|  |
| --- |
| ESPOL |
| Taller Refactoring |
| Diseño de Software |

|  |
| --- |
| Genesis Elizabeth Cortez Gavilanes  Bryan Jair Segovia Mariscal |

Contenido

[Large Class 2](#_Toc92400841)

[Primitive Obsession 4](#_Toc92400842)

[Data Clumps 5](#_Toc92400843)

[Data Class 6](#_Toc92400844)

[Comments 8](#_Toc92400845)

[Innapropiate Intimacy 9](#_Toc92400846)

[Speculative Generality 10](#_Toc92400847)

[Duplicate Code 12](#_Toc92400848)

[Feature Envy 13](#_Toc92400849)

# Large Class

Las clases Estudiante y Profesor, contienen muchos atributos; esto produce que sean muy largas y tengan varias responsabilidades lo que dificulta el desarrollo o modificación del código.

**Técnicas:**

* Extract Class: Crear una superclase Persona, y extraer los atributos en común de Estudiante y Profesor.

**Código Inicial**

Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente

**Código Final**

Texto

Descripción generada automáticamente Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Por otro lado, también se extraen los métodos de calcularNota y se los agrega en una clase usando la técnica Extract Method.

**Código Inicial:**

Texto

Descripción generada automáticamente

**Código Final:**

Texto

Descripción generada automáticamente

# Primitive Obsession

La clase estudiantes, tanto como la clase Profesor, tiene todos sus atributos de tipo primitivo. Sin embargo, muchos de estos siguen un comportamiento, que pueden ser encapsulados en un tipo especifico. El uso de los tipos primitivos hace nuestro código inestable y en este caso traería consecuencias al momento de modificarlo ya que, por ejemplo, la variable dirección, puede contener otros atributos como ciudad o calle.

**Técnica:**

* [Replace Data Value with Object](https://refactoring.guru/es/replace-data-value-with-object).

**Código Inicial:**

Texto

Descripción generada automáticamente

**Código Final:**

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente Texto

Descripción generada automáticamente

# Data Clumps

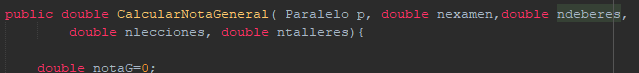
La clase Materia tiene grupos de variables parecidos, además de inutilizados, que pueden estar en su propia clase. Así como, parte del conjunto de parámetros de los métodos *calcularNota*. Tener estos grupos de parámetros y variables, afecta la organización de nuestro código, aumenta su tamaño o da más responsabilidades a la clase.

**Técnicas:**

* [Introduce Parameter Object](https://refactoring.guru/es/introduce-parameter-object).
* Extract Class.

**Código Inicial:**

Texto

Descripción generada automáticamente 

**Código Final:**

Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente

# Data Class

La clase “InformacionAdicionalProfesor” contiene atributos que perfectamente pueden pertenecer directamente a la clase “Profesor”. En esta clase no hay métodos que contengan acciones adicionales, solamente contiene campos que pertenecen a la clase “Profesor”, podemos eliminar la clase “InformacionAdicionalProfesor” y mover esos atributos a la clase “Profesor”. Además, podemos ver que en ambas clases estos atributos están públicos (no están encapsulados), lo cual no debería ser así.

**Consecuencias**

Sería necesario crear más métodos adicionales puesto que los atributos que se usarán se encuentran repartidos entre dos clases distintas, incluso corremos el riesgo de duplicar código en el programa.

**Técnicas usadas**

- Move Fields

- Encapsulate Fields

**Código Inicial**

Captura de pantalla de computadora

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

**Código Refactorizado**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

# Comments

La mayoría de las clases tienen comentarios innecesarios que buscan explicar métodos a pesar de que muchos están sobreentendidos, como de los getters y setters. Por otro lado, algunos métodos u operaciones necesitan el comentario para que sean entendidos, como *calcularNota* y *mostrarParalelos*. También hay comentarios que explican la ausencia de métodos para realizar ciertas funciones, como agregar o eliminar paralelos.

**Técnica:**

* Rename Method.
* Introduce Assertion: agregar métodos para añadir o eliminar paralelos.

**Código Inicial:**

Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente

**Código Final:**

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

# Innapropiate Intimacy

La clase “calcularSueldoProfesor” contiene un método que accede directamente a los atributos de otra clase, además de eso, podemos notar que este método no tendría que estar en esta clase, debido a que perfectamente podría estar en la clase “Profesor” y tomar de ahí los atributos, esto nos permite eliminar la clase “calcularSueldoProfesor”, puesto que está de más.

**Consecuencias**

El código se vuelve más complejo de mantener, incluso se complica el control sobre los datos de cada clase debido a que se accede directamente a ellos.

**Técnicas usadas**

- Inline Class

**Código Inicial**

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

**Código Refactorizado**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

# Speculative Generality

La clase profesor tiene muchos campos que no son utilizados. Por lo tanto, se tiene el code smell “Speculative Generality”. Si se refactoriza esta parte se tendrá un código más delgado y un soporte más sencillo.

**Consecuencias**

El código se vuelve largo y difícil de entender con tantos campos en una sola clase.

**Técnicas usadas**

Para refactorizar esto simplemente se eliminan estos campos que no se van a usar. En este caso solo se dejaran los campos de código, nombre y apellido para identificar al profesor y los años de trabajo y el bonofijo porque se usan en el método calcularSueldo().

**Código Inicial**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Código Refactorizado**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

# Duplicate Code

La clase estudiante tiene 2 métodos llamados CalcularNotaInicial y CalcularNotaFinal, los cuales tienen código duplicado.

**Consecuencias**

La duplicidad de código puede llevar a problemas a la hora de modificarlo, debido a que puede dejarse alguna duplicación sin modificar y por lo tanto introduciendo bugs que se pueden evitar.

**Técnicas usadas**

Para arreglar esto, se usó la técnica de refactorización **"Extract method"**, la cual extrae un fragmento de código en común que se puede agrupar en un solo método para facilitar la modificación de esa funcionalidad.

**Código inicial**

Texto

Descripción generada automáticamente

**Código Refactorizado**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

# Feature Envy

Los getters y setters de la clase “Ayudante” delegan sus funciones a los de la clase “Estudiante”, en otras palabras, la clase “Ayudante” accede más a atributos de otra clase que a los suyos propios, la clase que delega tiene funciones similares a la clase “Estudiante”, así que podemos reemplazar esto con una herencia.

**Consecuencias**

El código en este estado se vuelve más difícil de mantener y comprender por quien lo lea, además se puede ver que su función es las misma que la de otra clase.

**Técnicas usadas**

- Replace Delegation with Inheritance

**Código inicial**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Código Refactorizado**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente