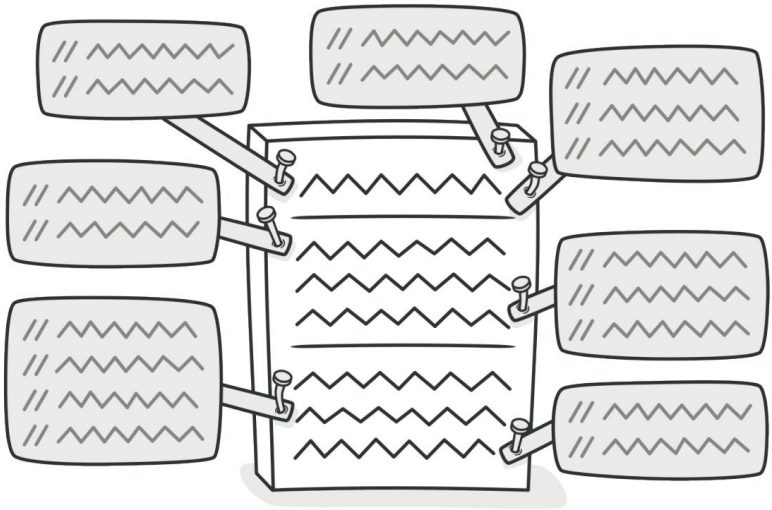
Machine Translated by Google

评论_

体征和症状

一个方法充满了解释性注释。



问题的原因

当作者意识到他或她的代码不直观或不直观时,通常会出于好意创建注释
明显的。在这种情况下,注释就像一个除臭剂面具,带有可以改进的
可疑代码的气味。

> 最好的注释是一个方法或类的好名字。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

如果您觉得没有注释无法理解代码片段,请尝试以以下方式更改代码结构
发表评论。

治疗

- 如果注释旨在解释复杂的表达式,则
表达式应拆分为可理解的子表达式
使用提取变量。

- 如果注释解释了一段代码,这部分可以是
通过提取方法变成了一个单独的方法。名字
可以从评论文本本身中获取新方法,
最有可能的。

- 如果一个方法已经被提取,但是注释仍然存在
有必要解释该方法的作用,给出该方法
一个不言自明的名称。为此使用重命名方法。

- 如果您需要断言有关状态所需的规则
系统工作,使用 Introduce Assertion。

清偿

代码变得更加直观和明显。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

何时忽略

评论有时很有用：

- 在解释为什么要在一个
  具体的办法。

- 当解释复杂的算法时（当所有其他用于简化算法的方法都被尝试过但都失败时）。

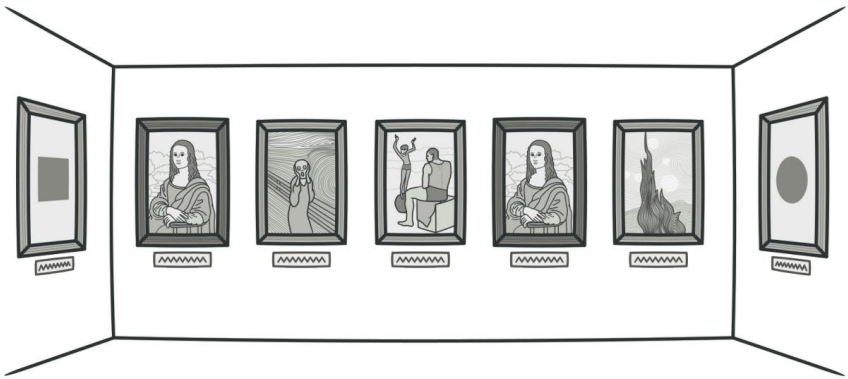
lukas.haigner@gmail.com (#15232)

Machine Translated by Google

重复的代码

体征和症状

两个代码片段看起来几乎相同。



问题的原因

当多个程序员同时处理同一程序的不同部分时,通常会发生重复。由于他们正在处理不同的任务,他们可能不知道他们的同事已经编写了类似的代码,这些代码可以重新用于他们自己的需要。

lukas.haigner@gmail.com (#15232)

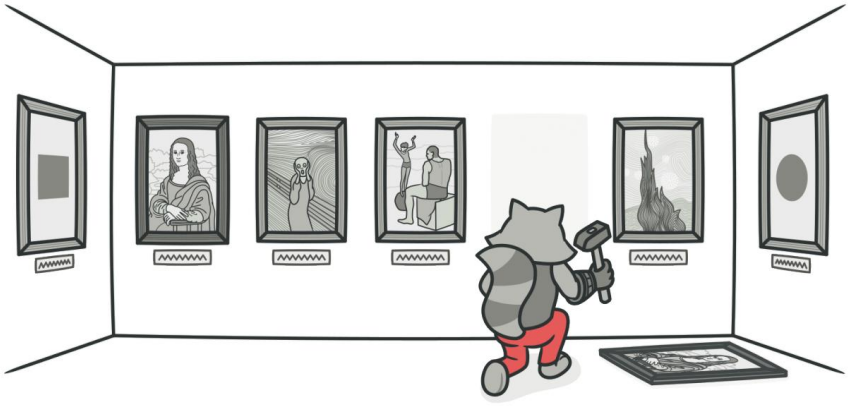
Machine Translated by Google

当代码的特定部分看起来不同但实际上执行相同的工作时,还有更微妙的重复。这种重复很难找到和修复。

有时重复是有目的的。当急于赶上最后期限并且现有代码“几乎适合”这项工作时,新手程序员可能无法抗拒复制和粘贴相关代码的诱惑。在某些情况下,程序员只是懒得去整理。

治疗

- 如果在相同的两个或多个方法中发现相同的代码
  类:使用提取方法并在两个地方调用新方法。



- 如果在同一级别的两个子类中发现相同的代码:

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

- 对两个类都使用提取方法,然后是上拉
  您正在使用的方法中使用的字段的字段
  拉起

- 如果重复代码在构造函数内,请使用上拉构造函数主体。

- 如果重复代码相似但不完全相同,
  使用表单模板方法。

- 如果两种方法做同样的事情但使用不同的算法,选择最佳算法并
  应用替代算法
  数学

- 如果在两个不同的类中发现重复代码:

  - 如果类不是层次结构的一部分,请使用提取超类为这些类创建单
    个超类
    它保留了所有以前的功能。

  - 如果创建超类很困难或不可能,请使用
    将 Class 提取到一个类中并在其中使用新组件
    另一个。

- 如果存在大量条件表达式并且
  执行相同的代码(仅在它们的条件上有所不同),
  使用 Consoli 日期条件表达式将这些运算符合并为一个条件,并使
  用提取方法放置

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

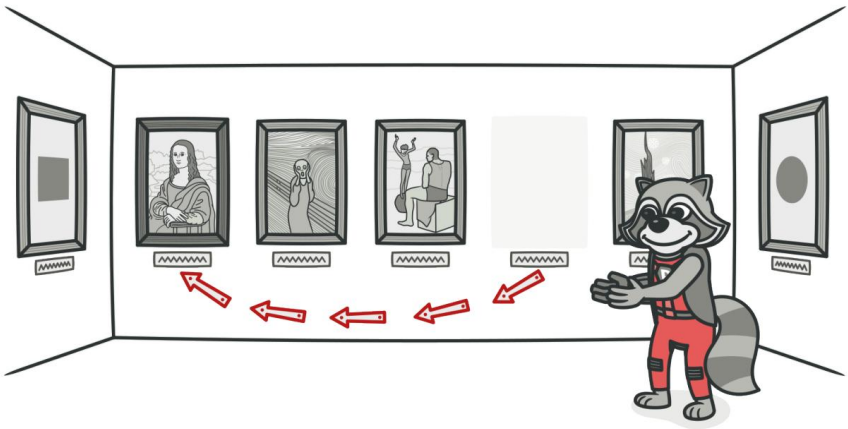
条件在一个单独的方法中,易于理解
站名。

- 如果在条件的所有分支中执行相同的代码作为表达式:将相同的代码放在条件之外
通过使用合并重复的条件片段来创建树。

清偿

- 合并重复代码可简化代码结构
并使其更短。

- 简化 + 简洁 = 更容易简化和简化的代码
支持更便宜。



lukas.haigner@gmail.com (#15232)

Machine Translated by Google

何时忽略

在极少数情况下,合并两个相同的代码片段
可以使代码不那么直观和明显。

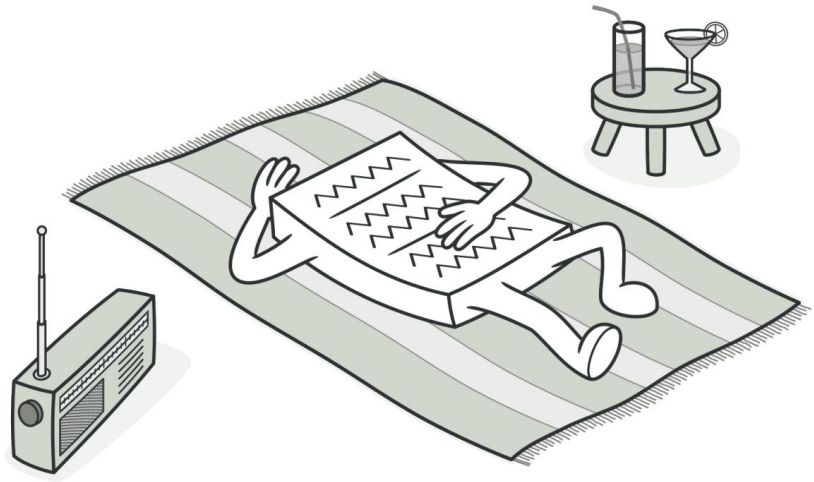
lukas.haigner@gmail.com (#15232)

Machine Translated by Google

懒惰的班级

体征和症状

理解和维护课程总是需要时间和金钱。因此，如果一门课程不足以引起您的注意，则应将其删除。



问题的原因

也许一个类被设计为功能齐全，但经过一些重构后，它变得小得离谱。

或者，它可能旨在支持未来从未完成的开发工作。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

治疗

- 几乎没用的组件应该被赋予 Inline 班级待遇。

- 对于功能很少的子类,请尝试折叠层次结构。

清偿

- 减少代码大小。

- 更容易维护。

何时忽略

有时会创建一个惰性类以描绘未来开发的意图,在这种情况下,请尝试在代码中保持清晰和简单之间的平衡。

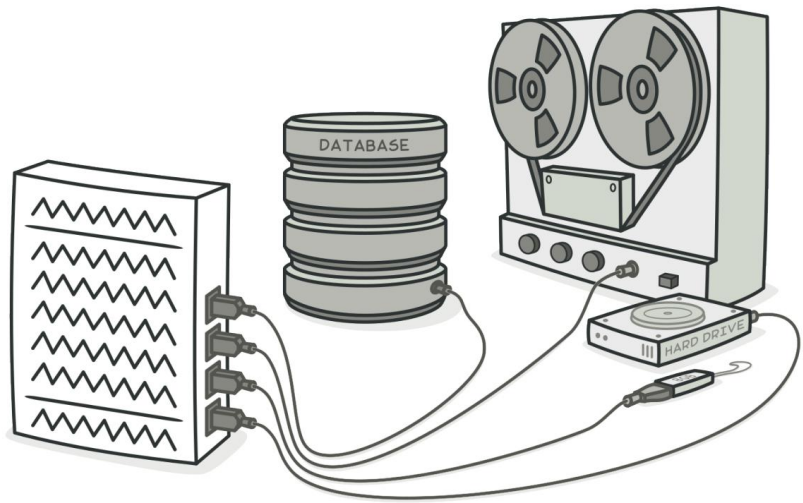
lukas.haigner@gmail.com (#15232)

Machine Translated by Google

一个数据类

体征和症状

数据类是指仅包含字段和访问它们的粗略方法（getter 和 setter)的类。这些只是其他类使用的数据的容器。这些类不包含任何附加功能，并且不能独立地对它们拥有的数据进行操作。



问题的原因

当一个新创建的类只包含几个公共字段（甚至可能是少数 getter/setter)时，这是很正常的事情。
但对象的真正力量在于它们可以包含对其数据的行为类型或操作。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

治疗

- 如果一个类包含公共字段，请使用封装字段隐藏它们以防止直接访问，并要求仅通过 getter 和 setter 执行访问。

- 对存储在集合中的数据使用封装集合（例如作为数组）。

- 查看使用该类的客户端代码。在其中，您可能会发现更好地位于数据类本身中的功能。如果是这种情况，请使用移动方法和提取方法将此功能迁移到数据类。

- 在类中填充了经过深思熟虑的方法后，您可能希望摆脱旧的数据访问方法，这些方法提供了对类数据的过于广泛的访问。为此，删除设置方法和隐藏方法可能会有所帮助。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

清偿

- 改进对代码的理解和组织。对特定数据的操作现在集中在一个地方，而不是在整个代码中随意进行。

- 帮助您发现客户端代码的重复。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

死代码

体征和症状

不再使用变量、参数、字段、方法或类（通常是因为它已过时）。

问题的原因

当对软件的要求发生变化或进行了更正时，没有人有时间清理旧代码。

当分支之一变得无法访问（由于错误或其他情况）时，也可以在复杂条件下找到此类代码。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

治疗

找到死代码的最快方法是使用好的 IDE。

- 删除未使用的代码和不必要的文件。
- 在不必要的类的情况下,如果使用子类或超类,则可以应用内联类或折叠层次结构。
- 要删除不需要的参数,请使用删除参数。

清偿

- 减少代码大小。
- 简单的支持。

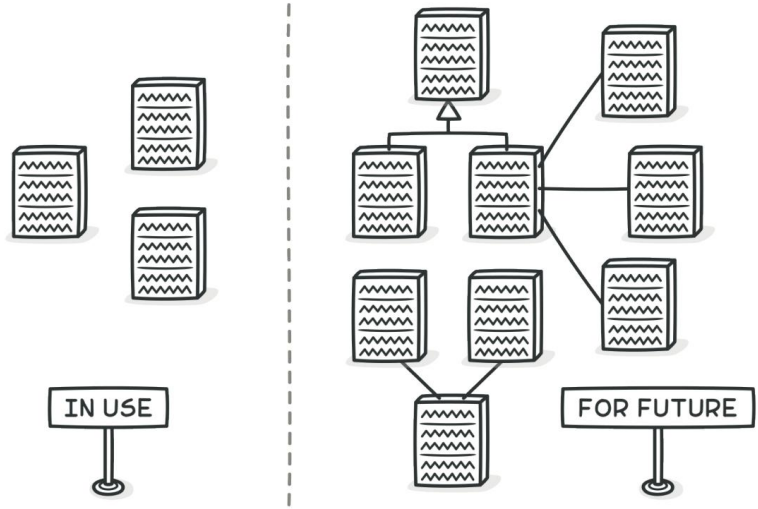
lukas.haigner@gmail.com (#15232)

Machine Translated by Google

投机的概论

体征和症状

有一个未使用的类、方法、字段或参数。



问题的原因

有时会创建代码以“以防万一”以支持从未实现的预期未来功能。结果，代码变得难以理解和支持。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

治疗

- 要删除未使用的抽象类,请尝试折叠层次结构。
- 将不必要的功能委托给另一个类可以
  通过内联类消除。
- 未使用的方法?使用内联方法摆脱它们。
- 应查看带有未使用参数的方法
  删除参数的帮助。
- 可以简单地删除未使用的字段。

清偿

- 精简代码。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

- 更轻松的支持。

何时忽略

- 如果您正在开发一个框架,那么非常合理
  创建框架本身未使用的功能,只要
  因为框架的用户需要该功能。

- 在删除元素之前,确保它们没有被用于
  单元测试。如果测试需要一种从类中获取某些内部信息或执行与测
  试相关的特殊信息的方法,则会发生这种情况
  行动。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

耦合器

该组中的所有气味都会导致类之间的过度耦合，或者表明如果耦合被过度委托取代会发生什么。

§ 功能羡慕

一个方法访问另一个对象的数据多于它自己的数据。

§ 不恰当的亲密关系

一个类使用另一个类的内部字段和方法
他上课。

§ 消息链

在代码中，您会看到一系列类似的调用

```
$a->b()->c()->d()
```

§ 中间人

如果一个类只执行一个动作，将工作委托给另一个类，那它为什么存在呢？

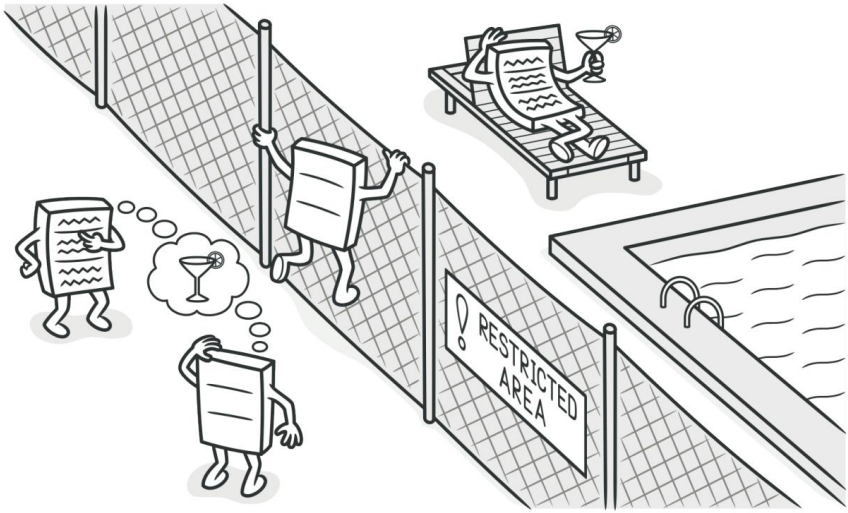
lukas.haigner@gmail.com (#15232)

Machine Translated by Google

功能嫉妒

体征和症状

一个方法访问另一个对象的数据多于它自己的数据。



问题的原因

在将字段移动到数据类之后，可能会出现这种气味。如果是这种情况，您可能还希望将数据操作移至此类。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

治疗

作为一个基本规则,如果事情同时发生变化,你应该
把它们放在同一个地方。通常数据和函数
使用这个数据一起改变（虽然例外是
可能的）。

- 如果一个方法显然应该移动到另一个地方,使用
  移动方法。

- 如果只有部分方法访问另一个对象的数据,
  使用提取方法移动有问题的部分。

- 如果一个方法使用了其他几个类的函数,首先
  确定哪个类包含使用的大部分数据。然后
  将该方法与其他数据一起放在此类中。或者,使用 Extract Method
  将方法拆分为几个
  可以放在不同类的不同地方的零件。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

清偿

- 更少的代码重复（如果数据处理代码放在一个 cen
  地方）。

- 更好的代码组织（接下来是处理数据的方法
  以实际数据为准）。

何时忽略

有时有目的地将行为与
保存数据的类。这样做的通常优点是
能够动态改变行为（参见策略、访问者和其他模式）。

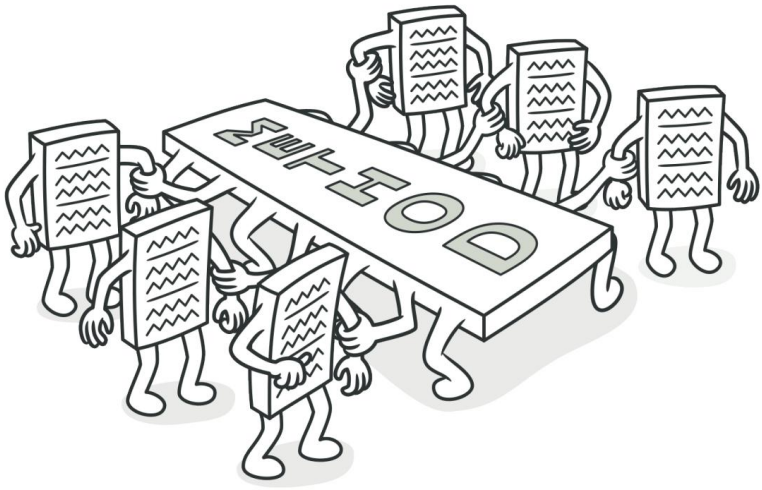
lukas.haigner@gmail.com (#15232)

Machine Translated by Google

不合适的亲密

体征和症状

一个类使用另一个类的内部字段和方法
他上课。



问题的原因

密切关注花太多时间在一起的课程。好的班级应该尽可能少地相互了解。这样的类更容易维护和重用。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

治疗

- 最简单的解决方案是使用移动方法和移动字段将一个类的部分移动到使用这些部分的类。但这只有在头等舱确实不需要这些部分时才有效。

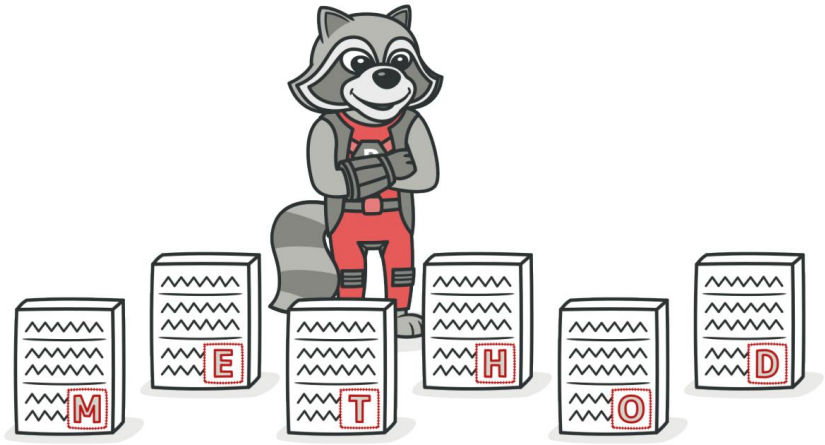
- 另一种解决方案是在类上使用Extract Class 和Hide Delegate 以使代码关系“正式”。

  - 如果类相互依赖,则应使用将双向关联更改为单向。

  - 如果这种“亲密关系”在子类和超类之间,请考虑将委托替换为继承。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google



清偿

- 改进代码组织。
- 简化支持和代码重用。

lukas.haigner@gmail.com (#15232)

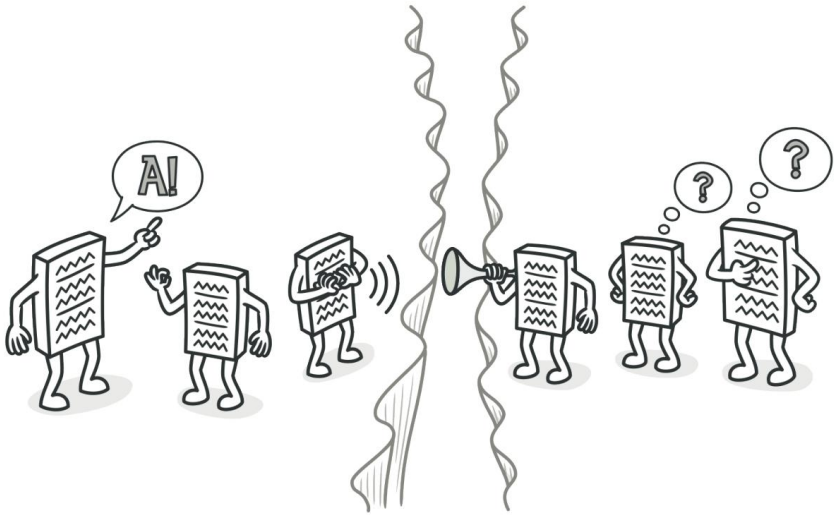
Machine Translated by Google

消息链

体征和症状

在代码中,您会看到一系列类似的调用

```
$a->b()->c()->d()
```



问题的原因

当客户端请求另一个对象,该对象又请求另一个对象,依此类推时,就会出现消息链。这些链意味着客户端依赖于类结构的导航。这些关系中的任何更改都需要修改客户端。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

治疗

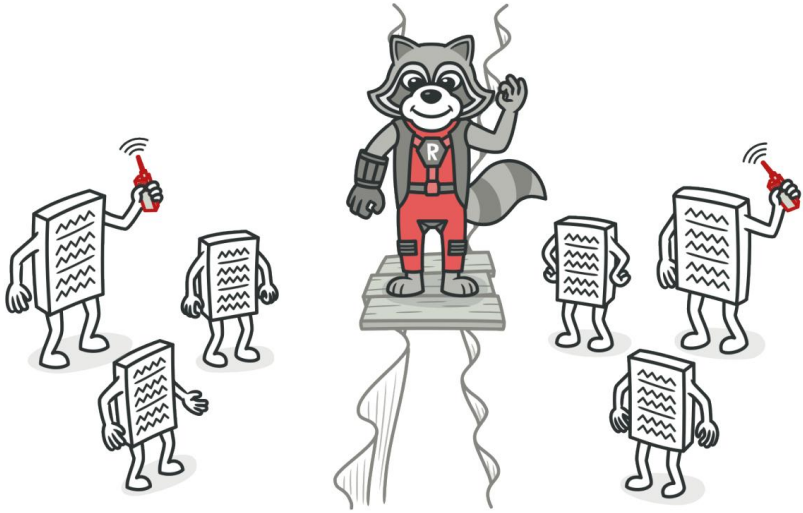
- 要删除消息链，请使用隐藏代理。
- 有时最好考虑一下使用最终对象的原因。也许对这个功能使用提取方法并通过使用移动方法将其移动到链的开头是有意义的。

清偿

- 减少链中类之间的依赖关系。
- 减少臃肿代码的数量。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google



何时忽略

过于激进的委托隐藏可能会导致代码难以看到功能实际发生的位置。这是另一种说法，也要避免中间人的气味。

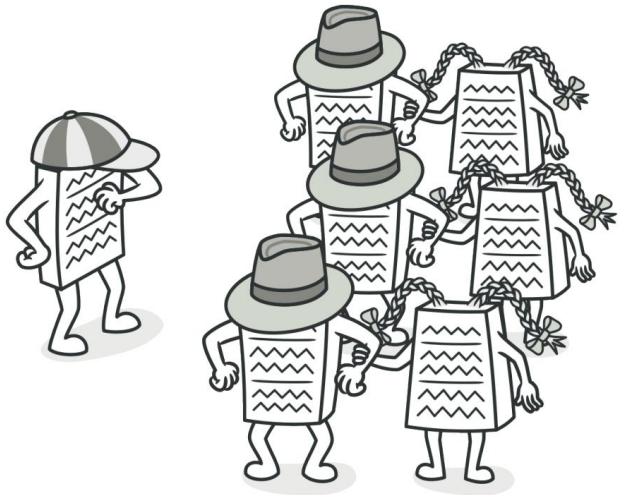
lukas.haigner@gmail.com (#15232)

Machine Translated by Google

一个中间人

体征和症状

如果一个类只执行一个动作,将工作委托给另一个类,那它为什么存在呢?



问题的原因

这种气味可能是过度消除 Mes 的结果
说链子。

在其他情况下,它可能是一个类的有用工作逐渐转移到其他类的结果。该类仍然是一个空壳,除了委托之外不做任何事情。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

治疗

如果一个方法的大部分类委托给另一个类，
删除中间人是为了。

清偿

更少的代码。

何时忽略

不要删除出于某种原因创建的中间人：

- 可能增加了一个中间人以避免课间
  依赖关系。

- 一些设计模式故意创建中间人（例如
  代理或装饰）。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

其他气味

以下是不属于任何广泛类别的气味。

§ 不完整的图书馆类

> 图书馆迟早会停止满足用户的需求。该问题的唯一解决方案 - 更改库 - 通常是不可能的,因为库是只读的。

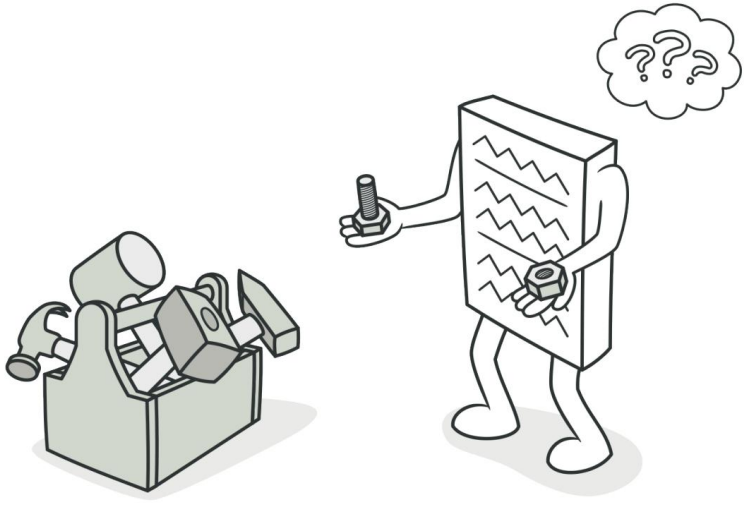
lukas.haigner@gmail.com (#15232)

Machine Translated by Google

不完整的图书馆班级

体征和症状

图书馆迟早会停止满足用户的需求。该问题的唯一解决方案 - 更改库 - 通常是不可能的,因为库是只读的。



问题的原因

该库的作者没有提供您需要的功能或拒绝实现它们。

lukas.haigner@gmail.com (#15232)

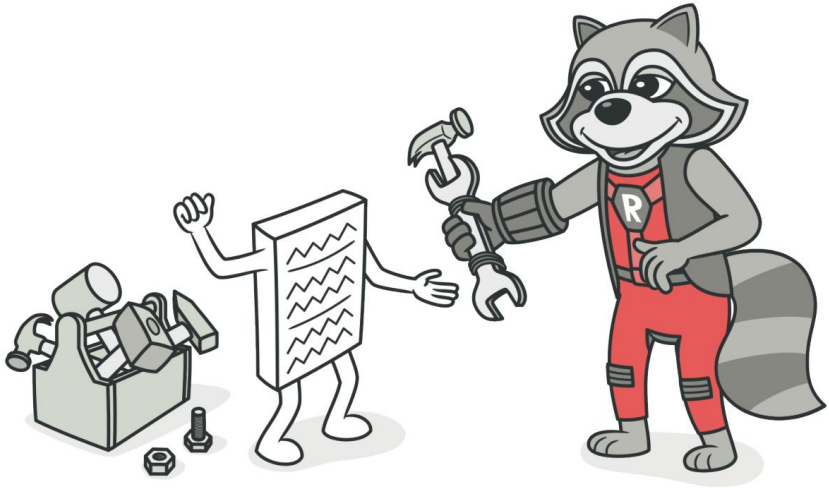
Machine Translated by Google

治疗

- 要为库类引入一些方法,请使用 Introduce Foreign Method。
- 对于类库中的重大更改,使用 Introduce Local Exten
  版本

清偿

减少代码重复（而不是从头开始创建自己的库,您仍然可以捎带现有的库）。



何时忽略

如果对库的更改涉及代码更改,则扩展库可能会产生额外的工作。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

重构技术

重构是在不创建新功能的情况下改进代码的可控过程。它将混乱转化为干净的代码和简单的设计。

干净的代码是易于阅读、编写和维护的代码。
干净的代码使软件开发可预测并提高最终产品的质量。

重构技术描述了实际的重构步骤。大多数重构技术都有其优点和缺点。因此，每次重构都应该有适当的动机并谨慎应用。

在前面的章节中，您已经看到特定的重构如何帮助解决代码问题。现在是时候更详细地查看重构技术了！

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

作曲方法

大部分重构都致力于正确组合方法。在大多数情况下，过长的方法是

万恶这些方法中的代码变幻莫测
执行逻辑并使该方法极难
理解 甚至更难改变。

该组中的重构技术简化了方法，消除了代码重复，并为未来铺平了道路

改进。

§ 提取方法

问题：你有一个可以分组的代码片段
一起。

解决方案：将此代码移至单独的新方法（或函数），并用对该方法的调用替换旧代码。

§ 内联方法

问题：当方法体比
方法本身，请使用此技术。

解决方案：用方法的内容替换对方法的调用，并删除方法本身。

§ 提取变量

问题：你有一个难以理解的表达方式。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

解决方案:将表达式的结果或其部分放在不言自明的单独变量中。

§ 内联温度

问题:您有一个临时变量,它分配了一个简单表达式的结果,仅此而已。

解决方案:用表达式本身替换对变量的引用。

§ 用查询替换 Temp

问题:您将表达式的结果放在局部变量中以供以后在代码中使用。

解决方案:将整个表达式移动到一个单独的方法中并从中返回结果。查询方法而不是使用变量。如有必要,将新方法合并到其他方法 ods 中。

§ 拆分临时变量

问题:您有一个用于存储 var 的局部变量
方法内部的 ious 中间值(循环变量除外)。

解决方案:对不同的值使用不同的变量。每个
变量应该只负责一件特定的事情。

§ 删除分配给参数

问题:一些值被分配给方法体内的参数。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

解决方案:使用局部变量而不是参数。

§ 用方法对象替换方法

问题:您有一个很长的方法,其中局部变量如此交织在一起,以至于您无法应用提取方法。

解决方案:将方法转换为单独的类,使局部变量成为类的字段。然后,您可以将该方法拆分为同一类中的多个方法。

§ 替代算法

问题:所以你想用新算法替换现有算法?

解决方案:用新算法替换实现算法的方法体。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

B提取方法

问题

您有一个可以组合在一起的代码片段。

```
1无效printOwing() {
2        打印横幅（）；
3
4        //打印详细信息。
5        System.out.println(  名称：                    +名称）；
6        System.out.println(  金额：     + getOutstanding());
7 }
```

解决方案

将此代码移动到单独的新方法（或函数)和用对方法的调用替换旧代码。

```
1无效printOwing() {
2        打印横幅（）；
3        printDetails(getOutstanding());
4 }
5
6 void printDetails（双未完成） {
7        System.out.println(  名称：                    +名称）；
8        System.out.println(  金额：     + 未完成);
```

lukas.haigner@gmail.com (#15232)

Machine Translated by Google


```
9 }
```

为什么要重构

在方法中找到的行越多,就越难弄清楚该方法的作用。这是造成这种情况的主要原因
重构。

除了消除代码中的粗糙边缘之外,提取方法也是许多其他重构方法中的一个步骤。

好处

- 更易读的代码!确保给新方法起一个描述方法用途的名称： createOrder()
  ,
  渲染客户信息（） , 等等。
- 更少的代码重复。通常,在方法中找到的代码可以在程序的其他地方重用。因此,您可以使用对新方法的调用来替换重复项。
- 隔离代码的独立部分,这意味着出错的可能性较小（例如,如果修改了错误的变量）。

如何重构

1. 创建一个新方法,并以使其纯粹的姿势不言而喻的方式命名。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

2. 将相关代码片段复制到您的新方法中。从旧位置删除片段并调用新位置

代替那里的方法。

查找此代码片段中使用的所有变量。如果它们在片段内部声明而不在片段外部使用，只需保持不变 它们将成为片段的局部变量

新方法。

3. 如果变量在您提取的代码之前声明，您需要将这些变量传递给您的新方法的参数，以便使用之前包含在其中的值。有时通过用查询替换临时来更容易摆脱这些变量。

---

4. 如果您在提取的代码中看到局部变量以某种方式发生变化，这可能意味着稍后在您的 main 方法中将需要此更改的值。再确认一次！如果确实如此，则将此变量的值返回给 main 方法以保持一切正常。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

反重构

§ 在线方法

类似的重构

§ 移动方法

帮助其他重构

§ 引入参数对象

§ 表格模板法

§ 引入参数对象

§ 参数化方法

消除异味

§ 重复代码

§ 长法

§ 功能羡慕

§ 切换语句

§ 消息链

§ 评论

§ 数据类

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

B在线方法

问题

当方法体比方法本身更明显时，
使用这种技术。

```
1类披萨外卖{
2        // ...
3        int getRating() {
4            返回moreThanFiveLateDeliveries() ？ 2： 1;
5        }
6        boolean moreThanFiveLateDeliveries() {
7            返回numberOfLateDeliveries > 5;
8        }
9    }
```

解决方案

用方法的内容替换对方法的调用，
删除方法本身。

```
1类披萨外卖{
2        // ...
3        int getRating() {
4            返回numberOfLateDeliveries > 5 ？ 2： 1;
```

lukas.haigner@gmail.com (#15232)

Machine Translated by Google


```
5         }
6 }
```

为什么要重构

一个方法只是委托给另一个方法。这个代表团本身没有问题。但是当有很多这样的方法时，它们就会变成一个难以理清的混乱纠结。

通常方法最初并不太短，但随着程序的更改而变得如此。所以不要羞于摆脱那些已经过时的方法。

好处

通过最小化不需要的方法的数量，您可以使代码更直接。

如何重构

1. 确保方法没有在子类中重新定义。如果
   方法被重新定义，避免使用这种技术。

2. 查找对该方法的所有调用。用 con 替换这些调用
   帐篷的方法。

3. 删除方法。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

反重构

§ 提取方法

消除异味

§ 推测的普遍性

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

B提取变量

问题

你的表情很难理解。

```
无效渲染横幅（）{
    if ((platform.toUpperCase().indexOf(  MAC  ) > -1) &&
            (browser.toUpperCase().indexOf(  IE  ) > -1) &&
             wasInitialized() && 调整大小 > 0 )
    {
        // 做点什么
    }
}
```

解决方案

将表达式的结果或其部分放在单独的
不言自明的变量。

```
无效渲染横幅（）{
    final boolean isMacOs = platform.toUpperCase().indexOf(  MAC  ) > -1;
    最终布尔isIE = browser.toUpperCase().indexOf(  IE  ) > -1;
    final boolean wasResized = resize > 0;

    if (isMacOs && isIE && wasInitialized() && wasResized) {
        // 做点什么
```

lukas.haigner@gmail.com (#15232)

Machine Translated by Google


```
8日         }
9 }
```

为什么要重构

提取变量的主要原因是为了做一个复杂的
通过将其分成中间部分,表达更容易理解。这些可能是:

- if()运算符或?:运算符的一部分的条件
  在基于 C 的语言中
- 没有中间结果的长算术表达式
- 长的多段线

提取变量可能是执行的第一步
如果您看到使用了提取的表达式,则提取方法
在你的代码的其他地方。

好处

更易读的代码!尽量给提取的变量好
大声而清晰地宣布变量用途的名称。
更具可读性,更少冗长的评论。去找名字
像 `客户税值`, `城市失业率,`
`clientSalutationString,` 等等。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

缺点

- 您的代码中存在更多变量。但这与阅读代码的便利性相抵消。

- 重构条件表达式时，请记住

  编译器很可能会对其进行优化以最小化

  确定结果值所需的计算。说你

  有以下表达式if (a() || b()) ... 。如果方法a返回true ，程序将不会调用方法b

  因为结果值仍然是true值返回b 。 ，无论

  但是，如果您将此表达式的部分内容提取到变量中，

  这两种方法都会被调用，这可能会损害程序的性能，特别是如果这些方法做了一些

  繁重的工作。

如何重构

1. 在相关表达式前插入新行并声明一个

   那里的新变量。将复杂表达式的一部分分配给

   这个变量。

2. 用新变量替换表达式的那部分。

3. 对表达式的所有复杂部分重复该过程。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

反重构

§ 内联温度

类似的重构

§ 提取方法

消除异味

§ 评论

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

B在线温度

问题

您有一个临时变量,它分配了一个简单表达式的结果,仅此而已。

```
1 boolean hasDiscount(Order order) {
2         双基价格 = order.basePrice();
3         返回基本价格 > 1000;
4 }
```

解决方案

用表达式本身替换对变量的引用。

```
1 boolean hasDiscount(Order order) {
2         返回order.basePrice() > 1000;
3 }
```

为什么要重构

内联局部变量几乎总是用作 Replace Temp with Query 的一部分或为 Extract Method 铺平道路。

lukas.haigner@gmail.com (#15232)

Machine Translated by Google

好处

这种重构技术本身几乎没有任何好处。但是,如果将变量分配为方法的结果,则可以通过删除不必要的变量来略微提高程序的可读性。

缺点

有时,看似无用的临时文件用于缓存重复使用多次的昂贵操作的结果。因此,在使用这种重构技术之前,请确保简单性不会以性能为代价。

如何重构

1. 找到所有使用该变量的地方。而不是变量,使用已分配给它的表达式。

2. 删除变量的声明及其赋值行。

帮助其他重构

§ 用查询替换 Temp

§ 提取方法

lukas.haigner@gmail.com (#15232)