**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

**FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN**

**DISEÑO DE SOFTWARE**

**Taller de Refactoring - I TÉRMINO 2020**

**Taller 8: Refactoring**

**Integrantes**

Pratt Garcia

Steven Choez

Kevin Castro

**Paralelo**

102

**Docente**

David Alonso Jurado Mosquera

**Fecha**

12 de Agosto del 2021

Tabla de contenido

**Objetivo3**

Objetivo3

Especificaciones3

Descripción3

**Sección A: Identificar Code Smells**4

Comments4

Comments5

Duplicate Code6

Dead Code7

Data Class8

Data Class9

Data Clumps10

Data Clumps11

Speculative Generality12

Inappropiate Intimacy13

**Sección B: Identificar Code Smells14**

**Objetivos**

* 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.

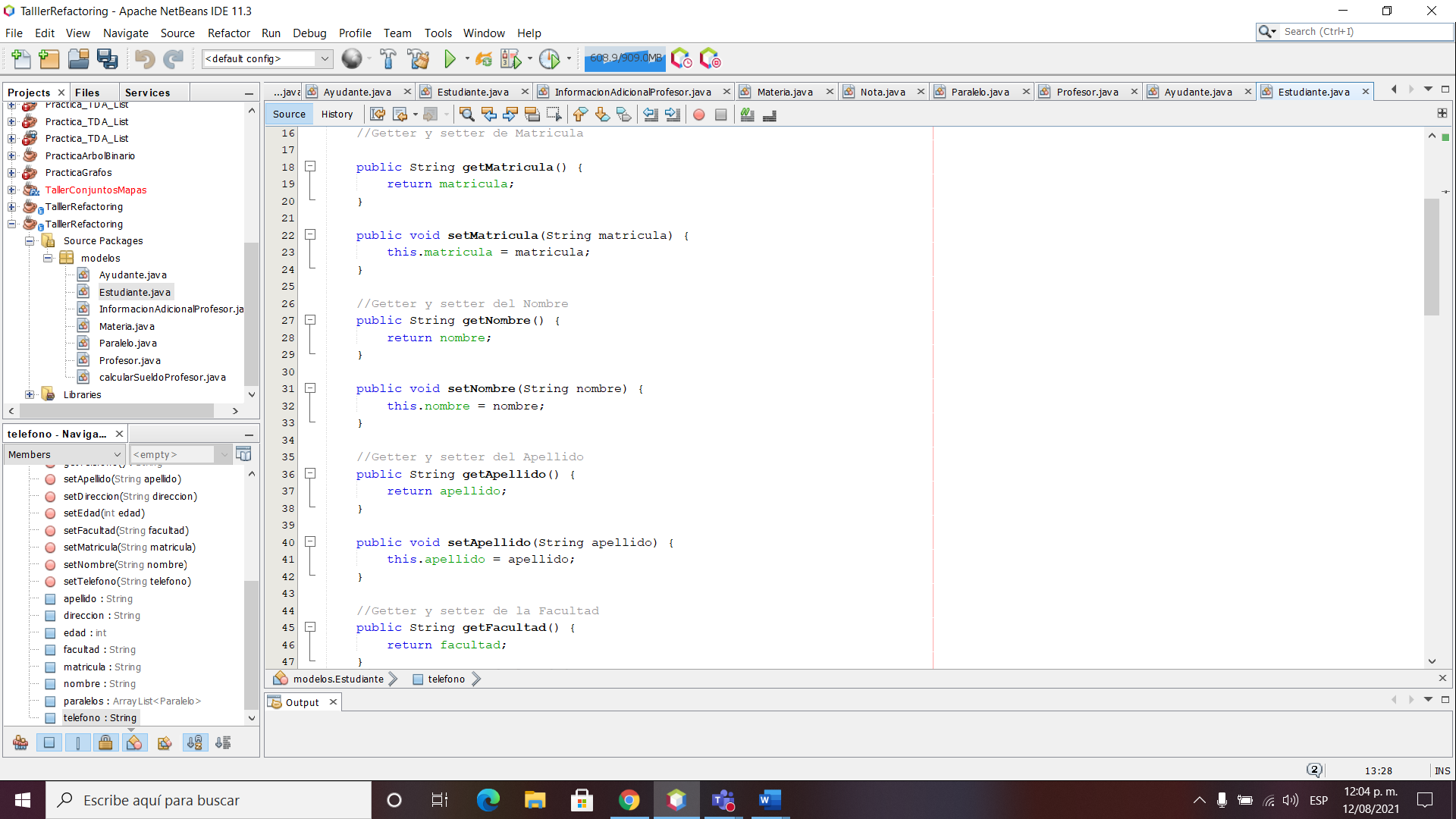
**Sección A**

**Sección A - CODE SMELLS DETECTADOS:**

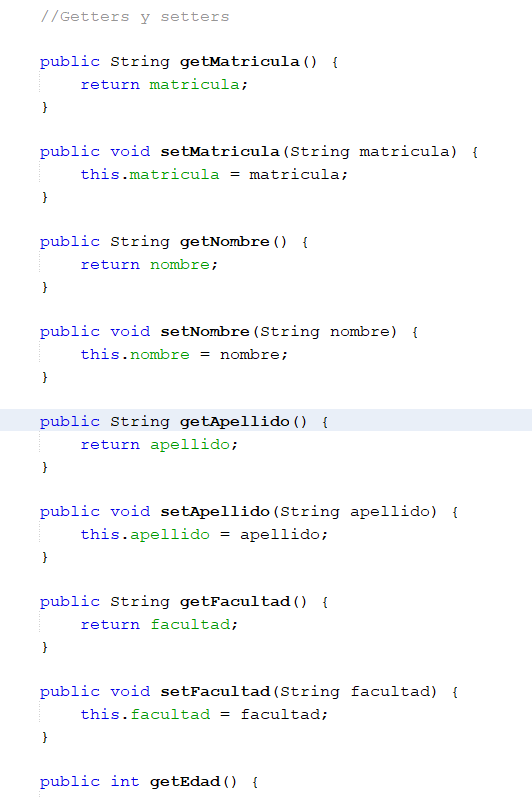
# Comments

* Consecuencias: En la clase Matricula poner comentarios para cada método creado provoca que se dificulte la lectura del código de dicha clase además de que da a entender que los nombres utilizados en los métodos no son lo suficientemente claros.
* Técnica de Refactorización: Esto puede solucionarse utilizando el método de refactorización llamado **Rename Method** en caso de que los nombres de los métodos no sean lo suficientemente claros, pero en esta ocasión optamos por eliminar los comentarios de casi todos los métodos y dejar un comentario general casi al principio de la clase especificando la idea central de todos los métodos que hay debajo.

Código Inicial:



Código Final:

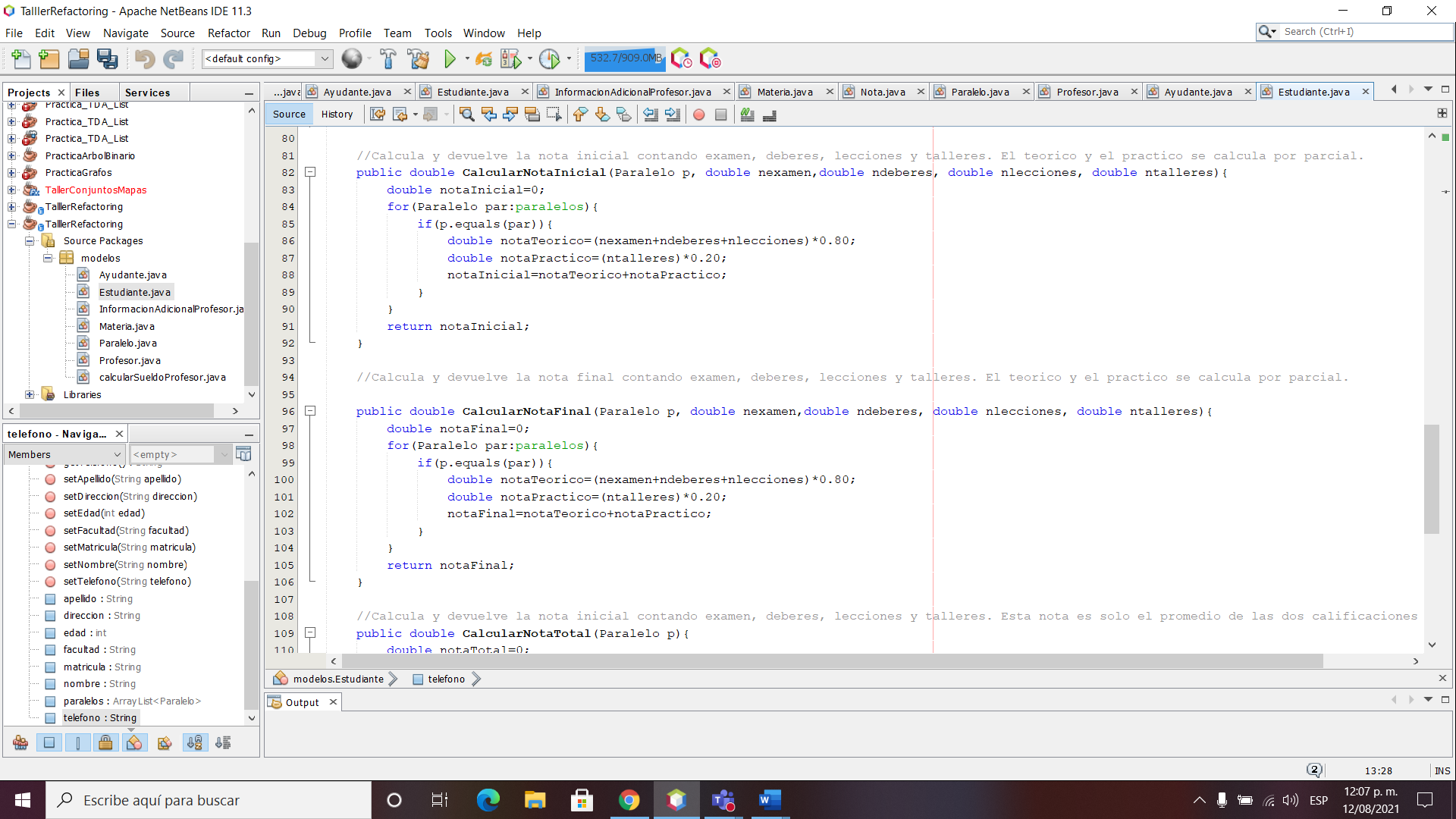


# Duplicate Code

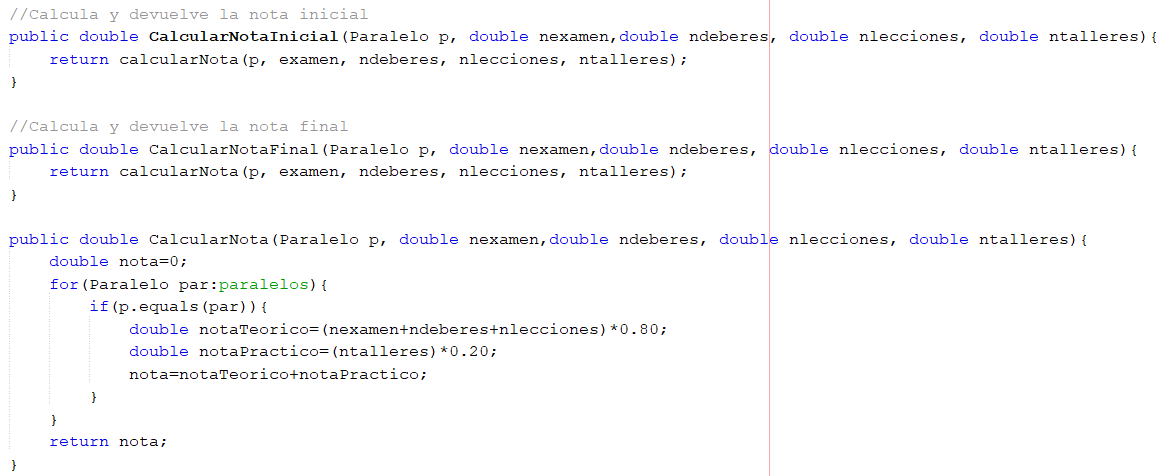
* Consecuencias: Es un tipo de duplicación sutil ya que ambos métodos al usar el mismo código crean largas secciones repetidas. El mantener un código duplicado hace que las clases sean más largas y que ocupen más espacio de memoria, además de hacer más difícil a la comprensión rápida del código.
* Tecnica de Refactorizacion: El **Composing methods** se encarga de remover códigos duplicados y de esta manera poder preparar el código para futuras modificaciones.

Ya que el mismo código se encuentra en dos métodos en la misma clase usamos **Extract Method** y realizamos llamadas para el nuevo método en ambos lugares.

Código Inicial:



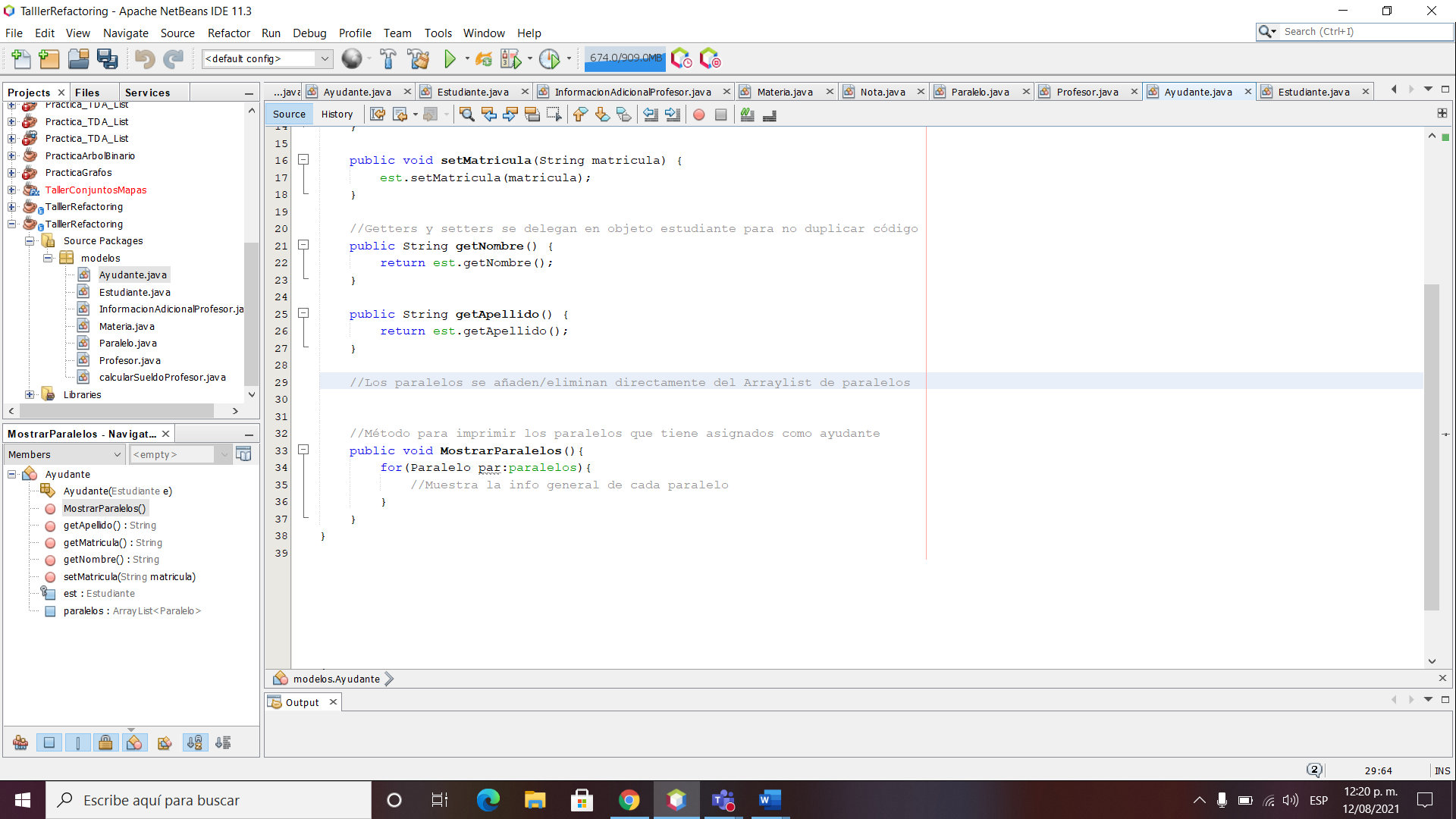
Código Final:



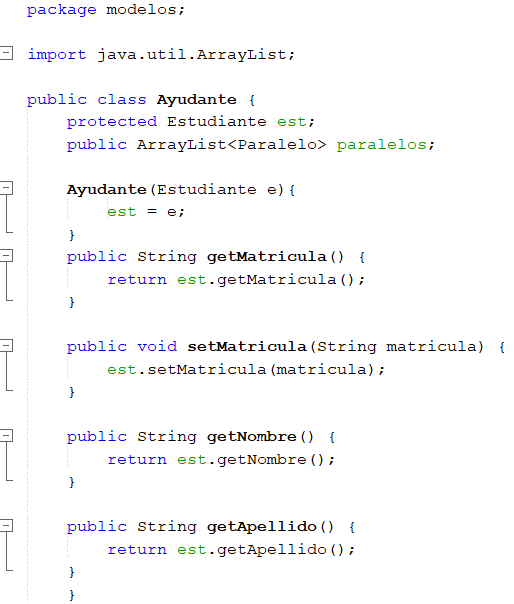
# Dead Code

* Consecuencias: En la clase Paralelo existe un método sin usar, estos métodos se crean por si acaso para en un futuro se pueda admitir características que nunca son implementadas como consecuencia de este code smell el código se vuelve difícil de entender y soportar.
* Técnica de refactorización: Al tener la clase Paralelo un método sin implementar se usará la técnica de refactorización de **Sustituir Algoritmo**. Para ello primero se debe asegurar que el método MostrarParalelos() no se encuentre redefinido en subclases y eliminar el método.

Código Inicial:



Código Final:

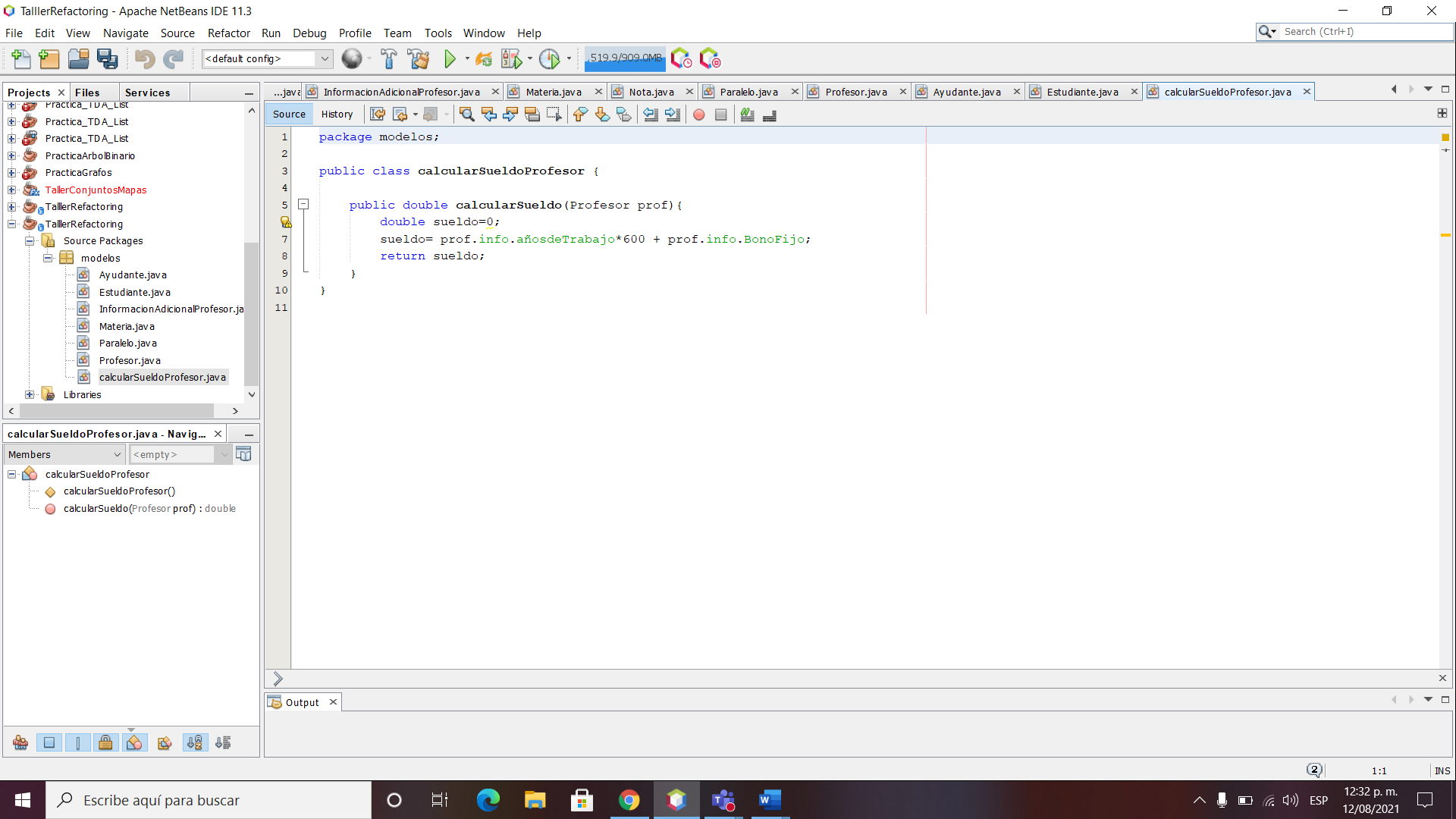


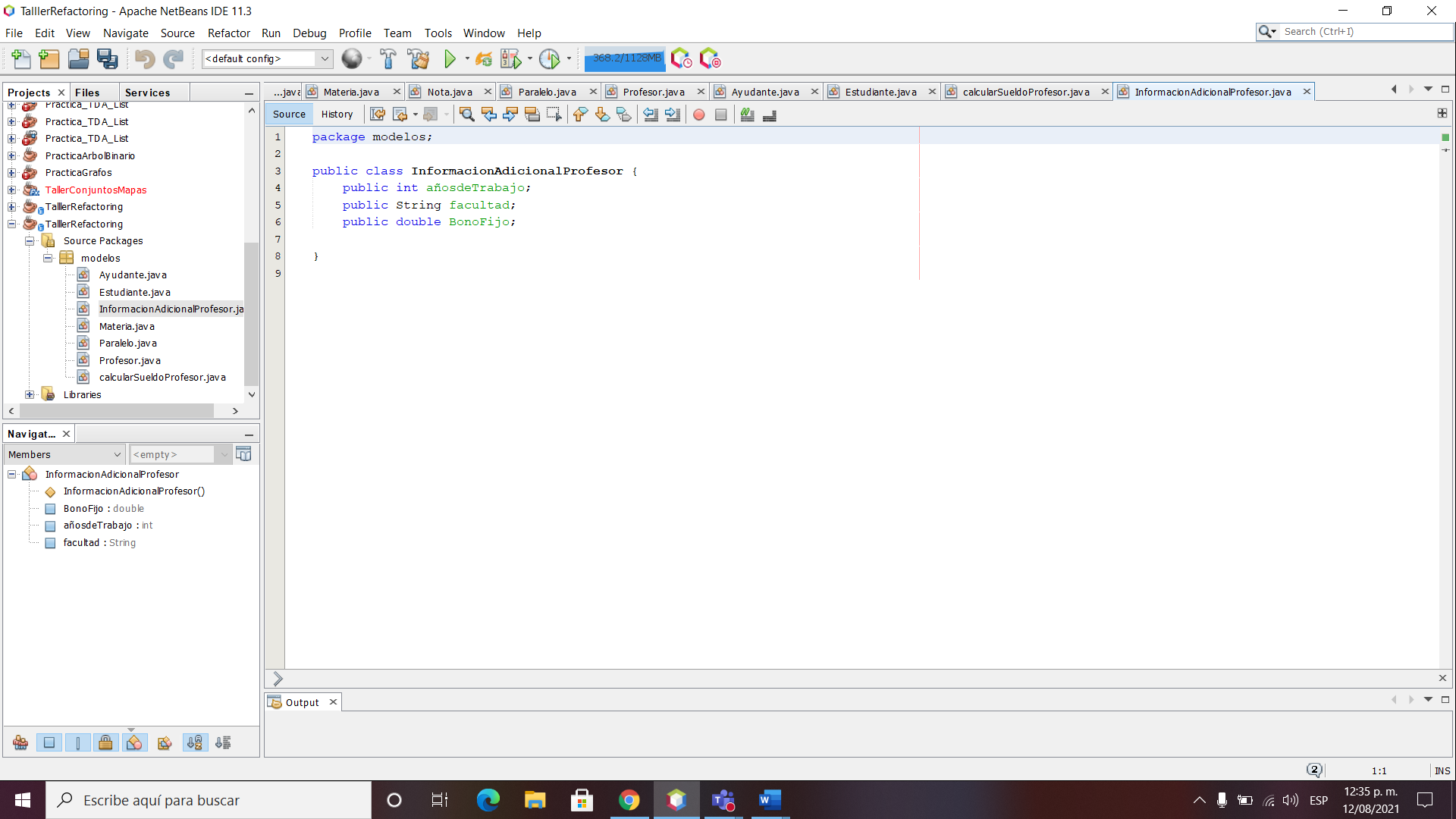
# Data Class

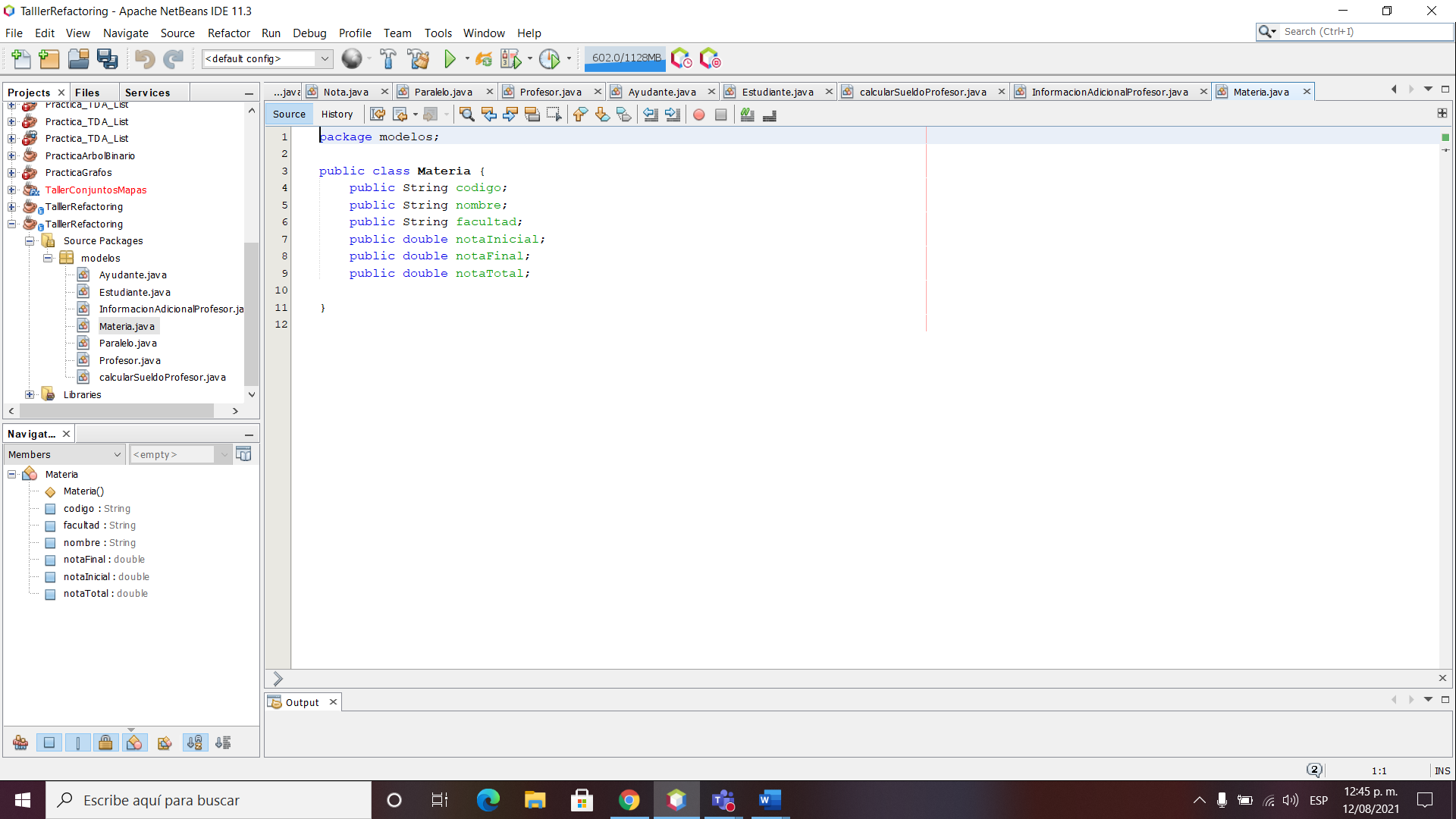
* Consecuencias: En las clases Materia e InformacionAdicionalProfesor contienen solo campos burdos para acceder a ellos, en este caso, solo contienen atributos públicos para acceder a ellos, por lo tanto, no contienen funcionalidad adicional y no pueden operar de forma independiente con los datos que poseen.
* Tecnica de refactorización: Como las clases contienen campos públicos se emplea la técnica de refactorización **Encapsulate Field** para ocultarlos del acceso directo, para que de esta manera requiera que el acceso se realice solo a través de getters y setters. Además en la clase InformacionAdicionalProfesor se incluyó el método CalcularSueldo de la clase calcularSueldoProfesor, eliminando la misma ya que solo contenía dicho método, el cual empleaba atributos de la clase InformacionAdicionalProfesor. También se agregaron los métodos de calcular nota que estaban en la clase estudiante

Por otro lado, cuando se movió el método calcularSueldo a la clase InformacionAdicionalProfesor se tuvo el mal olor Speculative Generality ya que no se usaba el atributo prof, por lo que se aplicó la técnica de refactorización remove parameter

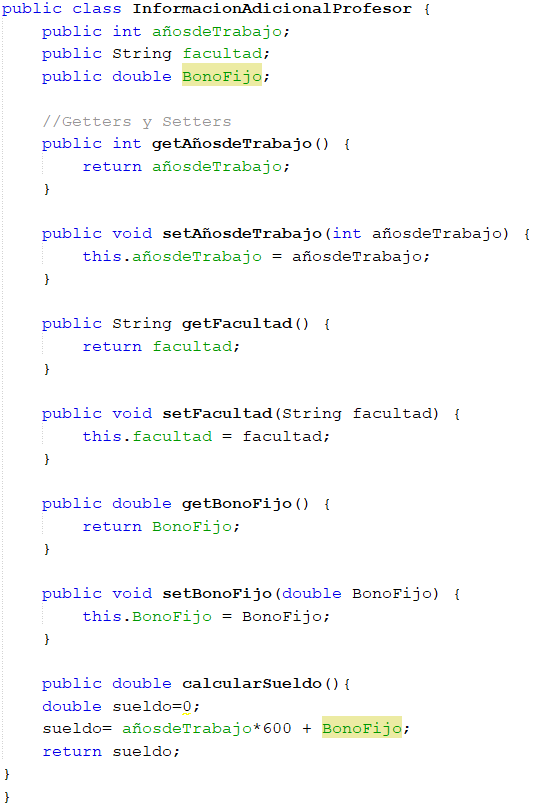
Codigo Inicial:

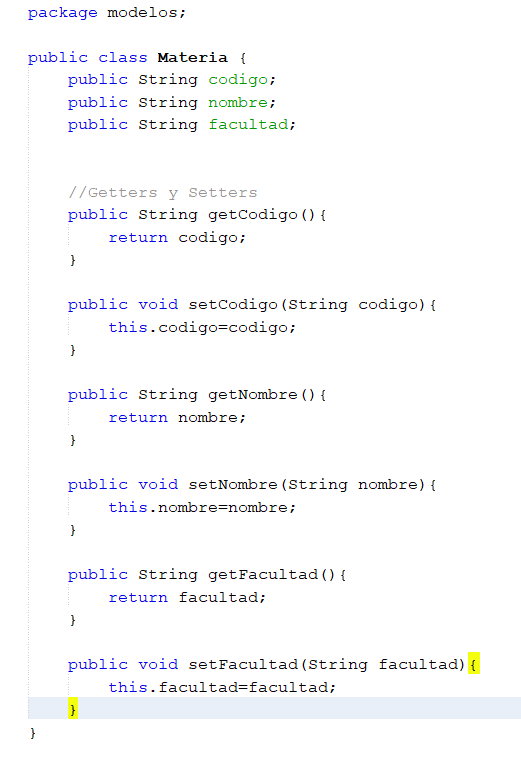






Codigo Final:





# Data Clumps

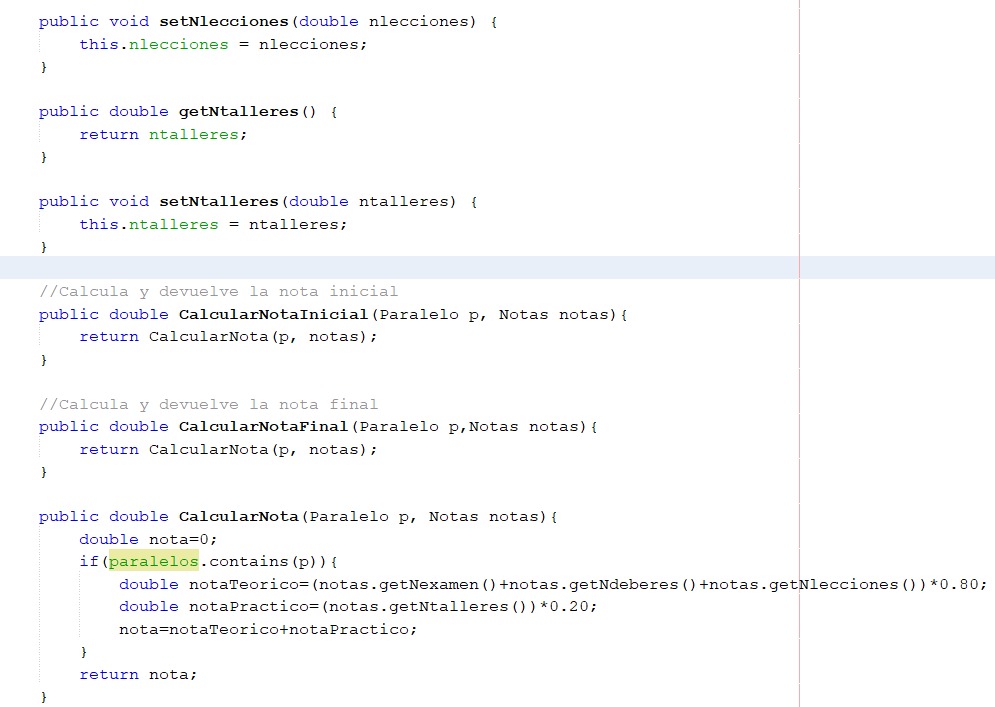
* Consecuencias: En la clase estudiante existen varios métodos que contienen el mismo grupo de parámetros (CalcularNotaIncial, CalcularNotaFinal y CalcularNota) que están haciendo el código más difícil de entender perjudicando la reutilización. Esto provoca la duplicación de código tanto de los propios parámetros como de las operaciones relacionadas. Se puede reconocer debido a que una sola de esas variables no tiene mucho sentido sin las demás, si dejamos solamente la nota de talleres denota la clara falta de los otros parámetros. Si se elimina este code smell mejorará la comprensión y organización del código.
* Tecnica de refactorización: La técnica de refactorización a utilizar **sería Introduce Parameter Object**, ya que se utiliza cuando los métodos contienen un grupo repetido de parámetros. Se soluciona reemplazando estos parámetros con un objeto. Al consolidar los parámetros en una sola clase, los métodos también se mueven a esta misma, dando un código más legible.

Codigo Inicial:



Codigo Final:

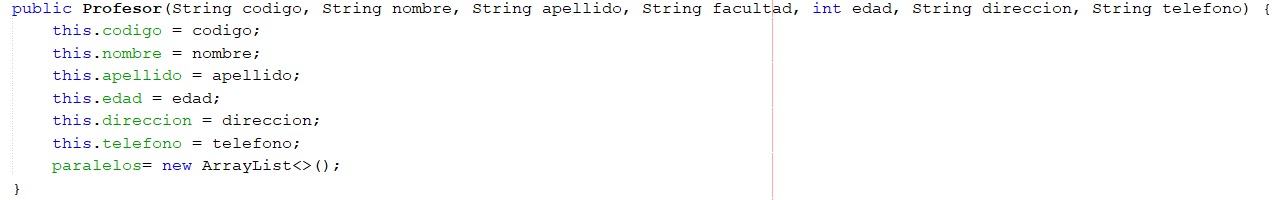




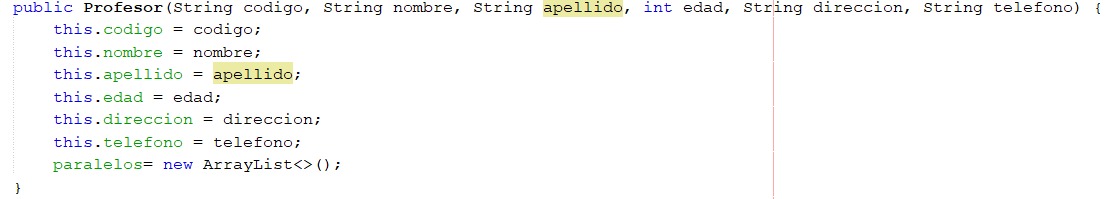
# Speculative Generality

* Consecuencias: En la clase Profesor, se implementa el respectivo constructor de dicha clase, y, entre uno de sus parámetros está declarada la variable facultad, la cual no es utilizada dentro de la implementación del método. Cada parámetro en una llamada a un método juega un rol importante ya que obliga al programador a leerlo para averiguar qué información se encuentra en este parámetro.
* Tecnica de refactorización: La técnica de refactorización que ayudará a solucionar este mal olor es **Remove Parameter**, la cual hará que dicho método contenga únicamente los parámetros que realmente requiere.

Codigo Inicial:



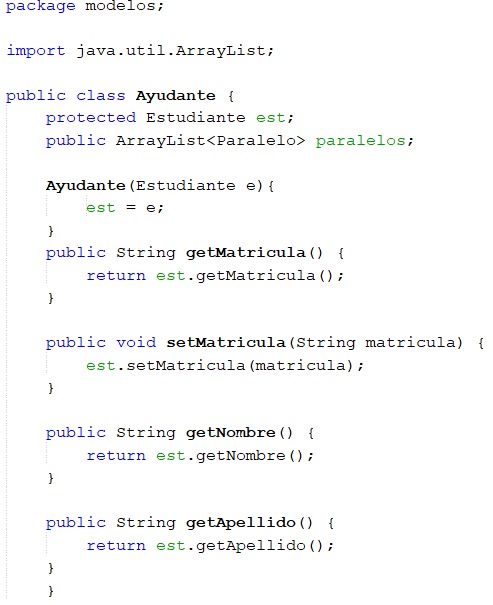
Codigo Final:



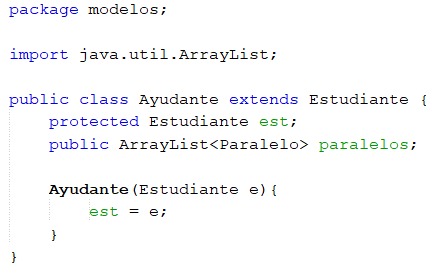
# Inapropiate Intimacy

* Consecuencias: En la clase Ayudante se encuentra como atributo un objeto de la clase Estudiante, siendo este de vital importancia en clase, sin embargo, su constructor y métodos tales como getMatricula(), getNombre(), getApellido() y demás, hacen referencias a métodos de la clase Estudiante.
* Tecnica de refactorización: Como la clase Ayudante contiene muchos métodos delegados de la clase Estudiante, entonces se procede a utilizar la técnica de **refactorización Replace Delegation with Inheritance**, de esta manera se heredarán los atributos y métodos de la clase Estudiante’

Codigo Inicial:



Codigo Final:



**Sección B - GITHUB:**

[**https://github.com/stevenchoez/TallerRefactoring**](https://github.com/stevenchoez/TallerRefactoring)