Java SE 11 與 Android 9.x 程式設計範例教本

習題參考解答 2018/12/25

第 1 章：Java 語言與流程圖的基礎

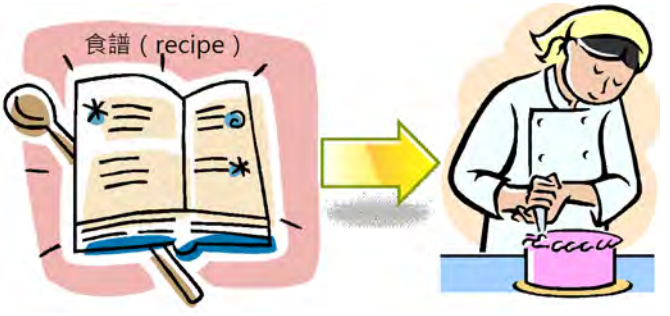
1.

認識程式

程式（Programs）或稱為「電腦程式」（Computer Programs）可以描述電腦如何完成指定工作，其內容是完成指定工作的步驟，撰寫程式就是寫下這些步驟，如同作曲寫下的曲譜、設計房屋的藍圖或烹調食物的食譜。例如：描述烘焙蛋糕過程的食譜（Recipe），可以告訴我們如何製作蛋糕；Word 電腦程式可以幫助我們編輯文件。

以電腦術語來說，程式是使用指定程式語言（Program Language）所撰寫沒有混淆文字、數字和鍵盤符號組成的特殊符號，這些符號組合成指令敘述和程式區塊，再進一步編寫成程式碼，程式碼可以告訴電腦解決指定問題的步驟。

基本上，電腦程式的內容主要分為兩大部分：資料（Data）和處理資料的操作（Operations）。對比烘焙蛋糕的食譜，資料是烘焙蛋糕原料的水、蛋和麵粉等成份，再加上器具的烤箱，在食譜描述的烘焙步驟是處理資料的操作，可以將這些成份製作成蛋糕，如下圖所示：



事實上，我們可以將程式視為一個資料轉換器，當使用者從電腦鍵盤或滑鼠輸入資料後，執行程式是在執行資料處理的操作，可以將資料轉換成有用的資訊，如下圖所示：

上述輸出結果可能是顯示在螢幕或從印表機印出，電腦只是依照程式的指令將輸入資料進行轉換，以產生所需的輸出結果。對比烘焙蛋糕，我們依序執行食譜描述的烘焙步驟，就可以一步一步混合、攪拌和揉合水、蛋和麵粉等成份後，放入烤箱來製作出蛋糕。

請注意！為了讓電腦能夠看懂程式，程式需要依據程式語言的規則、結構和語法，以指定文字或符號來撰寫程式，例如：使用 Java 語言撰寫的程式稱為 Java 程式碼（Java Code）或稱為「原始碼」（Source Code）。

程式邏輯

我們使用程式語言的主要目的是撰寫程式碼來建立程式，所以需要使用電腦的程式邏輯（Program Logic）來撰寫程式碼，如此電腦才能執

行程式碼來解決我們的問題，因為電腦才是真正的「目標執行者」（Target Executer），負責執行你寫的程式；並不是你的大腦在執行。

讀者可能會問撰寫程式碼執行程式設計（Programming）很困難嗎？事實上，如果你能夠一步一步詳細列出活動流程、導引問路人到達目的地、走迷宮、從電話簿中找到電話號碼或從地圖上找出最短路徑，就表示你一定可以撰寫程式碼。

請注意！電腦一點都不聰明，不要被名稱誤導，因為電腦真正的名稱應該是「計算機」（Computer），一台計算能力非常好的計算機，並沒有思考能力，更不會舉一反三，所以，我們需要告訴電腦非常詳細的步驟和資訊，絕對不能有模稜兩可的內容，而這就是電腦使用的程式邏輯。

例如：開車從高速公路北上到台北市大安森林公園，然後分別使用人類的邏輯和電腦的程式邏輯來寫出其步驟。

人類的邏輯

因為目標執行者是人類，對於人類來說，我們只需檢視地圖，即可輕鬆寫下開車從高速公路北上到台北市大安森林公園的步驟，如下所示：

Step 1：中山高速公路向北開。
Step 2：下圓山交流道（建國高架橋）。
Step 3：下建國高架橋（仁愛路）。
Step 4：直行建國南路，在紅綠燈右轉仁愛路。
Step 5：左轉新生南路。

上述步驟告訴人類的話（使用人類的邏輯），這些資訊已經足以讓我們開車到達目的地。

電腦的程式邏輯

對於目標執行者電腦來說，如果將上述步驟告訴電腦，電腦一定完全沒有頭緒，不知道如何開車到達目的地，因為電腦一點都不聰明，這些步驟的描述太不明確，我們需要提供更多資訊給電腦（請改用電腦的程式邏輯來思考），才能讓電腦開車到達目的地，如下所示：

- 從哪裡開始開車（起點）？中山高速公路需向北開幾公里到達圓山交流道？
- 如何分辨已經到了圓山交流道？如何從交流道下來？
- 在建國高架橋上開幾公里可以到達仁愛路出口？如何下去？
- 直行建國南路幾公里可以看到紅綠燈？左轉或右轉？
- 開多少公里可以看到新生南路？如何左轉？接著需要如何開？如何停車？

所以，撰寫程式碼時需要告訴電腦非常詳細的動作和步驟順序，如同教導一位小孩作一件他從來沒有作過的事，例如：綁鞋帶、去超商買東西或使用自動販賣機。因為程式設計是在解決問題，你需要將解決問題的詳細步驟一一寫下來，包含動作和順序（即設計演算法），然後將它轉換成程式碼，以本書為例就是撰寫 Java 程式碼。

2. CPU 執行機器語言程式只是一種例行工作，依序將儲存在記憶體的機器語言指令「取出和執行」（Fetch-and-execute）。簡單的說，CPU 是從記憶體取出指令，然後執行此指令，取出下一個指令，再執行它。CPU 執行程式的方式，如下所示：

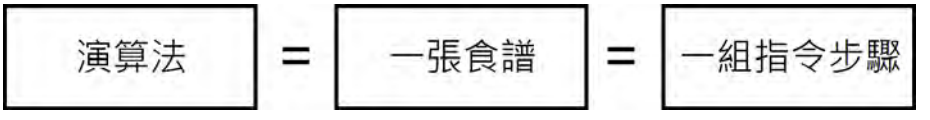
- 在電腦的主記憶體儲存機器語言的程式碼和資料。

- CPU 從記憶體依序取出一個個機器語言指令，然後依序的執行它。

所以，CPU 並非真正了解機器語言在作什麼？這只是 CPU 的例行工作，依序執行機器語言指令，所以使用者讓 CPU 執行的程式不能有錯誤，因為 CPU 只是執行它，並不會替您的程式擦屁股。

3. ASCII、Big 5、2

4. 「演算法」（Algorithms）簡單的說就是一張食譜（Recipe），提供一組一步接著一步（Step-by-step）的詳細過程，包含動作和順序，可以將食材烹調成美味的食物，例如：在第 1-1-1 節說明的蛋糕製作，製作蛋糕的食譜就是一個演算法，如下圖所示：



電腦科學的演算法是用來描述解決問題的過程，也就是完成一個任務所需的具體步驟和方法，這個步驟是有限的；可行的，而且沒有模稜兩可的情況。

認識流程圖

流程圖是使用簡單的圖示符號來表示程式邏輯步驟的執行過程，可以提供程式設計者一種跨程式語言的共通語言，作為與客戶溝通的工具和專案文件。如果我們可以畫出流程圖的程式執行過程，就一定可以將它轉換成指定的程式語言，以本書為例是撰寫成 Java 程式碼。

所以，就算你是一位完全沒有寫過程式碼的初學者，也一樣可以使用流程圖來描述執行過程，以不同形狀的圖示符號表示操作，在之間使

用箭頭線標示流程的執行方向，筆者稱它為圖形版程式（對比程式語言的文字版程式）。

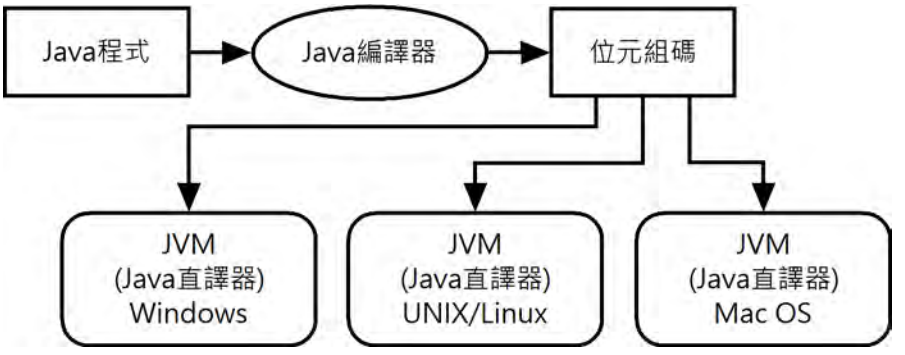
本書提供的 fChart 流程圖直譯器是建立圖形版程式的最佳工具，因為你不只可以編輯繪製流程圖，更可以執行流程圖來驗證演算法的正確性，完全不用涉及程式語言的語法，就可以輕鬆開始寫程式。

5. 「平台」（Platform）是一種結合硬體和軟體的執行環境，簡單的說，Java 程式是在平台上執行，因為 Java 屬於一種與硬體無關和跨平台的程式語言，所以 Java 平台是一種軟體平台，主要是由 JVM 和 Java API 兩個元件所組成。

Java 原始程式碼（副檔名.java）在編譯成位元組碼（副檔名.class）後，即可在 Windows、UNIX 或 Machintosh 的 Mac OS 作業系統上執行，只需作業系統安裝 JVM，同一個位元組碼檔案就可以跨平台在不同作業系統上，使用 Java 直譯程式來正確執行 Java 應用程式。

6. 「JVM」（Java Virtual Machine）虛擬機器是一台軟體的虛擬電腦，Java 原始程式碼並不是使用 Java 編譯器（Java Compiler）編譯成其安裝實體電腦可執行的機器語言，而是 JVM 虛擬機器的機器語言，稱為「位元組碼」（Bytecode）。

位元組碼是一種可以在 JVM 執行的程式，換句話說，電腦作業系統需要安裝 JVM，才能夠使用 Java 直譯器（Java Interpreter）來直譯和執行位元組碼，如下圖所示：



上述圖例的 Java 原始程式碼（副檔名.java）在編譯成位元組碼（副檔名.class）後，即可在 Windows、UNIX 或 Machintosh 的 Mac OS 作業系統上執行，只需作業系統安裝 JVM，同一個位元組碼檔案就可以跨平台在不同作業系統上，使用 Java 直譯器來執行 Java 應用程式。

7. Java 是一種高階和物件導向程式語言，其語法和 C/C++語言十分相似，支援 Windows、Solaris、Linux 和 Mac OS X 作業系統，針對不同需求提供多種版本，例如：企業版和標準版（Standard Edition、SE）。在本書說明的是 Java SE 標準版，其版本演進如下表所示：

版本	日期	說明
1.0	1996/01	Java Development Kit 1.0 版（JDK 1.0）
1.1	1997/02	Java Development Kit 1.1 版（JDK 1.1）
1.2	1998/12	Software Development Kit 1.2 版（SDK 1.2），開始稱為 Java 2 平台
1.3	2000/05	平台名稱 J2SE（Java 2 Platform, Standard Edition），產品名稱是 Software Development Kit 1.3 版（SDK 1.3），也稱為 Java 2 1.3 版

1.4	2002/02	平台名稱 J2SE，產品名稱是 SDK 1.4，也稱為 Java 2 1.4 版
5.0	2004/09	平台名稱 J2SE 5.0，產品名稱是 J2SE Development Kit 5.0 版（JDK 5.0），其開發版號為 JDK 1.5.0
SE 6	2006/12	Java SE Development Kit 6（JDK 6）
SE 7	2011/07	Java SE Development Kit 7（JDK 7）
SE 8	2014/04	Java SE Development Kit 8（JDK 8）
SE 9	2017/09	Java SE Development Kit 9（JDK 9）
SE 10	2018/03	Java SE Development Kit 10（JDK 10）
SE 11	2018/09	Java SE Development Kit 10（JDK 11）

8. Java 語言的開發環境需要安裝 JDK，然後使用編輯工具或整合開發環境來建立 Java 程式，在本書第一部分是使用 fChart 程式碼編輯器來建立 Java 程式，第二部分使用官方 Android Studio 整合開發工具來開發 Android 應用程式。

9. 請參閱第 1-4-1 節的說明。

10. 請參閱第 1-4-2 節的說明。

第 2 章：建立 Java 應用程式

1. 請參閱第 2-1 節的說明。

2. Java 程式檔是副檔名為.java 的純文字檔，內含同名 Java 類別宣告，即類別檔，例如：Ch2_2.java 的類別宣告是 Ch2_2，如下所示：

```
01: public class Ch2_2 {
02:         // 主程式
03:         public static void main(String[] args) {
04:                 System.out.println("第一個 Java 程式");
05:         }
06: }
```

上述 Java 程式架構是使用 public class 關鍵字和大括號括起的類別宣告，類別名稱是 Ch2_2。因為類別宣告在第 7 章才會詳細說明，在此之前讀者可以將它視為 Java 程式的基本結構，其說明如下所示：

- 程式註解：第 2 列是程式註解，詳細的註解說明請參閱第 2-5-3 節，如下所示：

```
02:         // 主程式
```

- 類別宣告：在第 1~6 列是與檔案名稱 Ch2_2 相同的類別宣告，如下所示：

```
01: public class Ch2_2 {
02:         // 主程式
03:         public static void main(String[] args) {
04:                 System.out.println("第一個 Java 程式");
05:         }
06: }
```

上述類別的程式區塊是使用 public 關鍵字宣告的類別，請注意！檔案名稱需要和宣告成 public 類別的名稱相同，不只如此，英文字母大小寫也需相同。

- 主程式：第 3~5 列的 main()方法（即其他程式語言的程序或函數）是 Java 程式的主程式，這是 Java 應用程式執行時的進入點，也就是說執行 Java 程式就是從此方法的第 1 行程式碼開始，如下所示：

```
03:     public static void main(String[] args) {
04:         System.out.println("第一個 Java 程式");
05:     }
```

上述 main()方法宣告成 public、static 和 void，表示是公開、靜態類別和沒有傳回值的方法，詳細修飾子的說明請參閱第 5 和 7 章。在第 6 列呼叫 System 子類別 out 的 println()方法顯示參數的字串，在下一小節有進一步的說明。

主控台的輸出

主控台程式是在命令提示字元輸入和輸出文字內容，我們可以使用 System.out 物件的 print()或 println()方法將文字內容輸出至主控台（Console）來顯示。

因為目前尚未說明物件和方法，請讀者先視為是一種標準 Java 輸出的程式碼，在之後的幾個章節，我們就是使用 println()和 print()兩個方法來輸出執行結果，如下所示：

```
System.out.println("第一個 Java 程式");
```

上述程式碼的 println()方法可以將括號內的參數字串輸出到螢幕顯示且換行，字串是使用「"」號括起一組字元集合。如果不換行就是使用 print()方法，如下所示：

```
System.out.print("第一個 Java 程式");
```

System.out 物件的 print()或 println()方法除了輸出字串內容外，也可以輸出變數值，如下所示：

```
System.out.print("姓名代碼: ");
System.out.println(name);
System.out.print("總額: " + total);
```

上述程式碼的 name 和 total 是變數，輸出的是變數值，如果同時輸出說明字串，請使用「+」加號來連接字串和變數值。

主控台的輸入

Java 主控台輸入是從 System.in 物件讀取資料，為了方便說明，筆者直接使用 java.util.Scanner 類別（此為類別全名，如此就不需匯入套件的類別）的 Scanner 物件來取得輸入資料，如下所示：

```
java.util.Scanner sc = new java.util.Scanner(System.in);
```

上述程式碼使用 new 運算子建立 Scanner 物件，建構子參數是基本輸入 System.in 物件，關於 new 運算子、建構子和套件說明，請參閱本書第 7 和 9 章。

在建立 Scanner 物件後，可以使用相關方法取得使用者輸入的資料，如下所示：

```
String name = sc.nextLine();
int grade = sc.nextInt();
double height = sc.nextDouble();
```

上述程式碼的 nextLine()方法可以取得使用者輸入字串的 String 物件（可以包含空白字元）；nextInt()方法取得輸入的整數值；nextDouble()方法取得浮點數值。完整 Java 程式是 Ch2_4_2.java。

3. main()、.java

4. Java 程式是由程式敘述（Statements）組成，一列程式敘述如同英文的一個句子，內含多個運算式、運算子或 Java 關鍵字（詳見第 3 章的說明）。

程式敘述的範例

一些 Java 程式敘述的範例，如下所示：

```
int total = 1234;
rate = 0.05;
interest = total * rate;
System.out.println("第一個 Java 程式");
```

「;」程式敘述結束符號

Java 的「;」符號代表程式敘述的結束，這是告訴編譯器已經到達程式敘述的最後。換句話說，我們可以使用「;」符號在同一列程式碼撰寫多個程式敘述，如下所示：

```
total = 1234; rate = 0.05; interest = total * rate;
```

上述程式碼在同一列 Java 程式碼列擁有 3 個程式敘述。

程式區塊

程式區塊（Block）是由多個程式敘述組成，它是使用「{」和「}」符號包圍，如下所示：

```
public static void main(String[] args) {
    System.out.println("第一個 Java 程式");
}
```

上述 main()方法的程式碼部分是一個程式區塊，在第 4 和 5 章說明的流程控制敘述和方法都擁有程式區塊。

Java 語言屬於一種「自由格式」（Free-format）的程式語言，我們可以將多個程式敘述寫在同一列，甚至可以將整個程式區塊置於同一列，程式設計者可以自由編排程式碼，如下所示：

```
public static void main(String[] args)  {  }
```

5.

```
System.out.println(“大家好！生日快樂”);
```

6. Test.java、

```
System.out.println(“大家好！”);
System.out.println(“生日快樂”);
```

7. 使用 System.out.println()方法來建立 Java 程式，如下所示：

```
System.out.println(“Java 程式設計入門教材”);
```

8. 使用 System.out.println()方法來建立 Java 程式，如下所示：

```
System.out.println("    **    ");
System.out.println("    **    ");
System.out.println("**********");
System.out.println("    **    ");
System.out.println("    **    ");
```

第 3 章：變數、資料型態與運算子

1. 電腦程式在執行時常常需要記住一些資料，所以程式語言會提供一個地方，用來記得執行時的一些資料，這個地方是「變數」（Variables）。Java 語言的變數名稱就是「識別字」（Identifier）。

變數是什麼

我們去商店買東西時，為了比較價格，我們會記下商品價格，同樣的，程式是使用變數儲存這些執行時需記住的資料，也就是將這些值儲存至變數，當變數擁有儲存值後，就可以在需要的地方取出變數值，例如：執行數學運算和比較等。

對比真實世界，當我們想將零錢存起來時，可以準備一個盒子來存放這些錢，並且隨時看看已經存了多少錢，這個盒子如同一個變數，我

們可以將目前的金額存入變數，或取得變數值來看看已經存了多少錢，如下圖所示：

請注意！真實世界的盒子和變數仍然有一些不同，我們可以輕鬆將錢幣丟入盒子，或從盒子取出錢幣，但是，變數只有兩種操作，如下所示：

- 在變數存入新值：指定變數成為一個全新值，我們並不能如同盒子一般，只取出部分金額。因為變數只能指定成一個新值，如果需要減掉一個值，其操作是先讀取變數值，在減掉後，再將變數指定成最後運算結果的新值。
- 讀取變數值：取得目前變數值，而且讀取變數值，並不會更改變數目前儲存的值。

命名規則

在 Java 程式碼中的「關鍵字」（Keywords），或稱「保留字」（Reserved Word）是一些對於編譯器而言，擁有特殊意義的名稱，其他非關鍵子的名稱都是程式設計者自訂的元素名稱，稱為「識別字」（Identifier），例如：變數、程序、函數和物件名稱等。

程式設計者在替元素命名時，需要遵循程式語言的語法。而且元素命名十分重要，因為一個好名稱如同程式註解，可以讓程式更容易了解。Java 語言的基本命名規則，如下所示：

- 名稱是合法的「識別字」（Identifier），識別字是使用英文字母、底線「_」或錢號「$」開頭，不限長度的 Unicode 統一字碼字元的字串，包含字母、數字、錢號「$」和底線「_」。例如：一些合法的名稱範例，如下所示：

  T, n, size, z100, long_name, helloWord, Test, apple, $total, _order
  Input_string, x, TITLE, APPLE, subtotal, _getTotal, $_32_cpu

- 名稱最長為 255 個字元，而且名稱區分英文字母的大小寫，例如：java、Java 和 JAVA 是不同的名稱。

- 名稱不能使用 Java 語法的「關鍵字」（Keyword），如下表所示：

abstract	boolean	break	byte	case
catch	char	class	const	continue
default	do	double	else	extends
false	final	finally	float	for
goto	if	implements	import	instanceof
int	interface	long	native	new
null	package	private	protected	public
return	short	static	super	switch

synchronized	this	throw	throws	transient
true	try	void	volatile	while

- 名稱在「範圍」（Scope）中必需是唯一的，例如：在程式中可以使用相同的變數名稱，不過各變數名稱需要在不同的範圍，詳細的範圍說明請參閱＜第 5 章：類別方法＞。

2. Total_Score、teamWork、_test、abc、j

3. 變數可以儲存程式執行時的暫存資料，Java 語言的變數在宣告時就可以指定初值，如下所示：

```
資料型態 變數名稱 = 初值;
```

上述語法使用資料型態開頭，在變數名稱後使用「=」等號指定變數初值，例如：175.5 和 76 等字面值。Java 語言變數初值的範例，如下所示：

```
int grade = 76;
double height = 175.5;
double weight = 75.5;
```

上述程式碼宣告 3 個變數，使用 int 和 double 開頭，和分別指定初值為 76、175.5 和 75.5，然後馬上可以使用 3 個 println()方法顯示這 3 個變數值，如下所示：

```
System.out.println("成績 = " + grade);
System.out.println("身高 = " + height);
System.out.println("體重 = " + weight);
```

Java 語言除了一列宣告一個變數外，我們也可以在同一列宣告多個變數，並且分別指定初值，其語法如下所示：

```
資料型態 變數名稱 1=初值 1[, 變數名稱 2=初值 2];
```

上述語法是在變數之間使用「,」號分隔。請另存 Ch3_2_1.java 成為 Ch3_2_1a.java（別忘了類別名稱也需改為 Ch3_2_1a），然後將原來 2 列的變數宣告和指定初值改在同一列，如下所示：

```
double height = 175.5, weight = 75.5;
```

上述程式碼宣告變數 height 和 weight，並且指定 2 個變數的初值，如下圖所示：

```
1  public class Ch3_2_1a {
2      // 主程式
3      public static void main(String[] args) {
4          int grade = 76;
5          double height = 175.5, weight = 75.5;
6          System.out.println("成績 = " + grade);
7          System.out.println("身高 = " + height);
8          System.out.println("體重 = " + weight);
```

程式 Ch3_2_1a.java 的執行結果和 Ch3_2_1.java 完全相同。

4. byte、short、int、long、float、double、char 和 boolean 八種資料型態。

5. 下列說明文字的最佳變數資料型態，如下所示：

- 圓半徑。(float)
- 父親的年收入。(float)
- 個人電腦的價格。(float)
- 地球和月球之間的距離。(double)
- 年齡、體重。(int、float)

6. 習題 5.的變數的最佳變數名稱，如下所示：

- 圓半徑。(r)
- 父親的年收入。(salary)
- 個人電腦的價格。(price)
- 地球和月球之間的距離。(distance)
- 年齡、體重。(age、weight)

7. 左值、位址、右值、值

8. 下列 Java 程式碼的常數值，決定變數 a 到 e 宣告使用的資料型態，如下所示：

```
a= 'r';          => char
b= 100;       => int
c= 23.14;     => double
d= 453.13f;   => float
e= 453.13d;   => double
```

9.

27 / 5 = 5

27 / (float) 5 = 5.4

(float) 27/ (float) 5 = 5.4

10. Java 程式片段，如下所示：

```
int a = 200;
System.out.println("幾打= " + a/12);
System.out.println("剩幾個= " + a%12);
```

11. Java 程式片段，如下所示：

```
amount = 1500000*1.02*1.02*1.02*1.02*1.02
                    *1.02*1.02*1.02*1.02*1.02;
```

12. 參閱第 3-2-4 節的程式範例，公式改為 perimeter = 2 * PI * r;，然後將 r 依序設為 10, 20, 50

第 4 章：流程控制結構

1. 循序結構（Sequential）、選擇結構（Selection）和重複結構（Iteration）

2. Java 程式區塊是一種最簡單的結構敘述，一般來說，流程控制敘述都是使用程式區塊來控制流程的執行。程式區塊（Blocks）的目的是將零到多列的程式敘述組合成一個群組，以便將整個程式區塊視為一列程式敘述來處理，如下所示：

```
{
    ……
    程式敘述;
    ……
}
```

上述程式區塊使用「{」和「}」大括號包圍的一到多個程式敘述，事實上，在大括號之內也可以不包含任何程式敘述，稱為「空程式區塊」（Empty Block）。

程式區塊可以群組化程式碼的編排，因為 Java 語言並沒有限制宣告變數的位置，所以程式區塊還可以用來隱藏變數宣告，如下所示：

```
{
    int temp;
    temp = a; a = b; b = temp;
}
```

上述程式區塊宣告整數變數 temp，變數 temp 只能在程式區塊之內使用，一旦離開程式區塊，就無法存取變數 temp，變數 temp 稱為程式

區塊的區域變數（Local to the Block）。關於方法的區域變數說明請參閱＜第 5-3-2 節：Java 的變數範圍＞。

3. if/else/if、switch

4. for、while、do/while、break

5. Java 條件運算式的結果，如下所示：

true

true

true

false

true

6. Java 程式片段，如下所示：

```
if ( x >= 1 && x <= 40 )
      y = x;
else   y = 150;
```

7. Java 程式執行結果，如下所示：

```
2
4
8
16
32
64
128
```

8. Java 程式執行結果，如下所示：

```
1
3
5
7
9
20
```

9. Java 程式片段，如下所示：

```
for (i = 1; i <= 150; i++) {
    if ((i % 2) != 0 && i >= 50 && i <= 90) {
        total += i;
        System.out.println(i);
    }
}
System.out.print("總和: “+ total);
```

10. Java 程式片段，如下所示：

```
if ( amount >= 2000 ) amount *= 0.85;
System.out.println("付款金額: " + amount);
```

11. Java 程式片段，如下所示：

```
if ( weight <= 5 )
    fee = weight * 50 + 199;
else
    fee = 5*50+(weight-5)*30+199;
System.out.println("運費: " + fee);
```

12. 使用 for 迴圈計算下列數學運算式的值，如下所示：

- 1+1/2+1/3+1/4~+1/n n=67

```
for ( n = 1; n <= 67 ; n++ )
    result += 1/n;
```

- 1*1+2*2+3*3~+n*n n=34

```
for ( n = 1; n <= 34 ; n++ )
    result += n*n;
```

13. 使用 for 迴圈從 4 到 100 顯示 4 的倍數，例如：4、8、12、16、20、32…..，如下所示：

```
for ( n = 4; n <= 100 ; n=+4 )
```

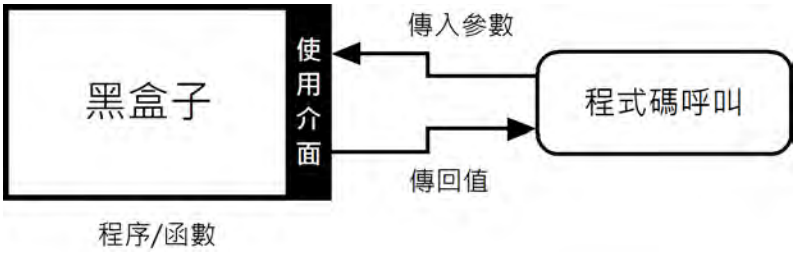
```
        System.out.println(n);
```

14. 使用 2 層 for 迴圈建立巢狀迴圈顯示九九乘法表，如下所示：

```
for ( i = 1; i <= 9; i++ ) {
    ……
    for ( j = 1; j <= 9; j++ ) {
        ……
    }
}
```

第 5 章：類別方法

1. 我們並不需要了解程序與函數實作的程式碼，也不想知道其細節，程序與函數如同是一個「黑盒子」（Black Box），只要告訴我們如何使用黑盒子的「使用介面」（Interface）即可，如下圖所示：



上述使用介面是呼叫程序或函數的對口單位，可以傳入參數和取得傳回值。簡單的說，使用介面是程序與函數對外的溝通管道，一個對外的邊界。程序與函數真正的內容是隱藏在使用介面之後，「實作」（Implementation）就是在撰寫程序與函數的程式碼。

一般來說，程序與函數的「語法」（Syntactic）說明程序與函數需要傳入何種資料型態的「參數」（Parameters）和傳回值，「語意」（Semantic）是描述程序與函數可以作什麼事？在撰寫程序與函數時，

我們需要了解程序與函數的語法規則；呼叫程序與函數時需要了解程序與函數的語意規則，如此才能夠正確的呼叫程序與函數。

2. return

3. void

static int sum(double c) { }

4. 在方法定義的參數稱為「正式參數」（Formal Parameters）或「假的參數」（Dummy Parameters），這些正式參數是識別字，其角色如同變數，需要指定資料型態，並且可以在方法的程式碼區塊中使用，如果參數不只一個請使用「,」符號分隔。

方法呼叫的參數稱為「實際參數」（Actual Parameters），這是參數的值，例如：1，也可以是運算式，例如：end 變數，其運算結果的值需要和正式參數定義的資料型態相同，方法的每一個正式參數需要對應一個同型態的實際參數。

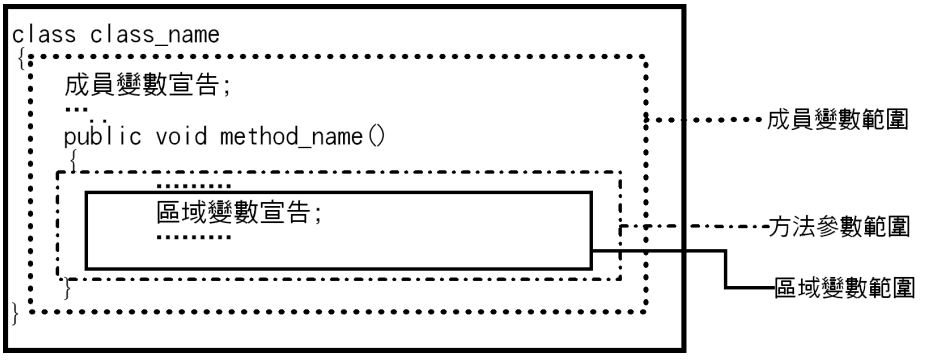
5. Java 方法傳入的參數有兩種不同的參數傳遞方式，如下表所示：

傳遞方式	**說明**
傳值呼叫（Call by Value）	將變數的值傳入方法，方法另外配置記憶體儲存參數值，所以並不會變更呼叫變數的值
傳址呼叫（Call by Reference）	將變數實際儲存的記憶體位置傳入，如果在方法變更參數值，也會同時變動原呼叫的變數值

6. Java 語言的變數分為類別的成員變數、「方法參數」（Method Parameters）和區域變數。「變數範圍」（Scope）影響變數值的存取，即有哪些程式碼可以存取此變數。Java 語言的變數範圍，如下所示：

- 區域變數範圍（Local Variable Scope）：在方法內宣告的變數，只能在宣告程式碼後的程式碼使用（不包括宣告前），在方法外的程式碼並無法存取此變數。
- 方法參數範圍（Method Parameter Scope）：傳入方法的參數變數範圍是整個方法的程式區塊，在方法外的程式碼並無法存取。
- 成員變數範圍（Member Variable Scope）：不論是 static 的類別變數或是沒有宣告 static（此為物件的實例變數，詳見第 10 章的說明），整個類別的程式碼都可以存取此變數。

接著筆者將上述各變數的範圍整理成圖形，如下圖所示：



7. Java 程式執行結果，如下所示：

```
<<<<<<$>>>>>>
```

8. Java 程式片段，如下所示：

```
no = (int)(Math.random()*50 + 1);
```

toRadians()方法

9. Java 方法如下所示：

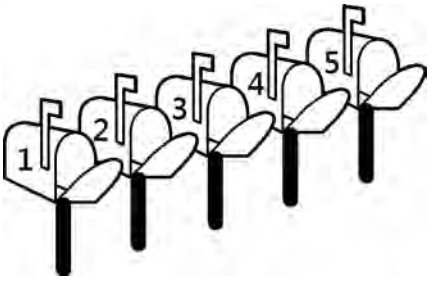
```
static int max(int a, int b) {
      if (a > b) return a;
      else return b;
}
static int min(int a, int b) {
      inf ( a>b) return b;
      else       return a;
}
```

10. Java 方法如下所示：

```
static int sum(int a) {
    if ( n == 1 )
       return 1;
    else
       return a + sum(a-1);
}
```

第 6 章：陣列與字串

1. 「陣列」（Arrays）是程式語言的一種基本資料結構，屬於一種循序性的資料結構，Java 陣列就是 Array 物件，一種參考資料型態。因此，Java 陣列變數值並不是陣列本身，而是指向陣列真正位址的參考。日常生活中最常見的陣列範例就是一排信箱，如下圖所示：



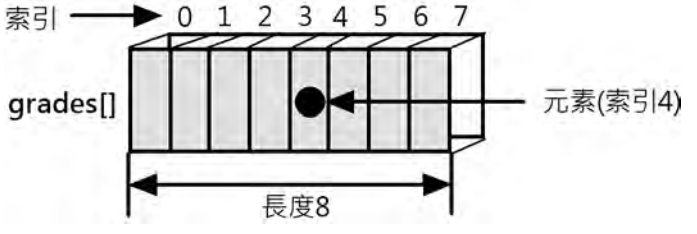
上述圖例是公寓或社區住家的一排信箱，郵差依信箱號碼投遞郵件，住戶依信箱號碼取出郵件，Java 陣列也是相同的道理。如果程式需要使用多個相同型態的變數時，我們可以宣告一堆變數。例如：班上 5 位學生的成績，如下所示：

```
int grade1, grade2, grade3, grade4, grade5;
```

上述程式碼宣告 5 個整數 int 變數 grade1~5，這是使用不同名稱來區分不同的學生。如果使用陣列變數，我們只需要宣告一個陣列變數，如下所示：

```
int[] grades = new int[5];
```

上述程式碼宣告一個一維陣列變數 grades，只需使用陣列變數名稱加上索引值，就可以存取指定陣列元素的值。Java 陣列就是將相同資料型態的變數集合起來，以一個名稱來代表，使用索引值存取元素，每一個元素相當於是一個變數，如下圖所示：



上述 grades[]陣列是一種固定長度的結構，陣列大小在編譯階段就已經決定，每一個「陣列元素」（Array Elements）可以使用「索引」（Index）存取，其索引值是從 0 開始到陣列長度減 1，即 0~7。

0

2. Array、String、length()

3. 將字串陣列連接成字串。

4. 建立 365x24 的二維陣列。

5.

```
float[] myArray = new float[10];
```

索引值序列：0、1、2、3

6. (1) 6　　(2) 23

7. 排序和搜尋都是在處理大量的資料，排序可以將資料依指定順序排列，搜尋則是從這些資料中，找出指定的值。

排序的基礎

排序工作是將一些資料依照特定原則排列成遞增或遞減的順序。例如：整數陣列 data[]的內容，如下所示：

```
data[0]=89 data[1]=34 data[2]=78 data[3]=45
```

上述陣列 data[]以整數大小將陣列元素依遞增順序進行排序，排序結果如下所示：

```
data[0]=34 data[1]=45 data[2]=78 data[3]=89
```

上述陣列 data[]已經排序，其大小順序如下所示：

```
data[0] < data[1] < data[2] < data[3]
```

搜尋的基礎

搜尋工作是在資料中找出是否存在與特定值相同的資料，搜尋的值稱為「鍵值」（Key），如果資料存在，就進行後續的資料處理。例如：查詢電話簿是為了找朋友的電話號碼，然後與他連絡。在書局找書也是為了找到後買回家閱讀。

搜尋方法依照搜尋的資料分為兩種，如下所示：

- 沒有排序的資料：針對沒有排序的資料執行搜尋，需要從資料的第 1 個元素開始比較，從頭到尾以確認資料是否存在。
- 已經排序的資料：搜尋就不需要從頭開始一個個的比較。例如：在電話簿找電話，相信沒有人是從電話簿的第一頁開始找，而是直接從姓名出現的頁數開始找，因為電話簿已經依照姓名排序好。

8. Java 方法如下所示：

```
static reverse(int[] arr) {
    int temp;
     for (int i = 0; i < arr.length/2; i++) {
         temp = arr[i];
         arr[i] = arr[arr.length - i ];
         arr[arr.length - i ] = temp;
     }
}
```

9. Java 程式片段，如下所示：

```
for ( a = 0; a < result.length; a++ )
    result[a] = arr1[a] + arr2[a]
```

10. 在 Java 程式片段，如下所示：

```
for ( i = 0; i < 3; i++ ) {
    for ( j = 0; j < 3; j++ ) {
        arr[i][j] = (int)(Math.random()*1000 + 1);;
        System.out.print(arr[i][j] + " ");
    }
    System.out.println();
}
for ( i = 0; i < 3; i++ ) {
    total = 0;
    for ( j = 0; j < 3; j++ )
        total += arr[i][j];
    System.out.println("Rows: " + i + "Total=" + total + "Average=" +total/3.0);
}
total = 0;
```

```
for ( i = 0; i < 3; i++ )
    for ( j = 0; j < 3; j++ )
        if ( i == j )
            total += arr[i][j];
System.out.println("對角線的和" + total);
```

11. Java 的 getMax()和 getMin()方法，如下所示：

```
public static int getMax(int data[]) {
    int i, max = 0;
    // 使用 for 迴圈找尋最大值
    for ( i = 0; i < data.length; i++ )
        if ( data[i] > max ) max = data[i];
    return max;
}
public static int getMin(int data[]) {
    int i, min = 500;
    // 使用 for 迴圈找尋最大值
    for ( i = 0; i < data.length; i++ )
        if ( min > data[i] ) min = data[i];
    return min;
}
```

12. 字元搜尋的二元搜尋法方法。

```
static boolean binary(char[] data,int l,int h,int k) {
    int m;                              // 宣告中間元素索引
    if ( l > h )                       // 遞迴的終止條件
        return false;                  // 沒有找到
    else {
        m = (l + h) / 2;               // 計算中間索引值
        if ( k == data[m] ) return true;      // 找到
        else if ( k < data[m] )// 前半部分
                return binary(data, l, m-1, k);
            else                       // 後半部分
                return binary(data, m+1, h, k);
    }
}
```

字元的泡沫排序法和線性搜尋方法。

```
static boolean sequential(char[] data,int k) {
    for ( int i = 0; i < data.length; i++ ) // 搜尋迴圈
```

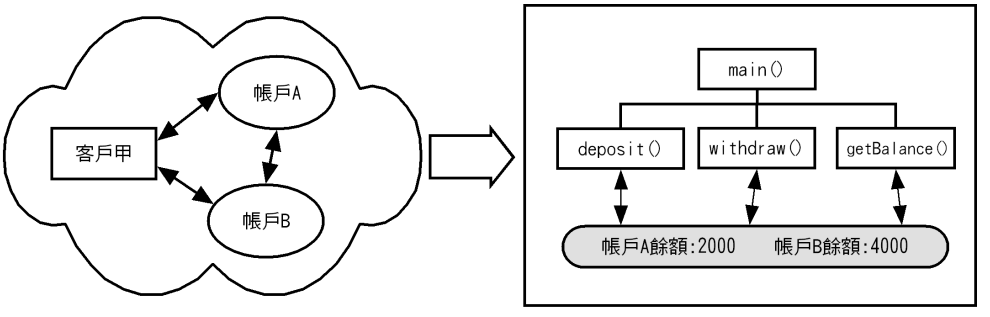
```
            // 比較是否是鍵值
            if ( data[i] == k ) return true;
        return false;
    static void bubbleSort(char[] data) {
        int i,j;                                    // 變數宣告
        int count = data.length;                    // 取得陣列尺寸
        char temp;
        for ( j = count; j > 1; j-- ) { // 第一層迴圈
            for ( i = 0; i < j - 1; i++ )// 第二層迴圈
                // 比較相鄰的陣列元素
                if ( data[i+1] < data[i] ) {
                    temp = data[i+1];           // 交換陣列元素
                    data[i+1] = data[i];
                    data[i] = temp;
                }
            // 顯示第一層迴圈執行後交換的字串
            System.out.print(count-j+1);
            System.out.println(": " + new String(data));
        }
    }
```

第 7 章：類別與物件

1. 物件導向的應用程式開發是一種思考程式問題上的革命，讓我們完全以不同於傳統應用程式開發的方式來思考問題。

傳統的應用程式開發

傳統的應用程式開發是將資料和操作分開來思考，著重於如何找出解決問題的程序或函數。例如：一家銀行的客戶甲擁有帳戶 A 和 B 共兩個帳戶，客戶甲在查詢帳戶 A 的餘額後，從帳戶 A 提出 1000 元，然後將 1000 元存入帳戶 B。傳統應用程式開發建立的模型，如下圖所示：



上述圖例的左邊是真實世界中，參與的物件和其關聯性，右邊是經過結構化分析和設計（Structured Analysis/Design）後建立的應用程式模型。

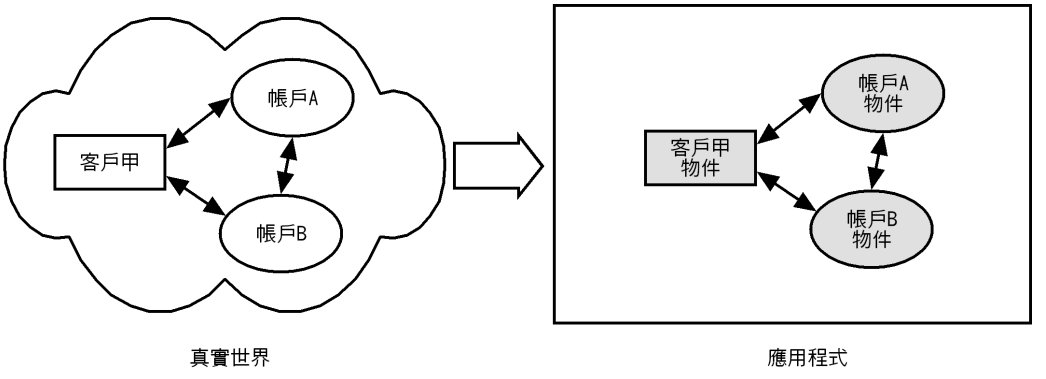
應用程式模型是解決問題所需的程序與函數，包含：存款的 deposit()函數、提款的 withdraw()函數和查詢餘額的 getBalance()函數。

在主程式 main()是一序列的函數呼叫，首先呼叫 getBalance()函數查詢帳戶 A 的餘額，參數是帳戶資料，然後呼叫 withdraw()函數從帳戶 A 提出 1000 元後，呼叫 desposit()函數將 1000 元存入帳戶 B。

物件導向的應用程式開發

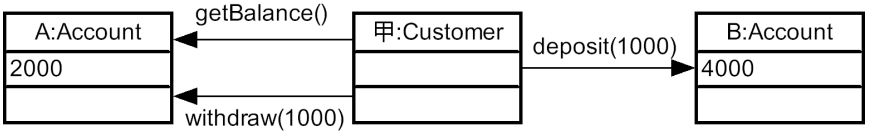
物件導向的應用程式開發是將資料和操作一起思考，其主要的工作是找出參與物件和物件之間的關聯性，並且透過這些物件的通力合作來解決問題。

例如：針對上一節相同的銀行存提款問題，使用物件導向應用程式開發所建立的模型，如下圖所示：



上述圖例是在電腦系統建立一個對應真實世界物件的模型，簡單的說，這是一個模擬真實世界的物件集合，稱為物件導向模型（Object-Oriented Model）。

物件導向應用程式開發因為是將資料和操作一起思考，所以帳戶物件除了餘額資料外，還包含處理帳戶餘額的相關方法：getBalance()、withdraw()和 deposit()方法，如下圖所示：



在上述圖例的客戶甲物件，先送出訊息給帳戶 A 物件，請求執行 getBalance()方法取得帳戶餘額 2000 元，然後再次送出訊息給帳戶 A 物件，執行 withdraw()方法提款 1000 元，所以目前餘額為 1000 元，最後送出訊息給帳戶 B 物件執行 deposit()方法存入 1000 元，帳戶 B 物件的餘額更新成 5000 元。

事實上，物件導向應用程式是物件集合，將合作物件視為節點，訊息為邊線連接成類似網路圖形的物件架構。在物件之間使用訊息進行溝

通，各物件維持自己的狀態（更新帳戶餘額），並且擁有獨一無二的物件識別（物件甲、A 和 B）。

2. 物件基礎程式語言（Object-based Languages）、物件導向程式語言（Object-oriented Languages）。物件導向程式語言（Object-oriented Languages）支援三種特性，如下所示：

封裝

封裝（Encapsulation）是將資料和處理資料的程序與函數組合起來建立物件。在 Java 語言定義物件是使用類別（Class），內含屬性和方法，屬於一種抽象資料型態，換句話說，就是替程式語言定義新的資料型態。

繼承

繼承（Inheritance）是物件的再利用，當定義好一個類別後，其他類別可以繼承這個類別的資料和方法，並且新增或取代繼承物件的資料和方法。

多型

多型（Polymorphism）屬於物件導向最複雜的特性，類別如果需要處理各種不同資料型態，並不需要針對不同的資料型態建立不同的類別，可以直接繼承基礎類別，繼承此類別建立同名方法處理不同的資料型態，因為方法的名稱相同，只是程式碼不同，所以也稱為「同名異式」。

Java 的類別可以用來建立物件，更正確的說是那些沒有宣告成 static 的部分，才是屬於物件的原型。它是一種使用者自行定義的資料型態。

如果使用類別原型建立物件，我們稱物件屬於類別的「實例」(Instance)，此時物件的變數稱為「實例變數」(Instance Variables)，而物件的程式或函數稱為「實例方法」(Instance Methods)，同一個類別可以當作範本建立無數個物件，而其中每一個物件都屬於類別的實例。

3. private、public

4. Java 的類別或實例方法都允許擁有 2 個以上的同名方法，同名方法只需傳遞參數個數或資料型態不同即可，稱為「過載」（Overload），或重載、多載。

因為物件導向程式設計的物件是依接收的訊息（Message）來執行不同的方法，換句話說，只需訊息有差異足以讓物件辨識，就算方法名稱相同，也一樣可以讓類別或物件執行正確的方法。

5. Java 語言的基本資料型態在宣告變數時會配置所需的記憶體，不過類別型態的變數並不會自動建立物件。如果類別沒有建構子，在使用 new 指令建立物件時，只會配置記憶體空間，並不會設定成員變數值。

如果希望類別如同基本資料型態，在宣告時指定變數初值，類別需要使用建構子，建構子是物件的初始方法，類別呼叫此方法來建立物件和指定初值。

類別的建構子

Java 類別的建構子擁有幾個特點，如下所示：

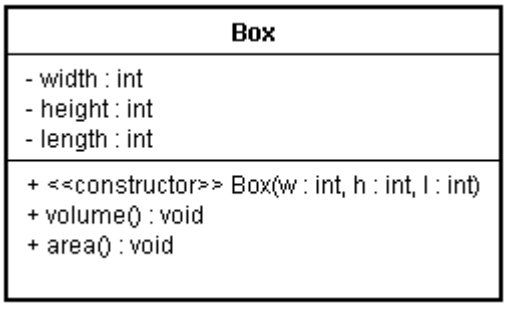
- 建構子與類別同名，例如：類別 Student 的建構子方法名稱為 Student()。
- 建構子沒有傳回值，也不用加上 void。
- 建構子支援方法的「過載」（Overload），建構子過載的詳細說明請參閱第 13-1 節。簡單的說，就是擁有多個同名的建構子，各建構子擁有不同的參數型態和個數。

6. 在類別宣告中宣告成 static 的類別變數和方法，在建立物件時，並不會替每一個物件實例建立類別的變數和方法，只有不是宣告成 static 部分才會建立獨立的實例變數和方法。

類別的變數和方法是屬於類別，並不是類別建立的物件，換句話說，所有的物件都是使用同一份類別變數和呼叫同一個類別方法。

7. 將程式 Ch7_3_3.java 的第 32 列 validAge()方法改為 static。

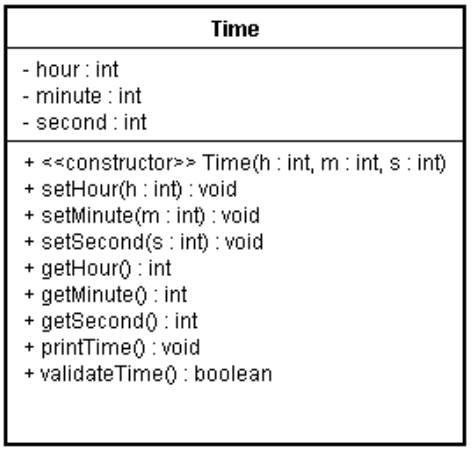
8. Box 類別宣告與類別圖，如下圖所示：




```
class Box {
    private int width;
    private int height;
    private int length;
```

```
    public Box(int w, int h, int l) {     }
    public void volume() {     }
    public void area() {     }
}
```

9. Time 類別宣告與類別圖，如下圖所示：




```
class Time {
    private int hour;
    private int minute;
    private int second;
    public Time(int h, int m, int s) {     }
    public void setHour(int h) {     }
    public void setMinute(int m) {     }
    public void setSecond(int s) {     }
    public int getHour() { return 0; }
    public int getMinute() { return 0; }
    public int getSecond() { return 0; }
    public void printTime() {     }
    public boolean validateTime() { return false; }
}
```

10. Counter 類別宣告，如下圖所示：

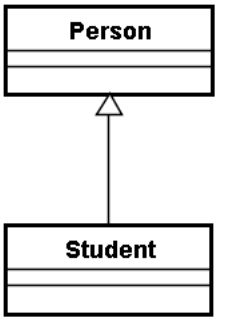
```
class Counter {
    public static int value;
```

```
    public Counter(int v) { vlaue = v;   }
    public void increment() { value++; }
    public void decrement() { value--;   }
    public int getCount() {   }
}
```

第 8 章：繼承、抽象類別與介面

1. 「繼承」（Inheritance）是物件導向程式設計的重要觀念，繼承是指宣告類別繼承現存類別的部分或全部的成員變數和方法、新增額外的成員變數和方法或覆寫和隱藏繼承類別的方法或變數。

類別的繼承關係可以讓我們建立類別架構。在 UML 類別關聯性中，繼承稱為一般關係（Generalization）。例如：類別 Student 是繼承自類別 Person，其類別架構如下圖所示：



上述 Person 類別是 Student 類別的父類別，反之 Student 類別是 Person 類別的子類別。UML 類別圖的繼承是使用空心的箭頭線來標示兩個類別之間的一般關係。

2. 兄弟類別、一般

3. 繼承的父類別方法如果不符合需求，在子類別可以宣告同名、同參數列和傳回值的方法來取代父類別繼承的方法，稱為「覆寫」（Override），或稱為覆蓋。

不過，物件的實例方法不能取代宣告成 static 的類別方法。如果父類別擁有類別方法，在子類別需要宣告同樣的類別方法來取代它，稱為「隱藏」（Hide）。

覆寫和隱藏的差異

覆寫和隱藏方法的差異，如下表所示：

	父類別的實例方法	**父類別的類別方法**
子類別的實例方法	覆寫	編譯錯誤
子類別的類別方法	編譯錯誤	隱藏

4. 類別方法、實例方法

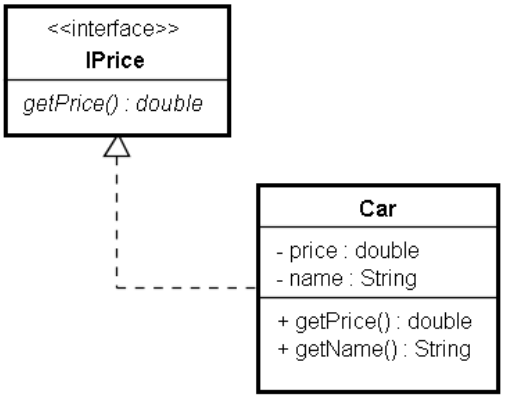
5. super、super

6. Java 介面（Interface）可以替類別的物件提供共同介面，就算類別間沒有任何關係（有關係也可以），一樣可以擁有共同介面。

如同網路通訊協定（Protocol）建立不同電腦網路系統間的溝通管道，不管 Windows 或 Unix 作業系統的電腦，只要說 TCP/IP 就可以建立連線。換句話說，介面是定義不同類別間的一致行為，也就是一些共同的方法。

例如：Car 和 CD 類別擁有共同方法 getPrice()取得價格，Java 程式可以將共同方法抽出成為介面 IPrice。如果 Book 類別也需要取得書價，就可以直接實作 IPrice 介面，反過來說，如果類別實作 IPrice 介面，就表示可以取得物件價格，這些類別都擁有相同行為：取得價格。

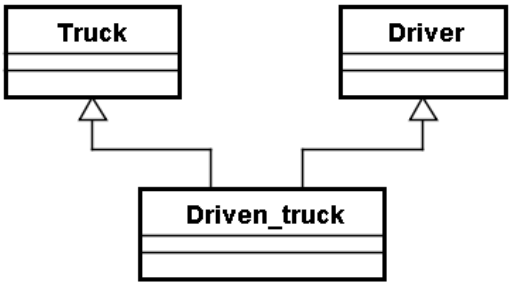
Java 介面也是使用 UML 類別圖，只是在類別名稱上方使用模板型態（Stereotype）<<interface>>指明此為介面，如下圖所示：



上述 IPrice 介面沒有屬性部分，而且介面方法前不需要存取修飾子的可見度。IPrice 提供介面給 Car 類別實作，這種類別與介面的關係稱為「實現化關係」（Realization）。

在 UML 類別圖的連接線類似一般關係的繼承，只是改用虛線連接 IPrice 介面和 Car 類別。

「多重繼承」（Multiple Inheritance）是指一個類別能夠繼承多個父類別，如下圖所示：



上述 Driven_truck 類別繼承自 Truck 和 Driver 兩個類別。對於 Driven_truck 類別來說，它擁有兩個父類別，這就是多重繼承。

Java 語言並不支援多重繼承，但是擁有相同目的的多重繼承 - 介面（Interface）。

7. print()、page()、footer()、header()共 4 個方法

8. 「常數類別」宣告成 final 表示這個類別不能被繼承，如果方法宣告成 final 表示此方法不可以覆寫，至於為什麼需要使用 final 修飾子，其理由如下所示：

- 保密的原因：基於保密的理由，可以將一些類別宣告成 final，以防止子類別的建立，存取或覆寫原類別的操作。
- 設計的原因：基於物件導向設計的需求，我們可以將某些類別宣告成 final，以避免子類別繼承。

Java 語言的類別如果使用 abstract 修飾子，表示這是一個「抽象類別」（Abstract Class），這個類別並不能建立物件，只能被繼承用來建立子類別。

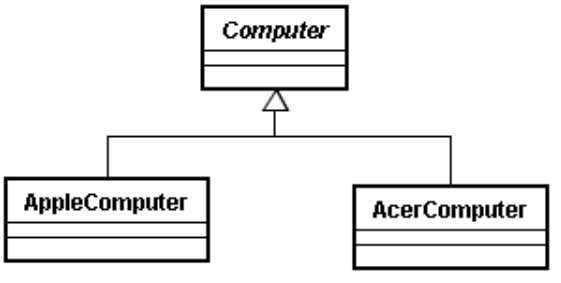
在抽象類別同時可以使用 abstract 宣告方法為抽象方法，表示這個方法只是一個原型宣告，實作的程式碼只能在子類別建立，而且繼承的類別一定要實作抽象方法。

9. 抽象類別和介面在架構上十分相似，抽象類別是讓其它類別繼承，介面可以讓類別實作方法，其主要的差異，如下所示：

- 抽象類別的方法可能只有宣告成抽象方法，但是仍然可以是一般方法，介面的方法只有宣告，一定不會有實作的程式碼。
- 介面不屬於類別的繼承架構，抽象類別一定屬於類別的繼承架構，而且一定是父類別，介面就算是毫無關係的類別一樣可以實作同一個介面。

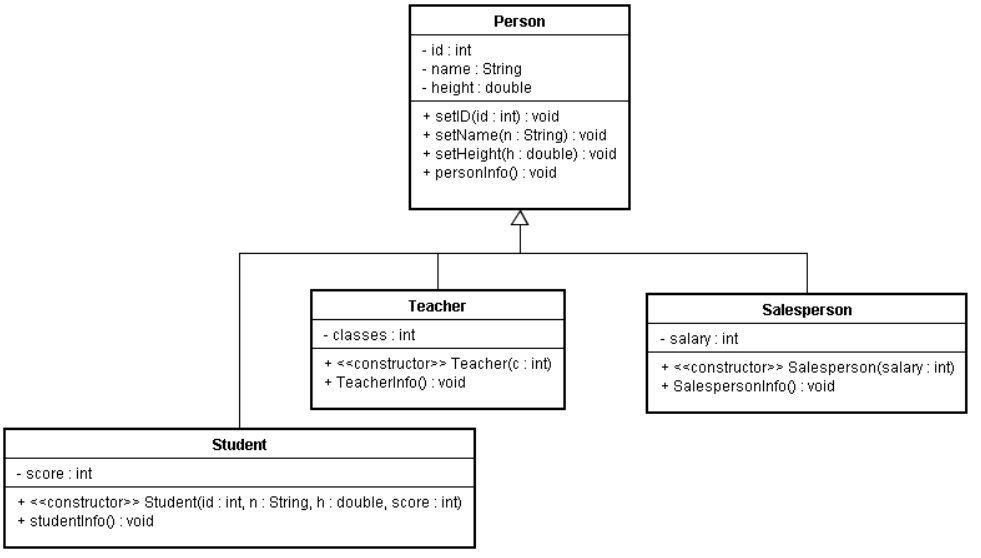
一個類別只能繼承一個抽象類別，但是可以同時實作多個介面。

10. Computer、AppleComputer 與 AcerComputer 類別架構，如下圖所示：



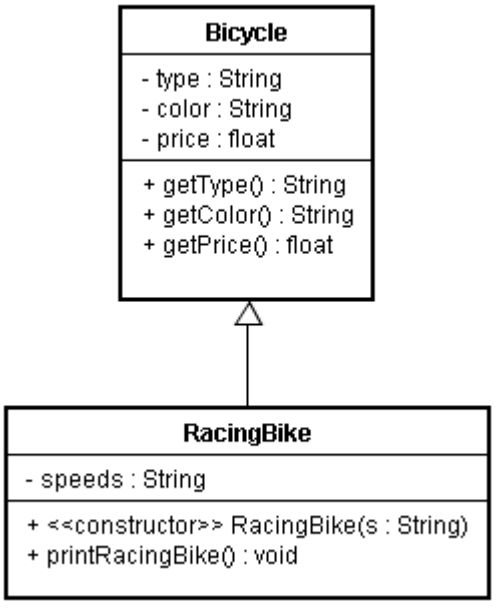
11. final、abstract

12. Teacher 與 Salesperson 類別宣告與類別圖，如下圖所示：




```
class Teacher extends Person {
    private int classes;
    public Teacher(int c) {     }
    public void TeacherInfo() {     }
}
class Salesperson extends Person {
    private int salary;
    public Salesperson(int salary) {     }
    public void SalespersonInfo() {     }
}
```

13. Bicycle 類別宣告與類別圖，如下圖所示：




```
class Bicycle {
    private String type;
    private String color;
    private float price;
    public String getType() { }
    public String getColor() { }
    public float getPrice() { }
}
class RacingBike extends Bicycle {
    private String speeds;
    public RacingBike(String s) { }
    public void printRacingBike() { }
}
```

14. IPrice 介面與 Car 類別宣告，如下所示：

```
interface IPrice {
     String getPrice();
}
```

```
class Car implements IPrice {
    public double price;
    public String name;
    public double getPrice() {   }
    public String getName() {   }
}
```

15. Person 抽象類別與 Student 類別宣告，如下所示：

```
abstract class Person {
    public String name;     // 姓名
    public int age;          // 年齡
    // 抽象方法: 計算總分或總價
    public abstract void total();
}
class Student extends Person {
    public String stdNo;
    public double test1, test2;
    // 建構子
    public Student(String no, String name, int age,
                        double t1, double t2) {
        stdNo = no;
        this.name = name;
        this.age = age;
        test1 = t1;
        test2 = t2;
    }
    // 成員方法: 實作抽象方法 total()
    public void total() { System.out.println("總分: " + (test1+test2)); }
}
```

16. CheckingAccount 類別宣告，如下所示：

```
class CheckingAccount extends Account {
    public boolean haveCheck;
    public CheckingAccount(String id, double amount,
                    double interest, boolean haveCheck) { … }
    public void calInterest() {
       double amount = getBalance();
        System.out.println("利息: " + (amount*interest));
    }
}
```

第 9 章：巢狀類別、多型與套件

1. 巢狀類別強調類別間的關聯性，強調內層類別一定需要外層類別，如果外層類別的物件不存在，內層類別物件也不會存在，內層的成員類別稱為「內層類別」（Inner Classes）。

巢狀類別的宣告

Order 巢狀類別的宣告，如下所示：

```
class Order {   // Order 外層類別
    ……
    class OrderStatus { // OrderStatus 內層類別
        ……
    }
    ……
}
```

上述 Order 類別擁有成員類別 OrderStatus 的內層類別，Order 是巢狀類別的「外層類別」（Enclosing Class）。

組成關係

2. Student.class、Student$Address.class

3. Java 內層類別如果沒有命名，稱為「匿名內層類別」（Anonymous Inner Classes），簡單的說，因為我們並沒有使用 class 關鍵字宣告類別名稱，所以稱為匿名，通常它是使用在 Java 的 Swing；Android 介面元件的事件處理，可以簡化複雜的事件處理程式碼。

匿名內層類別可以繼承現存類別，也可以實作介面來建立物件，其基本語法如下所示：

```
new 類別名稱( [ 參數列 ] ) { ... }
```

或：

```
new 介面名稱() { … }
```

上述實作介面的匿名內層類別隱含建立一個匿名類別（不是繼承的子類別）來實作介面。

4. 在匿名內層類別常常需要使用到 this 與 final 關鍵字，this 關鍵字可以取得匿名內層類別建立的物件本身；final 關鍵字可以取得匿名內層類別宣告外的變數值（不是類別的成員變數）。

匿名內層類別與 this 關鍵字

因為 this 關鍵字取得的是最接近類別的物件本身，換句話說，它只能存取到匿名內層類別建立的物件，如果需要存取外層類別，請加上類別名稱，如下所示：

```
class MyValue {
    private int v = 50;
    public IValue getIValueObject() {
        final int v2 = 10;
        IValue temp = new IValue() {
            private int v = 20;
            public int value() {
                int v = 30;
                return MyValue.this.v +
                            this.v + v + v2;
            }
        };
        return temp;
    }
}
```

上述 MyValue 類別擁有成員變數 v，匿名內層類別也擁有成員變數 v，取得外層類別的成員變數 v，需要使用 MyValue.this，如下所示：

```
return MyValue.this.v + this.v + v + v2;
```

上述 MyValue.this.v 是 MyValue 類別的成員變數 v，this.v 是匿名類別的成員變數 v。

匿名內層類別與 final 關鍵字

如果在內層匿名類別使用其所在方法的變數，必須宣告為 final。例如：在 value()方法使用 getIValueObject()方法宣告的區域變數 v2，如下所示：

```
....
public IValue getIValueObject() {
    final int v2 = 10;
    IValue temp = new IValue() {
        private int v = 20;
        public int value() {
            int v = 30;
            return MyValue.this.v +
                        this.v + v + v2;
        }
    };
    return temp;
}
```

上述區域變數 v2 是使用 final 關鍵字宣告，因為 v2 並不是直接拿到匿名內層類別中使用，而是在匿名內層類別中複製一份，作為成員變數來使用，因為是複本，所以不允許修改來影響到真正的區域變數 v2，所以需要宣告成 final，表示在匿名內層類別中不允許更改 v2 的值。

5. 多型是物件導向技術中最複雜的觀念，在這一節筆者準備使用類別繼承的方法覆寫來實作多型，也就是繼承抽象類別來建立多型。

抽象類別

Shape 抽象類別是建立多型所需的基礎類別，類別定義抽象方法 area()，其宣告如下所示：

```
abstract class Shape {
    public double x;
    public double y;
    public abstract void area();
}
```

建立多型

在 Java 程式可以繼承 Shape 抽象類別來建立 Circle（圓形）、Rectangle（長方形）和 Triangle（三角形）三個子類別來建立多型方法。其類別宣告如下所示：

```
class Circle extends Shape {
    ……
    public void area() { … }
}
class Rectangle extends Shape {
    ……
    public void area() { … }
}
class Triangle extends Shape {
    ……
    public void area() { … }
}
```

上述三個子類別都實作抽象方法 area()，只是內含程式碼不同，可以分別計算不同圖形的面積。現在我們可以使用抽象類別 Shape 宣告物件變數 s，如下所示：

```
Shape s;
```

上述物件變數 s 能夠參考 Circle、Rectangle 和 Triangle 物件，換句話說，物件變數 s 也可以呼叫物件的 area()方法，如下所示：

```
s.area();
```

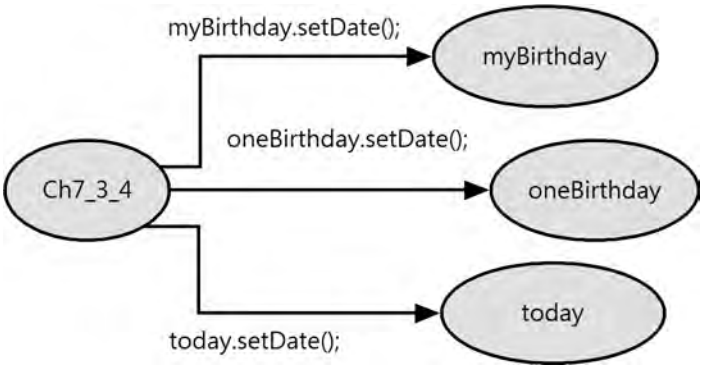
上述呼叫會依照物件變數 s 參考的物件來呼叫正確的方法。例如：如果 s 參考 Rectangle 物件，就會呼叫 Rectangle 物件的 area()方法，這個 area()方法就是多型。多型方法是直到執行階段，才會依照實際參考的物件來執行正確的方法。

6. 物件導向的過載與多型機制是架構在訊息和物件的「靜態連結」（Static Binding）與「動態連結」（Dynamic Binding）之上。

靜態連結

靜態連結（Static Binding）的訊息是在編譯階段，就決定其送往的目標物件。例如：類別的過載方法是在編譯時就建立訊息和物件的連結，也稱為「早期連結」（Early Binding）。

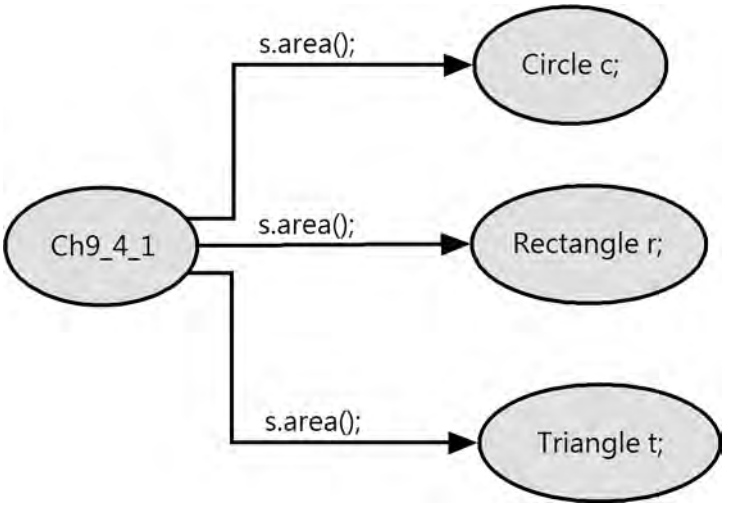
在第 7-3-4 節筆者說明的是靜態連結的過載方法，Ch7_3_4.java 程式的訊息是在編譯時就決定送達的目標物件是 myBirthday、oneBirthday 和 today，如下圖所示：



動態連結

動態連結（Dynamic Binding）的訊息是直到執行階段，才知道訊息送往的目標物件，這就是多型擁有彈性的主要原因，也稱為「延遲連結」（Late Binding）。

物件呼叫一個方法就是送一個訊息給物件，告訴物件需要執行什麼方法，現在 s.area()將訊息送到 s 物件變數參考的物件，如下圖所示：



上述 Java 類別 Ch9_4_1 的主程式 main()方法送出 3 個同名的 s.area()訊息（s 即 Shape 型態），這是實作的程式碼，等到動態連結在執行階段送出訊息時，才會依照物件變數參考的物件來送出訊息，所以實際送出的是下列 3 個訊息，如下所示：

```
c.area();
r.area();
t.area();
```

上述訊息可以送到指定物件 c、r 和 t，執行該物件的 area()方法，換句話說，我們並不用修改 Java 程式碼，只需新增繼承的子類別，再加上覆寫同名的方法，就可以馬上支援一種新圖形。

7. 多型是物件導向程式設計的重要觀念，Java 實作多型有三種方式，如下所示：

- 方法過載（Method Overloading）：方法過載也屬於多型，屬於一種靜態連結的多型。
- 類別繼承的方法覆寫（Method overriding through Inheritance）：繼承基礎類別覆寫同名方法或實作同名的抽象方法，就可以處理不同資料型態的物件。如果有新類別，也只需新增繼承的子類別和建立方法。
- Java 介面的方法覆寫（Method overriding through the Java Interface）：Java 介面是指同一個物件擁有多種型態，換個角度，不同物件也可以擁有相同的介面型態，所以一樣可以透過 Java 的介面來實作多型。

8. 類別繼承、Java 介面

9.

```
abstract class Vehicle {
    public abstract void show();
}
class Car extends Vehicle {
    ……
    public void show() { … }
}
class Trucks extends Vehicle {
    ……
    public void show() { … }
}
class Motocycle extends Vehicle {
    ……
    public void show() { … }
```

```
}
```

10.

```
interface IPrint {
    void print();
}
class Customer implements IPrint {
    ……
    public void print() { … }
}
class Student implements IPrint {
    ……
    public void print() { … }
}
class Teacher implements IPrint {
    ……
    public void print() { … }
}
class Sales implements IPrint {
    ……
    public void print() { … }
}
```

11.

```
class Polygon extends Shape {
    ……
    public void area() { … }
}
class Square extends Shape {
    ……
    public void area() { … }
}
```

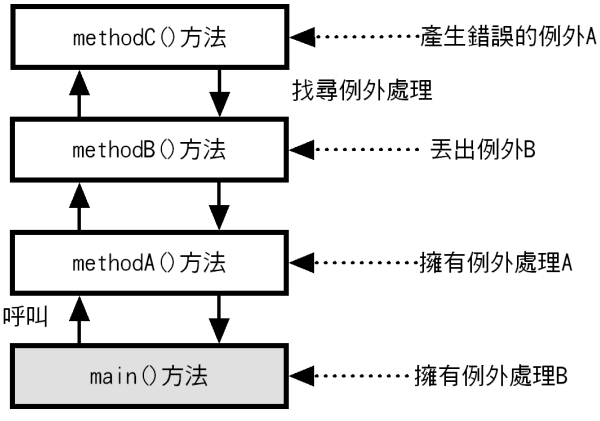
12.

```
abstract class Test {
    int score[10];
    public abstract void show();
}
```

```
class MidTeam extends Test {
    ……
    public void show() {    }
}
class Final extends Test {
    ……
    public void show() {    }
}
class Quiz extends Test {
    ……
    public void show() {    }
}
```

第 10 章：例外處理、執行緒、集合物件與 Lambda 運算式

1. Java 的例外處理架構是一種你丟我撿的架構，當 JVM 執行 Java 程式有錯誤發生時，就會產生例外物件。有了例外，JVM 開始尋找是否有方法可以處理，處理方法有兩種：一種是在方法上加上例外處理程式敘述來處理例外，或是將例外丟給其他方法處理，如下圖所示：



上述圖例是執行 Java 程式的呼叫過程，依序從 main()方法呼叫 methodA()方法，接著呼叫 methodB()方法，最後呼叫 methodC()方法。

呼叫方法的過程是存入稱為「呼叫堆疊」（Call Stack）資料結構來儲存呼叫方法的資料，以便返回時，能夠還原成呼叫前的狀態。

假設：methodC()方法發生錯誤，產生例外物件 A。JVM 就會倒過來找尋方法是否擁有例外處理，首先是 methodC()和 methodB()，因為沒有例外處理，所以例外會傳遞給 methodA()，在此方法擁有例外處理 A，所以可以處理例外物件 A。

不只如此，Java 方法也可以自行丟出例外，例如：methodB()丟出例外物件 B，同樣需要找尋是否有例外處理的方法可以進行處理。以此例雖然 methodB()擁有例外處理，但是因為例外類型不同，所以，直到 main()方法才找到正確的例外處理 B。

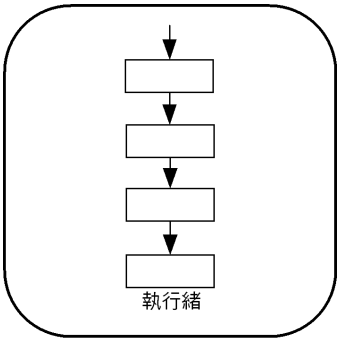
2. 雖然例外也是一種錯誤情況，只是這種錯誤沒有嚴重到需要終止程式的執行，我們可以使用例外處理防止程式執行終止，並且作一些補救。

例外處理的目的是為了讓程式能夠更加「強壯」(Robust)，如此這算程式遇到不尋常的情況，也不會造成程式「崩潰」(Crashing)，甚至導致整個系統當機的情況。

3. try、catch、finally、try、catch

4. ArithmeticException、ArrayStoreException 和 IllegalArgumentException 物件。

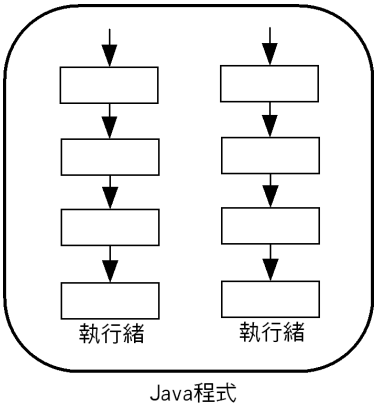
5. Java 的「執行緒」（Threads）也稱為「輕量行程」（Lightweight Process），其執行過程類似上述傳統程式執行，不過執行緒並不能單獨存在或獨立執行，一定需要隸屬於一個程式，由程式來啟動執行緒，如下圖所示：



上述圖例的 Java 程式產生一個執行緒在程式中執行，我們可以將它視為是包含在 Java 程式中的小程式。

換句話說，如果程式碼本身沒有先後依存的關係。例如：因為 b()方法需要使用到 a()方法的執行結果，需要在執行完 a()方法後，才能執行 b()方法，所以 a()方法和 b()方法並不能同時執行，也就無法使用 2 個執行緒來同步執行。

若程式能夠分割成多個同步執行緒來一起執行，這種程式設計方法稱為「平行程式設計」（Parallel Programming），如下圖所示：



上述圖例的 Java 程式擁有 2 個執行緒且是同步執行，也就是說，在同一個 Java 程式擁有多個執行流程，以同時執行多個執行緒來增加程式的執行效率。

6. Java 執行緒是在建立 Thread 類別的物件，一共有兩種方式建立多執行緒的應用程式，如下所示：

- 實作 Runnable 介面。
- 繼承 Thread 類別。

Rummable 介面、start()方法

7.

```
1
2
3
4
5
6
7
8
9
10
……..
40
41
42
43
44
45
46
47
48
49
……..
```

```
97
98
99
4950
```

8. 集合物件可以視為是一個容器，其儲存的元素是一個個物件，可以讓我們儲存不定數量的物件，簡單的說，集合物件就是一個物件集合。

Java 支援多種泛型集合類別，在這一節筆者準備說明常用的 HashSet 和 ArrayList 類別，不只如此，集合物件還提供 Iterator 和 ListIterator 介面的一致走訪方式來輸出集合物件的元素。

「泛型」（Generic）提供類似 C++語言的「樣版」（Templates）語法，新增編譯階段的集合物件型態檢查功能，可以減少程式碼中，取出集合物件元素所需的大量型態轉換。

集合物件的型態轉換

舊版 Java 集合物件在建立 HashSet 物件時，因為存入的是 Object 物件，所以可以存入 Byte、Float 和 String 等各種不同資料型態的元素。

不過，當從集合元素取出元素時，程式設計者需要自行記得存入的元素是哪一種資料型態，然後將 Object 物件型態轉換成正確的資料型態，如下所示：

```
Customer temp;
Iterator iterator = hset.iterator();
temp = (Customer) iterator.next();
```

上述程式碼是舊版 Java 語法，在使用 Iterator 介面取出 HashSet 集合物件 hset 的元素後，需要自行執行型態轉換成 Customer 類別的物件，才能指定給 temp 物件變數。

泛型型態

Java 可以使用「泛型型態」（Generic Types）來擴充 Java 語言。在建立集合物件時，使用泛型型態來指定集合物件儲存元素的資料型態，如下所示：

```
HashSet<Byte> hset0= new HashSet<Byte>();
HashSet<Float> hset1= new HashSet<Float>();
HashSet<String> hset2= new HashSet<String>();
```

上述程式碼在 HashSet 類別之後，使用「<」和「>」括起的資料型態就是泛型型態，可以指定集合物件儲存元素的資料型態，以便 Java 編譯程式自行追蹤記錄元素的資料型態，如下所示：

```
Customer temp;
Iterator<Customer> iterator = hset.iterator();
temp = iterator.next();
```

上述程式碼使用泛型型態，所以取出集合物件的元素時，就不需使用程式碼來執行型態轉換。

9. Lambda 運算式是一種匿名方法的運算式寫法（也可以擁有參數），可以用來當作方法的參數傳遞，或方法呼叫的傳回值。因為 Java 語言是一種徹頭徹尾的物件導向程式語言，除了一些基本資料型態之外，所有東西都是物件，並不允許單獨存在的方法，所以沒有辦法在方法的參數或傳回值直接使用方法，只能使用物件來取代。

Lambda 運算式的範例

當我們建立 Student 物件 Array 陣列物件後，如果希望使用 Student 物件的姓名 name 屬性進行排序，此時，我們需要建立 compare()方法進行姓名字串的比較，才能進行排序。但是，我們並不能直接使用方法，取而代之的是先建立匿名物件後才能建立方法，如下所示：

```
// 物件陣列的排序方法, 使用傳統 Comparator 物件
Arrays.sort(std1, new Comparator<Student>() {
    // 覆寫 compare()方法
    public int compare(Student first, Student second) {
        return first.getName().compareTo(second.getName());
    }
});
```

上述程式碼建立匿名 Comparator 物件後，覆寫 compare()方法來進行 Student 物件的姓名字串比較。Java 直接使用 Lambda 運算式來簡化程式碼，如下所示：

```
// 物件陣列的排序方法, 使用 Lambda 運算式
Arrays.sort(std2,
  (first,second) -> first.getName().compareTo(second.getName()));
```

上述方法的第 2 個參數就是一個 Lambda 運算式。事實上，Lambda 運算式就是在 Java 語言加入「函數程式設計」（Functional Programming）功能，讓方法（即函數）可以單獨存在。

舊版 Java 語言並不允許單獨存在的方法，Lambda 運算式是補足 Java 語言的缺失，因為近年來函數程式設計的重要性愈加重要，它特別適用在並行處理和事件驅動程式設計，這正是現代程式語言非常重要的部分。

Lambda 運算式的基本語法

Lambda 運算式允許我們在需要使用方法的地方，馬上建立一個匿名方法，特別是哪些只會使用 1 次的地方，如此，就不需要先宣告類別來建立所需的方法。Lambda 運算式的基本語法，如下所示：

```
(參數) -> 運算式或程式區塊{  }
```

上述語法的「->」左邊指定參數（如果有的話），右邊是運算式或程式區塊。例如：將數學函數 f(x) = x * 2 寫成 Lambda 運算式，如下所示：

```
(x) -> x * 2;     // Lambda 運算式
```

上述程式碼的「->」左邊是參數 x；右邊是運算式 x * 2。一些 Lambda 運算式的範例，如下所示：

```
(int a, int b) -> { return a + b; }
(String s) -> { System.out.printls(s); }
```

上述 2 個 Lambda 運算式分別有 2 個和 1 個參數，並且加上參數的型態，然後在右邊是程式區塊，可以傳回運算結果和顯示字串內容。一些沒有參數的 Lambda 運算式範例，如下所示：

```
() -> System.out.println("陳會安");
() -> 54
() -> { return 3.1415926; }
```

10. 函數介面（Functional Interface）是一種只有宣告單一虛擬方法的 Java 介面，例如：java.lang.Runnable 介面是函數介面，因為只有 1 個 run()方法；AWT 的 ActionListener 介面也是函數介面，只有 1 個 actionPerformed()方法。

每一個 Lambda 運算式隱含指定給一個函數介面。例如：建立 Runnable 介面參考來指定成 Lambda 運算式，如下所示：

```
// 建立 Runnable 介面參考指定 Lambda 運算式
Runnable r = () -> System.out.println("大家好!");
```

上述程式碼指定函數介面 Runnable。另一種情況是沒有指明函數介面，如下所示：

```
new Thread(
    () -> System.out.println("大家好!")     // Lambda 運算式
).start();
```

上述程式碼因為 Thread()建構子參數是 Runnable 介面，所以自動型態轉換成 Runnable 介面。

在 Java 7 之前版本的介面只有方法宣告，並沒有實作程式碼，其主要原因是為了避免多重繼承產生混淆（因為不知方法是從那 1 個父介面而來），加上介面與實作此介面的類別擁有緊密關係，在介面新增方法，所有實作此介面的類別都需要實作此方法。

為了與舊版相容，避免存在函數庫因為支援 Lambda 運算式新增介面方法，而需要所有實作此介面的類別都實作方法，所以新增「預設方法」（Default Methods）讓介面宣告的方法也可以有實作的程式碼。

11.

```
static double test(double a)
                    throws IllegalArgumentException
{
    ……
    throw new IllegalArgumentException("……");
}
// 主程式
public static void main(String[] args) {
    try {   test(a);    }
    catch( IllegalArgumentException e ) {
        // 處理 ArithmeticException 例外
        System.out.println("例外說明: " + e.getMessage());
        System.out.print("例外原因: ");
        e.printStackTrace();
    }
}
```

12.

```
static double eval(double a, double b) throws ArithmeticException {
    ……
```

```
        throw new ArithmeticException("…….");
    }
    // 主程式
    public static void main(String[] args) {
        try {       }
        catch( ArithmeticException e ) {
            // 處理 ArithmeticException 例外
            System.out.println("例外說明: " + e.getMessage());
            System.out.print("例外原因: ");
            e.printStackTrace();
        }
    }
```

13. 請參閱 Ch10_2_2.java 的程式架構，加上下列條件來丟出例外，如下所示：

```
        if (arg < 0 ) throw new IllegalArgumentException("…….");
        if (arg > 100 ) throw new ArithmeticException("…….");
```

14. Stack 類別宣告，如下所示：

```
class Stacks {
    // 建立 LinkedList 物件
    private LinkedList llist = new LinkedList();
    // 新增堆疊元素
    public void push(Object obj) {
        llist.addFirst(obj);
    }
    // 取出堆疊元素
    public Object pop() {
        Object obj = llist.getFirst();
        llist.removeFirst();
        return obj;
    }
    public boolean isEmpty() {
        return (llist.size()>0 ? false : true);
    }
}
```

第 11 章：Android 與 XML 的基礎

1. XML 和 HTML 文件的結構都是使用標籤和屬性來建立內容。不過，XML 結構比 HTML 網頁更多樣化，因為 XML 並沒有預定標籤，我們需要自行定義標籤和文件結構。

什麼是 XML

「XML」（Extensible Markup Language）可擴展標示語言也是一種標籤語言，XML 1.0 版規格在 1998 年 2 月正式推出，其寫法十分類似 HTML，繼承 SGML 自定標籤的優點，並且刪除一些 SGML 複雜的部分，在功能上能夠補足 HTML 標籤的不足，而且擁有更多的擴充性。

不過，XML 並不是用來編排內容，而是用來描述資料，因此，XML 沒有 HTML 的預設標籤，使用者需要自行定義描述資料所需的各種標籤。例如：在 XML 文件使用自訂標籤定義電腦圖書的資料，如下所示：

```
01: <?xml version="1.0" encoding="utf-8"?>
02: <!-- 程式語言學習系列 -->
03: <library>
04:     <book>
05:         <id>J123</id>
06:         <title>Java SE 與 Android 程式設計範例教本</title>
07:         <author>陳會安</author>
08:         <price>650</price>
09:     </book>
10:     <book>
11:         <id>V222</id>
12:         <title>用實例學 Visual Basic 程式設計</title>
13:         <author>陳會安</author>
14:         <price>600</price>
15:     </book>
16: </library>
```

上述 XML 文件的基本架構可以分為幾個部分，其說明如下所示：

- 文件宣告：第 1 列是 XML 文件宣告，定義 XML 文件的版本和使用的編碼（即字元集），以此例為 1.0 版，使用 utf-8 字元集。
- 根標籤：第 3 列和第 16 列是 XML 文件根元素的開始標籤<library>和結尾標籤</library>。
- 子元素：在第 4~15 列是根元素的兩個子元素 book，而第 5~8 列和第 11~14 列是 book 元素的子元素 id、title、author 和 price。

XML 的命名空間

XML 命名空間（Namespace）是用來解決與其他公司進行 XML 資料交換時，避免標籤或屬性名稱重複的問題。

XML 標籤的完整名稱

XML 標籤和屬性的完整名稱需要加上命名空間字首，如下所示：

```
<命名空間字首:標籤名稱>
```

上述標籤在「:」符號前是命名空間「字首」（Prefix），可以視為是代替命名空間字串「URI」（Uniform Resource Identifier）的縮寫。

位在「:」符號後才是命名的標籤名稱。所以，同一份 XML 文件，就算之後標籤名稱相同，只要命名空間不同，XML 剖析器一樣可以分辨是不同的標籤，因為擁有不同命名空間字首 bk 和 jb，如下所示：

```
<bk:title>
<jb:title>
```

在 Android 的 XML 檔使用的命名空間

在 Android 專案定義資源是使用 XML 文件檔案，其根元素需要宣告命名空間來識別這是 Android 系統的屬性，例如：宣告使用介面的 XML 檔案內容，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    …
</android.support.constraint.ConstraintLayout>
```

上述 ConstraintLayout 是 XML 文件的根元素，使用 xmlns 屬性宣告 2 個命名空間，第 1 個是 android，如下所示：

```
xmlns:android="http://schemas.android.com/apk/res/android"
```

上述「:」符號後是命名空間字首 android，位在「=」等號後是 URI 字串，用來識別抽象或實際資源，例如：URL 網址，在之後屬性加上此字首就表示屬於此命名空間，例如：android:id 等。

2. XML 文件是由標籤和內容組成，其組成元素有：元素、屬性、實體參考、註解等。

元素與屬性

XML 文件是由元素（Elements）組成，XML 元素和標籤不同，其意義如下所示：

- 標籤（Tags）：XML 能夠自己定義標籤，一個標籤是用來標示文件的部分內容。例如：標籤<id>、<title>和<price>等，標籤分為開始標籤<id>和結尾標籤</id>。

- 元素（Elements）：XML 元素是整個文件的主要架構，元素是標籤加上其中的文字內容，或是在元素內包含其他元素，元素是一個完整項目，包含開始標籤、屬性、標籤內的文字內容和結尾標籤。

XML 元素需要由開始和結尾標籤構成，在其中包含文字內容，例如：XML 元素 id，內含值 J123，如下所示：

```
<id>J123</id>
```

除了文字內容外，XML 元素還可以包含其他子元素。例如：在圖書 book 元素擁有 id、title、author 和 price 子元素，如下所示：

```
<book>
  <id>J123</id>
  <title>Java SE 7 與 Android 程式設計範例教本</title>
  <author>陳會安</author>
  <price>650</price>
</book>
```

在 XML 開始標籤的標籤名稱後，可以新增空白字元分隔的屬性（Attributes）清單，每一個屬性擁有屬性名稱和值，屬性值需要使用雙引號括起，如下所示：

```
<book id="J123">
```

上述<book>標籤擁有 id 屬性，其值為 J123。如果在開始和結尾標籤之中沒有任何內容，稱為空元素標籤，其寫法有兩種，如下所示：

```
<Button/>
<Button></Button>
```

上述兩種寫法都是空元素標籤，如果空元素標籤沒有結尾標籤，標籤需要使用「/>」符號結束。

實體參考

XML 標籤語言本身擁有一些保留符號。例如：標籤中的「<」符號，如果文件內容需要使用這些符號，請使用實體參考（Entity Reference）。在 XML 提供 5 個預先保留的實體參考，如下表所示：

實體參考	符號
&lt;	<
&gt;	>
&amp;	&
&apos;	'
&quot;	"

在上表的每一個實體參考都是由「&」符號開始，以「;」結束，例如：在 XML 元素的文字內容不允許「<」符號，如下所示：

```
<order>書價 < 650</order>
```

上述標籤內容有「<」符號，此時就需要使用實體參考，如下所示：

```
<order>書價 &lt; 650</order>
```

註解

XML 的註解（Comment）和 HTML 一樣都是由"<!--"符號開始和"-->"符號結尾，如下所示：

```
<!-- 程式語言學習系列 -->
```

3.

Android 是一套使用 Linux 作業系統為基礎開發的開放原始碼（Open Source）作業系統，最初主要是針對手機等行動裝置使用的作業系統，現在 Android 已經逐漸擴充到平板電腦、筆電和其他領域，例如：電子書閱讀器、MP4 播放器和 Internet 電視等。

Android 作業系統最初是 Andy Rubin 創辦的同名公司 Android, Inc 開發的行動裝置作業系統，在 2005 年 7 月 Google 收購此公司，之後 Google 拉攏多家通訊系統廠商、硬體製造商等在 2007 年 11 月 5 日組成「開放式手持裝置聯盟」（Open Handset Alliance），讓 Android 正式成為一套開放原始碼的作業系統。

換句話說，目前擁有 Android 作業系統的是非營利組織的開放式手持裝置聯盟，Google 公司則在幕後全力支援 Android 作業系統的開發計劃，並且在 Android 作業系統整合 Google 的 Gmail、Youtube、Google 地圖和 Android Market 等服務，作為主要的獲利來源。

在 2010 年 1 月 5 日 Google 正式販售自有品牌的智慧型手機 Nexus One，到了 2010 年末僅僅推出兩年的 Android 作業系統，已經快速成長且超越稱霸十數年的諾基亞 Symbian 系統，躍居成為世界最受歡迎的智慧手機平台，在 2011 年初更針對平板電腦推出專屬的 3.x 版，而且快速成為最廣泛使用的平板電腦作業系統之一。

在 2011 年 10 月 19 日推出 4.0 版 Ice Cream Sandwich（冰淇淋三明治），一套整合手機和平板電腦 2.x 和 3.x 版本的全新作業系統平台，換句話說，之後的 Android 就只有一個版本，不再區分手機和平板電腦兩種專屬版本。

對於程式開發者來說，Android 提供完整開發工具和框架，可以讓開發者快速建立行動裝置執行的應用程式，其專屬開發工具 Android

SDK 更提供模擬器來模擬行動裝置，換句話說，就算沒有實體行動裝置，我們也一樣可以進行 Android 應用程式的開發。

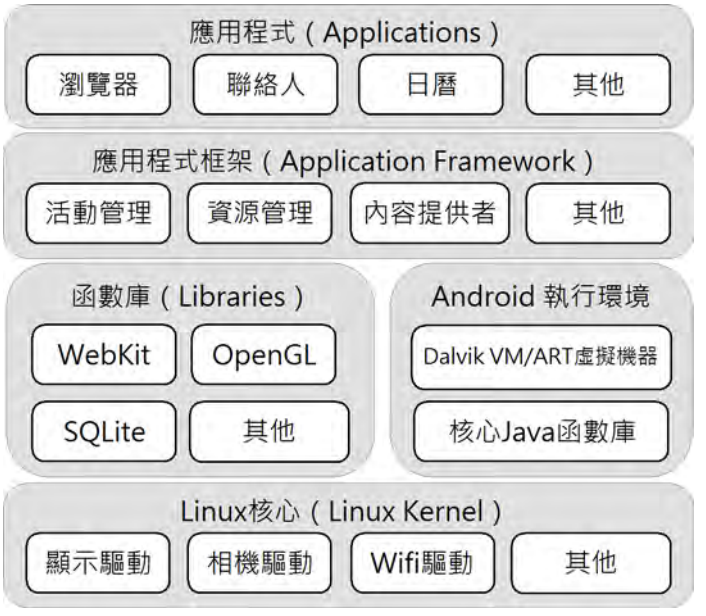
Android 的特點

Android 行動作業系統原則上沒有固定搭配的硬體配備或內建軟體，可以讓使用 Android 系統的製造廠商自行客製化行動裝置，依成本、市場定位和功能來搭配所需軟硬體，其特點如下所示：

- 硬體：支援數位相機、GPS、數位羅盤、加速感測器、重力感測器、趨近感測器、陀螺儀、NFC 近場通訊、指紋辨識和環境光線感測器等（並不是每一種行動裝置都具備完整硬體支援，可能只有其中幾項）。
- 通訊與網路：支援 GSM/EDGE、IDEN、GPRS、CDMA、EV-DO、UMTS、藍牙、WiFi、LTE 和 WiMAX 等。
- 簡訊：支援 SMS 和 MMS 簡訊。
- 瀏覽器：整合開放原始碼 WebKit 瀏覽器，支援 Chrome 的 JavaScript 引擎。
- 多媒體：支援常用音效、視訊和圖形格式，包含 MPEG4、H.264、AMR、AAC、MP3、MIDI、Ogg Vorbis、WAV、JPEG、PNG、GIF 和 BMP 等。
- 資料儲存：支援 SQLite 資料庫，一種輕量化的關聯式資料庫。
- 繪圖：支援 2D 函數庫的最佳化繪圖，和 3D 繪圖 OpenGL ES 規格。

- 其他：支援多點觸控、Flash、多工和可攜式無線基地台等。

4. Android 作業系統的系統架構像是在蛋糕店購買的一個多層蛋糕，在行動裝置的硬體和使用者之間是 Android 軟體堆疊（Android Software Stack），如下圖所示：



上述 Android 軟體堆疊可以分成很多層，各層的說明如下所示：

應用程式

使用者在 Android 作業系統執行的是應用程式 App，這也是使用者真正面對的 Android 作業系統，基本上，Android 預設就內建有多種核心應用程式，包含：瀏覽器、日曆、聯絡人和打電話等。

應用程式框架

位在應用程式之下的是應用程式框架，這一層提供高階建構元件，即一組類別集合來幫助我們建立 Android 應用程式，事實上，應用程式框架代表 Android 作業系統已經實作完整的 Java 類別，我們只需呼叫適當物件的方法，或繼承指定類別來覆寫或擴充其功能，就可以建立 Android 應用程式。

函數庫

位在應用程式框架之下的是函數庫，負責支援應用程式框架各元件的執行，這是一些使用 C/C++語言撰寫的函數庫，包含：瀏覽器引擎的 WebKit、3D 繪圖的 OpenGL、資料庫的 SQLite 和支援多種媒體播放的函數庫。

Android 執行環境

Android 執行環境是由虛擬機器和核心 Java 函數庫組成，其說明如下所示：

- 虛擬機器：Android 虛擬機器是 Google 針對行動裝置實作的 Java 執行環境，因為 Android 應用程式是使用 Java 撰寫和編譯，在虛擬機器上執行，4.4 之前版本是使用 Dalvik VM 虛擬機，在 4.4 版新增更有效率的 ART（Android Runtime），需要自行切換使用，5.x 版預設是 ART。所以，Java 程式碼在編譯成 Java 類別檔後，還需轉換成 Android 虛擬機器的格式，才能在 Dalvik VM 或 ART 虛擬機執行。

- 核心 Java 函數庫：核心 Java 函數庫，Android 支援的 Java API 是 Java SE API 的子集，刪除一些不需要的套件，例如：列印、視窗 Swing 或 AWT 等。

Linux 核心

Android 作業系統是架構在 Linux 作業系統之上，使用的 Linux 核心負責提供系統的核心服務，包含：執行緒、低階的記憶體管理、網路、行程管理、電源管理（Power Management）和硬體的驅動程式。

5. 在 Android 作業系統上執行的應用程式是由多種元件（Components）組成，這些元件就是一些 Android 框架的 Java 類別，我們只需繼承這些類別，覆寫方法和擴充其功能，就可以建立 Android 應用程式。

Android 應用程式的組成有四種元件：活動（Activities）、內容提供者（Content Providers）、廣播接收器（Broadcast Receivers）和服務（Services）。

基本上，Android 應用程式不一定四種元件都擁有，如果有多個活動、廣播接收器或服務，我們可以使用意圖（Intents）來啟動各元件，建立更複雜的 Android 應用程式。

活動

活動是 Android 應用程式與使用者互動的元件，可以用來定義使用者經驗，它也是唯一可以讓使用者看到的元件，即使用介面畫面。Android 應用程式可以建立一至多個活動來處理應用程式所需的不同互動。

一般來說，一個活動就是使用者在行動裝置上看到的單一螢幕畫面（大多是佔滿整個螢幕，不過也有可能是對話方塊），一個 Android 應用程式通常擁有一至多個活動，如同 Web 網站擁有多頁網頁。

意圖

意圖（Intents）是一個啟動其他 Android 活動、服務和廣播接收器的非同步訊息，非同步（Asynchronous）是指訊息的送出和接收是相互獨立。意圖可以告訴 Android 作業系統我想作什麼？執行什麼動作？此時，作業系統是使用意圖篩選（Intent Filters）來找出可以處理的元件，例如：啟動其他活動、告訴指定服務可以啟動或停止與送出廣播。

事實上，意圖是一個描述特定操作的機制，例如：寄送電子郵件、瀏覽網頁和打電話回家等特定操作或動作，當 Android 應用程式需要寄送電子郵件時，就可以使用意圖來啟動系統內建的預設電子郵件工具，或自行撰寫一個郵件工具來處理此意圖，也就是撰寫 Android 應用程式取代系統內建的相關工具。

內容提供者

內容提供者是在不同 Android 應用程式之間分享資料的介面，它是一組封裝的資料，提供客製化 API 來進行讀寫。例如：聯絡人應用程式並沒有儲存任何聯絡人資料，它是透過內容提供者取得聯絡人資訊：姓名、地址和電話等，換句話說，其他需要使用聯絡人資料的 Android 應用程式，都可以透過同一個內容提供者來存取聯絡人資料。

事實上，內容提供者的主要目的是切割儲存資料和實際使用資料的應用程式，以便增加 Android 作業系統的彈性，因為任何人都可以自行撰寫電話簿應用程式來存取相同的聯絡人資料。

廣播接收器

廣播接收器顧名思義是用來接收廣播並且做出回應，這是 Android 實作系統層級的廣播與回應機制，事實上，Android 系統本身就會常常發出廣播，例如：接到來電、收到簡訊、啟動相機裝置、時區改變、系統開機、電池剩餘電量過低或使用者選擇偏好語言時，Android 系統都會發出廣播。

在 Android 應用程式可以使用廣播接收器來接收有興趣的廣播，即回應廣播，或自行送出廣播讓其他應用程式知道你的應用程式有狀態的改變。

服務

服務是在背景執行的行程，可以執行和活動一樣的工作，只是沒有使用介面。例如：播放背景音樂時，之所以不會打斷我們發送簡訊或收發電子郵件，因為它是一個在背景執行的服務，才能讓音樂播放不會中斷。

Android 作業系統本身就內建許多系統服務，我們可以直接使用 API 來使用這些服務，例如：定位服務。

活動是 Android 應用程式與使用者互動的元件，可以用來定義使用者經驗，它也是唯一可以讓使用者看到的元件，即使用介面畫面。Android 應用程式可以建立一至多個活動來處理應用程式所需的不同互動。

一般來說，一個活動就是使用者在行動裝置上看到的單一螢幕畫面（大多是佔滿整個螢幕，不過也有可能是對話方塊），一個 Android 應用程式通常擁有一至多個活動，如同 Web 網站擁有多頁網頁。

意圖（Intents）是一個啟動其他 Android 活動、服務和廣播接收器的非同步訊息，非同步（Asynchronous）是指訊息的送出和接收是相互獨立。意圖可以告訴 Android 作業系統我想作什麼？執行什麼動作？此時，作業系統是使用意圖篩選（Intent Filters）來找出可以處理的元件，例如：啟動其他活動、告訴指定服務可以啟動或停止與送出廣播。

事實上，意圖是一個描述特定操作的機制，例如：寄送電子郵件、瀏覽網頁和打電話回家等特定操作或動作，當 Android 應用程式需要寄送電子郵件時，就可以使用意圖來啟動系統內建的預設電子郵件工具，或自行撰寫一個郵件工具來處理此意圖，也就是撰寫 Android 應用程式取代系統內建的相關工具。

6. Android 開發環境（Android Development Environment）是由三個主要元件組成：JDK、Android Studio 和 Android SDK。

7. 請參考第 11-5-2 節的說明。

8. 請參考第 11-5-3 節的說明。

9. Android 模擬器（Android Virtual Devices，英文簡稱 AVD）是一個非常有用的工具，它可以在 Windows 作業系統模擬一台執行 Android 作業系統的行動裝置，幫助我們測試 Android Studio 建立的 Android 應用程式，而不用真正購買一台實機的智慧型手機或平板電腦。

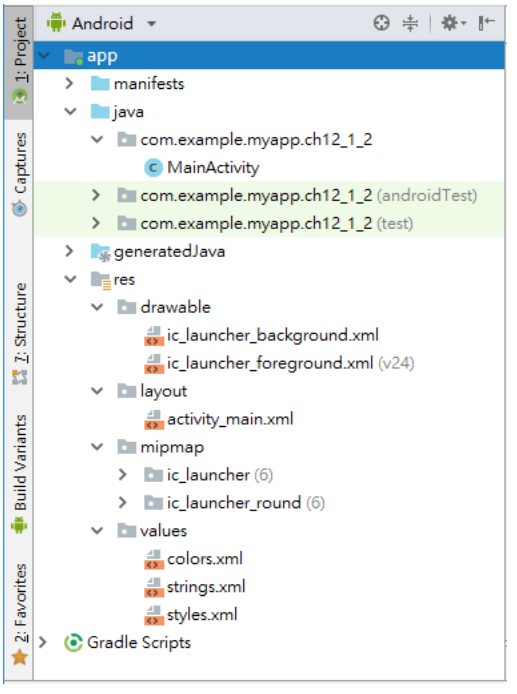
10. 請參考第 11-6-1 節的說明。

第 12 章：版面配置與使用介面元件

1. 請參考第 12-1-1 節的說明。

2. 請參考第 12-1-2 節的說明。

3. Android Studio 專案預設建立多個目錄、子目錄和檔案，而且提供多種檢視方式，預設是【Android】檢視。Android 檢視的專案結構，以 Ch12_1_2 專案為例，如下圖所示：



上述 Android 應用程式專案結構的主要目錄和檔案說明，如下所示：

app\manifests 目錄

在此目錄下的 AndroidManifest.xml 檔案是十分重要的檔案，提供 Android 作業系統關於應用程式的資訊，即一個功能清單。不同於 Windows 作業系統，Android 作業系統需要透過 AndroidManifest.xml 檔

案先認識這個應用程式，才能知道如何執行此應用程式。其主要提供的資訊有：

- 應用程式的完整名稱（包含 Java 套件名稱），一個唯一的識別名稱，可以讓 Android 作業系統和 Google Play 找到此應用程式。
- 應用程式包含的活動、內容提供者、廣播接收器和服務元件。
- 宣告應用程式執行時需要的權限，例如：存取網路和 GPS 等。

app\java 目錄

在此目錄下是 Java 類別的原始程式碼檔案（.java），位在套件 com.example.myapp.ch12_1_2 對應的路徑之下，展開套件可以看到之下的 Java 程式檔案清單，以此例是 MainActivity.java（活動類別的 Java 程式檔）。

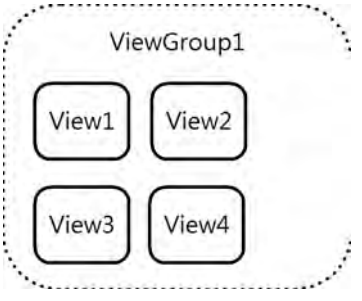
app\res 目錄

在此目錄下是一些資源子目錄，內容是 Android 應用程式使用到的相關資源檔案，一些常用子目錄的說明，如下表所示：

子目錄	內容說明
drawable	在目錄下是不同尺寸的 JPEG 或 PNG 格式圖檔案，hdpi 是 72x72 點；mdpi 是 48x48 點；xhdpi 是 96x96 點；xxhdpi 是 144x144 點，可以使用在高、中和低不同解析度的行動裝置螢幕
layout	定義使用介面版面配置的 XML 檔，例如：

	activity_main.xml
mipmap	取代 drawable 目錄存放圖片，可以最佳化在不同螢幕尺寸的縮放顯示，特別是在需要顯示動畫時，使用方式和 drawable 目錄相同，Android Studio 專案的圖示檔 ic_launcher.png 已經改置於此目錄
menu	顯示應用程式選單的 XML 檔
values	定義程式使用的陣列、字串、尺寸、色彩和樣式的常數值，例如：dimes.xml 是尺寸、strings.xml 是字串和 styles.xml 是樣式

4. 版面配置是 android.widget 套件的類別，一些看不見的容器物件（ViewGroup 物件），可以用來群組與編排介面元件（View 物件，也屬於 android.widget 套件），如下圖所示：



上述 ViewGroup1 物件是一個容器，在之中編排 4 個 View 物件。實作上，我們並不用撰寫程式碼來建立 ViewGroup 版面配置和 View 介面物件，而是建立版面配置資源的 XML 檔，使用宣告方式定義使用介面擁有哪些元件和如何編排。

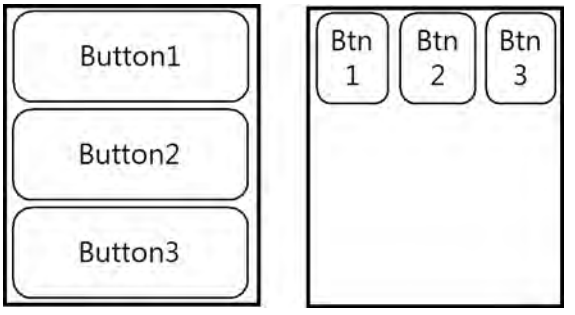
ViewGroup 版面配置類別

Android 提供多種擁有預設編排方式的版面配置物件，我們只需依照需求選擇版面配置來編排子介面元件（Views），即可快速編排出所需的使用者介面，常用類別的簡單說明，如下所示：

- ConstraintLayout 類別：使用元件和元件或邊界之間的距離和對齊的限制連接線來編排元件，可以建立自動調整和產生適用在不同螢幕尺寸的使用介面，即<ConstraintLayout>標籤。
- LinearLayout 類別：其包含的子介面元件是一個接著一個排列成水平或垂直一條直線，即<LinearLayout>標籤。
- TableLayout 類別：使用表格欄與列來編排子介面元件，每一個介面元件是新增至表格每一列的 TableRow 物件，即<TableLayout>和<TableRow>標籤。
- FrameLayout 類別：如同堆疊來編排多個子介面元件，所有子元件都是位在左上角的同一位置，每一個元件如同一頁圖層，即<FrameLayout>標籤。

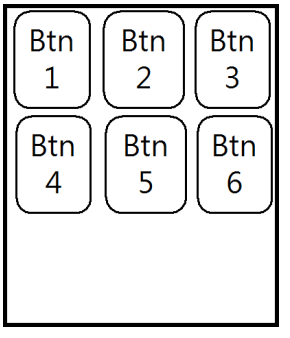
LinearLayout 版面配置

LinearLayout 版面配置是最常使用的版面配置，它可以將子介面元件排列成一列（垂直），或一欄（水平），一個接著一個排列成一直線，如下圖所示：



TableLayout 版面配置

TableLayout 版面配置使用表格的欄與列來編排子介面元件，每一個介面元件是新增至表格的每一列，即 TableRow 元件，如下圖所示：

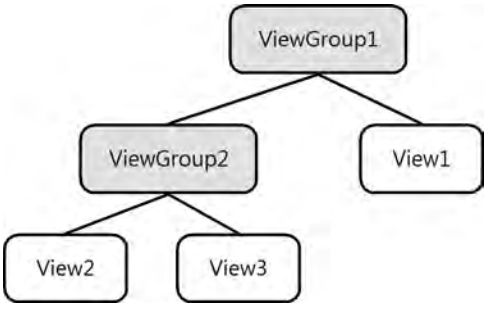


5. View 類別是所有使用介面元件的基礎類別（直接或間接父類別），其繼承的子類別分成兩大類，如下所示：

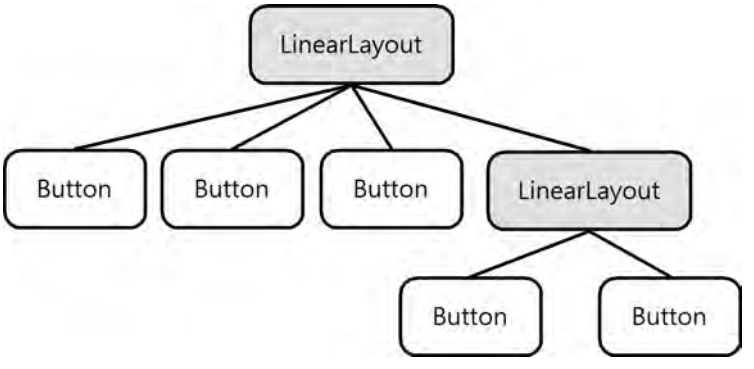
- 介面元件（Widgets，可稱為 View 物件）：正確的說，Android 的介面元件是 Widget；不是 View，Widget 是 View 的子類別，就是一些與使用者互動的圖形介面元件，例如：Button 和 EditText 元件等。
- 版面配置類別（Layout Class，可稱為 ViewGroup 物件）：ViewGroup 抽象類別是 View 的子類別，它是版面配置類別的父類

別，一種看不見的容器類別，用來組織其他介面元件和 ViewGroup 物件。

Android 應用程式的使用介面，以 Java 程式碼的角度來看，在活動視窗的使用介面是一棵 View 和 ViewGroup 物件組成的樹狀結構，如下圖所示：



上述樹狀結構代表螢幕上顯示介面元件的架構，在根 ViewGroup 物件之中可以包含多個 View 物件，或另一個 ViewGroup 物件，可以編排另一組 View 物件。例如：第 12-3-3 節的 Android 專案，其使用介面的樹狀結構，如下圖所示：



6. 在 Android 介面元件的 XML 屬性指定尺寸時，除了 fill_parent 和 wrap_content 常數外，我們還可以指定元件實際的尺寸，可以使用的單位說明，如下表所示：

單位	說明
dp 或 dip	Desity-independent Pixel 的簡稱，一英吋實際的螢幕尺寸相當於 160dp，這是 Android 建議使用的尺寸單位
sp	Scale-independent Pixel，類似 dp，建議使用在字型尺寸
pt	一點等於 1/72 英吋
px	實際螢幕上的點，Android 不建議使用此尺寸單位

7. TextView、EditText、CheckBox 和 RadioButton

8. 請參考第 12-4-3 節的步驟改為插入 TextView 元件。

9. 請參考第 12-1-2 和 12-4-4 節的步驟來建立使用介面。

10. 在 Android 應用程式建立披薩店訂購程式，在勾選和輸入數量後可以計算訂購的總價，其執行結果如下圖所示：

XML 文件：\res\layout\activity_main.xml

```
01: <?xml version="1.0" encoding="utf-8"?>
02: <LinearLayout xmlns:android=
        "http://schemas.android.com/apk/res/android"
03:     android:orientation="vertical"
04:     android:layout_width="fill_parent"
05:     android:layout_height="fill_parent">
06:     <CheckBox android:id="@+id/chkOriginal"
07:         android:text="原味披薩 $250"
08:         android:checked="true"
09:         android:layout_width="wrap_content"
10:         android:layout_height="wrap_content"/>
11:     <CheckBox android:id="@+id/chkBeef"
12:         android:text="牛肉披薩 $275"
13:         android:layout_width="wrap_content"
14:         android:layout_height="wrap_content"/>
15:     <CheckBox android:id="@+id/chkSeaFood"
16:         android:text="海鮮披薩 $350"
17:         android:layout_width="wrap_content"
18:         android:layout_height="wrap_content"/>
19:     <LinearLayout
20:         android:orientation="horizontal"
21:         android:layout_width="fill_parent"
22:         android:layout_height="wrap_content">
23:         <TextView
24:             android:layout_width="wrap_content"
25:             android:layout_height="wrap_content"
26:             android:text="數量(每種幾個):"/>
27:         <EditText android:id="@+id/txtQuantity"
28:             android:layout_width="fill_parent"
29:             android:layout_height="wrap_content"
30:             android:inputType="number"/>
31:     </LinearLayout>
32:     <Button android:id="@+id/btn1"
33:         android:text="計算訂購金額"
34:         android:layout_width="wrap_content"
35:         android:layout_height="wrap_content"
36:         android:onClick="btn1_Click"/>
37:     <TextView android:id="@+id/lblOutput"
38:         android:layout_width="wrap_content"
39:         android:layout_height="wrap_content"/>
```

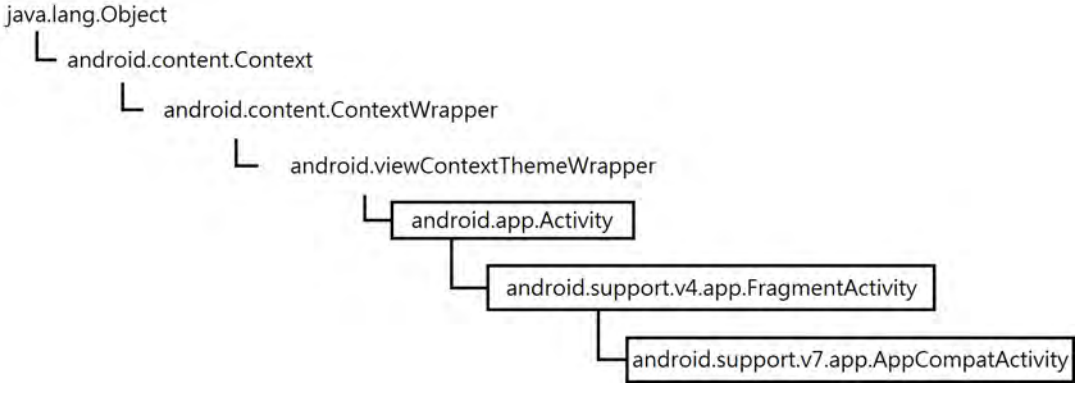
```
40: </LinearLayout>
```

第 13 章：活動與事件處理

1. 活動類別（Activity）是 Android 應用程式的核心，這也是使用者唯一會注意到的元件，因為大部分活動都會與使用者互動，我們建立 Android 應用程式的大部分時間都是在定義和實作每一個螢幕畫面的活動類別。

活動類別的類別架構

Activity 類別是 Android 應用程式最重要的類別，我們建立的活動類別可以繼承 Activity、FragmentActivity 或 AppCompatActivity 類別，其類別架構如下圖所示：



上述 FragmentActivity 和 AppCompatActivity 是 Activity 類別的子類別，它們是支援函數庫的子類別（為了相容舊版 Android），在中間的 2 個包裝類別是為了讓開發者更改 Context 物件的行為和佈景，基本上，Context 類別的抽象方法都是在 ContextWrapper 子類別實作。

在 Android 應用程式宣告活動

在 AndroidManifest.xml 檔案的 application 子元素宣告應用程式擁有的活動，例如：第 12 章 Ch12_1_1 專案的 AndroidManifest.xml 檔案，XML 標籤 application 元素的內容，如下所示：

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

上述 activity 子元素宣告應用程式擁有的活動 MainActivity，在 intent-filter 子元素定義此活動需要回應哪些操作或動作，2 個子元素的簡單說明，如下所示：

- action 元素：屬性值 android.intent.action.MAIN 表示活動是 Android 應用程式的進入點，也就是說，當使用者執行此應用程式，回應的操作就是執行此活動。

- category 元素：屬性值 android.intent.category.LAUNCHER 表示將程式置於啟動器的安裝程式清單中，所以，我們可以在清單中看到安裝的程式圖示。

2. Android 應用程式有多種方法來啟動活動，其簡單說明如下所示：

- 在 Manifest 檔宣告 Android 應用程式第一個進入的活動（相當於程式的進入點，即 Java 語言的 main()主程式方法）。
- 使用覆寫 ContentWrapper 物件的 startActivity()方法啟動活動。
- 使用 Activity 物件的 startActivityForResult()方法啟動活動。

3. 請參考第 13-2-1 和 13-2-2 節的說明。

4. 「事件」（Event）是在執行 Android 應用程式時，手指操作介面元件或鍵盤時所觸發的一些動作。Android 事件處理是一種「委託事件處理模型」（Delegation Event Model），分為「事件來源」（Event Source）和處理事件的「傾聽者」（Listener）物件，如下圖所示：

上述圖例的事件來源可能是按一下、長按和鍵盤事件或元件產生的選取或文字輸入事件，當事件產生時，註冊的傾聽者物件可以接收事件然後呼叫相關方法進行處理，傾聽者是一個委託處理指定事件的物件。

換句話說，當使用者按一下使用介面的 Button 元件，就會產生 Click 事件，因為我們已經註冊 Button 元件的傾聽者物件（例如：btnListener 物件）且實作 onClick()方法，此時，Click 事件就是執行傾聽者物件的 onClick()方法進行事件處理。

5. 使用介面元件的 android:id 屬性是身分證字號，用來找到指定介面元件的索引（在第 12-4-2 節已經說明過），如果需要撰寫 Java 程式碼更改介面元件的屬性值，或建立事件處理，介面元件一定要指定此屬性

值，例如：建立 Button 元件的事件處理方法、取得 EditText 元件輸入的內容和在 TextView 元件顯示輸出結果等。

android:id 屬性值的命名原則

在 XML 標籤指定的 android:id 屬性值，Android Studio 會自動編譯成 Java 語言的類別常數索引，一個 Java 語言的變數，所以一樣需要遵守 Java 變數的命名原則，例如：第 13-4-1 節的 activity_main.xml，我們替 EditText 元件命名為 txtC，如下所示：

```
<EditText
    android:layout_width="100dp"
    android:layout_height="wrap_content"
    android:inputType="number"
    android:ems="10"
    android:id="@+id/txtC"
    android:layout_column="0" />
```

上述 android:id 屬性值是「@+id」開頭，表示在目前 Android 應用程式的命名空間新增（因為有「+」加號）一個「/」符號之後的識別名稱 txtC，習慣上，我們都是以小寫字母開頭來進行命名。

在 Android Studio 專案參考此元件

Android Studio 專案參考此元件的 XML 屬性值和 Java 常數的寫法，如下所示：

- XML 文件：使用@id/txtC 或@android:id/txtC 來參考此元件。
- Java 程式碼：使用 R.id.txtC 參考此元件，因為 Android Studio 會自動編譯成 R.java 類別檔的常數，R 是指 R.java 類別檔；id 是指介面元件的 id 屬性，最後的 txtC 就是我們命名的元件名稱，id 屬性值。

在 Java 類別的方法可以使用上述類別常數來找出 EditText 元件 txtC，如下所示：

```
EditText txtC = (EditText) findViewById(R.id.txtC);
```

上述 findViewById()方法的參數是 R.id.txtC 類別常數，從資源索引檔 R.java 取得 txtC 的常數值，程式碼可以使用此值找出使用介面的 EditText 元件。

6. 請參考第 13-4-1 和第 13-4-2 節的說明。

7. Android 提供 android.util 套件的 Log 類別，可以建立記錄訊息，幫助我們進行程式偵錯。Log 類別常用的類別方法，如下表所示：

類別方法	說明
Log.e()	記錄錯誤訊息
Log.w()	記錄警告訊息
Log.i()	記錄一般資訊的訊息
Log.d()	記錄偵錯訊息
Log.v()	記錄詳細的訊息

我們可以使用 Log.d()方法來顯示呼叫各活動方法的訊息，如下所示：

```
Log.d(TAG, "ActivityCh13_2_3:onCreate");
```

上述方法有 2 個參數，第 1 個參數是標籤，可以用來識別是誰產生此訊息，第 2 個參數是訊息內容。一般來說，我們會使用一個類別常數來指定第 1 個參數，如下所示：

```
private static final String TAG = "Ch13_2_3";
```

請注意！使用 Log 類別方法記錄訊息會降低 Android 應用程式的執行效能，所以，通常只有在程式開發和偵錯階段使用，正式釋出版本就會刪除。

Toast 類別屬於 android.widget 套件，提供相關類別方法可以在行動裝置顯示一個彈跳訊息框，我們可以在此訊息框顯示一段訊息文字，而且只會保留一段時間。例如：使用 Toast 類別的 makeText()類別方法建立一段訊息，如下所示：

```
Toast.makeText(this, "按下 DEL 鍵...",
        Toast.LENGTH_SHORT).show();
```

上述 makeText()方法的第 1 個參數是 Context 物件，在 Activity 類別就是 this，如果在匿名內層類別是 Ch13_5_2Activity.this，第 2 個參數是顯示字串，或 strings.xml 的字串資源的索引，例如：R.string.hello，最後 1 個參數是顯示時間，Toast.LENGTH.SHORT 較短；Toast.LENGTH.LONG 較長。

最後使用串流呼叫 show()類別方法顯示 makeText()方法建立的訊息文字。

長按、鍵盤和觸控事件和如何處理這些事件的說明請參閱第 13-5 節。

8. 在 Android 應用程式建立 BMI 計算機，使用 2 個 EditText 元件輸入身高與體重，按下按鈕可以顯示計算結果的 BMI 值，其執行結果如下圖所示：

XML 文件：\res\layout\activity_main.xml

```
01: <?xml version="1.0" encoding="utf-8"?>
02: <LinearLayout xmlns:android=
        "http://schemas.android.com/apk/res/android"
03:     android:orientation="vertical"
04:     android:layout_width="fill_parent"
05:     android:layout_height="fill_parent">
06:     <LinearLayout
07:         android:orientation="horizontal"
08:         android:layout_width="fill_parent"
09:         android:layout_height="wrap_content">
10:         <TextView
11:             android:layout_width="wrap_content"
12:             android:layout_height="wrap_content"
13:             android:text="身高(公分):"/>
14:         <EditText android:id="@+id/txtHeight"
15:             android:layout_width="fill_parent"
16:             android:layout_height="wrap_content"
17:             android:inputType="number"/>
18:     </LinearLayout>
19:     <LinearLayout
20:         android:orientation="horizontal"
21:         android:layout_width="fill_parent"
22:         android:layout_height="wrap_content">
23:         <TextView
24:             android:layout_width="wrap_content"
25:             android:layout_height="wrap_content"
26:             android:text="體重(公斤):"/>
```

```
27:             <EditText android:id="@+id/txtWeight"
28:                 android:layout_width="fill_parent"
29:                 android:layout_height="wrap_content"
30:                 android:inputType="number"/>
31:         </LinearLayout>
32:         <Button android:id="@+id/btn1"
33:             android:text="計算 BMI 值"
34:             android:layout_width="wrap_content"
35:             android:layout_height="wrap_content"/>
36:         <TextView android:id="@+id/lblOutput"
37:             android:layout_width="wrap_content"
38:             android:layout_height="wrap_content"/>
39: </LinearLayout>
```

Java 程式：MainActivity.java

```
01: public class MainActivity extends Activity {
02:         @Override
03:         public void onCreate(Bundle savedInstanceState) {
04:             super.onCreate(savedInstanceState);
05:             setContentView(R.layout.activity_main);
06:             // 找出 Button 元件
07:             Button btn = (Button) findViewById(R.id.btn1);
08:             // 註冊傾聽者物件
09:             btn.setOnClickListener(listener);
10:         }
11:         // 建立傾聽者物件
12:         View.OnClickListener listener =
                                        new View.OnClickListener() {
13:           public void onClick(View v) {
14:             double height, weight, bmi;
15:             // 取得輸入值
16:             EditText txtHeight =
                        (EditText) findViewById(R.id.txtHeight);
17:             EditText txtWeight =
                        (EditText) findViewById(R.id.txtWeight);
18:             height = Double.parseDouble(
                                    txtHeight.getText().toString());
19:             weight = Double.parseDouble(
                                    txtWeight.getText().toString());
20:             // 計算 BMI
21:             height = height / 100.00;
```

```
22:             bmi = weight / (height * height);
23:             // 顯示 BMI
24:             TextView output =
                        (TextView) findViewById(R.id.lblOutput);
25:             output.setText(Double.toString(bmi));
26:         }
27:     };
28: }
```

9. 在 Android 應用程式建立披薩店訂購程式，在勾選和輸入數量後可以計算訂購的總價，其執行結果如下圖所示：

XML 文件：\res\layout\activity_main.xml

```
01: <?xml version="1.0" encoding="utf-8"?>
02: <LinearLayout xmlns:android=
            "http://schemas.android.com/apk/res/android"
03:         android:orientation="vertical"
04:         android:layout_width="fill_parent"
05:         android:layout_height="fill_parent">
06:         <CheckBox android:id="@+id/chkOriginal"
07:             android:text="原味披薩 $250"
08:             android:checked="true"
09:             android:layout_width="wrap_content"
10:             android:layout_height="wrap_content"/>
11:         <CheckBox android:id="@+id/chkBeef"
12:             android:text="牛肉披薩 $275"
13:             android:layout_width="wrap_content"
```

```
14:             android:layout_height="wrap_content"/>
15:         <CheckBox android:id="@+id/chkSeaFood"
16:             android:text="海鮮披薩 $350"
17:             android:layout_width="wrap_content"
18:             android:layout_height="wrap_content"/>
19:         <LinearLayout
20:             android:orientation="horizontal"
21:             android:layout_width="fill_parent"
22:             android:layout_height="wrap_content">
23:             <TextView
24:                 android:layout_width="wrap_content"
25:                 android:layout_height="wrap_content"
26:                 android:text="數量(每種幾個):"/>
27:             <EditText android:id="@+id/txtQuantity"
28:                 android:layout_width="fill_parent"
29:                 android:layout_height="wrap_content"
30:                 android:inputType="number"/>
31:         </LinearLayout>
32:         <Button android:id="@+id/btn1"
33:             android:text="計算訂購金額"
34:             android:layout_width="wrap_content"
35:             android:layout_height="wrap_content"
36:             android:onClick="btn1_Click"/>
37:         <TextView android:id="@+id/lblOutput"
38:             android:layout_width="wrap_content"
39:             android:layout_height="wrap_content"/>
40: </LinearLayout>
```

Java 程式：MainActivity.java

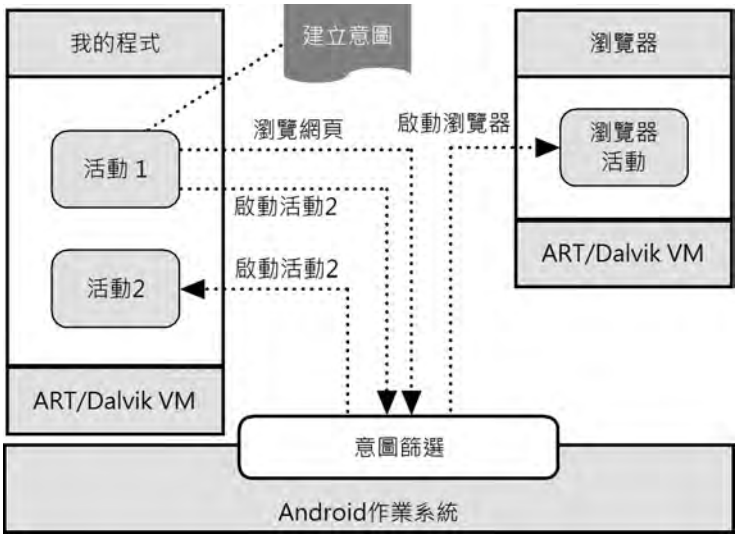
```
01: public class MainActivity extends Activity {
02:         @Override
03:         public void onCreate(Bundle savedInstanceState) {
04:             super.onCreate(savedInstanceState);
05:             setContentView(R.layout.activity_main);
06:         }
07:         public void btn1_Click(View view) {
08:             int amount = 0, quantity = 1;
09:             // 取得輸入的數量
10:             EditText txtQuantity =
                        (EditText) findViewById(R.id.txtQuantity);
11:             quantity = Integer.parseInt(
```

```
                        txtQuantity.getText().toString());
12:             // 檢查勾選哪些披薩
13:             CheckBox original =
                    (CheckBox) findViewById(R.id.chkOriginal);
14:             if (original.isChecked())
15:                 amount += 250 * quantity;
16:             CheckBox beef =
                    (CheckBox) findViewById(R.id.chkBeef);
17:             if (beef.isChecked())
18:                 amount += 275 * quantity;
19:             CheckBox seaFood =
                    (CheckBox) findViewById(R.id.chkSeaFood);
20:             if (seaFood.isChecked())
21:                 amount += 350 * quantity;
22:             // 顯示訂購金額
23:             TextView output =
                        (TextView) findViewById(R.id.lblOutput);
24:             output.setText(Integer.toString(amount));
25:         }
26: }
```

10. 請參考前上述 2 個習題的範例，其公式為: 新台幣 = 美金金額 * 兌換匯率

第 14 章：意圖與意圖篩選

1. Android 應用程式送出意圖的訊息需要經過 Android 作業系統來判斷接收者是誰，它是使用意圖篩選（Intent Filters）找出有能力處理的活動或內建應用程式，然後才將訊息送給接收者，如下圖所示：



上述圖例在【活動 1】建立和送出 2 個意圖（虛線），一個意圖指明是【活動 2】，作業系統可以依據指明類別找到此活動來轉送訊息，並且啟動此活動。另一個意圖是描述特定動作，即瀏覽網頁，作業系統依據此動作的意圖篩選找出可用的應用程式瀏覽器，然後啟動它。

Android 作業系統的意圖與意圖篩選機制，可以讓我們撰寫 Android 應用程式來取代系統內建的應用程式。例如：當撰寫一個新的瀏覽器後，作業系統透過意圖篩選可以找出 2 個處理網頁瀏覽的程式（一個是內建；一個是我們自建的工具），此時，作業系統會顯示一個選單讓使用者選擇使用哪一個工具，而且可以讓我們將新建工具指定為預設工具，換句話說，就是取代內建瀏覽器。

2. 在 Android 作業系統的意圖可以分為兩種，如下所示：

- 明確意圖（Explicit Intent）：指明目標活動接收者名稱，即明確指明是送給誰，通常是使用在連接同一個應用程式內部的多個活動，前述啟動活動 2 和第 14-2~14-4 節是明確意圖。

- 隱含意圖（Implicit Intent）：意圖只有指出執行的動作、型態和目錄，並沒有目標接收者的確實名稱，Android 作業系統任何可以完成此工作的應用程式都可以是接收者，即前述瀏覽網頁，在第 14-5 節就是隱含意圖。

3. 通常啟動同一個應用程式中的其他活動是使用明確意圖，當一個 Android 應用程式擁有多個活動時，我們需要使用意圖來啟動其他活動，其主要工作有三項，如下所示：

- 定義繼承 Activity 類別的新類別，它就是一個新活動。
- 在 AndroidManifest.xml 註冊新活動。
- 建立意圖來啟動新活動，如果需要可以在活動之間傳遞資料。

4. 在 Intent 物件攜帶傳遞資料是使用 Bundle 物件，它是一種目錄物件（Dictionary Object，儲存鍵與值的對應資料），可以儲存字串型態鍵值對應的各種資料型態資料。首先建立 Intent 物件，如下所示：

```
Intent intent = new Intent(this, FActivity.class);
```

然後就可以建立 Bundle 物件，它像是一個大包包，我們可以將資料直接放進去，如下所示：

```
Bundle bundle = new Bundle();
bundle.putString("TEMPC", txtC.getText().toString());
```

上述程式碼使用 putString()方法新增字串資料，第 1 個參數是字串的鍵值（之後我們需要使用此鍵值取出內容），第 2 個參數是值。常用的方法有 putInt()放入整數、putDouble()放入浮點數和 putByte()放入位元組資料等，如果資料不只一項，請重複呼叫 put???()方法將資料一一放入 Bundle 物件。

接著就可以使用 Intent 物件的 putExtras()方法來附加 Bundle 物件，和啟動活動，如下所示：

```
intent.putExtras(bundle);
startActivity(intent);
```

5. 請參考第 13 章學習評量 8 的解答和第 14-3 節的範例專案。

6. 請參考第 14-4 節的範例專案。

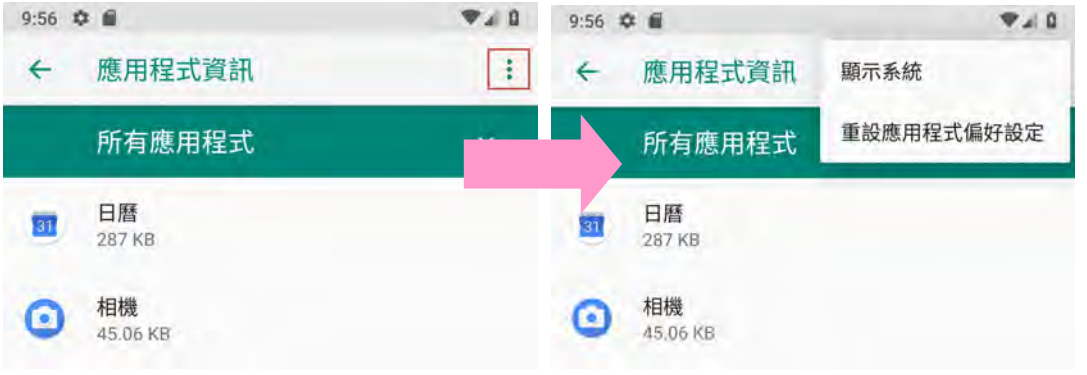
7. 請參閱第 14-5-1 節的說明。

8. 請參考第 14-5-2 節的範例專案。

第 15 章：選單、對話方塊與清單介面

1. Android 動作列是位在活動上方的一個固定區域，一種介面元件用來顯示活動的標題文字、圖示和切換與巡覽功能，也可以用來指示 Android 應用程式目前所在的活動，和顯示選單（Menu）。

Android 最常使用的是選項選單（Options Menu），當使用者按下行動裝置的實體【MENU】鍵，可以在下方顯示最多 6 個選項的選單。不過，目前 Android 行動裝置大多已經沒有實體【MENU】鍵，選項選單也整合至動作列成為「溢出選單」（Overflow），對於沒有實體【MENU】鍵的裝置來說，我們需要點選標題列最右方【垂直 3 點】圖示來開啟選單，如下圖所示：



上述【設定】程式上方顯示動作列的標題文字、圖示和最右邊【垂直 3 點】圖示，點選可以開啟選單，我們可以直接將選項置於動作列上來執行一些常用功能。

2. 請參閱第 15-1-2 節的說明。

3. 請參考第 15-1-2 節的範例專案。

4. 請參考第 13-4-2 節和第 15-1-2 節的範例專案。

5. 常用的相關方法說明，如下表所示：

方法	說明
setTitle()	指定對話方塊的標題文字為參數的字串
setMessage()	指定對話方塊的訊息內容為參數的字串，內容如需換行請使用「\n」符號
setCancelable()	指定對話方塊是否可取消，參數 true 是可以；false 為不可以

create()	傳回 AlertDialog.Builder 物件建立的 AlertDialog 物件
show()	顯示 AlertDialog.Builder 物件建立的 AlertDialog 對話方塊
setPositiveButton()	指定「確定」按鈕的標題文字和傾聽者物件
setNeutralButton()	指定「放棄」按鈕的標題文字和傾聽者物件
setNagtiveButton()	指定「取消」按鈕的標題文字和傾聽者物件

因為類別方法的傳回值都是 AlertDialog.Builder 物件，所以，可以直接使用串流呼叫方法（Method Chaining），如同項鍊的一串珠子一般依序呼叫各方法，直到 show()方法顯示對話方塊。

6. 請參考第 15-2-1 節和第 15-1-2 節的範例專案。

7. Android 提供兩種清單元件；Spinner 和 ListView 元件。

8. 接合器（Adapter）是一種介面物件，它是作為清單元件和資料來源之間的橋樑，也就是說，我們可以透過接合器從不同資料來源，建立 Spinner 和下一節 ListView 元件的項目，Android 預設提供三種接合器：ArrayAdapter 是陣列的資料來源；SimpleAdapter 是 XML 文件；CursorAdapter 是內容提供者。

9. 請參考第 15-3 節和第 15-1-2 節的範例專案。

10. 請參考第 15-4 節和第 13-4-1 和 13-4-2 節的範例專案。

第 16 章：偏好設定、檔案與 SQLite

資料庫

1. SharedPreferences 物件、檔案和 SQLite 資料庫。

2. 在 Android 提供 SharedPreferences 物件儲存簡單的應用程式資料，主要是指一些偏好設定的字型尺寸、使用者帳號、色彩或遊戲分數，實際上，它就是使用 XML 格式的偏好設定檔來儲存這些資料。

SharedPreferences

3. 請參考第 16-1 節的範例專案，在新增活動儲存偏好設定資料的匯率，在匯率轉換活動是取得偏好設定的匯率來執行轉換。

4. Java 語言的檔案處理是一種「串流」（Stream）模型，串流觀念最早是使用在 Unix/Linux 作業系統，串流模型如同水管的水流一般，當程式開啟來源的輸入串流（例如：檔案、記憶體和緩衝區等），Java 程式可以從輸入串流依序讀取資料。同理，如果程式需要輸出資料，可以開啟輸出串流（同樣是檔案、記憶體和緩衝區等），然後將資料寫入串流。

java.io

openFileInput()和 openFileOutput()方法

5. 請參考第 16-2 節的範例專案，在 MainActivity 活動將輸入溫度存入檔案，然後在 FActivity 活動讀取檔案資料來轉換溫度。

6. 請參考第 16-3 節的說明。

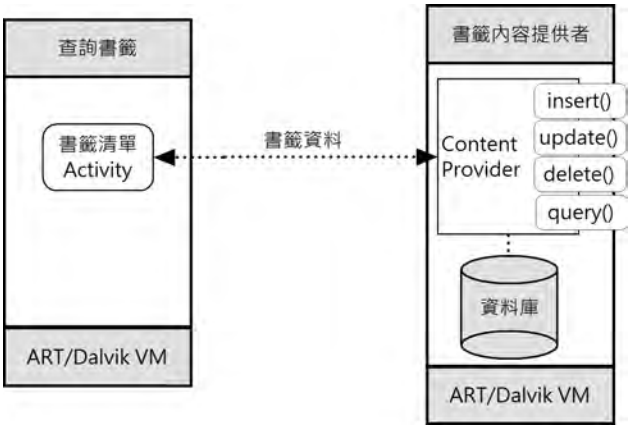
7. 請參考第 16-4 節的說明和範例專案。

8. 類似本章的學習評量 3，在新增活動將匯率存入資料庫的記錄資料，在匯率轉換活動是查詢資料庫的匯率來執行轉換。

第 17 章：內容提供者、廣播接收器與通知

1. 內容提供者的主要目的是切割儲存資料和實際使用資料的應用程式，以便增加整個 Android 作業系統的彈性，例如：任何人都可以自行撰寫電話簿應用程式來存取相同的聯絡人資料。

基本上，內容提供者是在不同 Android 應用程式之間分享資料的介面，它是一組封裝的資料，提供標準介面的方法來進行讀寫。例如：內建瀏覽器並沒有儲存任何書籤資料，它是透過內容提供者來取得書籤資訊：標題和 URL 網址等，如下圖所示：



上述圖例的【查詢書籤】應用程式是透過【書籤內容提供者】的內容提供者來提供書籤資料，換句話說，任何需要使用書籤資料的 Android 應用程式，都可以透過同一個內容提供者來存取書籤資料。

內容提供者是提供一組標準介面方法來存取資料，即 insert()、update()、delete()和 update()等方法，而將實際存取資料的程式碼包裝在內容提供者物件中，所以資料是儲存在資料庫、檔案、XML 文件或網路已經不重要了，只需透過介面方法就可以存取分享的資料。

2. 在 Android 作業系統已經內建多個內容提供者來儲存一些共享資訊，例如：聯絡人、瀏覽器書籤、通話記錄和媒體檔案等資訊。我們可以使用 android.provider 套件的相關類別來存取內容提供者，如下表所示：

內容提供者類別	說明
ContactsContract	聯絡人的相關資料，包含姓名、電話和電子郵件地址等
Browser	瀏覽器的相關資料，包含書籤和歷史記錄等
CallLog	通話的相關資料，包含未接來電、已接來電和通話記錄等
MediaStore	媒體檔案的相關資料，包含音樂、視訊和圖形檔
Settings	裝置設定和使用者偏好設定的相關資料

3.

內容提供者的 URI

內容提供者 URI 的基本語法，如下所示：

```
content://<內容提供者名稱>/<資料路徑>/<記錄編號>
```

上述語法是由「content://」字首開始，各部分語法的說明，如下所示：

- 內容提供者名稱：指定內容提供者的名稱，例如：聯絡人是 contacts；媒體檔案是 media；瀏覽器是 browser；通話是 call_log，如果是自行建立的內容提供者，就是類別的完整名稱。
- 資料路徑：指定從內容提供者請求哪一種資料，例如：聯絡人是 people。
- 記錄編號：指明是哪一筆記錄，例如：第 2 筆聯絡人資料。

一些內容提供者的 URI 查詢字串範例，如下表所示：

URI 查詢字串	**說明**
content://contacts/people	所有聯絡人資料的清單
content://browser/bookmarks	瀏覽器的所有書籤清單
content://call_log/calls	所有通話記錄的清單
content://media/internal/images	在內部儲存裝置的圖形檔清單
content://media/external/images	在外部儲存裝置 SD 卡的圖形檔清單

android.provider 套件的 URI 類別常數

Android SDK 在 android.provider 套件提供幫助類別（Helper Classes）的相關 URI 類別常數來代表上表的 URI 查詢字串，這是一些預先定義的類別常數，如下所示：

```
ContactsContract.Contacts.CONTENT_URI
Browser.BOOKMARKS_URI
CallLog.CONTENT_URI
MediaStore.Images.Media.INTERNAL_CONTENT_URI
MediaStore.Images.Media.EXTERNAL_CONTENT_URI
```

上述程式碼的 URI 類別常數可以取代上表的 URI 查詢字串（由上而下）。例如：CONTENT_URI，如下所示：

```
Uri contacts = ContactsContract.Contacts.CONTENT_URI;
```

上述程式碼可以取代 URI 查詢字串，如下所示：

```
Uri contacts = Uri.parse("content://contacts/people");
```

如果需要查詢指定記錄，請使用 ContentUris.withAppendedId()類別方法來新增記錄編號，即第 2 個參數值，第 1 個參數是 URI 類別常數，如下所示：

```
Uri person = ContentUris.withAppendedId(
                ContactsContract.Contacts.CONTENT_URI, 2);
```

4. 廣播接收器顧名思義是用來接收廣播並且做出回應，這是 Android 實作系統層級的廣播與回應機制。

Android 系統本身就會常常發出廣播，例如：接到來電、收到簡訊、啟動相機裝置、時區改變、系統開機、電池剩餘電量過低或使用者選擇偏好語言時，Android 系統都會發出廣播。

我們可以將廣播接收器想像是停車場上的多輛轎車，駕駛手上的遙控器就是廣播發送者，因為頻率不同，一輛轎車只能接收指定遙控器頻率的廣播，等到轎車接到廣播，就解開門鎖，駕駛才能打開車門進入車輛，其他駕駛的遙控器就沒有作用。

5. 廣播接收器本身並沒有任何使用介面，它是一個繼承 android.content.BroadcastReceiver 抽象類別的子類別，等到接收到指定的廣播而觸發時，即可在實作的 onReceive()抽象方法（因為是抽象方法，

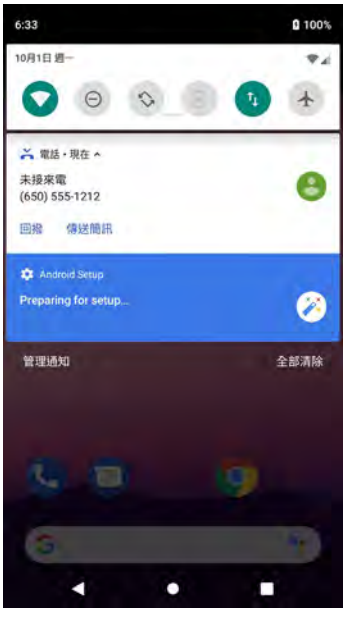
繼承的子類別一定要實作此方法）回應廣播來執行所需操作，如下所示：

```
public class MyBroadcastReceiver
                                extends BroadcastReceiver {
    public void onReceive(Context context, Intent intent) {
        // 處理接收的廣播
    }
}
```

上述類別是繼承自 BroadcastReceiver 類別，然後實作 onReceive()抽象方法來處理接收的廣播，在本節是處理系統廣播 PHONE_STATE。

6. Toast 類別顯示一段訊息、顯示一個對話方塊或啟動新活動來得到使用者的注意，另一種常用方法是在狀態列顯示訊息提醒。

狀態列（Status Bar）是行動裝置最上方的一條橫向的長條區域，「通知服務」（Notification Service）是一種系統服務，可以在狀態列顯示可向下捲動的通知訊息，例如：未接來電，如下圖所示：



上述狀態列顯示通知訊息的圖示，往下拖拉，可以展開通知的項目清單，顯示未接來電的電話號碼，點選通知項目就可以回撥電話。

7. 請參考第 17-4 節的說明。

PendingIntent 物件的內容就是包裝著 Intent 物件，可以讓其他應用程式在稍後觸發再來送出其中的 Intent 物件。

8. 請參考第 17-4 節的範例專案來建立 BMI 計算機的訊息提醒。

第 18 章：繪圖、多媒體與定位服務

1. ImageView 元件

2. VideoView 元件

3. 請參考第 18-3 節的說明。

4. 我們還可以在畫布上繪出一段旋轉文字。首先指定畫筆色彩、字型尺寸和繪出字串，如下所示：

```
paint.setColor(Color.BLACK);
paint.setTextSize(25);
String str = "旋轉的文字!";
canvas.rotate(-45, 200, 200);
paint.setStyle(Paint.Style.FILL);
canvas.drawText(str, 200, 200, paint);
```

上述 rotate()方法的第 1 個參數是旋轉角度，第 2 和 3 個參數是旋轉軸座標(200, 200)，以此座標為軸心來旋轉第 1 個參數的角度。

5. Android 行動裝置結合定位功能和 Google 地圖建立的「位置感知服務」（Location-based Service，LBS）是一項十分實用的功能，LBS 應用

程式可以追蹤你的位置和提供一些額外服務，例如：找出附近的咖啡廳、停車場、自動櫃員機或加油站等。

LBS 另一項常見的應用是路徑規劃的導航，除了行車導航外，對於大型展覽館、購物商場、城市觀光、野生動物園或主題樂園等，更可以提供導覽服務，結合定位功能來提供使用者更精確的位置資訊。

6. Android 作業系統的定位提供者（Provider）可以提供不同方式的定位服務，我們主要是使用下列兩種定位提供者，其說明如下所示：

- GPS 定位提供者：提供者名稱字串為"gps"，它是使用 GPS（Global Positioning System）的衛星訊號來定位，可以提供精確的位置資訊，但是無法收到衛星訊號的室內並無法使用。
- 網路定位提供者；提供者名稱字串為"network"，它是直接使用電信公司基地台來執行三角定位，其提供的位置資訊較不精確，但是可以在室內使用。

定位服務最主要的目的是找出行動裝置目前位置的經緯度座標，經緯度是經度與緯度合稱的座標系統，也稱為地理座標系統，它是使用三度空間的球面來定義地球表面各點的座標系統，能夠標示地球表面上的任何一個位置。經度與緯度的說明，如下所示：

- 緯度：地球表面某一點距離地球赤道以南或以北的度數，其值為 0 至 90 度，赤道以北的緯度叫北緯（符號為 N）；赤道以南的緯度稱南緯（符號為 S）。
- 經度：地球表面上某一點距離本初子午線（一條南北方向經過倫敦格林威治天文台舊址的子午線）以東或以西的度數，簡單的說，本

初子午線的經度是 0 度，其他地點的經度是向東從 0 到 180 度，即東經（符號為 W）或向西從 0 到 180 度，即西經（符號為 E）。

7. 請參考第 18-1 節和第 12-5-2 節的說明與範例專案。

8. 請參考第 18-4-2 節和第 12-5-1 節的說明與範例專案。