錯誤處理

很多時候在程式中的錯誤處理是必要的，這也是我們避免程式無法正常執行的**責任**。

錯誤處理很重要，但較多的狀況是散亂且大量的錯誤處理**模糊**了原演算法的意圖。

- **使用例外事件而非回傳錯誤碼（參考函式章節）**

ex:

```
if (Status.OK == recordCustomerBeforeDelete(customer)) {
    // do something ...
    if (Status.OK == deleteCustomer(customer)) {
        // do something ...
    } else {
        // do error control ...
    }
} else {
    // do error control ...
}
```

- 使用代碼
呼叫者必須處理非預期的代碼，將導致更深層的巢狀結構。

```
try{
    recordCustomerBeforeDelete(customer);
    deleteCustomer(customer);
}catch(Exception e){
    // do error control ...
}
```

- 使用例外處理
統一處理錯誤資訊，使程式更為整潔。

- **在開頭寫下你的 Try / Catch / Finally 敘述（參考函式章節）**

若 try / catch 混在程式中，讀者容易混淆程式的結構。

應將 try / catch 向外提取並寫在開頭，**錯誤處理本身就是一件事情**。

我們應將錯誤處理以及演算法拆開看待、維護它們。

ex:

```
public void delete(Customer customer) {
    try {
        deleteCustomerReferences(customer);
    } catch (Exception e) {
        // do error control ...
    }
}
```

```
protected void deleteCustomerReferences(Customer customer) {
    recordCustomerBeforeDelete(customer);
    deleteCustomer(customer);
}
```

試著在函式中拋出例外，並由函式層次中最外層的 try / catch 處理錯誤。

遵循這個規則，try / catch 將定義了一組函式的 scope，有點像是 **Transaction** 的概念。

- **使用不檢查型例外（Use Unchecked Exceptions）**

重要的函式庫或是工具類，檢查型例外有它的必要性。ex: method.invoke

而在一般的應用程式中，應使用不檢查型例外認真。

不檢查型例外指的是 **RuntimeException**，**Error**，以及他們的衍生子類別。

拋出檢查型例外，意味著該函式署名必須增加 throws 子句。

假設在一組函式中較低階層次的函式拋出檢查型例外，則會迫使較高階層次的函式署名一起變動或是捕捉相對應的錯誤。

更改一個低階函式，卻使得系統由下到上一連串的程式碼異動開發者怒。

這違反了**開放封閉原則** (註1)。

- **提供發生例外的相關資訊**

如果是開發者拋出的例外，請提供必要的錯誤資訊以及錯誤型態隨著例外一起被傳遞。

雖然 JAVA 能取得錯誤堆疊，但是如果僅看這些資訊一般無法預測錯誤的原始意圖為何。

- **從呼叫者的角度定義例外類別**

可以為同一個概念下類似的錯誤類別定義一個共同父類別。

在外層函式，僅需要捕捉這個父類別即可大幅簡化程式碼。

若是使用第三方 API ，可以使用**包裹類別** (Wrapper) 捕捉和翻譯第三方 API 的例外。

```
public class WrapperClass {

    Service thirdPartyAPI;

    public WrapperClass(Service thirdPartyAPI){
        this.thirdPartyAPI = thirdPartyAPI;
    }

    public void doThirdPartyAPI() {
        try{
            thirdPartyAPI.doSomething();
        }catch(ApiException e){
            // error control
        }
    }
}
```

包裹類別是相當實用的開發技巧，它有以下優點 …

1. 應用程式降低了對第三方 API 的依賴
2. 測試程式時可以透過包裹類別模擬第三方 API 的呼叫
3. 可客製化對第三方 API 的控制（如自定義錯誤資訊）

- **不要傳遞 null**

應盡可能避免傳遞 null 至參數中，null 的出現代表了有發生錯誤的可能性。

除了增加函式的工作份量（檢核是否為 null的瑣碎片段），也讓程式碼相對不整潔。

註

1. **開放封閉原則：**類別應該開放，以便擴充；應該關閉，禁止修改。