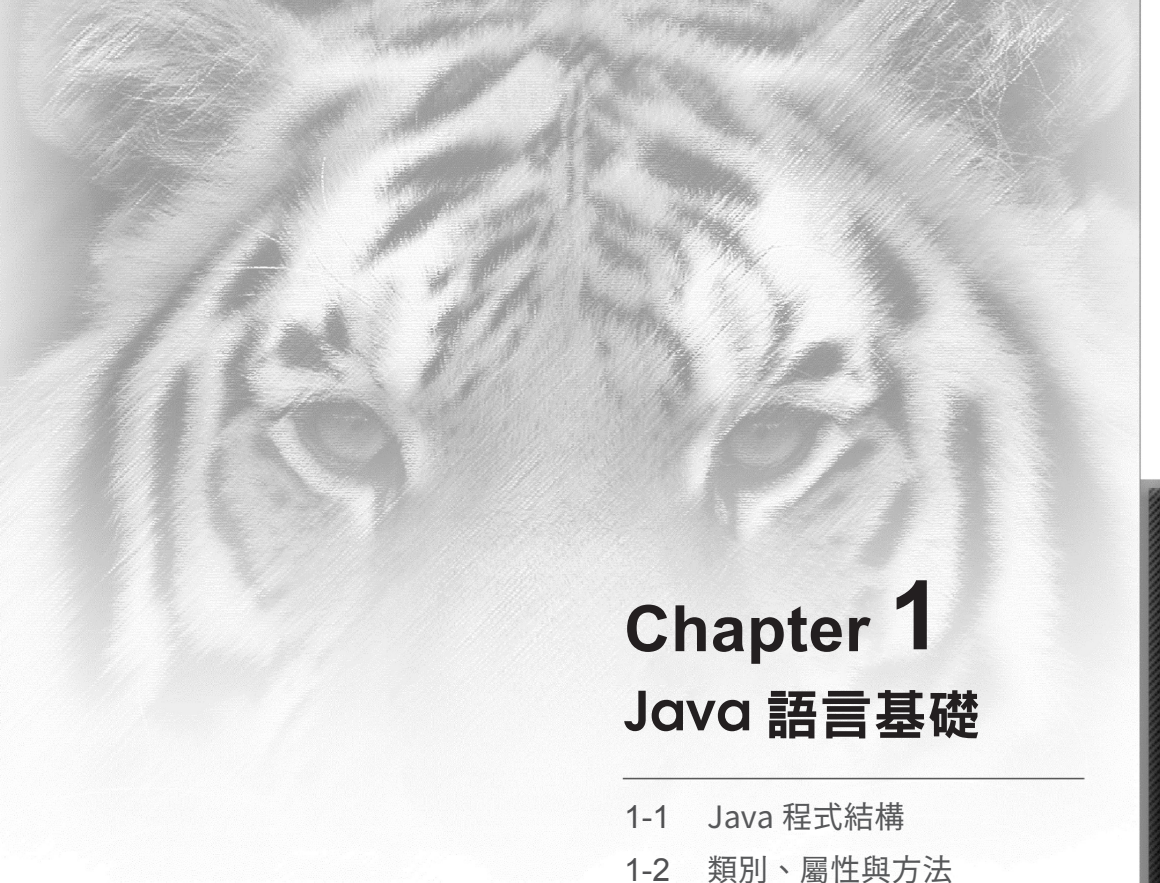
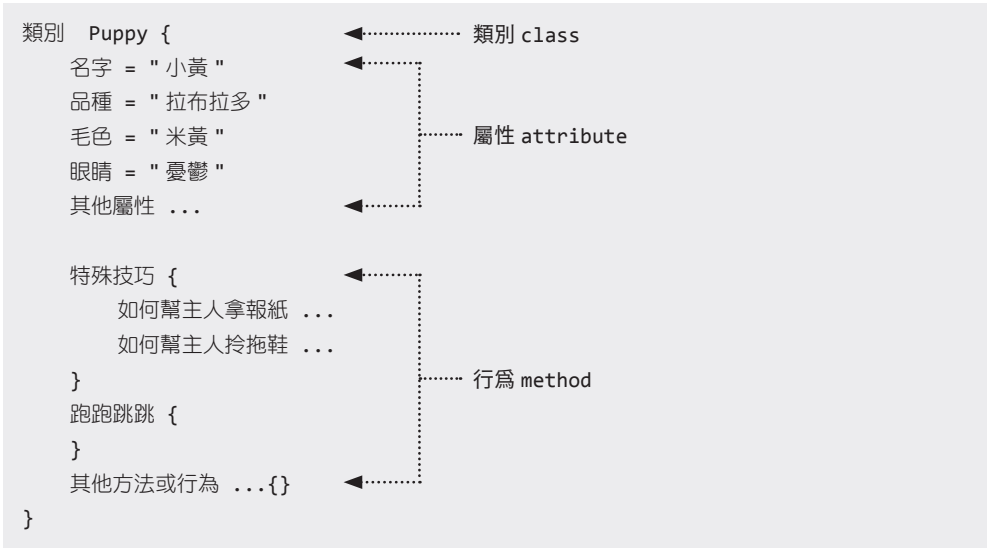


第二部分

Chapter 1
Java 語言基礎

1-1 Java 程式結構

Java 是物件導向的語言，我們所生活的環境中其實有許多「物件導向」的實例。當我們要描述一個東西時，勢必會敘述它的屬性與行爲。例如，我要描述一隻可愛的小狗，我會說：「有一隻狗名叫小黃，牠的品種是拉布拉多，牠身上的毛是米黃色的，牠的眼睛看起來很憂鬱，眞是可愛，牠還會幫主人拿報紙、拎拖鞋，眞是貼心」。將其抽象化後，我們可以將小狗設定成一個名叫 Puppy 的**類別**，牠的**屬性**便是狗的名字、品種與毛色等，牠具有幫主人拿報紙、拎拖鞋的**行爲**。這個 Puppy 類別下所包含的屬性和方法就可以幻化成 Java 語言的雛型，請參考下列的程式片段。



由此得知，類別下面會有二種成員分別爲屬性（也可稱爲變數）與方法。上述的例子若轉換成合法的 Java 語法，便可寫成如下的程式片段：

```
class Puppy {
    static String dogType = " 拉布拉多 ";
    String dogName = " 小黃 ";
    String dogColor = " 米黃 ";
```

```
    void skill() {
        String skill_1 = "拿報紙";
        String skill_2 = "拎拖鞋";
        System.out.println("幫主人"+ skill_1);
        System.out.println("幫主人"+ skill_2);
    }
    static void move() {
    }
}
```

以下爲一個完整的 Java 程式結構與對應名稱：

```
class MyClass {
    類別屬性、物件屬性
    類別方法 ()、物件方法 ()、建構子 ()
}
```

建構子也是一種方法，在方法內所宣告的變數稱之爲區域變數。

```
class Puppy {
    static String dogType = "拉布拉多";      ◄···· 類別 class
    String dogName = "小黃";                 ◄····
    String dogColor = "米黃";                ◄···· 物件變數
    void skill() {                           ◄···· 物件方法
        String skill_1 = "拿報紙";           ◄····
        String skill_2 = "拎拖鞋";           ◄···· 區域變數
        System.out.println("幫主人"+ skill_1);
        System.out.println("幫主人"+ skill_2);
    }
    static void move() {                     ◄···· 類別方法
    }
    Puppy() {                                ◄···· 建構子
    }
}
```

類別的成員中若加入 **static** 修飾字就會變成專屬該類別所有的屬性與方法，也就是「類別成員（static member）」。沒加上 static 修飾字的成員則爲物件所有的屬性與方法，稱爲「物件成員（non-static/instance member）」。建構子（constructor）是屬於物件成員，不可以加上 static 修飾字。

要撰寫一個 Java 程式必須從 class 來著手，依照 Java 語言的特性您可以先撰寫好許多實用的 class（零件），接下來再透過一個主程式及某一特定的 business logic（流程）將其拼裝成一個應用程式，藉由不同的流程又可以拼裝成更多的應用程式，這就是物件導向語言的彈性與擴充性。往後的章節當中，您將逐漸體會物件導向語言的強大功能與彈性機制。

程式進入點

Java 中的程式進入點是由一個特定的類別方法來負責執行：

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

該方法名稱爲 main，其方法的存取權限爲 public（有關「存取權限」請參考 1-5 節），static 屬於類別的方法，void 表示是一個沒有回傳值的方法，方法參數列固定放入 String[] 陣列且只能有一個。

1-2 類別、屬性與方法

1-2-1 類別

什麼是類別？類別（class）是物件的藍圖，亦即物件的基礎，可用來描述類別或物件內所包含的資料（attribute，屬性），以及該類別或物件可被操作的行爲（method，方法）。

宣告方式

```
[ 存取修飾字 ] + 宣告類別 + 類別名稱 + { 類別的內容與本體 }
```

語法範例

```
public class HelloWorld { ... }
```

1-2-2 方法

方法（method）爲類別與物件提供外界存取和呼叫的服務。

宣告方式

```
[ 存取修飾字 ] + [static] + 回傳值 + 方法名稱 ( 參數列 ) + { ... }
```

有加上 **static** 的稱爲「類別方法」，反之稱爲「物件方法」。

語法範例

```
static void skillA() {}       ◄····· 類別方法
void skillB() {}      ◄····· 物件方法
```

1-2-3 屬性

屬性（attribute）也可稱之爲變數或欄位，是類別與物件的資產。

宣告方式

```
[ 存取修飾字 ] + [static] + 資料型別 + 屬性名稱
```

有加上 **static** 的稱爲「類別變數」，反之稱爲「物件變數」。

語法範例

```
static String type = " 拉布拉多 ";  ◄····· 類別變數
String name = " 小白 ";    ◄····· 物件變數
```

1-2-4 區域變數

在方法或建構子裡面所定義的變數被稱爲區域變數，區域變數的生命週期僅存在於這個方法，方法一旦執行完畢，區域變數就自動歸還給系統。區域變數不可以加上 static 修飾字。

了解類別、屬性與方法後，接下來針對這些成員來作一些簡單的程式操控。請再回憶一下類別中有哪二個成員？屬性與方法，而利用 "." 運算子讓我們能輕鬆且直覺地存取到該類別的成員。

```
取得類別屬性  →  類別名稱 . 類別屬性
取得類別方法  →  類別名稱 . 類別方法 ()
System.out.println("Hello World"); 就是 . 運算子運用設計的最佳實例
```

設計 PetStore 與 Puppy 二個類別，程式進入點則設在 PetStore 類別，並在程式進入點的方法中操作 Puppy 類別的成員。UML Class Diagram 如圖 1-1。

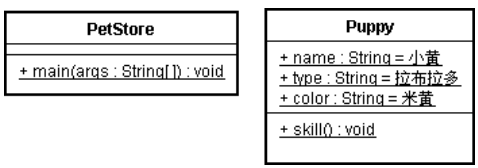


圖 1-1：UML Class Diagram

在 UML Class Diagram 中類別成員須加上底線（__）。

【範例程式一】Puppy.java

```
01  package book.java7.chapter1;
02
03  public class Puppy {
04      public static String name = "小黃";
05      public static String type = "拉布拉多";
06      public static String color = "米黃";
07
08      public static void skill() {
09          System.out.println("拿報紙與拎拖鞋");
10      }
11  }
```

package book.java7.chapter1; 是 Java 套件宣告，指的是編譯成 .class 檔後要放的路徑。有關 package 的用法與規則將在第 1-4 節「package 與 import」中介紹。

【範例程式二】PetStore.java

```
01  package book.java7.chapter1;
02
03  public class PetStore {
04      public static void main(String[] args) {
05          String dogName = Puppy.name;
06          String dogKind = Puppy.type;
07          String dogColor = Puppy.color;
08          System.out.println("我有一隻聰明的" + dogKind +
09                             "犬，名叫" + dogName +
10                             "，他的顏色是" + dogColor + "色的.");
11          System.out.print("他每天都會幫我");
12          Puppy.skill();
13      }
14  }
```

【編譯】javac PetStore.java

【執行】java PetStore

【執行結果】

```
我有一隻聰明的拉布拉多犬，名叫小黃，他的顏色是米黃色的．
他每天都會幫我拿報紙與拎拖鞋
```

利用 "." 運算子來存取 Puppy 類別所屬的類別成員。

1-2-5 具有回傳值的方法

void 屬於**無**回傳值的方法，若要在方法中設計**有**回傳值，必須在方法中明確定義回傳值的資料型別，Java 語法中不可設計**同時**具有回傳值和 void 的方法。

【範例程式】在 Puppy.java 中多設計一個具有回傳值功能的 skill2() 方法。

```
01  package book.java7.chapter1;
02
03  public class Puppy {
04      public static String name = " 小黃 ";
05      public static String type = " 拉布拉多 ";
06      public static String color = " 米黃 ";
07
08      public static void skill() {
09          System.out.println(" 拿報紙與拎拖鞋 ");
10      }
11      public static String skill2() {
12          String skill = " 拿報紙與拎拖鞋 ";
13          return skill;
14      }
15  }
```

【範例程式】PetStore.java

```
01  package book.java7.chapter1;
02
```

```
03  public class PetStore {
04      public static void main(String[] args) {
05          String dogName = Puppy.name;
06          String dogKind = Puppy.type;
07          String dogColor = Puppy.color;
08          System.out.println(" 我有一隻聰明的 " + dogKind +
09                             " 犬 , 名叫 " + dogName +
10                             ", 他的顏色是 " + dogColor + " 色的 .");
11          System.out.print(" 他每天都會幫我 ");
12          String dogSkill = Puppy.skill2();
13          System.out.println(dogSkill);
14          Puppy.skill2();
15      }
16  }
```

【 執行結果 】

```
我有一隻聰明的拉布拉多犬 , 名叫小黃 , 他的顏色是米黃色的 .
他每天都會幫我拿報紙與拎拖鞋
```

範例程式 PetStore.java 第 12 行，利用 dogSkill 區域變數來接收 Puppy.skill2() 所回傳的資料（skill2 的內容值：「拿報紙與拎拖鞋」）。值得注意的是，在 Java 語法中也**允許不接收具有回傳值方法的回傳值**，如範例程式 PetStore.java 第 14 行所示，雖然執行程式時沒有接收該方法的回傳值，但 Puppy.skill2() 的程式碼依然會執行。

1-2-6 何時使用 return 關鍵字？

在方法設計中最後都必須使用 return 關鍵字再加上法定回傳值回傳給呼叫端，但在 void 方法上是可以省略，當然若想要在 void 方法中使用也行，必須在 return 後面緊接著加上分號 ";"，如 "**return**;" 表示無回傳值。

此外，建構子好比是另一種無回傳值的方法，也可以加上 "**return**;"。一般說來，在 void 方法與建構子中加上 "**return**;"，實作上用來中斷該方法後續程式碼的執行，通常可以配合流程控制語法來撰寫例如 if。

1-3 註解

程式註解（comment）爲程式提供一段說明文字，JVM（Java 虛擬機器）在編譯和執行時會將其忽略。註解的作用在於記載某一段程式的功能或爲整段程式留下編寫記錄，好作爲 Java 程式開發人員維護程式時的參考。筆者建議平時養成寫註解的習慣，並且利用**文件註解**標記來製作程式說明書，以利日後自己或他人使用、查詢與維護。

在 Java 中註解分爲 4 種：

1. 單行註解 → // 註解文字

   當註解或說明文字只有一行時，使用單行註解。

2. 多行註解 → /*　註解文字　*/

   當註解或說明文字爲多行，或爲整段程式標示註解時使用多行註解。註解方式是在 /* 和 */ 之間撰寫說明資訊。例如：

```
/*  ◀····· 開始註解符號
    Puppy.java 範例程式碼
    多行註解
    …*/  ◀····· 結束註解符號
```

3. 文件註解 → /**　註解文字　*/

4. Annotation：一種 metadata 註解例如：

   假設要實作一個覆寫方法可以在方法上加入 @Override 來修飾

```
@Override
public void equals(Object obj) {
    … // block of code
}
```

   如此設計就容易區別一般方法與覆寫方法的實作區段。

利用文件註解 /** … */ 語法將註解資訊利用 javadoc.exe 來產生 Java API 文件，產生的過程會將程式碼中的文件註解內容寫到 API 文件中。

例如，我們可以將之前所撰寫的 Puppy.java 加上一些文件註解資訊：

```
package book.java7.chapter1;

/** 這是一個寵物的類別  */
public class Puppy {
    /** 寵物名稱 */
    public static String name = "小黃";
    /** 寵物品種 */
    public static String type = "拉布拉多";
    /** 寵物的外觀色澤 */
    public static String color = "米黃";
    /** 寵物的特殊才藝 */
    public static void skill() {
        System.out.println("拿報紙與拎拖鞋");
    }
}
```

撰寫好之後利用 javadoc.exe 指令來產生自己的 Java API 文件：

```
javadoc -verbose -private Puppy.java
```

private 等級（含）以上的存取權限
都可以寫到文件檔中。

執行所產生的 index.html 如圖 1-2 所示：

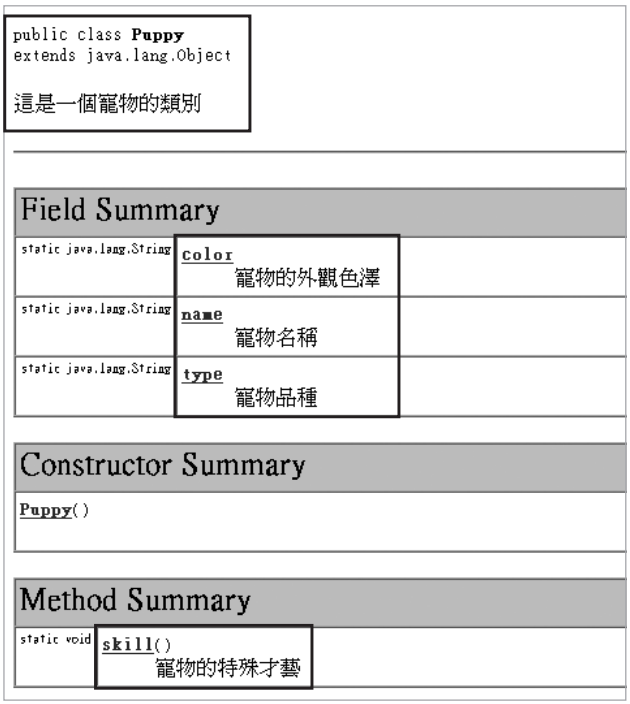


圖 1-2：Java API 文件

Oracle 的官方 javadoc 說明：

http://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html

不論使用哪一種註解或寫了多少註解字句，編譯器在編譯成 .class 檔的時候都會加以忽略，因此註解撰寫多寡與程式執行效率無關。

1-4 package 與 import

package 與 java 程式檔實際存放的位置息息相關，例如 package pet.water 表示該套件所有類別必須存放於 ...\pet\water 路徑之下，以便編譯器在執行編譯時能將所產生的 .class 檔放到指定路徑下。

【範例程式】欲將 Fish.java 與編譯後產生出來的 class 檔放置於 water 資料夾中

```
01  package book.java7.chapter1.water;
02
03  public class Fish {
04  }
```

經 javac Fish.java 編譯後，Fish.class 會放在 book/java7/chapter1/water 的資料夾之下，如圖 1-3。

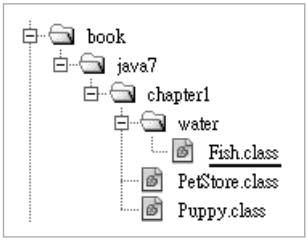


圖 1-3：存放 class 的資料夾

執行時期位於 package 下的 class 要如何能夠找到並存取呢？基本上有三種方式：

1. 打上完整路徑名稱（套件名稱 1. 套件名稱 2. ~ . 套件名稱 n. 類別名稱）
2. 透過 import 來定義
3. 設定 classpath 類別路徑

1-4-1 打上完整路徑名稱

首先我們先用第一種方式。

【範例程式】經由 PetStore2 類別來存取 water.Fish 類別的成員

```
01  package book.java7.chapter1.water;
02
03  public class Fish {
04      public static String name = "小金";
05      public static String type = "金魚";
06      public static String color = "金";
07      public static void skill() {
08          System.out.println("吐泡泡");
09      }
10  }
```

【範例程式】PetStore2.java

```
01  package book.java7.chapter1;
02
03  public class PetStore2 {
04      public static void main(String[] args) {
05          String fishName = book.java7.chapter1.water.Fish.name;
06          String fishKind = book.java7.chapter1.water.Fish.type;
07          String fishColor = book.java7.chapter1.water.Fish.color;
08          System.out.println("我有一隻優雅的" + fishKind +
09                             ", 名叫" + fishName +
10                             ", 他的顏色是" + fishColor + "色的.");
11          System.out.print("每當他肚子餓的時候都會");
12          book.java7.chapter1.water.Fish.skill();
13      }
14  }
```

程式（PetStore2.java）第 5 行，在 PetStore2 類別中要取得 Fish 類別的屬性 name，由於 PetStore2 與 Fish 並不在同一個 package 中，所以 PetStore2.java 的程式碼要明白指出 Fish 類別的確切位置，也就是類別完整全名（Fully qualified class name）：如 **book.java7.chapter7.water.Fish**.name。

1-4-2 透過 import 來定義

第二種方式是使用 import 來告知 java，如果在執行類別的目錄中找不到所指定 .class，應該要到哪裡去找？ import 有點類似 C# 中的 using 指令，不過就整體機制來說，java 的 import 指令又與它們有著許多不同。

上例中，可以使用「import book.java7.chapter1.water.Fish;」直接指定要使用的完整類別路徑，或使用「import book.java7.chapter1.water.*;」，因 water 套件裡所有的類別都是可能會參考到的對象，當然也自然會包含 Fish 類別囉！在程式實作上建議不要用星號（*），應將讓類別中有用到資源逐行 import，如此在理解程式邏輯之前就可以先知道該類別中有使用哪些資源。

【範例程式】PetStore3.java

利用 import 來存取 water.Fish 類別中的類別成員。Fish.java 請參考上一個程式碼。

```
01  import book.java7.chapter1.water.Fish;
02
03  public class PetStore3 {
04      public static void main(String[] args) {
05          String fishName = Fish.name;
06          String fishKind = Fish.type;
07          String fishColor = Fish.color;
08          System.out.println("我有一隻優雅的 " + fishKind +
09                             ", 名叫 " + fishName +
10                             ", 他的顏色是 " + fishColor + " 色的 .");
11          System.out.print("每當他肚子餓的時候都會 ");
12          Fish.skill();
13      }
14  }
```

注意，在引用類別名稱不同的情況下，import 可以帶來一些程式撰寫上的便利，不過程式中若需使用分屬在二個不同套件下的相同類別時，這時 import 可能就幫不上忙了，因爲編譯器不知道應選擇哪一個類別，所以要打上類別的完整全名，好讓編譯器能明確判別。

1-4-3 java.lang.* 基礎類別函式庫

爲何在 .java 檔中可以直接使用 System 類別，以及該類別所提供的成員？例如：

```
System.out.println(" 我愛 Java");
```

java 在編譯與執行時期直接可以找到 System 類別並加以運作。

在 java 程式規範中，預設會將 java.lang.*，也就是所謂的基礎類別函式庫（在 Java 程式中提供許多已經寫好的類別函式庫，名爲「Java Platform Packages」，供程式開發人員使用以加速程式的開發）自動 import 進來，該基礎類別的類別檔就被封裝在 C:\Program Files\openjdk\jre\lib\rt.jar 中，解開 rt.jar 後請參考圖 1-4。

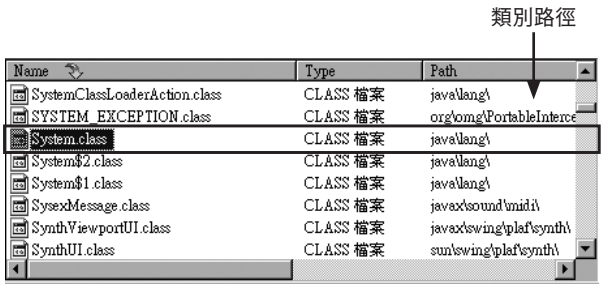
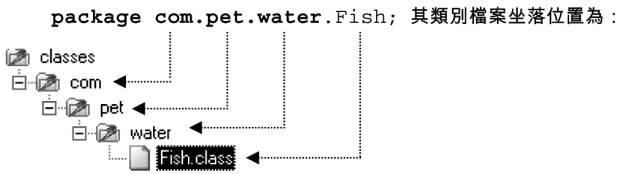


圖 1-4：透視 rt.jar

package 可連續使用「.」運算子來定義實際上存放的位置，也可用「*」來參照此套件下所有套件名稱。例如：



若您的環境變數有設定 CLASSPATH，Java 會根據您所設定的路徑去尋找，否則只會搜尋目前 Java 程式所在的工作目錄，因爲我們在先前的環境變數中是將 CLASSPATH 的預設值設爲 "."，也就是當前的目錄。

1-4-4 classpath（類別路徑）

第三種方式是設定 classpath，什麼是 classpath 呢？依據 JavaSE API 文件，它是 java 在執行時期用來搜尋類別與其它相關資源的路徑，classpath 是用來告訴 SDK 工具與 Java 應用程式要到哪裡尋找 third-party 或使用者自行定義的 classes、jar 或 zip 等資源，換言之，它們並不是 java 延伸套件或 java 平台的一部份。在預設的 java 環境中，只會去找 JDK 所提供的兩個類別套件：rt.jar 與 tools.jar。

在系統環境變數所設的 CLASSPATH，其內容將可以提供 java 程式在執行環境中找尋類別與其它相關資源的路徑。對於一些命令工具，例如：java、javac 與 javadoc 也提供 –classpath 指令，讓程式設計人員在每個獨立應用程式中能有自己定義的 classpath 路徑。

變更環境變數中的 CLASSPATH 指令（不同的 classpath 以 ";" 區隔）

```
C:\> set CLASSPATH=classpath1;classpath2...
```

設定獨立的 -classpath 指令（不同的 classpath 以 ";" 區隔）

```
C:\> sdkTool -classpath classpath1;classpath2...
```

此處的 sdkTool 是指 java.exe、javac.exe 或 javadoc.exe。另外，java.exe 這個工具程式還提供了 –classpath 的縮寫表示「–cp」。而 javac.exe 所提供的 –sourcepath 容易與 –classpath 混淆，–sourcepath 是用來設定 .java 檔的所在位置。

classpath1;classpath2：可用來指向副檔名爲 .jar、.zip 或 .class 的檔案，每一個 classpath 都應以檔名或目錄名稱作結束，而找尋的順序是先找 classpath1 再找 classpath2。

classpath 在撰寫上有三項規定：

1. 含有 .class 的 .jar 或 .zip 檔案，classpath 要以 .jar 或 .zip 作爲檔案的副檔名。
2. 未命名的 package，classpath 要以包括所有 .class 檔案的目錄名稱作結束。
3. 已命名的 package，classpath 要以包括 " 根 " package（完整 package 名稱的第一個 package）的目錄名稱作結束。

1-4-5 有 package 的 class 檔與 classpath

假設在目錄 **C:\prg\classes\book\java7\chapter1\water** 下有一個 Fish.class（類別完整全名：**book.java7.chapter1.water.Fish**），那麼 classpath 可設定成 C:\prg\classes 如下所示：

在 C:\> 目錄下，利用 –cp 或 –classpath 參數命令皆可。

```
C:\>java   -cp C:\prg\classes   book.java7.chapter1.water.Fish
```

若在 C:\prg> 目錄下必須設定如下：

```
C:\prg>java   -cp .\classes   book.java7.chapter1.water.Fish
```

在 C:\prg\classes> 目錄下，-cp 就可以不用設定了。

```
C:\prg\classes>java   book.java7.chapter1.water.Fish
```

1-4-6 jar 檔與 classpath

在目錄 C:\prg\classes> 下達 jar –cvf sample.jar 指令，將 classes 目錄（包含子目錄）中所有 class 包裹在 sample.jar 中，再將 C:\prg\classes 下的子目錄都刪除，讓 C:\prg\classes 的目錄中僅剩下 sample.jar 檔。

在 C:\ 目錄下：

```
C:\>java  -cp C:\prg\classes\sample.jar book.java7.chapter1.water.Fish
```

classpath 絕對路徑，且必須直接指向 sample.jar 檔。

完整類別名稱

在 C:\prg> 目錄下：

```
C:\prg>java  -cp .\classes\sample.jar  book.java7.chapter1.water.Fish
```

在 C:\prg\classes> 目錄下：

```
C:\prg\classes>java -cp sample.jar book.java7.chapter1.water.Fish
```

1-4-7 class、package 與 import 宣告時的先後順序

在程式中，宣告順序依序是：package → import → class。

合法的宣告

```
01  package water.*;
02  import java.io.*;
03  class MyTest {
04  }
```

不合法的宣告

```
01  import java.io.*;
02  package water.*;
03  class MyTest {
04  }
```

第一行必須與第二行的程式碼對調，因爲 package 必須宣告在 import 之前。