



ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

DISEÑO DE SOFTWARE

TALLER DE REFACTORIZACIÓN

GRUPO 9

PARALELO 2

INTEGRANTES:

- ANGIE ARGUDO
- ADRIANA GUILINDRO
- MANUEL LOOR

<i>Code Smell- Lazy Class.....</i>	<i>4</i>
Descripción.	4
Consecuencias.....	4
Técnica de Refactorización.....	4
Código inicial.....	4
Código refactorizado.	4
<i>Code Smell- Duplicate Code.....</i>	<i>5</i>
Descripción.	5
Consecuencias.....	5
Técnica de Refactorización.....	5
Código inicial.....	5
Código refactorizado.	5
<i>Code Smell – Data Class.</i>	<i>6</i>
Descripción.	6
Consecuencias.....	6
Técnica de refactorización.	6
Código Inicial.....	6
Código refactorizado.	6
<i>Code smell – Dead Code.</i>	<i>7</i>
Descripción.	7
Consecuencias.....	7
Técnica de refactorización.	7
Código Inicial.....	7
Código refactorizado.	7
<i>Code smell – Message Chains.....</i>	<i>8</i>
Descripción.	8
Consecuencias.....	8
Técnica de refactorización.	8
Código inicial.....	8
Código refactorizado.	8
<i>Code smell – Innappropriate Intimacy.....</i>	<i>9</i>
Descripción.	9

Consecuencias.....	9
Técnica de refactorización.	9
Código inicial.....	9
.....	9
Código refactorizado.	10

Code Smell- Lazy Class.

Descripción.

La clase **CalcularSueldoProfesor** es una clase que no tiene ninguna funcionalidad en la implementación del sistema del académico, ya que este solo requiere de consulta de notas y asignación de profesores y ayudantes en un paralelo, por lo que el uso de esta clase no es requerido.

Consecuencias.

Al tener una clase que no tiene ninguna utilidad, esto genera un costo de mantenimiento del sistema innecesario y de tiempo que se podría utilizar para otras clases o mantener el sistema mucho más eficiente y eficaz para los usuarios que utilizan este servicio.

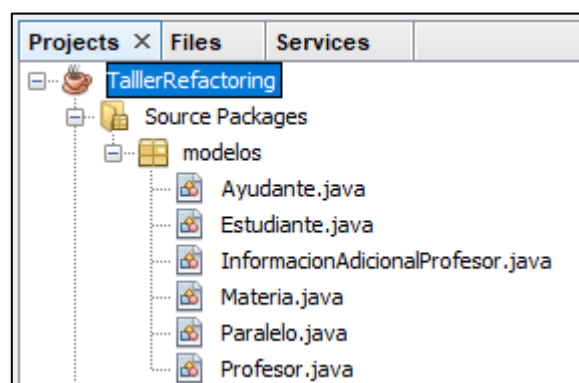
Técnica de Refactorización.

Para las clases que no son utilizadas en el sistema, la forma más sencilla de arreglar este code smell es eliminando la clase, en este caso **CalcularSueldoProfesor**.

Código inicial.



Código refactorizado.



Code Smell- Duplicate Code.

Descripción.

La clase Estudiante posee dos métodos llamados **CalcularNotaInicial()** y **CalcularNotaFinal()** que reciben los mismos parámetros y tienen la misma implementación para calcular la nota del estudiante.

Consecuencias.

Tener métodos con código repetido entorpece la lectura del código, volviéndolo más largo de lo que debería y también podría acarrear problemas a los usuarios que pueden llegar a confundir si existe alguna diferencia entre el cálculo de ambas notas.

Técnica de Refactorización.

Para la refactorización y corregir este code smell, se puede crear un método general llamado **CalcularNota()** el cual el usuario puede utilizar independientemente de qué nota esté calculando, si es una nota teórica con los trabajos que tiene o la final con todos los datos que posee.

Código inicial.

```
//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;
}
```

Código refactorizado.

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

Code Smell – Data Class.

Descripción.

La clase **Materia** emite un code smell de tipo **data class** debido a que el único motivo para acceder a ella es para utilizar sus campos.

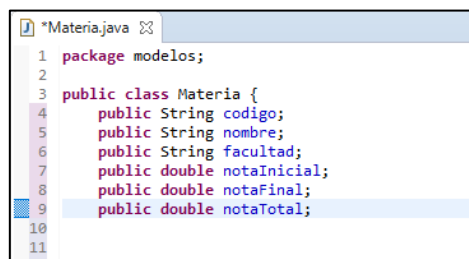
Consecuencias.

Mantener esa clase provocará que sea más difícil la organización y comprensión del código, ya que puede existir algún método que requiera de los campos de esa clase. Adicionalmente, será un obstáculo al momento de detectar la duplicación de código relacionado con la clase **Profesor**.

Técnica de refactorización.

La forma más sencilla para corregir el code smell presente en esta clase es aplicando el método **Encapsulate Field**, el cual propone convertir los modificadores de acceso de los campos de la clase a modificadores de acceso privados y crear métodos de acceso para estos campos.

Código Inicial.



```
*Materia.java
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 }
```

Código refactorizado.



```
Materia.java
3 public class Materia {
4     private String codigo;
5     private String nombre;
6     private String facultad;
7     private double notaInicial;
8     private double notaFinal;
9     private double notaTotal;
10
11     //Getters y Setters
12     public String getCodigo() {
13         return codigo;
14     }
15     public String getNombre() {
16         return nombre;
17     }
18     public String getFacultad() {
19         return facultad;
20     }
21     public double getNotaInicial() {
22         return notaInicial;
23     }
24     public double getNotaFinal() {
25         return notaFinal;
26     }
27     public double getNotaTotal() {
28         return notaTotal;
29     }
30     public void setCodigo(String codigo) {
31         this.codigo = codigo;
32     }
33     public void setNombre(String nombre) {
34         this.nombre = nombre;
35     }
36     public void setFacultad(String facultad) {
37         this.facultad = facultad;
38     }
39     public void setNotaInicial(double notaInicial) {
40         this.notaInicial = notaInicial;
41     }
42 }
```

Code smell – Dead Code.

Descripción.

Debido a que se eliminó la clase **CalcularSueldoProfesor**, la clase **InformacionAdicionalProfesor** ya no tiene ningún motivo de existir en nuestro proyecto, por lo que la clase es redundante.

Consecuencias.

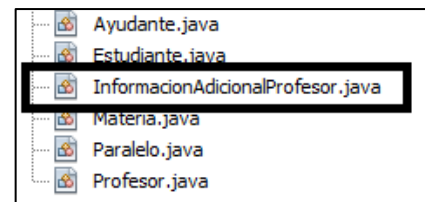
Si mantenemos esta clase, el soporte del código será complejo. Además, desaprovecharíamos la oportunidad de reducir el código.

Técnica de refactorización.

Debido a la actualización del proyecto, la cual fue eliminar la clase **CalcularSueldoProfesor**, la clase **InformacionAdicionalProfesor** es redundante, por lo que es necesario eliminar la clase. Para rescatar los campos de esta clase podemos utilizar el método **Inline Class** y ubicarlos en la clase **Profesor**.

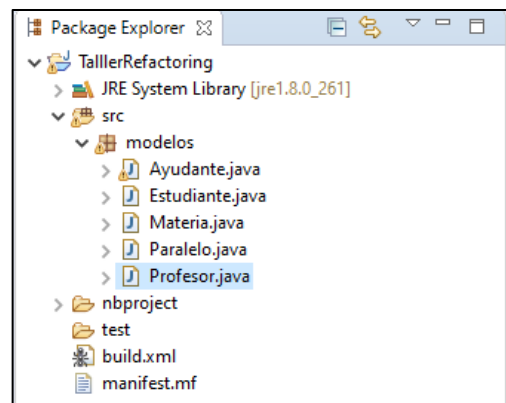
Código Inicial.

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



Código refactorizado.

```
Profesor.java
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    public String facultad;
14    public int aniosdeTrabajo;
15    public double BonoFijo;
16
17    public Profesor(String codigo, String nombre, String apellido, String facultad,
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    {
29        paralelos.add(p);
30    }
31
32 }
33
34 }
```



Una consecuencia de haber utilizado el método **Inline Class** es la modificación del método **CalcularNotaTotal** de la clase **Estudiante**:

```
Estudiante.java
95
96 //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta r
97 public double CalcularNotaTotal(Paralelo p){
98     double notaTotal=0;
99     for(Paralelo par:paralelos){
100         if(p.equals(par)){
101             notaTotal = (p.getMateria().getNotaInicial()+p.getMateria().getNotaFinal())/2;
102         }
103     }
104     return notaTotal;
105 }
```

Code smell – Message Chains.

Descripción.

Dentro de la clase Estudiante, se encuentra el método **CalcularNotaTotal()** en el cual se utiliza un objeto de tipo Paralelo para obtener su materia y luego, se utiliza el objeto de Materia para obtener la nota inicial. Lo mismo se realiza para obtener la nota final.

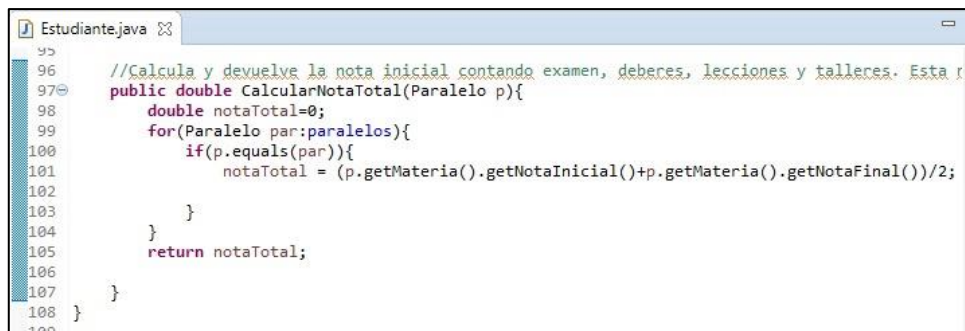
Consecuencias.

Demuestra dependencia dentro de la estructura de clases y trae complicaciones en futuras modificaciones, lo que hace que el código no sea eficiente.

Técnica de refactorización.

Utilizaremos el método **Hide delegate** creando dos métodos nuevos dentro de la clase Paralelo para obtener la nota inicial y final en dicha materia y de esta forma no se forme una cadena.

Código inicial.



```
95
96 //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta r
97 public double CalcularNotaTotal(Paralelo p){
98     double notaTotal=0;
99     for(Paralelo par:paralelos){
100         if(p.equals(par)){
101             notaTotal = (p.getMateria().getNotaInicial()+p.getMateria().getNotaFinal())/2;
102         }
103     }
104     return notaTotal;
105 }
106
107 }
108
109 }
```

Código refactorizado.

```
public class Paralelo {
    public int numero;
    public Materia materia;
    public Profesor profesor;
    public ArrayList<Estudiante> estudiantes;
    public Ayudante ayudante;

    public int getNumero() {
        return numero;
    }

    public void setNumero(int numero) {
        this.numero = numero;
    }

    public Materia getMateria() {
        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
    }

    public double getNotaInicial(){
        return materia.getNotaInicial();
    }

    public double getNotaFinal(){
        return materia.getNotaFinal();
    }
}
```

```
//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta r
public double CalcularNotaTotal(Paralelo p){
    double notaTotal=0;
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            notaTotal = (p.getNotaInicial()+p.getNotaFinal())/2;
        }
    }
    return notaTotal;
}
```


Code smell – Innappropriate Intimacy.

Descripción.

Las clases Ayudante, Estudiante, Paralelo y profesor tienen atributos declarados como públicos, lo que no restringe la relación entre las clases.

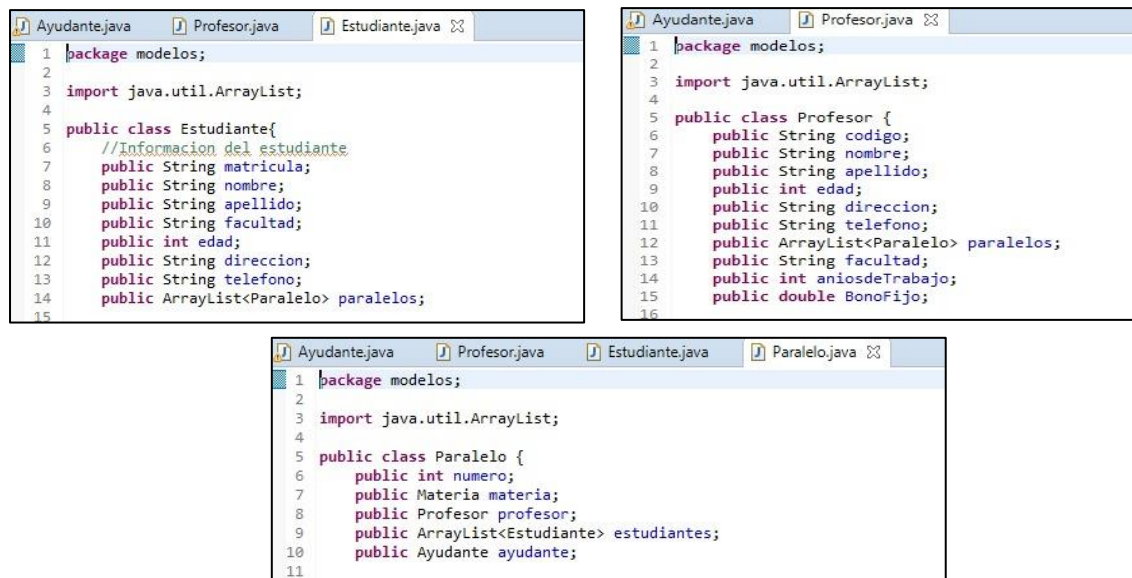
Consecuencias.

Las clases se encuentran altamente relacionadas cuando deberían conocerse lo menos posible, esto provoca que la calidad del código sea mala.

Técnica de refactorización.

Se coloca los atributos privados dentro de las clases y se utiliza el método **Change Bidirectional Association to Unidirectional** puesto que la clase Paralelo y Estudiante están relacionados mutuamente, pero en la clase Paralelo no se utiliza.

Código inicial.



```
1 package modelos;
2
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
16
17 package modelos;
18
19 import java.util.ArrayList;
20
21 public class Profesor {
22     public String codigo;
23     public String nombre;
24     public String apellido;
25     public int edad;
26     public String direccion;
27     public String telefono;
28     public ArrayList<Paralelo> paralelos;
29     public String facultad;
30     public int aniosdeTrabajo;
31     public double BonoFijo;
32
33 package modelos;
34
35 import java.util.ArrayList;
36
37 public class Paralelo {
38     public int numero;
39     public Materia materia;
40     public Profesor profesor;
41     public ArrayList<Estudiante> estudiantes;
42     public Ayudante ayudante;
43 }
```

Código refactorizado.

```
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 String facultad;
    private int aniosdeTrabajo;
    private double BonoFijo;

    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);
    }

    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 getApellido() {
        return apellido;
    }
}
```

```
public class Estudiante{
    //Informacion del estudiante
    private String matricula;
    private String nombre;
    private String apellido;
    private String facultad;
    private int edad;
    private String direccion;
    private String telefono;
    private 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;
    }
}
```

```
public class Paralelo {
    private int numero;
    private Materia materia;
    private Profesor profesor;
    private Ayudante ayudante;

    public int getNumero() {
        return numero;
    }

    public void setNumero(int numero) {
        this.numero = numero;
    }

    public Materia getMateria() {
        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
    }

    public double getNotaInicial(){
        return materia.getNotaInicial();
    }

    public double getNotaFinal(){
        return materia.getNotaFinal();
    }
}
```