B将 Temp 替换为询问

问题

将表达式的结果放在局部变量中
稍后在您的代码中使用。

```
1双计算总计（）{
2        double basePrice = 数量 * itemPrice;
3        如果（基本价格> 1000）{
4            返回基本价格 * 0.95;
5        }
6        否则{
7            返回基本价格 * 0.98;
8        }
9    }
```

解决方案

将整个表达式移动到单独的方法并返回
结果。查询方法而不是使用
多变的。将新方法合并到其他方法中，如果
必要的。

```
1双计算总计（）{
2        如果（基本价格（） > 1000）{
3            返回basePrice() * 0.95;
4        }
5        否则{
6            返回basePrice() * 0.98;
7        }
8    }
9双基价格（）{
10       退货数量 * itemPrice;
11   }
```

为什么要重构

这种重构可以为应用Extract奠定基础

方法的一部分很长。

有时也可以在其他方法中找到相同的表达式,这是考虑创建 com 的原因之一

蒙法。

好处

- 代码可读性。更容易理解的目的

  方法getTax()比行orderPrice() * 0.2 。

- 如果要替换的行是通过重复数据删除来精简代码

  用于多种方法。

很高兴知道

表现

这种重构可能会引发这样一个问题,即这种方法是否容易导致性能下降。诚实的答案是:是的,因为结果代码可能会因查询新方法而负担过重。但是对于当今快速的 CPU 和出色的编译器,负担几乎总是很小的。相比之下,由于这种重构方法,可读代码和在程序代码的其他地方重用此方法的能力是非常明显的好处。

尽管如此,如果您的临时变量用于缓存真正耗时的表达式的结果,您可能希望在将表达式提取到新表达式后停止此重构

方法。

如何重构

1. 确保在方法中为变量赋值一次且仅一次。如果不是,请使用拆分临时变量以确保该变量仅用于存储表达式的结果。

2. 使用提取方法将感兴趣的表达置于新方法中。确保此方法仅返回一个值,并且不会更改对象的状态。如果方法

影响对象的可见状态,使用分离查询
修饰符。

3. 将变量替换为对新方法的查询。

类似的重构

§ 提取方法

消除异味

§ 长法

§ 重复代码

B临时拆分多变的

问题

你有一个局部变量,用于存储各种

方法内部的中间值（循环变量除外）。

```
1双温度 = 2 * （高度 + 宽度）； 2 System.out.println(temp);
3温度 = 高度 * 宽度；

4 System.out.println(temp);
```

解决方案

对不同的值使用不同的变量。每个变量

应该只对一件特定的事情负责。

```
1最终双周长 = 2 * （高度 + 宽度）； 2 System.out.println(周长); 3最终双面
积=高*宽；

4 System.out.println(区域);
```

为什么要重构

如果您在函数内的变量数量上吝啬
并将它们重用于各种不相关的目的,你确定
在需要更改时立即遇到问题
到包含变量的代码。你将不得不纠正
每种情况下的变量使用以确保正确的值
被使用。

好处

- 程序代码的每个组件都应负责
  只为一件事。这使得维护代码变得更加容易,因为您可以轻松地替换任何特定的东西

  不用担心意外的影响。

- 代码变得更具可读性。如果变量创建时间很长
  之前匆忙中,它可能有一个无法解释任何事情的名称： k等。但是您
  可以通过以下方式解决这种情况
  以易于理解、不言自明的方式命名新变量。此类名称可能类似于customerTaxValue
  ，
  cityUnemploymentRate 、 clientSalutationString等。

- 如果您预计使用
  稍后提取方法。

如何重构

1. 找到代码中变量被赋值的第一个位置。在这里,您应该使用与分配的值相对应的名称来重命名变量。

2. 在出现这种情况的地方使用新名称而不是旧名称使用变量的值。

3. 根据需要对变量赋值的地方重复不同的价值。

反重构

§ 内联温度

类似的重构

§ 提取变量

§ 删除分配给参数

帮助其他重构

§ 提取方法

B删除分配给参数

问题

一些值被分配给方法体内的参数。

```
1 int折扣（int inputVal, int数量）{
2         如果（输入值> 50）{
3             输入值-= 2;
4         }
5         // ...
6 }
```

解决方案

使用局部变量而不是参数。

```
1 int折扣（int inputVal, int数量）{
2         整数结果 = 输入值;
3         如果（输入值> 50）{
4             结果-= 2;
```

```
5       }
6       // ...
7 }
```

为什么要重构

这种重构的原因与拆分临时变量的原因相同,但在这种情况下,我们处理的是参数,而不是局部变量。

首先,如果通过引用传递参数,则在方法内部更改参数值后,将该值传递给请求调用该方法的参数。

很多时候,这会意外发生并导致不幸的后果。即使在您的编程语言中参数通常通过值（而不是通过引用)传递,这种编码怪癖可能会疏远那些不习惯它的人。

其次,对单个参数多次赋值不同的值,让你很难知道在任何特定时间点参数中应该包含哪些数据。

如果记录了您的参数及其内容,但实际值可能与方法内部的预期值不同,则问题会变得更糟。

好处

· 程序的每个元素应该只负责一件事。这使得代码维护变得更加容易,因为您可以安全地替换代码而无需任何方面

效果。

· 这种重构有助于提取重复代码以分离
方法。

如何重构

1.创建一个局部变量并分配你的初始值
范围。

2. 在此行之后的所有方法代码中,替换参数
与您的新局部变量一起使用。

类似的重构

§ 拆分临时变量

帮助其他重构

§ 提取方法

B替换方法与方法对象

问题

你有一个很长的方法,其中局部变量是如此交织在一起,您不能应用提取方法。

```
1类订单{
2        // ...
3        公共双倍价格 () {
4            双primaryBasePrice;
5            双二级基础价格;
6            双tertiaryBasePrice;
7            // 执行长计算。
8日       }
9    }
```

解决方案

将方法转换为单独的类,以便本地变量成为类的字段。然后你可以拆分方法到同一个类中的多个方法。

```
class Order {
    // ...
    public double price() {
        return new PriceCalculator(this).compute();
    }
}

class PriceCalculator {
    private double primaryBasePrice;
    private double secondaryBasePrice;
    private double tertiaryBasePrice;

    public PriceCalculator(Order order) {
        // Copy relevant information
        // from the order object.
    }

    public double compute() {
        // Perform long computation.
    }
}
```

为什么要重构

一个方法太长了，你不能把它分开，因为有大量难以分离的局部变量

彼此。

第一步是将整个方法隔离成一个单独的
类并将其局部变量转换为类的字段。

首先,这允许在类级别隔离问题。
其次,它为将大而笨重的方法拆分成更小的方法铺平了道路,这些小方法无论如何都不符合原始类的目的。

好处

在自己的类中隔离一个长方法可以阻止方法的大小膨胀。这也允许将其拆分为类中的子方法,而不会使用实用方法污染原始类。

缺点

添加了另一个类,增加了程序的整体复杂性。

如何重构

1. 创建一个新班级。根据用途命名
   您正在重构的方法。

2. 在新类中,创建一个私有字段,用于存储对该方法先前所在类的实例的引用。如果需要,它可用于从原始类中获取一些所需的数据。

3. 为每个局部变量创建一个单独的私有字段
   方法。

4.创建一个接受参数值的构造函数

该方法的所有局部变量,并初始化corre

响应私有字段。

5. 声明主方法并将原方法的代码复制到其中,将局部变量替换为私有字段。

6、通过创建方法对象并调用其main方法来替换原类中原方法的主体。

类似的重构

§ 用对象替换数据值

对字段做同样的事情。

消除异味

§ 长法

B替代品算法

问题

所以你想用新的算法替换现有的算法？

```
1 个字符串foundPerson(String[] people){
2       for (int i = 0; i < people.length; i++) {
3           if (people[i].equals(   Don   )){
4               返回“唐”；
5           }
6           if (people[i].equals(   John   )){
7               返回“约翰”；
8           }
9           if (people[i].equals(   Kent   )){
10              返回“肯特”；
11          }
12      }
13      返回“”；
14 }
```

解决方案

替换实现方法的主体
算法与新算法。

```
1 个字符串foundPerson(String[] people){
2       列出候选人 =
3           Arrays.asList(new String[] {  Don  ,  John  ,  Kent  });
4       for (int i=0; i < people.length; i++) {
5           if (candidates.contains(people[i])) {
6               返回人[i];
7           }
8       }
9       返回 “” ;
10 }
```

为什么要重构

1. 逐步重构并不是改进程序的唯一方法。有时，一种方法充满了问题，以至于

更容易拆除该方法并重新开始。有可能
你找到了一个更简单、更丰富的算法
高效的。如果是这种情况，您应该简单地更换旧的
新算法。

2.随着时间的推移，你的算法可能会被纳入

一个著名的库或框架，你想摆脱
您的独立实施，为了简化
维护。

3. 你的程序的要求可能会发生很大的变化，以至于

您现有的算法无法为该任务挽救。

如何重构

1.确保您已将现有算法简化为
尽可能。将不重要的代码移至其他方法
使用提取方法。算法中的移动部件越少,更换起来就越容易。

2. 以新方法创建新算法。更换旧的
使用新算法并开始测试程序。

3.如果结果不匹配,返回旧实现
并比较结果。确定差异的原因。虽然原因通常是旧算法中的错误,但它是
更有可能是由于某些东西在新版本中不起作用。

4.当所有测试成功完成后,删除旧算法就好了!

消除异味

§ 重复代码

§ 长法

在之间移动特征对象

即使您以不太完美的方式在不同的类之间分配功能,仍然有希望。

这些重构技术展示了如何在类之间安全地移动功能、创建新类以及对公共访问隐藏实现细节。

§ 移动方法

问题:一个方法在另一个类中的使用比在它的类中使用的多自己的班级。

解决方案:在类中创建一个新方法,使用方法最多,然后将代码从旧方法移到那里。将原始方法的代码转换为对另一个类中新方法的引用,否则将其完全删除。

§ 移动领域

问题:一个字段在另一个类中的使用比在它的类中更多自己的班级。

解决方案:在一个新类中创建一个字段并重定向所有用户旧田地吧。

§ 提取类

问题:当一个班级做两个班级的工作时,尴尬
结果。

解决方案:相反,创建一个新类并将负责相关功能的字段和方法放入其中。

§ 在线课程

问题:一个类几乎什么都不做,也不对任何事情负责,也没有为它计划额外的职责。

解决方案:将所有特征从类移到另一个。

§ 隐藏代表

问题:客户端从对象A的字段或方法中获取对象B。然后客户端调用对象 B 的一个方法。

解决方案:在 A 类中创建一个新方法,将调用委托给对象 B。现在客户端不知道或不依赖于 B 类。

§ 移除中间人

问题:一个类有太多简单地委托给其他对象的方法。

解决方法:删除这些方法,强制客户端调用
直接结束方法。

§ 引进国外方法

问题:实用程序类不包含您需要的方法,并且您无法将方法添加到类中。

解决方案:将方法添加到客户端类,并将实用程序类的对象作为参数传递给它。

§ 引入本地扩展

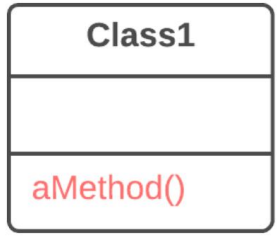
问题:实用程序类不包含您需要的一些方法。但是您不能将这些方法添加到类中。

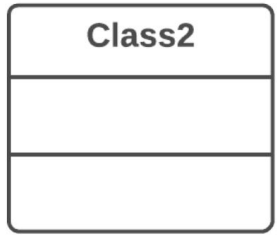
解决方案:创建一个包含方法的新类,并使其成为实用程序类的子类或包装器。

B移动方式

问题

一个方法在另一个类中的使用比在它自己的类中更多。

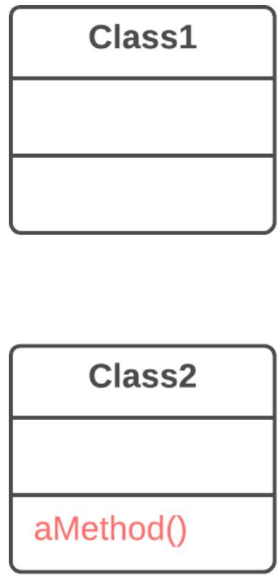




解决方案

在使用该方法的类中创建一个新方法

大多数,然后将代码从旧方法移到那里。将原始方法的代码转换为对另一个类中新方法的引用,否则将其完全删除。



为什么要重构

1.你想将一个方法移动到一个包含大部分内容的类中

该方法使用的数据。这使得类在内部更加连贯。

2.你想移动一个方法以减少或消除

调用该方法的类对其所在类的依赖关系。如果调用类已经依赖于您计划将方法移动到的类，这将很有用。这减少了类之间的依赖。

如何重构

1. 验证旧方法在其类中使用的所有特性。移动它们也可能是个好主意。通常，如果使用某个功能

只有通过正在考虑的方法,您当然应该将功能移至它。如果该功能也被其他方法使用,您也应该移动这些方法。有时移动大量方法比在不同类中建立它们之间的关系要容易得多。

确保该方法未在超类和子类中声明。如果是这种情况,您将不得不避免移动,或者在接收者类中实现一种多态性,以确保在捐助者类之间拆分方法的不同功能。

2. 在接收者类中声明新方法。您可能想为更适合的方法指定一个新名称

它在新班级。

3. 决定你将如何引用接收者类。您可能已经有一个返回适当对象的字段或方法,但如果没有,您将需要编写一个新方法或字段来存储接收者类的对象。

现在您有一种方法可以引用接收者对象和它的类中的一个新方法。有了这一切,你可以转动
旧方法变成对新方法的引用。

4. 看看:能不能把旧方法彻底删除?如果是这样,请在所有使用新方法的地方引用新方法

旧的

类似的重构

§ 提取方法

§ 移动领域

帮助其他重构

§ 提取类

§ 线下班

§ 引入参数对象

消除异味

§ 霰弹枪手术

§ 功能羡慕

§ 切换语句

§ 并行继承层次结构

§ 消息链

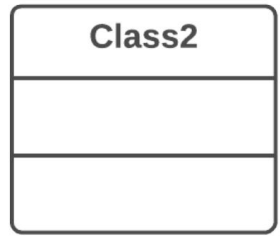
§ 不恰当的亲密关系

§ 数据类

B移动场

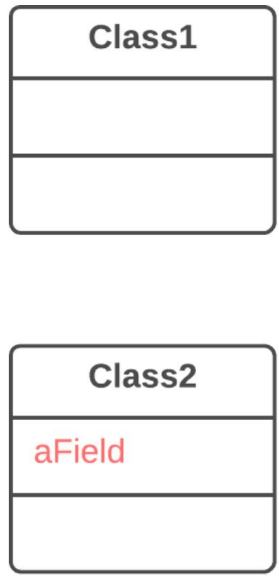
问题

一个字段在另一个类中的使用比在它自己的类中更多。



解决方案

在新类中创建一个字段并重定向旧类的所有用户
领域。



为什么要重构

通常,字段作为提取类技术的一部分进行移动。
决定离开该领域的班级可能很困难。这是我们的经验法则:将字段放在与使用它的方法相同的位置（或者大多数这些方法所在的位置）。

当字段只是位于错误的位置时,此规则将在其他情况下有所帮助。

如何重构

1. 如果字段是公共的,如果将字段设为私有并提供公共访问方法（为此，可以使用封装字段）,重构会容易得多。

2.在配方中创建与访问方法相同的字段
耳鼻喉科。

3. 决定你将如何引用接收者类。您可能已经有一个返回适当对象的字段或方法;如果没有，
您将需要编写一个新方法或字段来存储接收者类的对象。

4. 将所有对旧字段的引用替换为对接收者类中方法的适当调用。如果该字段不是私有的,请
在超类和子类中处理此问题。

5.删除原类中的字段。

类似的重构

§ 移动领域

帮助其他重构

§ 提取类

§ 线下班

消除异味

§ 霰弹枪手术

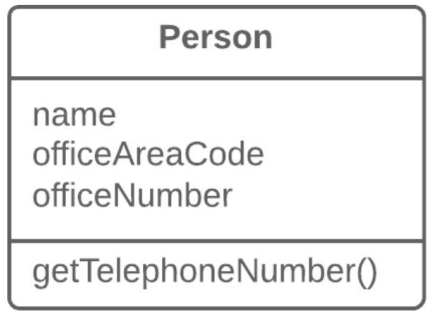
§ 并行继承层次结构

§ 不恰当的亲密关系

B提取类

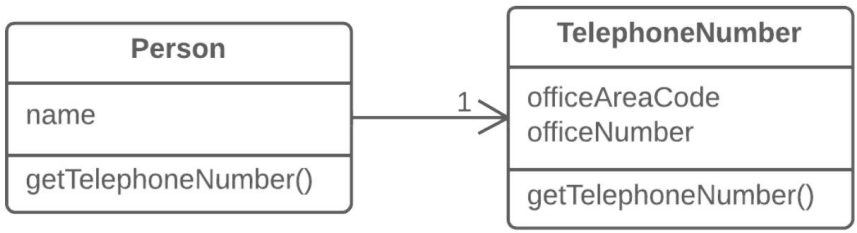
问题

当一个班级完成两个班级的工作时,就会产生尴尬。



解决方案

相反,创建一个新类并将负责相关功能的字段和方法放入其中。



为什么要重构

课程一开始总是清晰易懂。他们做自己的工作,照常处理自己的事情,与其他班级的工作打交道。但是随着程序的扩展,添加了一个方法,然后添加了一个字段……最终,一些类比以往任何时候都执行更多的职责

设想。

好处

- 这种重构方法将有助于保持对单一职责原则的遵守。您的课程代码将是

  更加明显和易于理解。

- 单一职责的班级更可靠,更能容忍变化。例如,假设您有一个班级负责十种不同的事情。当您更改此课程以使其对一件事更好时,您可能会冒着为其他九件事破坏它的风险。

缺点

如果您使用这种重构技术“过度使用”,您将拥有
诉诸内联课程。

如何重构

在开始之前,请确定您希望如何准确地划分班级的职责。

1. 创建一个包含相关功能的新类。

2. 在旧类和新类之间建立关系。

   理想情况下,这种关系是单向的;这允许重用第二类而没有任何问题。尽管如此,如果您认为双向关系是必要的,则始终可以建立这种关系。

3. 对每个字段和方法使用 Move Field和 Move Method

   你已经决定搬到新班级。对于方法,从私有的开始,以减少犯大量错误的风险。尝试一次移动一点点,并在每次移动后测试结果,以避免在最后出现错误修复的堆积。

   完成移动后,再看一下生成的类。可以重命名具有更改职责的旧类以提高清晰度。再次检查是否可以摆脱双向类关系（如果存在）。

4. 还要考虑从外部对新类的可访问性。您可以通过将其设为私有,通过旧类中的字段对其进行管理,从而完全对客户端隐藏该类。

   或者,您可以通过允许客户端直接更改值来将其设为公开。您在这里的决定取决于当对新类中的值进行意外的直接更改时,旧类的行为有多安全。

反重构

§ 线下班

类似的重构

§ 提取子类

§ 用对象替换数据值

消除异味

§ 重复代码

§ 大班

§ 发散变化

§ 数据块

§ 原始痴迷

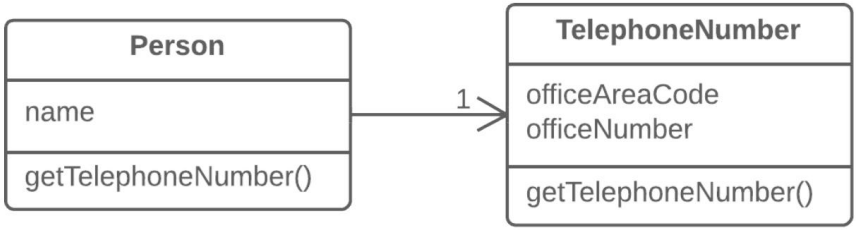
§ 临时场地

§ 不恰当的亲密关系

B内联类

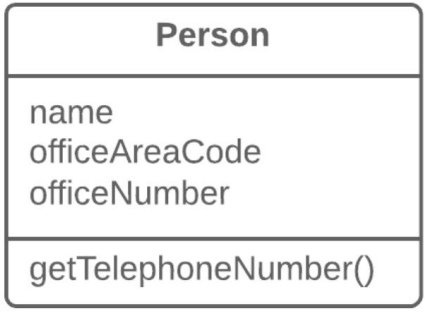
问题

一个类几乎什么都不做,也不对任何事情负责,也没有为它计划额外的责任。



解决方案

将所有功能从类移到另一个。



为什么要重构

在一个类的特征被“移植”到其他类之后,通常需要这种技术,使该类保持点亮状态

无事可做。

好处

消除不必要的类可以释放计算机上的操作内存 以及你头脑中的带宽。

如何重构

1. 在接受者类中,创建存在于捐赠者类中的公共字段和方法。方法应参考equiv

   施主类的天赋方法。

2. 将所有对供体类的引用替换为对
   接收者类的字段和方法。

3. 现在测试程序并确保没有添加任何错误。如果测试显示一切正常,请开始使用 Move
   Method和 Move Field 将所有功能从原始类完全移植到接收者类。

   继续这样做,直到原始类完全为空。

4.删除原来的类。

反重构

§ 提取类

消除异味

§ 霰弹枪手术

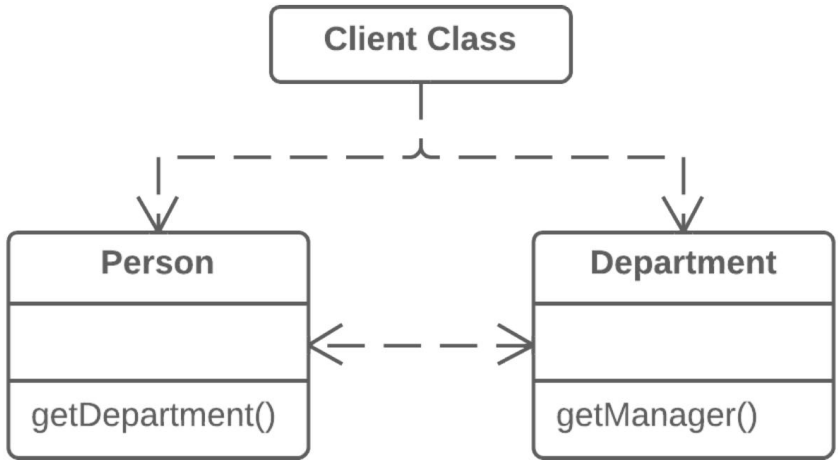
§ 懒人课

§ 推测的普遍性

B隐藏委托

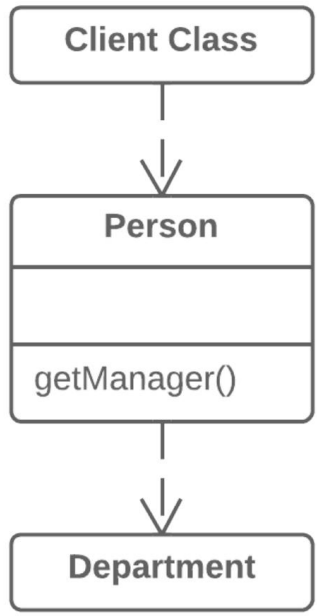
问题

客户端从对象 A 的字段或方法中获取对象 B。
然后客户端调用对象 B 的一个方法。



解决方案

在 A 类中创建一个新方法，将调用委托给对象 B。现在客户端不知道或不依赖于 B 类。



为什么要重构

首先,让我们看一下术语:

·服务器是客户端可以直接访问的对象。

·委托是包含客户端所需功能的最终对象。

当客户端请求一个对象时,会出现一个调用链
另一个对象,然后第二个对象请求另一个对象,
等等。这些调用序列涉及客户端在类结构中的导航。这些相互关系的
任何变化都需要客户端进行更改。

好处

对客户隐藏委托。客户端代码需要了解的对象之间关系的细节越少，就越容易对程序进行更改。

缺点

如果您需要创建过多的委托方法，服务器级可能会成为不必要的中间人，从而导致过多的中间人。

如何重构

1. 对于客户端调用的委托类的每个方法，在服务器类中创建一个方法，将调用委托给委托类。

2. 更改客户端代码，使其调用serv的方法
   他上课。

3. 如果您的更改使客户端不再需要委托类，您可以从服务器类中删除对委托类的访问方法（最初用于获取委托类的方法）。

反重构

§ 移除中间人

消除异味

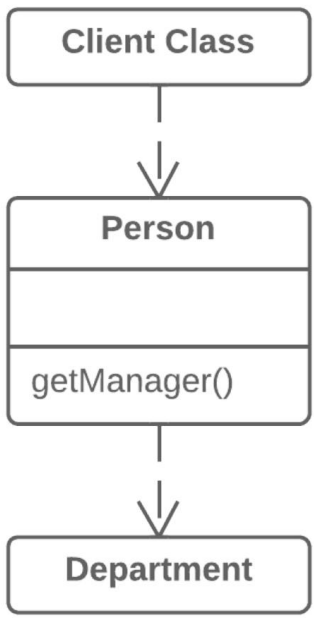
§ 消息链

§ 不恰当的亲密关系

B移除中间人

问题

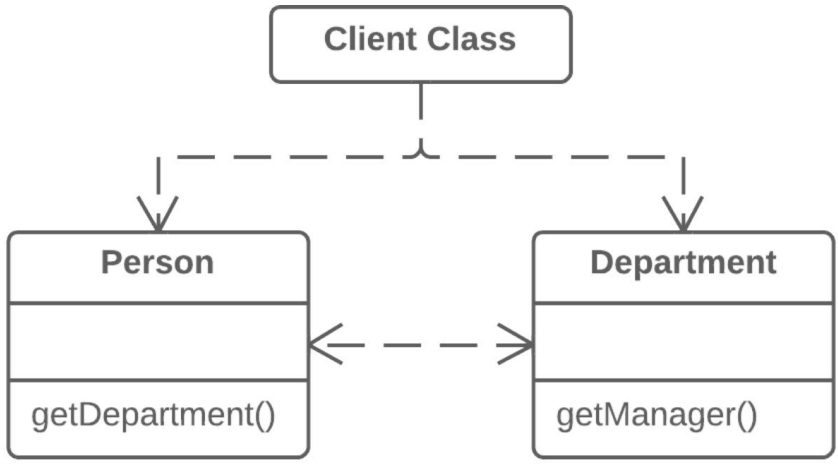
一个类有太多简单地委托给其他对象的方法。



解决方案

删除这些方法,强制客户端调用结束

方法直接。



为什么要重构

为了描述这种技术,我们将使用 Hide Delegate 中的术语,它们是:_____

·服务器是客户端可以直接访问的对象。

·委托是包含客户端所需功能的最终对象。

有两种类型的问题:

1.服务器类本身不做任何事情,只是创建
不必要的复杂性。在这种情况下,考虑这是否
根本需要上课。

2. 每次向委托添加新特性时,都需要在服务器类中为其创建委托方法。如果进行大量更改,
这将是相当令人厌烦的。

如何重构

1. 创建一个getter 用于从
服务器类对象。

2.用直接调用委托类中的方法替换对服务器类中委托方法的调用。

反重构

§ 隐藏代表

消除异味

§ 中间人

B引进外国方法

问题

实用程序类不包含您需要的方法,并且
您不能将该方法添加到类中。

```
课报告{
    // ...
    无效发送报告 () {
        日期nextDay = new Date(previousEnd.getYear(),
            previousEnd.getMonth(), previousEnd.getDate() + 1);
        // ...
    }
}
```

解决方案

将方法添加到客户端类并传递
实用程序类将其作为参数。

```
课报告{
    // ...
```

```
    无效发送报告（）{
        日期newStart = nextDay(previousEnd);
        // ...
    }
    私人静态日期nextDay（日期参数）{
        返回新日期（arg.getYear（），arg.getMonth（），arg.getDate（）+1）
    }
}
```

为什么要重构

您有使用特定数据和方法的代码
班级。你意识到代码看起来和工作起来都很好
在类中的一个新方法中。但是你不能添加
类的方法，因为例如，该类位于
在第三方库中。

当你想要的代码时，这种重构有很大的回报
移动到该方法在不同的地方重复几次
在你的程序中的地方。

由于您将实用程序类的对象传递给新方法的参数，因此您可以访问其所有字段。

在该方法中，您几乎可以做所有您想做的事情
想要，好像该方法是实用程序类的一部分。

好处

删除代码重复。如果您的代码在多个地方重复,您可以将这些代码片段替换为方法调用。
即使考虑到外部方法位于次优位置,这也比重复要好。

缺点

在您之后维护代码的人并不总是清楚在客户端类中使用实用程序类的方法的原因。如果
该方法可以在其他类中使用,您可以通过为实用程序类创建一个包装器并放置

那里的方法。当有 sev 时,这也是有益的
Eral 搜索实用方法。引入本地扩展可以提供帮助
有了这个。

如何重构

1. 在客户端类中新建一个方法。

2. 在此方法中,创建一个参数,实用程序类的对象将传递给该参数。如果此对象可以从客户端
   类中获取,则不必创建这样的参数。

3. 将相关代码片段提取到该方法中并
   用方法调用替换它们。

4. 确保在方法的注释中留下Foreign 方法标记,以及将此方法放置在实用程序类中的建议（如果以后可能的话）。对于那些将在

未来。

类似的重构

§ 引入本地扩展

> 将所有扩展方法移动到一个单独的类,该类是包装器或某个服务类的子类。

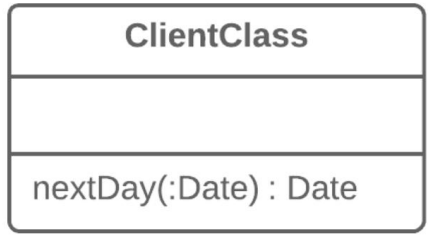
消除异味

§ 不完整的图书馆类

B引入本地延期

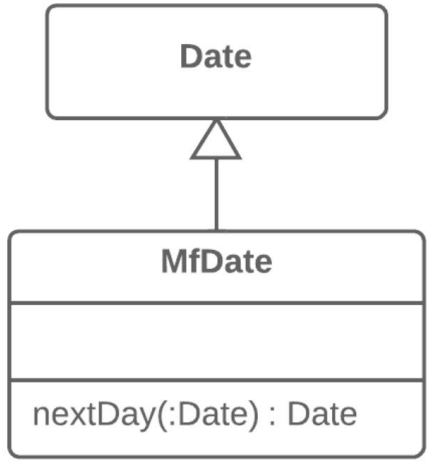
问题

实用程序类不包含您需要的某些方法。
但是您不能将这些方法添加到类中。



解决方案

创建一个包含方法的新类,并使其成为实用程序类的子类或包装器。



为什么要重构

您正在使用的类没有您需要的方法。更糟糕的是,您无法添加这些方法（例如,因为这些类在第三方库中）。那里有两个

出路：

- 从相关类创建一个子类,包含方法并从父类继承其他所有内容。这种方式更容易,但有时会被实用程序类本身阻止（由于final）。

- 创建一个包含所有新方法的包装类,其他地方将委托给实用程序类的相关对象。这种方法工作量更大,因为您不仅需要代码来维护包装器和实用程序对象之间的关系,而且还需要大量简单的委托方法来模拟实用程序类的公共接口。

好处

通过将其他方法移动到单独的扩展类
（包装器或子类），您避免使用不适合的代码混淆客户端类。程序组件更多

连贯且更可重用。

如何重构

1.新建一个扩展类：

- 选项 A：使其成为实用程序类的子项。

- 选项 B：如果您决定制作包装器，请创建一个
  其中的字段用于存储将进行委托的实用程序类对象。使用此选项时，您需要

  还创建重复的公共方法的方法
  实用程序类并包含对方法的简单委托
  实用程序对象。

2.创建一个使用构造函数参数的构造函数
实用程序类的门。

3. 还创建一个替代的“转换”构造函数，它采用
只有原始类的对象在其参数中。这会
帮助用扩展替换原点的对象
最后一班。

4. 在类中创建新的扩展方法。移居国外
从其他类到这个类的方法,否则删除 for
如果它们的功能已经存在于
延期。

5.用新的扩展类替换实用程序类的使用
需要其功能的地方。

类似的重构

§ 引进国外方法

> 如果您只需要一种特殊方法,该方法在服务类中不存在,并且您无法扩展它,请将其移至客户端类并将服务类的对象作为参数传递。

消除异味

§ 不完整的图书馆类

组织数据

这些重构技术有助于数据处理,用丰富的类功能替换原语。

另一个重要的结果是类关联的解开,这使得类更便携和可重用。

§ 自封装字段

问题:您使用对类中私有字段的直接访问。

解决方案:为该字段创建一个 getter 和 setter,并仅使用它们来访问该字段。

§ 用对象替换数据值

问题:一个类(或一组类)包含一个数据字段。该字段有自己的行为和相关数据。

解决方法:新建一个类,将旧的字段及其行为放在类中,将类的对象存放在原来的类中。

§ 将值更改为参考

问题:因此,您需要用单个对象替换单个类的许多相同实例。

解决方案:将相同的对象转换为单个参考对象。