ESCUELA SUPERIOR POLITECNICA DEL LITORAL

DISEÑO DE SOFTWARE

TALLER DE CODE SMELLS

INTEGRANTES

ALEX VELEZ

JAIME PIZARRO

VALERIA BARZOLA

CARLOS LOJA

EDDO ALVARADO

TERMINO

2020 – 1S

Contents

[**Sección A** 3](#_Toc48238612)

[Code Smell 1: Duplicate Code 3](#_Toc48238613)

[Consecuencias 3](#_Toc48238614)

[Técnicas de Refactorización 3](#_Toc48238615)

[Después 3](#_Toc48238616)

[Code Smell 2: Long Parameter List 4](#_Toc48238617)

[Consecuencias 4](#_Toc48238618)

[Técnicas de Refactorización 4](#_Toc48238619)

[Code Smell 3: Inappropriate Intimacy 5](#_Toc48238620)

[Consecuencias: 5](#_Toc48238621)

[Técnicas de Refactorización: 5](#_Toc48238622)

[Code Smell 4: Data Class 6](#_Toc48238623)

[Consecuencias: 6](#_Toc48238624)

[Técnicas de Refactorización: 6](#_Toc48238625)

[Code Smell 5: Primitive Obsession 7](#_Toc48238626)

[Consecuencias: 7](#_Toc48238627)

[Técnicas de Refactorización: 7](#_Toc48238628)

[Code Smell 6: Duplicate Code 8](#_Toc48238629)

[Consecuencias: 8](#_Toc48238630)

[Técnicas de Refactorización: 8](#_Toc48238631)

[Code Smell 7: Lazy Class 12](#_Toc48238632)

[Consecuencias: 12](#_Toc48238633)

[Técnicas de Refactorización: 12](#_Toc48238634)

[Code Smell 8: Dead Code 13](#_Toc48238635)

[Consecuencias: 13](#_Toc48238636)

[Técnicas de Refactorización: 13](#_Toc48238637)

[Code Smell 9: Inappropiate Intimacy 14](#_Toc48238638)

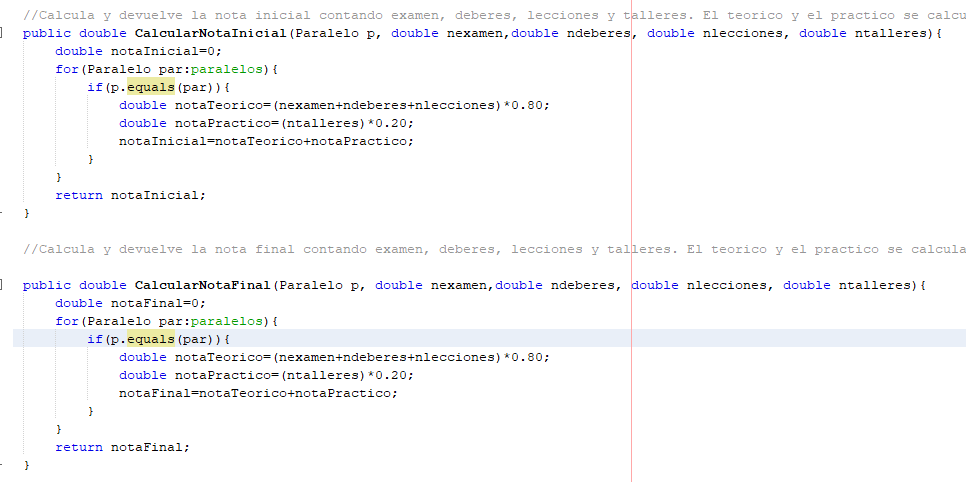
[Consecuencias: 14](#_Toc48238639)

[Técnicas de Refactorización: 14](#_Toc48238640)

[**Seccion B** 15](#_Toc48238641)

# **Sección A**

## Code Smell 1: Duplicate Code



### Consecuencias

Si se mantiene el código duplicado, hacemos el código más largo y difícil de mantener. En caso de que se quiera hacer cambio en la forma de calcular las notas, habrá que hacer los cambios en ambos códigos.

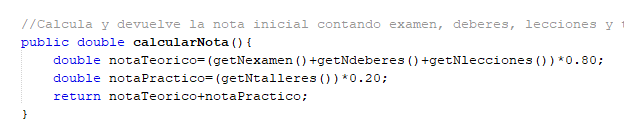
### Técnicas de Refactorización

**Extract Method**:

Extraer el código repetido en ambos métodos y ubicarlo en uno nuevo con un nombre que pueda generalizarse para ambos casos.

### Después

Esta parte del código tenía varios code smells que son revisados más abajo, aplicando todas las técnicas de refactorización el código final queda de la siguiente manera:



## Code Smell 2: Long Parameter List



### Consecuencias

Las consecuencias de mantener el método con todos los parámetros son muy negativas.

En primer lugar, hace el código menos entendible, mientras más parámetros tiene el método, hay mayor probabilidad de que no se entienda el por qué debe recibir esos métodos. En segundo lugar, el costo de mantenimiento del código aumenta. También hace el código menos extensible.

### Técnicas de Refactorización

**Extract Class**

Primero creamos una clase Nota que convierta los parámetros del método y los convierte en atributos de la clase. Esto lo hacemos ya que el estudiante no debe tener la responsabilidad de realizar los cálculos de las notas.

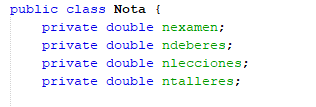
**Move Method**

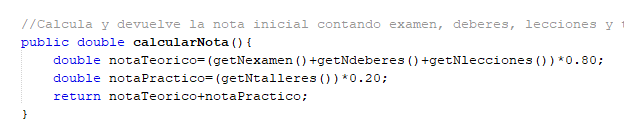
Creamos un nuevo método en la clase Nota que se llame calcularNota() que se base en el método calcularNotaFinal() o calcularNotaInicial() pero ya no tenga la necesidad de recibir las notas como parámetros sino simplemente calcule la nota promedio usando los atributos de la clase.

Self Encapsulate Field

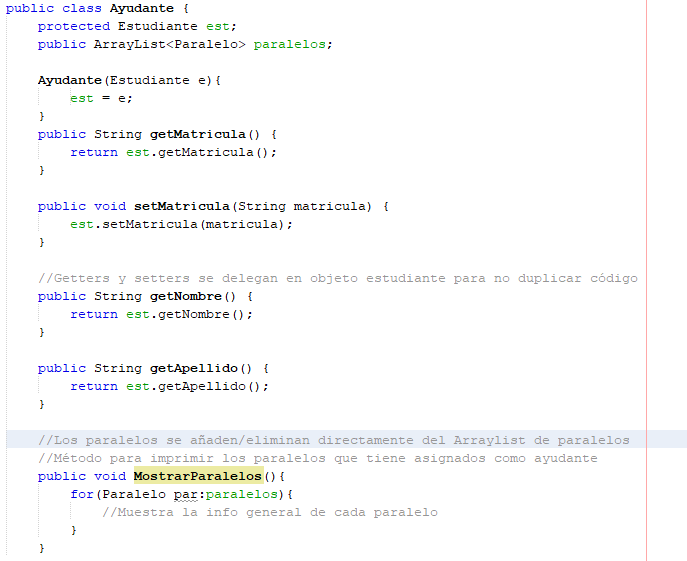
Usar los getters para acceder a los atributos de la clase.

### Después





## Code Smell 3: Inappropriate Intimacy



### Consecuencias:

Esta clase Ayudante usa los campos y métodos internos de la clase Estudiante contribuyendo así a un acoplamiento excesivo entre clases. Haciendo el código difícil de organizar y mantener.

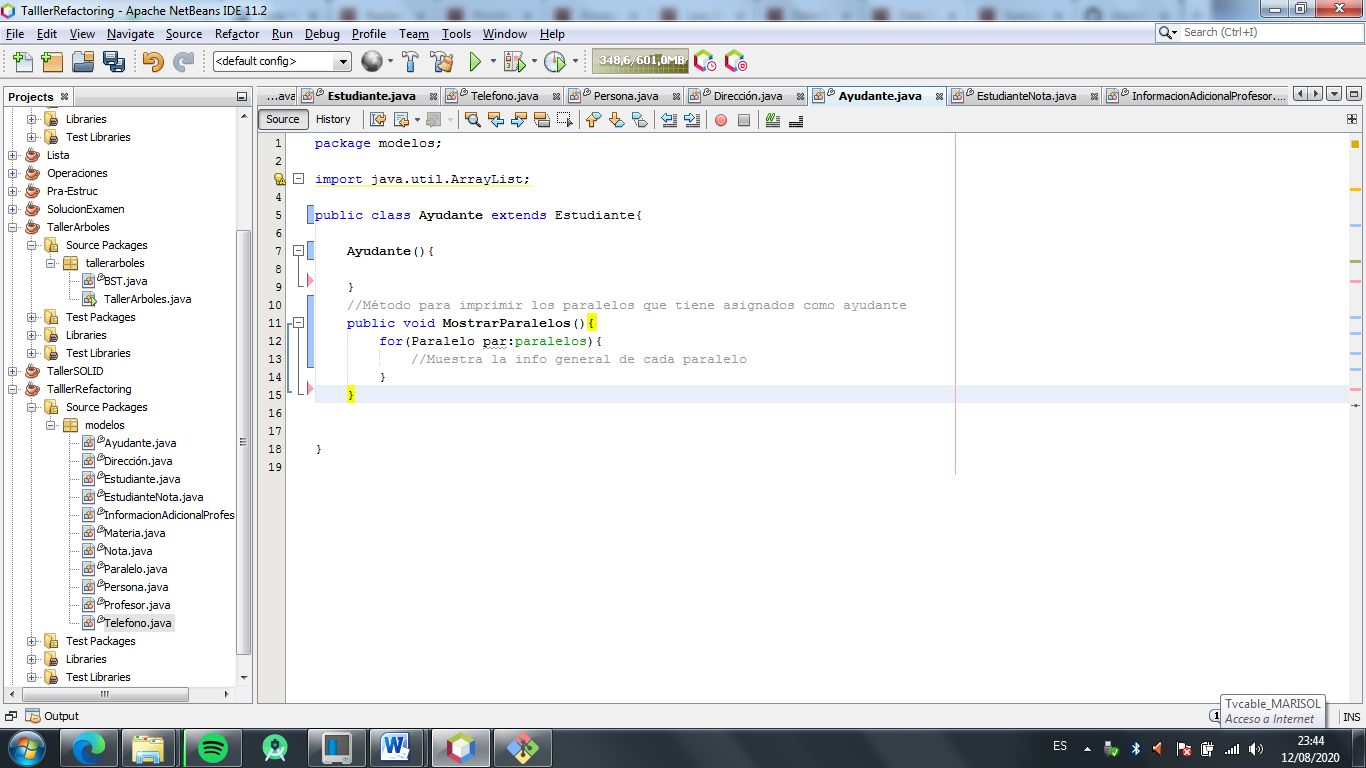
### Técnicas de Refactorización:

**Replace Delegation With Inherintance**

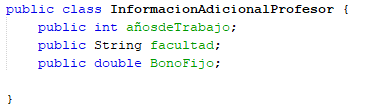
La técnica de refactorización utilizada para solucionar el code smell es Replace Delegation With Inherintance**,** esto se da porque la clase Ayudante utiliza a los métodos públicos de Estudiante, en este caso al utilizar este método evitamos la creación de métodos delegados del objeto de tipo Estudiante.

Por ello convertimos la clase Estudiante en una superclase de la clase ayudante, para así heredar directamente los métodos y no crear nuevos métodos

### Después



Code Smell 4: Data Class



### Consecuencias:

Esta clase contiene solo campos que son simplemente contenedores de datos utilizados por otras clases. Estas clases no contienen ninguna funcionalidad adicional y no pueden operar de forma independiente con los datos que poseen.

Dejar esta clase en el proyecto significaría aumentar el acoplamiento entre las clases, de manera innecesaria

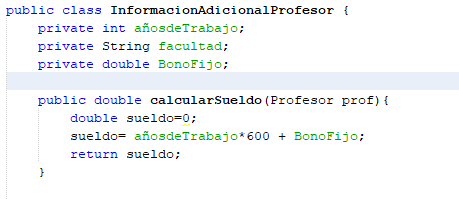
### Técnicas de Refactorización:

Revisando el código de cliente que usa la clase InforamacionAdicionalProfesor que es calcularSueldoProfesor, puede encontrar una funcionalidad que estaría mejor ubicada en la clase de datos en sí (InforamacionAdicionalProfesor). Dado este caso se utilizaría dos técnicas de refactorización:

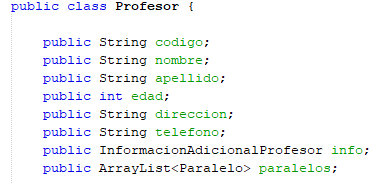
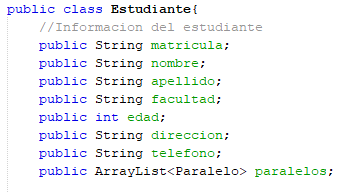
**Move method:** Crear un nuevo método en la clase que más usa el método, luego mover el código del método anterior allí.

**Encapsulate Field:** para ocultarlos del acceso directo y requiera que el acceso se realice solo a través de getters y setters.

### Después



## Code Smell 5: Primitive Obsession

### Consecuencias:

La consecuencia de mantener a las variables dirección y teléfono como primitivas, es que no permite describir el comportamiento adecuado de esos atributos, pues el tipo primitivo no le brinda una adecuada descripción a su data.

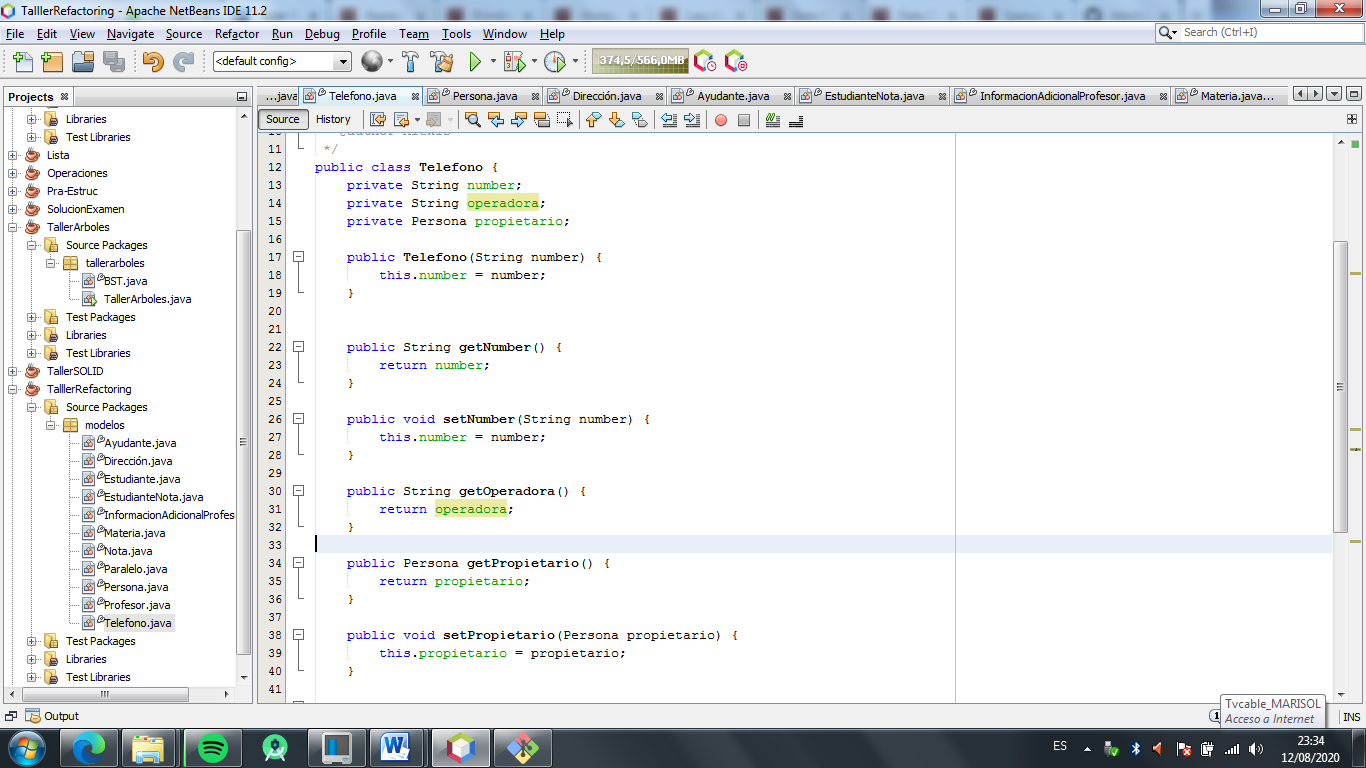
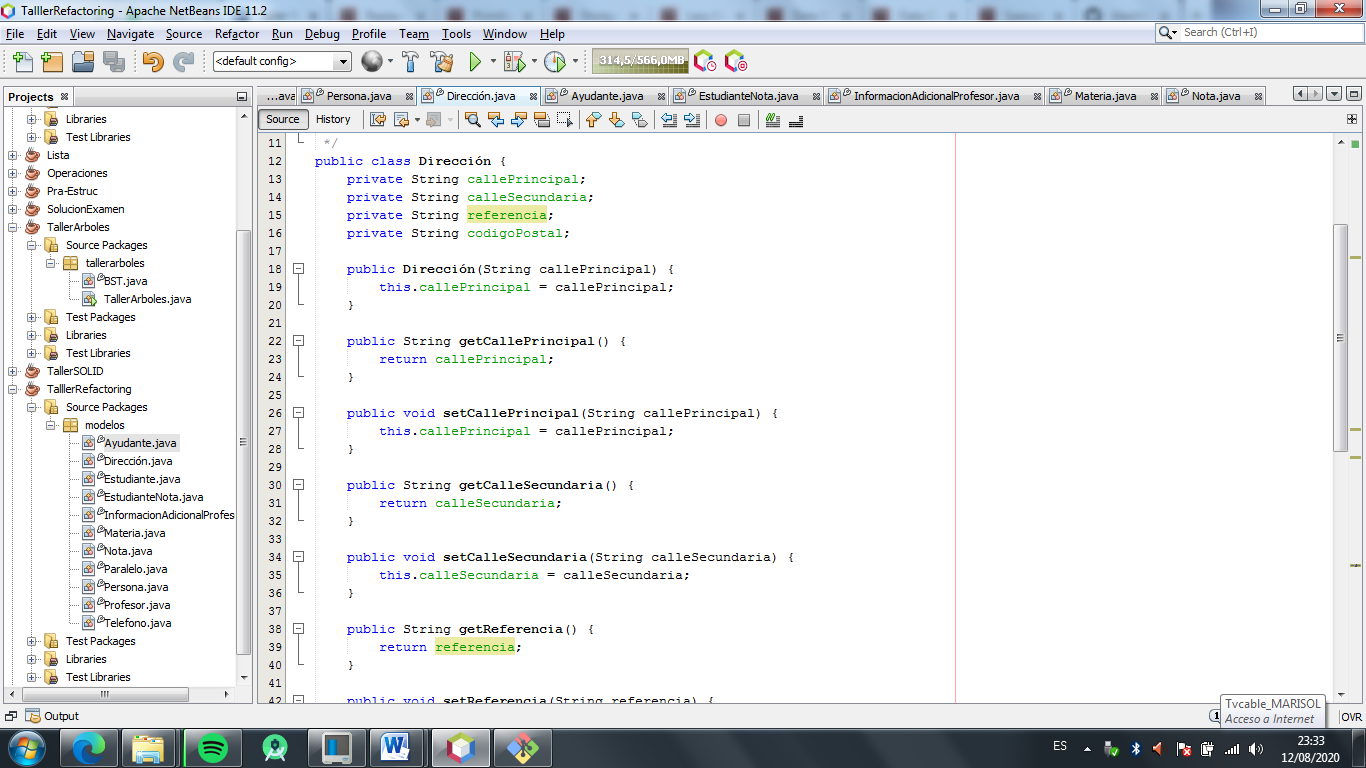
### Técnicas de Refactorización:

Revisando ambos atributos que usa las clases Profesor y Estudiante, y se decidió agregar un comportamiento descriptivo que estaría mejor ubicada en nuevas clases. Dado este caso se utilizaría la técnica de refactorización:

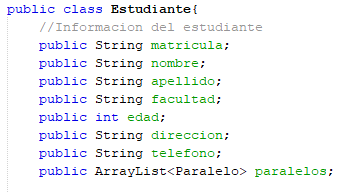
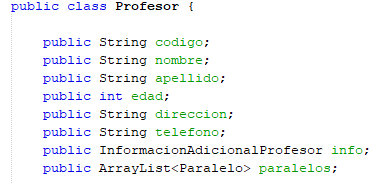
**Replace Data Value with Object:**

Como ambos atributos no cumplen con un comportamiento adecuado, se crean clases de con un nombre descriptivo y se agregan las funcionalidades descriptivas y adecuadas de cada una de las clases.

### Después:



## Code Smell 6: Duplicate Code



### Consecuencias:

Tener atributos de clases repetidos, causan que las clases tengan muchos atributos y por lo tanto sea una clase larga, sea mucho más difícil de mantener, menos mantenible y tenga un mayor costo de mantenibilidad.

### Técnicas de Refactorización:

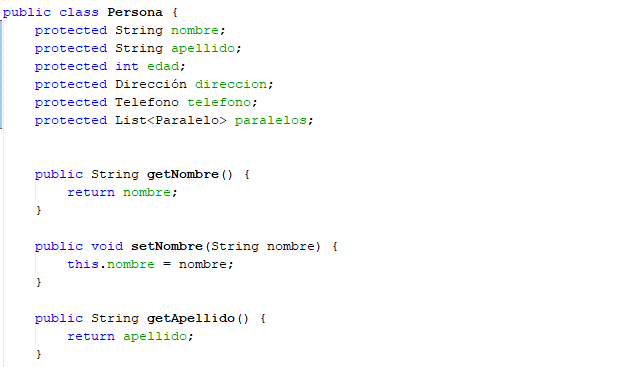
**Extract Superclass.**

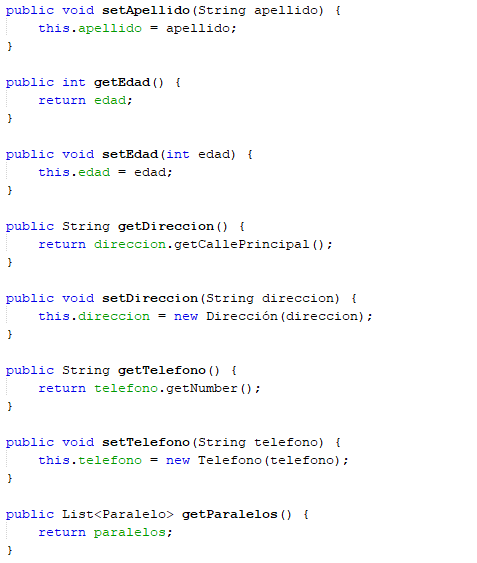
Como ambas clases tienen atributos similares, se debe crear una nueva clase que contenga a todos esos atributos similares, y hacer que ambas clases hereden de esa nueva clase.

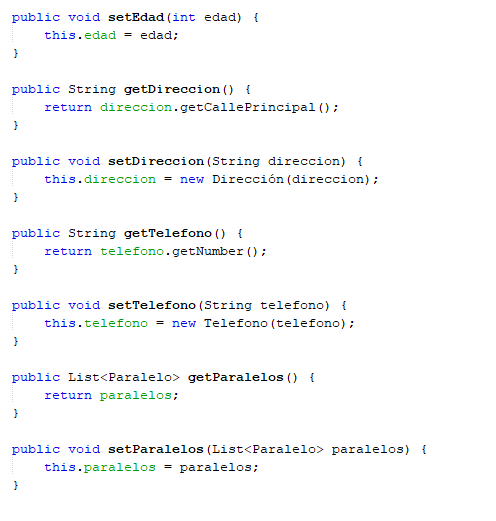
En este caso se creó a la clase Persona que es la encargada de tener estos atributos en común, los ponemos con un modificador de acceso protected, con sus getters y setters y Profesor y Estudiante heredan de la clase Persona, y mantienen los atributos que no heredan de Persona.

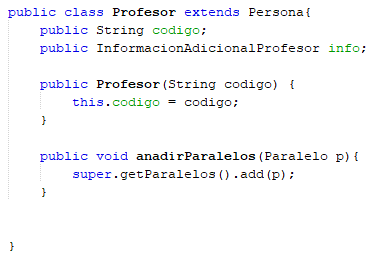
### Después

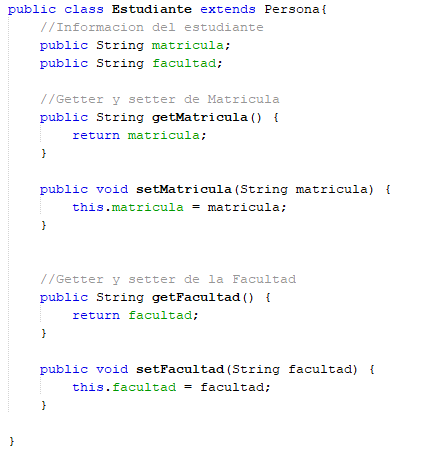
Luego de aplicar esta técnica de refactoring el código resultante sería el siguiente:







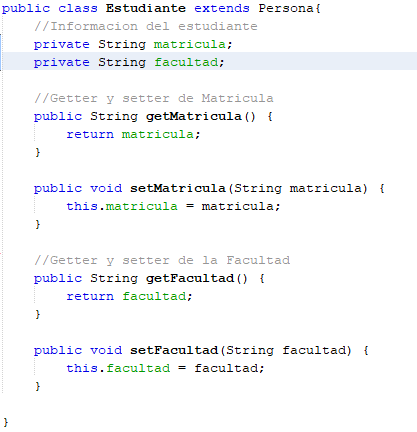




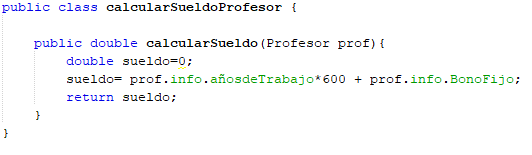
**Encapsulate Field**

Se cambia el modificador de acceso de los atributos de las clases Profesor y Estudiante de público a privado, y se incluyen los getters y setters para dar acceso a los atributos.





## Code Smell 7: Lazy Class



### Consecuencias:

Las consecuencias de tener una Lazy Class es que los costos de mantenimiento pueden elevarse, ya que habrá más código que leer, además que quitará tiempo.

### Técnicas de Refactorización:

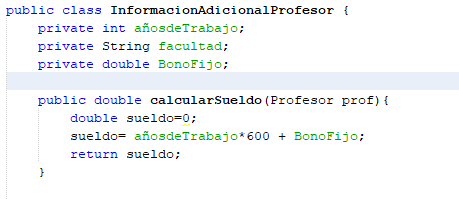
**Inline Class**

Para solucionar el code smell se utilizará el método Inline Classlo cual nos permitirá unir una clase con otra, en este caso podremos eliminar la clase CalcularSueldoProfesor.

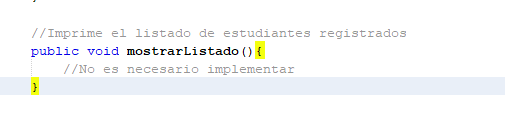
**Move Method**

y utilizar otro método de refactoring el cual es move method,el mismo nos ayudara a mover el método que contenía la clase CalcularSueldoProfesor a la clase InformacionAdicionalProfesor**.**

### Después



## Code Smell 8: Dead Code



### Consecuencias:

Provoca que el código sea más complicado de leer, que las clases sean más largas y complejas.

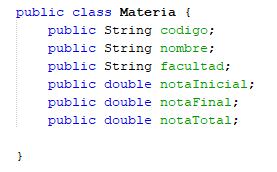
### Técnicas de Refactorización:

Como los métodos no son usados en ninguna clase se pueden eliminar sin generar problemas en ninguna otra clase.

### Después

La solución de este code smell es eliminar el método, por lo que no existe un bloque de código para mostrar en esta sección.

## Code Smell 9: Inappropiate Intimacy



### Consecuencias:

Se tienen atributos que no deben pertenecer a Materia, por lo que no cumple el principio de SOLID de Single Responsibility Principle. El tener estos atributos en materia, hace que deban tener instancias de esta clase para poder acceder a estos atributos, lo que produce que exista un acoplamiento probablemente innecesario con esta clase.

### Técnicas de Refactorización:

**Extract Class:**

Aplicamos el método de Extract Class para crear una nueva clase EstudianteNota que sirva como relación entre Estudiante y Paralelo. Esta clase servirá para tener las notas de los estudiantes.

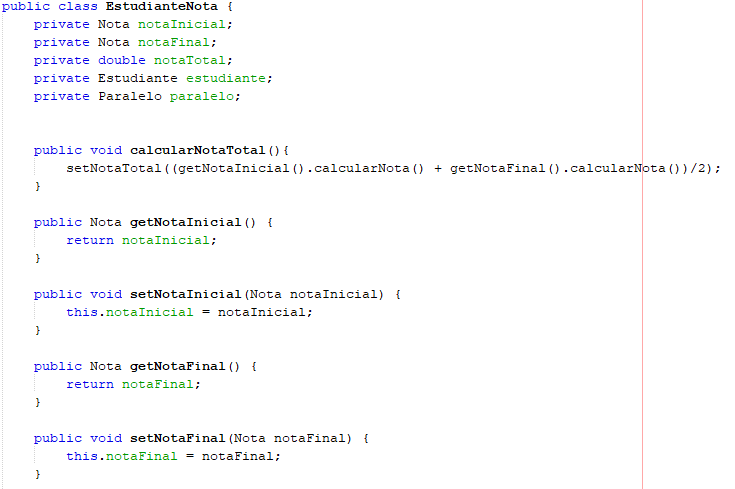
**Move Field:**

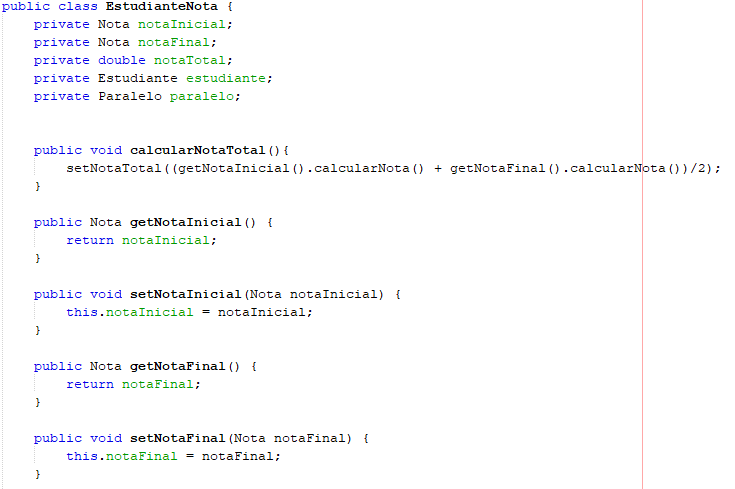
Notamos que es necesario mover los atributos notaInicial, notaFinal y notaTotal de la clase Materia a la nueva clase EstudianteNota.

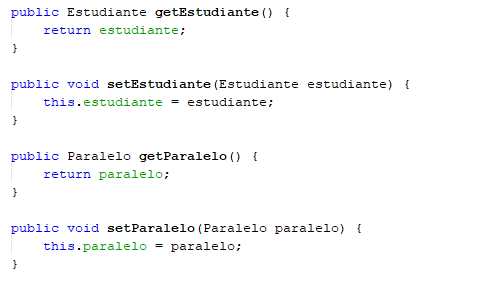
**Move Method:**

Movemos el método de calcularNotaTotal() que estaba estudiante a la clase EstudianteNota.

### Después







# **Seccion B**

Repositorio de github:

<https://github.com/AlexVelezLl/TallerDS_Refactoring>