# 物件導向程式設計進階（一）
# 繼承、多型

人工智慧與無線感應設備開發專班

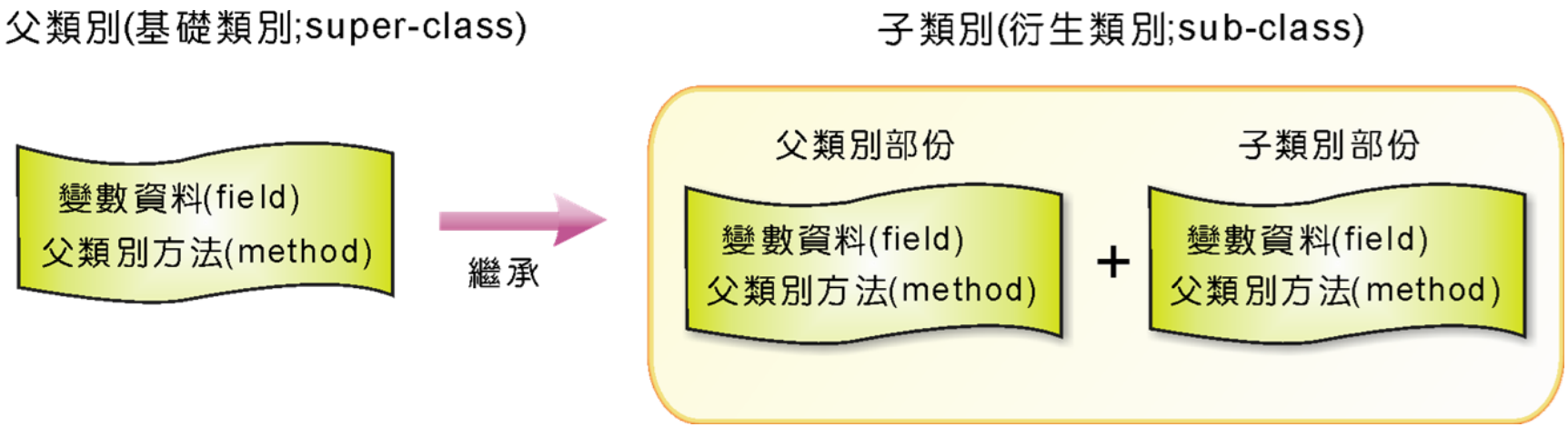
湜憶電腦知訊顧問股份有限公司

馬傳義

# 繼承

- 「繼承（Inheritance）」這種機制在「物件導向程式設計」中佔有非常重要的地位。
- 在「物件導向程式設計」中，原先已建立好的類別稱之為「基礎類別（Base Class）」，而經由繼承所產生的新類別就稱之為「衍生類別（Derived Class）」。
- 通常會將「基礎類別」稱之為「父類別」，而「衍生類別」稱之為「子類別」。
- 「子類別」除了「繼承」「父類別」的成員外，還能另外定義「父類別」所沒有的成員來使用（如下圖）。

# 繼承




- Java的「類別」都是「繼承」自「Object」這個類別，這是在「java.lang」「套件」中定義的類別。
- 一個父類別可以「衍生」很多子類別，而一個子類別「只能繼承一個」父類別。

# 繼承

- 父類別用「private」修飾子所宣告的成員，子類別並不能繼承；就算子類別所建立的資料成員，其名稱與父類別用private修飾子所宣告的資料成員「名稱相同」，兩個資料成員「彼此不影響」。
- 父類別用「protected」修飾子所宣告的成員，子類別能繼承，但該成員只能在「自身類別」和「被繼承」的「子類別」之「內部敘述使用」，可把它當成「子類別」的「私有」成員。
- 繼承可以將「已存在」的「類別成員」當成「基礎成員」或「功能」，進而「擴充」「其它功能」。

# 繼承

- extends：
  - 語法：

**存取修飾子 class 子類別名稱 extends 父類別名稱**
**{**
**敘述區段**
**}**

# 繼承

- 說明：
  - 存取修飾子：
    - 在相同「套件（package）」中（亦即同路徑下）的所有類別，都能對此類別進行存取動作。
    - Java所支援的類別存取修飾子有：
      - public：

        允許所有類別都可以存取此類別。
      - abstract：

        此類別僅能被繼承，無法直接進行實體化動作（建立物件）。
      - final：

        代表此類別無法作為其餘類別的繼承原型。
  - class：
    - 類別的關鍵字。

# 繼承

- 子類別名稱：
  - 自訂。

  注意
    - 此類別如果為主要類別，則必須與檔案名稱相符。
    - 新增類別的辨識名稱，不可與同程式或同套件（路徑）內已宣告的類別名稱互相衝突。
- extends：
  - 繼承的關鍵字。
- 父類別名稱：
  - 子類別所要向上繼承的類別名稱。
- 敘述區段：
  - 包含子類別新增的類別成員。

# 繼承

- 程式：

```
package CH07_01;

class CPenScore
{
    protected String name;
    protected int pen_score;

    public void SetPenData(String name, int score)
    {
        this.name = name;
        pen_score = score;
    }

    public void ShowPenScore()
    {
        System.out.println("\n筆試成績：");
        System.out.println("姓名：" + name);
        System.out.println("筆試分數：" + pen_score);
```

# 繼承

```
    }
}

class CTotalScore extends CPenScore
{
    private int oral_score;
    private int tot_score;

    public void SetOralData(int score)
    {
        oral_score = score;
    }

    public void ShowTotScore()
    {
        System.out.println("\n總成績：");
        System.out.println("姓名：" + name);
        System.out.println("筆試分數：" + pen_score);
        System.out.println("口試分數：" + oral_score);
```

# 繼承

```
        System.out.println("================");
        tot_score = pen_score + oral_score;
        System.out.println("總分：" + tot_score);
    }
}

public class CH07_01
{
    public static void main(String[] args)
    {
        CPenScore p1 = new CPenScore();
        p1.SetPenData("張三", 50);
        p1.ShowPenScore();

        CTotalScore p2 = new CTotalScore();
        p2.SetPenData("李四", 60);
        p2.SetOralData(20);
        p2.ShowTotScore();
    }
}
```

# 繼承

```
1  package CH07_01;
2
3  class CPenScore
4  {
5     protected String name;
6     protected int pen_score;
7
8     public void SetPenData(String name, int score)
9     {
10       this.name = name;
11       pen_score = score;
12    }
13
14    public void ShowPenScore()
15    {
16       System.out.println("\n筆試成績：");
17       System.out.println("姓名：" + name);
18       System.out.println("筆試分數：" + pen_score);
19    }
20 }
21
22 class CTotalScore extends CPenScore
23 {
24    private int oral_score;
25    private int tot_score;
26
```

# 繼承

```
27      public void SetOralData(int score)
28      {
29          oral_score = score;
30      }
31
32      public void ShowTotScore()
33      {
34          System.out.println("\n總成績：");
35          System.out.println("姓名：" + name);
36          System.out.println("筆試分數：" + pen_score);
37          System.out.println("口試分數：" + oral_score);
38          System.out.println("================");
39          tot_score = pen_score + oral_score;
40          System.out.println("總分：" + tot_score);
41      }
42  }
43
44  public class CH07_01
45  {
46      public static void main(String[] args)
47      {
48          CPenScore p1 = new CPenScore();
49          p1.SetPenData("張三", 50);
50          p1.ShowPenScore();
51
52          CTotalScore p2 = new CTotalScore();
```

# 繼承

```
53          p2.SetPenData("李四", 60);
54          p2.SetOralData(20);
55          p2.ShowTotScore();
56      }
57  }
```

- 執行結果：

```
筆試成績：
姓名：張三
筆試分數：50

總成績：
姓名：李四
筆試分數：60
口試分數：20
================
總分：80
```

- 說明：
  - 行01：
    - 定義「套件（package）」。

# 繼承

- 行03~行20：
  - 建立「CPenScore」類別。
  - 行05~行06：
    - 宣告類別的「資料成員」。
  - 行08~行12：
    - 建立「公開」的類別「方法成員」「SetPenData(String name, int score)」。
  - 行14~行19：
    - 建立「公開」的類別「方法成員」「ShowPenScore()」。
- 行22~行42：
  - 建立繼承自「CPenScore」父類別的「CTotalScore」子類別。
  - 行24~行25：
    - 宣告子類別的「資料成員」。

# 繼承

- 行27~行30：
  - 建立「公開」的子類別「方法成員」「SetOralData(int score)」。
- 行32~行41：
  - 建立「公開」的子類別「方法成員」「ShowTotScore()」。

- 行48：
  - 建立「CPenScore」類別的p1物件。
- 行49：
  - 執行p1物件的「方法成員」「SetPenData(String name, int score)」。
- 行50：
  - 執行p1物件的「方法成員」「ShowPenScore()」。

# 繼承

- 行52：
  - 建立「CTotalScore」類別的p2物件。
- 行53：
  - 執行p2物件的「方法成員」「SetPenData(String name, int score)」。

  注意：
  - 雖然在「CTotalScore」類別中，沒有「SetPenData(String name, int score)」的「方法成員」，但因為「CTotalScore」類別是繼承自「CPenScore」類別，所以也可以使用父類別的「SetPenData(String name, int score)」「方法成員」。
- 行54：
  - 執行p2物件的「方法成員」「SetOralData(int score)」。

# 繼承

- 行55：
  - 執行p2物件的「方法成員」「ShowTotScore()」。

# 繼承

- 建構子的執行順序：
  - 每個類別都預設有一個「不帶引數」的「建構子」。
    - 若該建構子在程式碼中被省略而沒有定義出來，則這個建構子事實上還是隱藏存在著，此建構子只是空敘述罷了。
  - 「類別」「繼承」後，在「建立子類別物件」時，會先「自動」執行「父類別」的「預設建構子」或「沒有參數的建構子」，再執行「子類別」的「預設建構子」或「沒有參數的建構子」。
    - 若為多代繼承時，會先執行最上層類別的「預設建構子」或「沒有參數的建構子」，再依序向下執行各代的「預設建構子」或「沒有參數的建構子」。

# 繼承

- 程式：

```
package CH07_02;

class CLine
{
   protected int leng;

   CLine()
   {
      System.out.println("\n執行CLine類別不帶引數的建構子");
   }

   CLine(int x, int y)
   {
      System.out.println("執行CLine(int x,int y)類別帶兩個引數的建構子");
      System.out.println("傳入建構子的引數x= " + x + "，y= " + y);
   }

   public void SetLeng(int l)
```

# 繼承

```
    {
        leng = l;
        System.out.println("直線長度 = " + leng);
    }
}

class CRectangle extends CLine
{
    private int wide;

    CRectangle()
    {
        System.out.println("執行CRectangle類別不帶引數的建構子");
    }

    CRectangle(int x, int y)
    {
        System.out.println("執行CRectangle類別有兩個引數的建構子");
        System.out.println("傳入建構子的引數 x= " + x + "，y= " + y);
    }
```

# 繼承

```
    public void SetValue(int l, int w)
    {
        leng = l;
        wide = w;
        System.out.println("矩形長度 = " + leng);
        System.out.println("矩形寬度 = " + wide);
    }
}

public class CH07_02
{
    public static void main(String[] args)
    {
        CLine sha1 = new CLine();
        sha1.SetLeng(20);
        System.out.println("\n==============================");
        CRectangle sha2 = new CRectangle();
        sha2.SetValue(10, 5);
```

# 繼承

```
        System.out.println("\n==============================");
        CRectangle sha3 = new CRectangle(2, 3);
        sha2.SetValue(10, 5);
    }
}
```

# 繼承

```
1  package CH07_02;
2
3  class CLine
4  {
5     protected int leng;
6
7     CLine()
8     {
9        System.out.println("\n執行CLine類別不帶引數的建構子");
10    }
11
12    CLine(int x, int y)
13    {
14       System.out.println("執行CLine(int x,int y)類別帶兩個引數的建構子");
15       System.out.println("傳入建構子的引數x=" + x + "，y=" + y);
16    }
17
18    public void SetLeng(int l)
19    {
20       leng = l;
21       System.out.println("直線長度 = " + leng);
22    }
23 }
24
25 class CRectangle extends CLine
26 {
```

# 繼承

```
27      private int wide;
28
29      CRectangle()
30      {
31          System.out.println("執行CRectangle類別不帶引數的建構子");
32      }
33
34      CRectangle(int x, int y)
35      {
36          System.out.println("執行CRectangle類別有兩個引數的建構子");
37          System.out.println("傳入建構子的引數 x=" + x + "，y=" + y);
38      }
39
40      public void SetValue(int l, int w)
41      {
42          leng = l;
43          wide = w;
44          System.out.println("矩形長度 = " + leng);
45          System.out.println("矩形寬度 = " + wide);
46      }
47  }
48
49  public class CH07_02
50  {
51      public static void main(String[] args)
52      {
```

# 繼承

```
53          CLine sha1 = new CLine();
54          sha1.SetLeng(20);
55          System.out.println("\n================================");
56          CRectangle sha2 = new CRectangle();
57          sha2.SetValue(10, 5);
58          System.out.println("\n================================");
59          CRectangle sha3 = new CRectangle(2, 3);
60          sha2.SetValue(10, 5);
61      }
62  }
```

# 繼承

- 執行結果：

```
執行CLine類別不帶引數的建構子
直線長度 = 20

================================

執行CLine類別不帶引數的建構子
執行CRectangle類別不帶引數的建構子
矩形長度 = 10
矩形寬度 = 5

================================

執行CLine類別不帶引數的建構子
執行CRectangle類別有兩個引數的建構子
傳入建構子的引數 x=2，y=3
矩形長度 = 10
矩形寬度 = 5
```

# 繼承

- 說明：
  - 行01：
    - 定義「套件（package）」。
  - 行03~行23：
    - 建立「CLine」類別。
    - 行05：
      - 宣告「保護」的類別「資料成員」。
    - 行07~行10：
      - 宣告類別「建構子」「CLine()」。
    - 行12~行16：
      - 宣告類別「建構子」「CLine(int x, int y)」。
    - 行18~行22：
      - 建立「公開」的類別「方法成員」「SetLeng(int l) 」。

# 繼承

- 行25~行47：
  - 建立繼承自「CLine」父類別的「CRectangle」子類別。
  - 行27：
    - 宣告「私有」的類別「資料成員」。
  - 行29~行32：
    - 宣告類別「建構子」「CRectangle()」。
  - 行34~行38：
    - 宣告類別「建構子」「CRectangle(int x, int y)」。
  - 行40~行46：
    - 建立「公開」的類別「方法成員」「SetValue(int l, int w)」。
- 行53：
  - 利用「CLine()」的「建構子」，建立sha1物件。
- 行54：
  - 執行sha1物件的「方法成員」「SetLeng(**int l**)」。

# 繼承

- 行56：
  - 利用「CRectangle()」的「建構子」，建立sha2物件。
  - 因為「Crectangle」類別是繼承「Cline」類別，故在建立「Crectangle」的sha2物件時，會先執行父類別沒有參數的建構子（即「CLine()」），再執行自己沒有參數的建構子（即「CRectangle()」）。
- 行57：
  - 執行sha2物件的「方法成員」「SetValue(int l, int w)」。
- 行59：
  - 利用「CRectangle(int x, int y)」的「建構子」，建立sha3物件。
  - 因為「Crectangle」類別是繼承「Cline」類別，故在建立「Crectangle」的sha3物件時，會先執行父類別沒有參數的建構子（即「CLine()」），再執行自己沒有參數的建構子（即「CRectangle(int x, int y)」）。

# 繼承

- 行60：
  - 執行sha3物件的「方法成員」「SetValue(int l, int w)」。

# 繼承

- super：
  - 在「類別」「繼承」的機制中，子類別用new建立物件時，會自動呼叫父類別的「預設建構子」或「不帶引數的建構子」。
  - 如果父類別有許多「帶有引數」的多載「建構子」，子類別要呼叫時，就要用「super」來呼叫。
    - 語法：

      **super(引數串列);**

# 繼承

- 「super」也可以用來呼叫父類別的「資料成員」或「方法成員」。
  - 語法：

    **super.資料成員;**

    或

    **super.方法成員;**

# 繼承

- 程式：

```
package CH07_03;

class CLine
{
    protected int leng;

    CLine(int l)
    {
        this.leng = l;
    }

    public void ShowLine()
    {
        System.out.println("\n**** 直線 ****");
        System.out.println("直線長度 = " + leng);
    }
}
```

# 繼承

```
class CRectangle extends CLine
{
    private int wide;

    CRectangle(int l, int w)
    {
        super(l);
        this.wide = w;
    }

    public void ShowRec()
    {
        System.out.println("\n**** 矩形 ****");
        System.out.println("矩形長度 = " + leng);
        System.out.println("矩形寬度 = " + wide);
        int peri = (leng + wide) * 2;
        int area = leng * wide;
        System.out.println("矩形周長 = " + peri);
```

# 繼承

```
        System.out.println("矩形面積 = " + area);

        System.out.print("\n呼叫父類別的ShowLine()方法 ： ");
        super.ShowLine();
        System.out.println("\n**** 顯示父類別leng的值 ****");
        System.out.println(super.leng);
    }
}

public class CH07_03
{
    public static void main(String[] args)
    {
        CLine sha1 = new CLine(20);
        sha1.ShowLine();

        CRectangle sha2 = new CRectangle(10, 5);
        sha2.ShowRec();
    }
}
```

# 繼承

```
1  package CH07_03;
2
3  class CLine
4  {
5     protected int leng;
6
7     CLine(int l)
8     {
9        this.leng = l;
10    }
11
12    public void ShowLine()
13    {
14       System.out.println("\n**** 直線 ****");
15       System.out.println("直線長度 = " + leng);
16    }
17 }
18
19 class CRectangle extends CLine
20 {
21    private int wide;
22
23    CRectangle(int l, int w)
24    {
25       super(l);
26       this.wide = w;
27    }
```

# 繼承

```
28
29      public void ShowRec()
30      {
31          System.out.println("\n**** 矩形 ****");
32          System.out.println("矩形長度 = " + leng);
33          System.out.println("矩形寬度 = " + wide);
34          int peri = (leng + wide) * 2;
35          int area = leng * wide;
36          System.out.println("矩形周長 = " + peri);
37          System.out.println("矩形面積 = " + area);
38
39          System.out.print("\n呼叫父類別的ShowLine()方法 ： ");
40          super.ShowLine();
41          System.out.println("\n**** 顯示父類別leng的值 ****");
42          System.out.println(super.leng);
43      }
44  }
45
46  public class CH07_03
47  {
48      public static void main(String[] args)
49      {
50          CLine sha1 = new CLine(20);
51          sha1.ShowLine();
52
53          CRectangle sha2 = new CRectangle(10, 5);
```

# 繼承

```
54        sha2.ShowRec();
55    }
56 }
```

- 執行結果：

```
**** 直線 ****
直線長度 = 20

**** 矩形 ****
矩形長度 = 10
矩形寬度 = 5
矩形周長 = 30
矩形面積 = 50

呼叫父類別的ShowLine()方法 :
**** 直線 ****
直線長度 = 10

**** 顯示父類別eng的值 ****
10
```

# 繼承

- 說明：
  - 行01：
    - 定義「套件（package）」。
  - 行03~行17：
    - 建立「CLine」類別。
    - 行05：
      - 宣告「保護」的類別「資料成員」。
    - 行07~行10：
      - 宣告類別「建構子」「CLine(int l)」。
    - 行12~行16：
      - 建立「公開」的類別方法「ShowLine()」。
  - 行19~行44：
    - 建立繼承自「CLine」父類別的「CRectangle」子類別。

# 繼承

- 行21：
  - 宣告「私有」的類別「資料成員」。
- 行23~行27：
  - 宣告類別「建構子」「CRectangle(int l, int w)」。
  - 行25：

    利用「super(l)」，呼叫「父類別」中，「有一個引數」的「建構子」「CLine(int l)」。
- 行29~行43：
  - 建立「公開」的類別方法「ShowRec()」。
  - 行40：

    執行父類別的「方法成員」「ShowLine()」。
  - 行42：

    顯示父類別「資料成員」「leng」的內容。

- 行50：
  - 利用「CLine(int l)」的「建構子」，建立sha1物件。

# 繼承

- 行51：
  - 執行sha1物件的「方法成員」「ShowLine()」。
- 行53：
  - 利用「CRectangle(int l, int w)」的「建構子」，建立sha2物件。
- 行54：
  - 執行sha2物件的「方法成員」「ShowRec()」。

# 繼承

- final：
  - 分為「成員變數」、「成員方法」及類別的「存取修飾子」來討論：
    - 成員變數：
      - 以final定義的類別成員變數，一旦經過初始化後，就只能供物件來讀取它的屬性值，不能再變更它的值，所以final成員變數通常在定義時就設定初始值。
    - 成員方法：
      - 以final定義的成員方法，是表示其衍生類別，不能「覆蓋（Override）」這個成員方法。
    - 類別的「存取修飾子」：
      - 表示該類別無法作為其餘類別的繼承原型。

# 繼承

- 程式：

```
package CH07_04;

final class CLine
{
    protected final int leng=5;

    CLine(int l)
    {
        this.leng = l;
    }

    public final void ShowLine()
    {
        System.out.println("\n**** 直線 ****");
        System.out.println("直線長度 = " + leng);
    }
}
```

# 繼承

```
class CRectangle extends CLine
{
    private int wide;

    CRectangle(int l, int w)
    {
        super(l);
        this.wide = w;
    }

    public void ShowLine()
    {
        System.out.println("\n**** 矩形 ****");
        System.out.println("矩形長度 = " + leng);
        System.out.println("矩形寬度 = " + wide);
    }

    public void ShowRec()
```

# 繼承

```
    {
        int peri = (leng + wide) * 2;
        int area = leng * wide;
        System.out.println("矩形周長 = " + peri);
        System.out.println("矩形面積 = " + area);
    }
}

public class CH07_04
{
    public static void main(String[] args)
    {
        CLine sha1 = new CLine(20);
        sha1.ShowLine();

        CRectangle sha2 = new CRectangle(10, 5);
        sha2.ShowLine();
        sha2.ShowRec();
    }
}
```

# 繼承

```
1  package CH07_04;
2
3  final class CLine
4  {
5     protected final int leng=5;
6
7     CLine(int l)
8     {
9        this.leng = l;
10    }
11
12    public final void ShowLine()
13    {
14       System.out.println("\n**** 直線 ****");
15       System.out.println("直線長度 = " + leng);
16    }
17 }
18
19 class CRectangle extends CLine
20 {
21    private int wide;
22
23    CRectangle(int l, int w)
24    {
25       super(l);
26       this.wide = w;
```

# 繼承

```
27      }
28
29      public void ShowLine()
30      {
31          System.out.println("\n**** 矩形 ****");
32          System.out.println("矩形長度 = " + leng);
33          System.out.println("矩形寬度 = " + wide);
34      }
35
36      public void ShowRec()
37      {
38          int peri = (leng + wide) * 2;
39          int area = leng * wide;
40          System.out.println("矩形周長 = " + peri);
41          System.out.println("矩形面積 = " + area);
42      }
43  }
44
45  public class CH07_04
46  {
47      public static void main(String[] args)
48      {
49          CLine sha1 = new CLine(20);
50          sha1.ShowLine();
51
52          CRectangle sha2 = new CRectangle(10, 5);
```

# 繼承

```
53          sha2.ShowLine();
54          sha2.ShowRec();
55      }
56  }
```

- 執行結果：

  此範例無法執行

- 說明：
  - 行01：
    - 定義「套件（package）」。
  - 行03~行17：
    - 建立「最終」的「CLine」類別。
    - 行05：
      - 宣告「保護」且「最終」的類別「資料成員」，並指定初值。
    - 行07~行10：
      - 宣告類別「建構子」「CLine(int l)」。

# 繼承

- 行09：
  因為「leng」被宣告為「final」，所以不能被修改內容。
- 行12~行16：
  - 建立「公開」且「最終」的類別方法「ShowLine()」。

- 行19~行43：
  - 建立繼承自「CLine」父類別的「CRectangle」子類別。
  - 因為「CLine」類別，被宣告為「final」，所以不能被「CRectangle」繼承。
  - 行21：
    - 宣告「私有」的類別「資料成員」。
  - 行23~行27：
    - 宣告類別「建構子」「CRectangle(int l, int w)」。

# 繼承

- 行25：

  利用「super(l)」，呼叫「父類別」中，「有一個引數」的「建構子」「CLine(int l)」。

  因為「leng」被宣告為「final」，所以不能被修改內容。

- 行29~行34：
  - 建立「公開」的類別方法「ShowLine()」。
- 行36~行42
  - 建立「公開」的類別方法「ShowRec()」。

- 行49：
  - 利用「CLine(int l)」的「建構子」，建立sha1物件。
- 行50：
  - 執行sha1物件的「方法成員」「ShowLine()」。

# 繼承

- 行52：
  - 利用「CRectangle(int l, int w)」的「建構子」，建立sha2物件。
- 行53：
  - 執行sha2物件的「方法成員」「ShowLine()」。
- 行54：
  - 執行sha2物件的「方法成員」「ShowRec()」。

# 多型

- 「多型（Polymorphism）」又稱為「同名異式」。
  - 可以使用相同名稱的方法，利用傳遞的引數數目或型別不一致，去執行不同敘述，而達到相同目的。

    例如：
    - 從甲地到乙地，坐公車、自行開車、搭火車、用跑的...等，都可以到達。
    - 即搭不同交通工具類別的物件，如果它們若皆有go()方法，皆可以使用來到達目的地。
- 「多型」的使用，可能會在父、子類別中定義名稱相同的方法（即「同名異式」）。
- 在這樣的機制下方法的定義與使用有四個主要概念：多載、覆蓋、物件參照、動態連結。

# 多型

- 多載：
  - 「多載（Overload）」在「類別」的繼承機制中是「多重定義方法」。
    - 即子類別繼承父類別的方法成員以後，在自身類別又定義了和父類別「名稱相同」的方法成員，但「傳入引數的型別或數目不同」，其呼叫後的結果也可能不同。
  - 前面曾提過同一「類別」的「方法多載」，也可算是多型。

# 多型

- 程式：

```
package CH07_05;

class CDogKind
{
    public void Show(String kind)
    {
        System.out.println("寵物種類：" + kind);
    }
}

class CDog extends CDogKind
{
    public void Show(String name, int age)
    {
        System.out.println("寵物名字：" + name);
        System.out.println("寵物年齡：" + age);
    }
}
```

# 多型

```
public class CH07_05
{
    public static void main(String[] args)
    {
        String kind = "米格魯";
        String name = "布丁";
        int age = 6;

        CDog dog = new CDog();
        dog.Show(kind);
        dog.Show(name, age);
    }
}
```

# 多型

```
1  package CH07_05;
2
3  class CDogKind
4  {
5     public void Show(String kind)
6     {
7        System.out.println("寵物種類：" + kind);
8     }
9  }
10
11 class CDog extends CDogKind
12 {
13    public void Show(String name, int age)
14    {
15       System.out.println("寵物名字：" + name);
16       System.out.println("寵物年齡：" + age);
17    }
18 }
19
20 public class CH07_05
21 {
22    public static void main(String[] args)
23    {
24       String kind = "米格魯";
25       String name = "布丁";
26       int age = 6;
```

# 多型

```
27
28          CDog dog = new CDog();
29          dog.Show(kind);
30          dog.Show(name, age);
31      }
32  }
```

- 執行結果：

```
寵物種類：米格魯
寵物名字：布丁
寵物年齡：6
```

- 說明：
  - 行01：
    - 定義「套件（package）」。
  - 行03~行09：
    - 建立「CDogKind」類別。
    - 行05~行08：
      - 建立「公開」的類別方法「Show(String kind)」。

# 多型

- 行11~行18：
  - 建立繼承自「CDogKind」父類別的「CDog」子類別。
  - 行13~行17：
    - 建立「公開」的類別方法「Show(String name, int age)」。
- 行24~行25：
  - 宣告字串變數，並指定初值。
- 行26：
  - 宣告整數變數，並指定初值。
- 行28：
  - 利用「CDog()」的「建構子」，建立dog物件。
- 行29：
  - 執行dog物件的「方法成員」「Show(String kind)」。

# 多型

- 行30：
  - 執行dog物件的「方法成員」「Show(String name, int age)」。

注意：

- 「CDog」類別是繼承「CDogKind」類別，雖然兩個類別均有「Show()」的方法成員，但因為引數的「資料型態」與「個數」都不相同，所以兩個「Show()」的方法成員都會存在，此為「多載（Overload）」的形式。
- 執行時會依照引數的「資料型態」與「個數」，自動找尋對應的方法成員來執行。

# 多型

- 覆蓋：
  - 「覆蓋（Override，也有被翻譯成「覆寫」）」在類別的繼承機制中，子類別重新定義父類別的方法。
    - 子類別繼承父類別的「方法成員」後，在「自身類別」內又定義了和父類別「傳回值型別」相同、「名稱」相同、「傳入引數的數目與型別」也相同的「方法成員」，則在父類別「相同名稱」的「方法成員」會被「子類別」的「方法成員」所「覆蓋」掉。

# 多型

- 程式：

```
package CH07_06;

class CDogKind
{
    private String kind;

    public void SetKind(String k)
    {
        kind = k;
    }

    public void Show()
    {
        System.out.println("寵物種類：" + kind);
    }
}

class CDog extends CDogKind
```

# 多型

```
{
    private String name;
    private int age;

    public void SetName(String n)
    {
        name = n;
    }

    public void SetAge(int a)
    {
        age = a;
    }

    public void Show()
    {
        System.out.println("寵物名字：" + name);
        System.out.println("寵物年齡：" + age);
    }
```

# 多型

```
}

public class CH07_06
{
    public static void main(String[] args)
    {
        System.out.println("*** dog1 ***");
        CDogKind dog1=new CDogKind();
        dog1.SetKind("米格魯");
        dog1.Show();

        System.out.println("\n*** dog2 ***");
        CDog dog2 = new CDog();
        dog2.SetKind("約克夏");
        dog2.SetName("布丁");
        dog2.SetAge(6);
        dog2.Show();
    }
}
```

# 多型

```
1  package CH07_06;
2
3  class CDogKind
4  {
5     private String kind;
6
7     public void SetKind(String k)
8     {
9        kind = k;
10    }
11
12    public void Show()
13    {
14       System.out.println("寵物種類：" + kind);
15    }
16 }
17
18 class CDog extends CDogKind
19 {
20    private String name;
21    private int age;
22
23    public void SetName(String n)
24    {
25       name = n;
26    }
```

# 多型

```
27
28      public void SetAge(int a)
29      {
30          age = a;
31      }
32
33      public void Show()
34      {
35          System.out.println("寵物名字：" + name);
36          System.out.println("寵物年齡：" + age);
37      }
38  }
39
40  public class CH07_06
41  {
42      public static void main(String[] args)
43      {
44          System.out.println("*** dog1 ***");
45          CDogKind dog1=new CDogKind();
46          dog1.SetKind("米格魯");
47          dog1.Show();
48
49          System.out.println("\n*** dog2 ***");
50          CDog dog2 = new CDog();
51          dog2.SetKind("約克夏");
52          dog2.SetName("布丁");
```

# 多型

```
53          dog2.SetAge(6);
54          dog2.Show();
55      }
56  }
```

- 執行結果：

```
*** dog1 ***
寵物種類：米格魯

*** dog2 ***
寵物名字：布丁
寵物年齡：6
```

- 說明：
  - 行01：
    - 定義「套件（package）」。
  - 行03~行16：
    - 建立「CDogKind」類別。

# 多型

- 行05：
  - 宣告「私有」的類別「資料成員」。
- 行07~行10：
  - 建立「公開」的類別方法「SetKind(String k)」。
- 行12~行15：
  - 建立「公開」的類別方法「Show()」。

- 行18~行38：
  - 建立繼承自「CDogKind」父類別的「CDog」子類別。
  - 行20~行21：
    - 宣告「私有」的類別「資料成員」。
  - 行23~行26：
    - 建立「公開」的類別方法「SetName(String n)」。
  - 行28~行31：
    - 建立「公開」的類別方法「SetAge(int a)」。

# 多型

- 行33~行37：
  - 建立「公開」的類別方法「Show()」。
- 行45：
  - 利用「CDogKind」的「建構子」，建立dog1物件。
- 行46：
  - 執行dog1物件的「方法成員」「SetKind(String k)」。
- 行47：
  - 執行dog1物件的「方法成員」「Show()」。
  - dog1物件是依據「CDogKind」類別所建立的物件，故執行dog1物件時，會執行「CDogKind」類別中的方法成員「Show()」。
- 行50：
  - 利用「CDog」的「建構子」，建立dog2物件。

# 多型

- 行51：
  - 執行dog2物件的「方法成員」「SetKind(String k)」。
- 行52：
  - 執行dog2物件的「方法成員」「SetName(String n)」。
- 行53：
  - 執行dog2物件的「方法成員」「SetAge(int a)」。
- 行54：
  - 執行dog2物件的「方法成員」「Show()」。
  - dog2物件是依據「CDog」類別所建立的物件，執行dog2物件時，「CDog」類別的方法成員「Show()」，會「覆蓋（Override）」「CDogKind」的方法成員，故會執行「CDog」類別中的方法成員「Show()」。

# 多型

- 物件參照：
  - 子類別繼承了父類別的成員後，有些父類別的方法成員會被子類別覆蓋。
  - 這時父類別物件可以使用物件參照指向子類別的物件，呼叫子類別的方法成員覆蓋父類別的方法成員。

    例如：
    - 「電視機」與「冷氣機」相信這是各位日常生活中不可或缺的生活家電。
    - 這些家電都有一個共通特性：
      可以利用「遙控器」進行控制動作。
  - 作法分為下列三部分：

# 多型

## 1、基礎類別宣告：

- 通常於實作物件多形時，並不會利用基礎類別物件，來呼叫基礎類別內的任何成員方法。
- 因此對於成員方法僅須進行象徵性的宣告動作，而實作的部份則交由衍生類別來重新定義即可。
- 如下類別程式碼片段所示：

```
class RemoteControl
{
   public void powerOn()
   {

   }
          ⋮
      其它程式敘述
          ⋮
}
```

# 多型

## 2、衍生類別宣告：

- 這些繼承自基礎類別的成員方法，主要是針對所有衍生類別共有的運算功能。
- 使用者可依照衍生類別特性的不同，分別進行重新定義動作。
- 如下列程式碼片段所示：

```
class MyTV extends RemoteControl
 {
   public void powerOn()
   {
      依照類別特性實作執行敘述
   }
                  ⋮
            其它程式敘述
                  ⋮
 }
```

# 多型

## 3、主程式區塊：

- 使用者必須先建立一個基礎類別物件，再透過「new」關鍵字所組成物件建構語法，將基礎類別物件轉型為衍生類別物件。
- 進而呼叫衍生類別所重載的成員方法，來執行相對應的運算工作。
- 如下列程式碼片段所示：

```
public static void main(String args[ ])
{
   RemoteControl myControl = new RemoteControl();
   myControl = new MyTV();
   myControl.powerOn();
            ⋮
      其它程式敘述
            ⋮
}
```

# 多型

- 程式：

```
package CH07_07;

import java.io.*;

class CPlayer
{
    public void Show()
    {

    }
}

class CCD extends CPlayer
{
    public void Show()
    {
        System.out.println("現在正在撥放 [音樂CD]");
    }
```

# 多型

```
}

class CDVD extends CPlayer
{
    public void Show()
    {
        System.out.println("現在正在撥放 [影片DVD]");
    }
}

public class CH07_07
{
    public static void main(String[] args) throws IOException
    {
        CPlayer play = new CPlayer();
        CCD cd = new CCD();
        CDVD dvd = new CDVD();

        BufferedReader keyin = new BufferedReader
                               (new InputStreamReader(System.in));
```

# 多型

```
        while (true)
        {
            System.out.println("請輸入 CD 或 DVD ？");
            String item = keyin.readLine();
            if (item.equals("CD") || item.equals("cd"))
                play = cd;
            else if (item.equals("DVD") || item.equals("dvd"))
                play = dvd;
            else
                continue;
            play.Show();
            break;
        }
    }
}
```

# 多型

```
1  package CH07_07;
2
3  import java.io.*;
4
5  class CPlayer
6  {
7     public void Show()
8     {
9
10    }
11 }
12
13 class CCD extends CPlayer
14 {
15    public void Show()
16    {
17       System.out.println("現在正在撥放[音樂CD]");
18    }
19 }
20
21 class CDVD extends CPlayer
22 {
23    public void Show()
24    {
25       System.out.println("現在正在撥放[影片DVD]");
26    }
```

# 多型

```
27  }
28
29  public class CH07_07
30  {
31     public static void main(String[] args) throws IOException
32     {
33        CPlayer play = new CPlayer();
34        CCD cd = new CCD();
35        CDVD dvd = new CDVD();
36
37        BufferedReader keyin = new BufferedReader(new InputStreamReader(System.in));
38
39        while (true)
40        {
41           System.out.println("請輸入 CD 或 DVD ？");
42           String item = keyin.readLine();
43           if (item.equals("CD") || item.equals("cd"))
44              play = cd;
45           else if (item.equals("DVD") || item.equals("dvd"))
46              play = dvd;
47           else
48              continue;
49           play.Show();
50           break;
51        }
52     }
53  }
```

# 多型

- 執行結果：

```
請輸入 CD 或 DVD ？
cd
現在正在撥放 [音樂CD]
```

```
請輸入 CD 或 DVD ？
DVD
現在正在撥放 [影片DVD]
```

- 說明：
  - 行01：
    - 定義「套件（package）」。
  - 行03：
    - 載入「java.io.*」套件。
  - 行05~行11：
    - 建立「CPlayer」類別。

# 多型

- 行07~行10：
  - 建立「公開」的類別方法「Show()」。
  - 方法成員「Show()」，是準備用來被實作多型的方法。
- 行13~行19：
  - 建立繼承自「CPlayer」父類別的「CCD」子類別。
  - 行15~行18：
    - 建立「公開」的類別方法「Show()」
    - 用來實作父類別多型的方法成員「Show()」。
- 行21~行27：
  - 建立繼承自「CPlayer」父類別的「CDVD」子類別。
  - 行23~行26：
    - 建立「公開」的類別方法「Show()」。
    - 用來實作父類別多型的方法成員「Show()」。

# 多型

- 行31：
  - main()方法後面，加上 throws IOException。
- 行33：
  - 利用「CPlayer」的「建構子」，建立play物件。
- 行34：
  - 利用「CCD」的「建構子」，建立cd物件。
- 行35：
  - 利用「CDVD」的「建構子」，建立dvd物件。
- 行37：
  - 利用「BufferedReader」的類別，建立keyin物件。
- 行39~行51：
  - 利用while迴圈，輸入資料，並依判斷式執行相關的程式敘述。

# 多型

- 行42：
  - 利用keyin物件的readLine()方法，讀取由鍵盤輸入的字串，並指派給字串變數item。
- 行43~行48：
  - 利用判斷式將「父類別的物件」「轉型」為「子類別物件」。
- 行49：
  - 「父類別的物件」「轉型」後，執行「子類別物件」的方法成員「Show()」。
- 行50：
  - 中斷此while迴圈的執行，並跳離此迴圈（行39~行51）。

# 多型

- 動態連結：
  - 若程式中所使用到的名稱（包括物件名稱及變數名稱），在編譯過程中即完成連結，稱為「靜態連結（Static Binding）」，又稱為「早期連結」。
  - 「動態連結（dynamic binding）」又稱為「晚期連結」。
  - 即物件的連結不是在編譯時期就決定了，而是在程式執行時期依物件當時的狀態決定。
  - 如何動態決定，是根據外邊送來的訊息做出適當的反應。
  - 透過這種動態連結的技巧，才是精典的多型，而多型若要應付外邊送來的訊息，須使用引數的傳遞。

# 多型

- 「動態連結」的觀念及說明如下：
  - 觀念：
    - 決定要呼叫改寫的類別方法是在「執行期間」，不是在「編譯期間」。
  - 理論基礎：
    - 基礎類別（父類別）的參考變數可以參考衍生類別（子類別）物件。
  - 分配流程：
    - 父類別呼叫改寫方法時，Java會根據父類別的參考物件所指向參考的物件類型是父類別還是子類別而有所不同。
    - 因此決定呼叫那一個改寫方法的是「被參考的物件類型」。

# 多型

- 程式：

```
package CH07_08;

import java.io.*;

class CFruit
{
    protected int price;

    public int Spend()
    {
        return 0;
    }
}

class CApple extends CFruit
{
    private int number;
```

# 多型

```
    CApple(int number, int price)
    {
        this.number = number;
        this.price = price;
}

    public int Spend()
    {
        return number * price;
    }
}

class CTomato extends CFruit
{
    private float kg;

    CTomato(float kg, int price)
    {
```

# 多型

```
        this.kg = kg;
        this.price = price;
    }

    public int Spend()
    {
        return (int) (kg * price);
    }
}

class CSum
{
    public static int tot;

    public void Total(CFruit f)
    {
        tot += f.Spend();
        System.out.println("小計：" + f.Spend() + "元\t\t累計：" + tot);
    }
```

# 多型

```
}

public class CH07_08
{
    public static void main(String[] args) throws IOException
    {
        CSum sum = new CSum();
        CApple apple;
        CTomato tomato;

        BufferedReader keyin = new BufferedReader
                               (new InputStreamReader(System.in));

        while (true)
        {
            System.out.print("結算的總類？（1、蘋果\t2、番茄\t3、離開）：");
            String item = keyin.readLine();
            if (item.equals("1"))
            {
```

# 多型

```
        System.out.print("購買幾顆？");

        int number = Integer.parseInt(keyin.readLine());
        System.out.print("單價（元）？");
        int price = Integer.parseInt(keyin.readLine());
        apple = new CApple(number, price);
        sum.Total(apple);
    }
    else if (item.equals("2"))
    {
        System.out.print("購買幾公斤？");
        float kg = Float.parseFloat(keyin.readLine());
        System.out.print("單價（元）？");
        int price = Integer.parseInt(keyin.readLine());
        tomato = new CTomato(kg, price);
        sum.Total(tomato);
    }
    else
        break;
```

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

# 多型

```
1  package CH07_08;
2
3  import java.io.*;
4
5  class CFruit
6  {
7     protected int price;
8
9     public int Spend()
10    {
11       return 0;
12    }
13 }
14
15 class CApple extends CFruit
16 {
17    private int number;
18
19    CApple(int number, int price)
20    {
21       this.number = number;
22       this.price = price;
23    }
24
25    public int Spend()
26    {
```

# 多型

```
27          return number * price;
28      }
29  }
30
31  class CTomato extends CFruit
32  {
33      private float kg;
34
35      CTomato(float kg, int price)
36      {
37          this.kg = kg;
38          this.price = price;
39      }
40
41      public int Spend()
42      {
43          return (int) (kg * price);
44      }
45  }
46
47  class CSum
48  {
49      public static int tot;
50
51      public void Total(CFruit f)
52      {
```

# 多型

```
53          tot += f.Spend();
54          System.out.println("小計：" + f.Spend() + "元\t\t累計：" + tot);
55      }
56  }
57
58  public class CH07_08
59  {
60      public static void main(String[] args) throws IOException
61      {
62          CSum sum = new CSum();
63          CApple apple;
64          CTomato tomato;
65
66          BufferedReader keyin = new BufferedReader(new InputStreamReader(System.in));
67
68          while (true)
69          {
70              System.out.print("結算的總類？（1、蘋果\t2、番茄\t3、離開）：");
71              String item = keyin.readLine();
72              if (item.equals("1"))
73              {
74                  System.out.print("購買幾顆？");
75                  int number = Integer.parseInt(keyin.readLine());
76                  System.out.print("單價（元）？");
77                  int price = Integer.parseInt(keyin.readLine());
78                  apple = new CApple(number, price);
```

# 多型

```
79              sum.Total(apple);
80          }
81          else if (item.equals("2"))
82          {
83              System.out.print("購買幾公斤？");
84              float kg = Float.parseFloat(keyin.readLine());
85              System.out.print("單價（元）？");
86              int price = Integer.parseInt(keyin.readLine());
87              tomato = new CTomato(kg, price);
88              sum.Total(tomato);
89          }
90          else
91              break;
92          System.out.println();
93      }
94  }
95  }
```

# 多型

- 執行結果：

```
結算的總類？（1、蘋果　2、番茄 3、離開）：1
購買幾顆？2
單價（元）？50
小計：100元　　累計：100

結算的總類？（1、蘋果　2、番茄 3、離開）：2
購買幾公斤？2
單價（元）？25
小計：50元　　累計：150

結算的總類？（1、蘋果　2、番茄 3、離開）：3
```

- 說明：
  - 行01：
    - 定義「套件（package）」。
  - 行03：
    - 載入「java.io.*」套件。

# 多型

- 行05~行13：
  - 建立「CFruit」類別。
  - price欄位會被繼承；方法成員「Spend()」，是準備用來被實作多型的方法。
  - 行07：
    - 宣告「保護」的類別「資料成員」。
  - 行09~行12：
    - 建立「公開」的類別方法「Spend()」。
- 行15~行29：
  - 建立繼承自「CFruit」父類別的「CApple」子類別。
  - 行17：
    - 宣告「私有」的類別「資料成員」。
  - 行19~行23：
    - 宣告類別「建構子」「CApple(int number, int price)」。

# 多型

- 行25~行28：
  - 建立「公開」的類別方法「Spend()」。
  - 用來實作父類別多型的方法成員「Spend()」。
- 行31~行45：
  - 建立繼承自「CFruit」父類別的「CTomato」子類別。
  - 行33：
    - 宣告「私有」的類別「資料成員」。
  - 行35~行39：
    - 宣告類別「建構子」「CTomato(float kg, int price)」。
  - 行41~行44：
    - 建立「公開」的類別方法「Spend()」。
    - 用來實作父類
- 行47~行56：
  - 建立「CSum」類別。

# 多型

- 行49：
  - 宣告「公開」「靜態」的「類別變數」。
- 行51~行55：
  - 建立「公開」的類別方法「Total(CFruit f)」。
  - 引數CFruit f是父類別的物件參照。
  - 此方法在行79與行88就分別指定了引數CFruit f引數的物件參照是子類別CApple的物件apple，或是子類別CTomato的物件tomato。
  - 若傳入的是子類別CApple的物件apple，則f物件參照指向apple，其行53的f.Spend()就會實作apple.Spend()（行25~行28）；若傳入的是子類別CTomato的物件tomato，則f物件參照指tomato，其行53的f.Spend()就會實作tomato.Spend()（行41~行44）。
  - 行53：

    累加apple.Spend()或tomato.Spend()實作所傳回的金額，並將結果存入「類別變數」「tot」中。

# 多型

- 行54：

  顯示呼叫apple.Spend()或tomato.Spend()的傳回值，及tot累加的值。

- 行60：
  - main()方法後面，加上 throws IOException。
- 行62：
  - 利用「CSum」的「建構子」，建立sum物件。
- 行63：
  - 宣告「CApple」的apple物件。
  - 這物件尚未用new來建立（因要動態指派CApple類別建構子的引數值，故於行78才建立該物件）。
- 行64：
  - 宣告「CTomato」的tomato物件。
  - 這物件尚未用new來建立（因要動態指派CTomato類別建構子的引數值，故於行87才建立該物件）。

# 多型

- 行66：
  - 利用「BufferedReader」的類別，建立keyin物件。
- 行68~行93：
  - 利用while迴圈，輸入資料，並依判斷式執行相關的程式敘述。
  - 行75：
    - 利用keyin物件的readLine()方法，讀取由鍵盤輸入的數字，並指派給整數變數number。
  - 行77：
    - 利用keyin物件的readLine()方法，讀取由鍵盤輸入的數字，並指派給整數變數price。
  - 行78：
    - 利用「CApple(int number, int price)」的「建構子」，建立apple物件。

# 多型

- 行79：
  - 動態連結「sum.Total(apple)」來實作「apple.Spend()」方法。
- 行84：
  - 利用keyin物件的readLine()方法，讀取由鍵盤輸入的數字，並指派給浮點數變數kg。
- 行86：
  - 利用keyin物件的readLine()方法，讀取由鍵盤輸入的數字，並指派給整數變數price。
- 行87：
  - 利用「CTomato(float kg, int price)」的「建構子」，建立tomato物件。
- 行88：
  - 動態連結「sum.Total(tomato)」來實作「tomato.Spend()」方法。

# 多型

- 行91：
  - 中斷此while迴圈的執行，並跳離此迴圈（行68~行93）。

# 資料來源


- 蔡文龍、何嘉益、張志成、張力元，JAVA SE 10基礎必修課，台北市，碁峰資訊股份有限公司，2018年7月，出版。
- 吳燦銘、胡昭民，圖解資料結構-使用Java(第三版)，新北市，博碩文化股份有限公司，2018年5月，出版。
- Ivor Horton，Java 8 教學手冊，台北市，碁峰資訊股份有限公司，2016年9月，出版。
- 李春雄，程式邏輯訓練入門與運用---使用JAVA SE 8，台北市，上奇科技股份有限公司，2016年6月，初版。
- 位元文化，Java 8視窗程式設計，台北市，松崗資產管理股份有限公司，2015年12月，出版。
- Benjamin J Evans、David Flanagan，Java 技術手冊 第六版，台北市，碁峰資訊股份有限公司，2015年7月，出版。
- 蔡文龍、張志成，JAVA SE 8 基礎必修課，台北市，碁峰資訊股份有限公司，2014年11月，出版。
- 陳德來，Java SE 8程式設計實例，台北市，上奇科技股份有限公司，2014年11月，初版。
- 林信良，Java SE 8 技術手冊 ，台北市，碁峰資訊股份有限公司，2014年6月，出版。
- 何嘉益、黃世陽、李篤易、張世杰、黃鳳梅，徐政棠譯，JAVA2 程式設計從零開始--適用JDK7，台北市，上奇資訊股份有限公司，2012年5月，出版。