

Materia: Diseño de Software

Taller Refactoring

Grupo # 9

Integrantes:

* Alex Villegas
* Ysrael Larco
* Guillermo Veintimilla

Contenido

[Code Smell: Data Class 3](#_Toc60956539)

[Code Smell: Inappropiate intimacy 4](#_Toc60956540)

[Code Smell: Data Clumps 5](#_Toc60956541)

[Code Smell: Temporary Field 6](#_Toc60956542)

[Code Smell: Dead Code 7](#_Toc60956543)

[Code Smell: Lazy Class 8](#_Toc60956544)

[Code Smell: Primitive Obsession 9](#_Toc60956545)

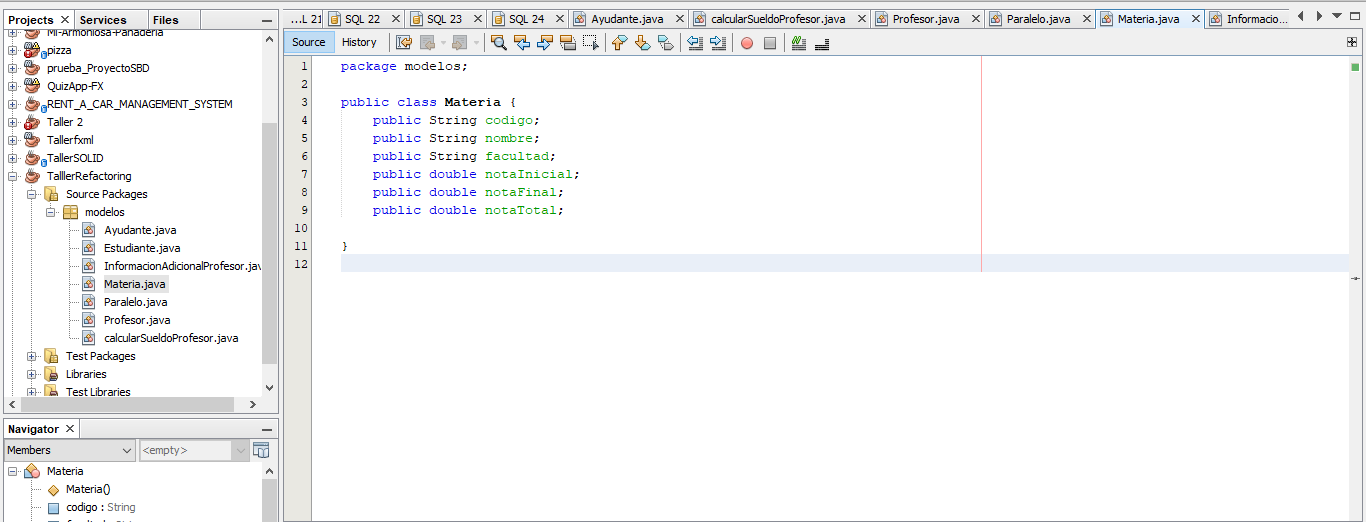
[Code Smell: Duplicated Code 10](#_Toc60956546)

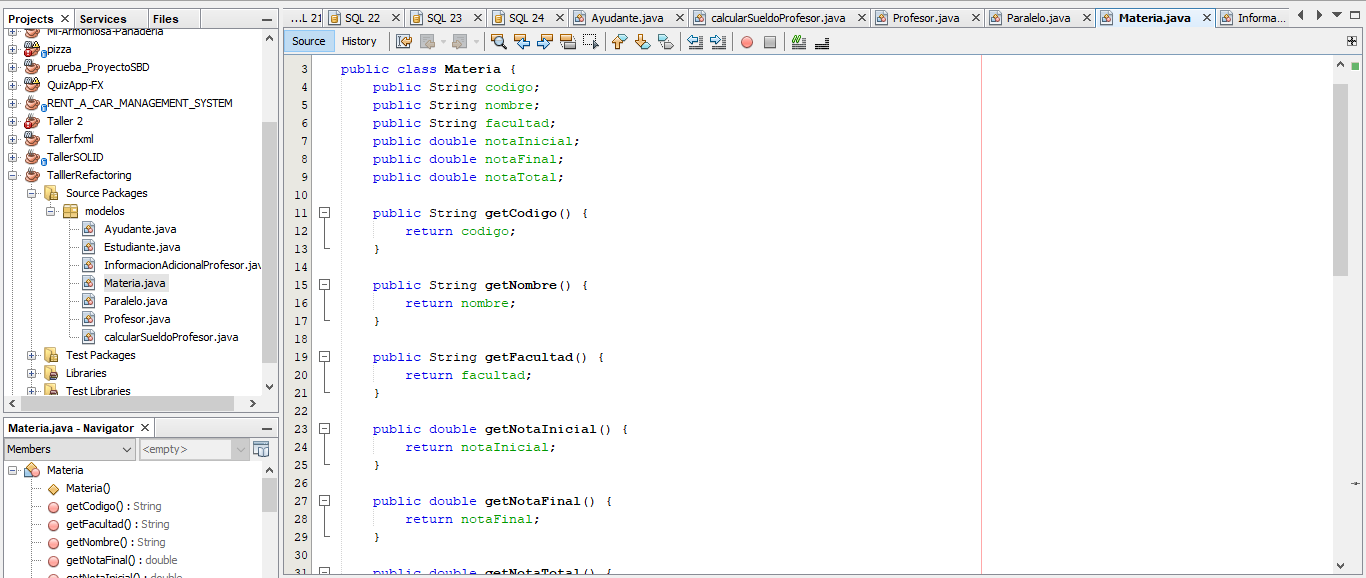
# Code Smell: Data Class

**Consecuencias:** En sí, mantener esta clase no afecta al funcionamiento normal del programa, sin embargo, se vuelve inútil al no contener ni getters ni setters o comportamiento alguno. Por lo tanto, estos deberían ser implementados ya que una característica principal de los objetos es el poder contener tipos de comportamientos u operaciones.

**Técnica de Refactorización para utilizar:** Encapsulate Field

Con esta técnica se estarían creando los getters y setters para la clase “Materia” con el fin de que exista encapsulación para que puedan ocultar sus datos de otros objetos y con esto, el código se vuelve más fácil de mantener.

Código anterior:

Código actualizado:

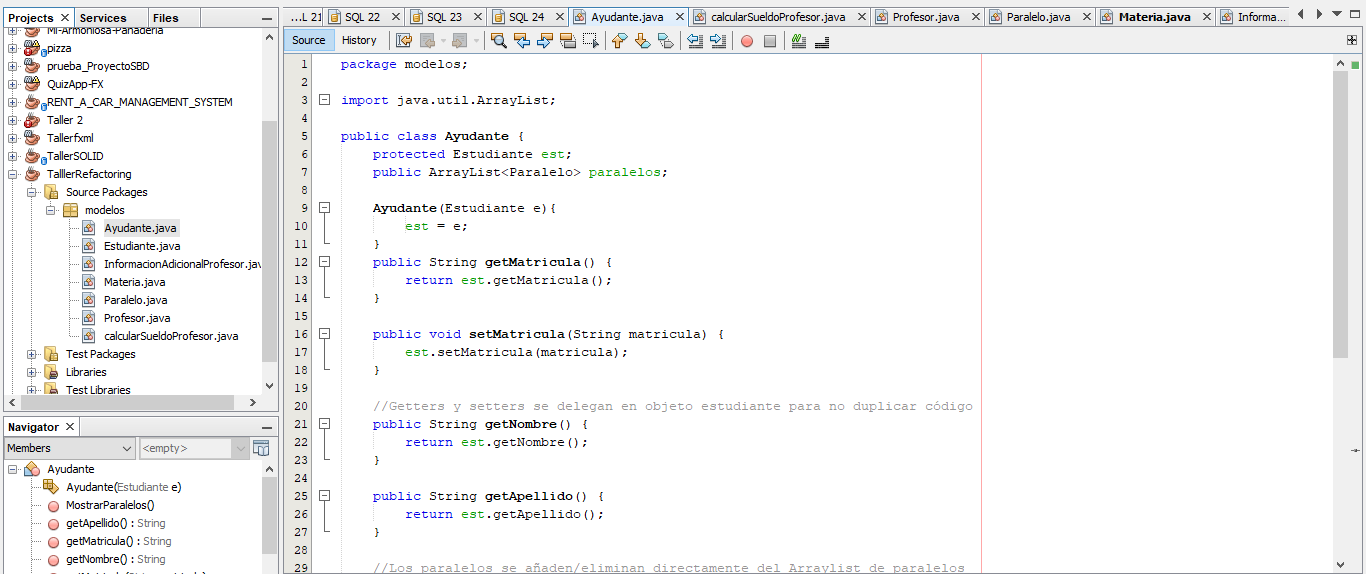
# Code Smell: Inappropiate intimacy

**Consecuencias:** En este caso, la clase “Ayudante” al tener como atributo un objeto de la clase “Estudiante”, accede a los datos sensibles de “Estudiante” y provoca que exista un gran acoplamiento entre las dos, es decir, si se realiza un cambio en “Estudiante” también se tendrán que aplicar cambios a “Ayudante”. Así, las clases se vuelven difíciles de reutilizar y mantener.

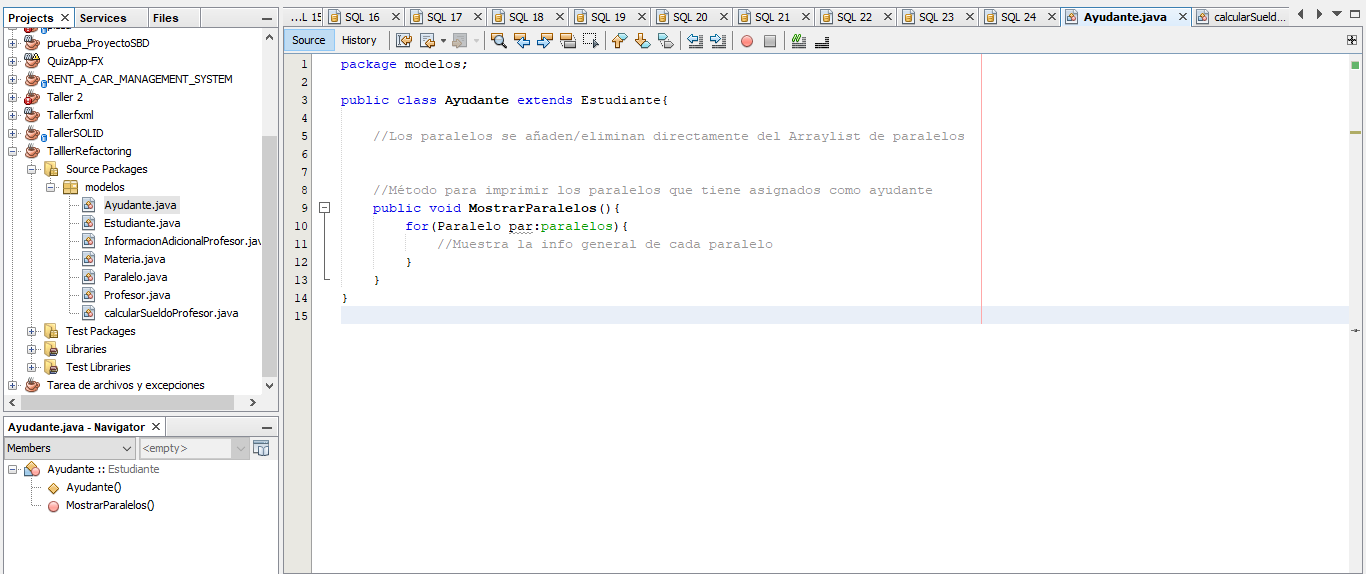
**Técnica de Refactorización para utilizar:** Replace delegation with inheritance

Esta técnica permitiría que la clase “Ayudante” herede de la clase “Estudiante” haciendo innecesarios los métodos de delegación en “Ayudante” y reduciendo el tamaño del código.

Código anterior:



Código actualizado:



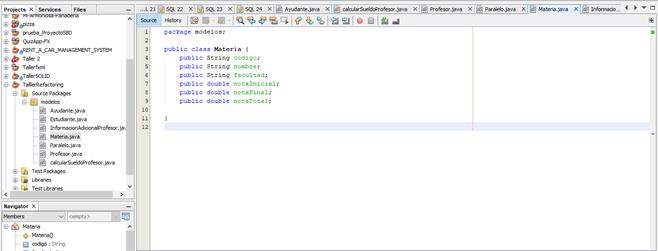
# Code Smell: Data Clumps

**Consecuencias:** Tener código repetido e idéntico en distintas partes del código representa problemas ya que se tiene una clase innecesariamente larga, y como no se asegura si algunas variables tienen relación entre si esto puede provocar inconvenientes al momento de manipularlas.

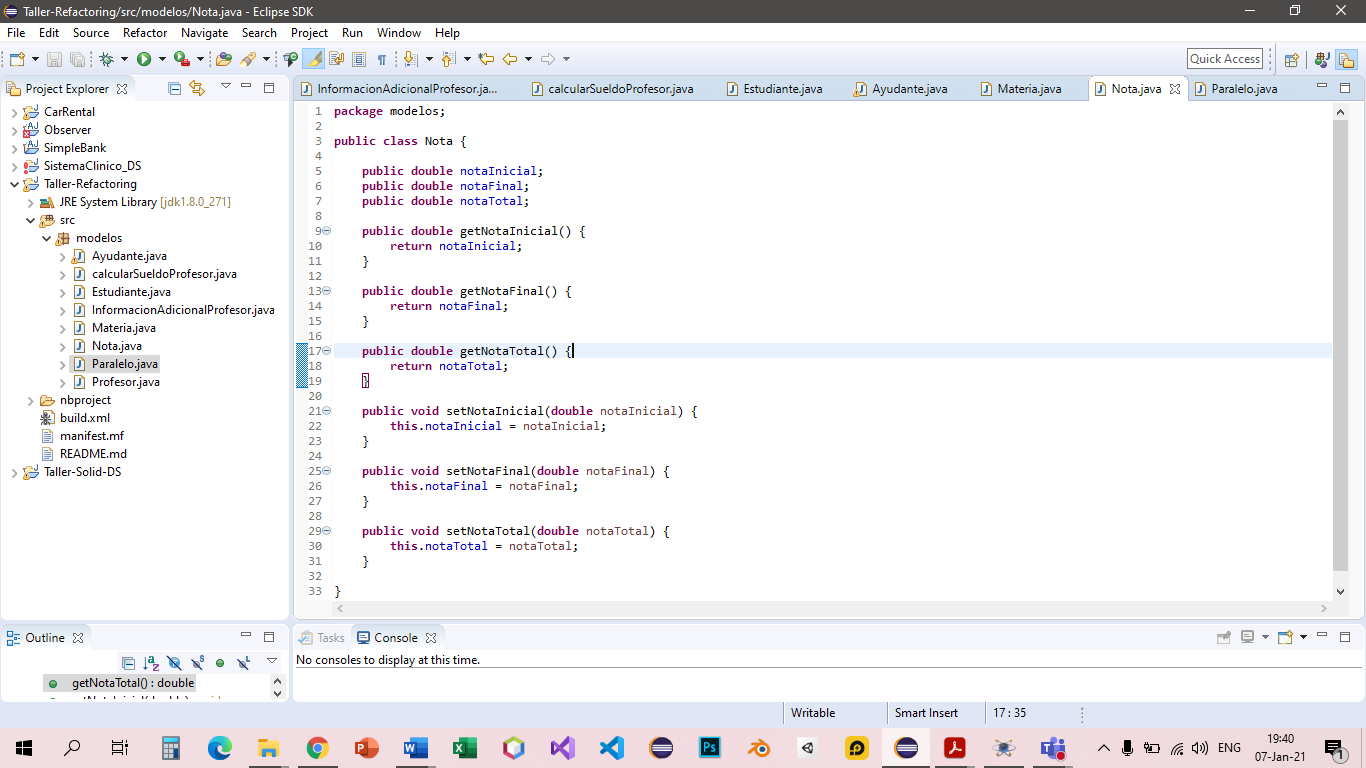
**Técnica de Refactorización para utilizar:** Extrac Class

Esta técnica permitiría separar la variable notainicial y notafinal en su propia clase asi se podrá editarla de una manera más eficiente sin que este presente problemas estando en otra clase como variable.

Codigo Anterior:



Codigo Nuevo:



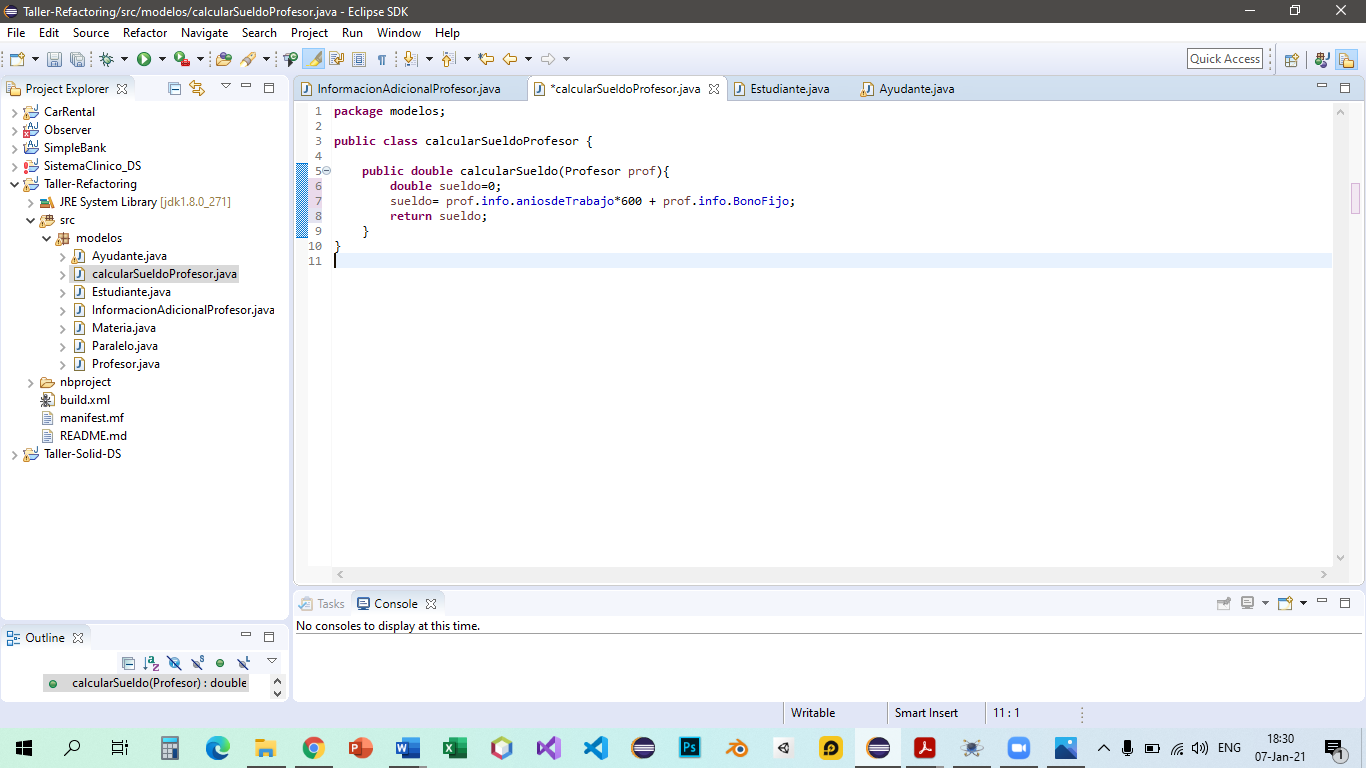
# Code Smell: Temporary Field

**Consecuencias:** Es una variable que solo se usa una vez usando recursos de manera innecesaria y aumenta líneas de código sin justificación.

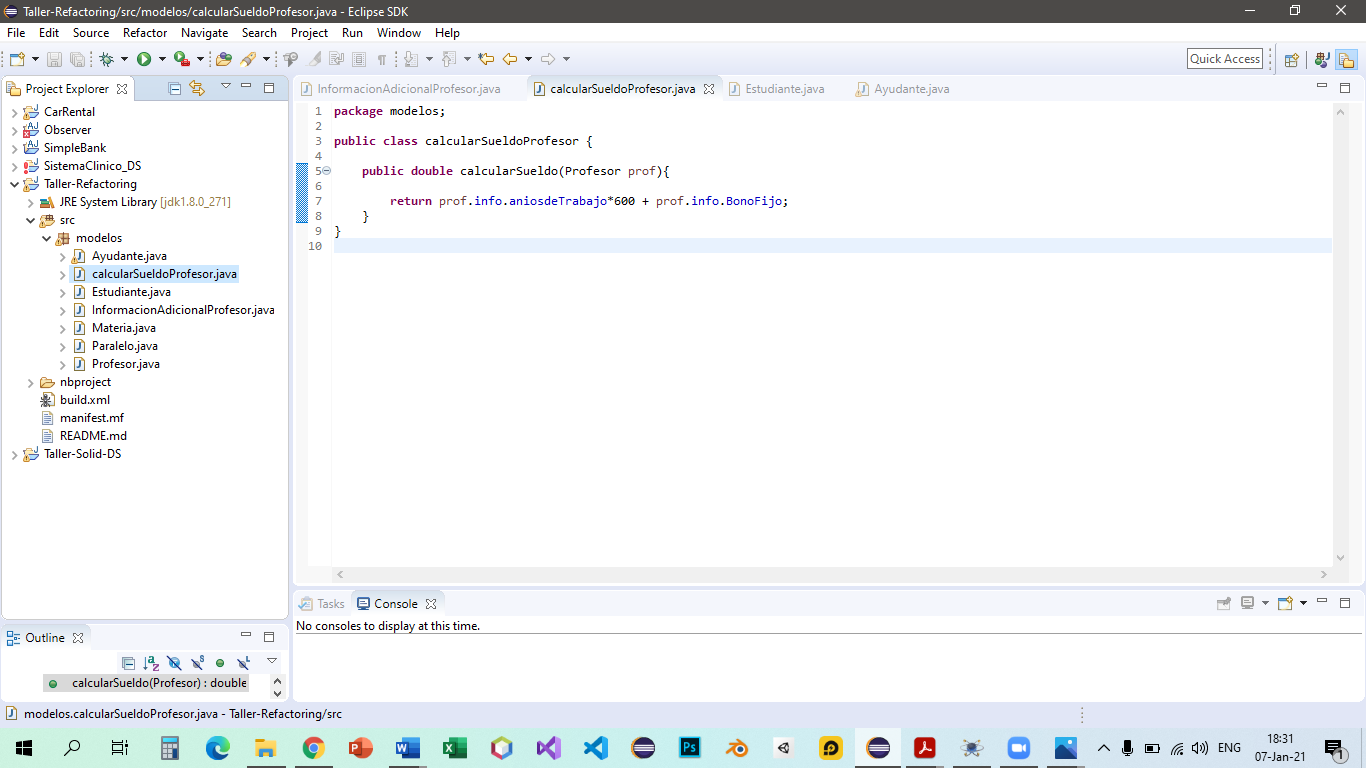
**Técnica de Refactorización para utilizar:** Inline Temp.

Esta técnica permitiría obtener el resultado de la expresión de manera directa y sin desperdiciar recursos.

Codigo Anterior:



Codigo Nuevo.



# Code Smell: Dead Code

Este code smell ocurre cuando los requisitos del software han cambiado o se han realizado correcciones, nadie tuvo tiempo de limpiar el código antiguo.

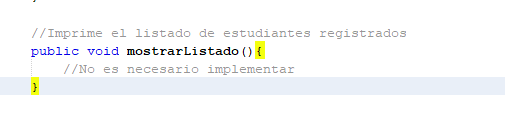
Dicho código también se puede encontrar en condicionales complejos, cuando una de las ramas se vuelve inalcanzable (debido a un error u otras circunstancias).

**Consecuencias:** Provoca que el código sea más complicado de leer, que las clases sean más largas y complejas.

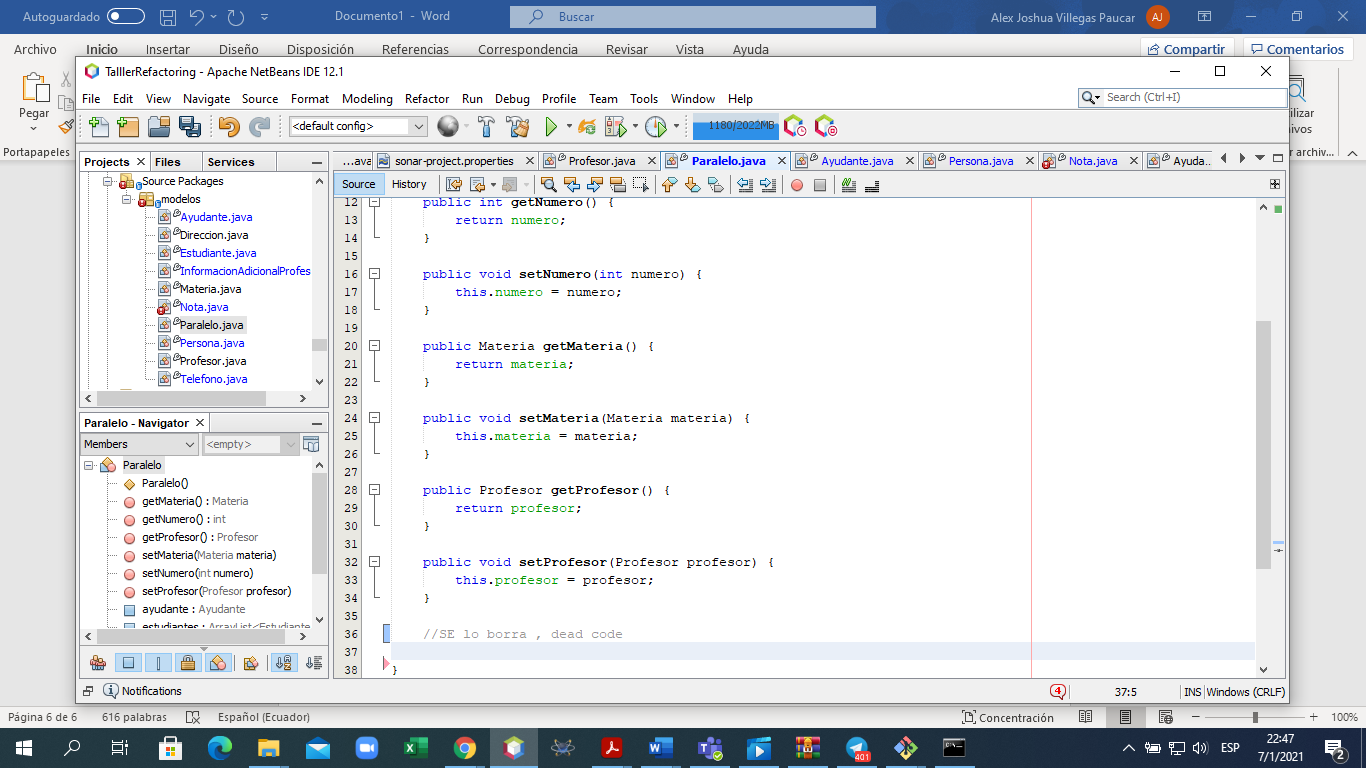
**Técnica de refactorización:**

Como los métodos no son usados en ninguna clase se pueden eliminar sin generar problemas en ninguna otra clase.

Código anterior



Código actualizado



# 

# Code Smell: Lazy Class

Este code smell ocurre cuando una clase fue diseñada para ser completamente funcional, pero después de algunas de las refactorizaciones, se ha vuelto ridículamente pequeña.

O quizás fue diseñado para apoyar el trabajo de desarrollo futuro que nunca se llevó a cabo.

**Consecuencias:** Las consecuencias de tener una Lazy Class es que los costos de mantenimiento pueden elevarse, ya que habrá más código que leer, además que quitará tiempo.

**Técnicas de refactorización a usarse:**

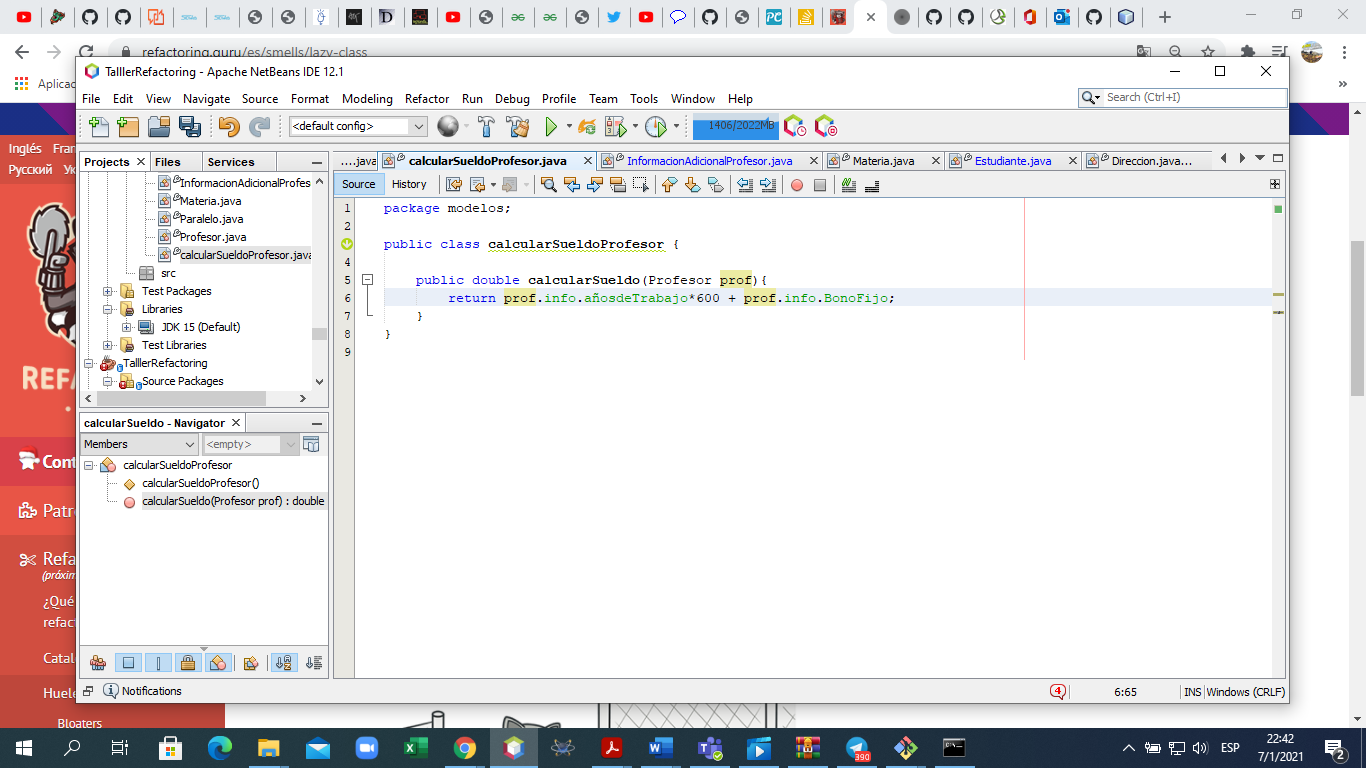
**Inline Class**

Para solucionar el code smell se utilizará el método Inline Classlo cual nos permitirá unir una clase con otra, en este caso podremos eliminar la clase CalcularSueldoProfesor.

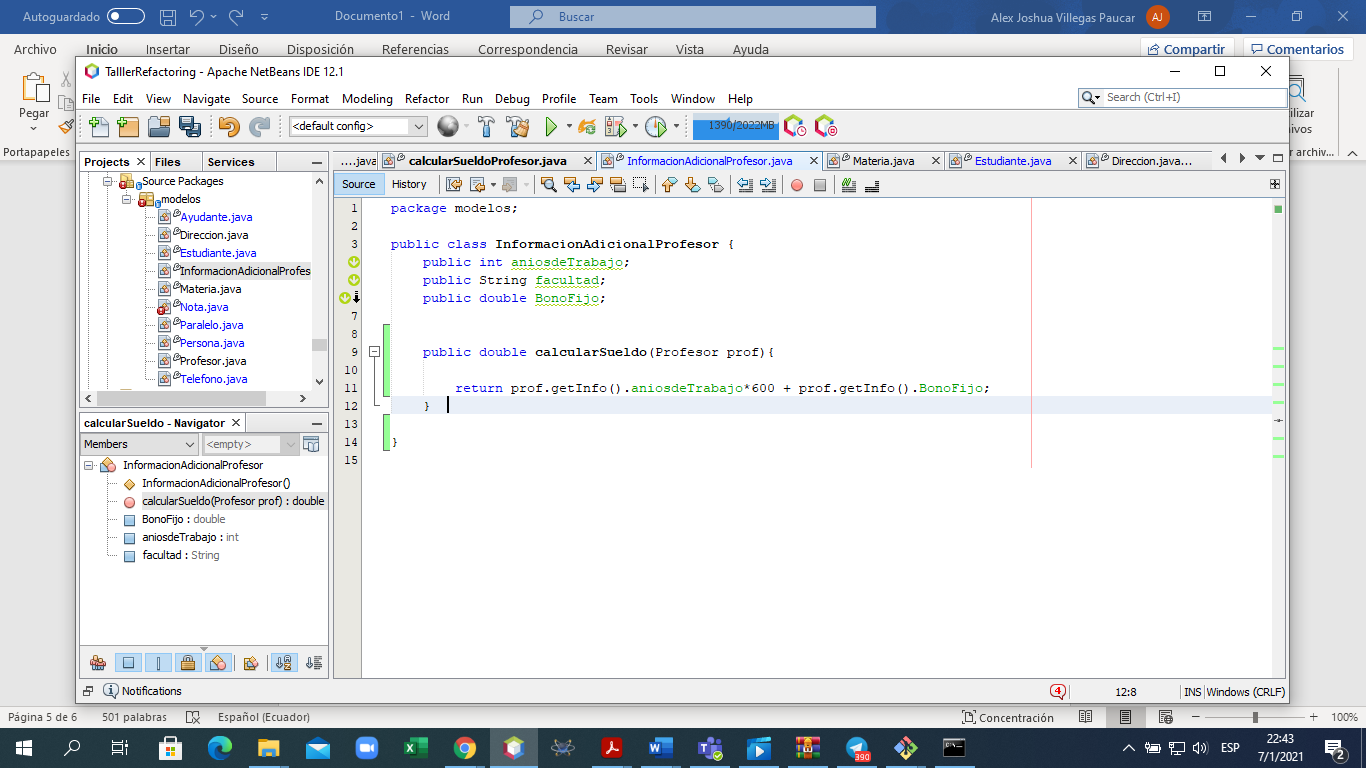
**Move Method**

Nos ayudará a mover el método que contenía la clase CalcularSueldoProfesor a la clase InformacionAdicionalProfesor**.**

Código Anterior



Código actualizado



# Code Smell: Primitive Obsession

Este code smell aparece cuando el código se basa demasiado en primitivas. Significa que un valor primitivo controla la lógica en una clase y este valor no es seguro para tipos. Por lo tanto, la obsesión primitiva es cuando tienes una mala práctica de usar tipos primitivos para representar un objeto en un dominio.

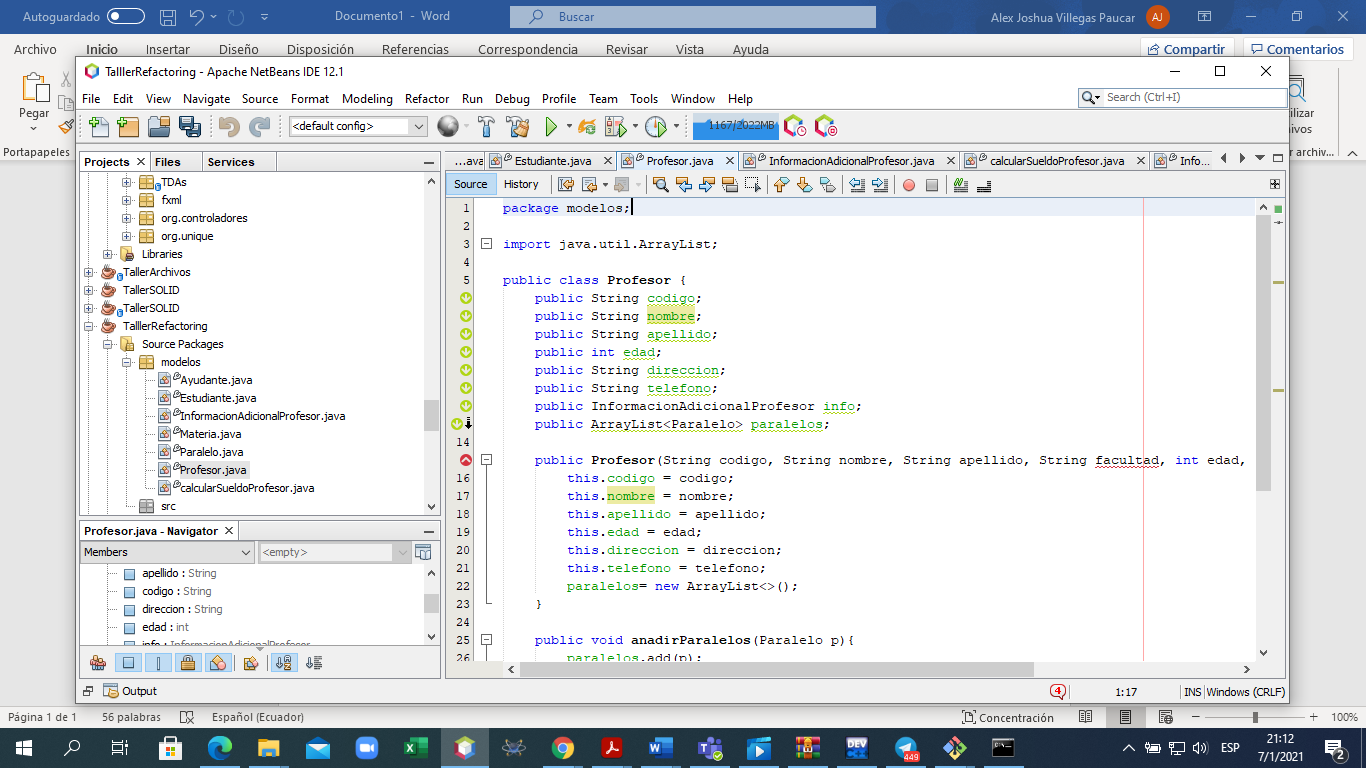
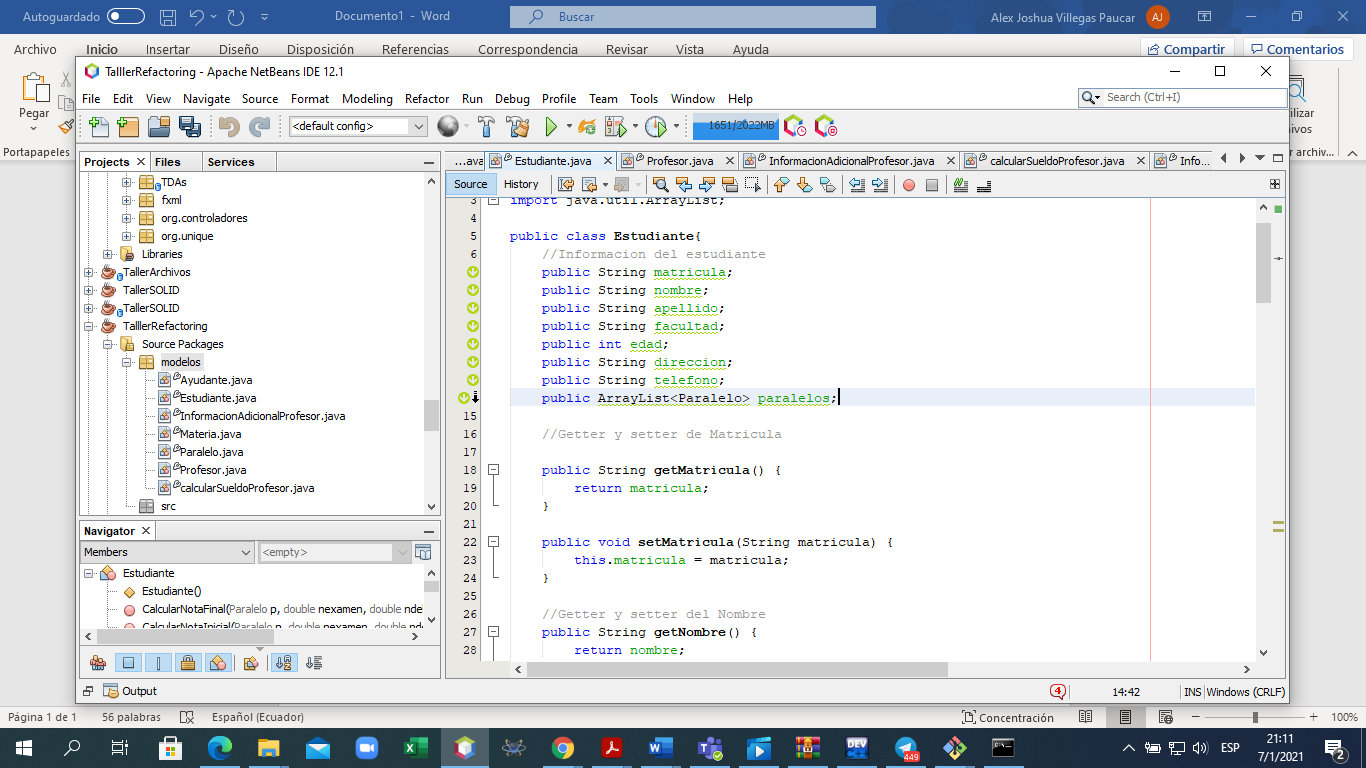
**Consecuencias:** La consecuencia de mantener a las variables dirección y teléfono como primitivas, es que no permite describir el comportamiento adecuado de esos atributos, pues el tipo primitivo no le brinda una adecuada descripción a su data.

**Técnicas de refactorización a usarse:**

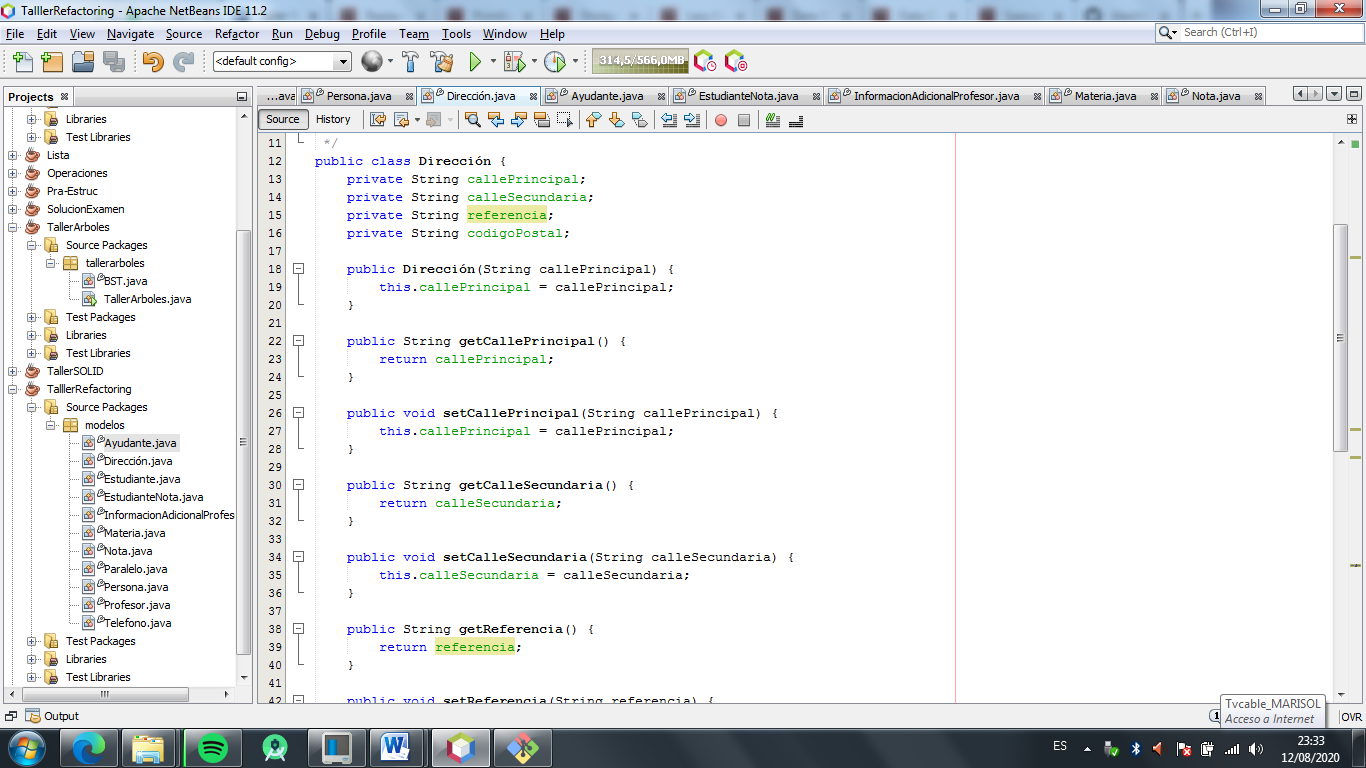
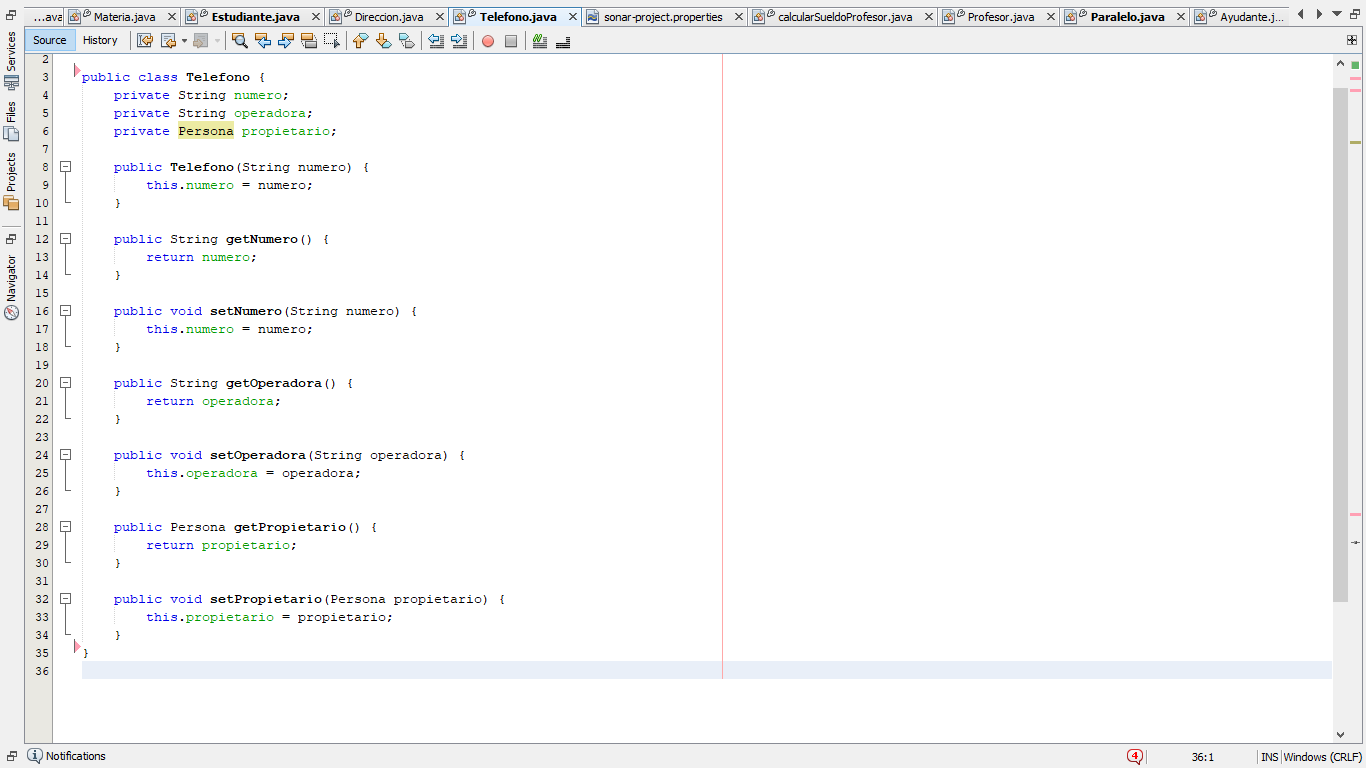
**Replace Data Value with Object:**

Como ambos atributos no cumplen con un comportamiento adecuado, se crean clases de con un nombre descriptivo y se agregan las funcionalidades descriptivas y adecuadas de cada una de las clases.

Código anterior



Código actualizado



# Code Smell: Duplicated Code

Este code smell ocurre cuando varios programadores están trabajando en diferentes partes del mismo programa al mismo tiempo. Dado que están trabajando en diferentes tareas, es posible que no sepan que su colega ya ha escrito un código similar que podría reutilizarse para sus propias necesidades.

**Consecuencias:** Tener atributos de clases repetidos, causan que las clases tengan muchos atributos y por lo tanto sea una clase larga, sea mucho más difícil de mantener, menos mantenible y tenga un mayor costo de mantenibilidad.

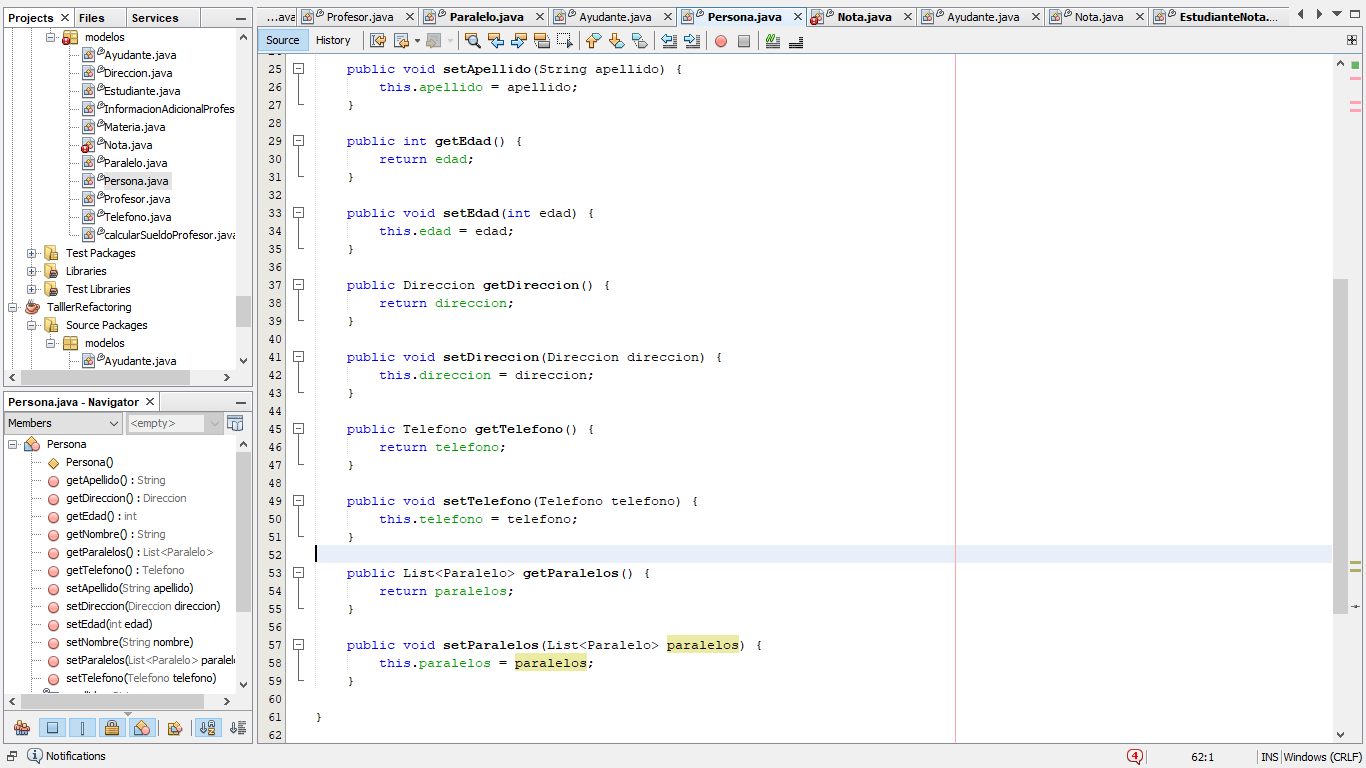
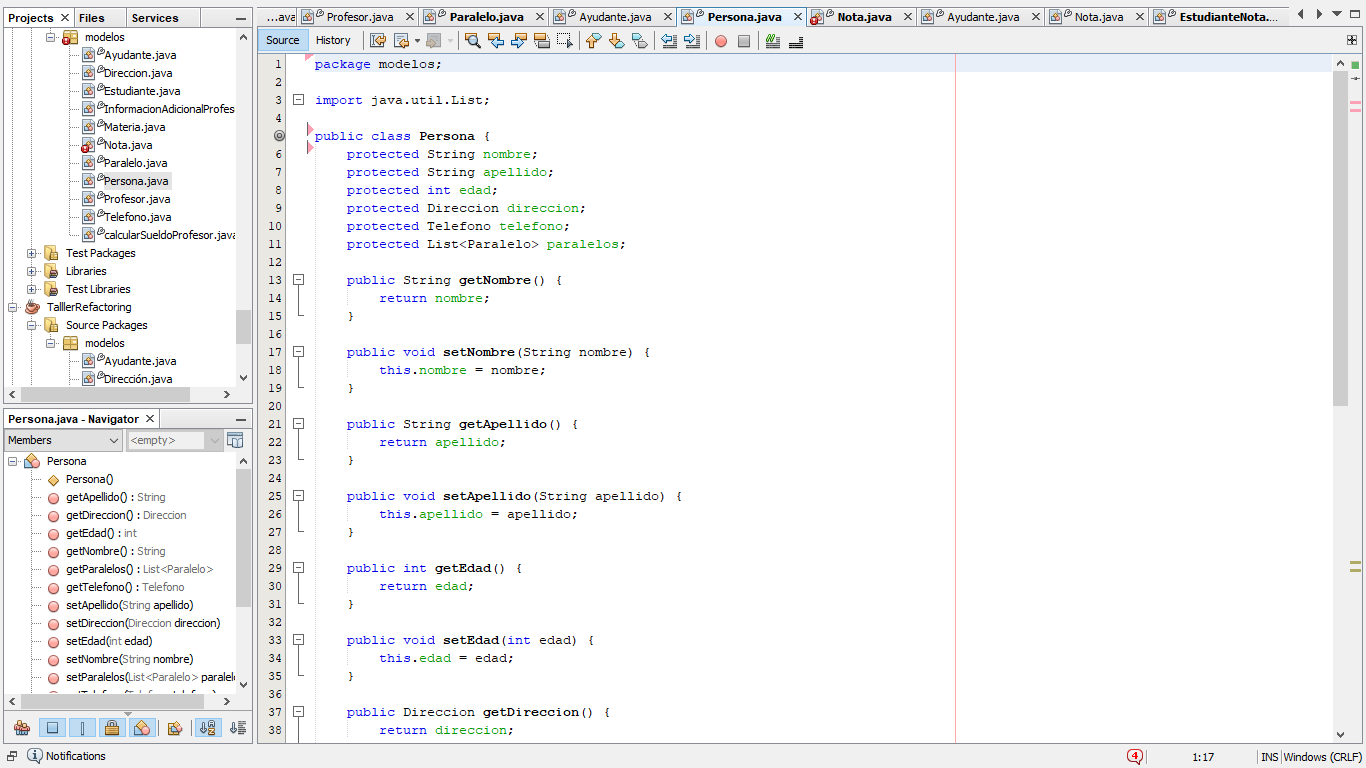
**Técnicas de refactorización a usarse:**

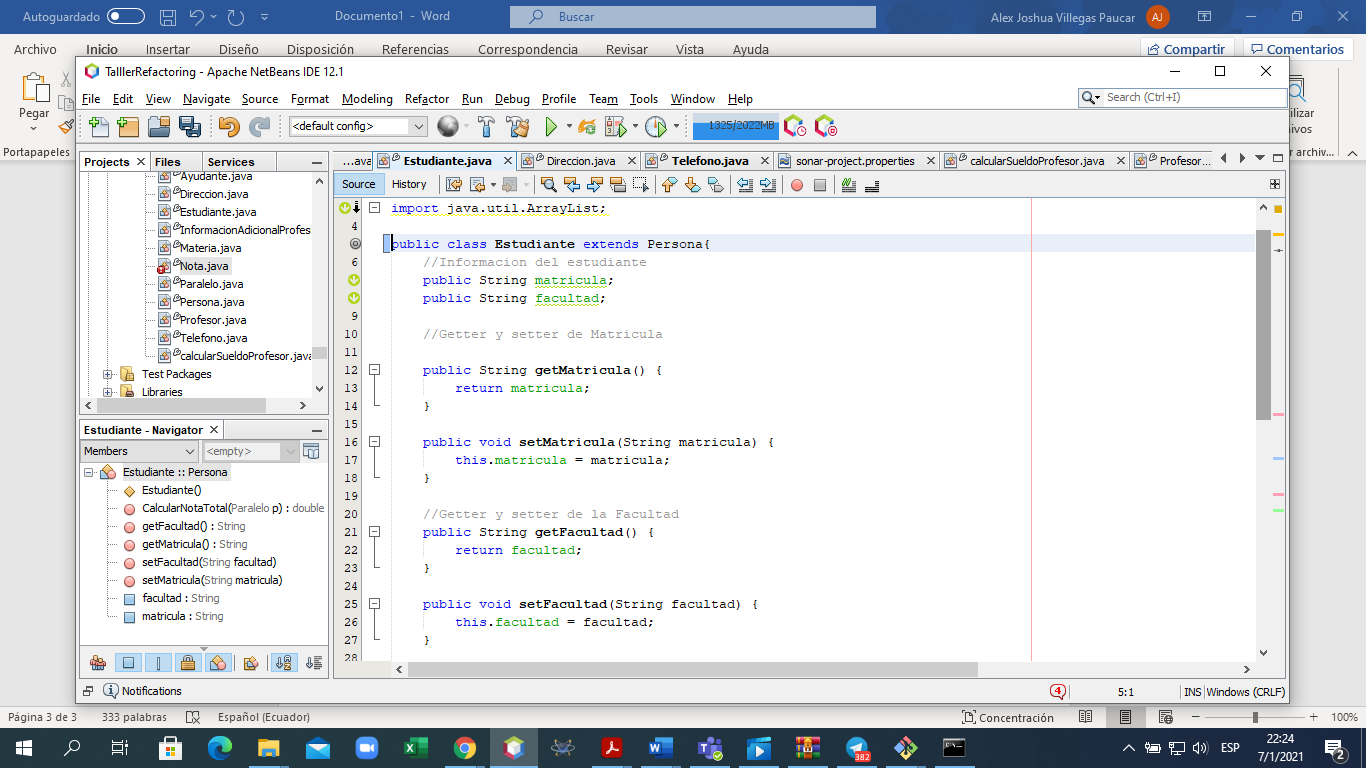
**Extract Superclass.**

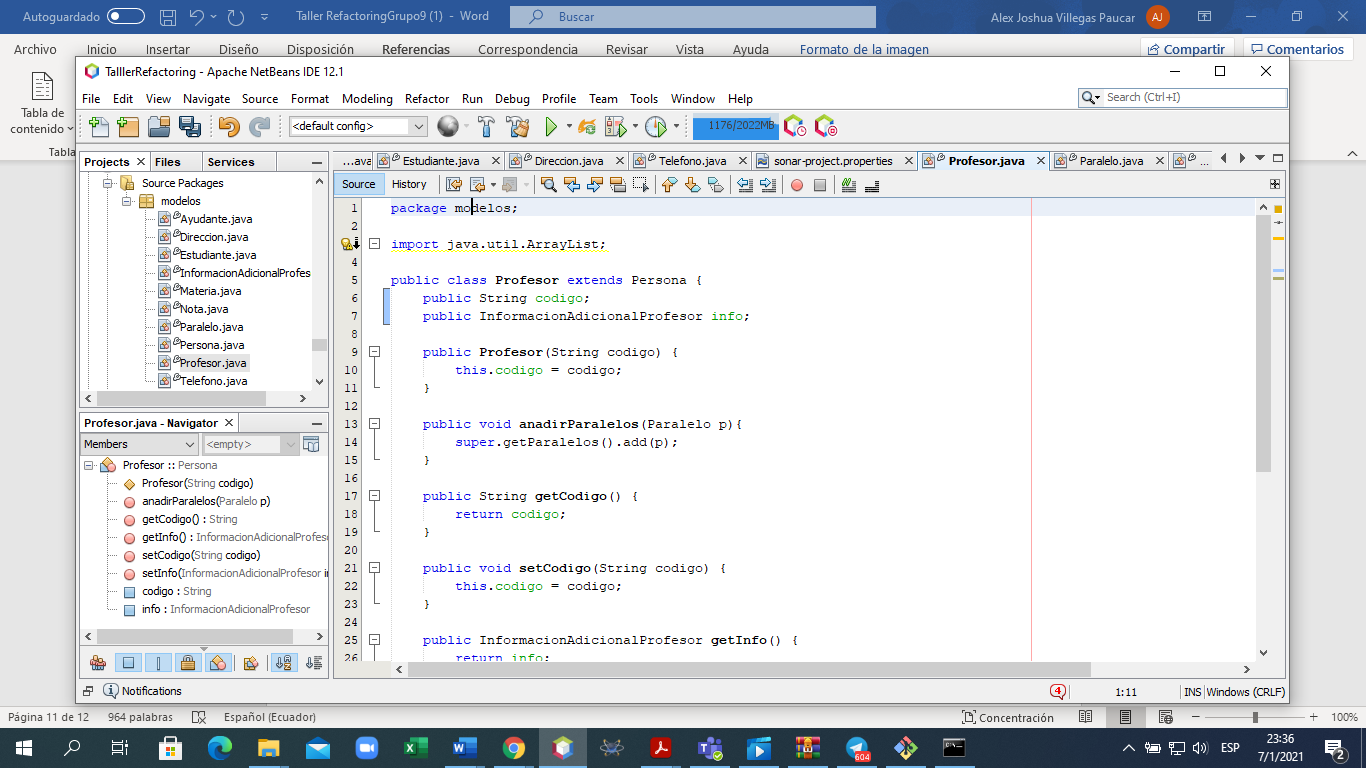
Como ambas clases tienen atributos similares, se debe crear una nueva clase que contenga a todos esos atributos similares, y hacer que ambas clases hereden de esa nueva clase.

En este caso se creó a la clase Persona que es la encargada de tener estos atributos en común, los ponemos con un modificador de acceso protected , con sus getters y setters y Profesor y Estudiante heredan de la clase Persona, y mantienen los atributos que no heredan de Persona.

Código anterior

****

Código actualizado



Adicionalmente se usa la técnica de Encapsulate field para poner en privado los atributos de las clases estudiante y profesor, y se crean los getters y setters en la clase profesor.

