ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

DISENO DE SOFTWARE

Taller

Refactoring

Integrantes:

DANIEL ELIAS TORRES ALVARADO

DAVID ALEJANDRO ALARCON GALEAS

MARCOS VICENTE RIOS CASTRO

IAN ERICK LOPEZ LLORENTTY

DANIEL FRANCISCO FERNANDEZ BUSTAMANTE

Descripción

Identificar Code Smells y técnicas para refactorizar

Indice

Code Smells

Middle Man:

Antes

Las consecuencias de permitir este Middle Man, están en que la clase Ayudante nos permite obtener información de la clase Estudiante mediante métodos que también están en esa clase. Como consecuencia, hay una ligera repetición de código.

El tratamiento propuesto es de removerlo haciendo que Ayudante extienda de Estudiante directamente, y podamos acceder a la información sin repeticiones de código en ambas clases.

Después

```
1   package modelos;
2
3   import java.util.ArrayList;
4
5   public class Ayudante extends Estudiante{//MIDDLE MAN debe extender de ESTUDIANTE
6       private ArrayList<Paralelo> paralelosAsignados;
7
8       Ayudante(Persona p, Estudiante e,ArrayList<Paralelo> paralelosAsignados){
9           super(p,e.getFacultad(),e.getMatricula(),e.getParalelos());
10          this.paralelosAsignados=paralelosAsignados;
11      }
12
13      public ArrayList<Paralelo> getParalelosAsignados() {
14          return paralelosAsignados;
15      }
16
17      public void setParalelosAsignados(ArrayList<Paralelo> paralelosAsignados) {
18          this.paralelosAsignados = paralelosAsignados;
19      }
20
21      //Método para imprimir los paralelos que tiene asignados como ayudante
22      public void MostrarParalelos(){
23          for(Paralelo par:paralelosAsignados){
24              //Muestra la info general de cada paralelo
25          }
26      }
27  }
```

Large Class:

Antes

```
    //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcu
    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

    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 l
    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;

    }
```

November 28

Las consecuencias de permitir que la clase Estudiante se quede con estos métodos, es que ahí posee métodos que violan el principio Single Responsability y en consecuencia tenemos una Large Class, por eso es que se propone que los métodos de cacularNotas***() sean uno solo, es decir cacularNotas(), para poder eliminar las líneas repetidas. En otras palabras, se usaría una extracción de interfaz que contenga al método y este sea implementado en la clase que se encargue de tener las notas de una materia.

Después

```
public double CalcularNota(Paralelo p, double nexamen,double ndeberes, double nlecciones, double ntalleres){
    double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
    double notaPractico=(ntalleres)*0.20;
    return notaTeorico+notaPractico;
}
```

Feature Invy:

Antes

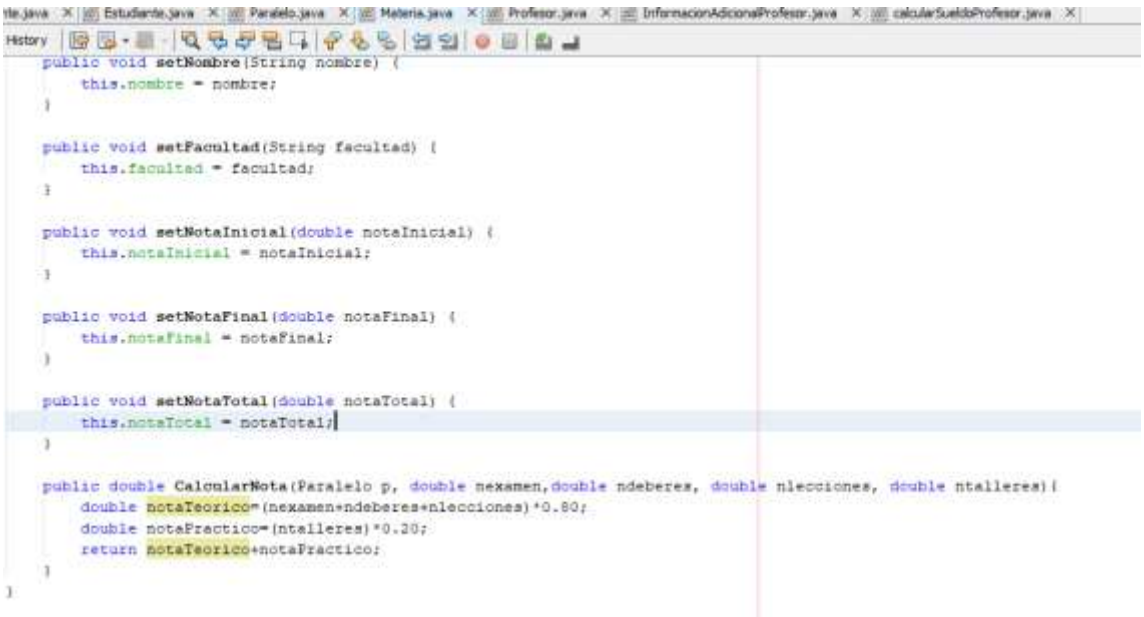
```
    public double CalcularNota(Paralelo p, double nexamen,double ndeberes, double nlecciones, double ntalleres){
        double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
        double notaPractico=(ntalleres)*0.20;
        return notaTeorico+notaPractico;
    }

    //SE QUEDA  AQUI
    //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta nota es solo el prome
    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;
    }
```

Las consecuencias de permitir que Estudiante conserve cualquiera de estos métodos, es que no son propios del mismo, tal y como lo es el método de calcularNota() que tiene el propósito de entregarnos las notas iniciales y finales en cada materia. Por lo tanto, debemos extraer el método a una clase que se encargue de implementar estos métodos y esa es la clase Materia que ya existe.

Después

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



Duplicate Code:

Antes

```
ublic 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;
```

```
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;
```

Las consecuencias de permitir estas líneas de código tanto en Estudiante como Profesor, es que hay líneas repetidas entre las clases. Por lo tanto, se propone extraer una superclase llamada “Persona” tal que Estudiante y Profesor hereden de esta sus atributos respectivos.

Después

```
public abstract class Persona {
    private String nombre;
    private String apellido;
    private int edad;
    private String direccion;
    private String telefono;
```

Long Paremeter List:

Antes

```
public class Profesor extends Persona {
    private String codigo;
    private ArrayList<Paralelo> paralelos;
    private int añosdeTrabajo;
    private String facultad;
    private double BonoFijo;
```

Tras resolver el code smell “Duplicate Code” anterior y un “Data Class” en InformacionAdicionalProfesor generamos una super clase que a pesar de reducir el número de parámetros para el constructor de la clase Profesor, aun tenemos mas de 4 en la firma del mismo. La resolución del Data Class nos da como resultado que los parámetros iniciales de InformacionAdicionalProfesor pasan a ser propios de Profesor, sin embargo, provocamos un Long Parameter List. Por estas razones, conservaremos la clase InformacionAdicionalProfesor y evitaremos este code smell.

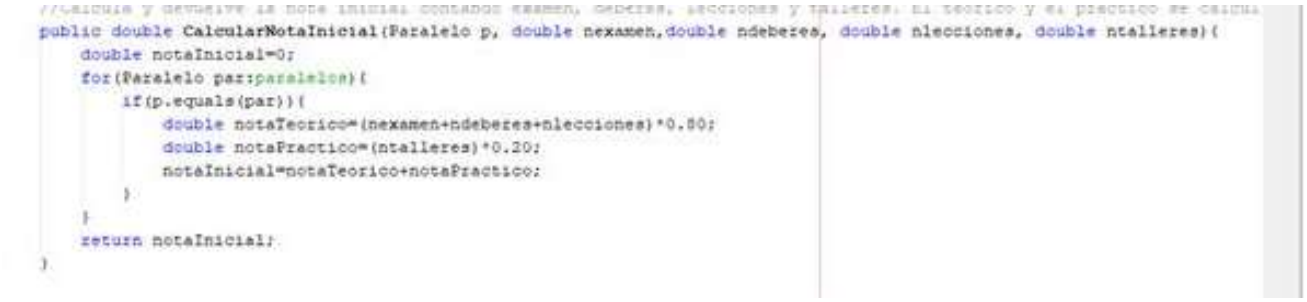
Después

```
public class Profesor extends Persona {
    private String codigo;
    private ArrayList<Paralelo> paralelos;
    private InformacionAdicionalProfesor info;
    //Long parameter list AGREGAR PERSONA AL CONSTRUCTOR/
    public Profesor(Persona p, String codigo, ArrayList<Paralelo> paralelos, InformacionAdicionalProfesor info) {
        super(p.getNombre(),p.getApellido(),p.getEdad(),p.getDireccion(),p.getTelefono());
        this.codigo = codigo;
        this.paralelos = paralelos;
        this.info=info;
    }
```

Temporary Fields:

Antes

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



```
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;
}
```

Las consecuencias de permitir estas variables fantasmas es que llenamos el stack de manera innecesaria cada vez que llamamos estos métodos. Por eso proponemos que en vez usarlas, retornemos directamente los cálculos deseados.

Después

```
public double CalcularNota(Paralelo p, double nexamen,double ndeberes, double nlecciones, double ntalleres){
    double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
    double notaPractico=(ntalleres)*0.20;
    return notaTeorico+notaPractico;
}
```

```
public double calcularSueldo(Profesor prof){
    return prof.getInfo().getAñosdeTrabajo()*600 + prof.getInfo().getBonoFijo();//temporary field
}
```

Lazy Class:

Antes

```
public class calcularSueldoProfesor {//Lazy Class

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

Las consecuencias de permitir este código tal como esta es que estaremos agregando al programa una clase que no siempre se va a usar. Además de esto, solo posee un método, y usar una clase como método es innecesario. Por lo tanto, se propone un colapso de jerarquía tal que este método se integre en la clase que lo usa.

Después

```
public class Profesor extends Persona {
    private String codigo;
    private ArrayList<Paralelo> paralelos;
    private InformacionAdicionalProfesor info;
    //Long parameter list AGREGAR PERSONA AL CONSTRUCTOR/
    public Profesor(Persona p, String codigo, ArrayList<Paralelo> paralelos, InformacionAdicionalProfesor info) {
        super(p.getNombre(),p.getApellido(),p.getEdad(),p.getDireccion(),p.getTelefono());
        this.codigo = codigo;
        this.paralelos = paralelos;
        this.info=info;
    }

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

    public InformacionAdicionalProfesor getInfo(){
        return info;
    }

    public double calcularSueldo(Profesor prof){
        return prof.getInfo().getAñosdeTrabajo()*600 + prof.getInfo().getBonoFijo();//temporary field
    }
}
```