第九章　例外處理與 Assertion

本章內容

- 9-1　例外的種類
- 9-2　使用「try-catch」敘述
- 9-3　使用「throws」敘述
- 9-4　自定例外
- 9-5　Assertion

9-1 例外的種類

- 程式可能產生的例外情況有:
  - 在編譯時期產生的錯誤一般稱為「**語法錯誤 (Syntax Error)**」

```
float i = 75.0;          // 必需是「 float i = 75.0F; 」
```

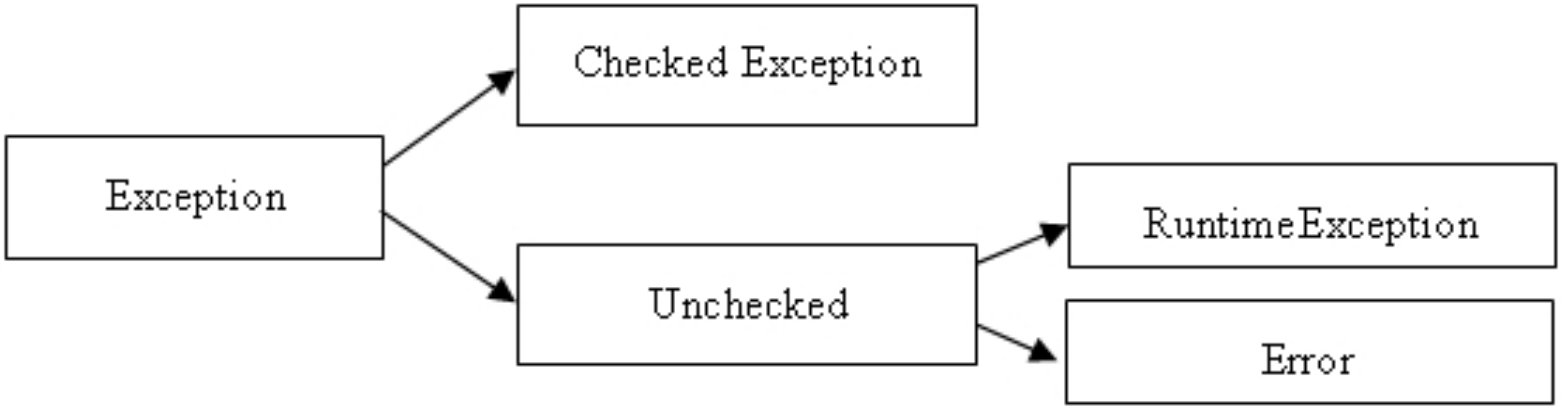
  - 執行的時候產生錯誤，這種錯誤一般稱為「**執行時期的錯誤 (Runtime Error)**」

```
int i, j, result;
i = 10;
j = 0;
result = i/j;
```

- Java 程式語言對程式的錯誤處理的機制，我們稱為「**例外處理機制 (Exception-handling Mechanism)**」。

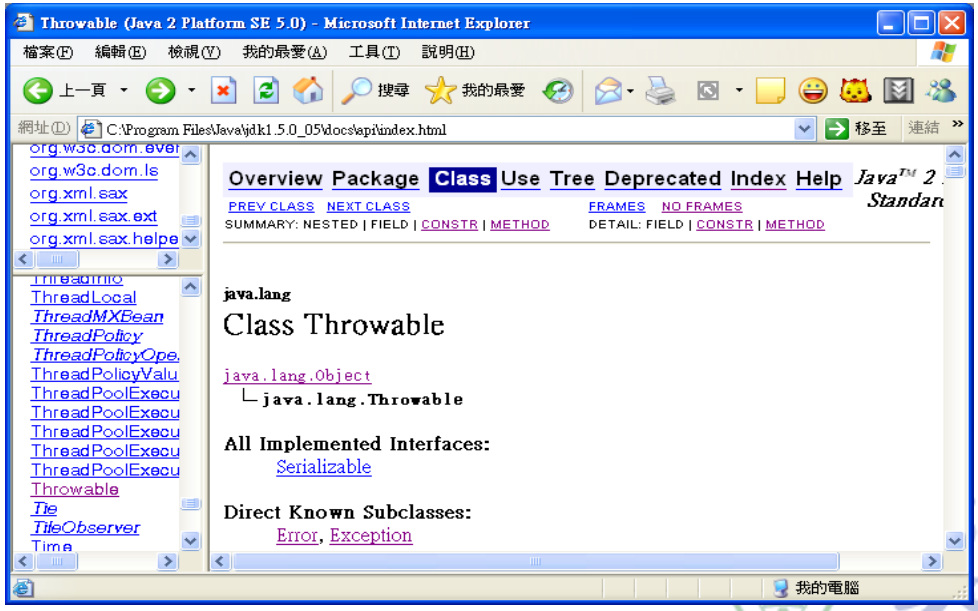
9-1-1 Java 的例外 (Exception) 與錯誤 (Error)

- 從程式設計者的角度而言，例外可以區分為兩大類:「**Checked Exception**」和「**Unchecked Exception**」，而「Unchecked Exception」又可以再區分為:「**RuntimeException**」和「**Error**」兩種情況，參考下圖:



9-1-1 Java 的例外 (Exception) 與錯誤 (Error)

- 在 Java 中，「Error」或是「Exception」類別，它們都是衍生自「Throwable」類別 。您可以參考 SDK 文件中的說明



9-1-1 Java 的例外 (Exception) 與錯誤 (Error)

- 在「Exception」的分類中，「Checked Exception」是屬於即使程式設計正確，也有可能會產生的例外種類。
- 在 Java 語言中，對於這一類型的 Exception，程式設計者必需要自行處理，否則，程式無法通過編譯。例如：

例外名稱	產生原因
ClassNotFoundException	找不到指定的類別，可能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
CloneNotSupportException	在類別中使用 clone() 方法，但該類別未實作「Cloneable」介面
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
IOException	檔案、網路 ... 的輸出、入錯誤時產生的例外

9-1-1 Java 的例外 (Exception) 與錯誤 (Error)

- 而「Unchecked Exception」通常不會在編譯時產生錯誤的訊息。Java 並不强迫設計者一定要處理這一類型的 Exception。 常見的有:

例外名稱	產生原因
ArithmeticException	運算式產生的例外，例如：除數為 0
ArrayIndexOutOfBoundsException	陣列的索引值指定錯誤，例如：超出索引值
ArrayStoreException	指定陣列內容錯誤時產生的錯誤
ClassCastException	類別轉型錯誤
IndexOutOfBoundsException	索引使用時超出範圍，這個類別是 ArrayIndexOutOfBoundException 的父類別
IllegalArgumentException	呼叫方法時，傳遞錯誤的參數
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
SecurityException	企圖違反安全性的限制

9-1-1 Java 的例外 (Exception) 與錯誤 (Error)

- 對於 Java 而言，每個 Exception 都是一個物件。也就是說，當程式發生 Exception 時，就會產生某個特定的 Exception 物件。如果程式在執行時真的產生了異常情況，JVM 會依照一定的程序來處理異常情況，過程如下：

  一、中止產生異常情況的指令的執行。

  二、產生此異常情況的 Exception 物件。

  三、如果設計者有撰寫處理該 Exception 的程式碼，則交由該區段的程式碼來處理；如果沒有設計處理的程式碼，則交由 JVM 處理。

  四、如果是由 JVM 來處理 Exception 物件，則程式會異常終止。

9-2 使用「try-catch」敘述

- 「try-catch」是 Java 程式的例外處理敘述，該語法中的處理如下：

```
try {
        // 可能產生例外的敘述
} catch (ExceptionType e1) {
        // 產生 ExceptionType 例外時，要處理的敘述
} catch (Exception e2) {
        // 產生其他的例外時，要處理的敘述
}
```

- 在語法中，「try」區段中包含了可能產生例外的程式碼。「catch」敘述後以「( )」表示需要處理的例外的類別，e1 或是 e2 則是產生的例外物件。

9-2 使用「try-catch」敘述…

- finally 區塊
  - 「finally」區段是一個不論例外是否發生，它都一定會執行的一個區段。 參考以下的程式碼:

```
try{
        result = i / j;
    } catch (ArithmeticException e){
            System.out.println(" 除數不得為零 ");
    } catch (Exception e){
            System.out.println(" 其餘的錯誤 ");
    } finally {
        System.out.println(" 執行 finally 區段 ");
    }
}
```

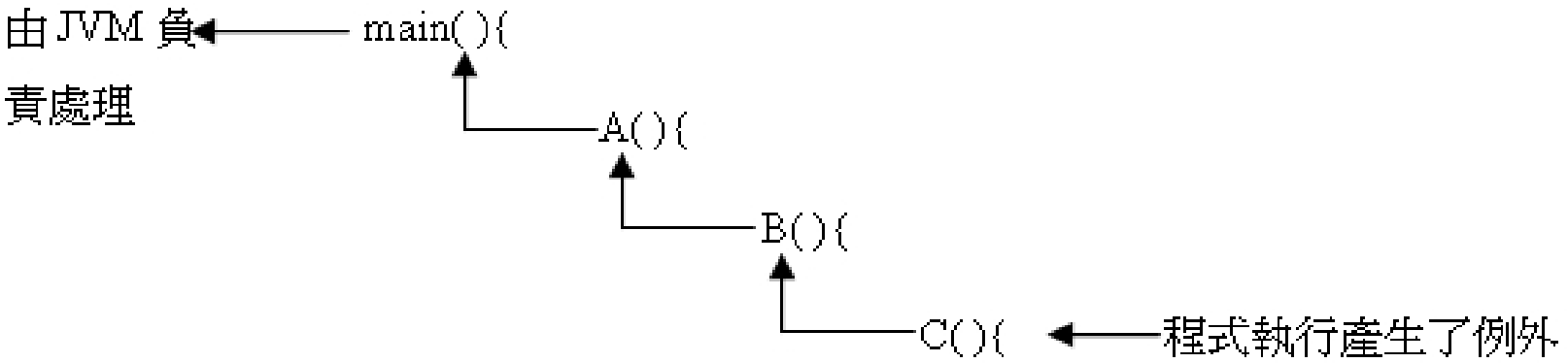
9-3 使用「throws」敘述

- Java 語言要求設計師必需自行處理可能產生 Checked Exception 的程式碼。
- 但其實，當例外產生時，您也可以不要使用「try-catch」機制立即處理它。可以取代的作法是：將它「丟 (throws)」出來給另一支程式處理。
  - 如果要使用這種方法，您必需使用「throws」關鍵字。例如以下的程式碼：

```
public static double calculate(int i, int j) throws ArithmeticException{
        return i/j;
}
```

9-3 使用「throws」敘述…

- Call Stack 機制
  - 在宣告方法時，如果使用了「throws」關鍵字將可能產生的例外丟出，那麼呼叫此方法的類別就必需使用「try-catch」來捕捉例外。這種將例外一直往上層丟出，直到該例外被處理為止的機制就稱為「**Call Stack**」機制。參考下圖:



9-4 自定例外

- 雖然 Java 語言已經提供了相當多的例外類別，但有時候，為了設計出更 User Friendly 的程式，我們可能需要自已設計例外處理的機制。
  - 在自定例外時，常會使用到「 throw 」關鍵字
- throw 關鍵字可以用來丟出例外物件，它可以丟出 Java 內部的例外物件，也可以丟出自定的例外物件， throw 的語法如下：

```
throw  例外物件 ;                                    // 丟出 Java 例外物件
throw new Exception(" 自定的例外訊息 ");     // 丟出匿名的例外物件
```

9-4 自定例外…

- 自定的例外類別常會衍生自 Exception 類別，該類別的建構子有：

建構子	作用
Exception()	建立一個 Exception 物件
Exception(String message)	建立一個 Exception 物件，並指定例外訊息為 message
Exception (String message, Throwable cause)	建立一個 Exception 物件，並指定例外訊息為 message，同時將產生例外的原因 cause 傳入
Exception(Throwable cause)	建立一個 Exception 物件，同時將產生例外的原因 cause 傳入

9-4-2 自定例外類別

- 如果您想要設定自己的例外類別來處理特殊的情況，您只要繼承 Exception 類別或是 Exception 的子類別就可以了。例如：以下的繼承都是合法的：

```
class MyException1 extends Exception {}    // 繼承自 Exception 類別
// 繼承自 RuntimeException 類別
class MyException2 extends RuntimeException
```

- 繼承而來的子類別能夠產生的例外物件是和父類別相關連的。子類別可以丟出和父類別相同的 Exception，或者是父類別的子類別。

9-4-2 自定例外類別…

- Exception 類別並沒有自己的方法，它的方法都是由 Throwable 類別中繼承來的。該類別中常用的方法有：

方法	作用
Throwable fillInStackTrace()	傳回包含完整堆疊追蹤的 Thorwable 物件
Throwable getCause()	傳回造成例外原因的 Throwable 物件
String getLocalizedMessage()	傳回本地語系的例外說明，使用此方法傳回的例外說明和 getMessage() 相同
String getMessage()	傳回例外說明
StackTraceElement[]getStackTrace()	傳回包含堆疊追蹤的陣列
Throwable initCause(Throwable cause)	將 cause 當作是例外發生的原因
void printStackTrace()	顯示追蹤堆疊
void printStackTrace(PrintStream s)	將追蹤堆疊顯示在串流物件之中
void printStackTrace(PrintWriter s)	將追蹤堆疊顯示在串流物件之中
String toString()	傳回簡短的例外描述字串

9-4-3 巢狀的 try 敘述

- Java 語言中是使用「try-catch」機制來捕捉例外事件。我們會在「try」區段中放入可能產生例外的敘述。
- 但是程式在執行其他的區段中也可能產生例外而需要使用另一組的「try-catch」再來捕捉例外物件。
- 不論另一組的「try-catch」是放在那一個區段中，程式都是形成巢狀的「try-catch」敘述。
  - 您必需要能判斷任何「try-catch」區段中的每行程式碼是否會被執行到。

9-5 Assertion

- Java 1.4 版中加入了 Assertion 機制，其目的是在確保程式執行的正確性。
- Assertion 常被用來檢查關鍵值。
  - 所謂的關鍵值是指當變數值錯誤時，會對程式造成重大的影響，甚至會造成程式無法執行。
- Assertion 所要檢查的關鍵值著重在測試或開發時期的檢查，一般而言，這些關鍵值在程式發佈後是不容許有錯誤產生的。

9-5-1 使用 assert 關鍵字

- assert 關鍵字是用於 Assertion 機制中，使用的基本語法如下：

```
assert bool_expr;
```

- 「bool_expr」代表需要檢查的邏輯判斷敘述句。bool_expr 可以是一個比較或是邏輯運算式，也可以是一個方法的執行結果。
- 當 bool_expr 的判斷結果為 false 時，則整行敘述會產生 AssertionError。例如以下的敘述：

```
assert x > 2;
```

9-5-1 使用 assert 關鍵字

- assert 關鍵字的另一個語法為:

```
assert bool_expr : detail_expr;
```

- detail_expr 可以是一個數值、字串、變數、或是一個物件。當 bool_expr 的判斷值為 false 時，程式會執行 detail_expr 敘述句。我們時常會將 detail_expr 寫成一個字串來表明錯誤的原因。 例如

```
assert income>=0 : " 薪資不可以為負值 ";
```

9-5-1 使用 assert 關鍵字

- 編譯具有 assert 語法的程式
  - 如果編譯器的版本是在 1.4 之前，編譯時，會產生 warning 的訊息 。您要用以下的語法編譯:

```
javac –source 1.4 Assertion1.java
```

  - Assertion 機制可以在執行時選擇開啓或是關閉，預設情況下， Assertion 機制是關閉的。執行時，要開啓 Assertion 機制的語法為:

```
java –ea Assertion1
```

```
java -enableassertions  Assertion1
```

9-5-2 Assertion 的使用建議

- Assertion 可以用在幾個方面:
  - private 方法的參數檢查:
    - Assertion 常被用於「前置狀態 (preconditions) 」的檢查。
  - 內部執行狀況判斷 (Internal Invariants) :
  - 控制流程的判斷 (Control Flow Invariants) :
  - 後置狀態的控管 (Postcondition Invariants) :
- 在 public 函式中不適合使用 Assertion 做「前置狀態」的檢查
  - 但可以利用 Assertion 做「後置狀態」的檢查