Taller Refactoring

DISEño de software

Integrantes:

Darien AlTAMIRANO

Stefany Lavayen

Andres Medina

Edwin eras

aaron villao

Contenido

[**Code Smell** 2](#_Toc60946028)

[**Primitive Obsession** 2](#_Toc60946029)

[Consecuencia: 2](#_Toc60946030)

[Solución: 2](#_Toc60946031)

[**Data class** 3](#_Toc60946032)

[Consecuencia: 3](#_Toc60946033)

[Solución: 3](#_Toc60946034)

[**Duplicate Code** 4](#_Toc60946035)

[Consecuencia: 4](#_Toc60946036)

[Solución: 4](#_Toc60946037)

[**Comments** 5](#_Toc60946038)

[Consecuencia: 5](#_Toc60946039)

[Solución: 5](#_Toc60946040)

[**Inappropriate Intimacy** 7](#_Toc60946041)

[Consecuencia: 7](#_Toc60946042)

[Solución: 7](#_Toc60946043)

[**Primitive Obsession** 9](#_Toc60946044)

[Consecuencia: 9](#_Toc60946045)

[Solución 9](#_Toc60946046)

[**Data Clumps** 11](#_Toc60946047)

[Consecuencia: 11](#_Toc60946048)

[Solución 12](#_Toc60946049)

[**Feature Envy** 13](#_Toc60946050)

[Consecuencia: 13](#_Toc60946051)

[Solucion: 13](#_Toc60946052)

[**Long Parameter List** 14](#_Toc60946053)

[Consecuencia: 14](#_Toc60946054)

[Solución: 14](#_Toc60946055)

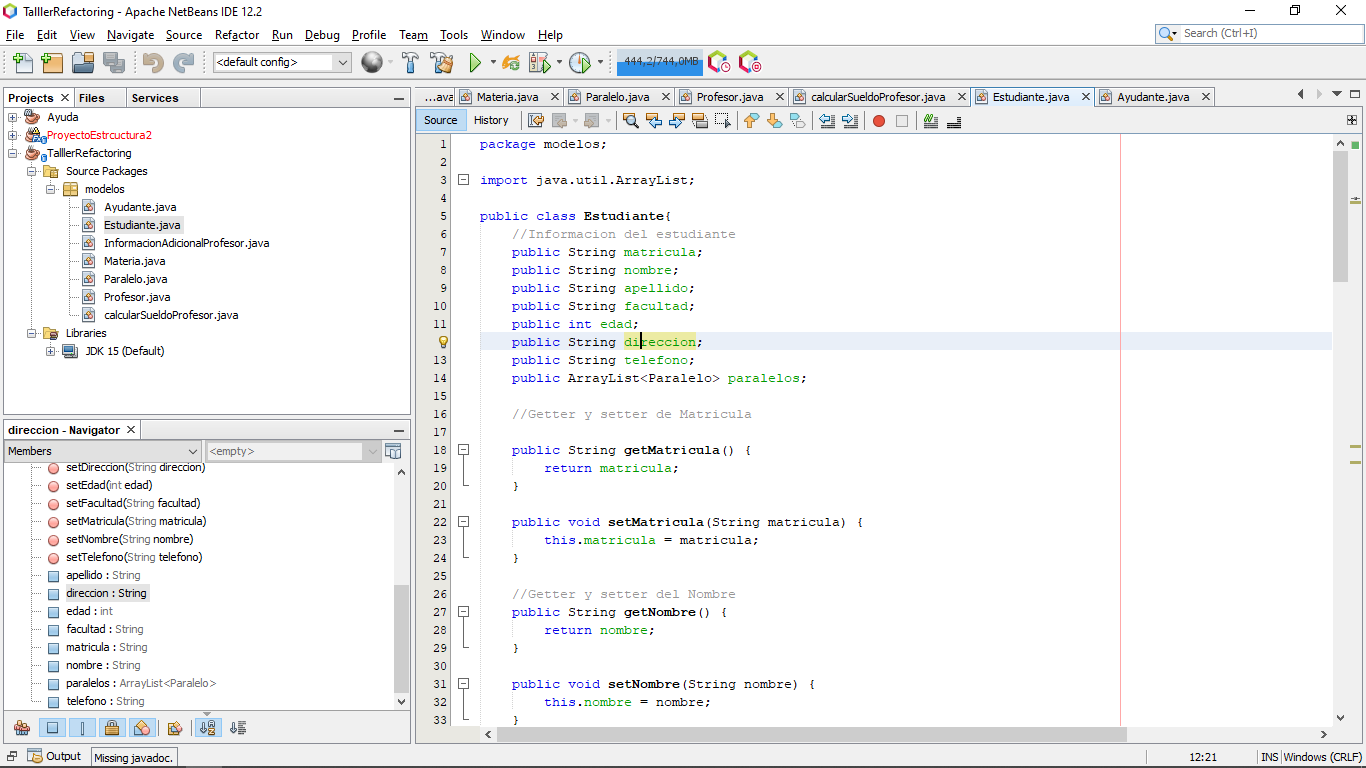
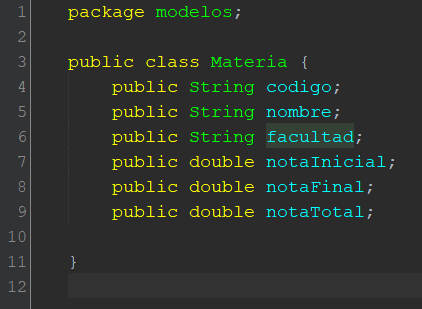
# **Code Smell**

## **Primitive Obsession**

En las clases Estudiante y Materia tiene un campo igual llamado facultad de tipo String. Considerando que es un sistema de universidad las facultades no deben ser consideradas como primitivos y repetirse en varias clases. Además, en la clase Materia se puede observar que tiene varios métodos de tipo double.

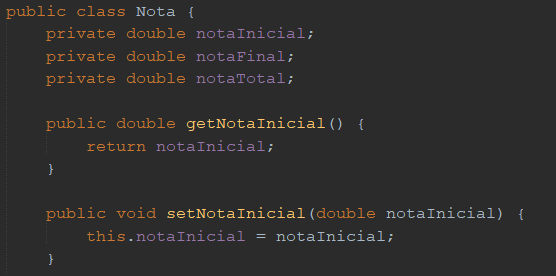
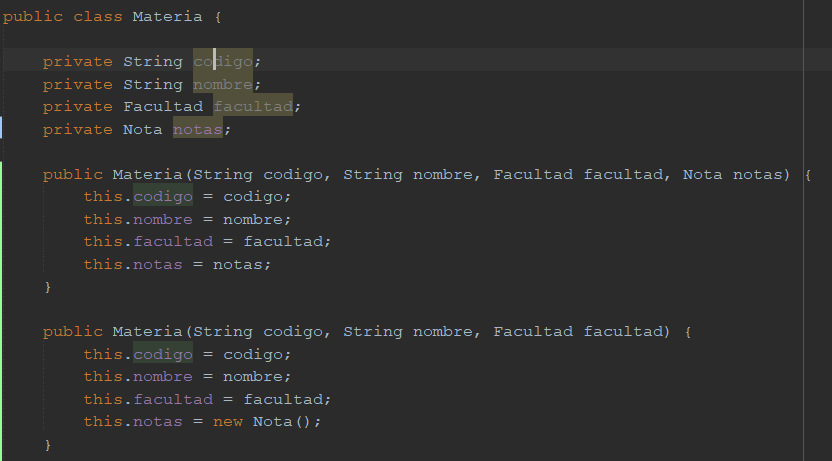
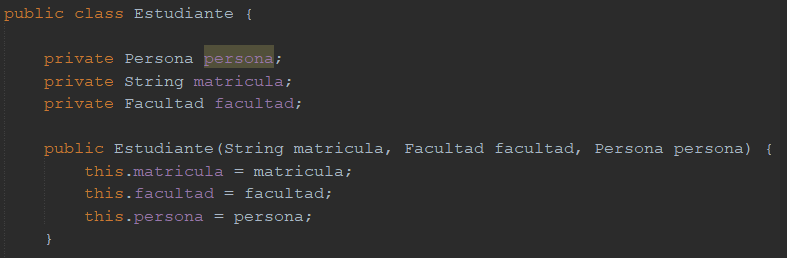
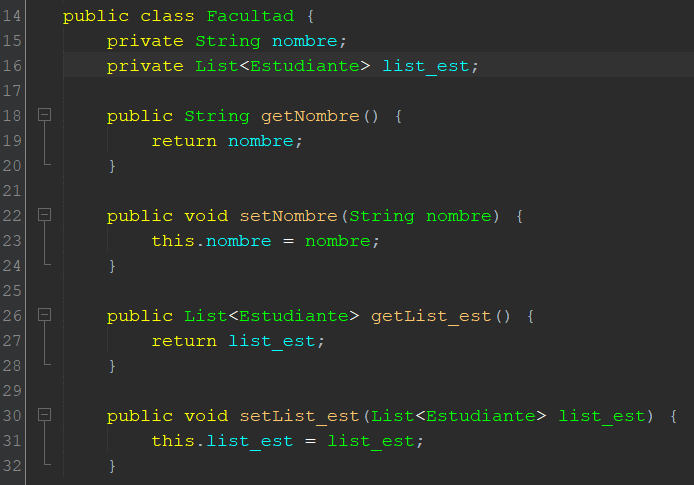
### Consecuencia:

Tener un código menos escalabre y con menor acoplamiento debido a la mala organización de las clases, lo que conlleva a un difícil entendimiento.

Solución:

Usar **EXTRACT CLASS** para crear una clase llamada Facultad la cual tenga su propio nombre y atributos y pueda extender funcionalidad en un futuro, y una clase Nota para agrupar los atributos de tipo double. Se considero

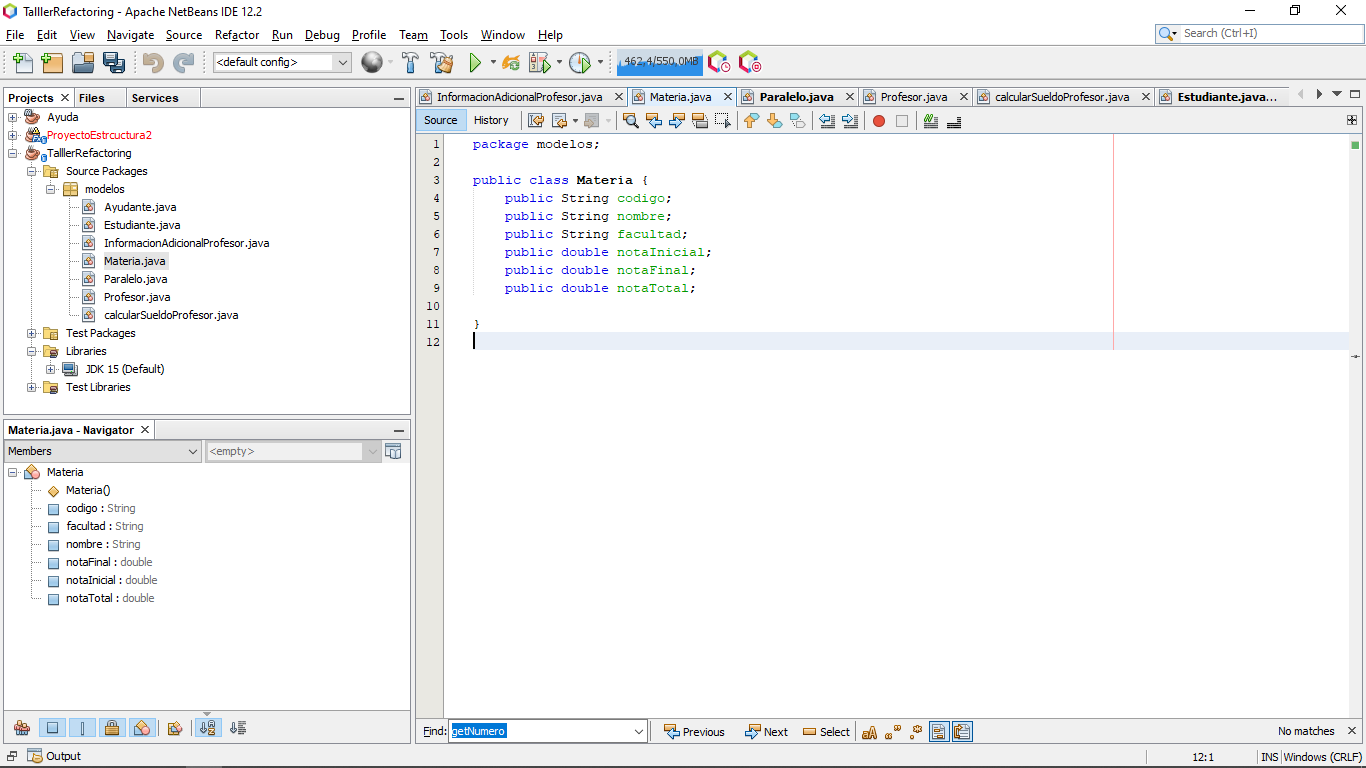


## **Data class**

La clase Materia solo tiene atributos con modificador público, lo que permite que cualquier clase puede acceder a estos. Añadiendo que en materia no hay otros métodos que caractericen al objeto como tal.

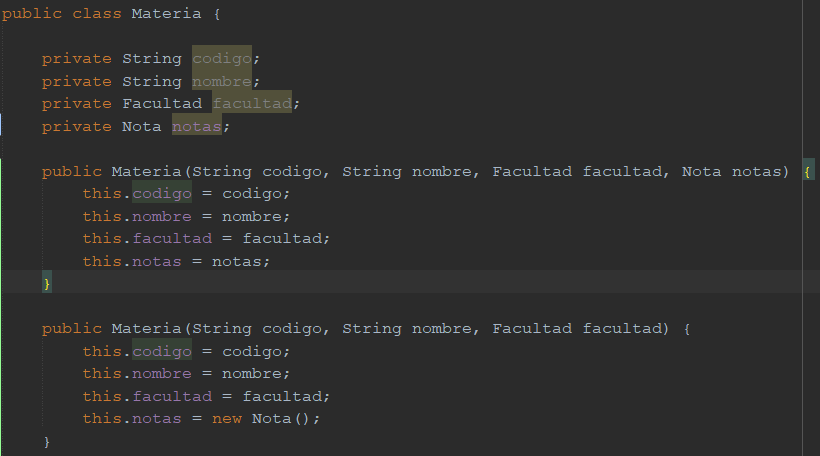
### Consecuencia:

Los valores de cada atributo pueden cambiar según lo indique la clase que lo llama u otras clases, y en el peor de los casos ocasionar alteraciones drásticas en el programa; además de la pérdida de tiempo al realizar el mantenimiento respectivo



### Solución:

Aplicar “**ENCAPSULATE FIELD**” para evitar que clases externas puedan alterar los datos de la clase, se decide también implementar los getters, setters y el respectivo Constructor.

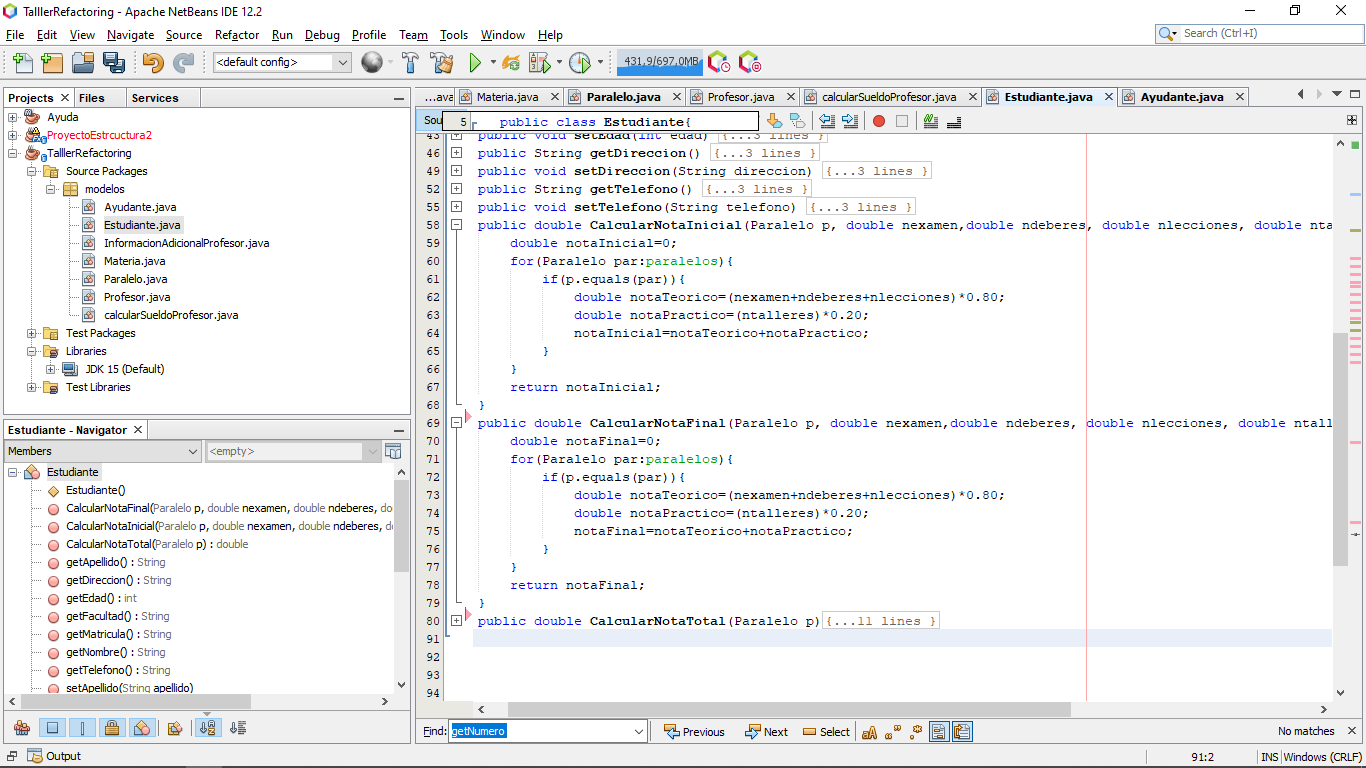


## **Duplicate Code**

En la clase Estudiante hay dos métodos (calcularNotaInicial, calcularNotaFinal) que reciben los mismos parámetros, y tienen las mismas líneas de código.

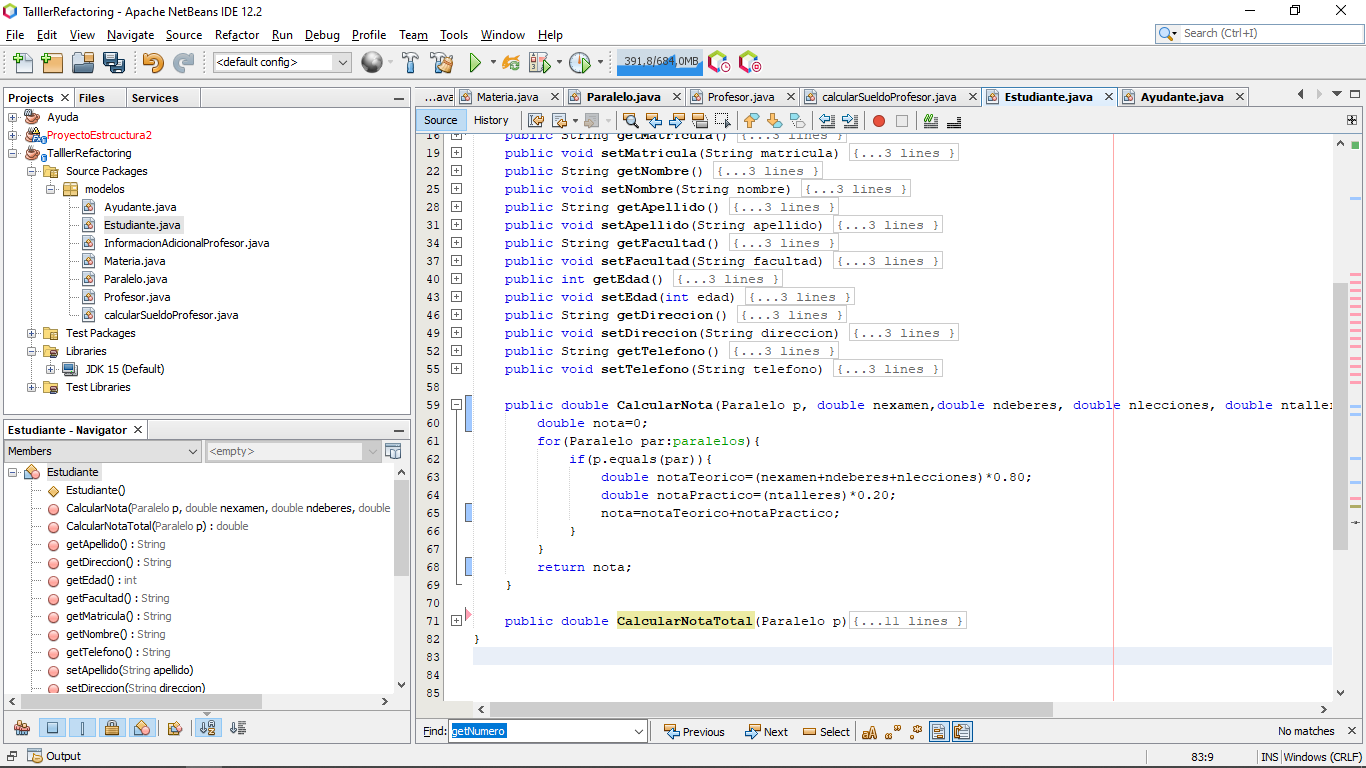
### Consecuencia:

Genera malestar y confusión al momento de realizar el respectivo mantenimiento que se torna complejo.



### Solución:

Se debe usar “**EXTRACT METHOD**” para quedarnos con un solo método, y las clases deben llamar a este método najo su propio contexto.

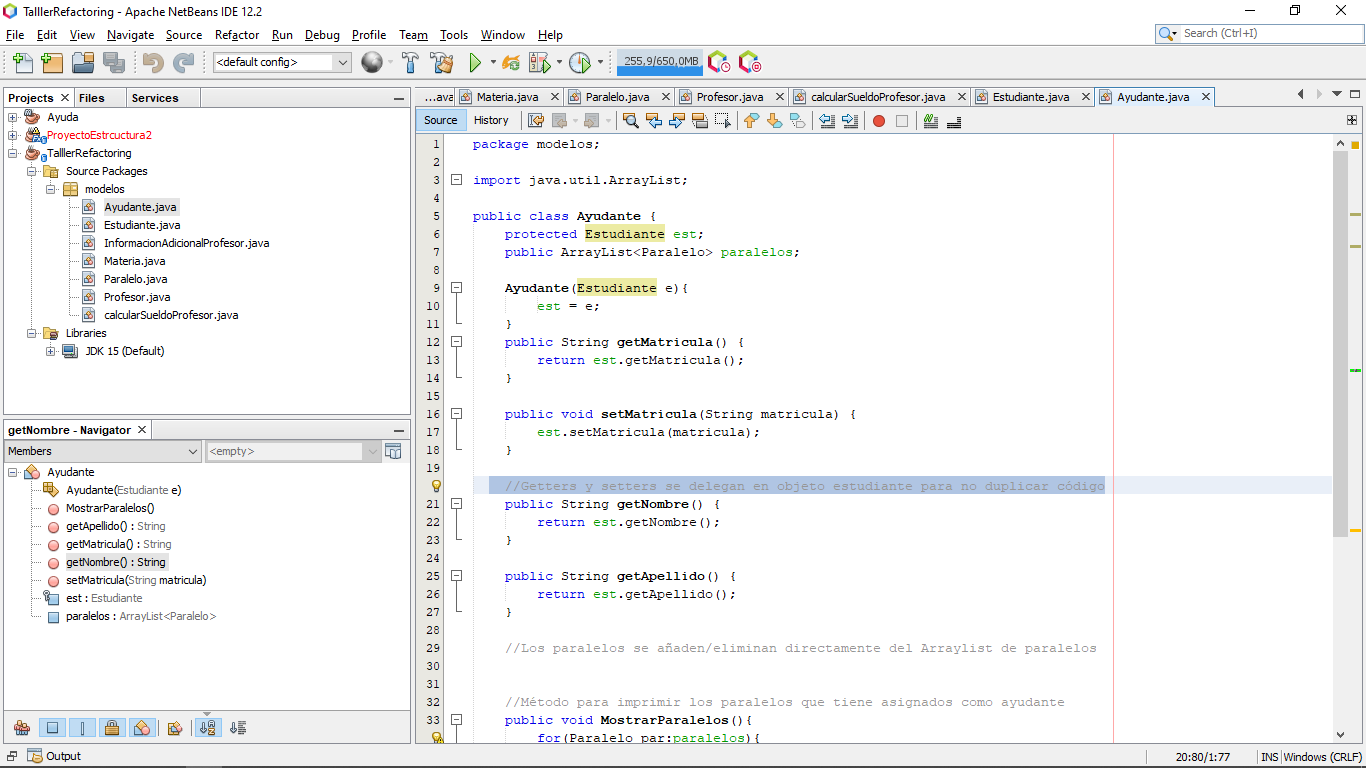


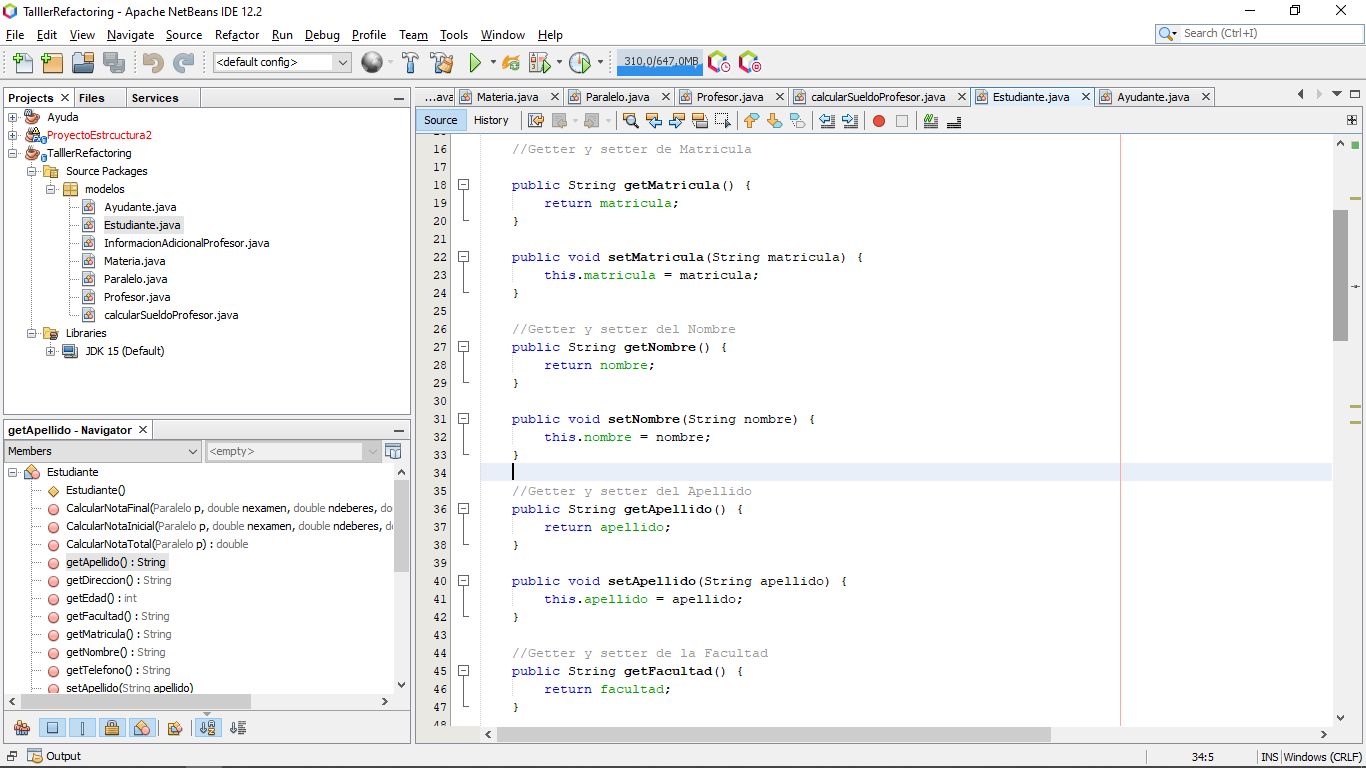
## **Comments**

En la clase Estudiante y Ayudante, contiene comentarios que describen los getters y setters cuando no en este caso no es necesario que se detallen lo que se realiza en esos métodos.

### Consecuencia:

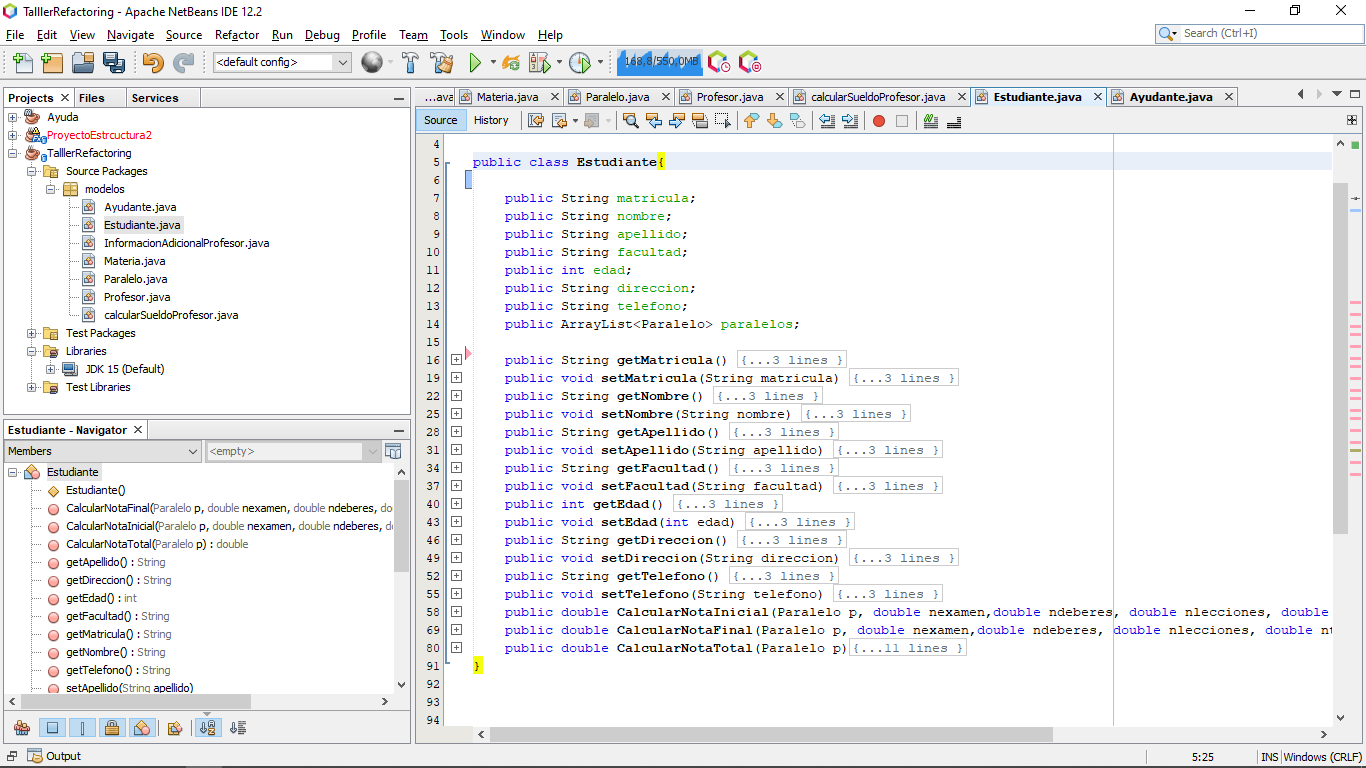
Los comentarios son engorrosos en métodos bastante obvios.

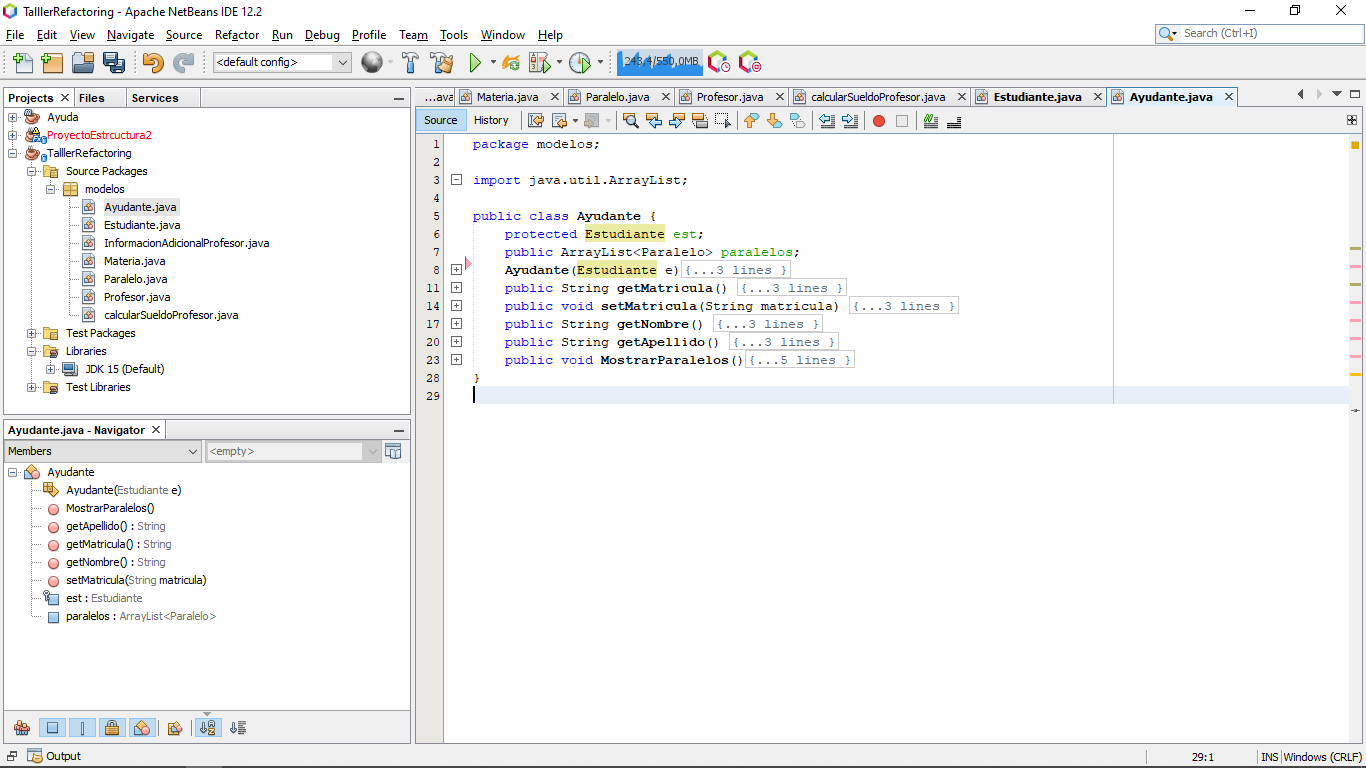




### Solución:

Los métodos ya cumplen “**EXTRACT METHOD**” e “**INTRODUCE ASSERTION**” por lo que basta con eliminar los comentarios por lo obvios que son.



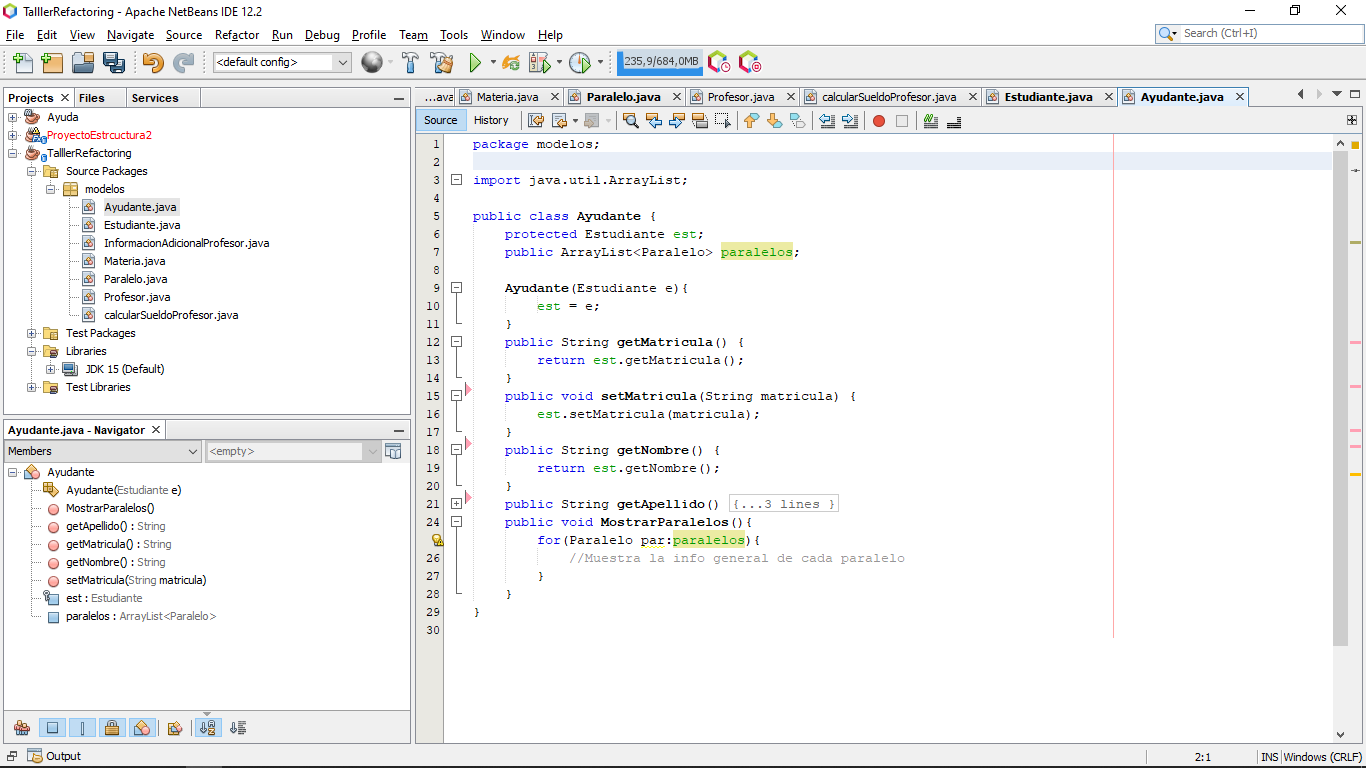
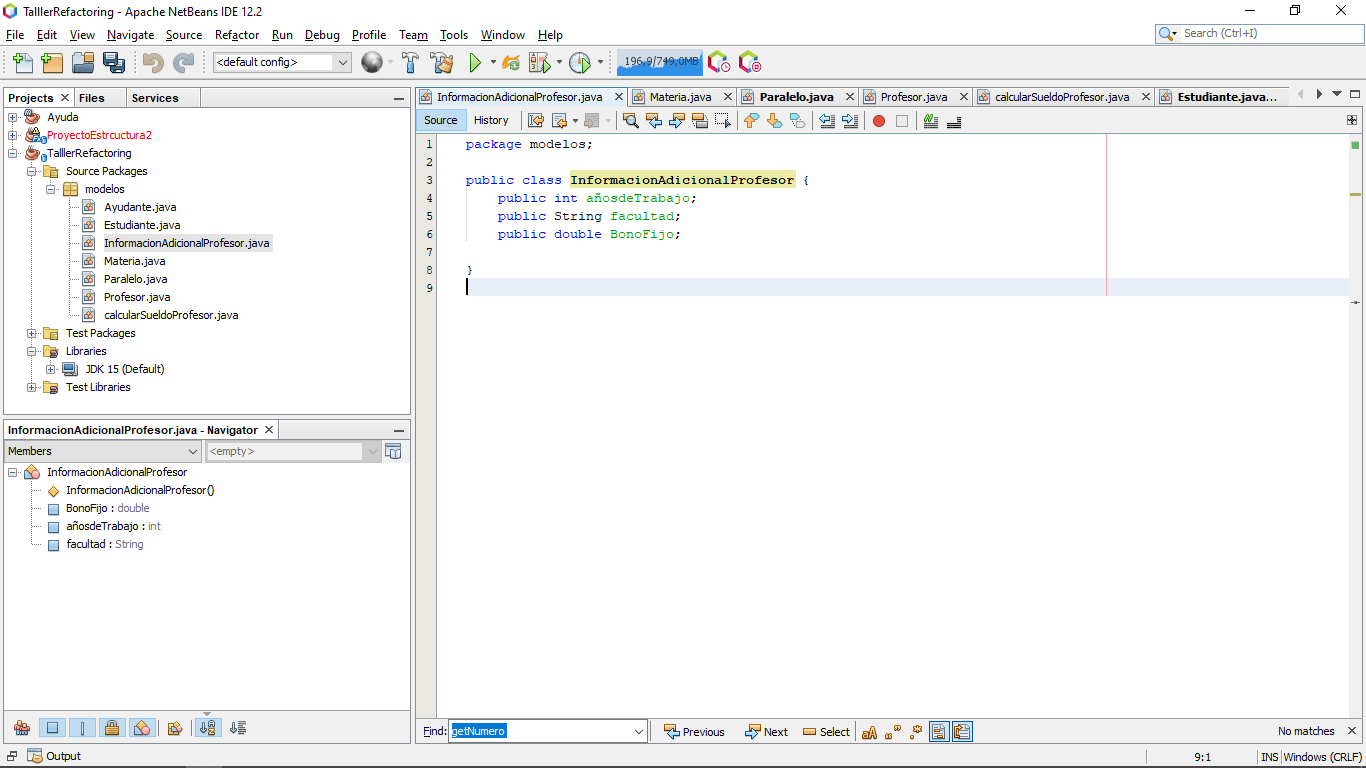
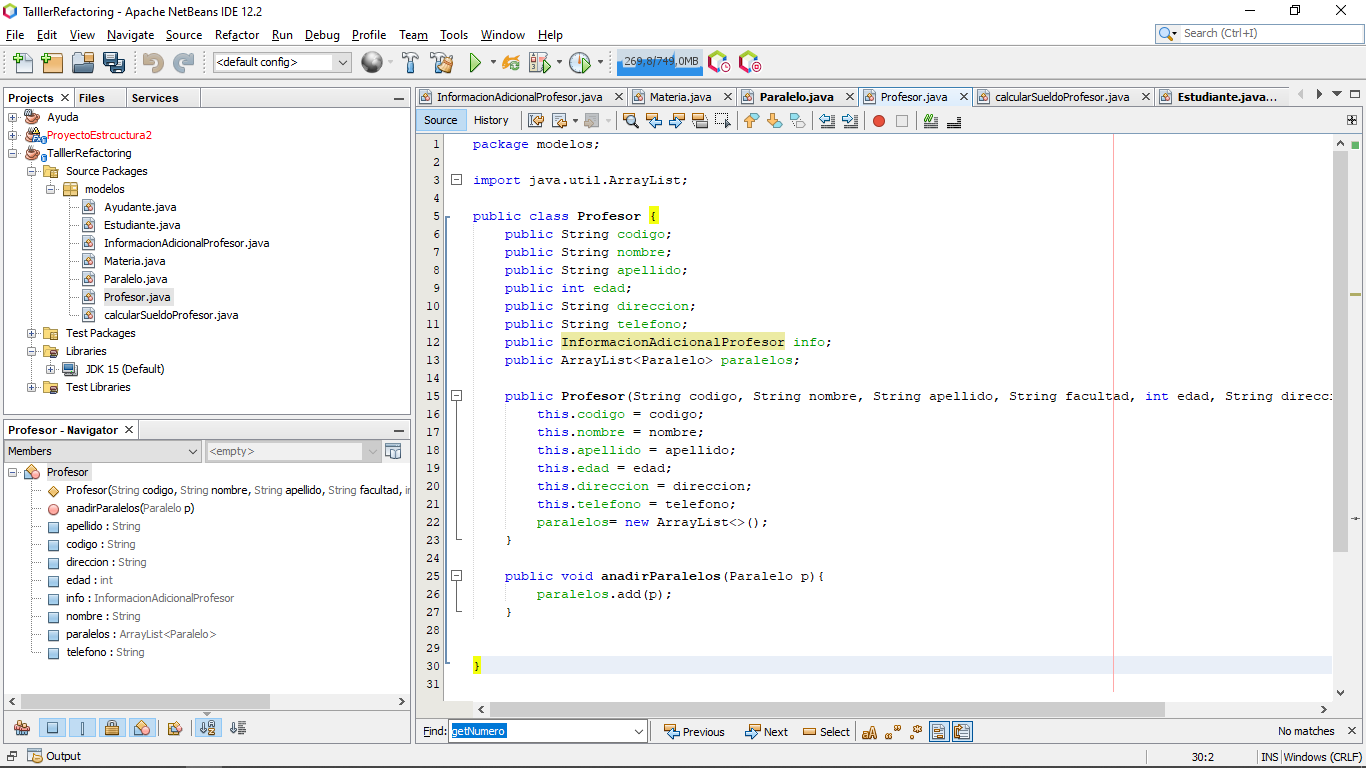
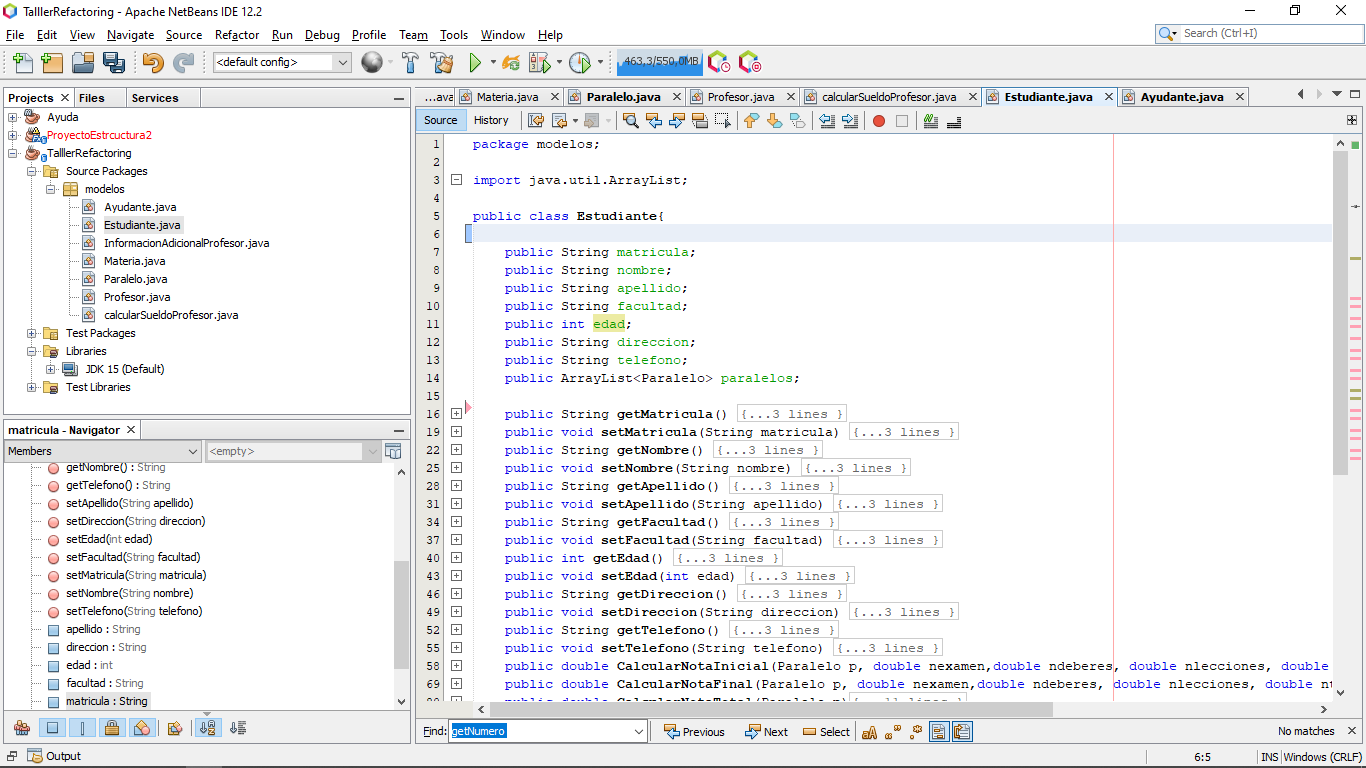


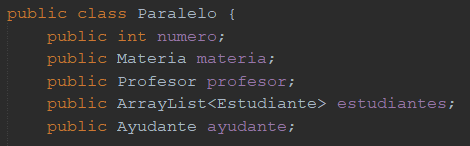
## **Inappropriate Intimacy**

En la clase *Ayudante, Paralelo, InformacionAdicionalProfesor, Estudiante y Profesor* tienen atributos que pueden ser utilizados en otras clases, y que pueden ser accedidas directamente al tener modificador público. Cabe resaltar, que en las clases que tiene como atributo paralelos, se agregan y eliminan directamente del arreglo *paralelos.* Esto puede causar varios problemas a futuro.

### Consecuencia:

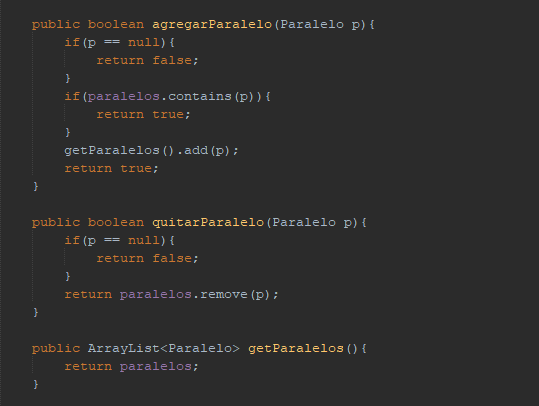
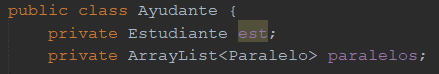
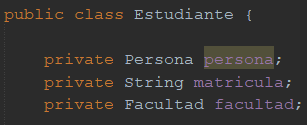
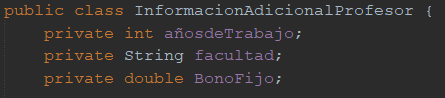
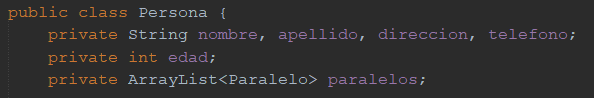
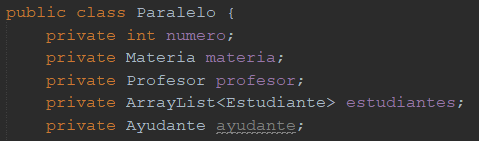
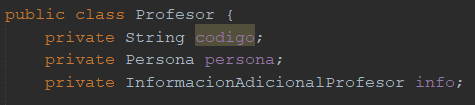
Si se desea cambiar la estructura de datos de los paralelos de las clases mencionadas, también deberán cambiar el código para agregar o eliminar paralelos las clases que las utilizan; o incluso otra clase podría eliminar o modificar de forma inesperada el arreglo ya que tiene acceso. Por lo que, no cuenta con la seguridad necesaria para ser una aplicación segura.





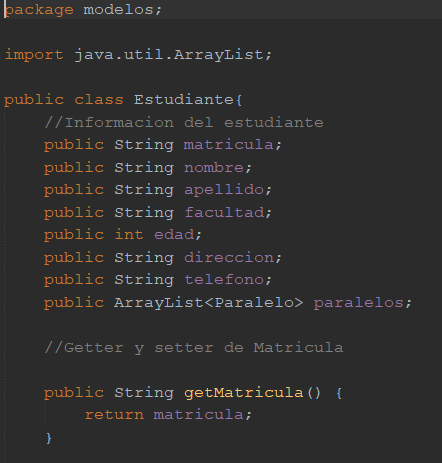
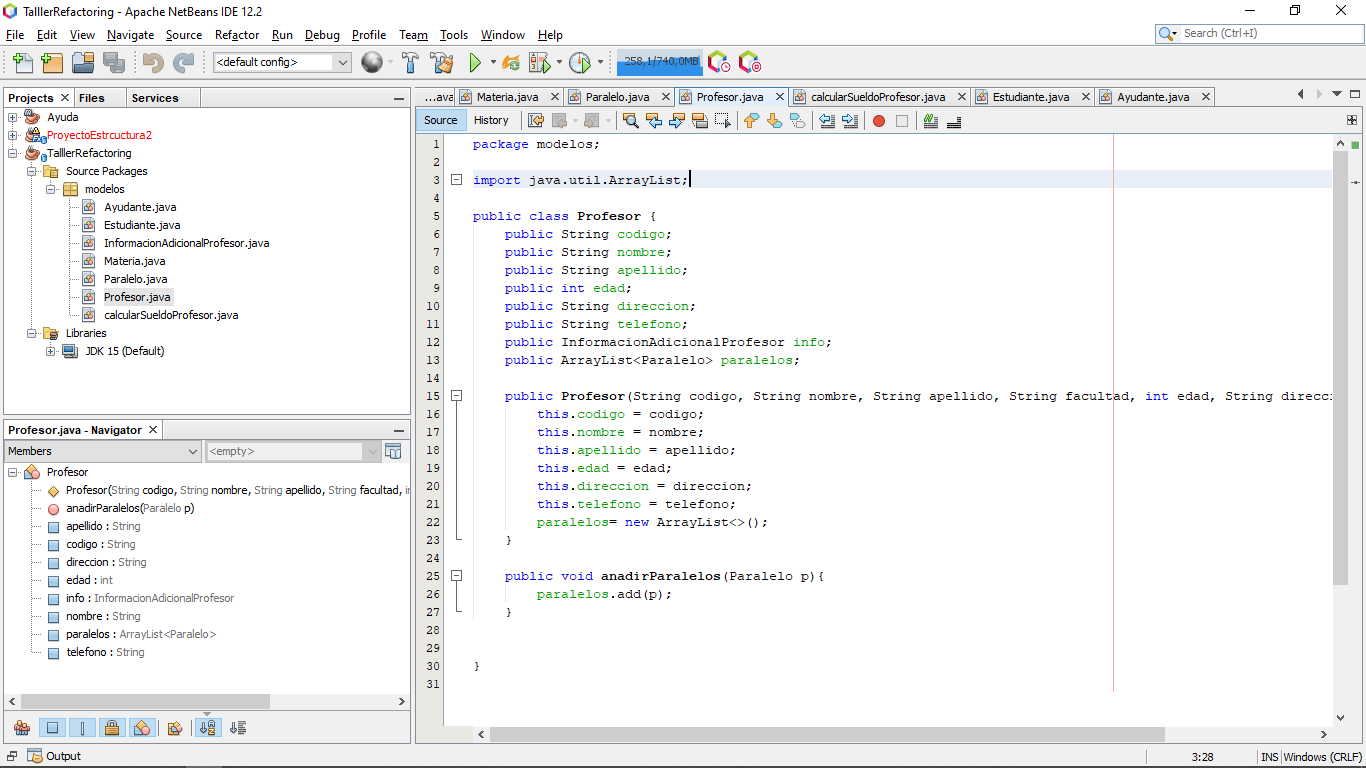
### Solución:

Lo preferible para este caso es que estas clasesmanejen mediante métodos sus respectivos campos, y los paralelos que se van a agregar o eliminar con el objetivo de impedir que se pueda setear el arreglo (**ENCAPSULATE COLLECTION** y **ENCAPSULATE FIELD**).

## **Data Clumps**

Las clases Estudiantes y Profesor tienen campos que pueden ser agrupados en otra clase, estos son nombre, apellido, edad, dirección y teléfono.

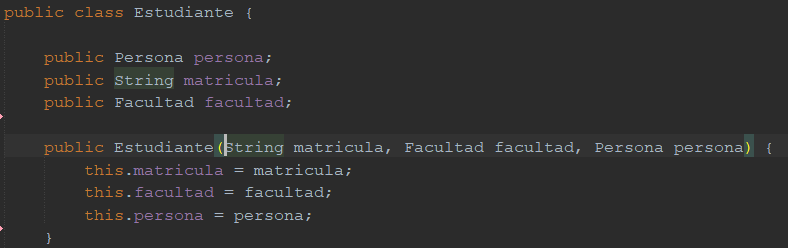
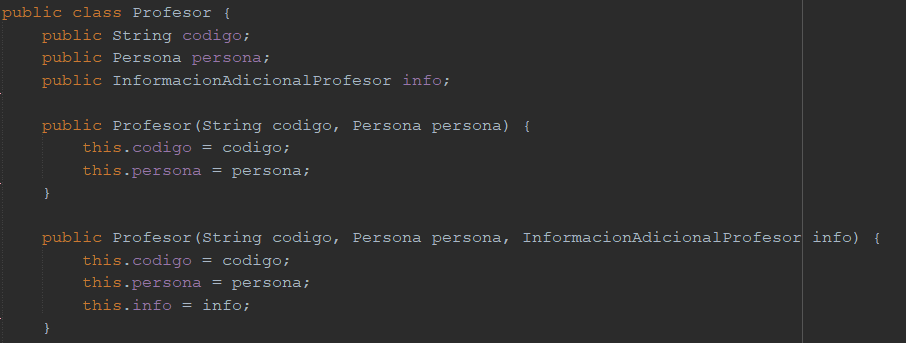
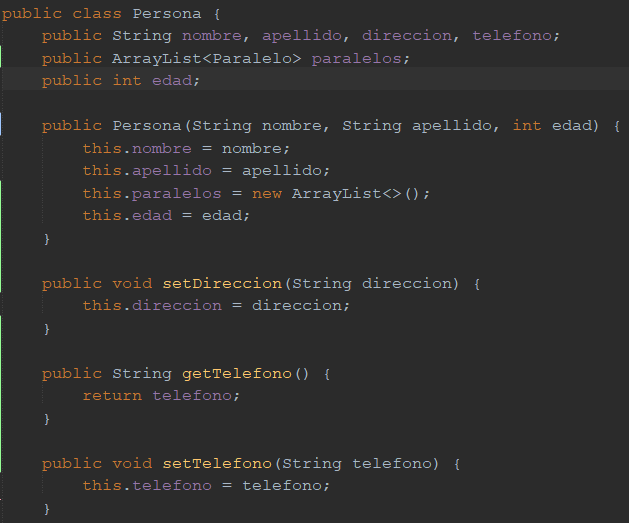
 

Consecuencia:

No se puede reutilizar y dificulta bastante el mantenimiento del proyecto.

Solución:

Usar **EXTRACT CLASS** y **REPLACE VALUE WITH OBJECT** para crear una clase llamada Persona la cual tenga los campos mencionados anteriormente y permita abreviar los respectivos atributos de la clase Estudiante y Profesor al hacer ese de un único objeto Persona.

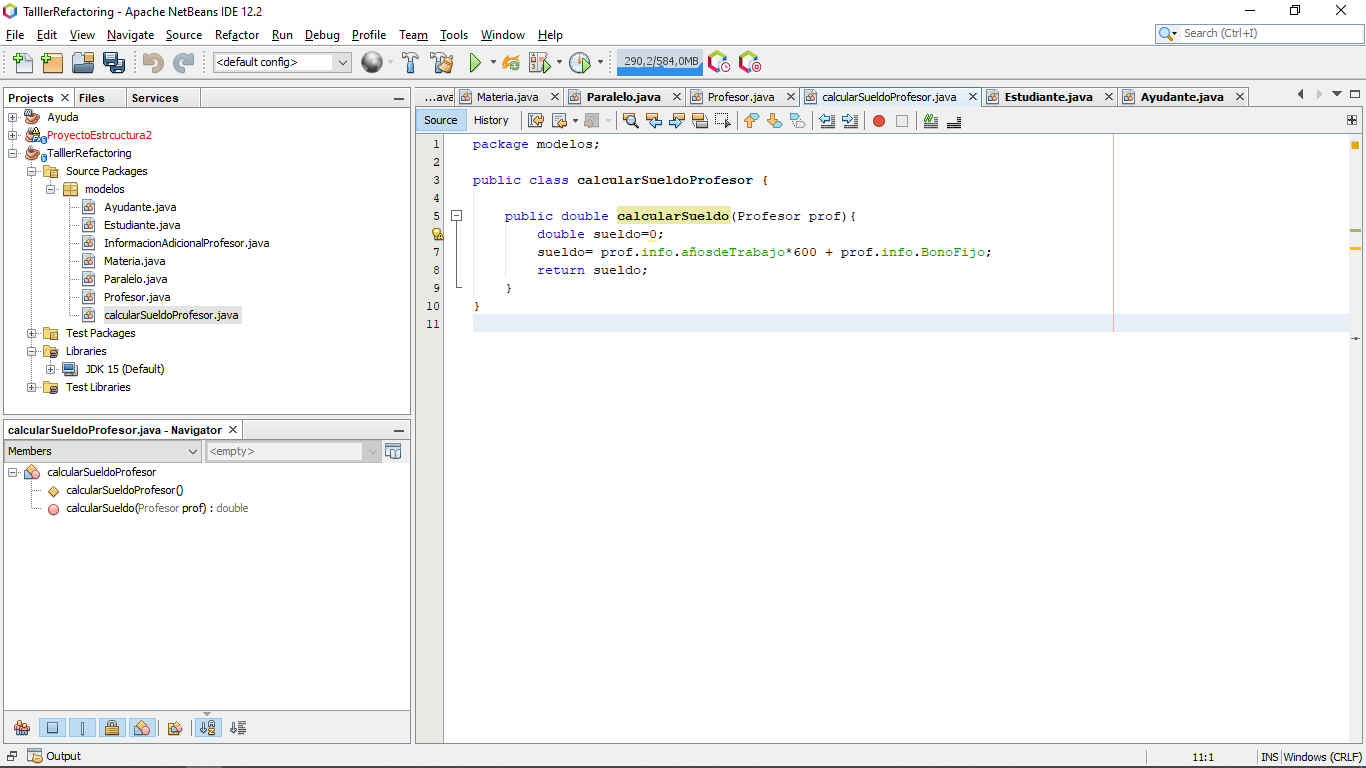


## **Feature Envy**

En la clase calcularSueldoProfesor hay un método calcularSueldo que recibe como parámetro un objeto Profesor y utiliza sus valores, mientras que esta clase mencionada no tiene atributos adicionales al único método indicado.

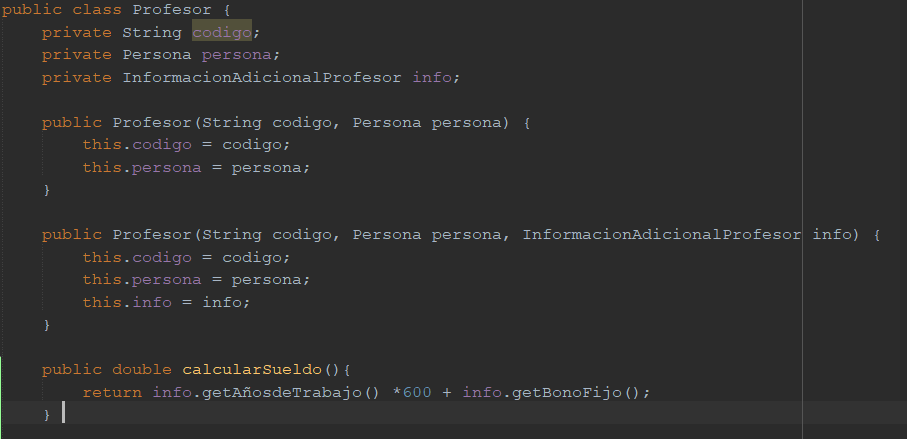
### Consecuencia:

Se tiene un clase unica adicional y pequeña para un solo método, por lo que se gastaria mas recursos.



### Solucion:

Aplicar **MOVE METHOD**, para trasladar calcularSueldo como método de instancia de la clase Profesor para que utilizar de sus atributos requeridos.

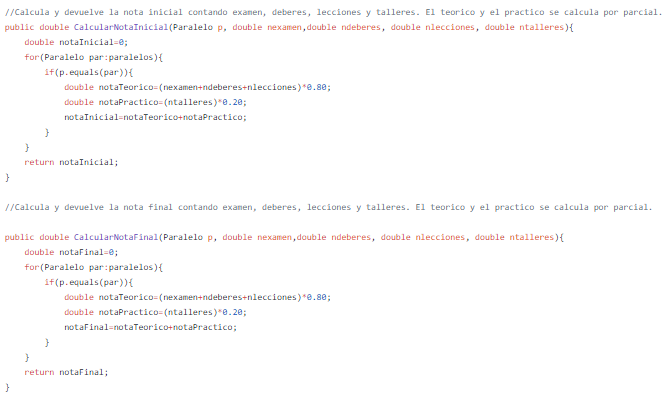
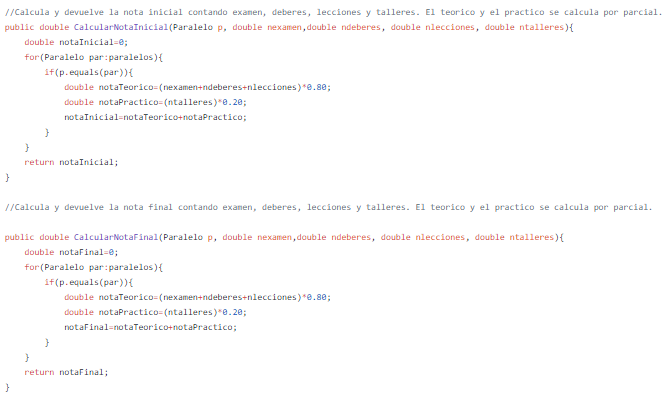


## **Long Parameter List**

En la clase Estudiante, los métodos calcularNotaInicial y calularNotaFinal que utilizan más de 4 parámetros para su código. Sin embargo, no hace falta pasar a un método toda la información que necesita para su ejecución, sino sólo aquella que es imprescindible para que pueda cumplir el fin del método.

### Consecuencia:

Una lista larga de parámetros genera una difícil compresión y legibilidad del objetivo del código, además, que al momento de llamar o probar el método, se tendrá un claro conflicto al poner muchos parámetros solo para un sencillo cálculo de notas.



### Solución:

Se aplica **EXTRACT METHOD** para poder crear el método calcularNota que reciba los parámetros que CalcularNota en la clase Estudiante recibía, luego se usa **MOVE METHOD** para ubicarlo en la clase Materia como un método estático para que así no sea necesario instanciar el objeto.

