Duplicated Code **重覆的程式碼** **(76)**	**Extract Method (提煉函式) (110)** 函式名稱與函式本體的語意距離有所落差。 由函式本體獨立出新函式，並給予可解釋用途之名稱。 **Extract Class (提煉類別) (149)** 某個 class 作了應由兩個 classes 作的事。 建立一個新 class，將相關欄位和函式從舊 class 搬移到新 class。 **Pull Up Method (函式上拉) (322)** 有些函式，在各個 subclasses 中產生完全相同的結果。 將該函式移至 superclass。 **Form Template Method (塑造模版函式) (345)** 兩個函式以相同順序執行大致相近的操作，但不盡相同。 將**執行各操作的順序移至 superclass**，並**以抽象函式定義那些有所不同的操作**，而倚賴多型保持其差異性。
Long Method **過長函式** **(76)**	**Extract Method (提煉函式) (110)** 函式名稱與函式本體的語意距離有所落差。 由函式本體獨立出新函式，並給予可解釋用途之名稱。 **Replace Temp With Query (以查尋取代暫時變數) (120)** 程式以一個暫時變數保存某一運算式的運算結果。 將此運算式獨立到一個**查詢函式**(query，**不得修改任何物件內容**)中。將此暫時變數的所有「被引用點」替換為「對新函式的呼叫」。新函式可被其他函式使用。 該函式不修改任何物件內容，否則需採取 **Separate Query from Modifier (279)**。 若暫時變數被賦值超過一次，考慮使用 **Split Temporary Variable (128)**。 **Replace Method With Method Object (以函式物件取代函式) (135)** 函式內，區域變數的使用導致無法採行 Extract Method (110)。 將此函式放入一個單獨物件(以用途名之)中，**區域變數成為新物件的欄位(field)**，**建構式則保留原函式之參數**，並新增 **compute()函式**負責原始操作。之後便可採取 **Extract Method (110)**。最後，舊函式的本體得以將工作**轉發**給此函式物件。Ex: return new Gamma(123,3).compute(); **Decompose Conditional (分解條件式) (238)** 有一個複雜的條件句 (if-then-else)。 先觀察是否可使用 **Replace Nested Conditional with Guard Clauses (250)**。若否，從 if、then、else 三個段落中分別提煉出**獨立函式**。

Large Class

過大類別

(78)

Extract Class (提煉類別) (149)

某個 class 作了應由兩個 classes 作的事。

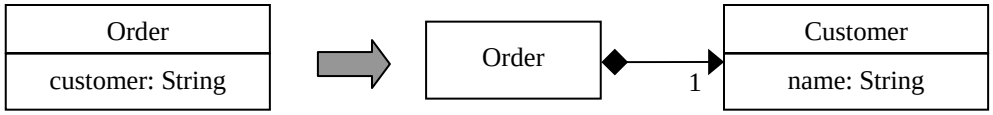
建立一個新 class，將相關欄位和函式從舊 class 搬移到新 class。並決定**是否讓新 class 曝光**：

1. reference object。
2. immutable object 或 immutable interface。
3. 保護性拷貝。 (**委託，class 可以數種不同方式變化**，對比 **Extract Subclass (330)**之繼承)。

Replace Data Value with Object (以物件取代資料值) (175)

你有一筆資料項(data item)，需要額外的資料和行為。

將這筆資料項變成一個**不可變的**(immutable)**實值物件**(Value Object)(**組合關係 composition**)。



作法：1. 新建一個 class，其中宣告型別與「代替換數值」相同的 **final 欄位**，並給定相應的**取值函式**(getter)及以其為參數的**建構式**，而 source class 的取值函式將委託新 class。

2. source class 中「代替換數值欄位」的型別換為新 class。
3. source class 中，代替換欄位的**設值函式**(setter)，令它為新 class 創建一個實體。

(可能進一步使用 **Change Value to Reference (179)**)。

Duplicate Observed Data (複製「被監視資料」) (189)

你有一些 domain data (業務邏輯)置身於 GUI 控件中，而 domain method 需要存取之。

將該筆資料拷貝到一個 domain object 中。建立一個 **Observer** 範式，用以對 domain object 和 GUI object 內的重覆資料進行同步控制。(操作步驟較複雜，請參照書本內容)。

Extract Subclass (提煉子類別) (330)

Class 中的某些特性只被某些(而非全部)實體用到。

新建一個 subclass，將上述的那一部份特性移到 subclass 中。

(**繼承，subclass 只能用以表現一組變化**，對比 **Extract Class (149)**之委託)

Extract Interface (提煉介面) (341)

某個 class 在不同環境下扮演截然**不同的角色**、若干客戶**使用 class 介面中的同一子集**，或者**兩個 classes 的介面有部分相同**。

將相同子集或角色提煉到一個獨立介面中。

Long Parameter List

過長參數列

(78)

Preserve Whole Object (保持物件完整) (288)

你從某個物件取出若干值，將它們作為某一次函式呼叫時的參數。

改傳遞**整個物件**。**(**如果將使依存結構惡化，就不該使用！但若物件有合適的取值函式，便不需操心依存問題)。

Replace Parameter with Method (以函式取代參數) (292)

物件喚起某函式，並將所得結果作為參數，傳遞給另一個函式。而接受該參數的函式也可以喚起前一個函式。

讓參數接受者去除該項參數，並**直接呼叫前一個函式**。(前提：呼叫端通過「其所屬物件內部的另一個函式」來計算參數，並在計算過程中「未曾引用呼叫端其他參數」，否則將需使用 Replace Parameter with Explicit Methods (285))。

Introduce Parameter Object (引入參數物件) (295)

某些參數總是很自然地同時出現。

以一個**不可變物件**(immutable object)取代這些參數。並觀察是否可採行 **Move Method (142)**或 **Extract Method (110)**。

Divergent Change **發散式變化** **(79)**	一旦需要修改，希望能夠跳到系統的某一點，只在該處修改。如果不能做到這點，即是此種壞味道。(**一個class受多種變化影響**)。 **Extract Class (提煉類別) (149)** 某個 class 作了應由兩個 classes 作的事。 建立一個新 class，將相關欄位和函式從舊 class 搬移到新 class。並決定**是否讓新 class 曝光**： 1. reference object。 2. immutable object 或 immutable interface。 3. 保護性拷貝。 (**委託，class 可以數種不同方式變化**，對比 **Extract Subclass (330)**之繼承)。
Shotgun Surgery **散彈式修改** **(80)**	如果每次遇到某種變化，就必需在許多不同的class內做出許多小修改。 **(一種變化引發多個classes相應修改)**。 **Move Method (搬移函式) (142)** 程式中，有個函式與其所駐 class 之外的另一個 class 進行更多交流：呼叫後者，或被後者呼叫。 在該函式最常引用的 class 中建立一個有著類似行為的新函式。將舊函式變成一個單純的**委託函式**，或是將舊函式**完全移除**。當需要使用 source class 特性： 1. 將此特性也移至 target class。 2. 建立或使用一個從 target class 到 source 的引用關係。 3. 將 source object 當作參數傳給 target method。 4. 若所需特性是個變數，將它當作參數傳給 target method。 **Move Field (搬移欄位) (146)** 程式中，某個欄位被其所駐 class 之外的另一個 class 更多地用到。 在 target class 建立一個 new field，修改 source field 的所有用戶，令它們改用 new field。 (所謂「使用」動作可能是通過設值/取值函式間接進行)。 **Inline Class (將類別內聯化) (154)** 某個 class 沒有作太多事情(沒有承擔足夠責任)。 將 class 的所有特性搬移到另一個 class 中，然後移除原 class。 如果「以一個獨立介面表示 source class 函式」更合適的話，先使用 **Extract Interface (341)**。

Feature Envy **依戀情節** **(80)**	函式對某個class的興趣高於對自己所處之host class的興趣。 **Extract Method (提煉函式) (110)** 函式名稱與函式本體的語意距離有所落差。 由函式本體獨立出新函式，並給予可解釋用途之名稱。實作類型： 1. 無區域變數。 2. 有區域變數：將區域變數作為**參數**。 3. 對區域變數在賦值：使用 **Remove Assignments to Parameters (131)**。 **Move Method (搬移函式) (142)** 程式中，有個函式與其所駐 class 之外的另一個 class 進行更多交流：呼叫後者，或被後者呼叫。 在該函式最常引用的 class 中建立一個有著類似行為的新函式。將舊函式變成一個單純的**委託函式**，或是將舊函式**完全移除**。當需要使用 source class 特性： 1. 將此特性也移至 target class。 2. 建立或使用一個從 target class 到 source 的引用關係。 3. 將 source object 當作參數傳給 target method。 4. 若所需特性是個變數，將它當作參數傳給 target method。 **Move Field (搬移欄位) (146)** 程式中，某個欄位被其所駐 class 之外的另一個 class 更多地用到。 在 target class 建立一個 new field，修改 source field 的所有用戶，令它們改用 new field。 (所謂「使用」動作可能是通過設值/取值函式間接進行)。
Data Clumps **資料泥團** **(81)**	**Extract Class (提煉類別) (149)** 某個 class 作了應由兩個 classes 作的事。 建立一個新 class，將相關欄位和函式從舊 class 搬移到新 class。並決定**是否讓新 class 曝光**： 1. reference object。 2. immutable object 或 immutable interface。 3. 保護性拷貝。(**委託，class 可以數種不同方式變化**，對比 **Extract Subclass (330)**之繼承)。 **Preserve Whole Object (保持物件完整) (288)** 你從某個物件取出若干值，將它們作為某一次函式呼叫時的參數。 改傳遞**整個物件**。(如果將使依存結構惡化，就不該使用！但若物件有合適的取值函式，便不需操心依存問題)。 **Introduce Parameter Object (引入參數物件) (295)** 某些參數總是很自然地同時出現。 以一個**不可變物件**(immutable object)取代這些參數。並觀察是否可採行 **Move Method (142)**或 **Extract Method (110)**。

Primitive Obsession

基本型別偏執

(81) - 1/3

Extract Class (提煉類別) (149)

某個 class 作了應由兩個 classes 作的事。

建立一個新 class，將相關欄位和函式從舊 class 搬移到新 class。並決定**是否讓新 class 曝光**：

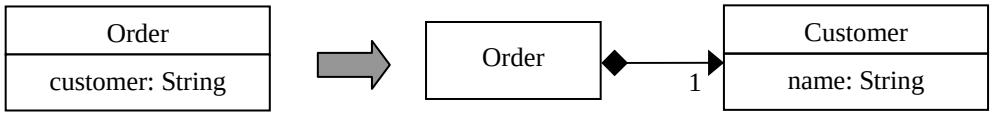
1. reference object。
2. immutable object 或 immutable interface。
3. 保護性拷貝。

(**委託，class 可以數種不同方式變化**，對比 **Extract Subclass (330)**之繼承)。

Replace Data Value with Object (以物件取代資料值) (175)

你有一筆資料項(data item)，需要額外的資料和行為。

將這筆資料項變成一個**不可變的**(immutable)**實值物件**(Value Object)(**組合關係 composition**)。



作法：

1. 新建一個 class，其中宣告型別與「代替換數值」相同的 **final 欄位**，並給定相應的**取值函式**(getter)及以其為參數的**建構式**，而 source class 的取值函式將委託新 class。
2. source class 中「代替換數值欄位」的型別換為新 class。
3. source class 中，代替換欄位的**設值函式**(setter)，令它為新 class 創建一個實體。

(可能進一步使用 **Change Value to Reference (179)**)。

Replace Array with Object (以物件取代陣列) (186)

你有一陣列，其中的元素各自代表不同的東西。

以物件替換陣列。對於陣列中的每個元素，以一個欄位表示之。並逐一對元素添加**取值/設值**函式。

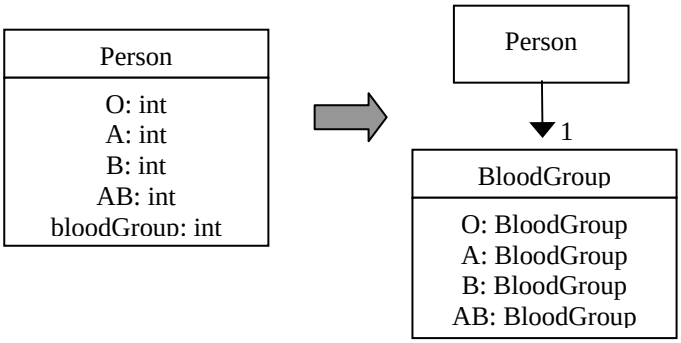
Replace Type Code with Class (以類別取代型別代碼) (218)

Class 之中有一個**數值**型別代碼(numeric type code)，但它並**不影響** class 的行為。

以一個新的 class 替換數值型別代碼(type code)。

原則：只有當 type code 是**純資料**時，才能以 class 來取代它。

否則應採用 **Replace Type Code with Subclasses (223)** 或 **Replace Type Code with State/Strategy (227)**。



作法：

1. 為 type code 建立一個 class。

包含：記錄 type code 的**欄位**(static，且型別不變)、對應的**取值函式**(getter，亦為 static)。

2. 為 source class 提供相應的 **取值/設值**函式 及 **建構子**，都將使用新的 class。

Primitive Obsession

基本型別偏執

(81) - 2/3

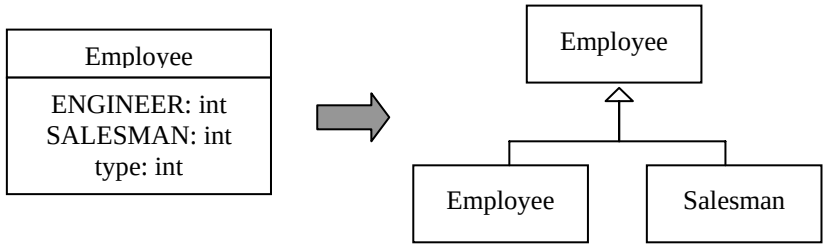
Replace Type Code with Subclasses (以子類別取代型別代碼) (223)

你有一個**不可變的** type code，它**會影響** class 的行為。

以一個 subclass 取代這個 type code。

當 **type code 值會在物件創建後發生改變** 或 **type code 宿主類別已經有了 subclasses**，那麼就不是使用本項重構，而是 **Replace Type Code with State/Strategy (227)**。

如果**宿主類別沒有出現條件式**，那麼 **Replace Type Code with Class (218)** 更合適。



作法：

1. 使用 **Self Encapsulate Field (171)**將 type code 自我封裝起來。

   如果 type code 被傳遞給建構式，需將建構式轉換為 factory method。

2. 為 type code 的每個數值建立相應的 subclass。在每個 subclass 中覆寫 type code 的**取值函式**(getter)，使其傳回相應的值。

3. super class 則將 type code 存取函式宣告為**抽象函式**。

   (可進一步採用 **Replace Conditional with Polymorphism (255)**)。

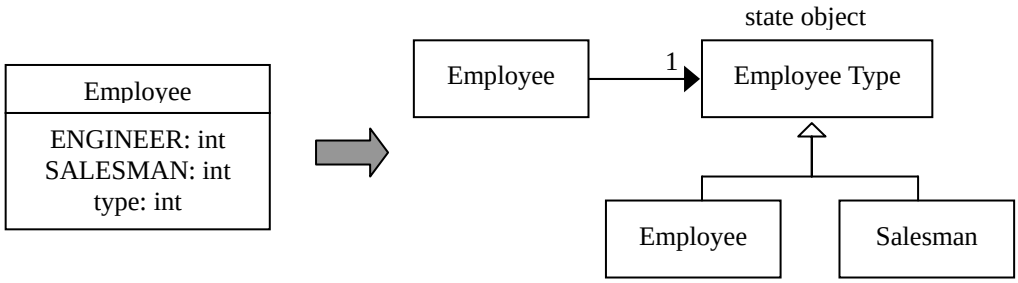
Replace Type Code with State/Strategy (以 State/Strategy 取代型別代碼) (227)

你有一個 type code，它會影響 class 的行為，但無法使用 subclassing。

以 state object 取代 type code。

本項重構使用 State 範式或 Strategy 範式。

如果完成本項重構將繼續採取 **Replace Conditional with Polymorphism (255)**，那麼 **Strategy** 範式比較合適。



作法：

1. 使用 **Self Encapsulate Field (171)**將 type code 自我封裝起來。
2. 新建一個 class，根據 type code 的用途為它命名。(此即為 **state object**)。
3. 為此新建的 class 添加 subclasses，每個 subclass 對應一種 type code。
4. superclass 中建立一個抽象的查詢函式(query)，用以傳回 type code，

   並交由 subclass 進行覆寫。

5. 在 source class 中建立欄位，以保存新建的 state object。
6. 調整 source class 中負責查詢 type code 的函式，將查詢動作轉發給 state object。
7. 調整 source class 中「為 type code 設值」的函式，將一個恰當的 state object subclass 賦值給「保存 state object」的欄位。

   (可進一步採用 **Replace Conditional with Polymorphism (255)**)。

Primitive Obsession

基本型別偏執

(81) - 3/3

Introduce Parameter Object (引入參數物件) (295)

某些參數總是很自然地同時出現。

以一個**不可變物件**(immutable object)取代這些參數。並觀察是否可採行 **Move Method (142)**或 **Extract Method (110)**。

Switch Statements

Switch 述句

(82) – 1/2

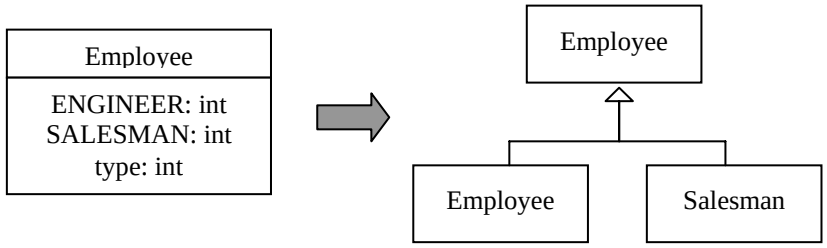
Replace Type Code with Subclasses (以子類別取代型別代碼) (223)

你有一個**不可變的** type code，它**會影響** class 的行為。

以一個 subclass 取代這個 type code。

當 **type code 值會在物件創建後發生改變** 或 **type code 宿主類別已經有了 subclasses**，那麼就不是使用本項重構，而是 **Replace Type Code with State/Strategy (227)**。

如果**宿主類別沒有出現條件式**，那麼 **Replace Type Code with Class (218)** 更合適。



作法：

1. 使用 **Self Encapsulate Field (171)**將 type code 自我封裝起來。
   如果 type code 被傳遞給建構式，需將建構式轉換為 factory method。
2. 為 type code 的每個數值建立相應的 subclass。在每個 subclass 中覆寫 type code 的**取值函式**(getter)，使其傳回相應的值。
3. super class 則將 type code 存取函式宣告為**抽象函式**。

(可進一步採用 **Replace Conditional with Polymorphism (255)**)。

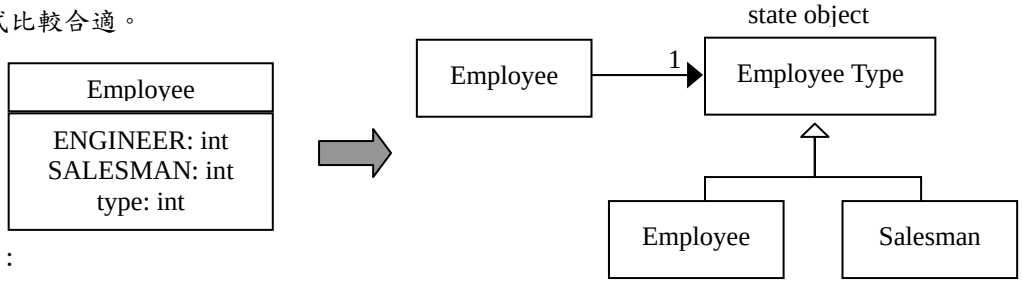
Replace Type Code with State/Strategy (以 State/Strategy 取代型別代碼) (227)

你有一個 type code，它會影響 class 的行為，但無法使用 subclassing。

以 state object 取代 type code。

本項重構使用 State 範式或 Strategy 範式。

如果完成本項重構將繼續採取 **Replace Conditional with Polymorphism (255)**，那麼 **Strategy** 範式比較合適。



作法：

1. 使用 **Self Encapsulate Field (171)**將 type code 自我封裝起來。
2. 新建一個 class，根據 type code 的用途為它命名。(此即為 **state object**)。
3. 為此新建的 class 添加 subclasses，每個 subclass 對應一種 type code。
4. superclass 中建立一個抽像的查詢函式(query)，用以傳回 type code，並交由 subclass 進行覆寫。
5. 在 source class 中建立欄位，以保存新建的 state object。
6. 調整 source class 中負責查詢 type code 的函式，將查詢動作轉發給 state object。
7. 調整 source class 中「為 type code 設值」的函式，將一個恰當的 state object subclass 賦值給「保存 state object」的欄位。

(可進一步採用 **Replace Conditional with Polymorphism (255)**)。

Switch Statements **Switch 述句** **(82) - 2/2**	**Replace Conditional with Polymorphism (以多型取代條件式) (255)** 你手上有個條件式，它根據物件型別的不同而選擇不同的行為。 將這些條件式的每個分支放進一個 subclass 內的**覆寫函式**中，然後將原始函式宣告為抽象函式。必需要有一個繼承結構，若沒有，可從 **Replace Type Code with Subclasses (223)** 或 **Replace Type Code with State/Strategy (227)**建立。 **Introduce Null Object (引入 Null 物件) (260)** 你需要再三檢查「某物件是否為 null value」。 將 null value (無效值) 替換為 null object (無效物)。null objects 一定是常數，且它的任何成分都不會發生變化，故可使用 **Singleton** 範式來實現。null object 可以藉由以下兩種方式實現： 1. **subclass** (必需提供 isNull()函式) 2. **interface** (可提供 isNull()函式 或 運用 instanceof 運算子檢查)
Parallel Inheritance Hierarchies **平行繼承體系** **(83)**	某個繼承體系的class名稱字首和另一個繼承體系的class名稱字首完全相同。每當為某個class增加一個subclass，也必需為另一個class相應增加一個subclass。 解決：讓一個繼承體系的實體引用另一個繼承體系的實體。 **Move Method (搬移函式) (142)** 程式中，有個函式與其所駐 class 之外的另一個 class 進行更多交流：呼叫後者，或被後者呼叫。 在該函式最常引用的 class 中建立一個有著類似行為的新函式。將舊函式變成一個單純的**委託函式**，或是將舊函式**完全移除**。當需要使用 source class 特性： 1. 將此特性也移至 target class。 2. 建立或使用一個從 target class 到 source 的引用關係。 3. 將 source object 當作參數傳給 target method。 4. 若所需特性是個變數，將它當作參數傳給 target method。 **Move Field (搬移欄位) (146)** 程式中，某個欄位被其所駐 class 之外的另一個 class 更多地用到。 在 target class 建立一個 new field，修改 source field 的所有用戶，令它們改用 new field。 (所謂「使用」動作可能是通過設值/取值函式間接進行)。
Lazy Class **懶惰的類別** **(83)**	**Inline Class (將類別內聯化) (154)** 某個 class 沒有作太多事情(沒有承擔足夠責任)。 將 class 的所有特性搬移到另一個 class 中，然後移除原 class。 如果「以一個獨立介面表示 source class 函式」更合適的話，先使用 **Extract Interface (341)**。 **Collapse Hierarchy (折疊繼承體系) (344)** superclass 和 subclass 之間無太大區別。 將它們合為一體。

Speculative Generality **夸夸其談未來性** **(83)**	**Inline Class (將類別內聯化) (154)** 某個 class 沒有作太多事情(沒有承擔足夠責任)。 將 class 的所有特性搬移到另一個 class 中，然後移除原 class。 如果「以一個獨立介面表示 source class 函式」更合適的話，先使用 **Extract Interface (341)**。 **Rename Method (重新命名函式) (273)** 函式的名稱未能揭示函式的用途。 修改函式名稱。可先考慮應該給這個函式寫上一句怎樣的註釋，然後再將註釋變成函式名稱。 **Remove Parameter (移除參數) (277)** 函式本體不再需要某個參數。 將該參數去除。 **Collapse Hierarchy (折疊繼承體系) (344)** superclass 和 subclass 之間無太大區別。 將它們合為一體。
Temporary Field **暫時欄位** **(84)**	某個instance變數只為某種特殊情勢而設。 **Extract Class (提煉類別) (149)** 某個 class 作了應由兩個 classes 作的事。 建立一個新 class，將相關欄位和函式從舊 class 搬移到新 class。並決定**是否讓新 class 曝光**： 1. reference object。 2. immutable object 或 immutable interface。 3. 保護性拷貝。 (**委託，class 可以數種不同方式變化**，對比 **Extract Subclass (330)**之繼承)。 **Introduce Null Object (引入 Null 物件) (260)** 你需要再三檢查「某物件是否為 null value」。 將 null value (無效值) 替換為 null object (無效物)。null objects 一定是常數，且它的任何成分都不會發生變化，故可使用 **Singleton** 範式來實現。null object 可以藉由以下兩種方式實現： 1. **subclass** (必需提供 isNull()函式) 2. **interface** (可提供 isNull()函式 或 運用 instanceof 運算子檢查)
Message Chains **過度耦合的訊息鍊** **(84)**	程式碼中你看到的可能是一長串getXXX()或一長串暫時變數。 **Hide Delegate (隱藏「委託關係」) (157)** 客戶直接呼叫其 server object(服務物件)的 delegate class。 在 server 端(某個 class)建立客戶所需的所有函式，用以隱藏委託關係(delegation)。 原則： 封裝意味每個物件都應盡可能減少瞭解系統的其他部分，一旦發生變化，需要瞭解這一變化的物件就會比較少。

Middle Man

中間轉手人

(85)

某個class介面有一半的函式都委託給其他class。

Inline Method (將函式內聯化) (117)

一個函式，其本體應該與其名稱同樣清楚易懂。

在函式呼叫點插入函式本體，然後移除函式。

(若 subclass 覆寫了這個函式，便無法使用此項重構)。

Remove Middle Man (移除中間人) (160)

某個 class 作了過多的簡單委託動作(simple delegation)。

讓客戶**直接呼叫 delegate(受託類別)**。

Replace Delegation with Inheritance (以繼承取代委託) (355)

在兩個 classes 之間使用委託關係，並經常為整個介面編寫許多極簡單的請託函式。

讓「**請託**(delegating) class」**繼承**「**受託**(delegate) class」。

也可考慮 **Remove Middle Man (160)**、**Extract Superclass (336)**或 **Extract Interface (341)**。

若沒有使用「受託 class」的**所有**函式，或者「**受託物件被一個以上的其他物件共享，且受託物件是可變的**(mutable)」，就不該使用此項重構。

Inappropriate Intimacy

狹暱關係

(85) – 1/2

兩個classes過於親密，花費太多時間探究彼此的private成分。

Move Method (搬移函式) (142)

程式中，有個函式與其所駐 class 之外的另一個 class 進行更多交流：呼叫後者，或被後者呼叫。

在該函式最常引用的 class 中建立一個有著類似行為的新函式。將舊函式變成一個單純的**委託函式**，或是將舊函式**完全移除**。當需要使用 source class 特性：

1. 將此特性也移至 target class。
2. 建立或使用一個從 target class 到 source 的引用關係。
3. 將 source object 當作參數傳給 target method。
4. 若所需特性是個變數，將它當作參數傳給 target method。

Move Field (搬移欄位) (146)

程式中，某個欄位被其所駐 class 之外的另一個 class 更多地用到。

在 target class 建立一個 new field，修改 source field 的所有用戶，令它們改用 new field。

(所謂「使用」動作可能是通過設值/取值函式間接進行)。

Hide Delegate (隱藏「委託關係」) (157)

客戶直接呼叫其 server object(服務物件)的 delegate class。

在 server 端(某個 class)建立客戶所需的所有函式，用以隱藏委託關係(delegation)。

原則：

封裝意味每個物件都應盡可能減少瞭解系統的其他部分，一旦發生變化，需要瞭解這一變化的物件就會比較少。

Inappropriate Intimacy

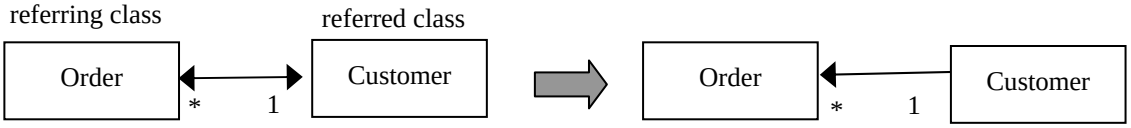
狹暱關係

(85) – 1/2

Change Bidirectional Association to Unidirectional (將雙向關聯改為單向) (200)

兩個 classes 之間有雙向關聯，但其中一個 class 如今不再需要另一個 class 的特性。

去除不必要的關聯。(大量的雙向關連容易引發「僵屍物件」，也造成高耦合)



對於「指標保存欄位」刪除後，相關函式的處理，可採取：

1. 將「被引用物件」(referred object)作為**參數**傳入。
2. 對**取值函式**(getter)使用 **Substitute Algorithm (139)**。

Replace Inheritance with Delegation (以委託取代繼承) (352)

某個 subclass 只用 superclass 介面中的一部份，或是根本不需要繼承而來的資料。

在 subclass 中**新建一個欄位用以保存 superclass**；調整 subclass 函式，令它改而委託 superclass；然後去掉兩者之間的繼承關係。(如此能更清楚表明只需要受委託類別的部分功能)。

Alternative Classes with Different Interfaces

異曲同工的類別

(85)

Move Method (搬移函式) (142)

程式中，有個函式與其所駐 class 之外的另一個 class 進行更多交流：呼叫後者，或被後者呼叫。

在該函式最常引用的 class 中建立一個有著類似行為的新函式。將舊函式變成一個單純的**委託函式**，或是將舊函式**完全移除**。當需要使用 source class 特性：

1. 將此特性也移至 target class。
2. 建立或使用一個從 target class 到 source 的引用關係。
3. 將 source object 當作參數傳給 target method。
4. 若所需特性是個變數，將它當作參數傳給 target method。

Rename Method (重新命名函式) (273)

函式的名稱未能揭示函式的用途。

修改函式名稱。可先考慮應該給這個函式寫上一句怎樣的註釋，然後再將註釋變成函式名稱。

Incomplete Library Class

不完美的程式庫類別

(86)

Introduce Foreign Method (引入外加函式) (162)

所用的 server class 需要**一個**額外函式，但無法修改這個 class。

在 client class 中建立一個函式，並**以一個 server class 實體作為第一引數(argument)**。

若為 server class 建立了大量外加函式，或發現許多 classes 都需要同樣的外加函式，就不該使用本向重構，而是使用 **Local Extension (164)**。

Introduce Local Extension (引入區域性擴展) (164)

你所使用的 server class 需要**一些**額外函式，但妳無法修改這個 class。

建立一個新 class，使它包含這些額外函式。讓這個擴展品成為 source class 的 **subclass** 或 **wrapper**。並在 extension class 中加入轉型建構式(接受原物作為參數)。(equality 議題請參照書本內容)。

Data Class **純稚的資料類別** **(86)**	Data Class：它們擁有一些欄位，以及用於存取這些欄位的函式，除此之外一無長物。 **Move Method (搬移函式) (142)** 程式中，有個函式與其所駐 class 之外的另一個 class 進行更多交流：呼叫後者，或被後者呼叫。 在該函式最常引用的 class 中建立一個有著類似行為的新函式。將舊函式變成一個單純的**委託函式**，或是將舊函式**完全移除**。當需要使用 source class 特性： 1. 將此特性也移至 target class。 2. 建立或使用一個從 target class 到 source 的引用關係。 3. 將 source object 當作參數傳給 target method。 4. 若所需特性是個變數，將它當作參數傳給 target method。 **Encapsulate Field (封裝欄位) (206)** 你的 class 中存在一個 public 欄位。 將它宣告為 private，並提供相應的存取函式。 **Encapsulate Collection (封裝群集) (208)** 有個函式傳回一個群集。 讓函式傳回該群集的一個**唯讀映件**(read-only-view)。Ex: Collection.unmodifiableXxx() 並在此 class 中提供「**添加/移除**」(add/remove)群集元素的函式。另外，不該為整個群集提供設值函式。
Refused Bequest **被拒絕的遺贈** **(87)**	如果subclass復用了superclass的行為，卻不願意支援superclass的介面。 **Replace Inheritance with Delegation (以委託取代繼承) (352)** 某個 subclass 只用 superclass 介面中的一部份，或是根本不需要繼承而來的資料。 在 subclass 中**新建一個欄位用以保存 superclass**；調整 subclass 函式，令它改而委託 superclass；然後去掉兩者之間的繼承關係。(如此能更清楚表明只需要受委託類別的部分功能)。
Comments **過多的註釋** **(87)**	Comments可以帶我們找到各種壞味道。 **Extract Method (提煉函式) (110)** 函式名稱與函式本體的語意距離有所落差。 由函式本體獨立出新函式，並給予可解釋用途之名稱。實作類型： 1. 無區域變數。 2. 有區域變數：將區域變數作為**參數**。 3. 對區域變數在賦值：使用 **Remove Assignments to Parameters (131)**。 **Introduce Assertion (引入斷言) (267)** 某一段程式碼需要對程式狀態(state)做出某種假設。 以 assertion (斷言)明確表現這種假設。 assertion 條件式應該總是為真，但請**只使用它來檢查「一定必需為真」的條件**，否則將造成難以維護的重覆邏輯。

重構名錄：

重新組織你的函式 (Composing Methods)

Extract Method (提煉函式) (110)

函式名稱與函式本體的語意距離有所落差。

由函式本體獨立出新函式，並給予可解釋用途之名稱。實作類型：

1. 無區域變數。
2. 有區域變數：將區域變數作為**參數**。
3. 對區域變數在賦值：使用 **Remove Assignments to Parameters (131)**。

Inline Method (將函式內聯化) (117)

一個函式，其本體應該與其名稱同樣清楚易懂。

在函式呼叫點插入函式本體，然後移除函式。(若 subclass 覆寫了這個函式，便無法使用此項重構)。

Inline Temp (將暫時變數內聯化) (119)

有一個暫時變數，只被一個簡單運算式賦值一次，而他妨礙了其他重構手法。

將所有對該變數的引用動作，替換為對它賦值的那個運算式本身。

Replace Temp with Query (以查詢取代暫時變數) (120)

程式以一個暫時變數保存某一運算式的運算結果。

將此運算式獨立到一個**查詢函式**(query，**不得修改任何物件內容**)中。將此暫時變數的所有「被引用點」替換為「對新函式的呼叫」。新函式可被其他函式使用。

該函式不修改任何物件內容，否則需採取 **Separate Query from Modifier (279)**。

若暫時變數被賦值超過一次，考慮使用 **Split Temporary Variable (128)**。

Introduce Explaining Variable (引入解釋性變數) (124)

你有一個複雜的運算式。

將該複雜運算式(或其中一部份)的結果放進一個暫時變數，以此變數名稱來解釋運算式用途。

建議優先考慮 **Extract Method (110)**來解釋程式碼的意義，除非重構工作需要較大的工作量，再改以此項重構。

Split Temporary Variable (解剖暫時變數) (128)

程式中某個暫時變數被賦值**超過一次**，既不是迴圈變數，也不是一個集用暫時變數(collecting temporary variable)。

針對每次賦值，創造一個獨立的、對應的暫時變數。(原則：**每個變數只承擔一個責任**)

Remove Assignments to Parameters (移除對參數的賦值動作) (131)

你的程式碼對一個參數進行賦值動作。

以一個**暫時變數**取代該參數的位置。

Replace Method with Method Object (以函式物件取代函式) (135)

函式內，區域變數的使用導致無法採行 Extract Method (110)。

將此函式放入一個單獨物件(以用途名之)中，**區域變數成為新物件的欄位(field)**，**建構式則保留原函式之參數**，並新增 **compute()函式**負責原始操作。之後便可採取 **Extract Method (110)**。最後，舊函式的本體得以將工作**轉發**給此函式物件。Ex: return new Gamma(123,3).compute();

Substitute Algorithm (替換你的演算法) (139)

想要把某個演算法替換為另一個更清晰的演算法。

將函式本體替換為另一個演算法。

在物件之間移動特性 (Moving Features Between Objects)

Move Method (搬移函式) (142)

程式中，有個函式與其所駐 class 之外的另一個 class 進行更多交流：呼叫後者，或被後者呼叫。

在該函式最常引用的 class 中建立一個有著類似行為的新函式。將舊函式變成一個單純的**委託函式**，或是將舊函式**完全移除**。當需要使用 source class 特性：

1. 將此特性也移至 target class。
2. 建立或使用一個從 target class 到 source 的引用關係。
3. 將 source object 當作參數傳給 target method。
4. 若所需特性是個變數，將它當作參數傳給 target method。

Move Field (搬移欄位) (146)

程式中，某個欄位被其所駐 class 之外的另一個 class 更多地用到。

在 target class 建立一個 new field，修改 source field 的所有用戶，令它們改用 new field。

(所謂「使用」動作可能是通過設值/取值函式間接進行)。

Extract Class (提煉類別) (149)

某個 class 作了應由兩個 classes 作的事。

建立一個新 class，將相關欄位和函式從舊 class 搬移到新 class。並決定**是否讓新 class 曝光**：

1. reference object。
2. immutable object 或 immutable interface。
3. 保護性拷貝。

(**委託，class 可以數種不同方式變化**，對比 **Extract Subclass (330)**之繼承)。

Inline Class (將類別內聯化) (154)

某個 class 沒有作太多事情(沒有承擔足夠責任)。

將 class 的所有特性搬移到另一個 class 中，然後移除原 class。

如果「以一個獨立介面表示 source class 函式」更合適的話，先使用 **Extract Interface (341)**。

Hide Delegate (隱藏「委託關係」) (157)

客戶直接呼叫其 server object(服務物件)的 delegate class。

在 server 端(某個 class)建立客戶所需的所有函式，用以隱藏委託關係(delegation)。

原則：

封裝意味每個物件都應盡可能減少瞭解系統的其他部分，一旦發生變化，需要瞭解這一變化的物件就會比較少。

Remove Middle Man (移除中間人) (160)

某個 class 作了過多的簡單委託動作(simple delegation)。

讓客戶**直接呼叫 delegate(受託類別)**。

Introduce Foreign Method (引入外加函式) (162)

所用的 server class 需要**一個**額外函式，但無法修改這個 class。

在 client class 中建立一個函式，並**以一個 server class 實體作為第一引數(argument)**。

若為 server class 建立了大量外加函式，或發現許多 classes 都需要同樣的外加函式，就不該使用本向重構，而是使用 **Local Extension (164)**。

Introduce Local Extension (引入區域性擴展) (164)

你所使用的 server class 需要**一些**額外函式，但妳無法修改這個 class。

建立一個新 class，使它包含這些額外函式。讓這個擴展品成為 source class 的 **subclass** 或 **wrapper**。並在 extension class 中加入轉型建構式(接受原物作為參數)。(equality 議題請參照書本內容)。

重新組織你的資料 (Organizing Data)

Self Encapsulate Field (自我封裝欄位) (171)

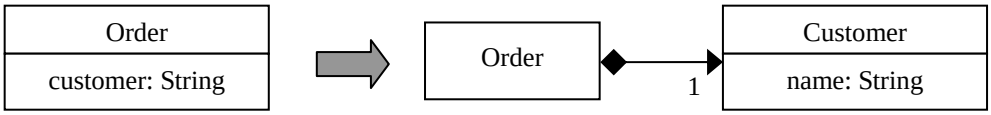
直接存取一個欄位，但與欄位之間的耦合關係逐漸變得笨拙。

為此欄位建立 **取值／設值函式**，並且只以這些欄位來存取。

Replace Data Value with Object (以物件取代資料值) (175)

你有一筆資料項(data item)，需要額外的資料和行為。

將這筆資料項變成一個**不可變的**(immutable)**實值物件**(Value Object)(**組合關係 composition**)。



作法：

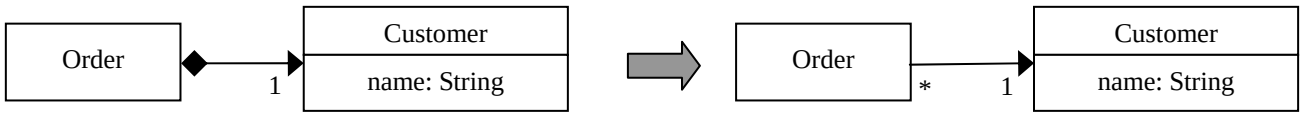
1. 新建一個 class，其中宣告型別與「代替換數值」相同的 **final 欄位**，並給定相應的**取值函式**(getter)及以其為參數的**建構式**，而 source class 的取值函式將委託新 class。
2. source class 中「代替換數值欄位」的型別換為新 class。
3. source class 中，代替換欄位的**設值函式**(setter)，令它為新 class 創建一個實體。

(可能進一步使用 **Change Value to Reference (179)**)。

Change Value to Reference (將實值物件改為引用物件) (179)

你有一個 class，衍生出許多相等實體，希望將它們替換為單一物件。

將這個**不可變的實值物件**(value object)變成一個**可變的引用物件**(reference object)。



作法：

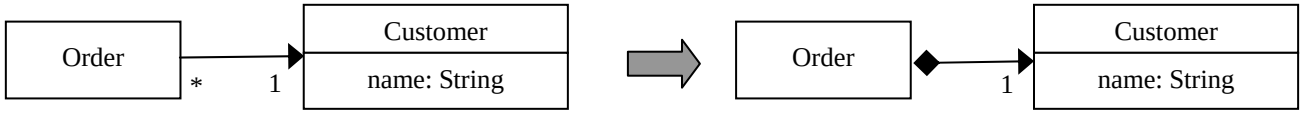
1. 使用 **Replace Constructor with Factory Method (304)**。
2. 決定由什麼物件負責提供**存取新物件**的途徑：

   靜態字典(static dictionary)或**註冊物件**(registry object)。 Ex: private static Dictionary dic = new Hashtable();
3. 決定這些 reference object 是**預先創建** 或 **動態創建**。
4. 修改 **factory method** (比 GoF 定義的還廣義)，令他傳回 reference object。

Change Reference to Value (將引用物件改為實值物件) (183)

你有一個引用物件(reference object)，很小且不可變(immutable)，而且不易管理。

將它變成一個**不可變的實值物件**(value object)。



作法：

1. 檢查**是否為不可變物件**，若否但可修改，先使用 **Remove Setting Method (300)**，否則不可使用本項重構。
2. 建立 **equals()** 和 **hashCode()** (可讀取 equals()使用之所有欄位的 hash codes 然後進行 bitwise xor ^ 操作)。
3. 考慮是否可**刪除 factory method**，並將建構式宣告為 public。

Replace Array with Object (以物件取代陣列) (186)

你有一陣列，其中的元素各自代表不同的東西。

以物件替換陣列。對於陣列中的每個元素，以一個欄位表示之。並逐一對元素添加**取值/設值**函式。

Duplicate Observed Data (複製「被監視資料」) (189)

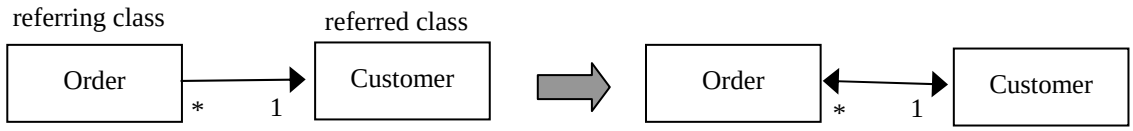
你有一些 domain data (業務邏輯)置身於 GUI 控件中，而 domain method 需要存取之。

將該筆資料拷貝到一個 domain object 中。建立一個 **Observer** 範式，用以對 domain object 和 GUI object 內的重覆資料進行同步控制。(操作步驟較複雜，請參照書本內容)(亦有使用事件監聽器的實作)。

Change Unidirectional Association to Bidirectional (將單向關聯改為雙向) (197)

兩個 classes 都需要使用對方特性，但其間只有一條單向連結。

添加一個反向指標，並使修改函式能夠同時更新兩條連結。



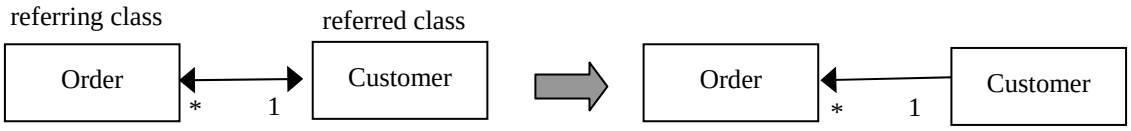
作法：

1. 在被引用端(rederred)增加新欄位，以保存反向指標。
2. 決定由那個 class 控制關連性：
   a. 若是「一對多」關係，由**前者**擔任控制者角色。
   b. 若是「多對多」關係，**皆可**當控制者。
   c. 若某物是另一物件的組成(component)，由**後者**作為控制者。
3. 在「被控端」建立輔助函式，負責**對指標兩端物件進行同步控制**。
4. 如果修改函式在「控制端」，讓它負責更新反向指標。
5. 如果修改函式在「被控端」，在「控制端」建立一個控制函式，以讓既有的修改函式呼叫。

Change Bidirectional Association to Unidirectional (將雙向關聯改為單向) (200)

兩個 classes 之間有雙向關聯，但其中一個 class 如今不再需要另一個 class 的特性。

去除不必要的關聯。(大量的雙向關連容易引發「僵屍物件」，也造成高耦合)



對於「指標保存欄位」刪除後，相關函式的處理，可採取：

1. 將「被引用物件」(referred object)作為**參數**傳入。
2. 對**取值函式**(getter)使用 **Substitute Algorithm (139)**。

Replace Magic Number with Symbolic Constant (以符號常數／字面常數 取代魔術數字) (204)

你有一個字面數值，帶有特別含意。

創造一個**常數**，根據其意義為它命名，並將上述的字面數值替換為這個常數。

若是 type code，請使用 **Replace Type Code with Class (218)**。

Encapsulate Field (封裝欄位) (206)

你的 class 中存在一個 public 欄位。

將它宣告為 private，並提供相應的存取函式。

Encapsulate Collection (封裝群集) (208)

有個函式傳回一個群集。

讓函式傳回該群集的一個**唯讀映件**(read-only-view)。Ex: Collection.unmodifiableXxx()

並在此 class 中提供「**添加/移除**」(add/remove)群集元素的函式。另外，不該為整個群集提供設值函式。

Replace Record with Data Class (以資料類別取代記錄) (217)

你需要面對傳統編程環境中的 record structure(記錄結構)。

為該 record 創建一個「啞」資料物件(只含資料及存取函式)，以便進一步的重構。

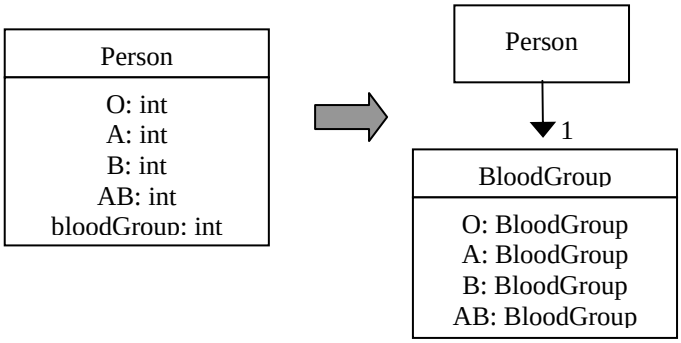
Replace Type Code with Class (以類別取代型別代碼) (218)

Class 之中有一個**數值**型別代碼(numeric type code)，但它並**不影響** class 的行為。

以一個新的 class 替換數值型別代碼(type code)。

原則：只有當 type code 是**純資料**時，才能以 class 來取代它。

否則應採用 **Replace Type Code with Subclasses (223)** 或 **Replace Type Code with State/Strategy (227)**。



作法：

1. 為 type code 建立一個 class。

   包含：記錄 type code 的**欄位**(static，且型別不變)、對應的**取值函式**(getter，亦為 static)。

2. 為 source class 提供相應的 **取值/設值**函式 及 **建構子**，都將使用新的 class。

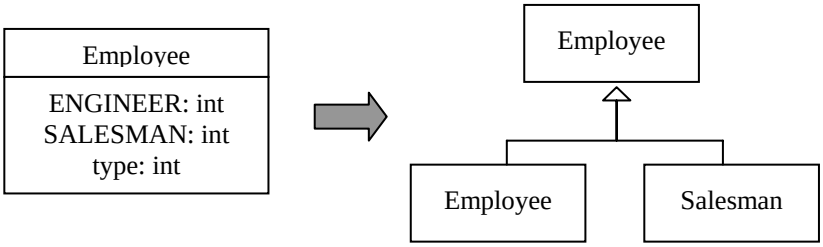
Replace Type Code with Subclasses (以子類別取代型別代碼) (223)

你有一個**不可變的** type code，它**會影響** class 的行為。

以一個 subclass 取代這個 type code。

當 **type code 值會在物件創建後發生改變** 或 **type code 宿主類別已經有了 subclasses**，那麼就不是使用本項重構，而是 **Replace Type Code with State/Strategy (227)**。

如果**宿主類別沒有出現條件式**，那麼 **Replace Type Code with Class (218)** 更合適。



作法：

1. 使用 **Self Encapsulate Field (171)**將 type code 自我封裝起來。

   如果 type code 被傳遞給建構式，需將建構式轉換為 factory method。

2. 為 type code 的每個數值建立相應的 subclass。在每個 subclass 中覆寫 type code 的**取值函式**(getter)，使其傳回相應的值。

3. super class 則將 type code 存取函式宣告為**抽象函式**。

(可進一步採用 **Replace Conditional with Polymorphism (255)**)。

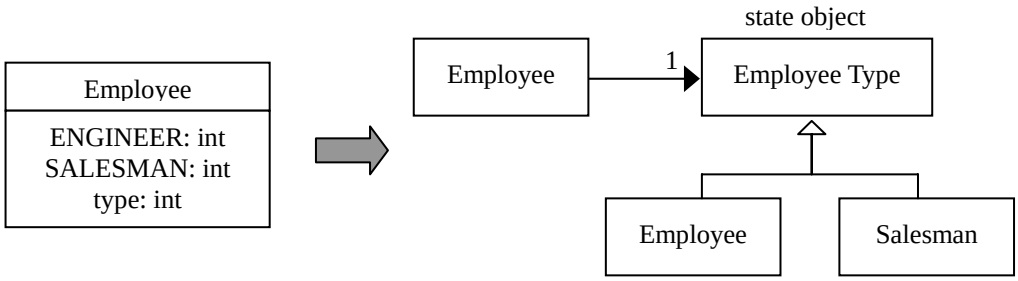
Replace Type Code with State/Strategy (以 State/Strategy 取代型別代碼) (227)

你有一個 type code，它會影響 class 的行為，但無法使用 subclassing。

以 state object 取代 type code。

本項重構使用 State 範式或 Strategy 範式。

如果完成本項重構將繼續採取 **Replace Conditional with Polymorphism (255)**，那麼 **Strategy** 範式比較合適。



作法：

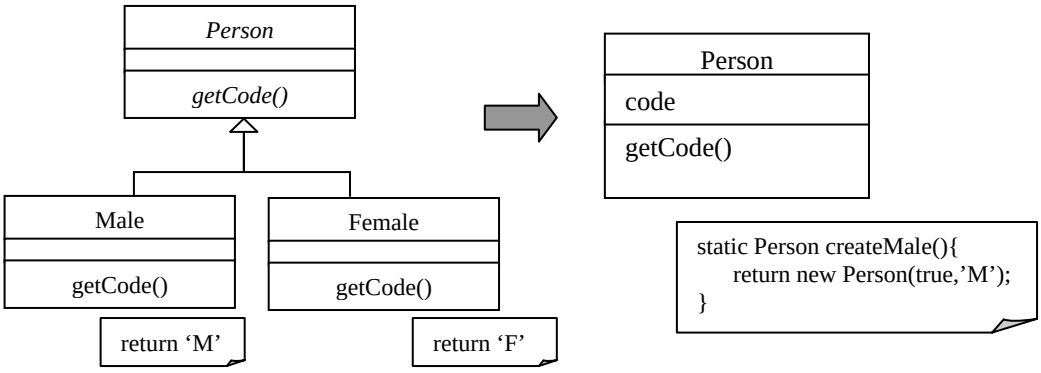
1. 使用 **Self Encapsulate Field (171)**將 type code 自我封裝起來。
2. 新建一個 class，根據 type code 的用途為它命名。(此即為 **state object**)。
3. 為此新建的 class 添加 subclasses，每個 subclass 對應一種 type code。
4. superclass 中建立一個抽像的查詢函式(query)，用以傳回 type code，並交由 subclass 進行覆寫。
5. 在 source class 中建立欄位，以保存新建的 state object。
6. 調整 source class 中負責查詢 type code 的函式，將查詢動作轉發給 state object。
7. 調整 source class 中「為 type code 設值」的函式，將一個恰當的 state object subclass 賦值給「保存 state object」的欄位。

   (可進一步採用 **Replace Conditional with Polymorphism (255)**)。

Replace Subclass with Fields (以欄位取代子類別) (232)

各個 subclasses 的唯一差別只在「傳回常數資料」的函式身上。

修改這些函式，使它們傳回 superclass 中的某個(新增)欄位，然後銷毀 subclasses。



作法：

1. 在 superclass 中對所有 subclasses 使用 **Replace Constructor with Factory Method (304)**。
2. 若有任何程式碼直接引用 subclass，令他改用 superclass。
3. 針對每個常數函式，在 superclass 中宣告一個 **final 欄位**。
4. 為 superclass 宣告一個 **protected 建構式**，用來初始化這些欄位。
5. 使用 Inline Method (117) 將 subclass 建構式內聯(inlining)到 superclass 的 **factory mehtod** 中。

簡化條件式 (Simplifying Conditional Expressions)

Decompose Conditional (分解條件式) (238)

有一個複雜的條件句 (if-then-else)。

先觀察是否可使用 **Replace Nested Conditional with Guard Clauses (250)**。若否，從 if、then、else 三個段落中分別提煉出**獨立函式**。

Consolidate Conditional Expression (合併條件式) (240)

你有一系列條件測試，都得到相同結果。

將這些測試合併為一個條件式，並將其提煉成為**一個獨立函式**。若這些條件彼此獨立，不應被視為同一次檢查，就不要使用本項重構。

Consolidate Duplicate Conditional Fragments (合併重複的條件片段) (243)

在條件式的每個分支尚有著相同的一段程式碼。

將這段重複程式碼搬移到條件式之外。

Remove Control Flag (移除控制旗標) (245)

在一系列布林運算式中，某個變數帶有「控制旗標」的作用。

以 **break** 述句、**continue** 述句或 **return** 述句取代控制旗標。

Replace Nested Conditional with Guard Clauses (以衛述句取代巢狀條件式) (250)

函式中的條件邏輯使人難以看清正常的執行路徑。

使用**衛述句**表現所有特殊情況，如果某個條件極其罕見，就應單獨檢查該條件，並在**該條件為真**時立刻從函式回返。(常可以將條件運算式逆反，從而實現本項重構)。

Replace Conditional with Polymorphism (以多型取代條件式) (255)

你手上有個條件式，它根據物件型別的不同而選擇不同的行為。

將這些條件式的每個分支放進一個 subclass 內的**覆寫函式**中，然後將原始函式宣告為抽象函式。必需要有一個繼承結構，若沒有，可從 **Replace Type Code with Subclasses (223)** 或 **Replace Type Code with State/Strategy (227)**建立。

Introduce Null Object (引入 Null 物件) (260)

你需要再三檢查「某物件是否為 null value」。

將 null value (無效值) 替換為 null object (無效物)。null objects 一定是常數，且它的任何成分都不會發生變化，故可使用 **Singleton** 範式來實現。null object 可以藉由以下兩種方式實現：

1. **subclass** (必需提供 isNull()函式)
2. **interface** (可提供 isNull()函式 或 運用 instanceof 運算子檢查)

Introduce Assertion (引入斷言) (267)

某一段程式碼需要對程式狀態(state)做出某種假設。

以 assertion (斷言)明確表現這種假設。

assertion 條件式應該總是為真，但請**只使用它來檢查「一定必需為真」的條件**，否則將造成難以維護的重覆邏輯。

簡化函式呼叫 (Making Method Calls Simpler)

Rename Method (重新命名函式) (273)

函式的名稱未能揭示函式的用途。

修改函式名稱。可先考慮應該給這個函式寫上一句怎樣的註釋，然後再將註釋變成函式名稱。

Add Parameter (添加參數) (275)

某個函式需要從呼叫端得到更多資訊。

為此函式**添加一個物件參數**，讓該物件帶進函式所需資訊。但可先考慮使用 Introduce Parameter Object (295)。

Remove Parameter (移除參數) (277)

函式本體不再需要某個參數。

將該參數去除。

Separate Query from Modifier (將查詢函式和修改函式分離) (279)

某個函式既傳回物件狀態值，又修改物件狀態(state)。

建立兩個不同的函式，將**查詢、修改責任獨立**。(任何有回返值的函式，都不應該有看得到的副作用)

Parameterize Method (令函式攜帶參數) (283)

若干函式作了類似的工作，但在函式本體中卻包含了不同的值。

建立單一函式，以參數表達那些不同的值。

Replace Parameter with Explicit Methods (以明確函式取代參數) (285)

你有一函式，其內完全取決於參數值而採取不同反應。

針對**該參數的每一個可能值，建立一個函式**。

Preserve Whole Object (保持物件完整) (288)

你從某個物件取出若干值，將它們作為某一次函式呼叫時的參數。

改傳遞**整個物件**。**(**如果將使依存結構惡化，就不該使用！但若物件有合適的取值函式，便不需操心依存問題)。

Replace Parameter with Methods (以函式取代參數) (292)

物件喚起某函式，並將所得結果作為參數，傳遞給另一個函式。而接受該參數的函式也可以喚起前一個函式。

讓參數接受者去除該項參數，並**直接呼叫前一個函式**。(前提：呼叫端通過「其所屬物件內部的另一個函式」來計算參數,並在計算過程中「未曾引用呼叫端其他參數」,否則將需使用 Replace Parameter with Explicit Methods (285))。

Introduce Parameter Object (引入參數物件) (295)

某些參數總是很自然地同時出現。

以一個**不可變物件**(immutable object)取代這些參數。並觀察是否可採行 **Move Method (142)**或 **Extract Method (110)**。

Remove Setting Method (移除設值函式) (300)

class 中的某個欄位，應該在物件初創時被設值，然後就不再改變。

去掉該欄位的所有**設值函式**(setter)。

Hide Method (隱藏某個函式) (303)

函式從來沒有被其他任何 class 用到。

將這個函式修改為 **private**。(當面對一個過於豐富、提供了過多行為的介面時，就值得將**取值、設值函式隱藏**起來)。

Replace Constructor with Factory Method (以「工廠函式」取代「建構式」) (304)

你希望在創建物件時，不僅僅是對它作簡單的建構動作。

將 constructor 替換為 factory method。(常在 subclassing 過程中以 factory method **取代 type code**，也是 **Change Value to Reference (179)**的基礎)。實作類型：

1. 根據**整數**(實際為 type code)創建物件。
2. 根據**字串**創建 subclass 物件：使用 **switch 述句** 或 (SuperclassName)**Class.forName**(SubclassName).newInstance()
3. 以**明確函式**創建 subclass。

Encapsulate Downcast (封裝「向下轉型」動作) (308)

某個函式傳回的物件，需要由函式呼叫者執行「向下轉型」動作。

將向下轉型動作移到函式中。(常在「傳回迭代器或群集」的函式身上發生)。

Replace Error Code with Exception (以異常取代錯誤碼) (310)

某個函式傳回一個特殊代碼，用以表示某種錯誤情況。

改用異常(Exception)。

需決定待拋異常是 **checked** (函式自行檢查)還是 **unchecked** (使用者檢查，RuntimeException)。

Replace Exception with Test (以測試取代異常) (315)

面對一個「呼叫者可預先加以檢查」的條件，你拋出了一個異常。

修改呼叫者，使他在呼叫函式之前先作檢查。

(異常**只應用於「異常的、罕見的行為」**，而不應成為「條件檢查」的替代品)。

處理概括關係 (Dealing with Generalization)

Pull Up Field (欄位上移) (320)

兩個 subclasses 擁有相同的欄位。

將此一欄位移至 superclass。

Pull Up Method (函式上移) (322)

有些函式，在各個 subclasses 中產生完全相同的結果。

將該函式移至 superclass：

1. 若被提升的函式引用了「只出現於 subclass 而不出現於 superclass」的函式(或特性)，可**將該函式也提升至 superclass**，或者**在 superclass 中建立一個抽象函式**。
2. 若兩個函式相似但不相同，可先以 **Form Template Method (345)**構造出相同的函式。

Pull Up Constructor Body (建構式本體上移) (325)

在各個 subclass 中擁有一些建構式，它們的本體(程式碼)幾乎完全一致。

在 superclass 中新建一個建構式，並**在 subclass 建構式中呼叫它**。

(過程中可能採用 **Extract Method (110)**和 **Pull Up Method (322)** 處理共同程式碼)。

Push Down Method (函式下移) (328)

Superclass 中的某個函式只與部分(而非全部) subclasses 有關。

將這些函式移到相關的那些 subclass 去。

Push Down Field (欄位下移) (329)

Superclass 中的某個欄位只被部分(而非全部) subclasses 用到。

將這個欄位移到需要它的那些 subclasses 去。

Extract Subclass (提煉子類別) (330)

Class 中的某些特性只被某些(而非全部)實體用到。

新建一個 subclass，將上述的那一部份特性移到 subclass 中。

若希望對用戶隱藏 subclass 的存在，可使用 **Replace Constructor with Factory Method (304)**。

(**繼承，subclass 只能用以表現一組變化**，對比 **Extract Class (149)**之委託)

Extract Superclass (提煉超類別) (336)

兩個 classes 有相似特性。

為這兩個 classes 建立一個 superclass，將相同特性移至 superclass，不同的函式則成為抽象函式。

(可進階改為 **Composite** 範式)

若考慮**委託**，可選擇 **Extract Class (149)**或者之後再採取 **Replace Inheritance with Delegation (352)**。

若各個 subclasses 中某個函式的整體流程很相似，也許可以使用 **Form Template Method (345)**。

Extract Interface (提煉介面) (341)

某個 class 在不同環境下扮演截然**不同的角色**、若干客戶**使用 class 介面中的同一子集**，或者**兩個 classes 的介面有部分相同**。

將相同子集或角色提煉到一個獨立介面中。

Collapse Hierarchy (折疊繼承體系) (344)

superclass 和 subclass 之間無太大區別。

將它們合為一體。

Form Template Method (塑造模版函式) (345)

兩個函式以相同順序執行大致相近的操作，但不盡相同。

將**執行各操作的順序移至 superclass**，並**以抽象函式定義那些有所不同的操作**，而倚賴多型保持其差異性。

Replace Inheritance with Delegation (以委託取代繼承) (352)

某個 subclass 只用 superclass 介面中的一部份，或是根本不需要繼承而來的資料。

在 subclass 中**新建一個欄位用以保存 superclass**；調整 subclass 函式，令它改而委託 superclass；然後去掉兩者之間的繼承關係。(如此能更清楚表明只需要受委託類別的部分功能)。

Replace Delegation with Inheritance (以繼承取代委託) (355)

在兩個 classes 之間使用委託關係，並經常為整個介面編寫許多極簡單的請託函式(delegating methods)。

讓「**請託**(delegating) class」**繼承**「**受託**(delegate) class」。

也可考慮 **Remove Middle Man (160)**、**Extract Superclass (336)**或 **Extract Interface (341)**。

若沒有使用「受託 class」的**所有**函式，或者「**受託物件被一個以上的其他物件共享，且受託物件是可變的**(mutable)」，就不該使用此項重構。

大型重構 (Big Refactorings)

(由於大型重構的步驟較為概括，所以在此不詳細記錄，僅節錄部分內容)。

Tease Apart Inheritance (梳理並分解繼承體系) (362)

某個繼承體系同時擔任兩項任務。

建立兩個繼承體系，並透過**委託關係**讓其中一個可以呼叫另一個。

Convert Procedural Design to Objects (將程序式設計轉化為物件設計) (368)

你手上有些程式碼，以傳統的程序式風格寫就。

將資料記錄變成物件，將行為分開，並將行為移入相關物件之中。

Separate Domain from Presentation (將領域和表述/顯示分離) (370)

某些 GUI classes 之中包含了 domain logic (領域邏輯)。

將 domain logic (領域邏輯)分離出來，為它們建立獨立的 domain classes。

Extract Hierarchy (提煉繼承體系) (375)

你有某個 class 作了太多工作，其中一部份工作是以大量條件式完成的。

建立繼承體系，以一個 subclass 表示一種特殊情況。