**Escuela Superior Politécnica del Litoral**

**DISEÑO DE SOFTWARE**

**Taller Refactoring**

**Integrantes**

Jorge Daniel Mawyin Cabay

Andrés Alfredo Cornejo Figueroa

Carlos Josue Alvia Alarcón

Contenido

[Sección A 3](#_Toc122562925)

[DATA CLASS 3](#_Toc122562926)

[CODIGO DUPLICADO 7](#_Toc122562927)

[INAPPROPRIATE INTIMACY 9](#_Toc122562928)

[SPECULATIVE GENERALITY 11](#_Toc122562929)

[DATA CLUMPS 12](#_Toc122562930)

[DEAD CODE 14](#_Toc122562931)

[SECCION B 16](#_Toc122562932)

[REPOSITORIO 16](#_Toc122562933)

# Sección A

## DATA CLASS

**Consecuencias**

Las clases Materia e InformacionAdicionalProfesor contienen solo campos burdos para acceder a ellos y se puede observar solo contienen atributos públicos para acceder a ellos, por lo tanto, no contienen funcionalidad adicional y no pueden operar de forma independiente con los datos que poseen.

**Técnica de refactorización**

Como las clases contienen campos públicos se emplea la técnica de refactorización “Encapsulate Field***”*** para ocultarlos del acceso directo (cumplir con la encapsulación de datos), para que de esta manera requiera que el acceso se realice solo a través de los métodos getters y setters. Además, en la clase “InformacionAdicionalProfesor” se incluyó el método CalcularSueldo() de la clase calcularSueldoProfesor, eliminando la misma ya que solo contenía dicho método, el cual empleaba atributos de la clase InformacionAdicionalProfesor. También se agregaron los métodos de calcular nota que estaban en la clase estudiante, y se creó una clase Nota para almacenar las variables que se necesitan para calcular las notas.

Por otro lado, cuando se movió el método calcularSueldo a la clase InformacionAdicionalProfesor se generó el code smell **Speculative Generality** ya que no se usaba el atributo prof, por lo que se aplicó la técnica de refactorización **remove parameter**.

**Código inicial**

Clase calcularSueldoProfesor

**Texto

Descripción generada automáticamente**

Clase InformacionAdicionalProfesor

Texto

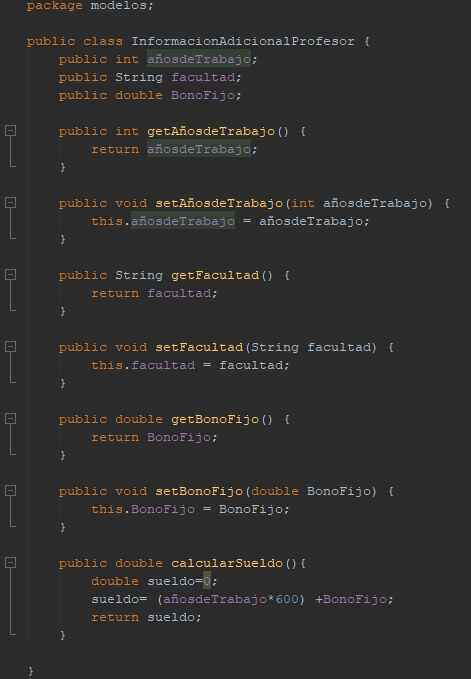
Descripción generada automáticamente

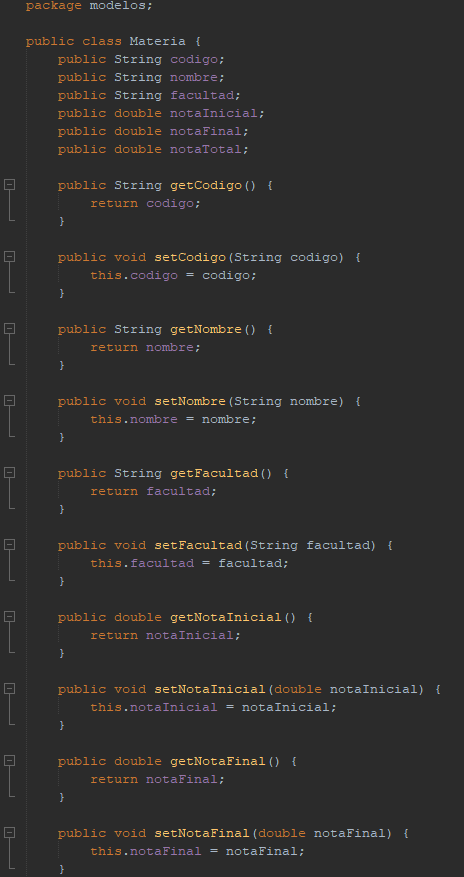
Texto

Descripción generada automáticamenteClase Materia

**Código Refactorizado**

Clase InformacionAdicionalProfesor



Clase Materia

Texto

Descripción generada automáticamente

## CÓDIGO DUPLICADO

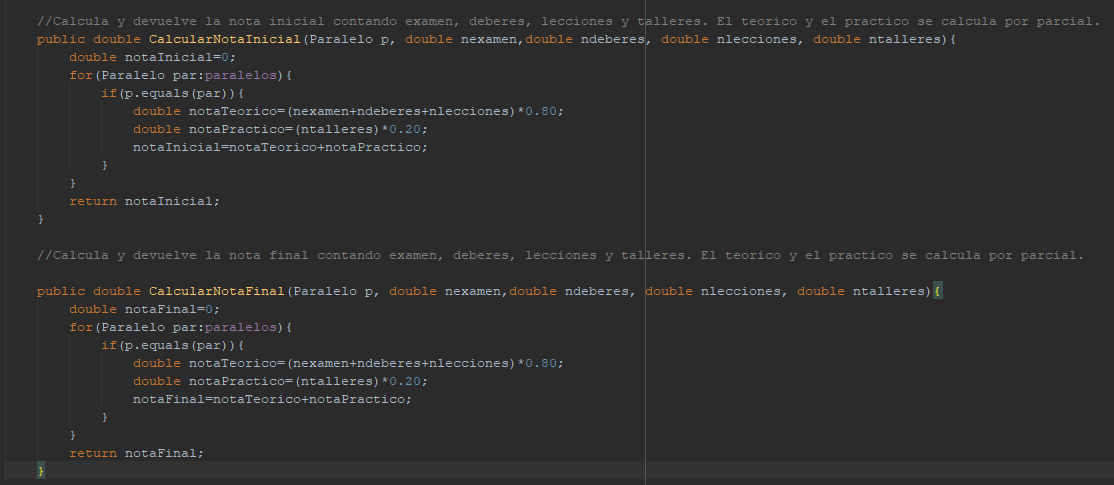
**Consecuencias**

Este smell es un tipo de duplicación sutil ya que ambos métodos al usar el mismo código crean largas secciones repetidas, estos son los métodos CalcularNotaInicial() y CalcularNotaFinal en la clase estudiante. El mantener un código duplicado hace que las clases sean más extensas y dificultan la comprensión del código, adicionalmente, si la lógica del código duplicado cambia en un futuro, esto generaría que se necesite cambiar la implementación en ambos métodos.

**Técnica de refactorización**

Se aplicó la técnica Composing methods ya que esta nos indica que en estos casos se debe remover códigos duplicados y de esta manera poder preparar el código para futuras modificaciones, además, dado que el mismo código se encuentra en dos métodos en la misma clase usamos Extract Method y realizamos llamadas al método extraído en los lugares donde se encontraba originalmente.

**Código inicial**

Clase estudiante

**Código final**

Clase materia

Texto

Descripción generada automáticamente

## INAPPROPRIATE INTIMACY

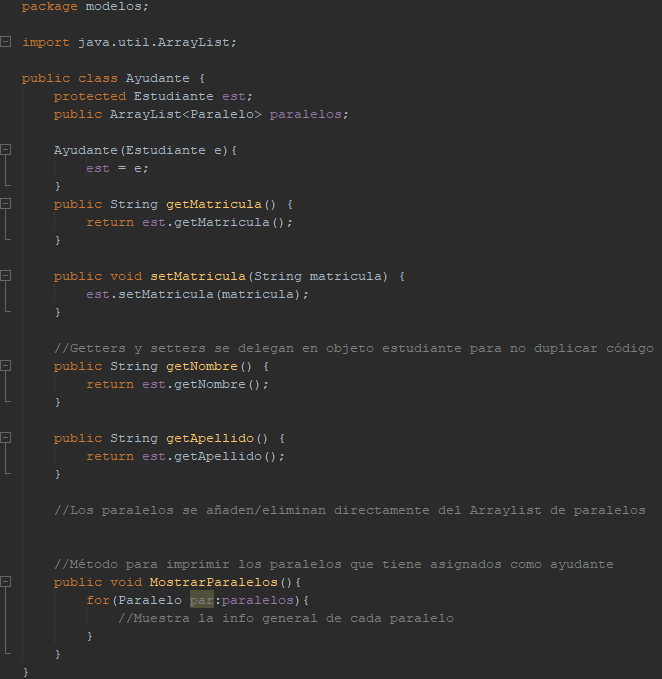
**Consecuencias**

En la clase Ayudantese encuentra como atributo un objeto de la clase Estudiante, sin embargo, su constructor y métodos tales como getMatricula(), getNombre() y demás, hacen referencias a métodos de la clase *Estudiante*.

**Técnica de refactorización**

Como la clase Ayudante contiene muchos métodos delegados de la clase Estudiante, entonces se procede a utilizar la técnica de refactorización**Replace Delegation with Inheritance*,*** de esta manera se heredarán los atributos y métodos de la clase Estudiante.

**Código inicial**

****Clase Ayudante

**Código final**

Texto

Descripción generada automáticamenteClase Ayudante

## 

## SPECULATIVE GENERALITY

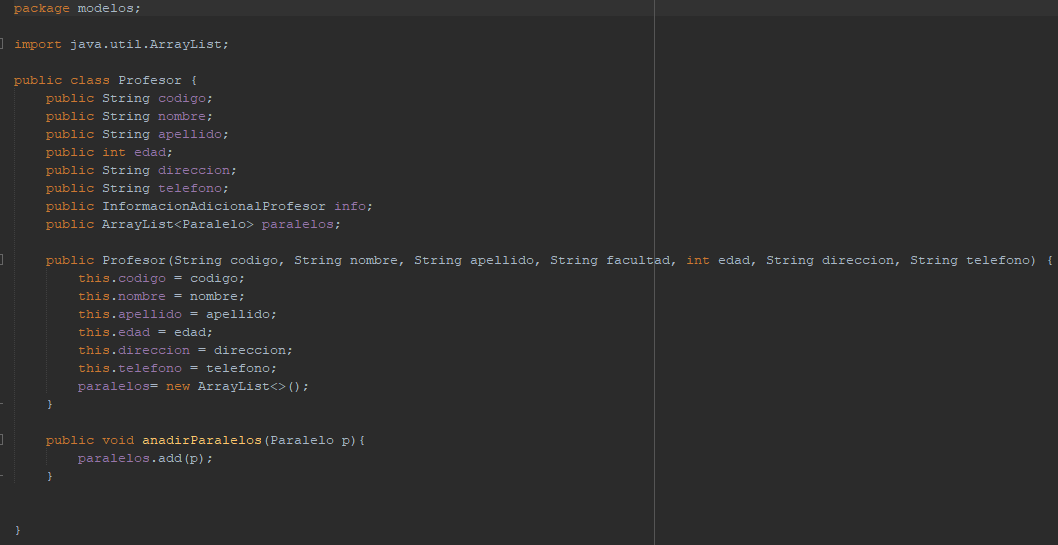
**Consecuencias**

Entro los parámetros del constructor de la clase profesor se encuentra la variable facultad, la cual no es utilizada dentro de la implementación de este. Cada parámetro en una llamada a un método juega un rol importante ya que obliga al programador a leerlo para averiguar qué información se encuentra en este parámetro. Y si un parámetro no se en el cuerpo del método, este es en vano. Adicionalmente, en la implementación del resto del código no se utiliza dicho atributo, permitiendo considerar que este campo se dejó definido para una implementación a futuro.

**Técnica de refactorización**

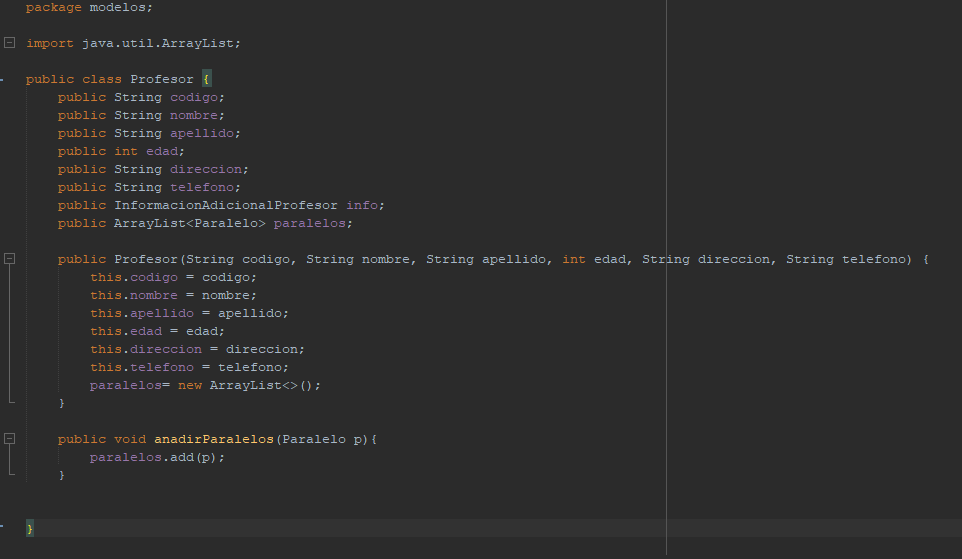
La técnica de refactorización que ayudará a solucionar este smell es **Remove Parameter**, la cual hará que dicho método contenga únicamente los parámetros que realmente requiere.

**Código inicial**

Clase profesor

**Código final**

Clase Profesor



## 

## DATA CLUMPS

**Consecuencias**

En la clase estudiante existen varios métodos que contienen el mismo grupo de parámetros (CalcularNotaInicial, CalcularNotaFinal y CalcularNota) que están haciendo el código más difícil de entender perjudicando la reutilización. Esto provoca la duplicación de código tanto de los propios parámetros como de las operaciones relacionadas. Se puede reconocer debido a que una sola de esas variables no tiene mucho sentido sin las demás, si dejamos solamente la nota de talleres denota la clara falta de los otros parámetros. Si se elimina este code smell mejorará la comprensión y organización del código.

**Técnica de refactorización**

La técnica de refactorización a utilizar sería **Introduce Parameter Object,** ya que se utiliza cuando los métodos contienen un grupo repetido de parámetros. Se aplica esta técnica reemplazando estos parámetros por una instancia de la clase Nota creada anteriormente.

**Código inicial**

Clase Estudiante

Texto

Descripción generada automáticamente

**Código final**

Clase Estudiante

Texto

Descripción generada automáticamente

## DEAD CODE

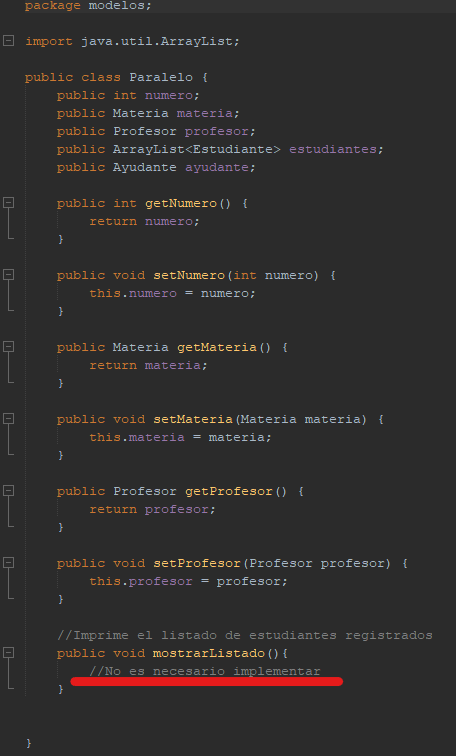
**Consecuencias**

En la clase Paralelo existe un método sin usar, estos métodos se crean por si en un futuro se puede admitir características que nunca son implementadas como consecuencia de este code smell el código puede generar confusión y se tiene un método obsoleto en la actualidad.

**Técnica de refactorización**

Al tener la clase Paralelo un método sin implementar se usará la técnica de refactorización de Sustituir Algoritmo. Para ello primero se debe asegurar que el método mostrarListado() no se encuentre redefinido en subclases y eliminar el método.

**Código inicial**

Clase Paralelo

**Código final**

Texto

Descripción generada automáticamenteClase Paralelo

# SECCION B

## REPOSITORIO

<https://github.com/JorgeMawyin/Taller08-Refactoring>