**ESCUELA SUPERIOR POLITECNICA DEL LITORAL**

**Diseño de Software**

**Taller 08 Refactoring**

**Grupo 5**

**Integrantes:**

* **Luis Adrian Litardo Calderon**
* **Yonkani Manuel Cedeno**
* **Jose Alberto Murillo Escudero**

**2021-2022**

Contenido

[Lazy Class: calcularSueldoProfesor 3](#_Toc79705323)

[Consecuencias: 3](#_Toc79705324)

[Refactoring: 3](#_Toc79705325)

[Captura Inicial: 3](#_Toc79705326)

[Captura Final: 3](#_Toc79705327)

[Inappropiate Intimacy: clase Profesor metodo calcularSueldoProfesor() 4](#_Toc79705328)

[Consecuencias: 4](#_Toc79705329)

[Refactoring: 4](#_Toc79705330)

[Codigo Inicial: 4](#_Toc79705331)

[Código Final: 4](#_Toc79705332)

[Duplicate Code: Clase estudiante 5](#_Toc79705333)

[Consecuencias: 5](#_Toc79705334)

[Refactoring: 5](#_Toc79705335)

[Captura Inicial: 5](#_Toc79705336)

[Captura Final: 5](#_Toc79705337)

[Speculative Generality: Clase Estudiante y clase Profesor. 6](#_Toc79705338)

[Consecuencias: 6](#_Toc79705339)

[Refactoring: 6](#_Toc79705340)

[Código Inicial: 6](#_Toc79705341)

[Código Final: 6](#_Toc79705342)

[Feature Envy : Clase Ayudante y Estudiante 7](#_Toc79705343)

[Consecuencias: 7](#_Toc79705344)

[Refactoring: 7](#_Toc79705345)

[Código Inicial: 7](#_Toc79705346)

[Código Final: 7](#_Toc79705347)

[Dead Code : Clase Materia 8](#_Toc79705348)

[Consecuencias: 8](#_Toc79705349)

[Refactoring: 8](#_Toc79705350)

[Código Inicial: 8](#_Toc79705351)

[Código Final: 8](#_Toc79705352)

[Data Class : Clase InformacionAdicionalProfesor 9](#_Toc79705348)

[Consecuencias: 9](#_Toc79705349)

[Refactoring: 9](#_Toc79705350)

[Código Inicial: 9](#_Toc79705351)

[Código Final: 9](#_Toc79705352)

# Lazy Class: calcularSueldoProfesor

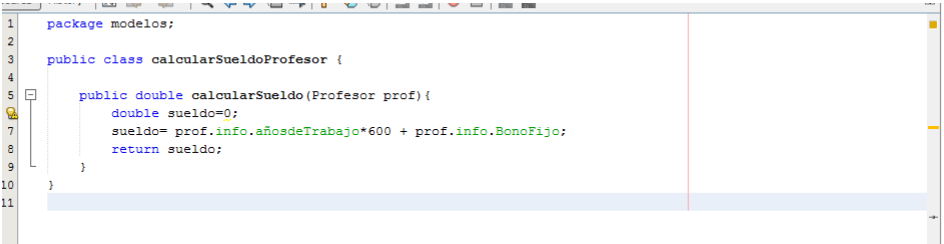
## Consecuencias:

La clase calcularSueldoProfesor solo tiene un metodo y este metodo hace llamada a los atributos de otra clase, mantenerla en el codigo solo causaria confusion ademas de aumentar el numero de clases y crear una dificultad al momento de revisar el codigo

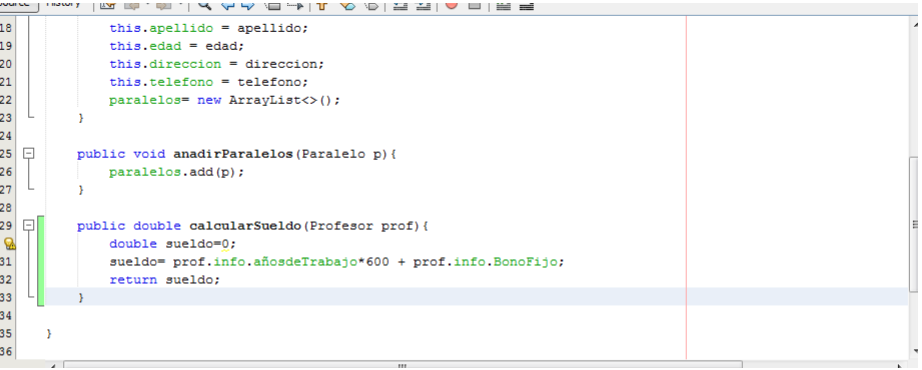
## Refactoring:

Se usara la tecnica de refactorizacion INLINE CLASS la cual nos permitira mover toda esta clase junto a sus atributos a otra que si sirva, en este caso moveremos el metodo a la clase profesor refactorizando asi el codigo

## Captura Inicial:



## Captura Final:



# Inappropiate Intimacy: clase Profesor metodo calcularSueldoProfesor()

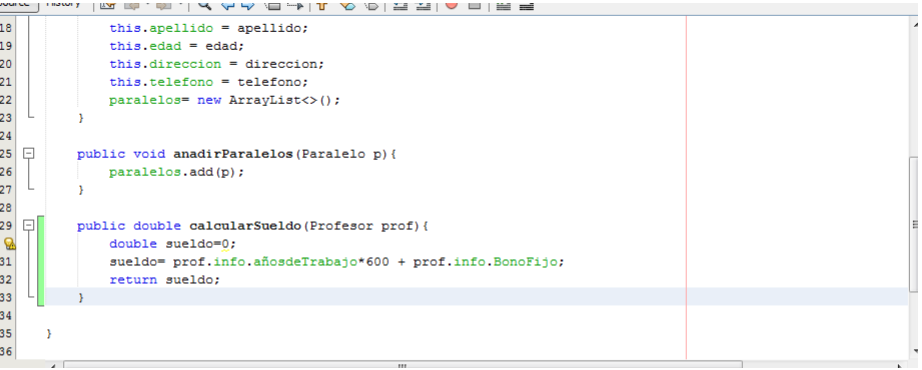
## Consecuencias:

Se produce cuando una clase usa los metodos o atributos de otra clase, en este caso en la clase Profesor el metodo calcularSueldoProfesor() hace llamado a los atributos de la clase InformacionAdicionalProfesor

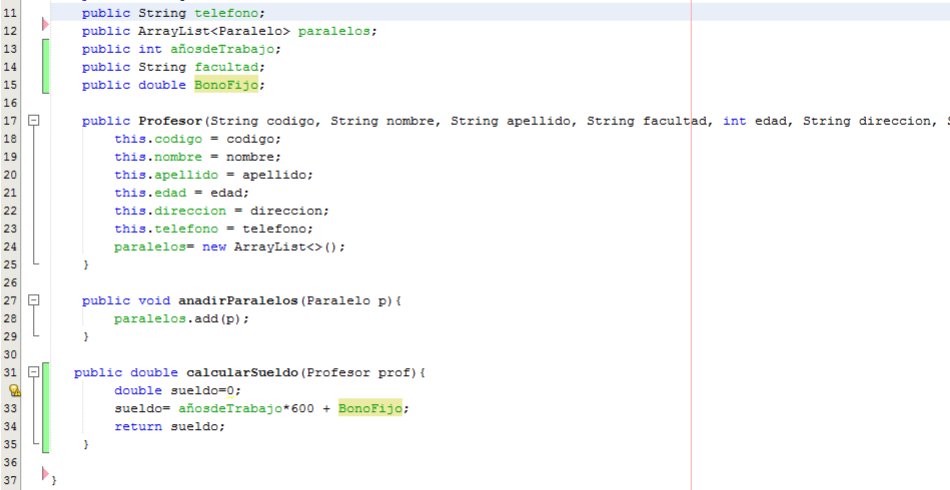
## Refactoring:

Se usa la tecnica de refactorizacion MOVE FIELD para poder mover los atributos de la clase InformacionAdicionalProfesor a la clase Profesor y que el metodo calcularSueldoProfesor() acceda a los metodos desde Profesor

## Codigo Inicial:



## Código Final:



# Duplicate Code: Clase estudiante

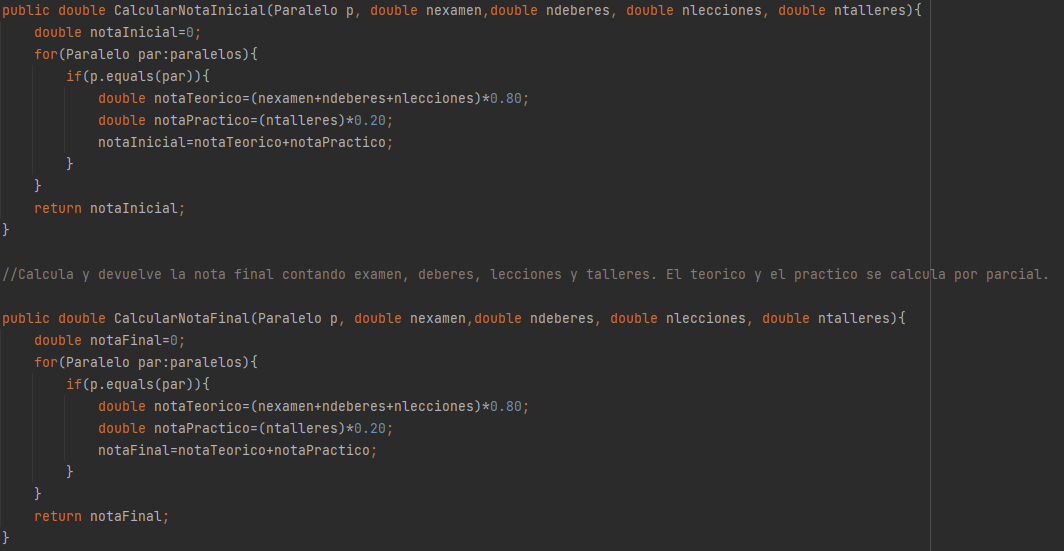
## Consecuencias:

Al tener código duplicado en un proyecto bastante grande puede ser difícil de encontrar y corregirlo a parte que el código que las clases se hagan mucho más grandes y poca legibilidad al momento de leer el código haciendo difícil la mantenibilidad y la simpleza.

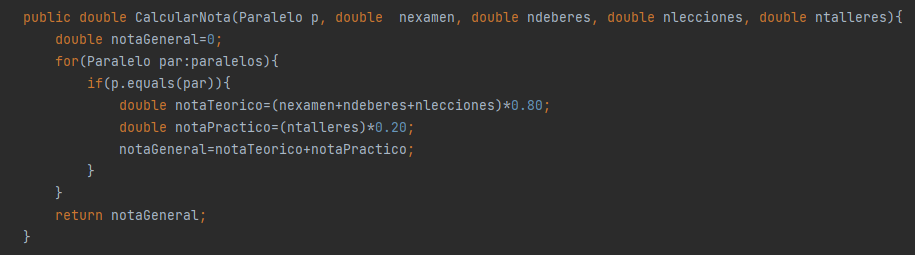
## Refactoring:

La técnica a utilizar es “Extract Method” que va a consistir en mover el código que este duplicado a un nuevo método con un nombre que vaya acorde a su propósito. Se copiará el fragmento del código relevante al nuevo método y se borrará el código viejo.

## Captura Inicial:



## Captura Final:



# Speculative Generality: Clase Estudiante y clase Profesor.

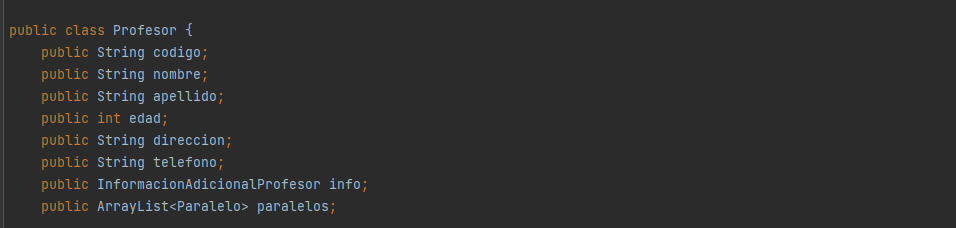
## Consecuencias:

Al existir este code smell, el código se vuelve mucho más grande haciendo que sea difícil de entender en otras palabras agregando complejidad y mantener. Además, que se hace uso memoria innecesaria al momento de crear variables que jamás se van a usar.

## Refactoring:

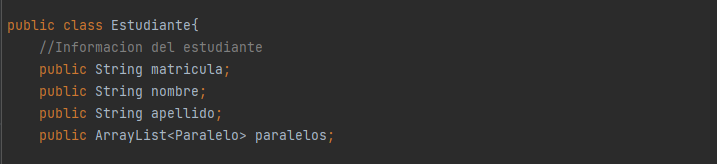
Los campos no utilizados van a ser removidos de sus clases correspondientes.

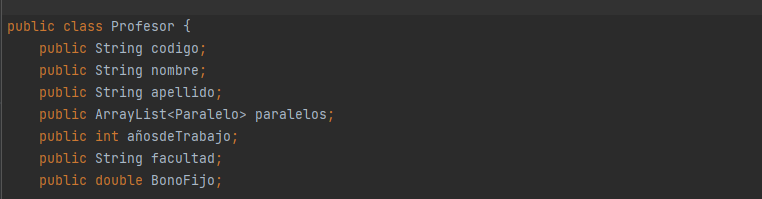
## Código Inicial:





## Código Final:





# Feature Envy : Clase Ayudante y Estudiante

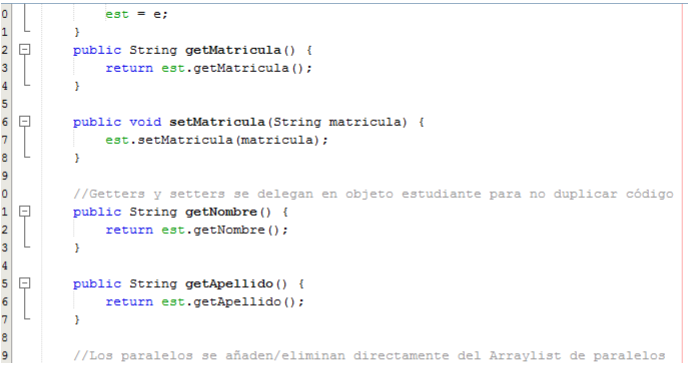
## Consecuencias:

Este code smell sucede cuando una clase accede a más objetos de otra clase que a las de la propia clase, en la clase Ayudante tenemos getters y setters que provienen de la clase Estudiante creando asi este Smell

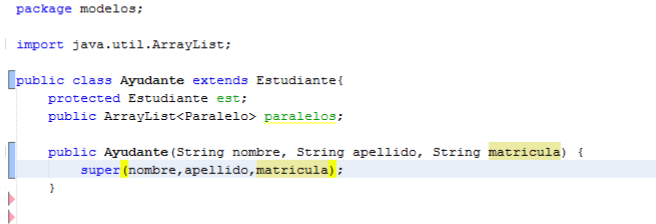
## Refactoring:

Para poder evitar todo este codigo podemos reemplazar con herencia de tal forma que Ayudante herede de Estudiante

## Código Inicial:



## Código Final:



# Dead Code : Clase Materia

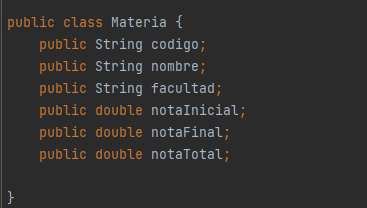
## Consecuencias:

Al momento de revisar el código hay clases que no ha sido utilizado provocando que exista confusión al momento de analizar el código además de que consume tiempo de cómputo en código que jamás se va a utilizar.

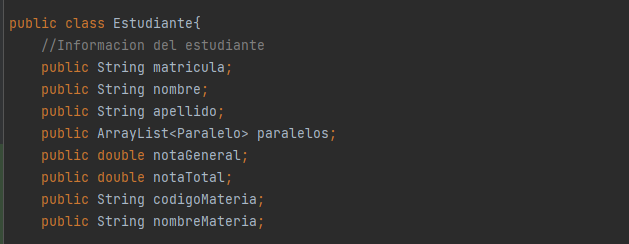
## Refactoring:

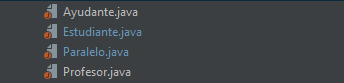
Se aplicará “inline class” que consiste en mover los atributos de una clase a otra hasta que esta clase quede totalmente vacía. Además, se eliminará la clase que ha quedado vacía se procederá a ser eliminada. Se reemplazara las referencias de los atributos de clase donante a la clase recipiente.

## Código Inicial:



## Código Final:





# Data Class: Clase InformacionAdicionalProfesor

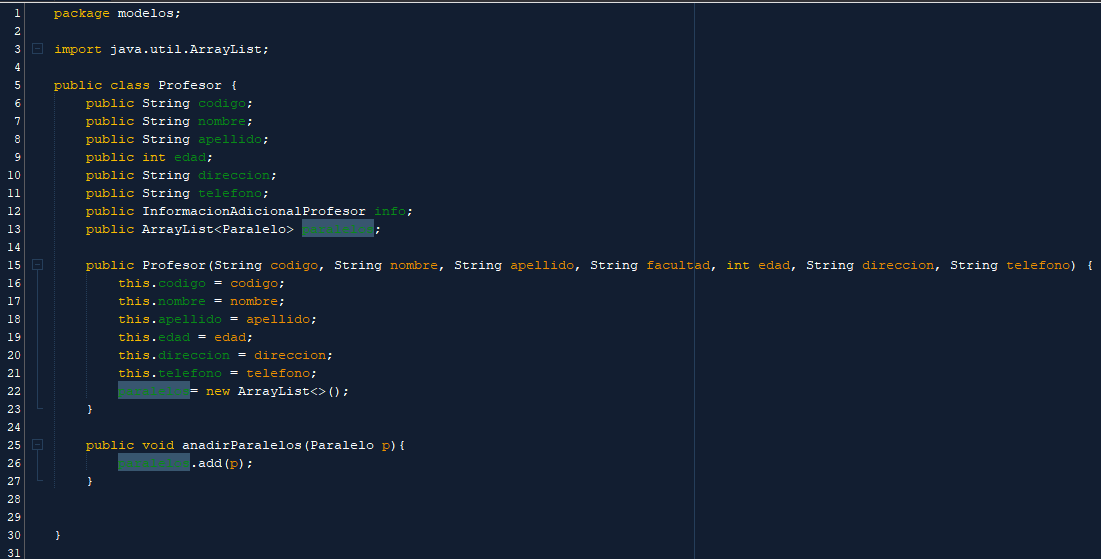
## Consecuencias:

Sería necesario crear más métodos adicionales puesto que los atributos que se usarán se encuentran repartidos entre dos clases distintas, incluso corremos el riesgo de duplicar código en el programa.

## Refactoring:

La clase “InformacionAdicionalProfesor” solamente contiene atributos que perfectamente pueden pertenecer directamente a la clase “Profesor”. En esta clase no hay métodos que contengan acciones adicionales, solamente contiene campos que pertenecen a la clase “Profesor”, podemos eliminar la clase “InformacionAdicionalProfesor” y mover esos atributos a la clase “Profesor”. Además, podemos ver que en ambas clases estos atributos están públicos (no están encapsulados), lo cual no debería ser así.

## Código Inicial:



## Código Final: