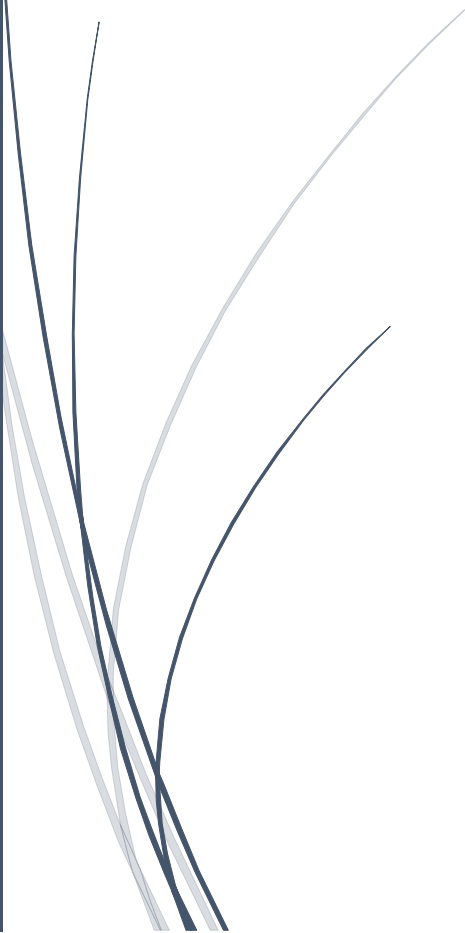


A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

13-8-2020

# Taller de Refactoring

Diseño de Software – I término  
2020-2021

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

Sebastián Mendoza, Petter De la Cruz, Edwards  
Sabando, Moisés Atupaña, Sandy Intriago  
ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

# Contenido

- 1. Code Smells encontrados .....2
- 2. Consecuencias y técnicas de refactorización .....2
- 3. Capturas de código .....7

## 1. Code Smells encontrados

- **Comments:** Clase Ayudante y clase Estudiante.
- **Duplicate Code:** Clase Estudiante.
- **Data Class:** Clase InformacionAdicionalProfesor y clase Materia.
- **Lazy Class:** Clase CalcularSueldoProfesor
- **Inappropriate Intimacy:** Clase CalcularSueldoProfesor y clase CalcularNotaTotal.
- **Long Parameter List:** Clase Profesor.
- **Data Clumps:** Clase Estudiante y clase Profesor.

## 2. Consecuencias y técnicas de refactorización

### Comment

#### **Consecuencias:**

El uso de comentarios es requerido al momento de que un código no sea intuitivo, de tal manera que no permita tener más claro la aplicabilidad de las líneas de códigos que estamos comentando, sin embargo, esto no presenta una solución eficiente cuando se está programando, porque enmascara el hecho de que podría mejorarse el código actual y que se menos complicado de entender, sumado a esto, el mantener comentarios innecesarios, ocasiona que se tengan líneas de más en el código, y al eliminarlas se tendrá como resultado un sistema más compacto y ligero.

#### **Técnicas de refactorización:**

##### ○ **Rename Method**

Este método de refactorización se aplica con la finalidad de que nuestro método sea más entendible, reemplazándolo por uno más descriptivo a la función que realiza.

#### **Clase Estudiante.**

Inicialmente dentro de la clase vemos que los nombres de cada método son lo suficientemente descriptivos para conocer la función que cumplen, por lo que se procederá a eliminar los comentarios innecesarios que se tienen para explicar cómo funcionan.

## **Clase Ayudante.**

En esta clase el método que se encarga de mostrar los paralelos tiene un comentario que describe el método, el cual es innecesario y solo aumenta la cantidad de líneas de código en caso de que fuera porque se puede malinterpretar el método lo ideal sería darle un nombre más entendible y pasar de comentarios.

### **Duplicate Code**

#### **Consecuencias:**

Este code smell se da debido a que hay dos fragmentos de código iguales, en este caso en los métodos `calcularNotaInicial(...)` y `calcularNotaFinal(...)` las cuales son métodos idénticos con una única variante en el nombre de una variable la cual no afecta el resultado de la operación realizada dentro del método. Como consecuencia, se genera un conflicto en la mantenibilidad del código, lo cual hace que sea más propenso a errores futuros y cueste más tiempo modificarlo.

#### **Técnicas de refactorización:**

- **Extract Method**

Esta técnica nos permite extraer lo esencial de los métodos similares y crear una nueva función la cual será llamada por los métodos que lo necesiten, en este caso se crea un nuevo método llamado `calcularNota(...)` la cual será llamado por `calcularNotaFinal` y `calcularNotaInicial(...)` cuando lo necesiten, logrando así simplificar la estructura de nuestro código y hacer que sea más mantenible.

### **Data Class**

#### **Consecuencias:**

Este code smell es generado porque la clase solo posee atributos los cuales además no se encuentran debidamente protegidos razón por la cual mantener este smell code puede causar problemas por datos inconsistentes.

## **Técnicas de refactorización:**

- **Encapsulate field**

Si se contienen atributos públicos, se los convierten a privados de tal manera que solo se pueda acceder a ellos por medio de getters o setters, de esta manera aumentamos el encapsulamiento de las clases.

### **Clase materia**

Actualmente la clase solo consta de atributos, a los cuales son accedidos de manera directa, por lo cual se pasarán sus atributos a privados y se implementaron los correspondientes métodos getters y setters.

### **Clase InformacionAdicionalProfesor**

Esta clase consta de solo atributos públicos que son pueden ser accedidos y modificados desde cualquier otra clase, razón por la cual se convertirán en atributos privados con sus respectivos métodos getter and setters.

## **Lazy Class**

### **Consecuencias:**

El code smell Lazy Class ocurre cuando la funcionalidad de una clase dentro del código es mínima, o bien, se encuentra totalmente sin utilidad. Puede darse el caso que dicha clase fue creada para darle soporte al código en el futuro, pero nunca sucedió.

## **Técnicas de refactorización:**

- **Inline Class**

Si existe una clase que no realiza casi nada, no es responsable de nada, y no tiene responsabilidades adicionales planeadas para ella, lo mejor es mover las características de esta clase a otra clase que sea adecuada para realizar dichas acciones.

### **Clase CalcularSueldoProfesor**

Esta clase solo consta de un solo método el cual devuelve un tipo de dato primitivo pero que accede a características de la clase Profesor. Por lo que, la mejor opción, es trasladar dicho método a Profesor y eliminar la clase CalcularSueldoProfesor.

## **Inappropriate Intimacy**

### **Consecuencias:**

Este code smell tiene lugar cuando una clase tiene acceso a atributos y métodos internos de otra clase. Esto es potencialmente peligroso, puesto que significa una falencia en el encapsulamiento de la información, exponiendo al sistema a posibles inconsistencias, puesto que los atributos que almacena podrían ser alterados por agentes o clases externas que no deberían tener acceso.

### **Técnicas de refactorización:**

- **Move Method**

Si un método es usado en otra clase en mayor medida que la clase donde se encuentra ese método, se debe crear un nuevo método igual al que se tiene, pero en la clase donde más se lo utilice. Luego, dicho método en su propia clase, o bien se le hace referencia al nuevo método, o es removido en su totalidad.

- **Encapsulate Field**

Si existen atributos que son de acceso público, se los establece como privados, y se generan métodos que permitan el acceso y modificación de estos, siendo estos de carácter público.

### **Clase CalcularNotaTotal**

Este método, presente en la clase Estudiante accede de manera directa a los atributos de una materia, sin emplear métodos que se lo permitan. Para resolver esto se establecieron los atributos notaInicial y notaFinal además de implementar sus getter y setter.

### **Clase CalcularSueldoProfesor**

Esta clase accede a los elementos directamente de la clase Profesor, lo cual puede ser porque esta clase solo existe para hacer un cálculo que involucra a la clase Profesor. Para resolver esto, se usa el Move Method para trasladar el método de esta clase a la clase Profesor.

## **Long Parameter List**

### **Consecuencias:**

Este code smell se debe al abuso de parámetros en los métodos, lo que los vuelve difíciles de leer y de mantener, especialmente si los nombres no son del todo indicativos.

### **Técnicas de refactorización:**

- **Introduce Parameter Object**

Este método de refactorización consiste en reemplazar un conjunto de datos que cumplían las funciones de parámetros en un método por un objeto del cual se pueden obtener estos datos, simplificando en gran medida el número de parámetros y haciendo el código más fácil de leer y mantener.

#### **Clase Profesor**

En la clase profesor se podía encontrar una gran cantidad de argumentos en el constructor, lo que representa un code smell. Esto fue solucionado con el reemplazo de varios parámetros con un objeto que permite el acceso a estos. Vale la pena destacar que este nuevo objeto nació originalmente del método Extract Class aplicado para un code smell previo.

## **Data clumps**

### **Consecuencias:**

Este code smell se da debido a que hay grupos de atributos idénticos en diferentes partes del código, lo cual genera que las clases se vuelvan más grandes y disminuya la comprensión y organización de código, además que puede dificultar las operaciones que se quieran realizar sobre estos atributos. En este caso se puede observar que las clases Profesor y Estudiante contienen grupos idénticos de atributos como el nombre, apellido, etc. Lo cual repercute en el tamaño de la clase Estudiante y asimismo en la clase profesor.

### **Técnicas de refactorización:**

- **Extract Class**

Esta técnica de refactorización se aplica al extraer el grupo de atributos de las clases Estudiante y Profesor en una nueva clase, logrando así reducir el tamaño de las clases de las cuales se separó el grupo de atributos y además ayuda a mejorar su comprensión y organización.

### 3. Capturas de código

#### Comments

#### Antes (Ayudante)

```
10
11
12 //Método para imprimir los paralelos que tiene asignados como ayudante
13 public void MostrarParalelos(){
14     for(Paralelo par:paralelos){
15         //Muestra la info general de cada paralelo
16     }
17 }
18 }
19
```

#### Después (Ayudante)

```
32
33 public void MostrarParalelosAyudante(){
34     for(Paralelo par:paralelos){
35         //Muestra la info general de cada paralelo
36     }
37 }
```

#### Antes (Estudiante)

```
public void setTelefono(String telefono) {
    this.telefono = telefono;
}

//Calcula y devuelve la nota inicial contando examen, deberes, lecciones
public double CalcularNotaInicial(Paralelo p, double nexamen, double ndeberes, double ndecciones) {
    double notaInicial=0;
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
            double notaPractico=(ntalleres)*0.20;
            notaInicial=notaTeorico+notaPractico;
        }
    }
    return notaInicial;
}

//Calcula y devuelve la nota final contando examen, deberes, lecciones
public double CalcularNotaFinal(Paralelo p, double nexamen, double ndeberes, double ndecciones) {
    double notaFinal=0;
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
            double notaPractico=(ntalleres)*0.20;
            notaFinal=notaTeorico+notaPractico;
        }
    }
    return notaFinal;
}
```

#### Después (Estudiante)

```
public double CalcularNotaInicial(Paralelo p, double nexamen, double ndeberes, double ndecciones) {
    double notaInicial=0;
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
            double notaPractico=(ntalleres)*0.20;
            notaInicial=notaTeorico+notaPractico;
        }
    }
    return notaInicial;
}

public double CalcularNotaFinal(Paralelo p, double nexamen, double ndeberes, double ndecciones) {
    double notaFinal=0;
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
            double notaPractico=(ntalleres)*0.20;
            notaFinal=notaTeorico+notaPractico;
        }
    }
    return notaFinal;
}
```



## Duplicate Code

### Antes

```
public double CalcularNotaInicial(Paralelo p, double nexamen, double nde
double notaInicial=0;
for(Paralelo par: paralelos){
    if(p.equals(par)){
        double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
        double notaPractico=(ntalleres)*0.20;
        notaInicial=notaTeorico+notaPractico;
    }
}
return notaInicial;
}

//Calcula y devuelve la nota final contando examen, deberes, lecciones

public double CalcularNotaFinal(Paralelo p, double nexamen, double ndebe
double notaFinal=0;
for(Paralelo par: paralelos){
    if(p.equals(par)){
        double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
        double notaPractico=(ntalleres)*0.20;
        notaFinal=notaTeorico+notaPractico;
    }
}
return notaFinal;
}
```

### Después

```
public double CalcularNotaInicial(Paralelo p, double nexamen, double ndebere
    return calcularNota(p, nexamen, ndeberes, nlecciones, ntalleres);

//Calcula y devuelve la nota final contando examen, deberes, lecciones y ta
public double CalcularNotaFinal(Paralelo p, double nexamen, double ndeberes,
    return calcularNota(p, nexamen, ndeberes, nlecciones, ntalleres);

public double calcularNota(Paralelo p, double nexamen, double ndeberes, doub
double nota=0;
for(Paralelo par: paralelos){
    if(p.equals(par)){
        double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
        double notaPractico=(ntalleres)*0.20;
        nota=notaTeorico+notaPractico;
    }
}
return nota;
```

## Data Class

### Antes (InformaciónAdicionalProfesor)

```
1 package modelos;
2
3 public class InformacionAdicionalProfesor {
4     public int añosdeTrabajo;
5     public String facultad;
6     public double BonoFijo;
7
8 }
```

### Después (InformaciónAdicionalProfesor)

```
1 package modelos;
2
3 public class InformacionAdicionalProfesor {
4     private int añosdeTrabajo;
5     private String facultad;
6     private double BonoFijo;
7
8     public int getAnosdeTrabajo(){
9         return añosdeTrabajo;
10    }
11    public String getFacultad(){
12        return facultad;
13    }
14    public double getBonoFijo(){
15        return BonoFijo;
16    }
17    public void setAnosdeTrabajo(int anoT){
18        añosdeTrabajo = anoT;
19    }
20    public void setFacultad(String facult){
21        facultad = facult;
22    }
23    public void setBonoFijo(double bono){
24        BonoFijo = bono;
25    }
26 }
```

### Antes (Materia)

```
public class Materia {
    public String codigo;
    public String nombre;
    public String facultad;
    public double notaInicial;
    public double notaFinal;
    public double notaTotal;
}
```

### Después (Materia)

```
public class Materia {
    private String codigo;
    private String nombre;
    private String facultad;
    private double notaInicial;
    private double notaFinal;
    private double notaTotal;

    public String getCodigo() {
        return codigo;
    }

    public void setCodigo(String codigo) {
        this.codigo = codigo;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getFacultad() {
        return facultad;
    }
}
```

## Lazy Class

### Antes

```
package modelos;

public class calcularSueldoProfesor {

    public double calcularSueldo(Profesor prof){
        double sueldo=0;
        sueldo= prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
        return sueldo;
    }
}
```

### Después

```
package modelos;

import java.util.ArrayList;

public class Profesor {
    public String codigo;
    public InformacionPersonal cedula;
    public InformacionAdicionalProfesor info;
    public ArrayList<Paralelo> paralelos;

    public Profesor(String codigo, String nombre, String apellido, String facultad, int edad, String direccion) {}

    public void anadirParalelos(Paralelo p){
        paralelos.add(p);
    }

    public double calcularSueldo(){
        double sueldo=0;
        sueldo= this.info.getAnosdeTrabajo()*600 + this.info.getBonoFijo();
        return sueldo;
    }
}
```

## Inappropriate Intimacy

### Antes (CalcularSueldoProfesor)

```
package modelos;

public class calcularSueldoProfesor {

    public double calcularSueldo(Profesor prof){
        double sueldo=0;
        sueldo= prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
        return sueldo;
    }
}
```

### Después (CalcularSueldoProfesor)

```
package modelos;

import java.util.ArrayList;

public class Profesor {
    public String codigo;
    public InformacionPersonal cedula;
    public InformacionAdicionalProfesor info;
    public ArrayList<Paralelo> paralelos;

    public Profesor(String codigo, String nombre, String apellido, String facultad, int edad, String direccion) {}

    public void anadirParalelos(Paralelo p){
        paralelos.add(p);
    }

    public double calcularSueldo(){
        double sueldo=0;
        sueldo= this.info.getAnosdeTrabajo()*600 + this.info.getBonoFijo();
        return sueldo;
    }
}
```

### Antes (Estudiante)

```
public double CalcularNotaTotal(Paralelo p){
    double notaTotal=0;
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            notaTotal=(p.getMateria().notaInicial+p.getMateria().notaFinal)/2;
        }
    }
    return notaTotal;
}
```

### Después (Estudiante)

```
public double CalcularNotaTotal(Paralelo p){
    double notaTotal=0;
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            notaTotal=(p.getMateria().getNotaInicial() + p.getMateria().getNotaFinal())/2;
        }
    }
    return notaTotal;
}
```

## Long Parameter List

### Antes (Profesor)

```
public Profesor(String codigo, String nombre, String apellido, String facultad, int edad, String direccion, String telefono) {  
    this.codigo = codigo;  
    this.nombre = nombre;  
    this.apellido = apellido;  
    this.edad = edad;  
    this.direccion = direccion;  
    this.telefono = telefono;  
    paralelos= new ArrayList<>();  
}
```

### Después (Profesor)

```
public class Profesor {  
    public String codigo;  
    public InformacionPersonal informacionPer;  
    public InformacionAdicionalProfesor info;  
    public ArrayList<Paralelo> paralelos;  
  
    public Profesor(String codigo, InformacionPersonal informacionPer) {  
        this.codigo = codigo;  
        this.informacionPer = informacionPer;  
        paralelos= new ArrayList<>();  
    }  
  
    public void anadirParalelos(Paralelo p){  
        paralelos.add(p);  
    }  
}
```

## Data clumps

### Antes (Estudiante)

```
3  import java.util.ArrayList;  
4  
5  public class Estudiante{  
6      //Informacion del estudiante  
7      public String matricula;  
8      public String nombre;  
9      public String apellido;  
10     public String facultad;  
11     public int edad;  
12     public String direccion;  
13     public String telefono;  
14     public ArrayList<Paralelo> paralelos;  
15 }
```

### Antes (Profesor)

```
3  import java.util.ArrayList;  
4  
5  public class Profesor {  
6      public String codigo;  
7      public String nombre;  
8      public String apellido;  
9      public int edad;  
10     public String direccion;  
11     public String telefono;  
12     public InformacionAdicionalProfesor info;  
13     public ArrayList<Paralelo> paralelos;  
14 }
```

## Después (Estudiante y Profesor)

```
2
3 import java.util.ArrayList;
4
5 public class Estudiante{
6     //Informacion del estudiante
7     public String matricula;
8     public String facultad;
9     public InformacionPersonal cedula;
10    public ArrayList<Paralelo> paralelos;
11
12    //Getter y setter de Matricula
13
14    public String getMatricula() {
15        return matricula;
16    }
```

```
import java.util.ArrayList;

public class Profesor {
    public String codigo;
    public InformacionPersonal cedula;
    public InformacionAdicionalProfesor info;
    public ArrayList<Paralelo> paralelos;
```

```
public class InformacionPersonal {
    private String nombre;
    private String apellido;
    private int edad;
    private String direccion;
    private String telefono;

    public InformacionPersonal(String nombre, String apellido, int edad, String direccion, String telefono) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.edad = edad;
        this.direccion = direccion;
        this.telefono = telefono;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApellido() {
        return apellido;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public String getDireccion() {
```