**ESCUELA SUPERIOR POLITECNICA DEL LITORAL**

**DISEÑO DE SOFTWARE**

**“TALLER REFACTORING”**

**R E P O R T E**

Integrantes:

**JOYCE ROJAS FALCONI**

**WESLEY BRIONES PINCAY**

**ANGIE FLORES MATIAS**

**MARCK MURILLO RONQUILLO**

**FRANKLIN ORDOÑEZ LOPEZ**

Herramienta de repositorio:

**GITHUB**

Docente:

**ING. DAVID JURADO MOSQUERA**

Enero, 2021

Tabla de contenido

[Code smell: Duplicate Code 3](#_Toc60940710)

[Código sin refactorizar: 3](#_Toc60940711)

[Código refactorizado: 3](#_Toc60940712)

[Consecuencias 3](#_Toc60940713)

[Code Smell: Long parameter list 4](#_Toc60940714)

[Código refactorizado: 4](#_Toc60940715)

[Consecuencias 4](#_Toc60940716)

[Code Smell : Inappropiate Intimacy 5](#_Toc60940717)

[Código sin refactorizar 5](#_Toc60940718)

[Consecuencias 6](#_Toc60940719)

[Code Smell: Dead code 7](#_Toc60940720)

[código sin refactorizar: 7](#_Toc60940721)

[Consecuencias 7](#_Toc60940722)

[Code Smell: Data Class 8](#_Toc60940723)

[Código sin refactorizar 8](#_Toc60940724)

[Código refactorizado 9](#_Toc60940725)

[Consecuencias 9](#_Toc60940726)

[Code Smell: Data Class 10](#_Toc60940727)

[Código sin refactorizar: 10](#_Toc60940728)

[Código refactorizado: 10](#_Toc60940729)

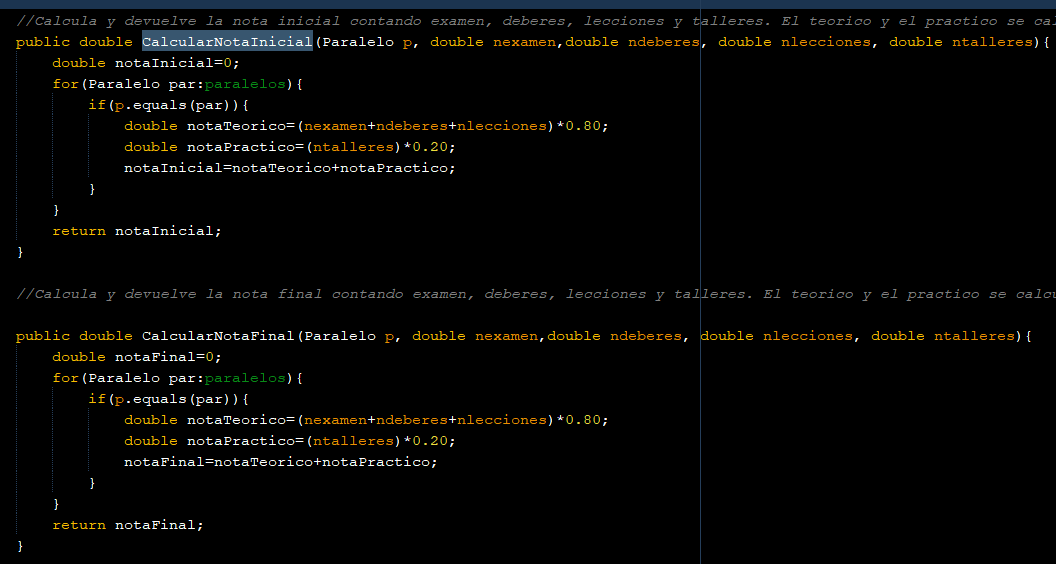
[Clase Materia: Código sin refactorizar 11](#_Toc60940730)

[Consecuencias 11](#_Toc60940731)

# Code smell: Duplicate Code

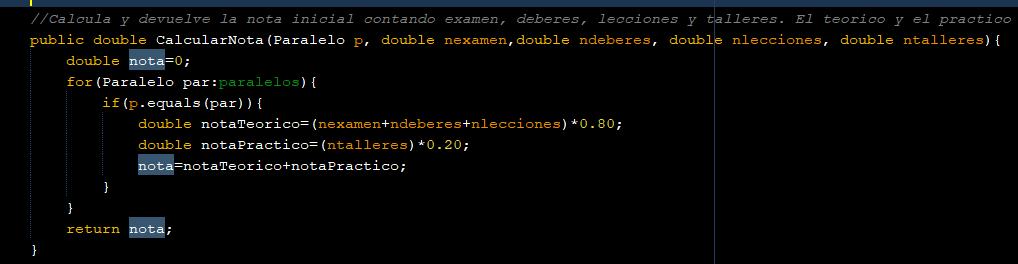
* Clase Estudiante

## Código sin refactorizar:



Se puede apreciar que los métodos de CalcuarNota inicial y final son exactamente iguales. El método de refactorización más simple aquí es unir los dos métodos en uno solo.

## Código refactorizado:



## Consecuencias

Las consecuencias de este Code Smell es que dificultad para leer el código y la existencia de métodos o porciones innecesarias que solo hacen el programa final más confuso y lento.

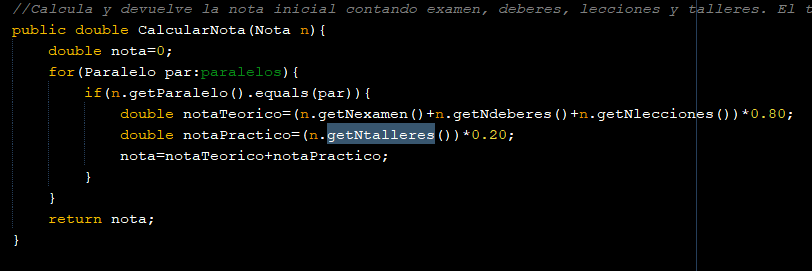
Sin embargo, todavía tenemos un problema.

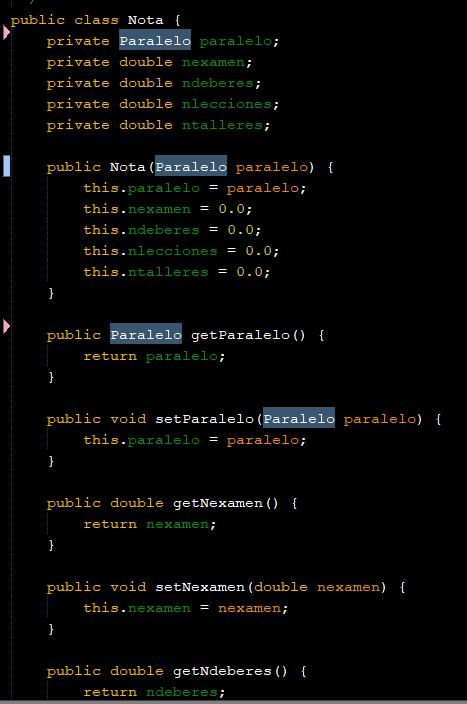
# 

# Code Smell: Long parameter list

El método tiene una lista de parámetros muy largo, esto se resuelve creando un objeto con todos los parámetros que recibe el método.

## Código refactorizado:





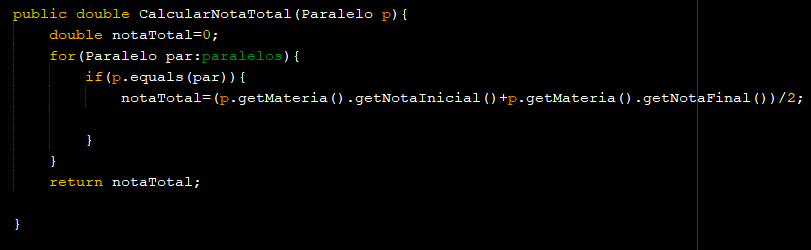
## Consecuencias

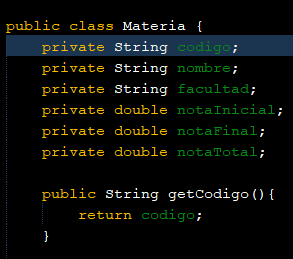
La consecuencia de este Code Smell es que sea el código más complejo de leer y refactorizar puede darnos indicios de código duplicado.

# Code Smell : Inappropiate Intimacy

* Clase Estudiante y clase Materia

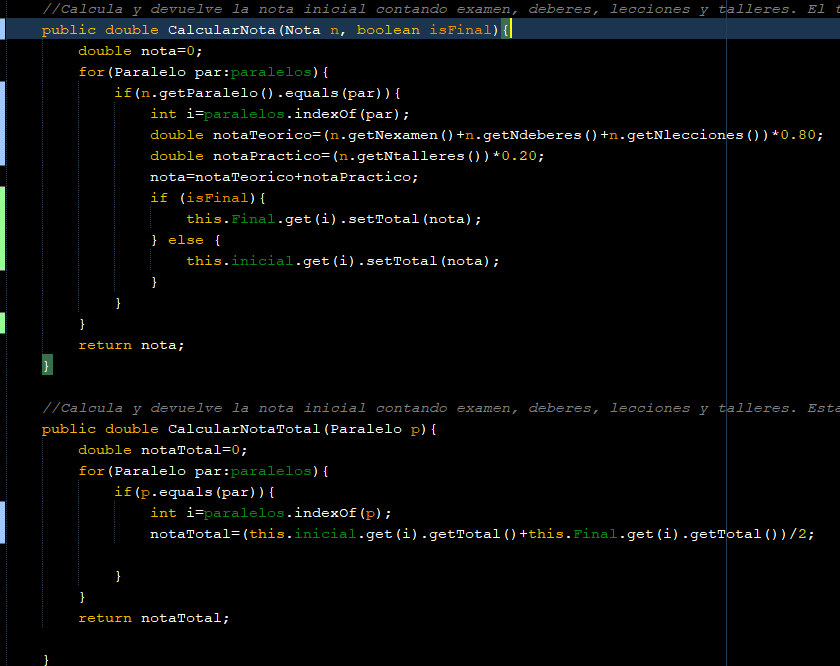
## Código sin refactorizar



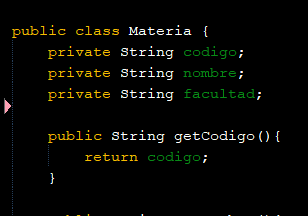


Se puede apreciar que tiene atributos que no son respectivos del objeto Materia, sino que tendría más sentido que fueran parte de Estudiante puesto que se trabaja con ellos

La solución de esto no es tan simple como pasar los atributos a estudiantes, puesto que un estudiante tiene diferentes notas para cada materia. Por lo tanto, se ha usado la misma clase que usamos para el anterior code smell para representar esto, añadiéndole un atributo total y dejando los métodos así:



Y la clase Materia así:



## Consecuencias

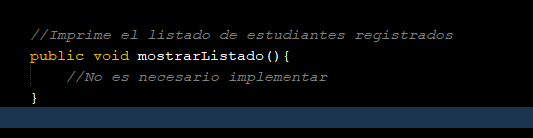
Las consecuencias de este code smell es una mala organización de nuestro código y confusión del mismo al momento de querer modificarlo o mejorarlo.

# 

# Code Smell: Dead code

* Clase Paralelo

## código sin refactorizar:



En la clase Paralelo se aprecia el inicio de la definición de un método que nunca se llegó a implementar, quizás, porque ya no era necesario. Sin embargo, su parte de código quedó como código muerto.

La forma de refactorización más rápida para lidiar con este Code Smell es simplemente eliminar el método sin definir.

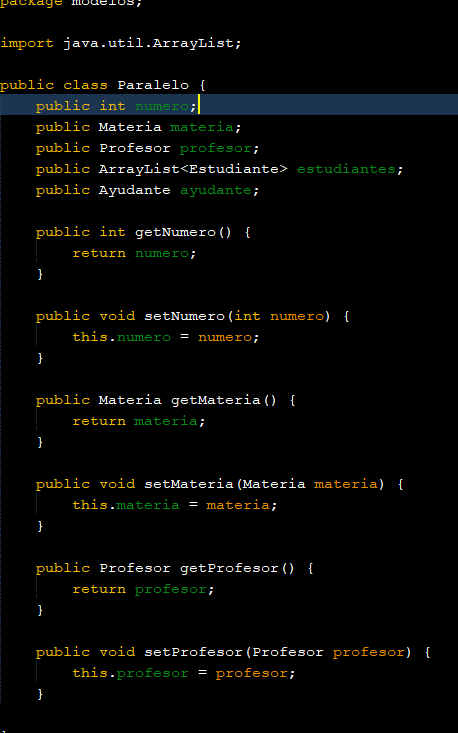
## Consecuencias

Las consecuencias de este Code Smell son molestosas, ya que no se entiende el porqué de su existencia y aumenta el tamaño del código, además de complicar el soporte del programa.

# Code Smell: Data Class

* Clase paralelo

## Código sin refactorizar



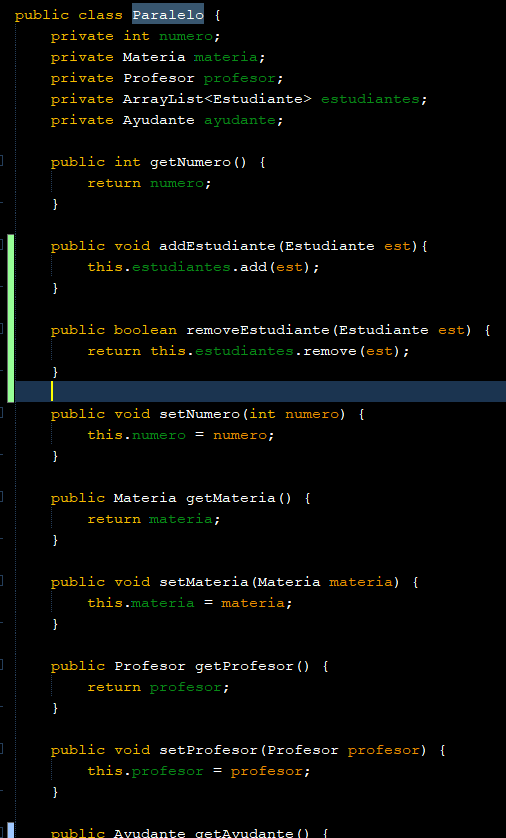
La solución sería aplicar los siguientes métodos de refactorización:

El primero es aplicar el método “Encapsulate Field”, lo que se hará es que los atributos solo se puedan acceder a ellos mediante los getter y setters.

Segundo, se aplicará “Encapsulate Collection” para el atributo estudiantes. Este método de refactorización hará que se creen métodos para añadir estudiantes y/o eliminar estudiantes de la colección mediante métodos sin acceder directamente al atributo.

Por lo tanto, la clase quedaría así:

## Código refactorizado



## Consecuencias

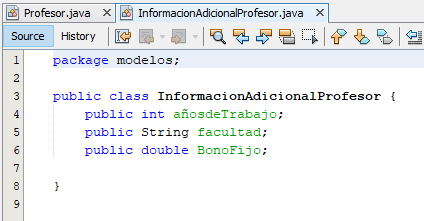
Las consecuencias de una Data Class sin refactorizar es que no hay organización del código y se arriesga a problemas de seguridad con la clase y además si refactorizas te ayuda a encontrar duplicados.

# Code Smell: Data Class

* Clase Información Adicional Profesor

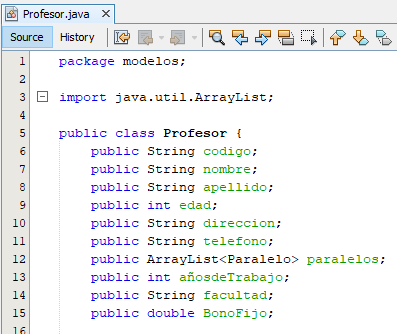
Los atributos enunciados en dicha clase realmente no hacen nada productivo ahí, es mejor colocarlos en una clase diferente.

## Código sin refactorizar:

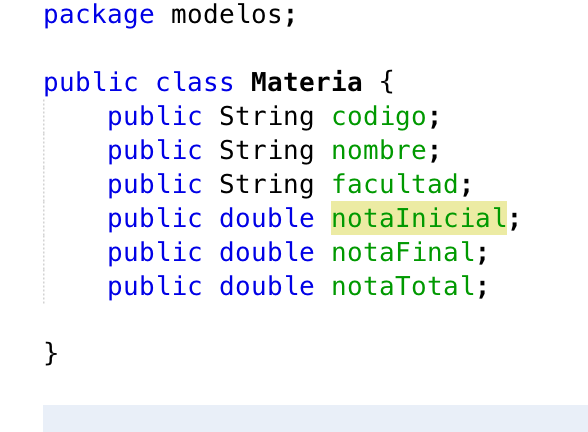


Se cambiará de lugar los atributos de InfomacionAdicionalProfesor.java a la clase Profesor.java, y se la elimino dicha clase resolviendo así el problema.

## Código refactorizado:

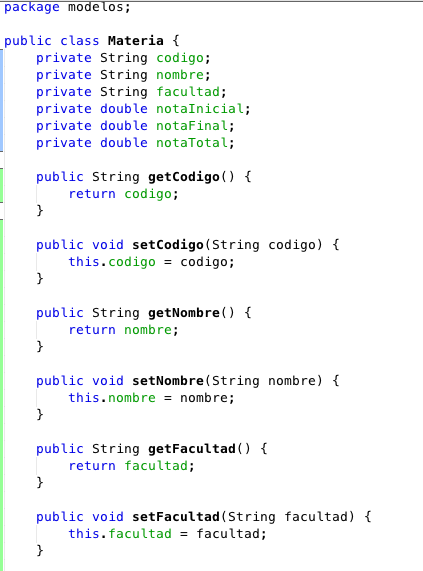


## Clase Materia: Código sin refactorizar



La clase Materia es un DataClass puesto a que esta clase contiene únicamente atributos y ningún método. Se cambiará el método de acceso a los atributos de la clase. Dichos atributos son actualmente públicos, por lo tanto, los cambiaremos a privados y posteriormente se escribirán sus getters y setters respectivos.

***Clase Materia: Código refactorizado***



## Consecuencias

Las consecuencias pueden ser problemas de seguridad con la clase y los datos que posee, además con la refactorización se puede encontrar duplicados.