ChartViewMatrix(Context context) {  **super**(context);  init();  }  **public** ChartViewMatrix(Context context, AttributeSet attrs) {  **super**(context,attrs);  init();  }    **private** **void** init() {  setBackgroundColor(Color.***YELLOW***);  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setStyle(Style.***STROKE***);  paint.setColor(Color.***BLUE***);  paint.setStrokeWidth(5);  paint.setStrokeCap(Cap.***ROUND***);  paint.setStrokeJoin(Join.***ROUND***);  textPaint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  textPaint.setTextAlign(Align.***CENTER***);  matrix = **new** Matrix();  }    @Override  **public** **void** onDraw(Canvas canvas) {  **if**(model == **null**) {  **if**(loading) {  canvas.drawText("Loading...", deviceWidth/2, deviceHeight/2, textPaint);  }  }  **else** {  canvas.drawPath(path, paint);  }  }    @Override  **protected** **void** onSizeChanged(**int** w, **int** h, **int** oldw, **int** oldh) { deviceHeight = h;  deviceWidth = w;  textPaint.setTextSize(h\*0.1f);  }    **public** **void** setLoading(**boolean** b) {  **this**.loading = b;  **this**.model = **null**;  invalidate();  }    **public** **void** setModel(ChartModel m) {  loading = **false**;  **this**.model = m;  updatePath();  invalidate();  }    **private** **void** updatePath() {  //Init matrix  matrix.reset();  matrix.postTranslate(0, -model.getMinPrice());  **float** scaleX = (**float**)  deviceWidth/(model.getMaxDay()-model.getMinDay());  **float** scaleY = (**float**)  deviceHeight/(model.getMaxPrice()-model.getMinPrice());  matrix.postScale(scaleX, -scaleY);  matrix.postTranslate(0, deviceHeight);  //Create path  path = **new** Path();  path.moveTo(0, model.getPrice(0));  **for**(**int** i = 1; i < model.getNumbersOfDays();i++) {  path.lineTo(i, model.getPrice(i));  }  //Convert coordinate  path.transform(matrix);  }  } |

We added one more constructor which is called when the view is created from an XML file, supplying attributes.

|  |
| --- |
| **public** ChartViewMatrix(Context context, AttributeSet attrs) {  **super**(context,attrs);  init();  } |

Also we moved all initialization code to a separate method.

|  |
| --- |
| **private** **void** init() {  setBackgroundColor(Color.***YELLOW***);  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setStyle(Style.***STROKE***);  paint.setColor(Color.***BLUE***);  paint.setStrokeWidth(5);  paint.setStrokeCap(Cap.***ROUND***);  paint.setStrokeJoin(Join.***ROUND***);  textPaint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  textPaint.setTextAlign(Align.***CENTER***);  matrix = **new** Matrix();  } |

We added loading flag and setLoading() method.

|  |
| --- |
| **private** **boolean** loading = **false**;  …  **public** **void** setLoading(**boolean** b) {  **this**.loading = b;  **this**.model = **null**;  invalidate();  } |

After we set loading and model variables, we called the invalidate() method to request the repaint of the view. This will trigger a called to onDraw() method.

|  |
| --- |
| @Override  **public** **void** onDraw(Canvas canvas) {  **if**(model == **null**) {  **if**(loading) {  canvas.drawText("Loading...", deviceWidth/2, deviceHeight/2, textPaint);  }  }  **else** {  canvas.drawPath(path, paint);  }  } |

If model variable is not null, a stock chart is displayed. If model variable is null and loading flag is set the “Loading…” message is displayed.

We also added setModel() method.

|  |
| --- |
| **public** **void** setModel(ChartModel m) {  loading = **false**;  **this**.model = m;  updatePath();  invalidate();  } |

It sets loading flag to false, stores a ChartModel in model, creates new Path object and calls invalidate() to refresh the view.

The updatePath() method creates a Path from the ChartModel. This is the same code we used in the previous chapter. Before it was in onSizeChange() method.

|  |
| --- |
| **private** **void** updatePath() {  //Init matrix  matrix.reset();  matrix.postTranslate(0, -model.getMinPrice());  **float** scaleX = (**float**)  deviceWidth/(model.getMaxDay()-model.getMinDay());  **float** scaleY = (**float**)  deviceHeight/(model.getMaxPrice()-model.getMinPrice());  matrix.postScale(scaleX, -scaleY);  matrix.postTranslate(0, deviceHeight);  //Create path  path = **new** Path();  path.moveTo(0, model.getPrice(0));  **for**(**int** i = 1; i < model.getNumbersOfDays();i++) {  path.lineTo(i, model.getPrice(i));  }  //Convert coordinate  path.transform(matrix);  } |

In the onSizeChange() method we stored the view width and height and calculated text size for the “Loading…” message.

|  |
| --- |
| @Override  **protected** **void** onSizeChanged(**int** w, **int** h, **int** oldw, **int** oldh) { deviceHeight = h;  deviceWidth = w;  textPaint.setTextSize(h\*0.1f);  } |

## 10.3 SurfaceView

In addition to the Handler class you can use the SurfaceView class to update the UI from a worker thread. The SurfaceView extends the View class, therefore you can override its onDraw() method to do the drawing as we did in previous chapters. In addition to that, the SurfaceView provides another drawing surface located behind the view. You can access this second drawing surface via the SurfaceHolder interface as shown below.

|  |
| --- |
| SurfaceView view;  …  SurfaceHolder holder = view.getHolder();  Canvas canvas = holder.lockCanvas(); |

Once the canvas object is locked by a thread, only this thread can draw on the Canvas. Note that this is not the same Canvas you get in onDraw() method. The lock is released by calling the following method.

|  |
| --- |
| holder.unlockCanvasAndPost(canvas); |

After this call, the UI will be updated and other threads can access the surface again.

You can update the drawing surface provided by the SurfaceHolder from any thread. If you want to update other views, such as buttons ro text fields, or the main drawing area of the SurfaceView inherited from the View class you have to do this in the main UI thread.

Remember that the second drawing surface is behind a window (main drawing area) of the SurfaceView. When you set the SurfaceView background color, it is applied to the windows of a view. If you use an opaquecolor, such as Color.WHITE(oxFFFFFFFF) you would not see the surface underneath the window (main drawing area).

If you want to receive information about changes to the surface, you have to implement the SurfaceHolder.Callback interface. It defines the following methods:

|  |
| --- |
| void surfaceCreated(SurfaceHolder holder);  void surfaceChanged(SurfaceHolder holder, int format, int width, int height);  void surfaceDestroyed(SurfaceHolder holder); |

The surfaceCreated() is called immediately after the surface is first created. It can be used to set a background color and show “Loading…” message.

The surfaceChanged() method is called when the format or size of a surface changes. The method is called at least once. It can be used instead of onSizeChanged() method to determine the size of a surface view.

The surfaceDestroyed() is called immediately before a surface is destroyed.

### 10.3.1 Example: Progress Indicator

In this Example we will create a circular progress indicator similar to the circular chart example discussed in Section 6.4 of Chapter6. We will use a SurfaceView to draw the indicator and a separate thread to update it. A progress thread starts when the application starts. It increases the progress value by 10% every half second until it reaches 100%. Below are several screenshots of the application.

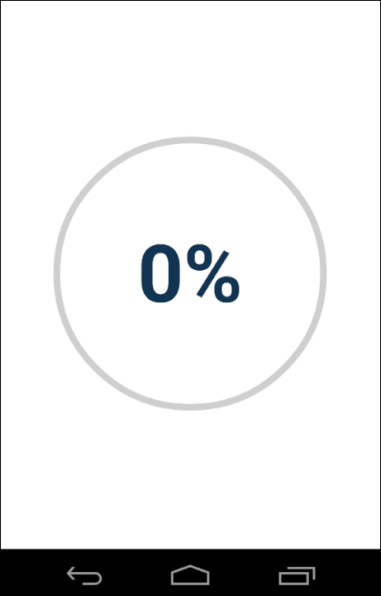


Figure 10-5.



Figure 10-6.



Figure 10-7.

Let’s review different classes we used in this example.

#### ProgressActivity

This is the main activity of our application.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.app.Activity;  **import** android.os.Bundle;  **public** **class** ProgressActivity **extends** Activity {  **private** ProgressView view;  **private** ProgressThread thread;  @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  view = **new** ProgressView(**this**);  this.setContentView(view);  }  @Override  **protected** **void** onStart()  {  **super**.onStart();  thread = **new** ProgressThread(view);  thread.start();  }    @Override  **protected** **void** onStop()  {  **super**.onStop();  **if**(thread == **null**) **return**;  thread.requestStop();  // Wait for the thread to stop  **boolean** retry = **true**;  **while**(retry)  {  **try**  {  thread.join();  retry = **false**;  }  **catch**(InterruptedException e)  {  }  }  thread = **null**;  }  } |

The onCreate() method is called when the activity is first created. In this method we created the progressView programmatically and set it as the content view of the activity.

|  |
| --- |
| view = **new** ProgressView(**this**);  this.setContentView(view); |

The onStart() method is called when the activity is becoming visible to the user. It is a convenient place to create and start a progressive thread.

|  |
| --- |
| thread = **new** ProgressThread(view);  thread.start(); |

The onStop() method is called when the activity is no longer visible to the user. This is a convenient place to stop a progress thread if it is still running. We used the following metod to stop the thread.

|  |
| --- |
| thread.requestStop(); |

We will discuss it when we review the ProgressThread class.

The following code is used to wait for a progress thread to stop.

|  |
| --- |
| **boolean** retry = **true**;  **while**(retry)  {  **try**  {  thread.join();  retry = **false**;  }  **catch**(InterruptedException e)  {  }  } |

The thread.join() method waits for the progress thread to stop, but it can throw an InterruptedException. Therefore we added a while loop to retry the thread.join() in case of an exception.

#### ProgressHelper

The ProgressHelper class is responsible for drawing the progress indicator.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Align;  **import** android.graphics.Paint.Style;  **import** android.graphics.Rect;  **import** android.graphics.RectF;  **import** android.graphics.Typeface;  **public** **class** ProgressHelper {  **private** **float** centerX;  **private** **float** centerY;  **private** **float** radius;  **private** **float** textY;  **private** RectF arcOval;  **private** Rect textBounds;  **private** Paint paint;  **private** Paint txtPaint;  **private** **int** color1 = Color.***LTGRAY***;  **private** **int** color2 = 0xff333377;  **public** ProgressHelper(Context context)  {  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setStyle(Style.***STROKE***);    txtPaint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  txtPaint.setColor(color2);  txtPaint.setTypeface(Typeface.***DEFAULT\_BOLD***);  txtPaint.setTextAlign(Align.***CENTER***);    arcOval = **new** RectF();  textBounds = **new** Rect();  }    **public** **void** draw(Canvas canvas, **float** value)  {  //Draw a circle  paint.setStrokeWidth(radius\*0.05f);  paint.setColor(color1);  canvas.drawCircle(centerX, centerY, radius, paint);    //Draw arc  paint.setColor(color2);  paint.setStrokeWidth(radius\*0.4f);  canvas.drawArc(arcOval, -90, value \* 360, **false**, paint);    //Draw label  String label = String.*format*("%.0f%%", value\*100);  canvas.drawText(label, centerX, textY, txtPaint);  }    **public** **void** changeSize(**int** w, **int** h)  {  //center  centerX = w/2.0f;  centerY = h/2.0f;    //Radius  radius = Math.*min*(w, h)\*0.7f/2;    //Text Size  txtPaint.setTextSize(radius \* 0.6f);    //Center text  String label = "100";  txtPaint.getTextBounds(label, 0, label.length(), textBounds);  textY = centerY - (textBounds.top + textBounds.bottom)\*0.5f;    //Arc oval  arcOval.left = centerX - radius;  arcOval.right = centerX + radius;  arcOval.top = centerY - radius;  arcOval.bottom = centerY + radius;  }  } |

This class is similar to the CircularChart class from Chapter 6. Unlike the CircularChart it doesn’t extend a View, but everything else is almost the same. In the constructor we initialized Paint objects and rectangles. We let Android align text horizontally, instead of calculating textX coordinate ourselves like we did in Chapter 6.

|  |
| --- |
| txtPaint.setTextAlign(Align.***CENTER***); |

In the changeSize() method which is similar to onSizeChange() from Chapter 6, we calculated the x- and y-coordinates of the center of the view and the radius of the circle. We set the text size and calculated the textY coordinate to center text vertically. We used the hard-coded value of 100 to measure the top and bottom of a label.

|  |
| --- |
| String label = "100";  txtPaint.getTextBounds(label, 0, label.length(), textBounds);  textY = centerY - (textBounds.top + textBounds.bottom)\*0.5f; |

Usually all digits of a font have the same top and bottom values, so instead of 100 we could have used any single digit. Finally we set the arc oval coordinates.

The draw() method takes two parameters: the Canvas to draw on and a value to draw. The value can be any number from 0 to 1. The code is similar to the onDraw() method of the CircularChart class from Section 6.4 of Chapter 6. In the CircularChart example we hard-coded values to 0.65 or 65%, but in the PargressHelper class we pass this value as a parameter.

#### ProgressView

This is the content view of the application.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.view.SurfaceHolder;  **import** android.view.SurfaceView;  **public** **class** ProgressView **extends** SurfaceView{  **private** ProgressHelper progress;  **public** ProgressView(Context context) {  **super**(context);  progress = **new** ProgressHelper(context);  }    @Override  **protected** **void** onSizeChanged(**int** width, **int** height, **int** oldw, **int** oldh)  {  progress.changeSize(width, height);  }  **public** **void** drawProgress(**int** value) {  SurfaceHolder holder = **this**.getHolder();  Canvas canvas = **null**;  **try** {  canvas = holder.lockCanvas();  **if**(canvas != **null**){  canvas.drawColor(Color.***WHITE***);  progress.draw(canvas, value/100.0f);  }  }  **finally** {  **if**(canvas != **null**){  holder.unlockCanvasAndPost(canvas);  }  }  }  } |

The ProgressiveView class extends the SurfaceView. It delegates most of the drawing to the ProgressiveHelper class.

In the constructor we created a ProgressiveHelper class.

|  |
| --- |
| progress = **new** ProgressHelper(context); |

The onSizeChanged() method overrides the corresponding method of the View class. We delegated all calculations to the ProgressHelper class.

|  |
| --- |
| progress.changeSize(width, height); |

The drawProgress() method is called by a worker thread to update the UI. This method takes a progressivevalue in 0 to 100 range as a parameter.

Before we can draw, we have to get a reference to the Canvas object from the SurfaceHolder.

|  |
| --- |
| SurfaceHolder holder = **this**.getHolder();  Canvas canvas = **null**;  **. . .**  canvas = holder.lockCanvas(); |

After we got the reference to the Canvas object, we cleared the view

|  |
| --- |
| canvas.drawColor(Color.***WHITE***); |

and delegated drawing of the progress indicator to the ProgressHelper class.

|  |
| --- |
| progress.draw(canvas, value/100.0f); |

We used try-finally statement to make sure that even in case of an exception, the Canvas is unlocked.

|  |
| --- |
| **try** {  . . .  }  **finally** {  . . .  holder.unlockCanvasAndPost(canvas);  } |

#### ProgressThread

This class acts as a timer. It increases the progress value by 10% every half second until it reaches 100% and updates the progress view.

|  |
| --- |
| **package** com.example.graphics2d;  **public** **class** ProgressThread **extends** Thread {  **private** **volatile** **boolean** stopRequested = **false**;  **private** ProgressView view;  **private** **int** progress;    **public** ProgressThread(ProgressView view){  **this**.view = view;  }    **public** **void** run() {  **while**(!stopRequested && progress <= 100) {  view.drawProgress(progress);  **try**  {  Thread.*sleep*(500);  }**catch**(Exception e) {  **break**;  }  progress +=10;  }  }    **public** **void** requestStop() {  stopRequested = **true**;  }  } |

In the constructor we stored the reference to the ProgressView.

|  |
| --- |
| **public** ProgressThread(ProgressView view)  {  **this**.view = view;  } |

In the run() method, which overrides the corresponding method of the Thread class, we loop while either stop is requested or the progress value reaches 100%.

|  |
| --- |
| **while**(!stopRequested && progress <= 100){  . . .  } |

In the loop we draw the progress indicator

|  |
| --- |
| view.drawProgress(progress); |

sleep for half a second.

|  |
| --- |
| Thread.*sleep*(500); |

And increase the progress value by 10

|  |
| --- |
| progress +=10; |

Let’s also discuss the stopRequested flag and requestStop() method.

|  |
| --- |
| **private** **volatile** **boolean** stopRequested = **false**;  . . .  **public** **void** requestStop() {  stopRequested = **true**;  } |

The common way to stop a Thread which performs repeating operations in a loop is to check a Boolean flag inside the loop. The flag can be changed from another thread. The volatile keyword is used to make sure that the value of stopRequested flag changed in another thread is immediately visible to the ProgressiveThread.

## 10.4 Example: SurfaceView and Handler in the same Application

You can use both the SurfaceView and Handler in the same application.

In this example we will change the stock chart application from Section 10.2 to use the SurfaceView to draw a chart and the “Loading…” message. The SurfaceView can be updated from any thread. We will draw a chart in a worker thread and the “Loading…” message in the main UI thread.

The rest of the UI can be only updated from the main UI thread. We will use the Handler class to send messages from a worker thread to the main UI thread to update the text field and Go button.

Let’s review modified classes and interfaces.

### 10.4.1 MessageListener

This interface defines a method which will be called by a Handler after a chart is drawn to enable the text field and Go button.

|  |
| --- |
| package com.example.graphics2d;  public interface MessageListener {  public void afterDrawChart();  } |

### 10.4.2 MessageHandler

The MessageHandler class extends the Handler class and overrides its handleMessage() method.

|  |
| --- |
| package com.example.graphics2d;  import android.os.Handler;  import android.os.Message;  public class MessageHandler extends Handler{  public static int MSG\_AFTER\_DRAW\_CHART = 1;  private MessageListener listener;  public MessageHandler(){  }  public void setMessageListener(MessageListener l){  this.listener = l;  }    @Override  public void handleMessage(Message msg){  if(listener == null) return;  if(msg.what == MSG\_AFTER\_DRAW\_CHART){  listener.afterDrawChart();  }  }  } |

It is similar to the MessageHandler from Section 10.2. The message has changed. It doesn’t have a ChartModel anymore. After a MSG\_AFTER\_DRAW\_CHART message is received, we notify the listener.

|  |
| --- |
| public static int MSG\_AFTER\_DRAW\_CHART = 1;  …  if(msg.what == MSG\_AFTER\_DRAW\_CHART){  listener.afterDrawChart();  } |

### 10.4.3 ChartSurfaceView

This class is responsible for displaying a stock chart and the “Loading…” message.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.content.Context;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Matrix;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Align;  **import** android.graphics.Paint.Cap;  **import** android.graphics.Paint.Join;  **import** android.graphics.Paint.Style;  **import** android.graphics.Path;  **import** android.util.AttributeSet;  **import** android.view.SurfaceHolder;  **import** android.view.SurfaceView;  **public** **class** ChartSurfaceView **extends** SurfaceView  **implements** SurfaceHolder.Callback {  **private** Matrix matrix;  **private** Paint paint, textPaint;  **private** Path path;  **private** **int** deviceWidth, deviceHeight;    **public** ChartSurfaceView(Context context) {  **super**(context);  init();  }    **public** ChartSurfaceView(Context context, AttributeSet attrs){  **super**(context, attrs);  init();  }    **private** **void** init(){  paint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  paint.setStyle(Style.***STROKE***);  paint.setColor(Color.***BLUE***);  paint.setStrokeWidth(5);  paint.setStrokeCap(Cap.***ROUND***);  paint.setStrokeJoin(Join.***ROUND***);  textPaint = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***);  textPaint.setTextAlign(Align.***CENTER***);  textPaint.setColor(Color.***BLACK***);  matrix = **new** Matrix();  **this**.getHolder().addCallback(**this**);  }    **public** **void** drawLoading() {  SurfaceHolder holder = **this**.getHolder();  Canvas canvas = **null**;  **try** {  canvas = holder.lockCanvas();  **if**(canvas != **null**){  canvas.drawColor(Color.***WHITE***);  canvas.drawText("Loading...",  deviceWidth/2, deviceHeight/2, textPaint);  }  }  **finally**{  **if**(canvas != **null**){  holder.unlockCanvasAndPost(canvas);  }  }  }  **public** **void** drawChart(ChartModel model) {  SurfaceHolder holder = **this**.getHolder();  Canvas canvas = **null**;  **try** {  canvas = holder.lockCanvas();  **if**(canvas != **null**){  updatePath(model);  canvas.drawColor(Color.***WHITE***);  canvas.drawPath(path, paint);  }  }  **finally** {  **if**(canvas != **null**){  holder.unlockCanvasAndPost(canvas);  }  }  }    **private** **void** updatePath(ChartModel model){  //Init transformation matrix  matrix.reset();  matrix.postTranslate(0, -model.getMinPrice());  **float** scaleX = (**float**) deviceWidth /  (model.getMaxDay()-model.getMinDay());  **float** scaleY = (**float**) deviceHeight /  (model.getMaxPrice()-model.getMinPrice());  matrix.postScale(scaleX, -scaleY);  matrix.postTranslate(0, deviceHeight);  //Create path  path = **new** Path();  path.moveTo(0, model.getPrice(0));  **for**(**int** i=1; i < model.getNumbersOfDays();i++){  path.lineTo(i, model.getPrice(i));  }  //Convert coordinates  path.transform(matrix);  }    @Override  **protected** **void** onSizeChanged(**int** w, **int** h, **int** oldw, **int** oldh) {  deviceHeight = h;  deviceWidth = w;  textPaint.setTextSize(h\*0.1f);  }    @Override  **public** **void** surfaceCreated(SurfaceHolder holder) {  Canvas canvas = **null**;  **try** {  canvas = holder.lockCanvas();  **if**(canvas != **null**){  canvas.drawColor(Color.***WHITE***);  }  }  **finally** {  **if**(canvas != **null**){  holder.unlockCanvasAndPost(canvas);  }  }  }  @Override  **public** **void** surfaceChanged(SurfaceHolder holder, **int** format,  **int** width, **int** height){  }  @Override  **public** **void** surfaceDestroyed(SurfaceHolder holder){  }  } |

The ChartSurfaceView extends the SurfaceView. We use its drawing surface instead of the main drawing area inherited from the View class.

Constructors are the same as before.

In the init() method we removed the setBackgroundColor() method.

|  |
| --- |
| setBackgroundColor(Color.***WHITE***); |

The Color.WHITE is opaque. If you keep the setBackgroundColor() method, you would not see the surface underneath the window(main drawing area).

We added drawLoading() methodto draw “Loading…” label and drawChart() method to draw a chart. Both methods get a Canvas from the SurfaceHolder.

|  |
| --- |
| SurfaceHolder holder = **this**.getHolder();  Canvas canvas = **null**;  ...  canvas = holder.lockCanvas(); |

After drawing is done, the canvas is unlocked and UI is refreshed.

|  |
| --- |
| holder.unlockCanvasAndPost(canvas); |

The code for drawing on the Canvas is the same as before.

The ChartSurfaceView implements the SurfaceHolder.Callback interface. We used its surfaceCreated() method to set the surface background color, right after the surface is created.

|  |
| --- |
| @Override  **public** **void** surfaceCreated(SurfaceHolder holder) {  Canvas canvas = **null**;  **try** {  canvas = holder.lockCanvas();  **if**(canvas != **null**){  canvas.drawColor(Color.***WHITE***);  }  }  **finally** {  **if**(canvas != **null**){  holder.unlockCanvasAndPost(canvas);  }  }  } |

### 10.4.4 ChartDataThread

The ChartDataThread class is a worker thread responsible for fetching marker data and drawing a chart.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.os.Message;  **public** **class** ChartDataThread **extends** Thread {  **private** MessageHandler handler;  **private** ChartSurfaceView view;    **public** ChartDataThread(ChartSurfaceView view,  MessageHandler handler, String ticker){  **this**.view = view;  **this**.handler = handler;  **this**.ticker = ticker;  }    **public** **void** run() {  **try** {  Thread.*sleep*(5000);  } **catch**(Exception ex){  }  ChartModel data = **new** ChartModel();  view.drawChart(data);  Message msg = Message.*obtain*();  msg.what = MessageHandler.*MSG\_AFTER\_DRAW\_CHART*;  handler.sendMessage(msg);  }  } |

As before, we used 5 seconds delay to sumulate a network communication delay.

|  |
| --- |
| Thread.*sleep*(5000); |

After that we created a ChartModel with hard-coded S&P 500 market data used for all queries

|  |
| --- |
| ChartModel data = **new** ChartModel(); |

and drew the chart.

|  |
| --- |
| view.drawChart(data); |

Next we created a MSG\_AFTER\_DRAW\_CHART message and sent it to the MessageHandler to notify the main UI thread that the chart is drawn and that the main UI thread can enable the text field and Go button.

|  |
| --- |
| Message msg = Message.*obtain*();  msg.what = MessageHandler.*MSG\_AFTER\_DRAW\_CHART*;  handler.sendMessage(msg); |

The main defference between this class and the ChartDataThread from Section 10.2 is that in this example we drew the chart in the worker thread. In Section 10.2 we sent a ChartModel to the main UI thread and drew the chart threr.

### 10.4.5 Layout

We changed the layout from Section 10.2 to use new ChartSurfaceView. Everything else is the same.

|  |
| --- |
| <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:orientation="vertical" >  <LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:orientation="horizontal" >  <EditText  android:id="@+id/txtTicker"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_weight ="1" />  <Button  android:id="@+id/btnGo"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Go" />  </LinearLayout>  <com.example.graphics2d.ChartSurfaceView  android:id="@+id/chartView"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_weight="1" />  </LinearLayout> |

### 10.4.6 Activity

The ChartActivity is also almost the same. It implements the OnClickListener to handle clicks on Go button and the MessageListener to get notifications from the MessageHandler.

|  |
| --- |
| **package** com.example.graphics2d;  **import** android.app.Activity;  **import** android.content.pm.ActivityInfo;  **import** android.os.Bundle;  **import** android.view.View;  **import** android.view.View.OnClickListener;  **import** android.view.Window;  **import** android.widget.Button;  **import** android.widget.EditText;  **public** **class** ChartActivity **extends** Activity  **implements** OnClickListener, MessageListener {  **private** Button btnGo;  **private** EditText txtTicker;  **private** ChartSurfaceView chartView;  **private** MessageHandler handler;    @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  **this**.setRequestedOrientation(  ActivityInfo.***SCREEN\_ORIENTATION\_LANDSCAPE***);  **this**.requestWindowFeature(Window.***FEATURE\_NO\_TITLE***);  setContentView(R.layout.***activity\_chart***);  txtTicker = (EditText) findViewById(R.id.***txtTicker***);  btnGo = (Button) findViewById(R.id.***btnGo***);  btnGo.setOnClickListener(**this**);  chartView = (ChartSurfaceView) findViewById(R.id.***chartView***);  handler = **new** MessageHandler();  handler.setMessageListener(**this**);  }    @Override  **public** **void** onClick(View v) {  btnGo.setEnabled(**false**);  txtTicker.setEnabled(**false**);  chartView.drawLoading();  String ticker = txtTicker.getText().toString();  ChartDataThread thread =  **new** ChartDataThread(chartView, handler, ticker);  thread.start();  }    @Override  **public** **void** afterDrawChart(){  btnGo.setEnabled(**true**);  txtTicker.setEnabled(**true**);  }  } |

We didn’t change the constructor.

In the onClick() method which is called when Go button is clicked, we disabled the text field and Go button to prevent a user from submitting a new request before the previous request is completed.

|  |
| --- |
| btnGo.setEnabled(**false**);  txtTicker.setEnabled(**false**); |

Next we drew the “Loading…” message.

|  |
| --- |
| chartView.drawLoading(); |

After getting a stock ticker from the EditText control we created and started a worker thread.

|  |
| --- |
| ChartDataThread thread =  **new** ChartDataThread(chartView, handler, ticker);  thread.start(); |

The worker thread fetches market data for a stock ticker and draws a chart. After that it sends a MSG\_AFTER\_DRAW\_CHART message to the MessageHandler. The MessageHandler handles the message and calls the afterDrawChart() method of the MessageListener interface.

|  |
| --- |
| @Override  **public** **void** afterDrawChart(){  btnGo.setEnabled(**true**);  txtTicker.setEnabled(**true**);  } |

In this method we enabled thetext field and Go button.

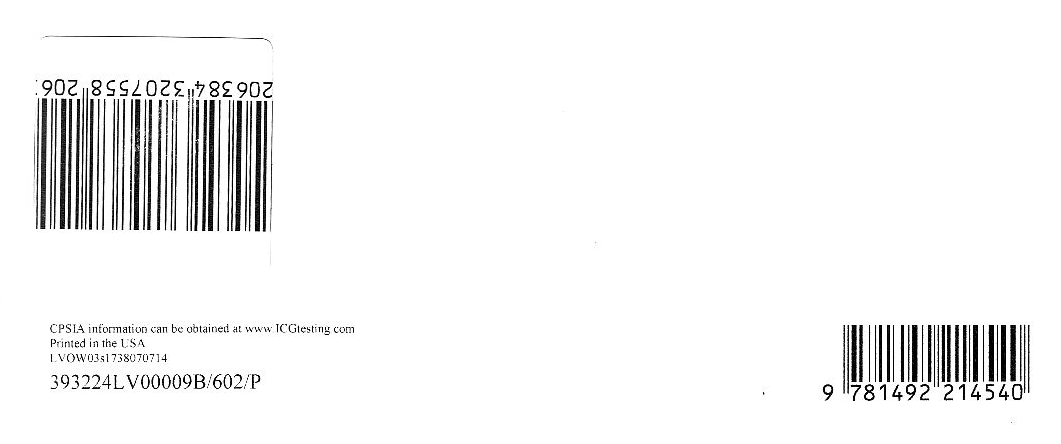
The main difference between this class and the old version from Section 10.2 is that before we drew a chart in the main UI thread in the onChartData() method by calling

|  |
| --- |
| chartView.setModel(model); |

In this example we drew a chart in a worker thread.

# Index

|  |  |
| --- | --- |
| Alignment  Custom,53 |  |



Contacting the Author

If you have any comments of find an error in the text or I the code, we would be grateful for your feedback. You can contact the author at yevgen.books@gmail.com.

## Android 2D Graphics with Canvas API

### Canvas API

Google doesn’t have any official name for this library. It is just a collection of Java classes in android.graphics package. The main class which provides methods for drawing graphics primitives is called Canvas, therefore some authors call this library Canvas or Canvas API. In this book we will also use this name.

Canvas API is an advanced two-dimensional graphics library. It provides methods for drawing text, lines, rectangles, circles and other graphics primitives. Since Android 3.0 the majority of the drawing done by the Canvas API can be hardware accelerated.

### Intended Audience

This book is intended for programmers interested in learning how to use Canvas API in Android. We expect the reader to have some knowledge of Java and Android SDK. We also assume you understand basic Android concepts like Activities and Views and know how to create and run simple Android applications.

In this book we will show how to use different graphics primitives, explain basics of coordinate systems and transformations, discuss two common approaches to writing multithreaded graphics applications, and provide a lot of examples.



glossary