Imagen que contiene dibujo, alimentos

Descripción generada automáticamente

ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD INGENIERÍA ELÉCTRICA Y COMPUTACIÓN

DISEÑO DE SOFTWARE

Taller Patrones de Diseño

Pertenece a:

Robinson Flores Y.

Daniel Guerrero R.

Ricardo Salazar S.

I TÉRMINO 2020

[Ilustración 1 Data Class en clase Materia 3](#_Toc48250652)

[Ilustración 2 Data Class en clase InformacionAdicionalProfesor 4](#_Toc48250653)

[Ilustración 3 Refactorización Data Class, clase Materia 4](#_Toc48250654)

[Ilustración 4 Refactorización Data Class, clase InformacionAdicionalProfesor 5](#_Toc48250655)

[Ilustración 5 Data Class en Estudiante, producto de refactorización 5](#_Toc48250656)

[Ilustración 6 Data Class en Estudiante refactorizado 6](#_Toc48250657)

[Ilustración 7 Duplicate Code en metodos calcularnota Inical y Final 6](#_Toc48250658)

[Ilustración 8 Refactorizacion de "Duplicate Code" 6](#_Toc48250659)

[Ilustración 9 Lazy Class en clase CalcularSueldoProfesor 7](#_Toc48250660)

[Ilustración 10 Refactorización Lazy Class en clase CalcularSueldoProfesor 7](#_Toc48250661)

[Ilustración 11 Inapropiate Intimazy en clase Ayudante 8](file:///C:\Users\tvsme\Documents\NetBeansProjects\Taller_Refactoring-master\Taller%20Refactorizacion%20Entregable.docx#_Toc48250662)

[Ilustración 12 Refactorización Inapropiate Intimazy en clase Ayudante 9](#_Toc48250663)

[Ilustración 13 Data Class en clase Profesor 9](#_Toc48250664)

[Ilustración 14 refactorización Data Class en clase Profesor 10](#_Toc48250665)

[Ilustración 15 Inappropiate Intimacy en metodo calcularSueldo 10](#_Toc48250666)

[Ilustración 16 Refactorización Inappropiate Intimacy 10](#_Toc48250667)

**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

**FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN**

**DISEÑO DE SOFTWARE**

**Taller de Refactoring - I TÉRMINO 2020**

**Objetivos**

* Identificar malos olores de programación en el código fuente adjunto y las técnicas de refactorización correspondientes.
* Aplicar técnicas de refactorización que pueden aplicarse para eliminar los malos olores previamente identificados.

**Descripción**

En equipos de trabajo, conformados por hasta tres estudiantes, (i) identifique los malos olores de programación encontrados en el código fuente adjunto, (ii) Identifique las técnicas de refactorización adecuadas para eliminar los malos olores encontrados y (iii) Refactorice el código fuente para obtener un código más limpio y fácil de leer. Justifique su respuesta.

**Especificaciones**

Considere un **sistema académico** que permite manejar los estudiantes registrados en ciertos paralelos de distintas materias. Además, cada materia tiene un profesor y podría tener asignado un ayudante.

**Code Smells**

**DISPENSABLES:**

“**Data Class**”: Este mal olor está presente en las clases Materia e InformacionAdicionalProfesor, las cuales solo tienen atributos que sirven para guardar datos necesarios para otras clases. Estas clases no operan sobre los datos que tienen como atributos, simplemente los almacenan.

Es necesario arreglar este “smell” para limpiar el código y hacerlo más fácil de entender. Estos datos deben moverse a la clase la cual realiza cambios sobre ellos. También disminuye tiempos y costo al tener que revisar menos clases.

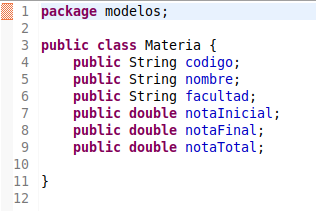


Ilustración 1 Data Class en clase Materia

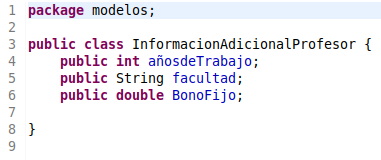


Ilustración 2 Data Class en clase InformacionAdicionalProfesor

Para dar solución a este problema se procede a la extracción y reubicación de métodos que estén haciendo uso de estas variables. Además, se encapsula sus datos como privados para poder ser accedidos por “getters” y “setters”.



Ilustración 3 Refactorización Data Class, clase Materia

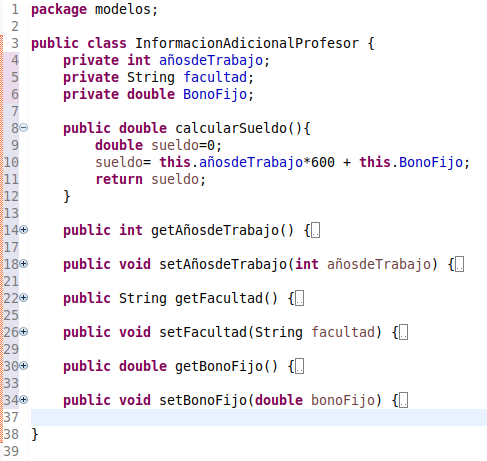


Ilustración 4 Refactorización Data Class, clase InformacionAdicionalProfesor

**“Data Class”:** Una vez realizado el cambio de mover el metodo a la clase identificada como “Data Class” se genera un nuevo mal olor de este tipo en la clase Estudiante, la cual presenta todos sus atributos publicos y los metodos “getters and setters” de los mismos, mas no realiza ningun cambio sobre los datos.

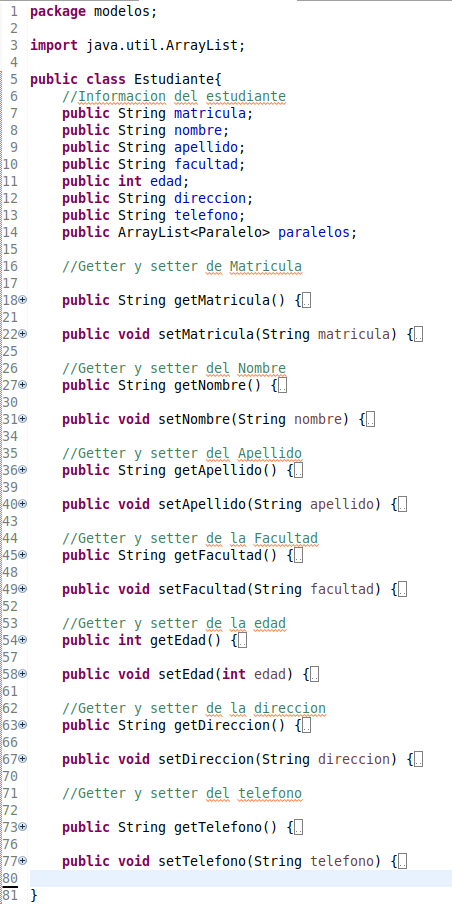


Ilustración 5 Data Class en Estudiante, producto de refactorización

Se debe agregar la funcionalidad para que la clase estudiante pueda trabajar sobre sus datos, ya que en el sistema académico se requiere que este pueda registrarse a nuevos paralelos, además se debe agregar su constructor. Se usa el método de “Refactoring” llamado “Encapsulate Field” con el motivo de hacer a los atributos solo accesibles por medio de las funciones “getters and setters”

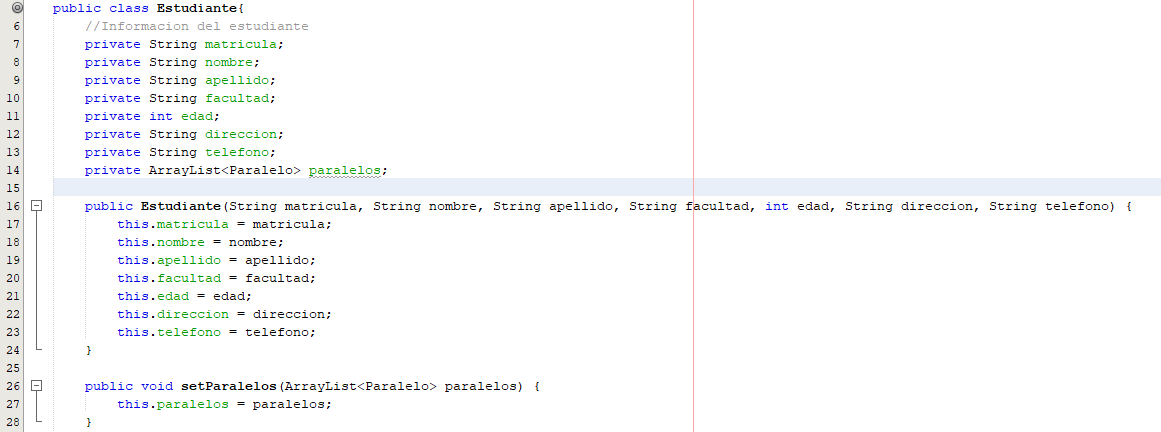


Ilustración 6 Data Class en Estudiante refactorizado

**“Duplicate Code”:** Este “code smell” se presenta cuando existe código que parece idéntico en varios métodos o clases. Dejar este tipo de errores siempre dificultara la lectura y entendimiento del código, además de ocupar líneas innecesarias de código.

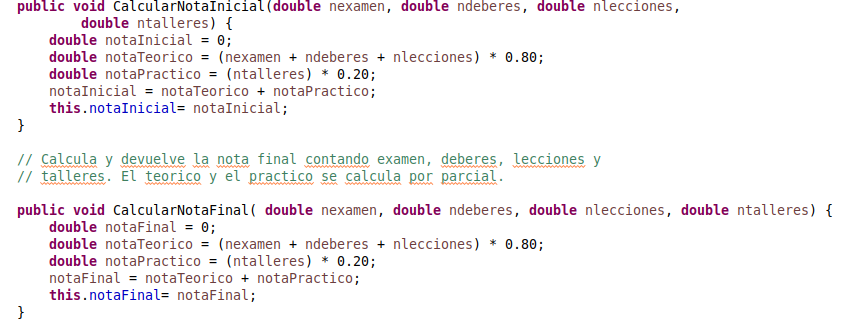


Ilustración 7 Duplicate Code en metodos calcularnota Inical y Final

Se procede a unificar el código ya que este funciona de la misma forma y lo único que varia es su nombre.

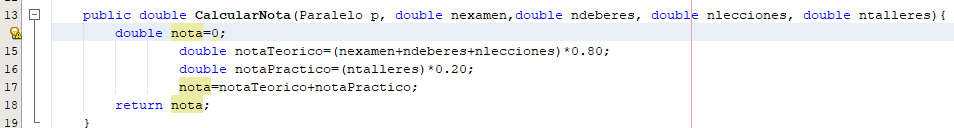


Ilustración 8 Refactorizacion de "Duplicate Code"

“**Lazy Class**” : Presente en la clase calcularSueldoProfesor, esta clase solo cuenta con un método que calcula el sueldo del profesor y no cuenta con atributos propios de sí misma.

Se debe solucionar el “code smell” ya que no presenta signos de mayor implementación de funcionalidades a futuro y el tamaño del código no es significativo para mantener esa porción de código en una clase dedicada, y se reduce el tiempo y coste de mantenimiento de todo el código del sistema.

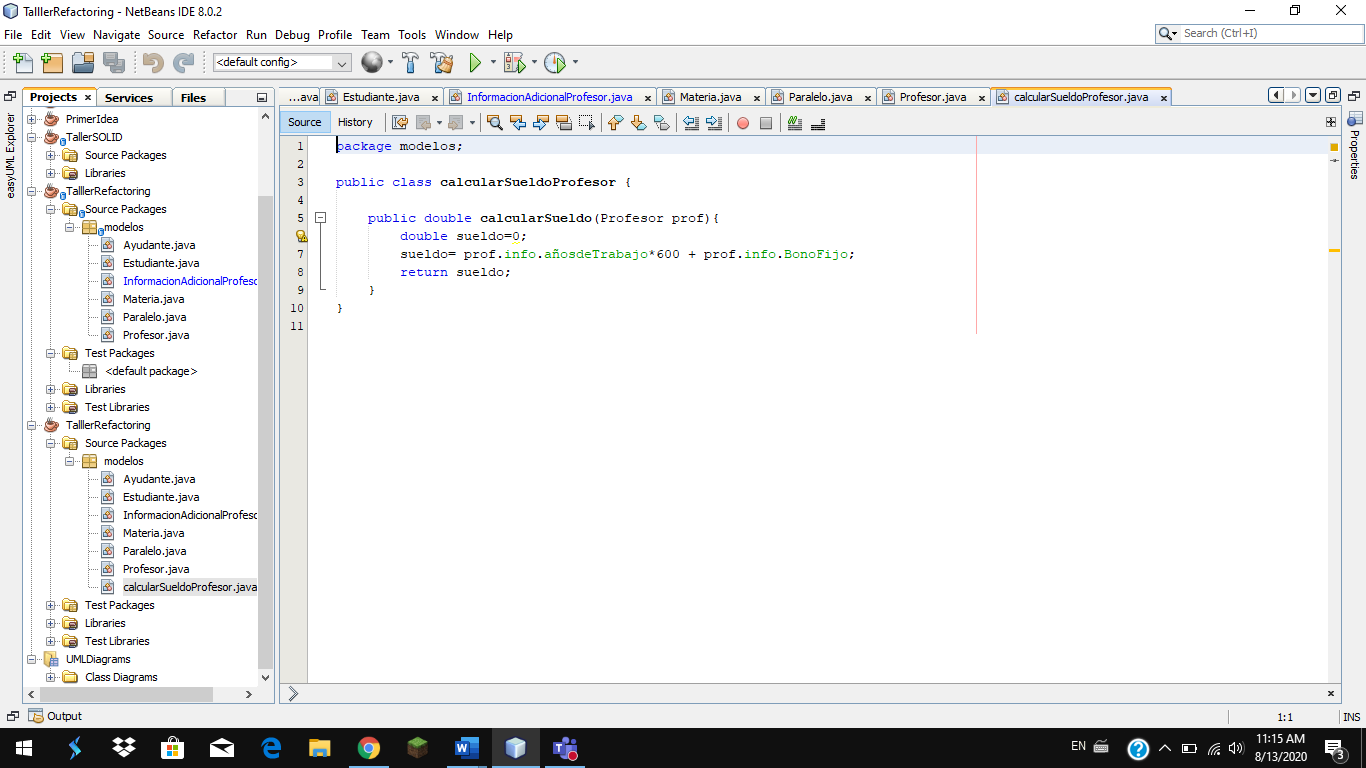


Ilustración 9 Lazy Class en clase CalcularSueldoProfesor

Se usa la técnica de refactorización “**Inline Class**” para mover el método a la clase InformaciónAdicionalProfesor ya que esta clase tiene atributos que el método calcularSueldoProfesor utiliza.

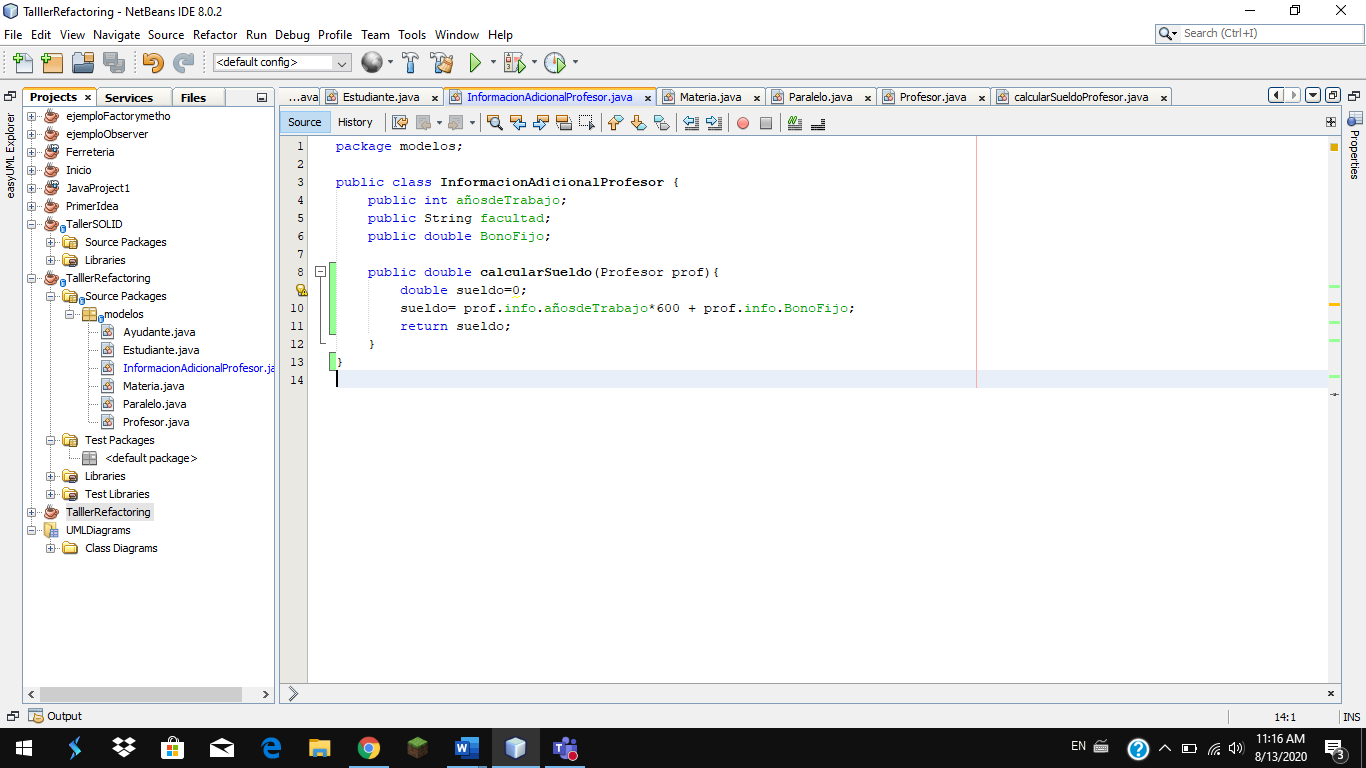


Ilustración 10 Refactorización Lazy Class en clase CalcularSueldoProfesor

**COUPLERS**:

“**Inappropriate Intimacy**”: Mal olor detectado en la clase Ayudante ya que usa métodos de la clase estudiante dentro de los que quiere llamar sus métodos, además tiene como parámetro un objeto estudiante y luego en el constructor se asigna un estudiante al atributo estudiante de la clase Ayudante para acceder a los métodos getters y setters de Estudiante.

Se debe corregir el mal olor para asegurar la abstracción y que ambas clases sepan lo menso posible entre si.

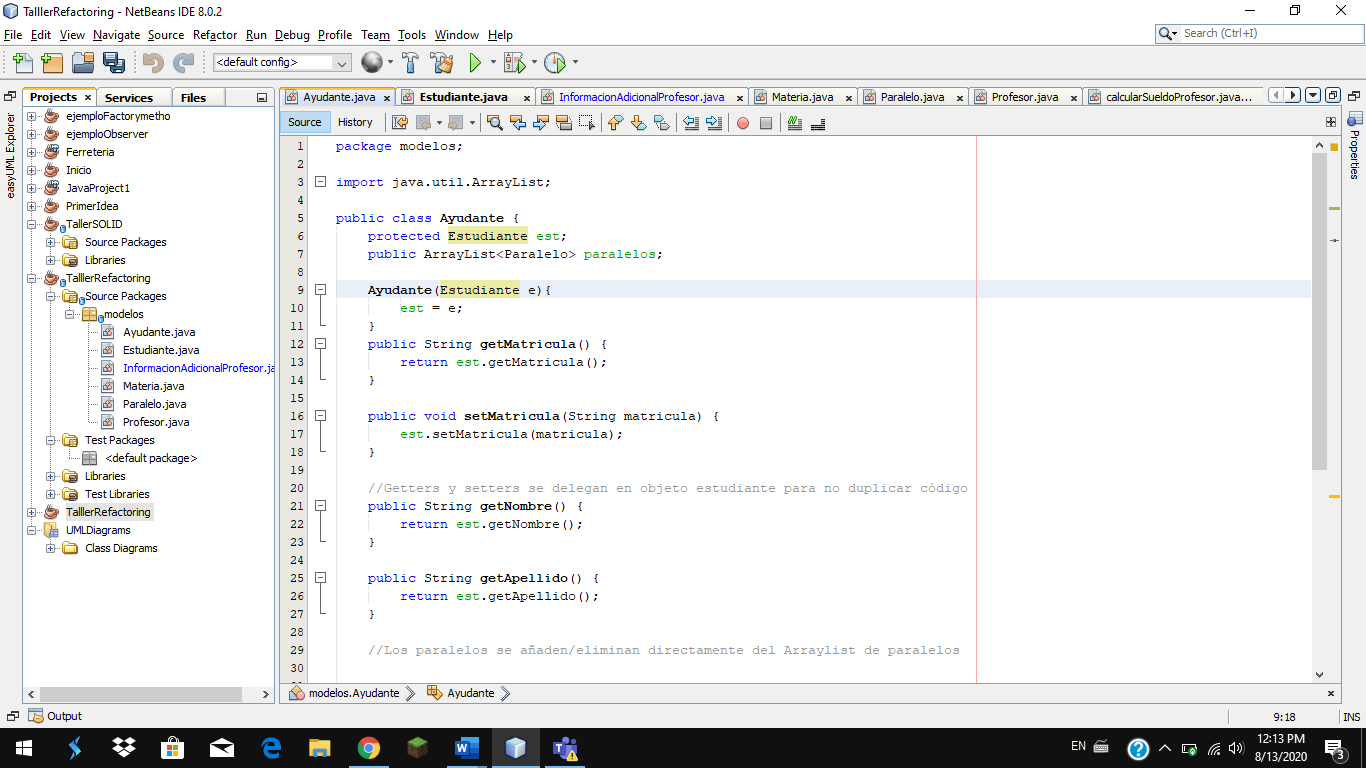


Ilustración 11 Inapropiate Intimazy en clase Ayudante

Par solucionar el mal olor se usa la tecnica de rafactorizacion “**Replace Delegation with Inheritance**” se debe hacer que clase Ayudante herede de la clase Estudiante y eliminar metodos asociados a la clase Estudiante de la clase Ayudante.

Eliminar el atributo est de la clase ayudante.

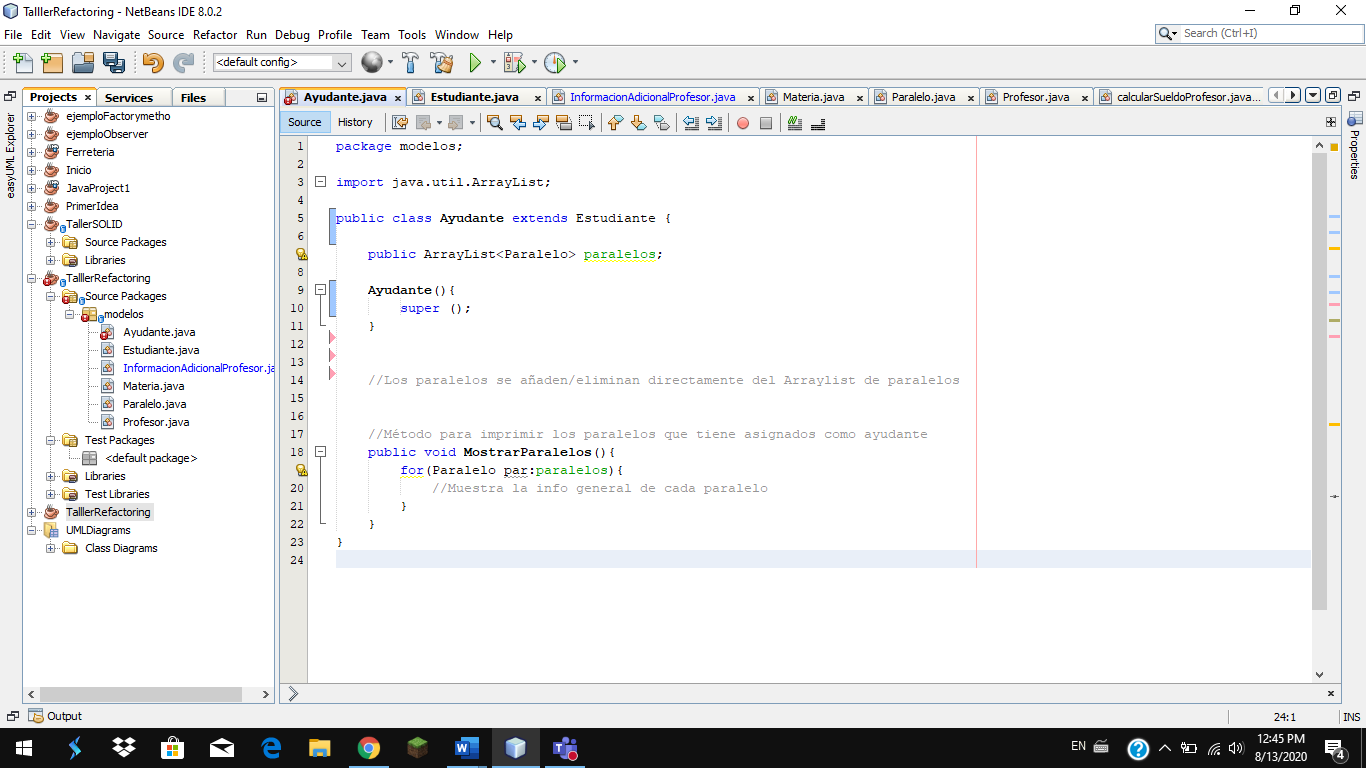


Ilustración 12 Refactorización Inapropiate Intimazy en clase Ayudante

**Data Class**

El mal olor en este código refiere a Data Class, debido a que dentro de Profesor los campos estan puestos en publico cuando deberían ser privados o protected.

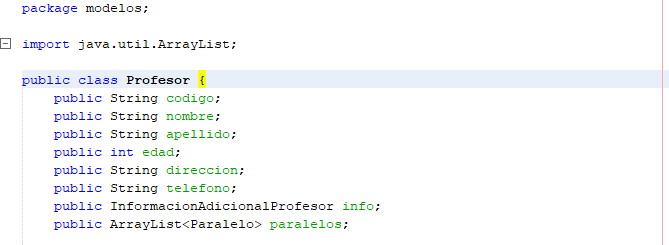


Ilustración 13 Data Class en clase Profesor

La técnica de refactorizacion que se utilizará aquí es Encapsulate Field el cual dice que se deben cambiar los atributos de una clase de publico a privado.

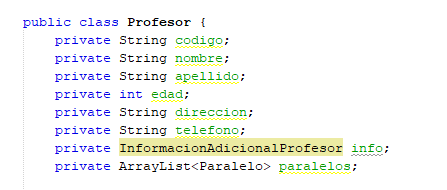


Ilustración 14 refactorización Data Class en clase Profesor

**Inappropiate Intimacy**

El code smell para esta porcion de código es inappropiate intimacy debido a que el metodo calcularSueldo de la clase CalcularSueldoProfesor usa un par de atributos internos de otra clase sin usar getters y setters. En este caso usa el campo privado info de la clase Profesor.

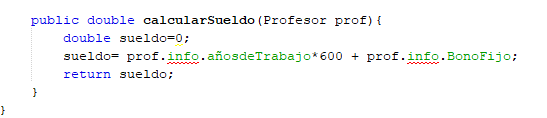


Ilustración 15 Inappropiate Intimacy en metodo calcularSueldo

La técnica de refactorización utilizda el Move Method moviendo este método a la clase profesor para que info pueda ser accedida.

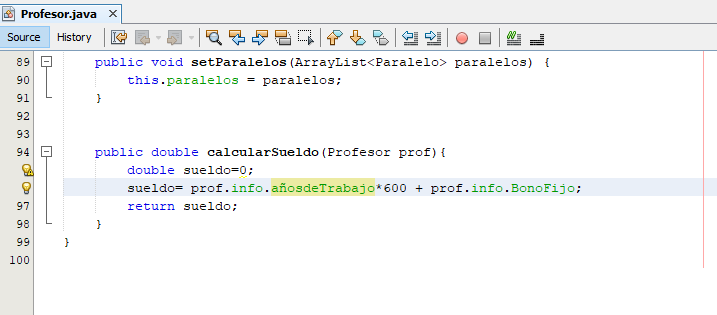


Ilustración Refactorización Inappropiate Intimacy