ESCUELA SUPERIOR POLITECNICA DEL LITORAL

**TALLER REFACTORING**

DISEÑO DE SOFTWARE

Ein Bild, das Screenshot enthält.

Automatisch generierte Beschreibung

**PARALELO 2**

*LUIS ABARCA*

*MIGUEL PARRA*

**2020-2021**

INDICE

[1. Temporary Field 1](#_Toc48247668)

[1.1. Consecuencias 1](#_Toc48247669)

[1.2. Técnica de Refactorización 1](#_Toc48247670)

[1.3. Código Inicial 1](#_Toc48247671)

[1.4. Código Final 1](#_Toc48247672)

[2. Lazy Class 2](#_Toc48247673)

[2.1. Consecuencias 2](#_Toc48247674)

[2.2. Técnica de Refactorización 2](#_Toc48247675)

[2.3. Código Inicial 2](#_Toc48247676)

[2.4. Código Final 2](#_Toc48247677)

[3. Feature Envy 3](#_Toc48247678)

[3.1. Consecuencias 3](#_Toc48247679)

[3.2. Técnica de Refactorización 3](#_Toc48247680)

[3.3. Código Inicial 3](#_Toc48247681)

[3.4. Código Final 3](#_Toc48247682)

# Temporary Field

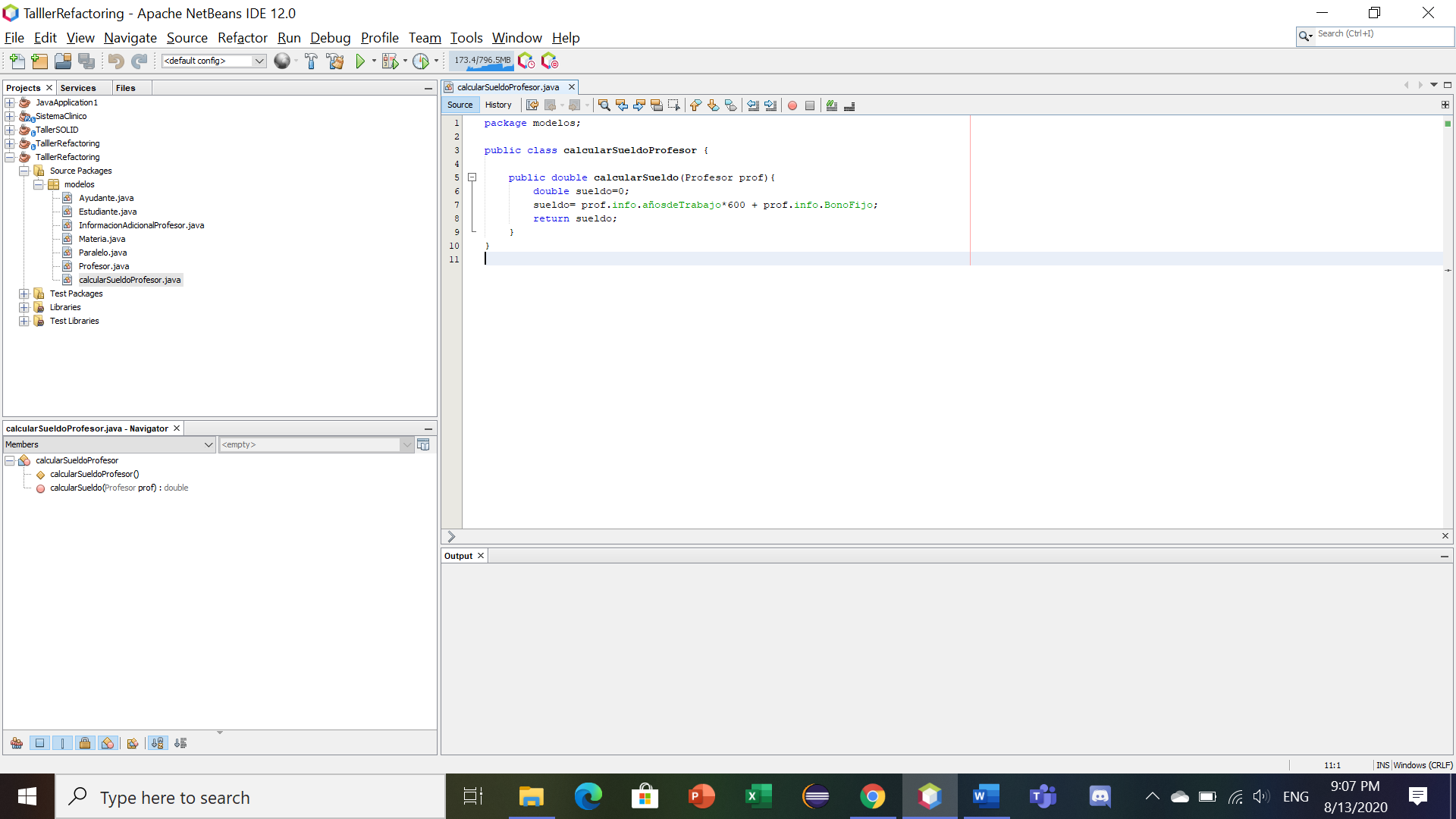
## Consecuencias

En ciertos casos, es correcto declarar una variable que contenga una expresión compleja y larga, pero en este código no aplica el caso ya que simplemente la variable “sueldo” está usando los getters de la clase Profesor. Si se mantiene asi el código, al crear la variable “sueldo” se estaría ocupando espacio en memoria innecesario para una simple expresión. Además, el código del método crece y eso hace que sea más complejo de leer.

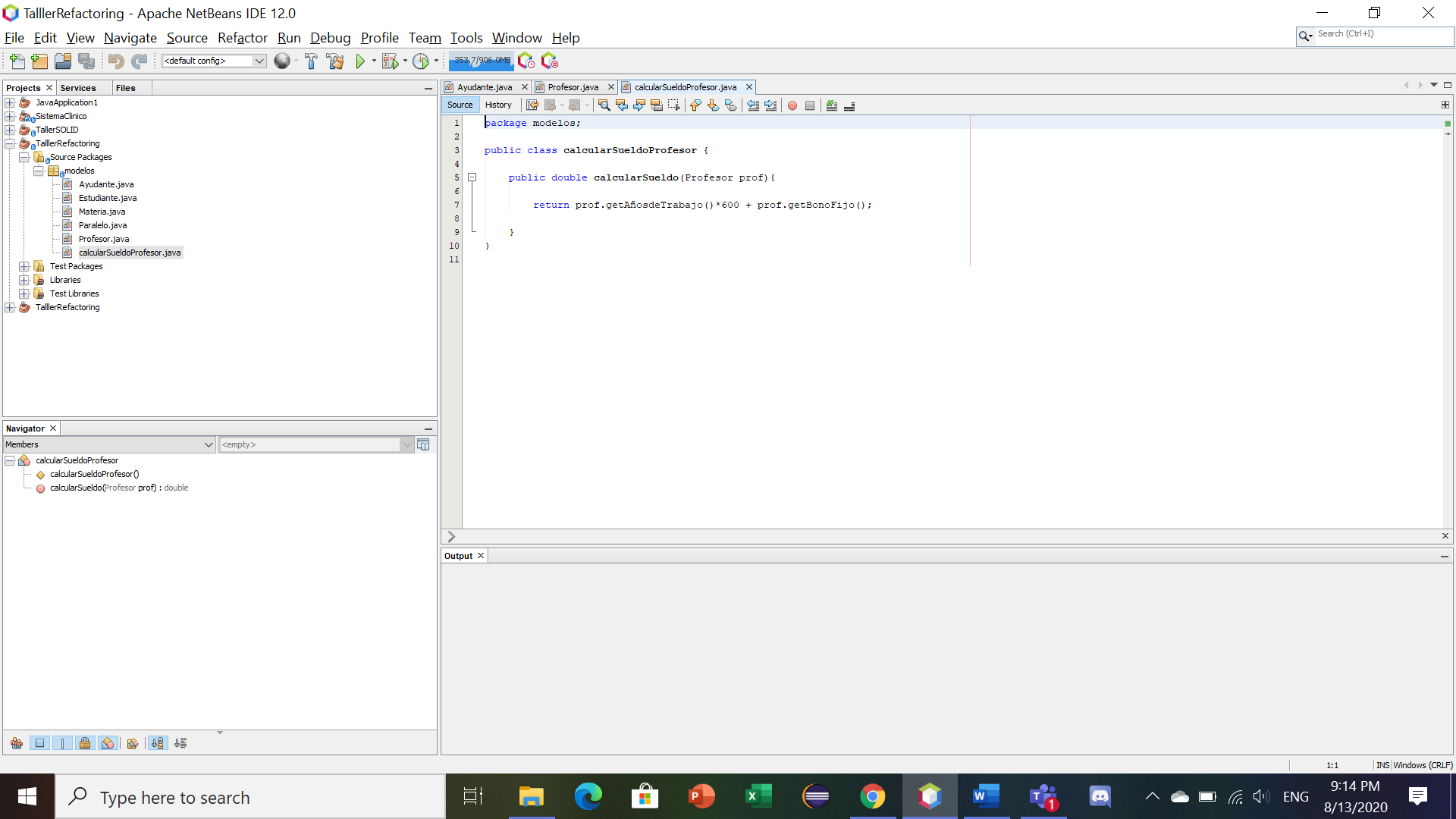
## Técnica de Refactorización

Para resolver este code smell se empleará Inline Temp, esto es, se eliminará la declaración e inicialización de la variable “sueldo” y directamente se retornará la expresión que este contenía.

## Código Inicial



## Código Final



# Lazy Class

## Consecuencias

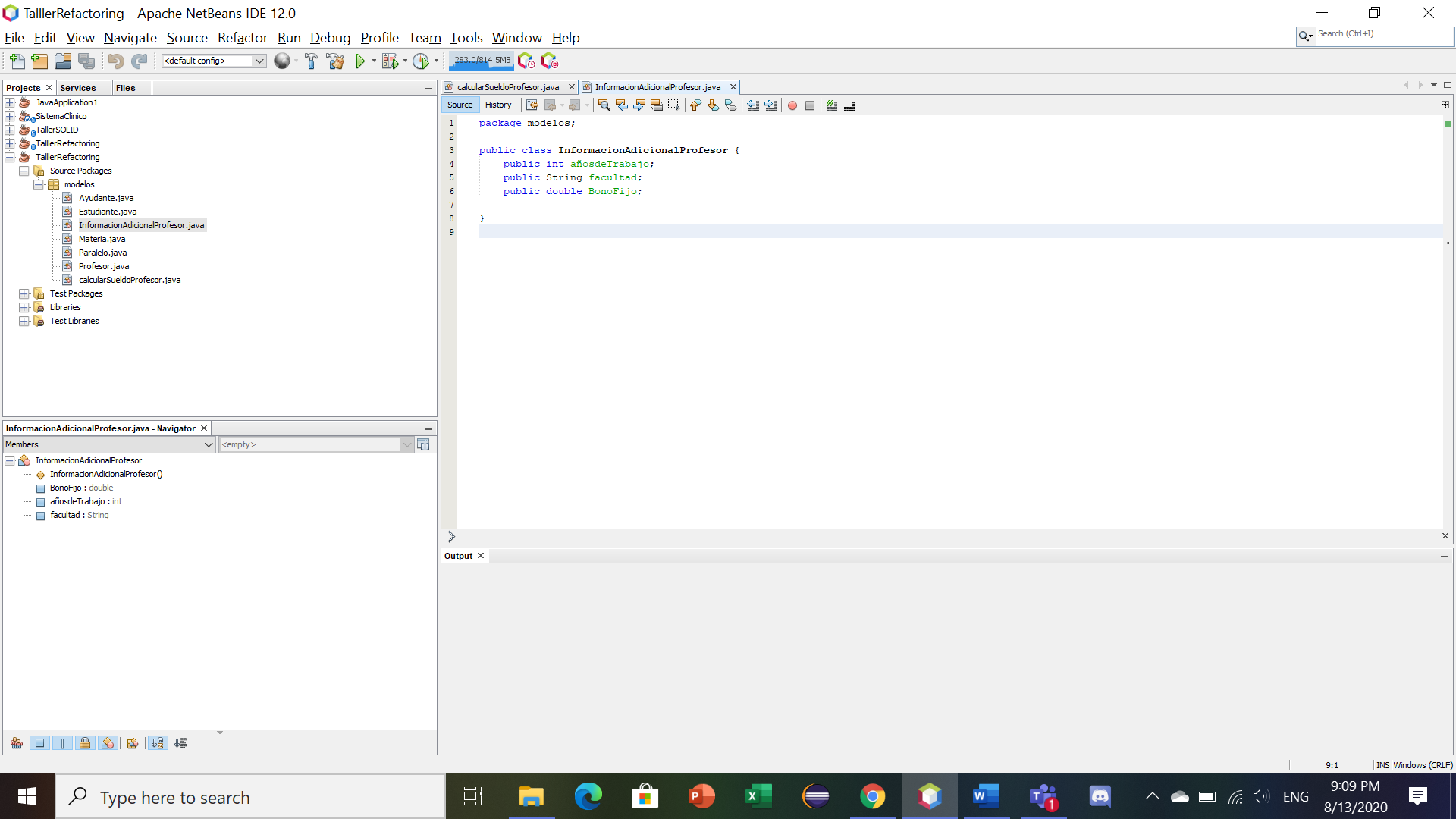
Si se mantiene esta clase, el proyecto se verá muy grande. También podría generar confusiones a la hora de modificar el proyecto en un futuro.

## Técnica de Refactorización

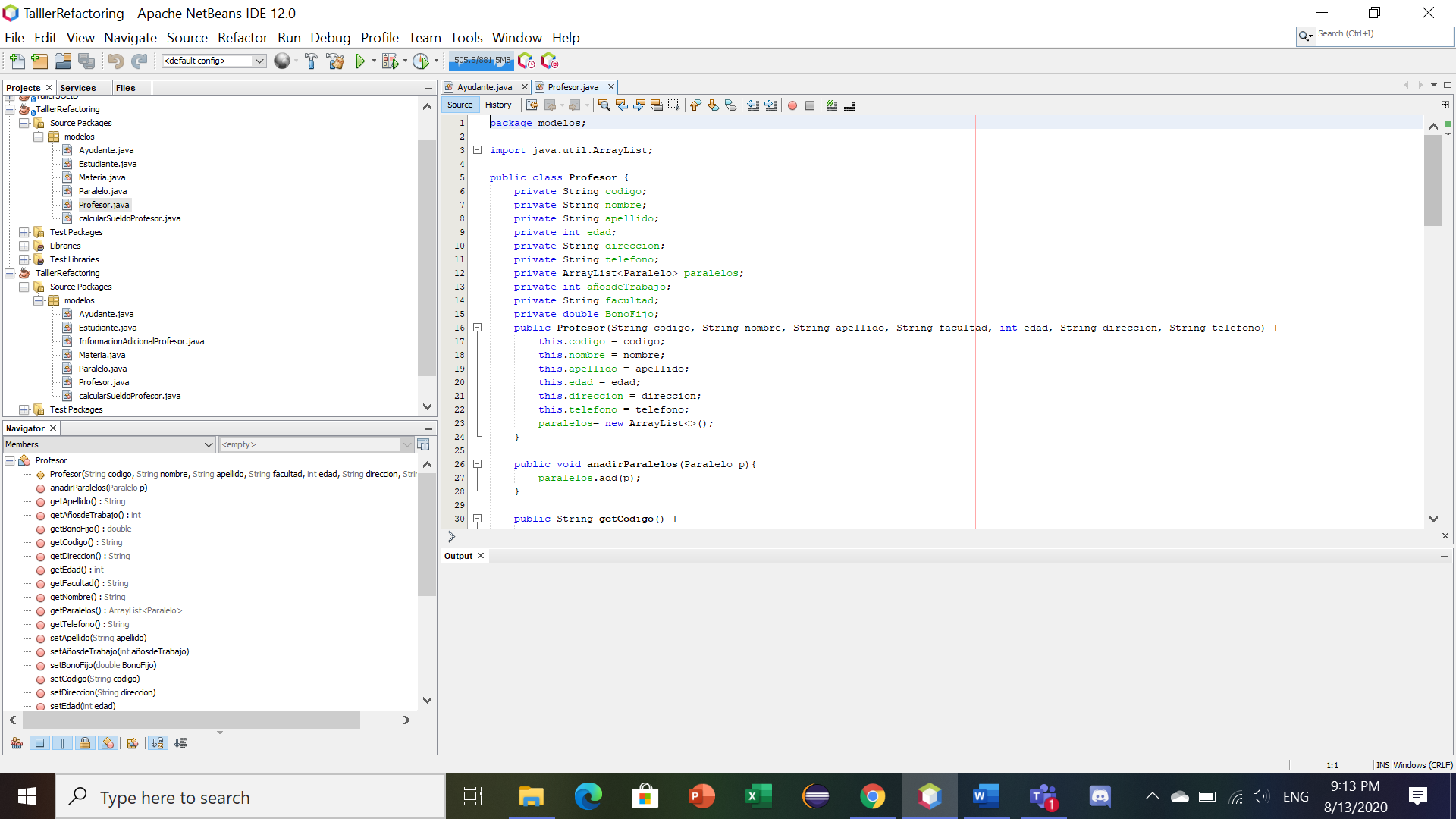
Para resolver este code smell se utilizará Inline Class, esto es, los atributos de esta clase se los añadirá a la clase Profesor y a su vez se la eliminará.

La clase InformacionAdicionalProfesor no se hace mucho, simplemente contiene 3 atributos que le pertenecen a los objetos Profesor. Además, si no se llega a eliminar, de igual forma a esta clase le faltaría un atributo de tipo Profesor, de tal manera pique se conozca a que profesor en particular le pertenecen dichos atributos, sino la clase quedaría completamente aislada de las demás.

## Código Inicial



## Código Final



# Feature Envy

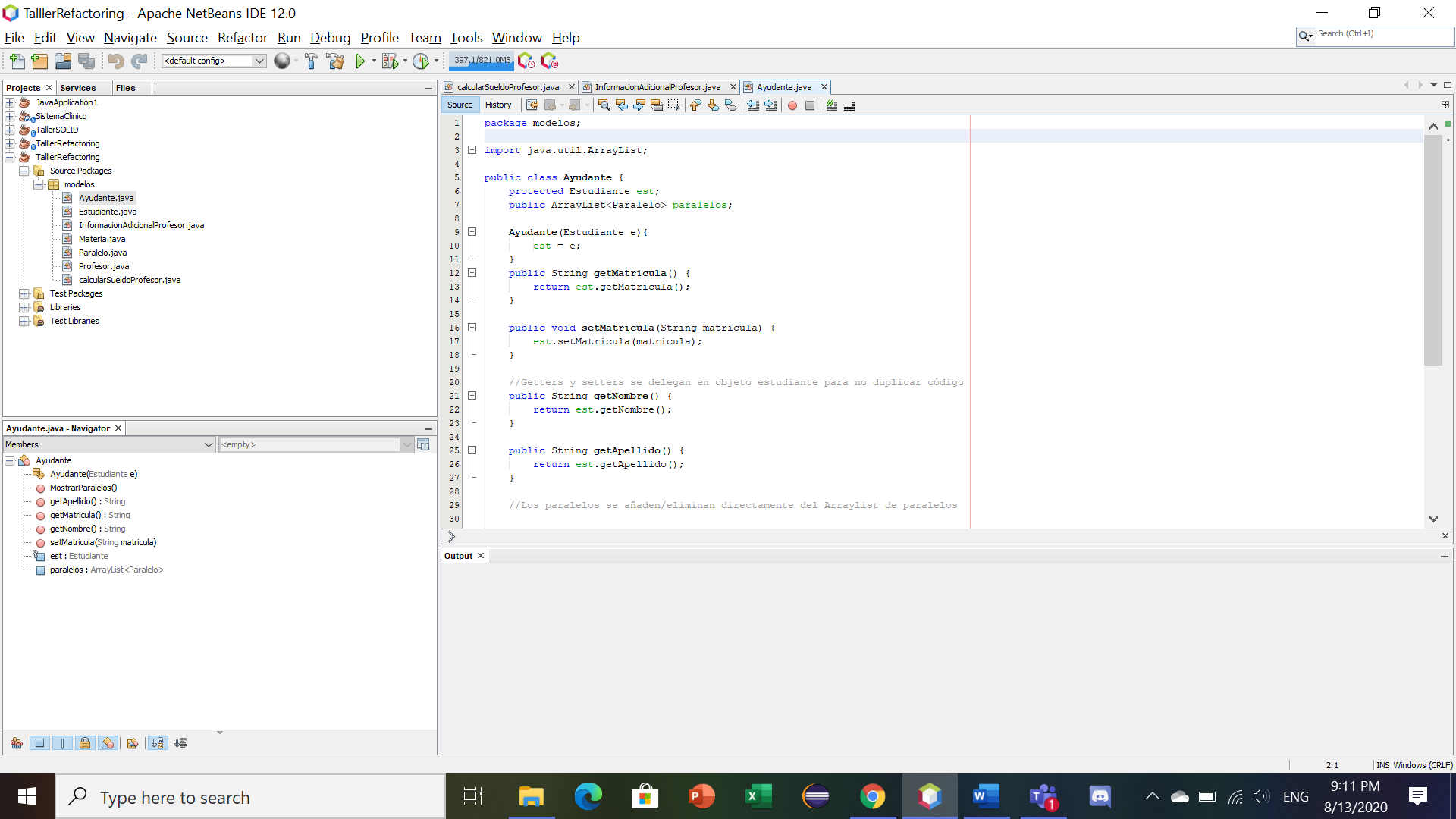
## Consecuencias

Se duplica el código, esto es, las clases Ayudante y Estudiante contiene el mismo código. A su vez, se puede notar que existe alto acoplamiento entre ambas clases, es decir, si se llega a eliminar código en la clase Estudiante puede que este le afecte a la operabilidad de la clase Ayudante.

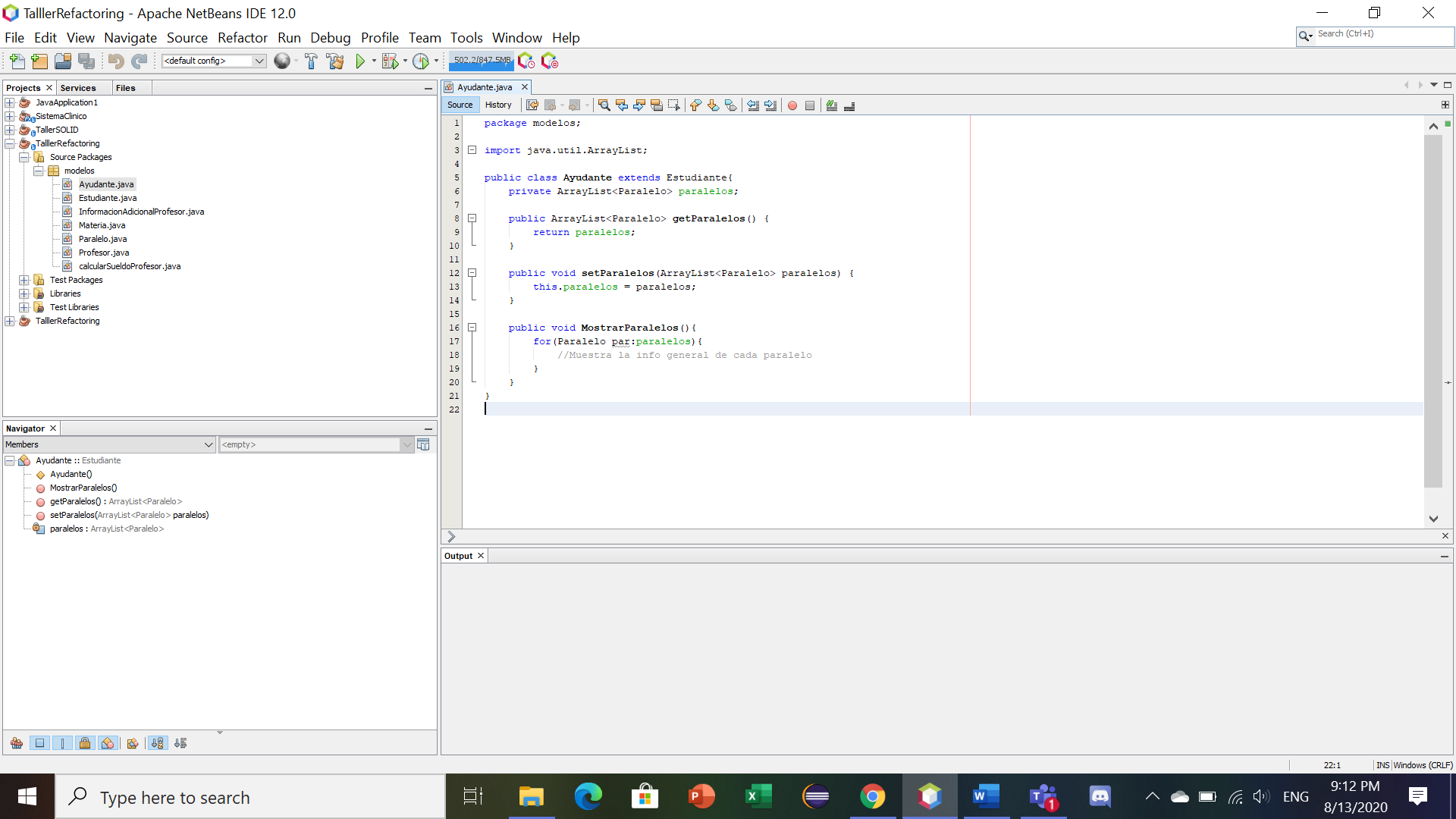
## Técnica de Refactorización

La técnica que se aplicará para resolver este code smell será Reemplazar Delegación con Herencia, ya que a través de la Herencia logramos que nuestro código en la clase Estudiante se reúse sin la necesidad de que haya código duplicado ,lo cual a su vez hace que la clase Ayudante sea más pequeña y fácil de mantener en un futuro.

## Código Inicial



## Código Final



# LARGE CLASS

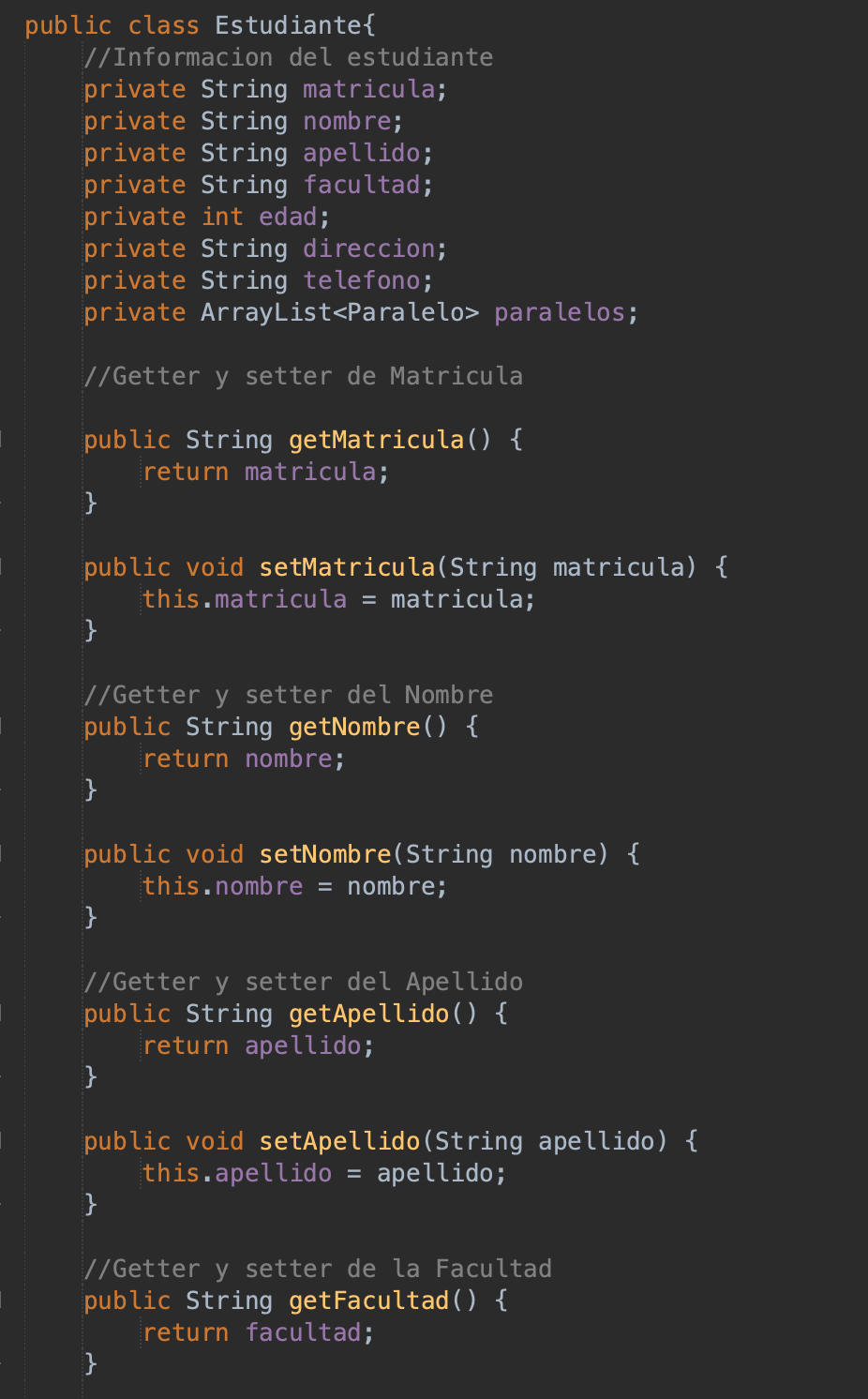
## Consecuencias

Este código efectúa el smell en el hecho de que hace que la clase se ve extremadamente grande, se lo puede visualizar en la clase estudiante y este método crea una confusión no solo para el usuario si no para el programador mismo, también incumple uno de los principios SOLID, el de single responsability cuando se implementan esa cantidad excesiva de gente.

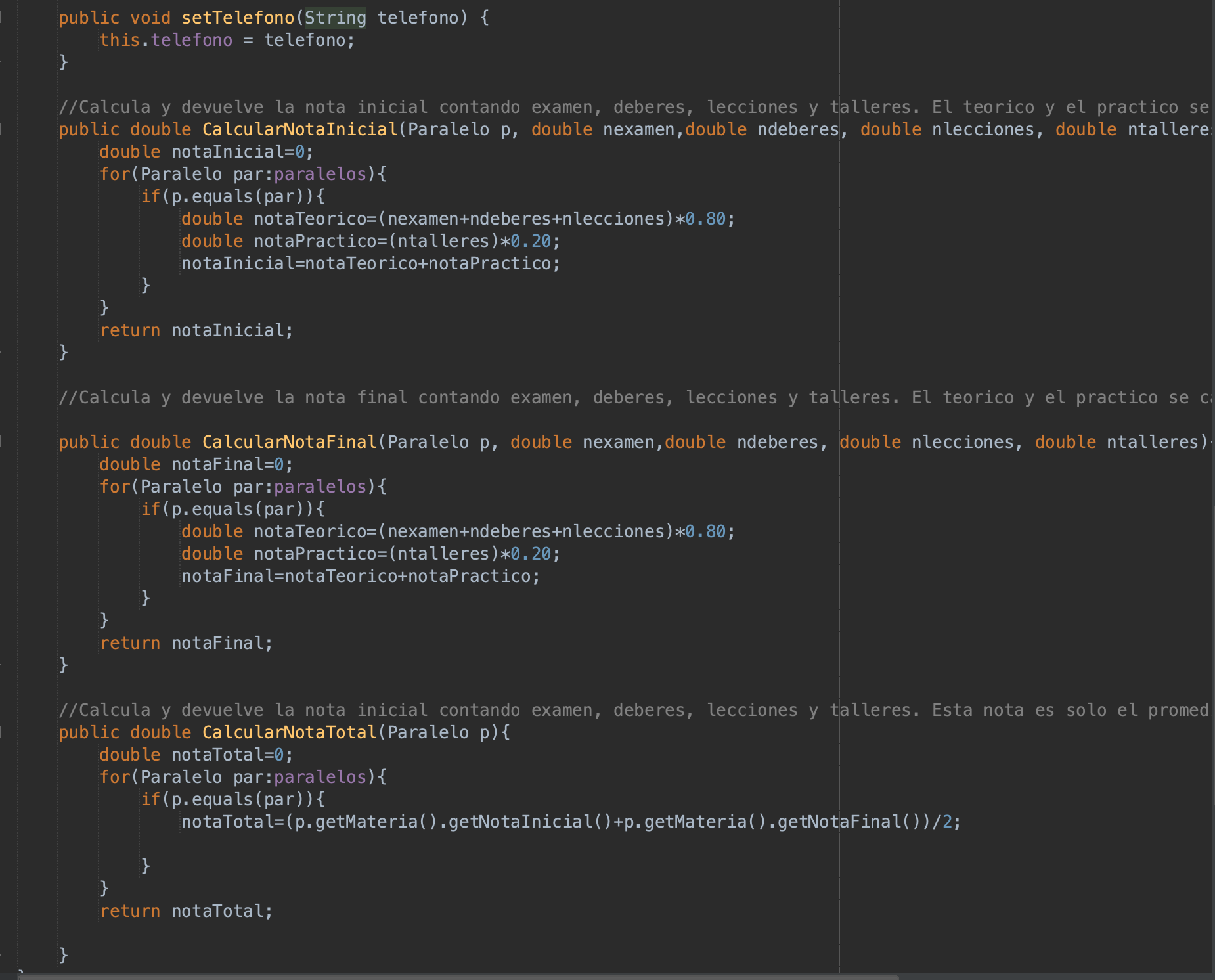
## Técnica de Refactorización

La técnica de refactorización que se usa en este smell es la de Extract class, se usó esa refactorización para poder separar muchos de los métodos que existen en esta clase y así hacer el código más flexible y más fácil de leer, se uso una nueva clase llamada obtenerNotas la cual nos informa de las notas que obtendra cada estudiante.

## Código Inicial



## Código Final



# LONG PARAMETER LIST

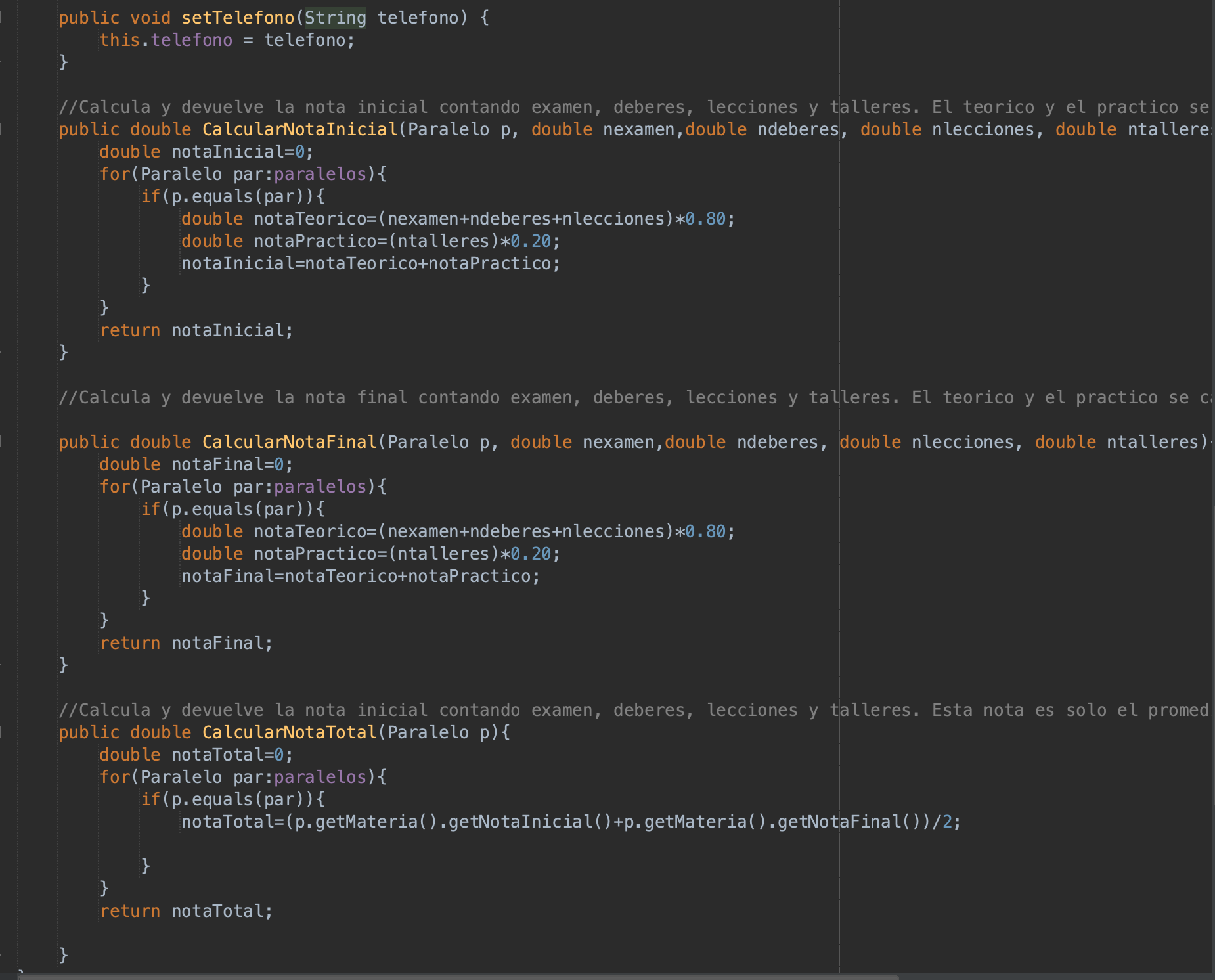
## Consecuencias

Consecuencias, una de las secuencias más importantes de tener parámetros largos es que el usuario se puede confundir con tanta información por ingresar y que también es muy poco estético haciendo que iluso de este método sea muy poco usado, también implique puede existir código muerto en el interior o que el método es muy largo

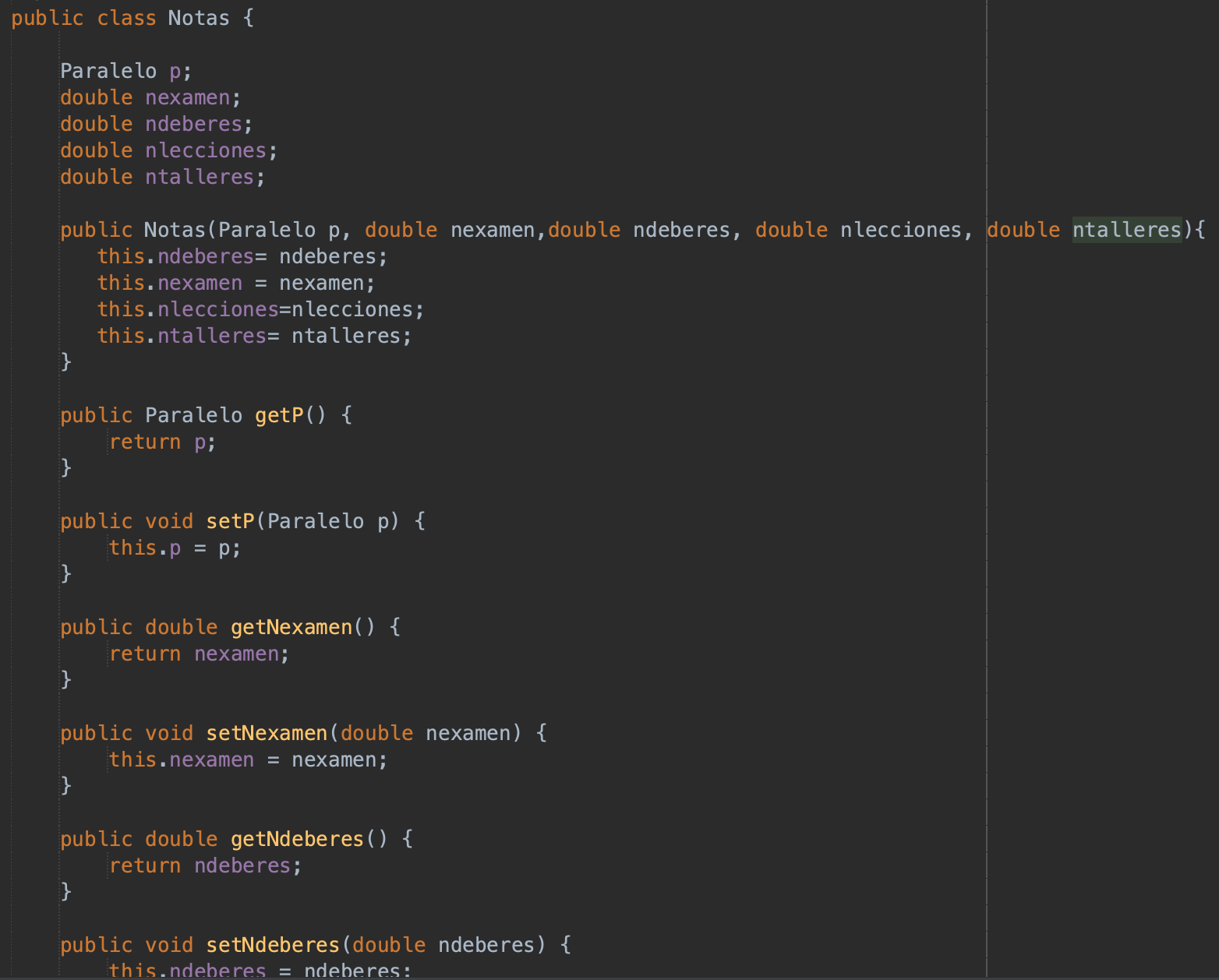
## Técnica de Refactorización

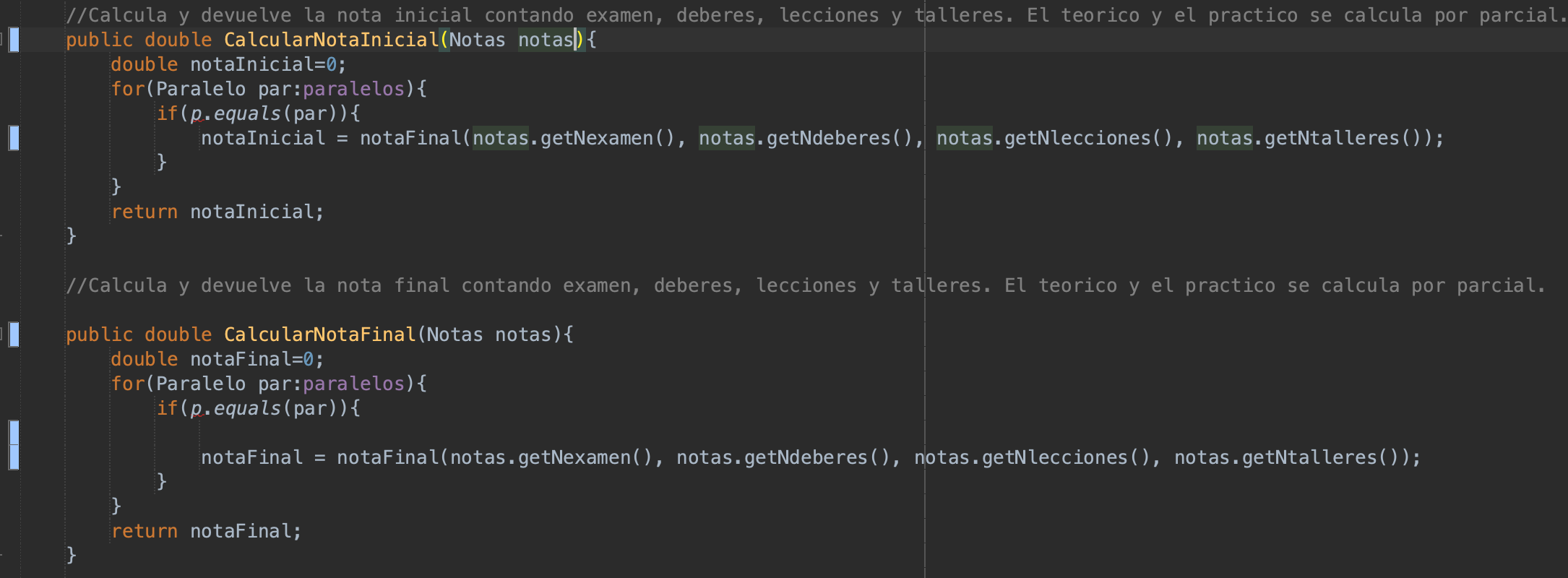
Una de las técnicas de refactorizaciones es la de distribuir responsabilidades implementando un objeto que abarque todos los parámetros necesarios para realizar la operación de este método y que solo se le ingrese un parámetro de este tipo de objeto.

## Código Inicial



## Código Final





# DUPLICADE CODE

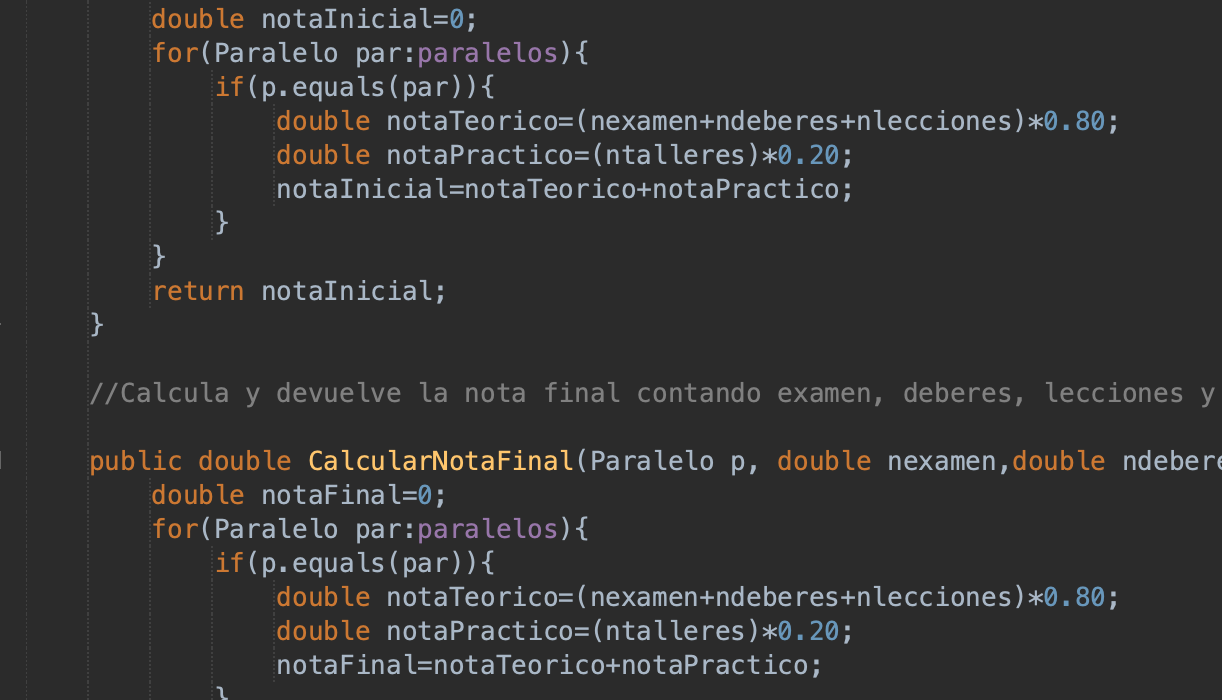
## Consecuencias

El hecho de tener un código duplicado hace que ocupe mucha memoria y se realicen procesos los cuales hacen que el código hace que sea más pesado y hace al usuario y programador propenso a confundirse

## Técnica de Refactorización

Una de las técnicas de refactorización es la de Extract method la cual nos ayuda a obtener el código que se ve involucrado en varias clases y almacenarlo en una función la cual cuando se vaya a usar ese código se la llame.

## Código Inicial



## Código Final

