

**DISEÑO DE SOFTWARE**

**TALLER:**

Refactoring

**Equipo T2**

Ricardo Aurelio Rivera Guerra

Michelle Stephania Mujica Meneses

Adriana Lourdes Riofrío Silva

Claudia Sofia Asanza Romero

Xavier Patricio García Baño

**PARALELO:** 1

**PROFESOR:** Dr. Carlos Mera

**FECHA DE PRESENTACIÓN:**

13 de Agosto del 2020

**GUAYAQUIL - ECUADOR**



Tabla de contenido

[**Sección A: Code Smells y Técnicas de Refactorización** 1](#_Toc48229964)

[**1.** **Lazy class** 1](#_Toc48229965)

[**1.1 Consecuencia** 1](#_Toc48229966)

[**1.2 Técnica de Refactorización** 2](#_Toc48229967)

[**2.** **Speculative Generality** 3](#_Toc48229968)

[**2.1** **Consecuencia** 3](#_Toc48229969)

[**2.1** **Técnica de Refactorización** 3](#_Toc48229970)

[**Inline Method** 3](#_Toc48229971)

[**3.** **Message Chains** 4](#_Toc48229972)

[**3.1** **Consecuencia** 4](#_Toc48229973)

[**3.2** **Técnica de Refactorización** 4](#_Toc48229974)

[**4.** **Inappropiate Intimacy** 5](#_Toc48229975)

[**4.1** **Consecuencia** 5](#_Toc48229976)

[**Replace Delegation with Inheritance** 5](#_Toc48229977)

[**4.1** **Técnica de Refactorización** 6](#_Toc48229978)

[**5.** **Duplicated Code** 6](#_Toc48229979)

[**5.1** **Consecuencia** 6](#_Toc48229980)

[**5.2** **Técnica de Refactorización** 6](#_Toc48229981)

[**6.** **Long Parameter List** 7](#_Toc48229982)

[**6.1** **Consecuencia** 7](#_Toc48229983)

[**6.2** **Técnica de Refactorización** 7](#_Toc48229984)

[**7.** **Temporary field** 9](#_Toc48229985)

[7.1 **Consecuencia** 9](#_Toc48229986)

[7.2 **Técnica de Refactorización** 9](#_Toc48229987)

[**Inline temp** 9](#_Toc48229988)

[**SECCION B** 10](#_Toc48229989)

[**Repositorio** 10](#_Toc48229990)

# **Sección A: Code Smells y Técnicas de Refactorización**

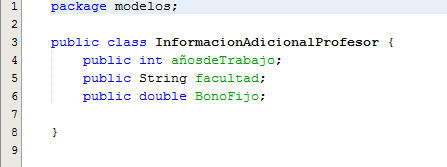
## **Lazy class**

### **1.1 Consecuencia**

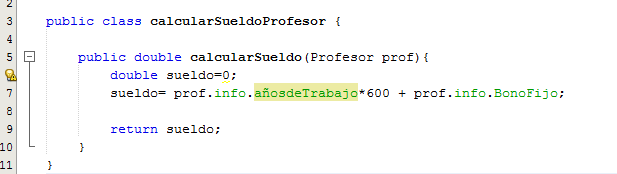
Estas clases hacen que el código sea menos entendible y eficiente

* InformacionAdicionalProfesor

Esta clase tiene algunos atributos que no son requeridos en el sistema, es decir no justifica su existencia



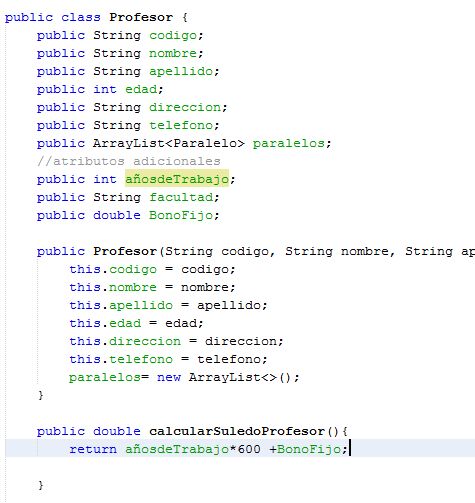
* CalcularSueldo

Esta clase solo tiene un método que puede estar dentro de Profesor para así disminuir el acoplamiento entre clases 

### **1.2 Técnica de Refactorización**

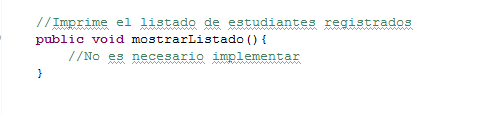
**Inline Class**

Mover las características de la clase InfoAdicional a la clase principal Profesor y a su vez el metodo Calcular



1. **Speculative Generality**
   1. **Consecuencia**

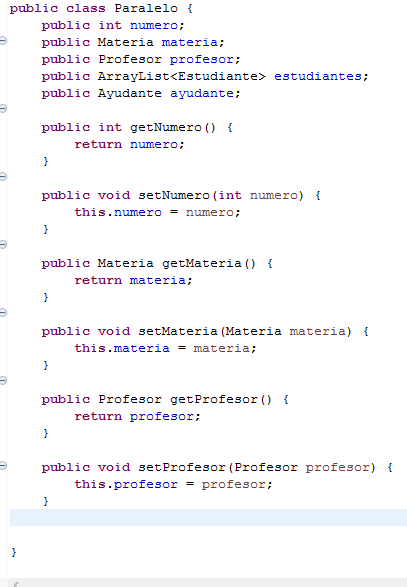
Código no utilizado o método no implementado vuelve al código más difícil de mantener como es el método de mostrarListado() que quizá este para futuras implementaciones, pero actualmente en el sistema no es usado.



* 1. **Técnica de Refactorización**

**Inline Method**

Eliminar el método no utilizado



1. **Message Chains**
   1. **Consecuencia**

|  |
| --- |
| **ANTES** |
| * Clase CalcularSueldoProfesor |
| * Aumenta el acoplamiento entre clases, por lo tanto, un cambio en una clase dentro de la cadena puede afectar el funcionamiento del método. |

* 1. **Técnica de Refactorización**

|  |
| --- |
| **MOVE METHOD** |
| * Clase Profesor |
| * Para no llamar a tantas clases es conveniente mover el método completo a la clase Profesor, dado que es esta clase la que utilizará este método siempre. Como consecuencia la clase CalcularSueldoProfesor es eliminada. |

1. **Inappropiate Intimacy**
   1. **Consecuencia**

|  |
| --- |
| **Antes** |
| * La clase Ayudante esta muy acoplada con la clase Estudiante esto causa que las clases sean difíciles de mantener, comprender, y cambiar. * La clase Ayudante se vuelve completamente dependiente de Estudiante, donde si se modifica esta clase, la clase Ayudante también debería cambiarse para cumplir con la funcionalidad adecuada. |

|  |
| --- |
| **Replace Delegation with Inheritance** |
|  |
| Ayudante extiende de Estudiante |

* 1. **Técnica de Refactorización**

1. **Duplicated Code**
   1. **Consecuencia**

|  |
| --- |
| **Antes** |
| * La clase estudiante tiene los métodos CalcularNotaInicial(), CalcularNotaFinal(), con código duplicado esto ocasiona que el método y la clase sea más larga de lo que debería ser, por lo que habría una posibilidad de bugs, haciéndolo difícil de mantener. * Hace que el código sea difícil de cambiar, ya que si se necesita cambiar alguna parte del código repetido este cambio debería hacerse para cada parte en donde se lo implementa. |

* 1. **Técnica de Refactorización**

|  |
| --- |
| **Extract Method** |
| **Despues** |
| Al momento de extraer un método que se llama en los métodos que tuvieron código duplicado esto hace que el código sea corto, facilitando su lectura, entendimiento, el debbug, y el mantenimiento de este. |

1. **Long Parameter List**
   1. **Consecuencia**

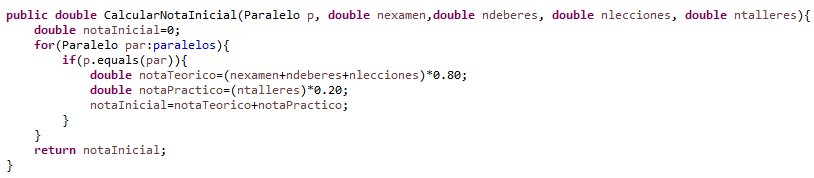
Una lista de parámetros mayor a 3 o 4 para un método hace que el código sea menos entendible, lo recomendable es tener un objeto y acceder a sus atributos.

* 1. **Técnica de Refactorización**

**Introduce Parameter Object**

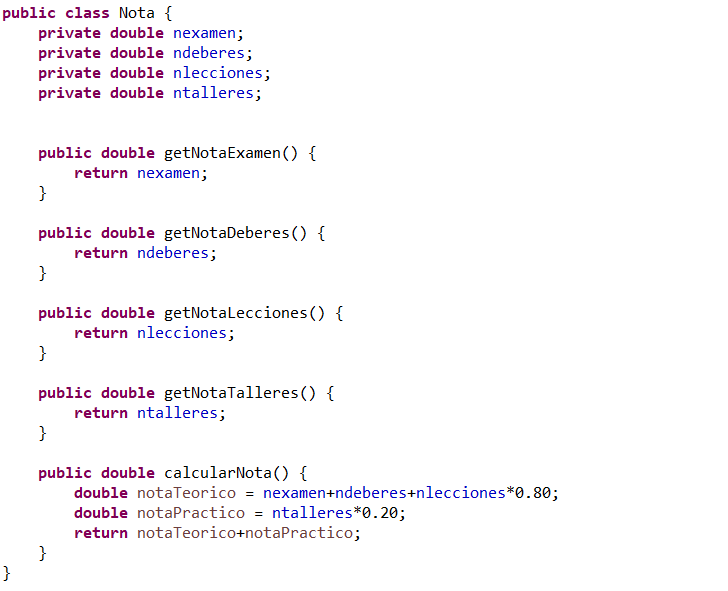
Los atributos nexamen, nlecciones y ntalleres está relacionados, así que es mejor agruparlos en un mismo objeto en lugar de enviarlos por separado. Para esto se crea una nueva clase llamada Nota, que además se encarga de hacer el cálculo de la calificación en base a estos parámetros.

**Antes**

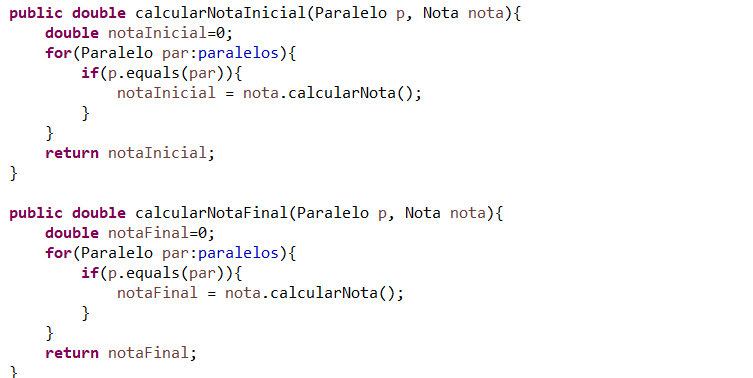


**Después**

Clase Nota



Clase Estudiante



1. **Temporary field**
   1. **Consecuencia**

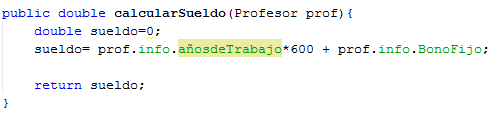
Los campos temporales ocupan espacio en las clases y alarga el código innecesariamente ya que solo se les asigna un valor y son usados bajo ciertas condiciones, el resto del tiempo están vacíos. Además, el código se vuelve más difícil de entender.

* 1. **Técnica de Refactorización**

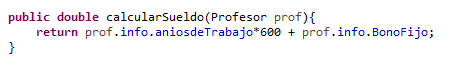
**Inline temp**

Para no usar una variable temporal, se pone directamente después de *return* el código con el que se calcula el valor a devolver.

**Antes**



**Después**



# **SECCION B**

## **Repositorio**

https://github.com/claudiasofia18/TallerRefactoring.git