

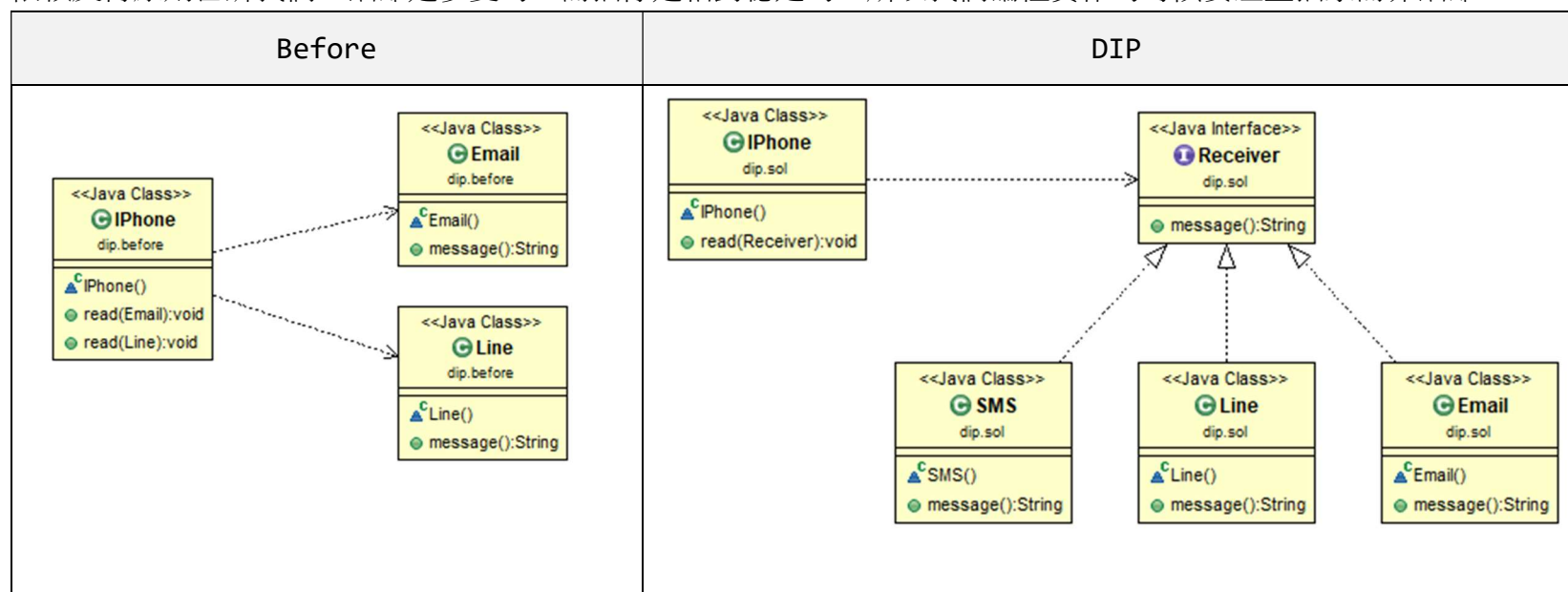
Java 物件導向設計原則 SOLID

段維瀚老師

9.1 依賴反轉原則 Dependency inversion principle (DIP)

- 一. 高層模組不應該依賴底層模組，它們都應該依賴抽象。
- 二. 抽象不應該依賴細節。細節應該依賴抽象。

依賴反轉原則告訴我們：細節是多變的，而抽象是相對穩定的。所以我們編程實作的時候要注重抽象而非細節。

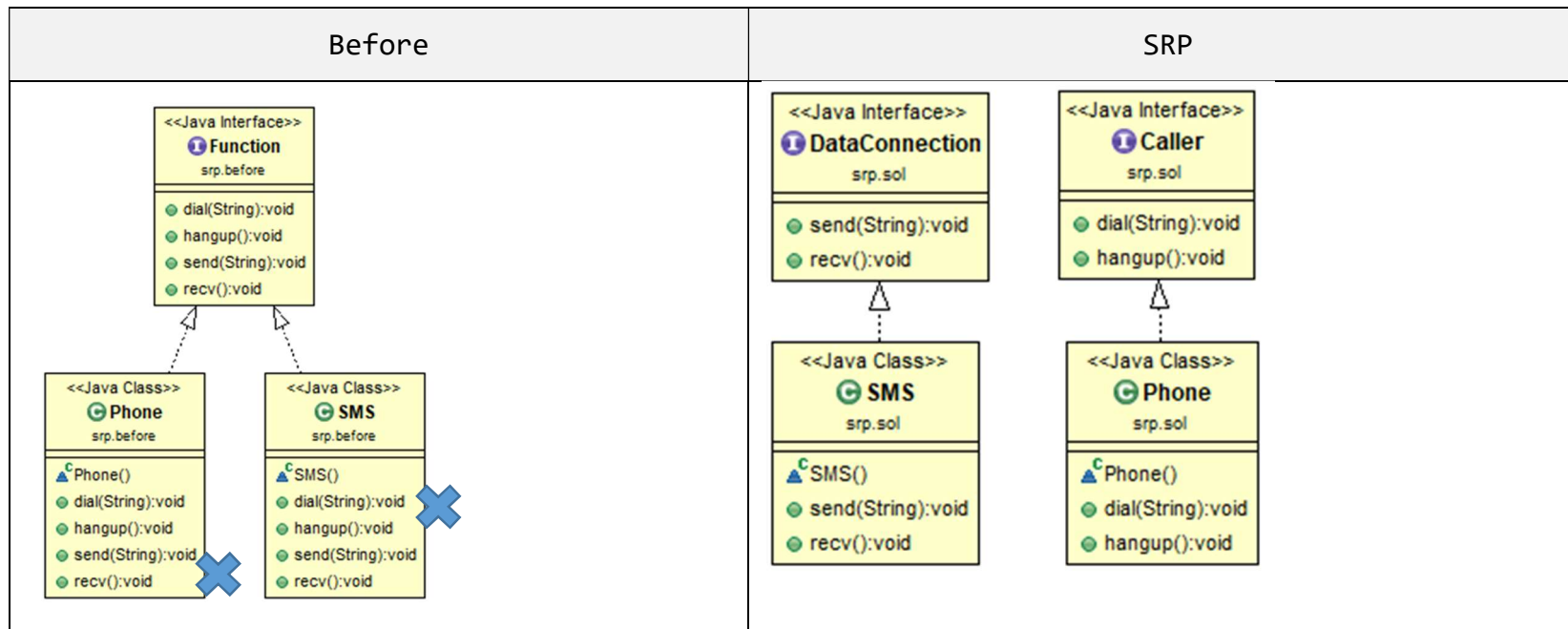


Java 物件導向設計原則 SOLID

段維瀚老師

9.2 單一職責原則 Single responsibility principle (SRP)

就一個類別而言，應該僅有一個引起它變化的原因。簡單地說，就是一個類別應就只負責一項職責。



Java 物件導向設計原則 SOLID

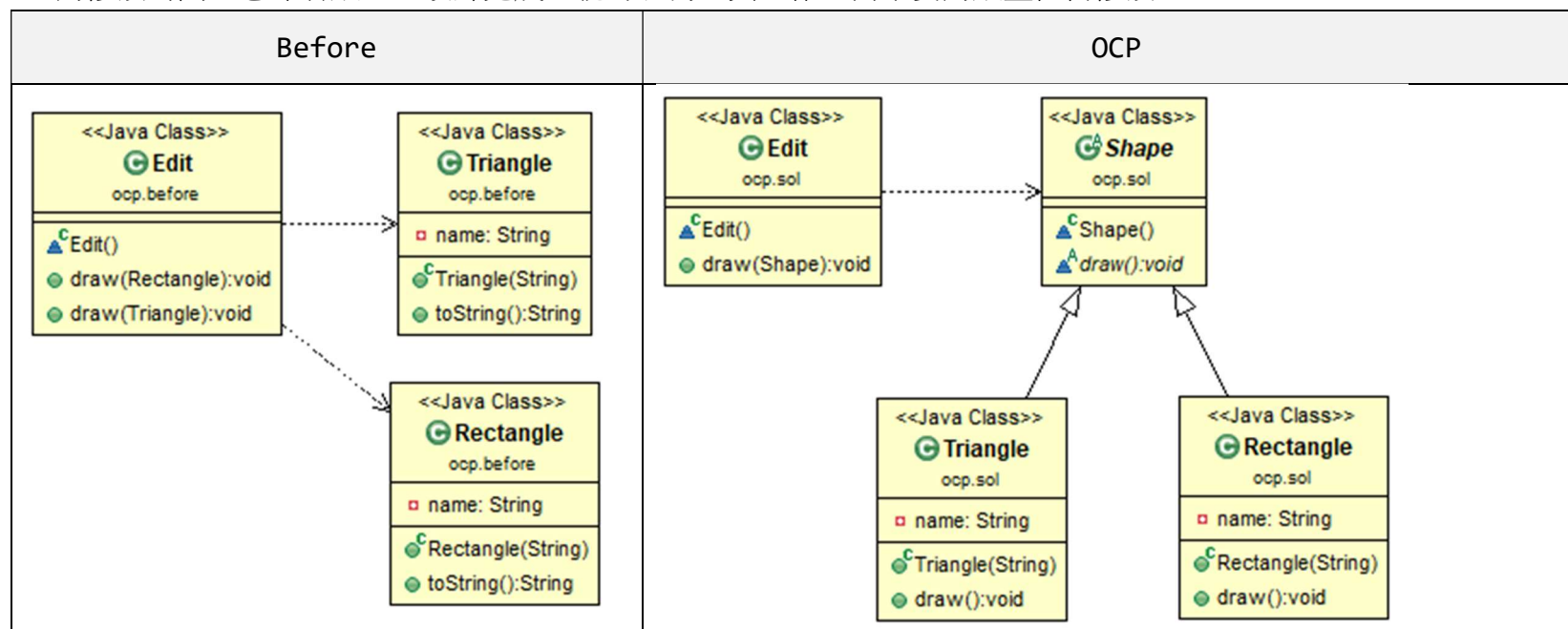
段維瀚老師

9.3 開放關閉原則 Open-Close principle (OCP)

軟件維護（類別、模組、函數等等）應該可以擴展（增加功能），但是不可修改。

開閉原則主要體現在兩個方面：

- 1、對擴展開放，意味著有新的需求或變化時，可以對現有代碼進行擴展，以適應新的情況。
- 2、對修改封閉，意味著類一旦設計完成，就可以獨立其工作，而不要對類盡任何修改。



Java 物件導向設計原則 SOLID

段維瀚老師

9.4 里氏替換原則 Liskov substitution principle (LSP)

繼承問題：

子類別可以無條件使用父類別的成員或方法，不過子類別可以透過覆寫機制，破壞這種結構，讓孫類別無法執行父類別的資源。

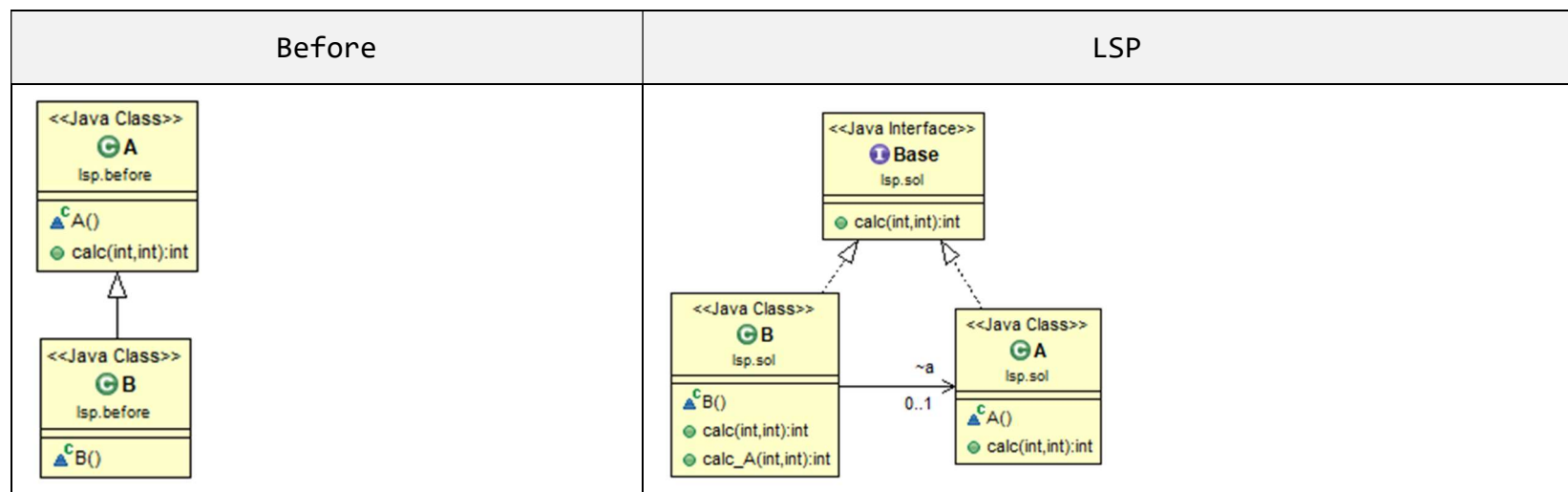
增加物件間的耦合：繼承會讓物件之間彼此耦合影響，造成物件的可移植性降低。

例如：若某一個父類別需要做修改時，他可能必須要考慮到子類別，因為子類別有可能因為他的修改而造成問題。

所以如何正確地繼承，這正是里約替換原則要解決的問題。

里約替換原則

繼承實際上就是增強了二個類別（父子類別）之間的耦合性，在適當的情況下，可以改採用聚合、組合或依賴等設計技巧來解決。

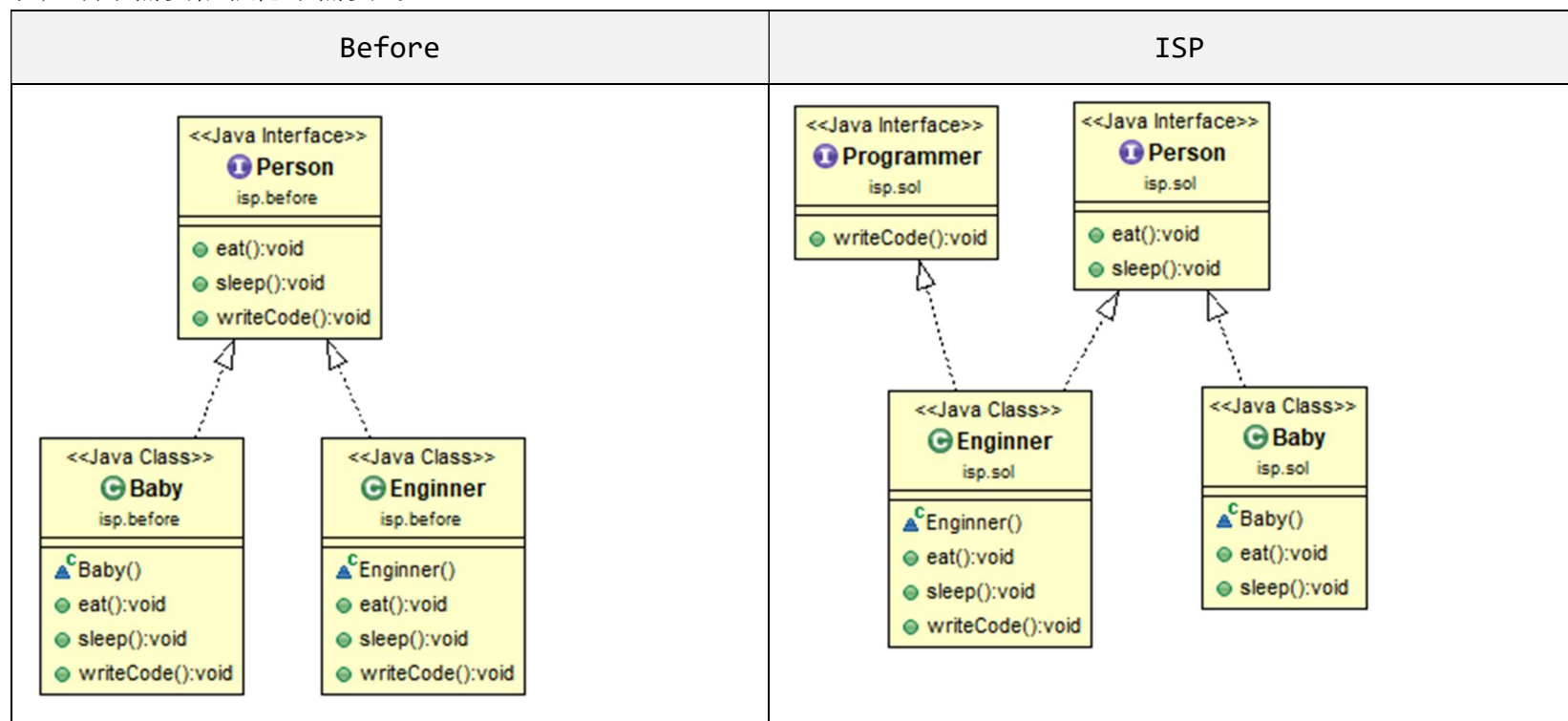


Java 物件導向設計原則 SOLID

段維瀚老師

9.5 介面隔離原則 Interface segregation principle (ISP)

客戶端不需要依賴他不需要的 interface



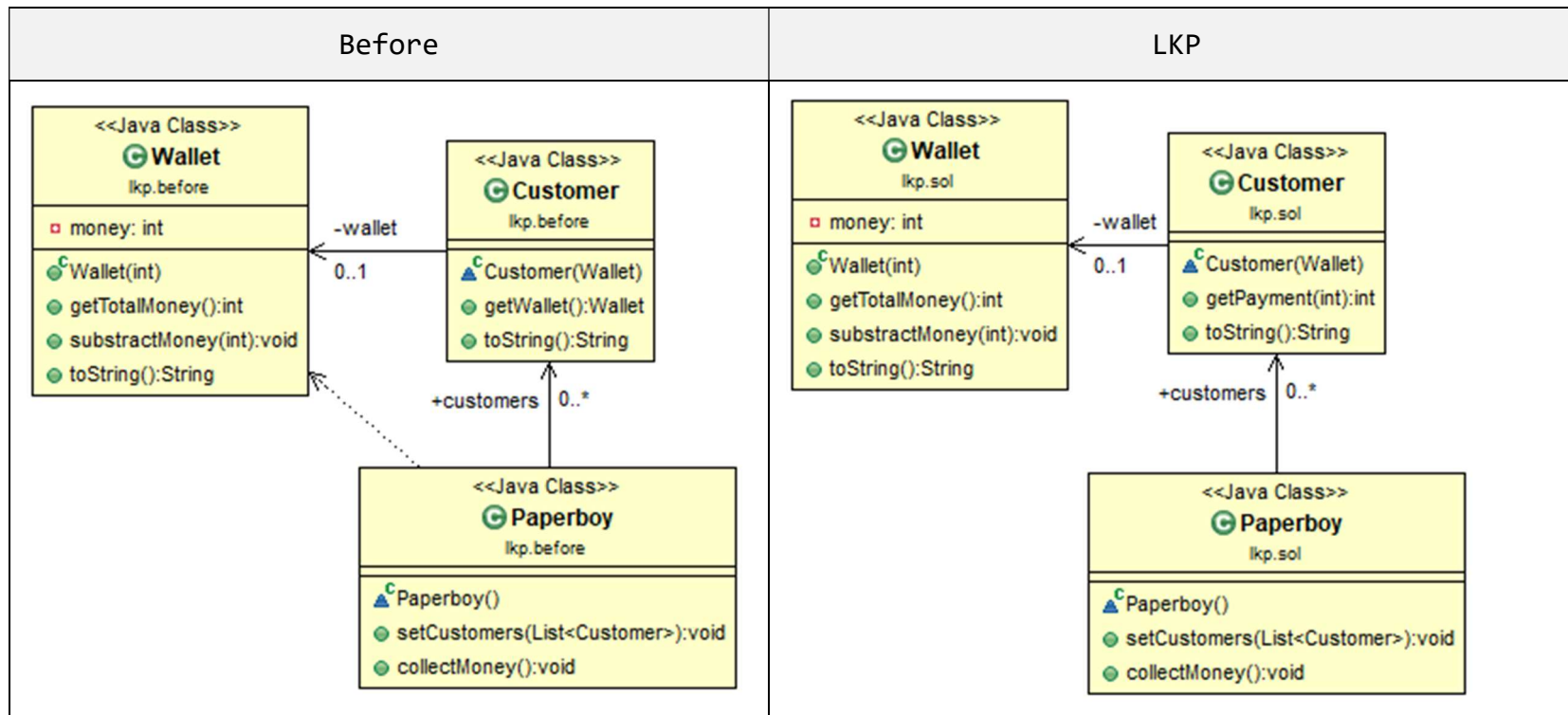
Java 物件導向設計原則 SOLID

段維瀚老師

9.6 最少知識原則 Least Knowledge Principle (LKP)

最少知識原則 (LKP) 也稱為迪米特法則 (Law of Demeter)

就是說一個對象應當對其他對象有盡可能少的了解，類與類之間的了解的越多，關係越密切，耦合度越大，當一個類發生改變時，另一個類也可能發生變化。其實說穿了就是**封裝**的設計。



Java 物件導向設計原則 SOLID

段維瀚老師

9.7 合成/聚合復用原則

要多使用合成(組合, 強聚合), 聚合或依賴, 盡量不要使用繼承。

