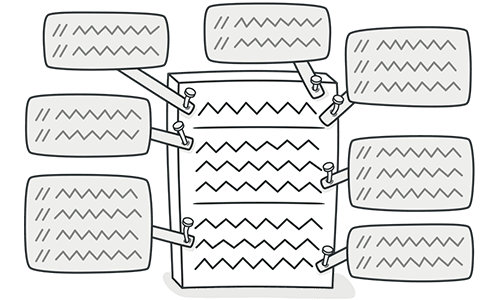
**[Comments](https://refactoring.guru/smells/comments)**

问题：方法中充满了解释性的注释



问题原因：作者意识到他写的代码理解的不太清晰，于是创建了注释用来帮助理解。在这种情况下，注释就掩盖了本身代码的理解问题。

最好的注释应该是**方法或者类的名称**

解决方法：

1. 如果注释旨在解释复杂表达式，则应使用 Extract Variable 将表达式拆分为可理解的子表达式。
2. 如果注释解释了一段代码，则可以通过 extractmethod 将此部分转换为单独的方法。新方法的名称从注释中提取。
3. 如果方法已经被提取，但仍然需要注释来解释方法的作用，那么使用Rename 给方法一个可以理解的名称。
4. 如果您需要断言系统工作所必需的状态规则，请使用 introduction Assertion。

重构优点：

代码变得直观和明显

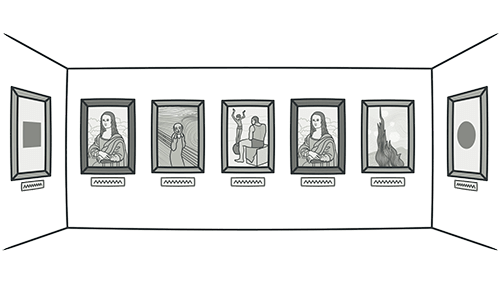


非必要重构：

1. 在解释为什么以特定方式实现某些内容时
2. 在解释复杂算法时

**[Duplicate Code](https://refactoring.guru/smells/duplicate-code)**

问题：两个代码块看起来几乎完全相同

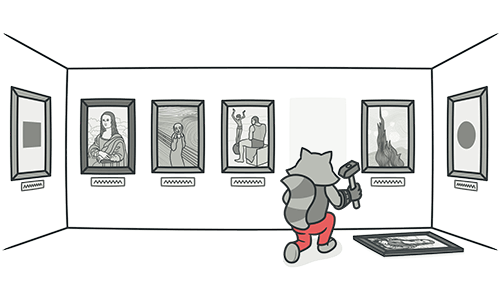


问题原因：

1. 多个coder隔离从事多个业务时，可能不会意识到已经有相似功能代码块被实现
2. 看似相同，实际在特定地方不同的代码块，并且几乎无法更改
3. 因为需求时间紧急，CV操作可以及时完成需求而忽略了代码的规范

解决方法：

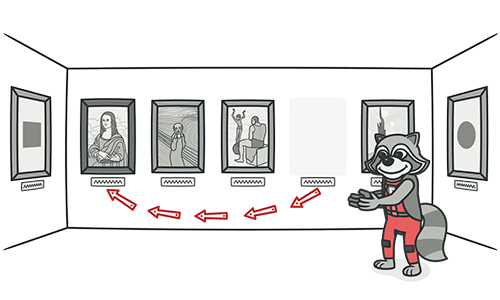
1. 如果在同一个类中的两个或多个方法中找到相同的代码，请在两个位置使用 Extract Method 并对新方法进行重新调用



1. 如果在同一级别的两个子类中发现了重复代码：
   1. 对这两个类都使用 Extract Method，把相同的属性提取至父类。
   2. 如果重复的代码在构造函数中，请在父类中创建构造函数，在子类中调用父类构造函数
   3. 如果重复代码相似但不完全相同，将相同的算法部分和步骤移到父类，在子类中保留不同的步骤实现
   4. 如果两种方法做相同的事情，但使用不同的算法，选择最好的算法和应用替代算法。
2. 如果在两个不同的类中发现重复的代码：
   1. 如果这些类不是层次结构的一部分，那么使用 Extract Superclass 来创建一个父类来维护以前的所有功能。
   2. 如果很难或不可能创建父类，那么在一个类中使用 Extract Class，在另一个类中使用新组件。
3. 如果存在大量条件表达式并执行相同的代码(只是条件不同) ，则使用合并条件表达式将这些运算符合并为单个条件，并使用 Extract Method 将条件放置在一个具有容易理解的名称的单独方法中。
4. 如果在条件表达式的所有分支中执行相同的代码，请使用合并重复条件片段将相同的代码放在条件树之外。

重构优点：

1. 简化代码结构，去除重复代码，提高代码复用性
2. 代码支持成本更低

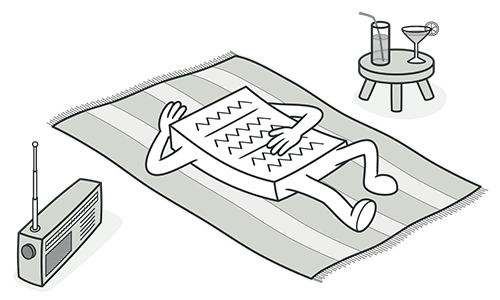


非必要重构：

在极少数情况下，合并两个相同代码会使代码变得不太直观和明显，这个时候可以选择不合并重复代码

**[Lazy Class](https://refactoring.guru/smells/lazy-class)**

问题：对于一个类的维护和理解都是需要花费成本的，如果一个类的功能已经不值得成本消耗，那这个类就不应该存在

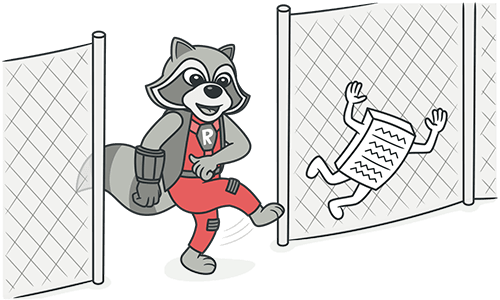


问题原因：

1. 一个类在设计之初可能是具有很多功能的，但是在不断的重构中功能被削减和分离
2. 可能是为了未来计划中的某些功能设计去配合功能完成的类，但是实际这个这些功能实际未被采用或者抛弃。

解决方法：

1. 将责任转移至其他类，删除这个类
2. 合并子类和父类为一个类



重构优点：

1.减少了代码大小

2.易于维护

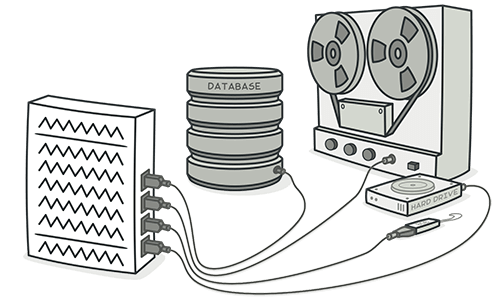
非必要重构：

创建惰性类是为了描绘未来开发的意图，在这种情况下，请尝试在代码的清晰度和简单性之间保持平衡

**[Data Class](https://refactoring.guru/smells/data-class)**

问题：

数据类只包含最基础的数据，为其他数据类提供作用，而自己没有对应的操作逻辑

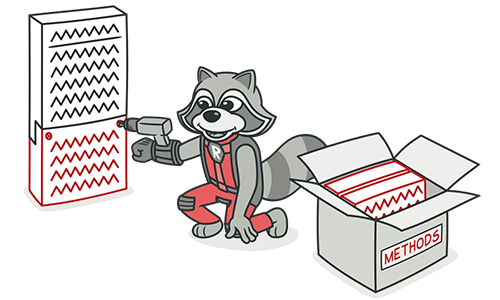


问题原因：

一个对象应该包括状态和行为，而数据类只有状态，没有行为

重构方法：

1. 如果类包含公共字段，请让他仅支持get/set方法访问
2. 对存储在集合中的数据使用封装集合
3. 你可能在代码里发现比数据类中更好的功能，这样的话请把该功能提取至数据类中
4. 替换了类里的老旧方法以后，可以通过删除方法和隐藏方法去除老旧方法的作用



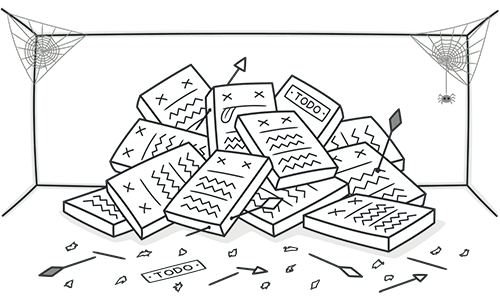
重构优点：

1. 对特定数据的操作集中收集在一个位置有利于提高对代码的理解和组织
2. 帮助发现代码中对于数据类的重复操作代码

**[Dead Code](https://refactoring.guru/smells/dead-code)**

问题：

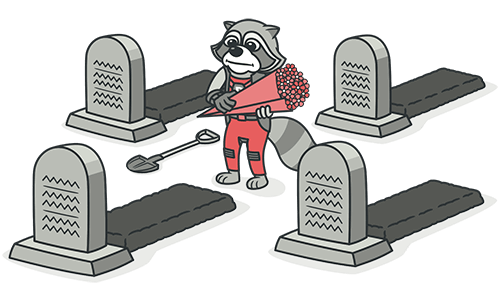
不再使用的变量，参数，字段，方法，类。



问题原因：

1. 当软件的要求发生了变化或者进行了更正时，没有人有时间清理旧代码
2. 如果有一部分的代码无法访问，那么也可以找到这类代码

重构方法：

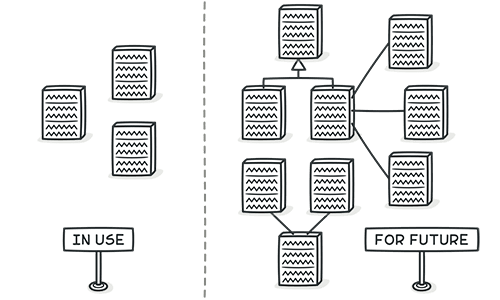
1. 删除未使用的代码和不需要的文件
2. 对于不需要的类，如果使用子类和父类，就应该折叠继承结构
3. 删除不需要的参数

重构优点:

1. 减少代码体积
2. 更容易维护

**[Speculative Generality](https://refactoring.guru/smells/speculative-generality)**

问题：有一个未使用的类，方法，字段和参数



问题原因：

有时，代码是"以防万一"创建的，以支持从未实现的预期未来功能。因此，代码变得难以理解和支持。

重构方法：

1. 删除未使用的抽象类，简化继承结构
2. 合并类消除不必要的功能，委派到另一个类
3. 合并方法来删除不使用的方法
4. 删除参数
5. 删除不使用的字段



重构优点：

1. 缩小代码体积
2. 代码更易于维护

何时不重构

1. 框架中存在不使用的功能是合理的，只要框架的目标用户会使用就可以
2. 删除元素之前，确保未在单元测试中使用