物件及資料結構

- **德摩特爾法則（The Law of Demeter）**

假設有個類別為 C，C 的某個函式為 F，則 C 只能使用以下事項 ...

1. C
2. 任何由 F 所產生的物件
3. 任何當作參數傳給 F 的物件
4. C 的實體變數所持有的物件

在 Head First Design Pattern 書中稱之為**極少化守則**，換句話說：**只和密友說話**。

- **火車事故**

```
Skill skill = customerMoudle.getCustomer().getSkill();
```

~~對，這就是一輛程式火車。~~

不但不好閱讀，且只要其中一層資料結構有異動就會出現編譯錯誤。

這是一種懶散的寫法，可以的話應避免或是說根本不該出現。

```
Customer customer = customerMoudle.getCustomer();
Skill skill = customer.getSkill();
```

較好的做法是適當切割，資料異動後更改幅度**或許**會小一點。

依據**德摩特爾法則**，我們不需要讓模組知道太多的資料結構。

當在函式中揭露的實現細節越多，耦合度相對的越高讀者越怒。

- **隱藏結構**

```
Skill skill = customerMoudle.getCurrentCustomerSkill();
```

依據**德摩特爾法則**，我們應將參考到的相關物件將之內部結構隱藏起來。

我們應告訴物件**去做什麼**，不需要讓使用這個物件的參考知道此物件內部的資料結構。

知道別人的私事越少，越好。這邊是指**物件**如果你喜歡八卦我也沒辦法

Hey！只和密友說話，還記得吧？