Tarea de Patrones de Diseño

Integrantes

Johan Gilces

Danny Loor

Joangie Márquez

Gabriela Mora

Emanuel Parra

Dr. Carlos Mera

Paralelo 1

23 de diciembre 2020

Escuela Politécnica del Litoral

Facultad de Ingeniería en Electricidad y Computación

Diseño de

Contenido

[Clase: Profesor 3](#_Toc59646417)

[Clase: CalcularSueldoProfesor 5](#_Toc59646418)

[Clase: Estudiante 6](#_Toc59646419)

[Clase: Materia 9](#_Toc59646420)

[Clase: Ayudante 11](#_Toc59646421)

Sección A

## Clase: Profesor

Code Smell

* Long Parameter List

Consecuencias

* Tener un método con muchos parámetros es una señal de baja cohesión. Esto hace que el código sea difícil de mantener, de probar, de reusar o incluso entender. Asimismo, puede confundir a los usuarios ya que fácilmente pueden equivocarse colocando los parámetros en un orden diferente.

Técnicas

* Extract Super Class
  + Reemplazar la composición con Persona por Herencia

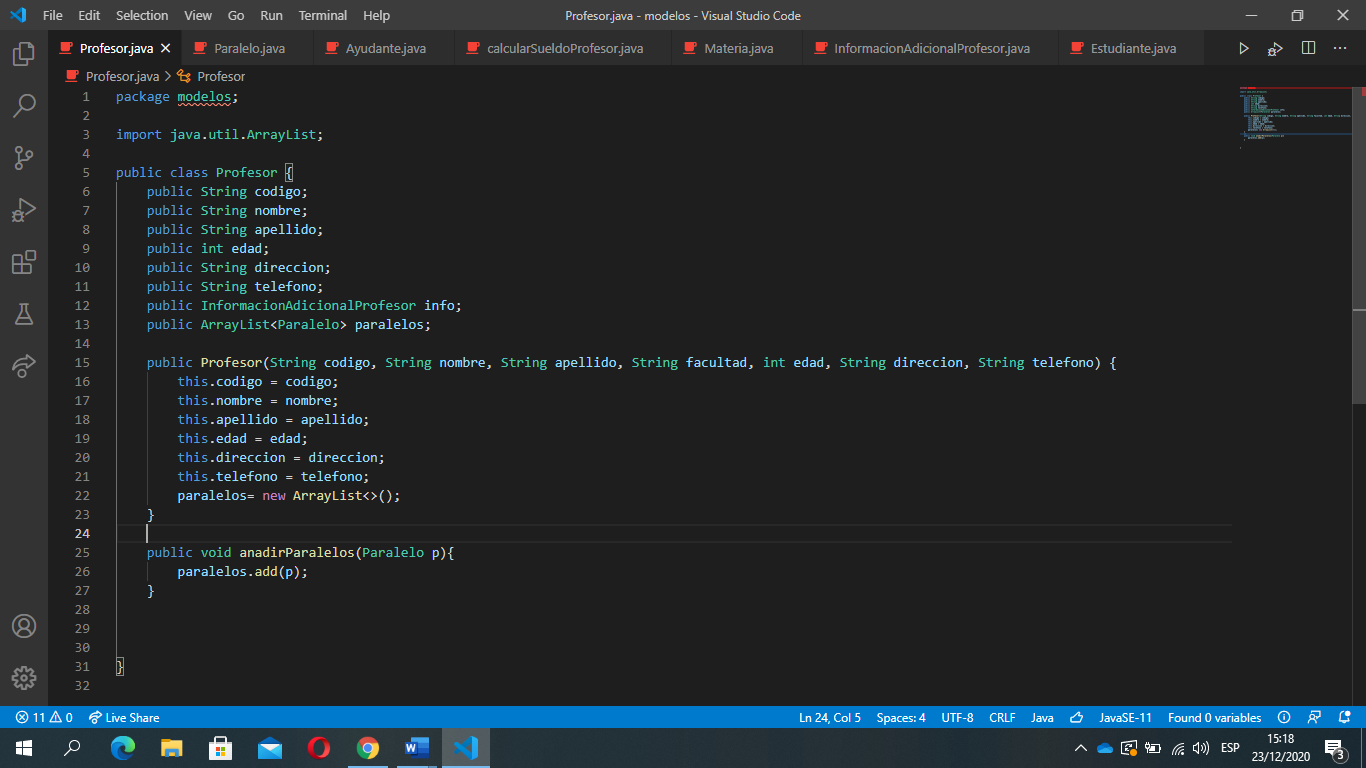
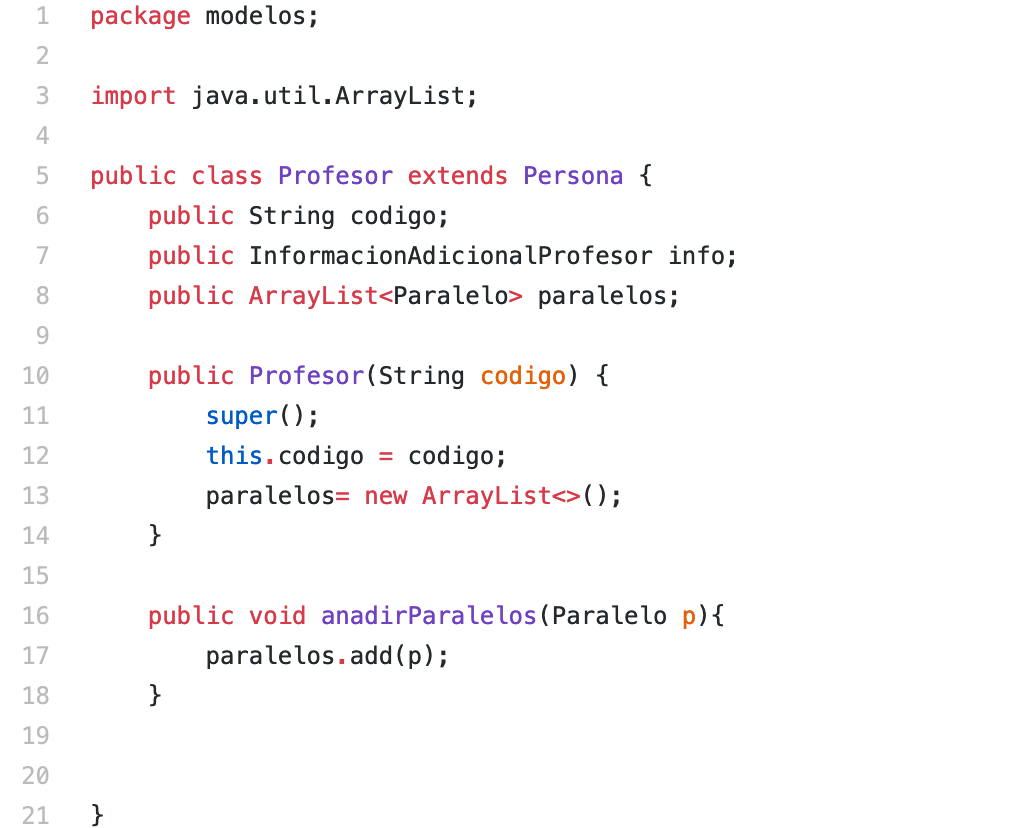


Ilustración 1Antes de la Refactorización





## Clase: CalcularSueldoProfesor

Code Smell

* Inappropriate Intimacy

Consecuencias

* Principalmente, el problema con ‘Inappropriate Intimacy’ es que hace el código más difícil de leer y de entender. Esto a su vez ocasiona que el mantenimiento del código tome más tiempo y compromete la encapsulación de la clase a la que se le solicita información. Además, dificulta implementar mejoras y nuevas características.

Técnicas

* Move Method
  + Mover el método calcularSueldo a la clase InformacionAdicionalProfesor

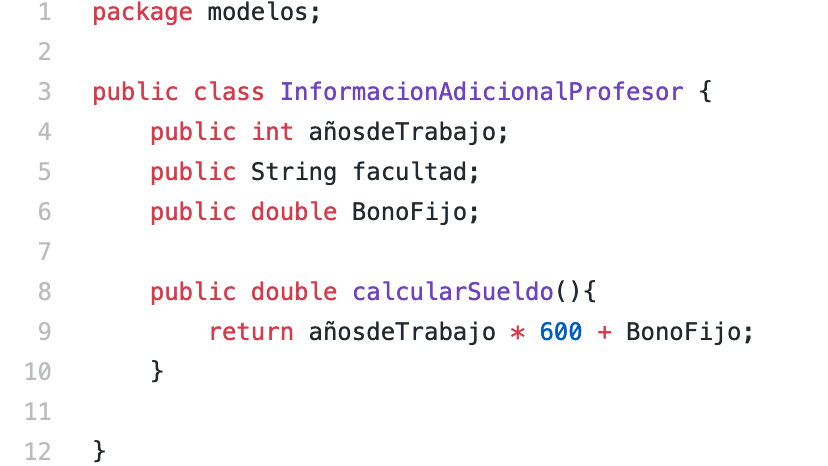
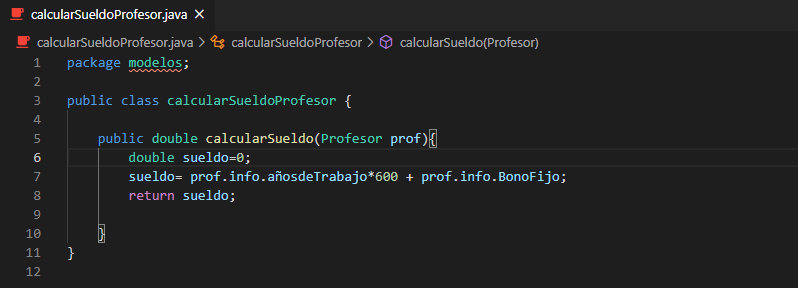


Ilustración 4Antes de la Refactorización

## Clase: Estudiante

Code Smell

* Comments

Consecuencias

* Los comentarios no deben ser utilizados para explicar los métodos ya que en un futuro podría evitar que estos sean mejorados, incluso si se llega a comentar cada método esto podría enmascarar otros code smells en el código.

Técnicas

* Delete comments
  + Los comentarios de los atributos y getters&setters están demás.

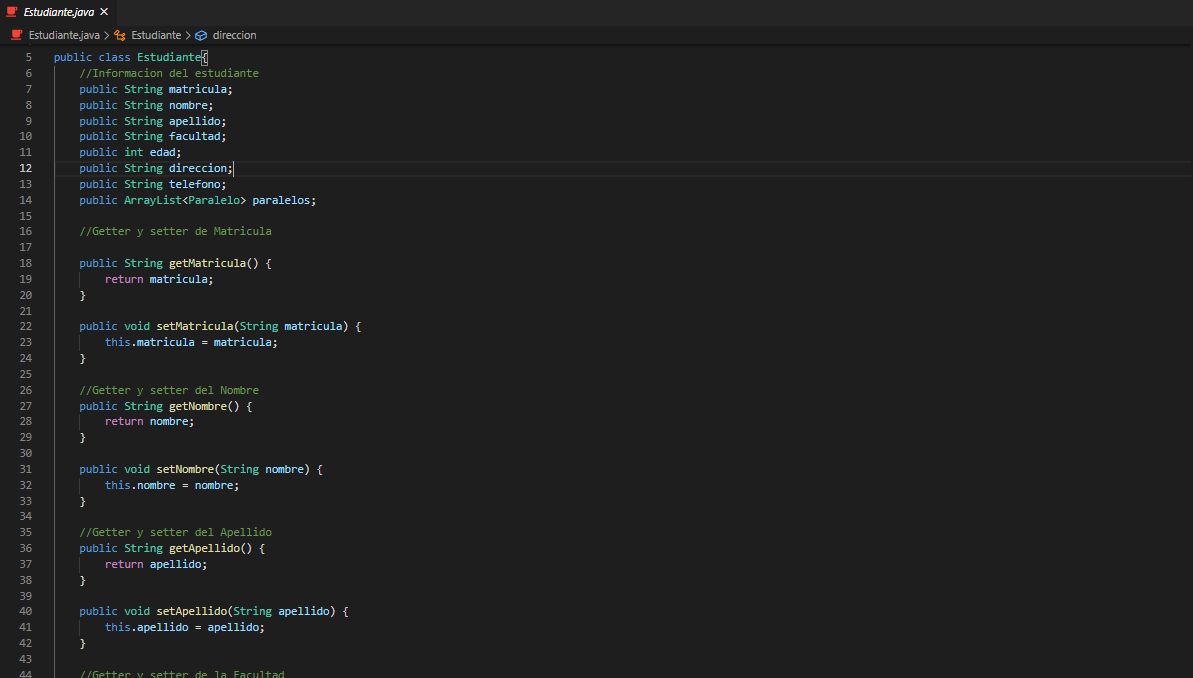


Ilustración 5Antes de la Refactorización

Code Smell

* Duplicate code

Consecuencias

* En este caso existen dos métodos que realizan la misma serie de pasos para calcular las notas de un estudiante difiriendo únicamente en que parcial se encuentra, y teniendo en cuenta que el cálculo para ambos es el mismo cada vez que se actualice el algoritmo en uno, el otro también debe mutar, robando mayor tiempo del desarrollador para realizar dicha acción.

Técnicas

* Rename Method & Merge Method & Move Method (extra)
  + CalcularNota se modificó, se cambió su nombre, se lo agregó a la clase Materia

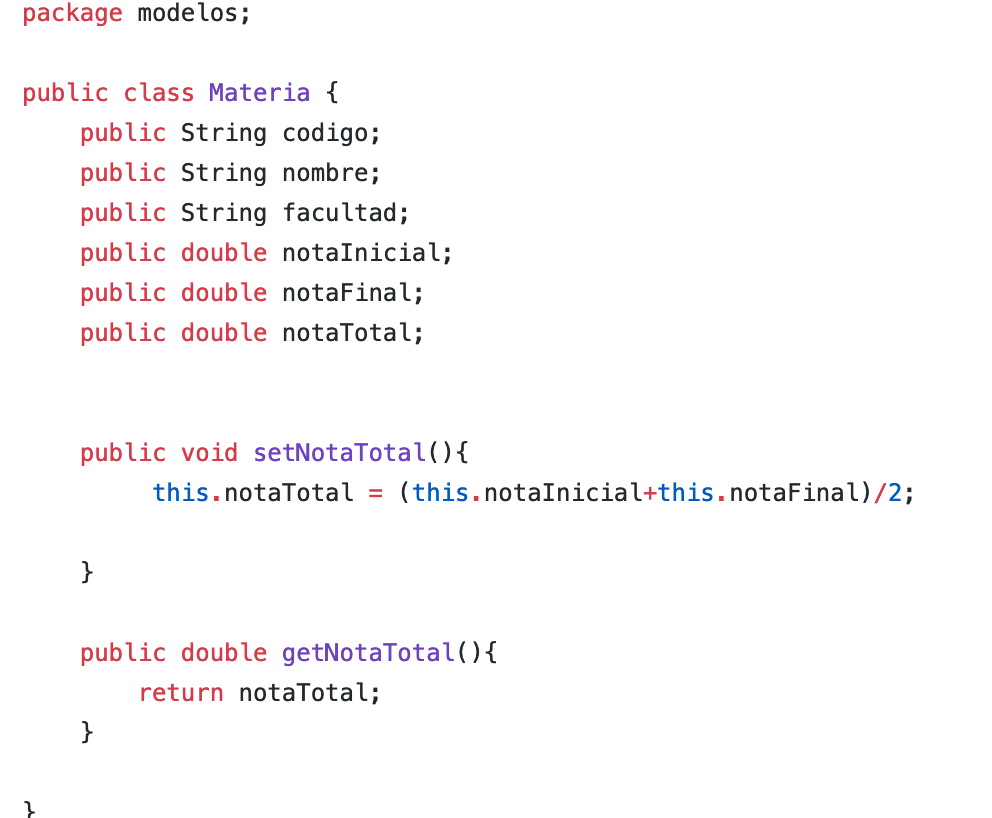
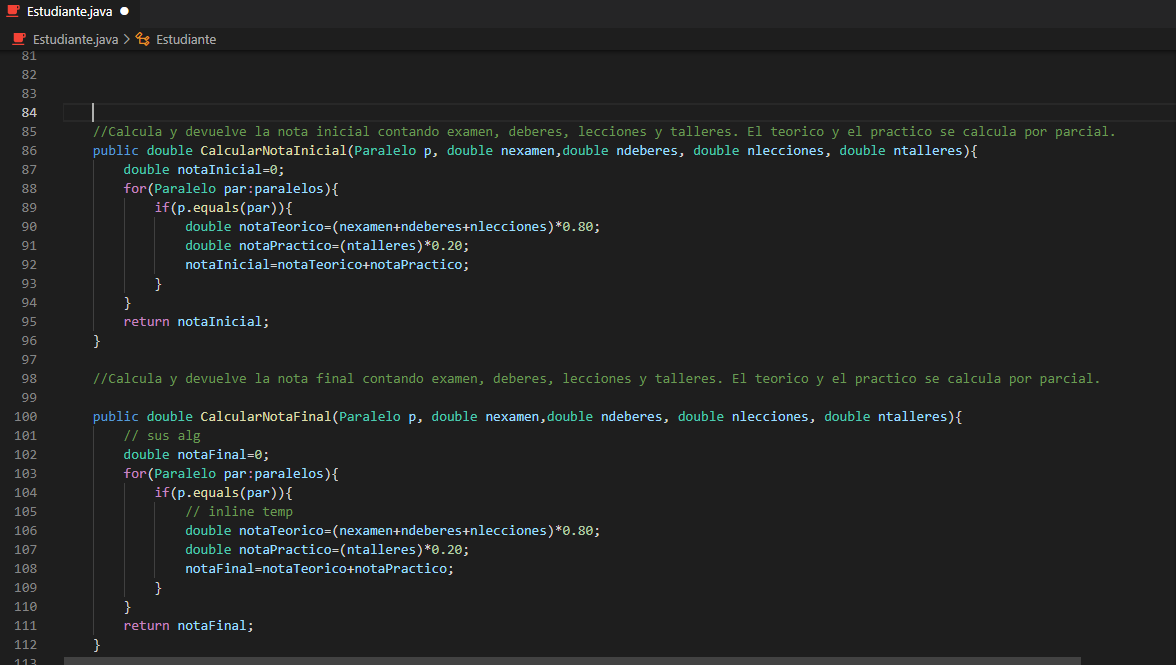


Ilustración 6 Antes de la Refactorización

## Clase: Materia

Code Smell

* Data Class

Consecuencias

* En la orientación a objetos se alienta a que las clases tengan responsabilidades que modelen su comportamiento. Además, probablemente existe un método que haga uso de atributos de esta clase y que modela mejor el comportamiento de esta que el de la otra clase.

Técnicas

* Move Method
  + Al aplicar la técnica de refactorización, el método de calcularNota ayuda a modelar el comportamiento de esta clase.

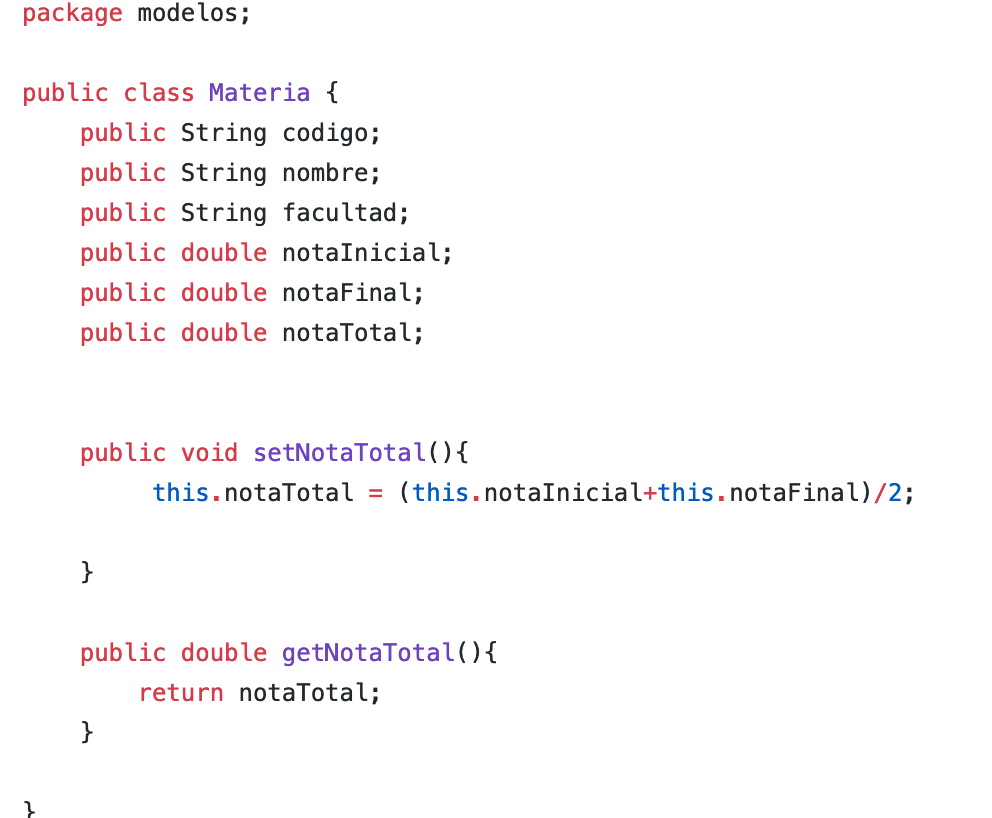
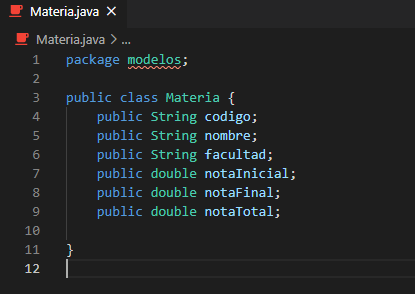


Ilustración 7Antes de la Refactorización

## Clase: Ayudante

Code Smell

* Middle man

Consecuencias

* A medida que pase el tiempo, y se decida aumentar o disminuir la información contenida en estudiante esta clase que hace de intermediaria también debe actualizar sus métodos con los cuales accede a estudiante, empleando un mayor esfuerzo y tiempo para realizar dos veces lo mismo.

Técnicas

* Extract Super Class
  + Reemplazar la composición con Estudiante por Herencia

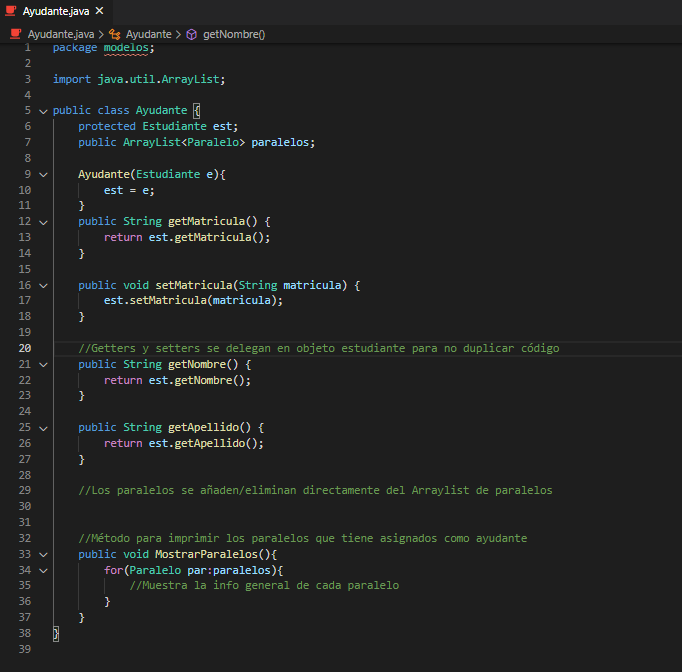


Ilustración 8Antes de la Refactorización

