

**Facultad de ingeniería en electricidad y computación**

**Diseño de software**

**Taller 4 segundo parcial**

**Tema:** Refactoring

**Integrantes:**

* Yepez Vera Tatiana Lisbeth
* Villagómez Borja Tommy Joel
* Riera Massache Julián Federico
* Pita Estrella Bryan Jorge

**Paralelo:** 2

**Profesor:** Ingeniero David Jurado

07/01/2021

PAO 2020 2T

Tabla de contenido

[Refactoring 4](#_Toc60926983)

[Duplicate Code (Code Smell) 4](#_Toc60926984)

[a) Consecuencias: 4](#_Toc60926985)

[b) Técnica de refactorización: 4](#_Toc60926986)

[c) Captura código inicial 4](#_Toc60926987)

[d) Captura código final: 5](#_Toc60926988)

[Long Parameter List (Code Smell) 5](#_Toc60926989)

[a) Consecuencias: 5](#_Toc60926990)

[b) Técnica de refactorización: 5](#_Toc60926991)

[c) Captura código inicial: 6](#_Toc60926992)

[d) Captura código final: 6](#_Toc60926993)

[Large Class (Code Smell) 7](#_Toc60926994)

[a) Consecuencias: 7](#_Toc60926995)

[b) Técnica de refactorización: 7](#_Toc60926996)

[c) Captura código inicial: 7](#_Toc60926997)

[d) Captura código final: 8](#_Toc60926998)

[Lazzy Class (Code Smell) 8](#_Toc60926999)

[a) Consecuencias: 8](#_Toc60927000)

[b) Técnica de refactorización: 8](#_Toc60927001)

[c) Captura código inicial: 9](#_Toc60927002)

[d) Captura código final: 9](#_Toc60927003)

[Tempory Field (Code Smell) 10](#_Toc60927004)

[a) Consecuencias: 10](#_Toc60927005)

[b) Técnica de refactorización: 10](#_Toc60927006)

[c) Captura código inicial: 10](#_Toc60927007)

[d) Captura código final: 10](#_Toc60927008)

[Feature Envy (Code Smell) 10](#_Toc60927009)

[a) Consecuencias: 10](#_Toc60927010)

[b) Técnica de refactorización: 10](#_Toc60927011)

[c) Captura código inicial: 11](#_Toc60927012)

[d) Captura código final: 12](#_Toc60927013)

[12](#_Toc60927014)

[Link del repositorio 12](#_Toc60927015)

# Refactoring

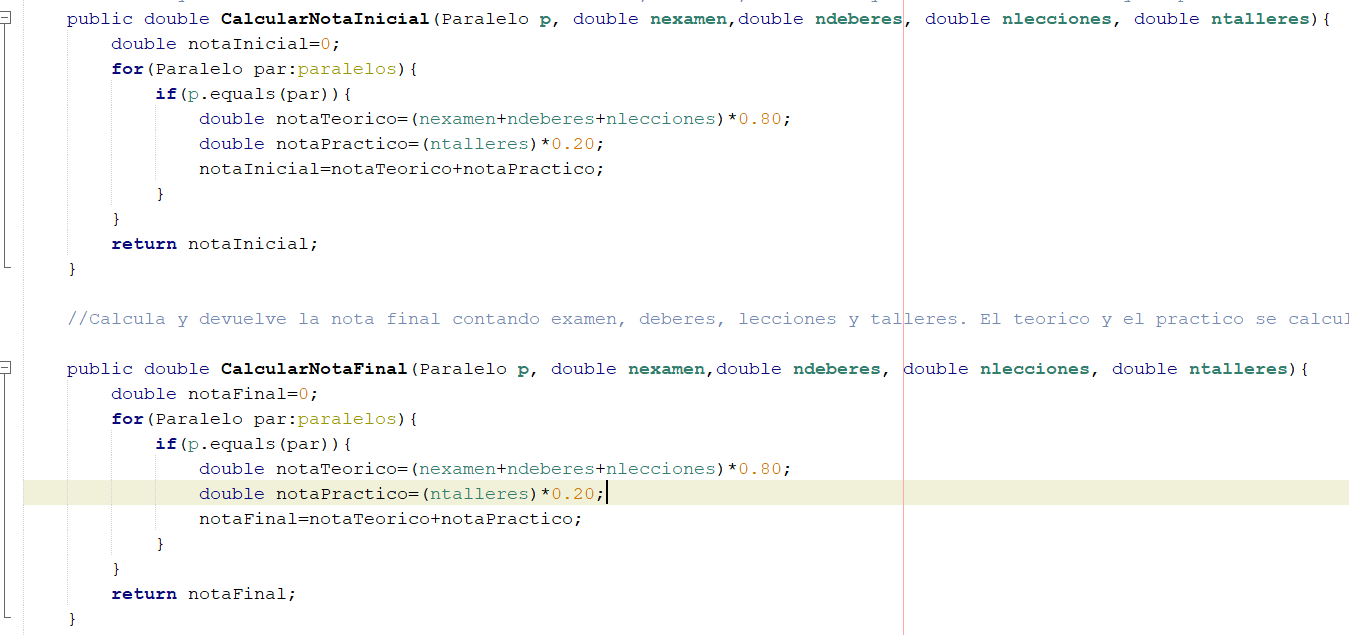
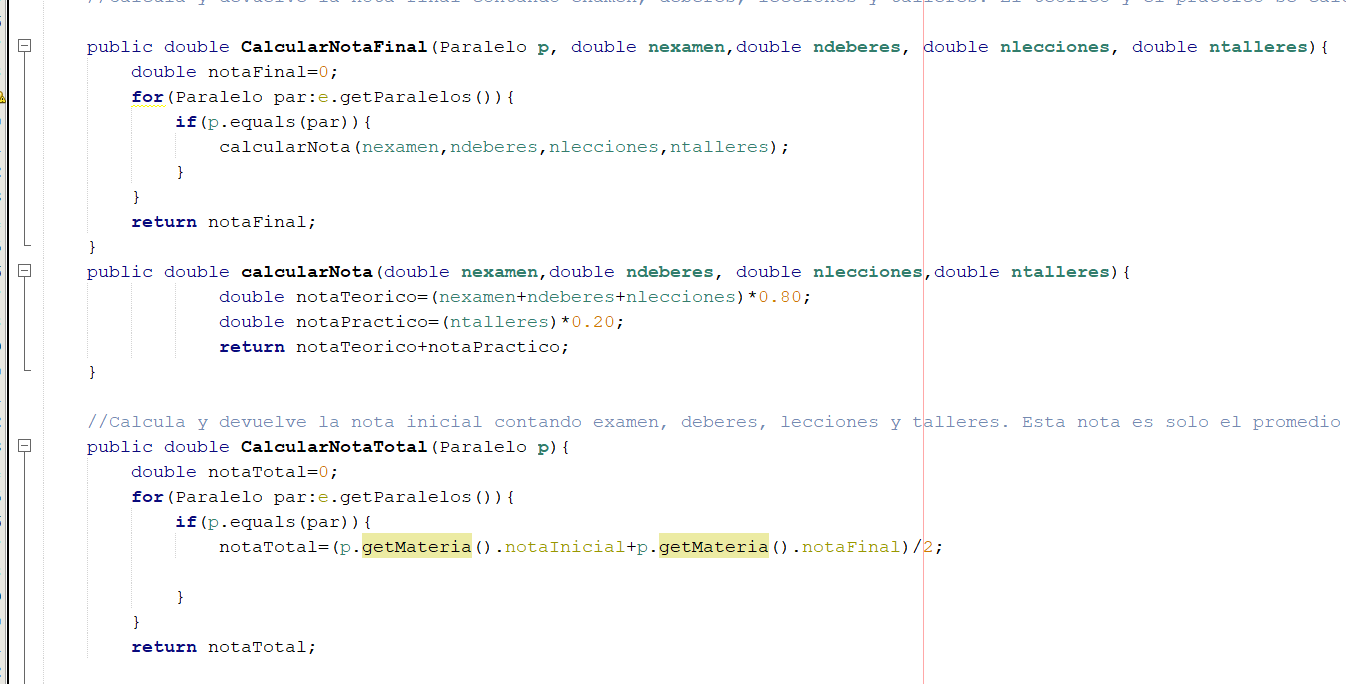
Duplicate Code (Code Smell)

1. Consecuencias:

Esto puede causar que el código necesite más memoria, de tal manera que lo vuelve mas complejo y a la vez más difícil de entender.

1. Técnica de refactorización:

Para poder solucionar esto podemos usar una técnica de refactorización Extract Method, esta nos va a ayudar a obtener el código duplicado en varios métodos o clases y almacenarlo en un solo método y podemos llamarlo cuando sea necesario, sin tener que volver a escribir demasiado código.

1. Captura código inicial
2. Captura código final:

Long Parameter List (Code Smell)

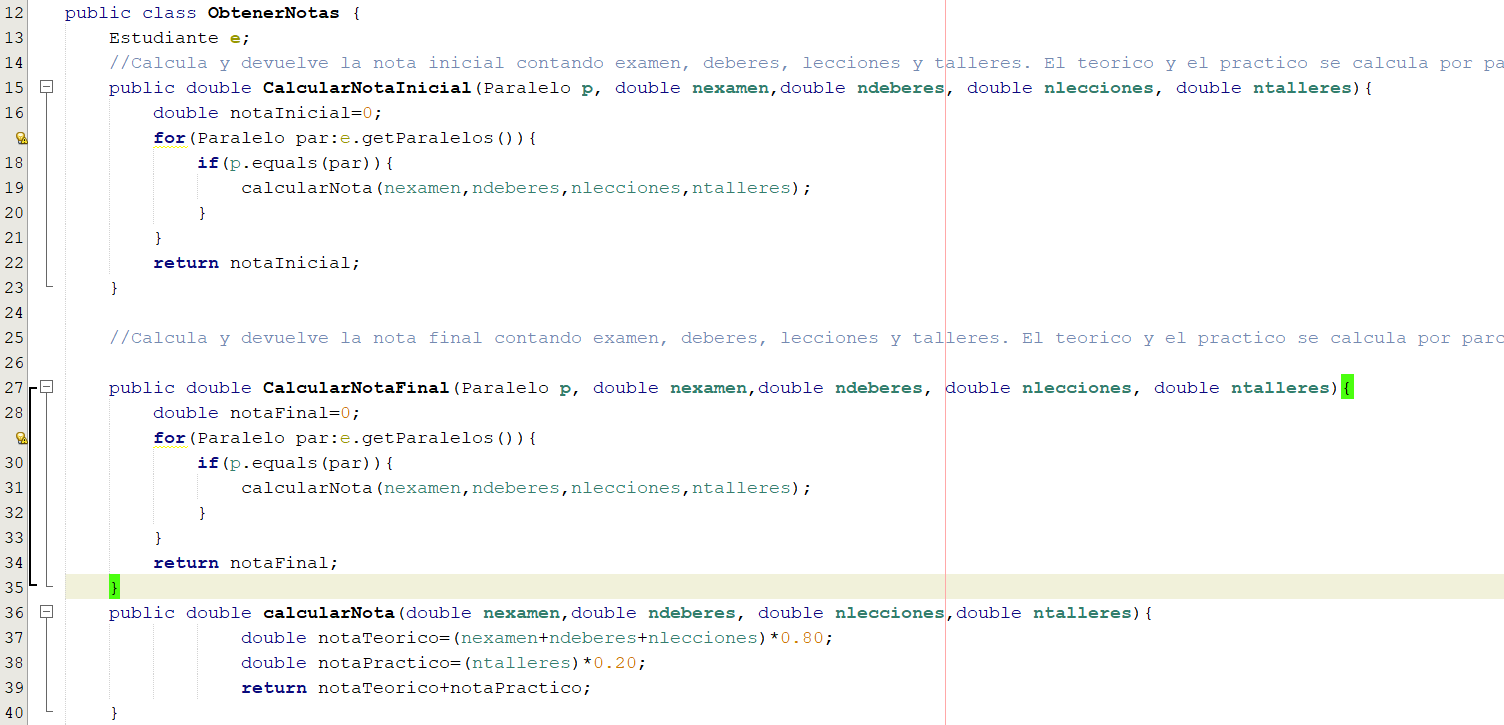
1. Consecuencias:

El usuario puede tener serios problemas al encontrarse con un programa con parámetros muy largos, debido a que tiende a confundir al propio usuario al tener que ingresar mucha información en ese parámetro en particular. Además, puede haber código muerto dentro de dicho parámetro que se hace difícil de identificar ya que su contenido es extenso. También se ve muy mal estéticamente hablando.

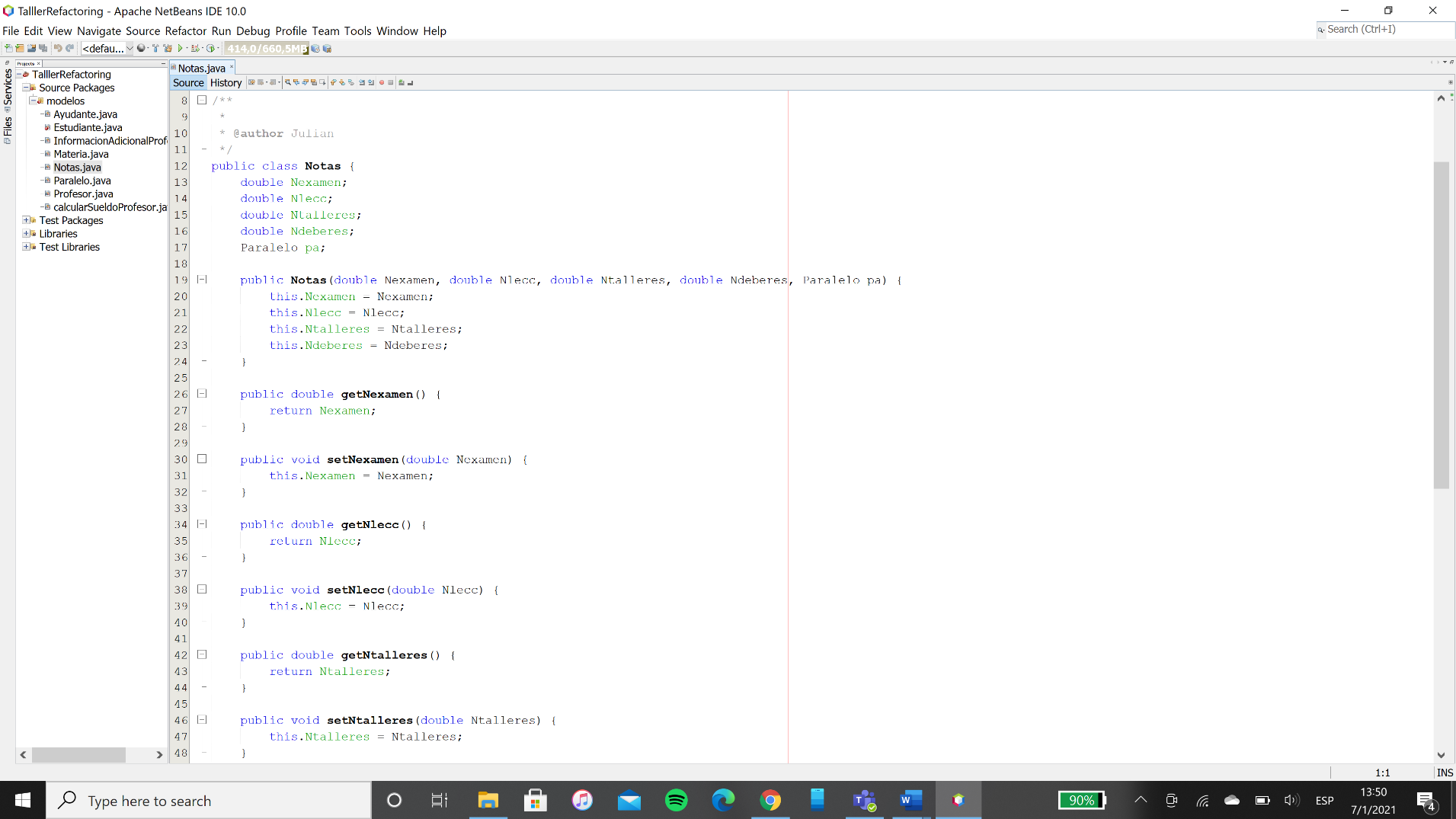
1. Técnica de refactorización:

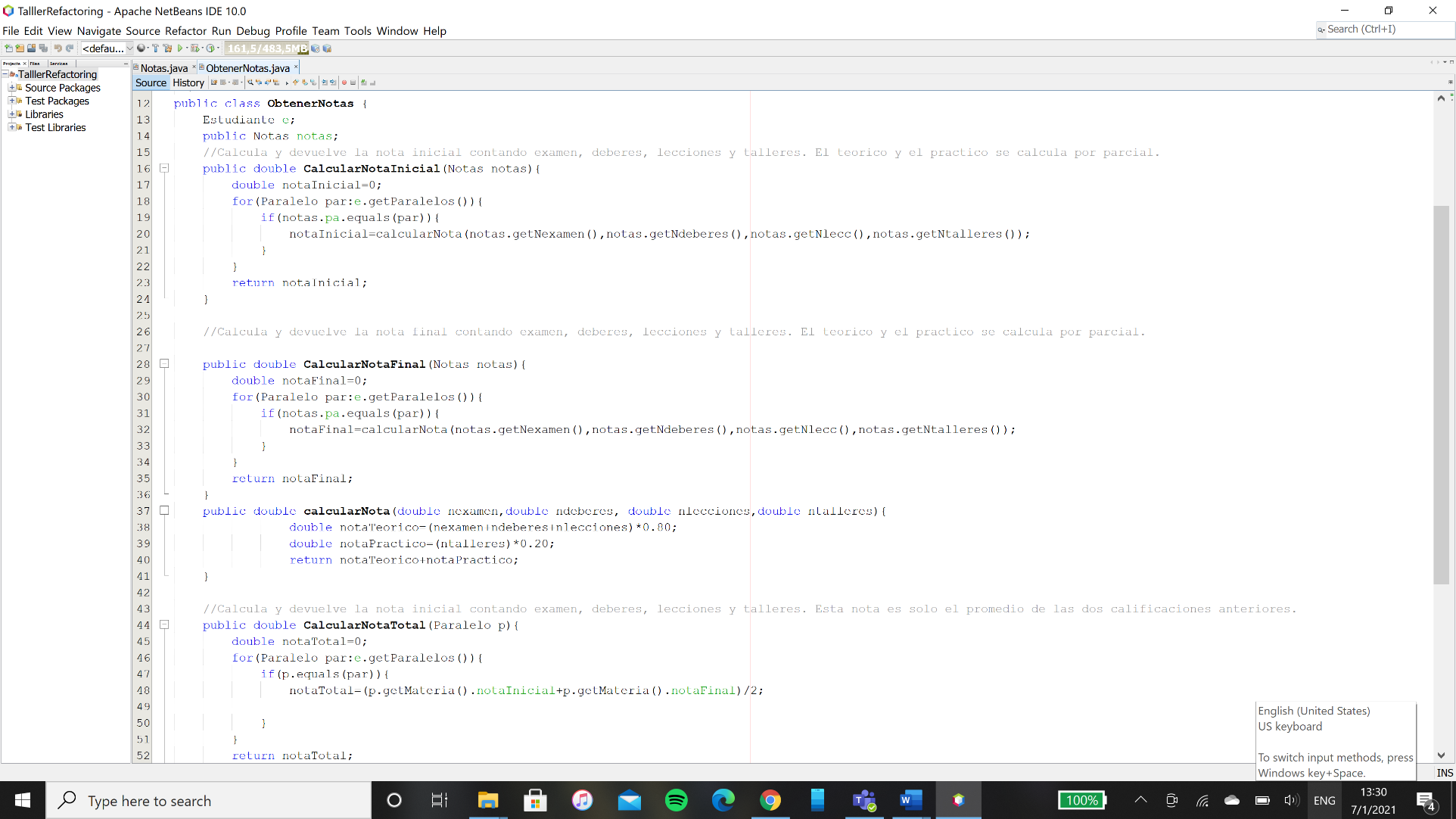
La técnica usada en este caso es la de distribución de responsabilidades. Esto se realiza haciendo un objeto que contenga los parámetros suficientes para hacer la operación dada por un método. De esta manera solo se tendrá que ingresar un parámetro de este tipo de objeto.

1. Captura código inicial:



1. Captura código final:





Large Class (Code Smell)

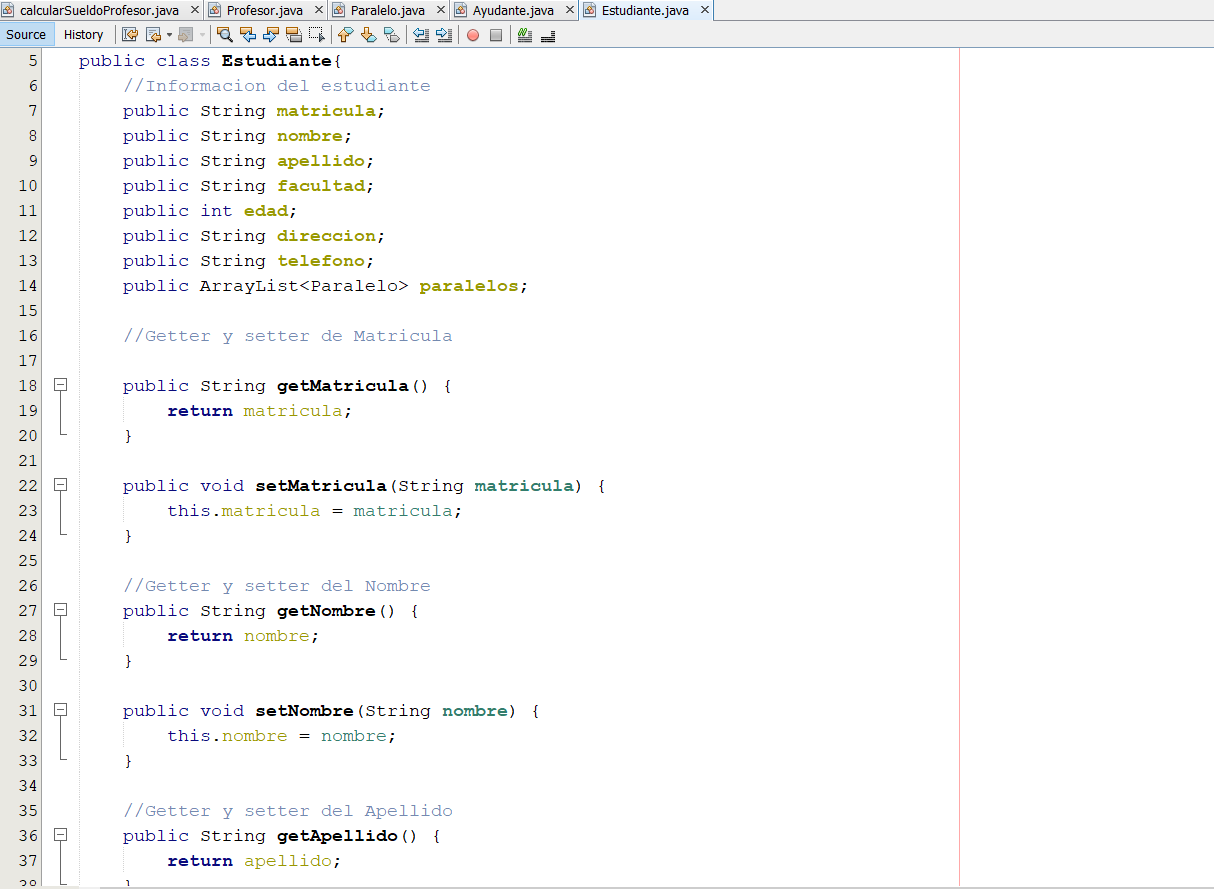
1. Consecuencias:

La clase es extremadamente grande lo cual hace el código difícil de entender y mantener, además que se viola el principio de Single responsability al usar métodos para calcular notas en la clase estudiante, lo cual necesariamente debe estar ahí

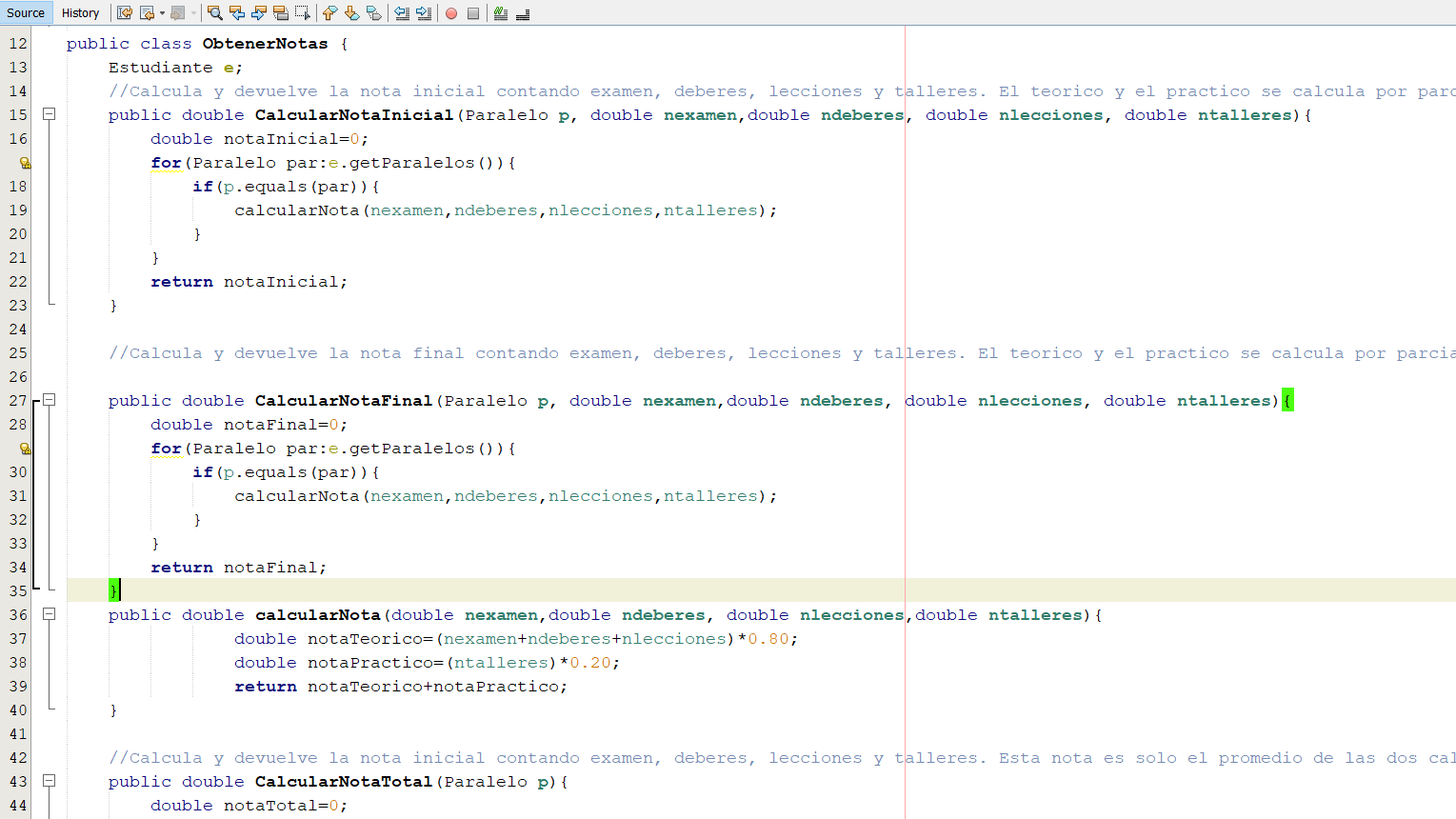
1. Técnica de refactorización:

La técnica de factorización que se utilizará para solucionar este code smell es Extract Class, se usó para crear una clase ObtenerNotas la cual contiene los métodos usando para obtener las notas de los estudiante, esto nos ayudara a tener un codigo más ordenado y fácil de entender.

1. Captura código inicial:



1. Captura código final:



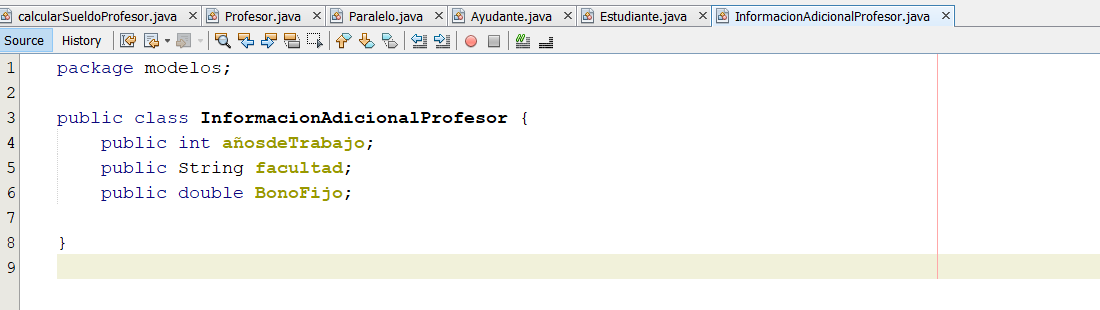
Lazzy Class (Code Smell)

1. Consecuencias:

Si se mantiene esta clase el proyecto aumenta de tamaño, además de requerir instancias de objetos adicionales para solo tres atributos de tipo primitivos (que están dentro de la clase InformacionAdicionalProfesor(), esto podría generar confusiones al modificar el proyecto en el futuro.

1. Técnica de refactorización:

Utilizada Inline class, es decir se eliminan estos atributos de la clase InformacionAdicionalProfesor() añadiéndolos a la clase profesor , lo que permite a su vez eliminar esta primera clase.

1. Captura código inicial:
2. Captura código final:

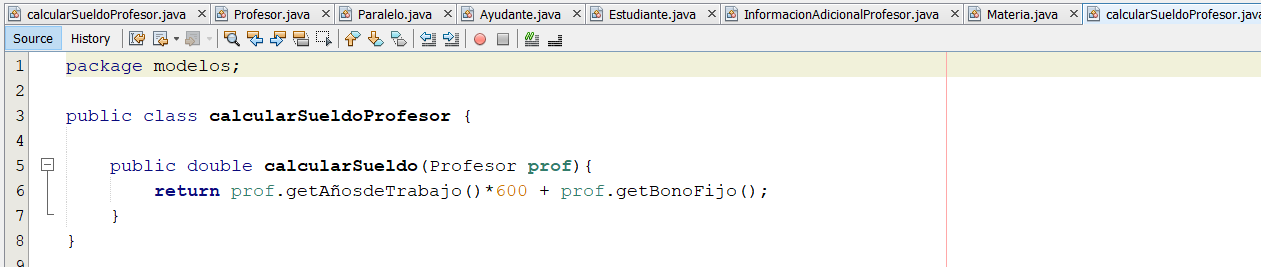
Tempory Field (Code Smell)

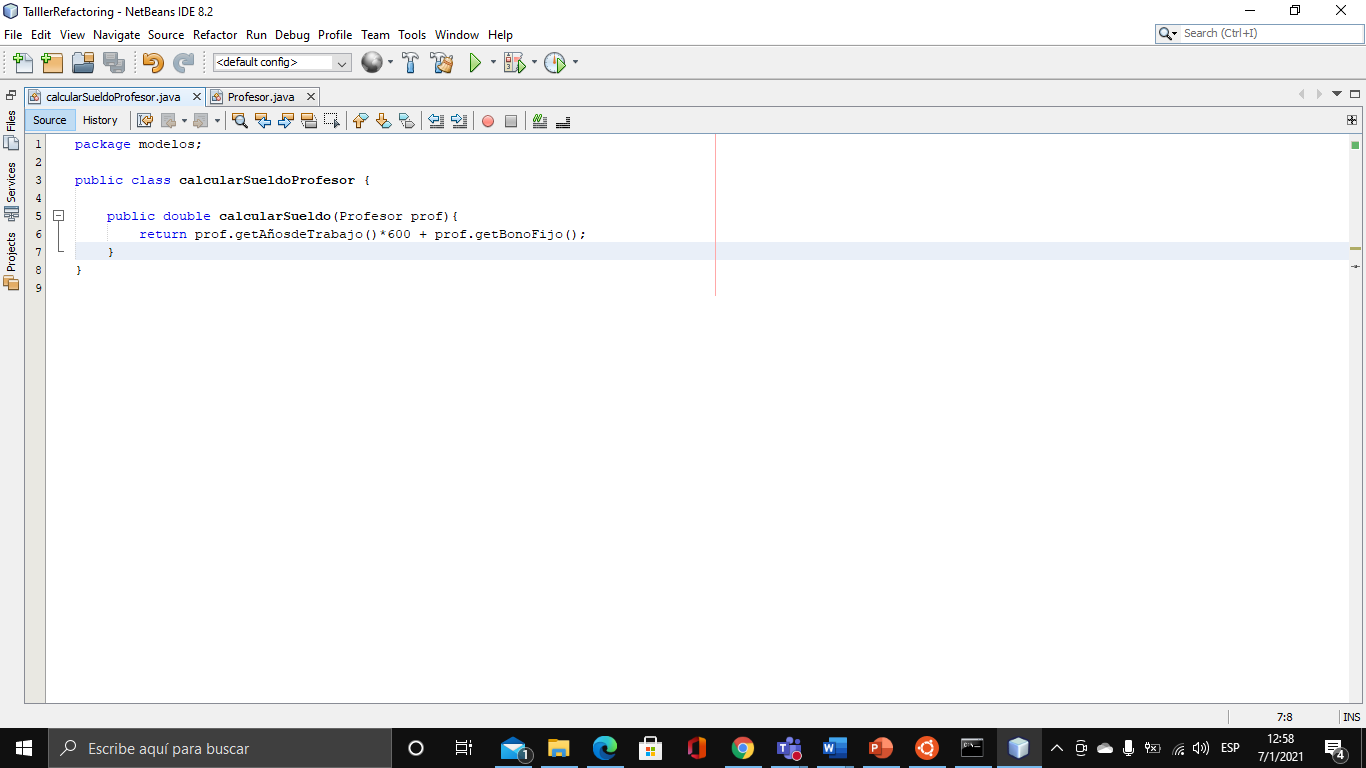
1. Consecuencias:

Crear una variable compleja para solo un return ocupa espacio innecesario de memoria además de hacer crecer innecesariamente el método en líneas de código

1. Técnica de refactorización:

Se usa inline temp, de esta forma se elimina la declaración y la inicialización de la variable sueldo, y directamente retornara la expresión (tener en cuenta que se cambia la expresión ya que se eliminó el objeto tipo info de la clase InformacionAdicionalProfesor() y se utilizó directamente los getter de los atribuidos de esta clase, debido al método de refactorización aplicado anteriormente(inline class)

1. Captura código inicial:
2. Captura código final:



Feature Envy (Code Smell)

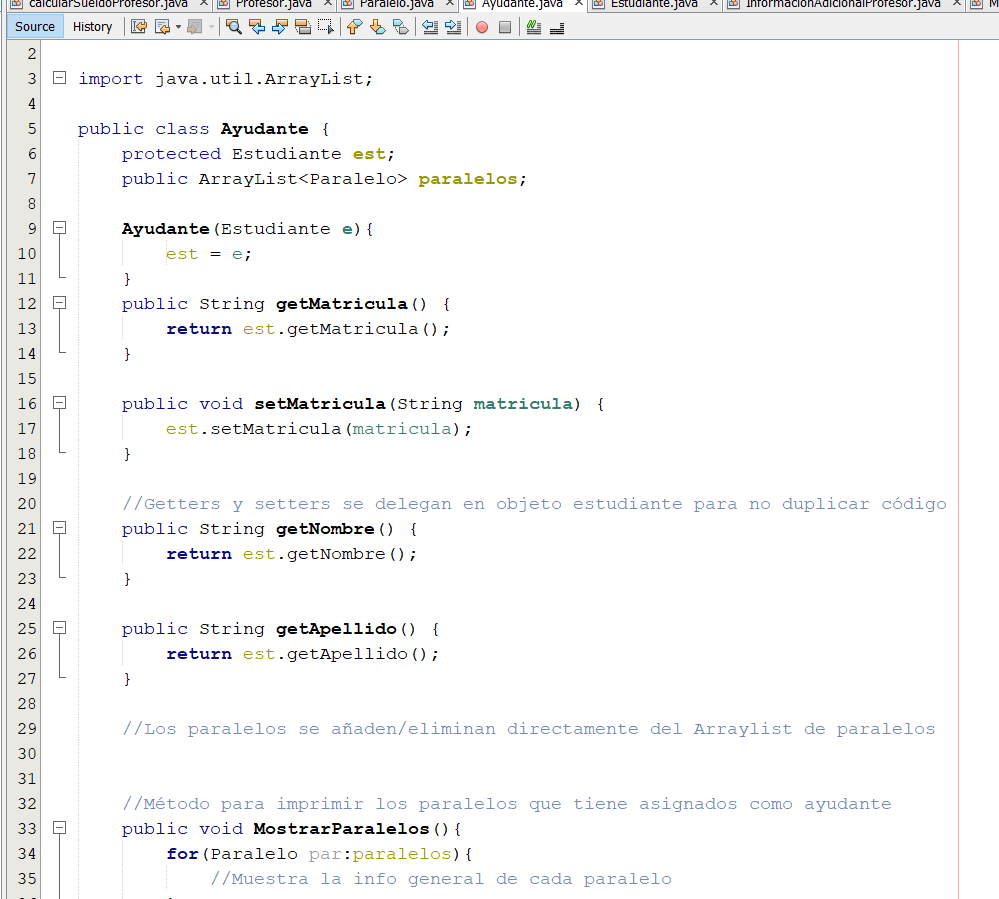
1. Consecuencias:

En el código inicial se tiene 2 clases diferentes que tienen código parecido, se puede notar que entre estas clases existe un alto acoplamiento, si se elimina o cambia la clase estudiante esto afectará a la clase ayudante.

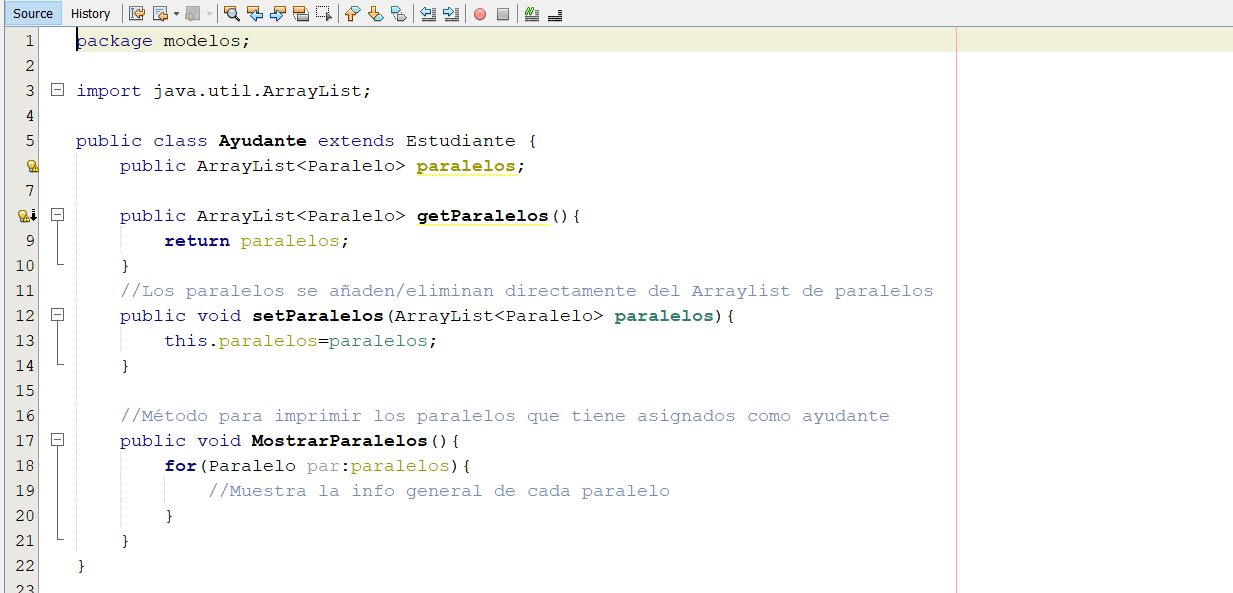
1. Técnica de refactorización:

La técnica aplicada para resolver este code smell es Reemplazar delegación con herencia, ya que a través de herencia se puede hacer que el código en la clase estudiante sea reutilizado por la clase ayudante.

1. Captura código inicial:



1. Captura código final:



## Link del repositorio

https://github.com/Jfede98/TallerRefactoring