**TALLER DE REFACTORING**

Encontrar tipos de Code Smells y solucionarlos

**Descripción breve:**

En equipos de trabajo, conformados por hasta tres estudiantes, (i) identifique los malos olores de programación encontrados en el código fuente adjunto, (ii) Identifique las técnicas de refactorización adecuadas para eliminar los malos olores encontrados y (iii) Refactorice el código fuente para obtener un código más limpio y fácil de leer. Justifique su respuesta.

INTEGRANTES:

* Benalcázar García Sebastián Elías
* Poveda Páez Alexis Steven
* Townsend Hinostroza Darinka Milena

**CONTENIDO**

**[OBJETIVOS ESPECÍFICOS](#_Toc48253739)** [3](#_Toc48253739)

[**DESCRIPCIÓN** 3](#_Toc48253740)

[**ESPECIFICACIONES** 3](#_Toc48253741)

[**Sección A** 3](#_Toc48253742)

[**Sección B** 3](#_Toc48253743)

[**SECCION A - CODE SMELLS IDENTIFICADOS** 4](#_Toc48253744)

[**1.** **Code Smells #1: Duplicate Code** 4](#_Toc48253745)

[**1.** **Consecuencia:** 4](#_Toc48253746)

[**2.** **Solución:** 4](#_Toc48253747)

[**2.** **Code Smells #2: Clase Perezosa** 5](#_Toc48253748)

[**1.** **Consecuencia:** 5](#_Toc48253749)

[**2.** **Solución:** 5](#_Toc48253750)

[**3.** **Code Smells #3: Dead Code** 6](#_Toc48253751)

[**1.** **Consecuencia:** 6](#_Toc48253752)

[**2.** **Solución:** 6](#_Toc48253753)

[**4.** **Code Smells #3: Data Class** 7](#_Toc48253754)

[**1.** **Consecuencia:** 7](#_Toc48253755)

[**2.** **Solución:** 7](#_Toc48253756)

[**5.** **Code Smells #5: Speculate Generality** 8](#_Toc48253757)

[**1.** **Consecuencia:** 8](#_Toc48253758)

[**2.** **Solución:** 8](#_Toc48253759)

[**6.** **Code Smells #6: Feature Envy** 9](#_Toc48253760)

[**1.** **Consecuencia:** 9](#_Toc48253761)

[**2.** **Solución:** 9](#_Toc48253762)

[**7.** **Code Smells #7: Comments** 10](#_Toc48253763)

[**1.** **Consecuencia:** 10](#_Toc48253764)

[**2.** **Solución:** 10](#_Toc48253765)

[**8.** **Code Smells #8: Long Class** 11](#_Toc48253766)

[**1.** **Consecuencia:** 11](#_Toc48253767)

[**2.** **Solución:** 11](#_Toc48253768)

# **OBJETIVOS ESPECÍFICOS**

* Identificar malos olores de programación en el código fuente adjunto y las técnicas de refactorización correspondientes.
* Aplicar técnicas de refactorización que pueden aplicarse para eliminar los malos olores previamente identificados.

# **DESCRIPCIÓN**

En equipos de trabajo, conformados por hasta tres estudiantes, (i) identifique los malos olores de programación encontrados en el código fuente adjunto, (ii) Identifique las técnicas de refactorización adecuadas para eliminar los malos olores encontrados y (iii) Refactorice el código fuente para obtener un código más limpio y fácil de leer. Justifique su respuesta.

# **ESPECIFICACIONES**

Considere un **sistema académico** que permite manejar los estudiantes registrados en ciertos paralelos de distintas materias. Además, cada materia tiene un profesor y podría tener asignado un ayudante.

## **Sección A**

Elabore un **reporte** en el que **identifique los code smells** encontrados en el código adjunto. Para cada code smell debe **indicar el nombre**, las **consecuencias** de mantener el mismo en dicho código y la(s) técnicas de refactorización utilizadas para eliminarlo. Para cada mal olor coloque una captura inicial y una del código refactorizado. Indique cualquier asunción que realice. **[60%]**

## **Sección B**

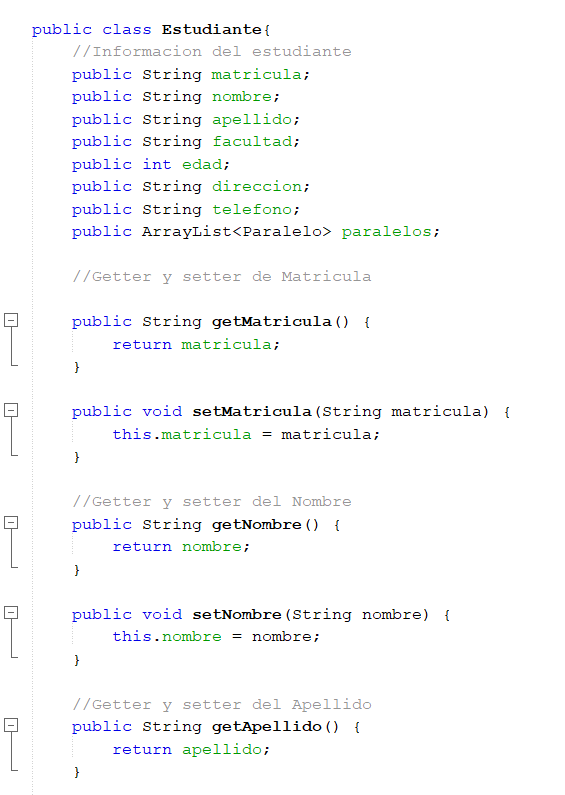
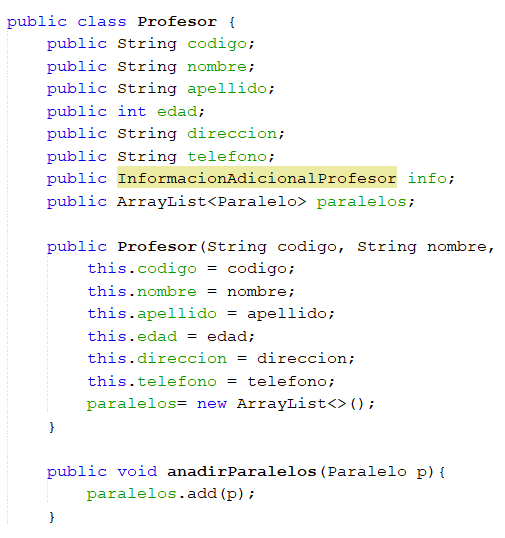
Cree un repositorio en GitHub para subir el reporte y realizar las mejoras sobre el código inicial y aplique las técnicas indicadas en el reporte. **[40%]**

# **SECCION A - CODE SMELLS IDENTIFICADOS**

## **Code Smells #1: Duplicate Code**

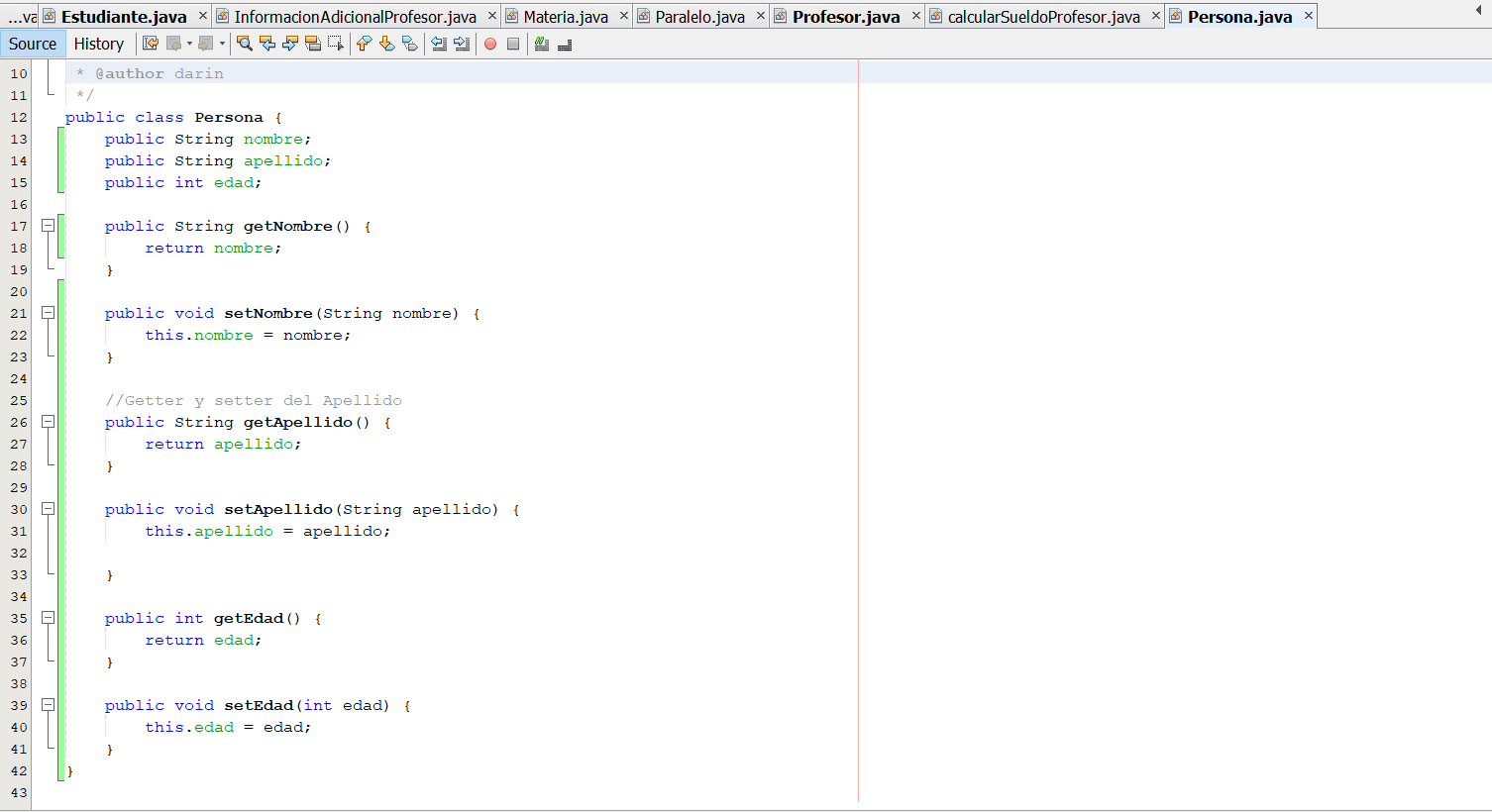
### **Consecuencia:**

Se puede identificar Duplicate Code en las clases Profesor y Estudiante, puesto que varios de sus atributos se repiten, como los atributos nombre, apellido, edad y dirección.



### **Solución:**

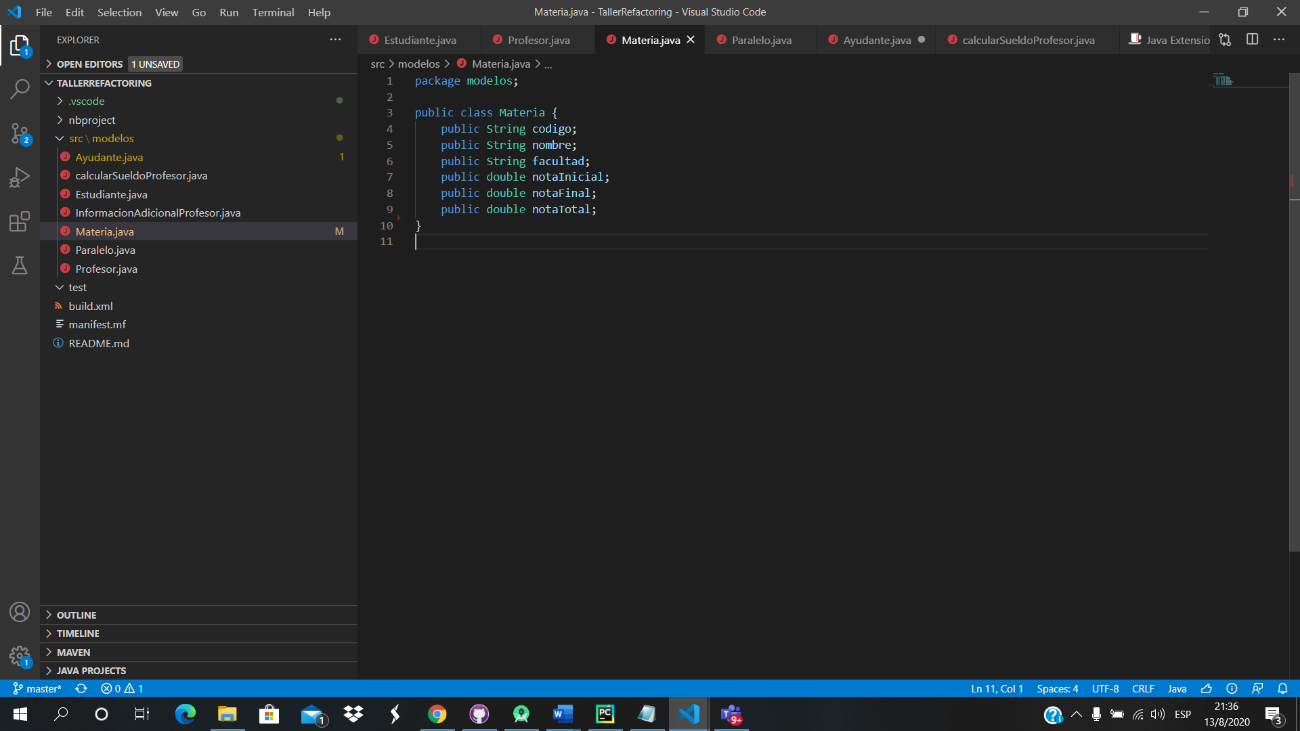
Para solucionar este code smell se utilizó extract method, creamos una nueva clase persona donde almacenará los atributos y métodos que se repetían en las clases profesor, estudiante y ayudante.



## **Code Smells #2: Clase Perezosa**

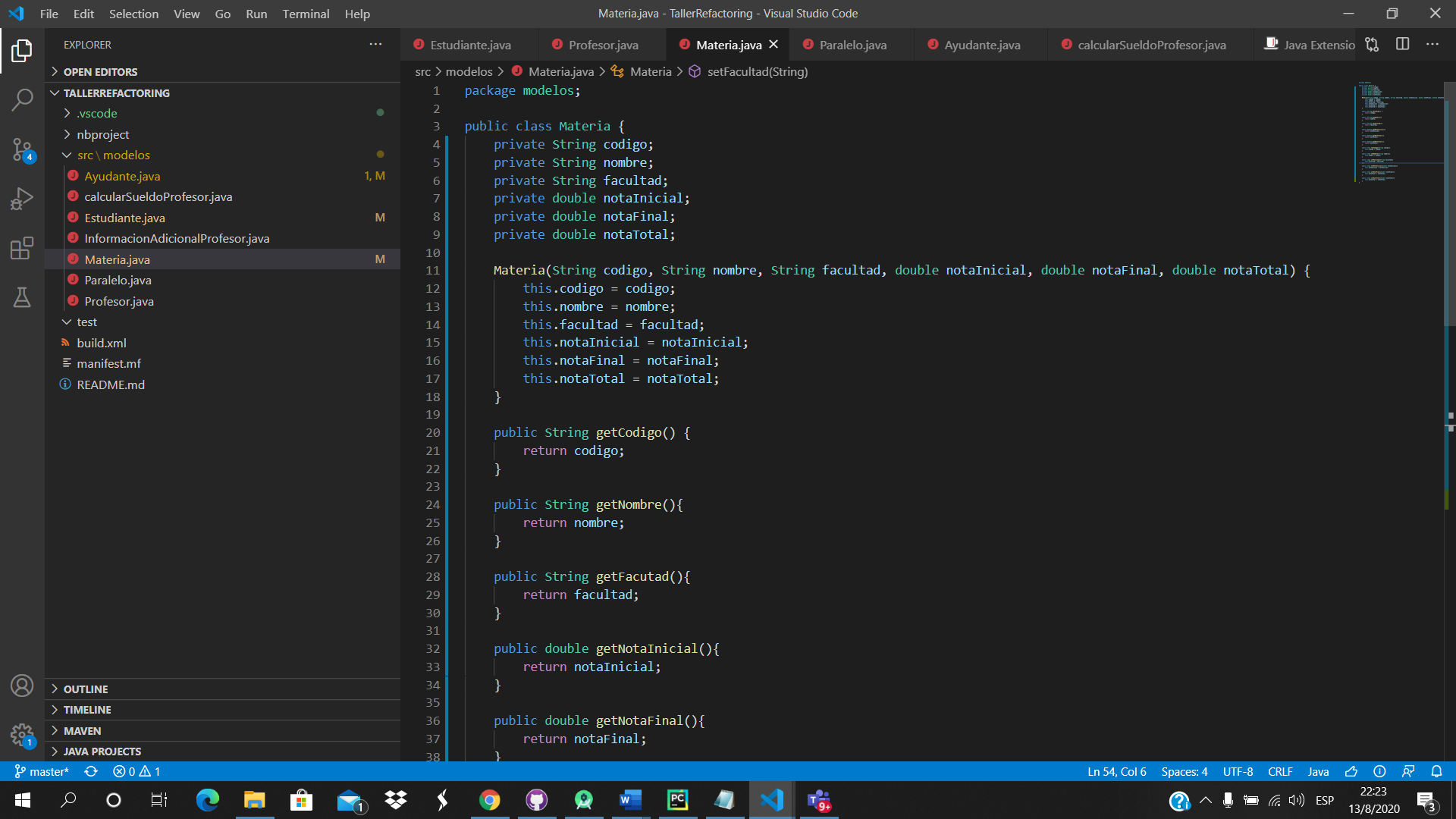
### **Consecuencia:**

Toda clase tiene un costo por ser mantenida y leída así sea mínima su construcción. La clase materia en este caso no cuenta con un constructor o métodos para poder instanciar sus atributos y obtener funcionamiento alguno dentro del sistema.



### **Solución:**

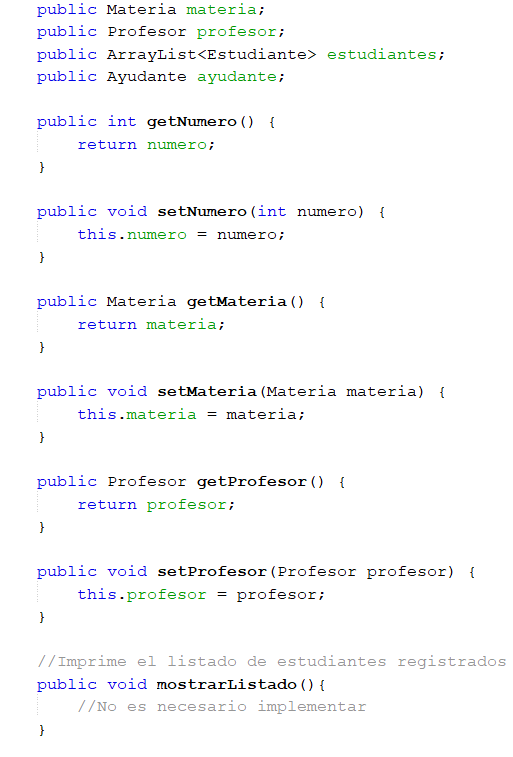
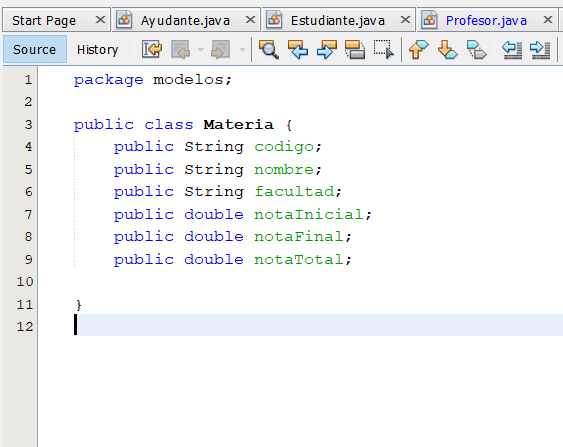
Para refactorizar el código de la clase perezosa, es necesario destruir herencias en el caso de que existan o si no existe una relación en línea, eliminarla. En esta ocasión al denotar la importancia de la clase Materia, se le dio más peso a la clase.



## **Code Smells #3: Dead Code**

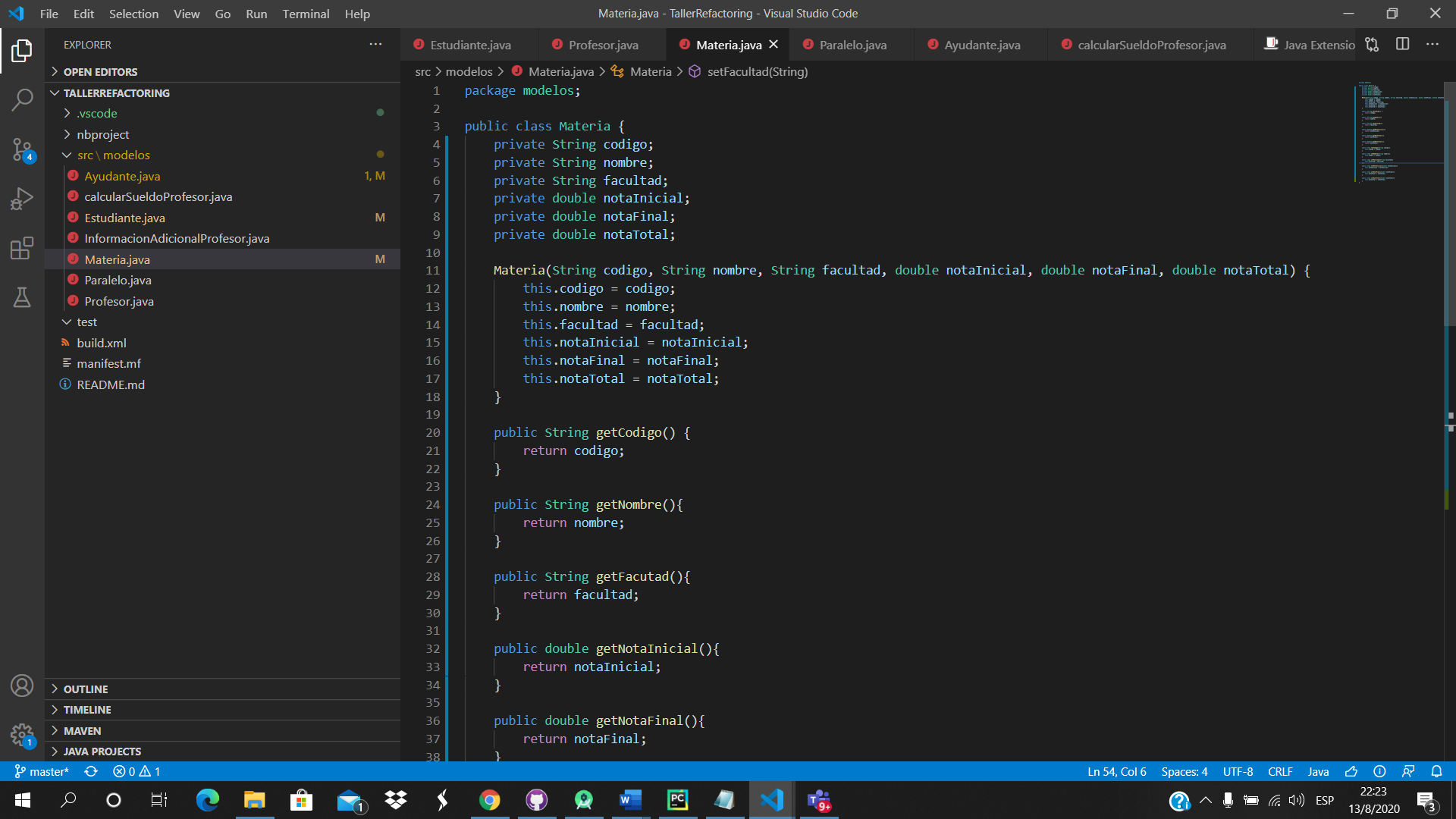
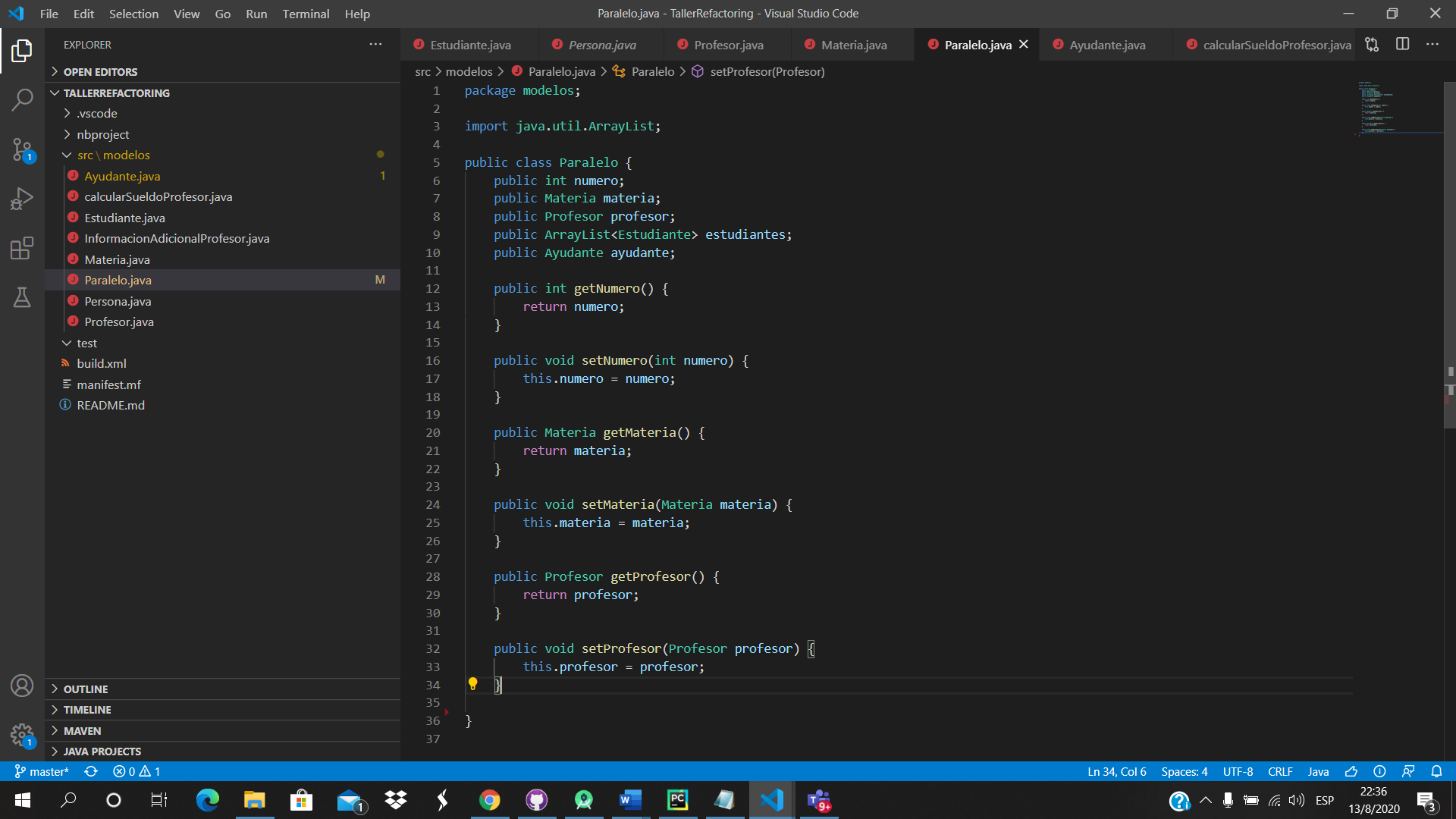
### **Consecuencia:**

Existe código sin utilizar dentro de las clases. En la clase Materia existen varios atributos no instanciados, al igual que en la clase paralelo existe un método llamado mostrarListado() el cual no cuenta con implementación



### **Solución:**

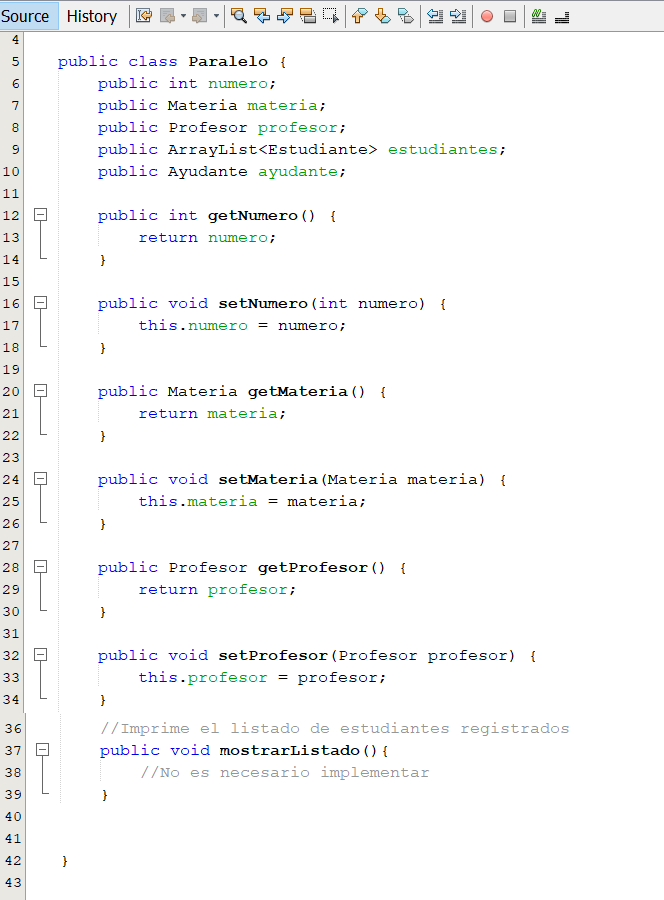
Se añadió el código necesario para que Materia tenga más responsabilidad y se eliminó el método mostrarListado() que no tenía implementación.



## **Code Smells #3: Data Class**

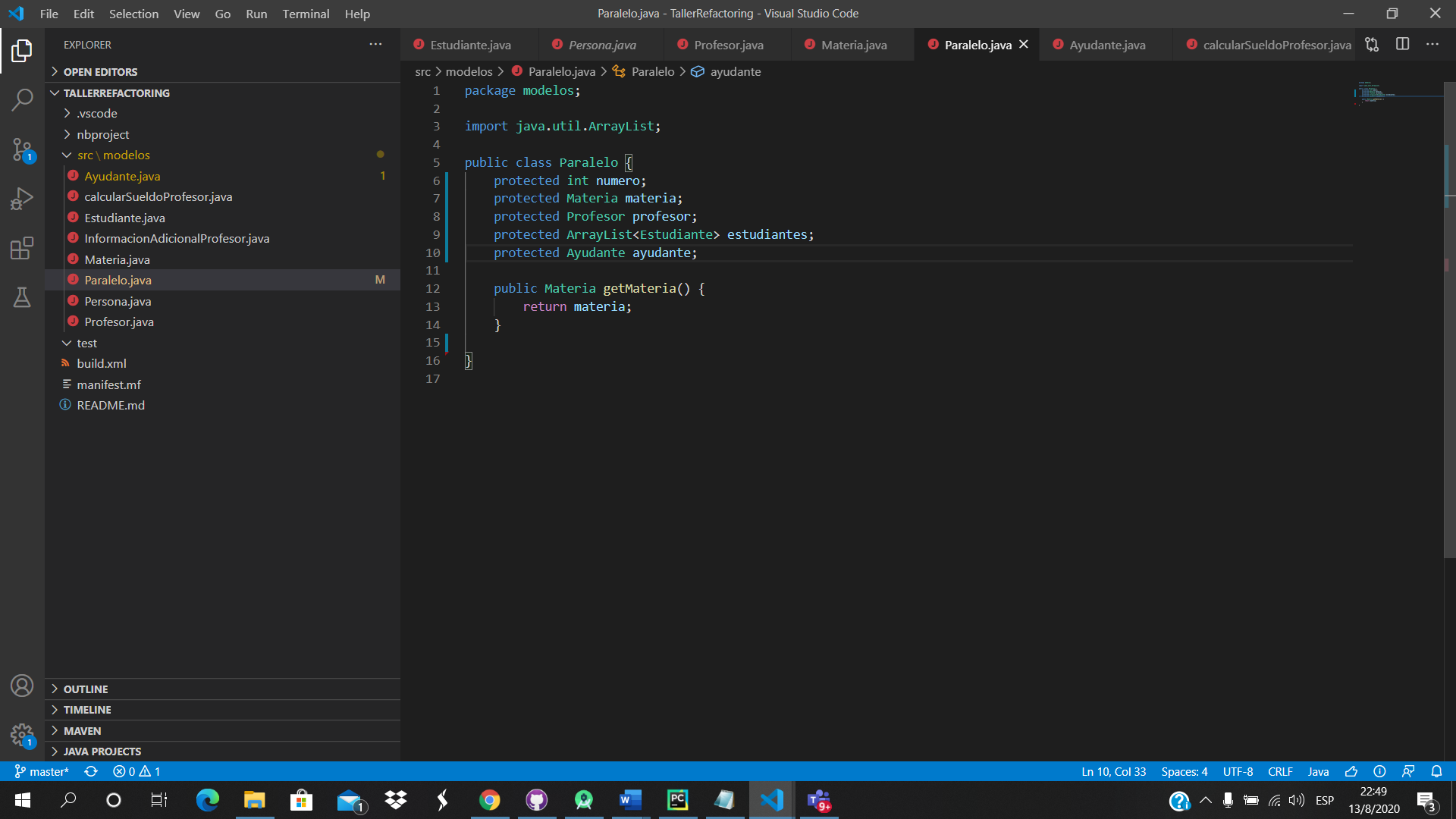
### **Consecuencia:**

El Code Smell Data class se puede identificar dentro de la clase Paralelo, puesto que esta clase solo tiene getters y setters, además de un método llamado mostrarParelo() que no es utilizado.



### **Solución:**

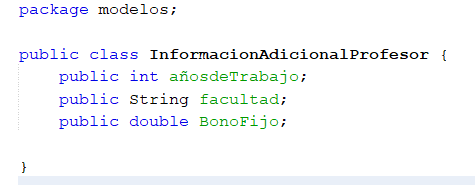
Se utilizó el método de refactorización Encapsulate Field. Para solucionar este problema se encapsularon los atributos cambiando su estado a privados.



# **Code Smells #5: Speculate Generality**

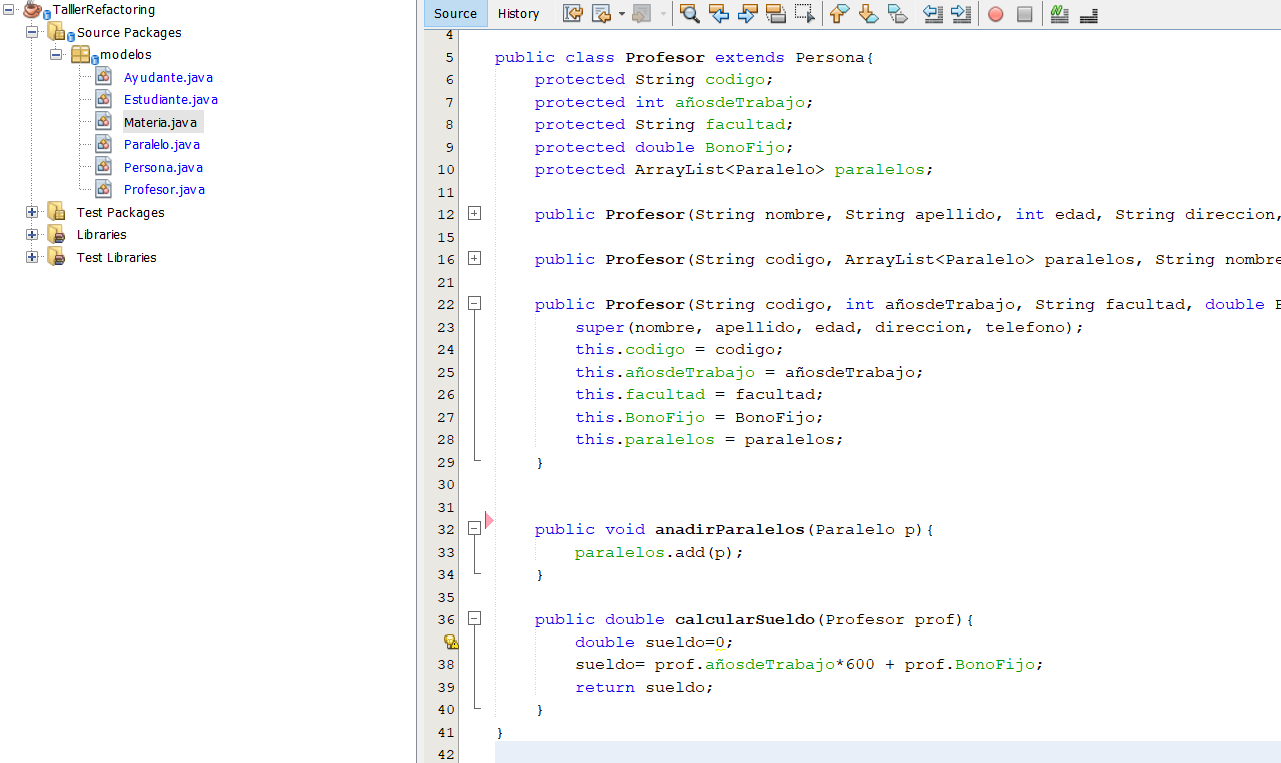
### **Consecuencia:**

Existe código que busca cubrir posibles casos futuros que no necesariamente puedan llegar a ocurrir. En la clase InformaciónAdicionalProfesor, existe un caso parecido, el nombre de la clase hacer referencia al posible caso de que se necesite agregar información adicional no requerida para los objetos de tipo Profesor.



### **Solución:**

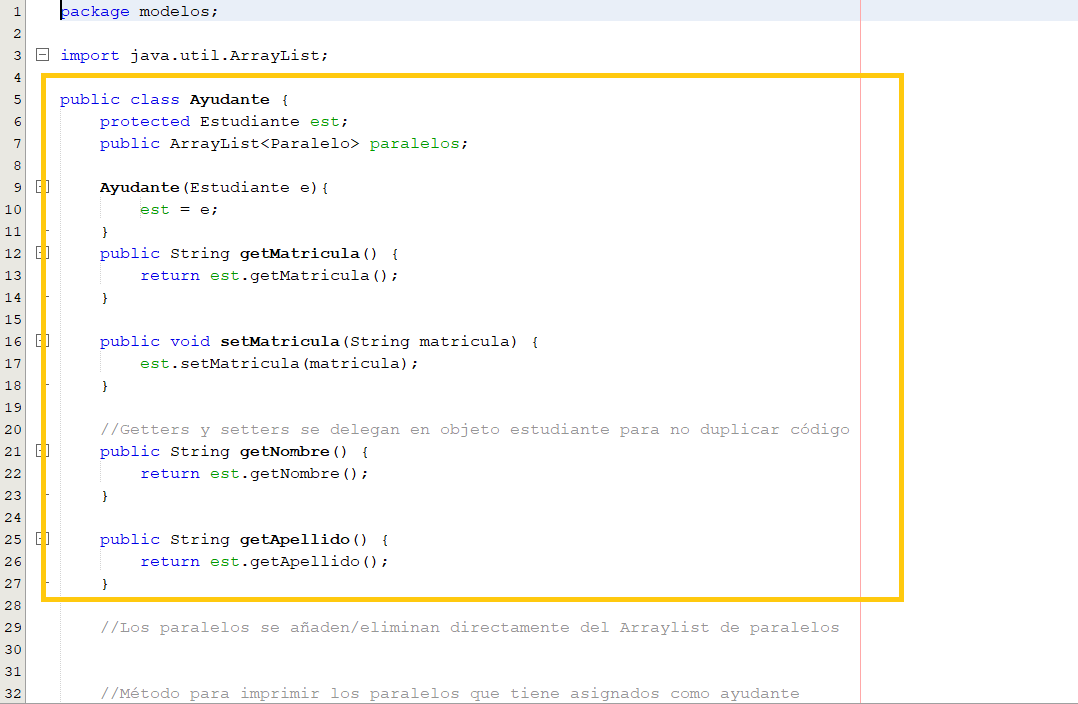
Para este caso en específico en donde la clase informaciónAdicional, posee dead code, es considerado una clase vaga y a su vez forma parte de una Speculate Generality la mejor forma de refactorizar fue Safely delete, tanto de la clase informaciónAdicional como la clase calcularSueldoProfesor moviendo sus atributos y método (respectivamente) a la clase profesor y generando un constructor para el caso de recibir la información adicional utilizada en el método calcular sueldo.



# **Code Smells #6: Feature Envy**

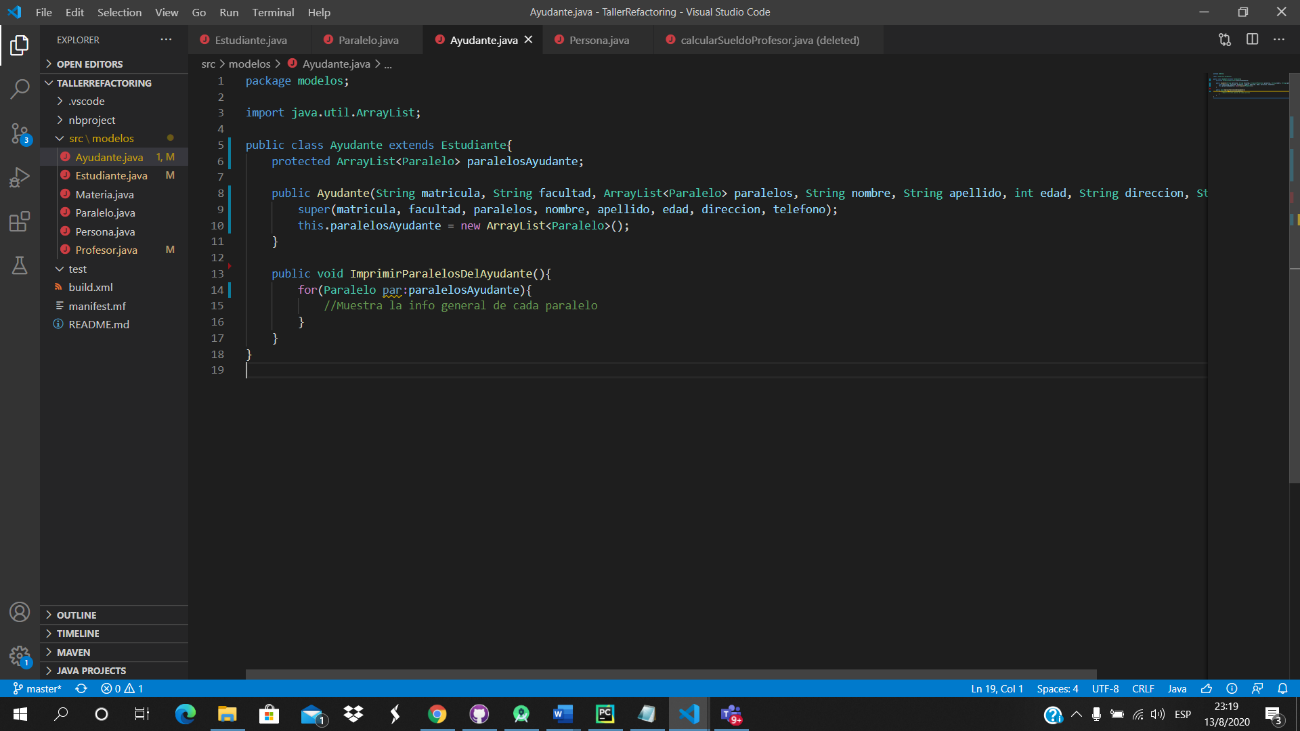
### **Consecuencia:**

Se logra identificar Feature Envy en la clase Ayudante, puesto esta clase utiliza los métodos y atributos de la clase Estudiante mucho más que sus propios métodos y atributos.



### **Solución:**

Se extendió la clase Ayudante a Estudiante para que use los métodos correspondientes de Estudiante y no envide sus métodos, ya que anteriormente usaba más los métodos de Estudiante que la misma.



# **Code Smells #7: Comments**

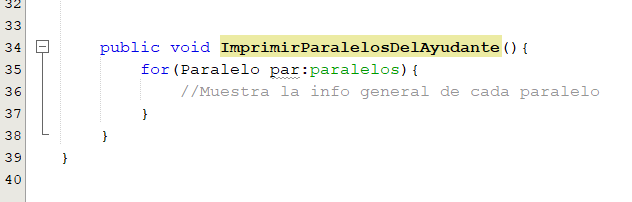
### **Consecuencia:**

Se puede identificar el Code Smell comments en la clase Ayudante. Los comentarios realizados ayudan a que el programados pueda entenderse, lo que no debería ser así puesto que el código debe ser intuitivo y no necesitar de códigos.



### **Solución:**

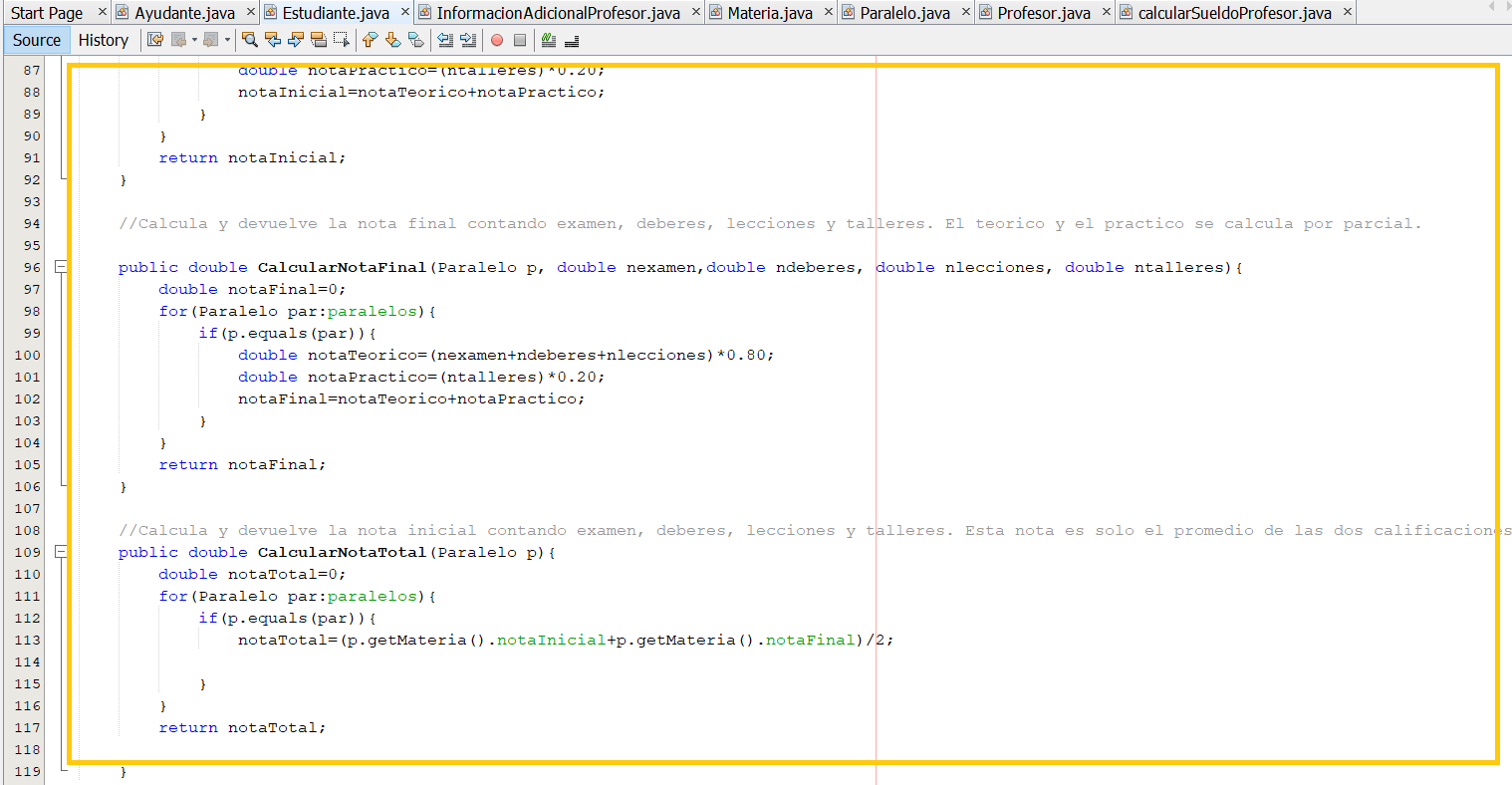
Para solucionar este Code Smell se debe utilizar el tipo refactoring Rename Method, así el nombre del método puede dar a entender al programador para que va a servir.



# **Code Smells #8: Long Class**

### **Consecuencia:**

Dentro de la clase Estudiante se puede identificar el Code Smell Long class, está clase es muy larga, tiene muchos métodos y trata de hacer muchas acciones las cuales algunas pueden ser realizadas por otra clase.



### **Solución:**

Para solucionar este Code Smell se puede utilizar el refactoring llamado Extract Class, la cual separa la clase Estudiante en dos, en este caso se creó una nueva clase Persona que almacena varios métodos y atributos de estudiantes, haciendo que la clase Estudiante sea más pequeña.

