**ESCUELA SUPERIOR POLITECNICA DEL LITORAL**

**TALLER REFACTORING**

**GRUPO 4**

**INTEGRANTES:**

**-Omar Coello**

**-David Rivera**

**-Ricardo Zaruma**

**Link del Repositorio:**

[**https://github.com/omcoello/TallerRefactoring.git**](https://github.com/omcoello/TallerRefactoring.git)

**Tabla de contenido**

Tabla de contenido

[**Data Class** 3](#_Toc79662631)

[**Innapropiate Intimacy** 4](#_Toc79662632)

[**Speculative Generality** 5](#_Toc79662633)

[**Duplicate Code** 6](#_Toc79662634)

[**Feature Envy** 7](#_Toc79662635)

[**Innapropiate Intimacy** 8](#_Toc79662636)

# **Data Class**

La clase “InformacionAdicionalProfesor” es una clase que posee atributos que podrían adaptarse para estar dentro de la clase “Profesor” sin ningún problema. No hay métodos relevantes y solo posee campos de la clase profesor. Eliminaremos esta clase y moveremos esos atributos a la clase “Profesor”. Además los atributos están públicos lo que no debería pasar.

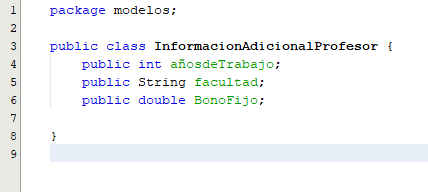
Consecuencias:

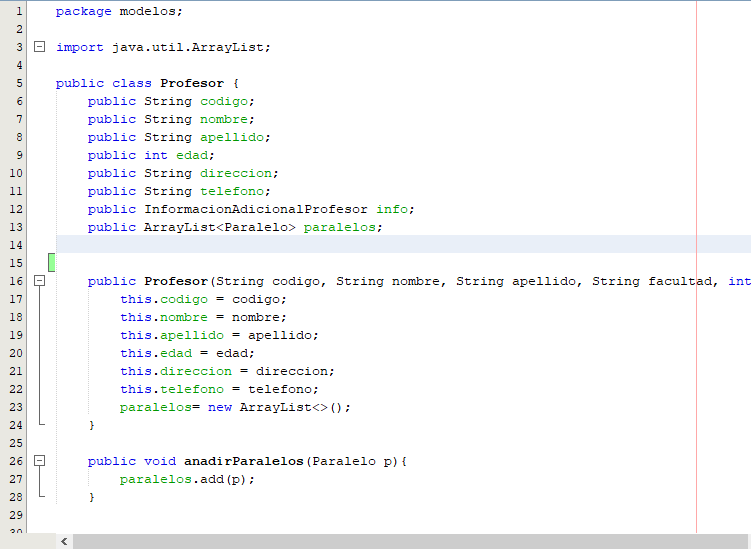
Se crearía métodos redundantes entre las dos clases haciendo el código más complicado innecesariamente.

Técnicas Refactoring:

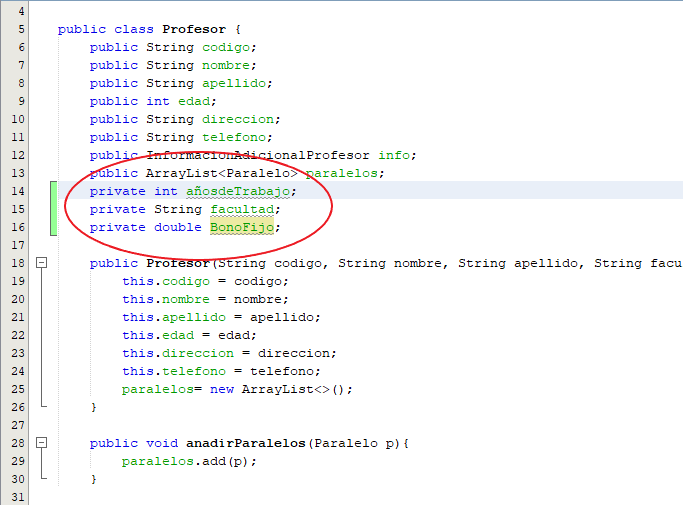
* Move Fields
* Encapsulate Fields

Código con Smells:





Código Corregido:



# **Innapropiate Intimacy**

La clase “CalcularSueldoProfesor” posee un método que accede directamente a los atributos de otra clase. Además esta clase se la podría arreglar y estar perfectamente en la clase profesor y tomar directamente los atributos de ahí.

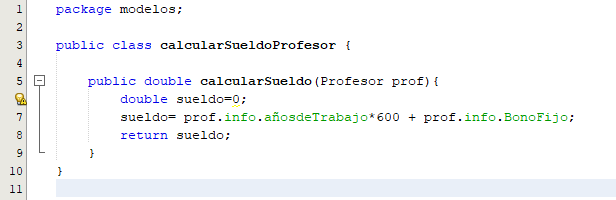
Consecuencias:

Aumenta la complejidad del código sobre todo el control sobre los datos de cada clase, puesto que se hay una dependencia bidireccional, y una actualización de esa información supondría altos costos e tiempo

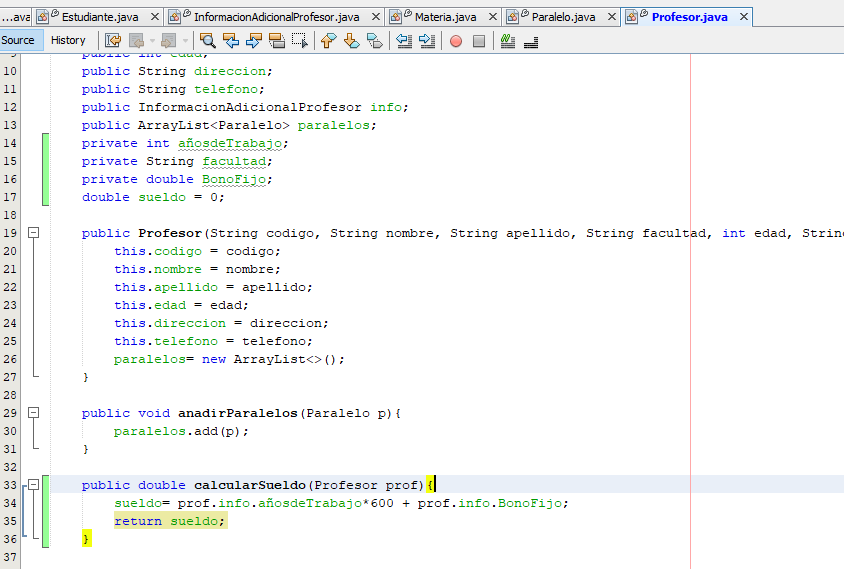
Técnicas Refactoring:

* Move Method
* Encapsulate Fields

Código con Smells:



Código Corregido:



# **Speculative Generality**

La clase “Profesor” tiene campos que no son utilizados. Si se refactoriza el código se tendrá como resultado un programa más fácil de leer y dar mantenimiento.

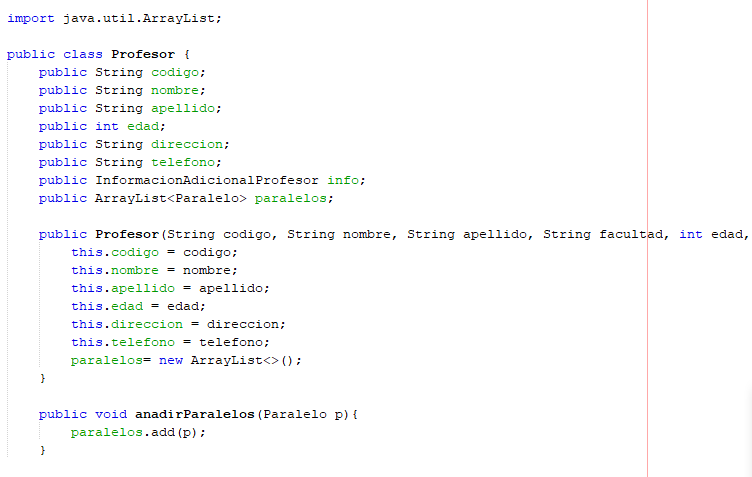
Consecuencias:

Aumenta la complejidad del código en una sola clase.

Técnicas Refactoring:

* Remove Parameter

Código con Smells:



Código Corregido:

# **Duplicate Code**

En la clase “Estudiante” tenemos dos métodos que usan gran parte del mismo código. Estos son “CalcularNotaInicial” y “CalcuclarNotaFinal”.

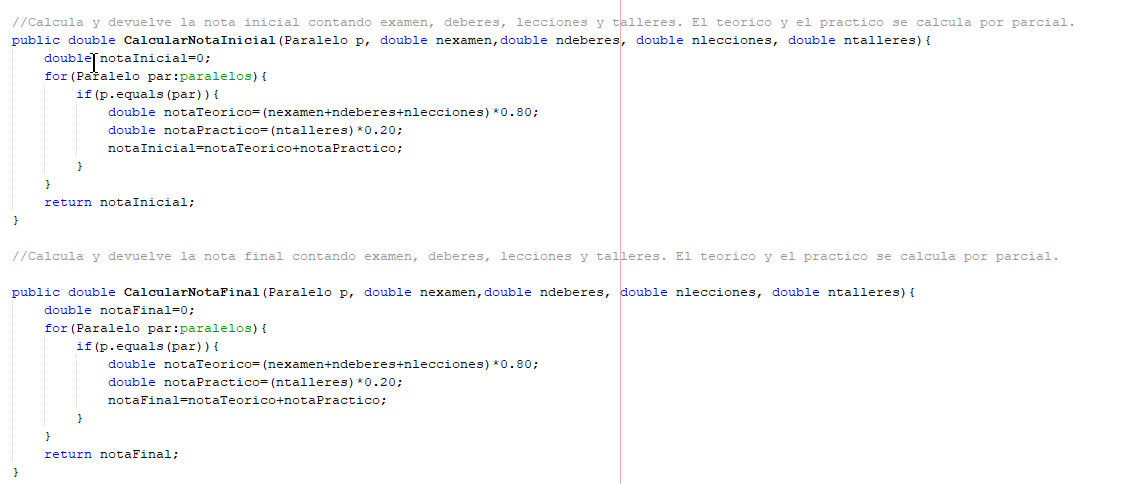
Consecuencias:

Esto puede complicar el código a la hora de realizar modificaciones debido a que se puede dejar un duplicado sin cambiar. Esto genera errores en los retornos de los métodos.

Técnicas Refactoring:

* Extract Method

Código con Smells:



Código Corregido:

# **Feature Envy**

La clase “Ayudante” accedes a atributos de la clase “Estudiante” demasiado. Lo que genera una delegación de sus funciones. Esto se puede arreglar con una herencia.

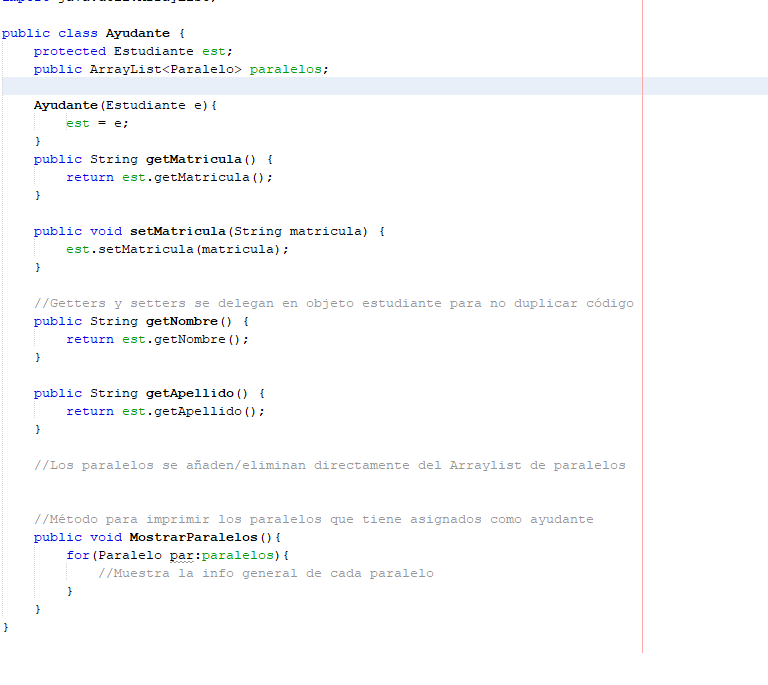
Consecuencias:

El código se vuelve más difícil de mantener e inentendible en ciertos casos. Además que un cambio en estudiante podría dañar métodos en ayudante.

Técnicas Refactoring:

* Replace Delegation with Inheritance

Código con Smells:



Código Corregido:

# **Innapropiate Intimacy**

El método “calcularNotaTotal” tiene como parámetro un objeto de tipo Paralelo, y este accede a su objeto Materia para poder acceder directamente a los atributos de la clase Materia. Además esto genera un mal encapsulamiento.

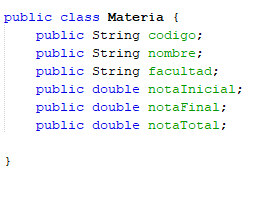
Consecuencias:

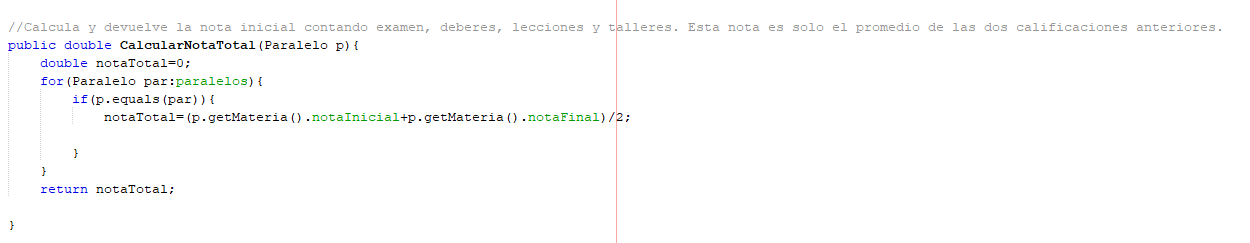
El acceso sin restricciones podría generar consecuencias como que la información pueda ser manipulada.

Técnicas Refactoring:

* Encapsulate Field

Código con Smells:





Código Corregido: