



INTEGRANTES:

CARLOS QUIÑONEZ ARROYO
RICARDO VILCACUNDO FLORES
HÉCTOR VILLEGAS BARRAGAN

MATERIA:

DISEÑO DE SOFTWARE

PARALELO:

102

FECHA:

13/08/2020

INDICE

Code Smell: LAZY CLASS	3
Técnica de refactorización: Inline Class	3
Code Smell: TEMPORARY FIELD	4
Técnica de refactorización: Inline Temp.	4
Code smell: LARGE CLASS	6
Técnica de refactorización: Extract class	6
Code smell: LONG PARAMETER LIST	9
Técnica de refactorización: Extract method y remove parameter	9
Code Smell: Feature envy	11
Técnica de refactorización: Replace Delegation with Inheritance.	11
Code Smell: Duplicate code.	13
Técnica de refactorización: Extract Method.	13

Code Smell: LAZY CLASS

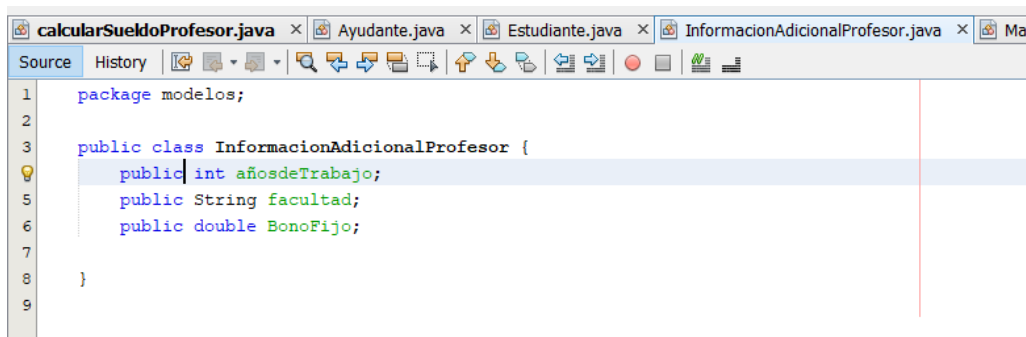
Ubicación: Lo ubicamos en la clase InformacionAdicionalProfesor.

Consecuencias: No deshacerse de este code smell puede implicar un gasto de recursos innecesario, al menos en este proyecto puede verse como ese, ya que esta clase solo cuenta con atributos y no tiene alguna funcionalidad extra.

Técnica de refactorización: Inline Class

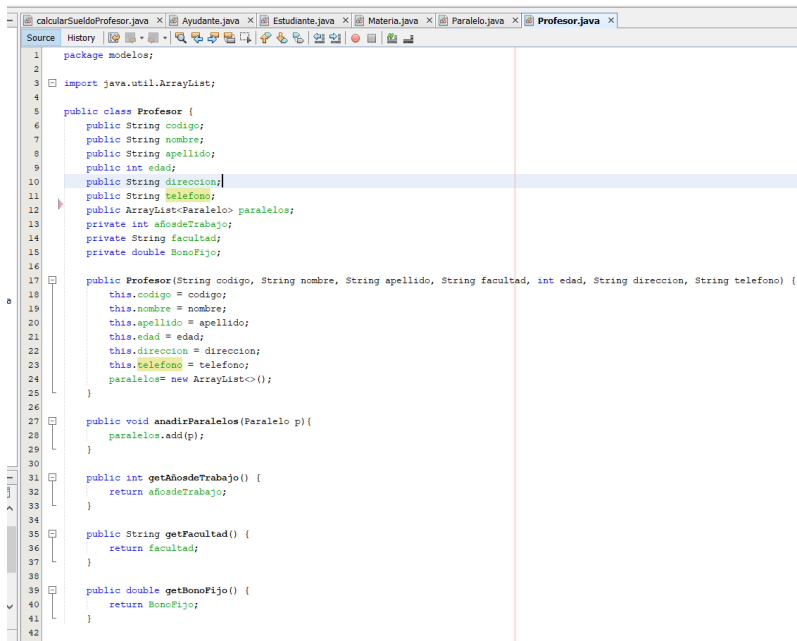
Solución: La solución a realizarse para deshacerse de este code smell radica en mover estos atributos de la clase InformacionAdicionalProfesor, ya que son valores que pueden ser atributos de la clase Profesor sin ningún problema. Solo tenemos que mover estos atributos de una clase a otra y eliminar la clase InformacionAdicionalProfesor.

ANTES:



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

DESPUES:



```
1 package modelos;
2
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 ArrayList<Paralelo> paralelos;
13    private int añosdeTrabajo;
14    private String facultad;
15    private double BonoFijo;
16
17    public Profesor(String codigo, String nombre, String apellido, String facultad, int edad, String direccion, String telefono) {
18        this.codigo = codigo;
19        this.nombre = nombre;
20        this.apellido = apellido;
21        this.edad = edad;
22        this.direccion = direccion;
23        this.telefono = telefono;
24        paralelos = new ArrayList<>();
25    }
26
27    public void anadirParalelos(Paralelo p) {
28        paralelos.add(p);
29    }
30
31    public int getAñosdeTrabajo() {
32        return añosdeTrabajo;
33    }
34
35    public String getFacultad() {
36        return facultad;
37    }
38
39    public double getBonoFijo() {
40        return BonoFijo;
41    }
42
43 }
```

Code Smell: TEMPORARY FIELD

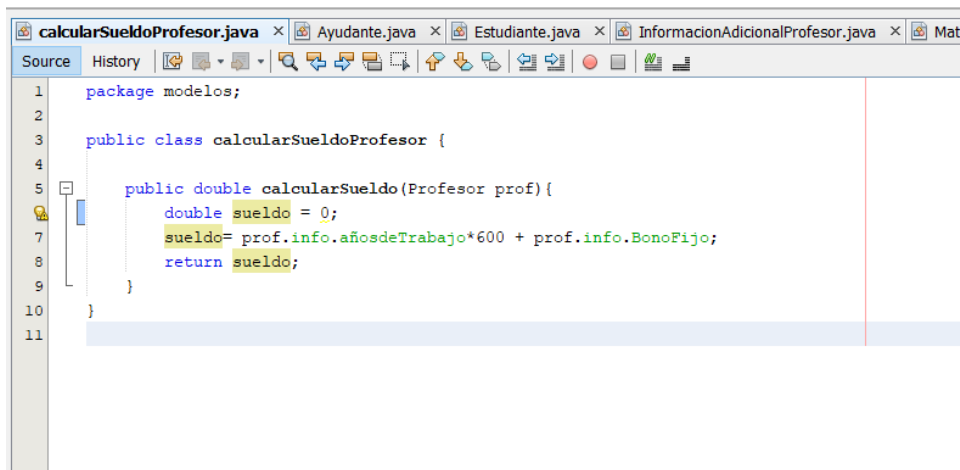
Ubicación: lo encontramos en la clase calcularSueldoProfesor.

Consecuencias: si no nos deshacemos de code smell como estos podríamos estar usando memoria sin que realmente sea necesario para el funcionamiento del programa. Instanciar variables usadas en una sola instancia podría ser un coste de recursos que se puede evitar.

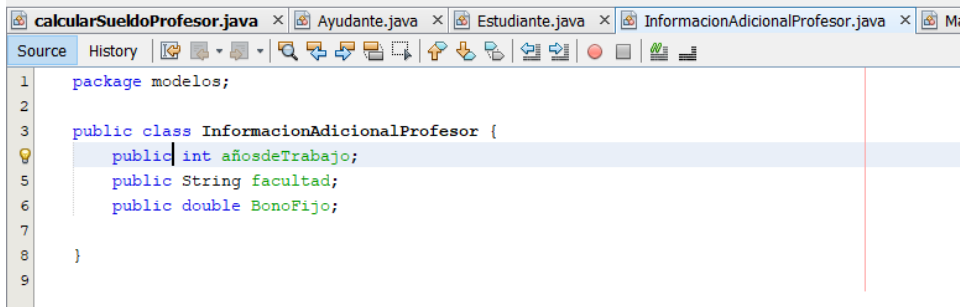
Técnica de refactorización: Inline Temp.

Solución: Lo que se hizo fue no instanciar la variable sueldo en el método, de hecho, la solución radica en un sencillo paso, el cual es retornar el valor calculado sin necesidad de guardarlo en una variable.

ANTES:

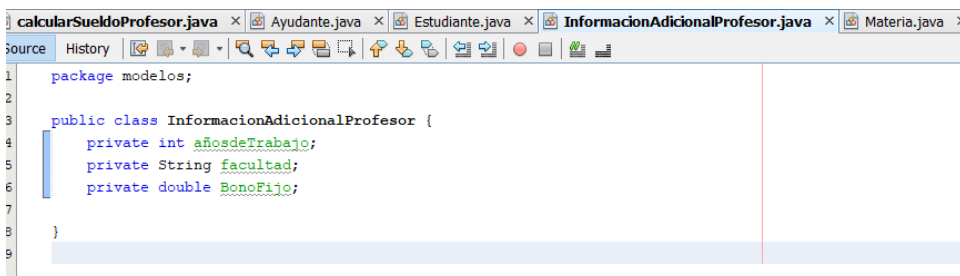


```
1 package modelos;
2
3 public class calcularSueldoProfesor {
4
5     public double calcularSueldo(Profesor prof) {
6         double sueldo = 0;
7         sueldo = prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
8         return sueldo;
9     }
10 }
11
```

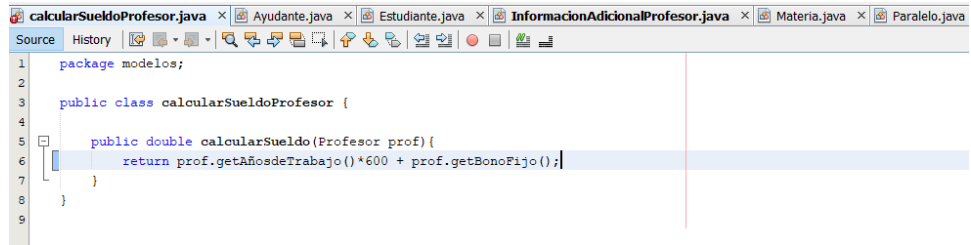


```
1 package modelos;
2
3 public class InformacionAdicionalProfesor {
4     public int añosdeTrabajo;
5     public String facultad;
6     public double BonoFijo;
7
8 }
9
```

DESPUES:



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



The screenshot shows an IDE window with several tabs: `calcularSueltoProfesor.java`, `Ayudante.java`, `Estudiante.java`, `InformacionAdicionalProfesor.java`, `Materia.java`, and `Paralelo.java`. The `calcularSueltoProfesor.java` tab is active, displaying the following code:

```
1 package modelos;
2
3 public class calcularSueltoProfesor {
4
5     public double calcularSuelto(Profesor prof) {
6         return prof.getAñosdeTrabajo() * 600 + prof.getBonoFijo();
7     }
8 }
9
```

Code smell: LARGE CLASS

Ubicación: Este code smell lo encontramos en la clase Estudiante.

Consecuencias: Dejar este code smell hace que la clase se vea muy grande, con una cantidad de datos que puede hacer confundir a la persona que intente dar mantenimiento al código, además para este caso en particular dejar los métodos que se muestran en dicha clase, hace que se incumpla uno de los principios SOLID, el de Single responsibility.

Técnica de refactorización: Extract class

Solución: Lo que se procedió hacer es usar la técnica extract class para poder separar los métodos que se encontraban en la clase Estudiante, por lo que se creó una clase nueva llamada CalcularNotas que será la que tenga la lógica de como calcular las calificaciones de los estudiantes.

ANTES:

```
public class Estudiante{
    //Informacion del estudiante
    public String matricula;
    public String nombre;
    public String apellido;
    public String facultad;
    public int edad;
    public String direccion;
    public String telefono;
    public ArrayList<Paralelo> paralelos;

    //Getter y setter de Matricula

    public String getMatricula() {
        return matricula;
    }

    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }

    //Getter y setter del Nombre
    public String getNombre() {
        return nombre;
    }

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

    //Getter y setter del Apellido
    public String getApellido() {
        return apellido;
    }

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

    //Getter y setter de la Facultad
    public String getFacultad() {
        return facultad;
    }

    public void setFacultad(String facultad) {
        this.facultad = facultad;
    }
}
```

```

//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
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 calcula por parcial.
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;
}

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

```

DESPUES:

```

public class Estudiante{
    //Informacion del estudiante
    public String matricula;
    public String nombre;
    public String apellido;
    public String facultad;
    public int edad;
    public String direccion;
    public String telefono;
    public ArrayList<Paralelo> paralelos;

    //Getter y setter de Matricula

    public String getMatricula() {
        return matricula;
    }

    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }

    //Getter y setter del Nombre
    public String getNombre() {
        return nombre;
    }

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

    //Getter y setter del Apellido
    public String getApellido() {
        return apellido;
    }

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

    //Getter y setter de la Facultad
    public String getFacultad() {
        return facultad;
    }

    public void setFacultad(String facultad) {
        this.facultad = facultad;
    }
}

```

```

public class CalcularNotas {
    Estudiante e;

    public CalcularNotas(Estudiante e) {
        this.e = e;
    }

    //Calcula y devuelve la nota inicial contando exámenes, deberes, lecciones y talleres. El teórico y el práctico se calcula por parcial.
    public double CalcularNotaInicial(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres) {
        double notaInicial = 0;
        for (Paralelo par : p.getParalelos()) {
            if (p.equals(par)) {
                double notaTeorico = (nexamen + ndeberes + nlecciones) * 0.60;
                double notaPractico = (ntalleres) * 0.20;
                notaInicial = notaTeorico + notaPractico;
            }
        }
        return notaInicial;
    }

    //Calcula y devuelve la nota final contando exámenes, deberes, lecciones y talleres. El teórico y el práctico se calcula por parcial.
    public double CalcularNotaFinal(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres) {
        double notaFinal = 0;
        for (Paralelo par : p.getParalelos()) {
            if (p.equals(par)) {
                double notaTeorico = (nexamen + ndeberes + nlecciones) * 0.60;
                double notaPractico = (ntalleres) * 0.20;
                notaFinal = notaTeorico + notaPractico;
            }
        }
        return notaFinal;
    }

    //Calcula y devuelve la nota inicial contando exámenes, deberes, lecciones y talleres. Esta nota es solo el promedio de las dos calificaciones anteriores.
    public double CalcularNotaTotal(Paralelo p) {
        double notaTotal = 0;
        for (Paralelo par : p.getParalelos()) {
            if (p.equals(par)) {
                notaTotal = (p.getMateria().notaInicial + p.getMateria().notaFinal) / 2;
            }
        }
        return notaTotal;
    }
}

```


Code smell: LONG PARAMETER LIST

Ubicación: En la clase Profesor encontramos este code smell.

Consecuencias: Dejar este code smell representa un problema a futuro, ya que se considera que la lista de parámetros es muy grande e innecesaria, para este caso particular no ha sido necesario estar creando nuevas instancias de profesor, lo cual es bueno ya que debido a la larga lista que posee su constructor sería muy complicado de hacer.

Técnica de refactorización: Extract method y remove parameter

Solución: Como se puede observar el método constructor posee una serie de parámetros que dificultan la creación de objetos de este tipo, para esto se decidió usar las técnicas de refactorización extraer método y eliminar parámetros, de esta manera lo primero que hicimos fue quitar aquellos parámetros que no son necesarios para la creación de un profesor, y en base a esos parámetros eliminados se crearon los métodos que me permiten asignar información y obtener información de este.

ANTES:

```
public class Profesor {
    public String codigo;
    public String nombre;
    public String apellido;
    public int edad;
    public String direccion;
    public String telefono;
    public InformacionAdicionalProfesor info;
    public ArrayList<Paralelo> paralelos;

    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<>();
    }

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

DESPUES:

```
public class Profesor {
    private String codigo;
    private String nombre;
    private String apellido;
    private int edad;
    private String direccion;
    private String telefono;
    private ArrayList<Paralelo> paralelos;
    private int añosdeTrabajo;
    private String facultad;
    private double BonoFijo;

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

    public int getEdad() {
        return edad;
    }

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

    public String getDireccion() {
        return direccion;
    }

    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }

    public String getTelefono() {
        return telefono;
    }

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

Code Smell: Feature envy

Ubicación: En la clase Ayudante.

Consecuencias: Este code smell genera código duplicado, aumenta la complejidad en la organización de las clases y el acoplamiento

Técnica de refactorización: [Replace Delegation with Inheritance.](#)

Solución: La clase Ayudante debe extender de Estudiante en lugar de contener un atributo de la Ayudante.

ANTES:

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

    Ayudante(Estudiante 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();
    }

    //Los paralelos se añaden/eliminan directamente del ArrayList de paralelos

    //Método para imprimir los paralelos que tiene asignados como ayudante
    public void MostrarParalelos(){
        for(Paralelo par:paralelos){
            //Muestra la info general de cada paralelo
        }
    }
}
```

DESPUES:

```
package modelos;

import java.util.ArrayList;

public class Ayudante extends Estudiante{

    private ArrayList<Paralelo> paralelosAyuda;

    public Ayudante(){
        super();
    }

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

    //Método para imprimir los paralelos que tiene asignados como ayudante
    public void MostrarParalelos(){
        for(Paralelo par:paralelos){
            //Muestra la info general de cada paralelo
        }
    }
}
```

Code Smell: Duplicate code.

Ubicación: En la clase CalcularNotas

Consecuencias: Este code smell vuelve compleja la estructura del código e innecesariamente largo, lo que dificulta su lectura.

Técnica de refactorización: Extract Method.

Solución: Crear un método que pueda encargarse de calcular la nota inicial y la final.

ANTES:

```
//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
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 calcula por parcial.
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;
}
```

DESPUES:

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