**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

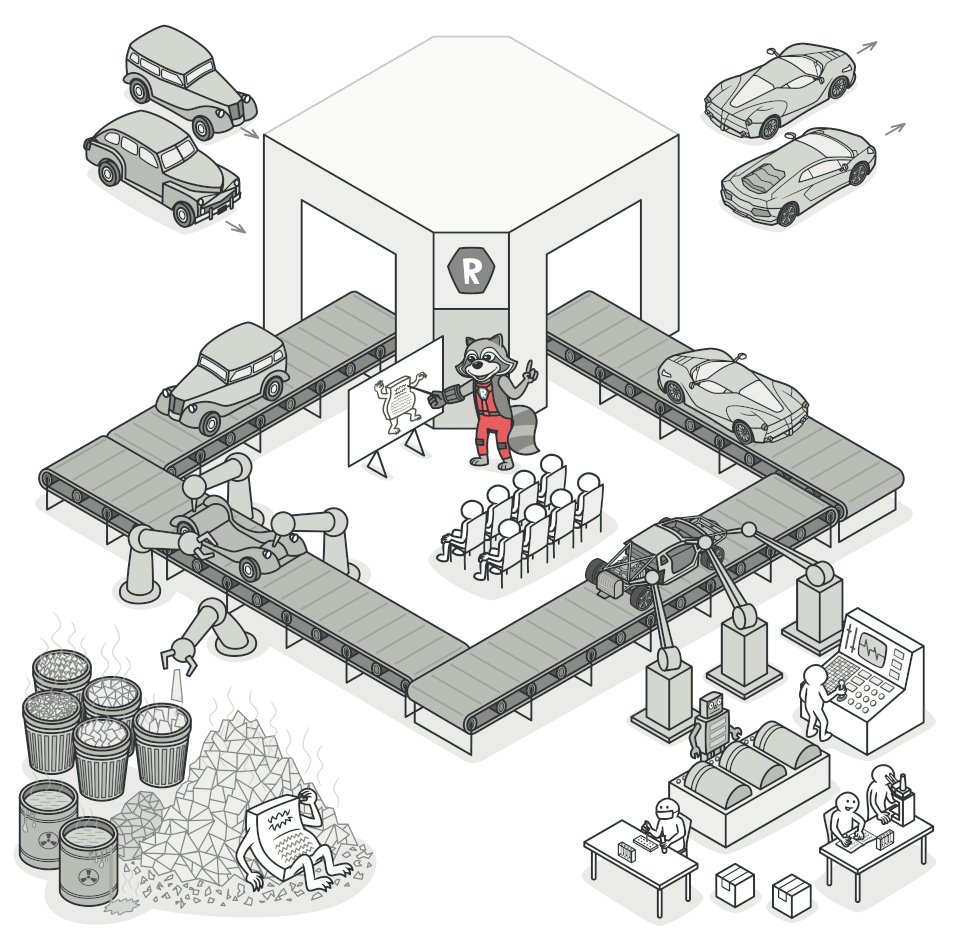
**FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN**

**DISEÑO DE SOFTWARE**

**Taller 08 REFACTORING - 2021**

**PARALELO 103**

**GRUPO 4:**



BRAVO SERNA DAVID ALVARO

FRANCO HIDALGO KEYLA FERNANDA

LOPEZ CASTILLO JOHANNA CECILIA

**Link del repositorio: https://github.com/KeylaFrancoH/Taller08-Refactoring.git**

Contenido

[SECCIÓN A 1](#_Toc79694070)

[1. IDENTIFICACIÓN DE MALOS OLORES 1](#_Toc79694071)

[1.1. Duplicate code 1](#_Toc79694072)

[1.1.1. Descripción 1](#_Toc79694073)

[1.1.2. Consecuencias 1](#_Toc79694074)

[1.1.3. Técnica para su eliminación - Extract Method 1](#_Toc79694075)

[1.1.1. Captura código al inicio 1](#_Toc79694076)

[1.1.2. Código después de refactorización 1](#_Toc79694077)

[1.2. Data Class 2](#_Toc79694078)

[1.2.1. Descripción 2](#_Toc79694079)

[1.2.2. Consecuencia 2](#_Toc79694080)

[1.2.3. Técnica para su eliminación – Move fields 2](#_Toc79694081)

[1.2.4. Captura código al inicio 2](#_Toc79694082)

[1.2.5. Código después de refactorización 2](#_Toc79694083)

[1.3. Inappropiate Intimacy 3](#_Toc79694084)

[1.3.1. Descripción 3](#_Toc79694085)

[1.3.2. Consecuencia 3](#_Toc79694086)

[1.3.3. Técnica para su eliminación – Move method y Move field 3](#_Toc79694087)

[1.3.4. Captura código al inicio 3](#_Toc79694088)

[1.3.5. Código después de refactorización 4](#_Toc79694089)

[1.4. Long Parameter List 5](#_Toc79694090)

[1.4.1. Descripción 5](#_Toc79694091)

[1.4.2. Consecuencia 5](#_Toc79694092)

[1.4.3. Técnica para su eliminación – Remove Parameter 5](#_Toc79694093)

[1.4.4. Captura código al inicio 5](#_Toc79694094)

[1.4.5. Código después de refactorización 5](#_Toc79694095)

[1.5. Feature Envy 7](#_Toc79694096)

[1.5.1. Descripción 7](#_Toc79694097)

[1.5.2. Consecuencias 7](#_Toc79694098)

[1.5.3. Técnica para su eliminación - Replace delegatin with inheritance 7](#_Toc79694099)

[1.5.4. Captura código al inicio 7](#_Toc79694100)

[1.5.5. Código después de refactorización 7](#_Toc79694101)

[1.6. Large class 9](#_Toc79694102)

[1.6.1. Descripción 9](#_Toc79694103)

[1.6.2. Consecuencias 9](#_Toc79694104)

[1.6.3. Técnica para su eliminación – Extract Class 9](#_Toc79694105)

[1.6.4. Captura código al inicio 9](#_Toc79694106)

[1.6.5. Código después de refactorización 9](#_Toc79694107)

**TALLER DE REFACTORING**

SECCIÓN A

1. IDENTIFICACIÓN DE MALOS OLORES
   1. Duplicate code
      1. Descripción

En la clase Estudiante se encuentran dos métodos que poseen el mismo código y realizan las mismas acciones, esto se nota en los métodos CalcularNotaInicial y CalcularNotaFinal.

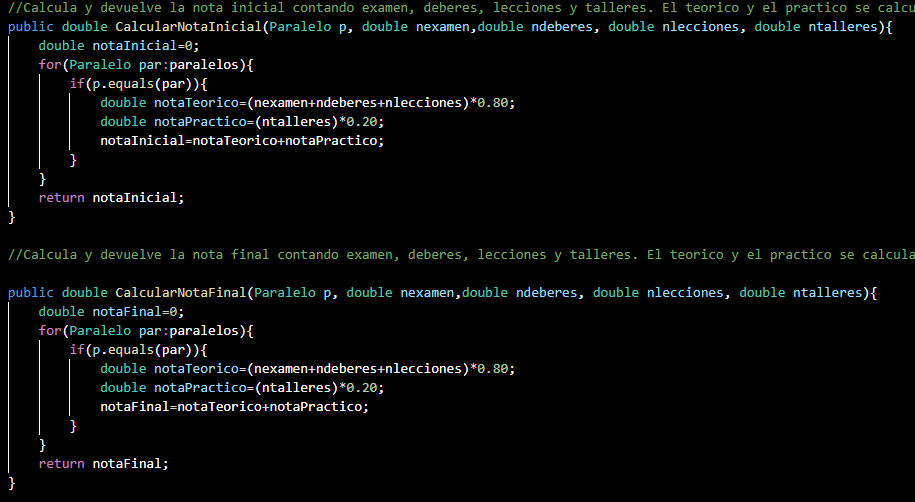
* + 1. Consecuencias

Se puede ocasionar problemas al momento de tener que modificar el código en cada uno de los métodos que lo utilizan, puede causar errores o el mal funcionamiento de este debido a que puede haber olvidos en alguna línea de código, lo cual alteraría el correcto funcionamiento del programa.

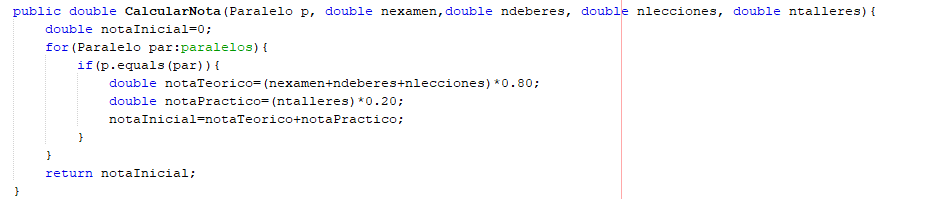
* + 1. Técnica para su eliminación - Extract Method

**Extract Method:** Esta técnica permite extraer el código para de esta manera poder agruparlo en un solo método separado y facilitar la modificación del método en cada lugar que se utilice.

* + 1. Captura código al inicio



* + 1. Código después de refactorización



* 1. Data Class
     1. Descripción

La clase InformacionAdicionalProfesor posee atributos que pueden ser modificados fuera de la clase, sin embargo, estos atributos pueden pertenecer directamente a la clase Profesor sin necesidad de dividirse en otra.

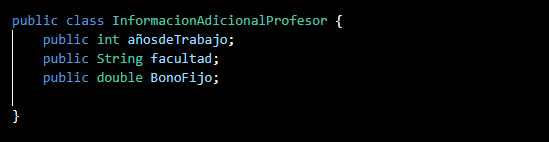
* + 1. Consecuencia

Se podría crear código duplicado ya que se podrían crear los atributos faltantes sin conocer que se encuentran en otra clase, también sería necesario crear más métodos adicionales correspondientes porque sus otros atributos se encuentran distribuidos en otras clases.

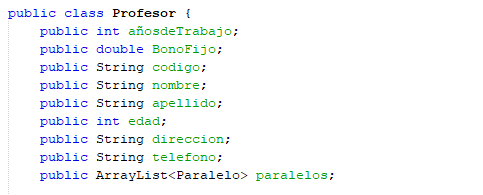
* + 1. Técnica para su eliminación – Move fields

**Move fields:** Se mueven los atributos de InformacionAdicionalProfesor a la clase Profesor y luego se elimina la clase.

* + 1. Captura código al inicio



* + 1. Código después de refactorización



* 1. Inappropiate Intimacy
     1. Descripción

En la clase estudiante y en la clase CalcularSueldoProfesor se encontraron métodos que usaban los campos internos de otra clase.

* + 1. Consecuencia

El código se vuelve más difícil de mantener, además, el control de datos de la otra clase es más complicado ya que la información puede ser manipulada. También se vuelve más difícil de comprender.

* + 1. Técnica para su eliminación – Move method y Move field

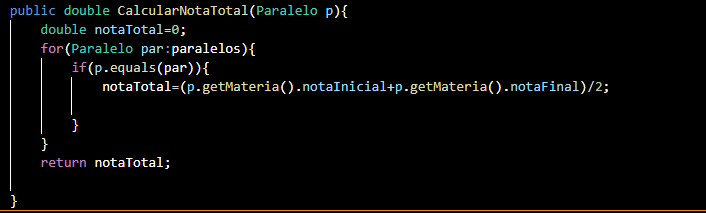
Move method – CalcularSueldoProfesor, el método estaba contenido en una clase llamada calcularSueldoProfesor, procedimos a cambiarla esa responsabilidad a la clase profesor, y que ella se encargara de procesarla

Move field – CalcularNotaTotal, se movieron los parámetros que usaba el método calcularNotaTotal a la clase Materia para no estar abusando de los parámetros de otra clase

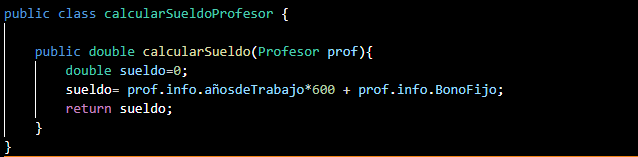
Move method – CalcularNotaTotal, se movió el método desde la clase estudiante a la clase Materia, para que la clase sea más coherente internamente

* + 1. Captura código al inicio

**CalcularNotaTotal – Clase Estudiante**

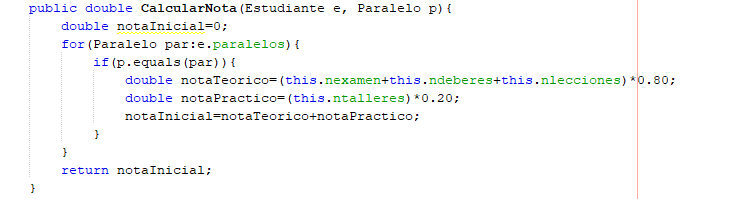


**CalcularSueldoProfesor – Clase CalcularSueldoProfesor**

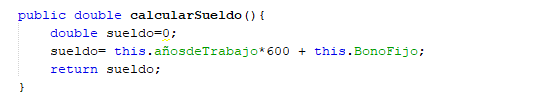


* + 1. Código después de refactorización

**CalcularNota – Clase Materia**



**CalcularSueldoProfesor – Clase Profesor**



* 1. Long Parameter List
     1. Descripción

En la clase Profesor, en el método constructor encontramos el code smell Long Parameter List, esto se genera debido a querer delegar menos entre clases utilizando demasiados parámetros para que el método actúe como se precisa, lo identificamos porque tiene más de 4 parámetros.

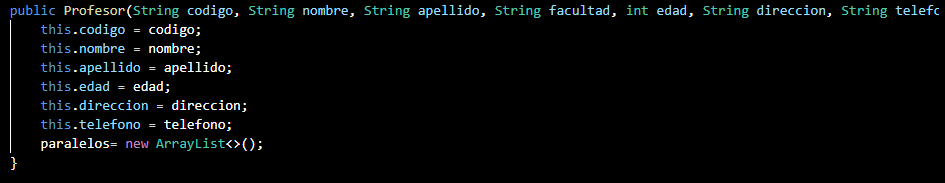
* + 1. Consecuencia

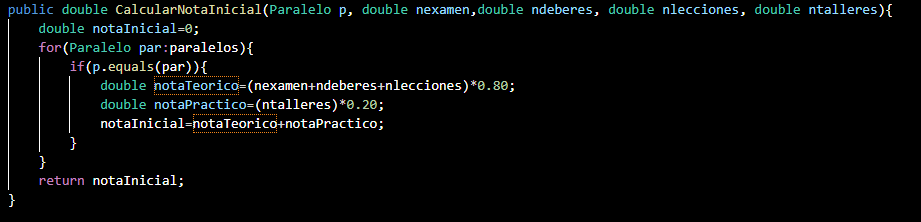
La lista de parámetros larga hace que el código sea más difícil de usar y comprender, este code smell puede deberse a métodos demasiados complejos y crea dependencias.

* + 1. Técnica para su eliminación – Remove Parameter

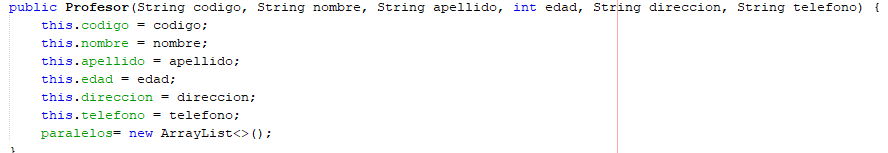
**Remove Parameter:** La técnica en este code smell permitió remover el parámetro facultad en el método constructor de la clase Profesor, en los métodos CalcularNotaInicial de la clase Materia también se removieron parámetros

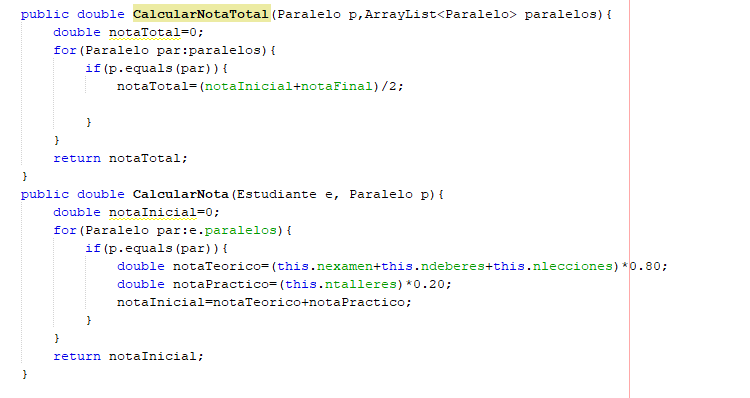
* + 1. Captura código al inicio





* + 1. Código después de refactorización





* 1. Feature Envy
     1. Descripción

La clase Ayudante accede a los atributos de la clase Estudiante, ya que sus métodos getters y setters delegan sus funciones a la clase Estudiante, además tienen funciones parecidas por lo cual se puede convertir en una herencia.

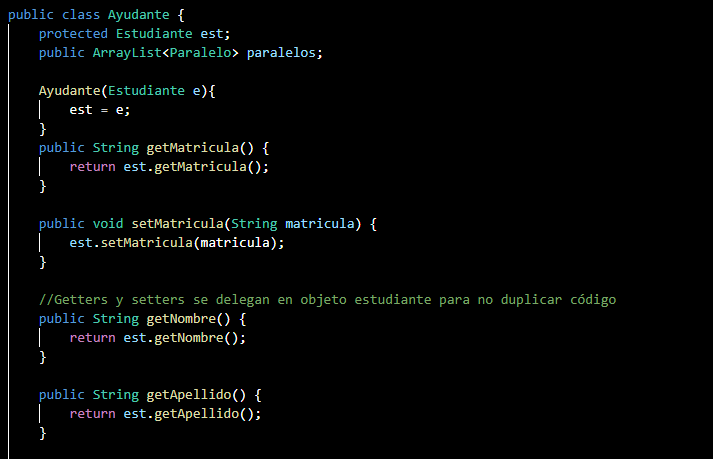
* + 1. Consecuencias

El código se vuelve más complicado de entender por el usuario, por otro lado, el código puede ser reemplazado por la herencia de la otra clase, poseyendo los métodos y atributos de Estudiante.

* + 1. Técnica para su eliminación - Replace delegatin with inheritance

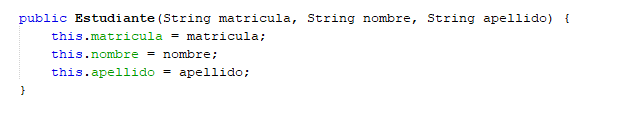
**Replace delegatin with inheritance:** Evita que la clase posea gran cantidad de métodos innecesarios, además, evita la necesidad de crearlos para cada nuevo método de clase delegada, en este caso la clase ayudante como debería heredar de la clase estudiante, se eliminó el atributo estudiante y ahora a la clase estudiante le pertenecen los atributos de la clase estudiante

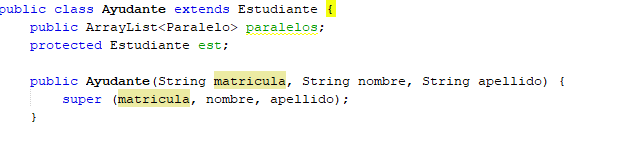
* + 1. Captura código al inicio



* + 1. Código después de refactorización

Se crea un constructor en la clase Estudiante para que Ayudante pueda heredarlo y usar super.





* 1. Large class
     1. Descripción

En la clase Estudiante tenemos el code smell Large class, la clase tiene los métodos CalcularNotaInicial, CalcularNotaFinal, CalcularNotaTotal, estos métodos no corresponden a esta clase porque se le está dando más responsabilidades a la clase Estudiante, haciendo crecer el código de esta clase

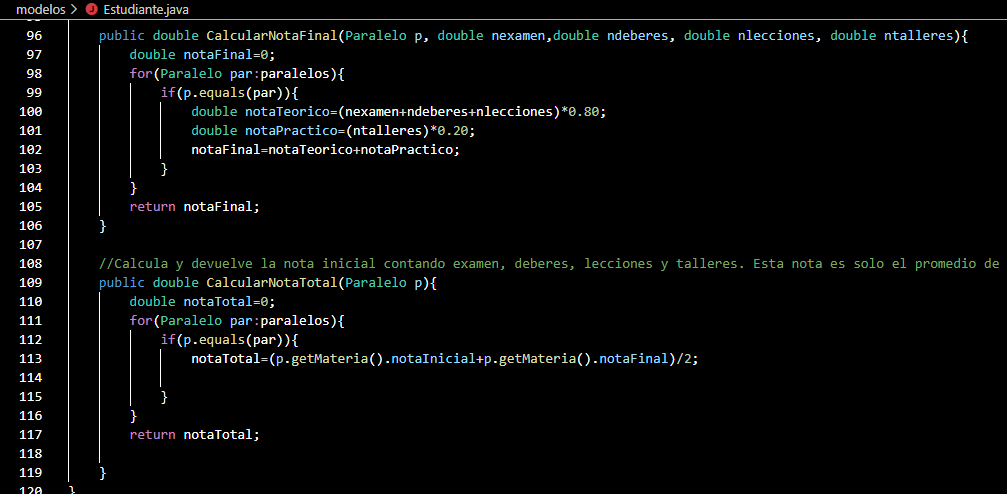
* + 1. Consecuencias

El código se vuelve más difícil de entender, leer y desarrollar, creando la tendencia de centralizar funciones en esta clase, aumentando las posibilidades de fallas en el futuro.

* + 1. Técnica para su eliminación – Extract Class

**Extract Class:** La técnica que usamos nos permite extraer parte del comportamiento de la clase que puede ser usado de distintas formas.

* + 1. Captura código al inicio



* + 1. Código después de refactorización

Se extrajeron los métodos CalcularNotaInicial, CalcularNotaFinal, CalcularNotaTotal, con la finalidad de reducir la clase y cumplir con el principio de Single Responsibility de los principios SOLID

