

# **ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

## **INGIENERÍA EN COMPUTACIÓN**

### **DISEÑO DE SOFTWARE**

#### **Taller 07: Refactoring I**

##### **Grupo #5**

##### **Integrantes:**

Fiorella Yerovi

Edison Reinoso

Jonathan García

Allison Recalde

Santiago Tumbaco

# INDICE DE CONTENIDO

<b>SECCIÓN A</b>	<b>3</b>
<b>Code Smell: Data Class</b>	<b>3</b>
Consecuencias de mantenerlo	3
Técnicas de refactorización utilizadas para eliminarlo	3
Captura inicial	3
Captura código refactorizado	3
<b>Code Smell: Lazy Class</b>	<b>4</b>
Consecuencias de mantenerlo	4
Técnicas de refactorización utilizadas para eliminarlo	4
Captura inicial	4
Captura código refactorizado	4
<b>Code Smell: Middle man</b>	<b>5</b>
Consecuencias de mantenerlo	5
Técnicas de refactorización utilizadas para eliminarlo	5
Captura inicial	5
Captura código refactorizado	5
Observaciones	6
<b>Code Smell: Dead Code</b>	<b>7</b>
Consecuencias de mantenerlo	7
Técnicas de refactorización para eliminarlo	7
Captura Inicial	7
Código refactorizado	7
<b>Code Smell: Long Parameter List</b>	<b>8</b>
Consecuencias de mantenerlo	8
Técnicas de refactorización para eliminarlo	8
Captura inicial	8
Captura de código refactorizado	8
Observación	8
<b>Code Smell: Duplicate Code</b>	<b>9</b>
Consecuencias de mantenerlo	9
Técnicas de refactorización utilizadas para eliminarlo	9
Captura inicial	9
Captura código refactorizado	9

# SECCIÓN A

## Identificación de Code Smells

### ***Code Smell: Data Class***

#### **Consecuencias de mantenerlo**

No tendría sentido tener una clase que solo tenga atributos de acceso público, además la clase no tiene funcionalidades por eso se utilizó Encapsuled Field para ocultar los datos y se añadió métodos getters y setters.

#### **Técnicas de refactorización utilizadas para eliminarlo**

Encapsuled Field

#### **Captura inicial**

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

#### **Captura código refactorizado**

```
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;  
    }  
}
```

## Code Smell: Lazy Class

### Consecuencias de mantenerlo

Mantener clases que no son del todo funcional como calcularSueldoProfesor cuesta dinero para mantener la misma y se aumenta la dificultad del código a la hora de mantenimiento. El código se hace más extenso de lo que debería por tanto y ocupa más memoria en la computadora.

### Técnicas de refactorización utilizadas para eliminarlo

Inline Class

#### Captura inicial

```
package modelos;

public class calcularSueldoProfesor {

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

#### Captura código refactorizado

Para solucionar este code smell se debe de utilizar la técnica Inline Class que nos ayuda a poder movilizar el contenido de la clase calcularSueldoProfesor a la clase InformacionAdicionalProfesor.

```
package modelos;

public class InformacionAdicionalProfesor {
    public int añosdeTrabajo;
    public String facultad;
    public double BonoFijo;

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

## Code Smell: *Middle man*

Cuando una clase delega su trabajo haciendo llamadas a otras clases. Entonces para que existe?

### Consecuencias de mantenerlo

Esta clase cuesta mantenerlo ya que solo crea métodos por cada atributo de estudiante es decir si estudiante tuviera mil atributos, ayudante implementaría estos métodos igual, pero delegando el trabajo a estudiante otra vez.

### Técnicas de refactorización utilizadas para eliminarlo

Remove middle man

#### Captura inicial

```
import java.util.ArrayList;

public class Ayudante {
    protected Estudiante est;
    public ArrayList<Paralelo> paralelos;

    Ayudante(Estududiante e) {
        est = e;
    }

    public String getMatricula() {
        return est.getMatricula();
    }

    public void setMatricula(String matricula) {
        est.setMatricula(matricula);
    }

    //Getters y setters se delegan en objeto estudiante para no duplicar código
    public String getNombre() {
        return est.getNombre();
    }

    public String getApellido() {
        return est.getApellido();
    }
}
```

#### Captura código refactorizado

```
import java.util.ArrayList;

public class Ayudante {
    protected Estudiante est;
    public ArrayList<Paralelo> paralelos;

    Ayudante(Estududiante e){
        est = e;
    }

    //retorna el estudiante para que implemente los metodos de estudiante
    public Estudiante getAyudante(){
        return est;
    }

    //Los paralelos se añaden/eliminan directamente del Arraylist de paralelos
}
```

### Observaciones

Al eliminar el resto de métodos y solo colocar este estamos obligando al usuario que implemente los métodos de estudiante para un ayudante en vez de crear por cada atributo de estudiante otro método en ayudante.

## Code Smell: Dead Code

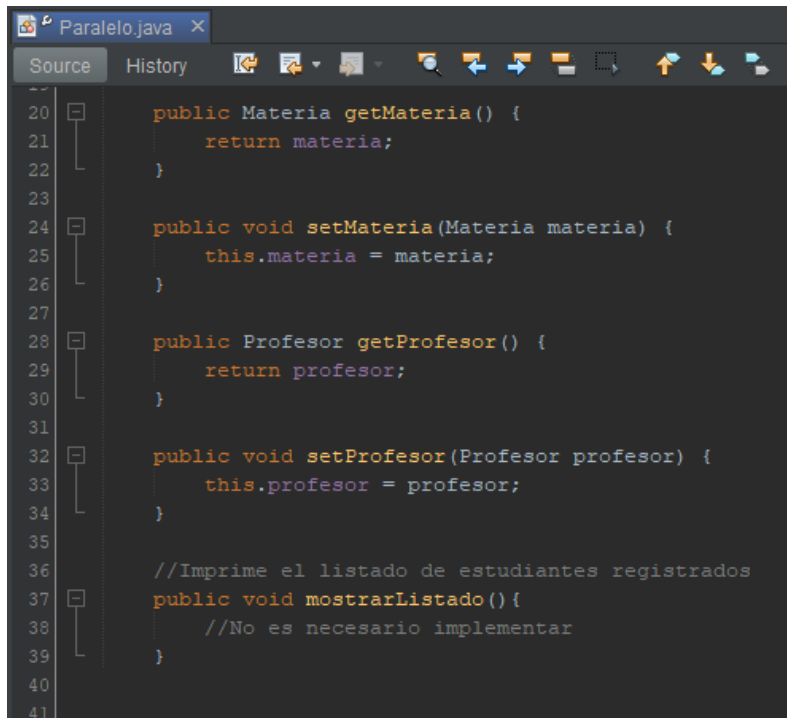
### Consecuencias de mantenerlo

Tener código sin uso implica desaprovechar el espacio de memoria en cosas que no aportan al código en general.

### Técnicas de refactorización para eliminarlo

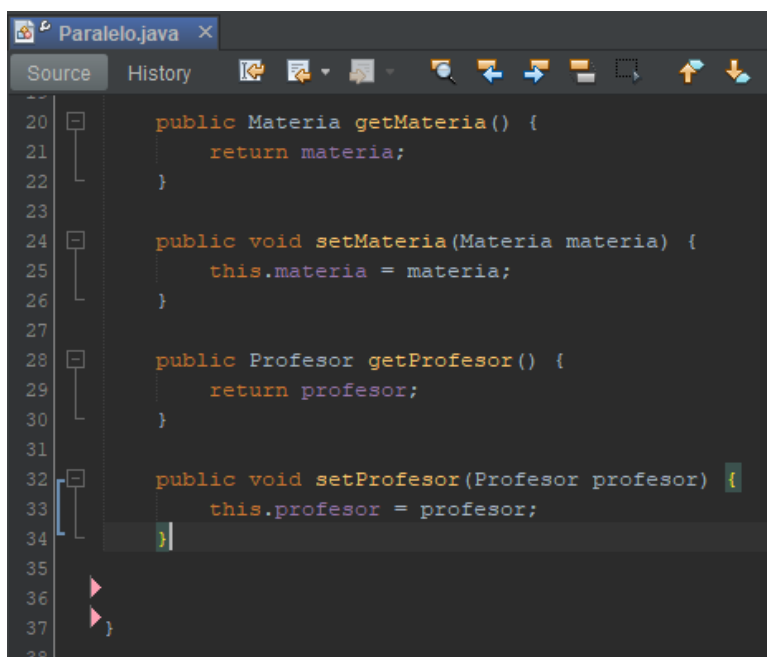
Delete unused code

### Captura Inicial



```
20 public Materia getMateria() {
21     return materia;
22 }
23
24 public void setMateria(Materia materia) {
25     this.materia = materia;
26 }
27
28 public Profesor getProfesor() {
29     return profesor;
30 }
31
32 public void setProfesor(Profesor profesor) {
33     this.profesor = profesor;
34 }
35
36 //Imprime el listado de estudiantes registrados
37 public void mostrarListado() {
38     //No es necesario implementar
39 }
40
41
```

### Código refactorizado



```
20 public Materia getMateria() {
21     return materia;
22 }
23
24 public void setMateria(Materia materia) {
25     this.materia = materia;
26 }
27
28 public Profesor getProfesor() {
29     return profesor;
30 }
31
32 public void setProfesor(Profesor profesor) {
33     this.profesor = profesor;
34 }
35
36
37
38
```

## Code Smell: Long Parameter List

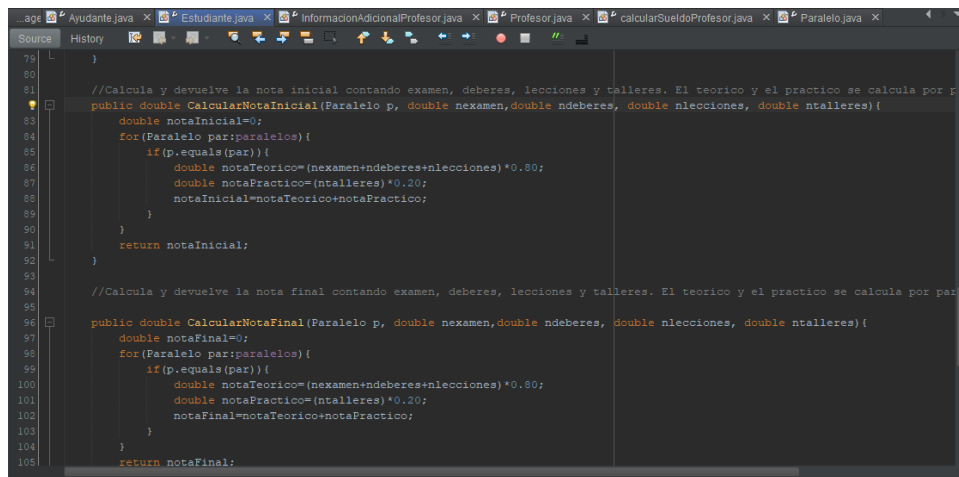
### Consecuencias de mantenerlo

A medida que se mantiene este code smell se harán más difíciles de entender y contradictorias para el lector debido a su longitud, lo que ocasionaría mayor esfuerzo para hacer cambios en ellas al estar pendiente de qué hace cada parámetro.

### Técnicas de refactorización para eliminarlo

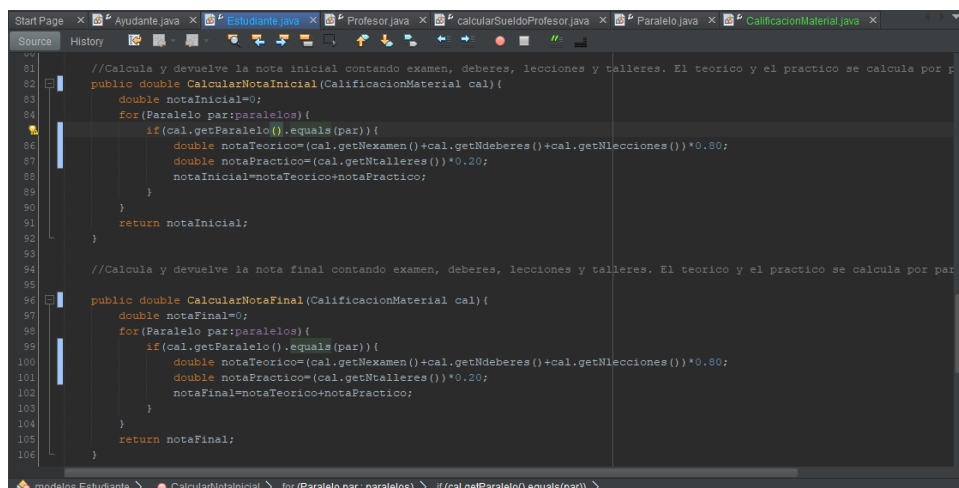
Introduce Parameter Object

#### Captura inicial



```
79 }
80
81 //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por p
82 public double CalcularNotaInicial(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres) {
83     double notaInicial=0;
84     for(Paralelo par: paralelos) {
85         if(p.equals(par)) {
86             double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
87             double notaPractico=(ntalleres)*0.20;
88             notaInicial=notaTeorico+notaPractico;
89         }
90     }
91     return notaInicial;
92 }
93
94 //Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por par
95 public double CalcularNotaFinal(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres) {
96     double notaFinal=0;
97     for(Paralelo par: paralelos) {
98         if(p.equals(par)) {
99             double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
100             double notaPractico=(ntalleres)*0.20;
101             notaFinal=notaTeorico+notaPractico;
102         }
103     }
104     return notaFinal;
105 }
```

#### Captura de código refactorizado



```
81 //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por p
82 public double CalcularNotaInicial(CalificacionMaterial cal) {
83     double notaInicial=0;
84     for(Paralelo par: paralelos) {
85         if(cal.getParalelo().equals(par)) {
86             double notaTeorico=(cal.getNexamen()+cal.getNdeberes()+cal.getNlecciones())*0.80;
87             double notaPractico=(cal.getNtalleres())*0.20;
88             notaInicial=notaTeorico+notaPractico;
89         }
90     }
91     return notaInicial;
92 }
93
94 //Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por par
95 public double CalcularNotaFinal(CalificacionMaterial cal) {
96     double notaFinal=0;
97     for(Paralelo par: paralelos) {
98         if(cal.getParalelo().equals(par)) {
99             double notaTeorico=(cal.getNexamen()+cal.getNdeberes()+cal.getNlecciones())*0.80;
100             double notaPractico=(cal.getNtalleres())*0.20;
101             notaFinal=notaTeorico+notaPractico;
102         }
103     }
104     return notaFinal;
105 }
```

#### Observación

Mediante la refactorización se puede notar similitud en el comportamiento del método que constituye otro code smell que se tratará posteriormente.



## Code Smell: Duplicate Code

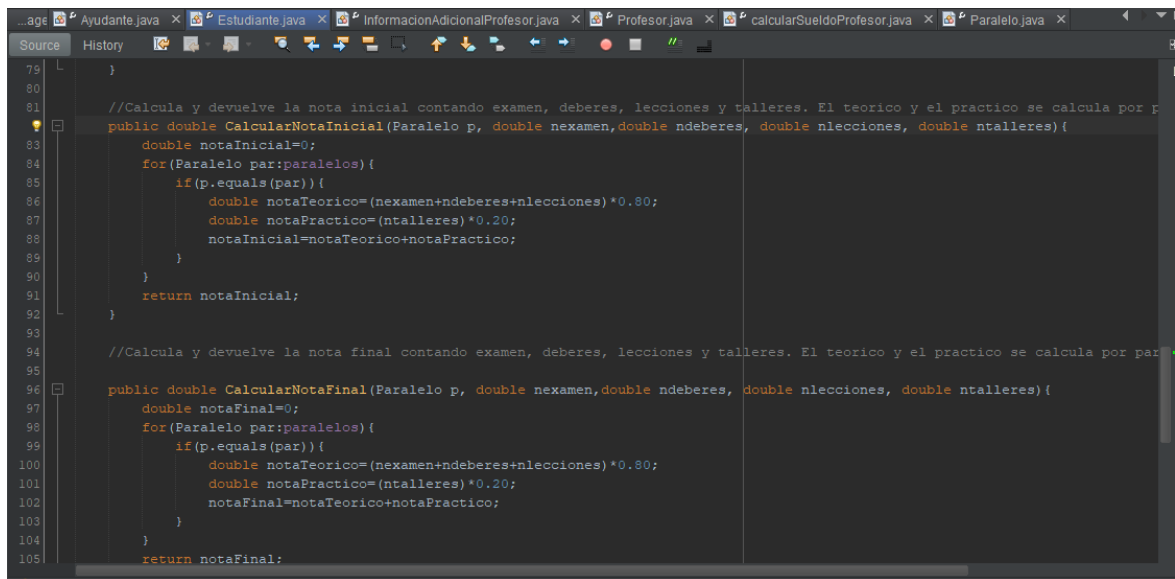
### Consecuencias de mantenerlo

Cuando se tiene duplicados en el código como en la captura que se puede observar que son lo mismo el código será más extenso y será más difícil de darle mantenimiento y por tanto más costoso.

### Técnicas de refactorización utilizadas para eliminarlo

Extract Method

#### Captura inicial



#### Captura código refactorizado

Para solucionar este code smell se elimina uno de los métodos de CalcularNota y al otro se lo edita para que se denomine CalcularNotaParcial y retorne la nota.

```
//Calcula y devuelve la nota del parcial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
public double CalcularNotaParcial(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres){
    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;
}
```