

例外處理

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

前言

- 在「編譯」程式或「執行」程式時，常會遇到各種不同錯誤，以致無法正常完成工作。
- 研發軟體時，最容易遇到的三種錯誤（Bug）：語法錯誤、執行時期錯誤（Runtime error）、邏輯錯誤。
 - ◆ 語法錯誤：
 - 語法錯誤是初學者最容易犯的錯誤。
 - 在編譯過程中，系統常能立即指出此種錯誤的所在，並要求程式設計者修正後才能正式執行。
 - 這樣錯誤最容易解決，只要熟悉語法多多練習就可以減少錯誤產生。

前言

◆ 執行時期錯誤：

- 程式在執行時，有時候會因下列情況而使得程式中斷執行，

例如：

「輸入資料不符合要求」或「在計算過程分母為0」或
「磁碟中無此檔案存在」或「陣列的索引值超出陣列宣告範圍」或...等。

- 這種錯誤的問題在編譯時，並不會發生，被Java稱為「例外」，而Java也提供了例外處理的方式來解決問題。

◆ 邏輯錯誤：

- 邏輯錯誤是最難找出的，尤其在大型應用程式最為明顯。

前言

- 程式在執行過程並沒有出現錯誤，也會有執行結果，甚至有時候結果是正確的。
- 除非仔細觀察，多人多次的測試，否則不見得會發現。
- 因此誤信程式完全正確，推出正式使用，應用過程又沒有注意異常狀況，往往造成很大損失。
- 有些系統有提供偵錯（Debug）工具，用來協助找出錯誤之處。
- 若沒有偵錯工具，就只能自己設定「偵測點」，輸出目前「主要變數內容」是否如「預測結果」，以推測可能錯誤之處，再仔細研讀程式，尋找邏輯上錯誤之處，加以修正。

Throwable

- Java產生的「錯誤（Error）」及「例外（Exception）」都是一個物件，屬於Object下的Throwable類別的子類別，關係如下：

```
java.lang.Object
├── java.lang.Throwable
│   ├── java.lang.Error
│   └── java.lang.Exception
```

Throwable

■ Throwable類別擁有兩個直接的子類別：

◆ Error：

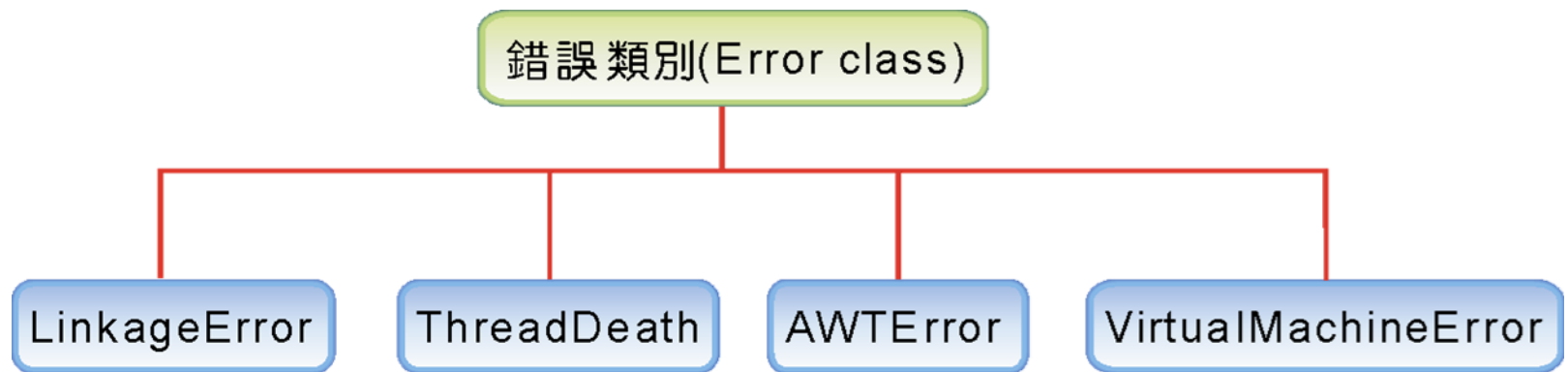
- 這個類別及其衍生的類別是屬於JVM的嚴重錯誤。
- 這種錯誤會導致程式終止執行，沒有辦法用使用「例外」來處理這種錯誤。
- 其衍生的類別有：
 - ◆ LinkageError
 - ◆ 類別間的連結或作用不當時所使用的Error類別。
 - ◆ 例如：類別型態錯誤（ClassFormatError）。
 - ◆ ThreadDeath
 - ◆ 程式執行時，發生不明狀況引起錯誤時所使用的Error類別。
 - ◆ 例如：除法運算的除數為零。

Throwable

- ◆ AWTError
 - ◆ 程式執行AWT（抽象視窗工具）所使用的Error類別。
- ◆ VirtualMachineError
 - ◆ Java虛擬機器發生錯誤所使用的Error類別。

例如：

超出記憶體使用範圍（OutOfMemoryError）。



Throwable

◆ Exception :

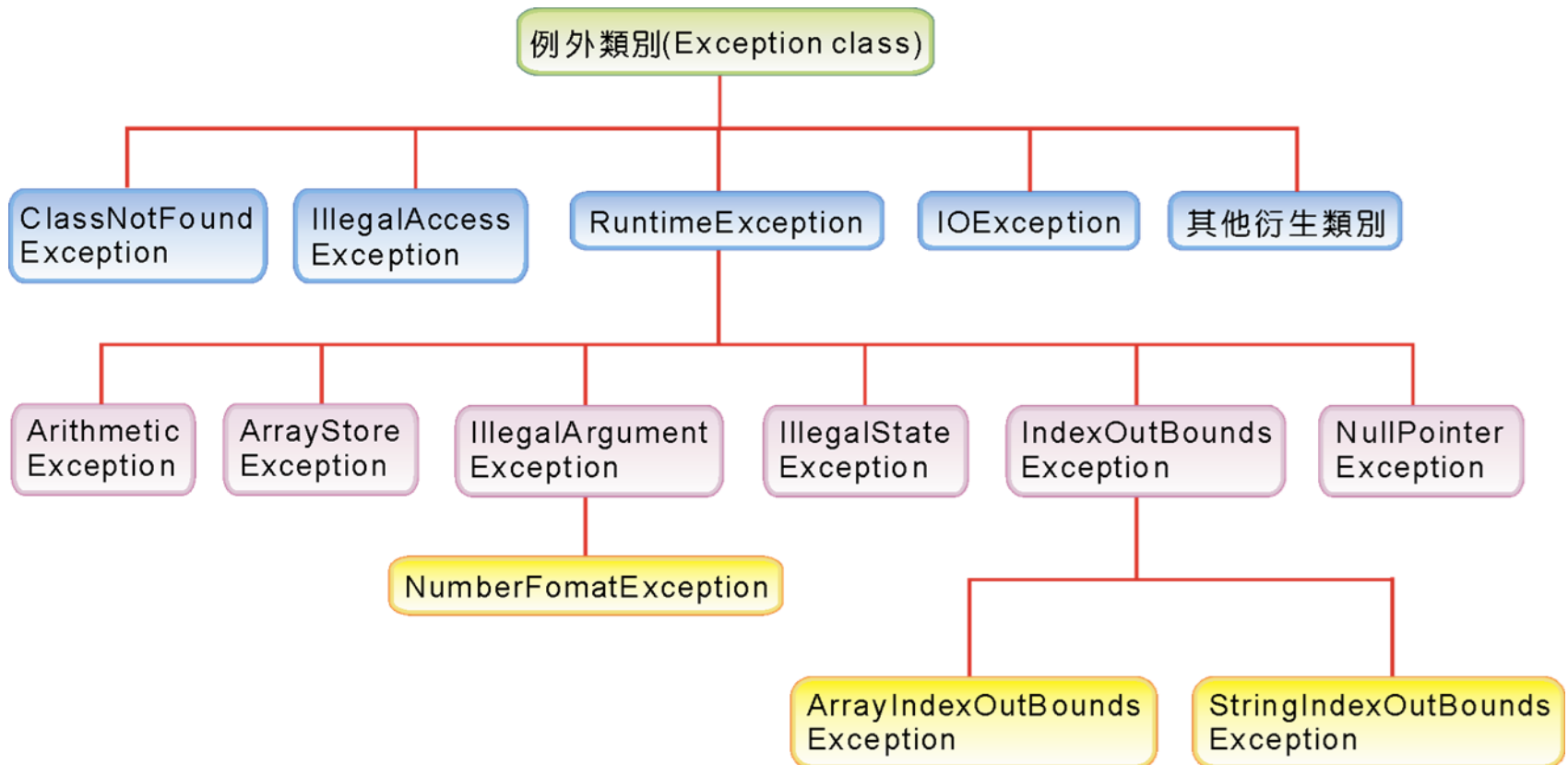
- 這個類別及其衍生的類別是屬於可以用「較不嚴重」的錯誤。
- 這種錯誤沒有嚴重到須要終止程式的執行。
- 可以使用「例外」來處理這種錯誤，防止程式執行終止。
- 常用的衍生類別有：
 - ◆ ClassNotFoundException
 - ◆ 應用程式載入.class檔而找不到時的錯誤。
 - ◆ IllegalAccessException
 - ◆ 載入的.class，方法或相關資料有權限問題時的錯誤。
 - ◆ RuntimeException
 - ◆ JVM執行時所產生的例外錯誤而使用的Exception類別。
◦ 此類別底下還有衍生類別。

Throwable

- ◆ IOException
 - ◆ 程式輸入、輸出時發生的錯誤，例如：檔案未關閉。
- ◆ ArithmeticException
 - ◆ 算數運算錯誤，例如：除數為0。
- ◆ ArrayStoreException
 - ◆ 存入陣列的資料與陣列宣告型別不同的錯誤。
- ◆ IllegalArgumentException
 - ◆ 呼叫方法時，參數型態不同所產生的錯誤。
- ◆ NumberFormatException
 - ◆ 字串轉成數值的錯誤。
- ◆ IllegalStateException
 - ◆ 應用程式與環境的狀態錯誤。
- ◆ ArrayIndexOutOfBoundsException
 - ◆ 陣列索引超出宣告範圍的錯誤。

Throwable

- ◆ StringIndexOutOfBoundsException
 - ◆ 字串索引超出範圍的錯誤。
- ◆ NullPointerException
 - ◆ 物件值為null產生的錯誤。



捕捉例外

- 當程式在執行時期發生錯誤或不正常狀況，稱之為「例外」。
- 進行「例外處理」是不希望程式「被中斷」執行，而是希望程式能「捕捉」到錯誤，讓程式開發人員能確切知道問題出在哪裡，並能讓程式繼續執行。

例如：

若錯誤是使用者輸入不正確資料所造成的，可以要求使用者輸入正確資料再繼續執行，或者不處理使用者輸入資料繼續做其他工作。

- 利用條件判斷式（if...else）來避免錯誤的發生，這樣的檢查方式在一些程式語言中經常出現。

捕捉例外

- 然而，處理「錯誤」的邏輯與「處理」業務的邏輯混在一起，如果有更多的錯誤狀況必須檢查的話，程式會更難以閱讀。
- 由於使用了一些判斷式，即使發生機率低的錯誤，也都必須一視同仁的進行判斷檢查，這會使得程式的執行效能受到一定程度的影響。
- Java是使用「try ... catch ... finally」敘述來捕捉並解決「例外」。
 - ◆ 它的方式是將被監視的敘述區段寫在「try」大括號內，當程式執行到「try」內的敘述有發生錯誤時，會逐一檢查「捕捉（catch）」該錯誤，以便執行該「catch」內敘述。

捕捉例外

 語法：

```
try
{
    可能發生例外的程式碼
}
catch(例外類別 例外物件)
{
    處理例外事件的區塊
}
finally
{
    一定會處理的區塊
}
```

捕捉例外

■ 說明：

◆ try：

- 會對此區塊的程式碼依序進行檢查，若有「例外」狀況發生，則會依據「例外」事件的類別將此「例外」物件丟給相關的「catch」區塊處理。
- 會產生的「例外」狀況不限一種，所以「catch」區塊的設計則相當重要。

注意：

- ◆ 一組「try ... catch ... finally」的敘述，只能有一個「try」區塊。

◆ catch：

- 處理捕捉到的「例外」物件。

捕捉例外

- ◆ 依據「catch」參數列中所使用的「例外」類別來決定要捕捉的是何種「例外」物件。
- ◆ 每個「catch」區塊，只能捕捉「一種」「例外」物件。
 - ◆ 可以設計「多個」「catch」區塊，讓每個區塊來負責捕捉「一種」「例外」物件。
 - ◆ 可以在最後一個「catch」區塊用「Exception」類別，來捕捉遺漏的「例外」。
- ◆ 捕捉到「例外」後，可透過該「例外」「物件」的「方法」，得到「例外」的資訊，常用的「方法」有：
 - ◆ toString()
以簡單的字串描述該例外。
 - ◆ getMessage()
列出細節訊息。
 - ◆ printStackTrace()
將堆疊資訊印在螢幕上，可幫助設計者快速找到錯誤點。

捕捉例外

注意：

- ◆ 若有多個「catch」區塊時，由上至下的「catch」逐一檢查，若遇到符合條件，則執行該對應敘述區段，以下「catch」就不再處理。
- ◆ 若用「Exception」類別，來捕捉遺漏的「例外」，一定要放在最後一個「catch」區塊

◆ finally：

- 不論是否有「例外」發生，都會執行此區塊。
- 既使在「try」區塊中，有發生「例外」狀況，或是「catch」區塊「沒有」「捕捉」到「例外」物件，都可以將「補救措施」放在這個區塊以作「最後處理」。

例如：

檔案的關閉動作。

捕捉例外

注意：

- 一個「try」區塊，必須有對應的「catch」區塊或是「finally」區塊。
 - ◆ 如果有設定「catch」區塊，則「finally」區塊可有可無。
 - ◆ 如果沒有定義「catch」區塊，則一定要有「finally」區塊。

捕捉例外

◆ 程式：

```
package CH08_01;
```

```
public class CH08_01
```

```
{
```

```
    static void Division(int num1, int num2)
```

```
    {
```

```
        try
```

```
        {
```

```
            int num3=num1/num2;
```

```
            System.out.println(num1 + "/" + num2 + " = " + num3);
```

```
        }
```

```
        catch(Exception ex)
```

```
        {
```

```
            System.out.println("錯誤類型：" + ex.toString());
```

```
        }
```

```
        finally
```

```
        {
```

```
            System.out.println("執行 finally 區塊\n");
```

捕捉例外

```
    }  
}  
  
public static void main(String[] args)  
{  
    Division(12,0);  
    Division(12,2);  
}  
}
```

捕捉例外

```
1 package CH08_01;
2
3 public class CH08_01
4 {
5     static void Division(int num1, int num2)
6     {
7         try
8         {
9             int num3 = num1 / num2;
10            System.out.println(num1 + " / " + num2 + " = " + num3);
11        }
12        catch (Exception ex)
13        {
14            System.out.println("錯誤類型：" + ex);
15        }
16        finally
17        {
18            System.out.println("執行 finally 區塊\n");
19        }
20    }
21
22    public static void main(String[] args)
23    {
24        Division(12, 0);
25        Division(12, 2);
26    }
27 }
```

捕捉例外

◆ 執行結果：

錯誤類型：[java.lang.ArithmeticException](#): / by zero
執行 finally 區塊

12 / 2 = 6
執行 finally 區塊

◆ 說明：

- 行01：
 - ◆ 定義「套件（package）」。
- 行05~行20：
 - ◆ 「Division(int num1, int num2)」方法為計算兩個整數相除，然後顯示其結果。
 - ◆ 在此方法中應用例外處理「try...catch...finally」技巧。

捕捉例外

- ◆ 行07~行11：
 - ◆ 「try」區塊。
 - ◆ 測試兩數相除，並顯示結果。
若發生「例外」，則拋出「例外」的類別，供「catch」捕捉「例外」。
- ◆ 行12~行15：
 - ◆ 捕捉「例外」，並顯示「例外」的類別。
- ◆ 行16~行19：
 - ◆ 無論有否執行「catch」，皆會執行「finally」內的敘述區塊。
- 行24：
 - ◆ 呼叫「Division(12, 0)」方法。
 - ◆ 因除數num2 = 0，所以第09行發生錯誤，跳到第12行執行「catch」，捕捉「例外」。
 - ◆ 本程式不會中斷執行。

捕捉例外

- 行25：
 - ◆ 呼叫「Division(12, 6)」方法，因此會顯示執行 $12 / 2 = 6$ 的結果。

自行拋出例外

■ 在Java中，雖然可以利用「try...catch...finally」來「捕捉」「例外情況」，但難免會出現利用「例外處理」仍然「抓不到」的錯誤。

例如：

- 將上例第05行中的num2及第09行中的num3更改成double形態，並將第28行更改為「Division(12, 0.0);」，則計算出來的商為「Infinity（無窮）」（也就是無限小數的意思），便會出現「捕捉不到」「例外」情況。

12 / 0.0 = Infinity
執行 finally 區塊

12 / 2.0 = 6.0
執行 finally 區塊

- ◆ 這是因為Java允許除數為浮點數0.0，所以不會產生錯誤。

自行拋出例外

- 另外，假設在「方法」中發生「IOException」或「ClassNotFoundException」等例外情況，但是「方法」中並沒有撰寫「try...catch...finally」區塊來捕捉這些「例外」時，則因為這些「例外」類別不屬於「RuntimeException類別」，因此「編譯程式」時就會發生錯誤。
- 上述兩種情況，就需要用到「自行拋出例外」。
- 「自行拋出例外」有下列兩種方式：
 - ◆ 在「程式敘述中」，使用「throw」關鍵字拋出「例外」。
 - ◆ 在「定義方法時」，使用「throws」關鍵字宣告該「方法」可能會拋出的「例外」。

自行拋出例外

使用「throw」關鍵字

◆ 使用時機：

- 在「程式敘述中」使用。
- 可「自訂」「錯誤訊息」。

◆ 語法：

throw new 例外物件(例外參數);

◆ 說明：

- throw
 - ◆ 關鍵字。
- new
 - ◆ 建立「例外」物件。

自行拋出例外

- 例外物件

- ◆ 為拋出的「例外」物件的「類別」。

- 例外參數

- ◆ 「自訂」的「錯誤訊息」。
- ◆ 使用「例外物件」的「getMessage()」方法時，所取得的「錯誤訊息」。
- ◆ 可以省略。

注意：

- ◆ 在程式中使用「throw」關鍵字拋出「例外」時，若有「例外參數」，則「例外參數」的「值（即錯誤訊息）」會覆蓋原本Java內建的「getMessage()」方法的「值（即錯誤訊息）」。

自行拋出例外

◆ 程式：

```
package CH08_02;
```

```
public class CH08_02
```

```
{
```

```
    static void Division(int num1, double num2)
```

```
    {
```

```
        try
```

```
        {
```

```
            if(num2==0)
```

```
                throw new ArithmeticException("除數為浮點數 0.0");
```

```
            double num3 = num1 / num2;
```

```
            System.out.println(num1 + "/" + num2 + " = " + num3);
```

```
        }
```

```
        catch(ArithmeticException ex)
```

```
        {
```

```
            System.out.println("錯誤類型：" + ex.getMessage());
```

```
        }
```

```
    finally
```

自行拋出例外

```
{  
    System.out.println("執行 finally 區塊\n");  
}
```

```
public static void main(String[] args)  
{  
    Division(12, 0.0);  
    Division(12, 2);  
}
```

自行拋出例外

```
1 package CH08_02;
2
3 public class CH08_02
4 {
5     static void Division(int num1, double num2)
6     {
7         try
8         {
9             if(num2==0)
10                 throw new ArithmeticException("除數為浮點數 0.0");
11             double num3 = num1 / num2;
12             System.out.println(num1 + "/" + num2 + " = " + num3);
13         }
14         catch(ArithmeticException ex)
15         {
16             System.out.println("錯誤類型：" + ex.getMessage());
17         }
18         finally
19         {
20             System.out.println("執行 finally 區塊\n");
21         }
22     }
23
24     public static void main(String[] args)
25     {
```

自行拋出例外

```
26     Division(12, 0.0);  
27     Division(12, 2);  
28 }  
29 }
```

◆ 執行結果：

錯誤類型：除數為浮點數 0.0
執行 finally 區塊

12 / 2.0 = 6.0
執行 finally 區塊

◆ 說明：

- 行01：
 - ◆ 定義「套件（package）」。
- 行05~行22：
 - ◆ 「Division(int num1, double num2)」方法為計算兩個相除，然後顯示其結果。
 - ◆ 在此方法中應用例外處理「try...catch...finally」技巧。

自行拋出例外

- ◆ 行07~行13：
 - ◆ 「try」區塊。
 - ◆ 測試兩數相除，並顯示結果。
 - ◆ 行09~行10：

判斷傳入的值，是否為「0.0」，若是，則拋出「自訂」的「例外」，並「自訂」「錯誤訊息」。
- ◆ 行14~行17：
 - ◆ 捕捉「例外」，並顯示「例外」的「錯誤訊息」。
- ◆ 行18~行21：
 - ◆ 無論有否執行「catch」，皆會執行「finally」內的敘述區塊。
- 行26：
 - ◆ 呼叫「Division(12, 0.0)」方法。
 - ◆ 因除數num2 = 0.0，所以第09行及第10行發生錯誤，跳到第14行執行「catch」，捕捉「例外」。

自行拋出例外

- ◆ 本程式不會中斷執行。
- 行27：
 - ◆ 呼叫「Division(12, 6)」方法，因此會顯示執行 $12 / 2 = 6$ 的結果。

自行拋出例外

使用「throws」關鍵字

◆ 使用時機：

- 在「定義方法時」時，讓呼叫該「方法」的程式來處理「例外」。
- 可以拋出「一個」或「多個」「例外」。
 - ◆ 如果要拋出「多個」「例外」，必須以「逗點（,）」隔開「每個」「例外」類別「名稱」。

◆ 語法：

[修飾子][static]傳回值資料型別 方法名稱([引數串列]) throws 例外類別 1,例外類別 2,...

```
{  
    敘述區段;  
    return 運算式;  
}
```

自行拋出例外

◆ 說明：

- throws
 - ◆ 關鍵字。
- 例外類別 1, 例外類別 2, ...
 - ◆ 有可能發生「例外」的類別。

自行拋出例外

◆ 程式：

```
package CH08_03;
```

```
import java.io.*;
```

```
public class CH08_03
```

```
{
```

```
    public static void main(String[] args) throws IOException
```

```
    {
```

```
        int score;
```

```
        String data;
```

```
        BufferedReader keyin;
```

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

```
        while(true)
```

```
        {
```

```
            System.out.print("輸入成績 0~100 (按「Enter」鍵離開) : ");
```

```
            data = keyin.readLine();
```

自行拋出例外

```
if(data.equals(""))
    break;

try
{
    score = Integer.parseInt(data);
    if (score < 0 || score > 100)
        throw new NumberFormatException();
}
catch(NumberFormatException ex)
{
    System.out.println("輸入錯誤");
    continue;
}
System.out.println("您所輸入的成績：" + score);
}
}
```

自行拋出例外

```
1 package CH08_03;
2
3 import java.io.*;
4
5 public class CH08_03
6 {
7     public static void main(String[] args) throws IOException
8     {
9         int score;
10        String data;
11
12        BufferedReader keyin;
13        keyin = new BufferedReader(new InputStreamReader(System.in));
14
15        while(true)
16        {
17            System.out.print("輸入成績 0~100 (按「Enter」鍵離開) : ");
18            data = keyin.readLine();
19
20            if(data.equals(""))
21                break;
22
23            try
24            {
25                score = Integer.parseInt(data);
26                if (score < 0 || score > 100)
```

自行拋出例外

```
27         throw new NumberFormatException();
28     }
29     catch (NumberFormatException ex)
30     {
31         System.out.println("輸入錯誤");
32         continue;
33     }
34     System.out.println("您所輸入的成績：" + score);
35 }
36 }
37 }
```

◆ 執行結果：

```
輸入成績 0~100 (按「Enter」鍵離開) : 80
您所輸入的成績：80
輸入成績 0~100 (按「Enter」鍵離開) : 0.3
輸入錯誤
輸入成績 0~100 (按「Enter」鍵離開) :
```

◆ 說明：

- 行01：
 - ◆ 定義「套件 (package)」。

自行拋出例外

- 行03：
 - ◆ 載入「java.io.*」套件。
- 行07：
 - ◆ main()方法後面，加上 throws IOException。
 - ◆ 若未加throws IOException，在編譯時會發生錯誤。
- 行09：
 - ◆ 宣告整數變數。
- 行10：
 - ◆ 宣告字串變數。
- 行12：
 - ◆ 宣告BufferedReader類別的物件。
- 行13：
 - ◆ 建立keyin物件。

自行拋出例外

- 行15~行35：
 - ◆ 判斷輸入的數字是否是整數，且在指定的範圍內；若不是，顯示錯誤，但程式不會中斷。
 - ◆ 行18：
 - ◆ 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data。
 - ◆ 行20~行21：
 - ◆ 判斷是否沒有輸入資料（即按「Enter」鍵），若是，則離開while迴圈。
 - ◆ 行23~行28：
 - ◆ 「try」區塊。
 - ◆ 行25：
 - 透過Integer.parseInt的方法，轉換成整數後，指派給整數變數score。

自行拋出例外

- ◆ 行27：
 - ◆ 若輸入的值，不在指定範圍中，則拋出「例外」物件。
- ◆ 行29~行33：
 - ◆ 「catch」區塊。
 - ◆ 捕捉「NumberFormatException」 「例外」。
 - ◆ 行31：
 - 顯示錯誤。
- ◆ 行34：
 - ◆ 顯示輸入的成績。

自定例外類別

- 如果在程式除錯時，找不到適當Java內建的「例外」「類別」可用，則可以「自行定義」需要的「例外」「類別」，然後在
 - 一. 「程式敘述中」使用「throw」拋出「自定」「例外」類別的「例外」物件
 - 或
 - 二. 在「方法成員」名稱後面加「throws」自定例外類別，讓呼叫該「方法」的程式來處理「例外」
- ，以協助程式的除錯工作。

自定例外類別

■ 因為Java的「例外」類別主要繼承自「Exception」「基礎類別」，所以自訂「例外」類別時，統一都是繼承於「Exception」「基礎類別」或其「衍生類別」。

◆ 語法：

class 使用者自行設計的例外名稱 extends Exception
(或Exception類別的子類別)

{

類別程式敘述；

}

自定例外類別

◆ 程式：

```
package CH08_04;
```

```
import java.io.*;
```

```
class CCheckNumberException extends NumberFormatException
{
    CCheckNumberException(String msg)
    {
        super(msg);
    }
}
```

```
public class CH08_04
{
    static void CheckScore(int num)
    {
        if (num < 0 || num > 100)
            throw new CCheckNumberException("輸入成績非 0~100");
    }
}
```

自定例外類別

```
}
```

```
public static void main(String[] args) throws IOException
{
    int score;
    String data;

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

    while (true)
    {
        System.out.print("輸入成績 0~100 (按「Enter」鍵離開) : ");
        data = keyin.readLine();

        if (data.equals(""))
            break;

        try
```

自定例外類別

```
{
    score = Integer.parseInt(data);
    CheckScore(score);
}
catch (CCheckNumberException ex)
{
    System.out.println("錯誤說明：" + ex.getMessage());
    continue;
}
catch (NumberFormatException ex)
{
    System.out.println("錯誤說明：輸入非整數");
    continue;
}
System.out.println("您所輸入的成績：" + score);
}
}
```

自定例外類別

```
1 package CH08_04;
2
3 import java.io.*;
4
5 class CCheckNumberException extends NumberFormatException
6 {
7     CCheckNumberException(String msg)
8     {
9         super(msg);
10    }
11 }
12
13 public class CH08_04
14 {
15     static void CheckScore(int num)
16     {
17         if (num < 0 || num > 100)
18             throw new CCheckNumberException("輸入成績非 0~100");
19     }
20
21     public static void main(String[] args) throws IOException
22     {
23         int score;
24         String data;
25
26         BufferedReader keyin;
```


自定例外類別

```
27     keyin = new BufferedReader(new InputStreamReader(System.in));
28
29     while (true)
30     {
31         System.out.print("輸入成績 0~100 (按「Enter」鍵離開) : ");
32         data = keyin.readLine();
33
34         if (data.equals(""))
35             break;
36
37         try
38         {
39             score = Integer.parseInt(data);
40             CheckScore(score);
41         }
42         catch (CCheckNumberException ex)
43         {
44             System.out.println("錯誤說明：" + ex.getMessage());
45             continue;
46         }
47         catch (NumberFormatException ex)
48         {
49             System.out.println("錯誤說明：輸入非整數");
50             continue;
51         }
52         System.out.println("您所輸入的成績：" + score);
```

自定例外類別

```
53 }  
54 }  
55 }
```

◆ 執行結果：

```
輸入成績 0~100 (按「Enter」鍵離開) : 102  
錯誤說明：輸入成績非 0~100  
輸入成績 0~100 (按「Enter」鍵離開) : 0.3  
錯誤說明：輸入非整數  
輸入成績 0~100 (按「Enter」鍵離開) : 80  
您所輸入的成績：80  
輸入成績 0~100 (按「Enter」鍵離開) :
```

◆ 說明：

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

自定例外類別

- 行05~行11：
 - ◆ 建立繼承自「NumberFormatException」父類別的「CCheckNumberException」子類別。
 - ◆ 行07~行10：
 - ◆ 宣告類別「建構子」
「CCheckNumberException(String msg)」。
 - ◆ 行09：
利用「super(msg)」，呼叫「父類別」中，「有一個引數」的「建構子」。

註：

因為「CCheckNumberException」的最上層類別是「Throwable」，所以「super(msg)」會一直傳到「Throwable」的「Throwable(String message)」建構子中，再將傳入的「msg」存入「私有」的「資料成員」
「detailMessage」，以供「getMessage()」方法讀取。

自定例外類別

- 行15~行19：
 - ◆ 建立「類別」方法「CheckScore(int num)」，用來檢查輸入的數字是否在指定範圍內；若沒有在指定範圍內，則拋出「例外」。
 - ◆ 行18：
拋出「例外」物件，並自訂「例外訊息」給「CCheckNumberException」的類別。
- 行21：
 - ◆ main()方法後面，加上 throws IOException。
 - ◆ 若未加throws IOException，在編譯時會發生錯誤。
- 行23：
 - ◆ 宣告整數變數。
- 行24：
 - ◆ 宣告字串變數。

自定例外類別

- 行26：
 - ◆ 宣告BufferedReader類別的物件。
- 行27：
 - ◆ 建立keyin物件。
- 行29~行53：
 - ◆ 判斷輸入的數字是否是整數，且在指定的範圍內；若不是，顯示錯誤，但程式不會中斷。
 - ◆ 行32：
 - ◆ 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data。
 - ◆ 行34~行35：
 - ◆ 判斷是否沒有輸入資料（即按「Enter」鍵），若是，則離開while迴圈。
 - ◆ 行37~行41：
 - ◆ 「try」區塊。


自定例外類別

- ◆ 行39：
透過Integer.parseInt的方法，轉換成整數後，指派給整數變數score。
- ◆ 行40：
執行「類別」方法「CheckScore(int num)」。
- ◆ 行42~行46：
 - ◆ 「catch」區塊。
 - ◆ 捕捉「CCheckNumberException」例外。
 - ◆ 行44：
顯示錯誤。
- ◆ 行47~行51：
 - ◆ 「catch」區塊。
 - ◆ 捕捉「NumberFormatException」例外。
 - ◆ 行49：
顯示錯誤。

自定例外類別

- ◆ 行52：
 - ◆ 顯示輸入的成績。

巢狀例外

 「例外處理」與迴圈一樣，可以有「好多層」，稱為「巢狀例外」。

- ◆ 「外層」的「try」區塊中又包含內層的「try...catch...finally」敘述，而「每一層」的「throw」敘述會執行對應該層的「catch」區塊（如右）。

```
try
{
    程式敘述;
    try
    {
        程式敘述;
    }
    catch()
    {
        程式敘述;
    }
    finally
    {
        程式敘述;
    }
}
catch()
{
    程式敘述;
}
finally
{
    程式敘述;
}
```


巢狀例外

◆ 程式：

```
package CH08_05;

import java.io.*;

public class CH08_05
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader keyin;
        keyin = new BufferedReader(new InputStreamReader(System.in));

        double base, exponent;

        System.out.print("請輸入底數：");
        base = Double.parseDouble(keyin.readLine());

        System.out.print("請輸入指數：");
        exponent = Double.parseDouble(keyin.readLine());
    }
}
```

巢狀例外

```
try
{
    if (base == 0 && exponent < 0)
        throw new ArithmeticException();

    if (base < 0)
    try
    {
        if (exponent > 0 && exponent < 1)
            throw new ArithmeticException();
    }
    catch (ArithmeticException ex)
    {
        System.out.println("底數小於 0，則指數不可介於 0~1 之間，無法  
計算!");

        return;
    }
    System.out.println("結果：" + Math.pow(base, exponent));
}
```

巢狀例外

```
    catch (ArithmeticException ex)
    {
        System.out.println("底數等於 0，則指數不可小於 0，無法計算！");
    }
}
```

巢狀例外

```
1 package CH08_05;
2
3 import java.io.*;
4
5 public class CH08_05
6 {
7     public static void main(String[] args) throws IOException
8     {
9         BufferedReader keyin;
10        keyin = new BufferedReader(new InputStreamReader(System.in));
11
12        double base, exponent;
13
14        System.out.print("請輸入底數：");
15        base = Double.parseDouble(keyin.readLine());
16
17        System.out.print("請輸入指數：");
18        exponent = Double.parseDouble(keyin.readLine());
19
20        try
21        {
22            if (base == 0 && exponent < 0)
23                throw new ArithmeticException();
24
25            if (base < 0)
26                try
```

巢狀例外

```
27     {
28         if (exponent > 0 && exponent < 1)
29             throw new ArithmeticException();
30     }
31     catch (ArithmeticException ex)
32     {
33         System.out.println("底數小於 0，則指數不可介於 0~1 之間，無法計算！");
34         return;
35     }
36     System.out.println("結果：" + Math.pow(base, exponent));
37 }
38 catch (ArithmeticException ex)
39 {
40     System.out.println("底數等於 0，則指數不可小於 0，無法計算！");
41 }
42 }
43 }
```

巢狀例外

◆ 執行結果：

```
請輸入底數：0  
請輸入指數：-1  
底數等於0，則指數不可小於0，無法計算！
```

```
請輸入底數：-1  
請輸入指數：0.5  
底數小於0，則指數不可介於0~1 之間，無法計算！
```

```
請輸入底數：2  
請輸入指數：3  
結果：8.0
```

◆ 說明：

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

巢狀例外

- 行07：
 - ◆ `main()`方法後面，加上 `throws IOException`。
 - ◆ 若未加`throws IOException`，在編譯時會發生錯誤。
- 行09：
 - ◆ 宣告`BufferedReader`類別的物件。
- 行10：
 - ◆ 建立`keyin`物件。
- 行12：
 - ◆ 宣告浮點數變數。
- 行15：
 - ◆ 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`base`。

巢狀例外

- 行18：
 - ◆ 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exponent。
- 行20~行37：
 - ◆ 外層「try」區塊。
 - ◆ 測試「底（base）」是否等於0，且「指數（exponent）」是否小於0。
 - ◆ 行23~行24：
 - ◆ 如果「底」是等於0，且「指數」是小於0，則拋出例外。
 - ◆ 行25：
 - ◆ 如果「底」是小於0，則
 - ◆ 行26~行30：
 - 內層「try」區塊。

巢狀例外

行28~行29：

如果「指數」是大於0，且「指數」是小於1，則拋出例外。

◆ 行31~行35：

內層「catch」區塊。

• 行38~行41：

◆ 外層「catch」區塊。

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。