Diseño de Software

Grupo #8

Integrantes:

Génesis Daniela Baquerizo Anastacio

Aharon Cruz Menendez

George Alberto Henriquez Ronquillo

Profesor: Ing. David Jurado

Paralelo 3

ÍNDICE

Acopladores

Intimacy Inappropriate

La encapsulación de todos los atributos de la clase Profesor son públicos, de mantenerse allí, **cualquier otra clase podría acceder a sus atributos** y debido a esto se originaría un intimacy inappropriate.

Solución

Antes

```
package modelos;

import java.util.ArrayList;

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;
```

Después

```
package modelos;

import java.util.ArrayList;

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

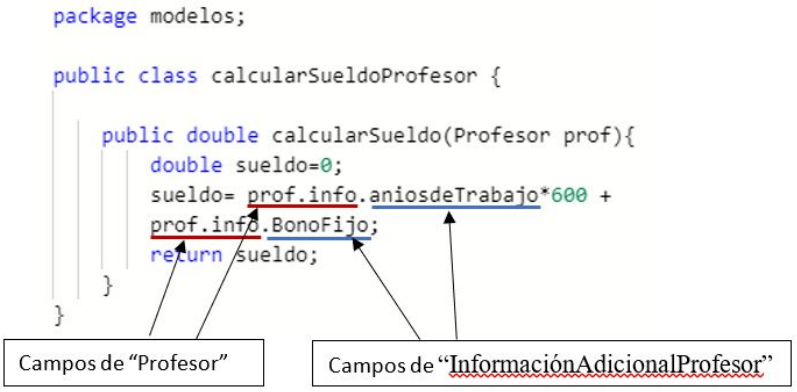
Este Code Smell tendría que resolverse cambiando el acceso de los atributos de esta clase de públicos a privados. De esta manera evitaríamos que otros usuarios que no les compete saber cierta información tengan libertad para conocerla.

Feature Envy

En la clase “calcularSueldoPorfesor” se detectó el smell Feature Envy debido a que el método “calcularSueldo” está accediendo a datos de dos clases distintas, las cuales son “Profesor” e “InformaciónAdicionalProfesor”. Como consecuencia esto conlleva a una desorganización en el código, ya que los datos que maneja el método están separados en otra clase.
Para resolver este problema, se usa la técnica de refactorización **Move Method** que consiste en mover un método a la clase que contiene la mayoría de datos que utiliza, en este caso el método calcularSueldo() es el que se traslada a la clase “InformaciónAdicionalProfesor”.

Solución

Antes



Después

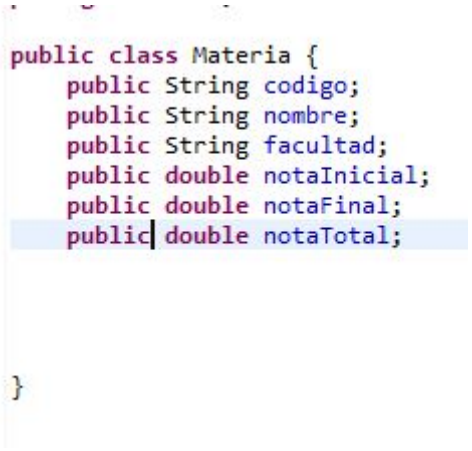
```
public class InformacionAdicionalProfesor {
    private int aniosdeTrabajo;
    private String facultad;
    private double BonoFijo;

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

Dispensables

Data class

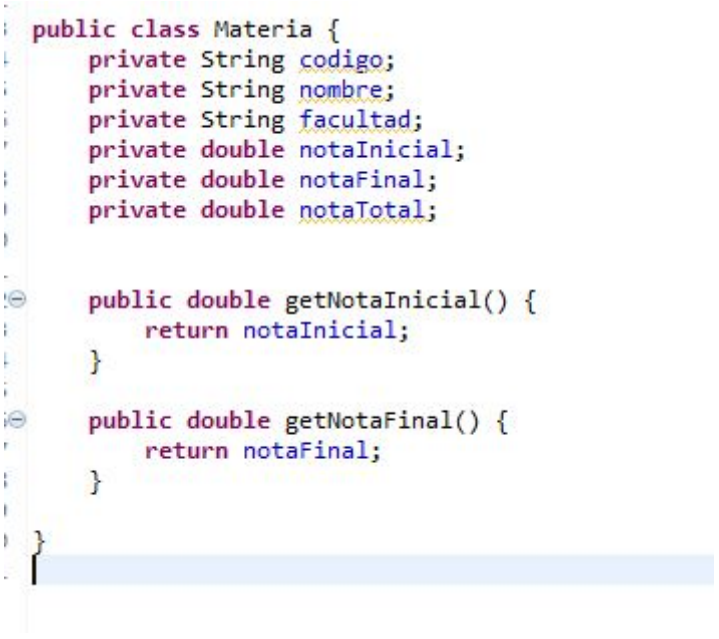
Este mal olor se presenta cuando una clase solamente tiene atributos.La técnica de refactorización a usar en este caso será **encapsulate field,** Los atributos se los cambia de públicos a privados y se le agregan los getter y setter correspondientes.

Antes



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



}
```

Después



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


    public double getNotaInicial() {
        return notaInicial;
    }

    public double getNotaFinal() {
        return notaFinal;
    }

}
```

Speculative Generality

Este mal olor se debe a que se asume que en un futuro se utilizaran ciertos métodos o atributos que en el momento no son usados por ninguna clase.

Antes

```
private String codigo;
private String nombre;
private String facultad;
private double notaInicial;
private double notaFinal;
private double notaTotal;
```

Después

```
public class Materia {

    private double notaInicial;
    private double notaFinal;


    public double getNotaInicial() {
        return notaInicial;
    }

    public double getNotaFinal() {
        return notaFinal;
    }

}
```

Código Duplicado

Esta code smell ocurre cuando tenemos dos fragmentos de nuestro código parecen casi iguales, y eso ocurre en los métodos **CalcularNotaInicial**() y **CalcularNotaFinal**(), ya que el código es básicamente el mismo, solo cambian las variables de notaInicial y notaFinal.

Para corregir este code smell debemos usar extract method para eliminar el código duplicado y así calcular en un solo método la nota de cualquier parcial cambiando el nombre del atributo local a notaParcial.

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;
}
```

Después

```
//Calcula y devuelve la nota 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 notaParcial=0;
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
            double notaPractico=(ntalleres)*0.20;
            notaParcial=notaTeorico+notaPractico;
        }
    }
    return notaParcial;
}
```

Bloaters

Long Parameter List

Este code smell se da cuando un método tiene más de tres o cuatro parámetros, lo que ocasiona que este sea más difícil de entender. Para solucionar este problema, se aplica la técnica de refactorización **Preserve Whole Object**, para esto se crea una nueva clase "Actividad" que va a almacenar los campos como "nexamen", "ndeberes", etc. Esta clase "Actividad" es la que se va a reemplazar a la lista de parámetros en el método calcular nota.

Antes

```
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;
}
```

Después

```
//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres.
public double CalcularNotaInicial(Paralelo p, Actividad a){
    double notaInicial=0;
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            double notaTeorico=(a.getNexamen()+a.getLecciones()+a.getNdeberes())*0.80;
            double notaPractico=(a.getNtalleres())*0.20;
            notaInicial=notaTeorico+notaPractico;
        }
    }
    return notaInicial;
}
```

```
public class Actividad {
    private double ndeberes;
    private double nexamen;
    private double lecciones;
    private double ntalleres;

    public double getLecciones() {
        return lecciones;
    }

    public void setLecciones(double lecciones) {
        this.lecciones = lecciones;
    }

    public double getNexamen() {
        return nexamen;
    }
    //....
```