第十二章 檔案處理

課前指引

除了使用鍵盤輸入資料之外，另一個常見的資料來源就是「檔案」。檔案也可以作為一個中介存放媒體，例如：我們可以將欲排序的資料存放在資料檔內，經由排序程式的處理，形成有用的已排序資訊再回存到檔案中，以便於下次搜尋資料時，可以使用比較快速的搜尋演算法來搜尋資料。

章節大綱

12.1 Java的檔案處理

- 除了使用鍵盤輸入資料之外，另一個常見的資料來源就是「檔案」。
  - 舉例來說，以往當我們在程式執行中輸入資料，資料將存放在某個變數或陣列中，一旦結束程式再重新啟動程式後，上一次輸入的資料將會消失，這是因為程式的資料是儲存在主記憶體中，當程式結束時，程式佔用的主記憶體空間將被釋放，因此資料無法被儲存到下一次重新執行程式時。
  - 不過，如果我們先將資料存放到檔案中，在下一次重新執行程式時，就可以由檔案中載入上次存放的資料，並且即使是重新開機，資料也不會消失，這是由於檔案是存放在磁碟中，而非主記憶體。

12.1 Java的檔案處理

- 除此之外，檔案也可以作為一個中介存放媒體，例如：我們可以將欲排序的資料存放在資料檔內，經由排序程式的處理，形成有用的已排序資訊再回存到檔案中，以便於下次搜尋資料時，可以使用比較快速的搜尋演算法來搜尋資料。

12.1 Java的檔案處理

- 除了程式本身的資料運算之外，Java將其餘由外部輸入或輸出到外部設備的資料交由java.io類別庫的類別來處理。例如檔案、印表機等都屬於java.io類別庫的服務範圍。
- Java將檔案的處理，視為一個串流(stream)，不論在哪種硬體及作業系統環境下，檔案都被看成是由眾多字元(characters)或位元(bits)所組成的串流，因此程式設計師在進行檔案方面的處理時，面對的其實是一個資料串流，如圖12-1示意。

12.1 Java的檔案處理

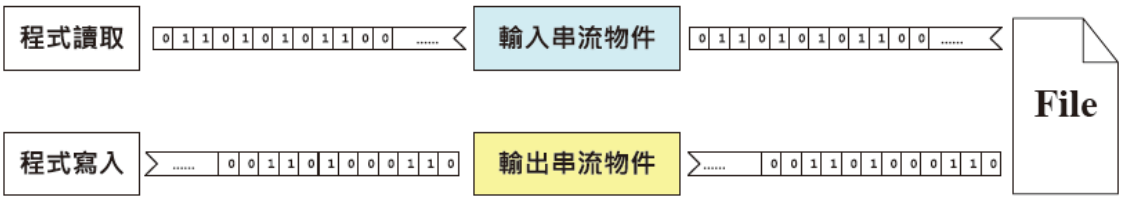


圖12-1 檔案是一種資料串流

- 串流分為輸入串流(input stream)與輸出串流(output stream)，分別用來對應資料的讀取與寫入。
- Java將處理串流的工作交由java.io類別庫完成，該類別庫可以讓Java進行所有IO的處理，而檔案也屬於其中一種，我們可以在需要進行檔案讀寫時，匯入該類別庫內的特定類別或者將整個java.io類別庫匯入。

12.1 Java的檔案處理

- 檔案類型與類別
  - 對於Java而言，檔案分為兩種：文字檔(Text file)與二進位檔(Binary file)，其特色如下。
  - 純文字檔：
    - 方便閱讀，但較無保密性。其他使用者也可以透過純文字編輯器開啟並成功閱讀。在Java程式中，純文字檔可使用Reader與Writer類別來進行讀寫工作。
  - 二進位檔：
    - I/O處理速度較快並具有保密性，但檔案內容需透過程式轉譯才能閱讀。二進位檔的資料是由一連串的位元或位元組所組合，通常使用在某些特殊用途，例如圖檔。在Java程式中，二進位檔可透過InputStream與OutputStream類別來進行讀寫工作。

12.1 Java的檔案處理

Coding 注意事項

純文字檔也可以透過InputStream與OutputStream類別進行讀寫工作。這是因爲，所有檔案的內容實際上都是以01等二進制檔案格式存放於磁碟中，只不過純文字檔的位元編碼具有特殊結構，當使用InputStream與OutputStream類別進行純文字檔的讀寫工作時，將忽略其編碼格式，例如複製檔案時，就可以適用。但若要將純文字檔以文字模式在Java程式中輸出，則仍以Reader與Writer類別較爲適當。

12.1 Java的檔案處理

- 事實上，Java程式在進行檔案的讀寫工作時，一般並未使用上述的四種類別直接進行工作，而是**使用其子孫類別進行工作**，因為這些子孫類別已經繼承了上述四種類別的重要方法
  - 例如，我們會使用FileReader類別讀取純文字檔，而當工作完畢欲將檔案關閉時所使用的close()方法，就是繼承自Reader類別（並在子孫類別中被改寫）。
  - 本章所介紹的相關類別繼承圖，如圖12-2、12-3中底色類別所示。

12.1 Java的檔案處理

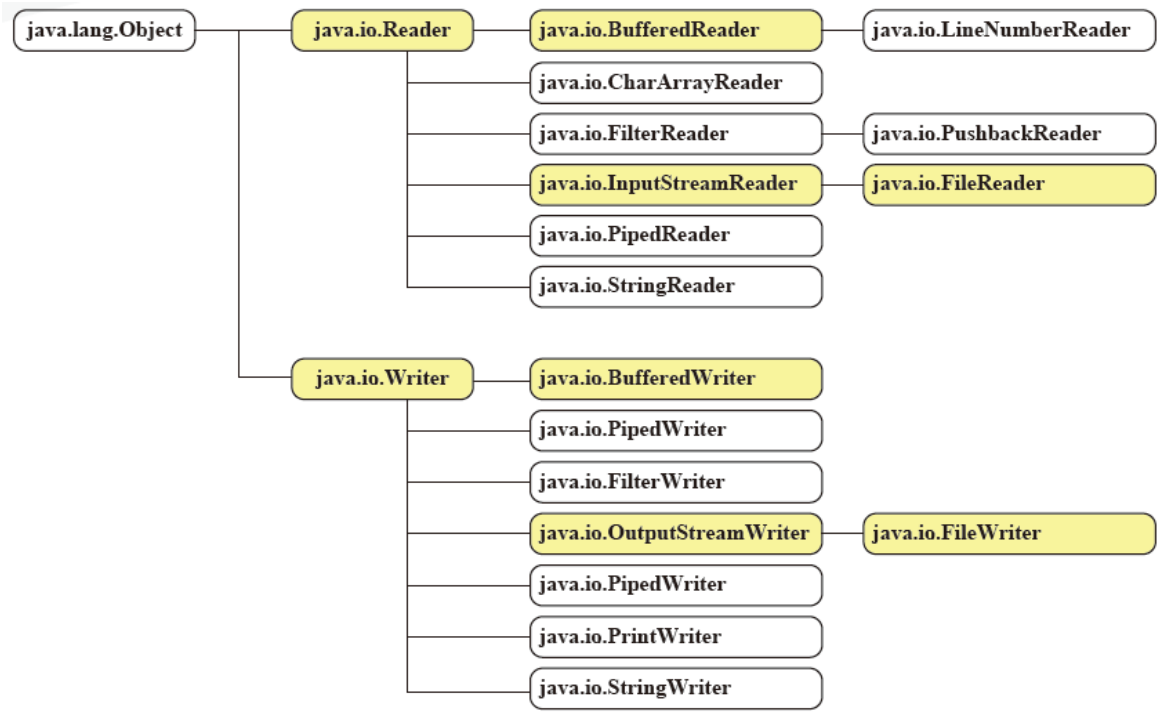


圖12-2 與IO有關的部分類別繼承圖（底色爲本章與純文字檔案有關之類別）

12.1 Java的檔案處理

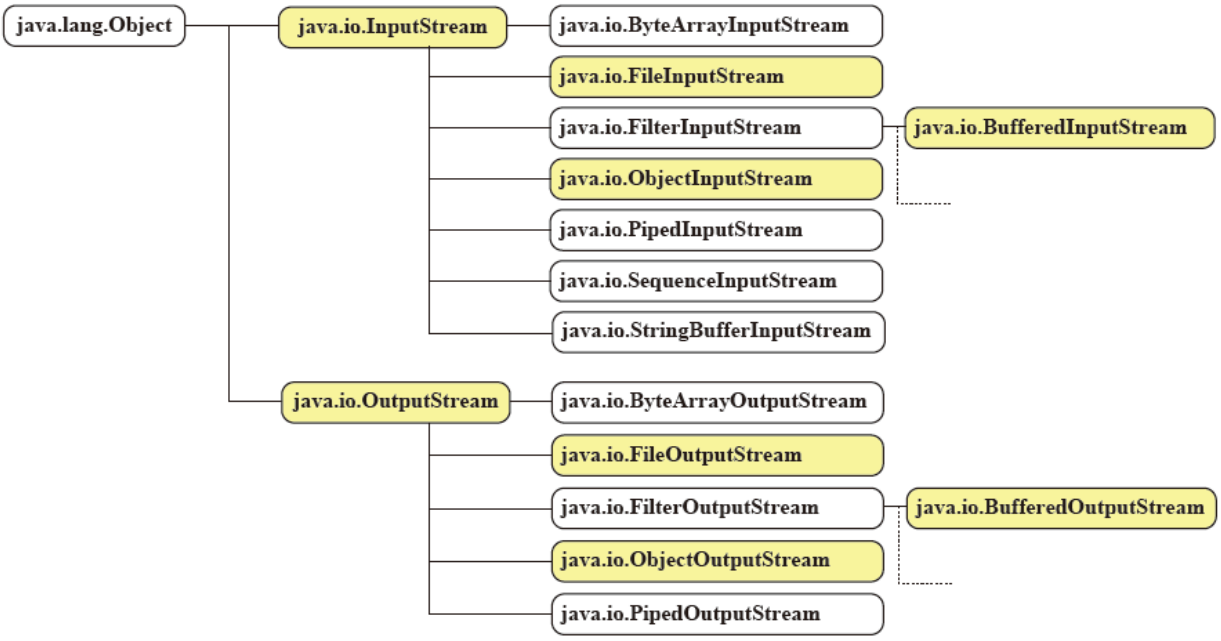


圖12-3 與IO有關的部分類別繼承圖（底色爲本章與二進位檔案有關之類別）

12.2 字元串流的檔案處理

- 字元串流亦即串流內以字元為單位，此處的字元指的是Unicode，因此，只有純文字檔適合使用字元串流進行讀寫。
  - Reader與Writer類別是以字元串流方式來處理純文字檔，這兩個類別提供了關於讀寫純文字檔的基本成員方法，如表12-1、12-2
  - 但我們無法直接使用這兩個類別的物件（原因後述），而是使用其衍生類別的物件進行實際的讀寫工作。
    - 當然，由於這些類別繼承自Reader/Writer類別，因此也可以使用表12-1、12-2的各種成員方法。

12.2 字元串流的檔案處理

方法宣告	用途	回傳值說明
abstract void close()	關閉字元串流。	無
int read()	讀取字元串流中的單一個字元。	-1代表串流已讀取完畢。其餘正整數或0則代表讀取字元成功。
int read(char[] cbuf)	讀取字元串流中的資料，並將之放入cbuf字元陣列中。	回傳值為讀取字元的數量，-1代表串流已讀取完畢。
abstract int read(char[] cbuf, int off, int len)	讀取字元串流中的資料，將之放入cbuf字元陣列，並由cbuf[off]開始存放，且指定最多讀取len個字元。	回傳值為讀取字元的數量，-1代表串流已讀取完畢。
long skip(long n)	略過n個字元不讀取。	實際被略過的字元數量。

表12-1 Reader的常用成員方法（上述方法會拋出IOException例外）

12.2 字元串流的檔案處理

方法宣告	用途	回傳值說明
Writer append(char c)	在串流內增加一個字元在末端。	回傳已經增加字元在末端的串流。
abstract void close()	關閉字元串流。	無
abstract void flush()	將緩衝區的內容寫入到檔案。當使用緩衝區時，必須記得使用此方法，否則資料僅會存放在串流中而不會寫入到檔案內。	無
void write(char[] cbuf)	將cbuf字元陣列的內容寫入串流。	無
abstract void write(char[] cbuf, int off, int len)	將字元陣列由cbuf[off]開始寫入len個字元到串流。	無
void write(int c)	寫入單一個字元到串流。	無
void write(String str)	寫入一個字串到串流。	無
void write(String str, int off, int len)	寫入一個字串的第off~off+len-1個字元到串流。	無

表12-2 Writer的常用成員方法（上述方法會拋出IOException例外）

12.2 字元串流的檔案處理

- 由上面的兩個表格中，我們可以發現Reader/Writer類別的有些成員方法被宣告為abstract，代表該類別無法直接產生物件，必須繼承後改寫(override)並實作這些abstract method才能產生物件。
  - 因此我們一般都是使用Reader/Writer類別的衍生類別來產生物件，並且這些衍生類別已經實作了abstract method，並不需要我們自行實作。

12.2.1 FileReader類別

- InputStreamReader為Reader的子類別，FileReader為InputStreamReader的子類別（見圖12-2）
  - 我們將使用FileReader作為讀取文字檔案的主要類別，因此，可使用的方法也包含祖先類別的方法。
- 任何對於檔案內讀寫動作可以分為三個步驟：分別是開檔、讀寫、關檔。
  - 其中，開檔動作包含在建立檔案物件時的建構子，例如FileReader的建構子如下：
- FileReader的建構子

```
FileReader(File file)
FileReader(FileDescriptor fd)
FileReader(String fileName)
//上述建構子會拋出FileNotFoundException例外，它是IOException例外類別的子類別
```

12.2.1 FileReader類別

- 【語法說明】：
  - 由於前兩個建構子的參數都是本書未介紹的類別型態，因此不多作介紹
  - 本書僅介紹第三類建構子FileReader(String fileName)，其中，fileName代表檔案路徑及檔名的字串，我們可以使用絕對路徑或相對路徑來表示。
  - Windows與Unix的目錄分隔符號不同
    - Unix/Linux為「/」、Windows為「\」，而「\」在Java字串中，被視為控制字元，故需要使用兩個「\」，也就是「\\」來表示目錄分隔符號。

- 【實用範例12-1】：開啟純文字檔，並使用字元串流方式讀取文字檔內容後輸出並統計所讀取的字元個數。
- 範例12-1：ch12_01.java（隨書光碟myJava\ch12\ch12_01.java）

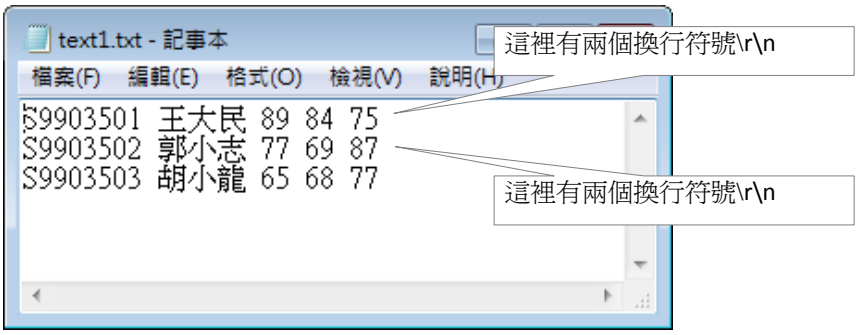
12.2.1 FileReader類別

```
1   /*   檔名:ch12_01.java          功能:以字元串流讀取文字檔   */
2
3   package myJava.ch12;
4   import java.lang.*;
5   import java.io.*;
6
7   public class ch12_01          //主類別
8   {
9       public static void main(String args[])
                              throws IOException,FileNotFoundException
10      {
11          char cbuf[] = new char[256];
12          FileReader fr =
                  new FileReader("c:\\myJava\\ch12\\file\\text1.txt");
13
14          int num = fr.read(cbuf);      //讀取最多256個字元到cbuf陣列中
15          String str1 = new String(cbuf,0,num);  //字元陣列轉換爲字串
16          System.out.println("總共讀取" + num + "個字元數");
17          System.out.println("檔案內容如下");
18          System.out.println(str1);
19          fr.close();         //關檔
20      }
21  }
```

使用字串指定檔名與路徑

12.2.1 FileReader類別

- text1.txt（隨書光碟 myJava\ch12\file\text1.txt）



- 執行結果：

```
總共讀取67個字元數
檔案內容如下
S9903501 王大民 89 84 75
S9903502 郭小志 77 69 87
S9903503 胡小龍 65 68 77
```

- 範例說明：
  - (1)FileReader建構子會產生FileNotFoundException例外，read()會產生IOException例外，所以在第9行，使用throws讓系統捕捉例外。

12.2.1 FileReader類別

- (2)第12行記錄了目標文字檔的路徑及檔名。文字檔的內容如上圖。
- (3)第14行，透過read讀取串流內的字元，由於cbuf字元陣列的大小為256，因此一次會讀取256個字元，如果不足，則讀取至最後一個字元為止，並且回傳值將會是實際讀取到的字元數量。
- (4)由執行結果中，可以發現一共讀取到67個字元，這是因為文字檔中，第一行與第二行分別有(21+2)個字元，最後一行則有21個字元。
  - 對於Java而言，不論中英文都算是一個字元。
  - 並且，前兩行多的兩個字元分別是「\r」與「\n」控制字元，在字元表中，分別代表的是Carriage Return（返回該行第一格）及NewLine（換行），而Windows在文字檔的每個換行處都會加上這兩個符號，當純文字編輯器讀取到這兩個符號時，就會顯示出換行的效果。
- (5)第19行是關檔。

12.2.1 FileReader類別

- 【實用範例12-2】：使用迴圈讀取純文字檔的所有內容。
- 範例12-2：ch12_02.java（隨書光碟 myJava\ch12\ch12_02.java）

```
1   /*  檔名:ch12_02.java          功能:使用迴圈讀取文字檔的全部內容  */
2
3   package myJava.ch12;
4   import java.lang.*;
5   import java.io.*;
6
7   public class ch12_02          //主類別
8   {
9       public static void main(String args[])
                        throws IOException,FileNotFoundException
10      {
11          char cbuf[] = new char[16];
12          FileReader fr =
                    new FileReader("c:\\myJava\\ch12\\file\\text1.txt");
13          int num;
14          String str1;
```


12.2.1 FileReader類別

```
15          while((num = fr.read(cbuf)) !=-1)
16          {
17              str1 = new String(cbuf,0,num);  //字元陣列轉換爲字串
18              System.out.println("總共讀取" + num + "字元數");
19              System.out.println(str1);
20              System.out.println("----------------------");
21          }
22          fr.close();         //關檔
23      }
24  }
```

使用迴圈讀取文字檔的全部內容

12.2.1 FileReader類別

- 執行結果
- 範例說明：
  - (1) 這個範例和範例12-1使用相同的目標文字檔，但這次我們將cbuf字元陣列的大小設定為16，明顯不足以一次讀取完畢字元串流的所有內容。因此在第15~21行，使用迴圈來分次讀取字元串流的內容，每次會自動讀取最多16個字元。
  - (2) 在執行結果中，可以發現每次讀取的字元數量，最後一次為3個字元，然後再嘗試讀取時，將會獲得-1回傳值而離開迴圈。

```
總共讀取16字元數
S9903501 王大民 89
----------------------
總共讀取16字元數
84 75
S9903502
----------------------
總共讀取16字元數
郭小志 77 69 87
S9
----------------------
總共讀取16字元數
903503 胡小龍 65 68
----------------------
總共讀取3字元數
 77
----------------------
```


12.2.1 FileReader類別

- try-with-resources
  - 在前面我們都使用了throws預防當讀取檔案時出現例外狀況，而在第10章時我們知道，如果不想使用throws，則應該在函式內使用try-catch來捕捉並處理例外。
  - 為了替這類因存取資源而撰寫的例外程式尋求一個更簡便的寫法，JDK7提供了try-with-resources機制，範例如下：
  - 【觀念及實用範例12-3】：使try-with-resources機制重新撰寫範例12-1，使得在main()函式內直接處理例外。
  - 範例12-3：ch12_03.java（隨書光碟myJava\ch12\ch12_03.java）

12.2.1 FileReader類別

```
1  /*  檔名:ch12_03.java          功能:try-with-resources  */
2
3  package myJava.ch12;
4  import java.lang.*;
5  import java.io.*;
6
7  public class ch12_03          //主類別
8  {
9      public static void main(String args[])
10     {
11         char cbuf[] = new char[256];
12
13         try(FileReader fr =
                   new FileReader("c:\\myJava\\ch12\\file\\text1.txt"))
14         {
15            int num = fr.read(cbuf);
16            String str1 = new String(cbuf,0,num);
17            System.out.println("總共讀取" + num + "個字元數");
18            System.out.println("檔案內容如下");
19            System.out.println(str1);
20            fr.close();        //關檔
21         }
22         catch(FileNotFoundException e)
23         {
24             System.out.println("例外發生:找不到該檔案");
25         }
```

try後面可以接上(...資源...)

12.2.1 FileReader類別

```
26         catch(final IOException e)
27         {
28             System.out.println("例外發生:檔案存取錯誤");
29         }
30     }
31 }
```

- 執行結果：（同範例12-1）
- 範例說明：
  - 以往在for迴圈的關鍵字for之後會有()，在其中宣告的變數屬於for迴圈專有。
  - JDK7提供的try-with-resources機制則是允許使用者在try()的括號內宣告變數並產生物件實體，而這些變數將在try{}內生效，如果在()內執行敘述時發生了例外，則例外可以被catch()捕捉。
  - 因此，我們通常會在需要使用外部資源時，將之撰寫於try(){}的小括號內，例如第13行開啟檔案如果不存在，則會被第22~25行的catch(FileNotFoundException e)捕捉到。

小試身手12-1

請將範例12-3第13行的檔名由text1.txt改爲不存在的text2.txt，然後進行編譯與執行，看看會獲得什麼樣的執行結果？

12.2.2 FileWriter類別

- 相對於FileReader的讀取功能，FileWriter類別則提供寫入文字檔的功能。
  - 同樣地，OutputStreamWriter為Writer的子類別，FileWriter為OutputStreamWriter的子類別（見圖12-2），因此，FileWriter產生的物件也可以使用祖先類別的方法。
    - 寫入的開檔動作同樣包含在建構子，FileWriter的建構子如下：
  - FileWriter的建構子

```
FileWriter(File file)
FileWriter(File file, boolean append)
FileWriter(FileDescriptor fd)
FileWriter(String fileName)
FileWriter(String fileName, boolean append)
//上述建構子會拋出IOException例外
```


12.2.2 FileWriter類別

- 【語法說明】：
  - 前三個建構子的參數都是本書未介紹的類別型態，因此不多作介紹，本書僅介紹後兩類建構子，其中，fileName代表檔案路徑及檔名的字串
  - 而append則是指定「是否覆蓋檔案原有內容」，若欲覆蓋原有檔案內容，則將之設定為false或不設定；若希望將新資料加在舊有資料後面，則設定為true。
- 【實用範例12-4】：將資料使用字元串流方式寫入純文字檔。
- 範例12-4：ch12_04.java（隨書光碟myJava\ch12\ch12_04.java）

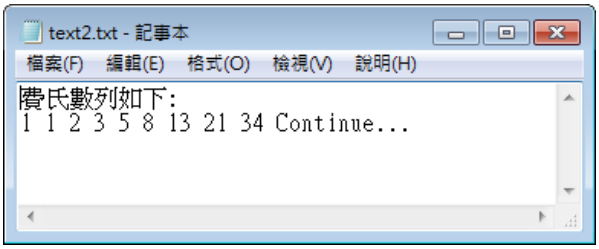
```
1  /*  檔名:ch12_04.java          功能:以字元串流寫入文字檔  */
2
3  package myJava.ch12;
4  import java.lang.*;
5  import java.io.*;
```

```
6
7   public class ch12_04          //主類別
8   {
9       public static void main(String args[]) throws IOException
10      {
11         String str1 = "費氏數列如下:";
12         char endCh[] = {'C','o','n','t','i','n','u','e','.','.','.'};
13         int numF;
14         FileWriter fw =
                   new FileWriter("c:\\myJava\\ch12\\file\\text2.txt");
15
16          fw.write(str1);
17          fw.write('\r');   fw.write('\n');   //寫入換行字元
18
19          for(int i=1;i<10;i++)
20          {
21              numF=Fib(i);
22              fw.write(numF+" ");
23          }
24
25          fw.write(endCh);
26          fw.close();        //關檔
27      }
28      public static int Fib(int n)
29      {
30          if((n==1) || (n==0))
31             return n;
32          else
33             return Fib(n-1)+Fib(n-2);
34      }
35  }
```

int與字串連結,會自動轉型爲字串

12.2.2 FileWriter類別

- 執行結果
  - 程式執行完畢，會在c:\myJava\ch12\file\目錄下產生text2.txt檔，內容如下



- 範例說明：
  - (1)範例執行完畢，會產生text2.txt，內容如上圖，其中的換行效果，是程式第17行的作用。
  - (2)第16行寫入字串。第25行寫入字元陣列。第17行則是寫入單一字元兩次。這三種是writer函式利用多載(overload)功能提供的三種參數列。

12.2.2 FileWriter類別

- (3)第22行，由於數值與字串透過「+」法運算時，執行的是字串連結，因此結果為字串，所以使用的是write(String str)函式。

- (4)本範例如果檔案原本不存在，則會建立新檔案。如果檔案原本已經存在，則原始檔案內容會被新的內容覆蓋。

小試身手12-2

延續小試身手12-1，請先執行範例12-4，產生text2.txt檔，然後再一次執行小試身手12-1，看看是否會獲得與小試身手12-1不一樣的執行結果？

12.2.2 FileWriter類別

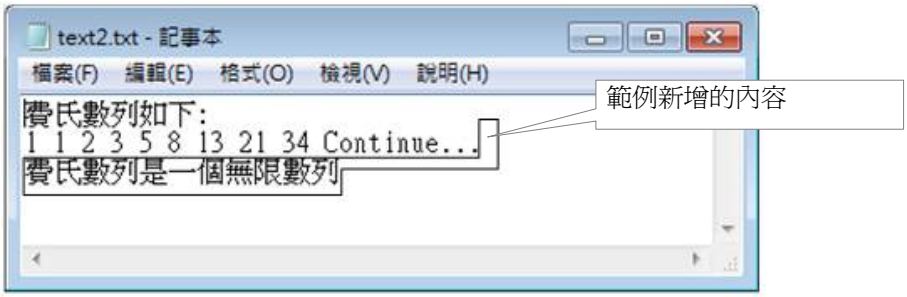
- 【實用範例12-5】：在原有文字檔案後面增加資料。
- 範例12-5：ch12_05.java（隨書光碟myJava\ch12\ch12_05.java）

```
1   /*  檔名:ch12_05.java          功能:加入文字在文字檔後方  */
2
3   package myJava.ch12;
4   import java.lang.*;
5   import java.io.*;
6
7   public class ch12_05          //主類別
8   {
9     public static void main(String args[]) throws IOException
10    {
11       String str1 = "費氏數列是一個無限數列";
12
13       FileWriter fw =
              new FileWriter("c:\\myJava\\ch12\\file\\text2.txt",true);
14       fw.write('\r');   fw.write('\n');   //寫入換行字元
15       fw.write(str1);
16
17       fw.close();       //關檔
18    }
19  }
```

true代表不要覆蓋原檔內容

12.2.2 FileWriter類別

- 執行結果（請先執行範例12-4後才執行本範例）



- 範例說明：
  - 第13行建立FileWriter物件fw時，在建構子設定了append參數為true，因此原始檔案內容不會被覆蓋。第14、15行為本範例新增的文字，其中換行字元也需要自行增加。
- 【實用範例12-6】：複製文字檔。
- 範例12-6：ch12_06.java（隨書光碟 myJava\ch12\ch12_06.java）

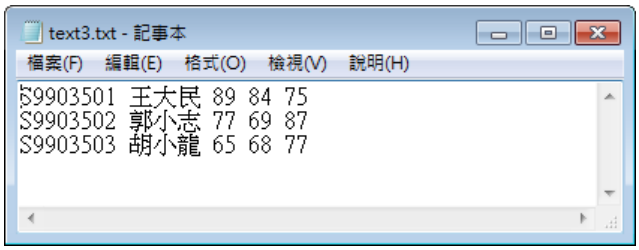
12.2.2 FileWriter類別

```
1   /*  檔名:ch12_06.java          功能:複製文字檔  */
2
3   package myJava.ch12;
4   import java.lang.*;
5   import java.io.*;
6
7   public class ch12_06          //主類別
8   {
9       public static void main(String args[])
                        throws IOException,FileNotFoundException
10      {
11          char cbuf[] = new char[1];
12          FileReader fr =
                    new FileReader("c:\\myJava\\ch12\\file\\text1.txt");
13          FileWriter fw =
                    new FileWriter("c:\\myJava\\ch12\\file\\text3.txt");
14          int num;
15          String str1;
16          while((num = fr.read(cbuf)) !=-1)
17                fw.write(cbuf);
18          fr.close();         //關檔
19          fw.close();         //關檔
20      }
21  }
```

使用迴圈逐一複製文字檔內容

12.2.2 FileWriter類別

- 執行結果（執行後出現text3.txt，內容與text1.txt相同）



- 範例說明：
  - 第16~17行使用迴圈逐一複製文字檔內容，由於cbuf的陣列大小為1，故一次複製一個字元。

12.2.3緩衝區

- 讀寫檔案時，可以採用緩衝區方式進行處理，減少對磁碟的存取動作（傳統磁碟存取屬於機械動作，故較慢）
  - 所謂緩衝區，代表的是在記憶體中預留一個空間作為緩衝區，當進行存取檔案時，會對緩衝區進行存取。
    - 若讀取檔案時，緩衝區已無資料可讀，則會從磁碟檔案中載入資料
    - 當緩衝區作為寫入使用時，若緩衝區已滿，則會自動將資料寫入磁碟檔案中，以清空緩衝區，使得程式仍舊可以對緩衝區寫入資料
    - 如圖12-4示意。
    - 而提供緩衝區功能的類別則為BufferedReader及BufferedWriter，其繼承關係請見圖12-2,12-3。

12.2.3緩衝區

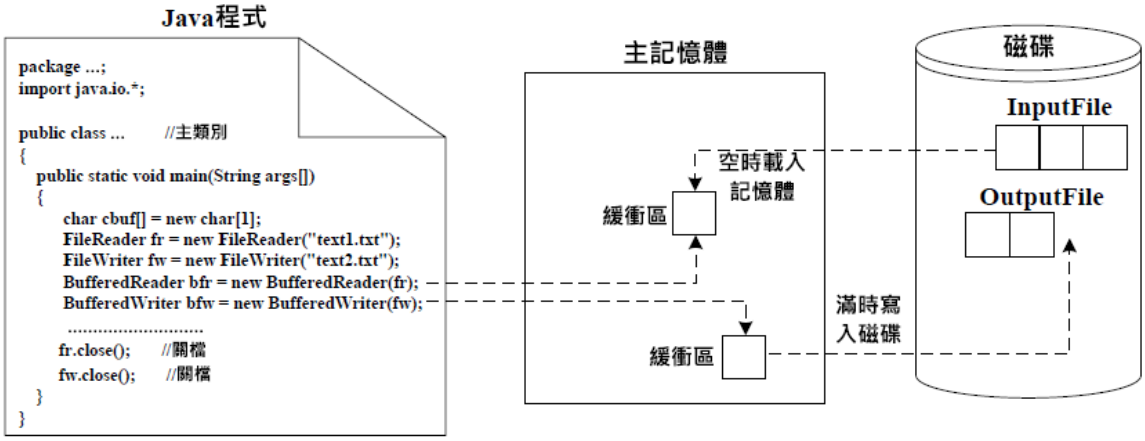


圖12-4 使用緩衝區讀寫檔案示意圖

12.2.3緩衝區

- BufferedReader
  - BufferedReader繼承自Reader類別，在執行建構子時，可以指定對應的檔案及緩衝區大小。建構子如下：

- BufferedReader的建構子

```
BufferedReader(Reader in)
BufferedReader(Reader in, int sz)
```

  - 【語法說明】：
    - 上述建構子可以產生輸入字元串流對應的緩衝區，雖然in參數被宣告為Reader類別，但我們可以傳入FileReader類別的物件，因為FileReader是Reader的子孫類別。
    - 而sz參數則為緩衝區的大小（sz大小需適中，否則使用緩衝區的意義不大），若未設定sz參數，則使用預設的緩衝區大小（JDK1.4預設為8192個字元）。

12.2.3緩衝區

- 當我們使用緩衝區方式來讀取檔案資料時，可以透過BufferedReader的眾多方法，進行更方便的讀取動作，方法如表12-3
  - 其中有些在Reader類別已經被定義，但在BufferedReader類別被改寫，有些則是新增的方法：

方法宣告	用途	回傳值說明
void close()	關閉字元串流。	無
int read()	讀取單一個字元。	-1代表串流已讀取完畢。其餘正整數或0則代表讀取字元成功。
int read(char[] cbuf)	讀取字元串流中的資料，並將之放入cbuf字元陣列中。	回傳值為讀取字元的數量，-1代表串流已讀取完畢。

12.2.3緩衝區

方法宣告	用途	回傳值說明
int read(char[] cbuf, int off, int len)	讀取字元串流中的資料，將之放入cbuf字元陣列，並由cbuf[off]開始存放，且指定最多讀取len個字元。	回傳值為讀取字元的數量，-1代表串流已讀取完畢。
long skip(long n)	略過n個字元不讀取。	實際被略過的字元數量。
public String readLine()	新增的方法，可以一次讀取一行字串	回傳一行字串，若已讀取完畢，則回傳null。

表12-3 BufferedReader的常用成員方法（上述方法會拋出IOException例外）

- 【實用範例12-7】：使用緩衝區讀取純文字檔內容，一次讀取一行直到讀取完畢為止。
- 範例12-7：ch12_07.java（隨書光碟myJava\ch12\ch12_07.java）

12.2.3緩衝區

```
1   /*  檔名:ch12_07.java          功能:使用緩衝區讀取文字檔  */
2
3   package myJava.ch12;
4   import java.lang.*;
5   import java.io.*;
6
7   public class ch12_07          //主類別
8   {
9       public static void main(String args[])
                        throws IOException,FileNotFoundException
10      {
11          String str1;
12          FileReader fr =
                  new FileReader("c:\\myJava\\ch12\\file\\text1.txt");
13          BufferedReader bufferIn = new BufferedReader(fr);
14
15          while((str1=bufferIn.readLine())!=null)
16          {
17                 System.out.println(str1);
18          }
19          fr.close();         //關檔
20      }
21  }
```

每次由緩衝區讀取一行資料

12.2.3緩衝區

- 執行結果：

```
S9903501 王大民 89 84 75
S9903502 郭小志 77 69 87
S9903503 胡小龍 65 68 77
```

- 範例說明：
  - 第15行使用readLine()讀取一行資料，當文字檔內容讀取完畢，則會回傳null。由執行結果可以看出，讀取該行資料時，回傳的字串並不包含換行字元。（執行結果的換行效果是由println產生）

12.2.3緩衝區

- BufferedWriter
  - BufferedWriter繼承自Writer類別，在執行建構子時，可以指定對應的檔案及緩衝區大小。建構子如下：

- BufferedWriter的建構子

```
BufferedWriter(Writer out)
BufferedWriter(Writer out, int sz)
```

  - 【語法說明】：
    - 上述建構子可以產生輸出字元串流對應的緩衝區，雖然out參數被宣告Writer類別，但我們可以傳入FileWriter類別的物件，因為FileWriter是Writer的子孫類別。
    - 而sz參數則為緩衝區的大小（sz大小需適中，否則使用緩衝區的意義不大），若未設定sz參數，則使用預設的緩衝區大小（JDK1.4預設為8192個字元）。

12.2.3緩衝區

- 當我們使用緩衝區方式來寫入檔案資料時，可以透過BufferedWriter的眾多方法，進行更方便的寫入動作，方法如表12-4
  - 其中有些在Writer類別已經被定義，但在BufferedWriter類別被改寫（僅繼承而未改寫者，則不列入表內），有些則是新增的方法：

方法宣告	用途	回傳值說明
void close()	關閉字元串流，在關閉之前必須進行flush動作，才會將最後一筆資料寫入檔案。	無
void flush()	將緩衝區的內容立刻寫入到檔案。	無
void newLine()	寫入換行字元。	無
void write(char[] cbuf, int off, int len)	將字元陣列由cbuf[off]開始寫入len個字元到串流緩衝區。	無

12.2.3緩衝區

方法宣告	用途	回傳值說明
void write(int c)	寫入單一個字元到串流緩衝區。	無
void write(String s, int off, int len)	寫入一個字串的第off~off+len-1個字元到串流緩衝區。	無

表12-4 BufferedWriter的常用成員方法（上述方法會拋出IOException例外）

- 【觀念及實用範例12-8】：使用緩衝區寫入純文字檔內容，並且觀察緩衝區大小與flush()功能。
- 範例12-8：ch12_08.java（隨書光碟myJava\ch12\ch12_08.java）

```
1  /*  檔名:ch12_08.java          功能:使用緩衝區寫入文字檔  */
2
3  package myJava.ch12;
4  import java.lang.*;
5  import java.io.*;
6
```

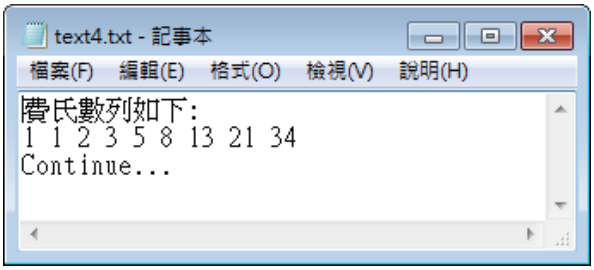


```
7  public class ch12_08          //主類別
8  {
9      public static void main(String args[]) throws IOException
10     {
11        String str1 = "費氏數列如下:";
12        char endCh[] = {'C','o','n','t','i','n','u','e','.','.','.'};
13        int numF;
14        FileWriter fw =
                 new FileWriter("c:\\myJava\\ch12\\file\\text4.txt");
15        BufferedWriter bufferOut = new BufferedWriter(fw,20);
16
17        bufferOut.write(str1);
18        bufferOut.newLine();          //寫入換行字元
19        for(int i=1;i<10;i++)
20        {
21            numF=Fib(i);
22            bufferOut.write(numF+" ");//int與字串連結,會自動轉型爲字串
23        }
24        bufferOut.newLine();          //寫入換行字元
25        bufferOut.write(endCh);
26        bufferOut.flush();            //重要,請看註解
27        fw.close();         //關檔
28    }
29    public static int Fib(int n)
30    {
31        if((n==1) || (n==0))
32           return n;
33        else
34           return Fib(n-1)+Fib(n-2);
35    }
36 }
```

寫入緩衝區的大小爲20個字元

12.2.3緩衝區

- 執行結果
  - 程式執行完畢，會在c:\myJava\ch12\file\目錄下產生text4.txt檔，內容如下



- 範例說明：
  - (1)第18行利用newLine()函式寫入該作業系統認定的換行字元。
  - (2)第15行建立的緩衝區大小為20。因此，欲寫入的全部資料將大於緩衝區的大小。第26行，利用flush()函式要求立即將緩衝區的內容寫入檔案，然後在第27行關閉檔案。若將第26行設定為註解，則執行結果中，最後面的「...」三個字元將不會被寫入檔案內，讀者可以自行測試看看。

12.2.3緩衝區

- (3)由這個範例，我們可以得知，緩衝區資料寫入檔案的時機可以分為(1)當緩衝區已滿時，自動寫入檔案內。(2)透過flush()強制立即將緩衝區內的資料寫入檔案。無論使用那一種方式寫入檔案，緩衝區在資料被寫入檔案後，都將被清空，以便容納新寫入的資料。
- (4)緩衝區的大小決定了檔案的寫入次數，例如若將本範例第15行的緩衝區大小設定為「1」，則第26行可以省略，因為緩衝區一旦寫入資料就會立刻額滿而必須自動寫入檔案中，這樣一來，是否使用緩衝區就變得沒有意義（因為無法減少實際寫入檔案的次數），但若將緩衝區設定過大，則必須系統提供夠多的記憶體可使用，因此適中的緩衝區大小必須視系統執行環境以及資料量的大小來決定，若無法預測者，建議還是使用預設的緩衝區大小即可。

小試身手12-3

請將範例12-8的第26行改爲註解，然後重新編譯與執行，看看會獲得什麼結果？

12.3位元串流的檔案處理

- 有些檔案的內容並非文字，例如圖檔等，此時要讀寫此類檔案則應該使用位元串流。
- InputStream與OutputStream類別是以位元串流方式來處理檔案，這兩個類別提供了關於讀寫檔案的基本成員方法，如表12-5、12-6

12.3位元串流的檔案處理

- 同樣地，我們無法直接使用這兩個類別的物件，而是使用其衍生類別的物件進行實際的讀寫工作。當然，由於這些類別繼承自InputStream/OutputStream類別，因此也可以使用表12-5、12-6的各種成員方法。
- 而在Java中讀取位元串流將以「位元組」為基本單位，而非位元。但位元組也是由位元構成，因此一般我們仍以位元串流來稱呼。

12.3位元串流的檔案處理

方法宣告	用途	回傳值說明
int available()	取得串流內可讀取或可略過的位元組數量。通常可用來代表檔案大小。	串流內可讀取或可略過的位元組數量。
void close()	關閉位元串流。	無
abstract int read()	讀取串流內的一個位元組資料。	回傳值代表為讀取位元組的值（0~255），若回傳-1代表串流已讀取完畢。
int read(byte[] b)	讀取位元串流中的資料，將之放入b位元組陣列。	回傳值為讀取位元組的數量，-1代表串流已讀取完畢。
int read(byte[] b, int off, int len)	讀取位元串流中的資料，將之放入b位元組陣列，並由b[off]開始存放，且指定最多讀取len個位元組。	回傳值為讀取位元組的數量，-1代表串流已讀取完畢。
long skip(long n)	略過n個位元組不讀取。	實際被略過的位元組數量。

表12-5 InputStream的常用成員方法（上述方法會拋出IOException例外）

12.3位元串流的檔案處理

方法宣告	用途	回傳值說明
void close()	關閉位元串流。	無
void flush()	將緩衝區的內容寫入到檔案。當使用緩衝區時，必須記得使用此方法，否則資料僅會存放在串流中而不會寫入到檔案內。	無
void write(byte[] b)	將b位元組陣列的內容寫入串流。	無
void write(byte[] b, int off, int len)	將位元組陣列由b[off]開始寫入len個位元組到串流。	無
abstract void write(int b)	寫入單一個位元組到串流。（b的前24位元忽略）	無

表12-6 OutputStream的常用成員方法（上述方法會拋出IOException例外）

12.3位元串流的檔案處理

- 同樣地，由於InputStream/OutputStream類別的某些成員方法被宣告為abstract，因此無法直接產生物件，必須繼承後改寫實作這些abstract method才能產生物件。
  - 而我們一般都是使用InputStream/OutputStream類別的衍生類別來產生物件，並且這些衍生類別已經實作了abstract method，並不需要我們自行實作。

12.3.1 FileInputStream與FileOutputStream類別

- FileInputStream與FileOutputStream分別為InputStream/OutputStream的子類別（見圖12-3）
  - 我們將使用FileInputStream與FileOutputStream作為讀取與寫入檔案的主要類別，因此，可使用的方法也包含父類別的方法。
    - 請特別注意，使用位元串流可以讀取非文字檔，也可以讀取文字檔，因為文字檔在檔案中仍舊是以位元方式存在，只不過文字檔的位元組存放方式具有特殊排列的意義，而當使用位元串流讀取檔案資料時，我們通常不關心位元組資料的實際意義。

12.3.1 FileInputStream與FileOutputStream類別

- 同樣地，對於一般檔案的讀寫動作也是分為三個步驟：分別是開檔、讀寫、關檔。
  - 其中，開檔動作包含在建立檔案物件時的建構子。
- FileInputStream的建構子

```
FileInputStream(File file)
FileInputStream(FileDescriptor fdObj)
FileInputStream(String name)
//上述建構子會拋出FileNotFoundException例外，它是IOException例外類別的子類別
```

- 【語法說明】：
  - 同樣地，本書僅介紹第三類建構子FileInputStream(String name)，其中，name為代表檔案路徑及檔名的字串，我們可以使用絕對路徑或相對路徑來表示。

12.3.1 FileInputStream與FileOutputStream類別

- FileOutputStream的建構子

```
FileOutputStream(File file)
FileOutputStream(File file, boolean append)
FileOutputStream(FileDescriptor fdObj)
FileOutputStream(String name)
FileOutputStream(String name, boolean append)
//上述建構子會拋出IOException例外
```

- 【語法說明】：
  - 同樣地，本書僅介紹後兩類建構子，其中，name為代表檔案路徑及檔名的字串，而append則是指定「是否覆蓋檔案原有內容」，若欲覆蓋原有檔案內容，則將之設定為false或不設定；若希望將新資料加在舊有資料後面，則設定為true。
- 【實用範例12-9】：透過位元串流複製圖片檔。
- 範例12-9：ch12_09.java（隨書光碟myJava\ch12\ch12_09.java）

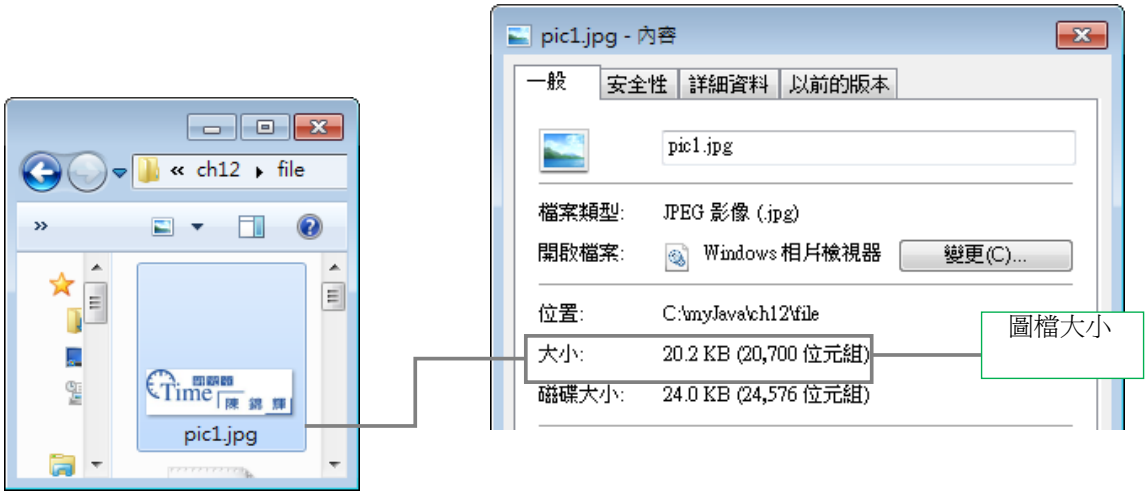
12.3.1 FileInputStream與FileOutputStream類別

```
1   /*  檔名:ch12_09.java          功能:複製檔案  */
2
3   package myJava.ch12;
4   import java.lang.*;
5   import java.io.*;
6
7   public class ch12_09          //主類別
8   {
9       public static void main(String args[])
                        throws IOException,FileNotFoundException
10      {
11          byte byteData[] = new byte[1];
12          FileInputStream fi =
               new FileInputStream("c:\\myJava\\ch12\\file\\pic1.jpg");
13          FileOutputStream fo =
              new FileOutputStream("c:\\myJava\\ch12\\file\\pic2.jpg");
14          int fileSize=fi.available();
15          int num;
16          while((num = fi.read(byteData)) !=-1)
17              fo.write(byteData);
18          System.out.println("檔案大小=" + fileSize + "位元組複製完畢");
19          fi.close();         //關檔
20          fo.close();         //關檔
21      }
22  }
```

使用迴圈逐一複製每一個位元組

12.3.1 FileInputStream與FileOutputStream類別

- pic1.jpg（隨書光碟 myJava\ch12\file\pic1.jpg）



12.3.1 FileInputStream與FileOutputStream類別

- 執行結果：

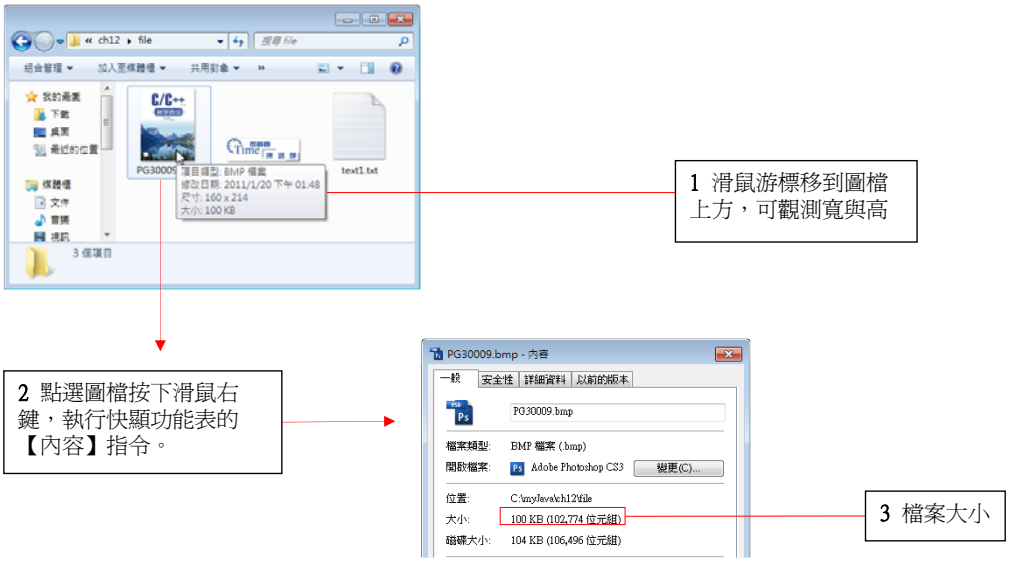
```
檔案大小=20700位元組複製完畢
```

- 範例說明：
  - (1)範例執行完畢，會在myJava\ch12\file\目錄中出現pic2.jpg檔，大小同為20700位元組。
  - (2)因byteData位元組陣列大小被宣告為1，因此最多只能存放一個位元組，第16~17行透過迴圈，每次複製一個位元組資料。
  - (3)第14行，available()函式回傳串流中可供讀取的最大位元組數量。

12.3.2 讀寫BMP圖片檔

- 在前面我們曾經提及，圖片檔屬於二進位檔，事實上，眾多圖片檔所存放的格式皆有所不同
  - 其中BMP檔為點陣圖。
  - BMP檔的內容包含了表頭資訊與像素資訊兩部分，我們首先可透過檔案總管觀察圖檔的大小如下。
    - 就該圖為例，PG30009.bmp是24位元色彩的BMP圖檔，檔案大小為102,774位元組，由於寬與高為160×214=34,240，而每一個像素點佔用3個位元組(24bits)，故要存放所有像素點必須使用34,240×3=102,720個位元組
    - 這和102,774仍相差了54個位元組，這54個位元組即為表頭資訊。

12.3.2 讀寫BMP圖片檔



12.3.2 讀寫BMP圖片檔

- BMP圖檔的表頭資訊分為兩大部分
  - 第一段佔用14個位元組（例如檔案大小記錄於此區間）
  - 第二段佔用40個位元組（例如寬與高記錄於此區間）
  - 詳細資訊整理如表12-7。

位元組編號	說明
（第一段）	代表點陣圖的標識。
#0-1	這兩個位元組固定爲424DH，對應ASCII則爲BM。
#2-5	代表檔案大小。
#6-9	保留（因此皆爲0），可作爲往後擴充使用。
#10-13	記錄圖形資料的起始位址。

12.3.2 讀寫BMP圖片檔

位元組編號	說明
（第二段）	一個常數值，用來描述影像區塊的大小。
#14-17	在不同的系統中，常數值並不相同，在Windows中爲28H
#18-21	點陣圖的寬度（以像素點數量表示）。
#22-25	點陣圖的高度（以像素點數量表示）。
#26-27	彩色平面數，通常爲 1 (十六色影像則爲4)。
#28-29	每個像素的顏色位元數，亦即深度。常用值是1、4、8（灰階）和24（全彩）。
#30-33	記錄所使用的壓縮演算法，可能的值爲0、1、2、3、4、5。 若未壓縮則其值爲0；JPEG壓縮則爲4。
#34-37	實際圖形檔的字組大小
#38-41	圖像水平解析度（單位爲像素/英吋）。
#42-45	圖像垂直解析度（單位爲像素/英吋）。
#46-49	所用顏色數目。
#50-53	保存所用重要顏色數目。若每個顏色都同等重要時，則與顏色數目相等。

表12-7 BMP圖檔的表頭資訊（54個位元組）

- 想要觀察上述的PG30009.bmp點陣圖內容，可透過debug指令或UltraEdit等軟體開啟它，通常會以16進制表示各位元組內容，如下圖：

12.3.2 讀寫BMP圖片檔

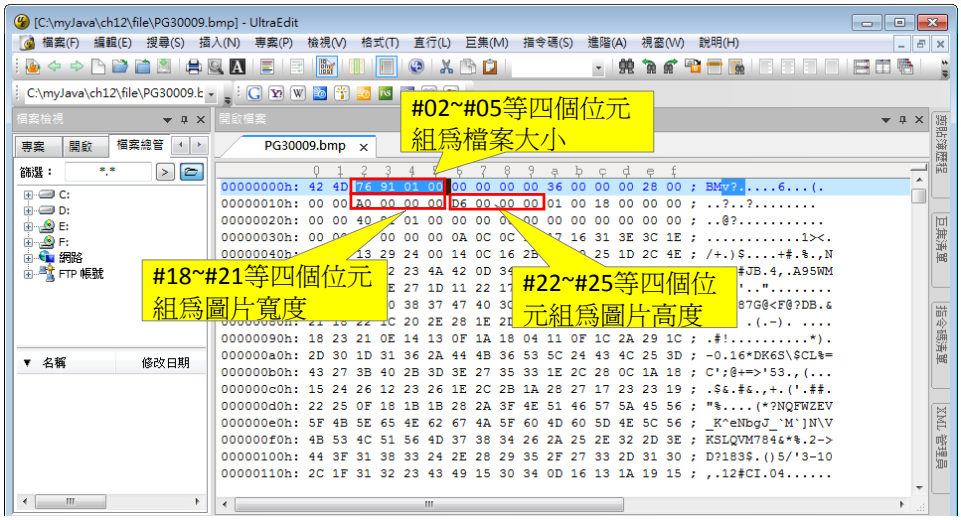


圖11-5 觀察點陣圖BMP檔內容

- 請注意，當以四個位元組代表一組資料時，必須反過來觀看
- 例如圖檔之大小顯示為「76」「91」「01」「00」，實際上應該是「00」「01」「91」「76」H個位元組 = 102,774個位元組。

12.3.2 讀寫BMP圖片檔

- 【觀念及實用範例12-10】：在BMP檔的表頭資訊中，取出檔案大小的資訊。
- 範例12-10：ch12_10.java（隨書光碟 myJava\ch12\ch12_10.java）

```
1   /*  檔名:ch12_10.java          功能:讀取二進位檔圖檔資料  */
2
3   package myJava.ch12;
4   import java.lang.*;
5   import java.io.*;
6
7   public class ch12_10        //主類別
8   {
9       public static void main(String args[])
                    throws IOException,FileNotFoundException
10      {
11        byte data[] = new byte[54];
12        int filesize,value1,value2,value3,value4,low,high;
13
14        FileInputStream fi =
            new FileInputStream("c:\\myJava\\ch12\\file\\PG30009.bmp");
15        int fileSize=fi.available();
16        int num;
```


12.3.2 讀寫BMP圖片檔

```
17        num = fi.read(data,0,54);      //讀取前54個位元組
18
19        int title[] = new int[6];
20        for(int i=2;i<5;i++)
21        {
22             if(data[i]<0)
23                 title[i]=(int)data[i]+256;
24             else
25                 title[i]=(int)data[i];
26
27        }
28
29        low=title[2]%16;
30        high=(title[2]-low)/16;
31        value1=high*16+low;
32
33        low=title[3]%16;
34        high=(title[3]-low)/16;
35        value2=high*16*16*16+low*16*16;
36
37        low=title[4]%16;
38        high=(title[4]-low)/16;
39        value3=high*16*16*16*16*16+low*16*16*16*16;
40
```

若資料>127(3FH) ，則會被判定爲負值，所以需要修正爲正值

12.3.2 讀寫BMP圖片檔

```
41          low=title[5]%16;
42          high=(title[5]-low)/16;
43          value4=high*16*16*16*16*16*16*16+low*16*16*16*16*16*16*16;
44
45          filesize=value1+value2+value3+value4;
46
47          System.out.println("檔案大小爲" + filesize + "個位元組");
48          fi.close();         //關檔
49      }
50  }
```

- 執行結果：

```
檔案大小爲102774個位元組
```

- 範例說明：
  - (1)第11行，宣告一個大小為54的陣列data，陣列元素資料型態為byte，其元素只佔用一個位元組，元素值介於-128~+127，。
  - (2)第17行：使用read函式讀取檔案的前54個位元組資料，並存入data陣列中。
  - (3)第19~45行：由前面的介紹可以得知檔案大小之資訊存放在data[2]~data[5]之中，但我們必須轉換進制才能求得正確之檔案大小。轉換時必須注意，每一個位元組若大於127，也就是十六進制的3F，會被認為負值，因此計算前要先修正為正值。

12.3.3緩衝區

- 一般檔案透過位元串流讀寫時，同樣可以採用緩衝區方式進行處理，以改善存取效率。
  - 而對應提供緩衝區功能的類別則為BufferedInputStream及BufferedOutputStream，其繼承關係請見圖12-3。
- BufferedInputStream
  - BufferedInputStream繼承自FilterInputStream，而FilterInputStream繼承自InputStream。在執行建構子時，可以指定對應的檔案及緩衝區大小。建構子如下：
- BufferedInputStream的建構子

```
BufferedInputStream(InputStream in)
BufferedInputStream(InputStream in, int size)
```

12.3.3緩衝區

- 【語法說明】：
  - 上述建構子可以產生輸入位元串流對應的緩衝區，in參數代表對應的位元串流。
  - 而size參數則為緩衝區的大小（size大小需適中，否則使用緩衝區的意義不大），若未設定size參數，則使用預設的緩衝區大小（JDK1.4預設為2048個位元組）。

12.3.3緩衝區

- 當我們使用緩衝區方式來讀取檔案資料時，可以透過BufferedInputStream及其父類別FilterInputStream的眾多方法，進行更方便的讀取動作，BufferedInputStream的方法如表12-8
  - 其中有些在父類別或祖父類別已經被定義，但在BufferedInputStream類別被改寫，有些則是新增的方法：

12.3.3緩衝區

方法宣告	用途	回傳值說明
int available()	取得串流內可讀取或可略過的位元組數量。通常可用來代表檔案大小。	串流內可讀取或可略過的位元組數量。
void close()	關閉位元串流。	無
int read()	讀取單一個位元組。	-1代表串流已讀取完畢。其餘0~255則為所讀取的位元組。
int read(byte[] b, int off, int len)	讀取位元串流中的資料，將之放入b位元組陣列，並由b[off]開始存放，且指定最多讀取len個位元組。	回傳值為讀取位元組的數量，-1代表串流已讀取完畢。
long skip(long n)	略過n個位元組不讀取。	實際被略過的位元組數量。

表12-8 BufferedInputStream的常用成員方法（上述方法會拋出IOException例外）

12.3.3緩衝區

- BufferedOutputStream
  - BufferedOutputStream繼承自FilterOutputStream，而FilterOutputStream繼承自OutputStream。在執行建構子時，可以指定對應的檔案及緩衝區大小。建構子如下：
- BufferedOutputStream的建構子

```
BufferedOutputStream(OutputStream out)
BufferedOutputStream(OutputStream out, int size)
```

  - 【語法說明】：
    - 上述建構子可以產生輸出位元串流對應的緩衝區，out參數為對應的位元串流。
    - 而size參數則為緩衝區的大小，若未設定size參數，則使用預設的緩衝區大小（JDK1.4預設為512個位元組）。

12.3.3緩衝區

- 當我們使用緩衝區方式來寫入檔案資料時，可以透過BufferedOutputStream的眾多方法，進行寫入動作，方法如表12-9
  - 這些方法都在父類別及祖父類別已經被定義，而BufferedOutputStream類別將之繼承或改寫：

方法宣告	用途	回傳值說明
void flush()	將緩衝區的內容立刻寫入到檔案。	無
void write(byte[] b, int off, int len)	將位元組陣列由b[off]開始寫入len個位元組到串流緩衝區。	無
void write(int b)	寫入特定位元組資料到串流緩衝區。	無

表12-9 BufferedOutputStream的常用成員方法（上述方法會拋出IOException例外）

12.3.3緩衝區

- 【實用範例12-11】：使用位元串流搭配緩衝區讀取文字檔內容，並寫入另一個檔案中。
- 範例12-11：ch12_11.java（隨書光碟myJava\ch12\ch12_11.java）

```
1  /*  檔名:ch12_11.java          功能:複製並輸出檔案  */
2
3  package myJava.ch12;
4  import java.lang.*;
5  import java.io.*;
6
7  public class ch12_11          //主類別
8  {
9      public static void main(String args[])
                        throws IOException,FileNotFoundException
10     {
11         FileInputStream fi =
             new FileInputStream("c:\\myJava\\ch12\\file\\text1.txt");
```

12.3.3緩衝區

```
12          FileOutputStream fo =
             new FileOutputStream("c:\\myJava\\ch12\\file\\text5.txt");
13          BufferedInputStream bufIn = new BufferedInputStream(fi);
14          BufferedOutputStream bufOut = new BufferedOutputStream(fo);
15          int fileSize=fi.available();
16          byte byteData[] = new byte[fileSize];
17          bufIn.read(byteData,0,fileSize);
18          bufOut.write(byteData,0,fileSize);
19          String str1 = new String(byteData);
20          bufOut.flush();   //記得要flush,否則串流內資料不會寫入檔案中
21          System.out.println("檔案複製完畢,內容如下");
22          System.out.println(str1);
23          fi.close();       //關檔
24          fo.close();       //關檔
25      }
26  }
```

轉換位元組陣列爲字串

- 執行結果：

```
檔案複製完畢,內容如下
S9903501 王大民 89 84 75
S9903502 郭小志 77 69 87
S9903503 胡小龍 65 68 77
```


12.3.3緩衝區

- 範例說明：
  - (1)程式執行完畢，會出現text5.txt，內容與text1.txt完全相同。這說明了位元串流也可以用來讀取、複製文字檔。因為不論是任何檔案，都是以0、1位元模式存在於檔案系統中。只不過該0、1位元是否具有某種編碼格式的差異罷了。
  - (2)第19行將位元組陣列內容轉換為字串。如果您將本範例的來源檔設定為圖檔pic1.jpg，則轉換出來的字串我們並無法解讀（會是一連串的亂碼），因為那是以jpg格式編碼的資料，而非文字資料，但您只要更改複製目標檔的副檔名為jpg，就能用看圖軟體顯示該檔案格式，因為看圖軟體看得懂jpg格式編碼的資料。

12.3.4序列化（Serialization）

- 一般檔案與文字檔最大的差別在於
  - 文字檔是所有純文字編輯器都可以輕鬆解讀的檔案，它可能使用Unicode與ASCII或其他編碼方式，但無論如何，要解讀純文字檔是非常容易的。
  - 而一般檔案則只有熟悉該檔案編碼邏輯的程式設計師（或程式）才能夠正確解讀，例如圖檔可以由看圖程式解讀
    - 因此，非文字檔較具有保密性。所以，某些遊戲將遊戲進度等採用非文字檔方式存放，這樣一來，不但可以於電源消失時保有資料，對於一般人或編輯器而言，也無法解讀檔案內的資料。
    - 這樣做的好處在於，遊戲資料不致被其他人快速破解而違背或喪失遊戲原本的規劃。

12.3.4序列化

- 類似地，對於寫入非文字檔格式，Java提供了一種「序列化」機制，它可以將物件完整寫入檔案中（採用位元串流方式），等到其他程式需要時，則可以將之取出，而一般人若要直接閱讀檔案，則無法理解檔案內的物件資料。
  - 因此，當物件需要存放於磁碟或於網路中傳輸時，通常會使用序列化技術來完成。

12.3.4序列化

- Java序列化功能的相關技術最主要的是Serializable介面，任何類別只要實作Serializable介面就可以享有預設序列化功能。
  - 而預設序列化則提供了最基本的完整寫入物件以及完整讀取物件的能力。
  - 至於更進階的序列化設計則超越本書介紹範圍，暫且不提。
    - 雖然採用位元串流的序列化提供了非文字檔的一般保密性，但它並不是絕對的保密，對於有經驗的Java程式設計師而言，仍舊可能讀取這些物件資料
    - 而預設的序列化寫入物件功能，只提供了完整寫入物件的功能（包含被宣告為private的資料），如果您不想要某些私有資料因此被竊取，可以利用transient關鍵字，強迫它在寫入物件時不被寫入，而由於未被寫入，因此也就不會被讀取。

12.3.4序列化

- readObject()與writeObject()是執行讀取與寫入物件的methods，類別只要提供了這兩個methods，當進行序列化時，這兩個methods就會自動發生作用。
  - 而本章重點在於檔案的讀寫，對於位元串流而言，有兩個重要的類別分別可以用來讀取及寫入物件，分別是ObjectInputStream與ObjectOutputStream類別，而它們分別提供了readObject()與writeObject()方法。

12.3.4序列化

- ObjectOutputStream類別與writeObject()方法
  - 由圖12-3中，我們可以發現ObjectOutputStream類別繼承自OutputStream類別，其建構子及writeObject()方法分別如下：

```
ObjectOutputStream(OutputStream out)          // ObjectOutputStream建構子
void writeObject(Obecjt obj)                  //寫入物件到位元串流
```

- ObjectInputStream類別與readObject()方法
  - 由圖12-3中，我們可以發現ObjectInputStream類別繼承自InputStream類別，其建構子及readObject()方法分別如下：

```
ObjectInputStream(InputStream in)          // ObjectInputStream建構子
Object readObject()                        //由位元串流讀出物件並回傳
```


12.3.4序列化

- 接下來，我們直接透過兩個範例來理解物件序列化的功能，首先我們將寫入物件，然後再由另一個範例讀取該物件。
- 【觀念範例12-12】：使用序列化寫入物件。
- 範例12-12：ch12_12.java（隨書光碟 myJava\ch12\ch12_12.java）

```
1   /*  檔名:ch12_12.java          功能:序列化  */
2
3   package myJava.ch12;
4   import java.lang.*;
5   import java.io.*;
6
7   public class ch12_12          //主類別
8   {
9       public static void main(String args[]) throws IOException
10      {
11        FileOutputStream fo =
              new FileOutputStream("c:\\myJava\\ch12\\file\\file6.tmp");
12        ObjectOutputStream oos = new ObjectOutputStream(fo);
13
```

12.3.4序列化

```
14          oos.writeObject(new CMyStudent("S9903501","王大民",89,84,75));
15          oos.writeObject(new CMyStudent("S9903502","郭小志",77,69,87));
16          oos.writeObject(new CMyStudent("S9903503","胡小龍",65,68,77));
17
18          oos.close();
19          fo.close();         //關檔
20      }
21  }
22
23  class CMyStudent implements Serializable
24  {
25      private String id;
26      private String name;
27      private int scoreComputer;
28      private int scoreMath;
29      private int scoreEnglish;
30      private int scoreSum;
31
32      public CMyStudent(String str1,String str2,int i,int j,int k)
33      {
34          id=str1;
35          name=str2;
36          scoreComputer=i;
37          scoreMath=j;
38          scoreEnglish=k;
39          computeSum();
40      }
```

CMyStudent實作Serializable介面

12.3.4序列化

12.3.4序列化

```
41      public void computeSum()
42      {
43          scoreSum=scoreComputer+scoreMath+scoreEnglish;
44      }
45      public void printSum()
46      {
47          //空敘述
48      }
49  }
```

- 執行結果：（執行完畢產生file6.tmp檔）
- 範例說明：
  - (1)第23~49行為自定的CMyStudent類別，包含資料、建構子、兩個成員方法。其中printSum()方法為空的敘述。

12.3.4序列化

  - (2)第14~16行透過writeObject()將三個物件寫入位元串流，由於未使用緩衝區，故會馬上寫入file6.tmp檔案內。若直接透過純文字編輯器觀看file6.tmp，會出現一些亂碼，如果使用UntraEdit等編輯器開啟file6.tmp，則可以看見實際存在的位元資料，但解讀不易
    - 若讀者有興趣，可以在網路上參閱「侯捷所著之Java的物件永續之道」文章，當中有解析實際存入物件時的格式細節。
  - 不論如何，對於一般使用者而言，file6.tmp已經具備一些保密性。我們將於下一個範例中，讀出這些物件。

12.3.4序列化

- 【觀念範例12-13】：使用序列化讀出物件。
- 範例12-13：ch12_13.java（隨書光碟 myJava\ch12\ch12_13.java）

```
1   /*  檔名:ch12_13.java          功能:序列化  */
2
3   package myJava.ch12;
4   import java.lang.*;
5   import java.io.*;
6
7   public class ch12_13          //主類別
8   {
9       public static void main(String args[])
                        throws IOException,ClassNotFoundException
10      {
11          FileInputStream fi =
              new FileInputStream("c:\\myJava\\ch12\\file\\file6.tmp");
12          ObjectInputStream ois = new ObjectInputStream(fi);
13          CMyStudent obj1;
14
```


12.3.4序列化

```
15          obj1=(CMyStudent)ois.readObject();
16          obj1.printSum();
17          obj1=(CMyStudent)ois.readObject();
18          obj1.printSum();
19          obj1=(CMyStudent)ois.readObject();
20          obj1.printSum();
21
22          ois.close();
23          fi.close();         //關檔
24      }
25  }
26
27  class CMyStudent implements Serializable
28  {
29      private String id;
30      private String name;
31      private int scoreComputer;
32      private int scoreMath;
33      private int scoreEnglish;
34      private int scoreSum;
35
```

12.3.4序列化

```
36      public CMyStudent(String str1,String str2,int i,int j,int k)
37      {
38
39          id=str1;
40          name=str2;
41          scoreComputer=i;
42          scoreMath=j;
43          scoreEnglish=k;
44          computeSum();
45
46      }
47      public void computeSum()
48      {
49          scoreSum=scoreComputer+scoreMath+scoreEnglish;
50      }
51      public void printSum()      //修改method內容,但method不可或缺
52      {
53            System.out.println(id + " " + name + " 總分=" + scoreSum);
54      }
55  }
```


12.3.4序列化

- 執行結果：

```
S9903501 王大民 總分=248
S9903502 郭小志 總分=233
S9903503 胡小龍 總分=210
```

- 範例說明：
  - (1)由於程式中執行了readObject()，而它可能引發ClassNotFoundException例外，故需要在main()函式後面加上例外丟出的程式碼。
  - (2)第27~55行為自定的CMyStudent類別，和上一個範例僅有printSum()的內容不同。
  - (3)第15、17、19行透過readObject()將檔案（位元串流）內的三個物件依序讀出，並在第16、18、20行執行printSum()方法，請注意，我們編譯及執行範例的順序為編譯ch12_12.java（此時會產生ch12_12.class與CMyStudent.class）、執行ch12_12，接著再編譯ch12_13.java（此時會產生ch12_13.class與CMyStudent.class，其中CMyStudent.class將會覆蓋上一個範例的同一類別檔）、執行ch12_13。故此時執行到printSum()時，會印出總分，如同執行結果。

12.3.4序列化

- (4)在這個範例中，我們可以得知，重新讀取物件後，可以重新撰寫某個method的內容，由於讀入的物件資料不包含method的原有內容，因此會執行新的method內容。
- (5)假設我們省略兩個範例中的任一欄位或任一method，則當執行本範例時將會發生執行時期的錯誤，原因是當存入物件時，會將所有欄位都寫入，並且所有的method也會寫入，但不包含method內容。
- (6)由這個範例可以發現，除非程式設計師知道原有物件的所有欄位及method，否則將無法正確讀出存放於檔案內的物件，因此具有保密性。

12.4 本章回顧

- 在本章中，我們介紹了Java對於檔案存取的支援技術，對於Java而言，檔案存取是透過串流來完成，而串流又分為字元串流與位元串流。
  - 其中，字元串流只能用來讀寫純文字檔，而位元串流則無限制。
    - 不過，對於存取文字檔而言，使用字元串流可以使用更多好用的成員方法，方便撰寫程式。

12.4 本章回顧

- 字元串流對應的類別為Reader/Writer類別及其子孫類別，而位元串流對應的類別則為InputStream/OutputStream類別及其子孫類別。
- 除了在進行串流讀寫時，立即反應於檔案外，我們也可以透過緩衝區來改善程式的效率
  - 也就是藉由緩衝區減少對檔案的讀寫次數，由於記憶體的存取時間較磁碟快許多，因此，我們可以利用緩衝區改善效率

12.4 本章回顧

  - 不過在寫入資料時，記得要在關閉檔案前執行flush()，強迫將緩衝區剩餘的資料寫入檔案，否則有可能會遺失最後的資料。
- 除了寫入一般資料外，我們也可以透過Java提供的序列化技術寫入物件到檔案中，於日後程式需要時再將之讀出。
  - 由於採用的是位元串流方式寫入，因此具有保密性，並且由於讀出時，必須了解當時類別的詳細規格，因此，即使是程式設計師，也不容易竊取物件資料。
- 讀寫檔案可能會發生例外的狀況，因此我們可以自行處理例外或在方法後面以throws標記，將例外轉交給呼叫method的程式來處理
  - 而JDK7為了方便程式設計師處理這類因資源存取而發生的例外，也提供了try-with-resources機制。

12.4 本章回顧

- 檔案不但是永久保存資料的方式，也是大型程式常見的暫存中繼格式，而Java對於檔案採用的串流概念，也將適用於網路程式，因為對於Java而言，這些資料都是由外部提供，而非原本就存在於記憶體中。
  - 礙於篇幅所限，若讀者對於串流方式有更進一步的需求，可以查閱Java I/O的專書。若對於網路程式設計有興趣，可以參閱「網路程式設計初學指引-使用Java」一書。

本章結束

Q&A討論時間