邊界

- **使用第三方軟體的程式碼**

在系統應用程式碼與第三方函式庫的「**邊界**」，應合理使用包裹物件將第三方函式庫進行封裝。

參考至錯誤處理章節之**從呼叫者的角度定義例外類別**

```
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）

- **探索及學習邊界**

我們常常為了解決某方面的問題並提高產出，需要導入新的第三方函式庫至我們的系統。

學習第三方函式庫很痛苦，整合新的第三方函式庫也很痛苦。

假設一起來真的是... ~~不要不要的~~。

建議的做法是使用學習式測試（learning tests），嘗試撰寫第三方函式庫的測試程式驗證我們知識範圍內所預期的結果以及實際的運作是否符合預期。

最後我們可以將學習式測試類別中獲得的**知識**以及**事實**封裝到我們自定義的類別中使用。

- **學習式測試比不花工夫更好**

學習式測試累積的程式碼除了驗證我們對第三方函式庫的了解程度以外，也可以在第三方函式庫版本升級之後快速的檢驗是否一切功能正常且符合原系統需求。

假設無法通過測試我們可以立刻的找出問題。

如果沒有這些「**邊界測試**」，預期第三方函式庫將停留在舊版本比預期長的時間。

- **使用尚未存在的程式**

開發時我們可能會遇到「**未知的邊界**」。

也就是說我們並不確定與我們系統橋接的子系統或是函式庫應當呈現什麼面貌。

此時比較做好的做法是儘早預先定義好期望的資料交換介面，取得系統整合主導權。

且使用 **Adapter** 模式（註1）整合邊界開發與測試。

由 **Adapter** 封裝邊界程式，一旦邊界程式升級與異動，也僅僅需要修改 **Adapter** 。

- **簡潔的程式邊界**

透過**包裹類別**或是 **Adapter** 妥善封裝邊界程式，可以保證在未來第三方函式庫或是其它子系統如果升級或是程式異動無法通過我們的測試時，僅需要做最少的修改且不影響到原來的程式。

系統的主導權，不能被第三方函式庫或是其它子系統控制。

我們必須控制他們。

~~雖然實際情況就像是你必須控制上臉書的慾望但顯然常常不如預期。~~

註

1. **Adapter**：又稱轉接器模式，請參考

https://github.com/xul4m4d93/DesignPattenLearn/tree/master/src/com/kb/patten/adapter