**DISEÑO DE SOFTWARE**

**TALLER 8**

**REFACTORING**

**PARALELO 1**

**INTEGRANTES:**

* NICOLAS MAURICIO CORONEL MENDOZA
* CRISTOPHER ALEJANDRO ARROBA GOMEZ

Contenido

[Code Smells y Refactoring 3](#_Toc111067153)

[Clase Ayudante --> Inappropiate Intimacy 3](#_Toc111067154)

[Clase Estudiante y Profesor--> Duplicate Code y Primitive Obssesion 5](#_Toc111067155)

[Clase Estudiante --> Large Class 8](#_Toc111067156)

[Clase Estudiante --> Temporary Field 12](#_Toc111067157)

[Clase calcularSueldoProfesor --> Feature envy 13](#_Toc111067158)

[Clase Estudiante --> Long Parameter List 14](#_Toc111067159)

[Clase Estudiante --> Duplicate Code 15](#_Toc111067160)

[Clase InformacionAdicionalProfesor y Materia --> Data Class 16](#_Toc111067161)

[Clase calcularSueldoProfesor --> Message Chains 19](#_Toc111067162)

[Link Repositorio 20](#_Toc111067163)

# Code Smells y Refactoring

## Clase Ayudante --> Inappropiate Intimacy

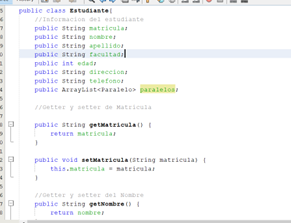
* Consecuencias:

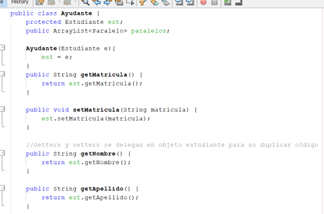
Este code smell se lo encuentra cuando una clase llama a los metodos de otra clase mas que a los suyos. En la clase ayudante tiene los getters y setters pero en estos metodos llama a los atributos de Estudiante, usando los getters y setters de estudiante.

* Refactoring:

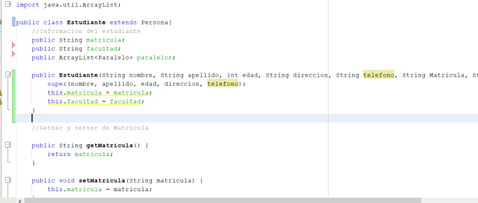
En este caso utilizaremos **Replace Delegation With Inheritance** asi los getters y setters no tendran que llamar a los getters y setters de estudiante, ya que no sera necesario. Esto mejorara la organizacion del codigo y lo reducira.

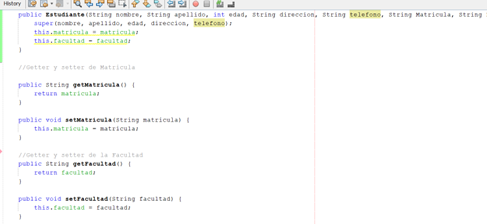
* Antes:

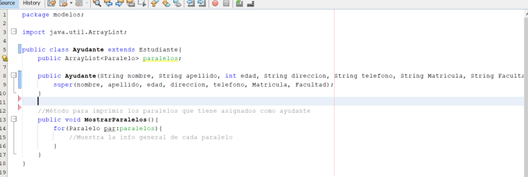




* Despues:







## Clase Estudiante y Profesor--> Duplicate Code y Primitive Obssesion

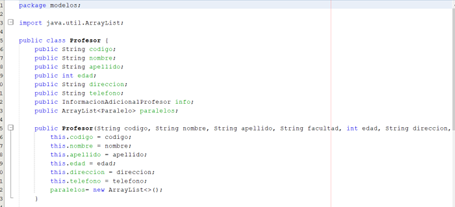
* Consecuencias:

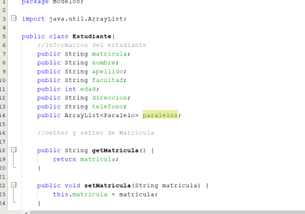
Este code smell aparece cuando dos fragmentos de codigo se ven iguales, en este caso la clase profesor y estudiante tienen casi los mismo atributos. Esto puede haber ocurrido porque una persona hizo la clase profesor y otra la clase estudiante o por copiar y pegar codigo, en cualquier caso esto seria un problema al crear mas clases o objetos que necesite estos atributos, como un decano o director de la materia ya que tendrian que copiar y pegar el mismo codigo en todas las clases. Ademas todos estos atributos son datos primitivos por lo cual creemos que tambien existe primitive obssesion. El cual se da al no hacer los datos una clase o objeto pequeno si no que solo lo pone string o int (datos primitivos).

* Refactoring:

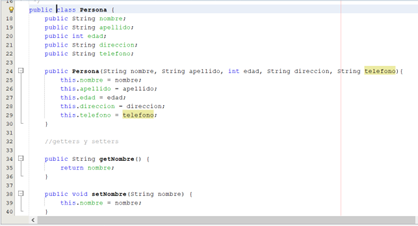
Lo que utilizaremos para este code smell sera **Extract Superclass** porque tenemos dos clases con los mismos atributos. Esto nos ayudara que los atributos se encuentren en un solo lugar y no tender que copiar el codigo en mas partes. Al hacer esto moveremos los datos a una clase padre y las clases podran extender de esta clase y usar dichos atributos juntos con sus metodos.

* Antes:

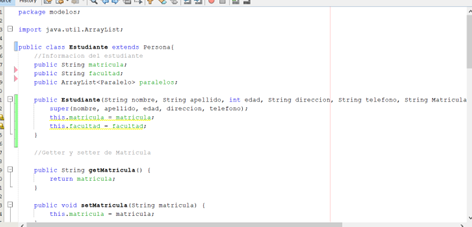


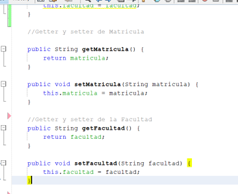


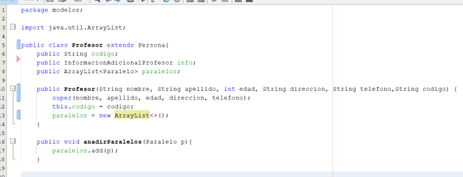
* Despues:











## Clase Estudiante --> Large Class

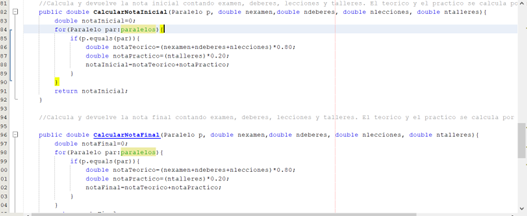
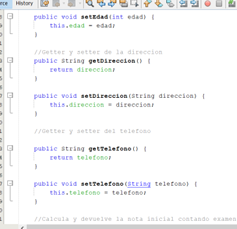
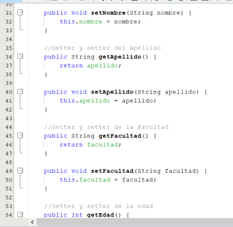
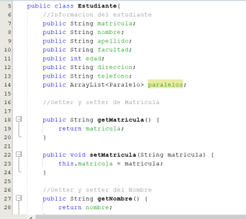
* Consecuencias:

Este code smell aparece cuando una clase es muy larga, tiene muchos metodos, fields o lineas de codigo, en este caso estudiante es una clase larga. Las clases asi no se ven bien y son mas dificilis de entender para las personas.

* Refactoring

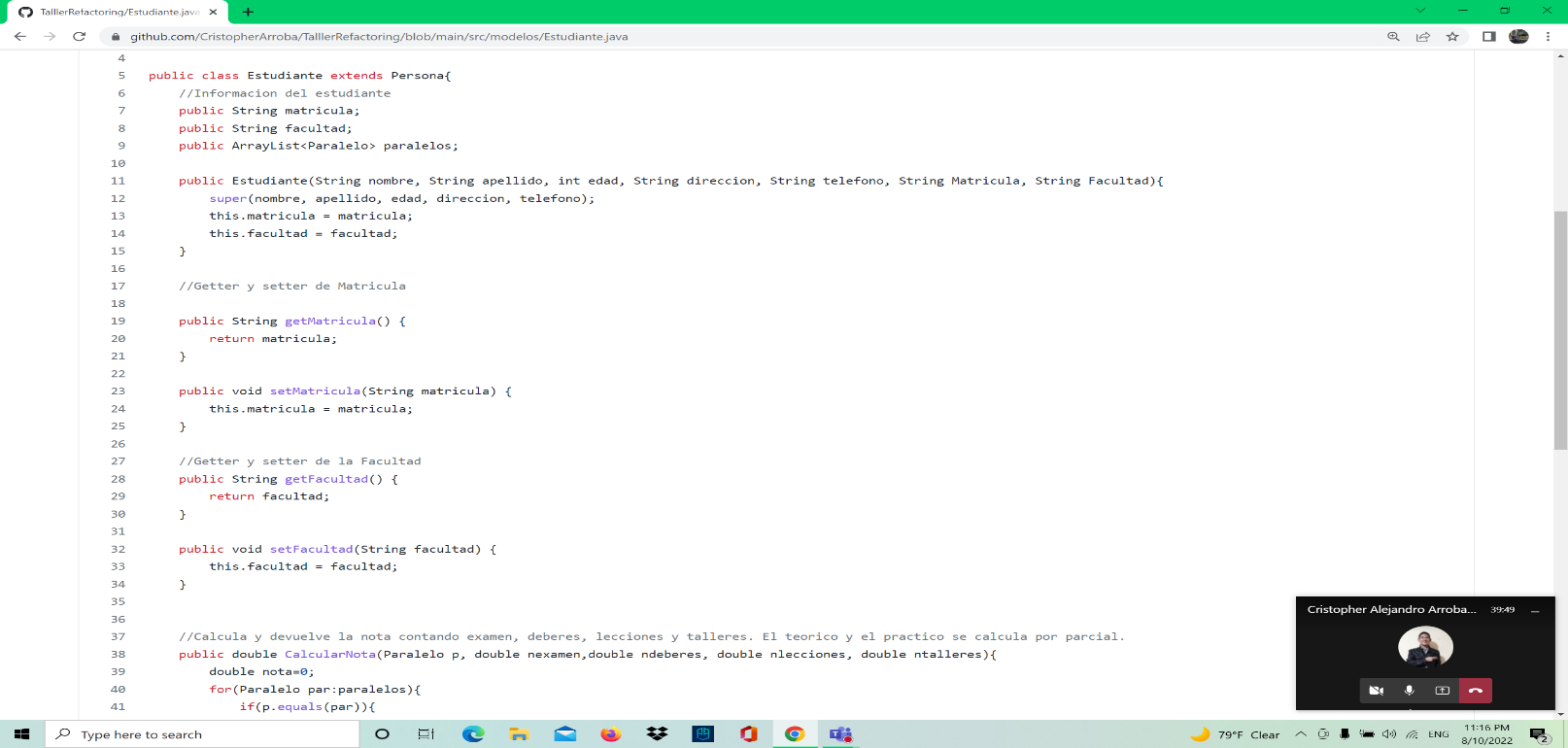
En este clase aplicaremos **Extract Superclass** por el codigo duplicado que tiene con profesor y **Extract Method** porque tiene codigo repetido en dos metodos, esto hara que la clase se vea mejor, tenga menos lineas de codigo y se mas sencilla de entender.

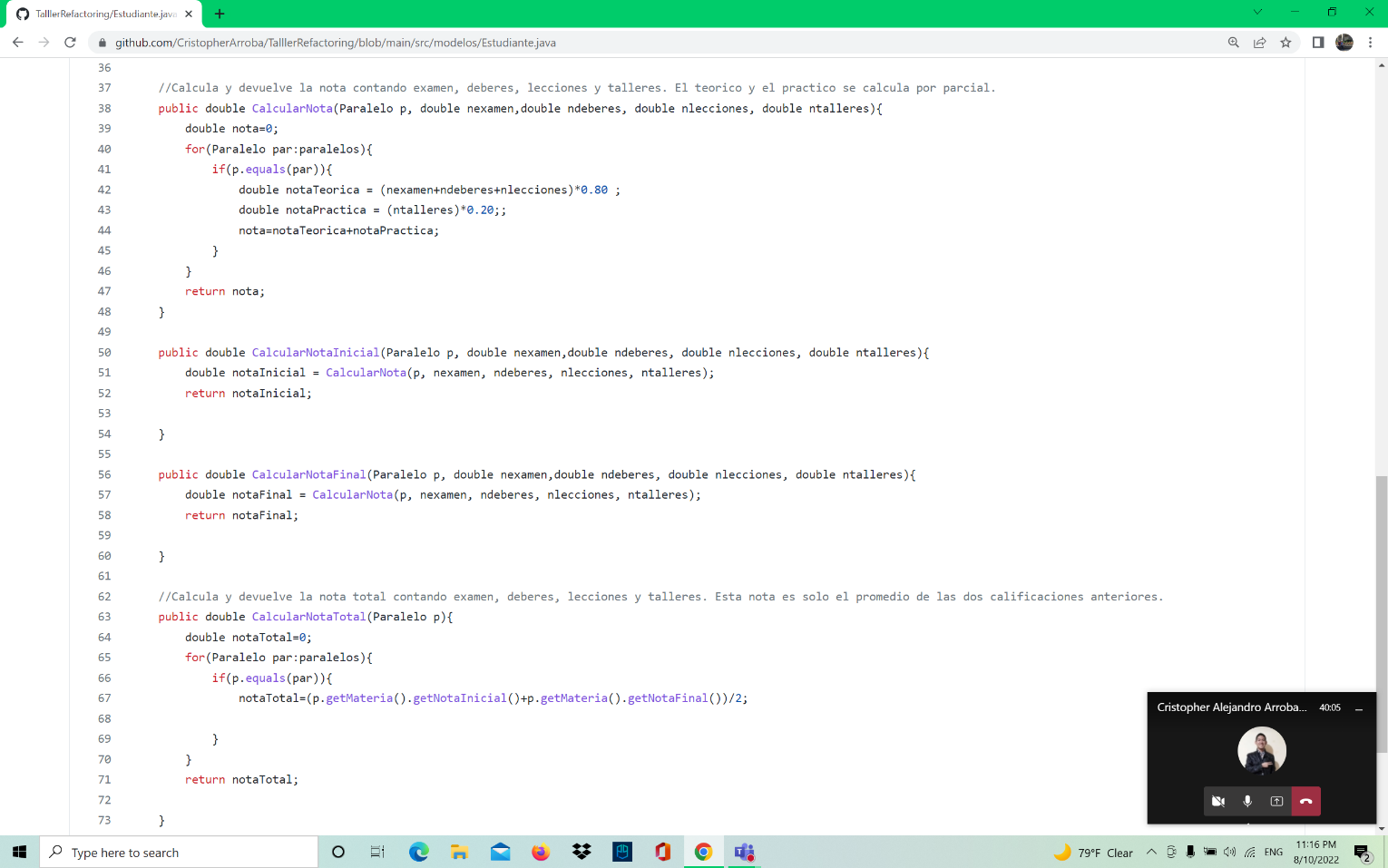
* Antes:





* Despues:





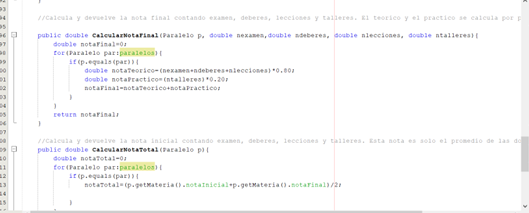
## Clase Estudiante --> Temporary Field

* Consecuencias:

Este code Smell aparece cuando usamos campos temporales para utilizarlos en funciones que necesitan varios inputs y en vez de crearlos usamos estos campos temporales que fuera de estas funciones son vacios. Esto hace mas dificil entender el codigo

* Refactoring

Para este code smell utilizaremos **Replace Method With Method Object** para mandar esos fields a la clase materia, y hacer un metodo que calcule la nota del teorico y la nota del practico. Asi el codigo se vera mejor y sera mas facil de enteder.



## Clase calcularSueldoProfesor --> Feature envy

* Consecuencias:

Este code smell ocurre cuando un metodo accede mas a los datos de otro objeto mas que a los suyos, los de su clase, esto ocurre con el metodo calcular suedo, que usa los datos de InformacionAdicionalProfesor para calcular el suedo.

* Refactoring:

Para este smell code usaremos **Move Method** para mover este metodo a la clase InformacionAdicionalProfesor que es de donde saca los datos para este metodo. Esto hara mas sencillo de entender el codigo y podremos eliminar la clase calculaeSueldoProfesor que se quedaria vacia.

* Antes:



## Clase Estudiante --> Long Parameter List

* Consecuencias:

En la clase Estudiante hay métodos con varios parámetros lo que ocasiona el origen de un code smell de Long Parameter list. Como consecuencia, el método se vuelve confuso al momento de implementarlo y por el orden que debe de seguir.

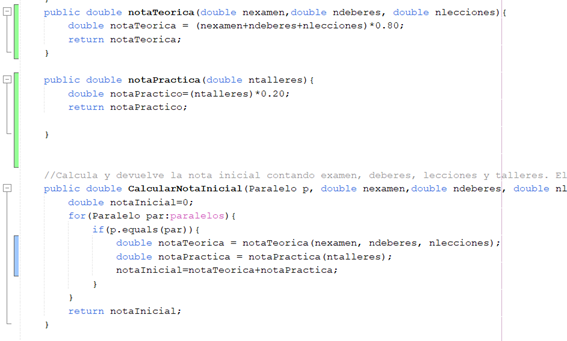
* Refactoring:

Para este code smell usaremos **Replace Parameter with Method Call,** con la finalidad de separar el proceso lógico de obtención de la nota, en métodos más simples.

* Antes:



* Despues:



## Clase Estudiante --> Duplicate Code

* Consecuencias:

En la clase estudiante hay dos métodos que son completamente idénticos y realizan el mismo proceso, por lo que solo está haciendo que un mismo proceso pueda ser hecho por varios métodos, cuando puede ser realizado con uno solo y simplificar el código del programa y hacerlo más legible.

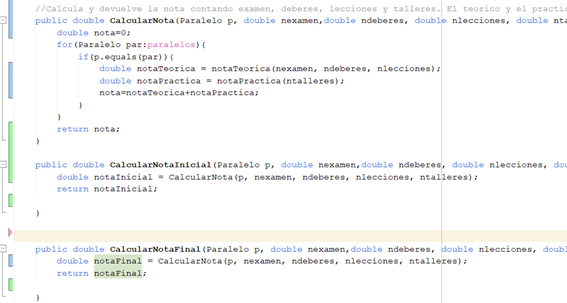
* Refactoring:

Para ello se procedera con **Extract Method,** para evitar la duplicación de esta parte del código, y reutilizar los métodos en el resto del programa donde se lo necesite.

* Antes:



* Despues



## Clase InformacionAdicionalProfesor y Materia --> Data Class

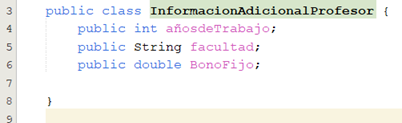
* Consecuencias:

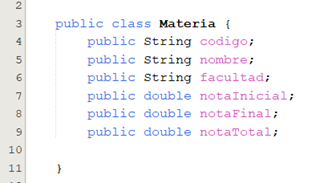
Estas clases solo tienen atributos que pueden ser accedidos y modificadospor cualquier otro objeto, ademas no cumple con el principio de la encapsulación. Por lo que la integridad del programa se ve afectada.

* Refactoring:

Para solucionar este code smell se aplicará E**ncapsulate field** para hacer sus atributos privados donde se puedan acceder de sus propios getters y setters.

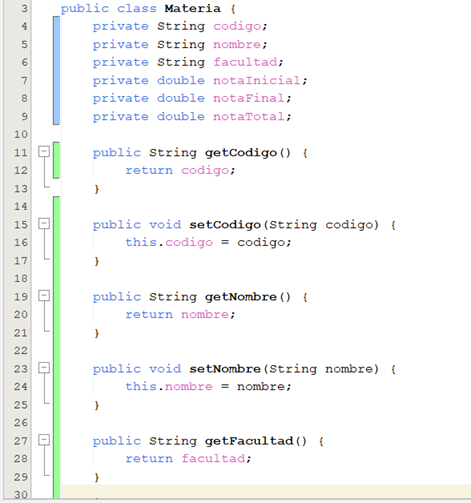
* Antes:





* Despues







## Clase calcularSueldoProfesor --> Message Chains

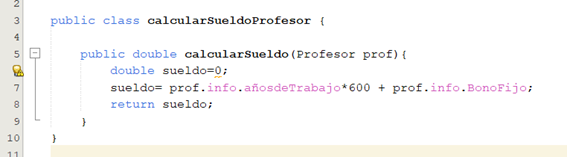
* Consecuencias:

Para calcular el sueldo primero obtiene la información del profesor, para con ese dato extraer de la clase “InformacionAdicionalProfesor” el dato requerido, lo que hace que el método tenga muchas depencias de otros métodos para realizar un proceso.

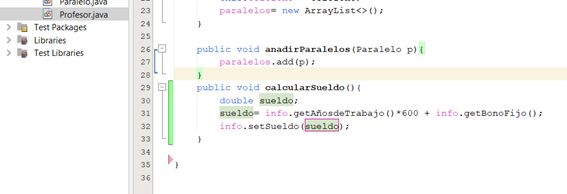
* Refactoring:

La solución a este code smell es **Move Method,** paramover este comportamiento a la clase “Profesor” que es de donde se están referenciando los datos, y así evitamos hacer varias llamadas en un mismo método.

* Antes:



* Después:



# Link Repositorio

https://github.com/CristopherArroba/TalllerRefactoring