

Taller Refactoring

Integrantes:

Paul del Pezo

Jose Vivanco

Paralelo:

102

Año Lectivo:

2020-2021

Profesor:

David Jurado

Ayudante:

Angie Tuarez

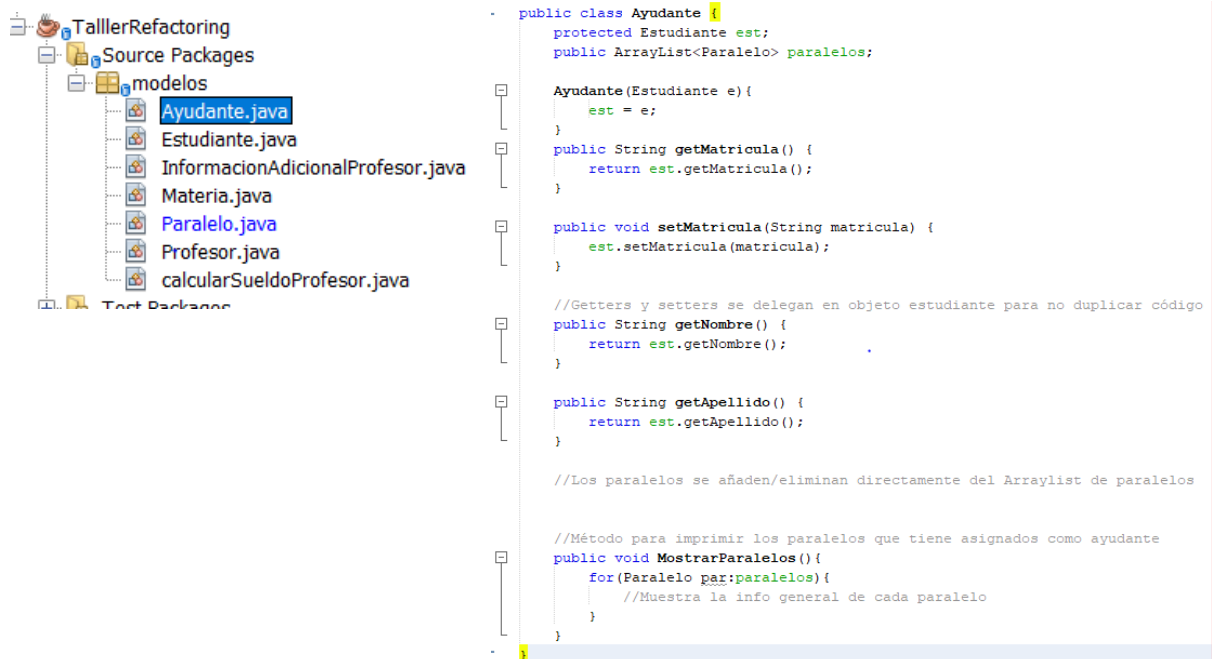
Contenido

Code Smell: Lazy Class	3
Codigo sin refactorizar	3
Codigo Refactorizado	3
Code Smell: Primitive Obsession.....	4
Caso 1	4
Codigo sin refactorizar	4
Código refactorizado.....	4
Caso 2:	5
Código sin refactorizar:	5
Codigo refactorizado:.....	5
Code Smell: Dead Code.....	6
Codigo sin refactorizar	6
Codigo refactorizado.....	6
Code Smell: Long Parameter List	7
Código sin factorizar	7
Código refactorizado.....	7
Code Smell: Shotgun Surgery	8
Código sin refactoriza	8
Código Refactorizado	8

Code Smell: Lazy Class

Este code smell se encuentra en la clase ayudante, dicha clase no tiene responsabilidad alguna y solo muestra información de otras cosas. El método de refactorización que se usó es el de “Inline Class”, que consiste en mover los métodos a una clase receptora.

Codigo sin refactorizar



The screenshot shows the project structure on the left with the 'modelos' package selected. The 'Ayudante.java' file is highlighted. On the right, the code for the 'Ayudante' class is displayed. The class has a protected 'Estudiante' attribute and a public 'ArrayList' of 'Paralelo' objects. It includes methods for getting and setting the matricula, getting the name and last name, and a method to show the list of paralelos. The code is as follows:

```
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);
    }

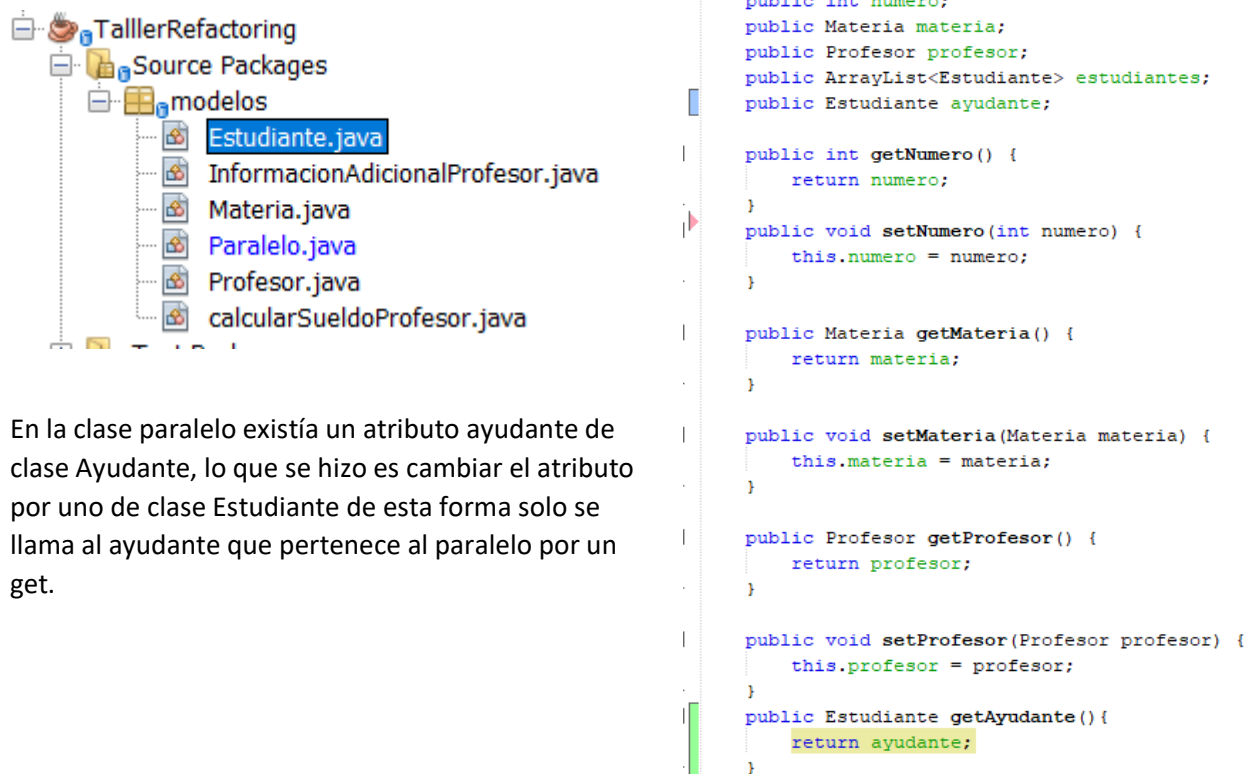
    //Getters y setters se delegan en objeto estudiante para no duplicar código
    public String getNombre() {
        return est.getNombre();
    }

    public String getApellido() {
        return est.getApellido();
    }

    //Los paralelos se añaden/eliminan directamente del ArrayList de paralelos

    //Método para imprimir los paralelos que tiene asignados como ayudante
    public void MostrarParalelos() {
        for(Paralelo par:paralelos){
            //Muestra la info general de cada paralelo
        }
    }
}
```

Codigo Refactorizado



The screenshot shows the project structure on the left with the 'modelos' package selected. The 'Estudiante.java' file is highlighted. On the right, the code for the 'Paralelo' class is displayed. The class has attributes for 'numero', 'Materia', 'Profesor', 'ArrayList' of 'Estudiante' objects, and an 'Estudiante' attribute named 'ayudante'. It includes methods for getting and setting the numero, materia, profesor, and ayudante. The code is as follows:

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

    public Estudiante getAyudante() {
        return ayudante;
    }
}
```

En la clase paralelo existía un atributo ayudante de clase Ayudante, lo que se hizo es cambiar el atributo por uno de clase Estudiante de esta forma solo se llama al ayudante que pertenece al paralelo por un get.

Code Smell: Primitive Obsession

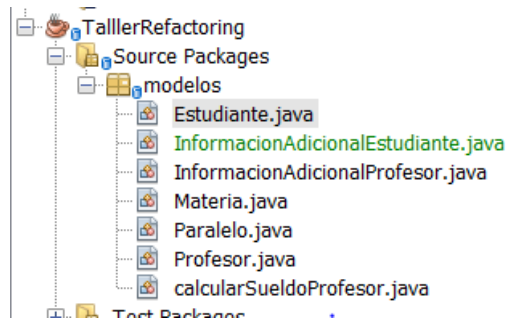
Caso 1

Este code smell aparece en la clase estudiante teniendo atributos que se pueden representar en una clase de esta forma haciendo que la clase sea más pequeña. El metodo de refactorización que se usó fue el de “Redefinir atributos en una nueva clase”

Codigo sin refactorizar

```
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;
}
```

Código refactorizado



Se crea una nueva clase llamada “InformaciónAdicionalEstudiante” que va a registrar datos adicionales del estudiante. Y que en la clase Estudiante solo sean llamados por un get.

```
public class InformacionAdicionalEstudiante {
    public int edad;
    public String direccion;
    public String telefono;

    public int getEdad() {
        return edad;
    }

    public String getDireccion() {
        return direccion;
    }

    public String getTelefono() {
        return telefono;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }

    public void setTelefono(String telefono) {
        this.telefono = telefono;
    }
}
```

Caso 2:

El sgt caso de este code smell se lo encuentra en la clase profesor teniendo algunos atributos que pueden ser agrupados en otra clase, reduciendo de esta forma el tamaño de esta.

Código sin refactorizar:

```
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;  
}
```

Código refactorizado:

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

```
public class Profesor {  
    public String codigo;  
    public String nombre;  
    public String apellido;  
    public InformacionAdicionalProfesor info;  
    public ArrayList<Paralelo> paralelos;  
  
    public Profesor(String codigo, String nombre, String apellido, InformacionAdicionalProfesor info) {...7 lines }  
  
    public InformacionAdicionalProfesor getInfo() {  
        return info;  
    }  
  
    public void anadirParalelos(Paralelo p) {  
        paralelos.add(p);  
    }  
}
```

Aprovechando que se tenía una clase creada como InformacionAdicionalProfesor se procede a enviar los datos que estaban en la clase Profesor a esta clase ya creada, y tan solo para acceder a ella se usa un get. El método de refactorización que se usó fue el de “Redefinir atributos en una nueva clase”

Code Smell: Dead Code

Este code smell se encuentra en la clase paralelo teniendo un método que ya no está siendo utilizado en ninguna clase y no hay la necesidad de implementarlo. El método de refactorización que se usó es tan eliminar el código.

Codigo sin refactorizar

```
//Imprime el listado de estudiantes registrados
public void mostrarListado(){
    //No es necesario implementar
}
```

Codigo refactorizado

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

    public Estudiante getAyudante(){
        return ayudante;
    }
}
```

Code Smell: Long Parameter List

Este code smell se encuentra en la clase Estudiante y siendo más específicos en dos métodos como lo son “CalcularNotaInicial” y “CalcularNotaFinal” la consecuencia de tener esta lista larga de parámetros es que se vuelven difíciles de entender, se vuelve contradictorias y difíciles de usar a medidas que crece nuestro programa. El método de refactorización a usar es de “Introduce Parameter Object”.

Código sin factorizar

```
//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

Se cambia esa extensa lista de parámetro por un objeto de una clase que se creó llamada “Notes” y teniendo como atributos a los parámetros que estaban en esos métodos.

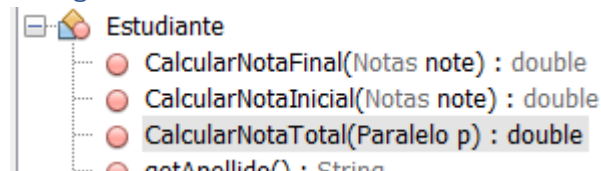
```
,
//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
public double CalcularNotaInicial(Notes note){
    double notaInicial=0;
    for(Paralelo par: paralelos){
        if(note.getP().equals(par)){
            double notaTeorico=(note.getNexamen()+note.getNdeberes()+note.getNlecciones())*0.80;
            double notaPractico=(note.getNtalleres())*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(Notes note){
    double notaFinal=0;
    for(Paralelo par: paralelos){
        if(note.getP().equals(par)){
            double notaTeorico=(note.getNexamen()+note.getNdeberes()+note.getNlecciones())*0.80;
            double notaPractico=(note.getNlecciones())*0.20;
            notaFinal=notaTeorico+notaPractico;
        }
    }
    return notaFinal;
}
```

Code Smell: Shotgun Surgery

Este code smell se presenta de nuevo en la clase “Estudiante” con los tres métodos de calculo que se presenta ahí, la consecuencia de esto es porque de forma lógica los estudiantes no se encargan de obtener sus notas, mas bien estos tres métodos se deben encargar los profesores. El metodo de refactorización a usar es “method move”, “Field move” y “Sustituir Metodo”.

Código sin refactoriza



```
public double CalcularNotaInicial(Notas note){
    double notaInicial=0;
    for(Paralelo par:paralelos){
        if(note.getP().equals(par)){
            double notaTeorico=(note.getNexamen()+note.getNdeberes()+note.getNlecciones())*0.80;
            double notaPractico=(note.getNtalleres())*0.20;
            notaInicial=notaTeorico+notaPractico;
        }
    }
    return notaInicial;
}

//Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico

public double CalcularNotaFinal(Notas note){
    double notaFinal=0;
    for(Paralelo par:paralelos){
        if(note.getP().equals(par)){
            double notaTeorico=(note.getNexamen()+note.getNdeberes()+note.getNlecciones())*0.80;
            double notaPractico=(note.getNlecciones())*0.20;
            notaFinal=notaTeorico+notaPractico;
        }
    }
    return notaFinal;
}

//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta nota es solo de la materia
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;
}
```

Cod

Código Refactorizado

El código fue simplificado eliminando algunas variables, y de paso se movieron estos métodos a la clase Profesor, además de enviar los cálculos de la nota a la clase “Materia” como correspondía

Estos métodos son serán privados ya que solo serán cálculos que no deben ser mostrados en pantallas y solo se verán reflejados en la nota.


```

private void CalcularNotaInicial(Notas note){
    for(Paralelo par:paralelos){
        if(note.getP().equals(par)){
            double notaTeorico=(note.getNexamen()+note.getNdeberes()+note.getNlecciones())*0.80;
            double notaPractico=(note.getNtalleres())*0.20;
            note.getP().getMateria().setNotaInicial(notaTeorico+notaPractico);
        }
    }
}

//Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico se (

private void CalcularNotaFinal(Notas note){
    for(Paralelo par:paralelos){
        if(note.getP().equals(par)){
            double notaTeorico=(note.getNexamen()+note.getNdeberes()+note.getNlecciones())*0.80;
            double notaPractico=(note.getNlecciones())*0.20;
            note.getP().getMateria().setNotaFinal(notaTeorico+notaPractico);
        }
    }
}

//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta nota es solo el prome(
private void CalcularNotaTotal(Paralelo p){
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            double notaTotal=(p.getMateria().notaInicial+p.getMateria().notaFinal)/2;
            p.getMateria().setNotaTotal(notaTotal);
        }
    }
}

```