1

重構，第一個案例

Refactoring, a First Example

我該怎麼開始介紹重構（**refactoring**）呢？按照傳統作法，一開始介紹某個東西時，首先應該大致講講它的歷史、主要原理等等。可是每當有人在會場上介紹這些東西，總是誘發我的瞌睡蟲。我的思緒開始游蕩，我的眼神開始迷離，直到他或她拿出實例，我才能夠提起精神。實例之所以可以拯救我於太虛之中，因爲它讓我看見事情的真正行進。談原理，很容易流於泛泛，又很難說明如何實際應用。給出一個實例，卻可以幫助我把事情認識清楚。

所以我決定以一個實例作爲本書起點。在此過程中我將告訴你很多重構原理，並且讓你對重構過程有一點感覺。然後我才能向你提供一般慣見的原理介紹。

但是，面對這個介紹性實例，我遇到了一個大問題。如果我選擇一個大型程式，對程式本身的描述和對重構過程的描述就太複雜了，任何讀者都將無法掌握（我試了一下，哪怕稍微複雜一點的例子都會超過 100 頁）。如果我選擇一個夠小以至於容易理解的程式，又恐怕看不出重構的價值。

和任何想要介紹「應用於真實世界中的有用技術」的人一樣，我陷入了一個十分典型的兩難困境。我將帶引你看看如何在一個我所選擇的小程式中進行重構，然而坦白說，那個程式的規模根本不值得我們那麼做。但是如果我給你看的程式碼是大系統的一部分，重構技術很快就變得重要起來。所以請你一邊觀賞這個小例子，一邊想像它身處於一個大得多的系統。

1.1 起點

實例非常簡單。這是一個影片出租店用的程式，計算每一位顧客的消費金額並列印報表（statement）。操作者告訴程式：顧客租了哪些影片、租期多長，程式便根據租賃時間和影片類型算出費用。影片分爲三類：普通片、兒童片和新片。除了計算費用，還要爲常客計算點數；點數會隨著「租片種類是否爲新片」而有不同。

我以數個 classes 表現這個例子中的元素。圖 1.1 是一張 UML class diagram（類別圖），用以顯示這些 classes。我會逐一列出這些 classes 的程式碼。

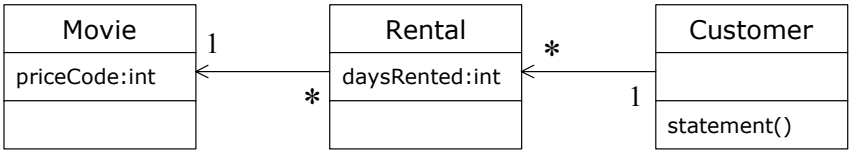

圖 1.1 本例一開始的各個 classes。此圖只顯示最重要的特性。圖中所用符號是 UML（Unified Modeling Language，統一建模語言，[Fowler, UML]）。

Movie（影片）

`Movie`只是一個簡單的data class（純資料類別）。

```
public class Movie {
  public static final int CHILDRENS = 2;
  public static final int REGULAR = 0;
  public static final int NEW_RELEASE = 1;

  private String _title;   // 名稱
  private int _priceCode;  // 價格（代號）

  public Movie(String title, int priceCode){
      _title = title;
      _priceCode = priceCode;
  }
```

```
    public int getPriceCode(){
        return _priceCode;
    }

    public void setPriceCode(int arg){
        _priceCode = arg;
    }

    public String getTitle(){
        return _title;
    }
  }
```

Rental（租賃）

`Rental` class表示「某個顧客租了一部影片」。

```
class Rental {
  private Movie _movie;          // 影片
  private int _daysRented;       // 租期

  public Rental(Movie movie, int daysRented) {
    _movie = movie;
    _daysRented = daysRented;
  }

  public int getDaysRented() {
    return _daysRented;
  }

  public Movie getMovie() {
    return _movie;
  }
}
```

譯註：中文版（本書）支援網站提供本章重構過程中的各階段完整程式碼（共分七個階段），並含測試。網址見於封底。

Customer（顧客）

`Customer` class用來表示顧客。就像其他classes一樣，它也擁有資料和相應的存取函式（accessor）：

```
class Customer {
  private String _name;                          // 姓名
  private Vector _rentals = new Vector();        // 租借記錄

  public Customer(String name) {
      _name = name;
  }

  public void addRental(Rental arg) {
      _rentals.addElement(arg);
  }

  public String getName() {
      return _name;
  }

  // 譯註：續下頁...
```

`Customer`還提供了一個用以製造報表的函式（method），圖 1.2 顯示這個函式帶來的交互過程（interactions）。完整程式碼顯示於下一頁。

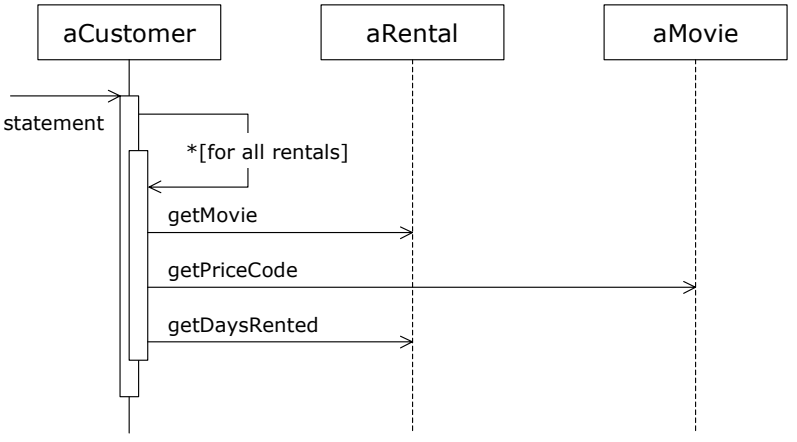

圖 1.2 `statement()` 的交互過程（interactions）

```
public String statement() {
  double totalAmount = 0;           // 總消費金額
  int frequentRenterPoints = 0;     // 常客積點
  Enumeration rentals = _rentals.elements();
  String result = "Rental Record for " + getName() + "\n";

  while(rentals.hasMoreElements()){
    double thisAmount = 0;
    Rental each = (Rental) rentals.nextElement(); // 取得一筆租借記錄

    //determine amounts for each line
    switch(each.getMovie().getPriceCode()) {  // 取得影片出租價格
      case Movie.REGULAR:           // 普通片
        thisAmount += 2;
        if(each.getDaysRented()>2)
          thisAmount += (each.getDaysRented()-2)*1.5;
        break;

      case Movie.NEW_RELEASE:       // 新片
        thisAmount += each.getDaysRented()*3;
        break;

      case Movie.CHILDRENS:         // 兒童片
        thisAmount += 1.5;
        if(each.getDaysRented()>3)
          thisAmount += (each.getDaysRented()-3)*1.5;
        break;
    }

    // add frequent renter points（累加 常客積點）
    frequentRenterPoints ++;
    // add bonus for a two day new release rental
    if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE &&
       each.getDaysRented() > 1)
      frequentRenterPoints ++;

    // show figures for this rental（顯示此筆租借資料）
    result += "\t" + each.getMovie().getTitle() + "\t" +
              String.valueOf(thisAmount) + "\n";
    totalAmount += thisAmount;
  }

  // add footer lines（結尾列印）
  result += "Amount owed is " + String.valueOf(totalAmount) + "\n";
  result += "You earned " + String.valueOf(frequentRenterPoints) +
            " frequent renter points";
  return result;
}
```

對此起始程式的評價

這個起始程式給你留下什麼印象？我會說它設計得不好，而且很明顯不符合物件導向精神。對於這樣一個小程式，這些缺點其實沒有什麼關係。快速而隨性（quick and dirty）地設計一個簡單的程式並沒有錯。但如果這是複雜系統中具有代表性的一段，那麼我就真的要對這個程式信心動搖了。`Customer`裡頭那個長長的`statement()`做的事情實在太多了，它做了很多原本應該由其他class完成的事情。

即便如此，這個程式還是能正常工作。所以這只是美學意義上的判斷，只是對醜陋程式碼的厭惡，是嗎？在我們修改這個系統之前的確如此。編譯器才不會在乎程式碼好不好看呢。但是當我們打算修改系統的時候，就涉及到了人，而人在乎這些。差勁的系統是很難修改的，因爲很難找到修改點。如果很難找到修改點，程式員就很有可能犯錯，從而引入「臭蟲」（bugs）。

在這個例子裡，我們的用戶希望對系統做一點修改。首先他們希望以HTML格式列印報表，這樣就可以直接在網頁上顯示，這非常符合潮流。現在請你想一想，這個變化會帶來什麼影響。看看程式碼你就會發現，根本不可能在列印HTML報表的函式中復用（*reuse*）目前`statement()`的任何行爲。你惟一可以做的就是編寫一個全新的`htmlStatement()`，大量重複`statement()`的行爲。當然，現在做這個還不太費力，你可以把`statement()`複製一份然後按需要修改就是。

但如果計費標準發生變化，又會發生什麼事？你必須同時修改`statement()`和`htmlStatement()`，並確保兩處修改的一致性。當你後續還要再修改時，剪貼（copy-paste）問題就浮現出來了。如果你編寫的是一個永不需要修改的程式，那麼剪剪貼貼就還好，但如果程式要保存很長時間，而且可能需要修改，剪貼行爲就會造成潛在的威脅。

現在，第二個變化來了：用戶希望改變影片分類規則，但是還沒有決定怎麼改。他們設想了幾種方案，這些方案都會影響顧客消費和常客積點的計算方式。作爲一個經驗豐富的開發者，你可以肯定：不論用戶提出什麼方案，你惟一能夠獲得的保證就是他們一定會在六個月之內再次修改它。

爲了應付分類規則和計費規則的變化，程式必須對`statement()`作出修改。但如果我們把`statement()`內的程式碼拷貝到用以列印HTML報表的函式中，我們就必須確保將來的任何修改在兩個地方保持一致。隨著各種規則變得愈來愈複雜，適當的修改點愈來愈難找，不犯錯的機會也愈來愈少。

你的態度也許傾向於「儘量少修改程式」：不管怎麼說，它還執行得很好。你心裡頭牢牢記著那句古老的工程學格言：「如果它沒壞，就別動它」。這個程式也許還沒壞掉，但它帶來了傷害。它讓你的生活比較難過，因爲你發現很難完成客戶所需的修改。這時候就該重構技術粉墨登場了。

如果你發現自己需要爲程式添加一個特性，而程式碼結構使你無法很方便地那麼做，那就先重構那個程式，使特性的添加比較容易進行，然後再添加特性。

1.2 重構的第一步

每當我要進行重構的時候，第一個步驟永遠相同：我得爲即將修改的程式碼建立一組可靠的測試環境。這些測試是必要的，因爲儘管遵循重構準則可以使我避免絕大多數的臭蟲引入機會，但我畢竟是人，畢竟有可能犯錯。所以我需要可靠的測試。

由於statement()的運作結果是個字串（string），所以我首先假設一些顧客，讓他們每個人各租幾部不同的影片，然後產生報表字串。然後我就可以拿新字串和手上已經檢查過的參考字串做比較。我把所有測試都設置好，俾得以在命令列輸入一條Java命令就把它們統統執行起來。執行這些測試只需數秒鐘，所以一如你即將見到，我經常執行它們。

測試過程中很重要的一部分，就是測試程式對於結果的回報方式。它們要不說"OK"，表示所有新字串都和參考字串一樣，要不就印出一份失敗清單，顯示問題字串的出現行號。這些測試都屬於自我檢驗（self-checking）。是的，你必須讓測試有能力自我檢驗，否則就得耗費大把時間來回比對，這會降低你的開發速度。

進行重構的時候，我們需要倚賴測試，讓它告訴我們是否引入了臭蟲。好的測試是重構的根本。花時間建立一個優良的測試機制是完全值得的，因爲當你修改程式時，好測試會給你必要的安全保障。測試機制在重構領域的地位實在太重要了，我將在第 4 章詳細討論它。

重構之前，首先檢查自己是否有一套可靠的測試機制。這些測試必須有自我檢驗（self-checking）能力。

1.3 分解並重組 statement()

第一個明顯引起我注意的就是長得離譜的statement()。每當看到這樣長長的函式，我就想把它大卸八塊。要知道，程式碼區塊愈小，程式碼的功能就愈容易管理，程式碼的處理和搬移也都愈輕鬆。

本章重構過程的第一階段中，我將說明如何把長長的函式切開，並把較小塊的程式碼移至更合適的 class 內。我希望降低程式碼重複量，從而使新的（列印 HTML 報表用的）函式更容易撰寫。

第一個步驟是找出程式碼的邏輯泥團（*logical clump*）並運用*Extract Method*（110）。本例一個明顯的邏輯泥團就是`switch`述句，把它提煉（*extract*）到獨立函式中似乎比較好。

和任何重構準則一樣，當我提煉一個函式時，我必須知道可能出什麼錯。如果我提煉得不好，就可能給程式引入臭蟲。所以重構之前我需要先想出安全作法。由於先前我已經進行過數次這類重構，所以我已經把安全步驟記錄於書後的重構名錄（refactoring catalog）中了。

首先我得在這段程式碼裡頭找出函式內的區域變數（local variables）和參數（parameters）。我找到了兩個：`each`和`thisAmount`，前者並未被修改，後者會被修改。任何不會被修改的變數都可以被我當成參數傳入新的函式，至於會被修改的變數就需格外小心。如果只有一個變數會被修改，我可以把它當作回返值。`thisAmount`是個暫時變數，其值在每次迴圈起始處被設爲 0，並且在`switch`述句之前不會改變，所以我可以直接把新函式的回返值賦予它。

下面兩頁展示重構前後的程式碼。重構前的程式碼在左頁，重構後的程式碼在右頁。凡是從函式提煉出來的程式碼，以及新程式碼所做的任何修改，只要我覺得不是明顯到可以一眼看出，就以粗體字標示出來特別提醒你。**本章剩餘部分將延續這種左右比對形式**。

```
public String statement() {
  double totalAmount = 0;           // 總消費金額
  int frequentRenterPoints = 0;     // 常客積點
  Enumeration rentals = _rentals.elements();
  String result = "Rental Record for " + getName() + "\n";

  while(rentals.hasMoreElements()){
    double thisAmount = 0;
    Rental each = (Rental) rentals.nextElement(); // 取得一筆租借記錄

    //determine amounts for each line
    switch(each.getMovie().getPriceCode()) {  // 取得影片出租價格
      case Movie.REGULAR:           // 普通片
        thisAmount += 2;
        if(each.getDaysRented()>2)
            thisAmount += (each.getDaysRented()-2)*1.5;
        break;

      case Movie.NEW_RELEASE:       // 新片
        thisAmount += each.getDaysRented()*3;
        break;

      case Movie.CHILDRENS:         // 兒童片
        thisAmount += 1.5;
        if(each.getDaysRented()>3)
            thisAmount += (each.getDaysRented()-3)*1.5;
        break;
    }

    // add frequent renter points（累加 常客積點）
    frequentRenterPoints ++;
    // add bonus for a two day new release rental
    if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
         each.getDaysRented() > 1)
      frequentRenterPoints ++;

    // show figures for this rental（顯示此筆租借資料）
    result += "\t" + each.getMovie().getTitle() + "\t" +
              String.valueOf(thisAmount) + "\n";
    totalAmount += thisAmount;
  }

  // add footer lines（結尾列印）
  result += "Amount owed is " + String.valueOf(totalAmount) + "\n";
  result += "You earned " + String.valueOf(frequentRenterPoints) +
            " frequent renter points";
  return result;
}
```

```
public String statement() {
  double totalAmount = 0;
  int frequentRenterPoints = 0;
  Enumeration rentals = _rentals.elements();
  String result = "Rental Record for " + getName() + "\n";
  while (rentals.hasMoreElements()){
    double thisAmount = 0;
    Rental each = (Rental) rentals.nextElement();

    thisAmount = amountFor(each);    // 計算一筆租片費用

    // add frequent renter points
    frequentRenterPoints ++;
    // add bonus for a two day new release rental
    if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
        each.getDaysRented() > 1)
      frequentRenterPoints ++;

    // show figures for this rental
    result += "\t" + each.getMovie().getTitle() + "\t" +
              String.valueOf(thisAmount) + "\n";
    totalAmount += thisAmount;
  }
  // add footer lines
  result += "Amount owed is " + String.valueOf(totalAmount) + "\n";
  result += "You earned " + String.valueOf(frequentRenterPoints) +
            " frequent renter points";
  return result;
}

private int amountFor(Rental each) {     // 計算一筆租片費用
  int thisAmount = 0;
  switch(each.getMovie().getPriceCode()) {
    case Movie.REGULAR:          // 普通片
      thisAmount += 2;
      if(each.getDaysRented() > 2)
         thisAmount += (each.getDaysRented()-2)*1.5;
      break;
    case Movie.NEW_RELEASE:      // 新片
      thisAmount += each.getDaysRented() * 3;
      break;
    case Movie.CHILDRENS:        // 兒童片
      thisAmount += 1.5;
      if(each.getDaysRented() > 3)
         thisAmount += (each.getDaysRented() - 3) * 1.5;
      break;
  }
  return thisAmount;
}
```

每次做完這樣的修改之後，我都要編譯並測試。這一次起頭不算太好 — 測試失敗了，有兩筆測試數據告訴我發生錯誤。一陣迷惑之後我明白了自己犯的錯誤。我愚蠢地將amountFor() 的回返值型別宣告爲int，而不是double。

```
private double amountFor(Rental each) {  // 計算一筆租片費用
  double thisAmount = 0;
  switch(each.getMovie().getPriceCode()) {
    case Movie.REGULAR:          // 普通片
      thisAmount += 2;
      if(each.getDaysRented() > 2)
        thisAmount += (each.getDaysRented()-2)*1.5;
      break;
    case Movie.NEW_RELEASE:      // 新片
      thisAmount += each.getDaysRented() * 3;
      break;
    case Movie.CHILDRENS:        // 兒童片
      thisAmount += 1.5;
      if(each.getDaysRented() > 3)
        thisAmount += (each.getDaysRented() - 3) * 1.5;
      break;
  }
  return thisAmount;
}
```

我經常犯這種愚蠢可笑的錯誤，而這種錯誤往往很難發現。在這裡，Java無怨無尤地把double型別轉換爲int型別，而且還愉快地做了取整數動作 [Java Spec]。還好此處這個問題很容易發現，因爲我做的修改很小，而且我有很好的測試。藉著這個意外疏忽，我要闡述重構步驟的本質：由於每次修改的幅度都很小，所以任何錯誤都很容易發現。你不必耗費大把時間除錯，哪怕你和我一樣粗心。

重構技術係以微小的步伐修改程式。如果你犯下錯誤，很容易便可發現它。

由於我用的是Java，所以我需要對程式碼做一些分析，決定如何處理區域變數。如果擁有相應的工具，這個工作就超級簡單了。Smalltalk的確擁有這樣的工具：Refactoring Browser。運用這個工具，重構過程非常輕鬆，我只需標示出需要重構的程式碼，在選單中點選*Extract Method*，輸入新的函式名稱，一切就自動搞定。而且工具絕不會像我那樣犯下愚蠢可笑的錯誤。我非常盼望早日出現Java版本的重構工具!