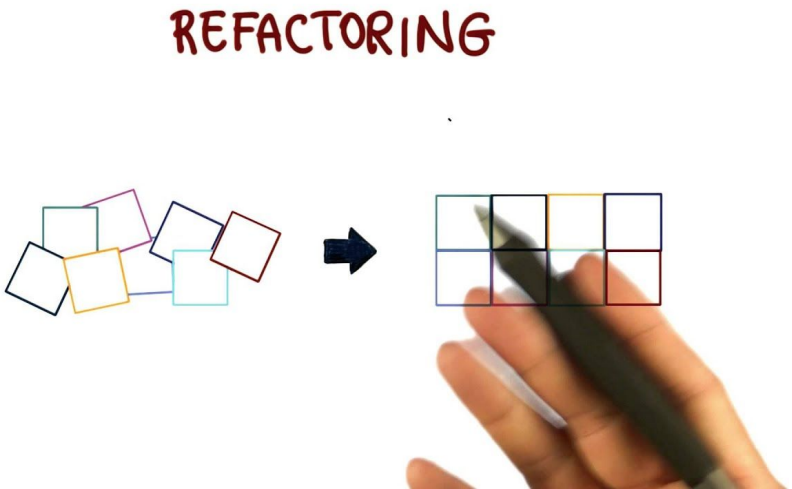
[7507 / 9502]

Algoritmos y Programación III

Refactorización

Qué es refactorizar?

Refactorizar es el proceso de cambiar un sistema de software mejorando su estructura interna sin alterar el comportamiento externo del código.



Los dos sombreros - Kent Beck

Cuando se usa refactorizaciones para desarrollar software, el tiempo se divide en dos:

- Agregar funcionalidades
- Refactorizar

¿Por qué deberías refactorizar?

- Mejorar el diseño del software
- Hace que el software sea más fácil de entender
- Ayuda a encontrar errores
- Ayuda a programar más rápido

¿Cuándo deberías refactorizar?

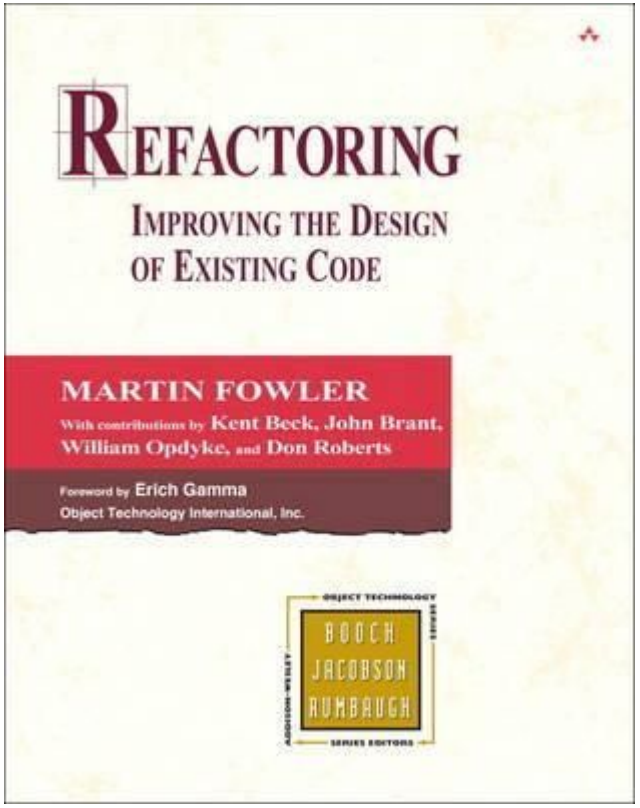
- La regla de tres
- Cuándo agregas una función
- Cuándo necesitas arreglar un error
- Mientras revisas el código (code reviews)

Code Smells

.UBAfiuba
FACULTAD DE INGENIERÍA

75.07
95.02

Kent Beck



Martin Fowler

.UBAfiuba
FACULTAD DE INGENIERÍA

75.07
95.02

Code Smells clásicos

- Alternative Classes with Different Interfaces
- Comments
- Data Class
- Data Clumps
- Divergent Change
- Duplicated Code
- Feature Envy
- Inappropriate Intimacy
- Incomplete Library Class
- Large Class
- Lazy Class
- Long Method
- Long Parameter List
- Message Chains
- Middle Man
- Parallel Inheritance Hierarchies
- Primitive Obsession
- Refused Bequest
- Shotgun Surgery
- Speculative Generality
- Switch Statements
- Temporary Field

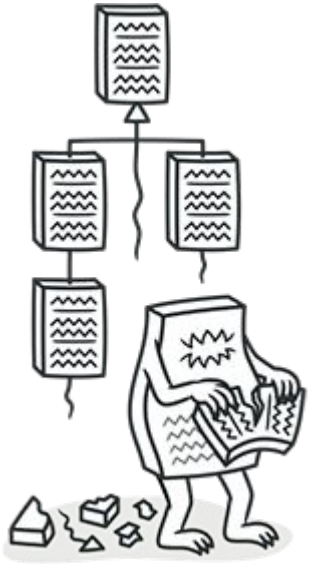
75.07
95.02

Bloaters

- Long Method
- Large Classes
- Data Clumps
- Long Parameter List

Object-Orientation (Tools) Abusers

- Switch stament
- Refused Bequest
- Alternative Classes with Different Interfaces
- Temporary Field



Change Preventers

- Divergent Change
- Shotgun Surgery
- Parallel Inheritance Hierarchies

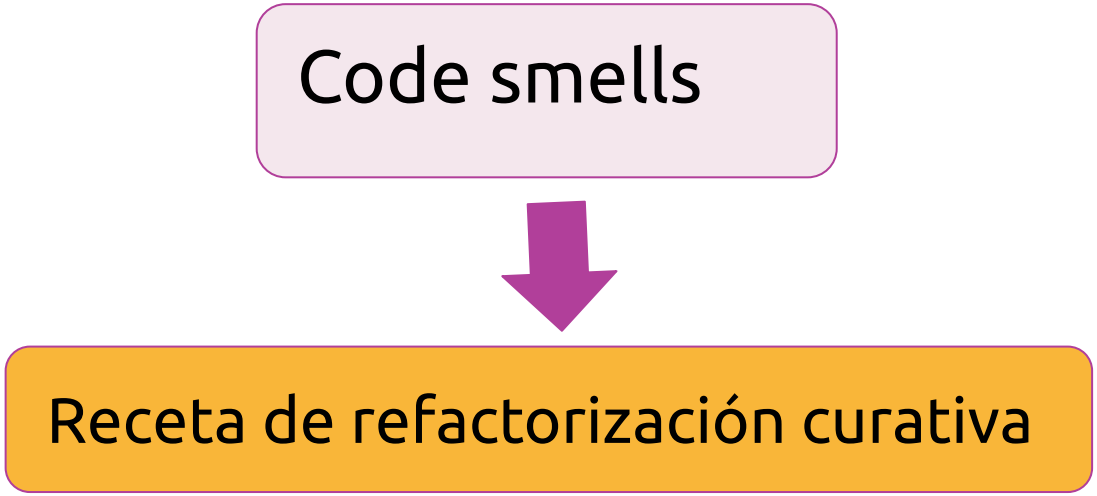
Dispensables (prescindibles)

- Lazy Class
- Speculative Generality
- Data Class
- Duplicate Code

Couplers

- Feature Envy
- Inappropriate Intimacy
- Message Chain
- Middle Man

Receta de refactorización



No reinventar la rueda

https://www.industriallogic.com/img/blog/2005/09/smellstorefactorings.pdf

Smells to Refactorings Quick Reference Guide

Smell	Refactoring
Alternative Classes with Different Interfaces: occurs when the interfaces of two classes are different and yet the classes are quite similar. If you can find the similarities between the two classes, you can often refactor the classes to make them share a common interface [F 85, K 43]	Unify Interfaces with Adapter [K 247]
	Rename Method [F 273]
	Move Method [F 142]
Combinatorial Explosion: A subtle form of duplication, this smell exists when numerous pieces of code do the same thing using different combinations of data or behavior. [K 45]	Replace Implicit Language with Interpreter [K 269]
Comments (a.k.a. Deodorant): When you feel like writing a comment, first try "to refactor so that the comment becomes superfluous" [F 87]	Rename Method [F 273]
	Extract Method [F 110]
	Introduce Assertion [F 267]
Conditional Complexity: Conditional logic is innocent in its infancy, when it's simple to understand and contained within a few lines of code. Unfortunately, it rarely ages well. You implement several new features and suddenly your conditional logic becomes complicated and expansive. [K 41]	Introduce Null Object [F 260, K 301]
	Move Embellishment to Decorator [K 144]
	Replace Conditional Logic with Strategy [K 129]
	Replace State-Altering Conditionals with State [K 166]
Data Class: Classes that have fields, getting and setting methods for the fields, and nothing else. Such classes are dumb data holders and are almost certainly being manipulated in far too much detail by other classes. [F 86]	Move Method [F 142]
	Encapsulate Field [F 206]
	Encapsulate Collection [F 208]
Data Clumps: Bunches of data that that hang around together really ought to be made	Extract Class [F 149]

.UBAfiuba
FACULTAD DE INGENIERIA

75.07
95.02

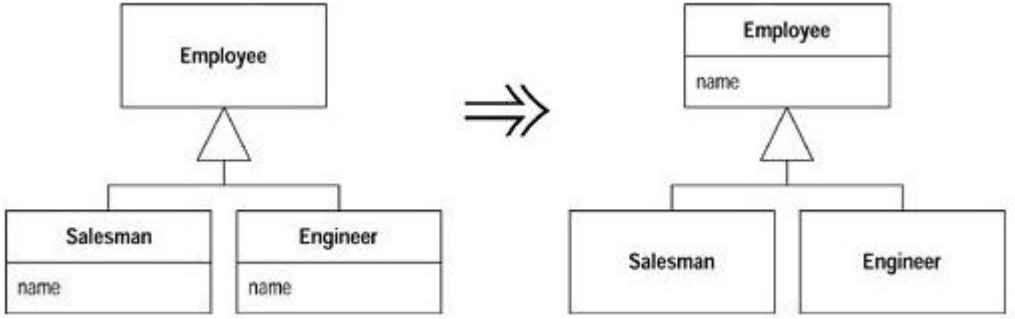
Consolidar expresión condicionales

```
double disabilityAmount() {
  if (seniority < 2) {
    return 0;
  }
  if (monthsDisabled > 12) {
    return 0;
  }
  if (isPartTime) {
    return 0;
  }
  // Compute the disability amount.
  // ...
}
```

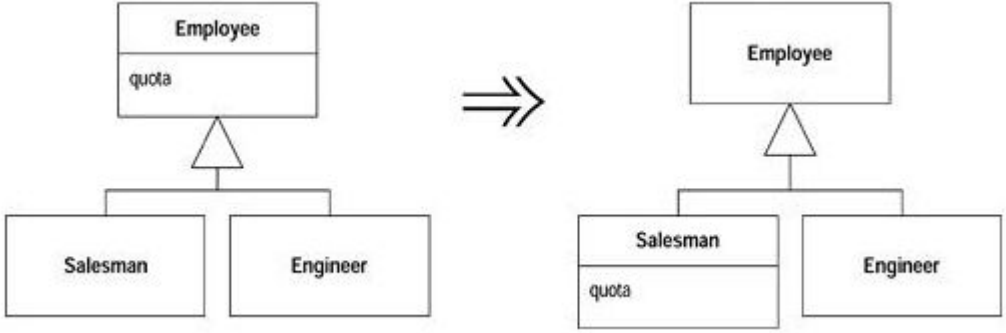
```
double disabilityAmount() {
  if (isNotEligibleForDisability()) {
    return 0;
  }
  // Compute the disability amount.
  // ...
}
```

Lidiar con la generalización

Pull up

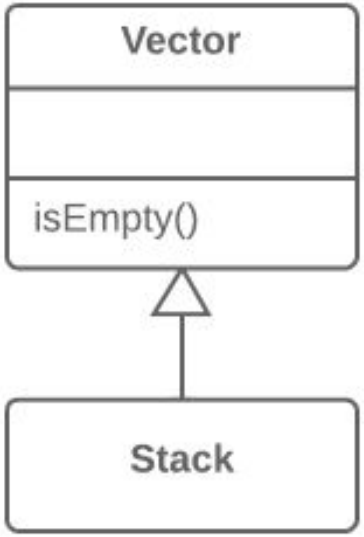


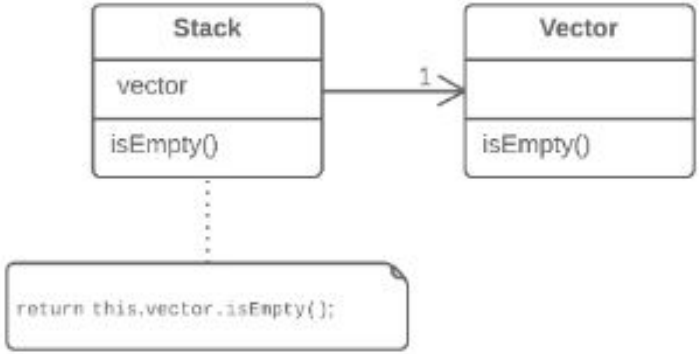
Push down



75.07
95.02

Reemplazar herencia con delegación





75.07
95.02

Muchas otras...