CHAPTER 5

設計大型程式(ch13)

例外處理與輸入輸出處理(ch14)

執行緒(ch15)

本節介紹

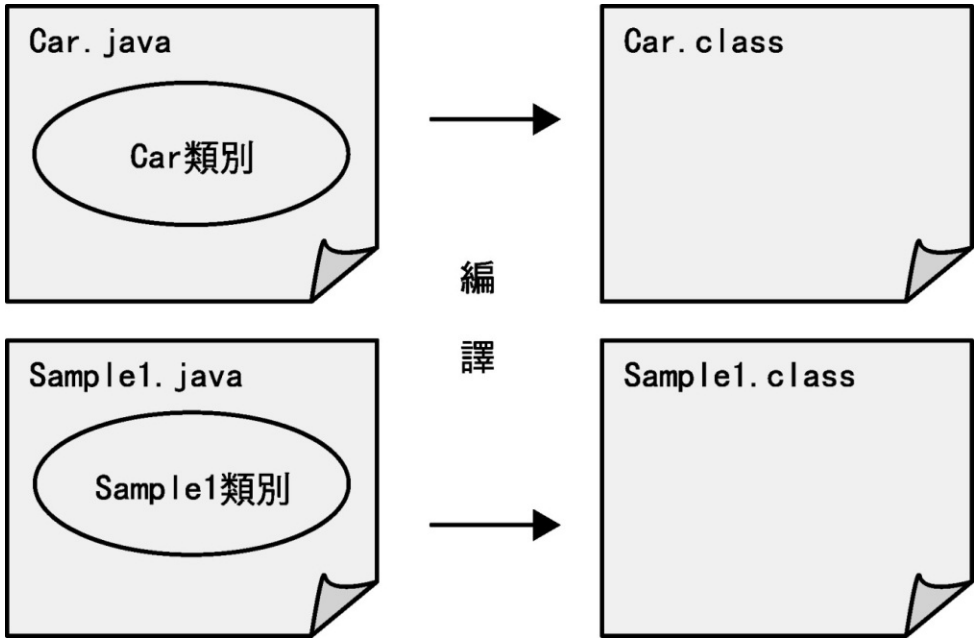
- 分割檔案
- 套件的使用方式
- 匯入

關鍵詞彙

- 分割檔案
- 套件
- 子套件
- 匯入

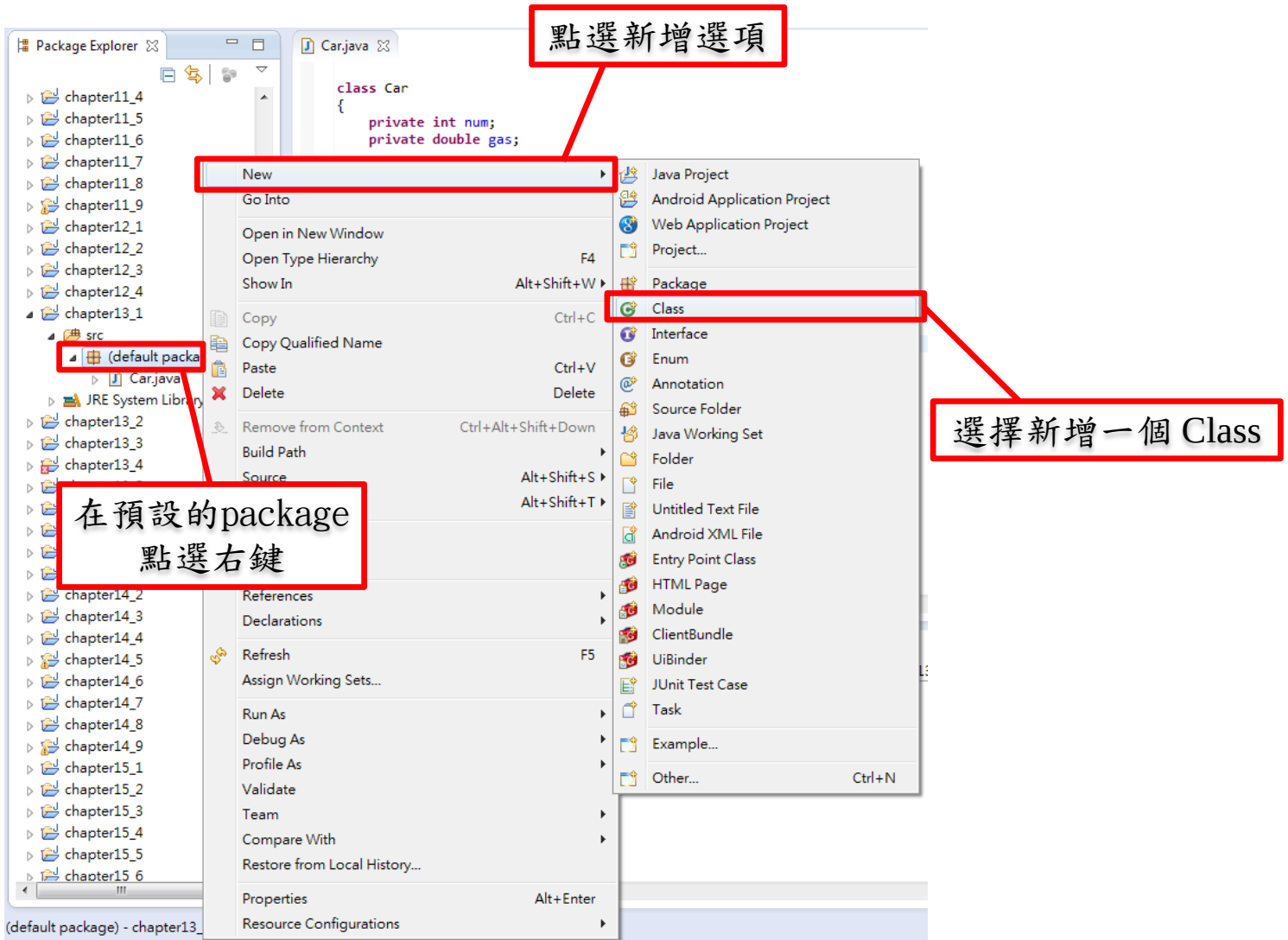
分割檔案(1/4)

由好幾位程式設計師共同完成一個大型程式是非常普遍的事情，有時候還會用寫好的類別來設計更大型的新程式。此時，像之前的範例那樣將好幾個類別寫在同一個檔案中是非常不方便的。因此，本章節會先介紹將各個類別分別寫在不同的檔案中的技巧。

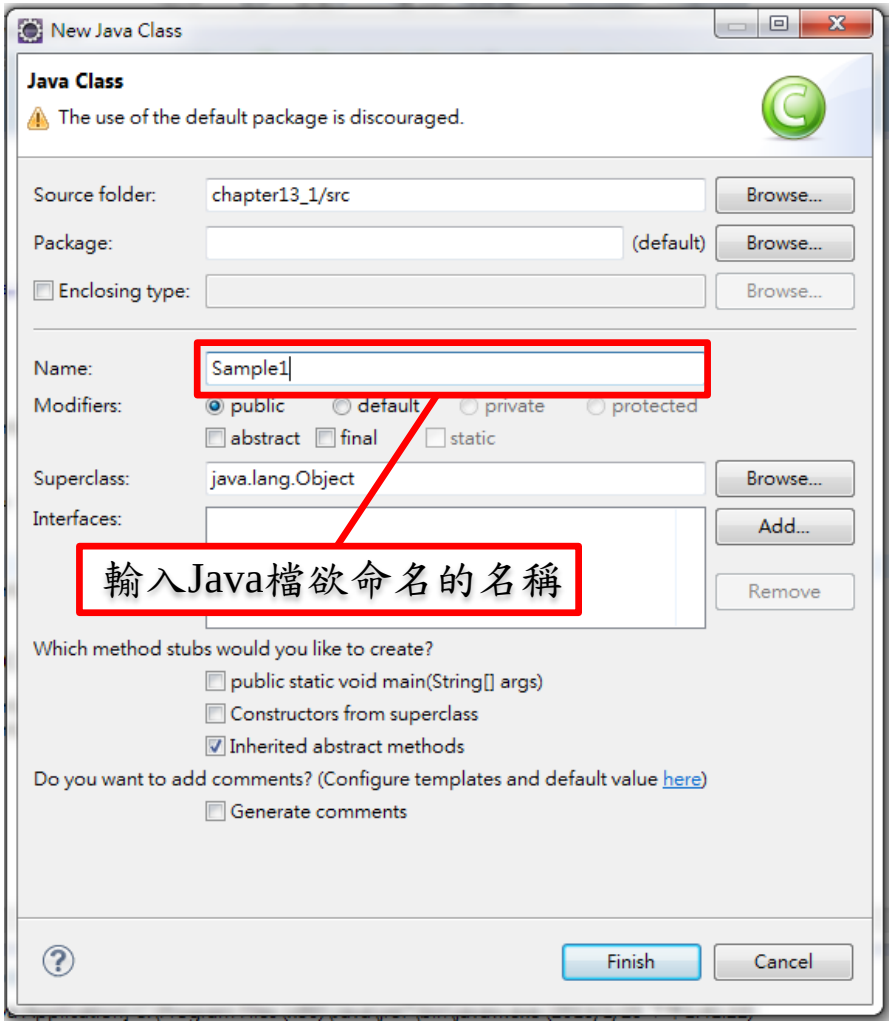


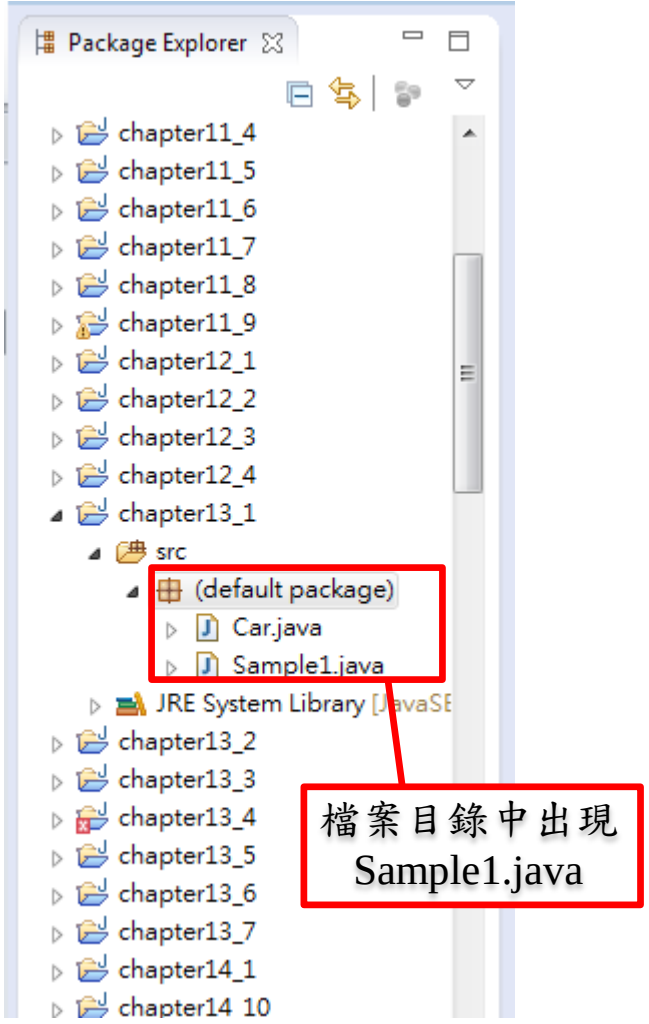
分割檔案(2/4)

如何新增一個 .Java 檔案?



分割檔案(3/4)


New Java Class
Java Class
The use of the default package is discouraged.
Source folder: chapter13_1/src
Browse...
Package:
(default)
Enclosing type:
Name: Sample1
Modifiers: public default private protected
abstract final static
Superclass: java.lang.Object
Interfaces:
Add...
Remove
輸入Java檔欲命名的名稱
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
Generate comments
Finish
Cancel


Package Explorer
chapter11_4
chapter11_5
chapter11_6
chapter11_7
chapter11_8
chapter11_9
chapter12_1
chapter12_2
chapter12_3
chapter12_4
chapter13_1
src
(default package)
Car.java
Sample1.java
JRE System Library
chapter13_2
chapter13_3
chapter13_4
chapter13_5
chapter13_6
chapter13_7
chapter14_1
chapter14_10
檔案目錄中出現
Sample1.java

分割檔案(4/4)

Sample5_1.java

```
1  class Sample5_1
2  {
3     public static void main (String[] args)
4     {
5        Car car1 ;
6        car1 = new Car();
7        car1.show();
8     }
9  }
```

將兩個class分別寫在兩個java檔

Car.java

```
1  class Car
2  {
3     private int num;
4     private double gas;
5
6     public Car()
7     {
8        num = 0;
9        gas = 0.0;
10       System.out.println("已生產了汽車");
11    }
12
13    public void setCar(int n, double g)
14    {
15       num = n;
16       gas = g;
17
18       System.out.println("將車號設為"+num+"汽油量設為"+gas);
19    }
20
21    public void show()
22    {
23       System.out.println("車號是"+num);
24       System.out.println("汽油量是"+num);
25    }
26 }
```

Console | Markers

<terminated> Sample5_1
已生產了汽車
車號是0
汽油量是0

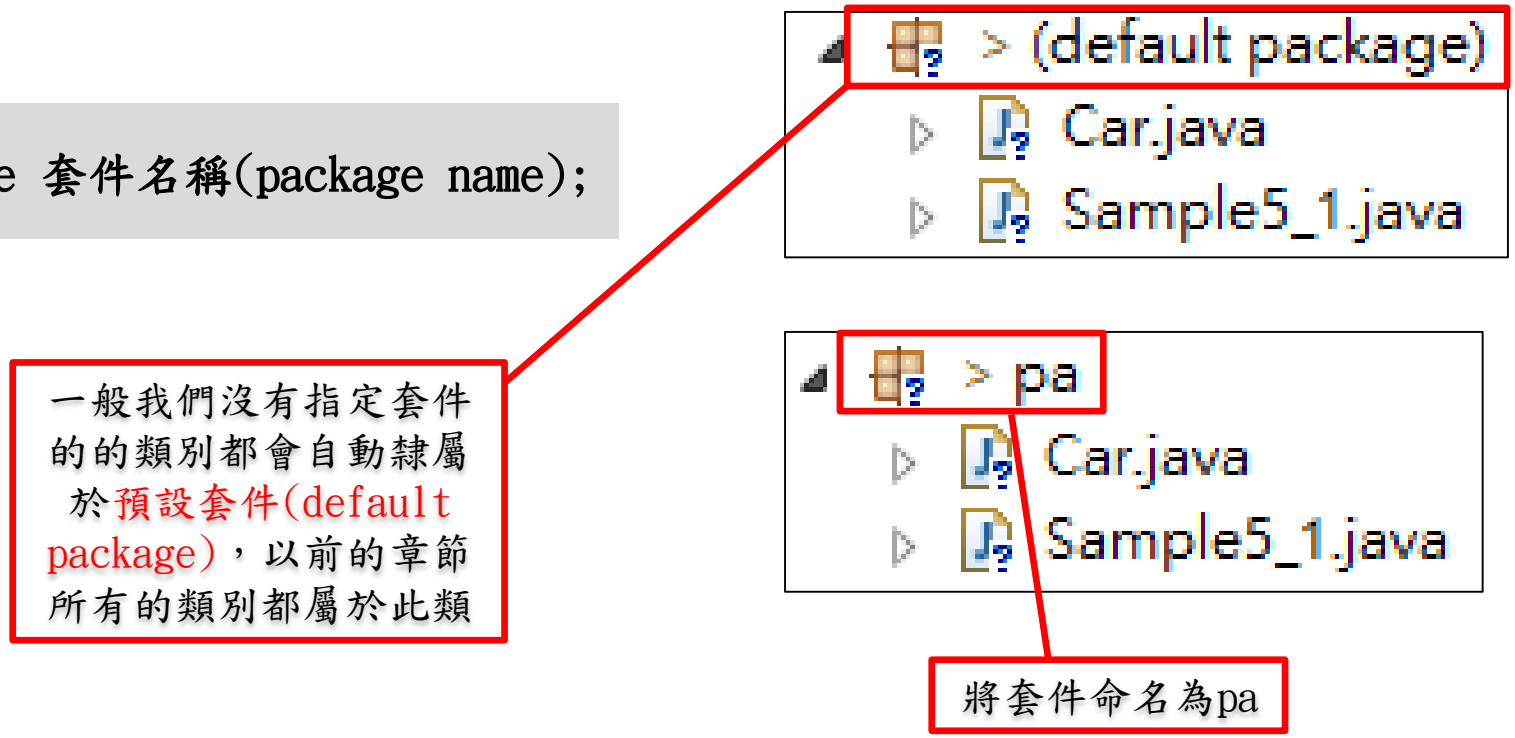
輸出結果

套件的使用方式(1/6)

大型程式中經常會用到別人設計的各種類別，因此有時候會需要同時使用由不同作者所寫的同名類別。此時，java 採用**套件(package)**這種機制來區分類別名稱。

語法

```
package 套件名稱(package name);
```



套件的使用方式(2/6)

Sample5_2.java

```
1  package pa;
2  class Sample5_2
3  {
4     public static void main (String[] args)
5     {
6        Car car1 ;
7        car1 = new Car();
8        car1.show();
9     }
10 }
```

可以看到現在兩個class都隸屬套件pa

Car.java

```
1  package pa;
2  class Car
3  {
4     private int num;
5     private double gas;
6
7     public Car()
8     {
9        num = 0;
10       gas = 0.0;
11       System.out.println("已生產了汽車");
12    }
13
14    public void setCar(int n, double g)
15    {
16       num = n;
17       gas = g;
18
19       System.out.println("將車號設為"+num+"汽油量設為"+gas);
20    }
21
22    public void show()
23    {
24       System.out.println("車號是"+num);
25       System.out.println("汽油量是"+num);
26    }
27 }
```

Console Markers

<terminated> Sample5_2

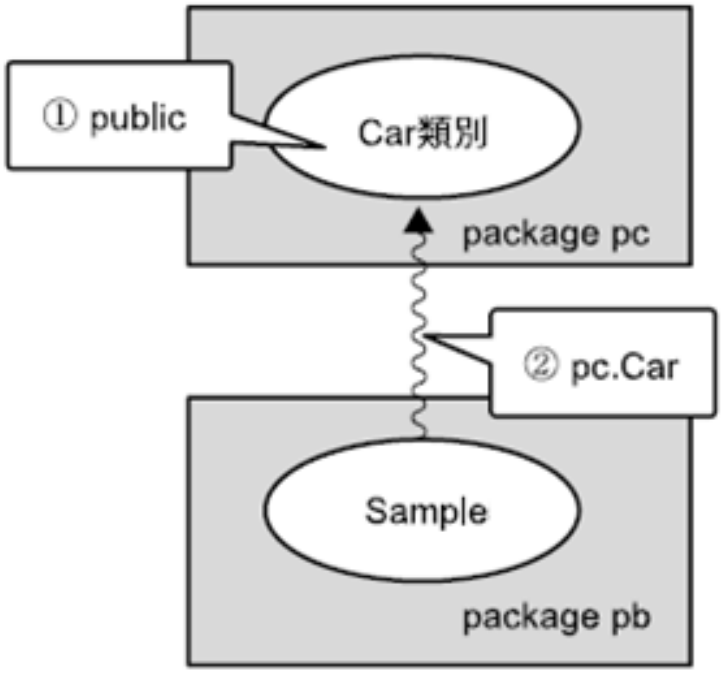
已生產了汽車
車號是0
汽油量是0

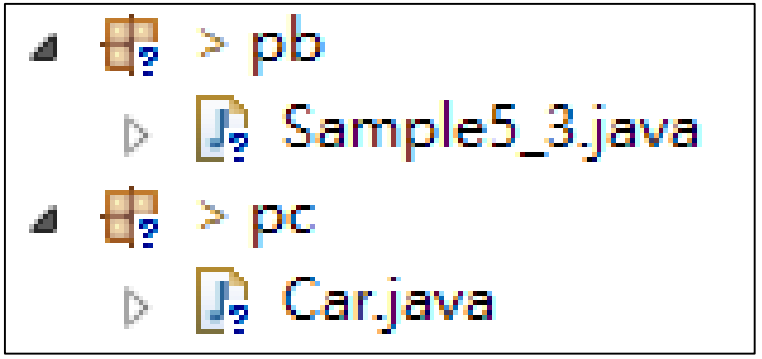
輸出結果

套件的使用方式(3/6)

再來我們要介紹如何使用不同套件(package)的類別。使用前必須要準備兩件事:

- ✓ 在被存取的類別加上public修飾子
- ✓ 在存取其他套件類別時，要加上套件名稱。





套件的使用方式(4/6)

Sample5_3.java

```
1  package pb;
2
3  class Sample5_3
4  {
5    public static void main (String[] args)
6    {
7      pc.Car car1 = new pc.Car();
8      car1.show();
9    }
10 }
```

類別Sample5_3隸屬套件pb

在類別名稱前要加套件名稱

Car.java

```
1  package pc;
2
3  public class Car
4  {
5    private int num;
6    private double gas;
7
8    public Car()
9    {
10     num = 0;
11     gas = 0.0;
12     System.out.println("已生產了汽車");
13   }
14
15   public void setCar(int n, double g)
16   {
17     num = n;
18     gas = g;
19
20     System.out.println("將車號設為"+num+"汽油量設為"+gas);
21   }
22
23   public void show()
24   {
25     System.out.println("車號是"+num);
26     System.out.println("汽油量是"+num);
27   }
28 }
```

類別Car隸屬套件pc

加上public才能被其他套件的類別使用

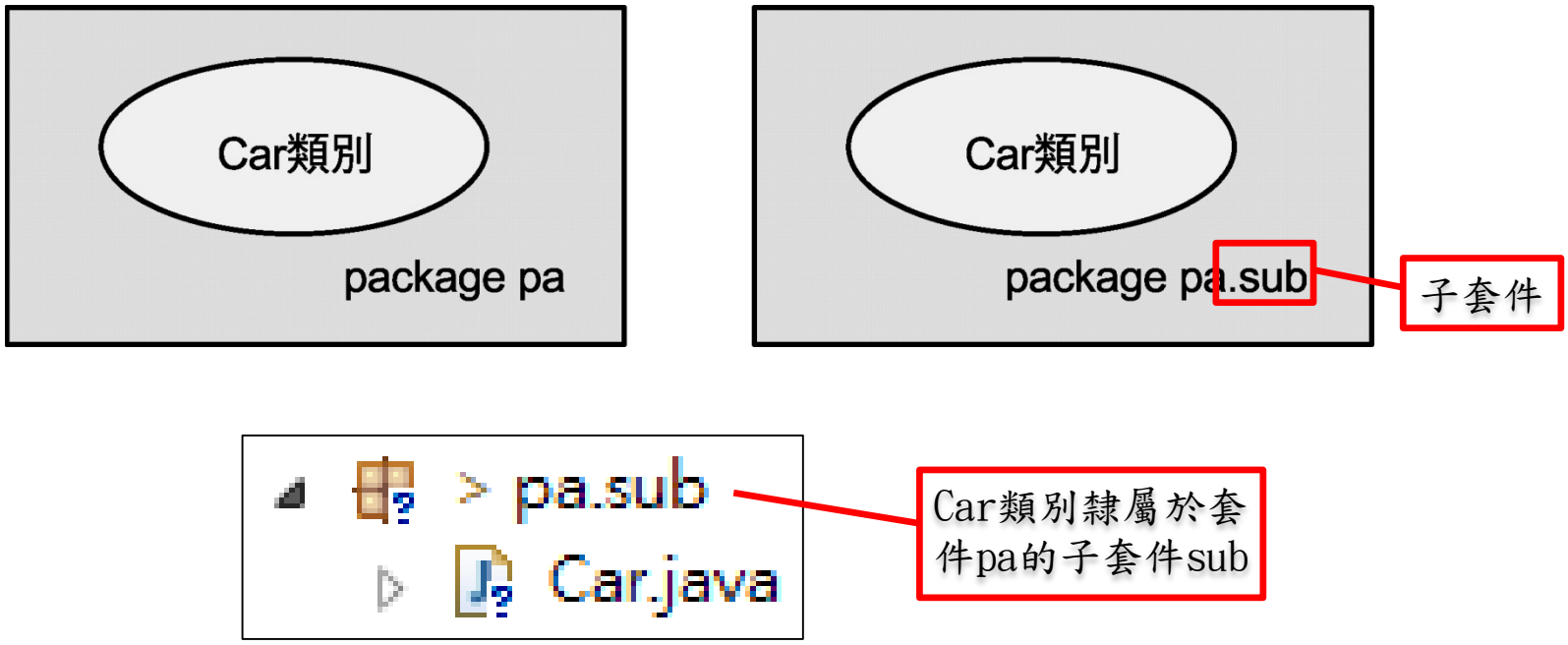
Console Markers

<terminated> Sample5_3
已生產了汽車
車號是0
汽油量是0

輸出結果

套件的使用方式(5/6)

套件還有更便利的作法，那就是可以建立套件的階層，只要將套件建立為階層，就能依照功能來分類各種類別，而套件的下一層套件就稱為**子套件(subpackage)**。



套件的使用方式(6/6)

類別、介面使用的修飾子	功能
沒有特別指定修飾子時	表示該類別或介面只能被同一套件的程式存取
public	類別或介面前面加上public表示其他套件下的程式也可以進來存取

成員、建構子使用的修飾子	功能
沒有特別指定修飾子時	表示該成員或建構子只能被同一個套件內的程式存取
public	前面加上public表示任何類別都可以對它存取。(但是如果成員或建構子所屬的類別本身不是宣告為public的話，即使將成員和建構子宣告為public，還是只有同一個套件內的程式能對其存取)
private	成員或建構子前面加上private表示只有位於同一類別的程式才能對其存取
protected	前面加上protected表示只有同一個套件下的類別、以及其他套件的子類別可以對它存取

匯入(1/4)

在上一節中曾經說過，為了要存取其它套件中的類別，就必須在類別名稱之前加上套件名稱。但是，經常會用到其他套件的類別時，在每個類別之前都加上套件名稱會相當麻煩，因此，可以在程式的開頭先進行**匯入(import)**的動作。

語法

```
import 套件名稱.類別名稱;
```

匯入(2/4)

Sample5_4.java

```
1  package pb;
2  import pc.Car;
3
4  class Sample5_4
5  {
6    public static void main (String[] args)
7    {
8      Car car1 = new Car();
9      car1.show();
10   }
11 }
```

匯入套件pc的Car類別

匯入以後就不用加套件名稱

Car.java

```
1  package pc;
2
3  public class Car
4  {
5      private int num;
6      private double gas;
7
8      public Car()
9      {
10       num = 0;
11       gas = 0.0;
12       System.out.println("已生產了汽車");
13     }
14
15     public void setCar(int n, double g)
16     {
17       num = n;
18       gas = g;
19
20       System.out.println("將車號設為"+num+"汽油量設為"+gas);
21     }
22
23     public void show()
24     {
25       System.out.println("車號是"+num);
26       System.out.println("汽油量是"+num);
27     }
28 }
```

Console Markers

<terminated> Sample5_4

已生產了汽車
車號是0
汽油量是0

輸出結果

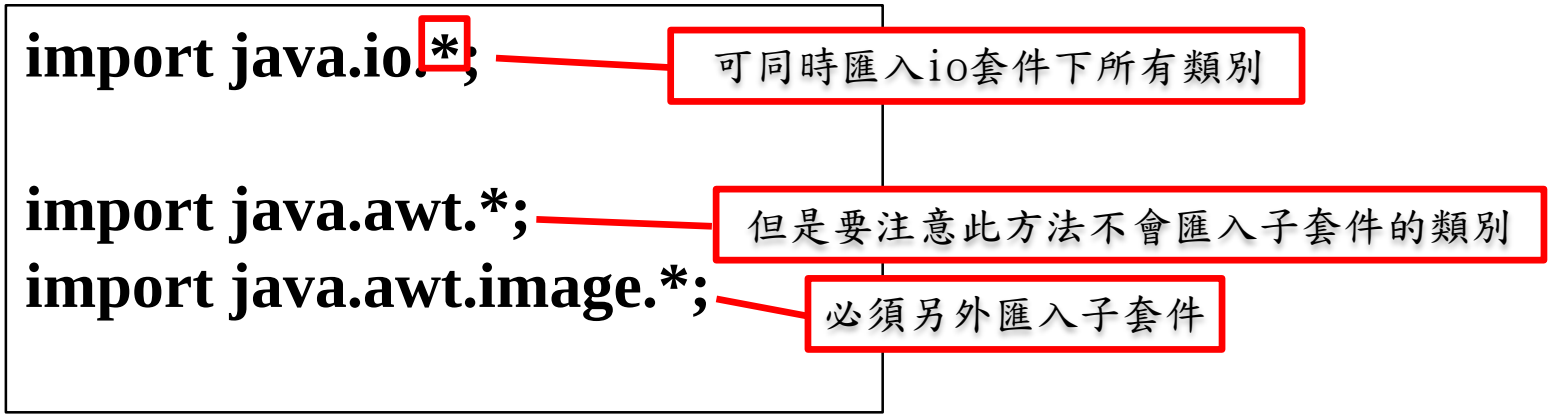
匯入(3/4)

在 Java 中已經把功能相近的類別 (class) 歸類到不同的套件當中，下表是比較常用的套件。

套件名稱	套件所包含之類別其主要功能
java.applet	與Java Applet相關的類別
java.awt	與視窗元件設計相關的類別
java.awt.event	與事件相關的類別
java.long	最基本的類別
java.io	與I/O相關的類別
java.net	與網路相關的類別
java.util	與Utility相關的類別

匯入(4/4)

如果想匯入同一個套件中的多個類別時，依照其面提到的方法就必須一個一個指定，匯入的工作會變得相當麻煩。因此，我們可以在後面加(*)一口氣匯入同一個套件中所有的類別。



本節介紹

- 例外的基礎知識
- 例外與類別
- 拋出例外
- 輸入與輸出的基礎知識

關鍵詞彙

- 例外
- 例外處理
- 拋出例外
- 子類別
- 輸入與輸出
- 串流
- 命令列引數

例外的基礎知識(1/7)

程式執行的時候經常會產生各種錯誤，例如下列都是可能發生情況：

- ✓ 執行一個處理檔案的程式，但是該程式卻找不到要處理的檔案在哪裏
- ✓ 執行一個存取陣列的程式，但是您所指定的陣列卻超出陣列元素的範圍，但是您卻在不知情的狀況下指定資料給這個超過範圍的陣列元素
- ✓ 設計一個程式將使用者輸入的字串轉成整數，但是使用者輸入的不是整數字串，導致轉換時發生錯誤

例外的基礎知識(2/7)

由於超過了陣列元素的數量，所以會發生例外

Sample5_1.java

```
1  class Sample5_1
2  {
3     public static void main (String[] args)
4     {
5        int[] test;
6        test = new int[5];
7
8        System.out.println("將值指定給test[10]");
9
10       test[10] = 80;
11       System.out.println("將80指定給test[10]");
12       System.out.println("順利地執行完畢");
13    }
14 }
```

Console | Markers | Properties | Servers | Data Source

```
<terminated> Sample5_1 [Java Application] C:\Program Files\Jav
將值指定給test[10]
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at Sample5_1.main(Sample5_1.java:10)
```

輸出結果

由於發生了例外，此兩行不會繼續處理

ArrayIndexOutOfBoundsException

例外的基礎知識(3/7)

如果想要更深入針對例外設計適當的處理方式，當然可以再建立相對的錯誤處理程式，這種錯誤處理程式稱為例外處理(exception handling)。

語法

```
try{
   需要例外檢查的程式敘述 ;
   …
}
catch(例外的類別 變數名稱){
   例外發生時要處理方式的程式敘述 ;
   …
}
```

例外的基礎知識(4/7)


包在try區塊的敘述會被檢查是否有例外
例外所以此行不會作處理
Sample5_2.java
1 class Sample5_2
2 {
3 public static void main (String[] args)
4 {
5 try
6 {
7 int[] test;
8 test = new int[5];
9
10 System.out.println("將值指定給test[10]");
11
12 test[10] = 80;
13 System.out.println("將80指定給test[10]");
14 }
15
16 catch (ArrayIndexOutOfBoundsException e)
17 {
18 System.out.println("超過陣列的範圍了");
19 }
20
21 System.out.println("順利地執行完畢了");
22 }
23 }
Console
Markers
<terminated> Sample5_2 [Ja
將值指定給test[10]
超過陣列的範圍了
順利地執行完畢了
輸出結果
由於超過了陣列元素的數量，所以會發生例外
發生超過了陣列元素的數量會被ctach區塊作處理
此行還是會繼續作處理

例外的基礎知識(5/7)

例外處理基本上是依循下列的順序進行處理：

- ✓ Step1： try程式區塊當中的例外發生時，原本執行中的程式必須中斷
- ✓ Step2：產生的例外種類，如果和catch()括號內的例外相符的話，catch會擷取這個例外，然後進入catch程式區塊內部繼續處理
- ✓ Step3： catch程式區塊處理結束後，繼續執行try～catch以後的其他程式

catch程式區塊

try程式區塊內所拋出的例外，由catch()接收後能夠進行適當的例外處理。如果找不到可以對try程式區塊所發生的例外加以處理的catch程式區塊，Java會回到處理中方法的原來呼叫程式，去尋找能進行處理的catch 區塊。

例外的基礎知識(6/7)

finally程式區塊專門用來記載與例外發生無關、但是希望最後一定會執行的程式。特別的是，即使例外狀況完全沒有發生，仍然會執行這裏面的程式內容。

語法

```
try{
   需要例外檢查的程式敘述;
   …
}catch(例外的類別 變數名稱){
   例外發生時的處理方式;
   …
}finally{
   絕對會執行的程式;
   …
}
```

例外的基礎知識(7/7)

Sample5_3.java

```
1  class Sample5_3
2  {
3    public static void main (String[] args)
4    {
5      try
6      {
7        int[] test;
8        test = new int[5];
9
10       System.out.println("將值指定給test[10]");
11
12       test[10] = 80;
13       System.out.println("將80指定給test[10]");
14     }
15
16     catch (ArrayIndexOutOfBoundsException e)
17     {
18       System.out.println("超過陣列的範圍了");
19     }
20
21     finally
22     {
23       System.out.println("最後一定會執行這個處理");
24     }
25
26     System.out.println("順利地執行完畢了");
27   }
28 }
```

不管有沒有例外最後一定都會作finally區塊內處理

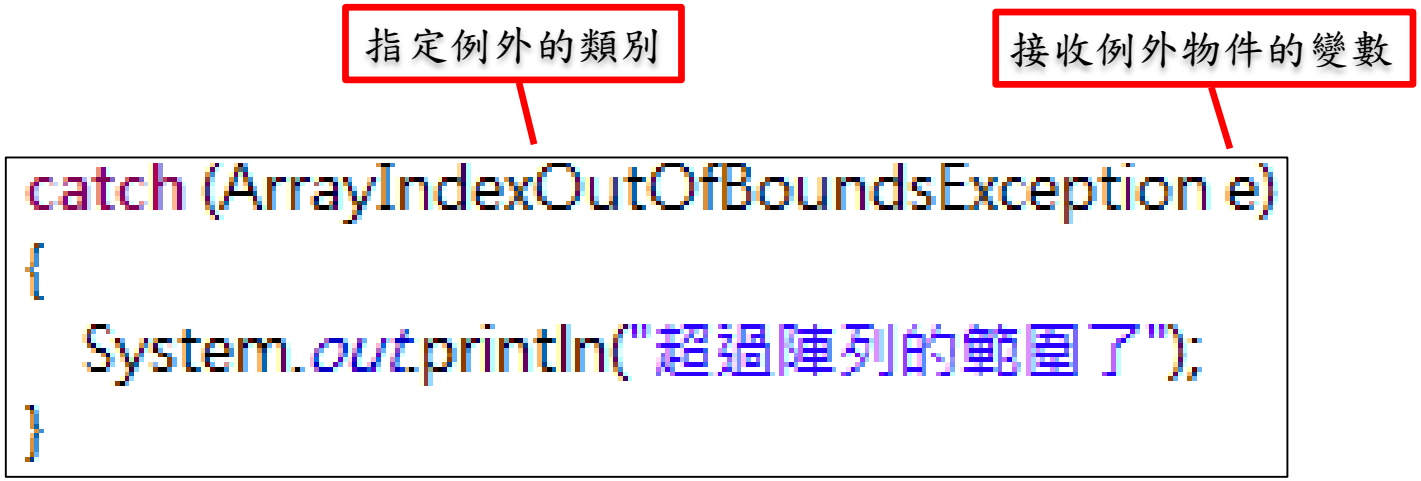
Console | Markers

<terminated> Sample5_3
將值指定給test[10]
超過陣列的範圍了
最後一定會執行這個處理
順利地執行完畢了

輸出結果

例外和類別(1/4)

本章所存取的「例外」，都是屬於類別庫當中：「**Throwable**這個類別→繼承而來的子類別→所產生的物件」，因此才會在catch()的括號內指定類別，緊接著再設定一個可以用來接收該物件的變數。之前的範例的例外是由Throwable類別的子類別延伸而成的ArrayIndexOutOfBoundsException類別的物件。



例外和類別(2/4)

Sample5_4.java

```
1  class Sample5_4
2  {
3    public static void main (String[] args)
4    {
5      try
6      {
7        int[] test;
8        test = new int[5];
9
10       System.out.println("將值指定給test[10]");
11
12       test[10] = 80;
13       System.out.println("將80指定給test[10]");
14     }
15
16     catch (ArrayIndexOutOfBoundsException e)
17     {
18       System.out.println("超過陣列的範圍了");
19       System.out.println("發生了"+e+"這個例外");
20     }
21
22     finally
23     {
24       System.out.println("最後一定會執行這個處理");
25     }
26
27     System.out.println("順利地執行完畢了");
28   }
29 }
```

接收到例外

輸出例外的種類

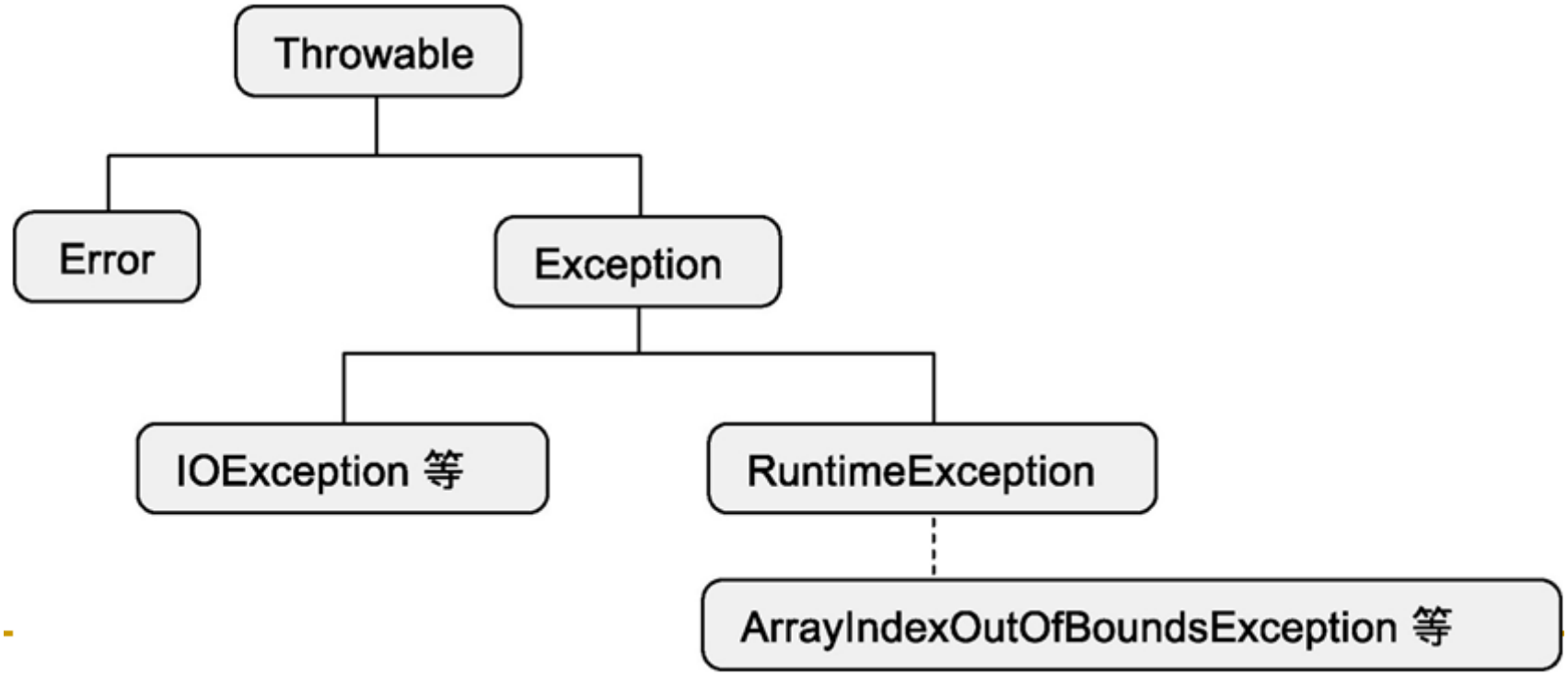
Console | Markers | Properties | Servers | Dat

```
<terminated> Sample5_4 (1) [Java Application] C:\Program
將值指定給test[10]
超過陣列的範圍了
發生了java.lang.ArrayIndexOutOfBoundsException: 10這個例外
最後一定會執行這個處理
順利地執行完畢了
```

輸出結果

例外和類別(3/4)

從Throwable類別延伸出Error類別和Exception類別。Error類別是指那些造成程式無法繼續執行的錯誤，通常也是比較嚴重的錯誤，在此我們討論到的例外處理都是屬於Exception類別。



例外和類別(4/4)

下面列出一些常發生的例外。

套件名稱	套件所包含之類別其主要功能
ClassNotFoundException	找不到指定的類別，可以發生在使用某個類別的方法，但卻找不到該方法所屬的類別
FileNotFoundException	找不到指定的檔案
IOException	檔案、網路…的輸出、入錯誤時產生的例外
InterruptedException	當某個執行緒中斷時，而另一個執行緒試圖使用「interrupt()」方法來中斷已停止執行的執行緒
ArrayIndexOutOfBoundsException	陣列的索引值指定錯誤，例如：超出索引值
NullPointerException	使用物件時，該物件的參考值為null
NumberFormatException	將字串轉為文字時，產生無法轉換的錯誤
ArithmeticException	運算式產生的例外，例如：除數為0

拋出例外(1/4)

前面介紹了接收例外並進行處理的技巧，但其實也可以反過來主動發生例外，讓自己建的類別能夠拋出例外。主要有下面兩個步驟:

- ✓ **宣告例外類別：**宣告延伸了Throwable類別之子類別繼承的例外類別。

- ✓ **拋出例外:** 使用throw敘述將例外拋出。

語法

```
傳回值的型態  method名稱（引數清單） throws  例外類別
```

語法

```
throw 指向例外物件的某變數;
```

拋出例外(2/4)

```
26  class Car
27  {
28    private int num;
29    private double gas;
30
31    public Car()
32    {
33      num = 0;
34      gas = 0.0;
35      System.out.println("已生產了汽車");
36    }
37
38    public void setCar(int n, double g) throws CarException
39    {
40      if (g < 0)
41      {
42        CarException e = new CarException();
43        throw e;
44      } else
45      {
46        num = n;
47        gas = g;
48        System.out.println("將車號設為" + num + "，汽油量設為" + gas);
49      }
50    }
51
52    public void show() {
53      System.out.println("車號是" + num);
54      System.out.println("汽油量是" + gas);
55    }
56  }
```

宣告此方法有可能拋出CarException例外

g小於零時被拋出例外

```
22  class CarException extends Exception
23  {
24  }
```

繼承Exception類別的自定義例外CarException

此頁範例接續下一頁

拋出例外(3/4)

Sample5_5.java

```
1  class Sample5_5
2  {
3    public static void main(String[] args)
4    {
5
6      Car car1;
7      car1 = new Car();
8
9      try {
10       car1.setCar(1234, -10.0);
11     }
12
13     catch (CarException e) {
14       System.out.println("拋出" + e + "了");
15     }
16
17     car1.show();
18   }
19
20 }
```

呼叫此方法來拋出例外

Console

<terminated> Sample5_

已生產了汽車
拋出CarException了
車號是0
汽油量是0.0

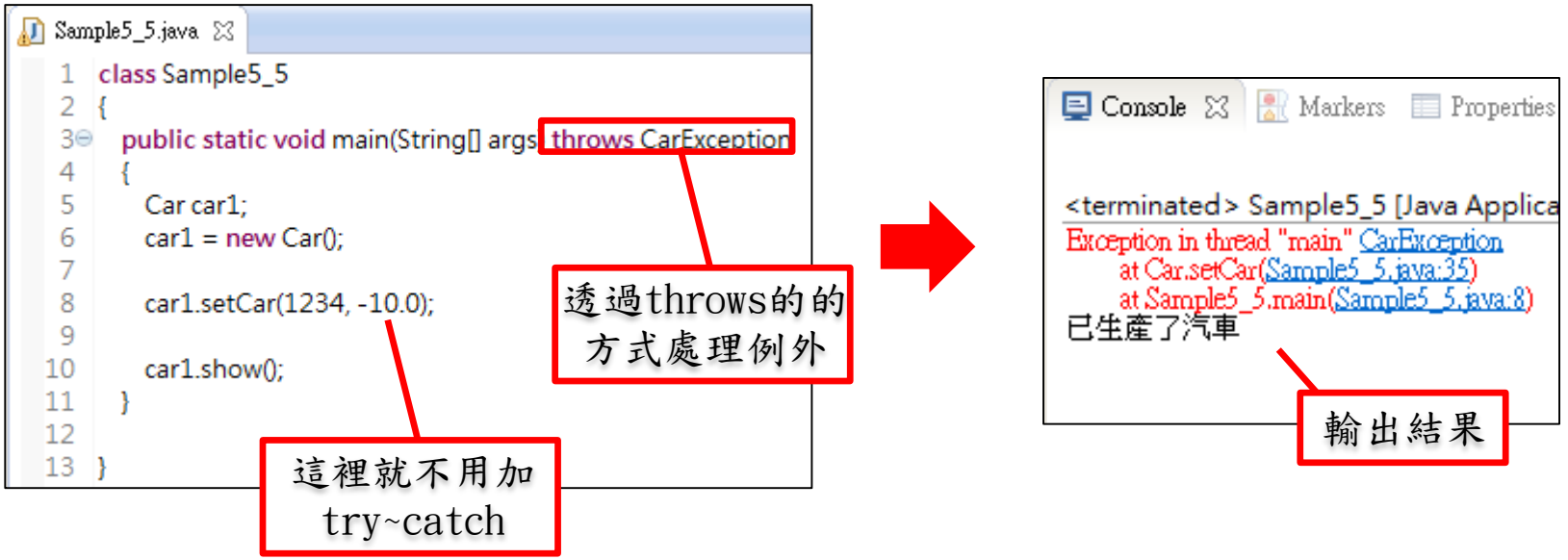
輸出結果

拋出例外(4/4)

撰寫一個可能會送出例外的方法時，有兩種處理方法：

- ✓ 使用**try～catch**，在方法內直接處理例外
- ✓ 透過**throws**，把處理例外的工作交給原呼叫程式所在的方法

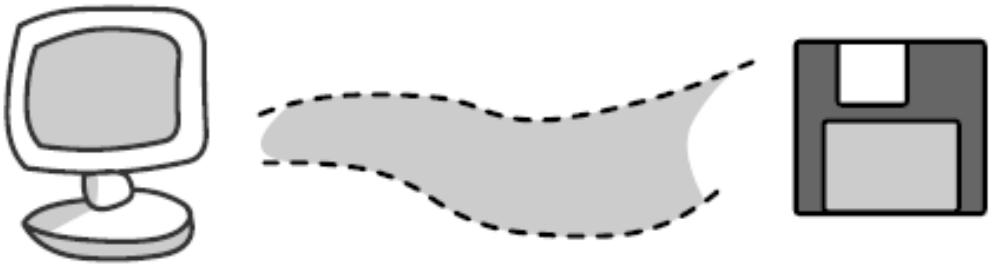
剛剛的範例5-5是使用try~catch的方式，現在我們改寫成用throws的方式。



輸入與輸出的基礎知識(1/8)

對於各式各樣的設備有著統一的輸入方法，此種輸出輸入的功能，就稱為串流(stream)。串流是指可以輸出到螢幕畫面的程式碼，或是從鍵盤輸入的程式碼。

- ✓ System.out : 標準輸出串流
- ✓ System.in : 標準輸入串流



輸入與輸出的基礎知識(2/8)

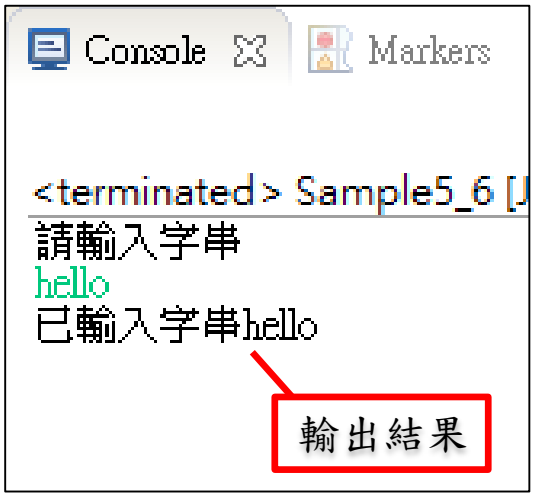
Sample5_6.java

```
1  import java.io.BufferedReader;
4
5  class Sample5_6
6  {
7    public static void main(String[] args)
8    {
9      System.out.println("請輸入字串");
10
11     try
12     {
13       BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
14
15       String str = br.readLine();
16
17       System.out.println("已輸入字串"+str);
18     }
19
20     catch(IOException e)
21     {
22       System.out.println("輸入輸出有誤");
23     }
24   }
25
26 }
```

指定標準輸入並建立文字串流

透過緩衝區來讀入

從緩衝區讀取一行資料



輸入與輸出的基礎知識(3/8)

將資料寫入到檔案

輸入與輸出的基礎知識(4/8)

從檔案讀取資料(此範例的text1.txt要從上一個範例的輸出路徑取得)

Sample5_8.java

```
1  import java.io.BufferedReader;
7
8  class Sample5_8
9  {
10   public static void main(String[] args)
11   {
12     try
13     {
14       BufferedReader br = new BufferedReader(new FileReader("test1.txt"));
15
16       String str1 = br.readLine();
17       String str2 = br.readLine();
18
19       System.out.println("寫入到檔案中的兩個字串是");
20       System.out.println(str1);
21       System.out.println(str2);
22
23       br.close();
24     }
25
26     catch(IOException e)
27     {
28       System.out.println("輸出入錯誤");
29     }
30   }
31 }
```

指定檔案名稱並建立從檔案讀取資料的文字串流

透過緩衝區讀入資料

從緩衝區一行一行讀入資料

Console | Markers

<terminated> Sample5_8 [Ja

寫入到檔案中的兩個字串是
Hello!
GoodBye!

輸出結果

輸入與輸出的基礎知識(5/8)

使用命令列引數

命令列引數就是會直接擷取使用者在程式執行階段所輸入的資料，再傳遞給內部程式進行處理的功能。命令列引數會傳到下列main()方法的參數中。

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


}
```

接收使用者輸入的字串

輸入與輸出的基礎知識(6/8)

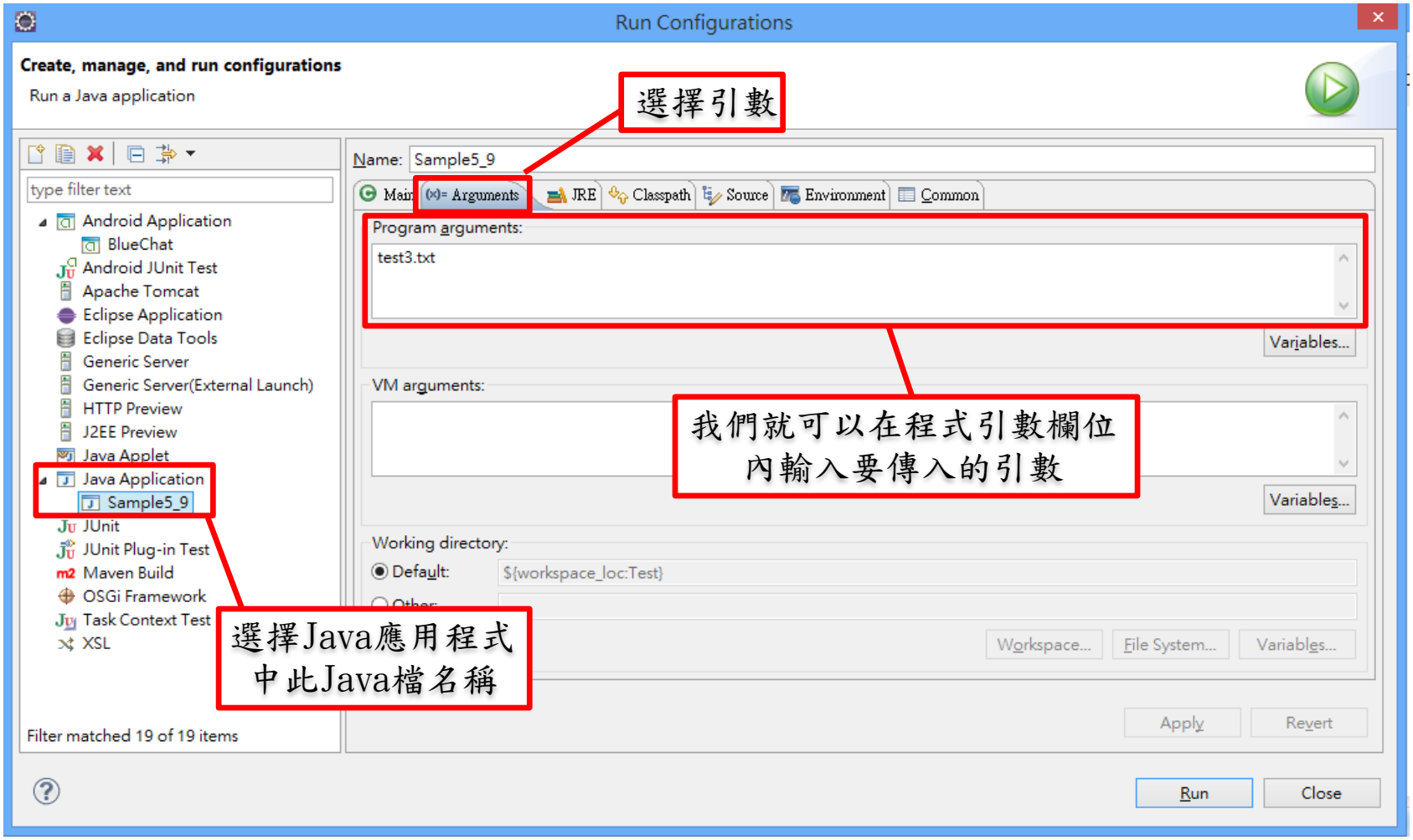
一般Java在使用命令提示字元執行值會下指令來執行程式，例如:

但是我們可以在將要傳入的檔案名稱加進去，便可以Java的Main方法的參數，例如:

後面的test3.txt字串就是命令列引數，但是我們是使用eclipse的開發平台，所以無法使用此方式傳入，在下一頁將介紹如何用eclipse傳入引數。

輸入與輸出的基礎知識(7/8)

在eclipse上方的工具列點選Run=>Run Configurations，會看到下圖Run Configurations頁面。



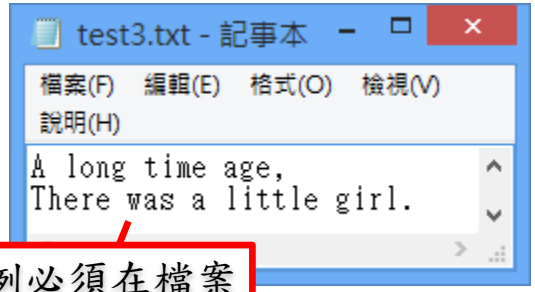
輸入與輸出的基礎知識(8/8)

Sample5_9.java

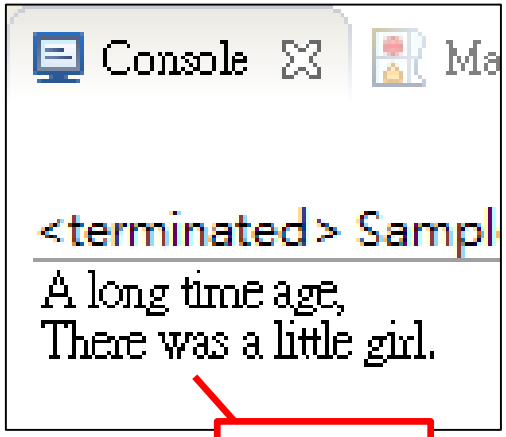
```
1  import java.io.BufferedReader;
7
8  class Sample5_9
9  {
10   public static void main(String[] args)
11   {
12     if (args.length !=1)
13     {
14       System.out.println("請指定正確的檔案名稱");
15       System.exit(1);
16     }
17
18     try
19     {
20       BufferedReader br = new BufferedReader(new FileReader(args[0]));
21
22       String str;
23       while ((str = br.readLine()) !=null)
24       {
25         System.out.println(str);
26       }
27
28       br.close();
29     }
30
31     catch (IOException e)
32     {
33       System.out.println("輸出入錯誤");
34     }
35   }
36 }
```

檢查使用者輸入的字串數量

從輸入的第一個字串(檔案名稱)開始建立文字串流



本範例必須在檔案目錄下建好此檔案



輸出結果

本節介紹

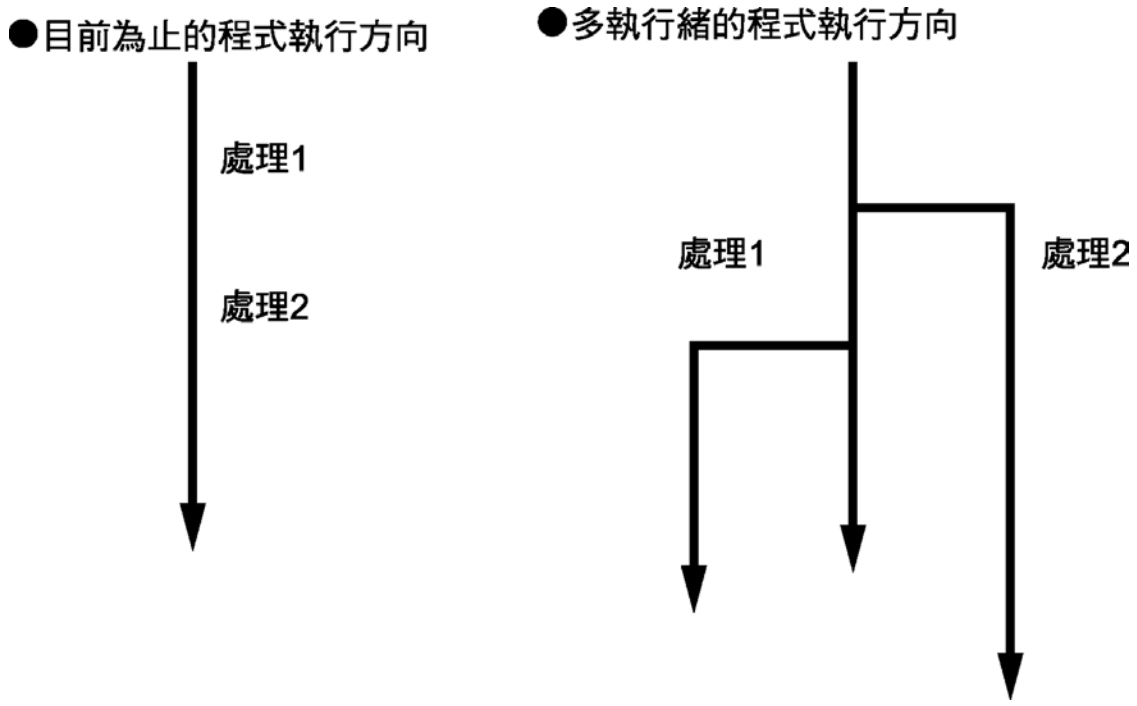
- 執行緒的基礎知識
- 同步化

關鍵詞彙

- 執行緒
- run
- sleep
- join
- Runnable
- 同步化

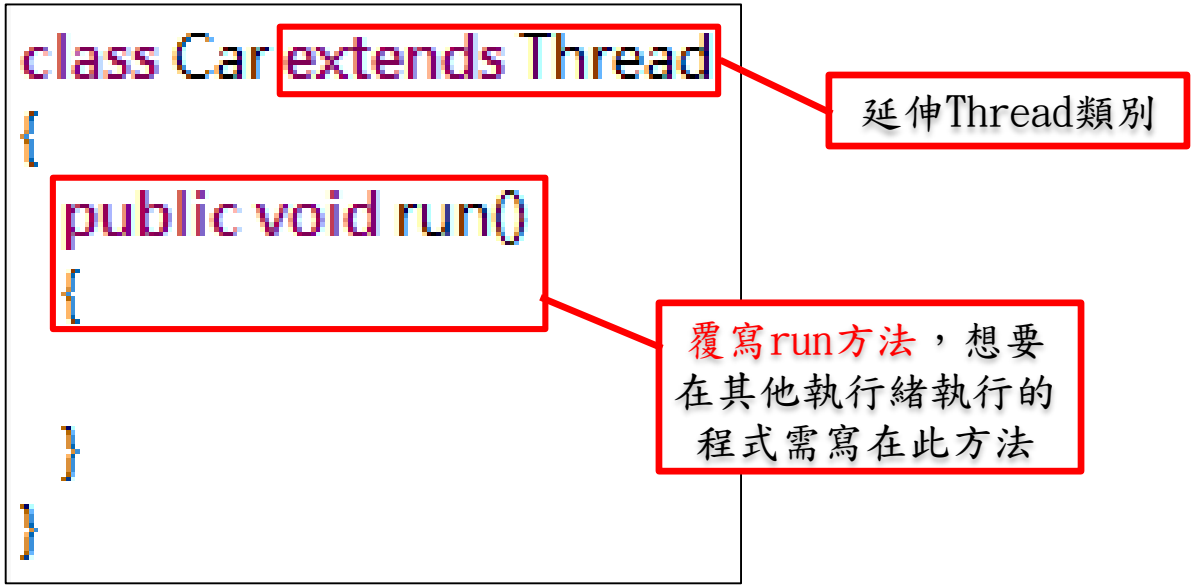
執行緒的基礎知識(1/9)

在本章之前的範例程式都是循著一條流程來進行處理的，在Java中可以同時擁有多個處理流程。也就是說可以同時執行程式碼中不同的部分，這樣的一個程式處理流程就稱為**執行緒(thread)**。



執行緒的基礎知識(2/9)

啟動執行緒之前，必須做好執行緒的準備工作，也就是從類別庫的Thread類別當中，延伸出一個子類別。



執行緒的基礎知識(3/9)

Sample6_1.java

```
1  class Sample6_1
2  {
3    public static void main(String[] args)
4    {
5      Car car1 = new Car("1號車");
6      car1.start();
7
8      for(int i=0; i<5; i++)
9      {
10       System.out.println("正在進行main()的處理工作");
11     }
12   }
13 }
14
15 class Car extends Thread
16 {
17   private String name;
18
19   public Car(String nm)
20   {
21     name = nm;
22   }
23
24   public void run()
25   {
26     for(int i=0; i<5; i++)
27     {
28       System.out.println("正在進行" + name + "的處理工作");
29     }
30   }
31 }
```

啟動新執行緒

延伸Thread類別

覆寫run方法，想要在其他執行緒執行的程式需寫在此方法

由於兩個執行緒時同時在作處理，所以每次完成的順序也可能有差異

Console Markers

<terminated> Sample6_1 [Java

```
正在進行main()的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
```

輸出結果

Console Markers

<terminated> Sample6_1 [Jav

```
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
```

輸出結果

執行緒的基礎知識(4/9)

啟動多個執行緒。

Sample6_2.java

```
1  class Sample6_2
2  {
3    public static void main(String[] args)
4    {
5      Car car1 = new Car("1號車");
6      car1.start();
7
8      Car car2 = new Car("2號車");
9      car2.start();
10
11     for(int i=0; i<5; i++)
12     {
13       System.out.println("正在進行main()的處理工作");
14     }
15   }
16 }
17
18 class Car extends Thread
19 {
20   private String name;
21
22   public Car(String nm)
23   {
24     name = nm;
25   }
26
27   public void run()
28   {
29     for(int i=0; i<5; i++)
30     {
31       System.out.println("正在進行" + name + "的處理工作");
32     }
33   }
34 }
```

啟動新執行緒

在啟動一個新執行緒

Console Markers

<terminated> Sample6_2 [Java

```
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行2號車的處理工作
正在進行2號車的處理工作
正在進行2號車的處理工作
正在進行2號車的處理工作
正在進行2號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
```

輸出結果

Console Markers

<terminated> Sample6_2 [Java

```
正在進行main()的處理工作
正在進行2號車的處理工作
正在進行2號車的處理工作
正在進行2號車的處理工作
正在進行2號車的處理工作
正在進行2號車的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
```

輸出結果

執行緒的基礎知識(5/9)

暫時停止執行緒sleep方法。

Sample6_3.java

```
1  class Sample6_3
2  {
3    public static void main(String[] args)
4    {
5      Car car1 = new Car("1號車");
6      car1.start();
7
8      for(int i=0; i<5; i++)
9      {
10       System.out.println("正在進行main()的處理工作");
11     }
12   }
13 }
14
15 class Car extends Thread
16 {
17   private String name;
18
19   public Car(String nm)
20   {
21     name = nm;
22   }
23
24   public void run()
25   {
26     for(int i=0; i<5; i++)
27     {
28       try
29       {
30         sleep(1000);
31         System.out.println("正在進行" + name + "的處理工作");
32       }
33       catch(InterruptedException e)
34       {
35       }
36     }
37   }
38 }
```

每當處理到這個敘述句，執行緒會暫停1000毫秒

可能被sleep方法拋出的例外

此部分每隔一秒輸出一次

Console Markers

<terminated> Sample6_3 [J

```
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
```

輸出結果

執行緒的基礎知識(6/9)

暫時停止主執行緒(main thread)。

Sample6_4.java

```
1  class Sample6_4
2  {
3    public static void main(String[] args)
4    {
5      Car car1 = new Car("1號車");
6      car1.start();
7
8      for(int i=0; i<5; i++)
9      {
10       try
11       {
12         Thread.sleep(1000); //讓執行緒小睡1秒
13         System.out.println("正在進行main()的處理工作");
14       }
15       catch(InterruptedException e)
16       {
17       }
18     }
19   }
20 }
21
22 class Car extends Thread
23 {
24   private String name;
25
26   public Car(String nm)
27   {
28     name = nm;
29   }
30
31   public void run()
32   {
33     for(int i=0; i<5; i++)
34     {
35       System.out.println("正在進行" + name + "的處理工作");
36     }
37   }
38 }
```

每當處理到這個敘述句，主執行緒會暫停1000毫秒

可能被sleep方法拋出的例外

Console | Markers

<terminated> Sample6_4 [Ja

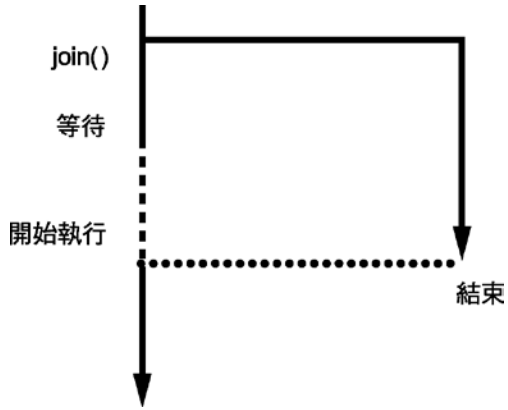
```
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行1號車的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
```

此部分每隔一秒輸出一次

輸出結果

執行緒的基礎知識(7/9)

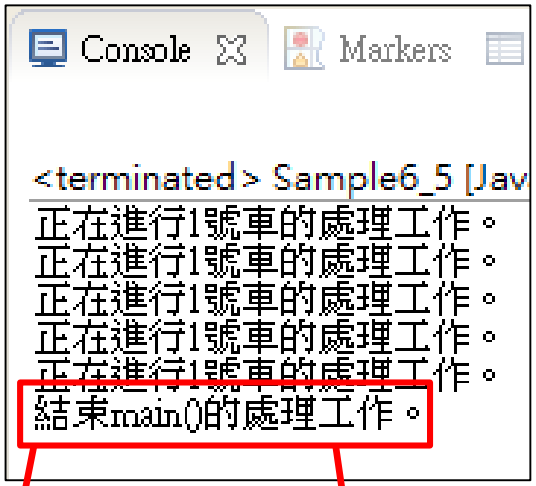
等待執行緒處理完畢join方法。



Sample6_5.java

```
1  class Sample6_5
2  {
3    public static void main(String[] args)
4    {
5      Car car1 = new Car("1號車");
6      car1.start();
7
8      try
9      {
10       car1.join();
11     }
12
13     catch(InterruptedException e)
14     {
15
16     }
17
18     System.out.println("結束main()的處理工作。");
19   }
20 }
21
22 class Car extends Thread
23 {
24   private String name;
25
26   public Car(String nm)
27   {
28     name = nm;
29   }
30
31   public void run()
32   {
33     for(int i=0; i<5; i++)
34     {
35       System.out.println("正在進行" + name + "的
36     }
37   }
38 }
```

中斷現在在處理的執行緒，直到car1執行緒處理完才繼續執行



等到car1執行緒處理完才執行main的處理

輸出結果

執行緒的基礎知識(8/9)

如果本身已經是Thread類別的子類別，卻又不得不同時從其他類別加以繼承的話，此時可使用類別庫當中的Runnable介面，透過它還是可以正常啟動執行緒。

```
class Car extends Thread, Vehicle
{

}
```

無法延伸兩個類別

透過實作Runnable介面
也可以新增執行緒

```
class Car extends Vehicle implements Runnable
{

}
```

執行緒的基礎知識(9/9)

Sample6_6.java

```
1  class Sample6_6
2  {
3    public static void main(String[] args)
4    {
5      Car car1 = new Car("一號車");
6
7      Thread th1 = new Thread(car1);
8      th1.start();
9
10     for (int i=0; i<5; i++)
11     {
12       System.out.println("正在進行main()的處理工作");
13     }
14   }
15 }
16
17 class Car implements Runnable
18 {
19   private String name;
20
21   public Car(String nm)
22   {
23     name = nm;
24   }
25
26   public void run()
27   {
28     for (int i=0; i<5; i++)
29     {
30       System.out.println("正在進行"+name+"的處理");
31     }
32   }
33 }
```

建立Thread類別的物件

啟動執行緒

實作Runnable介面

覆寫run方法

小知識

run方法本身就是定義在Runnable介面的方法，而Thread類別本身實作Runnable介面，所以我們之前才能繼承Thread類別父寫run方法

Console Markers

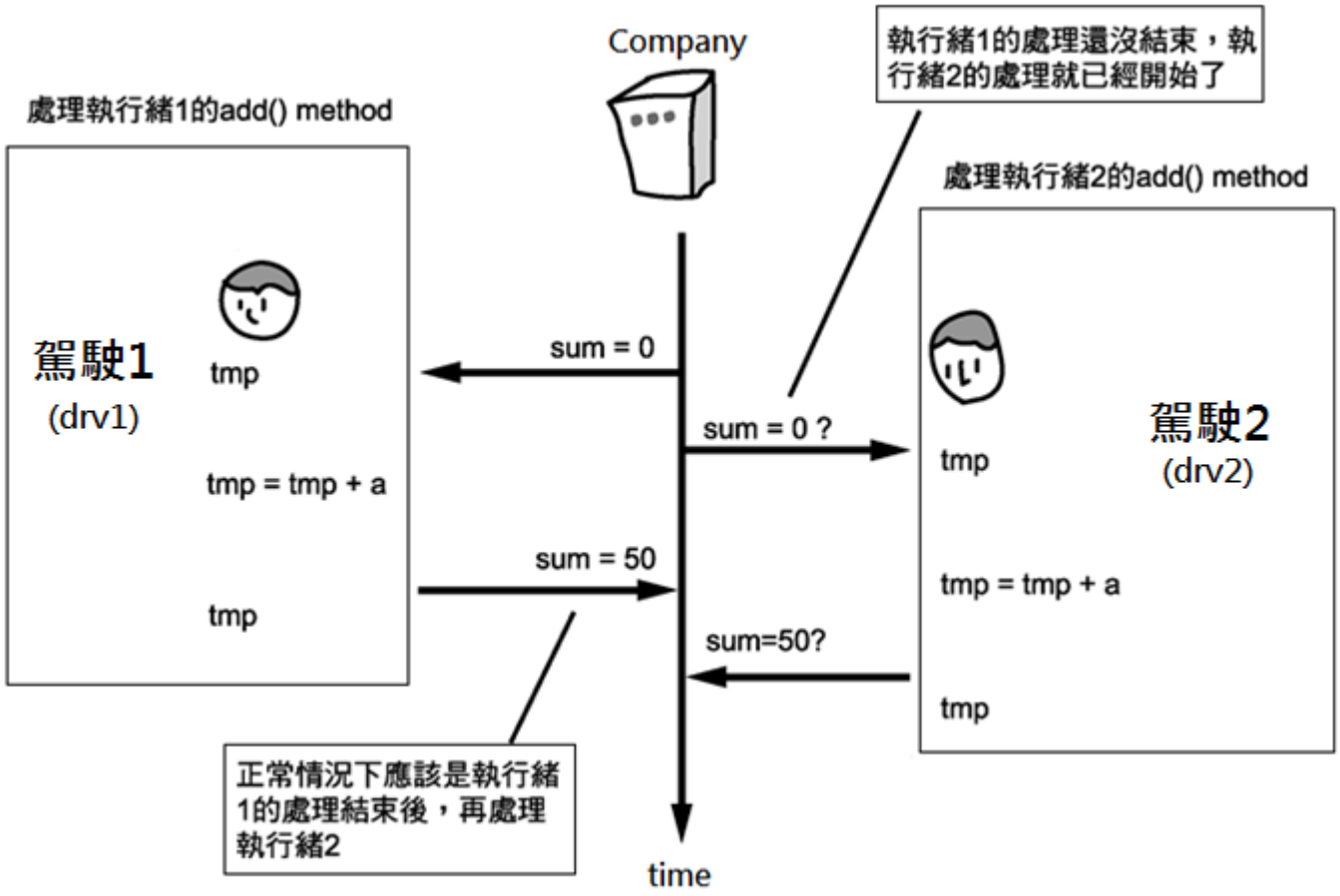
<terminated> Sample6_6 [Java

```
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行main()的處理工作
正在進行一號車的處理
正在進行一號車的處理
正在進行一號車的處理
正在進行一號車的處理
正在進行一號車的處理
```

輸出結果

同步化(1/5)

我們在處理多執行緒時也可能會發生一些預期外的錯誤，所以有時要控制執行緒間的處理時機，這樣的機制就叫做同步化(synchronization)。



同步化(2/5)

建立公司物件

```
Sample6_7.java
1  class Sample6_7
2  {
3    public static void main(String[] args)
4    {
5      Company cmp = new Company();
6
7      Driver drv1 = new Driver(cmp);
8      drv1.start();
9
10     Driver drv2 = new Driver(cmp);
11     drv2.start();
12   }
13 }
```

建立駕駛物件1

建立駕駛物件2

匯入金額的處理過程

```
14 class Company
15 {
16   private int sum = 0;
17   public  void add(int a) {
18     int tmp = sum;
19     System.out.println("目前的合計金額是" + tmp + "元");
20     System.out.println("賺到" + a + "元了。");
21     tmp = tmp + a;
22     System.out.println("合計金額是" + tmp + "元");
23     sum = tmp;
24   }
25 }
26 class Driver extends Thread
27 {
28   private Company comp;
29   public Driver(Company c){
30     comp = c;
31   }
32
33   public void run(){
34     for(int i=0; i<3; i++){
35       comp.add(50);
36     }
37   }
38 }
```

由於兩名駕駛同時使用add方法才導致的這種錯誤的結果

進行匯入金額

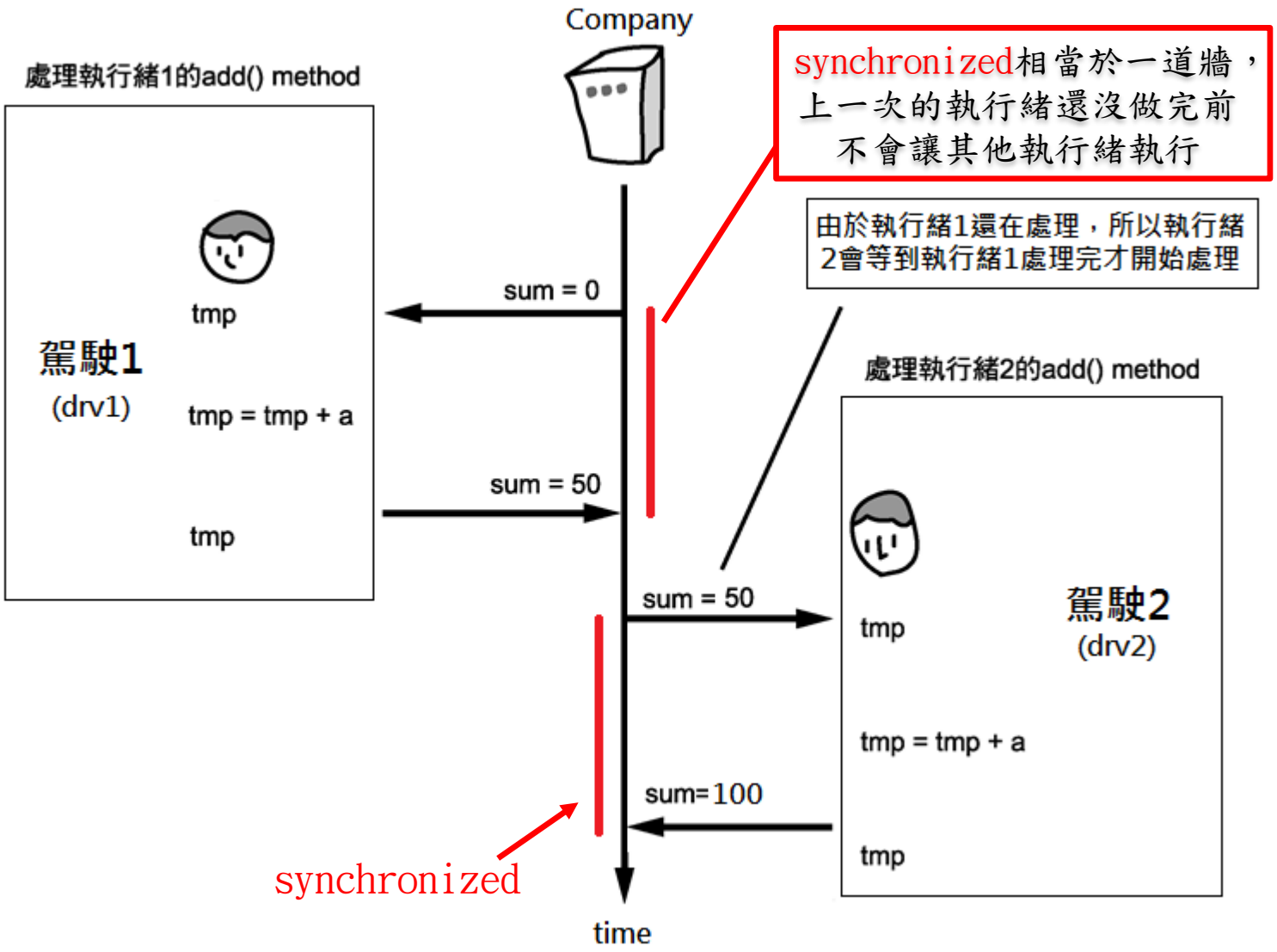
```
Console  Markers
<terminated> Sample6_7 [
目前的合計金額是0元
賺到50元了。
合計金額是50元
目前的合計金額是50元
賺到50元了。
合計金額是100元
目前的合計金額是100元
賺到50元了。
合計金額是150元
目前的合計金額是0元
賺到50元了。
合計金額是50元
目前的合計金額是50元
賺到50元了。
合計金額是100元
目前的合計金額是100元
賺到50元了。
合計金額是150元
```

輸出結果

同步化(3/5)

為了避免上個範例多個執行緒同時呼叫導致錯誤的結果，我們可以使用**synchronized修飾子**，當某個執行緒正在使用加了此修飾子的方法，其他執行緒就必須等它做完才能呼叫。

同步化(4/5)


Company
處理執行緒1的add() method
tmp
駕駛1
(drv1)
tmp = tmp + a
tmp
sum = 0
sum = 50
synchronized相當於一道牆，
上一次的執行緒還沒做完前
不會讓其他執行緒執行
由於執行緒1還在處理，所以執行緒
2會等到執行緒1處理完才開始處理
處理執行緒2的add() method
sum = 50
tmp
駕駛2
(drv2)
tmp = tmp + a
sum=100
tmp
synchronized
time

5.3 執行緒

同步化(5/5)

加上此修飾子避免多執行緒造成的錯誤

Sample6_8.java

```
1  class Sample6_8
2  {
3    public static void main(String[] args)
4    {
5      Company cmp = new Company();
6
7      Driver drv1 = new Driver(cmp);
8      drv1.start();
9
10     Driver drv2 = new Driver(cmp);
11     drv2.start();
12   }
13 }
```

```
14 class Company
15 {
16   private int sum = 0;
17   public synchronized void add(int a)  {
18     int tmp = sum;
19     System.out.println("目前的合計金額是" + tmp + "元");
20     System.out.println("賺到" + a + "元了。");
21     tmp = tmp + a;
22     System.out.println("合計金額是" + tmp + "元");
23     sum = tmp;
24   }
25 }
26 class Driver extends Thread
27 {
28   private Company comp;
29   public Driver(Company c){
30     comp = c;
31   }
32
33   public void run(){
34     for(int i=0; i<3; i++){
35       comp.add(50);
36     }
37   }
38 }
```

Console Markers

<terminated> Sample6_8 [J

```
目前的合計金額是50元
賺到50元了。
合計金額是100元
目前的合計金額是100元
賺到50元了。
合計金額是150元
目前的合計金額是150元
賺到50元了。
合計金額是200元
目前的合計金額是200元
賺到50元了。
合計金額是250元
目前的合計金額是250元
賺到50元了。
合計金額是300元
```

輸出結果