**TALLER REFACTORING**

**Descripción breve:**

En equipos de trabajo, conformados por hasta cinco estudiantes.

* Identifique los malos olores de programación encontrados en el código fuente adjunto.
* Identifique las técnicas de refactorización adecuadas para eliminar los malos olores encontrados.
* Refactorice el código fuente para obtener un código más limpio y fácil de leer. Justifique su respuesta.

**INTEGRANTES:**

* Sabando Macias Douglas Javier
* Betancourt Ramírez Brayan
* Rojas Lindao Geovanny Miguel
* Parra Guevara Astrid Alejandra
* Coronel Unda Moisés

**CONTENIDO**

[**SECCION A - CODE SMELLS IDENTIFICADOS** 4](#_Toc60945823)

[**1.** **CODE SMELL:** DATA CLASS 4](#_Toc60945824)

[ **Descripción/Consecuencia:** 4](#_Toc60945825)

[ **Técnica Refactorización:** INLINE CLASS 4](#_Toc60945826)

[**2.** **CODE SMELL:** COMMENTS 5](#_Toc60945827)

[ **Descripción/Consecuencia:** 5](#_Toc60945828)

[ **Técnica Refactorización:** RENAME METHOD 5](#_Toc60945829)

[**3.** **Code Smell:** DUPLICATE CODE 6](#_Toc60945830)

[ **Descripción/Consecuencia:** 6](#_Toc60945831)

[ **Técnica Refactorización:** Extract Method 6](#_Toc60945832)

[**4.** **Code Smell:** DEAD CODE 7](#_Toc60945833)

[ **Descripción/Consecuencia:** 7](#_Toc60945834)

[ **Técnica Refactorización:** 7](#_Toc60945835)

[**5.** **Code Smell:** Temporary Field 8](#_Toc60945836)

[ **Descripción/Consecuencia:** 8](#_Toc60945837)

[ **Técnica Refactorización:** INLINE TEMP 8](#_Toc60945838)

[**6.** **Code Smell:** LAZY CLASS 9](#_Toc60945839)

[ **Descripción/Consecuencia:** 9](#_Toc60945840)

[ **Técnica Refactorización:** 9](#_Toc60945841)

[**7.** **Code Smell:** Lazy Class 10](#_Toc60945842)

[ **Descripción/Consecuencia:** 10](#_Toc60945843)

[ **Técnica Refactorización:** MOVE METHOD 10](#_Toc60945844)

[**8.** **CODE SMELL:** INAPPROPRIATE INTIMACY 11](#_Toc60945845)

[ **Descripción/Consecuencia:** 11](#_Toc60945846)

[ **Técnica Refactorización:** Replace Delegation with inheritance 11](#_Toc60945847)

[**9. CODE SMELL :** PRIMITIVE OBSESSION 13](#_Toc60945848)

[ **Descripción/Consecuencia:** 13](#_Toc60945849)

[ **Técnica de Refactorización:** Replace Data Value With Object 13](#_Toc60945850)

[**10. CODE SMELL: Alternative Classes with Different Interfaces** 14](#_Toc60945851)

[ **Descripción/Consecuencia:** 14](#_Toc60945852)

[ **Técnica Refactorización:** Extract Superclass 14](#_Toc60945853)

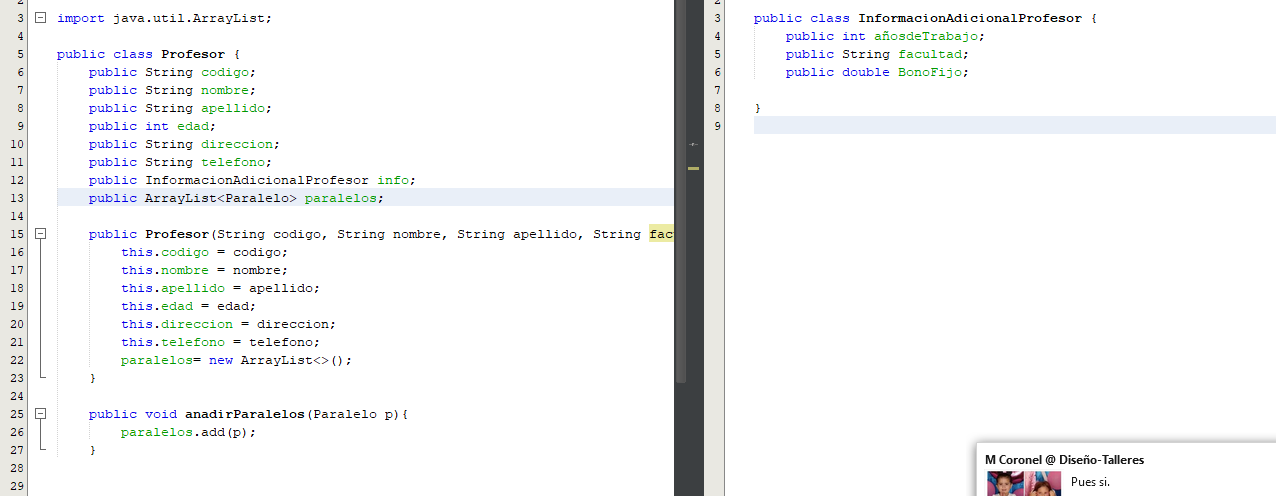
[**SECCION B – REPOSITORIO EN GITHUB** 15](#_Toc60945854)

# **SECCION A - CODE SMELLS IDENTIFICADOS**

## **CODE SMELL:** DATA CLASS

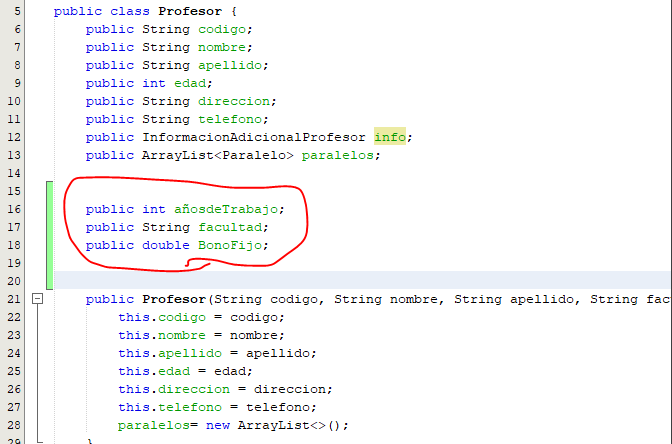
### **Descripción/Consecuencia:**

La clase informacionAdicionalProfesor solo contiene atributos y esta clase sirve como contenedor para la clase Profesor, ademas de que no presenta un comportamiento propio, una clase adicional que no hace nada más que agrupar datos.



### **Técnica Refactorización:** INLINE CLASS

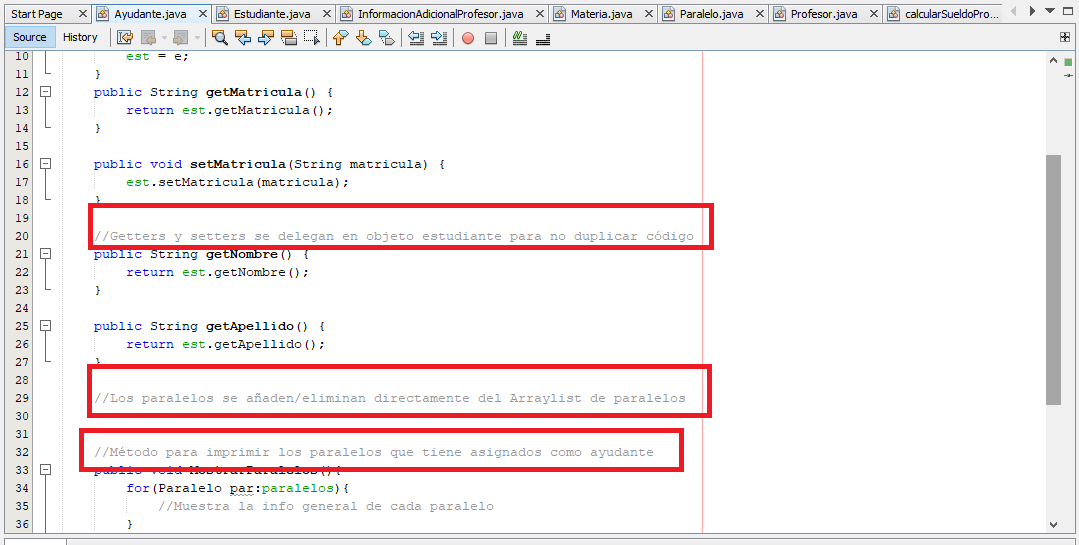
La solución es mover los campos de informacionAdicionalProfesor a Profesor y eliminar la clase informacionAdicionalProfesor.



## **CODE SMELL:** COMMENTS

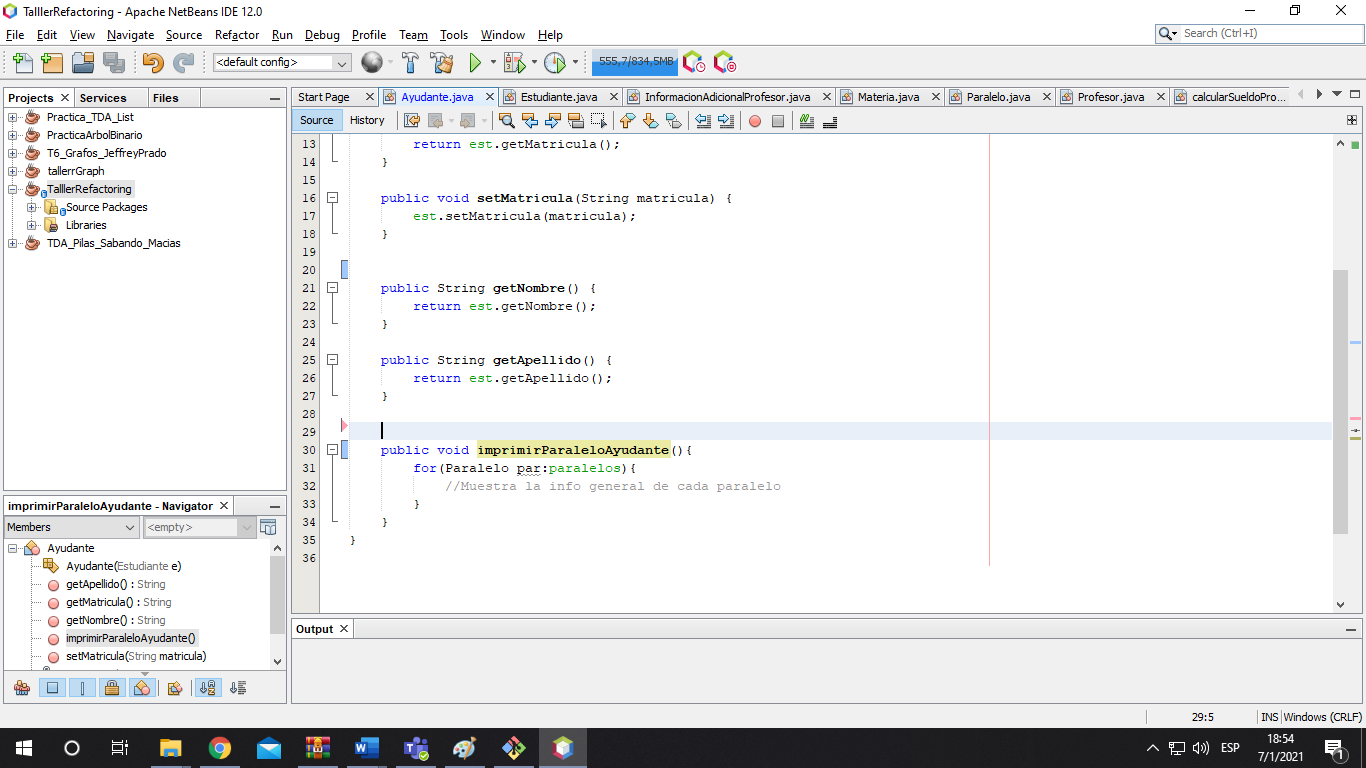
### **Descripción/Consecuencia:**

En la clase ayudante podemos evidenciar el exceso de comentarios, estos comentarios intentan explicar las acciones que el código va a realizar, pero simplemente ocupan espacio en el código y no deberían existir ya que el código debería ser intuitivo y fácil de entender, al menos en métodos cortos y sencillos.



### **Técnica Refactorización:** RENAME METHOD

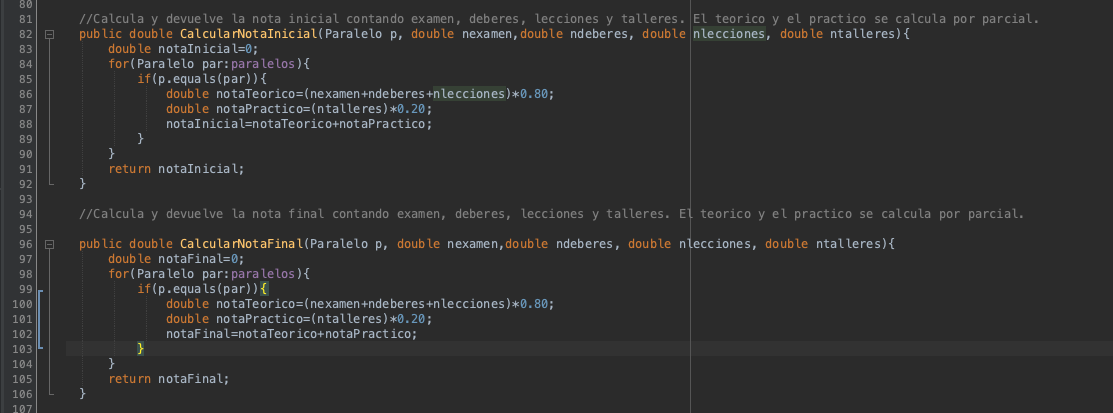
Para la refactorización de este code smell utilizaremos la técnica llamada **RENAME METHOD**, de esa manera el nombre nos va a ayudar a entender el uso y/o aplicación del método, adicionalmente se eliminarán los comentarios innecesarios.



## **Code Smell:** DUPLICATE CODE

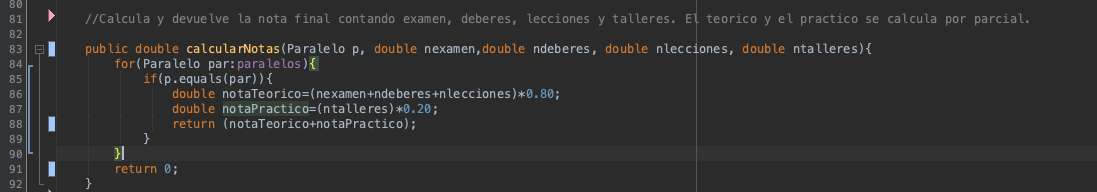
### **Descripción/Consecuencia:**

La clase Estudiante tiene dos métodos con los mismos parámetros, y que hacen prácticamente lo mismo, la única diferencia es en el nombre de una variable y el nombre del método. Como consecuencia estamos añadiendo líneas de código innecesarias.



### **Técnica Refactorización:** Extract Method

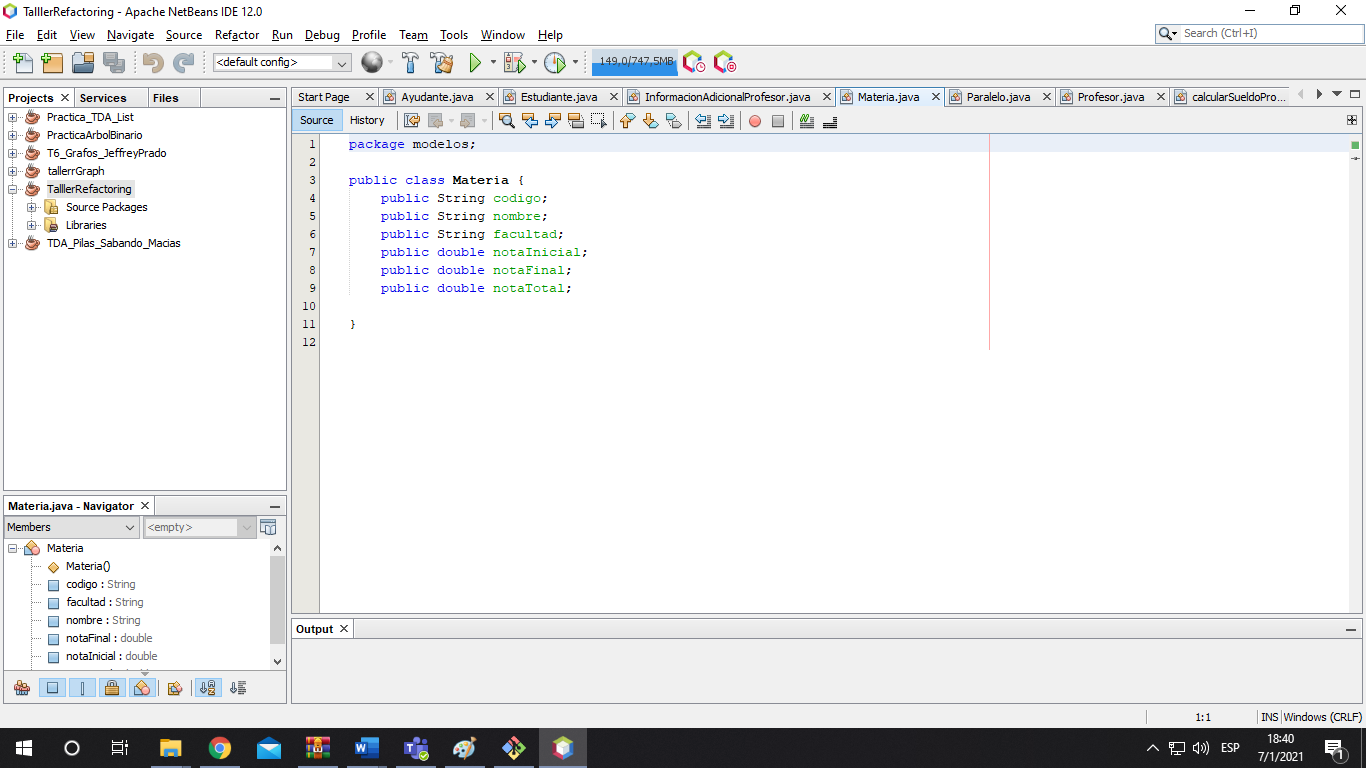
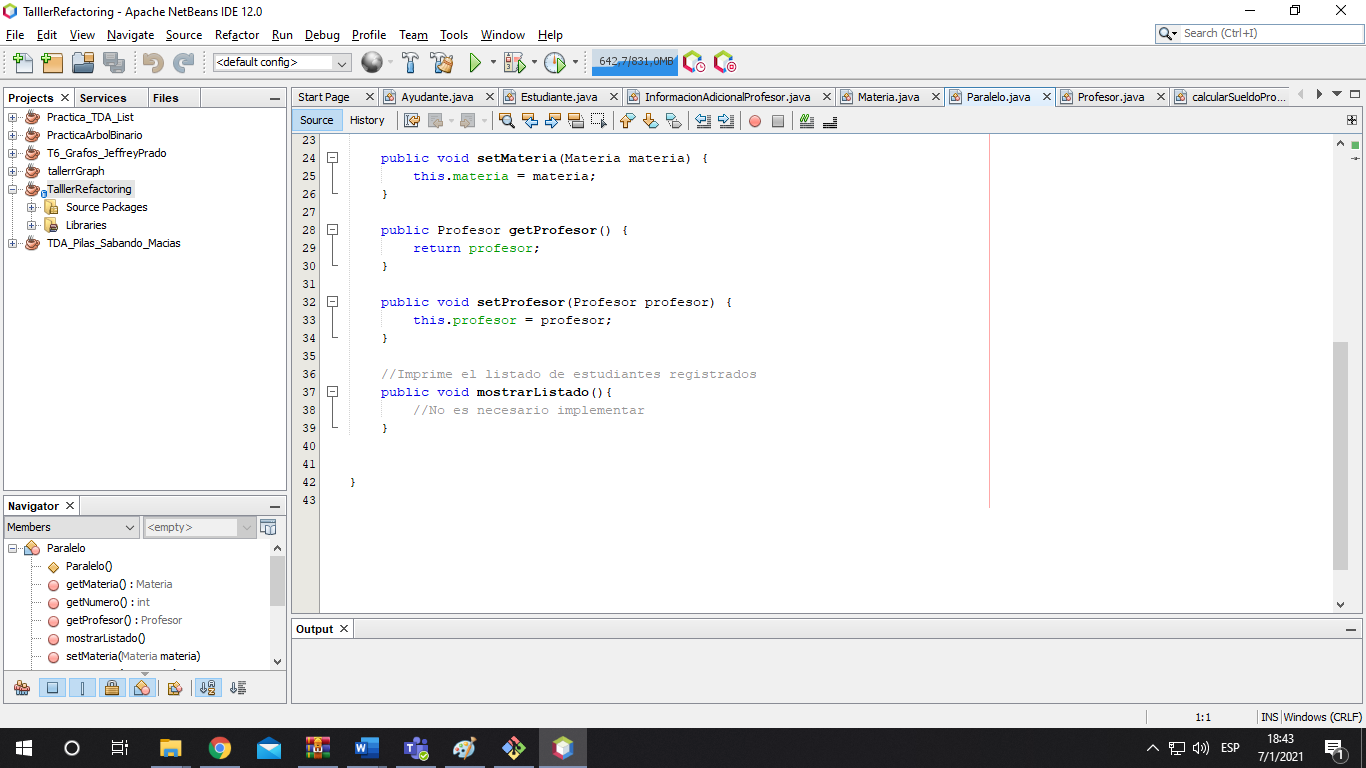
Extraemos un solo método y lo generalizamos a calcularNotas, en la implementación de calificaciones parciales se debería preguntar que parcial necesita calificar y se elimina la variable temporal de “nota” al momento de solo retornar la suma de la nota teórica y práctica, si el paralelo no es encontrado se retorna un 0.



## **Code Smell:** DEAD CODE

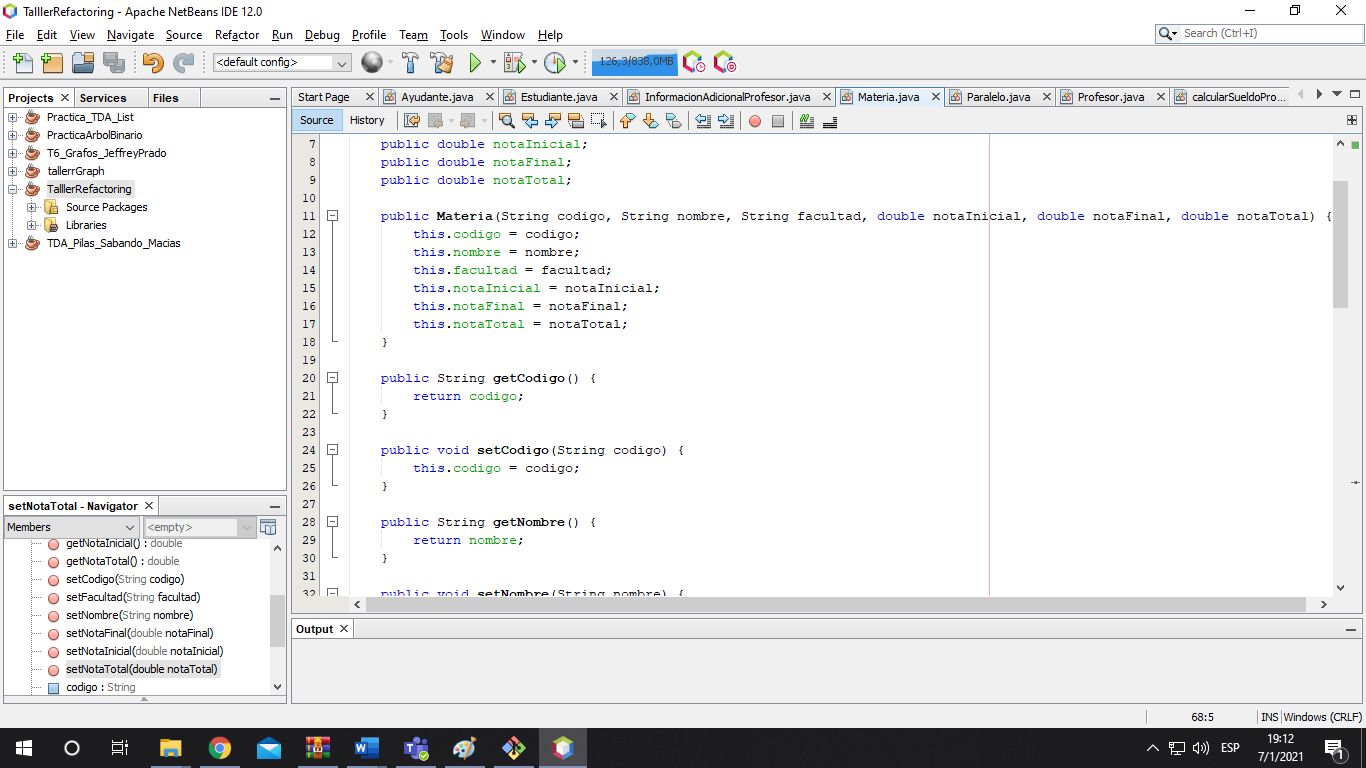
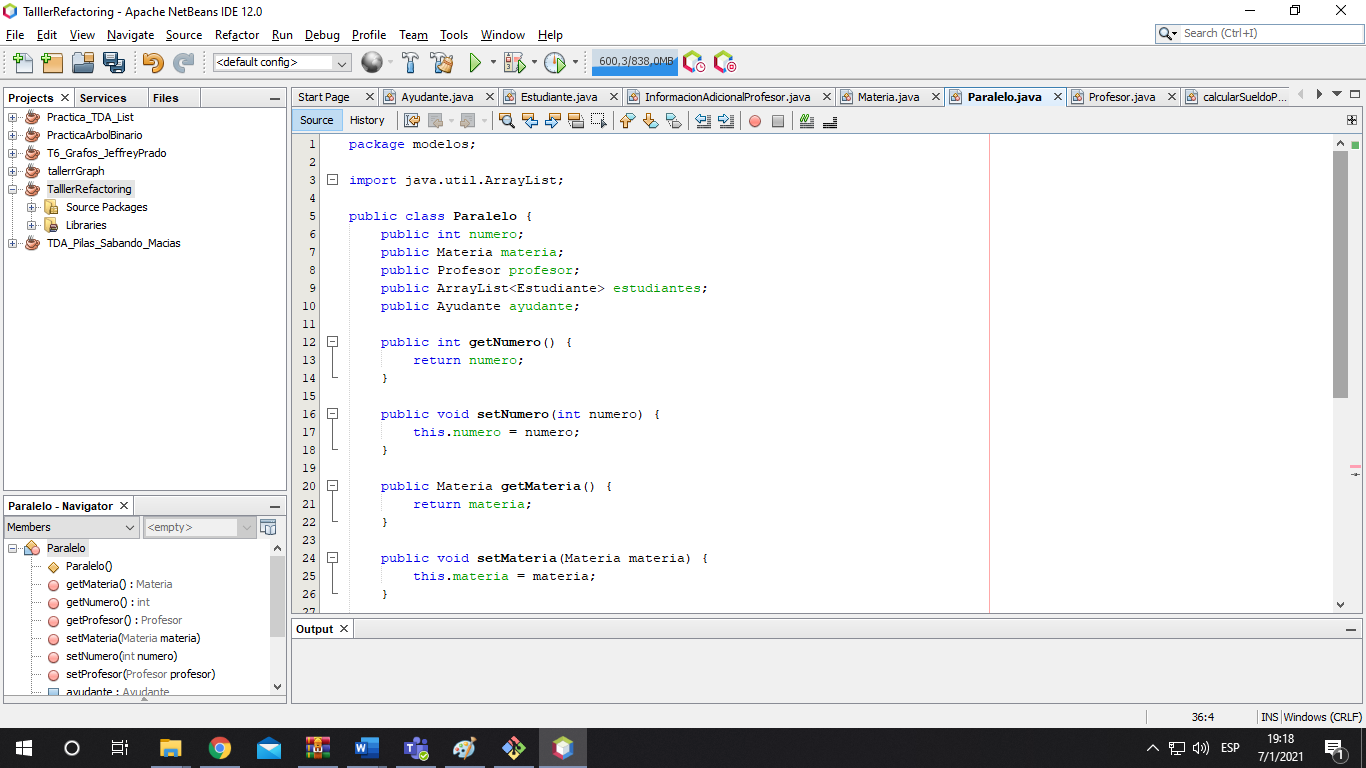
### **Descripción/Consecuencia:**

Existen atributos en la clase Materia que no son instanciados y dentro de la clase Paralelo existe un método llamado mostrarListado que no es utilizado ni está implementado, por lo tanto, esto constituye en el code smell denominado **DEAD CODE** que no cumple ninguna función en nuestro programa.

### **Técnica Refactorización:**

Para eliminar este codeSmell se borrará los métodos excedentes y se dará el peso correspondiente a la clase materia, a través de la implementación de código para que esta pueda ser accedida e instanciada.

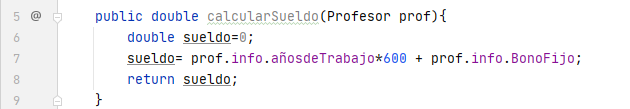


## **Code Smell:** Temporary Field

### **Descripción/Consecuencia:**

El método instancia la variable “sueldo” con valores innecesarios, esto incide en un gasto innecesario de memoria.

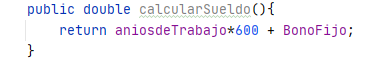
Código antes de la refactorización:



### **Técnica Refactorización:** INLINE TEMP

Se aplica la técnica Inline Temp con la finalidad de reemplazar las referencias innecesarias y retornar la expresión en sí.

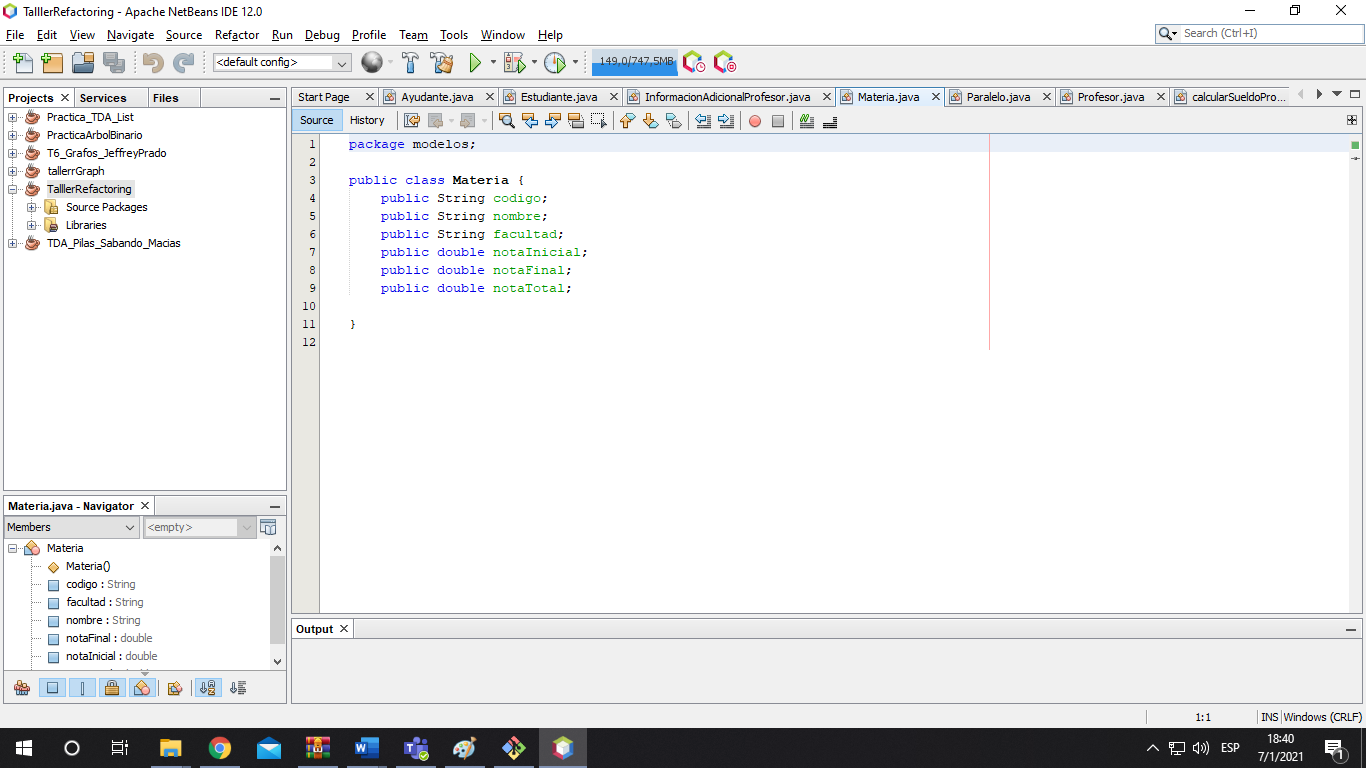
Código luego de la refactorización:



## **Code Smell:** LAZY CLASS

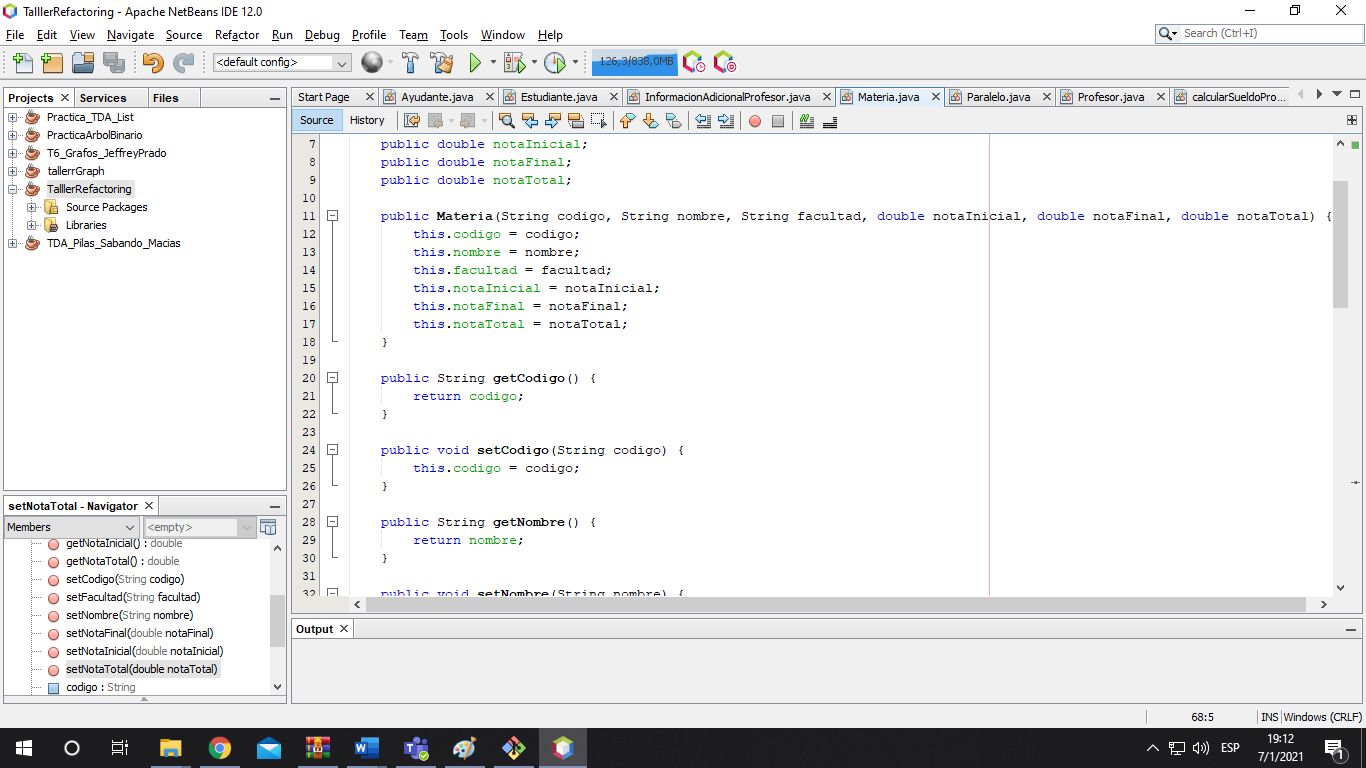
### **Descripción/Consecuencia:**

La clase Materia no realiza ninguna acción ni es instanciada, no posee constructor ni atributos para poder usarla, así que está consumiendo recursos para ser mantenida.



### **Técnica Refactorización:**

Para eliminar este code smell vamos a crear un constructor junto con getters y setter a la clase materia, de esa materia se podrá instanciar de manera correcta cuando se requiera.

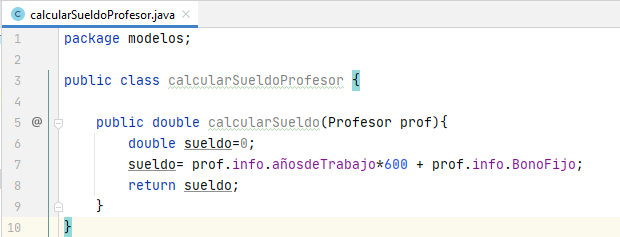


## **Code Smell:** Lazy Class

### **Descripción/Consecuencia:**

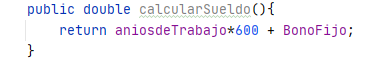
La clase CalcularSueldoProfesor solo tiene un método, mismo que utiliza información que además está en la clase Profesor, por lo que se podría inferir también que existe Future Envy. Sin embargo, esta clase está consumiendo recursos de mantenimiento innecesarios.

Código antes de la refactorización:



### **Técnica Refactorización:** MOVE METHOD

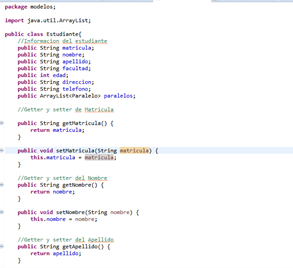
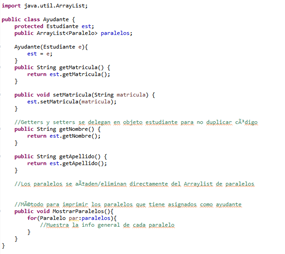
Se utiliza la técnica Move Method, así *calcularSueldo*  pasa a ser parte de la clase Profesor. Como la técnica Inline Temp ya ha sido aplicada anteriormente para suprimir el uso innecesario de variables, el resultado se mantiene igual.



## **CODE SMELL:** INAPPROPRIATE INTIMACY

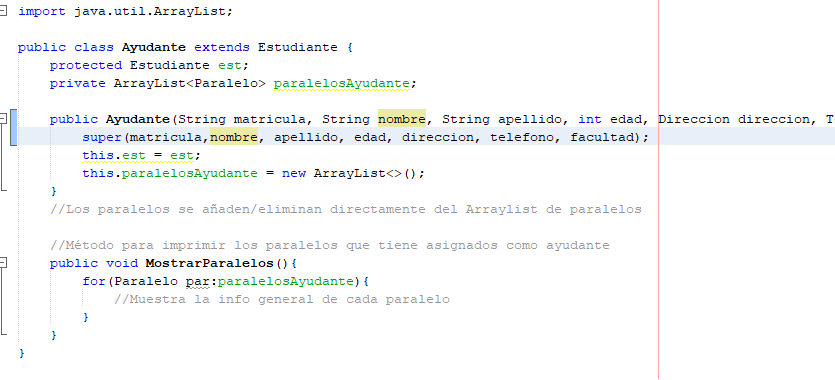
### **Descripción/Consecuencia:**

La clase Ayudante usa los getters y setters de la clase Estudiante además de sus atributos, creando código repetido, ya que sus atributos son los mismos, es una práctica innecesaria considerando que un ayudante es un estudiante



### **Técnica Refactorización:** Replace Delegation with inheritance

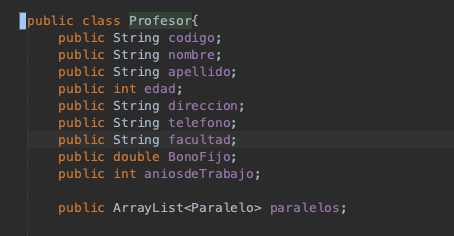
Considerando que un Ayudante es un Estudiante, se procede cambiar a la clase Ayudante una clase hija de Estudiante, así podrá heredar los atributos de Estudiantes al igual que sus métodos evitando reescribir ese código.



# **9. CODE SMELL :** PRIMITIVE OBSESSION

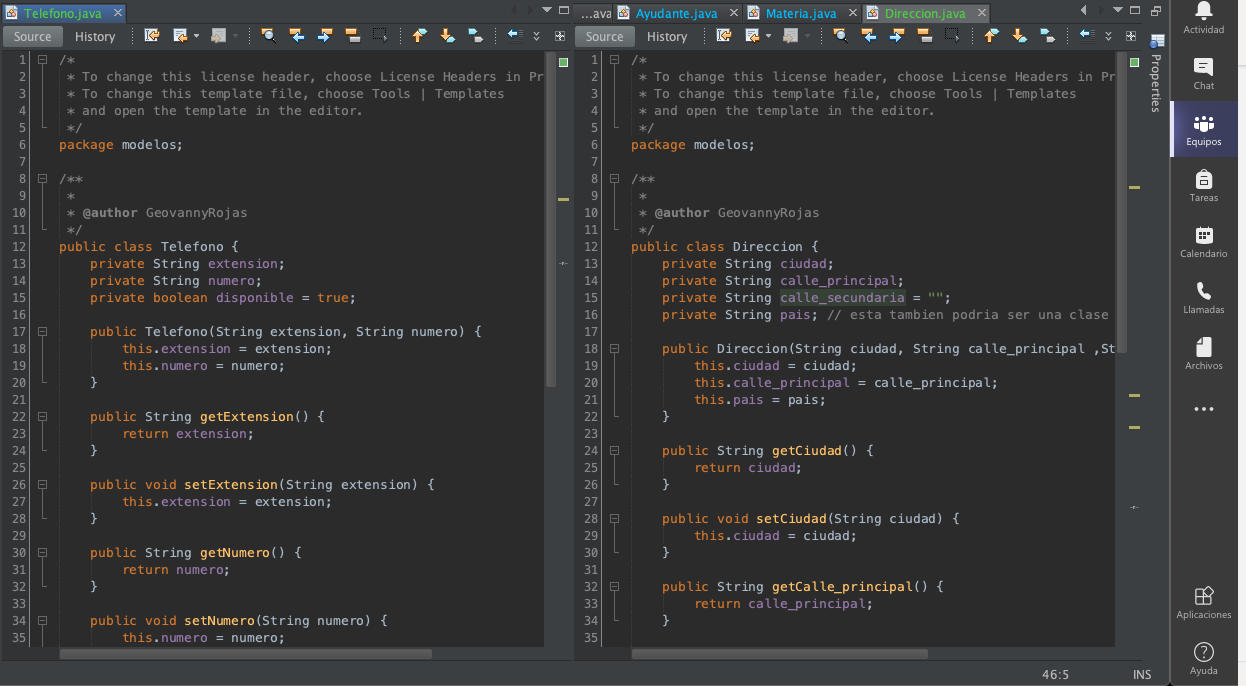
### **Descripción/Consecuencia:**

El uso de String en dirección y teléfono es primitive obsession debido a que estos dos atributos pueden tener más elementos implementados. En consecuencia, si el teléfono necesita un tipo extensión no se podría identificar y en la dirección se guardaría mejor los datos en una clase.

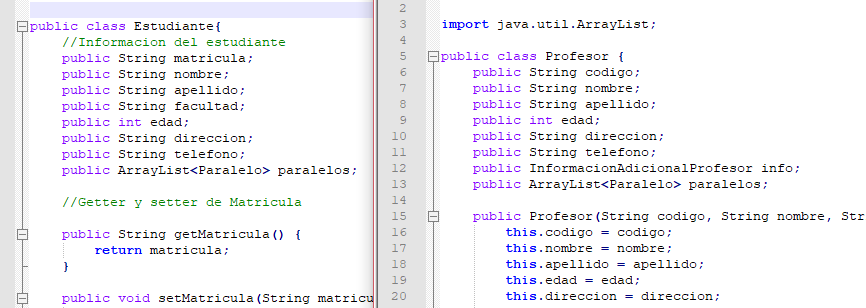


### **Técnica de Refactorización:** Replace Data Value With Object

Para solucionar este Code Smell debemos reemplazar los datos primitivos por un objeto, así tener mejor manejo de los mismo.



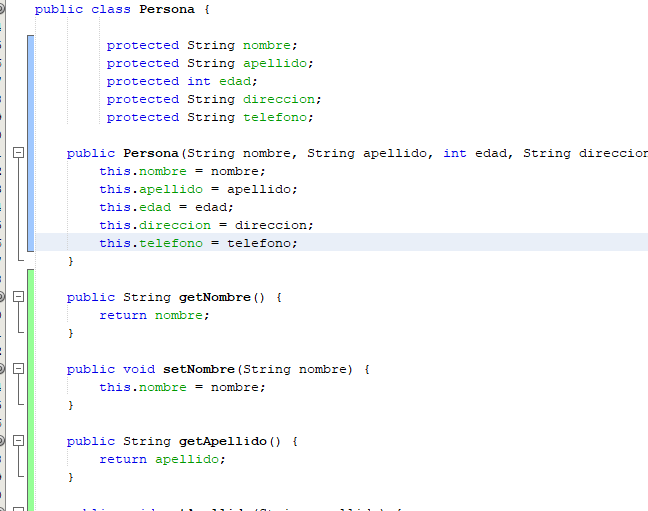
## **10. CODE SMELL: Alternative Classes with Different Interfaces**



### **Descripción/Consecuencia:**

Las clases Estudiante y Profesor comparten muchos atributos, esto crea un problema de duplicidad de comportamiento y de código

### **Técnica Refactorización:** Extract Superclass



Se crea una superclase con sus atributos como protected y métodos getters y setters, Profesor y Estudiante extienden de esta

# **SECCION B – REPOSITORIO EN GITHUB**

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