

Module II - I

例外處理

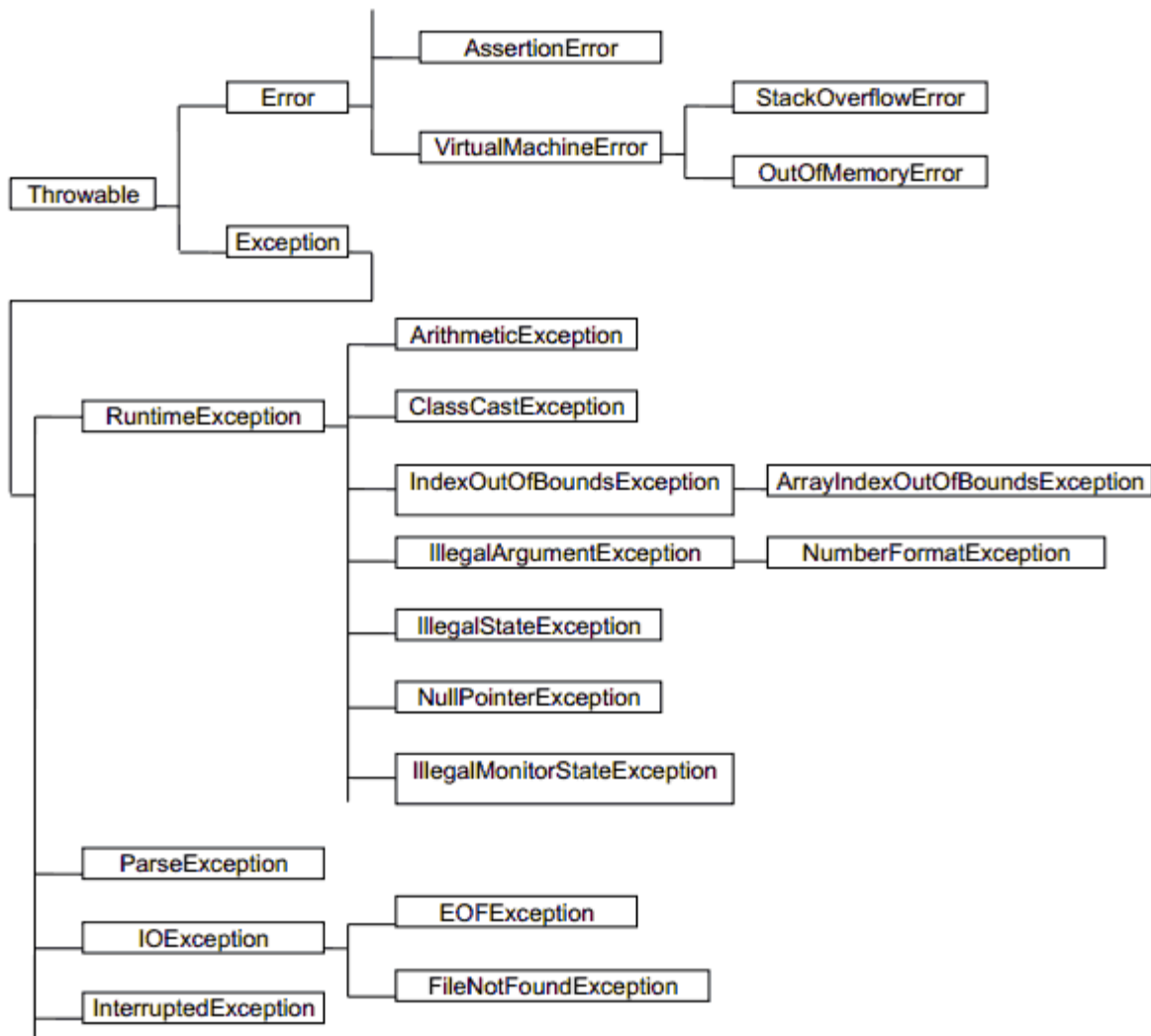
1. Exception物件
2. 例外的類別層級
3. 例外處理 (try – catch – finally)
 4. throws
 5. throw
 6. 自訂例外
7. 例外與覆寫(override)
8. 例外處理流程
9. Assertions

Exception物件

- Java程式執行時，如果發生異常狀況，可借助例外處理
 - 程式發生(產生)例外時，視為產生了一個**Exception物件**
- Java的例外處理可將一般**正常處理程序**與**錯誤處理程序**做到**分開敘述**，讓程式可讀性提高
 - 5個關鍵字：**try**、**catch**、**finally**、**throws**與**throw**
 - 在沒有支援例外處理的程式語言裡，錯誤必須自行檢查，然後再手動處理，可能要用到很多**if...else**，非常麻煩
- 適合使用的時機：如資料庫連結失敗、找不到檔案、除以0、參數為空值、參數型態不符、陣列索引超出範圍等...

例外的類別層級架構

Throwable的階層架構圖



try - catch

- Java的例外處理：

try {

正常處理程序的程式碼...;

} **catch** (MyException e) {錯誤處理程序的程式碼...;}

catch (Exception e) {錯誤處理程序的程式碼...;}

- try { } :

- 將**正常處理程序**的程式碼置於 try { } 的程式區塊中

- catch (MyException e) { } :

- 當捕捉到例外物件時
- 將**錯誤處理程序**的程式碼置於 catch { } 的程式區塊中

- try { } 程式區塊後應緊接著 catch { } 程式區塊

- try { } 程式區塊後可接多個 catch { } 程式區塊，至類別位階**高者**需置於類別位階低者的**後面**

try – catch - finally

- 將**一定要執行的程式碼**放在 **finally { }** 的程式區塊裡
- **finally { }** 是無論發生什麼情況**皆會執行**的程式區塊，可用來釋放有限的資源，例如關閉(close)資料庫連線、檔案讀取等
- **finally { }** 置於**所有catch { }**的後面

```
try {  
    正常處理程序程式碼...;  
} catch (MyException e) {  
    錯誤處理程序程式碼...;  
} catch (Exception e) {  
    錯誤處理程序程式碼...;  
} finally {  
    一定要執行的程式碼...;  
}
```

throws關鍵字 (1/4)

- Java有兩種不同型態的例外(Exceptions)
 - 執行時期的例外
 - 這一類的例外，不一定要處理
 - 也稱為不必檢查的例外 (**unchecked exceptions**)
 - 譬如 **Runtime Exception** 及其子類別
 - 非執行時期的例外
 - 這一類的例外，一定要處理
 - 也稱為必須檢查的例外 (**checked exceptions**)
 - 譬如 **IOException**、**SQLException**...等

throws 關鍵字 (2/4)

- 在方法定義時，可使用 **throws** 關鍵字將可能發生的例外，丟出給呼叫此方法的程式去處理，用法如下：
 - `void method() throws MyException {...}`
 - `public static int parseInt(String s) throws NumberFormatException {...}`
 - `public int read() throws IOException {...}`
 - `public static Connection getConnection() throws SQLException {...}`
- 對於 **checked exceptions**，在呼叫有 **throws** 關鍵字的方法時，**必須** 將該方法置於下列兩者之一：
 - 將該方法置於 **try { }** 程式區塊中
 - (或) 將該方法置於定義有 **throws** 關鍵字的方法中
 - 此為再透過 **throws** 丟出例外，然後再由下一個呼叫者來處理

throws關鍵字 (3/4)

```
1 public class Test3_throws {
2     String[] strs = {"Hello1", "Hello2", "Hello3"};
3
4     public void printStrs(int i) throws Exception {
5         System.out.println(strs[i]);
6     }
7
8     public static void main(String[] args) {
9         int i = 0;
10        Test3_throws t3 = new Test3_throws();
11        while(i < 4) {
12            try {
13                t3.printStrs(i);
14            } catch (ArrayIndexOutOfBoundsException e) {
15                System.out.println("1-已超出陣列的長度");
16            } catch (Exception e) {
17                System.out.println("2-發生Exception");
18            }
19            i++;
20        }
21    }
22 }
```

將該方法置於 try {}
程式區塊中

throws關鍵字 (4/4)

```
1 public class Test4_throws {
2     String[] strs = {"Hello1", "Hello2", "Hello3"};
3
4     public void printStrs(int i) throws Exception {
5         System.out.println(strs[i]);
6     }
7
8     public static void main(String[] args) throws Exception {
9         int i = 0;
10        Test4_throws t4 = new Test4_throws();
11        while(i < 4) {
12            t4.printStrs(i);
13            i++;
14        }
15    }
16 }
```

將該方法置於定義有
Throws關鍵字的方法中

throw 關鍵字 (1/2)


- 可使用 throw 關鍵字，將方法內的例外手動丟出
- throw的指令格式：
 - throw 「一個可被丟出的物件」
 - 該物件必須是 java.lang.**Throwable** 類別的子類別

throw 關鍵字 (2/2)


範例：
TestThrowDemo.java

```
1 public class Test5_ThrowDemo {
2     public static double method(double i, double j) throws ArithmeticException {
3         double result;
4         if (j == 0) {
5             throw new ArithmeticException("喂！ 除到0 ！ 算數錯誤!");
6         }
7         result = i / j;
8         return result;
9     }
10
11     public static void main(String[] args) {
12         try {
13             System.out.println(method(1, 0));
14         } catch (ArithmeticException e) {
15             System.out.println(e.getMessage());
16             //或
17             e.printStackTrace();
18         }
19     }
20 }
```

喂！ 除到0 ！ 算數錯誤！



java.lang.ArithmeticException: 喂！ 除到0 ！ 算數錯誤！
at Test5_ThrowDemo.method(Test5_ThrowDemo.java:5)
at Test5_ThrowDemo.main(Test5_ThrowDemo.java:13)



自訂例外

- 要自訂例外類別時：
 - 必須繼承 `Throwable` 或 `Exception` 或 `RuntimeException` 之一
 - 自訂的例外類別，通常會包含兩個建構子：
 - `public 建構子名稱 () { }`
 - `public 建構子名稱 (String message) {
 super(message);
}`
- 程式內一樣可利用 **throw** 關鍵字，將例外拋給負責處理此例外的 `catch { }` 區塊處理

課堂練習

- 請建立一個正立方體Cube.java檔案，並定義邊長屬性(double length)，建構子(Constructor)與getter/setter方法
- 產生一個cube物件並同時傳入邊長值，若是值為0或負數，則拋出自行定義的例外CubeException，並顯示「正立方體邊長不得為0或是負數」的訊息
- 若是傳入邊長的值沒有問題，則顯示體積

Exception 與 Override

- 子類別覆寫其父類別定義有 **throws** 的方法時，子類別所 throws 的 Exception 必須與父類別被覆寫方法的 Exception 一樣或是更低階

```
1. public class BaseClass {  
    public void method() throws IOException { }  
2. public class OK_A extends BaseClass {  
    public void method() throws IOException { }  
3. public class OK_B extends BaseClass {  
    public void method() { }  
4. public class NG_C extends BaseClass {  
    public void method() throws Exception { }    // Error!
```

例外處理的流程 (1/2)

```
public class ExceptionFlow {  
    public static void main(String[] args) {  
        try {  
            method();  
            System.out.println("output 0");  
        } catch (Exception1 e1) {  
            System.out.println("output 1");  
        } catch (Exception2 e2) {  
            System.out.println("output 2");  
        } finally {  
            System.out.println("output 3");  
        }  
        System.out.println("output 4");  
    }  
}
```

例外處理的流程 (2/2)

- 若method()執行成功，未發生任何例外錯誤，則程式輸出結果為 output 0, output 3, output 4
- 若method()發生 **Exception1** 的例外錯誤，則程式輸出結果為 output 1, output 3, output 4
- 若method()發生 **Exception1, 2 以外**的例外錯誤，則程式輸出結果只有 output 3

例外類型多重捕捉

- 以往catch區塊只能處理一個例外，Java 7開始一個catch區塊可以處理一個以上的例外類型，精簡程式碼
- **注意：catch括號內的例外類型不可以有繼承關係**

```
28 // before, 一個catch區塊只能處理一個例外類型
29 try {
30     methodA("A");
31     methodB("B");
32 } catch (ExceptionB e) {
33     e.printStackTrace();
34 } catch (ExceptionA e) {
35     e.printStackTrace();
36 }
```

```
38 // Java 7, 一個catch區塊可以處理一個以上的例外類型，可以精簡程式碼
39 try {
40     methodA("A");
41     methodB("B");
42 } catch (ExceptionA | ExceptionB e) {
43     e.printStackTrace();
44 }
```

改良重新拋出例外的類型檢查

- 編譯器可以更精確地分析需重新拋出的例外類型，在方法宣告的 throws 子句中可指定更多明確的例外型別

```
5    static class ExceptionA extends Exception { }
6    static class ExceptionB extends Exception { }
7    // before
8    public static void methodA(String exceptionName) throws Exception {
9        try {
10            if (exceptionName.equals("A")) {
11                throw new ExceptionA();
12            } else {
13                throw new ExceptionB();
14            }
15        } catch (Exception e) {
16            throw e;
17        }
18    }
19
20    // Java 7
21    public static void methodB(String exceptionName) throws ExceptionA, ExceptionB {
22        try {
23            if (exceptionName.equals("A")) {
24                throw new ExceptionA();
25            } else {
26                throw new ExceptionB();
27            }
28        } catch (Exception e) {
29            throw e;
```

try – with – resources

- 以往程式設計師需自行處理資源關閉，Java 7開始try – with – resources 可確保物件(資源)最後一定會關閉
- 實作**java.lang.AutoCloseable**與**java.io.Closeable**介面的物件皆可視為資源

```
14 // before, finally內自行處理資源的關閉
15 try {
16     br = new BufferedReader(new FileReader(file));
17     String bookInfo = "";
18     while ((bookInfo = br.readLine()) != null) {
19         System.out.println(bookInfo);
20     }
21 } catch (IOException ex) {
22     ex.printStackTrace();
23 } finally {
24     try {
25         br.close();
26     } catch (IOException ex) {
27         ex.printStackTrace();
28     }
29 }
30
31 // Java 7, try-with-resources可確保物件(資源)在最後都會被關閉
32 try (BufferedReader in = new BufferedReader(new FileReader(file));) {
33     String bookInfo = "";
34     while ((bookInfo = in.readLine()) != null) {
35         System.out.println(bookInfo);
36     }
37 } catch (IOException ex) {
38     ex.printStackTrace();
39 }
```

常見的 Runtime Exception (補充1)

例外(Exception)的類別名稱	例外狀況說明
ArithmeticException	分母為 0 時
NullPointerException	呼叫到一個空值 null 變數
IllegalArgumentException NumberFormatException	傳入的參數型態不符 傳入的數字型態不符(前者的子類別)
IndexOutOfBoundsException ArrayIndexOutOfBoundsException	索引值超過物件上限 索引值超過陣列上限(前者的子類別)
ClassCastException	類別的轉型(Casting)失敗
SecurityException	違反安全原則

取得錯誤訊息的方法 (補充2)

- 取得錯誤訊息的方法 (Throwable 類別所定義)

取得錯誤訊息的方法		說明
String	getMessage()	Returns the detail message string of this throwable
void	printStackTrace()	Prints this throwable and its backtrace to the standard error stream
void	printStackTrace (PrintStream s)	Prints this throwable and its backtrace to the specified print stream 至指定的輸出設備
String	toString()	Returns a short description of this throwable 簡短描述

Assertions

- 什麼是Assertion
 - 用來維護程式使之更堅固(robust)，零錯誤
 - Assertion通常用來檢查一些關鍵的值，避免這些值有錯誤時，讓程式無法繼續執行
- Assertion語法
 - **assert <boolean_expression> :**
 - 當boolean_expression為 false 時，會丟出AssertionError，程式即中斷
 - **assert <boolean_expression> : <detail_expression>;**
 - 當boolean_expression為 false 時，會執行後面的運算式，最常用為字串，以說明錯誤的原因
 - 如：assert obj != null : “這物件不得為null”;
 - 如：assert k != 0 : ”k值不得為0”;
- 執行：java **-ea** TestAssertion