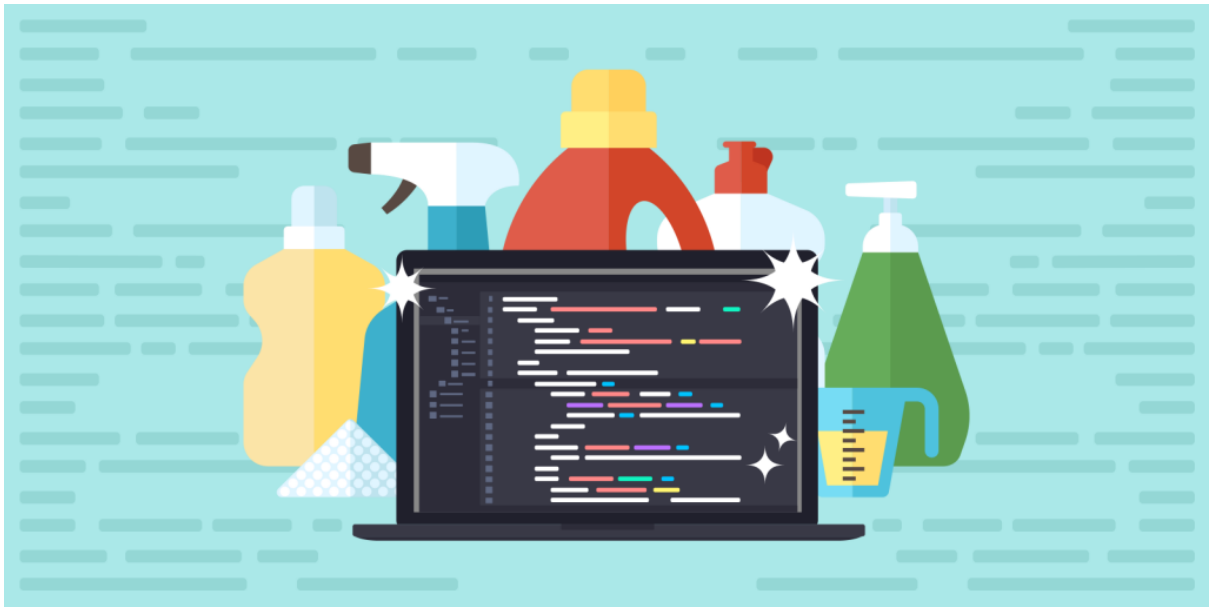
ESCUELA SUPERIOR POLITECNICA DEL LITORAL

TALLER REFACTORING

DISEÑO DE SOFTWARE



PARALELO 2

LUIS ABARCA

MIGUEL PARRA

2020-2021

INDICE

1. Temporary Field

1.1. Consecuencias

En ciertos casos, es correcto declarar una variable que contenga una expresión compleja y larga, pero en este código no aplica el caso ya que simplemente la variable “sueldo” está usando los getters de la clase Profesor. Si se mantiene asi el código, al crear la variable “sueldo” se estaría ocupando espacio en memoria innecesario para una simple expresión. Además, el código del método crece y eso hace que sea más complejo de leer.

1.2. Técnica de Refactorización

Para resolver este code smell se empleará Inline Temp, esto es, se eliminará la declaración e inicialización de la variable “sueldo” y directamente se retornará la expresión que este contenía.

1.3. Código Inicial

```
package modelos;

public class calcularSueldoProfesor {

    public double calcularSueldo(Profesor prof){
        double sueldo=0;
        sueldo= prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
        return sueldo;
    }
```

1.4. Código Final

```
package modelos;

public class calcularSueldoProfesor {

    public double calcularSueldo(Profesor prof){

        return prof.getAñosdeTrabajo()*600 + prof.getBonoFijo();

    }
}
```

2. Lazy Class

2.1. Consecuencias

Si se mantiene esta clase, el proyecto se verá muy grande. También podría generar confusiones a la hora de modificar el proyecto en un futuro.

2.2. Técnica de Refactorización

Para resolver este code smell se utilizará Inline Class, esto es, los atributos de esta clase se los añadirá a la clase Profesor y a su vez se la eliminará.
La clase InformacionAdicionalProfesor no se hace mucho, simplemente contiene 3 atributos que le pertenecen a los objetos Profesor. Además, si no se llega a eliminar, de igual forma a esta clase le faltaría un atributo de tipo Profesor, de tal manera pique se conozca a que profesor en particular le pertenecen dichos atributos, sino la clase quedaría completamente aislada de las demás.

2.3. Código Inicial

```java
package modelos;

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

}
```

2.4. Código Final

```java
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 ArrayList<Paralelo> paralelos;
    private int añosdeTrabajo;
    private String facultad;
    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);
    }
```

3. Feature Envy

3.1. Consecuencias

Se duplica el código, esto es, las clases Ayudante y Estudiante contiene el mismo código. A su vez, se puede notar que existe alto acoplamiento entre ambas clases, es decir, si se llega a eliminar código en la clase Estudiante puede que este le afecte a la operabilidad de la clase Ayudante.

3.2. Técnica de Refactorización

La técnica que se aplicará para resolver este code smell será Reemplazar Delegación con Herencia, ya que a través de la Herencia logramos que nuestro código en la clase Estudiante se reúse sin la necesidad de que haya código duplicado ,lo cual a su vez hace que la clase Ayudante sea más pequeña y fácil de mantener en un futuro.

3.3. Código Inicial

```java
package modelos;

import java.util.ArrayList;

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
```

3.4. Código Final

```java
package modelos;

import java.util.ArrayList;

public class Ayudante extends Estudiante{
    private ArrayList<Paralelo> paralelos;

    public ArrayList<Paralelo> getParalelos() {
        return paralelos;
    }

    public void setParalelos(ArrayList<Paralelo> paralelos) {
        this.paralelos = paralelos;
    }

    public void MostrarParalelos(){
        for(Paralelo par:paralelos){
            //Muestra la info general de cada paralelo
        }
    }
```

4. LARGE CLASS

4.1. Consecuencias

Este código efectúa el smell en el hecho de que hace que la clase se ve extremadamente grande, se lo puede visualizar en la clase estudiante y este método crea una confusión no solo para el usuario si no para el programador mismo, también incumple uno de los principios SOLID, el de single responsability cuando se implementan esa cantidad excesiva de gente.

4.2. Técnica de Refactorización

La técnica de refactorización que se usa en este smell es la de Extract class, se usó esa refactorización para poder separar muchos de los métodos que existen en esta clase y así hacer el código más flexible y más fácil de leer, se uso una nueva clase llamada obtenerNotas la cual nos informa de las notas que obtendra cada estudiante.

4.3. Código Inicial

```java
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;
    }

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

4.4. Código Final

```
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
public double CalcularNotaInicial(Paralelo p, double nexamen,double ndeberes, double nlecciones, double ntallere
    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 c

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 promed
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;

}
```

5. LONG PARAMETER LIST

5.1. Consecuencias

Consecuencias, una de las secuencias más importantes de tener parámetros largos es que el usuario se puede confundir con tanta información por ingresar y que también es muy poco estético haciendo que iluso de este método sea muy poco usado, también implique puede existir código muerto en el interior o que el método es muy largo

5.2. Técnica de Refactorización

Una de las técnicas de refactorizaciones es la de distribuir responsabilidades implementando un objeto que abarque todos los parámetros necesarios para realizar la operación de este método y que solo se le ingrese un parámetro de este tipo de objeto.

5.3. Código Inicial

```
    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
    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 ca

    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 promed
    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;

    }
```

5.4. Código Final

```
public class Notas {

    Paralelo p;
    double nexamen;
    double ndeberes;
    double nlecciones;
    double ntalleres;

    public Notas(Paralelo p, double nexamen,double ndeberes, double nlecciones, double ntalleres){
       this.ndeberes= ndeberes;
       this.nexamen = nexamen;
       this.nlecciones=nlecciones;
       this.ntalleres= ntalleres;
    }

    public Paralelo getP() {
        return p;
    }

    public void setP(Paralelo p) {
        this.p = p;
    }

    public double getNexamen() {
        return nexamen;
    }

    public void setNexamen(double nexamen) {
        this.nexamen = nexamen;
    }

    public double getNdeberes() {
        return ndeberes;
    }

    public void setNdeberes(double ndeberes) {
        this.ndeberes = ndeberes;
```

```java
//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
public double CalcularNotaInicial(Notas notas){
    double notaInicial=0;
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            notaInicial = notaFinal(notas.getNexamen(), notas.getNdeberes(), notas.getNlecciones(), notas.getNtalleres());
        }
    }
    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(Notas notas){
    double notaFinal=0;
    for(Paralelo par:paralelos){
        if(p.equals(par)){

            notaFinal = notaFinal(notas.getNexamen(), notas.getNdeberes(), notas.getNlecciones(), notas.getNtalleres());
        }
    }
    return notaFinal;
}
```

6. DUPLICADE CODE

6.1. Consecuencias

El hecho de tener un código duplicado hace que ocupe mucha memoria y se realicen procesos los cuales hacen que el código hace que sea más pesado y hace al usuario y programador propenso a confundirse

6.2. Técnica de Refactorización

Una de las técnicas de refactorización es la de Extract method la cual nos ayuda a obtener el código que se ve involucrado en varias clases y almacenarlo en una función la cual cuando se vaya a usar ese código se la llame.

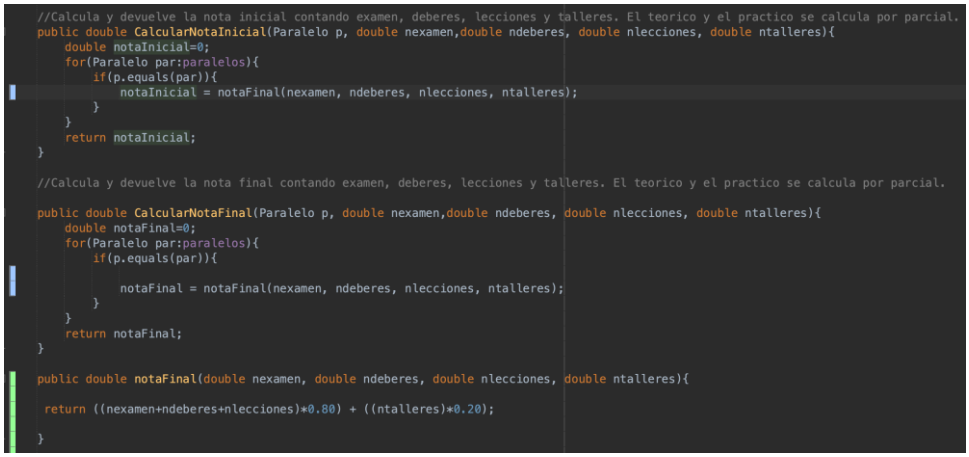
6.3. Código Inicial

```java
        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

    public double CalcularNotaFinal(Paralelo p, double nexamen,double ndeber
        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;
            }
```

6.4. Código Final



```java
//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)){
            notaInicial = notaFinal(nexamen, ndeberes, nlecciones, ntalleres);
        }
    }
    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)){

            notaFinal = notaFinal(nexamen, ndeberes, nlecciones, ntalleres);
        }
    }
    return notaFinal;
}

public double notaFinal(double nexamen, double ndeberes, double nlecciones, double ntalleres){

 return ((nexamen+ndeberes+nlecciones)*0.80) + ((ntalleres)*0.20);

}
```