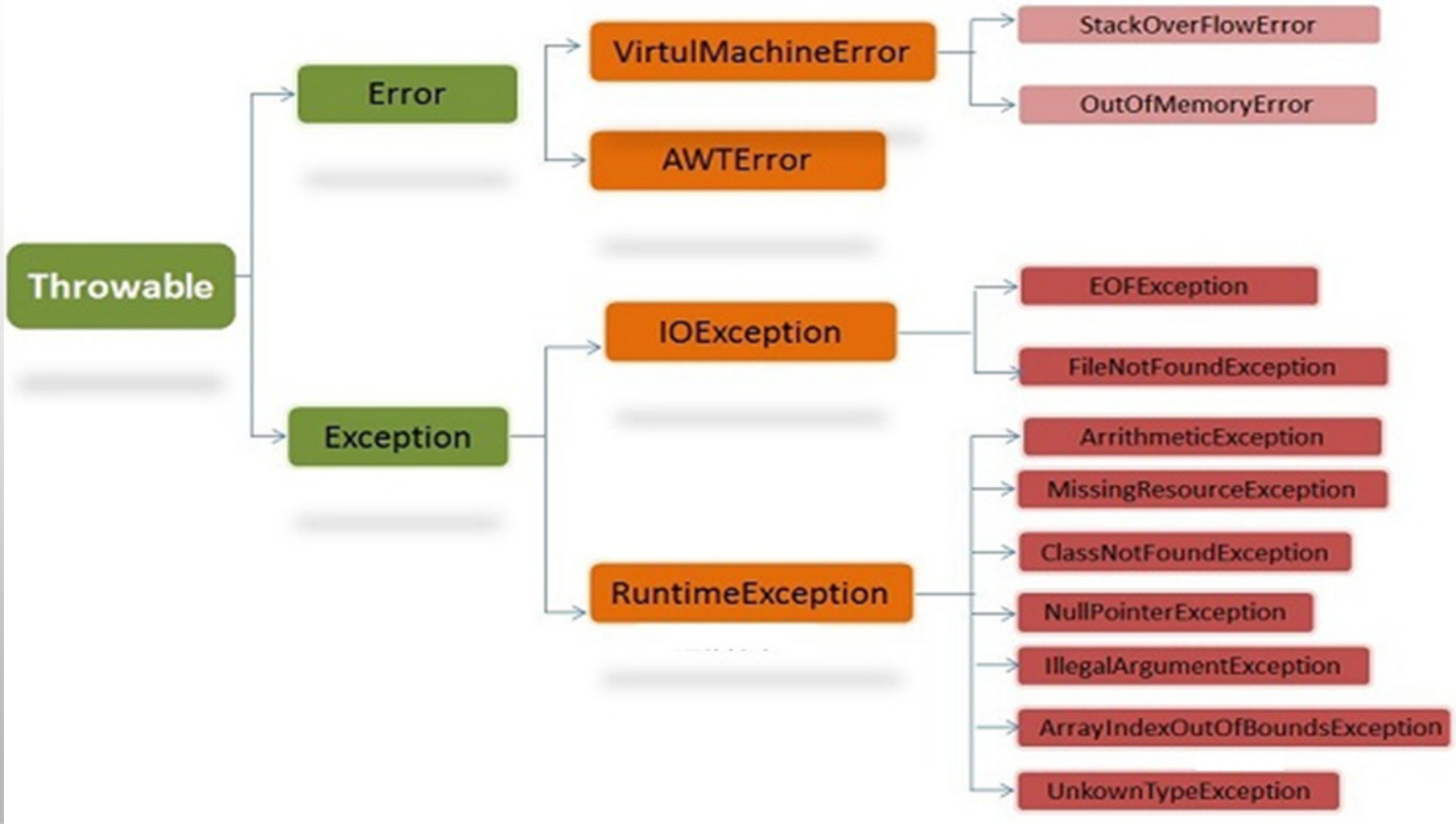
例外處理

學習目標

- 使用TRY、CATCH處理例外
- 認識例外繼承架構
- 認識THROW、THROWS的使用時機
- 運用FINALLY關閉資源
- 使用自動關閉資源語法
- 認識AUTOCLOSEABLE介面

Throwable類別繼承階層



Throwable類別繼承階層

- 在Java中，所有的例外都有一個共同的祖先Throwable（可拋出）。
- Throwable： 有兩個重要的子類，即Exception（例外）和 Error（錯誤），二者都是 Java例外處理的重要子類別，各自都包含大量子類別。

Throwable類別繼承階層

錯誤與例外的區別

- Error（錯誤）：是指程式無法處理的錯誤，表示運行應用程式中較嚴重問題。
- Exception（例外）：是程式本身可以處理的例外。
  Exception 這種例外分兩大類：運行時例外和非運行時例外（編譯例外）。程式中應當盡可能去處理這些例外。

Throwable類別繼承階層

- 一個程式出現問題主要是由三類原因引起的：語法問題、運行時問題和邏輯問題。
- 語法問題:是指程式的格式錯了，或者某個缺少括弧。
- 運行時問題是指在程式運行的時候出現的問題，如：空指標例外:陣列越界，除數為零等。
- 邏輯問題:是指運行結果與預想的結果不一樣，這種問題最難解決。
- 輸出入例外:是指在程式運行時出現的問題。如：檔找不到、網路連接失敗、非法參數等。

例外Exception

Exception類別

其子類別為各種例外物件，即例外處理可以處理的部分。以下為常見的例外：

例外名稱	例外狀況
ArithmeticException	分母為零時
IndexOutOfBoundsException	索引值超過物件上限時
ArrayIndexOutOfBoundsException	陣列索引值超過上限時 (子類別)
StringIndexOutOfBoundsException	字串索引值超過上限時 (子類別)
NegativeArraySizeException	陣列時索引值為負數
NullPointerException	呼叫內容為 null 的變數
ArrayStoreException	陣列元素型態與陣列所宣告不合
ClassCastException	類別轉型 (Casting) 失敗
IllegalArgumentException	傳入參數型態不符
SecurityException	動作違反安全原則

例外Exception

常見的 Checked Exceptions 介紹

例外名稱	例外情況
ClassNotFoundException	當存取一個不存在或不正確的類別時，就會發出此例外。
DataFormatException	嘗試將一個數值指定給型態不符的變數時，就會發出此例外。
PrinterException	當列表機發生問題時，就會發生此例外。
IOException	檔案不存在，或開檔不成功時，就會發出此例外。

例外Exception

- Java處理執行時期間發生錯誤的機制，稱為例外處理(exception handling) 。
- 捕捉例外:使用者可以「嘗試(try)」著去「捕捉(catch)」到執行時期錯誤，並直接加以處理，以避免程式因為執行時期的錯誤而立即停止。
- 拋出例外THROWS:當程式於執行時期產生錯誤時會產生例外物件，稱為「有一個例外被丟出(throw)了」。
- 當例外被丟出時若未能立即被捕捉(catch)並加以處理，則程式的執行會立即終止。
- 所有的例外相關類別都是由java.lang類別庫的Throwable(可丟出的)類別衍生出來的。

例外Exception

拋出例外throws

- 當一個方法出現錯誤引發例外時，方法創建例外物件並交付運行時系統，例外物件中包含了例外類型和例外出現時的程式狀態等例外資訊。運行時系統負責尋找處置例外的程式並執行。

捕獲例外try、catch、finally

- 在方法例外被丟出之後，運行時系統將轉為尋找合適的例外處理器（exception handler）。潛在的例外處理器是例外發生時依次存留在呼叫堆疊中的方法的集合。

ArithmeticException範例

```
package ch8_1;
public class ch8_1 {
    public static void main(String[] args) {
      // 產生除以零的例外
        for ( int i = 2; i > -1; i-- ) {
            System.out.println("計算結果: " + 6/i);
        }
            System.out.println("程式結束!");
     }
}
```

結果:

```
計算結果: 3
計算結果: 6
Exception in thread "main"
java.lang.ArithmeticException: / by zero
at ch8_1.ch8_1.main(ch8_1.java:8)
```

try, catch and finally

例外處理器（exception handler）

- 由 try、catch、finally 區塊所組成

處理例外常用的方法：

方法	解釋
String getMessage()	傳回錯誤訊息
void printStackTrace()	印出 “錯誤處理呼叫鍊” (Error Handling Calling Chain) 至標準錯誤設備 (System.err)
void printStackTrace (PrintStream s)	印出 “錯誤處理呼叫鍊” 至指定的輸出設備
Throwable fillInStackTrace()	將此錯誤再度放回 Java VM 中，供他人繼續使用
String toString()	印出此 Throwable 物件的簡短描述

try-catch語句

（一）try-catch語句

在Java中，通過try-catch語句捕獲例外，這兩個語句必須同時使用。其一般語法形式為：

```
try {
    //檢查此區塊裡程式是否有例外產生，若有就丟出例外
} catch (例外類型例外的變數名id1){
    //抓取並處理例外類型id1
} catch (例外類型例外的變數名id2){
    //抓取並處理例外類型Type2
}
```

try-catch語句

```
package ch8_2;
public class ch8_2 {
   public static void main(String[] args) {
        int a[] = new int[5];
        System.out.println("為陣列元素賦值");
        try {
             a[5] = 62;
             System.out.println("能執行這行程式嗎？");
        } catch (ArrayIndexOutOfBoundsException e) {
             System.out.println("陣列越界例外");
        }
        System.out.println("賦值完畢");
   }
}
```

結果:

```
為陣列元素賦值
陣列越界例外
賦值完畢
```

try-catch-finally語句

（二）try-catch-finally語句

try-catch-finally語句的一般語法形式為：

```
try {
   // 檢查此區塊裡程式是否有例外產生，若有就丟出例外
} catch (Type1 id1) {
   // 抓取並處理try拋出的例外類型Type1
} catch (Type2 id2) {
   // 抓取並處理try拋出的例外類型Type2
} finally {
   // 無論是否發生例外，都將執行的語句區塊
}
```

try-catch-finally語句

- try區塊：用於捕獲例外。其後可接零個或多個catch區塊，如果沒有catch區塊，則必須跟一個finally區塊。
- catch區塊：用於處理try捕獲到的例外。
- finally區塊：無論是否捕獲或處理例外，finally區塊裡的語句都會被執行。當在try塊或catch區塊中遇到return語句時，finally語句區塊將在方法返回之前被執行。

try-catch-finally語句

在以下4種特殊情況下，finally區塊不會被執行：
1）在finally語句區塊中發生了例外；
2）在前面的程式中用了System.exit()退出程式。
3）程式所在的執行緒停止。
4）關閉CPU。

try、catch、finally語句區塊的執行順序

（三）try、catch、finally語句區塊的執行順序

- 當try沒有捕獲到例外時：try語句區塊中的語句逐一被執行，程式將跳過catch語句區塊，執行finally語句區塊和其後的語句；
- 當try捕獲到例外，catch語句區塊裡沒有處理此例外的情況：當try語句區塊裡的某條語句出現例外時，而沒有處理此例外的catch語句區塊時，此例外將會拋給JVM處理，finally語句塊裡的語句還是會被執行，但finally語句區塊後的語句不會被執行；

try、catch、finally語句區塊的執行順序

（三）try、catch、finally語句區塊的執行順序

•當try捕獲到例外，catch語句塊裡有處理此例外的情況：在try語句區塊中是按照順序來執行的，當執行到某一條語句出現例外時，程式將跳到catch語句區塊，並與catch語句區塊逐一匹配，找到與之對應的處理常式，其他的catch語句區塊將不會被執行，

•而try語句區塊中，出現例外之後的語句也不會被執行，catch語句區塊執行完後，執行finally語句區塊裡的語句，最後執行finally語句區塊後的語句。

Exception

```
package ch8_3;
public class ch8_3 {
    public static void main(String[] args) {
// 例外處理程式敘述
        try { // 產生除以零的例外
             for (int i = 2; i > -1; i--)
                 System.out.println("計算結果: " + 6 / i);
        } catch (ArithmeticException ex) { // 顯示例外資訊
                 System.out.println("例外說明: " + ex.getMessage());
                 System.out.println("例外原因: ");
                 ex.printStackTrace();
        } finally {
                 System.out.println("錯誤處理結束");
        }
                 System.out.println("程式結束!");
    }
}
```

結果:

```
計算結果: 3
計算結果: 6
例外說明: / by zero
例外原因:
java.lang.ArithmeticException: / by zero
at ch8_3.ch8_3.main(ch8_3.java:9)
錯誤處理結束
程式結束!
```

try, catch and finally

```
try  {
      檢查此區塊裡程式是否有例外產生，若有就丟出例外;
} catch (例外類別  例外變數)  {
      例外處理程式碼;
} catch (例外類別  例外變數)  {
      例外處理程式碼;
} catch (例外類別  例外變數)  {
      例外處理程式碼;
} …
finally   {
      善後處理程式碼;
      (比如：將霸佔的記憶體歸還，或已開啟的檔案關閉等
等)
}
```

try, catch and finally

```
package ch8_4;
public class ch8_4 {
    public static void main(String[] args) {
            // 例外處理程式敘述
        try {
             String str = args[0]; // 產生超過陣列範圍例外
            // 產生除以零的例外
             for (int i = 2; i > -1; i--)
                 System.out.println("計算結果: " + 6 / i);
        } catch (ArithmeticException ex) {
                // 處理除以零的例外
                System.out.println("例外說明: " + ex.getMessage());
                System.out.print("例外原因: ");
                ex.printStackTrace();
        } catch (ArrayIndexOutOfBoundsException ex) {
                 // 處理超過陣列範圍例外
                System.out.println("例外說明: " + ex.getMessage());
                System.out.print("例外原因: ");
                ex.printStackTrace();
        } finally {
                System.out.println("錯誤處理結束");
        }
    }
}
```

結果:

```
例外說明: Index 0 out of bounds for length 0
例外原因: java.lang.ArrayIndexOutOfBoundsException: Index 0 out of
bounds for length 0
at ch8_3.ch8_3.main(ch8_3.java:8)
錯誤處理結束
```

例外處理

有一些例外必須由程式員來決定拋出，系統無法拋出這種例外，有兩種方式：

- 第一種是在方法頭寫出需要拋出的例外（利用throws報告）；
- 第二種方法是在方法體內拋出例外（利用throw拋出）。
- throw一般要與if語句配合使用。

throws例外報告

（一）**throws**例外報告

throws語句用在方法定義時宣告該方法要拋出的例外類型，如果拋出的是Exception例外類型，則該方法被宣告為拋出所有的例外。多個例外可使用逗號分割。

throws語句的語法格式為：

```
方法名 throws Exception1,Exception2,..,ExceptionN {
//方法體
}
```

throws例外報告

- throws只能出現在方法定義的頭部，不能出現在方法體中，也不能出現在方法呼叫部分。
- 如果一個方法宣告時用throws子句宣告了某個例外類別可能在該方法中出現，則該方法體中不需要用catch語句捕獲該例外，但是呼叫該方法的方法中必須對該例外進行捕獲並處理。
- 一個方法中產生例外且該例外必須處理時，要麼被方法的try-catch-finally捕獲並處理，要麼用throws拋給方法的呼叫者。

throws例外報告

```
package ch8_5;
public class ch8_5 {
    public static void main(String[] args) {
        try {
             showSalary("王小明", 35000);
             showSalary("李小華", 50000);
        } catch (IllegalArgumentException e) { // 捕捉自行拋出的例外
        System.out.println("例外內容：" + e.getMessage());
        }
    }

    static void showSalary(String name, int money) throws IllegalArgumentException {
       System.out.println("員工：" + name);
       if (money >= 20000 && money <= 40000)
          System.out.printf("\t底薪：%d\t獎金：%.1f %n", money, (int) money * 0.08);
       else
          throw new IllegalArgumentException("呼叫方法引數錯誤"); // 自行拋出例外
    }
}
```

結果:

```
員工：王小明
底薪：35000獎金：2800.0
員工：李小華
例外內容：呼叫方法引數錯誤
```

throw例外丢出

（二）throw例外丢出

throw總是出現在函數體中，用來拋出一個Throwable類型的例外。程式會在throw語句後立即終止，它後面的語句執行不到，然後在包含它的所有try塊中（可能在上層呼叫函數中）從裡向外尋找含有與其匹配的catch子句的try塊。

throw例外丟出

```
package ch8_6;
public class ch8_6 {
    static void math_div(int n1, int n2) {
    try {
        if (n2 == 0)
            throw new ArithmeticException("除數為零"); // 自行拋出例外物件
        int div = n1 / n2;
        System.out.println(n1 + "/" + n2 + " = " + div);
    } catch (ArithmeticException e) { // 捕捉自行拋出的例外
        System.out.println("例外內容:" + e.getMessage());
    }
}

public static void main(String[] args) {
    int num1, num2;
    num1 = 6;
    num2 = 3;
    System.out.println("被除數 = " + num1 + ", 除數 = " + num2);
    math_div(num1, num2);
    num1 = 6;
    num2 = 0;
    System.out.println("\n被除數 = " + num1 + ", 除數 = " + num2);
    math_div(num1, num2);}
}
```

結果:

```
被除數 = 6, 除數 = 3
6/3 = 2

被除數 = 6, 除數 = 0
例外內容:除數為零
```

throw和throws區分

Throw

- 只會出現在方法體中，當方法在執行過程中遇到例外情況時，將例外資訊封裝為例外物件，然後throw出去。throw關鍵字的一個非常重要的作用就是例外類型的轉換。

Throws

- 只能出現在方法定義的頭部，不能出現在方法體中，表示出現例外的一種可能性，並不一定會發生這些例外；throw則是拋出了例外，執行throw則一定拋出了某種例外物件。

Checked/Unchecked 例外

不檢查(Unchecked)例外

- RuntimeException類別及Error類別的衍生例外類別，不需要用throws關鍵字宣告可能的丟出。

檢查(checked)例外。

- 除了 RuntimeException 以外，所有 Exception 的衍生類別都屬於此類,要用throws關鍵字宣告可能的丟出。

自訂例外類別

創建自訂例外類別

自訂的例外類別至少必須繼承Throwable類別（它為Throwable的子類別）。一般情況下通過繼承Exception類別或它的子類別，來實現自訂例外類別。

語法格式如下：

```
class 自定例外類別名稱 extends Exception
 {
    public 自定例外類別名稱() //建構式
    {
        super(); //必須為第一行 //其他敘述
    } //其他成員
}
```

自訂例外類別

```
package ch8_7;
public class MyException extends Exception {
	public void ErrorAnswer() {
		System.out.println("答案錯誤");
	}
}
```

```
package ch8_7;
public class ch8_7 {
	public static void main(String[] args) {
		java.util.Scanner in = new java.util.Scanner(System.in);
		try {
			System.out.println("2+3=?");
			if (5 != in.nextInt()) {
				throw new MyException();
			} else {
				System.out.println("回答正確");
			}
		} catch (MyException e) {
			e.ErrorAnswer();
		}
	}
}
```

結果:

2+3=?
5
回答正確