

tALLER 08

Refactoring



12 de agosto de 2021

ESPOL

**Hans Ramos Mendoza**

**Paralelo 101**

Contenido

[Sección A 2](#_Toc79698759)

[**Clase:** Ayudante. 2](#_Toc79698760)

[**Code smell:** Alternative Classess with different interfaces. 2](#_Toc79698761)

[**Clase:** CalcularSueldoProfesor. 3](#_Toc79698762)

[**Code smell:** Feature Envy. 3](#_Toc79698763)

[**Clase:** Materia. 4](#_Toc79698764)

[**Code Smell:** Data class. 4](#_Toc79698765)

[**Clase:** InformacionAdicionalProfesor 6](#_Toc79698766)

[**Code Smell:** Data class 6](#_Toc79698767)

[**Clase:** Estudiante 7](#_Toc79698768)

[**Code smell:** Duplicate code. 7](#_Toc79698769)

[**Clase:** Paralelo 8](#_Toc79698770)

[**Code smell:** Dead Code 8](#_Toc79698771)

# Sección A

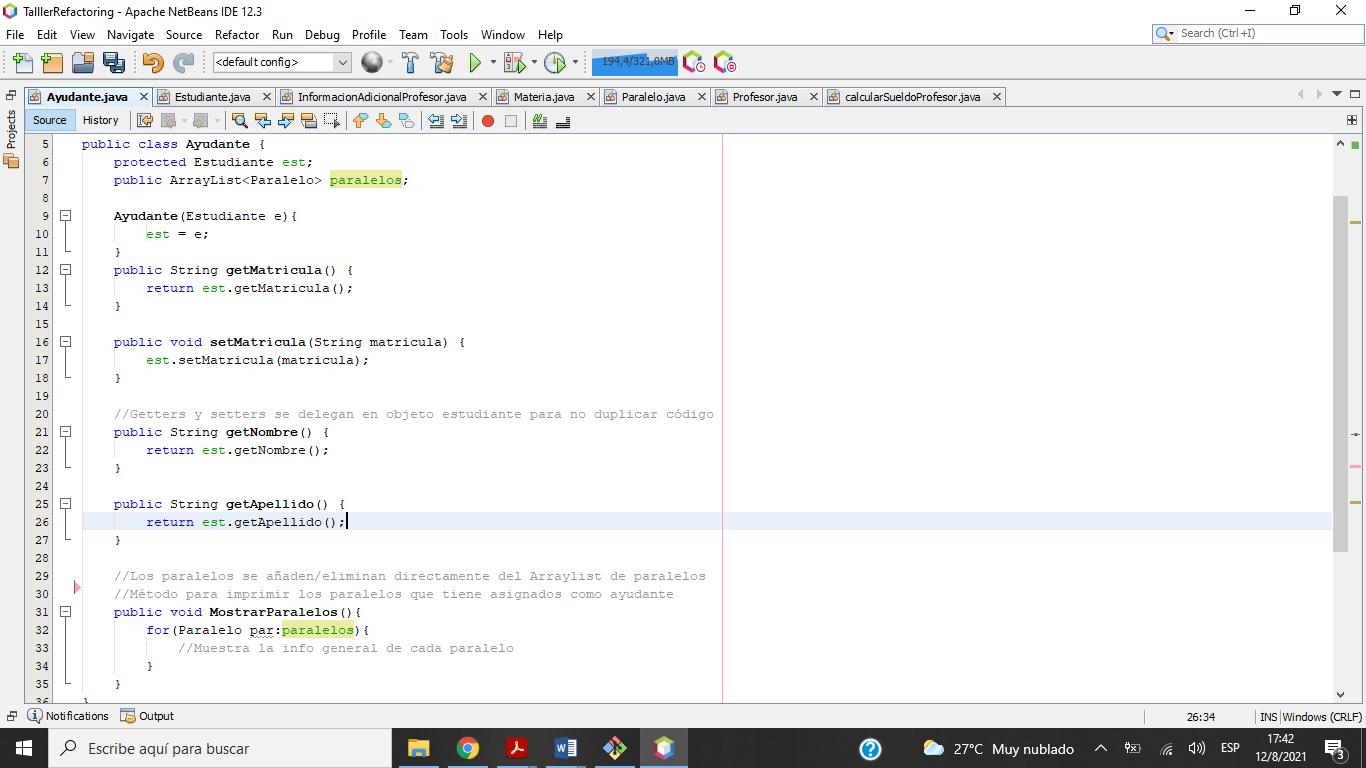
# **Clase:** Ayudante.

## **Code smell:** Alternative Classess with different interfaces.

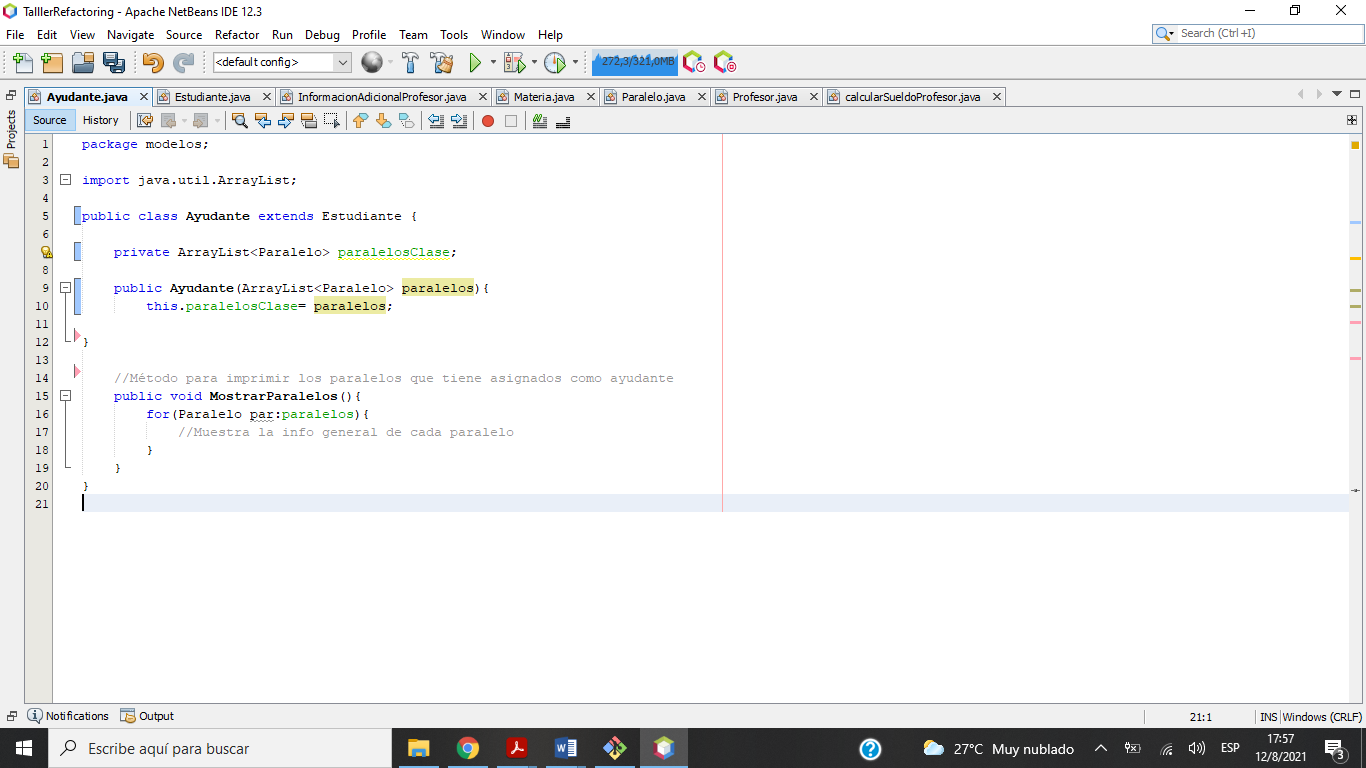
**Consecuencias:** Sucede cuando el desarrollador usa dos clases que tienen ciertas funcionalidades iguales, pero aun así permanecen separadas, o en este caso, usa la composición para arreglar el problema. El problema que surge es que mediante composición no permite usar todos los métodos de la clase compuesta.

**Técnicas usadas:**  Se usa Extract Superclass para extraer la superclase que permite heredar todas las funcionalidades que necesiten sus clases hijas, en este caso la clase hija será ayudante y la padre estudiante.

**Antes.**



**Después.**



# **Clase:** CalcularSueldoProfesor.

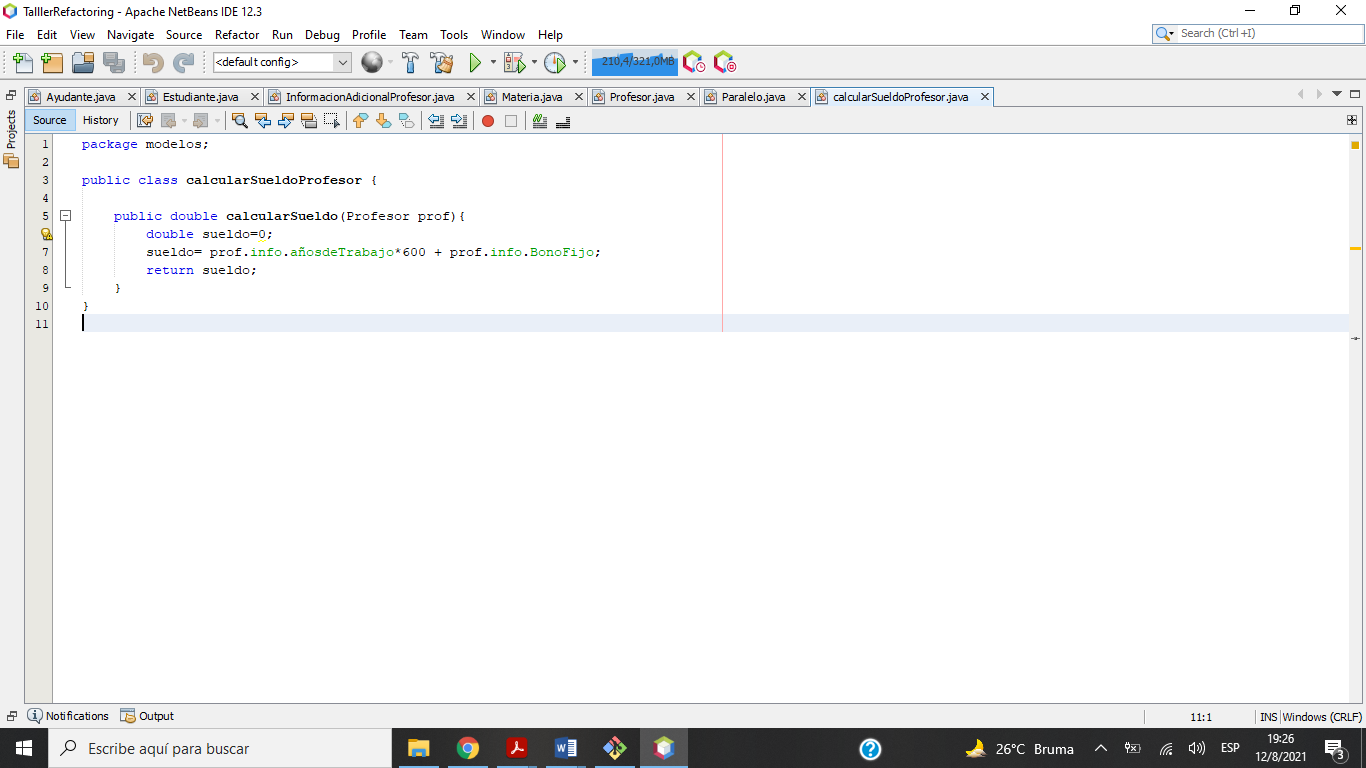
## **Code smell:** Feature Envy.

**Consecuencias:** Sucede cuando un objeto accede a datos de otra clase en lugar de su propia clase.

Las consecuencias de seguir usando este código sin refactorizar es que cuando tengamos que llamar al método necesario para realizar la acción, habrá confusiones en cuando a cuál objeto llamar, por lo que podría resultar en obstrucciones con respecto al tiempo de programación.

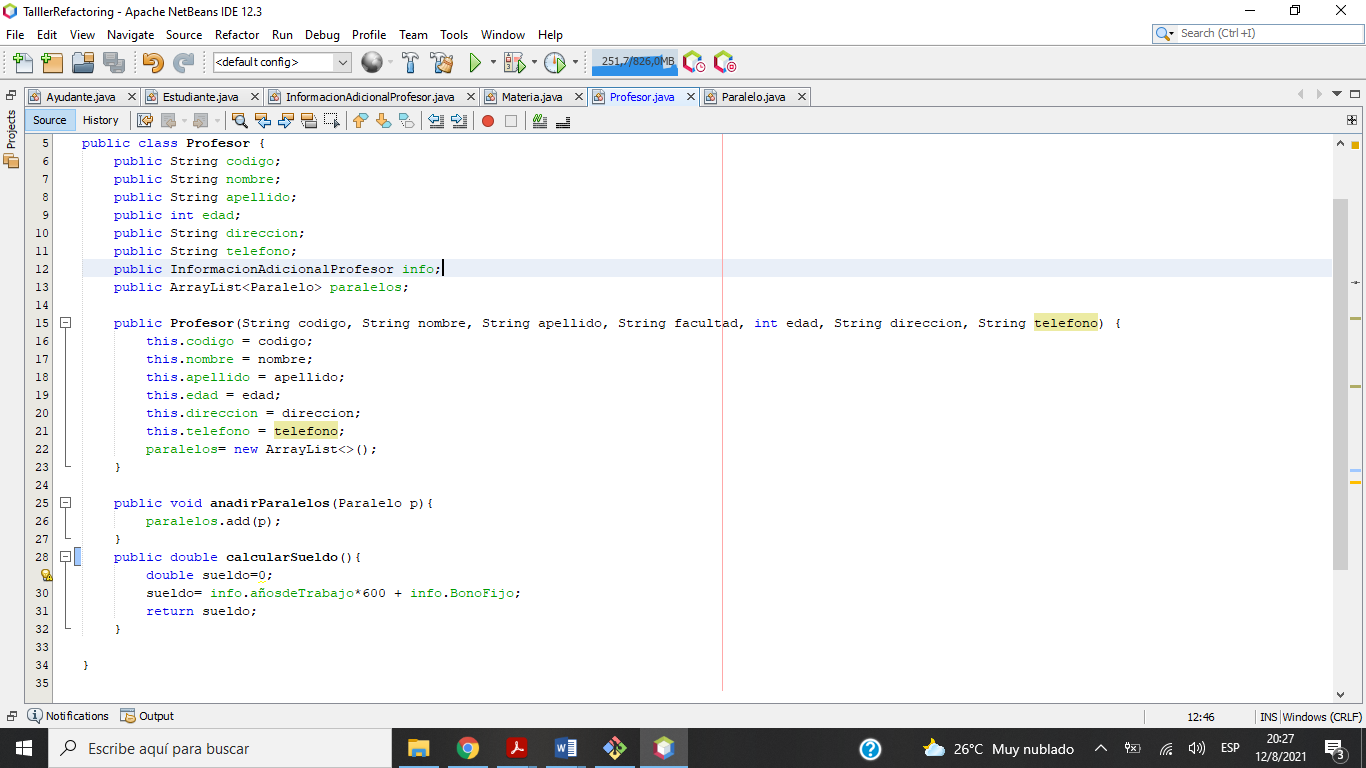
**Técnicas usadas:** Dado que en este caso la clase CalcularSueldoProfesor solo tiene un método que usa los datos del objeto profesor, usaremos **Extract method**, para enviar el método a su clase correspondiente y luego eliminaremos la clase.

**Antes:**



**Después:**

**//**Método añadido a la clase Profesor.



# **Clase:** Materia.

## **Code Smell:** Data class.

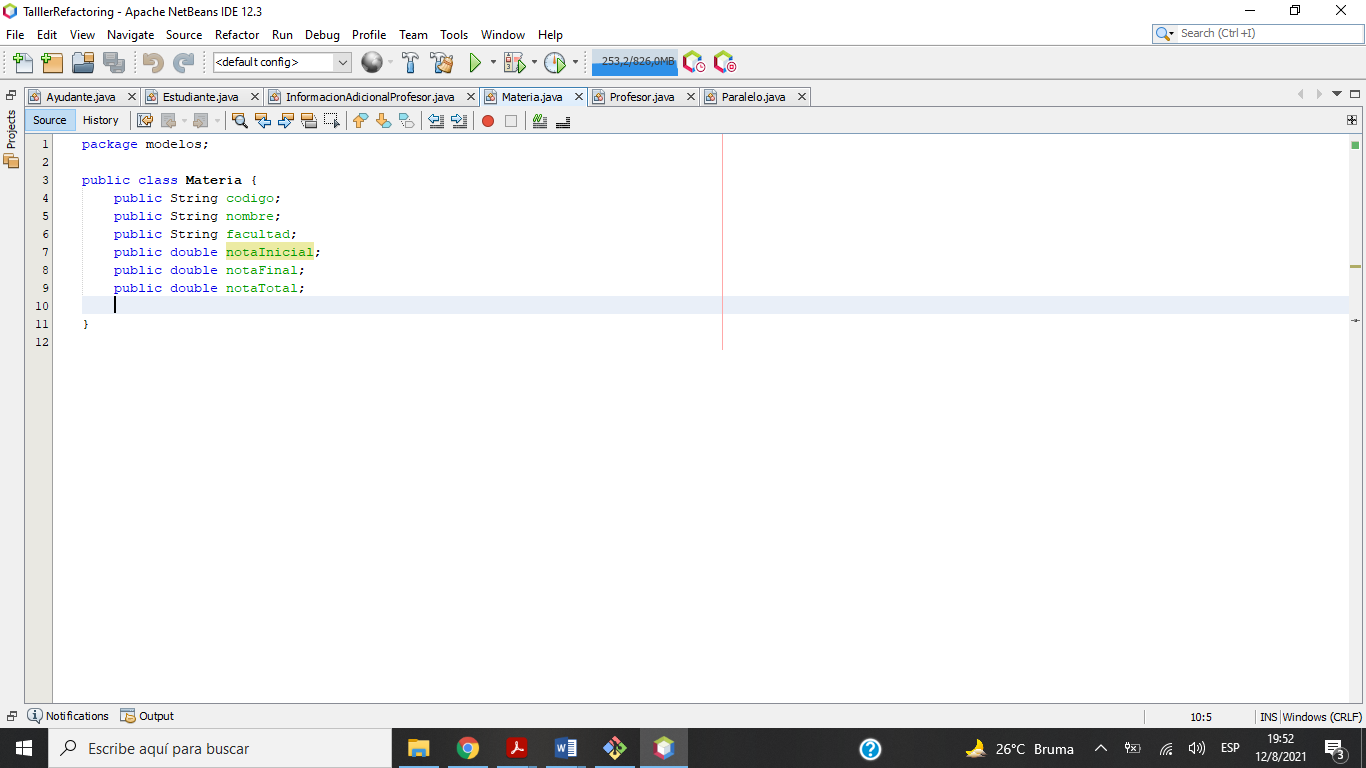
**Consecuencias:** Este smell aparece cuando se usa una clase solo para guardar atributos, los cuales no están encapsulados correctamente (se encuentran públicos) y además no contiene métodos para interactuar con dichos atributos.

La consecuencia es que cuando se quiera modificar los datos de una materia se tengan que usar métodos alojados en otra clase, lo que puede causar complicación y retrasar la escritura de código.

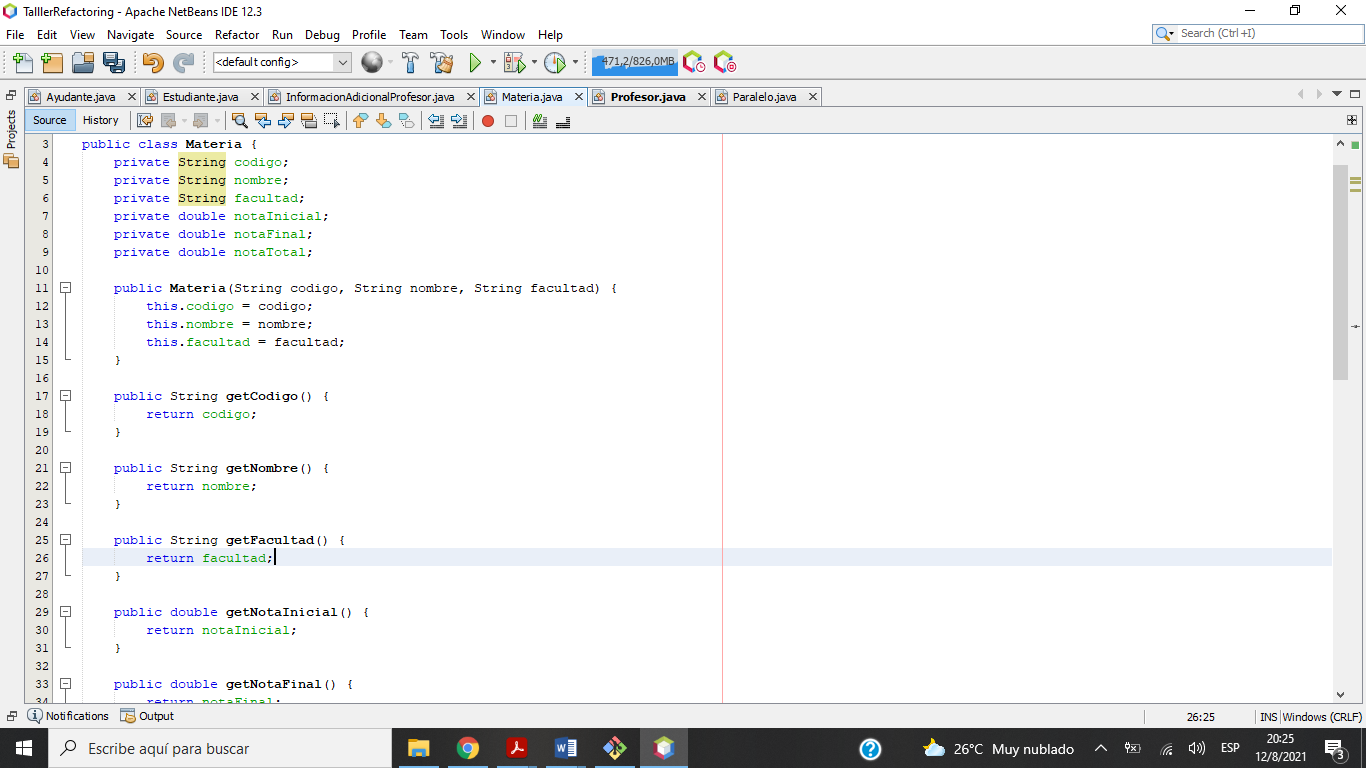
Además de tener código duplicado por el hecho de tener que acceder directamente a los atributos sin un método establecido para dicha acción.

**Técnicas usadas: Encapsulate field** lo usamos para encapsular correctamente los atributos de manera que solo puedan ser accedidos a través de la clase a la cual pertenecen, además de establecer métodos para que se pueda acceder a sus datos solo a través de ellos.

**Antes:**



**Después:**



# **Clase:** InformacionAdicionalProfesor

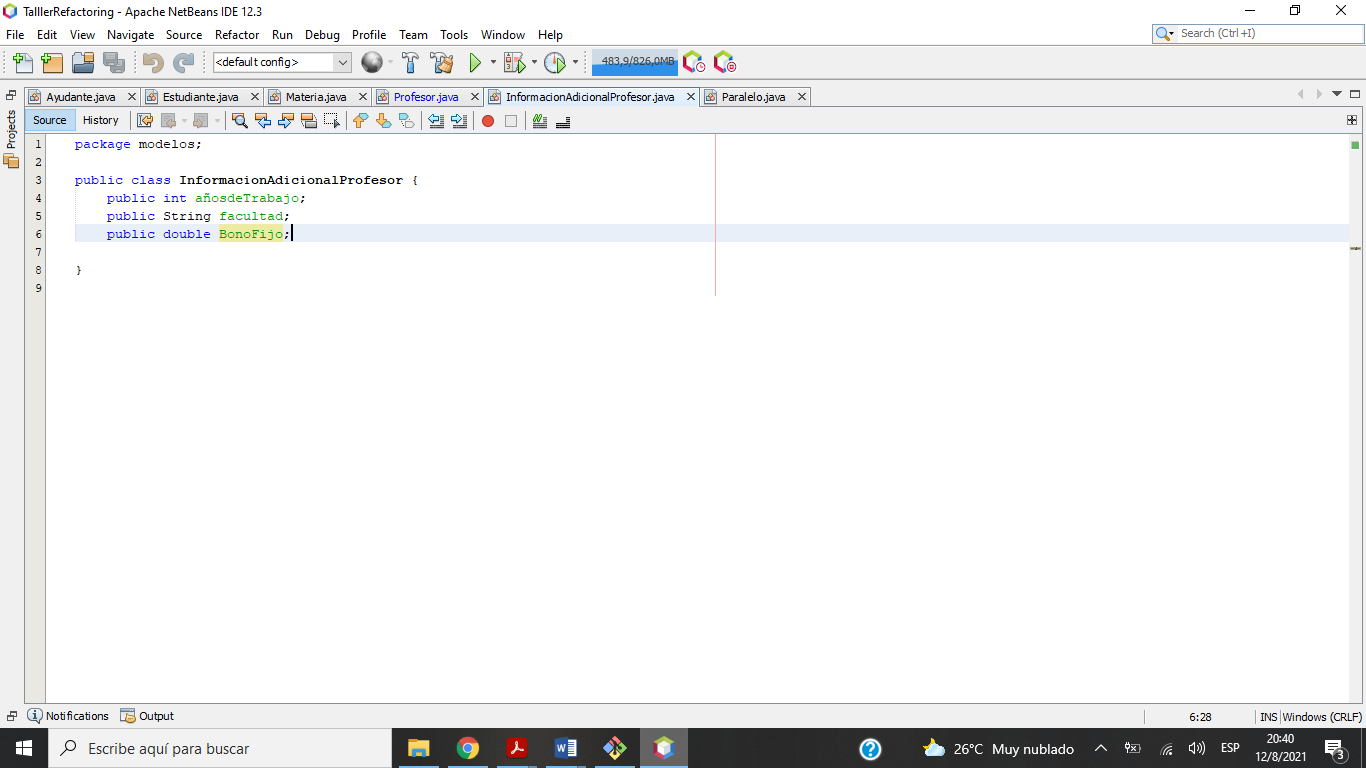
## **Code Smell:** Data class

**Consecuencias:** Este smell aparece cuando se usa una clase solo para guardar atributos, los cuales no están encapsulados correctamente (se encuentran públicos) y además no contiene métodos para interactuar con dichos atributos.

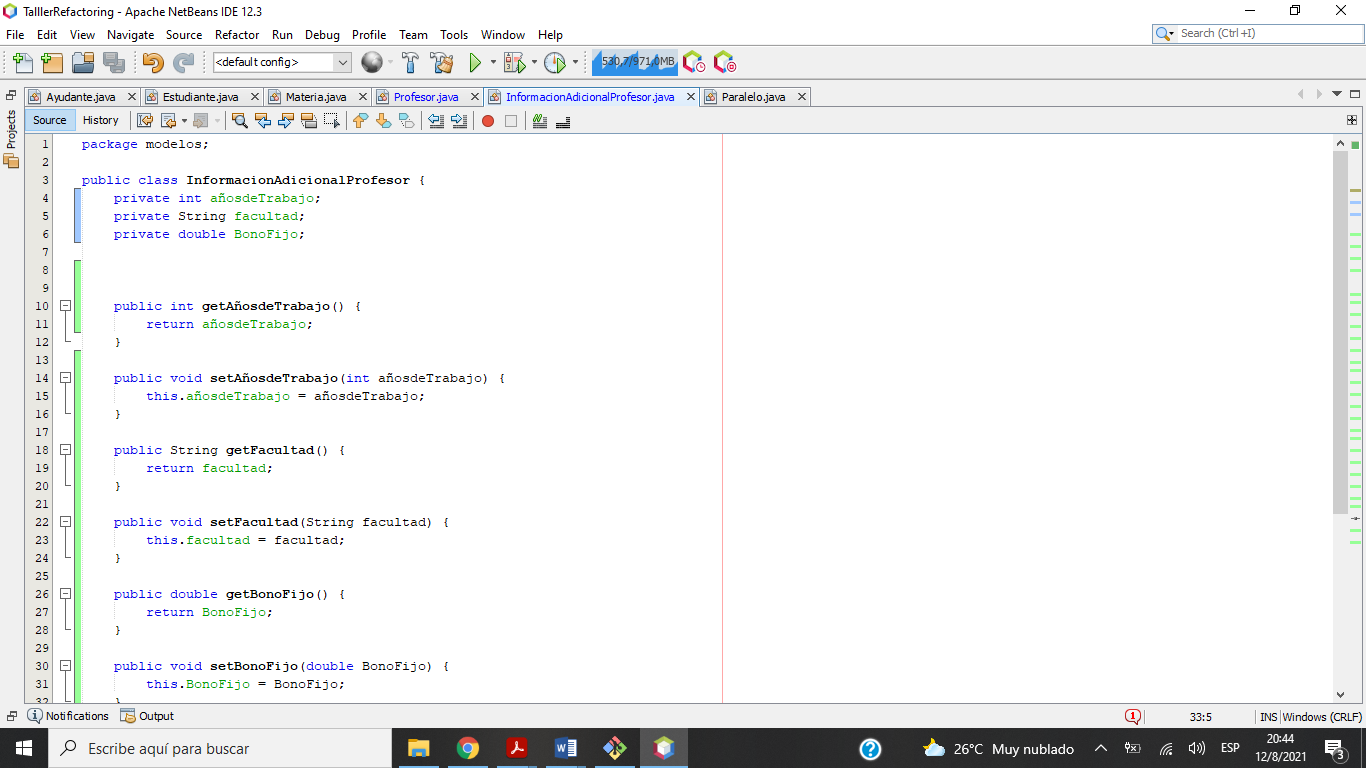
Además de tener código duplicado por el hecho de tener que acceder directamente a los atributos sin un método establecido para dicha acción.

**Técnicas usadas:** Se eliminó la clase y se uso **inline class** para pasar los atributos a su clase correspondiente.

**Antes:**



**Después:**



# **Clase:** Estudiante

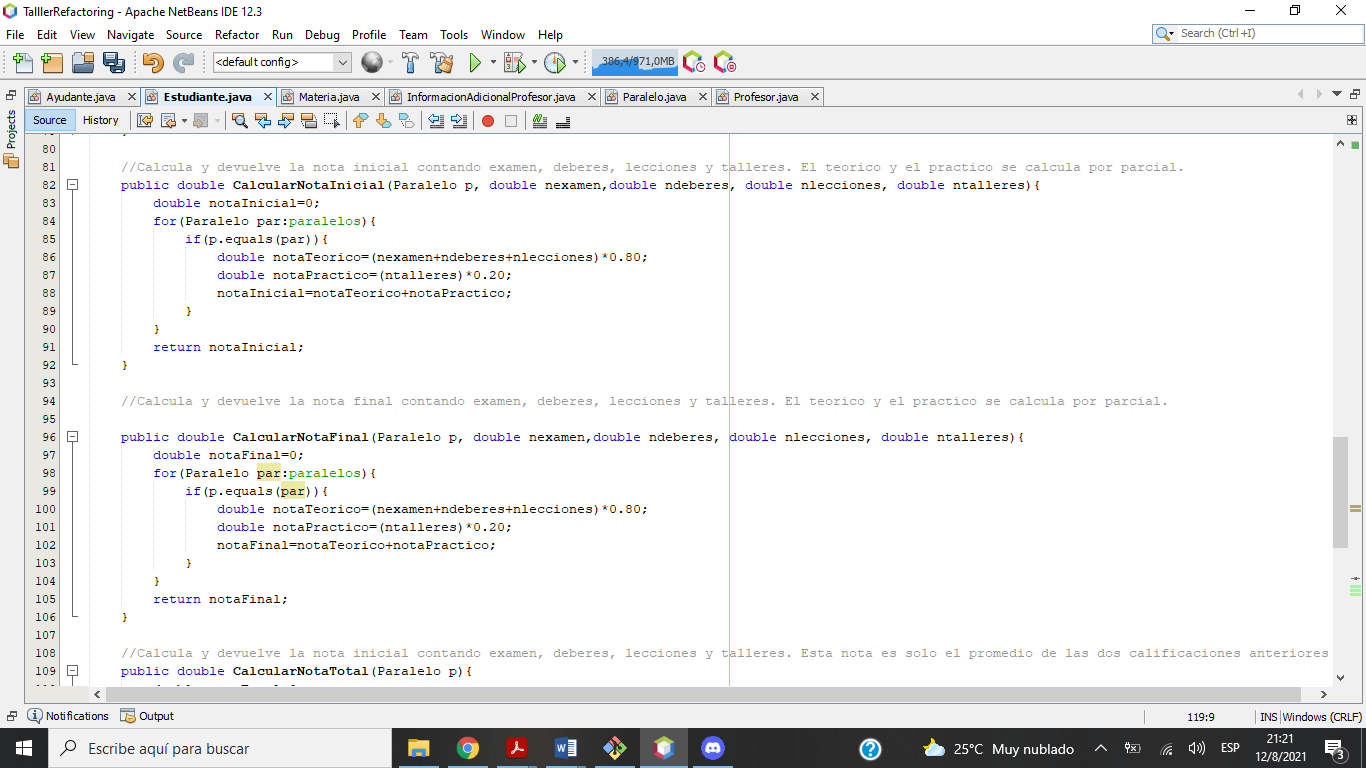
## **Code smell:** Duplicate code.

**Consecuencias:** Sucede cuando existe código repetido en el software, es necesario reemplazarlo para que no surjan futuras confusiones, además de acortar la clase en cuestión.

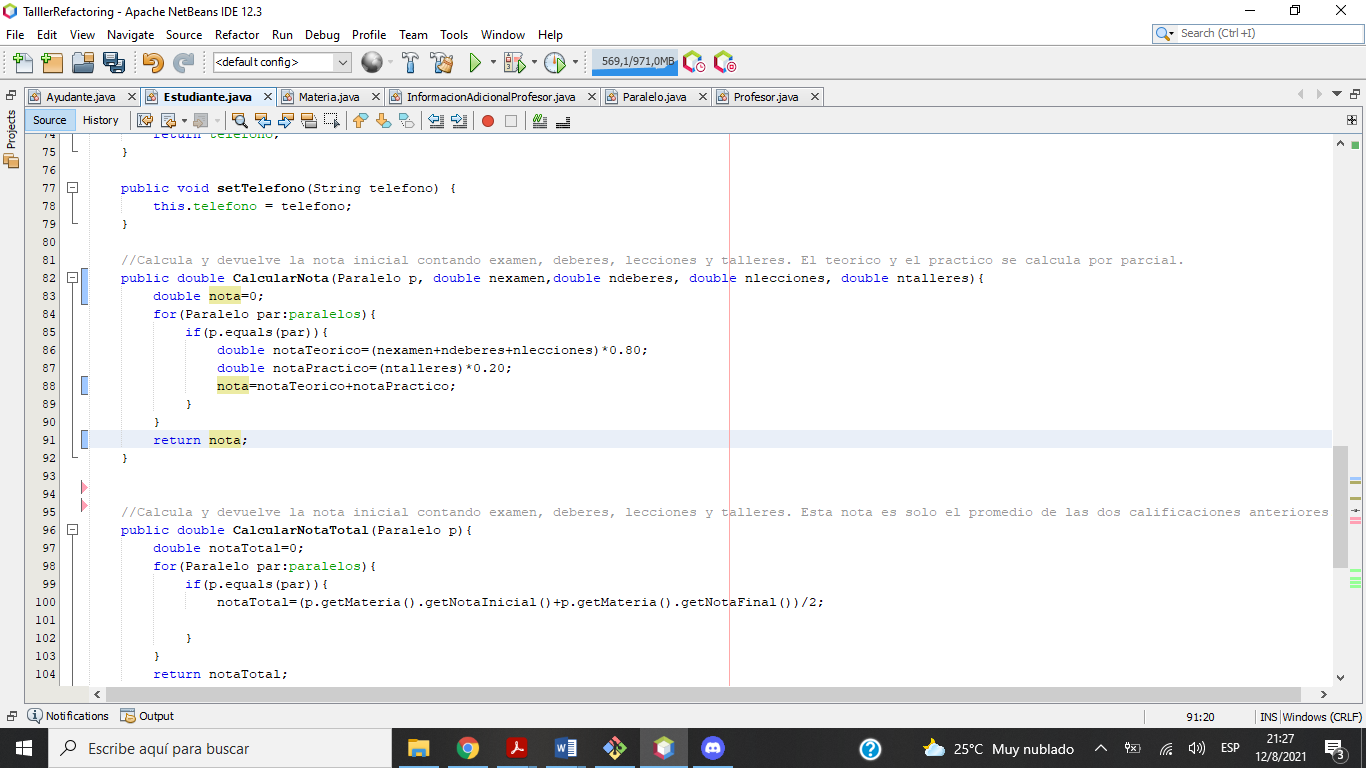
**Técnicas usadas:**

Se usa **Extract method** con un método duplicado y con el siguiente se reestablece el nombre de las variables para que sean generales y no específicos (como estaban antes de refactorizar).

**Antes:**



**Después:**



# **Clase:** Paralelo

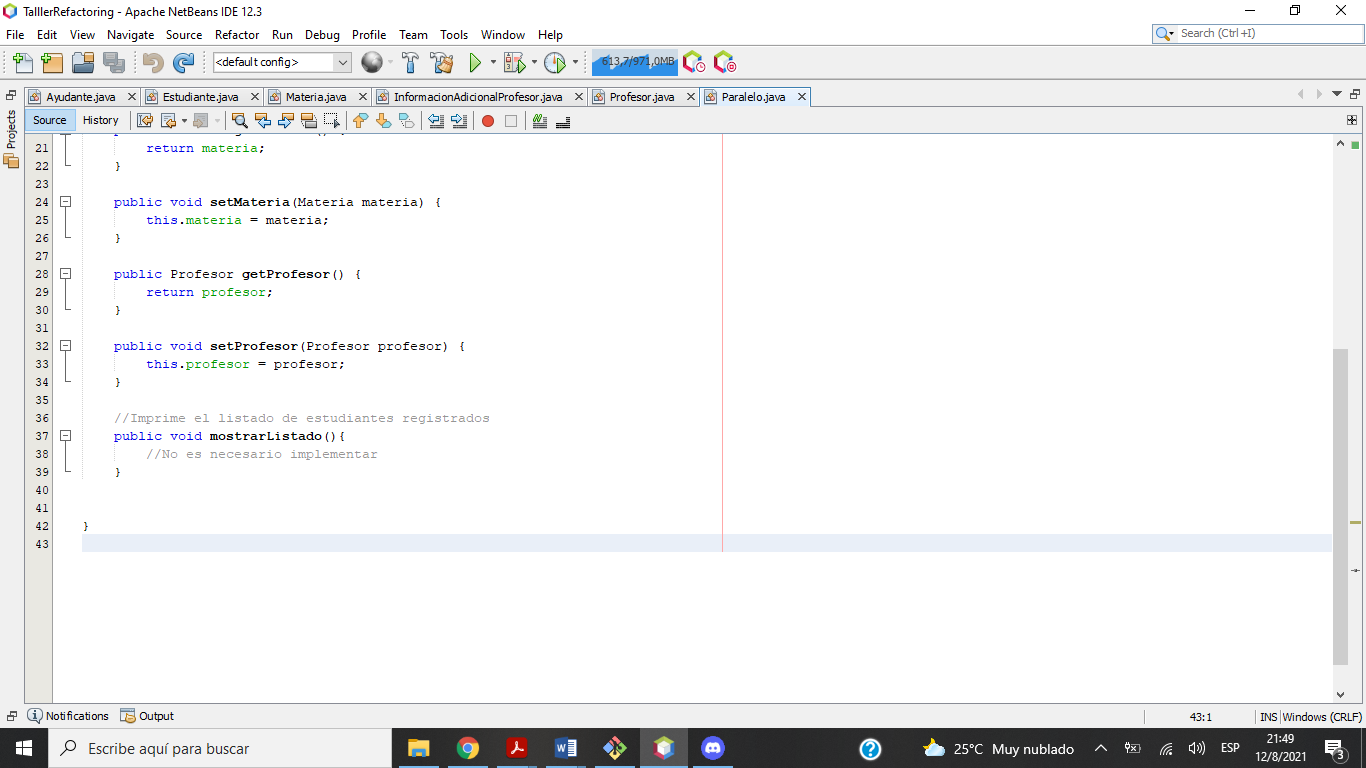
## **Code smell:** Dead Code

**Consecuencias:** Se obtiene cuando una parte de código no se usa, debido a una actualización o alguna otra razón.

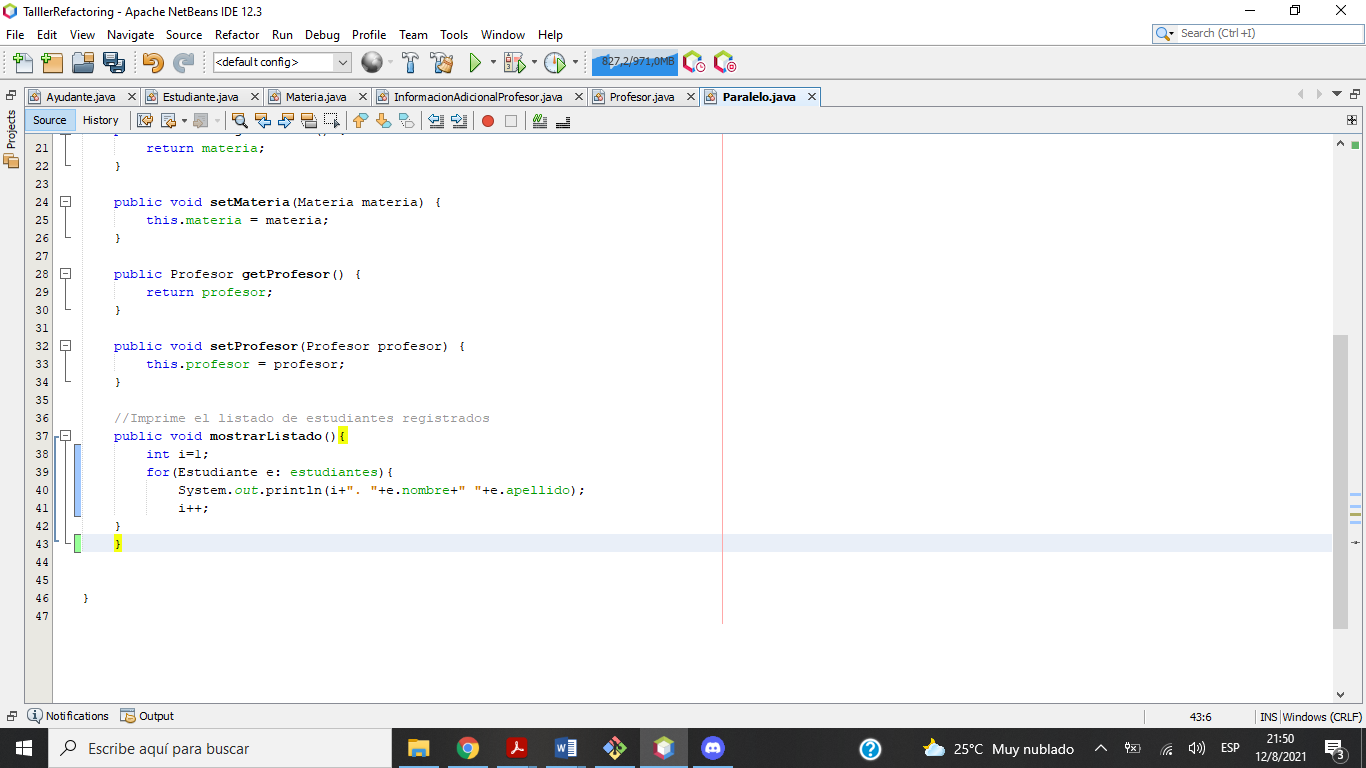
Si no se lo actualiza, va a representar un espacio no usado dentro de las líneas de código del software.

**Técnicas usadas:** Bien se puede eliminar esa parte del código o implementarla, yo decidí implementarla.

**Antes:**



**Después:**



**Sección B**

**https://github.com/HansRamos99/Taller-Refactoring.git**