CS2001
软件工程

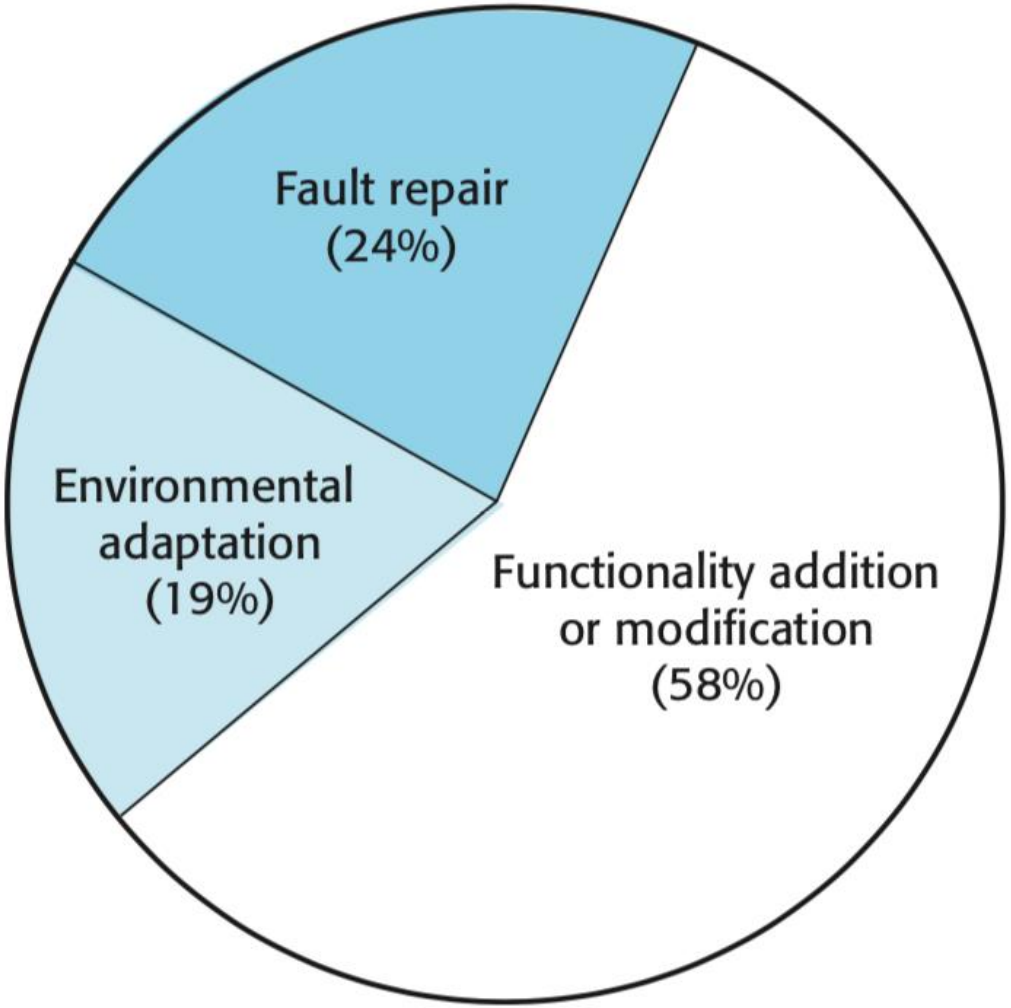
12. 软件设计
—演化式设计

课件来源　　复旦大学软件学院

软件演化

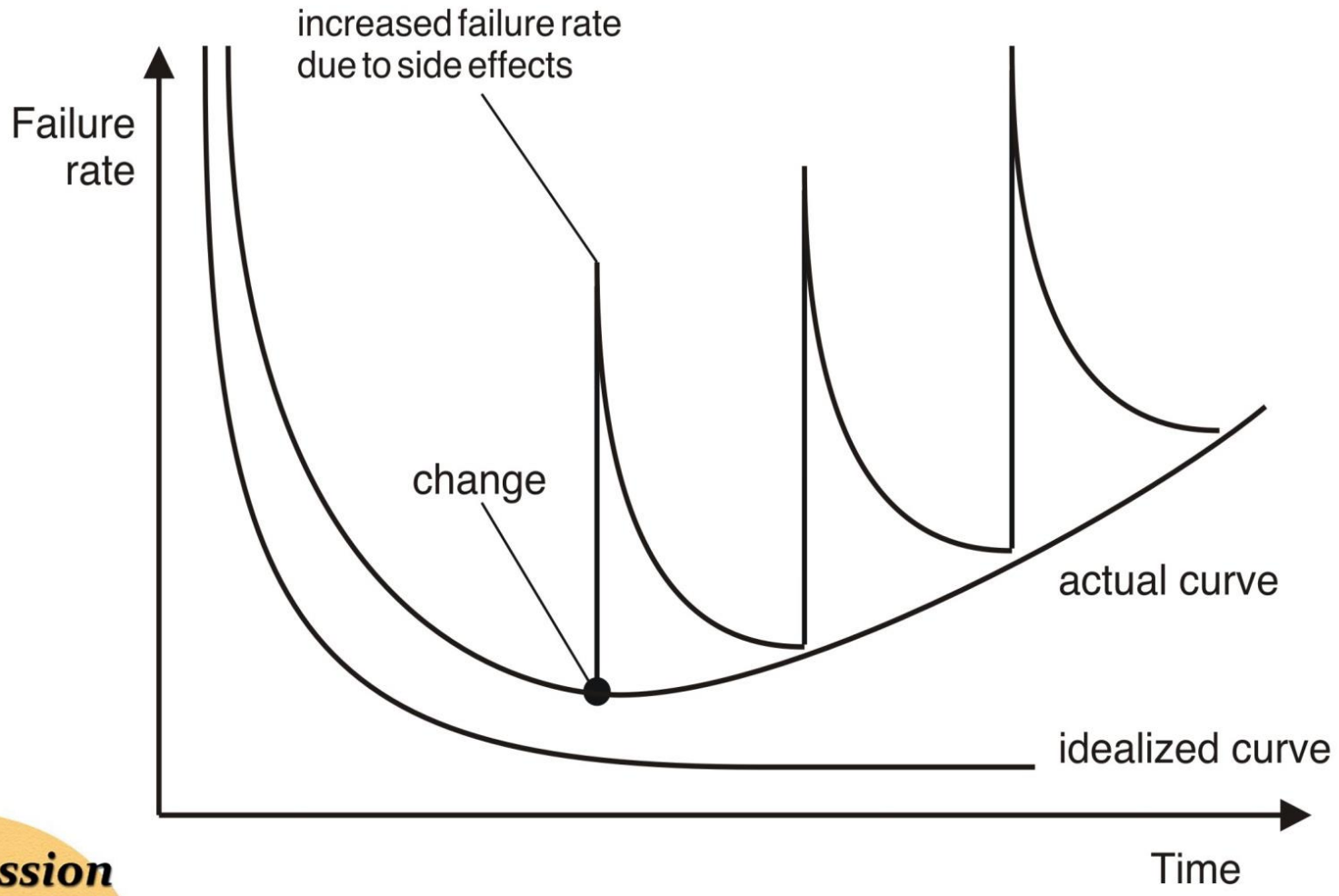
- 软件必须不断演化以保持“有用性”
  - ✓ 修复陆续发现并报告的问题
  - ✓ 满足客户/用户不断涌现的新需求
  - ✓ 适应不断变化的技术和业务环境
- 软件演化花费巨大
  - ✓ 60%-90%的软件花费是软件演化上的花费 (Lientz and Swanson 1980; Erlikh 2000)
  - ✓ 2006年美国的IT企业中有大约75%的工作都与软件演化有关，在可预见的将来还将继续保持此趋势 (Jones 2006)

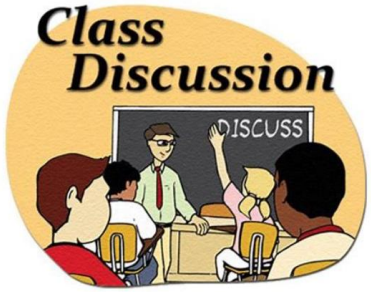
软件维护开销



软件初次交付之后的变更过程被称为**软件维护**

软件质量的退化





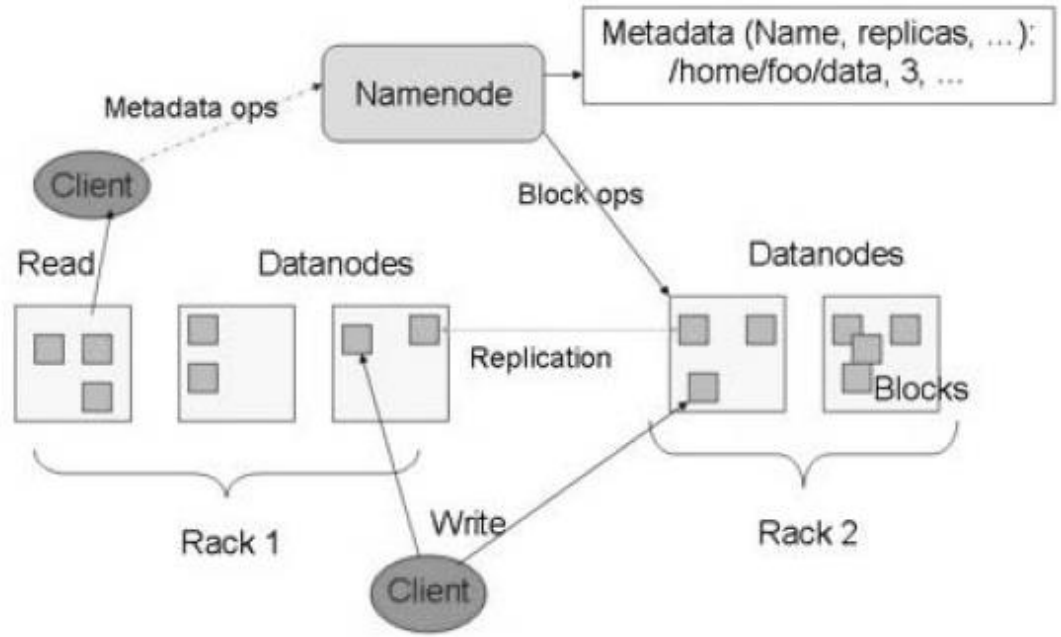
如何理解软件修改后的副作用？

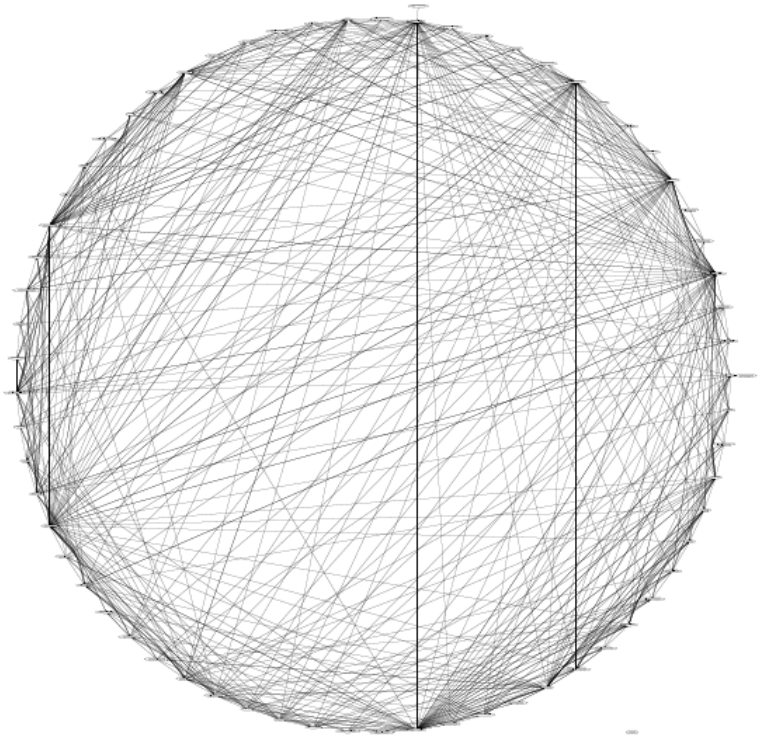
软件维护的冰山

我们希望那些一眼可见的就是所有实际存在的，但是我们知道在表面之下存在大量潜在的问题和成本。

Roger S. Pressman. Software Engineering: A Practitioner's Approach.

文档中的设计 Vs. 代码中的设计





Hadoop文档中的设计结构

Hadoop的实际实现结构

Nenad Medvidovic. Adapting Our View of Software Adaptation: An Architectural Perspective. SEAMS'14, June 2-3, 2014, Hyderabad, India.

冰山的形成（问题的累积）

- 时间和成本上的限制
- 预见性不足导致的不适应
- 文档缺失或未能同步更新
- 人员变动
- 责任意识（过渡心理）

冰冻三尺非一日之寒

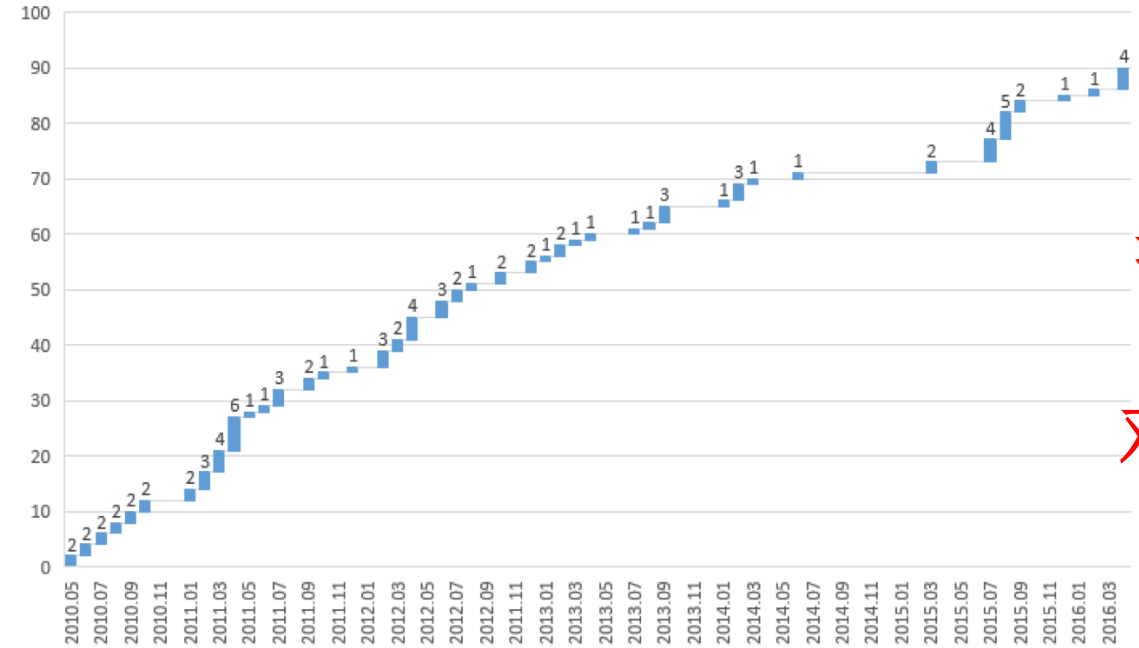
一个糟糕设计（Feature Envy）的形成

src/java/org/apache/cassandra/config/Config.java

```
@@ -34,6 +34,7 @@
 public Integer concurrent_reads = 8;
 public Integer concurrent_writes = 32;
+public Integer memtable_flush_writers = null; // will get set to the length of data dirs in
 public Double flush_data_buffer_size_in_mb = new Double(32);
 public Double flush_index_buffer_size_in_mb = new Double(8);
```

src/java/org/apache/cassandra/config/DatabaseDescriptor.java

```
@@ -218,7 +218,17 @@ else if (conf.disk_access_mode == Config.DiskAccessMode.mmap_index_only)
         {
             throw new ConfigurationException("concurrent_writes must be at least 2");
         }
+
+        /* Memtable flush writer threads */
+        if (conf.memtable_flush_writers != null && conf.memtable_flush_writers < 1)
+        {
+            throw new ConfigurationException("memtable_flush_writers must be at least 1");
+        }
+        else if (conf.memtable_flush_writers == null)
+        {
+            conf.memtable_flush_writers = conf.data_file_directories.length;
+        }
```



Feature Envy：方法对其他类的兴趣远大于自己所处的类的兴趣

开源分布式NoSQL数据库系统Cassandra中DatabaseDescriptor类对Config类依赖关系增长趋势

技术债（Technical Debt）

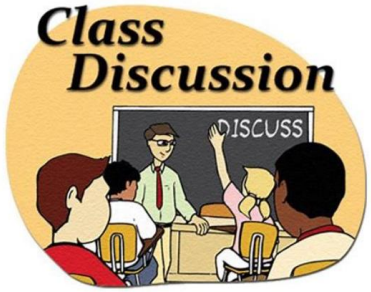
- 从短期效应的角度选择了一个易于实现的方案，但从长远来看则带来了更消极的影响
- 所积累的问题成为债务，每次维护所付出的额外代价只是“利息”
- 归还“本金”需要通过重构等手段消除问题

敏捷开发中的演化

- 一方面是传统的缺陷或需求/环境变化驱动的演化
- 另一方面是迭代式开发自身所带来的演化性
  - ✓ 短周期迭代：从数周到数月
  - ✓ 每次迭代都能产生可交付的软件制品
  - ✓ 每次迭代都是在上一次迭代交付物基础上的进一步开发
  - ✓ 演化式开发：开发与演化之间的界限进一步模糊

两种实施设计的方式

- 事先计划的设计（Planned Design）
  - ✓ 在实现开始前做好高层设计，然后交给另一组开发者进行实现
  - ✓ 相当于仔细画好图纸再施工
- 演化式设计（Evolutionary Design）
  - 计随着实现过程的进展而逐步发展
  - 相当于边盖房子边改设计图纸



这两种软件设计方式有什么优缺点？

两种软件设计方式各自的问题

- 事先计划的设计的问题
  - ✓ 无法对实现中可能遇到的所有问题进行预判，从而导致实现过程中对设计产生质疑
  - ✓ 设计人员和编码人员的“文化”和“技能”不完全相同
    - ■ 建筑业中设计人员和施工人员的技能界限很明确
    - ■ 软件业中开发人员很容易会优秀到足以质疑设计人员的设计
  - ✓ 几乎不可避免的需求变更要求设计具有灵活性
- 演化式设计的问题
  - ✓ 设计主要来自于一堆即兴的决定
  - ✓ 设计的贫乏使得软件越来越难以应对变化
  - ✓ 似乎回到了“编了再改”（Code and fix）的时代，随着时间的推移修复bug的成本越来越高

Martin Fowler

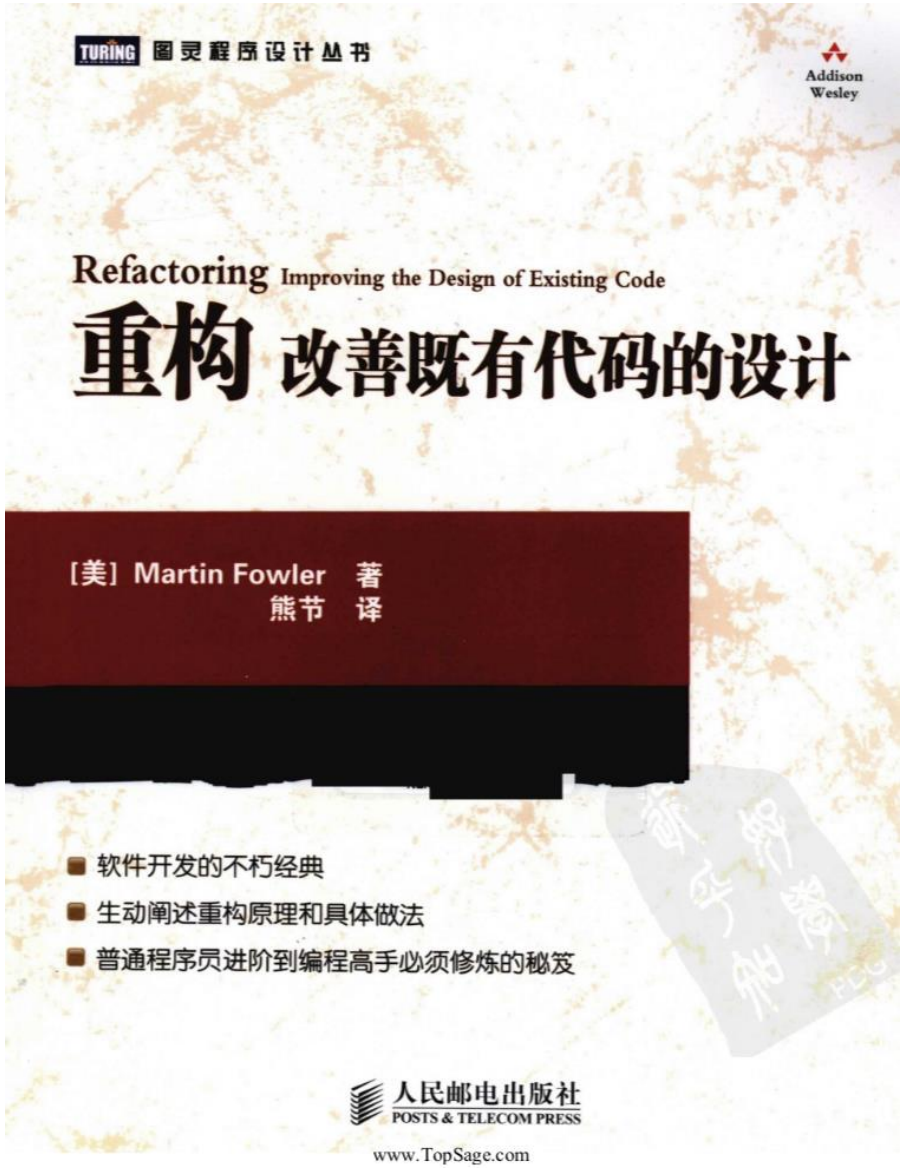
Martin Fowler. Is Design Dead? https://www.martinfowler.com/articles/designDead.html

寻找平衡点

- 抚平变更曲线（越晚的修改代价越高）
  - ✓ 软件测试：带来安全性
  - ✓ 持续集成：使团队保持同步，进行更改而不用担心与其他成员的集成问题
  - ✓ 软件重构：将设计决策推迟到演化过程中
- 在计划与演化式设计之间取得平衡
  - ✓ 事先计划的设计起主导作用是因为我们假设我们将很难在稍后改变主意
  - ✓ 当变化的成本降低后，我们可以将更多的设计作为重构推后进行
  - ✓ 没有完全抛弃事先计划的设计，只是找到了二者的平衡

当我应用重构之前，我就好像只用了一只手来做设计。

重构（Refactoring）

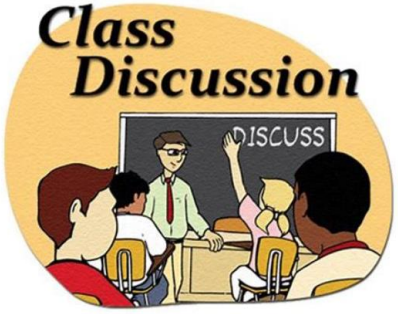


抚平曲线有多个可行的措施...重构也有同样的效果...这个巨大的变化驱使我写了关于它的整整一本书。

Martin Fowler

重构的定义

重构是以一种不改变代码外部行为、同时改进其内部结构的方式对软件系统进行修改的过程。

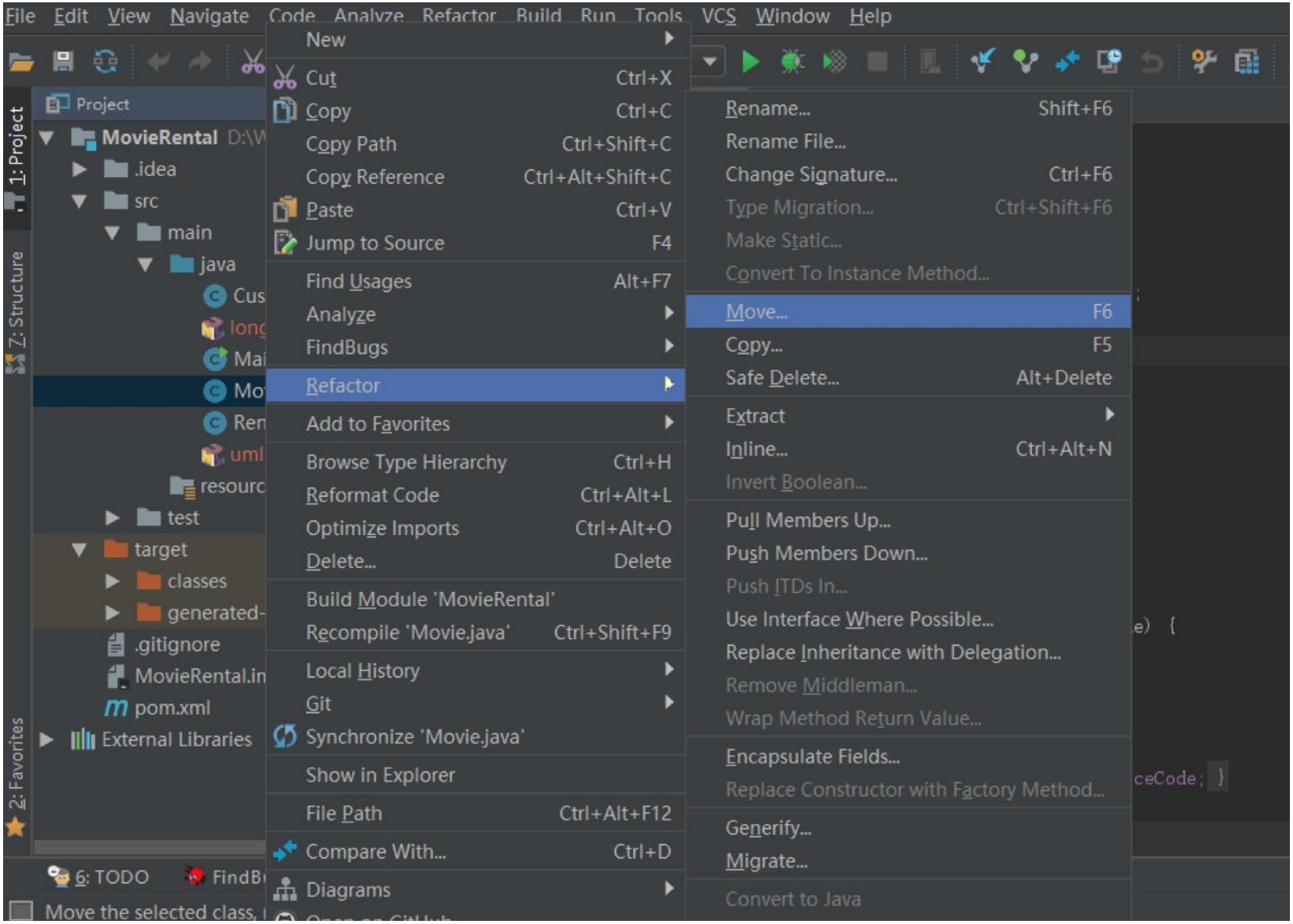


如何确保不改变代码外部行为？

常见的重构操作

- Rename Method/Field
- Move Method/Field
- Extract Class/Interface/Method
- Replace Temp with Query
- Introduce Parameter Object
- Pull Up/Push Down Members
- Replace Inheritance with Delegation

IntelliJ IDEA的重构支持



重构：消除代码坏味道（Bad Smell）

决定何时重构...去苏黎世拜访Kent Beck时，我正在为这个问题大伤脑筋。也许是因为受到刚出生的女儿的气味影响吧，他提出用味道来形容重构时机...我们看过很多代码，它们所属的项目从大获成功到奄奄一息的都有。观察这些代码时，我们学会了从中寻找某种特定结构，这些结构指出（有时甚至就像尖叫呼喊）重构的可能性。

— Martin Fowler

1. Duplicate Code
2. Long Method
3. Large class
4. Long Parameter List
5. Divergent Change
6. Shotgun Surgery
7. Feature Envy
8. Data Clumps
9. Primitive Obsession
10. Switch Statements
11. Parallel Inheritance Hierachies
12. Lazy Class
13. Speculative Generality
14. Temporary Field
15. Message Chains
16. Middle Man
17. Inappropriate Intimacy
18. Alternative Classes with Different Interfaces
19. Incomplete Library Class
20. Data Class
21. Refused Bequests
22. Comments

坏味道：重复代码（Duplicate Code）

- 相同的程序结构出现在不同地方
- 重构建议
  - ✓ 利用Extract Method重构将重复的代码提取出来，原方法中改为调用新提取的方法
  - ✓ 如果是兄弟子类之间的重复代码，那么提取方法后可以进一步利用Pull Up Method重构将其推到父类中
  - ✓ 如果是不完全相同的相似代码，那么先利用Extract Method重构将重复部分与差异部分区分开，然后对共性部分进行抽取和封装（例如提取参数）

坏味道：长方法（Long Method）

- 方法过长，承担了过多的职责
- 重构建议
  - ✓ 利用Extract Method重构对方法进行分拆
  - ✓ 如果存在大量参数和临时变量，那么需要利用Replace Temp with Query等重构减少临时变量，利用Introduce Parameter Object等重构缩短参数列表

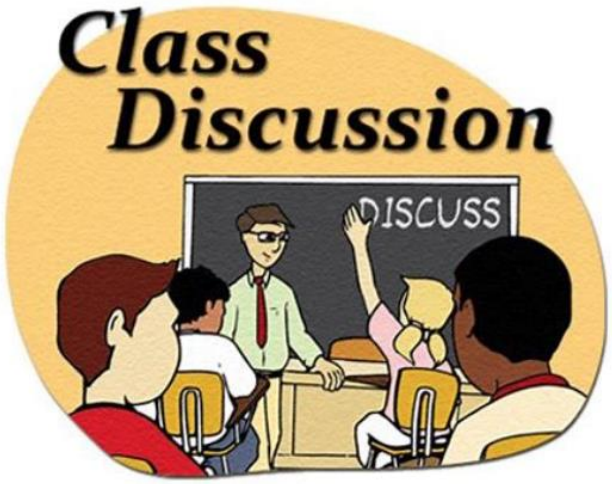
坏味道：发散式变化（Divergent Change）

- 一个类由于不同原因在不同的方向上发生变化
- 原因在于类的内聚度不高
- 重构建议
  - ✓ 对类进行分拆，使得所得到的每个类都只会由于一种原因进行修改
  - ✓ 针对每种不同的变化原因所影响的方法和属性，用Extract Class重构将它们提取出来

坏味道：霰弹式修改（Shotgun Surgery）

- 由于某种原因所导致的修改涉及许多不同的类中的小修改
- 需要修改的地方散落各处，修改难度大且容易遗漏
- 重构建议
  - ✓ 使用Move Method/Field重构将需要修改的代码放到同一个类中
  - ✓ 使用Inline Class重构将相关的行为放入同一个类

课堂讨论：发散式变化和霰弹式修改



课堂讨论：发散式变化和霰弹式修改有什么区别？

发散式变化：问题在混杂，即针对不同关注点的实现混杂到一起，出现在同一个类中

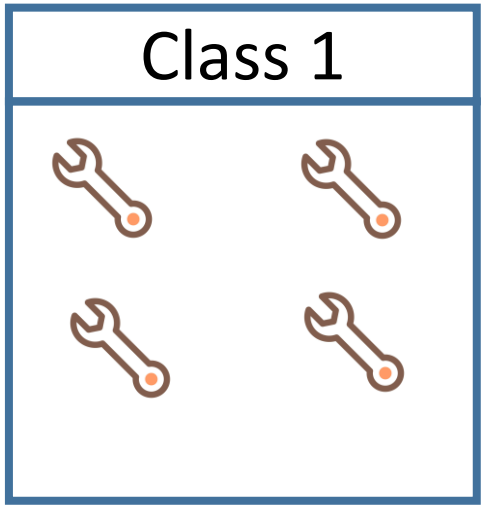
霰弹式修改：问题在于散布，即针对同一关注点的实现散布在多处，出现在多个类中

发散式变化和霰弹式修改

发散式变化

改下登录方式

改下欢迎消息

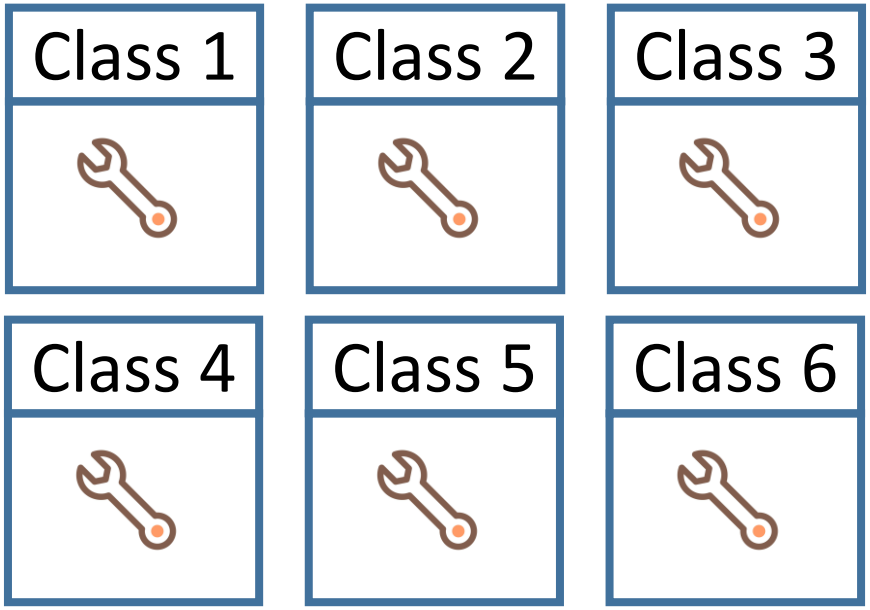


改下借书规则

改下催还方式

霰弹式修改

改下借书规则



重构操作：Extract Method

```
void printOwing() {

    Enumeration e = _orders.elements();
    double outstanding = 0.0;

    // print banner
    System.out.println ("**************************");
    System.out.println ("***** Customer Owes ******");
    System.out.println ("**************************");

    // calculate outstanding
    while (e.hasMoreElements()) {
        Order each = (Order) e.nextElement();
        outstanding += each.getAmount();
    }

    //print details
    System.out.println ("name:" + _name);
    System.out.println ("amount" + outstanding);
}
```

重构操作：Extract Method

```
void printOwing() {

    Enumeration e = _orders.elements();
    double outstanding = 0.0;

    printBanner();

    // calculate outstanding
    while (e.hasMoreElements()) {
        Order each = (Order) e.nextElement();
        outstanding += each.getAmount();
    }

    //print details
    System.out.println ("name:" + _name);
    System.out.println ("amount" + outstanding);
}
```

```
void printBanner() {
    // print banner
    System.out.println ("**************************");
    System.out.println ("***** Customer Owes ******");
    System.out.println ("**************************");
}
```

重构操作：Extract Method（含局部变量）

```
void printOwing() {

    Enumeration e = _orders.elements();
    double outstanding = 0.0;

    printBanner();

    // calculate outstanding
    while (e.hasMoreElements()) {
        Order each = (Order) e.nextElement();
        outstanding += each.getAmount();
    }

    //print details
    System.out.println ("name:" + _name);
    System.out.println ("amount" + outstanding);
}
```

重构操作：Extract Method（含局部变量）

```
void printOwing() {

    Enumeration e = _orders.elements();
    double outstanding = 0.0;

    printBanner();

    // calculate outstanding
    while (e.hasMoreElements()) {
        Order each = (Order) e.nextElement();
        outstanding += each.getAmount();
    }

    printDetails(outstanding);
}
```

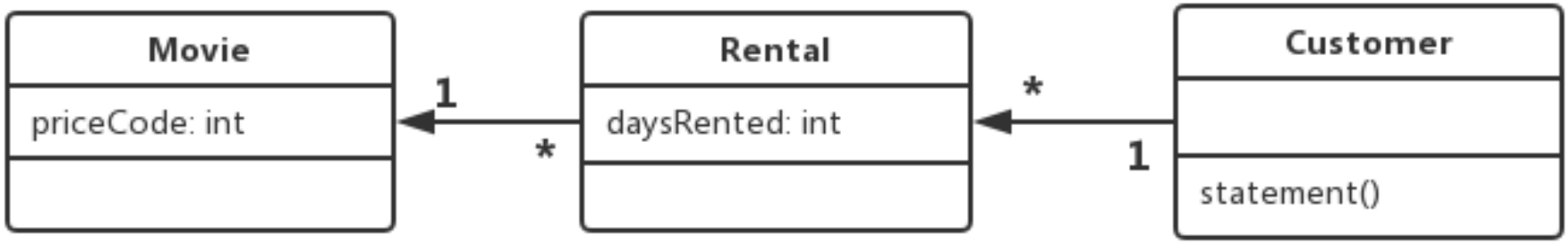
```
void printDetails (double outstanding) {
    System.out.println ("name:" + _name);
    System.out.println ("amount" + outstanding);
}
```

复杂重构

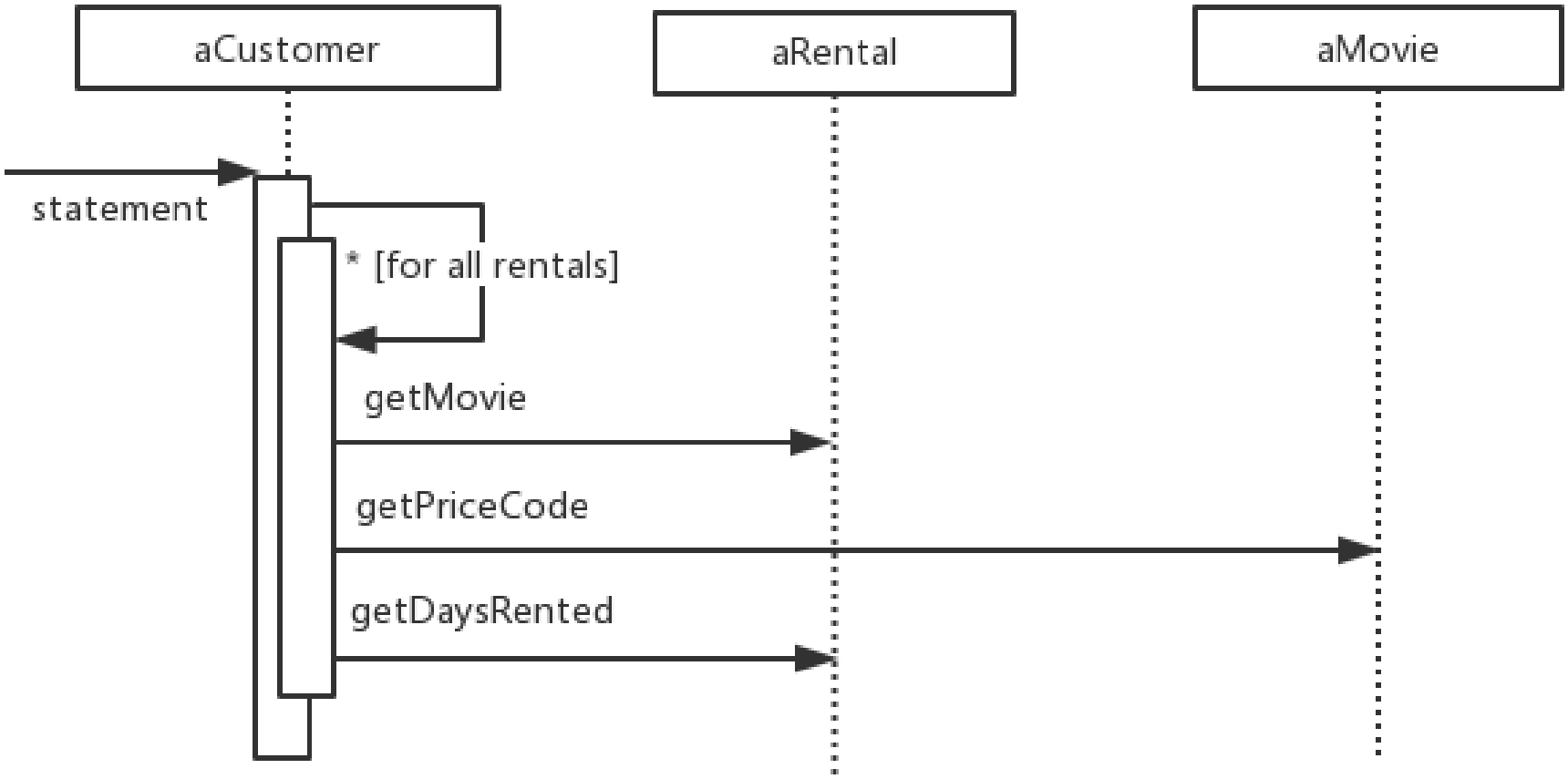
- 在相对大的范围内的整体重构
- 重构操作的执行会不断带来新的重构机会
- 需要考虑整体设计方案的优化，同时组合一系列基本重构操作

重构案例：电影租借系统

- Movie类：数据类，包括影片类型及相应的租赁费用
- Customer类：表示顾客信息，包含一个提供订单生成详单的方法statement()
- Rental类：表示一位顾客租借一部电影的相关信息



Statement方法交互过程

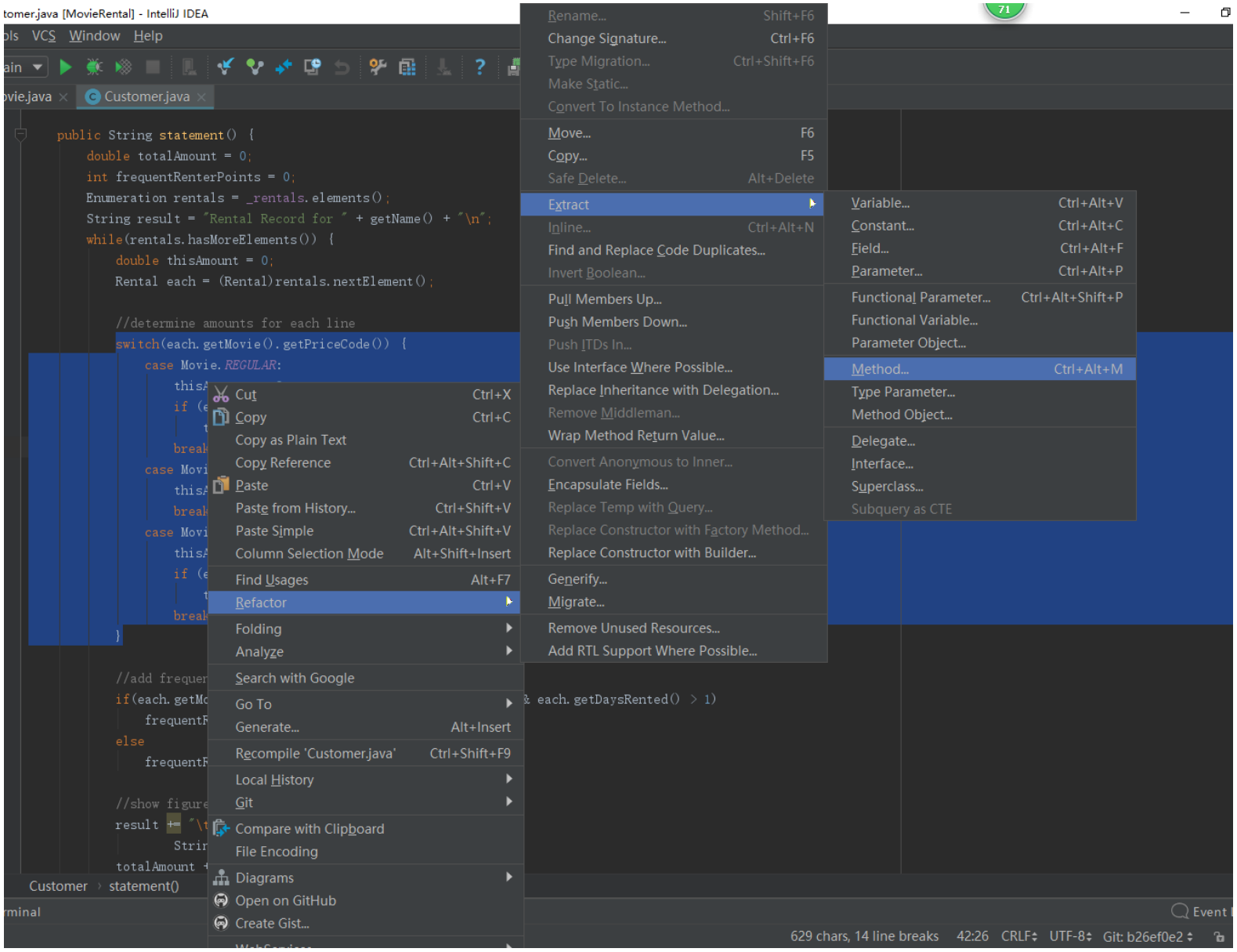


Customer类的Statement方法实现

```
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();
        //determine amounts for each line
        switch(each.getMovie().getPriceCode()) {
            case Movie.REGULAR:
                thisAmount += 2;
                if (each.getDaysRented() > 2)
                    thisAmount += (each.getDaysRented() - 2) * 1.5;
                break;
            case Movie.NEW_RELEASE:
                thisAmount += each.getDaysRented() * 3;
                break;
            case Movie.CHILDRENS:
                thisAmount += 1.5;
                if (each.getDaysRented() > 3)
                    thisAmount += (each.getDaysRented() - 3) * 1.5;
                break;
        }
        //add frequent renter points
        if(each.getMovie().getPriceCode() == Movie.NEW_RELEASE && each.getDaysRented() > 1)
            frequentRenterPoints += 2;
        else
            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;
}
```

复杂的switch语句，同时导致方法过长。可以考虑将其提炼为独立的方法。

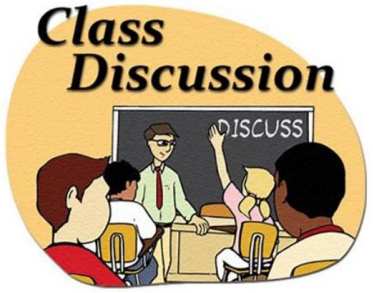
提取方法（Extract Method）重构



方法提取结果

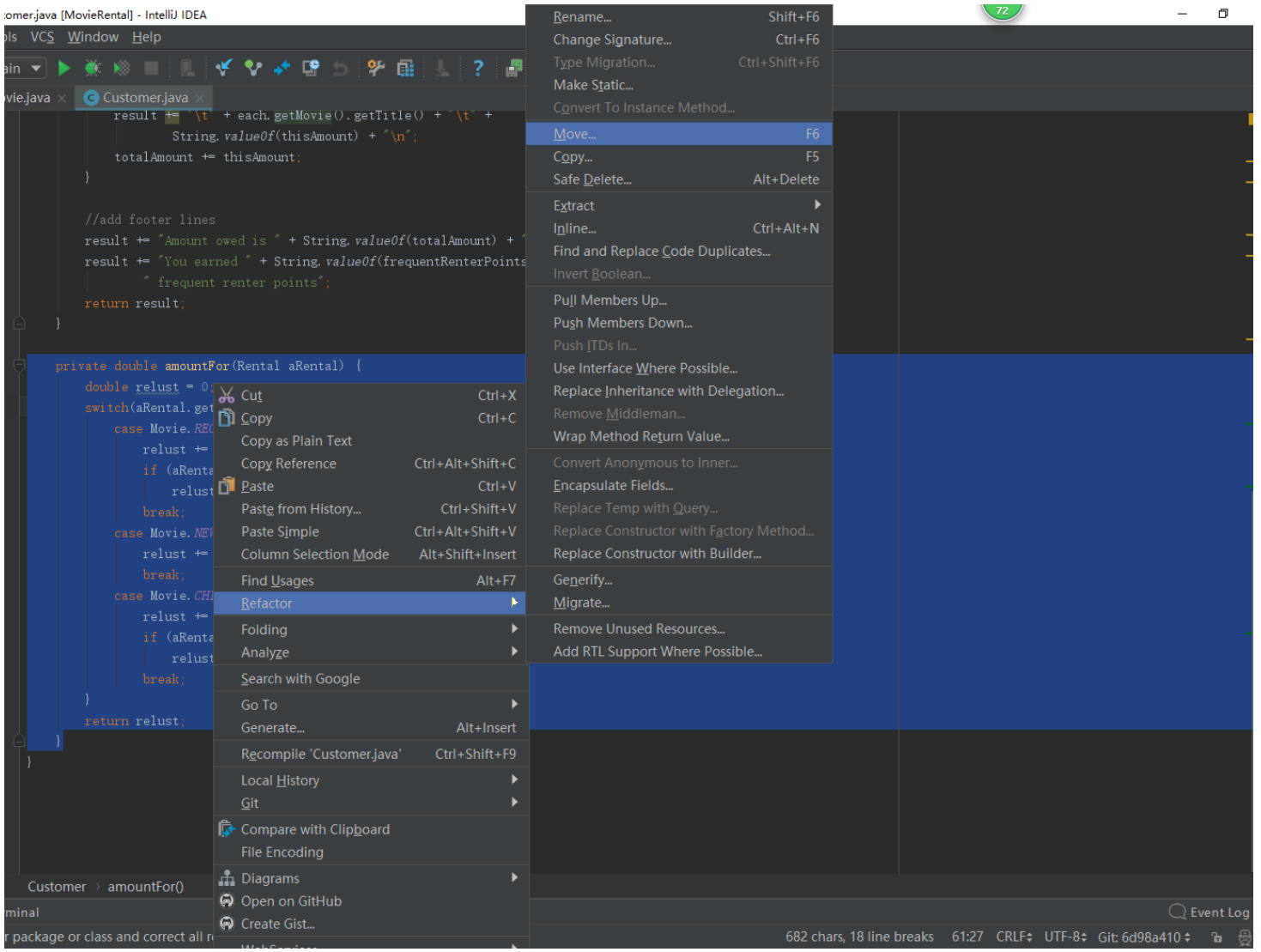
```
        //determine amounts for each line
        thisAmount = amountFor(each);

        //add frequent renter points
        if(each.getMovie().getPriceCode() == Movie.NEW_RELEASE && each.getDaysRented() > 1)
            frequentRenterPoints += 2;
        else
            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;
}

public double amountFor(Rental each) {
    double thisAmount = 0;
    switch(each.getMovie().getPriceCode()) {
        case Movie.REGULAR:
            thisAmount += 2;
            if (each.getDaysRented() > 2)
                thisAmount += (each.getDaysRented() - 2) * 1.5;
            break;
        case Movie.NEW_RELEASE:
            thisAmount += each.getDaysRented() * 3;
            break;
        case Movie.CHILDRENS:
            thisAmount += 1.5;
            if (each.getDaysRented() > 3)
                thisAmount += (each.getDaysRented() - 3) * 1.5;
            break;
    }
    return thisAmount;
```

该方法仅使用了Rental类的信息，故应该采用Move Method将该方法移到Rental类中。

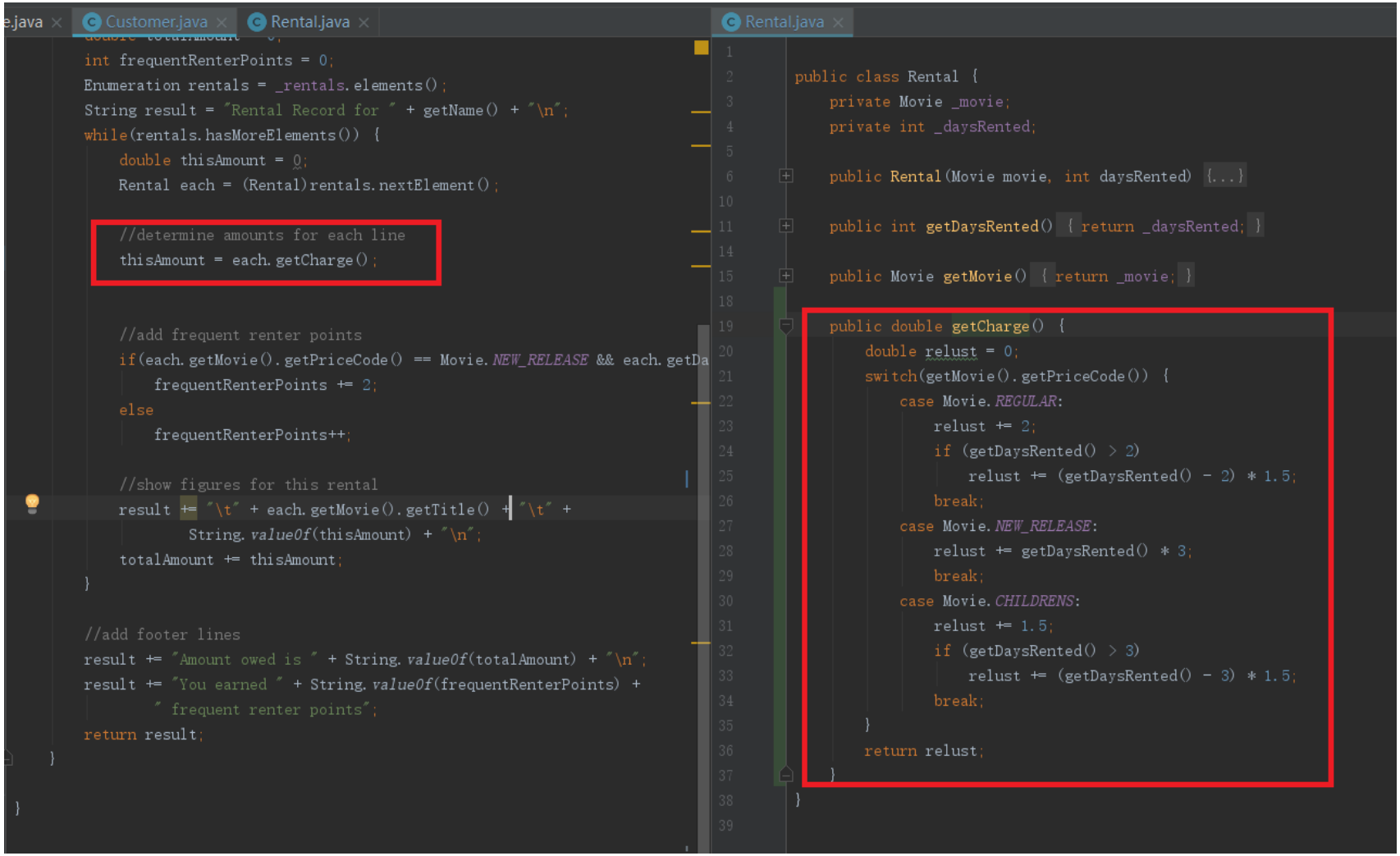


还有进一步的改进重构空间吗？

移动方法（Move Method）重构



移动方法并重命名后的结果



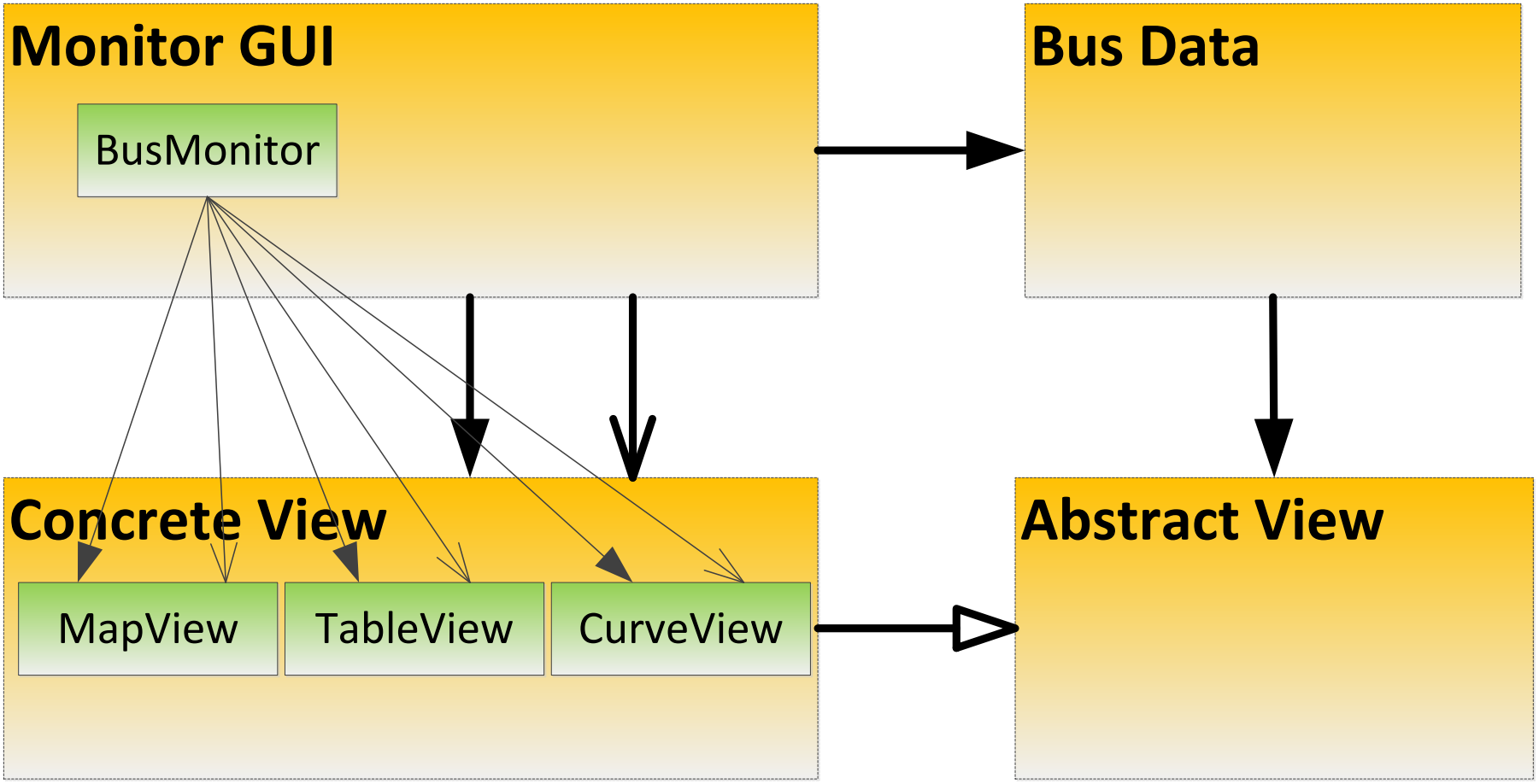
```
int frequentRenterPoints = 0;
Enumeration rentals = _rentals.elements();
String result = "Rental Record for " + getName() + "\n";
while(rentals.hasMoreElements()) {
    double thisAmount = 0;
    Rental each = (Rental)rentals.nextElement();

    //determine amounts for each line
    thisAmount = each.getCharge();


    //add frequent renter points
    if(each.getMovie().getPriceCode() == Movie.NEW_RELEASE && each.getDa
        frequentRenterPoints += 2;
    else
        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;
}

}
```

```
public class Rental {
    private Movie _movie;
    private int _daysRented;

    public Rental(Movie movie, int daysRented) {...}

    public int getDaysRented() { return _daysRented; }

    public Movie getMovie() { return _movie; }

    public double getCharge() {
        double relust = 0;
        switch(getMovie().getPriceCode()) {
            case Movie.REGULAR:
                relust += 2;
                if (getDaysRented() > 2)
                    relust += (getDaysRented() - 2) * 1.5;
                break;
            case Movie.NEW_RELEASE:
                relust += getDaysRented() * 3;
                break;
            case Movie.CHILDRENS:
                relust += 1.5;
                if (getDaysRented() > 3)
                    relust += (getDaysRented() - 3) * 1.5;
                break;
        }
        return relust;
    }
}
```

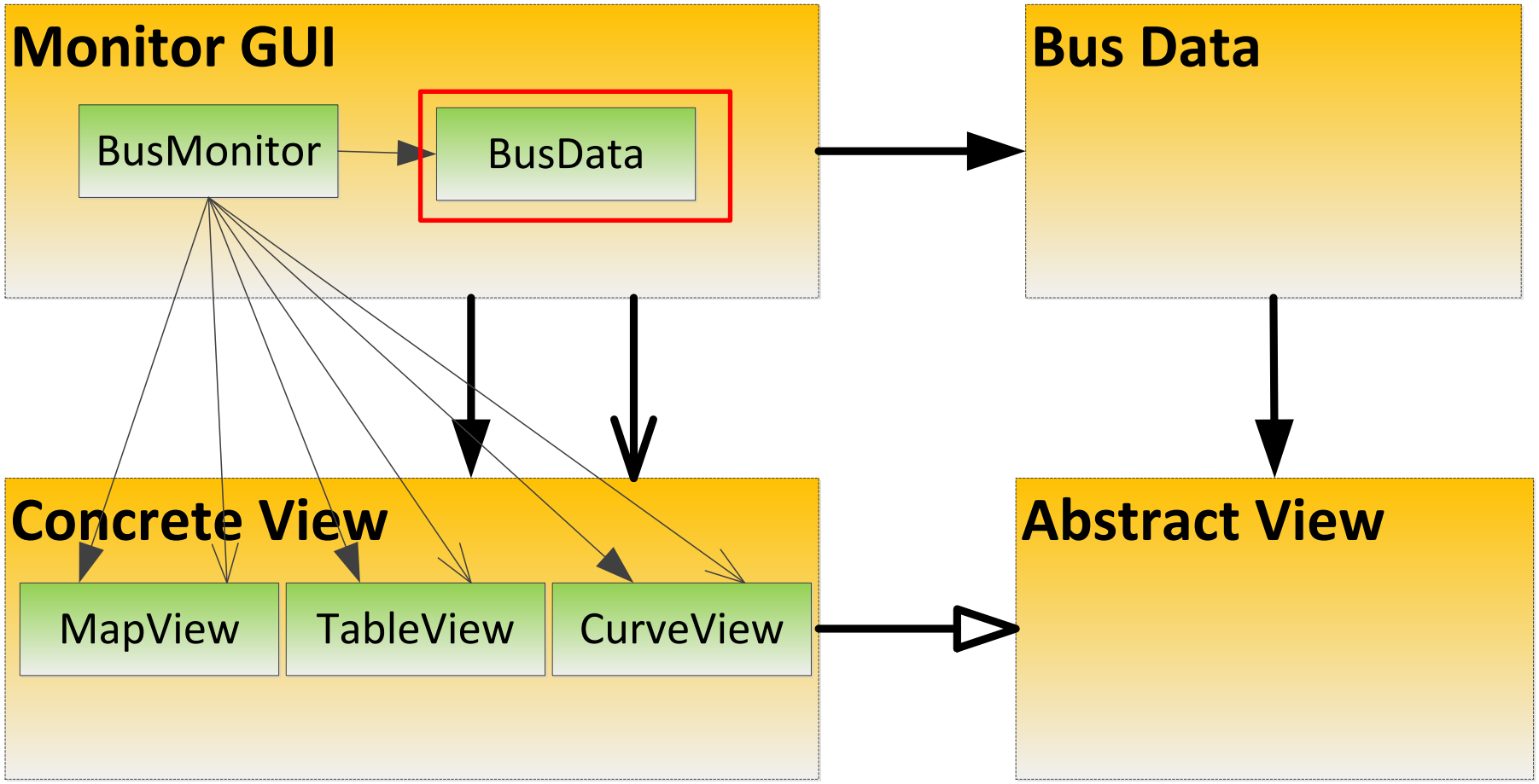
最终重构结果


Movie
priceCode: int
Rental
daysRented: int
getCharge()
Customer
statement()
1
*
*
1

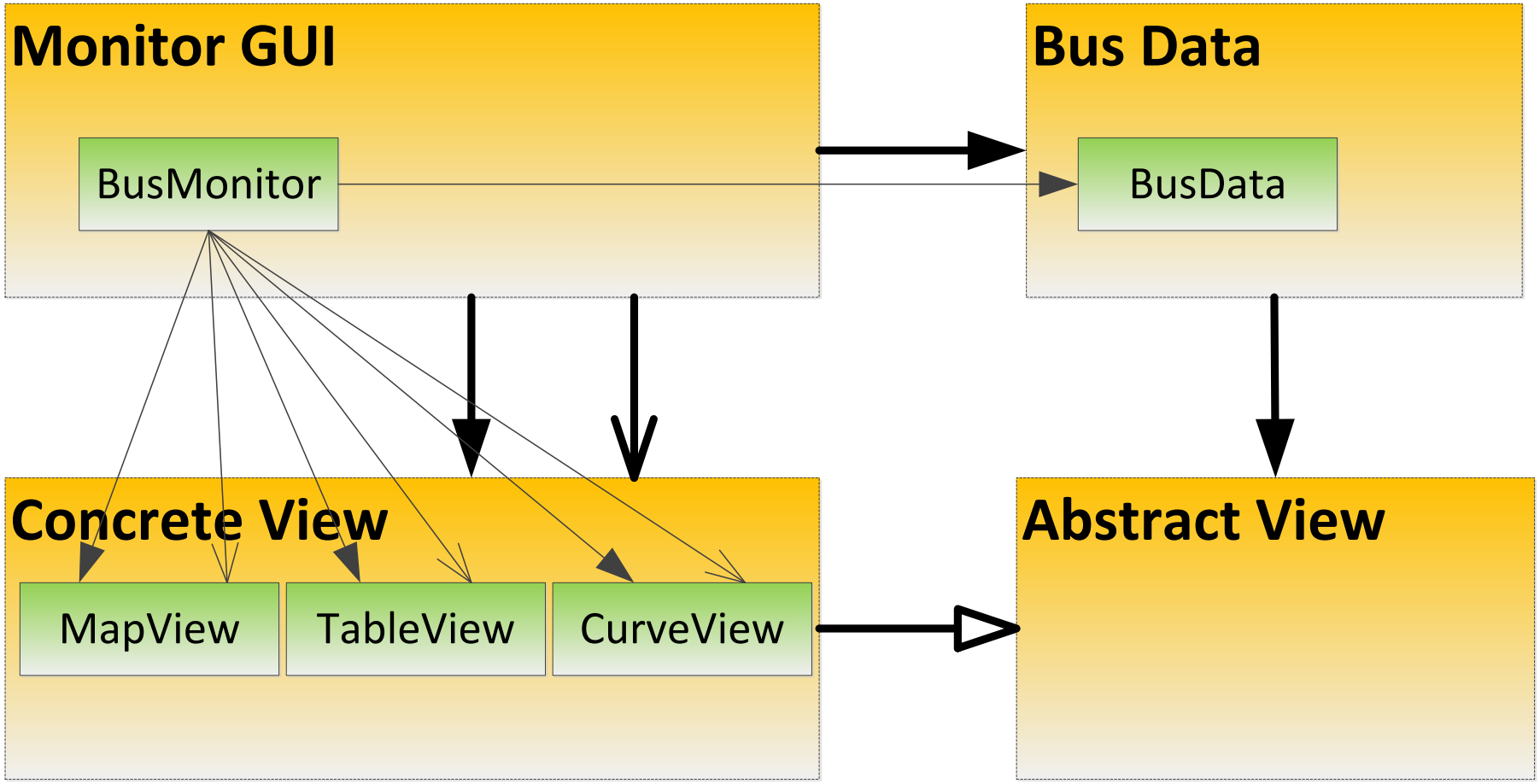
设计级复杂重构示例


Monitor GUI
BusMonitor
Bus Data
Concrete View
MapView
TableView
CurveView
Abstract View

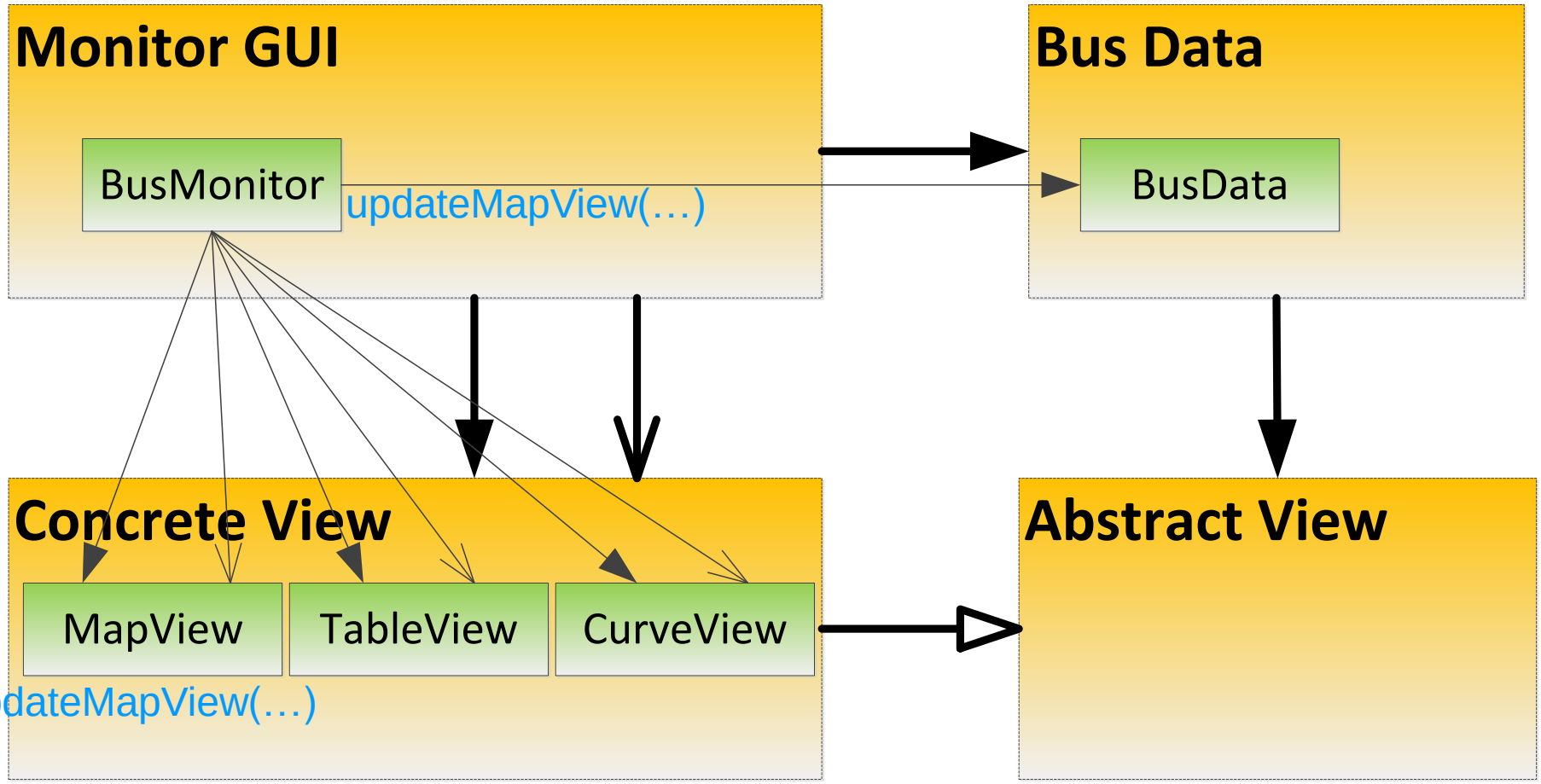
步骤1：分拆BusMonitor类



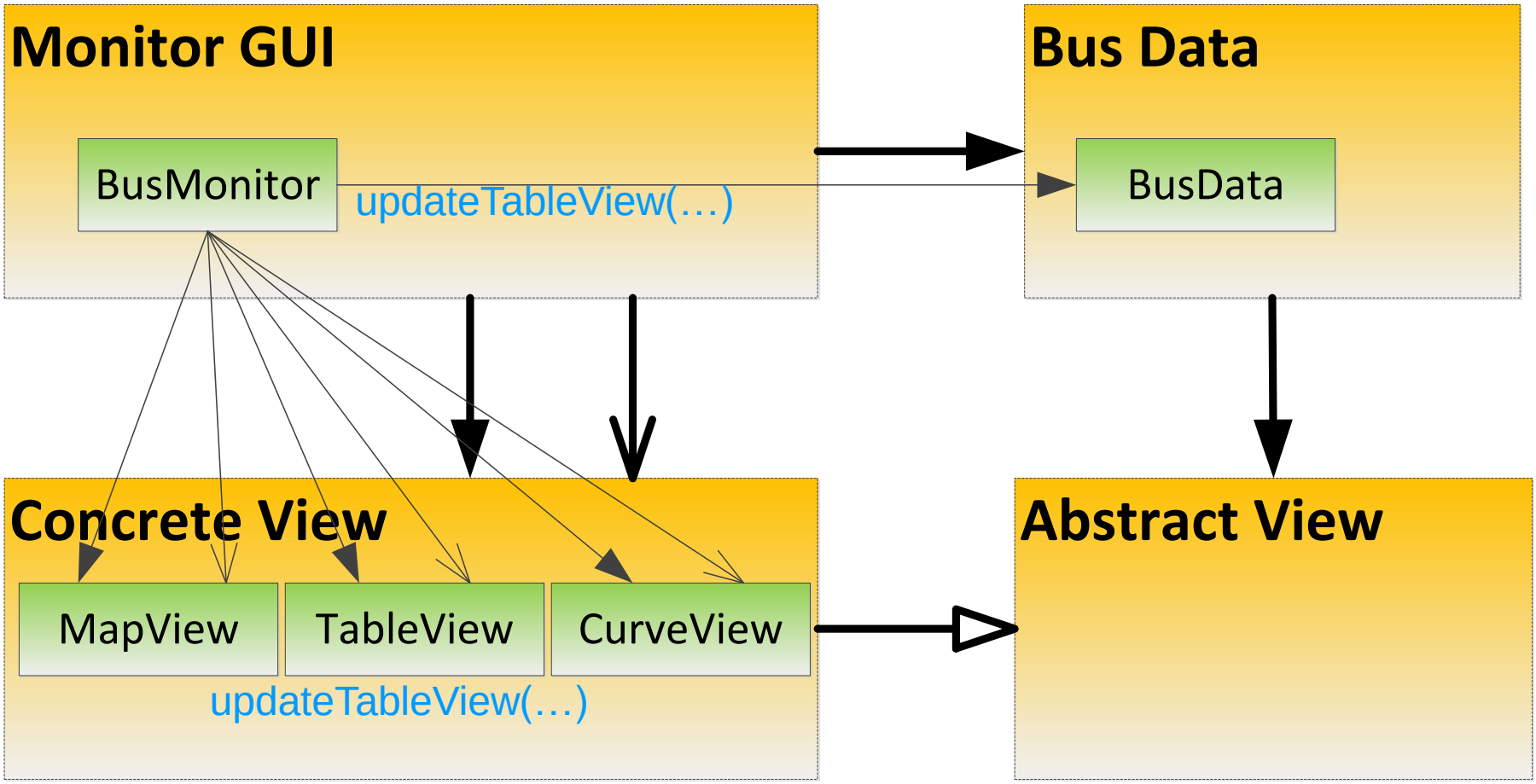
步骤1：分拆BusMonitor类



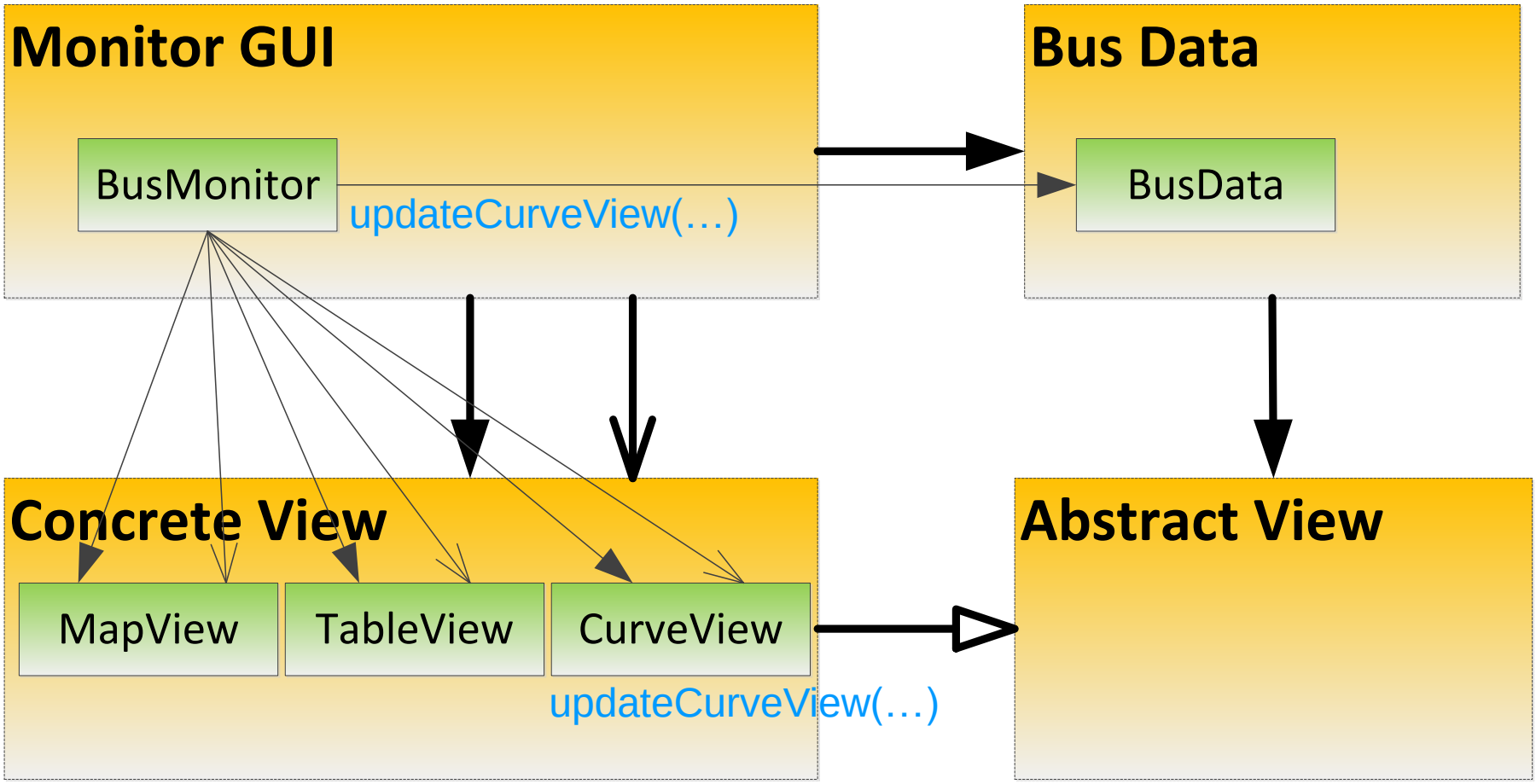
步骤2：将update*View方法移动到*View中



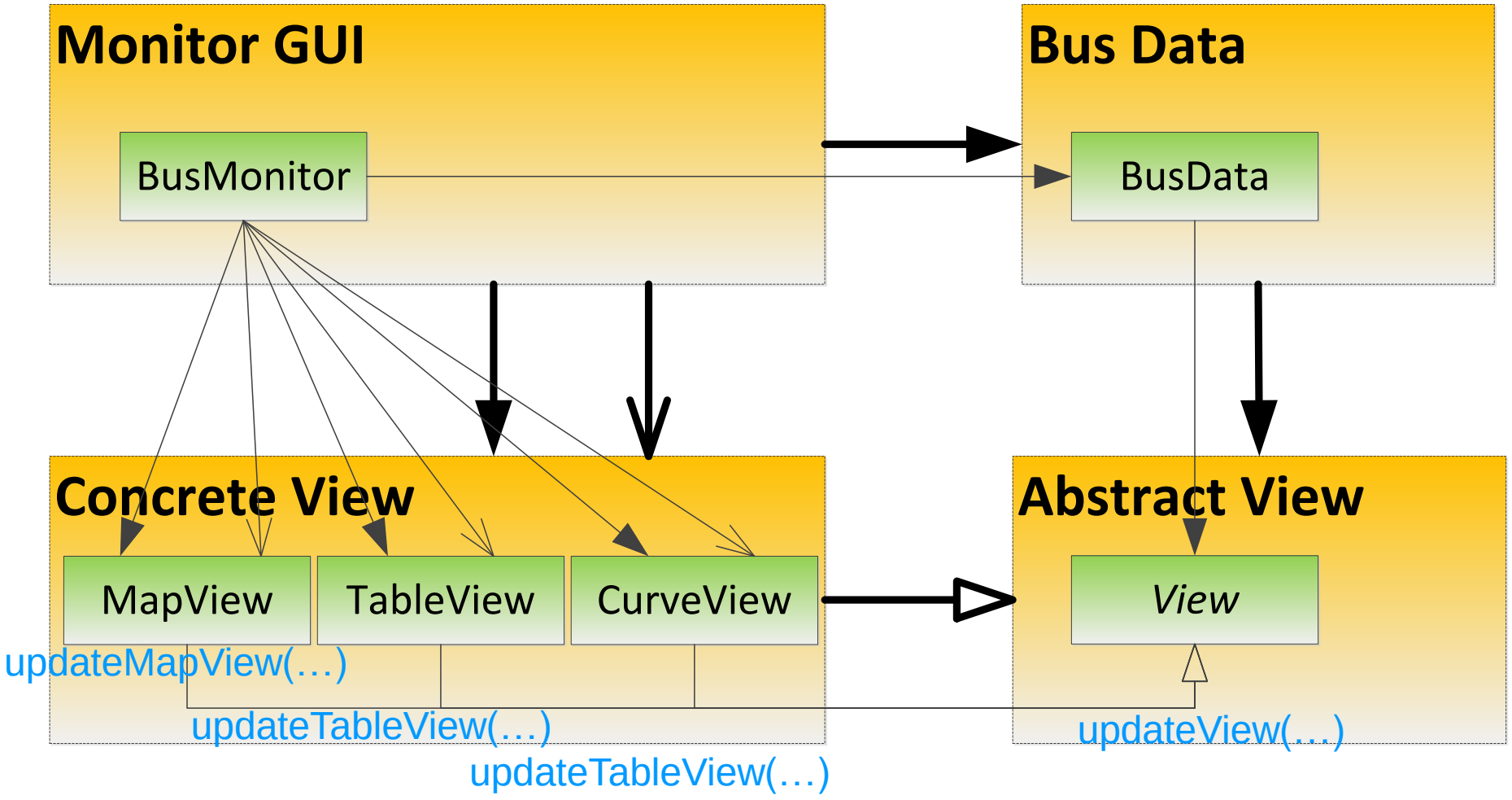
步骤2：将update*View方法移动到*View中



步骤2：将update*View方法移动到*View中



步骤3：提取抽象类View



阅读建议

- 《软件工程》9.3
- 《代码大全》24
- 《实践者研究方法》11.3

快速阅读后整理问题
在QQ群中提出并讨论

CS2001
软件工程

End

12. 软件设计
—演化式设计