

Taller 08 – Refactoring

Guido Flores

Stiven Rivera

22 de Diciembre de 2022



Contenido

1.Reporte	3
1.1. Code smells	3
1.1.1. Duplicate Code.....	3
1.1.2. Feature Envy	4
1.1.3. Middleman.....	5
1.1.4. Long Parameter list.....	6
1.1.5. Large Class	6
1.1.6. Temporary Field.....	9
1.1.7. Lazy Class	10

1.Reporte

1.1. Code smells

1.1.1. Duplicate Code

Los métodos `calcularNotaInicial()` y `calcularNotaFinal()` tienen códigos idénticos ,funcionan prácticamente igual y reciben los mismo parámetros, haciendo que sea innecesario tener los 2 métodos en la misma clase ya que ambos hacen lo mismo, con uno solo bastaría.

Consecuencias:

Tener más de un método que tiene un funcionamiento idéntico, aparte de usar memoria adicional innecesariamente, puede confundir a quienes revisen el código haciendo que se les dificulte entenderlo.

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

Refactoring/Solución:

Se eliminó uno de los métodos idénticos para evitar el código duplicado.

```
public double CalcularNota(Paralelo p, Calificaciones c){
    double nota=0;
    for(Paralelo par:e.paralelos){
        if(p.equals(par)){
            nota=CalcularNotaTeorico(c.nexamen,c.ndeberes, c.nlecciones)+CalcularNotaPractico(c.ntalleres);
        }
    }
    return nota;
}
```

1.1.2. Feature Envy

El método `calcularSueldo()` de la clase `calcularSueldoProfesor`, está usando los atributos `añosdeTrabajo` y `BonoFijo` que no pertenecen a su clase sino que pertenecen a la clase `InformacionAdicionalProfesor`, aunque no lo hace de la forma habitual (con getters y setters), siguen manteniendo este code smell presente.

Consecuencias:

Usar constantemente atributos de una clase en otra a la que no pertenecen puede hacer que la persona que esté revisando el código se confunda un poco y malinterprete el funcionamiento para la cual la clase fue diseñada.

```
1 package modelos;
2
3 //middle men
4 public class calcularSueldoProfesor {
5
6     //feature envy
7     public double calcularSueldo(Profesor prof) {
8         //temporary fields >>> inline temp
9         double sueldo=0;
10        sueldo= prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
11        return prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
12    }
13 }
```

Refactoring: Move method

Podemos mover el método `calcularSueldo()` a la clase `InformacionAdicionalProfesor` que es la clase a donde pertenecen los atributos o campos que utiliza para poder hacer que funcione adecuadamente.

```
package modelos;

public class InformacionAdicionalProfesor {

    public int añosdeTrabajo;
    public String facultad;
    public double BonoFijo;

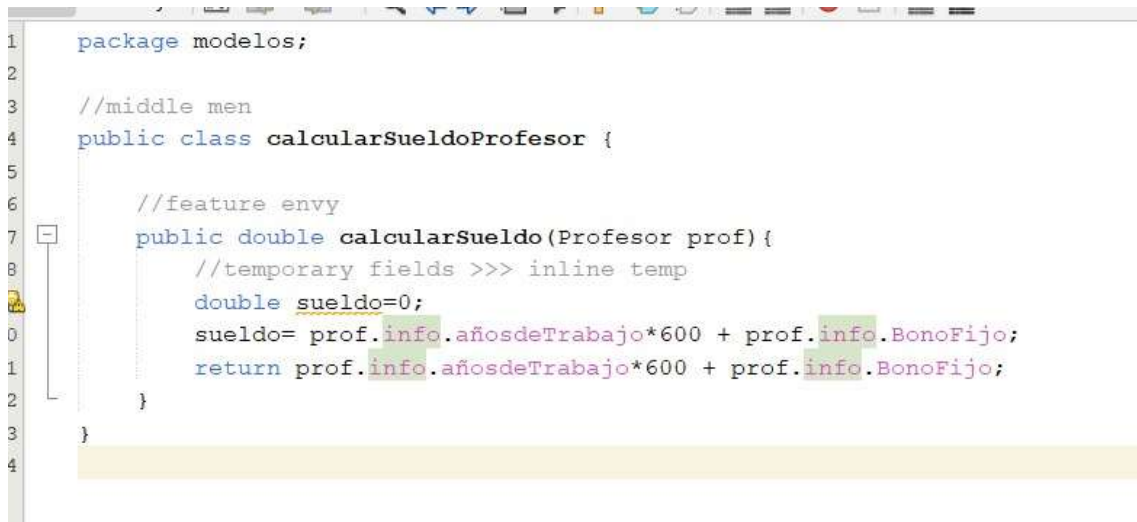
    public double calcularSueldo(Profesor prof){
        return prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
    }
}
```

1.1.3. Middleman

La clase `calcularSueldoProfesor` tiene un método que solo realiza un trabajo, delegando trabajo hacia otras clases

Consecuencias:

Mantener una clase que solo realiza un trabajo en el programa puede hacer que a futuro si ya no se necesita el funcionamiento de dicha clase, se convierta en una clase vaga haciendo que ocupe memoria innecesariamente.



```
1 package modelos;
2
3 //middle men
4 public class calcularSueldoProfesor {
5
6     //feature envy
7     public double calcularSueldo(Profesor prof){
8         //temporary fields >>> inline temp
9         double sueldo=0;
10        sueldo= prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
11        return prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
12    }
13 }
14
```

Refactoring: Move method

Podemos mover el método de la clase a otra en la que su funcionamiento sea importante, en este caso podríamos mover el método `calcularSueldo()` hacia la clase `InformacionAdicionalProfesor` ya que el funcionamiento del método destaca mejor en dicha clase

```
package modelos;

public class InformacionAdicionalProfesor {

    public int añosdeTrabajo;
    public String facultad;
    public double BonoFijo;

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

1.1.4. Long Parameter list

Los métodos `calcularNotaFinal()` y `calcularNotaInicial()` de la clase `Estudiante` reciben más de un parámetro para ejecutarse.

Consecuencias:

Si a futuro se desea hacer que los métodos implementen funcionalidades adicionales que requieran usar nuevos campos o atributos, la cantidad de parámetros que recibe se extendería aun más haciendo que entender el funcionamiento del programa se vuelva tedioso.

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

Refactoring/Solución:

Se crearon las clases `Calificaciones` que contiene los números de lecciones, examen, talleres (atributos que pedían los métodos de cálculo de notas) y la clase `Nota` que tiene los métodos para calcular notas, pero ahora en lugar de recibir varios parámetros, ahora recibe un objeto de tipo `Estudiante` y un objeto de tipo `Calificaciones` reduciendo en gran medida la cantidad de parámetros que recibe.

```
public class Nota {
    public Estudiante e;
    //duplicate code >>
    //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
    public double CalcularNotaTeorico(double nexamen, double ndeberes, double nlecciones){
        return (nexamen+ndeberes+nlecciones)*0.80;
    }
    public double CalcularNotaPractico(double ntalleres){
        return (ntalleres)*0.20;
    }
    public double CalcularNota(Paralelo p, Calificaciones c){
        double nota=0;
        for(Paralelo par: paralelos){
            if(p.equals(par)){
                nota=CalcularNotaTeorico(c.nexamen, c.ndeberes, c.nlecciones)+CalcularNotaPractico(c.ntalleres);
            }
        }
        return nota;
    }
}
```

1.1.5. Large Class

La clase Estudiante posee muchos atributos y métodos dentro de sí mismo.

Consecuencias:

Si se desea agregar aún más funcionalidades, se deberían agregar más métodos y campos haciendo que la clase se extienda demasiado hasta el punto de que mantenerla y entenderla se vuelva complicado.

```
1 package modelos;
2
3 import java.util.ArrayList;
4 //large class
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    //Getter y setter de Matricula
17
18    public String getMatricula() {
19        return matricula;
20    }
21
22    public void setMatricula(String matricula) {
23        this.matricula = matricula;
24    }
25
26    //Getter y setter del Nombre
27    public String getNombre() {
28        return nombre;
29    }
```

Refactoring: Extract Class

Dividir la clase larga en varias clases que realicen trabajos acordes a su funcionamiento y repartir atributos y métodos entre ellos, en este caso lo dividimos en la clase Estudiante que contiene los campos de los estudiantes y la clase Notas que contiene los métodos para calcular las notas de alumnos y paralelos.

```

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() {

```

```

public class Notas {
    public Estudiante e;
    //duplicate code >>
    //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
    public double calcularNotaTeorico(double examen, double deberes, double lecciones){
        return (examen+deberes+lecciones)*0.66;
    }
    public double calcularNotaPractico(double talleres){
        return (talleres)*0.33;
    }
    public double calcularNota(Paralelo p, Calificaciones c){
        double nota=0;
        for(Paralelo par:e.paralelos){
            if(p.equals(par)){
                nota=calcularNotaTeorico(c.examen,c.deberes, c.lecciones)+calcularNotaPractico(c.talleres);
            }
        }
        return nota;
    }

    //Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
    public double calcularNotaInicial(Paralelo p, Calificaciones c){
        return calcularNota(p, c);
    }

    public double calcularNotaFinal(Paralelo p, Calificaciones c){
        return calcularNota(p, c);
    }

    //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:e.paralelos){
            if(p.equals(par)){

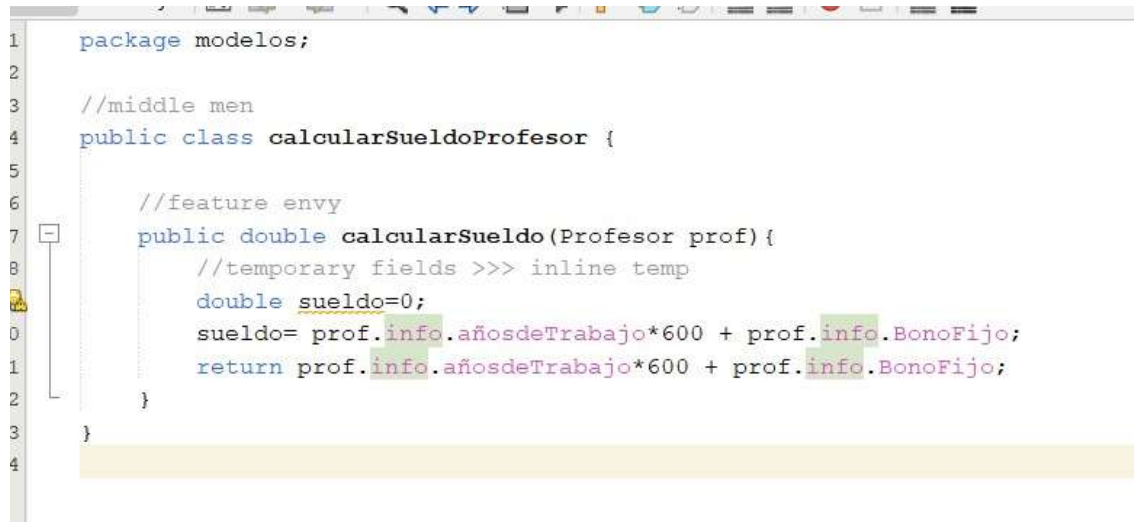
```


1.1.6. Temporary Field

En la función `calcularSueldoProfesor()`, encontramos la variable `sueldo` que usa memoria en vano y es una Temporary Field.

Consecuencias:

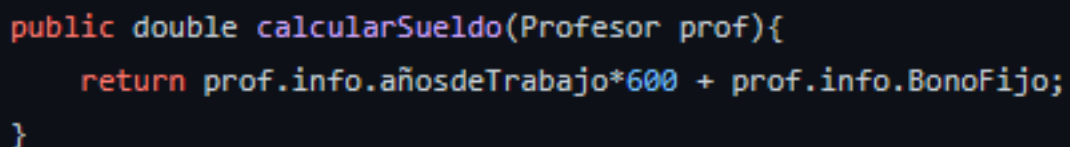
La variable `sueldo` es un campo creado que casi no se usará y además pasara mucho tiempo vacía, haciendo un mal uso de la memoria, es por esto, que lo mejor sería eliminarla.



```
1 package modelos;
2
3 //middle men
4 public class calcularSueldoProfesor {
5
6     //feature envy
7     public double calcularSueldo(Profesor prof){
8         //temporary fields >>> inline temp
9         double sueldo=0;
10        sueldo= prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
11        return prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
12    }
13 }
14
```

Refactoring: Inline temp

Podemos usar inline temp, y así borrar las líneas de código donde se crea la variable y donde se la inicializa, en lugar de eso, podemos poner directamente el valor que se le quiere asignar a esa variable.



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

1.1.7. Lazy Class

La clase `InformacionAdicionalProfesor` solo tiene atributos y no realiza ninguna acción relevante.

Consecuencias:

Conservar esta clase hace que esta parte del programa se encuentre en un punto muerto en el que no hace nada importante.

```
package modelos;

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

Refactoring/solución:

Podemos añadirle alguna funcionalidad para que la clase no se encuentre 'vaga', por ejemplo, un método relevante o añadirle algún método que al estar en otra clase, le delegaba más trabajo a las otras (MiddleMan), en este caso trasladamos el método **calcularSueldo** de la clase **calcularSueldoProfesor** hacia la clase **InformacionAdicionalProfesor**.

```
package modelos;

public class InformacionAdicionalProfesor {

    public int añosdeTrabajo;
    public String facultad;
    public double BonoFijo;

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