

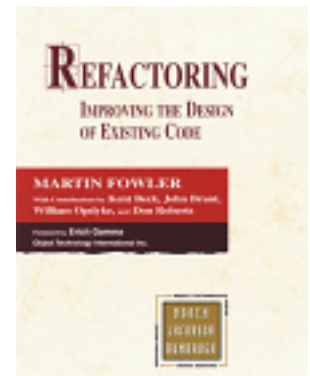
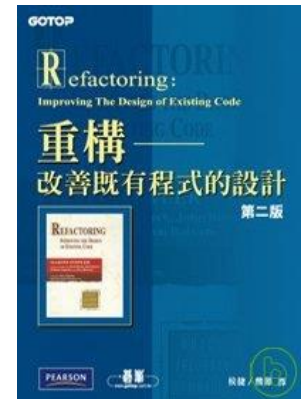
# Part IV 程式碼的壞味道

成大資訊系 李信杰 副教授 編修

取自99年度教育部資訊軟體人才培育推動中心軟體工程推廣分項計畫  
編輯委員 國立台北科技大學 劉建宏教授  
陳偉凱教授  
郭忠義教授

# 程式中的臭味

- 程式碼中有「不好」的寫法時，我們會說這段程式碼很臭(bad smell)
- 如何找出臭的程式碼
  - 同儕審查(Peer Review)



# Unresolved warnings

- 仍然可以產生可執行檔，並且可以執行
- 是一種隱性的錯誤
  - 雖然不是語法上的錯誤，但是在執行期間可能會發生無法預期的行為

```
1 public void printSomething() {  
2     int size = 3  
3     String target = null;  
4  
5     for(int i = 0; i < size; i++) {  
6         System.out.println("i = " + i);  
7     }  
8  
9     System.out.println(target.toString());  
10 }
```

15 Null pointer access: The variable target can only be null at this location

```
i = 0  
i = 1  
i = 2
```

Exception in thread "main" java.lang.NullPointerException  
at Examples.main(Examples.java:15)

# Long method(1/2)

- 一個函式應該只負責一項任務
- 過長函式的問題
  - 過長的函式不容易看得懂
  - 過長的函式難以重複利用
  - 過長的函式難以維護
  - 不容易取名
- 一個函式長度應該要小於一個螢幕的可視範圍
  - 約20行
  - 超過一個螢幕就需要經常捲動畫面了解上下文

# Long method(2/2)

- 很難有合適的函式名稱
- 使用 *Extract Method* 將之拆解成短函式

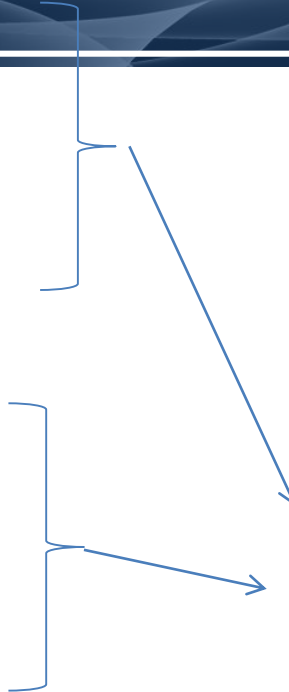
```
1 public void createPartControl(Composite parent) {  
2     _failnodes = new HashSet<Object>();  
3     _comps = new ConcurrentLinkedQueue<IComponent>();  
4     _viewer = new TreeViewer(parent, SWT.MULTI | SWT.H_SCROLL);  
5     _viewer.setInput(getViewSite());  
6     ...  
59     _selectionHandler = new SelectionChangHandler();  
60     _selectionHandler.setViewer(_viewer);  
61 }
```

```
Person person1 = new Person();  
person1.setName("tom");  
person1.setAge(20);  
person1.setTel("1234567");  
person1.setAddress("abcabcabc abcabcabc");
```

```
Person person2 = new Person();  
person2.setName("mary");  
person2.setAge(30);  
person2.setTel("7654321");  
person2.setAddress("abcabcabc abcabcabc");
```

```
List<Person> personList = new ArrayList<Person>();  
personList.add(person1);  
personList.add(person2);
```

```
for (Person person : personList)    {  
    System.out.println(person.getName());  
}
```



Public static **createPerson**(  
String name,  
int age, String tel,  
String address);

# Feature envy

- 某函式對於某個類別(Class)的興趣高於本身所處的類別
  - 經常需要取得另一個類別的多個變數或是呼叫多個函式

```
1  Public ClassB(){  
2      public void doSomething() {  
3          ClassA a = new ClassA();  
4          int x = a.getX();  
5          int y = a.getY();  
6          int z = a.calculateSomething(x + y, y);  
7          a.setZ(z);  
8      }  
9  }
```

```
1  public ClassA() {  
2      getX(){...}  
3      getY(){...}  
4      calculateSomething(a,b){...}  
5      setZ(){...}  
6  }
```

# Feature envy

- 使用*Move Method*將該函式移至另一個類別

```
1 public ClassA() {  
2     getX(){...}  
3     getY(){...}  
4     calculateSomething(a,b){...}  
5     setZ(){...}  
6     public void doSomething() {  
7         ...  
8     }  
9 }
```



# Unsuitable naming

- 好的命名令人易讀易懂
  - Class name
  - Member method, Member data
  - Local variable

```
1 public class T() {  
2     boolean b = false;  
3  
4     public int xyz(int x, int y, int z) {  
5         int r = 0;  
6         r = (x + y) * z / 2;  
7         return r;  
8     }  
9 }
```

```
1 public class Trapezoid() {  
2     boolean isIsosceles = false;  
3  
4     public int calculateArea(int top, int bottom, int height) {  
5         int area = 0;  
6         area = (top + bottom) * height / 2;  
7         return area;  
8     }  
9 }
```

# All assigned variables have proper type consistency or casting

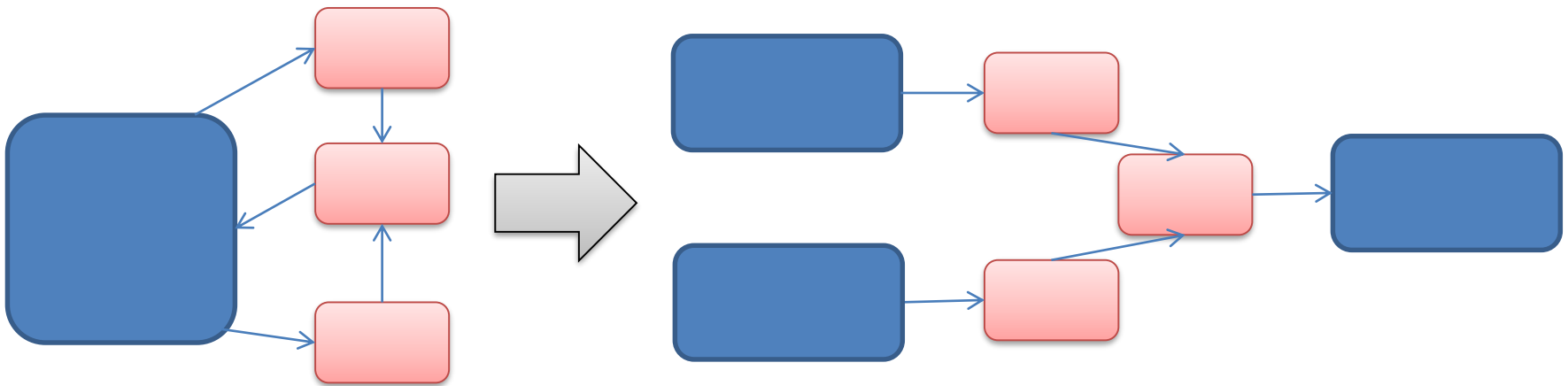
- 強制轉型(Casting)是一種危險的語言構件
- 除非你很清楚欲轉換的型別，否則應盡量避免使用
- 因變數型態不一致，可能造成無法預期的執行結果

```
1 void testType() {  
2     unsigned short x = 65535;  
3     short y = x;  
4  
5     for(int i = 0; i < y; i++) {  
6         Do something  
7     }  
8 }
```

```
public static void casting()  
{  
    int x = 65535;  
    System.out.println(x);  
    short y;  
    y = (short) x;  
    System.out.println(y);  
}
```

# Are any modules excessively complex and should be restructured

- 太過複雜的模組，可能使得該模組與其他模組之間產生不必要的 Coupling
  - 應將過於龐大的模組拆解成數個小模組



# Loop termination conditions are obvious and invariably achievable

- 迴圈的設計步驟
  - 設定初始條件
  - 執行過程中，逐步修改初始條件
  - 謹慎定義結束條件

```
1 for(int i = 1; (i % 2) ? ((i + 100) < 200) : ((i * 30) < 50); i++) {  
2     Do something  
3 }
```

```
4  
5 for(int i = 0; i < 100; i++) {  
6     Do something  
7     i = i * 5;  
8 }
```

```
9  
10 int i = 0;  
11 while(i < 10) {  
12     Do something  
13 }
```

```
static int i = 10;  
  
public static void loopTerm()  
{  
    while (i > 0)  
    {  
        i -= 1;  
        someBadProc();  
    }  
    System.out.println("terminated.");  
}  
  
private static void someBadProc()  
{  
    i += 1;  
}
```

```
1 for(int i = 1; i < 10; i++) {  
2     Do something  
3 }  
4  
5 for(int i = 0; i < 100; i++) {  
6     Do something  
7 }  
8  
9  
10 int i = 0;  
11 while(i < 10) {  
12     Do something  
13     i++;  
14 }
```

# Parentheses are used to avoid ambiguity

- 適當的運用括號
  - 增加可讀性
  - 可避免邏輯上的錯誤

```
1 public int trapezoidArea(int top, int bottom, int height) {  
2     int area = top + bottom * height / 2;  
3     return area;  
4 }  
5  
6 if (isOk && getX() *  
7     Do something  
8 }
```

```
1 public int trapezoidArea(int top, int bottom, int height) {  
2     int area = (top + bottom) * height / 2;  
3     return area;  
4 }  
5  
6 if ((isOk) && (getX() * getY() == 2000) && (!isFinished)) {  
7     Do something  
8 }
```

# Lack of comments(1/2)

- 良好的註解可提高可讀性
- 註解撰寫時機
  - 需要產生 API 文件(Java Doc 式的註解)
  - 複雜的演算法
  - 輔助設計與思考
  - 記錄這段程式碼可能缺陷
  - 註解決策原因

# Lack of comments(2/2)

```
1 public RSSIMapCollection() {  
2     _maps = new Hashtable<String, RSSIMap>();  
3     _listeners = new Vector<RSSIMapCollectionEventListener>();  
4     _stabilizes = new SelectionProperty(STABILIZES_LABEL);  
5     _stabilizes.addElement(Stabilize.NONE);  
6     _stabilizes.addElement(Stabilize.THRESHOLD);  
7     _stabilizes.addEle  
8     _stabilizes.addEle  
9     _stabilizes.setSel  
10 }
```

```
1 public RSSIMapCollection() {  
2     _maps = new Hashtable<String, RSSIMap>();  
3     _listeners = new Vector<RSSIMapCollectionEventListener>();  
4  
5     // Initialize a selection property for multiple stabilizations  
6     _stabilizes = new SelectionProperty(STABILIZES_LABEL);  
7     _stabilizes.addElement(Stabilize.NONE);  
8     _stabilizes.addElement(Stabilize.THRESHOLD);  
9     _stabilizes.addElement(Stabilize.AVERAGE);  
10    _stabilizes.addElement(Stabilize.WIEGHTED);  
11    _stabilizes.setSelectedItem(Stabilize.THRESHOLD);  
12 }
```

# 註解和程式碼不一致

```
/**  
 * Count the profile of a company based on tax rate  
 * @param rate  
 */  
public static void countProfit(int amount, double rate)  
{  
    //do something  
}
```



- 不要做沒有意義的註解

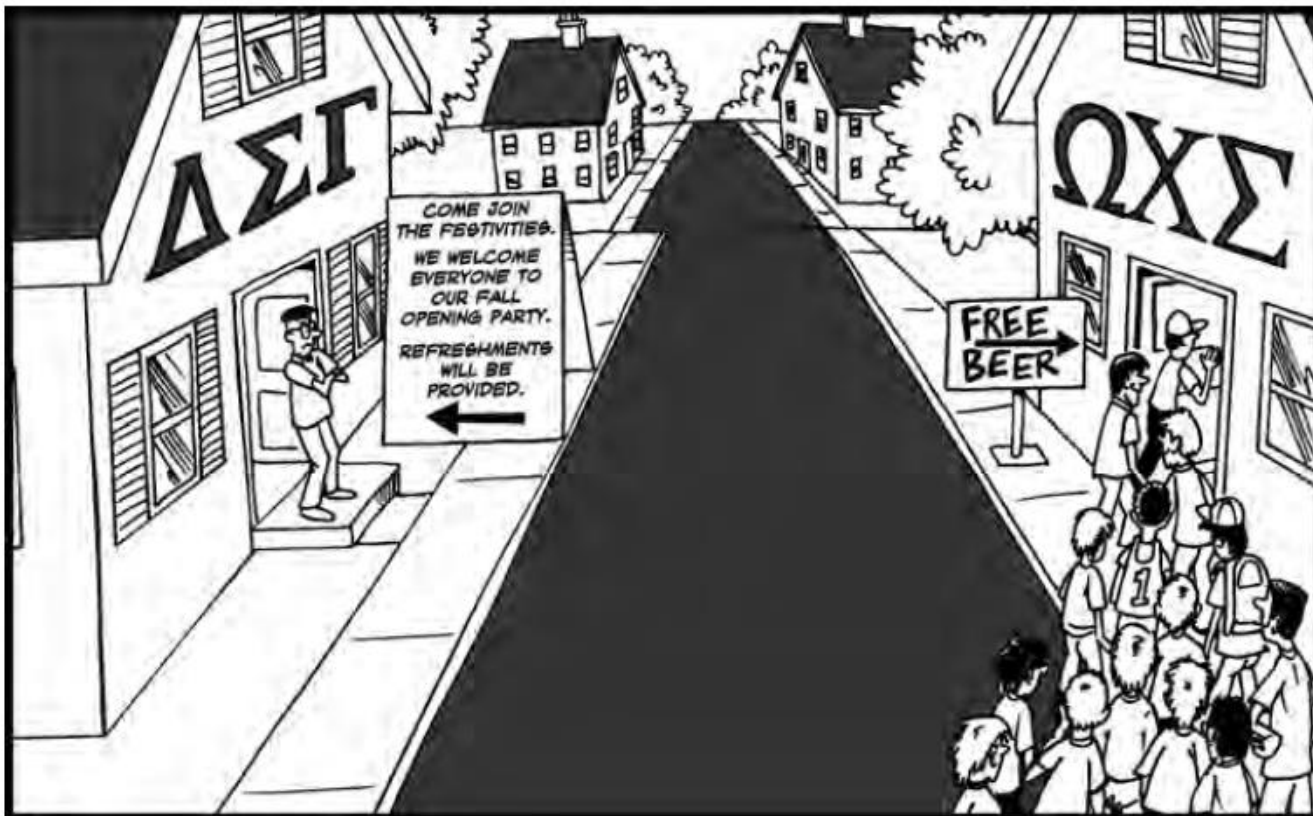
```
// The class definition for Account
class Account {
    public:
        // Constructor
        Account();

        // Set the profit member to a new value
        void SetProfit(double profit);

        // Return the profit from this Account
        double GetProfit();
};
```

# 註解

- 註解要力求精確與簡潔



- 不要為了註解而註解
  - 以下這段註解所傳達的資訊從Method名稱就可以看出来了

```
// Find the Node in the given subtree, with the given name, using the given depth.  
Node* FindNodeInSubtree(Node* subtree, string name, int depth);
```

- 需要註解的方法名稱，可能代表該方法命名有問題

```
// Releases the handle for this key. This doesn't modify the actual registry.  
void DeleteRegistry(RegistryKey* key);
```



```
void ReleaseRegistryHandle(RegistryKey* key);
```

# Duplicated Code (1/2)

- 一段段重複的程式碼在一個或多個類別或方法中出現。

```
1 public class ClassAReport {  
2     ...  
3     public int calculateAverage(List<Integer> scores) {  
4         int sum, average = 0;  
5         for (int i = 0; i < scores.size(); i++) {  
6             sum += scores.get(i);  
11        }  
20        average = sum / scores.size();  
21        return average;  
22    }  
23    ...  
}
```

```
1 public class ClassBReport {  
2     ...  
3     public int calculateAverage(List<Integer> scores) {  
4         int sum, average = 0;  
5         for (int i = 0; i < scores.size(); i++) {  
6             sum += scores.get(i);  
11        }  
20        average = sum / scores.size();  
21        return average;  
22    }  
23    ...  
}
```

This piece of code occurs more than once!

# Duplicated Code (2/2)

- 重複的程式碼應當抽出來改寫成類別或是副程式讓其他程式得以共享。

```
1  public class classAReport {  
2      private List<Integer> classAScores;  
3      ... // initialize scores  
4      public int calculateAverage (AverageCalculator ac) {  
5          retrun ac.calculateAverage(classAScores);  
6      }  
7      // Another Class  
8      public class classBReport {  
9          private List<Integer> classBScores;  
10         ... // initialize scores  
11         public int calculateAverage (AverageCalculator ac) {  
12             retrun ac.calculateAverage(classBScores);  
13         }  
14     }  
15 }
```

possible  
average.

the math.

# All methods have appropriate access modifiers and return types (1/2)

- 副程式都沒有加上適當的access modifier，使其他類別可以無限制的存取該類別應當保護的資料或是副程式。
  - 應按照設計原則 (如Information Hiding)替成員函式程式以及成員變數冠上適當的access modifier.

```
1 Class Account {  
2   public:  
3     string _password;  
4     string getPassword();  
5     ...  
};
```

```
1 Class Account {  
2   public:  
3     string getPassword();  
4     ...  
5   private:  
6     string _password;  
7     ...  
};
```



# All methods have appropriate access modifiers and return types (2/2)

- 副程式都沒有加上適當return type，無法得知副程式的執行是否正常。(C++ as example language)

```
1 int main() {  
2     1 bool openA  
3     2 ifstream i  
4     3 ifs.open(f  
5     4 if (!ifs.is_  
6     5 return  
7     ...  
8     10 return tru  
9     11 }  
10  
11 }
```

```
1 Int main() {  
2     string filePath;  
3     cin >> filePath;  
4     bool isFileOpenedAndProcessed;  
5     isFileOpenedAndProcessed =  
6         openAndProcessFile(filePath);  
7     if(isFileOpenedAndProcessed) {  
8         ... // do something  
9     } else {  
10        ... // error handling code  
11    }  
12 }
```

Client checks if the  
file is opened &  
processed.



# Are there any redundant or unused variables?

- 副程式 / 類別內存在多餘的變數。這些變數至始至終都沒有被存取及使用過。
  - 刪除副程式或類別中未曾使用存取過的變數。

```
1 public int calculateClassAverage (List<Integer> scores) {  
2     int rank = 0; // never used  
3     int sum, average = 0;  
4     for (int i = 0; i < scores.size(); i++) {  
5         sum += scores.get(i);  
6     }  
7     return average;  
8 }
```


```
1 public int calculateClassAverage (List<Integer> scores) {  
2     int sum, average = 0;  
3     for (int i = 0; i < scores.size(); i++) {  
4         sum += scores.get(i);  
5     }  
6     return average;  
7 }
```

Delete unused  
variable


# Indexes or subscripts are properly initialized, just prior to the loop

- 迴圈下標或條件值無適當地初始化。
  - 迴圈下標或條件值應適當地初始化。

```
1 int i;  
2 while (i < 0) {  
3     doSomething();  
4     i++;  
5 }
```

```
1 int i = -10;  initialized  
2 while (i < 0) {  
3     doSomething();  
4     i++;  
5 }
```

```
1 int i;  
2 for (i ; i < someInt; i++) {  
3     doSomething();  
4 }
```

```
1 int i = 0;  initialized  
2 for (i ; i < someInt; i++) {  
3     doSomething();  
4 }
```

# Are divisors tested for zero?

- 做除法運算未檢查除數是否為0
  - 做除法運算前應檢查除數是否為0

```
1  int divisor;  
2  int dividend;  
3  cin >> divisor;  
4  cin >> dividend;  
5  int quotient = dividend / divisor;  
6  ...  
}
```

```
1  int divisor;  
2  int dividend;  
3  cin >> divisor;  
4  cin >> dividend;  
5  
6  if (divisor == 0) {  
7      throw "divisor is 0";  
8  }  
9  int quotient = dividend / divisor;  
10 ...  
}
```

# Inconsistent coding standard

- 程式碼與規定的程式碼標準不一致
  - 程式碼應與規定的程式碼標準一致

- 1 成員變數名稱前應加底線。
- 2 務必使用有意義的命名。

```
1 class Car {  
2 public:  
3   int getAbc();  
4   string getXyz();  
5   ...  
6 private:  
7   int id;  
8   string manufactureDate;  
9   ...  
10  };  
    
```

meaningless naming

Inconsistent coding standard

```
1 class Car {  
2 public:  
3   int getVehicleId ();  
4   string getManufactureDate();  
5   ...  
6 private:  
7   int _id;  
8   string _manufactureDate;  
9   ...  
10  };  
    
```

# Data clumps

- 指常常在不同地方成群出現的相同資料項

```
1 public class Customer {  
2     private String name;  
3     private String title;  
4     private String house;  
5     private String street;  
6     private String city;  
7     private String postcode;  
8     private String country;  
9     ...  
10 }
```

```
1 public class Staff {  
2     private String lastname;  
3     private String firstname;  
4     private String house;  
5     private String street;  
6     private String city;  
7     private String postcode;  
8     private String country;  
9     ...  
10 }
```

# Data clumps

- 指常常在不同地方成群出現的相同資料項

```
1 public class Address {  
2     private String house;  
3     private String street;  
4     private String city;  
5     private String country;  
6     ...  
7 }
```

```
1 public class Customer {  
2     private String name;  
3     private String title;  
4     private Address customerAddr;  
5  
6  
7  
8     ...  
9 }
```

```
1 public class Staff {  
2     private String lastname;  
3     private String firstname;  
4     private Address staffAddr;  
5  
6  
7  
8     ...  
9 }
```

# Large class

- 通常發生在一個類別作了太多的事情，必須將這種類別去做妥善的分工。

```
1 public class 班長() {  
2     public void 辦班遊() {  
3         ...  
4         規劃行程();  
5         收錢();  
6         ...  
7     }  
8     public void 維持秩序() {...}  
9     public void 規劃行程() {...}  
10    public void 收錢() {...}  
11    ...  
12 }
```

# Large class

- 通常發生在一個類別作了太多的事情，必須將這種類別去做妥善的分工。

```
1 public class 班長() {  
2     public void 辦班遊() {  
3         ...  
4         康樂.規劃行程();  
5         總務.收錢();  
6         ...  
7     }  
8     public void 維持秩序() {  
11         風紀.維持秩序();  
12     }  
13 }
```

```
1 public class 風紀() {  
2     public void 維持秩序() {  
3         ...  
4     }  
5 }  
6 public class 總務() {  
7     public void 收錢() {  
8         ...  
9     }  
10 }  
11 public class 康樂() {  
12     public void 規劃行程() {  
13         ...  
14     }  
15 }
```



# Long parameter list

- 建議可以使用 Introduce Parameter Object 重構方法將過長的參數包裝成一個「參數物件」。
- Not good:

```
1 public class Member {  
2     public createMember(  
3         Name name,  
4         String country,  
5         String postcode,  
6         String city,  
7         String street,  
8         String house) {  
9         ...  
10    }  
11 }
```

- Better solution:

```
1 public class Member {  
2     public createMember(  
3         Name name,  
4         Address address) {  
5         ...  
6     }  
7 }
```

# 太多參數

- 參數數目若超過4個就太多了！
- 如果發現同一組參數重覆出現在很多方法，就該考慮將這些參數組成新類別

# 太多分支

```
public static void tooManyIf(String x)
{
    if (x.charAt(0) == 'A')
    {
        if (x.charAt(1) == 'B')
        {
            if (x.charAt(2) == 'C')
            {
                System.out.println(x);
            } else if (x.charAt(2) == 'D')
            {
                System.out.println(x);
            }
        }
    }
}
```

# Literal constants

- 應該先用(static) const或define關鍵字(視語言而異)定義常數，然後再使用之。
- Not good:

```
1 public double potentialEnergy(double mass, double height) {  
2     return mass * 9.81 * height;  
3 }
```

- Better solution:

```
1 public double potentialEnergy(double mass, double height) {  
2     final static double GRAVITATION = 9.81;  
3     return mass * GRAVITATION * height;  
4 }
```

# 沒有定義常數

- 應該先用public static final關鍵字(視語言而異)定義常數，然後再使用之

```
public double noLiteralConstant(double radius) {  
    return 2 * 3.14 * radius  
}
```



```
public static final double PI = 3.14;
```

```
public double literalConstant(double radius) {  
    return 2 * PI * radius  
}
```

# Every variable is properly initialized

- 在寫程式時總是會宣告一些必須使用的變數，每個變數都需要有正確的初始化。
- Not good:

```
1 Person person;  
2 Manager = person.getManager();  
3 int workHours, hourlyWage;  
4 Int salary = workHours * hourlyWage;
```

- Better solution:

```
1 Person person = new Person();  
2 Manager = person.getManager();  
3 int workHours = 40, hourlyWage = 120;  
4 Int salary = workHours * hourlyWage;
```

# There are uncalled or unneeded procedures or any unreachable code

- 任何 uncalled、unneeded、unreachable 的程式存在在專案中，總是會造成閱讀上的不便和負擔，也會讓程式太過冗長，浪費不必要的空間。

# There are uncalled or unneeded procedures or any unreachable code

```
1  if(i < 60) {  
2    //unreachable  
3    if(i == 60) {  
4      System.out.println("PASS");  
5    }  
6    else{  
7      System.out.println("NOT PASS");  
8    }  
9  }  
10 else{  
11   System.out.println("PASS");  
12 }
```

```
1  public class Client {  
2    public createMember(Name name)  
3    {  
4      Name name = new Name();  
5      Member.createMember(name);  
6    }  
7  }
```

```
1  public class Member {  
2    public Member createMember(  
3      Name name  
4    ) {...}  
5    //uncalled or unneeded procedure  
6    public Member createMember(  
7      String lastName,  
8      String firstName,  
9    ) {...}  
10 }
```



# Does every switch statement have a default?

- 對於每個 switch-case ，都必須有宣告的 default 動作。
- Not good:
- Better solution:

```
1 switch(weekday) {  
2     case 'Monday':  
3         System.out.println("國文課");break;  
4     case 'Tuesday':  
5         System.out.println("英文課");break;  
6     case 'Thursday':  
7         System.out.println("數學課");break;  
8 }
```

```
1 switch(weekday) {  
2     case 'Monday':  
3         System.out.println("國文課");break;  
4     case 'Tuesday':  
5         System.out.println("英文課");break;  
6     case 'Thursday':  
7         System.out.println("數學課");break;  
8     default:  
9         System.out.println("休息");break;  
12 }
```

# The code avoids comparing floating-point numbers for equality

- 為了避免浮點數運算時的誤差，應該要盡力去避免做浮點數值的相等比較。
- Not good:

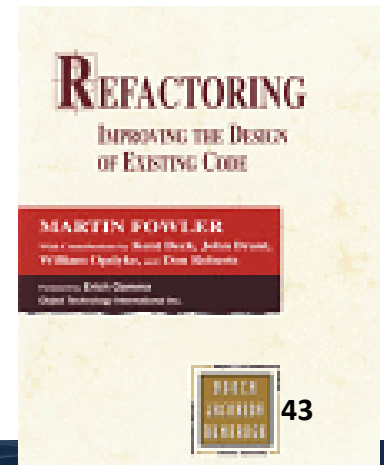
```
1 double x = 1e-10, y1 = 20e-10, y2 = 19e-10;  
2 double y = y1 - y2;  
3 if(x == y) {  
4     System.out.println("X == Y");//並不會成立  
5 }
```

- Better solution:

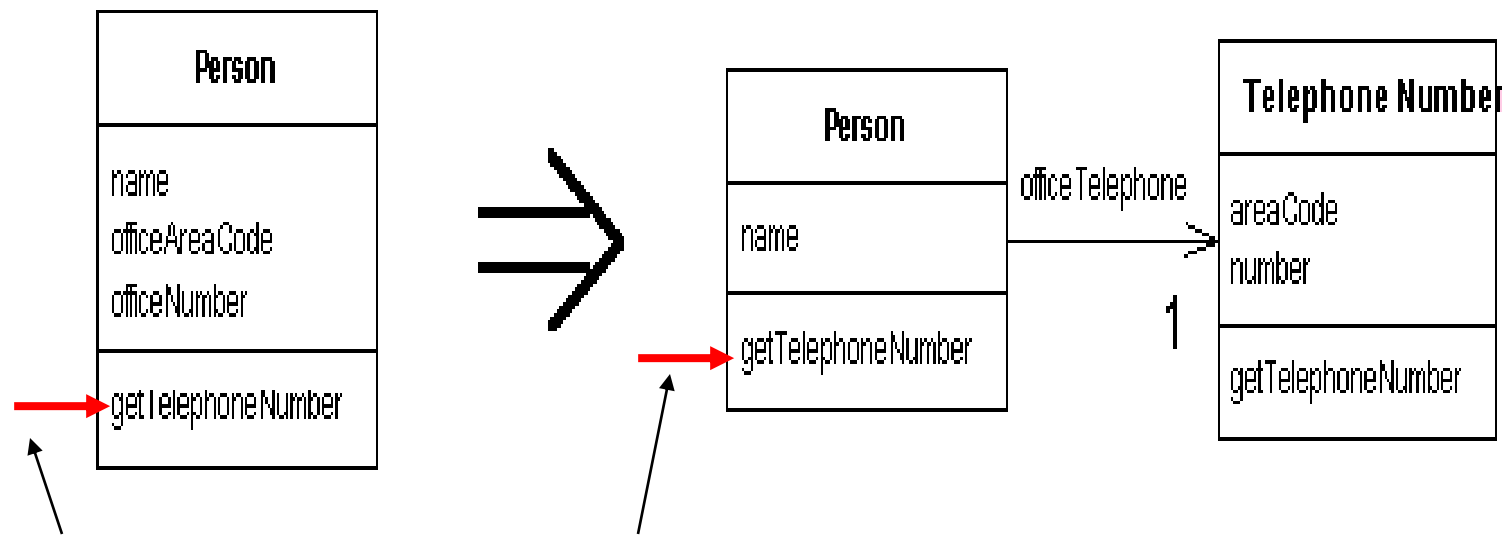
```
1 double x = 1e-10, y1 = 20e-10, y2 = 19e-10;  
2 double y = y1 - y2;  
3 if(Math.abs(x - y) < 1e-5) {  
4     System.out.println("X == Y");//成立  
5 }
```

# Refactoring

- 重構是什麼？
  - 由M. Fowler提出，改善程式碼臭味的方法論
  - 對軟體內部結構的一種調整，在不改變「外部可觀察到的行為」的前提下，提高軟體的可理解性，降低修改成本
  - 如何確保「外部可觀察到的行為」不變
    - 進行單元測試



# 實例化與多態性



外部使用的類別對其變化沒有感覺

# 為何要重構?

- 不重構軟體很快就會爛掉
- 增進可重用性
  - Design Patterns為重構提供了目標。
  - 有機會做抽象化並找出Framework。
- Lets you worry about generality tomorrow
  - 可以不需要在一開始就寫出漂亮的程式，可以先開發再重構

# 重構幫助設計師看到 更多設計層面的問題

- M.Fowler
  - 隨著程式碼漸趨簡潔，我發現自己可以看到一些以前看不到的設計層面的東西
  - 如果不對程式碼做這些修改，也許永遠看不到他們，因為我的聰明才智不足以在腦中把這一切都想像出來
- Ralph Johnson
  - 擦掉窗戶上的污垢，使你看得更遠

# 何時要進行重構？

- 功能增加時
- 修補Bug時
- Code Review之後
- 完成一個功能之後，又察覺bad smell時

# 如何重構

- 一次一小步: Make changes as small as possible.
  - Many small changes are easier than one big change.
- 每次改變都要測試: Test after each change



# Lab 4-1: 請找出可能的壞味道

```
1  import java.util.Random;
2
3  public class DiceGame
4  {
5      private Random randomNumbers = new Random();
6
7      public void play()
8      {
9          int myPoint = 0;
10         int gameStatus;
11
12         int die1 = 1 + randomNumbers.nextInt( 6 );
13         int die2 = 1 + randomNumbers.nextInt( 6 );
14
15         int sumOfDice = die1 + die2;
16
17         switch ( sumOfDice )
18         {
19             case 7:
20             case 11:
21                 gameStatus = 1;
22                 break;
23             case 2:
24             case 3:
25             case 12:
26                 gameStatus = -1;
27                 break;
```

```

28         default:
29             gameStatus = 0;
30             die1 = 1 + randomNumbers.nextInt( 6 );
31             die2 = 1 + randomNumbers.nextInt( 6 );
32             myPoint = die1 + die2;
33             System.out.printf( "Point is %d\n", myPoint );
34             break;
35     }
36
37     while ( gameStatus == 0 )
38     {
39         die1 = 1 + randomNumbers.nextInt( 6 );
40         die2 = 1 + randomNumbers.nextInt( 6 );
41
42         sumOfDice = die1 + die2;
43
44         if ( sumOfDice == myPoint )
45             gameStatus = 1;
46         else if ( sumOfDice == 7 )
47             gameStatus = -1;
48     }
49
50     if ( gameStatus == 1 )
51         System.out.println( "Player wins" );
52     else
53         System.out.println( "Player loses" );
54 }
55
56 public static void main( String args[] )
57 {
58     DiceGame game = new DiceGame();
59     game.play();
60 }
61 }

```

# Lab 4-2: 請找出可能的壞味道

```
1 public class Employee {
2
3     public String firstName;
4     public String midName;
5     public String lastName;
6     public String socialSecurityNumber;
7     public String employeeID;
8     public double monthlySalary;
9     public boolean isManager;
10    public int managerType;
11
12    public Employee(String first, String mid, String last, String ssn, String id, double pay, boolean isMgr, int mgrType) {
13        firstName = first;
14        midName = mid;
15        lastName = last;
16        socialSecurityNumber = ssn;
17        employeeID = id;
18        monthlySalary = pay;
19        isManager = isMgr;
20        managerType = mgrType;
21    }
22
23    public String getFirstName() {
24        return firstName;
25    }
26
27    public String getMidName() {
28        return midName;
29    }
30
31    public String getLastName() {
32        return lastName;
33    }
34
35    public String getSocialSecurityNumber() {
36        return socialSecurityNumber;
37    }
```

```

38 public String getEmployeeID() {
39     return employeeID;
40 }
41 public void setMonthlySalary(double salary) {
42     monthlySalary = salary < 0.0 ? 0.0 : salary;
43 }
44 public double getMonthlySalary() {
45     return monthlySalary;
46 }
47 public boolean isManager() {
48     return isManager;
49 }
50 public int getManagerType() {
51     return managerType;
52 }
53 public double getManagerBonus() {
54     if (managerType == 1) return 10000;
55     else if (managerType == 2) return monthlySalary * 0.1;
56     else return 6000;
57 }
58 public String toString() {
59     if (isManager)
60         return String.format("%s %s %s\nsocial security number: %s\nemployeeID: %s\nmonthly salary plus bonus: $%,.2f\n",
61             getFirstName(), getMidName(), getLastName(), getSocialSecurityNumber(), getEmployeeID(), getMonthlySalary());
62     else
63         return String.format("%s %s %s\nsocial security number: %s\nemployeeID: %s\nmonthly salary: $%,.2f\n",
64             getFirstName(), getMidName(), getLastName(), getSocialSecurityNumber(), getEmployeeID(), getMonthlySalary());
65 }
66 public static void main(String args[]) {
67     Employee employee1 = new Employee("Martin", "", "Fowler", "A123456789", "001", 5000, false, -1);
68     Employee employee2 = new Employee("Erich", "", "Gamma", "Z987654321", "002", 8000, true, 2);
69     System.out.println(employee1);
70     System.out.println(employee2);
71 }
72 }

```