Dive Into *Refactoring*

Offline Edition

REFACTORING
·GURU·

by Alexander Shvets

潜入重构

离线版 (Java)

v2019-1.3

无聊的版权页

你好!我的名字是 Alexander Shvets,我是 Dive 在线课程的作者进入重构,其中包括这本书。

感谢您尊重我多年来为创建课程和本书所做的辛勤工作!

Alexander Shvets,Refactoring.Guru,2019 年
support@refactoring.guru

Dmitry Zhart 的插图

前言

我试图将整个课程中所有可用的信息都塞进本书中。在大多数情况下,我成功了。但是有些东西,比如活生生的例子,是不可能以静态电子书的形式呈现的。因此,将本书视为辅助材料,而不是完整的重构课程的替代品。

本书分为两大部分:代码味道和重构技术。第一部分描述脏代码的各种迹象和症状。第二部分展示了处理脏代码并使其干净的不同方法。

这本书既可以从头到尾阅读,也可以在
随机顺序。尽管所有主题都紧密交织在一起,但您可以使用散布在文本中的大量链接轻松地跳过章节。

该版本本书中的代码示例使用 Java。
还有其他版本可供下载
在您的帐户中。

代码气味

代码异味是需要重构的关键标志。在里面
在重构过程中，我们去除异味，进一步启用
以相同或更快的速度开发应用程序。

缺乏定期重构，可能会导致项目随着时间的推移完全瘫痪，浪费几年的开发时间，需要
你再花几年的时间来重写

从头开始。

因此，有必要消除代码异味，同时
仍然很小。

腹胀

Bloaters 是代码、方法和类，它们已经增加到难以处理的巨大比例。

通常这些气味不会立即出现，而是随着程序的发展而随着时间的推移而积累（尤其是当没有人努力消除它们时）。

§ 长法

一个方法包含太多的代码行。一般来说，任何超过十行的方法都应该让你开始提问。

§ 大班

一个类包含许多字段/方法/代码行。

§ 原始痴迷

- 在简单任务中使用原语而不是小对象（例如货币、范围、电话号码的特殊字符串等）
- 使用常量来编码信息（例如常量USER_ADMIN_ROLE = 1用于引用具有管理员权限的用户。）
- 使用字符串常量作为数据数组中的字段名称。

§ 长参数列表

一个方法的参数超过三个或四个。

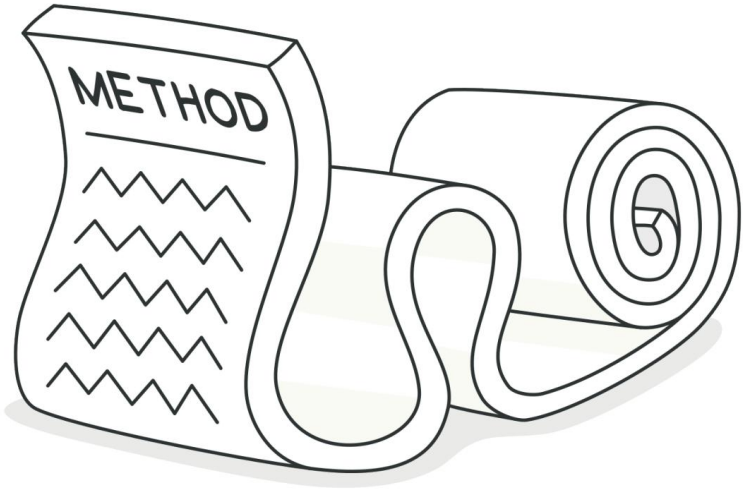
§ 数据块

> 有时代码的不同部分包含相同的变量组（例如用于连接数据库的参数）。
> 这些团块应该变成它们自己的类。

一个长方法

体征和症状

一个方法包含太多的代码行。一般来说,任何超过十行的方法都应该让你开始提问。



问题的原因

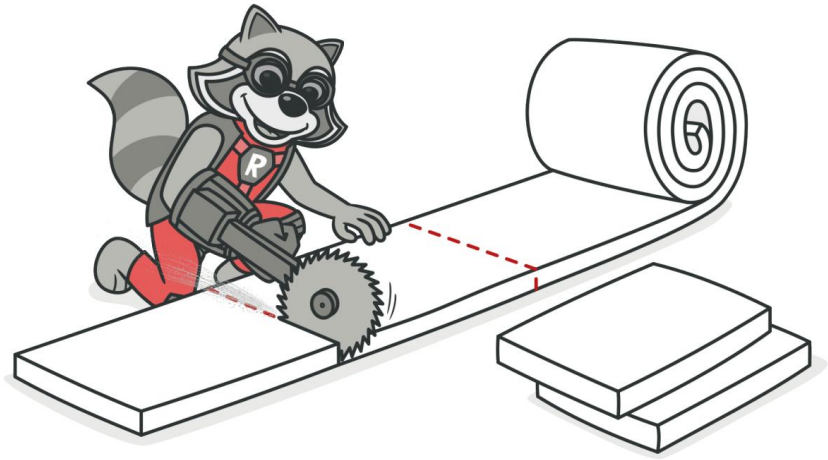
就像加利福尼亚酒店一样,总是在方法中添加一些东西,但从来没有取出任何东西。由于编写代码比阅读代码更容易,因此这种“气味”在方法变成丑陋的、超大的野兽之前不会引起注意。

在精神上,创造一种新方法通常比添加
对现有的:“但这只是两行,仅仅为此创建一个完整的方法是没有用的……”这意味
着添加了另一行,然后又添加了另一行,从而产生了纠结

意大利面条代码。

治疗

根据经验,如果您觉得需要对方法中的某些内容进行注释,您应该获取此代码并将其
放入
一种新方法。即使是单行也可以而且应该分开
如果需要解释,请放入单独的方法中。如果
方法有一个描述性的名称,没有人需要看
代码看看它做了什么。



· 要减少方法主体的长度,请使用提取方法。

- 如果局部变量和参数干扰提取
  方法,使用 Replace Temp with Query， Introduce Parameter
  对象或保留整个对象。

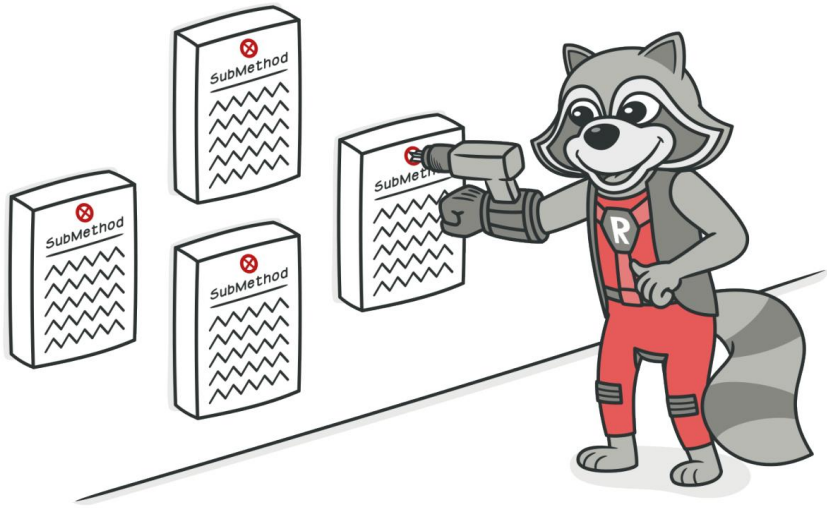
- 如果以前的食谱都没有帮助,请尝试移动整个
  通过用方法替换方法到单独的对象的方法
  对象。

- 条件运算符和循环是一个很好的线索,代码可以
  移至单独的方法。对于条件句,使用 Decompose Conditional。如果有循环，
  请尝试提取方法。

清偿

- 在所有类型的面向对象代码中,具有短
  方法寿命最长。方法或函数越长，
  更难理解和维护它。

- 此外,长方法提供了完美的藏身之处
  不需要的重复代码。



表现

方法数量的增加是否会像许多人所说的那样损害性能?在几乎所有情况下,影响都微不足道,甚至不值得担心。

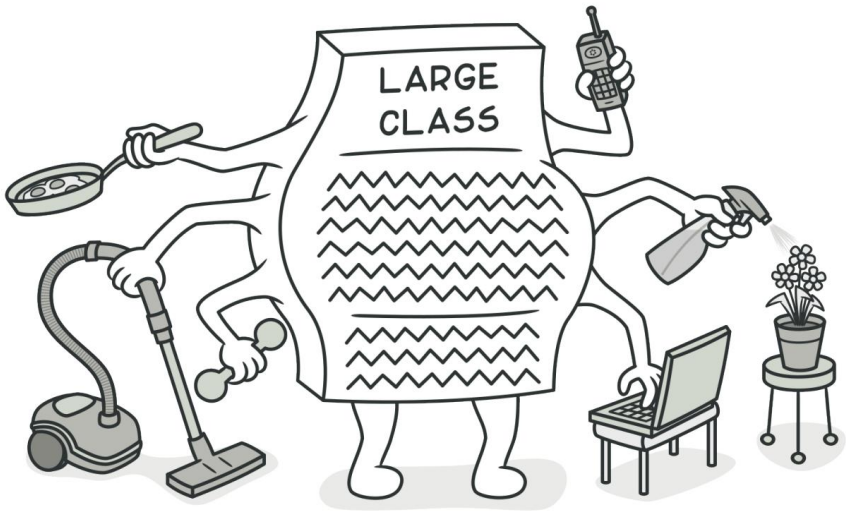
此外,既然您拥有清晰易懂的代码,您就更有可能找到真正有效的方法来重构代码并在需要时获得真正的性能提升

出现。

大班

体征和症状

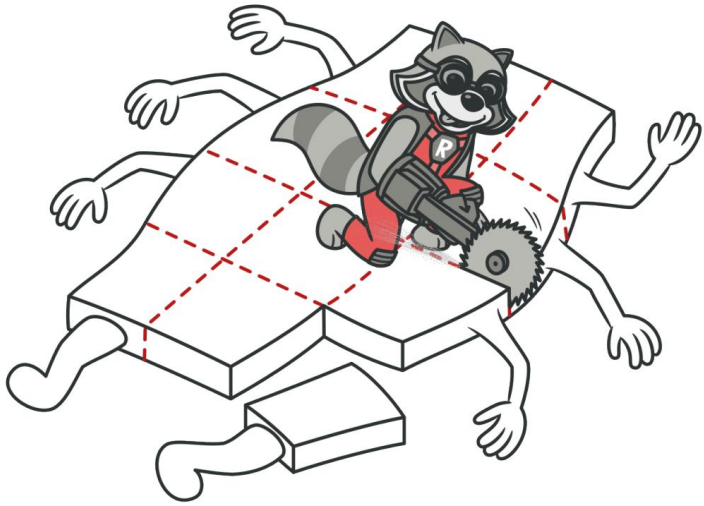
一个类包含许多字段/方法/代码行。



问题的原因

课程通常从小开始。但随着时间的推移，随着程序的发展，它们会变得臃肿。

与长方法的情况一样，程序员通常会发现在现有类中放置一个新特性比为该特性创建一个新类更容易。



治疗

当一个班级戴着太多（功能性）帽子时，请考虑将其分开：

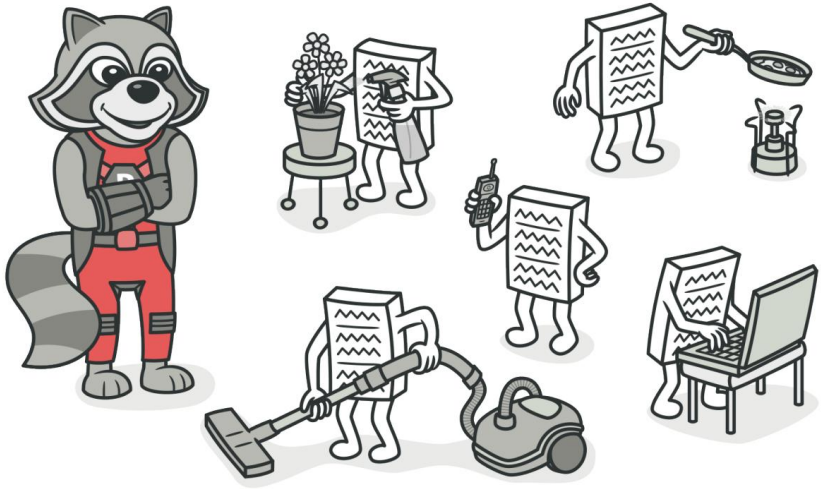
·如果可以将大类的部分行为拆分为单独的组件，则提取类会有所帮助。

· 如果大类的部分行为可以以不同的方式实现或在极少数情况下使用，则提取子类会有所帮助。

· 如果有必要列出客户可以使用的操作和行为，则提取接口会有所帮助。

· 如果一个大类负责图形界面，您可能会尝试将它的一些数据和行为移动到一个单独的域对象中。在这样做时，可能需要存储

在两个地方复制一些数据并保持数据一致。 Duplicate Observed Data提供了一种方法来做到这一点。



清偿

- 这些类的重构使开发人员无需记住一个类的大量属性。

- 在许多情况下,将大类拆分为多个部分可以避免代码和功能的重复。

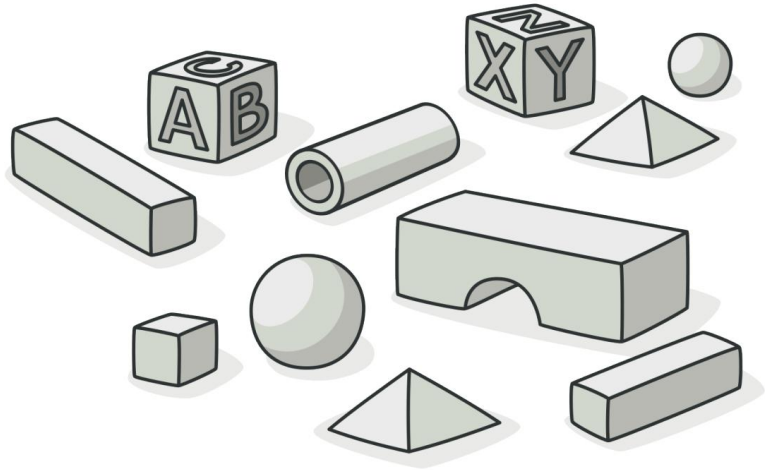
原始的痴迷

体征和症状

- 在简单任务中使用原语而不是小对象（例如货币、范围、电话号码的特殊字符串等）

- 使用常量来编码信息（例如常量USER_ADMIN_ROLE = 1用于引用具有管理员权限的用户。）

- 使用字符串常量作为数据数组中的字段名称。



问题的原因

像大多数其他气味一样,原始的痴迷是在虚弱的时刻产生的。 “只是一个存储一些数据的字段!”程序员说。创建一个原始字段比创建一个全新的类要容易得多,对吧?就这样完成了。

然后需要另一个字段并以相同的方式添加。瞧,这门课变得庞大而笨拙。

基元通常用于“模拟”类型。因此,您拥有一组数字或字符串,而不是单独的数据类型,这些数字或字符串构成了某些实体的允许值列表。然后通过常量为这些特定的数字和字符串赋予易于理解的名称,这就是它们广泛传播的原因

和远。

原始使用不佳的另一个例子是现场模拟。该类包含大量不同数据和字符串常量(在类中指定)用作获取此数据的数组索引。

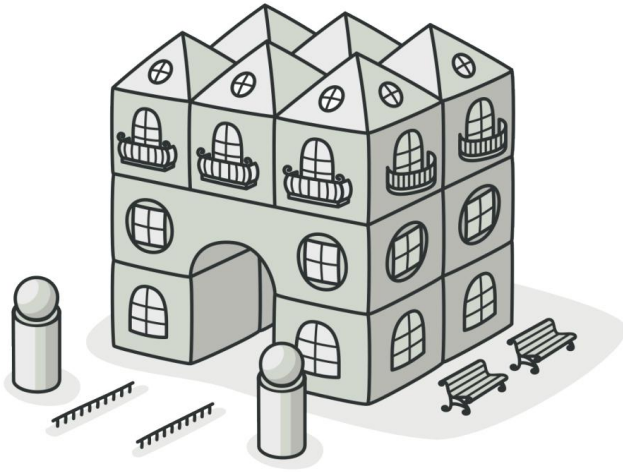
治疗

- 如果你有大量的原始字段,它可能是可能的
  在逻辑上将其中一些分组到他们自己的类中。更好的是,将与此数据
  相关的行为移到类中
  也。对于此任务,请尝试将数据值替换为对象。

- 如果方法参数中使用原始字段的值,
  使用 Introduce Parameter Object 或 Preserve Whole Object。

- 在变量中编码复杂数据时,使用替换类型
  带类的代码,用子类替换类型代码或替换
  带有状态/策略的类型代码。

- 如果变量之间存在数组,请使用 Replace Array with
  对象。



清偿

- 由于使用对象而不是原语,代码变得更加灵活。

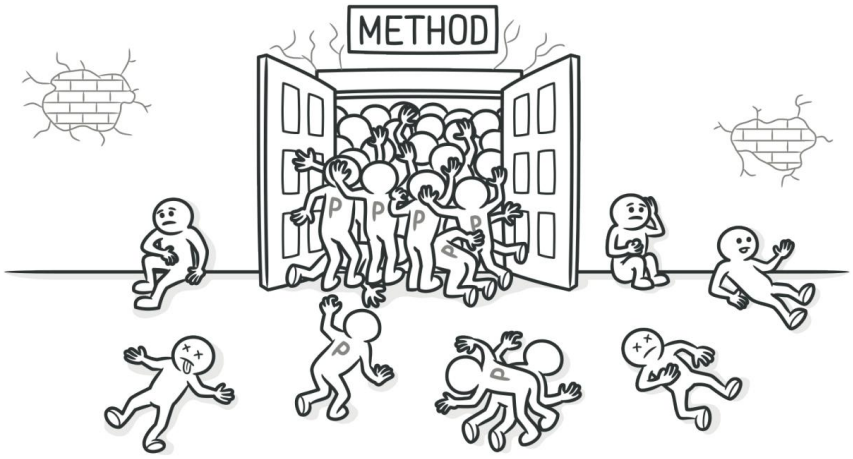
- 更好地理解和组织代码。对特定数据的操作都在同一个地方,而不是分散。无需再猜测所有这些奇怪常量的原因以及它们为何存在于数组中。

- 更容易找到重复代码。

很长的参数列表

体征和症状

一个方法的参数超过三个或四个。



问题的原因

在将几种类型的算法合并为一个方法后,可能会出现一长串参数。可能已经创建了一个长列表来控制将运行哪个算法以及如何运行。

长参数列表也可能是努力使类彼此更加独立的副产品。例如,用于创建方法中所需的特定对象的代码是

从方法移到调用方法的代码,但创建的对象作为参数传递给方法。这样原来的类就不再知道对象之间的关系了,依赖也减少了。

但是如果创建了几个这样的对象,每个对象都需要自己的参数,这意味着更长的参数名单。

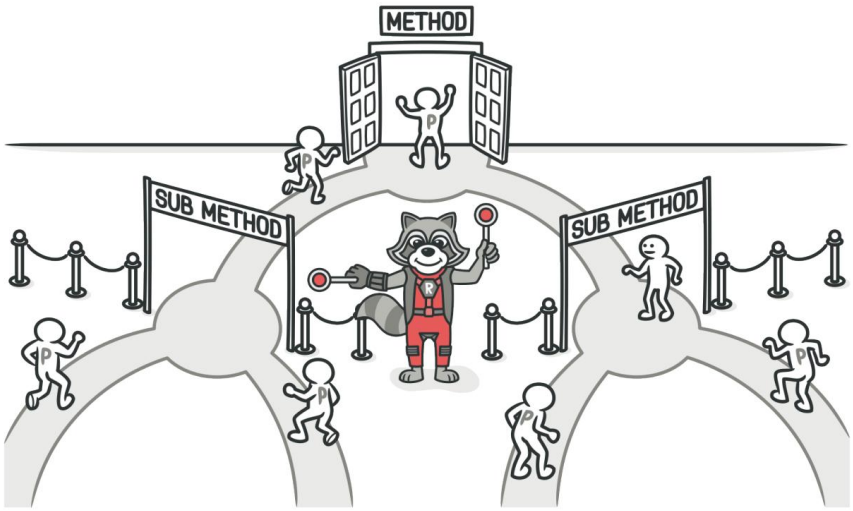
很难理解这样的列表,随着它们的增长,它们变得矛盾且难以使用。一个方法可以使用它自己对象的数据,而不是一长串参数。如果当前对象不包含所有必要的数据,则可以将另一个对象(它将获取必要的数据)作为方法参数传递。

治疗

- 检查传递给参数的值。如果某些参数只是另一个对象的方法调用的结果,请使用方法调用替换参数。这个对象可以放在自己类的字段中,也可以作为方法参数传递。

- 不是将从另一个对象接收到的一组数据作为参数传递,而是通过使用保留整个对象将对象本身传递给方法。

- 如果有多个不相关的数据元素,有时您可以通过 Introduce Parameter Object 将它们合并为单个参数对象。



清偿

- 更易读、更短的代码。
- 重构可能会揭示以前未被注意的重复代码。

何时忽略

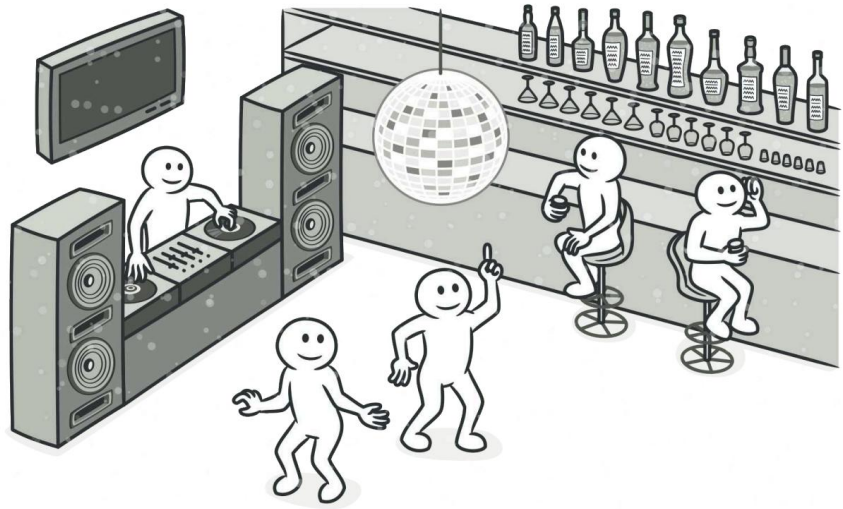
如果这样做会导致类之间不需要的依赖关系,请不要删除参数。

一个数据块

体征和症状

有时代码的不同部分包含相同的变量组（例如用于连接数据库的参数）。

这些团块应该变成它们自己的类。



问题的原因

通常这些数据组是由于糟糕的程序结构或“copypasta 编程”造成的。

如果要确定某些数据是否是数据块，只需删除其中一个数据值，看看其他值是否仍然有意义。如果不是这种情况，这是一个

这组变量应该组合成一个好兆头
反对。

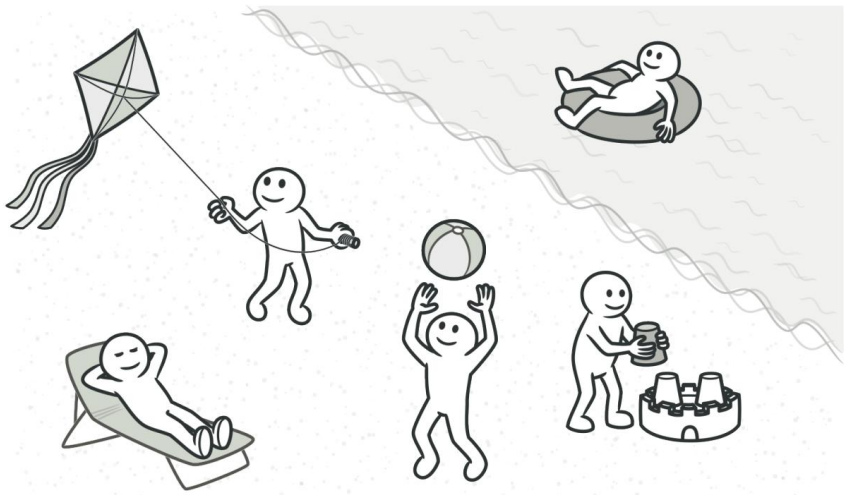
治疗

- 如果重复数据包含类的字段,请使用提取
  类将字段移动到自己的类。

- 如果在方法的参数中传递相同的数据块,请使用 Introduce Parameter
  Object 将它们设置为一个类。

- 如果将部分数据传递给其他方法,请考虑
  将整个数据对象传递给方法,而不仅仅是
  个别领域。保留整个对象将对此有所帮助。

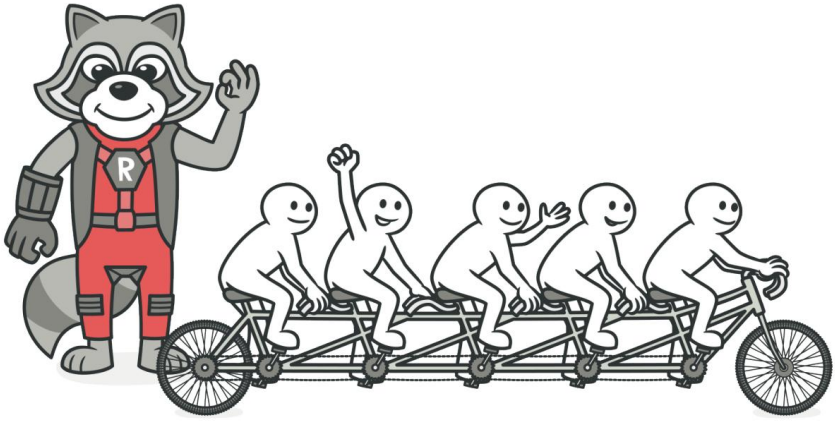
- 查看这些字段使用的代码。这可能是个好主意
  将此代码移动到数据类。



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。

- 减少代码大小。



何时忽略

在方法的参数中传递整个对象,而不是仅传递其值（原始类型）,可能会在两个类之间产生不希望的依赖关系。

面向对象滥用者

所有这些气味都是面向对象编程原则的不完整或错误应用。

§ 切换语句

您有一个复杂的switch运算符或if序列
陈述。

§ 临时场地

临时字段只有在某些情况下才能获得它们的值（因此是对象需要的）。在这些情况之外，它们是空的。

§ 拒绝请求

如果子类仅使用从其父类继承的一些方法和属性，则层次结构是不平衡的。不需要的方法可能只是不使用或重新定义并发出异常。

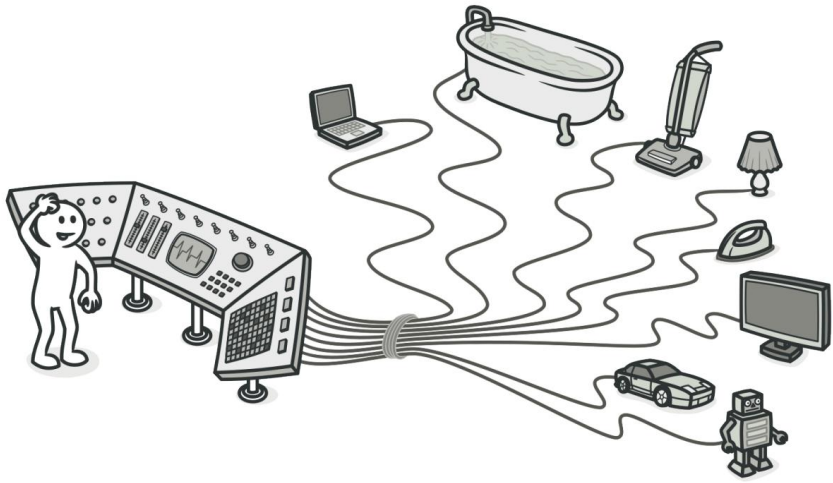
§ 具有不同接口的替代类

两个类执行相同的功能但具有不同的方法名称。

一个switch 语句

体征和症状

您有一个复杂的switch运算符或if序列
陈述。



问题的原因

相对较少使用switch和case运算符是其中之一
面向对象代码的特点。通常单个开关的代码可以分散在程序的不同
位置。添加新条件时，您必须找到所有

切换代码并修改它。

根据经验,当你看到switch时,你应该想到
多态性。

治疗

- 要隔离开关并将其放在正确的类中,您可能需要
  提取方法,然后移动方法。

- 如果开关基于类型代码,例如当程序的
  切换运行时模式,使用 Replace Type Code with Sub classes 或 Replace
  Type Code with State/Strategy。

- 指定继承结构后,使用Replace Conditional with Polymorphism。

- 如果运算符中没有太多条件并且它们
  都用不同的参数调用相同的方法,多态性
  将是多余的。如果是这种情况,您可以打破该方法
  使用 Replace Parameter with 转换为多个较小的方法
  显式方法并相应地更改开关。

- 如果条件选项之一是空对象。 , 使用引入 Null

清偿

改进的代码组织。

何时忽略

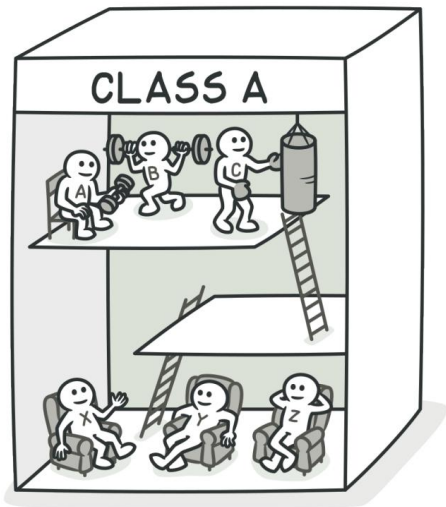
- 当开关操作员执行简单的操作时，没有理由更改代码。

- 工厂设计模式（工厂方法或抽象工厂）经常使用开关操作符来选择创建的类。

临时场

体征和症状

临时字段获取它们的值（因此需要
对象）仅在某些情况下。在这些环境之外，它们是空的。



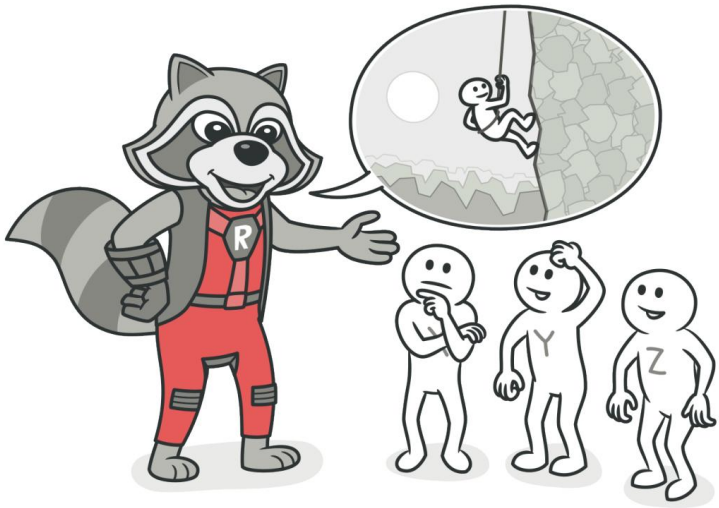
问题的原因

通常，会创建临时字段以用于需要大量输入的算法。所以而不是

在方法中创建大量参数时，程序员决定在类中为这些数据创建字段。

这些字段仅在算法中使用，未使用
其余时间。

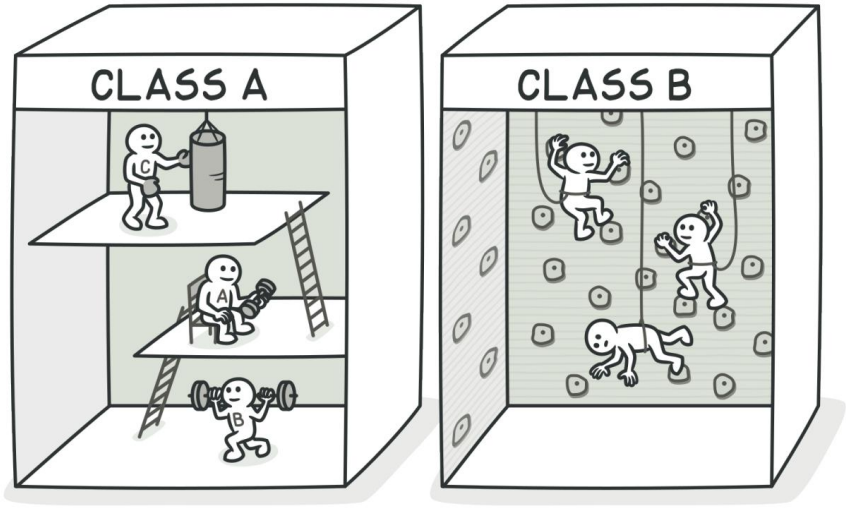
这种代码很难理解。你期待看到
对象字段中的数据，但由于某种原因，它们几乎
总是空的。



治疗

- 可以放入临时字段和对其进行操作的所有代码
  通过提取类单独的类。换句话说，您正在创建一个方法对象，获得的结果与您
  用方法对象执行替换方法。

- 引入 Null Object 并集成它来代替用于检查临时字段值的条件代码
  存在。



清偿

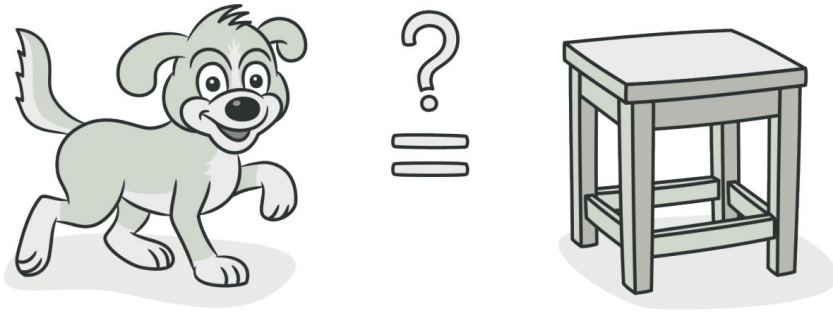
更好的代码清晰度和组织。

被拒绝的请求

体征和症状

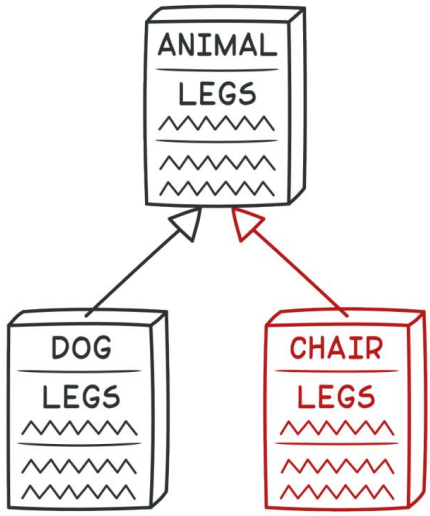
如果子类仅使用从其父类继承的一些方法和属性，则层次结构是不平衡的。这

不需要的方法可能会简单地不用或重新定义
放出例外。



问题的原因

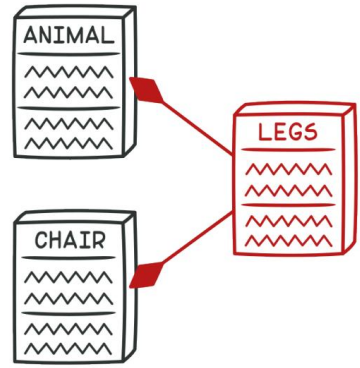
有人被激励在类之间创建继承
它只是希望重用超类中的代码。但是
超类和子类完全不同。



治疗

- 如果继承没有意义,而子类确实如此
  与超类没有任何共同之处,消除继承以支持用委托替换继承。

- 如果继承是合适的,去掉不需要的字段和
  子类中的方法。提取所有需要的字段和方法
  由父类的子类编辑,将它们放在一个新的
  子类,并将两个类都设置为从它继承（提取超类）。



清偿

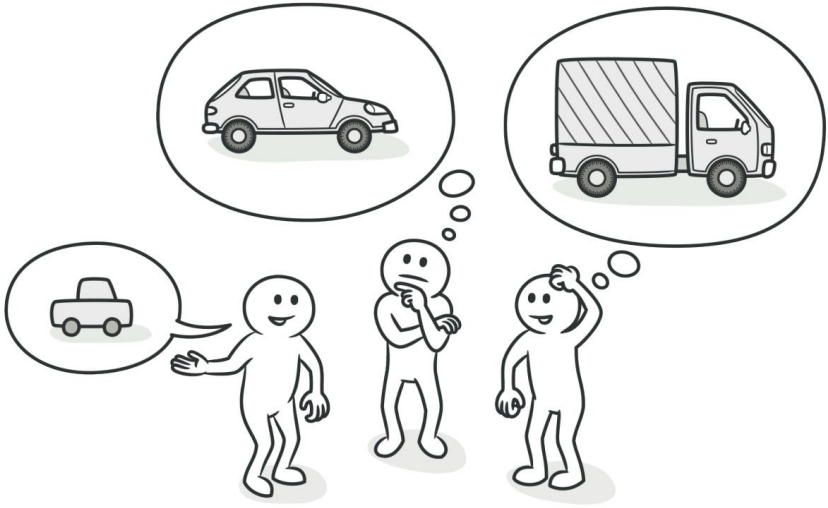
提高代码的清晰度和组织性。您将不再需要怀疑为什么Dog类是从Chair类继承的（即使它们都有 4 条腿）。

具有不同的替代类

接口

体征和症状

两个类执行相同的功能但具有不同的方法名称。



问题的原因

创建其中一个类的程序员可能不知道功能等效的类已经

存在。

治疗

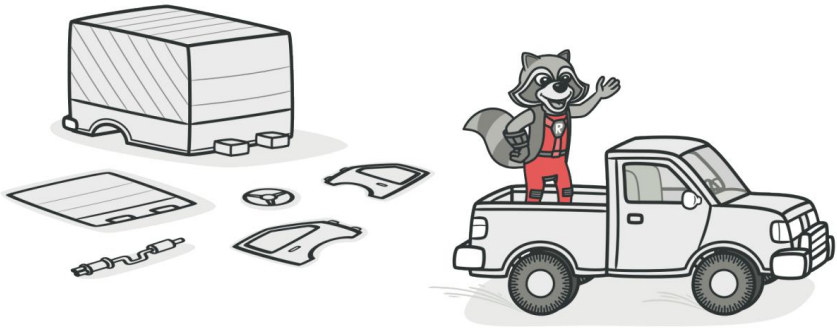
尽量把类的接口放在一个共同的方面
分母：

- 重命名方法以使它们在所有备选方案中都相同
  类。
- 移动方法、添加参数和参数化方法，使方法的签名和实现相同。
- 如果只复制了类的部分功能，请尝试使用提取超类。在这种情况下，现有的类将
  成为子类。
- 在您确定使用并实施了哪种处理方法之后，您可以删除其中一个类。

清偿

- 你摆脱了不必要的重复代码，使结果
  荷兰国际集团的代码不那么笨重。

· 代码变得更具可读性和可理解性（您不再需要猜测创建第二个类执行与第一个类完全相同的功能的原因）。



何时忽略

有时合并类是不可能的，或者是非常困难的，以至于毫无意义。一个例子是当替代类

位于不同的库中，每个库都有自己的版本
班上

改变预防者

这些气味意味着如果您需要在代码中的某个位置更改某些内容,则您也必须在其他位置进行许多更改。结果,程序开发变得更加复杂和昂贵。

§ 发散变化

当您对一个类进行更改时,您会发现自己不得不更改许多不相关的方法。例如,添加新产品类型时,您必须更改查找、显示和订购产品的方法。

§ 霰弹枪手术

进行任何修改都需要您对许多不同的类进行许多小的更改。

§ 并行继承层次结构

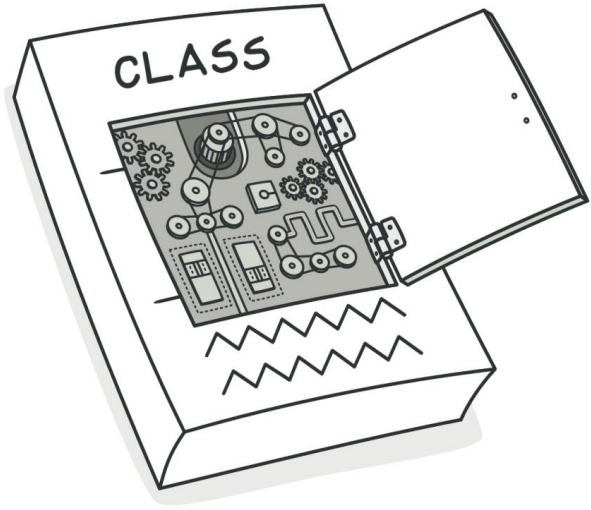
每当您为一个类创建一个子类时,您会发现自己需要为另一个类创建一个子类。

不同的变化

发散变化类似于霰弹枪手术，但实际上是相反的气味。发散变化是当许多
对单个类进行更改。霰弹枪手术
指同时对多个classes进行单个更改时。

体征和症状

你发现自己不得不改变许多不相关的方法
当您更改课程时。例如，当添加
一种新的产品类型，您必须更改查找、显示和订购产品的方法。

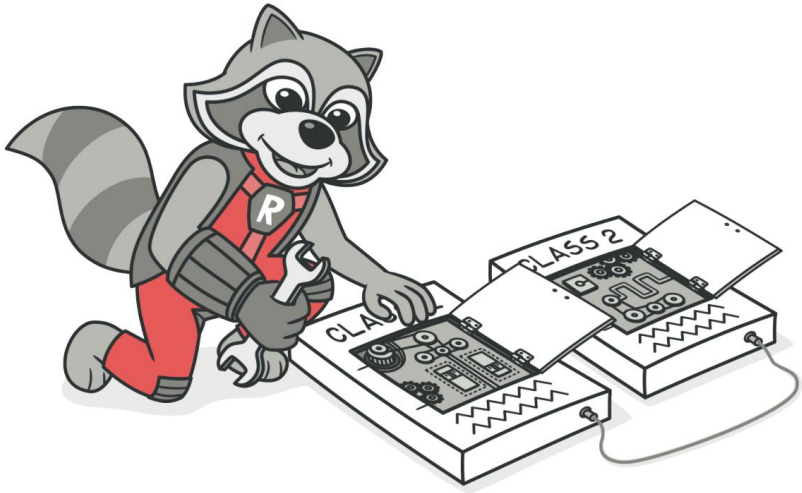


问题的原因

通常这些不同的修改是由于糟糕的程序结构或“copypasta 编程”造成的。

治疗

- 通过Extract Class 拆分类的行为。
- 如果不同的类具有相同的行为，您可能希望通过继承来组合这些类（提取超类和提取子类）。



清偿

- 改进代码组织。
- 减少代码重复。

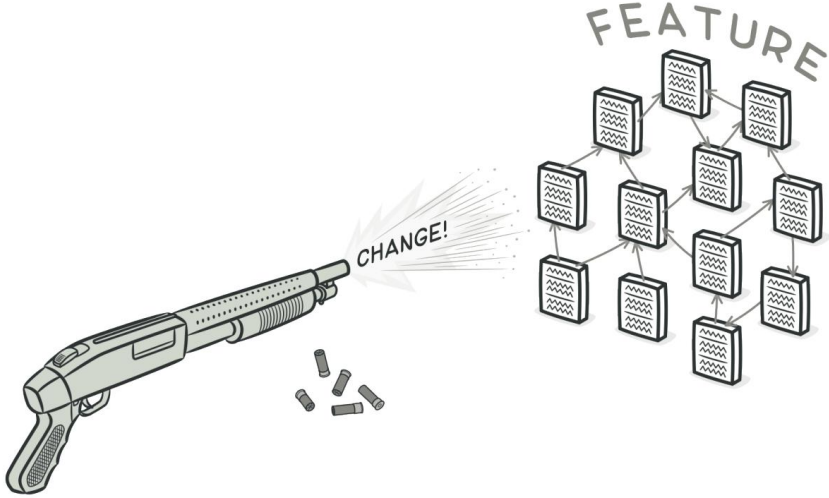
- 简化支持。

霰弹枪手术

霰弹枪手术类似于发散变化，但实际上是相反的气味。发散变化是当许多
对单个类进行更改。霰弹枪手术
指同时对多个classes进行单个更改时。

体征和症状

进行任何修改都需要您做很多小事
更改为许多不同的类。



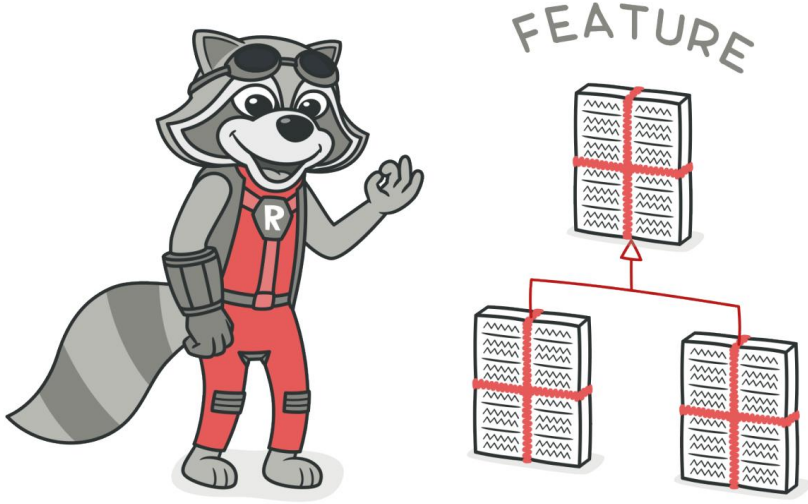
问题的原因

一个单一的职责被分配到大量的类中。这可能发生在过度热心地应用发散变化之后。

治疗

- 使用移动方法和移动字段将现有的类行为移动到一个类中。如果没有适合此的课程，请创建一个新课程。

- 如果将代码移动到同一个类使原始类几乎是空的，请尝试通过以下方式摆脱这些现在冗余的类
在行班。



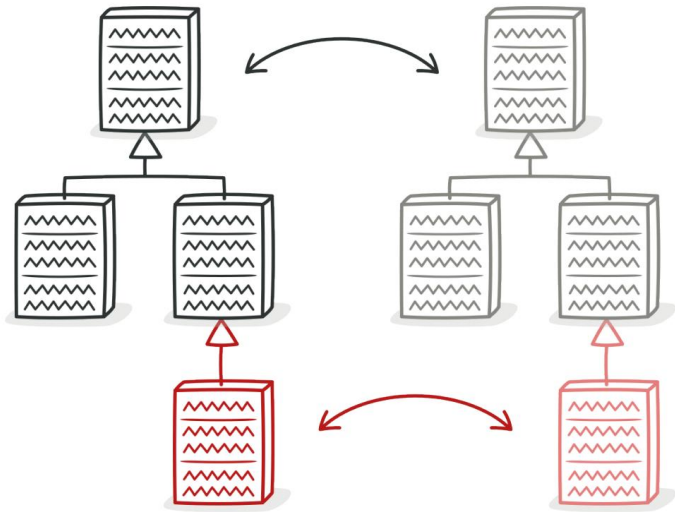
清偿

- 更好的组织。
- 更少的代码重复。
- 更容易维护。

并行继承层次结构

体征和症状

每当您为一个类创建一个子类时,您会发现自己需要为另一个类创建一个子类。



问题的原因

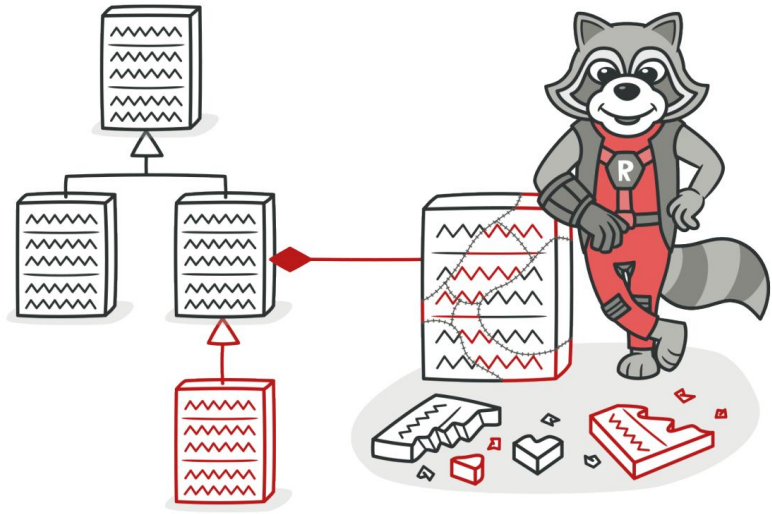
只要等级制度保持较小,一切都很好。但是随着新课程的加入,做出改变变得越来越难。

治疗

您可以分两步对并行类层次结构进行去重。
首先,使一个层次结构的实例引用另一个层次结构的实例。然后,使用Move Method和 Move Field 删除引用类中的层次结构。

清偿

- 减少代码重复。
- 可以改进代码的组织。



何时忽略

有时拥有并行的类层次结构只是避免程序架构出现更大混乱的一种方式。如果您发现对层次结构进行重复数据删除的尝试会产生什

更丑陋的代码,只需退出,还原所有更改并获得
习惯了那个代码。

可有可无

一次性是毫无意义和不需要的东西,如果没有它会使代码更干净、更高效、更容易呃理解。

§ 评论

一个方法充满了解释性注释。

§ 重复代码

两个代码片段看起来几乎相同。

§ 懒人课

理解和维护课程总是需要时间和金钱。所以如果一堂课没有做足够的事情来引起你的注意，
它应该被删除。

§ 数据类

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。

§ 死代码

不再使用变量、参数、字段、方法或类（通常是因为它已过时）。

§ 推测的普遍性

> 有一个未使用的类、方法、字段或参数。