Escuela Superior Politécnica del Litoral

Diseño de Software

**Taller #08**:

# “Refactoring”

**Grupo 3:**

Andres Moscoso

Henry Quiñonez Chávez

Sergio Basurto

**Paralelo: 1**

**PAO I 2022**

[“Refactoring” 1](#_Toc111039666)

[1.- Data class: 2](#_Toc111039667)

[2.- Lazy Class 3](#_Toc111039668)

[3.- Inappropriate-intimacy: 3](#_Toc111039669)

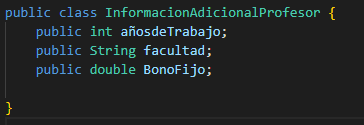
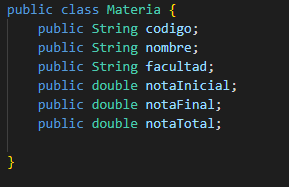
[4.-Duplicate Code: 4](#_Toc111039670)

[5.- Comments 5](#_Toc111039671)

[6.-Large Class: 7](#_Toc111039672)

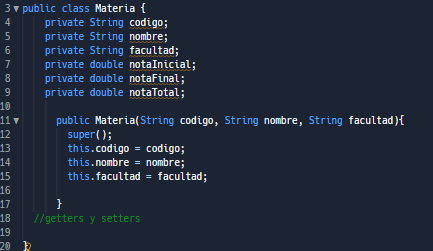
**Code Smells encontrados:**

1.- Data class: Materia.java, InformacionAdicionalProfesor.java

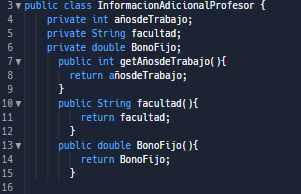


Justificación: Las clases Materia e InformacionAdicionalProfesor únicamente tienen atributos para que sean utilizados por otras clases.

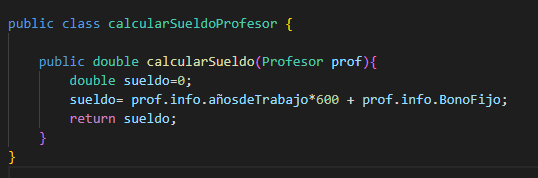
Con respecto a la clase Materia.java vamos a Usar el método de refactoring “Encapsulated Field” por lo que a los atributos públicos los convertiremos en privados y en la clase colocaremos a su propio constructor, sus getters, y sus setters.



A la clase InformacionAdicionalProfesor lo que vamos a hacer es comvertirlas en private los atributos y que se acceda por medio de los getters.

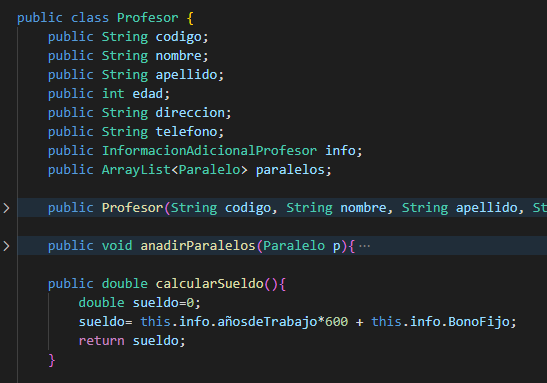


2.- Lazy Class: calcularSueldoProfesor.java



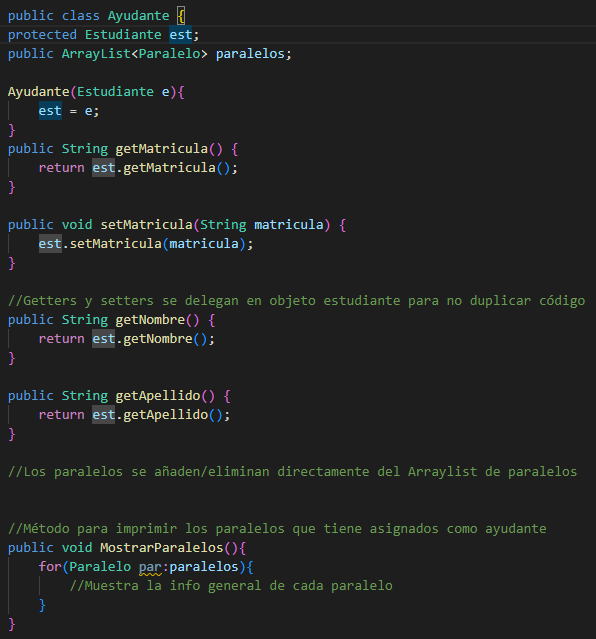
Justificación: La clase calcularSueldoProfesor contiene únicamente un método que utiliza los atributos de profesor, por lo que la clase tiene una única función, y el método podría fácilmente agregarse a Profesor.

Métodos de refactorización: Se utilizará Inline Class.



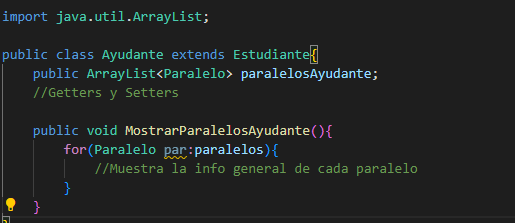
El único método ha sido transferido a la clase profesor, y se han cambiado las referencias al profesor que recibía por this, ya que se puede referir a la misma clase.

3.- Inappropriate-intimacy: Ayudante.java



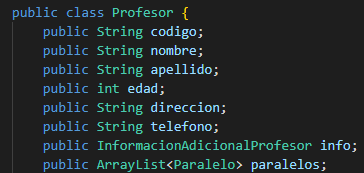
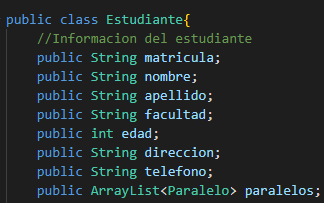
Justificación: Todos los atributos y métodos de la clase Ayudante, con la excepción de su lista paralelos, y MostrarParalelos, son delegados a la clase Estudiante, por lo que es mínima la utilidad de esta clase.

Métodos de refactorización: Se utilizará Replace Delegation with Inheritance.



Ahora, en lugar de delegar para cada atributo y método, Ayudante utiliza los heredados, y sólo necesita implementar los relacionados con su atributo de Ayudante.

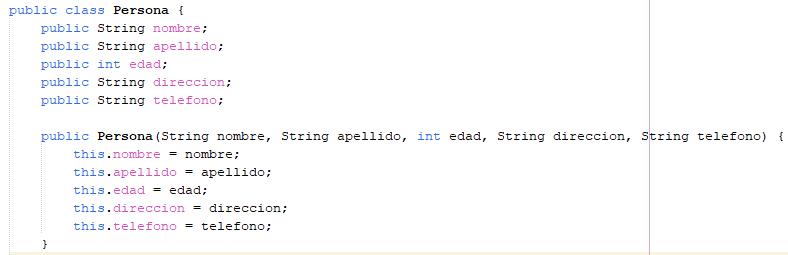
4.-Duplicate Code: Estudiante.java, Profesor.java



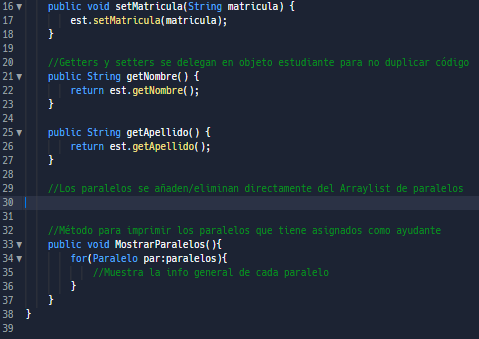
Justificación: La clase Estudiante y la clase Profesor, son excesivamente parecidos, ya que comparten la mayoría de sus atributos, por lo que se está duplicando el código de lidiar los mismos.

Métodos de refactorización: Se utilizará Extract Superclass.

Se ha creado una clase persona que contiene los atributos en común de profesor y estudiante, así estos extienden esos atributos de su clase padre.



5.- Comments: Estudiante.java, Ayudante.java

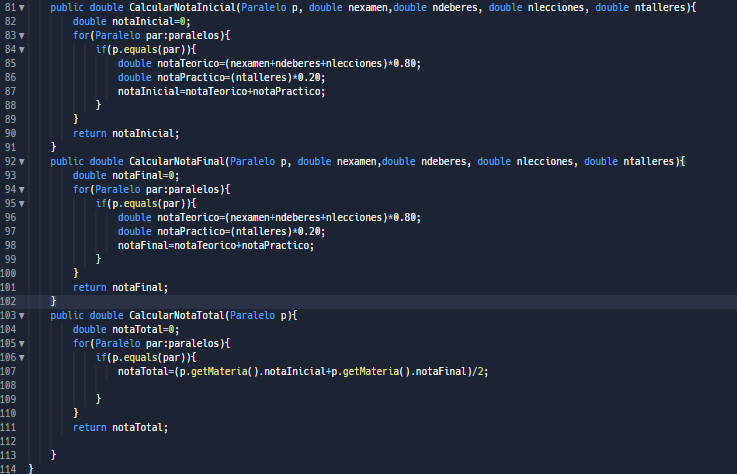
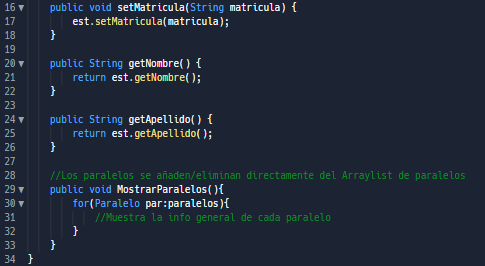


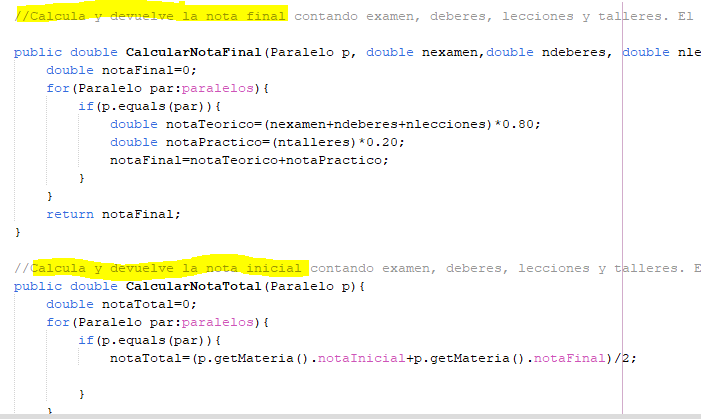




Justificación: Los comentarios en los métodos de calcular notas en la clase estudiante son innecesarios ya que la clase provee suficiente información de lo que realizan en su implementación, por ejemplo, el método CalcularNotaFinal es suficientemente explícito para entender lo que hará, además de los atributos tienen nombres entendibles.

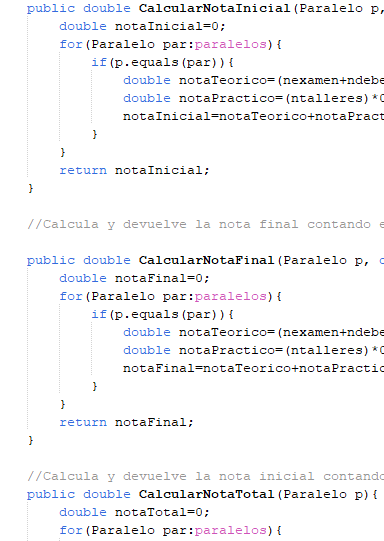
Para solucionar este code smell lo que vamos a hacer es borrar los comentarios que estan en el codigo.





6.-Large Class: Estudiante.java

Justificación: Cuando las clases comienzan a implementar muchos métodos se empieza a volver una large class, esto sucede cuando la persona que ha desarrollado el código le es menos agotador colocar más metodos en un clase que crear otra donde estos sean más acorde a la funcionalidad, por ejemplo tenemos tres clases para calcular notas.



Solución: Llevar estos métodos que han alargado la clase a otra clase más acorde, por ejemplo en una clase materia.