

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

DISEÑO DE SOFTWARE

TALLER 8 REFACTORING

HERRAMIENTAS: EDITOR DE TEXTO, GITHUB

INTEGRANTES

NATALIA RAMIREZ YEPEZ

LIZBETH PEÑA

KATHERINE MORALES TIERRA

PARALELO 102°

PAO-2021

***Tabla de Contenido***

[**Duplicate Code** 4](#_Toc79703158)

[Consecuencia 4](#_Toc79703159)

[Solución 4](#_Toc79703160)

[Código Después 4](#_Toc79703161)

[**Data Class** 5](#_Toc79703162)

[Consecuencia 5](#_Toc79703163)

[Solución 5](#_Toc79703164)

[Código después 5](#_Toc79703165)

[**Long Parameter List** 6](#_Toc79703166)

[Consecuencia: 6](#_Toc79703167)

[Solución 6](#_Toc79703168)

[Código después 6](#_Toc79703169)

[**Data Class** 7](#_Toc79703171)

[Consecuencia 7](#_Toc79703172)

[Solución 7](#_Toc79703173)

[Código después 7](#_Toc79703174)

[**Large Class** 8](#_Toc79703175)

[Consecuencia 8](#_Toc79703176)

[Solución 9](#_Toc79703177)

[Código después 9](#_Toc79703178)

[**Comments** 10](#_Toc79703179)

[Consecuencia 10](#_Toc79703180)

[Solución 10](#_Toc79703181)

[Código Después 11](#_Toc79703182)

[**Data Clumps** 12](#_Toc79703183)

[Consecuencia 12](#_Toc79703184)

[Solución 12](#_Toc79703185)

[Código Después 12](#_Toc79703186)

[**Alternative Classes with Different Interfaces** 13](#_Toc79703187)

[Consecuencia 13](#_Toc79703188)

[Solución 13](#_Toc79703189)

[Código Después 13](#_Toc79703190)

# **Duplicate Code**

Consecuencia**:**

Los métodos CalculoNotaFinal y CalculoNotaInicial tienen código duplicado, por lo cual la clase Estudiante en donde se encuentran se convierte en una large class, además de que complica la estructura de la clase. Además, estos métodos le están otorgando responsabilidades de más a la clase.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Solución**:**

Se puede solucionar aplicando Extract Method, al aplicar esta técnica reducimos las líneas del código, además de que ahora sería mucho más fácil de comprender.

**Técnica utilizada:** Extract Method

Código Después**:**

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

# **Data Class**

Consecuencia**:**

En esta clase encontramos Data class ya que solo contiene atributos para proporcionar información a otras clases. Y no posee los get y set.

Texto, Carta

Descripción generada automáticamente

## Solución

Esto se puede solucionar aplicando EncapsulateField, ya que tiene sus campos públicos y podemos crear los getters y setters para ocultar el acceso directo. Después de aplicar la clase sigue teniendo otro code smell, Lazy Class, pero ya no es posible corregirlo con los datos que tenemos puesto que a futuro tal vez se implemente funcionalidad.

**Técnica utilizada:** Encapsulate Field

## Código después

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

# **Long Parameter List**

## Consecuencia:

Los métodos CalculoNotaFinal y CalculoNotaInicial reciben muchos parámetros los cuales pueden dificultar la implementación de los métodos y también dificultan la lectura del código.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Solución**:**

Esto se puede arreglar con Preserve Whole Object, ya que los argumentos que recibe son varios valores de un objeto. Es mucho mejor solo pasar el objeto.

**Técnica utilizada:** Preserve Whole Object.

## Código después

## Interfaz de usuario gráfica Descripción generada automáticamente con confianza baja

Imagen que contiene Texto

Descripción generada automáticamente

# **Data Class**

Consecuencia**:**

En esta clase encontramos Data class ya que solo contiene atributos para proporcionar información a otras clases. Y no posee los getters y setters.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

## Solución

Esto se puede solucionar aplicando EncapsulateField, ya que tiene sus campos públicos y podemos crear los getters y setters para ocultar el acceso directo. Después de aplicar la clase sigue teniendo otro code smell, Lazy Class, pero ya es posible corregirlo con los datos que tenemos.

**Técnica utilizada:** Encapsulate Field

## Código después

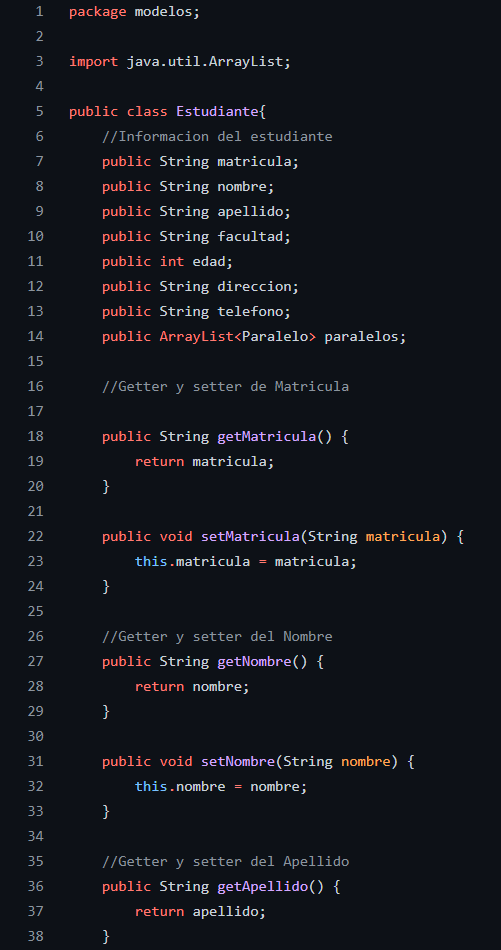
Imagen que contiene Texto

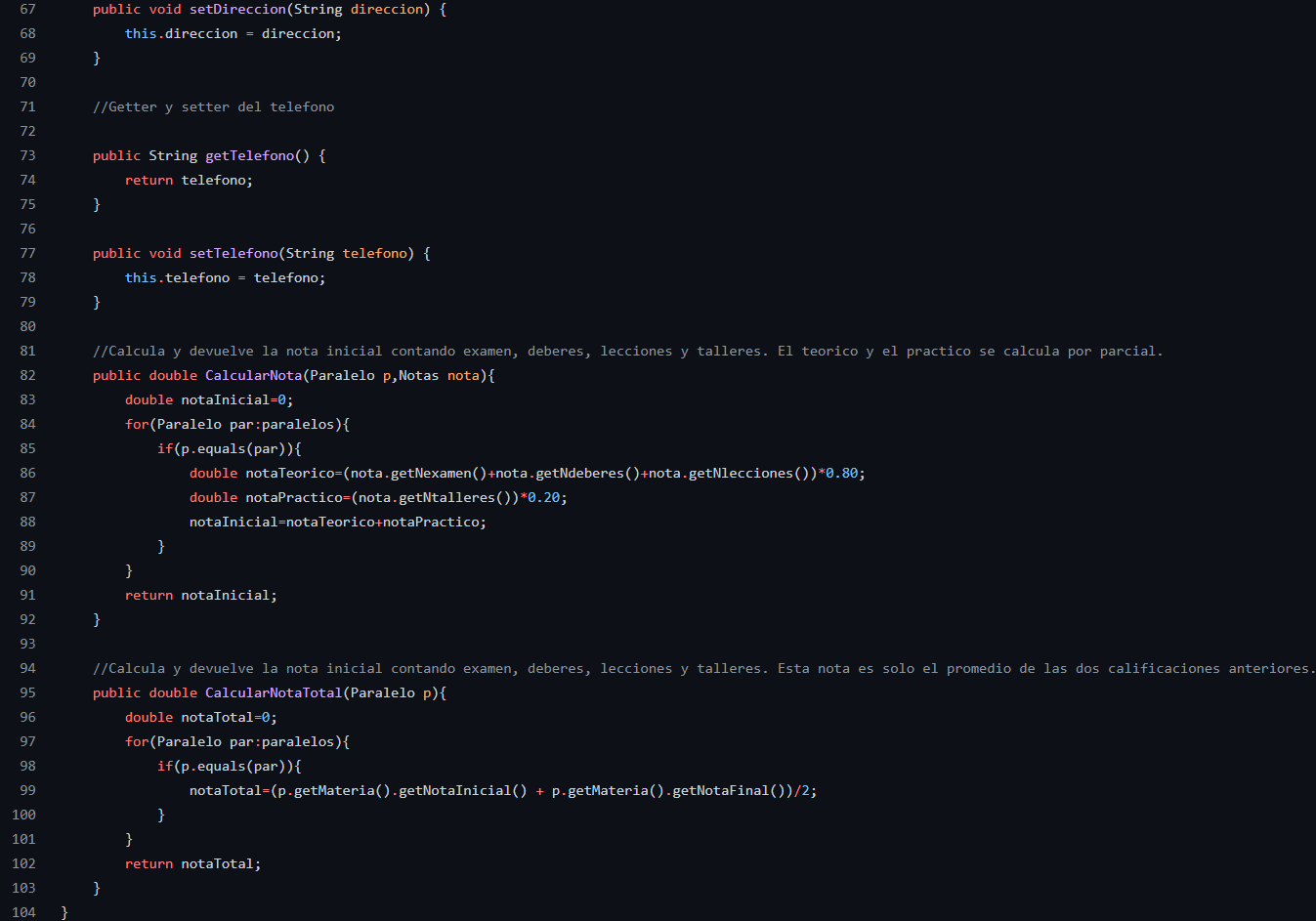
Descripción generada automáticamente

# **Large Class**

Consecuencia**:**

La clase Estudiante tiene muchas líneas de código que incluyen responsabilidades que no son parte de lo que debería realizar la clase como los métodos que calculan notas iniciales, finales y totales. Esto genera confusión para entender el código y su funcionalidad.



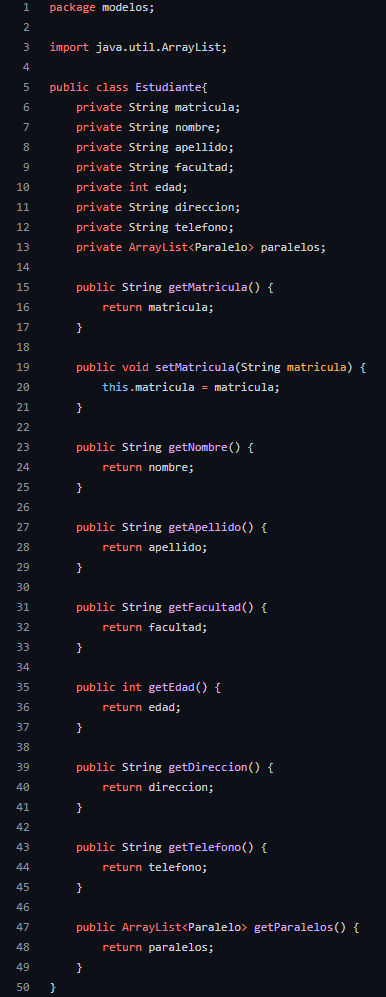


## Solución

Esto se puede solucionar aplicando Extract Class, de modo que la clase creada anteriormente, Notas, podría contener dichos métodos para tener en Estudiante un código más limpio. De aquí que también tenemos que eliminar los setters de los atributos de los estudiantes ya que se asume que la información del estudiante no debe ser modificada.

**Técnica utilizada:** Extract Class

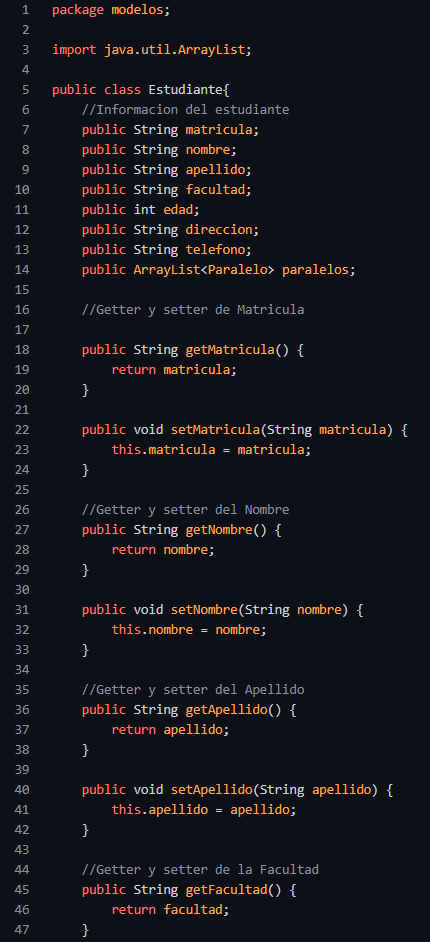
## Código después



# **Comments**

## Consecuencia

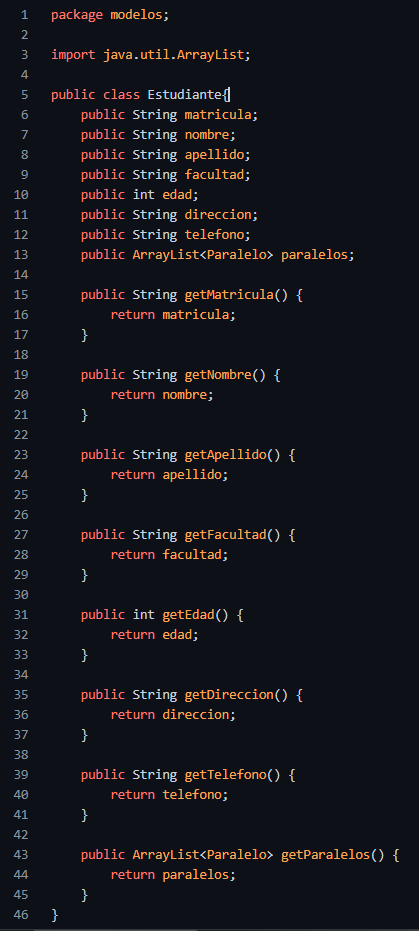
En varias partes del código se encontraron comentarios explicativos de atributos y/o métodos los cuales no son partes de la documentación necesaria. Esto hace las clases más largas y el código más abrumador y desordenado.



## Solución

La solución es eliminar estos comentarios cuando el código ya se entiende por sí mismo. Al no entenderse por sí mismo se evalúa qué otro code smell se encuentra ahí para aplicar refactoring. Esto resulta en un código mucho más limpio.

## Código Después



# **Data Clumps**

## Consecuencia

En la clase Materia se tienen los atributos notaInicial, notaFinal y notaTotal los cuales no son usados en la clase y no tienen relación con ella; sin embargo, estos atributos juntos tienen relación entre ellos. La clase Materia termina con demasiadas responsabilidades al manejar estos atributos.

Texto, Carta

Descripción generada automáticamente

## Solución

La solución para este code smell es extract class. Se crea una nueva clase llamada Notas la cual manejará el cálculo y otros asuntos de las calificaciones de los estudiantes. Esta poseerá estos atributos y se los eliminará de la clase Materia.

**Técnica utilizada:** Extract Class

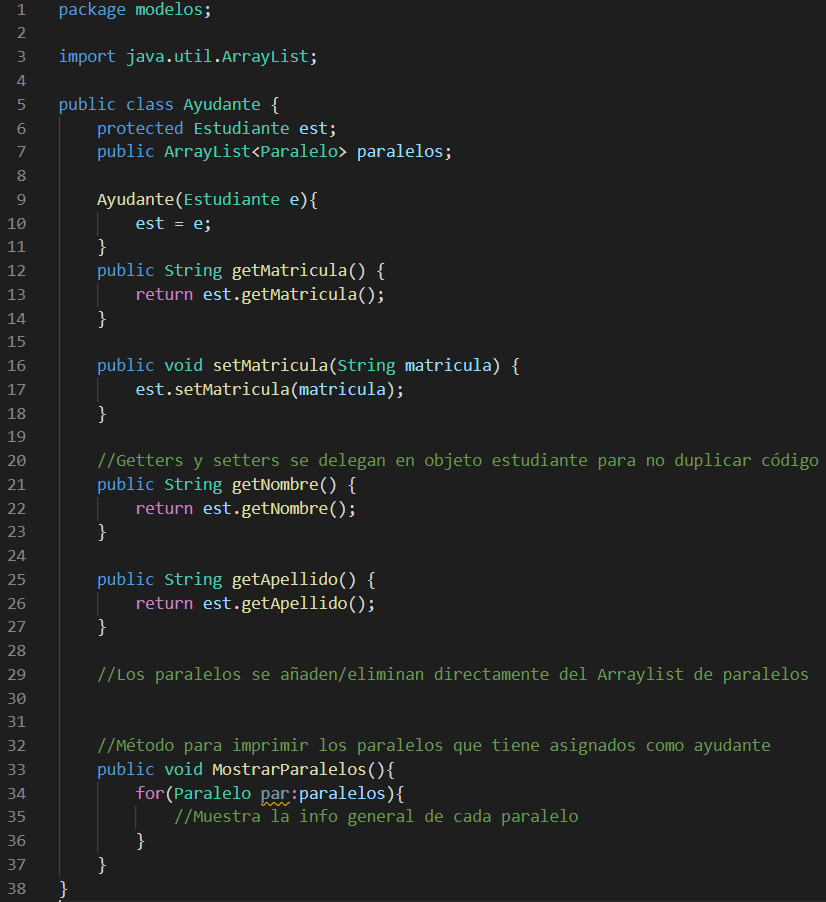
## Código Después

****

# **Alternative Classes with Different Interfaces**

## Consecuencia

La clase Estudiante y Ayudante tienen métodos y funcionalidades prácticamente idénticos con excepción de uno que pertenece a Ayudante.



## Solución

Se utilizará la técnica Extract Superclass donde se creará una clase padre para las clases Estudiante y Ayudante que tengan los atributos y métodos en común de ambas para que así se pueda heredar de la misma.

**Técnica utilizada:** Extract Superclass

## Código Después

