Java 物件導向程式設計

1. Java 的基本語法

1.1 前言

Java 是一種物件導向語言，Java/C#/C++ 等語言都是由 C 語言衍生而來的，因此其語法與 C 語言相當類似。

這些從 C 衍生出來的語言都具備某些共同的特性，例如使用 char, int, float 作為基本型態，使用大括弧 {…} 代表區塊結構，使用方括弧 […] 代表陣列元素用圓括弧 (…) 作為運算式結構，然後用 if 作為條件控制結構，用 for, while 作為迴圈結構，使用 type f(arg1, …, argk) { … } 作為函數定義結構。

舉例而言，以下的程式片段在 C/Java/C#/C++ 當中都是完全一致的，當您將這些程式碼正確的嵌入到語言當中時，將可以完全不需修改程式碼就能正確的編譯。

```
int sum(int n) {
   int s=0, i;
   for (i=1; i<=n; i++) {
         s += i;
   }
}

float max(float a, float b) {
   if (a > b)
      return a;
   else
      return b;
}
```

但是 Java 與 C 語言也有相當程度的不同點，這些不同點主要是由於物件導向這個概念而來的。例如 Java 的程式一定要放在物件結構當中，而且具有不同的函式庫架構。因此像下列的 C 語言程式就無法直接被 Java 編譯器正確的編譯，當然也就無法正確的執行了。

```
#include <stdio.h>
int main() {
```

```
    printf("Hello!");
}
```

如果要將上述程式寫成 Java 的版本，可以使用如下的語法。這個語法與 C 語言有兩個明顯的差異，第一個是整個程式被一個 class Hello {…} 的結構所包住，第二個是 printf() 函數被改為 System.out.println() 函數。

```
class Hello {
    public static void main(String[] args)   {
        System.out.println("Hello!");
    }
}
```

如果您更仔細的查看，將會發現一些更為細緻的差異，像是 public static 等關鍵字，以及 main 函數的參數 String[] args 等，這些都是 Java 與 C 語言有明顯不同的地方。

我們將在以下的各個小節中，進一步的說明 Java 與 C 語言的不同點，造成這些不同點的關鍵因素可以說只有一個，那就是物件導向的概念。

1.2 基本型態

在 C 語言當中，有 char (字元), short(短整數), int (整數), long (長整數), float (浮點數) 、double (雙精度浮點數)等基本型態，這些型態在 Java 當中也能正常的使用，但是 Java 還多了 boolean (布林) 與 byte (位元組) 這兩個型態，其中 boolean 用來代表真假值，而 byte 則用來代表 8 個 bit 所形成的位元組。

即便如此，C 語言中的基本型態與 Java 有少許的不同點。舉例而言，在 C 語言當中，char 占用的空間是 1 個位元組 (8 個 bit)，但是在 java 當中由於採用了 Unicode 的關係，因此每個 char 佔用兩個位元組 (byte)。

Java 當中的布林值只能是 true 或 false 兩者之一，不像 C 語言當中用 0 代表假值，1 (或者非零) 代表假值。

C 語言當中的 long 有時候只佔 32 bits，但是在 Java 當中的 long 則必定佔用 64 bits 的空間。

除此之外，Java 由於有物件的概念，因此具有字串 String 物件，不像 C 語言中必須使用 char* 型態代表字串。這些物件可以透過 new 關鍵字建立後，就能在程式中被使用，而且不需要使用像 free() 這樣的函數去釋放記憶體，因為 Java 具有垃圾回收機制，可以自動回收記憶體中不再被使用到的空間，因此撰寫 Java 程式比較不容易出現某些與記憶體相關的錯誤。

C 語言的指標型態，通常是在基本型態後加上星號 (*) 所達成的，但是在 Java 當中沒有指標這種型態，因為指標雖然是很強大的技術，但是卻會造成安全性的問題，以及記憶體回收上的困難，因此並

沒有被納入到 Java 語言當中。

1.3 流程控制

Java 在流程控制上，主要仍採用 C 語言的 if, while, for, do {…} while 等結構，因此其寫法幾乎與 C 語言完全一致，所以我們將不另外進行說明。

但是 Java 有一個 C 語言所沒有的 for 迴圈控制語法，這種語法在 Python, C# 等語言當中稱為 foreach，這種語法可以循序取出集合中的每個元素，用法比傳統的 for 語法更為簡潔，以下的 Java 的範例說明了這種 for 語法的使用方式。

```
class ForEach1 {
   public static void main(String[] args) {
      double[] a = {1.2, 3.0, 0.8};
      double sum = 0.0;
      for (double x : a) {
         sum += x;
      }
      System.out.println("sum="+sum);
   }
}
```

上述程式的編譯執行過程如下所示。

```
D:\ccc>javac ForEach1.java

D:\ccc>java ForEach1
sum=5.0
```

1.4 陣列

Java 的陣列宣告與 C 語言很像，但是又有少許不同。舉例而言，以下是一個具有合法陣列宣告的 C 語言程式。這個程式的 a 宣告可以在 Java 當中合法的使用，但是 b 宣告卻不是合法的 Java 語句，因為在 Java 當中陣列的大小是不需要宣告的。

```
int main() {
      int a[] = {1, 2, 3, 4, 5};
      int b[2][2] = {{1, 2}, {3, 4}};
}
```

在 Java 程式中，陣列宣告的 [] 符號可以被置放在變數的前面，但在 C 語言中卻不行，因此下列 Java

程式中的 c,d,e 是合法的 Java 宣告式，但卻不是合法的 C 語言宣告式。

```
class Array {
   public static void main(String[] args)   {
      int a[] = {1, 2, 3, 4, 5};
      int b[][] = {{1, 2}, {3, 4}};
      int[] c = {1, 2, 3, 4, 5};
      int d[][] = {{1, 2}, {3, 4}};
      int[][] e = {{1, 2}, {3, 4, 5, 6}};

      System.out.println("a.length="+a.length);
      System.out.println("b.length="+b.length);
      System.out.println("e[1].length="+e[1].length);
   }
}
```

上述程式中的宣告 b 雖然在 Java 當中是合法的，但是卻無法被某些 C 語言編譯器正確編譯，因為有些編譯器無法自動決定該二維陣列的大小。

雖然宣告 e 這個二維陣列的第一列與第二列的大小不一致，但卻是一個合法的 Java 陣列宣告，因為 Java 的二維陣列並非單純的線性結構，而是一個具有很多陣列元素陣列，因此每一列的大小不需要一樣，這點與 C 語言有相當大的不同。

在 C 語言當中，我們無法於執行時期決定一個陣列的大小，但是在 Java 當中可以，我們只要使用 .length 就可以取得某個陣列的子元素個數。

上述範例程式的編譯與執行過程如下所示，請仔細觀察其內容，自然可以理解 Java 的陣列與 C 語言有何不同了。

```
D:\ccc\JA>javac Array.java

D:\ccc\JA>java Array
a.length=5
b.length=2
e[1].length=4
```

1.5 字串

C 語言中並沒有所謂的字串，通常我們使用字元指標 char* 代表字串，這種做法可以節省一些記憶空間，但也讓字串成為一個不太容易使用的形態。在 Java 當中，具有 String 這樣的字串物件，這讓我們很容易在 Java 中使用字串，而且不需要擔心空間分配的問題。

以下是一個 Java 中使用字串的範例，這個範例可以將一個陣列轉為字串後輸出。

```
class String1 {
   public static void main(String[] args) {
      double[] a = {1.2, 3.0, 0.8};
      String str = "[";
      for (double x : a) {
         str += x + " ";
      }
      str = str.trim() + "]";
      System.out.println("str="+str);
   }
}
```

上述程式的編譯執行過程如下所示。

```
D:\ccc>javac String1.java

D:\ccc>java String1
str=[1.2 3.0 0.8]
```

1.6 輸出入

C 語言中的輸出入通常依靠 printf(), scanf() 等函數，這是 C 語言當中相當好用的函式庫。但是 Java 的函式庫並沒有採用這種方式，因為在 Java 當中，字串可以透過加法符號 + 進行連接的動作，因此我們可以方便的將想要輸出的參數直接透過 + 符號連接後輸出及可。以下是一個 Java 的輸出入範例程式，該範例可以讓使用者輸入姓名後，印出打招呼的語句。

```
import java.io.*;

class IO1 {
   public static void main(String[] args) throws Exception {
      System.out.print("請輸入姓名：");
      InputStreamReader is = new InputStreamReader(System.in);
      BufferedReader in = new BufferedReader(is);
      String name = in.readLine();
      System.out.println(name+"您好，很高興見到您!");
   }
}
```

上述程式的執行結果如下所示。

```
D:\ccc>javac IO1.java

D:\ccc>java IO1
請輸入姓名：王小明
王小明您好，很高興見到您!
```

1.7 錯誤處理

C 語言當中雖然具有某些錯誤處理機制，像是 setjump, longjump, signal 等，但由於這是高等技巧，使用上並不方便，一般人通常不會去使用它。為了讓錯誤處理更加容易，在 Java 語言當中使用了一種稱為 try {… } catch {…} 的錯誤捕捉機制，這種語法可以讓您捕捉任何在 try {…} 區塊所發生的錯誤，然後在 catch {…} 區塊進行錯誤處理，以下是一個 Java 的錯誤處理範例。

```
import java.io.*;

class Try1 {
   public static void main(String[] args) {
      try {
         int a[] = {1,2,3};
         int b = a[8];
         System.out.println("b="+b);
      } catch (Exception e) {
         System.out.println("Error:"+e);
      }
   }
}
```

在上述程式中，由於陣列 a 只有三個元素，也就是 a[0], a[1], a[2]，因此根本沒有 a[8] 這個元素，當程式執行 b=a[8] 這個指令時，就會引發錯誤而跳到 catch 區塊當中。由於這個錯誤被放入變數 e 中，因此才會印出 Error:java.lang.ArrayIndexOutOfBoundsException: 8 這個錯誤訊息。

```
D:\ccc>javac Try1.java

D:\ccc>java Try1
Error:java.lang.ArrayIndexOutOfBoundsException: 8
```

2. Java 的物件概念

2.1 簡介

在程式語言的發展歷史上，大致可以分為三個階段，第一階段是草創時期，這個時期直接將組合語言中的 jump 指令，提升為高階語言中的 goto 跳躍指令，這種程式由於結構混亂不容易理解，因而導致第二階段的結構化程式語言興起。

結構化程式語言是採用 if, for, while, 函數等結構化語句撰寫程式的方法，C 語言便是一種結構化程式語言。雖然結構化語言已經相當好用，但是在撰寫程式時仍然不容易構思，因為整個結構化的思想建構在流程控制之上，但是真實的世界往往是由「物體」與「概念」所組成的，因此在寫程式的過程中就發生了難以將現實世界對應到程式流程上的問題，於是進入的第三階段的物件導向時期。物件導向的概念從 ALGOL 語言開始逐漸形成，直到 Smalltalk 80 時完全成熟了。後來，許多人試圖去擴充 C 語言成為物件導向式的語言，於是乎就誕生了 C++, Objective C, Java, C# 等語言。

為了理解這三個階段的程式設計方式，且讓我們舉一個例子，以便說明程式語言的演進過程。以下我們分別用 (1) 非結構化 (2) 結構化 (3) 物件導向 的方式，撰寫出一個具有堆疊功能的物件程式。

(1) 非結構化

```
int stack[100];
int top = 0;

void push(int o) {
   printf("push(%d)\n", o);
   stack[top++] = o;
   return;
}

int pop() {
   top--;
   int o = stack[top];
   printf("pop(%d)\n", o);
   return o;
}

int main() {
   int a[] = {1,2,3,4,5};
   int i=0, o;
```

(2) 結構化

```
int stack[100];
int top = 0;

void push(int o) {
   printf("push(%d)\n", o);
   stack[top++] = o;
   return;
}

int pop() {
   top--;
   int o = stack[top];
   printf("pop(%d)\n", o);
   return o;
}

int main() {
   int a[] = {1,2,3,4,5};
   int i;
   for (i=0; i<5; i++)
```

(3) 物件導向

```
class Stack {
   int stack[] = new int[100];
   int top = 0;

   void push(int o) {
      System.out.println("push("+o+")");
      stack[top++] = o;
      return;
   }

   int pop() {
      top--;
      int o = stack[top];
      System.out.println("pop("+o+")");
      return o;
   }

   public static void main(String[] args) {
      Stack s = new Stack();
      int a[] = {1,2,3,4,5};
```

```
PUSH_LOOP:
   push(a[i]);
   i++;
   if (i<5)
      goto PUSH_LOOP;
POP_LOOP:
   pop(a);
   if (top>0)
      goto POP_LOOP;
}
```

```
      push(a[i]);
   while (top > 0)
      pop(a);
}
```

```
      int i;
      for (i=0; i<5; i++)
         s.push(a[i]);
      while (s.top > 0)
         s.pop();
   }
}
```

上述程式的執行結果如下所示。

```
push(1)
push(2)
push(3)
push(4)
push(5)
pop(5)
pop(4)
pop(3)
pop(2)
pop(1)
```

透過上述的程式，讀者應該可以很容易的理解程式語言演進的三階段過程了。現在，就讓我們將焦點放在物件導向的特性上，進一步的理解物件導向的程式設計方式吧！

2.2 封裝

物件導向技術具有三個主要的特性，這三個特性分別是「封裝」、「繼承」與「多型」，其中的封裝是指「將程式與資料封裝在一個稱為物件的結構當中」。舉例而言，假如我們希望建立一個圓形物件，可以根據半徑 r 計算圓的面積 area()，那麼我們就可以透過下列程式，將半徑 r 與面積函數 area() 封裝在一個稱為圓 (Circle) 的類別當中，這個類別就是物件的樣版結構。

檔案：Circle1.java

```
class Circle {
   double r;

   double area() {
      return 3.14 * r * r;
   }
```

```
    public static void main(String[] args) {
        Circle c = new Circle();
        c.r = 1.0;
        System.out.println("c.area()="+c.area());
    }
}
```

上述程式的編譯執行過程如下所示。

```
D:\ccc>javac Circle1.java

D:\ccc>java Circle
c.area()=3.14
```

在上述程式中，我們必須在用 Circle c = new Circle(); 指令之後，再用 c.r = 1.0 以設定 r 欄位的值。其實，我們只要在上述程式中加入一個與 Circle 同名的函數，就可以在 new 指令中直接指定半徑 r 的內容了，這種函數就稱為建構函數。

檔案：Circle1.java

```
class Circle {
    double r;

    Circle(double r) {
        this.r = r;
    }

    double area() {
        return 3.14 * r * r;
    }

    public static void main(String[] args) {
        Circle c = new Circle(1.0);
        System.out.println("c.area()="+c.area());
    }
}
```

上述程式的編譯執行過程如下所示。

```
D:\ccc>javac Circle2.java

D:\ccc>java Circle
c.area()=3.14
```

2.3 繼承

假如我們定義了一個動物的類別 Animal，這個動物有個名稱 name，那麼我們就可以用下列程式宣告一些動物並且列印出每個動物的名稱。

檔案：Animal1.java

```
class Animal {
   String name;

   Animal(String name) {
      this.name = name;
   }

   public static void main(String[] args) {
      Animal[] animals = { new Animal("鼠"), new Animal("牛"), new Animal("虎") };
      for (Animal a : animals)
         System.out.println(a.name);
   }
}
```

上述程式的編譯執行過程如下所示。

```
D:\ccc>javac Animal1.java

D:\ccc>java Animal
鼠
牛
虎
```

接著讓我們說明繼承的觀念，假如我們想再定義一種動物稱為貓 (Cat)，那麼我們就可以繼承原本的 Animal 類別，然後衍生出貓的特殊屬性即可。舉例而言，我們可以將上述程式擴充如下，就可以加入一個稱為 Cat 的類別。同樣的，我們也可以加上狗的類別，於是得到了下列程式。

檔案：Animal2.java

```
class Animal {
   String name;

   Animal(String name) {
      this.name = name;
```

```
    }

    public static void main(String[] args) {
        Animal[] animals = { new Animal("鼠"), new Cat(), new Dog() };
        for (Animal a : animals)
            System.out.println(a.name);
    }
}

class Cat extends Animal {
    Cat() { super("貓"); }
}

class Dog extends Animal {
    Dog() { super("狗"); }
}
```

上述程式的編譯執行過程如下所示。

```
D:\ccc>javac Animal2.java

D:\ccc>java Animal
鼠
貓
狗
```

2.4 多型

接著，假如我們希望為動物們加上叫聲，於是我們可以為動物、狗和貓分別加上 speak() 函數，讓它們可以「說話」，於是程式修改如下所示。

檔案：Animal3.java

```
class Animal {
    String name;

    Animal(String name) {
        this.name = name;
    }

    String speak() { return name+"叫"; }
```

```
    public static void main(String[] args) {
        Animal[] animals = { new Animal("鼠"), new Cat(), new Dog() };
        for (Animal a : animals)
            System.out.println(a.speak());
    }
}

class Cat extends Animal {
    Cat() { super("貓"); }
    String speak() { return name+"叫喵喵"; }
}

class Dog extends Animal {
    Dog() { super("狗"); }
    String speak() { return name+"叫汪汪"; }
}
```

上述程式的編譯執行過程如下所示。

```
D:\ccc>javac Animal3.java

D:\ccc>java Animal
鼠叫
貓叫喵喵
狗叫汪汪
```

在此，必須請讀者仔細觀察上述程式，特別將焦點放在這三個物件的 speak() 函數上。由於我們分別為 Animal, Cat, Dog 分別加上了 speak() 函數，而且都具有不同的行為，因此在主程式的 for (Animal a : animals) System.out.println(a.speak()); 這兩行程式上，java 虛擬機會根據物件的類型，分別呼叫 Animal, Cat, Dog 特有的 speak() 函數，於是讓同一個 speak() 指令產生了不同的行為，這種特性就被稱為多型，這是物件導向中非常有用的一種技術。我們將會在後面的小節中看到這種技巧的強大用途。

2.5 介面

有時候，我們在定義上層的類別時，會定義出一種稱為抽象類別的東西，因為我們雖然知道下層類別會具有這些函數，但是卻由於抽象性的緣故而無法實作出某些函數，這時候就可以將上層類別定義為抽象類別，這種類別可以擁有不具有內容的抽象函數。舉例而言，下列程式中的 Shape 就是個抽象類別，所以在 class 前面加上了 abstract 這個關鍵字，而其中的 area() 函數也被加上了 abstract 關鍵字，代表該函數是一個抽象函數，因此我們不需要定義 area() 函數的程式碼，也就是 area() 函數可以沒有 {…} 所框住的內容，直接用分號 「;」 結尾即可。

檔案：ShapeAbstract.java

```
abstract class Shape {
   abstract double area();
}

class Circle extends Shape {
   double r;
   public Circle(double r) {
      this.r = r;
   }
   public double area() {
      return 3.14 * r * r;
   }
}

class Rectangle extends Shape {
   double width, height;
   public Rectangle(double w, double h) {
      this.width = w;
      this.height = h;
   }
   public double area() {
      return width*height;
   }
}

class ShapeTest {
   public static void main(String[] args) {
      Shape shapes[] = { new Circle(1.0), new Rectangle(2.0, 3.0) };
      for (Shape s : shapes)
         System.out.println("s.area()="+s.area());
   }
}
```

上述程式的編譯執行過程如下所示。

```
D:\ccc>javac ShapeAbstract.java

D:\ccc>java ShapeTest
s.area()=3.14
```

```
s.area()=6.0
```

在某種更極端的情況之下，當抽象類別內只有抽象函數，但是沒有任何欄位時，我們也可以將該抽象類別定義為「介面 」 (interface)，此時其繼承者將不能以 extends 進行繼承動作，而必須改用 implements 關鍵字，以下程式是我們將上述程式中的 Shape 改用 interface 定義後的結果。

檔案：ShapeInterface.java

```
interface Shape {
   double area();
}

class Circle implements Shape {
   double r;
   public Circle(double r) {
      this.r = r;
   }
   public double area() {
      return 3.14 * r * r;
   }
}

class Rectangle implements Shape {
   double width, height;
   public Rectangle(double w, double h) {
      this.width = w;
      this.height = h;
   }
   public double area() {
      return width*height;
   }
}

class ShapeTest {
   public static void main(String[] args) {
      Shape shapes[] = { new Circle(1.0), new Rectangle(2.0, 3.0) };
      for (Shape s : shapes)
         System.out.println("s.area()="+s.area());
   }
}
```

上述程式的編譯執行過程如下所示。

```
D:\ccc>javac ShapeInterface.java

D:\ccc>java ShapeTest
s.area()=3.14
s.area()=6.0
```

由於 Java 是一個採用單一繼承法的程式語言，因此一個類別不能 extends 多個父類別，但是卻可以用 implements 實作多個介面，因此當基礎類別是完全抽象的類別時，最好以 interface 進行定義，如此才能讓程式具有更高的彈性。

3. Java 的函式庫

對於物件導向程式設計者而言，最難學習的不是語言，而是為數龐大的函式庫。Java 的函式庫非常龐大，幾乎沒有任何一個人可以學得完，但事實上也沒有需要學完，只要能掌握一些重要的物件，就可以寫出大部分的 Java 程式了。在本單元中，我們將會介紹最重要的幾個 Java 物件，讓讀者能以最快的速度進入 Java 函式庫的世界。

3.1 字串處理

字串是 Java 函式庫中最常用的物件，因為大部分的程式都必須進行某些字串處理，下表列出了 Java 中最常使用到的字串函數，。

函數	說明	範例	傳回值
length()	字串長度	"abc".length()	3
equals()	判斷兩字串是否相等	"abc".equals("abc");	true
toLowerCase()	字串轉小寫	"ABC".toLowerCase();	abc
toUpperCase()	字串轉大寫	"abc". toUpperCase();	ABC
indexOf()	尋找子字串的位置	"ABC".indexOf("BC");	1
charAt()	第 i 個字元	"ABC".charAt(2);	C
compareTo()	字串比較	"ABC".compareTo("abc");	-32
split()	字串分割	"a,b,c".split(",")	[a, b ,c]
matches()	字串比對	"abc".matches("[a-z]+");	true
replaceAll()	全部取代	"abca".replaceAll("a", "e");	ebce
replaceFirst()	取代第一個	"abca".replaceFirst("a", "e");	ebca
format()	格式化	String.format("%6.2f", 1.2345);	1.23

為了展式上述函數的功能，我們撰寫了一個測試程式，以幫助讀者從實務練習中理解這些函數的意義。

```
import java.util.*;
```

```
class StringTest {
   public static void main(String[] args) {
      String abc = "abc", ABC="ABC", abca="abca";
      System.out.println("abc.length()="+abc.length());
      System.out.println("abc.equals('abc')="+abc.equals("abc"));
      System.out.println("ABC.toLowerCase()="+ABC.toLowerCase());
      System.out.println("abc.toUpperCase()="+abc.toUpperCase());
      System.out.println("ABC.indexOf('BC')="+ABC.indexOf("BC"));
      System.out.println("ABC.compareTo('abc')="+ABC.compareTo(abc));
      System.out.println("'a,b,c'.split(',')="+Arrays.asList("a,b,c".split(",")));
      System.out.println("abc.matches('[a-z]+')="+abc.matches("[a-z]+"));
      System.out.println("abca.replaceAll('a', 'e')="+abca.replaceAll("a", "e"));
      System.out.println("abca.replaceFirst('a', 'e')="+abca.replaceFirst("a", "e"));
      System.out.println("String.format('%6.2f', 1.2345)="+String.format("%6.2f", 1.2345));
   }
}
```

上述程式的編譯執行過程如下所示。

```
D:\ccc\javaBook>javac StringTest.java

D:\ccc\javaBook>java StringTest
abc.length()=3
abc.equals('abc')=true
ABC.toLowerCase()=abc
abc.toUpperCase()=ABC
ABC.indexOf('BC')=1
ABC.compareTo('abc')=-32
'a,b,c'.split(',')=[a, b, c]
abc.matches('[a-z]+')=true
abca.replaceAll('a', 'e')=ebce
abca.replaceFirst('a', 'e')=ebca
String.format('%6.2f', 1.2345)=    1.23
```

3.2 資料結構

常見的資料結構包含陣列、串列、堆疊、搜尋樹、雜湊表等， 這些結構在 Java 函式庫當中都已經定義在 java.util.* 這個函式庫中了，您只要知道何時應該用何種結構即可，下表列出了筆者常用的一些結構。

類別	資料結構	說明
Vector	動態陣列	可伸縮大小的循序結構的陣列。
LinkedList	鏈節串列	一個接一個的串接結構
Stack	堆疊	後進先出的堆疊結構
HashMap	雜湊表	使用雜湊函數 (Hash Function) 查找資料的表格
TreeMap	紅黑樹	使用紅黑樹儲存資料的二元樹狀結構

檔案：UtilTest.java

```
import java.util.*;

class UtilTest {
   public static void main(String[] args) {
      String[] peoples = {"John", "Mary", "George"};
      int[] scores = { 75, 55, 90 };
      Vector vector = new Vector(Arrays.asList(peoples));
      LinkedList list = new LinkedList(Arrays.asList(peoples));
      System.out.println("vector="+vector);
      System.out.println("list="+list);
      HashMap hashMap = new HashMap();
      TreeMap treeMap = new TreeMap();
      for (int i=0; i<peoples.length; i++) {
            hashMap.put(peoples[i], scores[i]);
            treeMap.put(peoples[i], scores[i]);
      }
      System.out.println("hashMap="+hashMap);
      System.out.println("treeMap="+treeMap);
      System.out.println("hashMap.get('Mary')="+hashMap.get("Mary"));
      System.out.println("treeMap.get('Peter')="+treeMap.get("Peter"));
   }
}
```

上述程式的編譯執行過程如下所示。

```
D:\ccc\javaBook>javac UtilTest.java
Note: UtilTest.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

D:\ccc\javaBook>java UtilTest
vector=[John, Mary, George]
list=[John, Mary, George]
hashMap={Mary=55, John=75, George=90}
```

```
treeMap={George=90, John=75, Mary=55}
hashMap.get('Mary')=55
treeMap.get('Peter')=null
```

3.3 檔案處理

Java 的基本輸出入物件為 File 與 RandomAccessFile，其中的 File 除了用來代表檔案之外，也可以用來代表資料夾 (目錄)。

大致上來說，Java 的輸出入物件可以分為兩類，第一類是二進位輸出入物件，稱為串流 (Stream)，第二類是文字型輸出入物件，通常其讀取器被稱為 Reader，寫入器被稱為 Writer。

這些物件可以互相套疊，以變得到您所想要的功能，以下是 Java 中常見的輸出入物件列表。

二進位輸出入 (串流 Stream)		文字型輸出入 (Reader & Writer)	
輸入	輸出	輸入	輸出
InputStream	OutputStream	Reader	Writer
FileInputStream	FileOutputStream	FileReader	FileWriter
DataInputStream	DataOutputStream	InputStreamReader	OutputStreamWriter
BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
ObjectInputStream	ObjectOutputStream		

Java 的輸出入函式庫架構相當複雜，常常讓初學者感到相當困擾，為了讓使用者理解這些物件的使用方式， 我們撰寫了一個 FileTest.java 程式，可以將其自身印出並轉換成 UTF-8 的 Unicode 格式儲存在 FileTestUtf8.java 當中，以下是該程式的內容。

檔案：FileTest.java

```
import java.io.*;

public class FileTest {
   public static void main(String args[]) throws Exception {
      String text = fileToText("FileTest.java");
      System.out.println(text);
      textToFile(text, "FileTestUtf8.java", "UTF-8");
   }

   public static String fileToText(String fileName) throws Exception {
      File f = new File(fileName);
      int length = (int)(f.length());
      FileInputStream fin = new FileInputStream(f);
```

```
        DataInputStream in = new DataInputStream(fin);
        byte[] bytes = new byte[length];
        in.readFully(bytes);
        return new String(bytes);
    }

    public static void textToFile(String pText, String outFile, String pEncode) throws Exception {
        FileOutputStream fos = new FileOutputStream(outFile);
        Writer writer;
        if (pEncode == null)
            writer =new OutputStreamWriter(fos);
        else
            writer = new OutputStreamWriter(fos, pEncode);
        writer.close();
    }
}
```

上述程式的編譯執行過程如下所示。

```
D:\ccc\javaBook>javac FileTest.java

D:\ccc\javaBook>java FileTest
import java.io.*;

… 略 (同程式碼)
```