



---

# TALLER 08

---

Refactoring



**Hans Ramos Mendoza**  
**Paralelo 101**

12 DE AGOSTO DE 2021  
ESPOL

## Contenido

Sección A .....	2
<b>Clase:</b> Ayudante. ....	2
<b>Code smell:</b> Alternative Classess with different interfaces. ....	2
<b>Clase:</b> CalcularSueldoProfesor. ....	3
<b>Code smell:</b> Feature Envy.....	3
<b>Clase:</b> Materia. ....	4
<b>Code Smell:</b> Data class. ....	4
<b>Clase:</b> InformacionAdicionalProfesor.....	6
<b>Code Smell:</b> Data class .....	6
<b>Clase:</b> Estudiante.....	7
<b>Code smell:</b> Duplicate code. ....	7
<b>Clase:</b> Paralelo .....	8
<b>Code smell:</b> Dead Code .....	8

## Sección A

### Clase: Ayudante.

**Code smell:** Alternative Classess with different interfaces.

**Consecuencias:** Sucede cuando el desarrollador usa dos clases que tienen ciertas funcionalidades iguales, pero aun así permanecen separadas, o en este caso, usa la composición para arreglar el problema. El problema que surge es que mediante composición no permite usar todos los métodos de la clase compuesta.

**Técnicas usadas:** Se usa Extract Superclass para extraer la superclase que permite heredar todas las funcionalidades que necesiten sus clases hijas, en este caso la clase hija será ayudante y la padre estudiante.

**Antes.**

```
5 public class Ayudante {
6     protected Estudiante est;
7     public ArrayList<Paralelo> paralelos;
8
9     Ayudante(Estudiante e) {
10         est = e;
11     }
12     public String getMatricula() {
13         return est.getMatricula();
14     }
15
16     public void setMatricula(String matricula) {
17         est.setMatricula(matricula);
18     }
19
20     //Getters y setters se delegan en objeto estudiante para no duplicar código
21     public String getNombre() {
22         return est.getNombre();
23     }
24
25     public String getApellido() {
26         return est.getApellido();
27     }
28
29     //Los paralelos se añaden/eliminan directamente del ArrayList de paralelos
30     //Método para imprimir los paralelos que tiene asignados como ayudante
31     public void MostrarParalelos() {
32         for (Paralelo par: paralelos) {
33             //Muestra la info general de cada paralelo
34         }
35     }
36 }
```

Después.

```
1 package modelos;
2
3 import java.util.ArrayList;
4
5 public class Ayudante extends Estudiante {
6
7     private ArrayList<Paralelo> paralelosClase;
8
9     public Ayudante(ArrayList<Paralelo> paralelos){
10         this.paralelosClase= paralelos;
11     }
12
13
14     //Método para imprimir los paralelos que tiene asignados como ayudante
15     public void MostrarParalelos(){
16         for(Paralelo par:paralelos){
17             //Muestra la info general de cada paralelo
18         }
19     }
20 }
```

## Clase: CalcularSueldoProfesor.

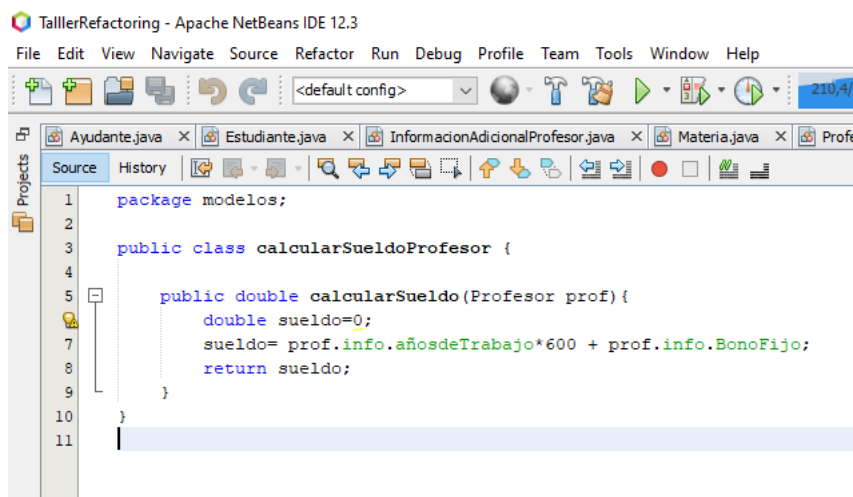
Code smell: Feature Envy.

**Consecuencias:** Sucede cuando un objeto accede a datos de otra clase en lugar de su propia clase.

Las consecuencias de seguir usando este código sin refactorizar es que cuando tengamos que llamar al método necesario para realizar la acción, habrá confusiones en cuando a cuál objeto llamar, por lo que podría resultar en obstrucciones con respecto al tiempo de programación.

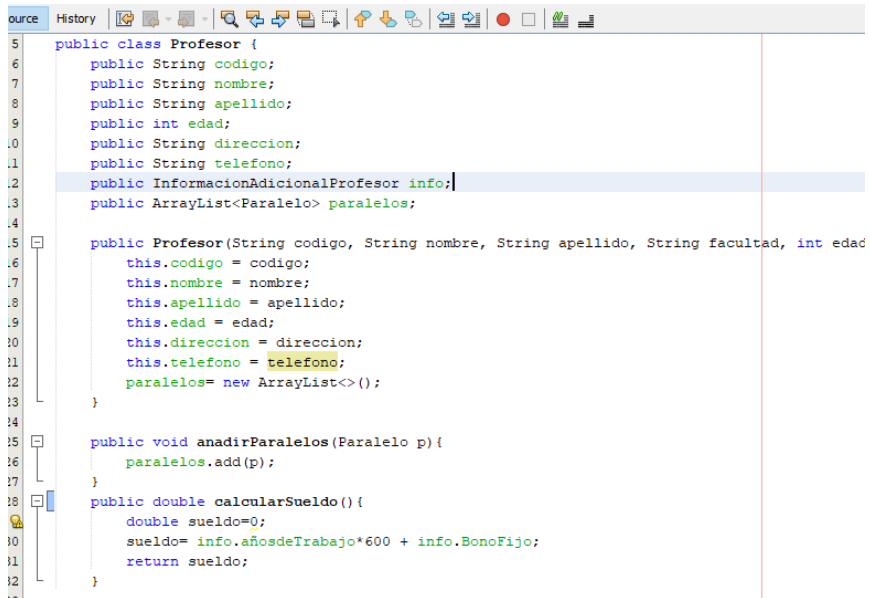
**Técnicas usadas:** Dado que en este caso la clase CalcularSueldoProfesor solo tiene un método que usa los datos del objeto profesor, usaremos **Extract method**, para enviar el método a su clase correspondiente y luego eliminaremos la clase.

Antes:



**Después:**

**//Método añadido a la clase Profesor.**



```
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
15    public Profesor(String codigo, String nombre, String apellido, String facultad, int edad) {
16        this.codigo = codigo;
17        this.nombre = nombre;
18        this.apellido = apellido;
19        this.edad = edad;
20        this.direccion = direccion;
21        this.telefono = telefono;
22        paralelos = new ArrayList<>();
23    }
24
25    public void anadirParalelos(Paralelo p) {
26        paralelos.add(p);
27    }
28    public double calcularSueldo() {
29        double sueldo = 0;
30        sueldo = info.añosdeTrabajo * 600 + info.BonoFijo;
31        return sueldo;
32    }
33 }
```

## Clase: Materia.

### Code Smell: Data class.

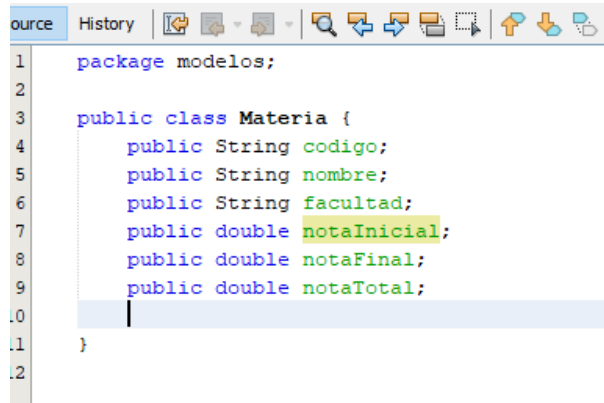
**Consecuencias:** Este smell aparece cuando se usa una clase solo para guardar atributos, los cuales no están encapsulados correctamente (se encuentran públicos) y además no contiene métodos para interactuar con dichos atributos.

La consecuencia es que cuando se quiera modificar los datos de una materia se tengan que usar métodos alojados en otra clase, lo que puede causar complicación y retrasar la escritura de código.

Además de tener código duplicado por el hecho de tener que acceder directamente a los atributos sin un método establecido para dicha acción.

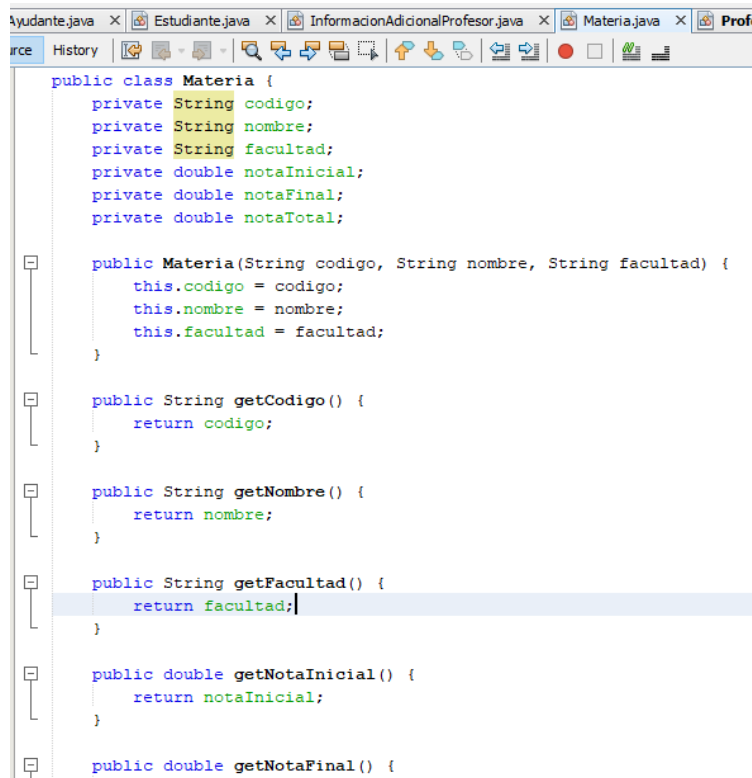
**Técnicas usadas:** **Encapsulate field** lo usamos para encapsular correctamente los atributos de manera que solo puedan ser accedidos a través de la clase a la cual pertenecen, además de establecer métodos para que se pueda acceder a sus datos solo a través de ellos.

## Antes:



```
1 package modelos;
2
3 public class Materia {
4     public String codigo;
5     public String nombre;
6     public String facultad;
7     public double notaInicial;
8     public double notaFinal;
9     public double notaTotal;
10
11 }
12
```

## Después:



```
Ayudante.java x Estudiante.java x InformacionAdicionalProfesor.java x Materia.java x Profi
ource History

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

    public Materia(String codigo, String nombre, String facultad) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.facultad = facultad;
    }

    public String getCodigo() {
        return codigo;
    }

    public String getNombre() {
        return nombre;
    }

    public String getFacultad() {
        return facultad;
    }

    public double getNotaInicial() {
        return notaInicial;
    }

    public double getNotaFinal() {

```

## Clase: InformacionAdicionalProfesor

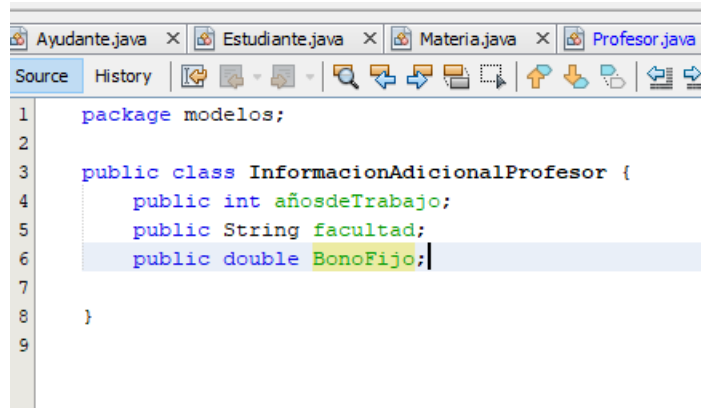
### Code Smell: Data class

**Consecuencias:** Este smell aparece cuando se usa una clase solo para guardar atributos, los cuales no están encapsulados correctamente (se encuentran públicos) y además no contiene métodos para interactuar con dichos atributos.

Además de tener código duplicado por el hecho de tener que acceder directamente a los atributos sin un método establecido para dicha acción.

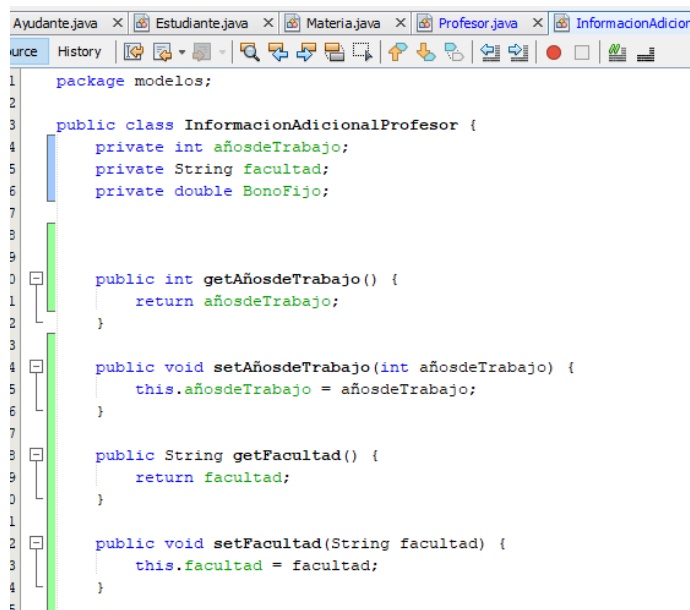
**Técnicas usadas:** Se eliminó la clase y se usó **inline class** para pasar los atributos a su clase correspondiente.

#### Antes:



```
1 package modelos;
2
3 public class InformacionAdicionalProfesor {
4     public int añosdeTrabajo;
5     public String facultad;
6     public double BonoFijo;
7
8 }
9
```

#### Después:



```
1 package modelos;
2
3 public class InformacionAdicionalProfesor {
4     private int añosdeTrabajo;
5     private String facultad;
6     private double BonoFijo;
7
8     public int getAñosdeTrabajo() {
9         return añosdeTrabajo;
10    }
11
12    public void setAñosdeTrabajo(int añosdeTrabajo) {
13        this.añosdeTrabajo = añosdeTrabajo;
14    }
15
16    public String getFacultad() {
17        return facultad;
18    }
19
20    public void setFacultad(String facultad) {
21        this.facultad = facultad;
22    }
23 }
```

## Clase: Estudiante

Code smell: Duplicate code.

**Consecuencias:** Sucede cuando existe código repetido en el software, es necesario reemplazarlo para que no surjan futuras confusiones, además de acortar la clase en cuestión.

### Técnicas usadas:

Se usa **Extract method** con un método duplicado y con el siguiente se reestablece el nombre de las variables para que sean generales y no específicos (como estaban antes de refactorizar).

### Antes:

```
//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calculan
public double CalcularNotaInicial(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres){
    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 y talleres. El teorico y el practico se calculan
public double CalcularNotaFinal(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres){
    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 void setTelefono(String telefono) {
        this.telefono = telefono;
    }

    //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por pa
    public double CalcularNota(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;
    }

    //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta nota es solo el promedio de las dos c
    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;
    }

```

## Clase: Paralelo

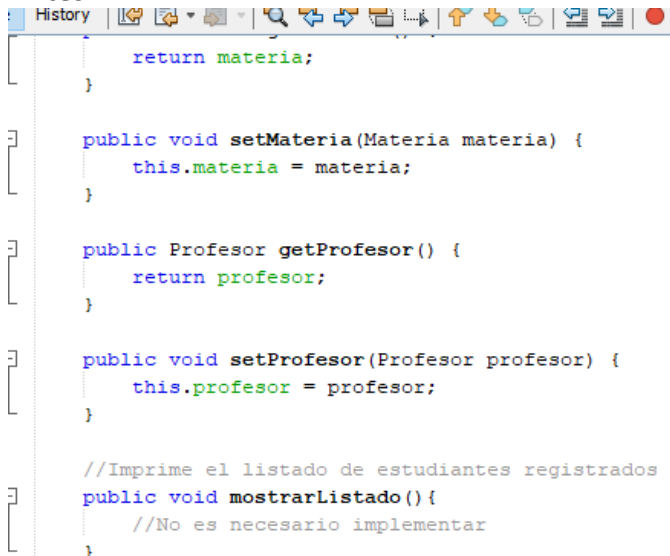
### Code smell: Dead Code

**Consecuencias:** Se obtiene cuando una parte de código no se usa, debido a una actualización o alguna otra razón.

Si no se lo actualiza, va a representar un espacio no usado dentro de las líneas de código del software.

**Técnicas usadas:** Bien se puede eliminar esa parte del código o implementarla, yo decidí implementarla.

### Antes:



```

    return materia;
}

public void setMateria(Materia materia) {
    this.materia = materia;
}

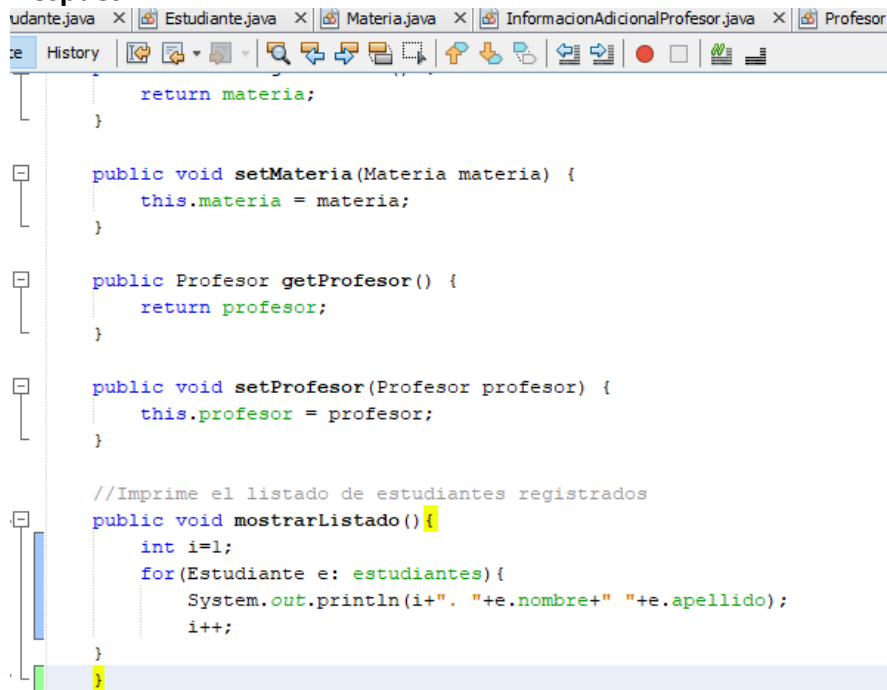
public Profesor getProfesor() {
    return profesor;
}

public void setProfesor(Profesor profesor) {
    this.profesor = profesor;
}

//Imprime el listado de estudiantes registrados
public void mostrarListado() {
    //No es necesario implementar
}

```

Después:



```
udante.java X Estudiante.java X Materia.java X InformacionAdicionalProfesor.java X Profesor
History [Icons]

    return materia;
}

public void setMateria(Materia materia) {
    this.materia = materia;
}

public Profesor getProfesor() {
    return profesor;
}

public void setProfesor(Profesor profesor) {
    this.profesor = profesor;
}

//Imprime el listado de estudiantes registrados
public void mostrarListado() {
    int i=1;
    for(Estudiante e: estudiantes){
        System.out.println(i+" "+e.nombre+" "+e.apellido);
        i++;
    }
}
```

## Sección B

<https://github.com/HansRamos99/Taller-Refactoring.git>