Working Effectively with Legacy code

CH21, 22, 23, 24

CH21: 修改大量相同程式

Context

1. 修改程式碼
2. 發現其他類似的程式碼
3. 修改第二個地方
4. 修改第三個地方
5. 修改第四個地方
6. .....

程式範例

請花一分鐘時間閱讀以下程式碼:

https://gist.github.com/hungyanbin/24b255b17f5f38cfb854b517554936a9

重構策略

刪除小塊重複程式碼（小步邁進）

```
outputStream.write(userName.getBytes());
outputStream.write( b: 0x00);
outputStream.write(passwd.getBytes());
outputStream.write( b: 0x00);
```

```
void writeField(OutputStream outputStream, String field) throws Exception {
    outputStream.write(field.getBytes());
    outputStream.write( b: 0x00);
}
```

```
public void write(OutputStream outputStream)
    outputStream.write(header);
    outputStream.write(getSize());
    outputStream.write(commandChar);
    outputStream.write(userName.getBytes());
    outputStream.write( b: 0x00);
    outputStream.write(passwd.getBytes());
    outputStream.write( b: 0x00);
    outputStream.write(footer);
}
```

```
public void write(OutputStream outputStr
    outputStream.write(header);
    outputStream.write(getSize());
    outputStream.write(commandChar);
    writeField(outputStream, userName);
    writeField(outputStream, passwd);
    outputStream.write(footer);
}
```

AddEmployeeCmd

```
public void write(OutputStream outputStream) throv
    outputStream.write(header);
    outputStream.write(getSize());
    outputStream.write(commandChar);
    outputStream.write(name.getBytes());
    outputStream.write( b: 0x00);
    outputStream.write(address.getBytes());
    outputStream.write( b: 0x00);
    outputStream.write(city.getBytes());
    outputStream.write( b: 0x00);
    outputStream.write(state.getBytes());
    outputStream.write( b: 0x00);
    outputStream.write(yearlySalary.getBytes());
    outputStream.write( b: 0x00);
    outputStream.write(footer);
}
```

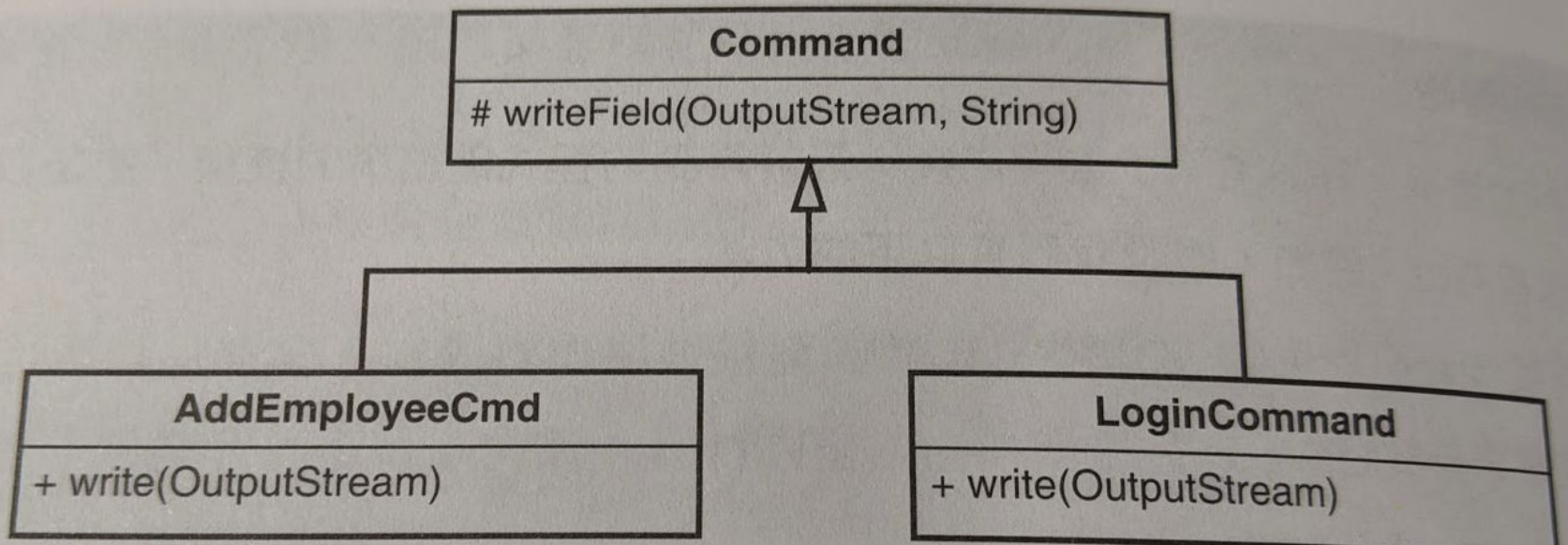


圖 21-2 命令類別繼承體系

AddEmployeeCmd

```
public void write(OutputStream outputStream) throw
    outputStream.write(header);
    outputStream.write(getSize());
    outputStream.write(commandChar);
    outputStream.write(name.getBytes());
    outputStream.write( b: 0x00);
    outputStream.write(address.getBytes());
    outputStream.write( b: 0x00);
    outputStream.write(city.getBytes());
    outputStream.write( b: 0x00);
    outputStream.write(state.getBytes());
    outputStream.write( b: 0x00);
    outputStream.write(yearlySalary.getBytes());
    outputStream.write( b: 0x00);
    outputStream.write(footer);
}
```

```
public void write(OutputStream outputStream)
    outputStream.write(header);
    outputStream.write(getSize());
    outputStream.write(commandChar);
    writeField(outputStream, name);
    writeField(outputStream, address);
    writeField(outputStream, city);
    writeField(outputStream, state);
    writeField(outputStream, yearlySalary);
    outputStream.write(footer);
}
```

AddEmployeeCmd

```
public void write(OutputStream outputStream)
    outputStream.write(header);
    outputStream.write(getSize());
    outputStream.write(commandChar);
    writeField(outputStream, name);
    writeField(outputStream, address);
    writeField(outputStream, city);
    writeField(outputStream, state);
    writeField(outputStream, yearlySalary);
    outputStream.write(footer);
}
```

LoginCmd

```
public void write(OutputStream outputStre
    outputStream.write(header);
    outputStream.write(getSize());
    outputStream.write(commandChar);
    writeField(outputStream, userName);
    writeField(outputStream, passwd);
    outputStream.write(footer);
}
```

AddEmployeeCmd

```
private void writeBody(OutputStream outputStre
    writeField(outputStream, name);
    writeField(outputStream, address);
    writeField(outputStream, city);
    writeField(outputStream, state);
    writeField(outputStream, yearlySalary);
}
```

```
public void write(OutputStream outputStream)
    outputStream.write(header);
    outputStream.write(getSize());
    outputStream.write(commandChar);
    writeBody(outputStream);
    outputStream.write(footer);
}
```

LoginCmd

```
private void writeBody(OutputStream output
    writeField(outputStream, userName);
    writeField(outputStream, passwd);
}
```

```
public void write(OutputStream outputStream)
    outputStream.write(header);
    outputStream.write(getSize());
    outputStream.write(commandChar);
    writeBody(outputStream);
    outputStream.write(footer);
}
```

重構提示

小心警慎

AddEmployeeCmd

```
byte[] header = {(byte) 0xde, (byte) 0xad};
byte[] commandChar = {(byte) 0x01};
byte[] footer = {(byte) 0xbe, (byte) 0xef};
int SIZE_LENGTH = 1;
int CMD_BYTE_LENGTH = 1;
```

LoginCmd

```
byte[] header = {(byte) 0xde, (byte) 0xad};
byte[] commandChar = {(byte) 0x02};
byte[] footer = {(byte) 0xbe, (byte) 0xef};
int SIZE_LENGTH = 1;
int CMD_BYTE_LENGTH = 1;
```

```java
public abstract class Command {

    public static final byte[] header = {(byte) 0xde, (byte) 0xad};
    public static final byte[] footer = {(byte) 0xbe, (byte) 0xef};
    public static final int SIZE_LENGTH = 1;
    public static final int CMD_BYTE_LENGTH = 1;

    protected abstract byte[] getCommandChar();
    protected abstract int getSize();
    protected abstract void writeBody(OutputStream outputStream) throws Exception;

    protected void writeField(OutputStream outputStream, String field) throws Exception {
        outputStream.write(field.getBytes());
        outputStream.write( b: 0x00);
    }

    public void write(OutputStream outputStream) throws Exception {
        outputStream.write(header);
        outputStream.write(getSize());
        outputStream.write(getCommandChar());
        writeBody(outputStream);
        outputStream.write(footer);
    }
}
```

AddEmployeeCmd

```
protected int getSize() {
    return header.length +
            SIZE_LENGTH +
            CMD_BYTE_LENGTH +
            footer.length +
            name.getBytes().length + 1 +
            address.getBytes().length + 1 +
            city.getBytes().length + 1 +
            state.getBytes().length + 1 +
            yearlySalary.getBytes().length + 1;
}
```

LoginCmd

```
protected int getSize() {
    return header.length + SIZE_LENGTH + CMD_BYTE_LENGTH +
            footer.length + userName.getBytes().length + 1 +
            passwd.getBytes().length + 1;
}
```

AddEmployeeCmd

```
protected int getSize() {
    return header.length +
           SIZE_LENGTH +
           CMD_BYTE_LENGTH +
           footer.length +
           getBodySize();
}
```

```
protected int getBodySize() {
    return name.getBytes().length + 1 +
           address.getBytes().length + 1 +
           city.getBytes().length + 1 +
           state.getBytes().length + 1 +
           yearlySalary.getBytes().length + 1;
}
```

LoginCmd

```
protected int getSize() {
    return header.length +
           SIZE_LENGTH +
           CMD_BYTE_LENGTH +
           footer.length +
           getBodySize();
}
```

```
protected int getBodySize() {
    return userName.getBytes().length + 1 +
           passwd.getBytes().length + 1;
}
```

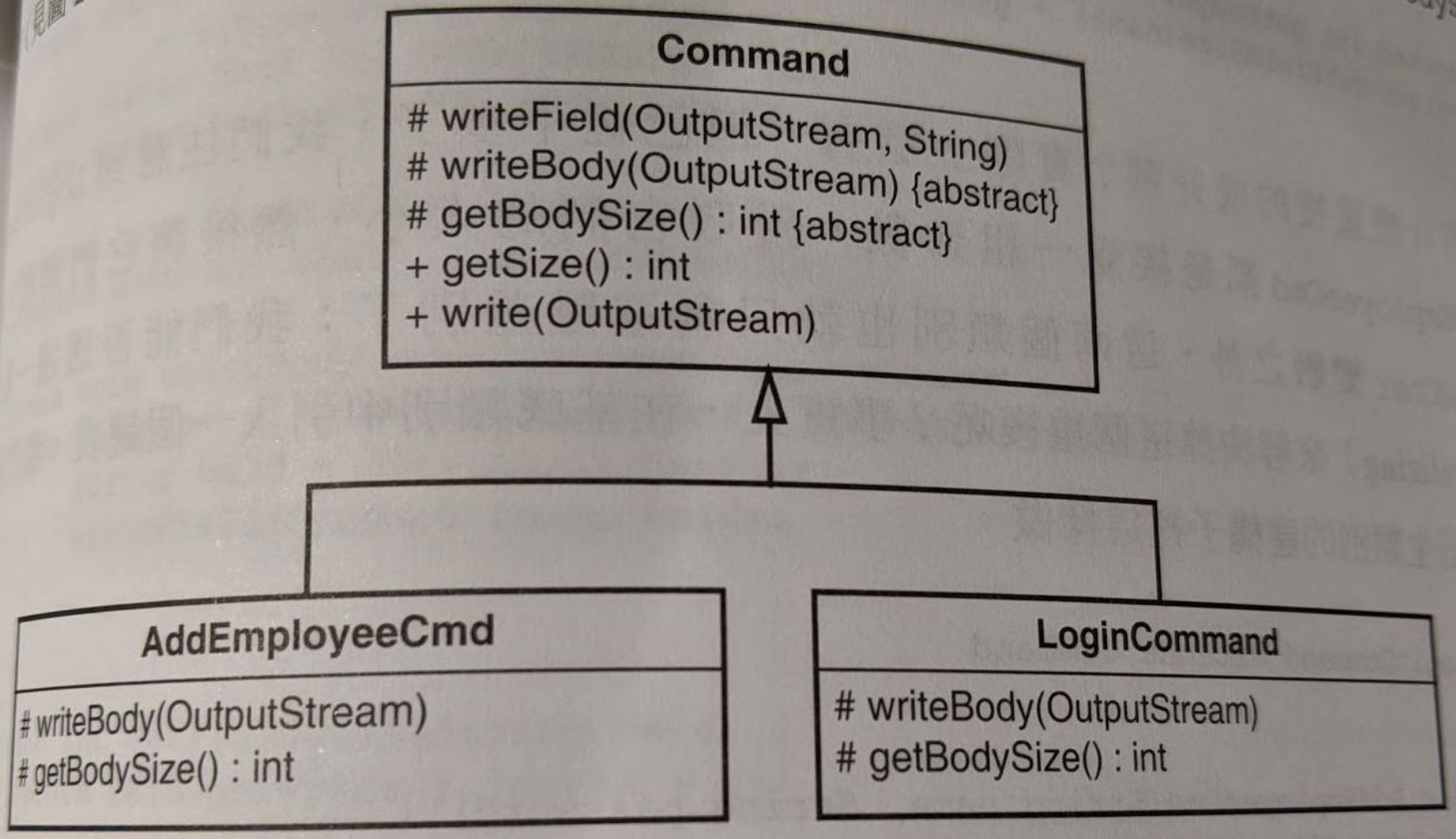
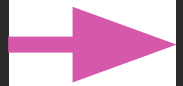


圖 21-4 提升 getSize

Further refactor

```
protected int getBodySize() {
    return name.getBytes().length + 1 +
           address.getBytes().length + 1 +
           city.getBytes().length + 1 +
           state.getBytes().length + 1 +
           yearlySalary.getBytes().length + 1;
}
```



```
protected int getBodySize() {
    return getFieldSize(name) +
           getFieldSize(address) +
           getFieldSize(city) +
           getFieldSize(state) +
           getFieldSize(yearlySalary);
}
```

停下來看一看

https://gist.github.com/hungyanbin/c0b54a705d5ababbb510ec15d4df0bf6

```java
public class AddEmployeeCmd extends Command{
    String name;
    String address;
    String city;
    String state;
    String yearlySalary;

    public AddEmployeeCmd(String name, String address, String city, String state, int yearlySalary) {
        this.name = name;
        this.address = address;
        this.city = city;
        this.state = state;
        this.yearlySalary = Integer.toString(yearlySalary);
    }

    @Override
    protected byte[] getCommandChar() { return new byte[]{(byte) 0x02}; }

    @Override
    protected int getBodySize() {
        return getFieldSize(name) +
                getFieldSize(address) +
                getFieldSize(city) +
                getFieldSize(state) +
                getFieldSize(yearlySalary);
    }

    @Override
    protected void writeBody(OutputStream outputStream) throws Exception {
        writeField(outputStream, name);
        writeField(outputStream, address);
        writeField(outputStream, city);
        writeField(outputStream, state);
        writeField(outputStream, yearlySalary);
    }
}
```

Refactor to loop

```java
@Override
protected int getBodySize() {
    int totalSize = 0;
    for (String field : fields) {
        totalSize += getFieldSize(field);
    }
    return totalSize;
}

@Override
protected void writeBody(OutputStream outputStream) throws Exception {
    for (String field : fields) {
        writeField(outputStream, field);
    }
}
```

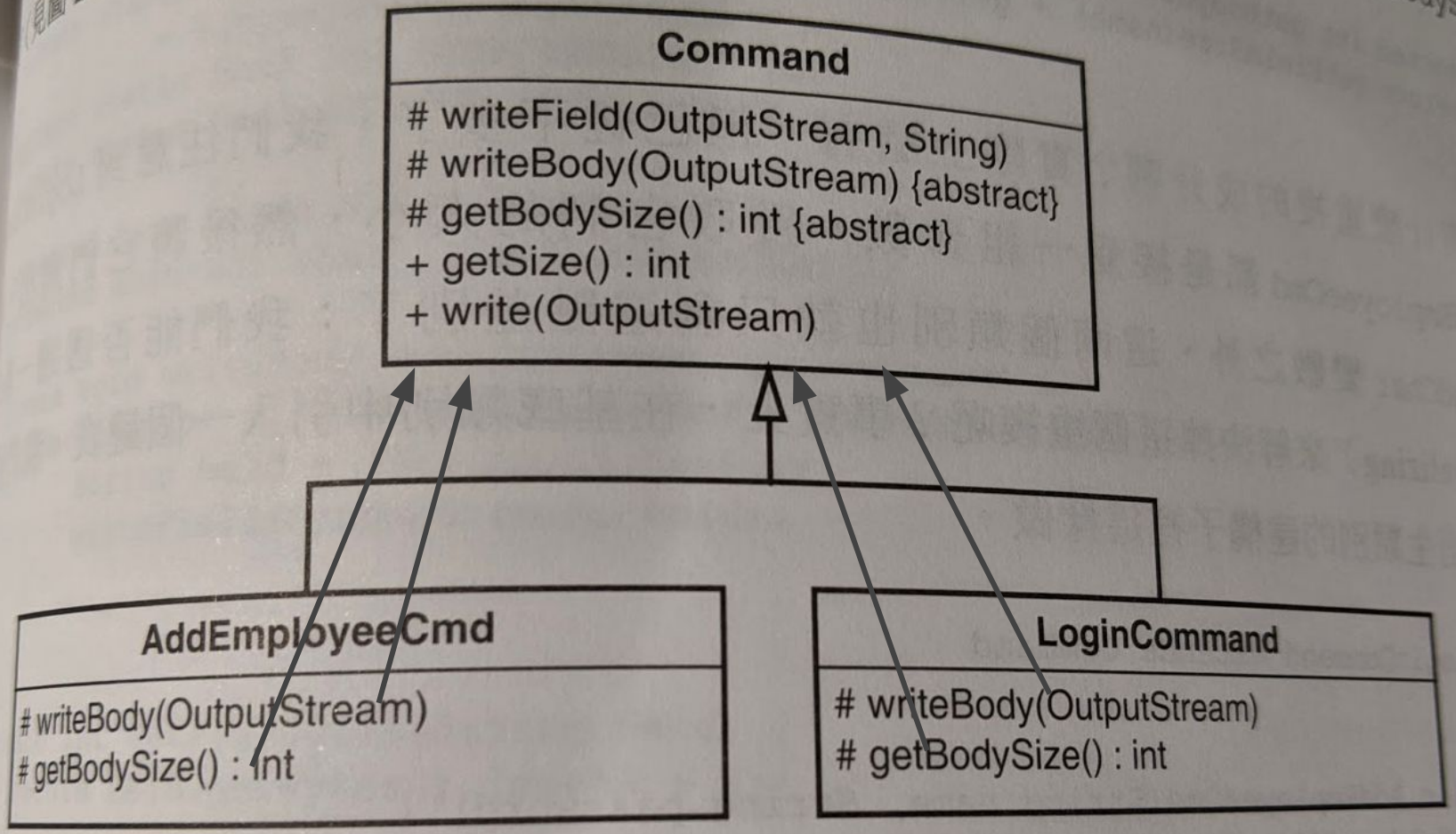


圖 21-4 提升 getSize

最後成果

https://gist.github.com/hungyanbin/662d44a4b3a577393b333a6db49634f6

探討

- 是否可以拿掉 AddEmployeeCmd, LoginCmd?
  - 客戶端程式碼需要變動
  - 意圖
- 重新命名 :LoginCmd -> LoginCommand
  - 避免用縮寫
- 重構之後的好處
  - 新增命令，重構前 V.S. 重構後
  - 非字串型參數？
  - 正交性（orthogonality）

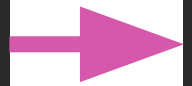
P.S. Orthogonality 在 Pragmatic Programmer 有更詳細的解釋

Open closed principle (開放封閉原則)

小結

- 一個持續不斷的迭代：重複-> 重構 -> 重複 -> 重構
- 警慎重構，有測試最好
- 重構過程中會發現其中隱藏的意圖，讓設計更加完善。

```
public void write(OutputStream outputStream) throws Exception {
    outputStream.write(header);
    outputStream.write(getSize());
    outputStream.write(getCommandChar());
    writeBody(outputStream);
    outputStream.write(footer);
}
```



```
public void write(OutputStream outputStream)
    writeHeader(outputStream);
    writeBody(outputStream);
    writeFooter(outputStream);
}
```

CH22: 測試巨型方法

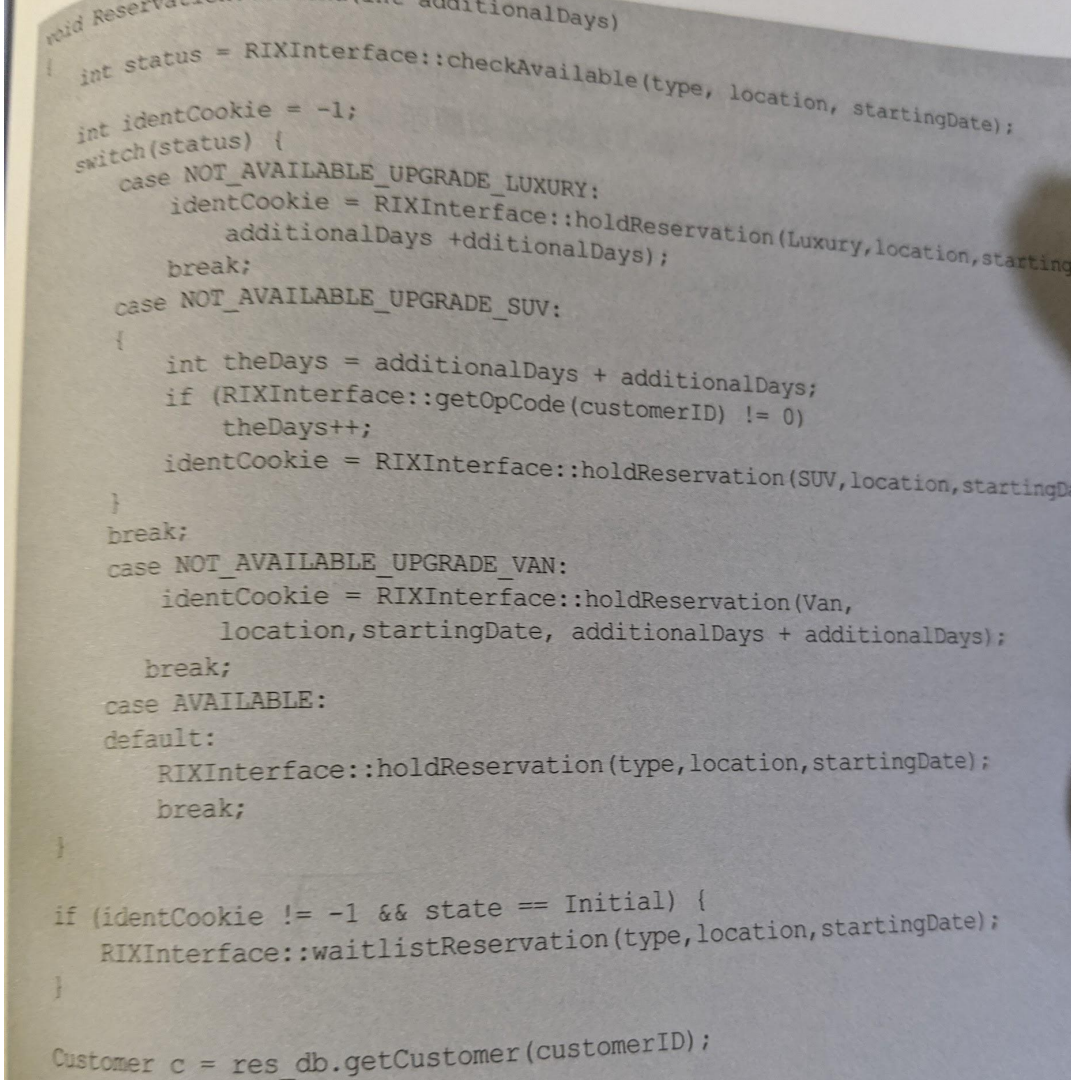
22.1 巨型方法種類

1. 項目列表式方法
2. 鋸齒狀方法

項目列表式方法

- If 或是 switch case
- 危險的區域變數

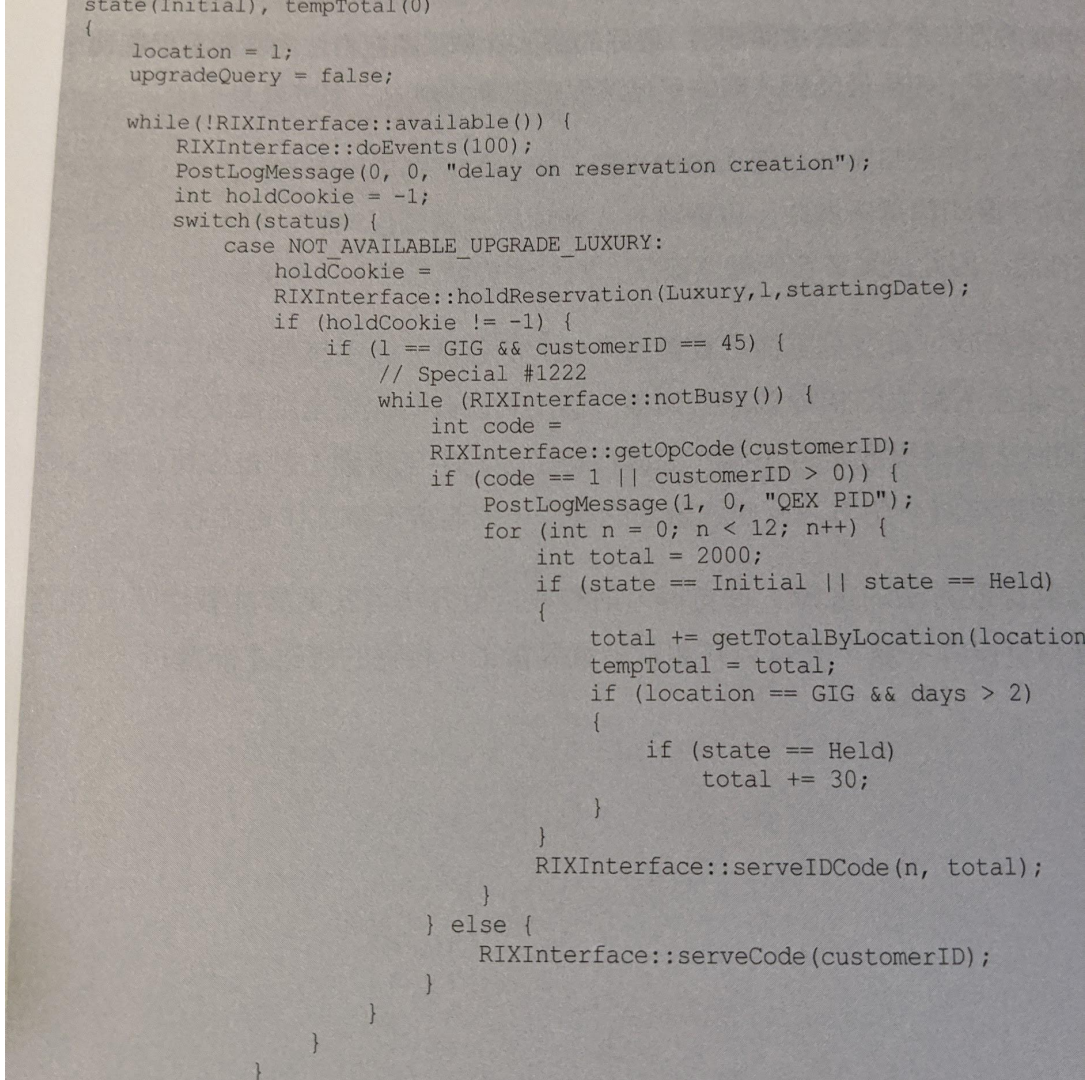
- （自以為）每一個 case 都很獨立，可以一直加下去



鋸齒狀方法

- 數量龐大的 if 判斷式
- Callback hell

- 不想讀懂裡面在幹嘛，最外面包一層最快
  - Check null
  - Add flag
  - try-catch



```
state(Initial), tempTotal(0)
{
    location = 1;
    upgradeQuery = false;

    while(!RIXInterface::available()) {
        RIXInterface::doEvents(100);
        PostLogMessage(0, 0, "delay on reservation creation");
        int holdCookie = -1;
        switch(status) {
            case NOT_AVAILABLE_UPGRADE_LUXURY:
                holdCookie =
                RIXInterface::holdReservation(Luxury,1,startingDate);
                if (holdCookie != -1) {
                    if (l == GIG && customerID == 45) {
                        // Special #1222
                        while (RIXInterface::notBusy()) {
                            int code =
                            RIXInterface::getOpCode(customerID);
                            if (code == 1 || customerID > 0)) {
                                PostLogMessage(1, 0, "QEX PID");
                                for (int n = 0; n < 12; n++) {
                                    int total = 2000;
                                    if (state == Initial || state == Held)
                                    {
                                        total += getTotalByLocation(location
                                        tempTotal = total;
                                        if (location == GIG && days > 2)
                                        {
                                            if (state == Held)
                                                total += 30;
                                        }
                                    }
                                    RIXInterface::serveIDCode(n, total);
                                }
                            } else {
                                RIXInterface::serveCode(customerID);
                            }
                        }
                    }
                }
```

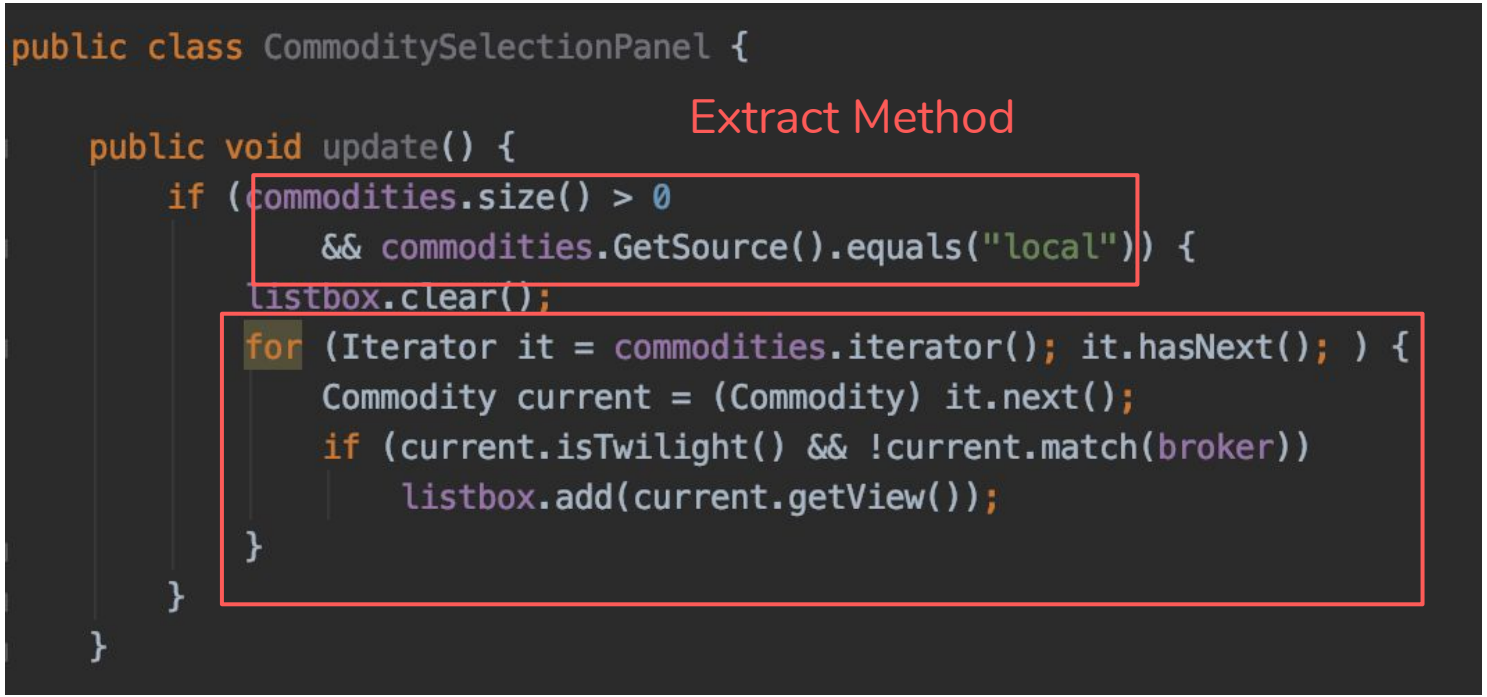
22.2 自動重構工具

- 不要參雜手動編排
  - 語句重排
  - 語句分組
- 目標
  - 分離依賴
  - 引入接縫

範例

```
public class CommoditySelectionPanel {

    public void update() {
        if (commodities.size() > 0
                && commodities.GetSource().equals("local")) {
            listbox.clear();
            for (Iterator it = commodities.iterator(); it.hasNext(); ) {
                Commodity current = (Commodity) it.next();
                if (current.isTwilight() && !current.match(broker))
                    listbox.add(current.getView());
            }
        }
    }
```

範例


public class CommoditySelectionPanel {
Extract Method
public void update() {
if (commodities.size() > 0
&& commodities.GetSource().equals("local")) {
listbox.clear();
for (Iterator it = commodities.iterator(); it.hasNext();) {
Commodity current = (Commodity) it.next();
if (current.isTwilight() && !current.match(broker))
listbox.add(current.getView());
}
}
}

```
public void update() {
    if (commoditiesAreReadyForUpdate()) {
        listbox.clear();
        updateCommodities();
    }
}

private void updateCommodities() {
    for (Iterator it = commodities.iterator(); it.hasNext(); ) {
        Commodity current = (Commodity) it.next();
        if (current.isTwilight() && !current.match(broker))
            listbox.add(current.getView());
    }
}

private boolean commoditiesAreReadyForUpdate() {
    return commodities.size() > 0
            && commodities.GetSource().equals("local");
}
```

Display list

```
public class CommoditySelectionPanel {

    public void update() {
        if (commoditiesAreReadyForUpdate()) {
            clearDisplay();
            updateCommodities();
        }
    }

    private void updateCommodities() {
        for (Iterator it = commodities.iterator(); it.hasNext(); ) {
            Commodity current = (Commodity) it.next();
            if (current.isTwilight() && !current.match(broker)) {
                displayCommodity(current.getView());
            }
        }
    }

    private boolean commoditiesAreReadyForUpdate() {
        return commodities.size() > 0
                && commodities.GetSource().equals("local");
    }

    private void clearDisplay() {
        listbox.clear();
    }

    private void displayCommodity(CommodityView view) {
        listbox.add(view);
    }
```

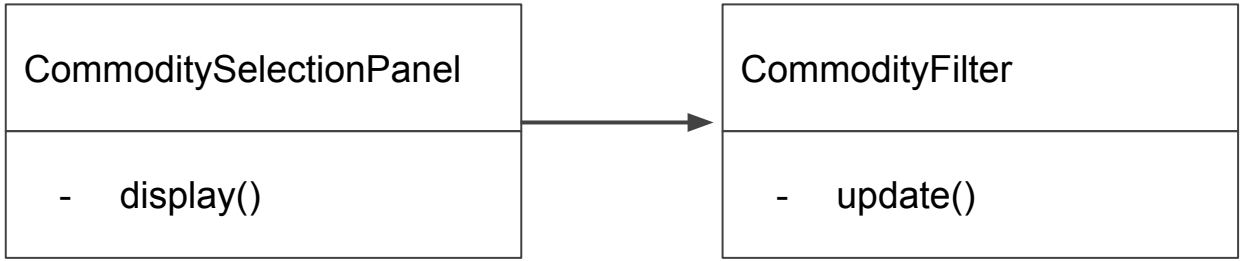
自動化重構工具任務已完成

引入接縫來解依賴

```
public class CommoditySelectionPanel {

    public void update() {
        if (commoditiesAreReadyForUpdate()) {
            clearDisplay();
            updateCommodities();
        }
    }

    private void updateCommodities() {
        for (Iterator it = commodities.iterator(); it.hasNext(); ) {
            Commodity current = (Commodity) it.next();
            if (current.isTwilight() && !current.match(broker)) {
                displayCommodity(current.getView());
            }
        }
    }

    private boolean commoditiesAreReadyForUpdate() {
        return commodities.size() > 0
                && commodities.GetSource().equals("local");
    }

    private void clearDisplay() {
        listbox.clear();
    }

    private void displayCommodity(CommodityView view) {
        listbox.add(view);
    }
```

protected(Seam)

精煉出更好的設計



Separation of Concern

22.3 手動重構的挑戰

- 忘記傳遞變數
- 取了相同方法(變數)名稱
- 錯誤的返回值
- 更多...

手動重構技術

1. 引入感測變數
2. 只提取你所了解的
3. 依賴收集
4. 分解出方法物件

1.引入感測變數

```
public class DOMBuilder

{
    public boolean nodeAdded = false;
    ...
    void processNode(XDOMNSnippet root, List childNodes)
    {
        if (root != null) {
            if (childNodes != null)
                root.addNode(new XDOMNSnippet(childNodes));
            root.addChild(XDOMNSnippet.NullSnippet);
        }
        List paraList = new ArrayList();
        XDOMNSnippet snippet = new XDOMNReSnippet();
        snippet.setSource(m_state);
        for (Iterator it = childNodes.iterator();
                    it.hasNext(); ) {
            XDOMNNode node = (XDOMNNode)it.next();
            if (node.type() == TF_G || node.type() == TF_H ||
                    (node.type() == TF_GLOT && node.isChild())) {
                paraList.add(node);
                nodeAdded = true;
            }
```

添加測試

```
void testAddNodeOnBasicChild()
{
    DOMBuilder builder = new DomBuilder();
    List children = new ArrayList();
    children.add(new XDOMNNode(XDOMNNode.TF_G));
    Builder.processNode(new XDOMNSnippet(), children);

    assertTrue(builder.nodeAdded);
}
```

```
void testNoAddNodeOnNonBasicChild()
{
    DOMBuilder builder = new DomBuilder();
    List children = new ArrayList();
    children.add(new XDOMNNode(XDOMNNode.TF_A));
    Builder.processNode(new XDOMNSnippet(), children);

    assertTrue(!builder.nodeAdded);
}
```

重構

```
public class DOMBuilder
{
    void processNode(XDOMNSnippet root, List childNodes)
    {
        if (root != null) {
            if (childNodes != null)
                root.addNode(new XDOMNSnippet(childNodes));
            root.addChild(XDOMNSnippet.NullSnippet);
        }
        List paraList = new ArrayList();
        XDOMNSnippet snippet = new XDOMNReSnippet();
        snippet.setSource(m_state);
        for (Iterator it = childNodes.iterator();
                it.hasNext();) {
            XDOMNNode node = (XDOMNNode)it.next();
            if (isBasicChild(node)) {
                paraList.addNode(node);
                nodeAdded = true;
            }
            ...
        }
        ...
    }
    private boolean isBasicChild(XDOMNNode node) {
        return node.type() == TF_G
            || node.type() == TF_H
            || node.type() == TF_GLOT && node.isChild());
    }
    ...
}
```

2. 只提取你所了解的

- 小塊程式碼 (3-5 行)
- 關注耦合數 (越少越好)
  - 參數 + 回傳值 = 耦合數
  - 不包含成員變數
- 命令式方法
  - 耦合數為 0
- 積少成多

3. 依賴收集

1. 容易驗證的行為
2. 負責顯示的程式碼
3. 不添加測試保護
4. 提取到其他地方

1. 不容易驗證的行為
2. 複雜的商業邏輯
3. 添加測試保護
4. 保留程式碼

4. 分解出方法物件（Break out Method Object）

- 建立新類別
- 在該類別裡新增方法：像是run(), execute()
- 將原來方法的程式碼移至該類別

22.4 策略

1. 主幹提取
2. 序列發現
3. 優先提取到目前類別中

1. 主幹提取

```
public class CommoditySelectionPanel {

    public void update() {
        if (commodities.size() > 0
                && commodities.GetSource().equals("local")) {
            listbox.clear();
            for (Iterator it = commodities.iterator(); it.hasNext(); ) {
                Commodity current = (Commodity) it.next();
                if (current.isTwilight() && !current.match(broker))
                    listbox.add(current.getView());
            }
        }
    }
```

2. 序列發現

```
public class CommoditySelectionPanel {

    public void update() {
        if (commodities.size() > 0
                && commodities.GetSource().equals("local")) {
            listbox.clear();
            for (Iterator it = commodities.iterator(); it.hasNext(); ) {
                Commodity current = (Commodity) it.next();
                if (current.isTwilight() && !current.match(broker))
                    listbox.add(current.getView());
            }
        }
    }
```

3. 優先提取到目前類別中

```
public void update() {
    if (commoditiesAreReadyForUpdate()) {
        listbox.clear();
        updateCommodities();
    }
}

private void updateCommodities() {
    for (Iterator it = commodities.iterator(); it.hasNext(); ) {
        Commodity current = (Commodity) it.next();
        if (current.isTwilight() && !current.match(broker))
            listbox.add(current.getView());
    }
}

private boolean commoditiesAreReadyForUpdate() {
    return commodities.size() > 0
            && commodities.GetSource().equals("local");
}
```

Display list

移到新類別?


修但幾勒
先不要

小結

- 自動重構工具優先，不要混雜手動重構
- 手動重構技術
- 策略 (手動 && 自動)

CH23: 降低修改的風險

23.1 超感編輯

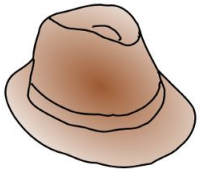
打的每一個字都有回饋，在靜態語言中，IDE 可以隨時檢查輸入的變數、型態是否正確（還有自動完成）。其中作者還期望未來會有隨時執行單元測試的 IDE 。

23.1 超感編輯

打的每一個字都有回饋，在靜態語言中，IDE 可以隨時檢查輸入的變數、型態是否正確（還有自動完成）。其中作者還期望未來會有隨時執行單元測試的 IDE 。

PS: 別再用 VIM 寫扣了（戰）

23.2 單一目標的編輯



Refactoring

When refactoring every change you make is a small behavior-preserving change. You only refactor with green tests, and any test failing indicates a mistake. By stringing together a series of small changes like this you can move more quickly and with less risk because you shouldn't get trapped in debugging.

Adding Function

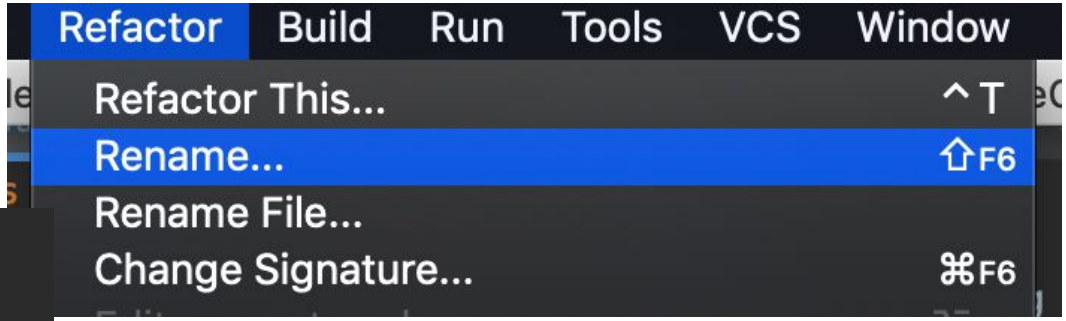
Any other change to the code is adding function. You will add new tests and break existing tests. You aren't confined to behavior-preserving changes (but it's wise to keep changes small and return to green tests swiftly).

During programming you may swap frequently between hats, perhaps every couple of minutes. But...

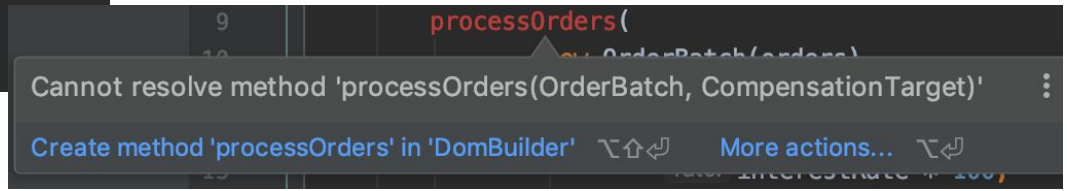
You can only wear one hat at a time

Ref: Refactoring by Martin Fowler

23.4 依靠編輯器




```
public void process(List orders,
                    int dailyTarget,
                    double interestRate,
                    int compensationPercent) {
    processOrders(
            Create method 'processOrders' in 'DomBuilder'
            Rename reference
                    dailyTarget,
                    rate: interestRate * 100,
                    compensationPercent)
    );
}
```



23.5 Pair Programming

- 多一雙眼睛
- 適時阻止過度設計/ 離需求越來越遠
- 多一個人可以記 Todo list
- 三個臭皮匠, 勝過一個諸葛亮

小結

1. 超感編輯
2. 單一目的的編輯
3. 依靠編輯器
4. Pair programming

CH24: 當你感到絕望時

- 維護舊系統 V.S. 打掉重練
- 找到動力
  - 跟同事學習
  - ~~繳房貸，養小三~~
  - 旅遊
- 社群
  - 聊八卦
  - 抱怨
  - 學習新知
  - 認識大神
- Side project
  - 練習 TDD
  - Coding kata

這是最後一頁

程式碼實作

Tennis-kata

- https://github.com/emilybache/Tennis-Refactoring-Kata
- 打開 TennisTest 並執行測試
- 打開 TennisGame2 開始重構
  - 盡量使用 IDE 提供的自動化重構工具
  - 做完一個段落請提交一個 commit
  - 每個 commit 只能是自動化重構結果或是手動重構結果其中之一，不能一起做。