§ 更改对值的引用

问题:您有一个参考对象太小且很少更改,无法证明管理其生命周期是合理的。

解决方案:把它变成一个值对象。

§ 用对象替换数组

问题:您有一个包含各种类型数据的数组。

解决方案:将数组替换为每个元素都有单独的速率字段的对象。

§ 重复观测数据

问题:域数据是否存储在负责

图形用户界面?

解决方案:那么最好将数据分离到单独的类中,确保连接和同步

在域类和 GUI 之间。

§ 将单向关联更改为双向

问题:你有两个类,每个类都需要使用 fea

彼此的关系,但它们之间的关联只是单向的。

解决方案:将缺少的关联添加到需要它的类中。

§ 将双向关联更改为单向

问题:你有一个类之间的双向关联

es,但是其中一个类不使用另一个类的功能。

解决方案:删除未使用的关联。

§ 用符号常数替换幻数

问题:您的代码使用了一个具有特定含义的数字。

解决方案:将此数字替换为具有人类可读名称的常量,以解释数字的含义。

§ 封装字段

问题:你有一个公共领域。

解决方案:将字段设为私有并为其创建访问方法。

§ 封装集合

问题:一个类包含一个集合字段和一个用于处理集合的简单 getter 和 setter。

解决方案:将 getter 返回的值设为只读,并创建用于添加/删除集合元素的方法。

§ 用类替换类型代码

问题:一个类有一个包含类型代码的字段。该类型的值不用于操作符条件,也不影响程序的行为。

解决方案:创建一个新类并使用其对象而不是类型代码值。

§ 用子类替换类型代码

问题:您有一个编码类型,它直接影响每克行为（该字段的值触发条件中的各种代码）。

解决方案:为编码类型的每个值创建子类。
然后将原始类中的相关行为提取到这些子类中。用多态替换控制流代码。

§ 用状态/策略替换类型代码

问题:您有一个影响行为的编码类型,但您不能使用子类来摆脱它。

解决方案:将类型代码替换为状态对象。如果需要用类型代码替换字段值,则“插入”另一个状态对象。

§ 用字段替换子类

问题:您的子类仅在它们的（常量返回)方法上有所不同。

解决方法:用父类中的字段替换方法
并删除子类。

B自封装字段

自封装不同于普通封装字段:执行此处给出的重构技术在私人领域。

问题

您可以直接访问类中的私有字段。

```
1类范围{
2        私人int低,高;
3        布尔包含（int arg） {
4            返回arg >= 低 && arg <= 高;
5        }
6 }
```

解决方案

为该字段创建一个 getter 和 setter,并仅将它们用于访问该字段。

```
类范围{
        私人int低,高;
        布尔包含（int arg） {
            返回arg >= getLow() && arg <= getHigh();
        }
        int getLow() {
            返回低;
        }
        int getHigh() {
            返回高位;
        }
}
```

为什么要重构

有时直接访问类中的私有字段只是
不够灵活。您希望能够启动一个字段
第一次查询时的值或在分配时对字段的新值执行某些操作，或者

也许在子类中以各种方式完成所有这些。

好处

·对字段的间接访问是指通过访问对字段进行操作
方法（getter 和 setter）。这种方法更
比直接访问字段更灵活。

- 首先，当数据在
  字段被设置或接收。延迟初始化和验证

字段值很容易在字段获取器中实现,并且
二传手。

◦ 其次,更重要的是,您可以重新定义 getter 并设置
子类中的术语。

· 您可以选择根本不为字段实现setter。字段值将仅在构造函数中指定,从而使该字段在
整个对象生命周期内都不可更改。

缺点

当使用直接访问字段时,代码看起来更简单、更美观,尽管灵活性降低了。

如何重构

1. 为该字段创建一个 getter（和可选的 setter）。他们应该
要么受保护,要么公开。

2.找到该字段的所有直接调用并将它们替换为
getter 和 setter 调用。

类似的重构

§ 封装字段

隐藏公共字段,提供 getter 和 setter。

帮助其他重构

§ 重复观测数据

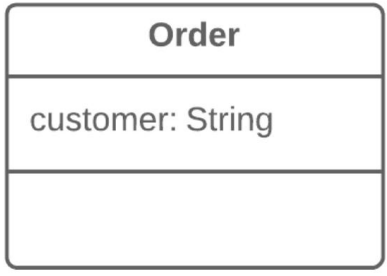
§ 用子类替换类型代码

§ 用状态/策略替换类型代码

B用对象替换数据值

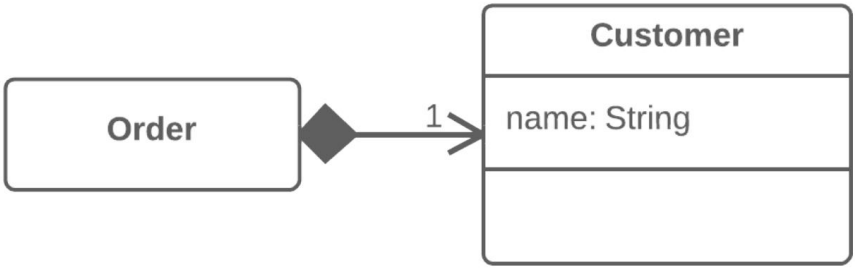
问题

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。



解决方案

创建一个新类，将旧字段及其行为放在类中，并将类的对象存储在原类中。



为什么要重构

这种重构基本上是Extract Class的一个特例。
使它与众不同的是重构的原因。

在 Extract Class 中,我们有一个单独的类负责不同的事情,我们希望拆分它的职责。

通过用对象替换数据值,我们有了一个原始字段（数字、字符串等）,由于程序的增长,它不再那么简单,现在有了相关的数据和行为。一方面,这些领域本身并没有什么可怕的。但是,这个字段和行为系列可以同时存在于多个类中,从而创建重复的代码。

因此,对于这一切,我们创建一个新类并移动
字段及其相关的数据和行为。

好处

提高类内部的相关性。数据和相关行为都在一个类中。

如何重构

在开始重构之前,请查看类中是否有对该字段的直接引用。如果是这样,请使用自封装字段将其隐藏在原始类中。

1. 创建一个新类并将您的字段和相关 getter 复制到其中。

   另外,创建一个接受简单值的构造函数
   领域的。由于每个新字段,此类将没有设置器

   发送到原始类的值将创建一个新的值对象。

2.在原类中,将字段类型更改为新类。

3.在原类的getter中,调用asso的getter

   等价物。

4. 在设置器中,创建一个新的值对象。如果之前已经为该字段设置了初始值,您可能还需要在构造函数中创建一个新对象。

下一步

应用此重构技术后,明智的做法是在包含对象的字段上应用 Change Value to Reference 。这允许存储对与值对应的单个对象的引用,而不是存储数十个对象

一个和相同的值。

当您想让一个对象负责一个现实世界的对象（例如用户、订单、文档等)时,通常需要这种方法。同时,这种方法对日期、金钱、范围等对象没有用处。

类似的重构

§ 提取类

§ 引入参数对象

§ 用对象替换数组

§ 用方法对象替换方法

> 对方法的代码做同样的事情。

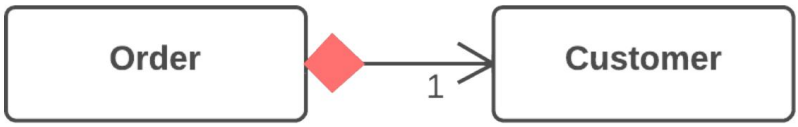
消除异味

§ 重复代码

B将值更改为参考

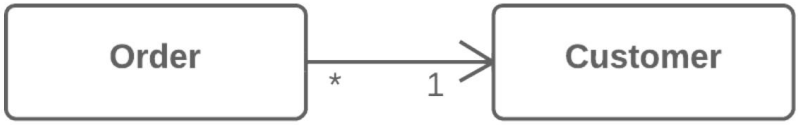
问题

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。



为什么要重构

在许多系统中,对象可以分类为值或
参考。

- 引用:当一个真实世界的对象只对应程序中的一个对象时。参考通常是用户/订单/产品/等。对象。

- 值:一个真实世界的对象对应于程序中的多个对象。这些对象可以是日期、电话号码、地址、颜色等。

参考与价值的选择并不总是明确的。
有时会有一个带有少量不变数据的简单值。然后每次访问对象时都需要添加可变数据并传递这些更改。在这种情况下,有必要将其转换为

参考。

好处

对象包含有关特定实体的所有最新信息。如果对象在程序的某个部分发生更改,则这些更改可以从使用该对象的程序的其他部分访问。

缺点

引用更难实现。

如何重构

1.在要从中生成引用的类上使用 Replace Constructor with Factory Method。

2. 确定哪个对象将负责提供对引用的访问。当您需要一个新对象时，您现在需要从存储对象或静态字典字段中获取它，而不是创建一个新对象。

3. 确定是提前创建引用还是

根据需要动态。如果对象是预先创建的，
确保在使用前加载它们。

4. 更改工厂方法，使其返回引用。如果对象是预先创建的，则决定在请求不存在的对象时如何处理错误。您可能还需要使用重命名方法来通知该方法仅返回现有对象。

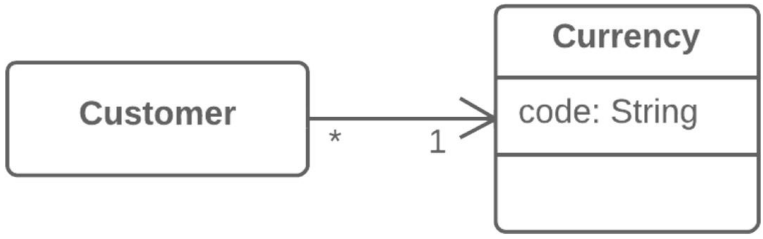
反重构

§ 更改对值的引用

B更改对值的引用

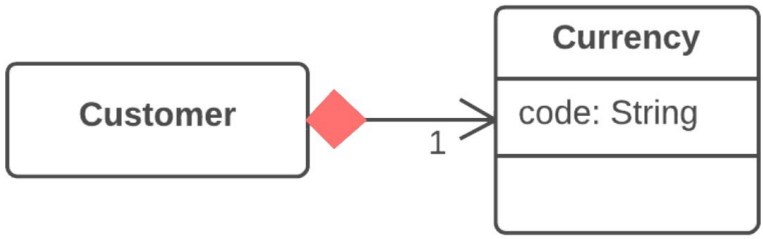
问题

您有一个太小且很少更改的引用对象,无法证明管理其生命周期是合理的。



解决方案

把它变成一个值对象。



为什么要重构

从引用切换到值的灵感可能来自使用引用的不便。引用需要您进行管理：

- 他们总是要求从
  贮存。
- 内存中的引用可能不方便使用。
- 与参考文献相比，使用参考文献特别困难
  值，在分布式和并行系统上。

  如果您宁愿拥有不可更改的对象而不是状态可能在其生命周期内发生变化的对象，则值特别有用。

好处

- 对象的一个重要属性是它们应该是不可更改的。每个返回对象值的查询都应该收到相同的结果。如果这是真的，那么如果有许多对象代表同一事物，则不会出现问题。
- 价值观更容易实现。

缺点

如果一个值是可更改的,请确保如果任何对象发生更改,则表示同一实体的所有其他对象中的值都会更新。这是非常繁重的,因此为此目的创建引用会更容易。

如何重构

1. 使对象不可更改。该对象不应有任何设置器或其他更改其状态和数据的方法（删除设置方法可能在这里有所帮助）。数据应该分配给值对象的字段的唯一位置是

   构造函数。

2. 创建一个比较方法,以便能够比较两个值。

3.检查是否可以删除工厂方法并公开对象构造函数。

反重构

§ 将值更改为参考

B将数组替换为对象

这种重构技术是用对象替换数据值的一种特殊情况。

问题

您有一个包含各种类型数据的数组。

```
1字符串 []行 =新字符串 [2]； 2行 [0] = 利物浦 ；
3行[1] = “15”；
```

解决方案

将数组替换为每个元素都具有单独字段的对象。

```
1性能行 = new Performance();
2 row.setName( 利物浦 );
3 row.setWins( 15 );
```

为什么要重构

数组是存储单一类型数据和集合的绝佳工具。但是,如果您使用像邮政信箱这样的数组,将用户名存储在框 1 中,将用户的地址存储在框 14 中,那么您总有一天会非常不高兴。当有人将某些东西放入错误的“盒子”时,这种方法会导致灾难性的失败,并且还需要您花时间弄清楚

哪些数据存储在哪里。

好处

- 在生成的类中,您可以放置以前存储在主类或其他地方的所有相关行为。

- 类的字段比元素更容易记录

  数组的元素。

如何重构

1. 创建包含数组数据的新类。

   将数组本身作为公共字段放在类中。

2、在原类中创建一个存放该类对象的字段。不要忘记在您启动数据数组的位置创建对象本身。

3. 在新类中,为每个数组元素一个一个地创建访问方法。给他们一个不言自明的名字

表明他们在做什么。同时,将主代码中每一个数组元素的使用替换为对应的访问

方法。

4. 为所有元素创建访问方法后,

将数组设为私有。

5. 对于数组的每个元素,在类中创建一个私有字段,然后更改访问方法,以便他们使用该字段

而不是数组。

6. 当所有数据都被移动后,删除数组。

类似的重构

§ 用对象替换数据值

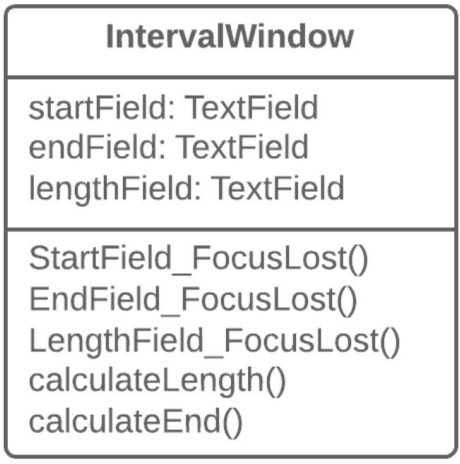
消除异味

§ 原始痴迷

B重复观察数据

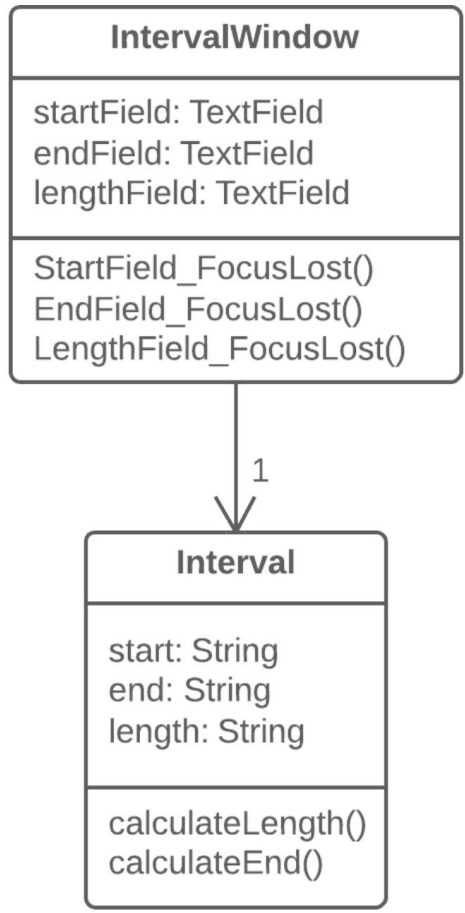
问题

域数据是否存储在负责 GUI 的类中?



解决方案

然后最好将数据分离到单独的类中,确保域类和 GUI 之间的连接和同步。



为什么要重构

您希望为相同的数据拥有多个界面视图（例如,您同时拥有一个桌面应用程序和一个移动应用程序）。

如果您无法将 GUI 与域分开,您将很难避免代码重复和大量

的错误。

好处

- 您在业务逻辑类和
  演示类（参见单一职责原则），
  这使您的程序更具可读性和可理解性
  有能力的。

- 如果需要添加新的界面视图，创建新的表示类；您无需接触业务代码
  逻辑（参见开放/封闭原则）。

- 现在不同的人可以处理业务逻辑和
  用户界面。

什么时候不使用

- 这种重构技术，其经典形式是使用 Observer模板执行的，不适用于 Web
  应用程序，其中所有类都在查询之间重新创建
  网络服务器。

- 都一样，提取业务逻辑的一般原则
  对于 Web 应用程序，也可以将它们分成单独的类。
  但这将根据您的系统设计方式使用不同的重构技术来实现。

如何重构

1.隐藏对GUI类中域数据的直接访问。为此,最好使用自封装字段。所以你创建了吸气剂

和此数据的设置器。

2. 在GUI 类事件的处理程序中,使用 setter 设置新的字段值。这将让您将这些值传递给关联的域对象。

3. 创建域类并从GUI复制必要的字段

给它上课。为所有这些字段创建 getter 和 seter。

4. 为这两个类创建一个观察者模式:

- 在域类中,创建一个数组来存储观察者对象(GUI 对象),以及注册、删除和通知它们的方法。
- 在GUI 类中,创建一个用于存储对域类的引用的字段以及update()方法,该方法将对对象的变化做出反应并更新GUI 类中字段的值。注意值更新应该直接在方法中建立,以避免

  递归。
- 在GUI 类构造器中,创建域类实例并将其保存在您创建的字段中。将GUI 对象注册为域对象中的观察者。

- 在域类字段的设置器中,调用
  通知观察者（换句话说， GUI类中的更新方法）,以便将新值传递给GUI。

- 更改GUI 类字段的设置器,以便它们直接在域对象中设置新值。注意确保值不是通
  过域类设置器设置的

  否则将导致无限递归。

实现设计模式

§ 观察员

消除异味

§ 大班

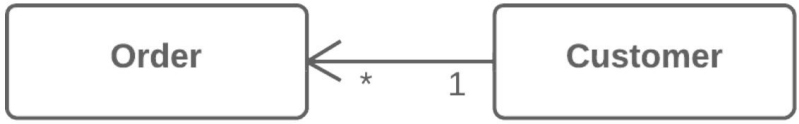
B改变_

单向
协会
双向

问题

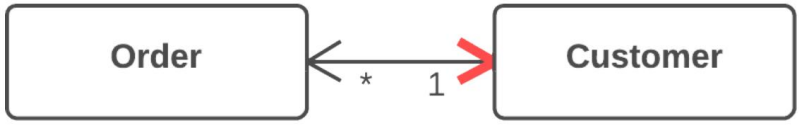
您有两个类,每个类都需要使用

其他,但它们之间的关联只是单向的。



解决方案

将缺少的关联添加到需要它的类。



为什么要重构

最初,这些类具有单向关联。但随着时间的推移,客户端代码需要访问

协会。

好处

如果一个类需要反向关联,你可以简单地计算它。但如果这些计算很复杂,最好保留
反向关联。

缺点

- 双向关联更难实现,并且
  保持比单向的。

- 双向关联使类相互依赖。通过单向关联,其中一个可以独立于另一个使用。

如何重构

1.添加一个字段用于持有反向关联。

2. 决定哪一类将是“主要的”。这个类将包含
   在添加或更改元素时创建或更新关联的方法,在其类中建立关联

并调用实用方法在关联对象中建立关联。

3. 创建用于在“非支配”类中建立关联的实用方法。该方法应使用参数中给出的内容来完成该字段。给该方法起一个明显的名称，以便以后不会将其用于任何其他目的。

4. 如果控制单向关联的旧方法位于“主导”类中，则通过从关联对象调用实用方法来补充它们。

5. 如果控制关联的旧方法在“非支配”类中，则在“支配”类中创建方法，调用它们，并将执行委托给它们。

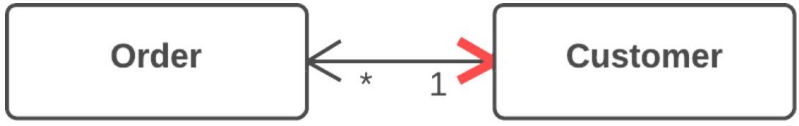
反重构

§ 将双向关联更改为单向

B改变双向协会单向

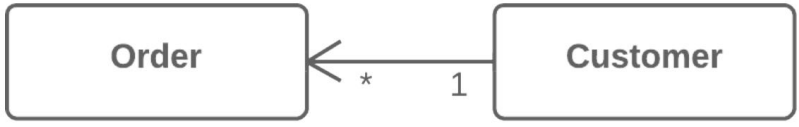
问题

你有一个类之间的双向关联,但是一个的类不使用其他的功能。



解决方案

删除未使用的关联。



为什么要重构

双向关联通常比维护更难
一个单向的,需要额外的代码才能正确
创建和删除相关对象。这使得每克更复杂。

此外,不正确实施的双向关联可能会导致垃圾收集问题（反过来导致
未使用的对象导致内存膨胀）。

示例:垃圾收集器从内存中删除不再被其他对象引用的对象。比方说

对象对User - Order已创建、使用,然后
弃。但是这些对象不会从内存中清除
因为它们仍然相互引用。也就是说,这个问题是
由于编程的进步而变得不那么重要
语言,现在可以自动识别未使用的对象
引用并将它们从内存中删除。

还有类之间相互依赖的问题。在双向关联中,两个类必须知道

关于彼此,意味着它们不能单独使用。如果存在许多这些关联,则不同的部分
的程序变得过于依赖彼此和任何
一个组件的变化可能会影响其他组件。

好处

- 简化不需要关系的类。较少的
  代码等于更少的代码维护。

- 减少类之间的依赖。独立类更容易维护,因为对类的任何更改只影响
  那堂课。

如何重构

1. 确保您的课程满足以下条件之一:
   - 不使用关联。
   - 还有另一种获取关联对象的方法,例如通过数据库查询。
   - 关联对象可以作为参数传递给
     使用它的方法。
2. 根据您的情况,使用包含与另一个对象关联的字段应替换为参数或方法调用,以便以不同
   的方式获取对象。
3. 删除将关联对象分配给字段的代码。
4. 删除现在未使用的字段。

反重构

§ 将单向关联更改为双向

消除异味

§ 不恰当的亲密关系

B替换魔法编号与符号常数

问题

您的代码使用了一个具有特定含义的数字。

```
1双势能（双倍质量,双倍高度）{
2        返回质量 * 高度 * 9.81;
3 }
```

解决方案

将此数字替换为具有人类可读名称的常量,以解释数字的含义。

```
1静态最终双GRAVITATIONAL_CONSTANT = 9.81;
2
3双势能（双倍质量,双倍高度）{
4        返回质量 * 高度 * GRAVITATIONAL_CONSTANT;
5 }
```

为什么要重构

幻数是在源中遇到但没有明显含义的数值。这种“反模式”使得理解程序和重构代码变得更加困难。

然而，当你需要改变这个神奇的数字时，就会出现更多的困难。查找和替换对此不起作用：相同的数字可能在不同的地方用于不同的目的，这意味着您必须验证使用它的每一行代码

数字。

好处

- 符号常量可以作为
  其价值的意义。

- 更改常量的值比在整个代码库中搜索这个数字要容易得多，而且不会有意外更改其他地方用于不同目的的相同数字的风险。

- 减少代码中数字或字符串的重复使用。当值复杂且长时（例如3.14159或0xCAFEBABE），这一点尤其重要。

很高兴知道

并非所有数字都是神奇的。

如果数字的用途很明显,则无需替换它。一个经典的例子是:

```
1 for (i = 0; i < count; i++) { ... }
```

备择方案

1. 有时可以用方法调用来代替幻数。例如,如果您有一个表示集合中元素数量的幻数,则不需要使用它来检查集合的最后一个元素。相反,使用标准方法来获取集合长度。

2. 幻数有时用作类型代码。假设您有两种类型的用户,并且您使用类中的数字字段来指定哪个是哪个:管理员是1和普通

用户是2。

在这种情况下,您应该使用其中一种重构方法来避免类型代码:

- 将类型代码替换为类
- 用子类替换类型代码

- 将类型代码替换为状态/策略

如何重构

1. 声明一个常数并赋值幻数的值
   给它。

2. 找出所有提到的幻数。

3. 对于你找到的每个数字，仔细检查这个特殊情况下的幻数是否对应于常数的目的。如果是，请将数字替换为您的常数。这是一个重要的步骤，因为相同的数字可能意味着完全不同的东西（并根据情况用不同的常数代替）。

B封装字段

问题

你有一个公共领域。

```
类人{
    公共字符串名称；
}
```

解决方案

将字段设为私有并为其创建访问方法。

```
类人{
    私有字符串名称；

    公共字符串getName() {
        返回名称；
    }
    公共无效集合名称（字符串参数） {
        名称 = 参数；
    }
}
```

为什么要重构

面向对象编程的支柱之一是封装,即隐藏对象数据的能力。否则,所有对象都将是公共的,其他对象可以获取和修改您的对象的数据,而无需任何制衡!数据与与该数据相关的行为分离,程序部分的模块化受到损害,维护变得复杂。

好处

- 如果组件的数据和行为密切相关,并且在代码中位于同一位置,那么您维护和开发该组件会容易得多。

- 您还可以执行与访问对象字段相关的复杂操作。

什么时候不使用

在某些情况下,出于性能考虑,封装是不明智的。这些情况很少见,但当它们发生时,这种情况非常重要。

假设您有一个图形编辑器,其中包含拥有 x 和 y 坐标的对象。这些领域在未来不太可能发生变化。更重要的是,该程序涉及存在这些字段的许多不同对象。

因此直接访问坐标字段可以节省大量
否则会被调用 access 占用的 CPU 周期
方法。

作为这种不寻常情况的一个例子， Java 中有 Point 类。此类的所有字段都是公共的。

如何重构

1. 为字段创建一个 getter 和 setter。

2. 查找该字段的所有调用。将字段值的接收替换为 getter，并替换新字段值的设置

与二传手。

3.所有字段调用都被替换后，使字段
私人的。

下一步

封装字段只是将数据和涉及此数据的行为更紧密地结合在一起的第一步。在为访问字段创建简单方法后，您应该重新检查调用这些方法的位置。这些区域中的代码很可能在

访问方法。

类似的重构

§ 自封装字段

为字段创建 getter 和 setter,而不是在类的方法中直接访问。

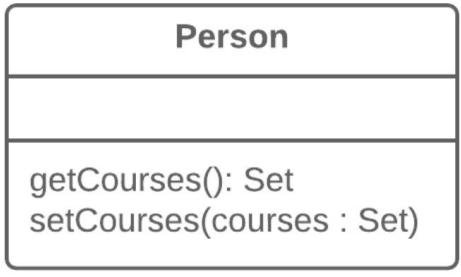
消除异味

§ 数据类

B封装收藏

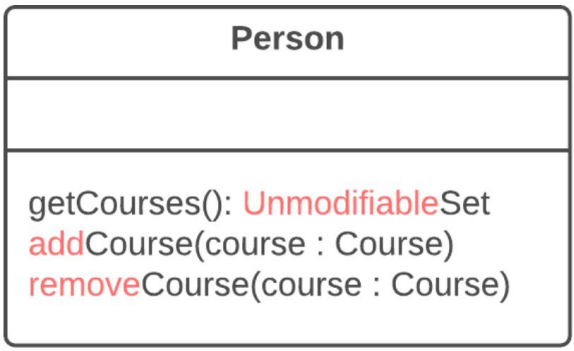
问题

一个类包含一个集合字段和一个用于处理集合的简单 getter 和 setter。



解决方案

将 getter 返回的值设为只读,并创建用于添加/删除集合元素的方法。



为什么要重构

一个类包含一个包含对象集合的字段。
该集合可以是数组、列表、集合或向量。已经创建了一个普通的 getter 和 setter 来处理该集合。

但是集合应该由与其他数据类型使用的协议有点不同的协议使用。 getter 方法不应该返回集合对象本身，因为这会让客户端在所有者类不知道的情况下更改集合内容。此外，这会向客户端显示太多对象数据的内部结构。

获取集合元素的方法应返回一个不允许更改集合或泄露有关其结构的过多数据的值。

此外，不应该有为集合分配值的方法。相反，应该有添加和删除元素的操作。感谢楼主

对象获得对集合的添加和删除的控制权
元素。

这样的协议适当地封装了一个集合,最终降低了所有者之间的关联
程度
类和客户端代码。

好处

- 集合字段被封装在一个类中。当。。。的时候
  调用 getter 时,它返回集合的副本,这可以防止在不知道包含该类的类的情况
  下意外更改或覆盖集合元素

  收藏。

- 如果集合元素包含在原始类型中,
  例如数组,您可以创建更方便的方法
  与收藏合作。

- 如果集合元素包含在非原始
  容器(标准集合类),通过封装集合可以限制对集合的不需要的标
  准方法的访问(例如通过限制添加新的

  元素)。

如何重构

1. 创建添加和删除集合元素的方法。
   它们必须在其参数中接受集合元素。

2. 为该字段分配一个空集合作为初始值,如果
   这不是在类构造函数中完成的。

3. 找到集合字段设置器的调用。更改设置器,使其使用添加和删除元素的操作,或使这些操
   作调用客户端代码。

   请注意,setter 只能用于将所有集合元素替换为其他元素。因此,建议将 setter 名称
   (重命名方法)更改为替换.

4. 查找集合 getter 的所有调用,之后集合发生变化。更改代码,使其使用新方法从集合中添
   加和删除元素。

5. 更改 getter 使其返回只读表示
   的集合。

6. 检查使用该集合的客户端代码
   在集合类本身内部看起来会更好。

消除异味

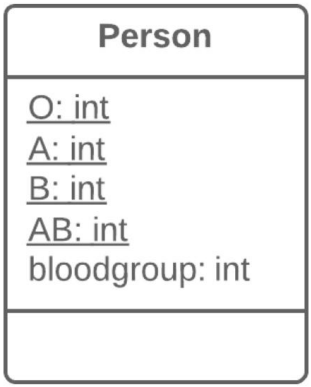
§ 数据类

B用类替换类型代码

什么是类型代码?当您拥有一组数字或字符串,而不是单独的数据类型时,就会出现类型代码,这些数字或字符串构成了某个实体的允许值列表。
通常这些特定的数字和字符串通过常量被赋予易于理解的名称,这就是原因
为什么会遇到如此多的类型代码。

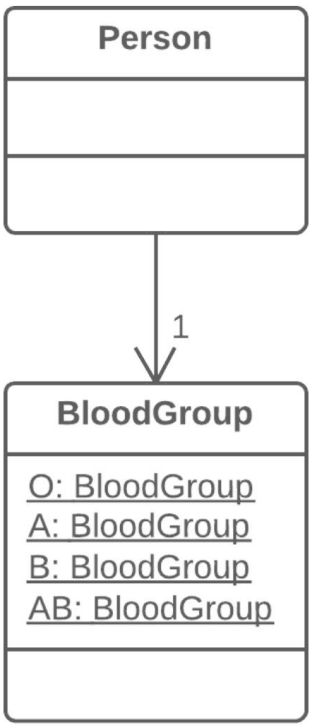
问题

一个类有一个包含类型代码的字段。该类型的值不用于操作符条件,也不影响程序的行为。



解决方案

创建一个新类并使用它的对象而不是类型代码
价值观。



为什么要重构

类型代码最常见的原因之一是使用
数据库,当数据库有一些复杂的字段时
概念用数字或字符串编码。

例如,您有带有字段的用户类
其中包含有关 access priv user_role的信息,

每个用户的文件,无论是管理员、编辑还是普通用户
用户。因此,在这种情况下,此信息在字段中编码为
`一种`, `乙`, 和U分别。

这种方法的缺点是什么?字段设置器
经常不检查发送的是哪个值,这可能会导致大
有人向其发送意外或错误值时的问题
这些领域。

此外,这些字段无法进行类型验证。它是
可以向他们发送任何数字或字符串,这不会是
由您的 IDE 检查类型,甚至允许您的程序运行
（稍后崩溃）。

好处

- 我们要转换原始值的集合　这就是编码的内容
  类型是 - 进入具有所有好处的成熟类
  面向对象编程必须提供。

- 通过用类替换类型代码,我们允许类型提示
  在编程语言级别传递给方法和字段的值。

例如,虽然编译器以前在将值传递给方法时没有看到数字常量和任意数字之间的差
异,但现在当数据

不符合指定的类型类已通过,警告您
IDE 内部的错误。

· 因此,我们可以将代码移动到该类型的类中。如果您需要在整个程序中对类型值执行复杂的操作,现在这段代码可以“活”在一个或多个类型类中。

什么时候不使用

如果编码类型的值在控制流结构（ if等)中使用并控制类行为,则应使用以下两种重构类型代码技术之一:

· 用子类替换类型代码

· 将类型代码替换为状态/策略

如何重构

1. 创建一个新类,并给它一个与编码类型的用途相对应的新名称。这里我们称它为类型类。

2. 将包含类型代码的字段复制到类型类中,并将其设为私有。然后为该字段创建一个吸气剂。只会从构造函数中为此字段设置一个值。

3. 对于编码类型的每个值,在类型类中创建一个静态方法。它将创建一个与编码类型的值相对应的新类型类对象。

4.在原类中,将编码字段的类型替换为类型类。在构造函数和字段设置器中创建此类型的新对象。更改字段 getter,使其调用类型类getter。

5.将任何提及编码类型的值替换为相关类型类静态方法的调用。

6. 从原始类中删除编码类型常量。

类似的重构

§ 用子类替换类型代码

§ 用状态/策略替换类型代码

消除异味

§ 原始痴迷

B用子类替换类型代码

什么是类型代码?当您拥有一组数字或字符串,而不是单独的数据类型时,就会出现类型代码,这些数字或字符串构成了某个实体的允许值列表。
通常这些特定的数字和字符串通过常量被赋予易于理解的名称,这就是原因
为什么会遇到如此多的类型代码。

问题

你有一个直接影响程序行为的编码类型（该字段的值触发条件中的各种代码）。