**Taller #5**

**Integrantes**

* Carrillo Carrillo Hayleen Ivilin.
* Pazmiño Guerrero Gabriela Nicole.

**Grupo:** 5

**Tema**:

Code smells y refactorización

Contenido

[**Code Smell: Large class** 3](#_Toc48249222)

[**Code Smell: Duplicate Code** 6](#_Toc48249223)

[**Code Smell: Temporary fields** 7](#_Toc48249224)

[**Code Smell: Long Paramether List** 8](#_Toc48249225)

[**Code Smell: Data Class** 10](#_Toc48249226)

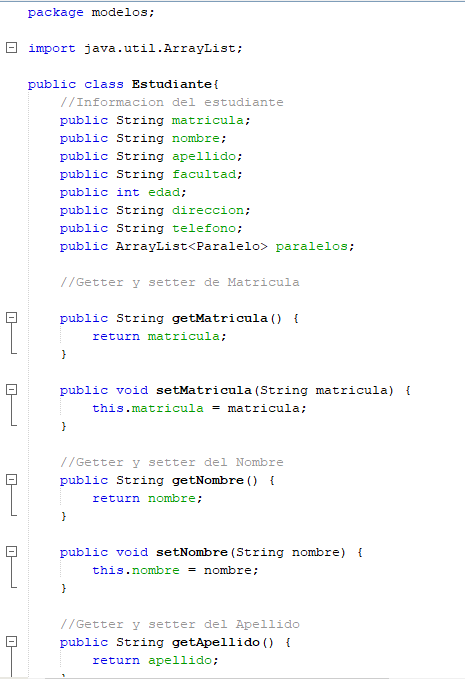
[**Code smell: Feature envy** 11](#_Toc48249227)

# **Code Smell: Large class**

**Clase donde se encuentra:**

Estudiante

**Captura Inicial:**





**Explicación:**

Dentro de esta clase existen métodos como calcularNotaInicial() y calcularNotaFina() que no tienen mucha relación con la clase Estudiante. A más de eso existen muchos getters y setters innecesarios ya que todos los atributos son públicos y no necesitan métodos para acceder a estos datos. La posible solución es eliminar todo lo que este de más y refactorizar los métodos duplicados (enviarlos a una nueva clase que solo cumpla con la función que se requiere, en este caso, calcular notas del estudiante.

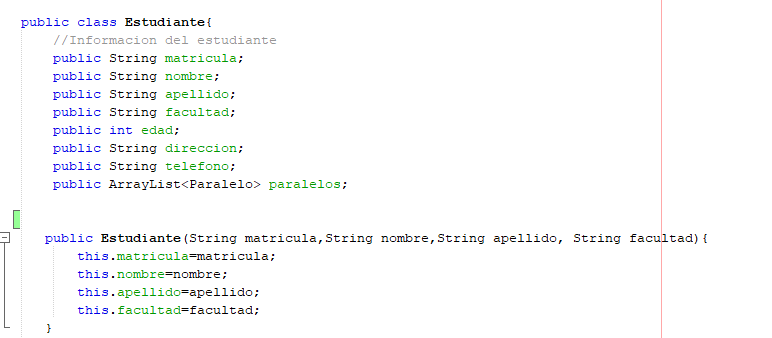
**Consecuencias:**

Todo esto hace que la clase se vuelva demasiado larga y realice funcione que no le correspondan.

**Técnica de refactorización:**

Redefinir métodos como una nueva clase (los métodos sobre calcular notas serán una nueva clase).

**Código Refactorizado:**

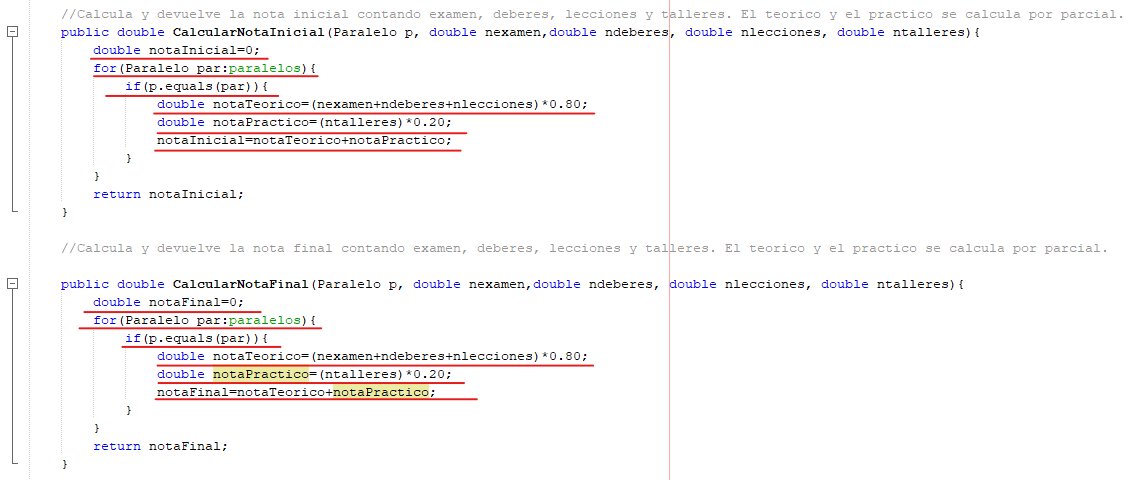


****

# **Code Smell: Duplicate Code**

**Clase donde se encuentra:** calcularNotasEstudiante

**Captura Inicial:**



**Explicación:**

En esta clase tenemos los mismos parámetros y funcionamiento entre los métodos calcularNotaInicial() y calcularNotaFinal. Estos fragmentos de códigos son idénticos, pero poseen un nombre diferente, entonces, una posible solución es combinar ambos métodos y simplificar la estructura del código, haciéndolo más corto y funcional.

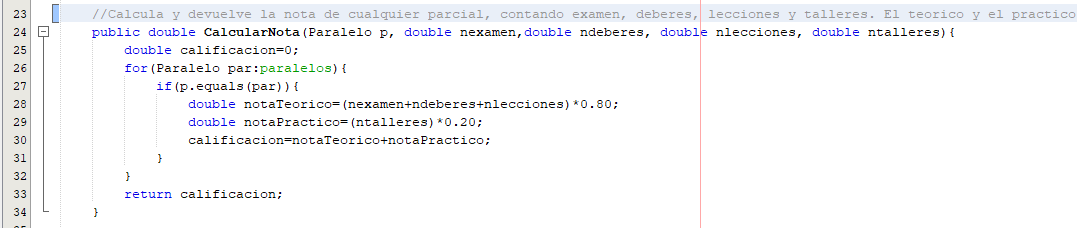
**Consecuencias:**

Hacer que el código se vuelva más largo innecesariamente.

**Técnica de refactorización:**

Extraer métodos.

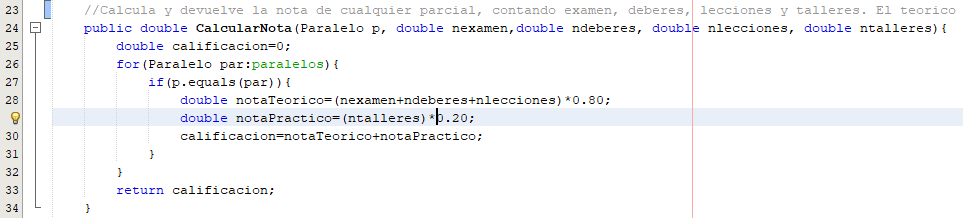
**Código Refactorizado:**

****

# **Code Smell: Temporary fields**

**Clase donde se encuentra:** calcularNotasEstudiante

**Captura Inicial:**



**Explicación:**

En este segmento de código (ya refactorizado) se generan dos variables temporales, que son notaTeorico y notaPractico, que solo se usan para agregarlas al valor de calificación, lo cual podría solucionarse con la adición de los valores generados, pero de forma directa a la variable calificación.

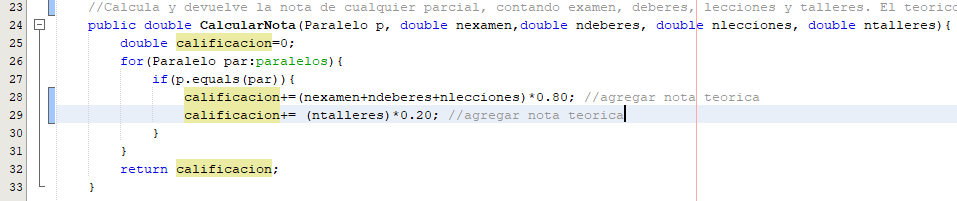
**Consecuencias:**

Hacer que el código ocupe más cantidad de memoria, debido a que se generan variables temporales cuando, en lugar de ello, podría sumarse directamente a la variable calificación (para mejor entendimiento del usuario, se agregará de forma individual la parte teórica y la práctica).

**Técnica de refactorización:**

Inline TEMP.

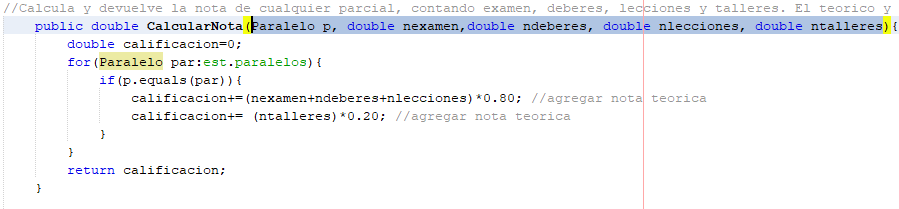
**Código Refactorizado:**

****

# **Code Smell: Long Paramether List**

**Clase donde se encuentra:** calcularNotasEstudiante

**Captura Inicial:**

****

**Explicación:**

Dentro de la clase calcularNotasEstudiante el método CalcularNota recibe muchos parámetros, en este caso, el usuario deberá ingresar cinco parámetros, lo cual podría generar confusión debido a que un parámetro podría tomar el lugar del otro (debido al orden específico que se tiene), entonces, lo que se debe hacer es reestructurar dicho método y reducir los parámetros utilizados en los mismos.

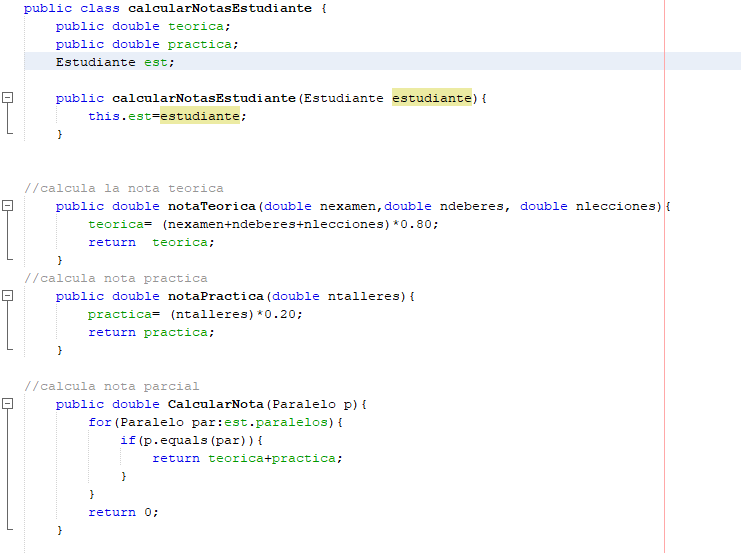
**Consecuencias:**

Esta cantidad de parámetros puede resultar tediosa para el usuario ya que al ingresar tantos datos puede cometer errores en el orden y generar inconvenientes en el programa.

**Técnica de refactorización:**

Sustituir algoritmo (es mejor simplificar dicho método para que se pueda utilizar de forma general en el futuro; para este caso, se dividió el método calcularNota en teórica y práctica para evitar que el usuario ingrese muchos parámetros, luego, se pide el paralelo del estudiante y, si se encuentra dentro de sus paralelos registrados, se realiza una suma entre los resultados de ambos métodos para obtener la nota parcial del estudiante, caso contrario, retornará cero).

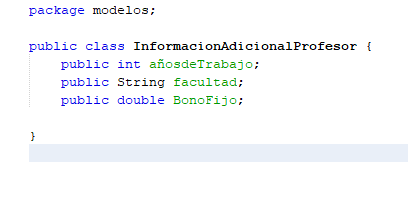
**Código Refactorizado:**

****

# **Code Smell: Data Class**

**Clase donde se encuentra:** InformacionAdicionalProfesor

**Captura Inicial:**



**Explicación:**

Esta es una clase que funciona como contenedor de datos dentro del programa. En ella solo hay atributos extras del profesor. Esta clase no tiene ninguna funcionalidad y no puede operar de manera independiente con los datos que posee. Lo ideal es colocar los atributos de esta clase en la clase Profesor y manejarlos directamente ahí. Esto mejoraría el tamaño y organización del programa.

**Consecuencias:**

El tener una clase que solo contiene datos como InformacionAdicionalProfesorgenera perdida de dinero y tiempo a más de agregar tamaño innecesario al código.

**Técnica de refactorización:** Fusionar clase

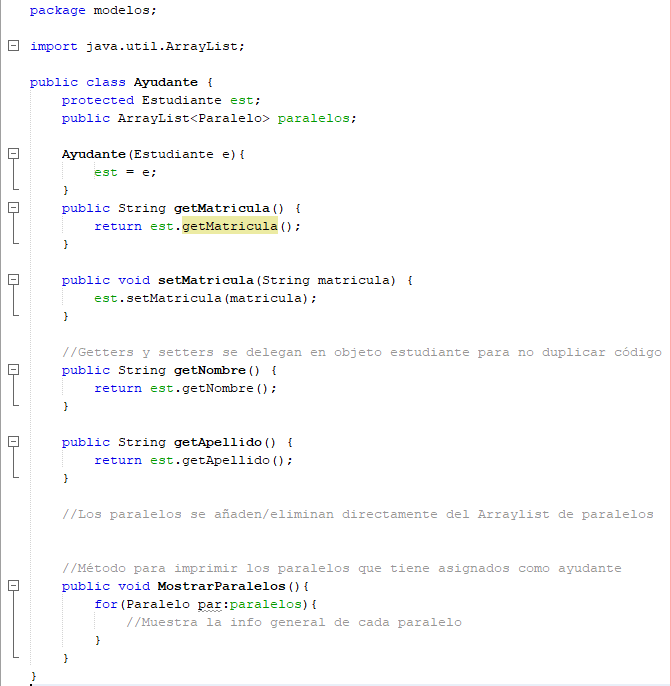
**Código Refactorizado:**

# **Code smell: Feature envy**

**Clase donde se encuentra:**

Ayudante

**Captura Inicial:**



**Explicación:**

Existe una instancia de estudiante en la clase mencionada, tanto así que, para que esta clase sea construida, se utiliza al objeto Estudiante, llegando a utilizar casi todos los datos de este objeto para que la clase Ayudante funcione correctamente.

**Consecuencias:**

Invasión total a una clase diferente a Ayudante, debido a que se utilizan la mayoría de los atributos para que funcione, incluso, lo único "nuevo" que realiza es la implementación del método Mostrar paralelos, en el que se presentan los paralelos en los que es ayudante.

**Técnica de refactorización:**

Reemplazar delegación con herencia

**Código Refactorizado:**

