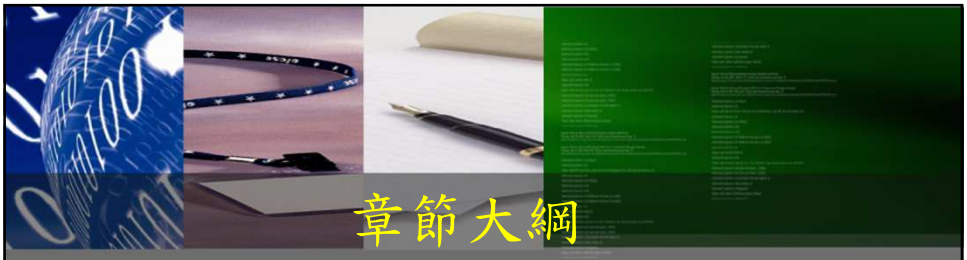
第一章 Java簡介

課前指引

在本章中，我們將針對Java程式語言的定位與特色做一些說明，並且示範如何編譯與執行Java程式，以配合往後章節的範例練習。



章節大綱

備註：可依進度點選小節

1.1 程式與程式語言

- 電腦硬體如果沒有軟體就無法發揮功效，要怎麼開發軟體呢？
  - 選擇一種程式語言來撰寫程式
  - 透過程式操作電腦硬體完成我們的要求。
- 程式語言
  - 人若要和電腦溝通的話，就必須使用電腦『懂』的語言，這種語言稱為程式語言(Programming Language)。
- 自然語言
  - 一般我們用來與人溝通的語言稱為自然語言(Natural Language)。

3

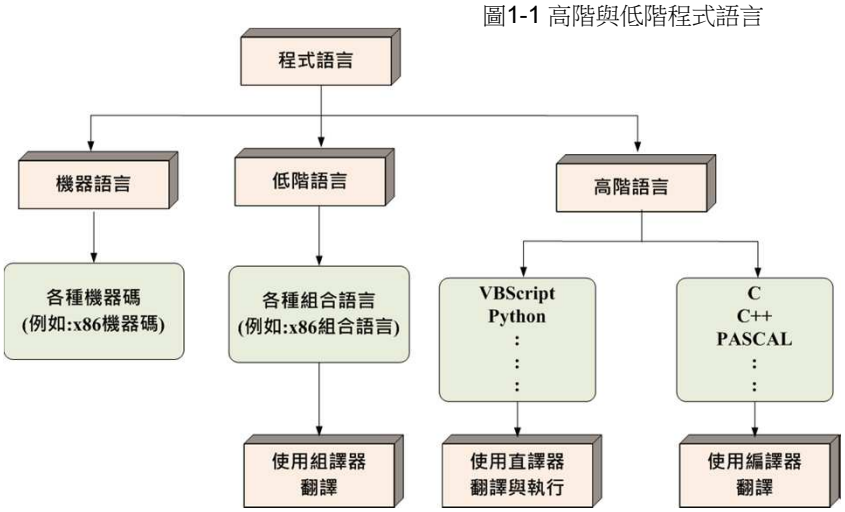
1.1 程式與程式語言

- 自然語言的相似度，程式語言可以分為3種：
  - 機器語言、低階語言及高階語言。
    - 高階語言與人類所使用的自然語言最為相近
    - 而機器語言則和人類所使用的語言南轅北轍。

4

1.1 程式與程式語言

圖1-1 高階與低階程式語言



5

1.3.1 1.1.1 機器語言

- 機器語言（Machine Language）
  - 是電腦硬體唯一能辨識、能解讀的語言
  - 機器語言就是一連串的0、1二進位數字組合，又稱為機器碼。
    - 這些0、1 的組合數字，可能代表某種資料，也可能代表某個指令
  - 大多數的人無法了解或記憶這一連串0、1數字所代表的涵義，因而發展了低階語言與高階語言。
    - 【註】：某些書籍或軟體會將機器語言使用16進制來加以表達。例如：01001001→49H。

6

1.1.2 低階語言

- 低階語言（Low-level Language）
  - 接近機器語言，不過卻使用人類比較容易記憶的單字形式來對應一連串的0、1組合。
  - **組合語言**(Assembly Language)。
    - 使用運算子與運算元來表示一連串的0、1組合，而這些運算子則使用類似英文的縮寫以利人類的記憶與理解，故稱為助憶碼。
    - 例如：使用INC來代表Increment（累加指令）。

8051機器語言指令	8051組合語言指令	意義
00000100	INC	執行累加1
10000100	DIV	執行除法

表1-1 8051 組合語言指令與機器語言指令的對應

7

1.1.2 低階語言

- 組合語言的指令與機器語言的指令是一對一的對應關係，但是卻比機器語言容易記憶
  - 其他關於組合語言的設計都與機器語言的設計相同
- 與機器語言一對一的特性
  - 組合語言可以完全掌控電腦的硬體結構
  - 在執行效率上自然也就完全交由程式設計師決定。
- 不同的CPU必須使用不同的組合語言
  - 必須對於該CPU的組織結構有充分認知
- 一個完整的組合語言指令恰好對應一組機器語言的0、1串列
  - 組譯程式（Assembler；又稱組譯器）
    是一種用來將組合語言轉換為機器語言的一套程式。

8

1.1.3 高階語言

- 高階語言（High-level Language）
  - 一種比組合語言更接近自然語言且不因更換機器而改變語法的程式語言。
  - 可使用更接近人類思維的方式來設計程式。
  - 當程式設計完成之後，必須先通過翻譯程式的翻譯，才能夠被電腦執行。
  - 運算子通常具有比較強大的功能，單一行的高階語言程式可能被翻譯成許多的機器碼，因此可完成較為複雜的工作。
  - 翻譯高階語言的工具程式分為兩大類：
    - 編譯器（compiler）與直譯器（interpreter）。

9

1.1.3 高階語言

- 編譯器
  - 採用整批作業(Batch)方式來處理程式翻譯的工作
    - 當我們將程式設計完畢並交由編譯器翻譯之後，編譯器會將之先轉換為中間碼，再將中間碼翻譯為目的碼並存入目的檔(object file)中，而這個目的檔再經由連結其他目的檔及程式庫之後，會形成可執行檔(execute file)，然後才能在電腦上直接執行。
  - 使用編譯器翻譯的程式語言
    - C、C++
      - 編譯器為gcc、g++、Visual C++、C++ Builder

10

1.1.3 高階語言

一個使用編譯器翻譯的程式處理流程如下圖

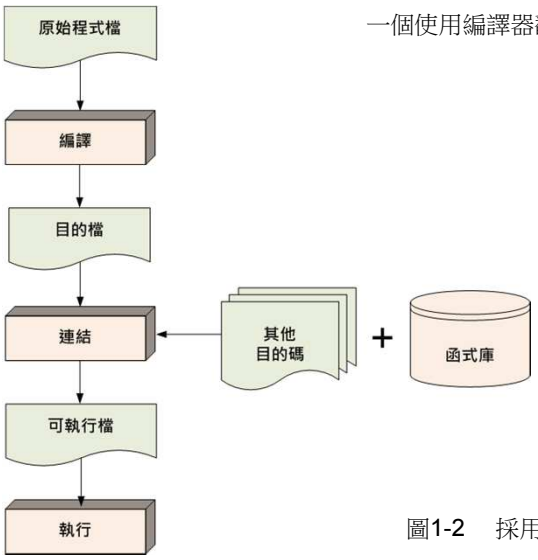


圖1-2　採用編譯器翻譯的程式處理流程

11

1.1.3 高階語言

- 使用模組化技巧來設計程式
  - 把某些具有特殊功能的片段程式獨立成一個個的函式，並將之集合成一個函式庫檔案
  - 讓需要使用該功能的程式透過連結的方式加以結合，縮短撰寫程式的時程。
  - 例如：math函式庫中就包含許多求三角函數值的函式，所以程式設計者不需撰寫求三角函數值的步驟，只要在需要求三角函數值時，引用math函式庫即可。
- 在圖1-2中，若原始程式中有些錯誤，必須重新回到編輯器修改原始程式，並重新執行整個流程，在開發除錯階段不方便。
  - 整合式開發環境（Integrated Development Environment；簡稱IDE）的發展，讓使用編譯器開發程式時，也同樣具有監督程式逐行執行的能力，改善了開發除錯的困難。

12

1.1.3 高階語言

- 直譯器
  - 直譯器並不產生目的檔或可執行檔。
  - 直譯器會逐行翻譯中間碼並送交電腦執行，因此，每一次要執行程式時，都必須啟動直譯器來重新翻譯程式。
  - 當程式中有錯誤發生時，前面正確程式碼對應的中間碼仍會被執行，並且停留在錯誤的那一行程式
  - 具有監督執行狀況的效果，故較適合用於程式開發過程。常見的JavaScript、VBScript、Python等都是採用直譯器翻譯的程式語言

圖1-3 採用直譯器翻譯的程式處理流程

13

1.1.3 高階語言

- 高階語言通常具有較高的硬體獨立性(machine independent)，也就是具有可攜性(portable)
  - 程式設計師在A機器上所撰寫的高階語言程式，如果想要拿到B機器上執行時，通常不需要修改程式（或只需要小幅度的修改）
    - 因為程式設計師只需要更換另外一套在B機器上執行的編譯器或直譯器，就可以將程式重新翻譯並在B機器上順利執行。
  - 高階語言的語法統一
    - 不像組合語言般必須考量到執行平台的因素。

14

1.2 程式語言簡介

- Java
  - 是一個高階語言
  - 也是一個物件導向程式語言（Object-Oriented Programming Language；簡稱**OOPL**）。
  - 並且在Java8時也加入了函數式程式語言的優點。
- 高階語言的分類與發展歷程

15

1.2 程式語言簡介

- 程式語言從型態上來區分，可以分為
  - 1. 命令式型態
    - 例如C/C++、BASIC、Pascal、FORTRAN、COBOL、Java等
  - 2. 函數式型態
    - LISP
  - 3. 邏輯式型態等等
    - Prolog
      - 其中函數式與邏輯式型態大多使用在人工智慧方面。

16

1.2 程式語言簡介

- 命令式型態程式語言(Imperative Paradigm Programming Language)
  - 透過一連串**指定敘述(assign statement)**組合而成，這些指定敘述又稱為**表示式(expression)**，只要依序執行這些指定敘述就可以得到結果
    - x = x+1;
  - 上述是一個Java語言表示式
  - 若以數學式子來表示，則為$x_{t+1} = x_t + 1$，其中t代表時間
  - 通常提供了幾種基本的控制流程結構，包含循序流程結構、選擇流程結構、迴圈流程結構、無條件跳躍等等
    - 程式設計師必須設計演算法，並藉由語言所提供的各種控制流程來實現演算法，以完成軟體的目的。
    - 除了無條件跳躍之外，我們將在第四章介紹Java所支援的幾種流程控制。

17

1.2 程式語言簡介

- 宣告式程式語言(Declarative Programming Language)
  - 命令式語言必須藉由流程來實現目標，宣告式語言只著重在目標，而非流程。
    - 命令式語言必須設計演算法明確告知下一步該怎麼做，電腦才知道該怎麼做，進而在最終達成目標。
    - 宣告式語言採用固定的演算法來完成各種操作，程式設計師只需要依照其語言規則來撰寫程式，語言執行器就會自動按照這些規則來執行
      - 這些規則有時具有推論與重寫性質，因此常用來解決人工智慧的設計問題。

18

1.2 程式語言簡介

- 宣告式程式語言(Declarative Programming Language)
  - 降低了平行處理的難度，因此在多核心、雲端的時代，越來越被重視。
  - 常見的宣告式語言及其套件如
    - SQL、
      正規表示式（Regular Expression）、
      邏輯式程式語言（Prolog）、
      函數式程式語言（如Haskell、Scala、R）。

19

1.2 程式語言簡介

- 函數式程式語言(Functional programming Language)
  - 函數式程式語言屬於宣告式程式語言的一種，提供以函數來撰寫程式的方式
  - 避免依靠程式目前的狀態與常變動的變數
  - 基礎則是λ演算（lambda calculus）。
    - Lambda演算強調規則的變換，與實現的機器無關，較偏向軟體的設計。
  - 在λ演算中，每個運算式都是一個函數，而且只有一個參數，也只回傳一個值。
    - 特別的是，不論是參數和回傳值，也都是個只有單一參數的函數。
    - 函數的參數也是個函數就是λ演算的特色。

20

1.2 程式語言簡介

- 函數式程式語言(Functional programming Language)
  - λ演算的函數是匿名的，它只使用運算式來表達這個函數對參數做了些什麼操作。
  - 例如有一個『+5』函數
    - 那麼它的運算式就是λx->x+5或λy->y+5。
      而傳統的數學函數則會寫成f(x)=x+5。
    - 我們還可以把λx->x+5當作是參數傳遞給另一個函數。
      一直重複這樣的動作，直到解決我們的問題為止。
    - 常見的函數式程式語言有Haskell、Scala、R等等。

21

1.2 程式語言簡介

- 函數式程式語言(Functional programming Language)
  - 一個Scala的範例程式，可求出$4^2+(3+2)^2$：

```
scala> def square(x: Double) = x * x
square: (Double)Double
scala> square(3)
unnamed0: Double = 9.0
scala> square(7 + 1)
unnamed1: Double = 64.0
scala> square(square(3))
unnamed2: Double = 81.0
scala> def sumOfSquares(x: Double, y: Double) = square(x) + square(y)
sumOfSquares: (Double,Double)Double
scala> sumOfSquares(4, 3 + 2)
unnamed3: Double = 41.0
```

22

1.2 程式語言簡介

- 結構化程式語言(Structural Programming Language)
  - 命令式型態的程式語言在撰寫時，為了要將程式的再利用率提高以利於維護，因此發展成結構化程式語言
    - 例如C語言。
  - 採用結構化分析設計的方法，將問題由上向下，由大到小切割成許多子問題，分別尋得子問題的解答，最後再結合起來解決所要解決的問題。
    - 也稱為模組化設計。
    - 解決子問題的方法被分成一個個的模組，這些模組在某些語言（例如C語言）中被稱為**函式**
  - 使用for、while等條件式迴圈來取代goto無條件跳躍指令
    - 有利於日後的程式維護。

23

1.2 程式語言簡介

- 結構化程式語言(Structural Programming Language)
  - 結構化程式設計也可以利用函式來進行設計
    - 例如平方和功能，可撰寫如下程式（以C語言為例）：

```
……
double square(double x) { return x*x; }
double sum(double x,double y) { return x+y; }
double sumOfSquares(double x,double y)
{ return sum(square(x),square(y)); }
double result=sumOfSquares(4,sum(3,2));
……
```

    - 雖然上述的程式可以完成兩數的平方和。但一般我們不會用
      result=
      sum(sum(sum(square(1),square(2)),square(3)),square(4));
      的方式去完成一個$1^2+2^2+3^2+4^2$的敘述（除非是故意的）。

24

1.2 程式語言簡介

- 結構化程式語言(Structural Programming Language)
  - 更常見的作法是將之寫成下列的迴圈敘述：

```
double result=0;
int i;
for(i=1;i<=4;i++)
   result = sum(result,square(i));
```

  - 加入迴圈的構想已經代表著設計者在構思演算流程了。
  - 一旦牽涉到迴圈運算，在平行化效率方面就有一段差距

25

1.2 程式語言簡介

- 結構化程式語言(Structural Programming Language)
  - 例如有一個LISP-like的函數式程式語言，可以接受樹狀的運算式，也就是((3+5)+(7*8))+((2+4)*(1*9))為一個運算樹的中序表示法，也為該語言的敘述，則其編譯器或直譯器就可充分利用CPU的多核心自動進行平行處理
    - 例如將(3+5)、(7*8)、(2+4)、(1*9)分散到四個核心去計算，以便加快執行速度。
  - 非函數式的程式語言若想要撰寫平行處理程式，
    - 程式設計師必須先思考各執行緒完成的先後順序，自行撰寫平行處理程式。

26

1.2 程式語言簡介

- 物件導向程式語言（Object-Oriented Programming Language；簡稱OOPL）
  - 結構化程式設計卻在發展大型軟體時遭遇到許多問題。
  - 多個模組可能存取同一個資料結構，這將導致模組之間的獨立性質降低
    - 例如，某兩個模組可能都會修改陣列資料結構中的元素，因此當我們希望將某一個模組的操控對象從「陣列」改為「鏈結串列」時，另外一個模組也必須跟著改變。
  - 結構化程式設計雖然可以快速尋得解決方案，但對於日後的維護則顯得不足，故後來又發展了以物件導向為基礎的程式語言。

27

1.2 程式語言簡介

- 物件導向程式語言（OOPL）
  - **物件導向程式設計**以**物件**為出發點，藉由物件與物件之間的互動完成問題的解答，比較符合真實世界環境。
    - 由於每一個物件都是一個獨立的個體，因此更動某一物件的內容時，其他的物件並不需要跟隨著變動。
    - 在尋求大型或超大型方案的解答時，比結構化程式設計容易分析問題，更有助於日後的維護與修改。
    - **物件導向型態**的程式語言最早起源於1961~1967年所發展的Simula I和Simula 67。爾後則有SmallTalk、C++、Object Pascal、Java、Visual Basic.NET、C#等等語言

28

1.3 Java與物件導向

- Java是一種純物件導向程式的程式語言
  - 物件導向的特性
  - Java的發展歷史。

29

1.3.1 物件導向程式設計

- **物件導向程式設計（Object-Oriented Programming；簡稱OOP）**
  - 是一種以物件觀念來設計程式的方法。
    - 在純物件導向的程式中，每一個運作實體都可視為某種類別衍生出來的物件
    - 而類別則較具抽象的概念，也就是某些具有共同特性物件的集合。
      - 換句話說，物件就是類別的實體。
    - 每個物件都擁有某些特性(attribute)與行為(behavior)，在物件導向程式設計中則稱之屬性(property)與方法(method)。

30

1.3.1 物件導向程式設計

- 物件導向具有封裝、繼承、多型等特性
- 原則上每個物件相互獨立且無關聯性
- 物件導向程式設計是依照物件的方法產生互動以完成要求。
  - 而物件之間的互動是透過訊息(Message)的傳遞來達成
  - 發送訊息的物件稱之為發送者(sender)
  - 接受訊息的物件稱之為接收者(receiver)
  - 當物件需要其他物件服務時，會發送訊息給接收者
  - 接收者在收到訊息後，會執行符合要求的方法完成任務以提供服務。

31

1.3.1 物件導向程式設計

- 物件（Object）
  - 真實世界中的所有具體或抽象的事物，都可以將之視為一個『物件』。
  - Java的物件是一些程式碼和資料的組合
  - 物件必須能夠單獨成為一個完整單元，也可以組合成更大的物件。

32

1.3.1 物件導向程式設計

- 屬性（Property）
  - 物件擁有許多特性，這些特性代表了一個物件的外觀或某些性質。
  - 這些特性在物件導向程式設計中稱之為**屬性(Property)**
  - 事實上，有的時候在取得物件的某些屬性時，所得到的也可能是另一個（子）物件（若該屬性的資料型態是一個類別）。
  - 在程式設計或執行階段，我們可以藉由改變屬性值來改變整個物件的某些特性，完成我們想要的物件表示形式。
    - 例如：將民航機物件的機殼漆成藍色，將按鈕背景顏色設為黃色。

33

1.3.1 物件導向程式設計

- 屬性（Property）
  - 在Java語言中，**屬性(Property)**被稱為欄位，並被區分為不同的保護等級
    - 某些等級的欄位並不允許外部程式加以修改，而必須透過該物件的方法，才能夠修改這些欄位，因此具備保護資料的功能。

34

1.3.1 物件導向程式設計

- **方法（Method）**
  - 每個物件都擁有不同數量的行為，這些行為在物件導向程式設計中稱之為**方法(Method)**。
    - 同類物件擁有的方法大致相同。
  - 為了完成該物件某些目標的處理方式。
    - 例如：飛機類別的物件有許多方法，如起飛、降落、逃生等等。每個方法都有許多的細節
    - 例如起飛，可能包含『發動引擎、、、、直到拉動操縱桿』，這些就是起飛方法的細節。

35

1.3.1 物件導向程式設計

- **方法（Method）**
  - 我們也將**方法**稱之為函式（但隸屬於特定類別）
    - 這是為了C/C++語言愛好者學習Java之便。
  - Java的方法也被區分為多種等級
    - 某些等級的方法不允許外部程式呼叫執行（只能做為內部方法呼叫執行的對象），因此達到封裝物件的功能。
  - 由他人完成的物件同樣必須提供關於物件的方法
    - 如此，我們就不需要了解這些方法的細節，而能夠快速運用物件來完成工作。
    - 例如飛機物件提供了起飛方法，我們只要指定執行起飛方法，而不需要了解起飛過程的細節，飛機就會起飛。

36

1.3.1 物件導向程式設計

- 【註】
某些物件導向程式語言的屬性與本處所提之屬性有些不同，這裡所提的屬性相當於Java和VB.NET的欄位，而VB.NET的屬性則為方法與欄位的綜合體，亦即使用特性名稱的方法（如Set、Get）存取特定的欄位。

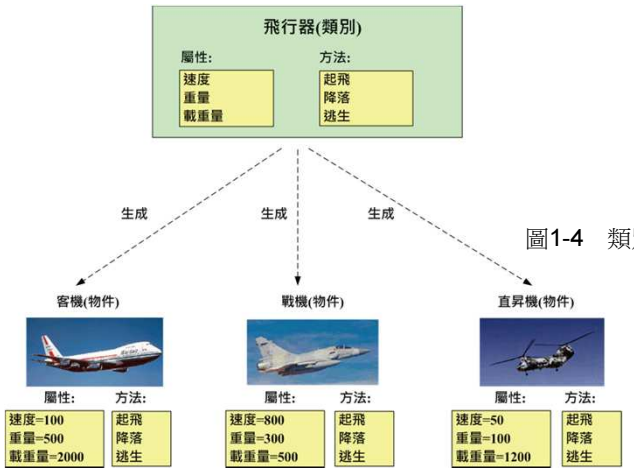


圖1-4 類別與物件

37

1.3.1 物件導向程式設計

- 類別(Class)
  - 不同的物件可以擁有相同的屬性及方法
    - 例如：「一架民航機」和「一架戰鬥機」各是一個物件，但兩者的速度、爬升力、重量、載重量雖不相同，但卻同樣擁有這些屬性以便表達完整物件的各個特性。
    - 所以，不同的物件若擁有共同的屬性，但因為屬性內容的不同，因此可以創造出同類性質但卻獨立的不同物件
  - 而同類型的物件，則構成了**類別(Class)**。
  - **類別才是物件導向程式設計最基本的單元**。
    - 不同的只是在建立該類別的實體物件時，給予不同的屬性而已。

38

1.3.1 物件導向程式設計

- 類別(Class)
  - 在實際的物件導向程式設計中，我們必須先定義類別，然後才能夠透過類別宣告各個屬於該類別下的物件。
  - 在純物件導向的程式中，程式碼是以類別為基礎，物件只是類別的實體
    - 就如同汽車是一個類別，它是一個抽象的名詞；而車牌號碼為010、015等的車輛是汽車類別的兩個物件，實際運作時通常依靠的是物件。

39

1.3.1 物件導向程式設計

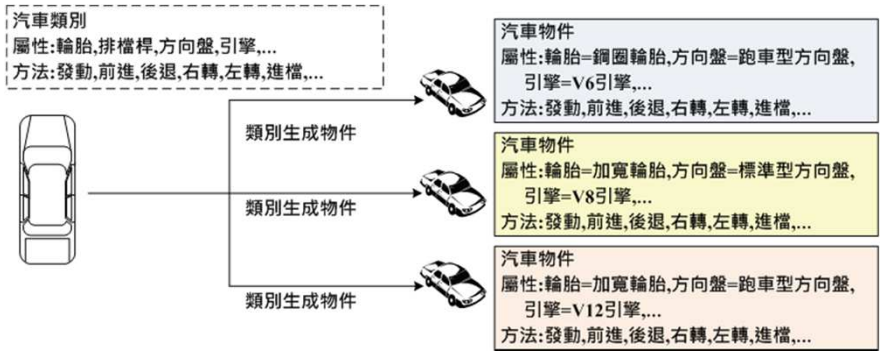


圖1-5 類別生成物件

40

1.3.2 物件導向程式設計的特點

- 封裝性(encapsulation)：
  - 封裝性可以將物件區分為可被外界使用的特性及受保護的內部特性
    - 除非是允許外部程式存取的資料，否則外部程式無法改變物件內的資料。如此就能夠達到封裝保護資料的特性
  - 更細分來說，物件導向程式設計將物件的資料與方法「至少」區分為三種等級：
    - public ：公用等級
    - private ：私用等級
    - protected ：保護等級
      - （註：Java的封裝等級還需要考量到package的影響，留待第11章說明）

41

1.3.2 物件導向程式設計的特點

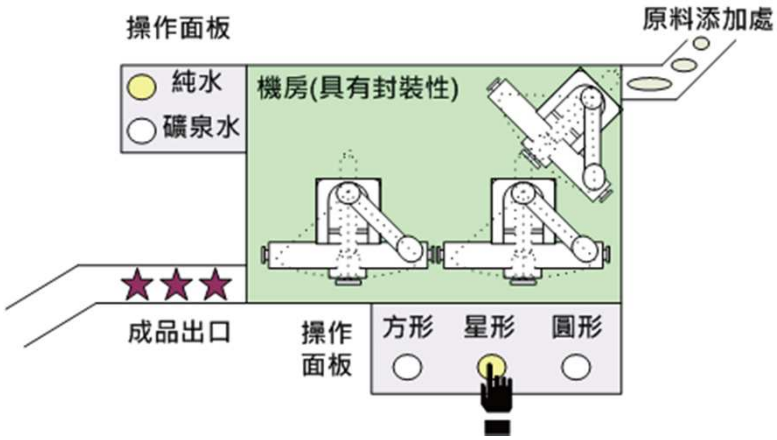


圖1-6　封裝性

42

1.3.2 物件導向程式設計的特點

機房物件

公開屬性：原料、成品
私有屬性：水……
公開方法：使用純水、使用礦泉水、方形餅乾製作、星形餅乾製作、圓形餅乾製作
私有方法：機器A啟動、機器B啟動、機器A快轉……

- 以圖1-9為例
  - 只開放原料與成品兩個可存取的屬性
  - 提供製作方形餅乾、製作星形餅乾、製作圓形餅乾等操作方法的按鈕。
  - 提供使用純水、使用礦泉水等操作方法的按鈕。
  - 被封裝的屬性(水)是不可存取的，除非透過外部方法間接存取
    - 無法隨意將水變更為海洋深層水

43

1.3.2 物件導向程式設計的特點

- 封裝性使得問題變為分層負責的狀態，更符合生活上的作事原則。
  - 設計物件者有責任將物件設計的完整且不會出錯，也需要留下一些對外可操作的方法，必須時也可留下一些外部可存取的屬性。
  - 而使用物件者，則只能夠透過物件對外的屬性與方法來利用物件達到自己的需求。

小試身手1-1
「電子鬧鐘」物件具有封裝性，試以電子鬧鐘為例，說明電子鬧鐘的哪一些屬性為公開屬性？哪一些為私有屬性？哪一些為公開方法？哪一些為私有方法？

44

1.3.2 物件導向程式設計的特點

- 繼承性(inheritance)：
  - 繼承性是為了達成重覆使用的目的所採取的一種策略。
    - 在物件導向程式設計中，允許使用者定義**基底類別**(base class)與**衍生類別**(derived class)，其中衍生類別允許**繼承**基底類別的屬性及方法，並「加入新的屬性及方法」或者改寫(override)某些繼承的方法，改成適用於本身的方法。
  - 有了這項特性，在開發大型程式時，我們就可以延續已經完成的技術，再加擴充。

45

1.3.2 物件導向程式設計的特點

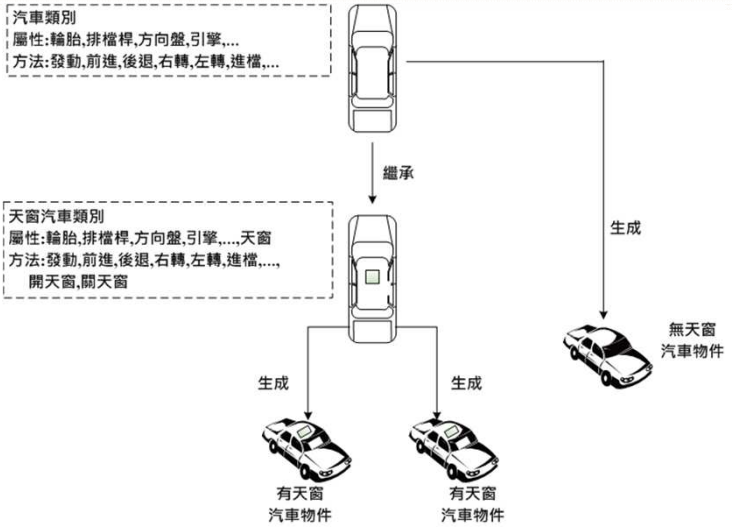


圖1-7　繼承性

46

1.3.2 物件導向程式設計的特點

- 多型性(polymophism)：
  - 就多型性而言，其實它是一個非常抽象的名詞，代表著一種彈性。
    - 舉例來說，所謂開水人人會煮，各有巧妙不同，基本上，瓦斯爐、電磁爐、熱水瓶都可以煮開水
    - 所以『煮開水』其實是一個相當抽象的名詞
    - 使用瓦斯爐、電磁爐、熱水瓶的煮開水方式都不一樣
  - 但『煮開水』一詞卻涵蓋了這些差異性的行為，而此種特質則稱為「多型性」。

47

1.3.2 物件導向程式設計的特點

- 多型的實作方法有很多種，其中一種是改寫（override；或稱覆寫），它與繼承有關
  - 例如瓦斯爐衍生類別、電磁爐衍生類別都繼承自加熱器基底類別，該基底類別中定義了加熱的方法
  - 我們則必須在衍生類別內，改寫這個方法，使得加熱的細節有所不同
- 如此一來，由不同衍生類別所產生的物件，執行方法時的細節就會不同了。

48

1.3.2 物件導向程式設計的特點

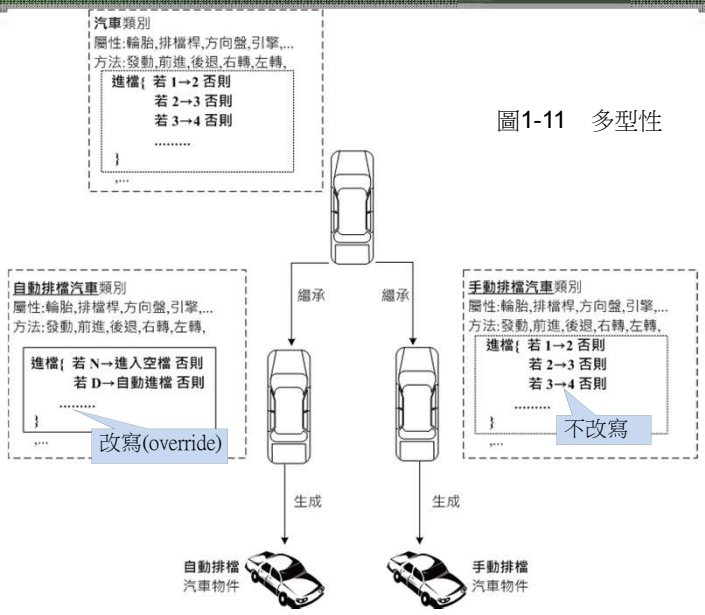


圖1-11　多型性

49

1.3.3　Java簡介

- Java
  - 是1991年，由Sun公司的James Gosling根據C++的物件導向理念而發展的另一種物件導向語言。
  - 基本語法與C++的語法相似，但Java是一個更純正的物件導向程式語言，擁有物件導向程式語言所有的特性。
  - 原名是Oak，用在消費性電子產品的跨平台整合，但在1993年，網際網路蓬勃發展的時代被大量使用，而於1994年時重新命名為Java，並在1995年由Sun正式發表。

50

1.3.3 Java簡介

Java大事記

年代	大事
1991	Green計畫，由James Gosling領導執行，推出Java的前身Oak。
1995	Sun正式命名為Java，並推出Java 1.0 Netscape決定將Java加入Netscape 2.0瀏覽器中
1996	JDK(Java Development Toolkit) 1.0版公佈 Netscape公佈 Java-enable Netscape 2.0瀏覽器
1997	JDK 1.1版公佈
1998	JDK 1.2版公佈，並更名為Java2
1998	Java2企業平台J2EE發布。
1999	Sun發布Java的三個版本：標準版(J2SE)、企業版(J2EE)和微型版(J2ME)

51

1.3.3 Java簡介

Java大事記(續)

表 1-3 Java大事紀

年代	大事
2000	JDK1.3公佈
2000	JDK1.4公佈
2001	J2EE1.3公佈
2002	J2SE1.4公佈
2004	J2SE1.5公佈(JDK1.5)
2005	由於Java2已經問世一段時間，因此Sun也取消了中間的「2」，故自此之後，J2EE更名為Java EE，J2SE更名為Java SE，J2ME更名為Java ME，所以J2SE 1.5稱為Java SE 5.0版。（JDK 1.5更名為JDK5）
2006	Java SE 6.0版公佈（即JDK6；也就是JDK 1.6版）。
2009	Sun被Oracle收購
2011	7月7號，Java SE 7.0版公佈（即JDK7；也就是JDK 1.7版）。
2014	3月18日，Oracle發表Java SE 8。

52

1.3.3 Java簡介

- 由表1-3的大事紀中，我們可以得知Java SE 8.0 版（不過目前許多人都簡稱為Java 8）指的是JDK8。
- 而什麼又是JDK呢？
  - JDK的全文是Java Development Toolkit
  - 顧名思義，這是一套Java的發展工具套件。

53

1.3.4 JDK與JVM簡介

- 介紹JDK之前，首先我們必須釐清Java究竟是使用編譯器還是直譯器來翻譯程式？
  - 答案是「一半一半」。
  - 傳統編譯器的處理流程如圖1-9，編譯器會將原始程式經過語法和語意的分析後，將之轉為中間碼(IR)，而中間碼的格式與組合語言已經非常接近。
    - 後面的步驟則在進行最佳化以及產生目的碼。
  - 在產生中間碼之前的步驟都是屬於與機器無關的處理程序
    - 這也使得高階語言具有可攜性。

54

1.3.4 JDK與JVM簡介

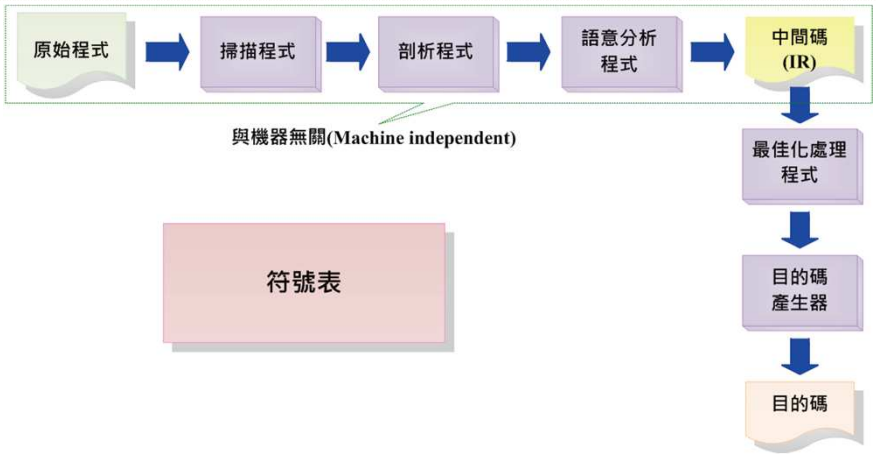


圖1-9 傳統編譯器的處理流程

55

1.3.4 JDK與JVM簡介

- 常見的中間碼有幾種表達方式
  - 包含後置式、三位址碼、P-code、語法樹等等。
  - 其中的P-code是一種可以在虛擬堆疊機器(Virtual Stack Machine；VSM)上所執行的程式碼。
- 至於直譯器的處理程序則如圖1-3所示
  - 它會讀入原始程式，一行一行的解釋並執行，所以速度比較慢。

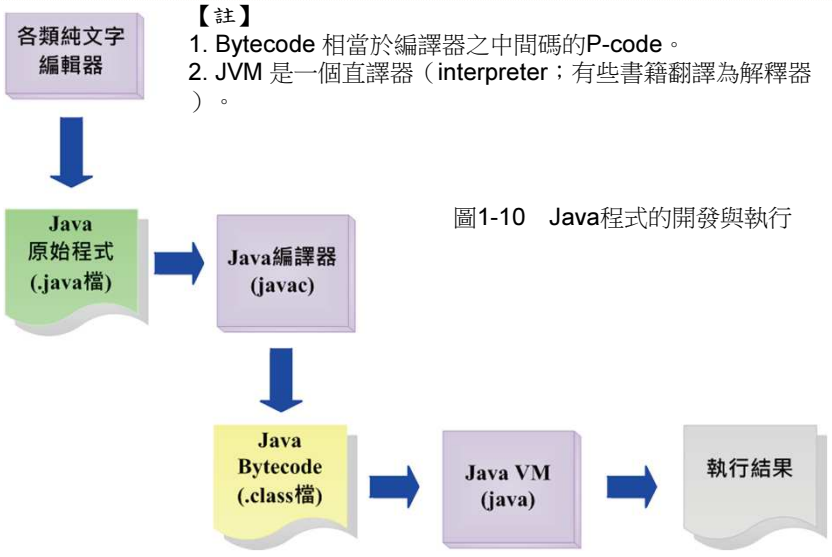
56

1.3.4 JDK與JVM簡介

- Java語言和其他傳統高階語言的處理程序都不相同
  - 為了要達到絕對的跨平台，採用了編譯與直譯混合的模式。
  - 取出編譯器中與機器無關(machine independent)的處理程序
    - 前半段→編譯
  - 後半段則交由JVM直譯器來執行。
    - 後半段→直譯
    - Java程式會經過編譯器(compiler)編譯為**位元組碼(Bytecode)**
    - 而"Bytecode"可以由Java Virtual Machine（簡稱Java VM或JVM）來執行。

57

1.3.4 JDK與JVM簡介



【註】
1. Bytecode 相當於編譯器之中間碼的P-code。
2. JVM 是一個直譯器（interpreter；有些書籍翻譯為解釋器）。

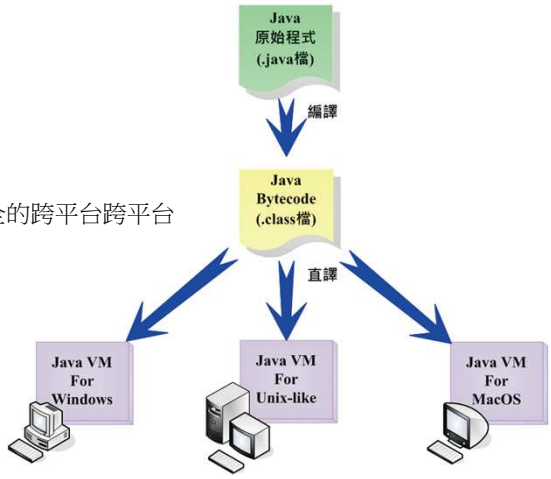
圖1-10 Java程式的開發與執行

58

1.3.4 JDK與JVM簡介

- Oracle提供了各平台所需的JVM，用來直譯"Bytecode"，（如下圖）
  - 因此，Java做到了絕對的跨平台

圖1-11 Java 能夠達到完全的跨平台跨平台



59

1.3.4 JDK與JVM簡介

- 由圖1-10中，我們可以得知，要使用Java撰寫並執行程式，必須擁有編輯器、Java編譯器、JVM等三種軟體。
  - 現今所有的作業系統都提供了一些基本的編輯器，例如Unix/Linux的Vim、Windows的記事本。
  - 而Java編譯器與JVM則需要由Oracle來提供，我們可以至下列網站下載相關軟體。
    - http://www.oracle.com/technetwork/java/javase/downloads/index.html

60

1.3.4 JDK與JVM簡介

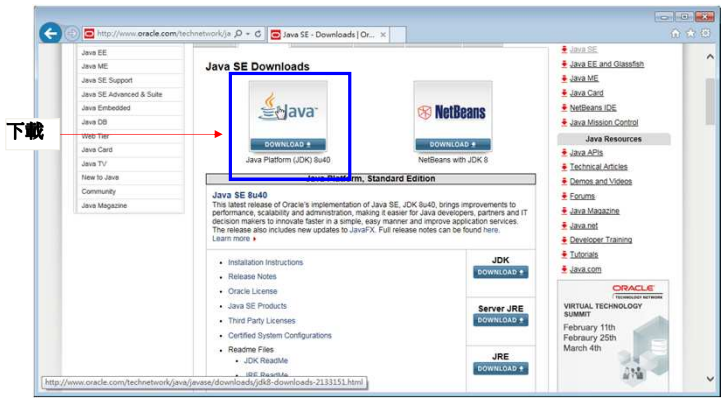


圖1-12 http://www.oracle.com/ 網站

61

1.3.4 JDK與JVM簡介

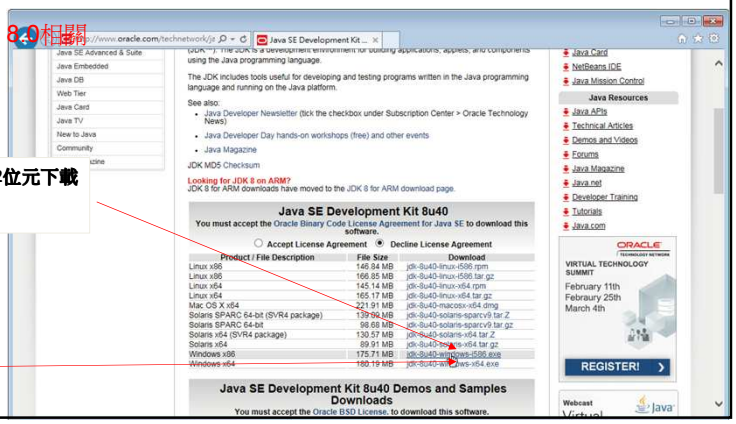
- 在上述網站的左側，我們可以發現Java有三種套餐可供選擇，分別是Java SE、Java EE與Java ME其特色如下：
  - **Java SE**：Java Standard Edition的縮寫，也就是標準版，它提供了開發一般程式的套件與平台。
    - 本書所使用的軟體版本為Java SE 8.0版。
  - **Java EE**：EE：Java Enterprise Edition的縮寫，也就是企業版，提供企業伺服器級的套件與平台，例如發展JSP動態網頁就需要使用這個版本。
    - 通常Java EE版本更新的速度會晚於Java SE。
  - **Java ME**：Java Micro Edition的縮寫，也就是微型版，提供微型設備的開發套件與平台。

62

1.3.4 JDK與JVM簡介

- 本書版本為Java SE 8.0
- 您可以按下上圖中左方的Java Platform(JDK) 8uXX ，就可以依照機器種類下載相關套件，如下圖。

圖1-13 Java SE 8.0相關下載套件



1.3.4 JDK與JVM簡介

- JDK8中又包含了一些其他的軟體，其用途解釋如下：（安裝詳見附錄A）
  - JDK 8：JDK是Java Development Kit的縮寫，也就是開發Java程式所需要的必備工具，它包含了編譯器、JRE與JVM。JDK是我們主要的開發工具，請下載此套件。
  - NetBeans with JDK8：在圖1-12的右方有一個NetBeans圖形超鏈結，按下後可選擇下載JavaSE and NetBeans Cobundle，其中NetBeans with JDK8是一個整合開發環境(IDE)【見延伸學習】，它包含了JDK。
  - JRE：Server JRE 與 JRE。JRE是Java Runtime Environment的縮寫，它包含了主要的API（即Java Runtime Classes）以及JVM。若只是要執行Java程式而不開發Java程式，可選擇只安裝JRE元件。
  - Java SE 8 Documentation：這是用來說明Java SE的技術文件，可下載當作參考手冊。

64

1.3.4 JDK與JVM簡介

- 對於一般使用者而言，若能取得編譯過的程式（即類別檔），則只要安裝JRE來執行即可。
- 對於Java程式設計師來說，則至少必須安裝JDK或各種IDE，例如NetBeans with JDK8 。
  - 其關係圖如圖1-14所示

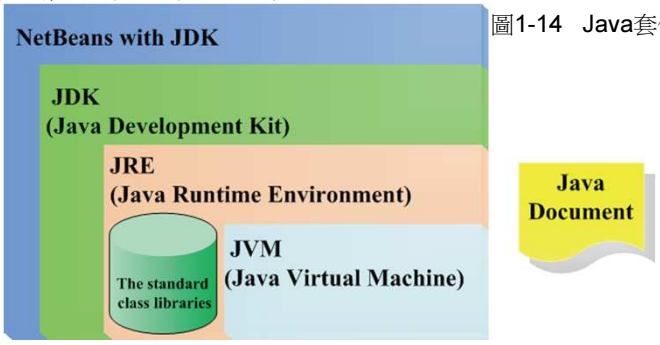


圖1-14 Java套件的關係圖

65

1.3.4 JDK與JVM簡介

延伸學習：IDE

傳統的編譯程式開發流程有些繁複，程式設計師必須先用文書編輯器撰寫原始程式檔，然後再交由編譯器、連結器將原始程式檔編譯為具有除錯功能的可執行檔，最後經過除錯無誤後，才重新編譯連結為最終的可執行檔。**不過這個流程通常不會一次就完成，因為程式可能發生語法、語意等錯誤**，而必須在編輯器、編譯器、除錯器間切換許多次，才能夠得到最後正確的原始程式與執行檔

整合開發環境（Integrated Development Environment；簡稱**IDE**）是一套整合性軟體，**它將程式發展所需要的各類軟體整合在同一套軟體之內，以方便程式設計師開發程式之用**。例如對於Java 語言的IDE而言，它至少必須包含，編輯器、Java編譯器、類別庫、JVM等等。而目前大多數的視窗整合開發環境除了提供這些軟體功能之外，也提供了許多的視窗套件供程式設計師使用。

Java常見的IDE有Oracle官方建議的NetBeans與最普及的**Eclipse**。開發Android App時，還可以選擇**Android Studio IDE**等專門為了開發App而設計的IDE。

66

1.3.4 JDK與JVM簡介

延伸學習：API

API是Application Programming Interface的縮寫。

由於軟體程式日漸龐大，要從無到有完整開發一個頗具規模的程式幾乎是不可能的任務。因此，軟體廠商們發展了許多好用的函式庫或**類別庫**供程式設計師使用，這些函式庫或類別庫有時統稱為API，例如微軟提供了Win32 API，可用來發展Windows視窗程式。在Java領域中，Oracle在JRE當中也提供了標準類別庫，我們將在本書中大量使用這些類別庫所提供的類別來設計程式。

67

1.3.5 Java的特色

- Java的特色
  - (1)Java可快速開發各類應用程式：
    - Java是一個物件導向語言，由大廠支援，類別庫完整且橫跨眾多應用領域，包含視窗程式、資料庫應用、網路應用、Web應用、智慧型手機Android作業系統應用等等
  - (2)Java易學：
    - 對於物件導向程式語言來說，Java屬於比較容易學習的一種程式語言，它去除了C++的多重繼承，使得學習上較為容易。
    - 尤其是加入了lambda運算式之後，也能如函數式語言般更容易也更直覺地進行程式設計。

68

1.3.5 Java的特色

- (3)Java可完全跨平台：
  - Java利用虛擬機器的觀念，使得能在各平台上執行Java程式（因為實際上是在JVM內執行）。
  - Java編譯器會將原始程式碼編譯為Bytecode，而各平台都有適用的JVM可以執行Bytecode，更換平台不但不必重新撰寫程式，甚至也不必重新編譯程式。
- (4) JVM可跨語言：
  - 現在有些編譯器可以將其他程式語言原始碼編譯為Bytecode，甚至還支援動態型別程式語言（如JavaScript或Ruby），這使得Java具備了未來的發展性。

69

1.3.5 Java的特色

- (5)Java是結構獨立的：
  - 這和跨平台有極類似之處，但此處強調的是Java面對的是虛擬機器，因此可以假定Java程式要執行的平台永遠是相同的，也就是Java Virtual Machine。而上述跨平台所強調的是Sun必定會提供各類平台所需的JVM。
  - 這和跨平台有極類似之處，但此處強調的是Java面對的是虛擬機器，因此可以假定Java程式要執行的平台永遠是相同的，也就是Java Virtual Machine。
  - 而上述跨平台所強調的是Oracle必定會提供各類平台所需的JVM。
    - 由這兩點，因此，早期Sun提出一個保證的口號，Write once, run anywhere （簡稱 WORA），也就是只要撰寫一次，就可以在各平台上執行。

70

1.3.5 Java的特色

- (6)Java比較安全：
  - Java去除了C/C++語言的指標（Java不允許程式設計師直接使用指標），因此降低了記憶體存取錯誤的機會。
  - 同時，Java在JVM中，也加入了許多的安全機制，以保證安全地在沙箱(Sandbox)中執行Bytecode，其步驟如圖1-15所示。

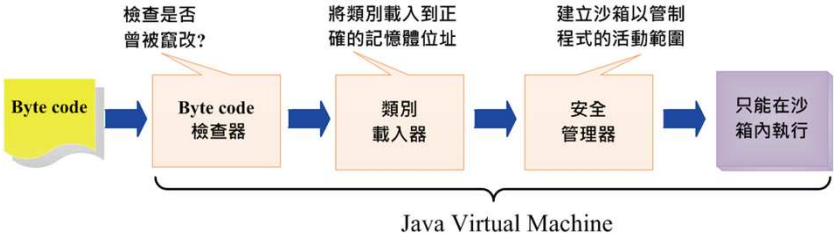


圖1-15 JVM的細節

71

1.3.5 Java的特色

- (7)Java軟體容易發佈：
  - 當客戶端未安裝JVM而想要執行Java程式時，會主動要求下載JVM來執行程式，下載並安裝完成後，日後會自動檢查軟體的更新。
- (8)Java是一種極佳的教學程式語言：
  - Java不但簡單，並且也是純物件導向語言，因此有助於讓學生從物件的角度來設計程式，並且又提供了眾多的API，足以在教學上取代C++、Object Pascal等傳統物件導向語言。
- (9)Java速度快又慢：
  - 使用Java開發的應用程式，整體而言，執行速度比C++還要慢。但若只和使用直譯器的程式語言比較，Java算是快的
  - Java支援多執行緒，可透過多工方式，加速工作的完成
  - Oracle在JVM與編譯器上很用心。
  - Java SE8在平行處理方面提供了更多的開發工具，可開發出效率極佳的應用程式。

72

1.3.5 Java的特色

- (10)Java能夠充分利用記憶體：
  - Java雖然需要啟動JVM來執行程式，但它也支援動態載入機制，可以在需要某個物件時，才將物件載入，因此JVM所使用的記憶體能夠有效的被利用。
  - 此外，Java還支援記憶體回收管理機制可收回成為垃圾的記憶體，並再度利用這些記憶體。
- (11)Java是穩固的(robust)：
  - Java支援例外機制，使得程式設計師可以針對各種例外狀況加以處理，以求交付更穩固的應用程式給客戶。

73

1.3.6 Java的相關工具

- Java包含了三大部分
  - 亦Java語言本身、JVM以及相關的類別庫。
  - 如果我們要開發Java應用程式，至少須下載並安裝JDK（安裝NetBeans之類的IDE也可以）
    - 因為JDK包含了最重要的Java編譯器、各層級的API以及執行用的JVM，如下圖。

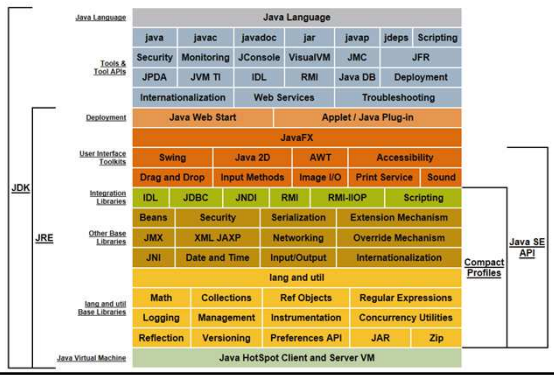


圖1-16 JDK已包含所有必須的工具
《圖片取自Java Docs》

74

1.3.6 Java的相關工具

- 由上圖的第二層中，我們可以發現到許多的工具，我們列舉幾個重要工具的功能，說明如下：
  - javac：
    - 這是Java的編譯器，可將.java原始檔編譯為.class類別檔。
  - java：
    - 這是JVM虛擬機器的執行檔，可用來執行Java Application的Bytecode（也就是.class類別檔）
    - 【註】本書介紹的Java程式可以分為兩大類，本文的所有Java程式皆為Java Application，而電子附錄介紹的程式則為可在瀏覽器中執行的Java Applet 。

75

1.3.6 Java的相關工具

- javadoc：
  - 這個工具可以將Java的特殊註解輸出為HTML格式（類似Java document格式） 。
- jar：
  - 可將.class檔包裝成壓縮檔的工具。
- javap：
  - 可反組譯.class檔的工具。
- 【註】下載與安裝JDK，請見附錄A。

76

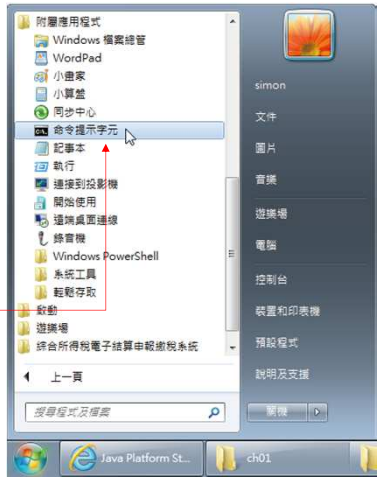
1.4 編譯與執行範例

- 在本節中，我們將直接編譯一個Java程式並執行它
  - 您不需要了解該程式，程式設計的細節將在後面章節中陸續介紹。
  - 在此，**請您先按照附錄A.1~A.3的指示，安裝JDK等相關工具並設定工作環境後**，按照下列範例進行編譯與執行。
- 【實作範例1-1】編譯與執行一個Java程式。
  - Step0：將光碟的範例檔ch1_01.java，預備在C:\myJava\ch01\中。
    - 執行書附光碟的myJava.exe解壓縮檔即可

77

1.4 編譯與執行範例

- Step1：開啟Dos視窗。執行【開始／所有程式／附屬應用程式／命令提示字元】指令。

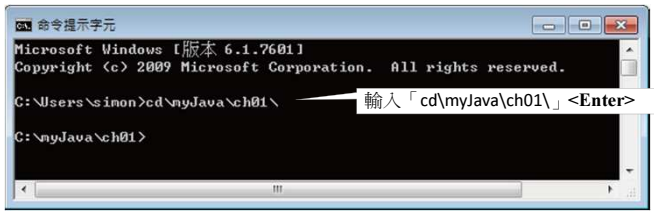


執行

78

1.4 編譯與執行範例

- Step2：切換到C:\myJava\ch01\。
  - 假設您位於其他磁碟槽，請先輸入「C:\」<Enter>
  - 若已位於C槽，請直接輸入「cd\myJava\ch01\」<Enter>。



- Step3：執行**編譯**，輸入「javac ch1_01.java」<Enter> 。

79

1.4 編譯與執行範例

- Step4：此時會多出一些類別檔，當中主類別檔之名稱為ch1_01.class。您可使用dir指令觀察，結果如下。

- Step5：開啟另一個Dos視窗。執行【開始／所有程式／附屬應用程式／命令提示字元】指令。（同Step1）

80

1.4 編譯與執行範例

- Step6：切換到C槽根目錄。假設您位於其他磁碟槽，請先輸入「C:\」 <Enter> 。若已位於C槽，請直接輸入「cd\」 <Enter> 。

- Step7：**請JVM執行程式**，輸入java myJava.ch01.ch1_01（後面不加副檔名） <Enter> 。

81

1.4 編譯與執行範例

- Step8：此時會出現一個視窗
  - 我們將在後面章節學習如何製作此視窗程式，目前您只要學會如何編譯與執行程式即可。
  - 最後請按視窗右上角的關閉鈕關閉視窗。

82

1.4 編譯與執行範例

- 【說明】
  - 1.在這個範例的Step1與Step5，我們分別開啟兩個Dos視窗，一個用來編譯，一個用來執行類別檔
  - 2.使用java執行程式時，後面不加上「.class」副檔名，因為java本身已經預設目標檔為類別檔
  - 3.使用java執行程式時，前方的「myJava.ch01」代表類別庫名稱，Java規定使用目錄層級作為類別庫分類，並且將目錄分隔符號「\」改為「.」

83

本章結束

Q&A討論時間

84