Facultad de Ingeniería en
Electricidad y Computación

GRUPO #4

1. Juan Francisco Quimí Granados
2. Loberlly Noelia Salazar Aspiazu
3. Kevin Elihan Muñoz Calva

TALLER 8 – REFACTORING

Paralelo. - 2

Profesor. - Jurado Mosquera David Alonso

Fecha De Presentación. -

12/8/2021

Contenido

Sección A: Identificar los code smells en el código 3

Code Smell: Long Parameter List 3

Smell 1: 3

Smell 2: 4

Code Smell: Comments 6

Smell 1: 6

Code Smell: Inappropriate Intimacy 7

Smell 1: 7

Smell 2: 8

Smell 3: 8

Code Smell: Duplicate Code 11

Smell 1: 11

Smell 2: 11

Code Smell: Data Class 13

Code Smell: Dead Code 14

Smell 1: 14

Smell 2: 14

Code Smell: Divergent Change 15

Smell 1: 15

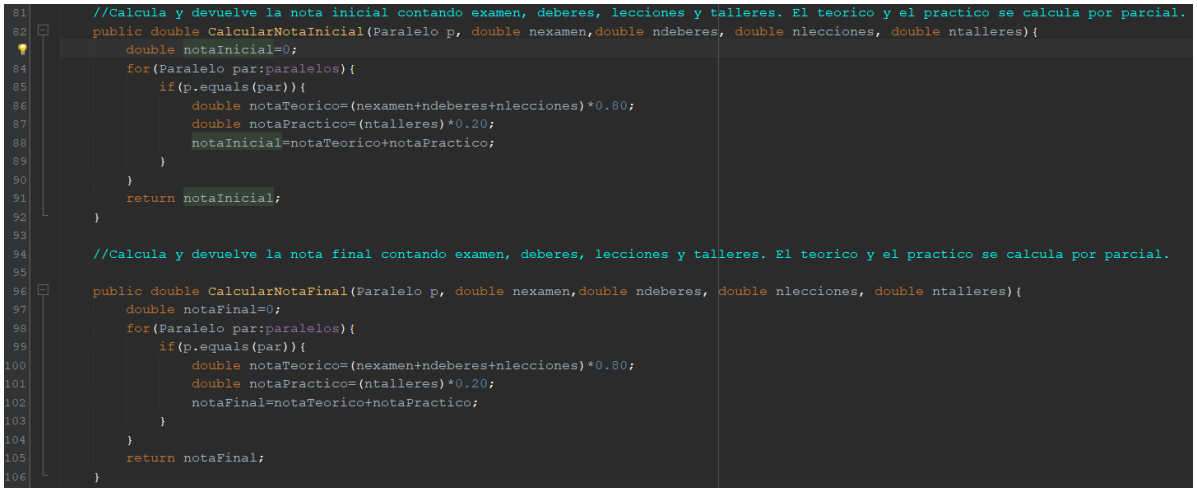
Sección A: Identificar los code smells en el código

Code Smell: Long Parameter List

Consecuencias. - Código más largo y tedioso de leer, puede llegar a provocar código duplicado.

Smell 1:

Ambos métodos para calcular nota reciben demasiados parámetros.



```
81      //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
82      public double CalcularNotaInicial(Paralelo p, double nexamen,double ndeberes, double nlecciones, double ntalleres){
83          double notaInicial=0;
84          for(Paralelo par:paralelos){
85              if(p.equals(par)){
86                  double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
87                  double notaPractico=(ntalleres)*0.20;
88                  notaInicial=notaTeorico+notaPractico;
89              }
90          }
91          return notaInicial;
92      }
93
94      //Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
95
96      public double CalcularNotaFinal(Paralelo p, double nexamen,double ndeberes, double nlecciones, double ntalleres){
97          double notaFinal=0;
98          for(Paralelo par:paralelos){
99              if(p.equals(par)){
100                 double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
101                 double notaPractico=(ntalleres)*0.20;
102                 notaFinal=notaTeorico+notaPractico;
103             }
104         }
105         return notaFinal;
106     }
```

Técnica de Refactoring. - Replace Type Code with Class

Se creó una nueva clase “Nota” para hacer uso de los parámetros de una forma más sencilla mediante getters y setters, en lugar de ser valores específicos.

```java
public class Nota {
    private double nexamen;
    private double ndeberes;
    private double nlecciones;
    private double ntalleres;
    private double valor;

    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;
    }

    public double getNlecciones() {
        return nlecciones;
    }

    public void setNlecciones(double nlecciones) {
        this.nlecciones = nlecciones;
    }
```

Smell 2:

El constructor de la clase profesor recibe demasiados parámetros.

```java
15      public Profesor(String codigo, String nombre, String apellido, String facultad, int edad, String direccion, String telefono) {
16          this.codigo = codigo;
17          this.nombre = nombre;
18          this.apellido = apellido;
19          this.edad = edad;
20          this.direccion = direccion;
21          this.telefono = telefono;
22          paralelos= new ArrayList<>();
23      }
```

Técnica de Refactoring. - Replace Parameter with Method Call

Con la creación de la clase padre Persona y su respectivo constructor de 3 parámetros, se acorta significativamente la longitud del constructor de la clase Profesor. Las acciones tanto de set y get de los parametros extras existentes en la clase Profesor serán responsabilidad de sus respectivos getters y setters.

```java
public class Profesor extends Persona{
    private String codigo;
    private InformacionAdicionalProfesor info;

    public Profesor(String nombre, String apellido, int edad)
        super(nombre, apellido, edad);
    }

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

    public String getCodigo() {
        return codigo;
    }

    public void setCodigo(String codigo) {
        this.codigo = codigo;
    }

    public InformacionAdicionalProfesor getInfo() {
        return info;
    }

    public void setInfo(InformacionAdicionalProfesor info) {
        this.info = info;
    }
```

Code Smell: Comments

Consecuencias. - Aumenta considerablemente el tamaño del código, puede sobrecargar al lector.

Smell 1:

Repartidos por todo el código existen comentario alguno completamente innecesarios que explican los métodos y otro que si son necesario ya que dicho no se logra entender sin estos.

```
29        //Los paralelos se añaden/eliminan directamente del Arraylist de paralelos
```

Técnica de Refactoring. -

Se eliminaron los comentarios innecesarios y los metods que si necesitaban sus comentarios se resolvieron en otros codes smells.

Code Smell: Inappropriate Intimacy

Consecuencias. - Puede llegar a dificultar la organización del código, además, complica el mantenimiento y la reutilización del código

Smell 1:

Este metodo de la clase calcularSueldoProfesor utiliza directamente los atributos de la clase InformacionAdicionalProfesor debido a que estos son publicos.

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

}
```

```
3   public class calcularSueldoProfesor {
4
5       public double calcularSueldo(Profesor prof){
6           double sueldo=0;
7           sueldo= prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
8           return sueldo;
9       }
10  }
```

Técnica de Refactoring. -

Se utilizaron las técnicas de “Extract Method” para sacra ese método de la clase y así enviarlo directamente a la clase InformacionAdicionalProfesor y de “Inline Method” ya que antes este metodo creaba una variable innecesaria solo para retornarla pudieron hacer esto directamente en la línea del return.

```
3   public class InformacionAdicionalProfesor {
4       private int añosdeTrabajo;
5       private String facultad;
6       private double BonoFijo;
7
8       public double calcularSueldo(Profesor prof){
9           return prof.getInfo().getAñosdeTrabajo()*600 + prof.getInfo().getBonoFijo();
10      }
11
```

```
108         //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y taller
109         public double CalcularNotaTotal(Paralelo p){
110             double notaTotal=0;
111             for(Paralelo par:paralelos){
112                 if(p.equals(par)){
113                     notaTotal=(p.getMateria().notaInicial+p.getMateria().notaFinal)/2;
114 
115                 }
116             }
117             return notaTotal;
```

Smell 2:

Varias clases no encapsulan sus atributos siendo estos accesibles por cualquier otra clase.

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

}
```

```
5   public class Paralelo {
6       public int numero;
7       public Materia materia;
8       public Profesor profesor;
9       public ArrayList<Estudiante> estudiantes;
10      public Ayudante ayudante;
```

Técnica de Refactoring. - Encapsulate field
Se encapsuló apropiadamente las clases cambiando la privacidad de los atributos.

```
8       private String nombre;
9       private String apellido;
10      private int edad;
11      private String direccion;
12      private String telefono;
13      private ArrayList<Paralelo> paralelos;
```

Smell 3:

La clase Ayudante tiene delegado un estudiante y su única funcionalidad son getter y setter que acceden a los getters y setters de estudiante.

```
12      public String getMatricula() {
13          return est.getMatricula();
14      }
15
16      public void setMatricula(String matricula) {
17          est.setMatricula(matricula);
18      }
19
20      //Getters y setters se delegan en objeto estudiante para no duplicar código
21      public String getNombre() {
22          return est.getNombre();
23      }
24
25      public String getApellido() {
26          return est.getApellido();
27      }
```

Técnica de Refactoring. - Replace delegation with inheritance

Se creo la herencia Estudiante->Ayudante para eliminar la delegación en los getters y setters.

```
5   public class Ayudante extends Estudiante{
6       protected Estudiante est;
7       private ArrayList<Paralelo> paralelos;
8
9       public Ayudante(String nombre, String apellido, int edad) {
10          super(nombre, apellido, edad);
11      }
12
13      //Los paralelos se añaden/eliminan directamente del Arraylist de paralelos
14
15
16      //Método para imprimir los paralelos que tiene asignados como ayudante
17      public void MostrarParalelos(){
18          for(Paralelo par:paralelos){
19              //Muestra la info general de cada paralelo
20          }
21      }
22  }
```

Smell 4:

La clase Paralelo no hace uso de ninguna de las funcionalidades de la clase Materia.

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

Técnica de Refactoring. - Change Bidirectional Association to Unidirectional

Se eliminó la asociación de Paralelo con Materia.

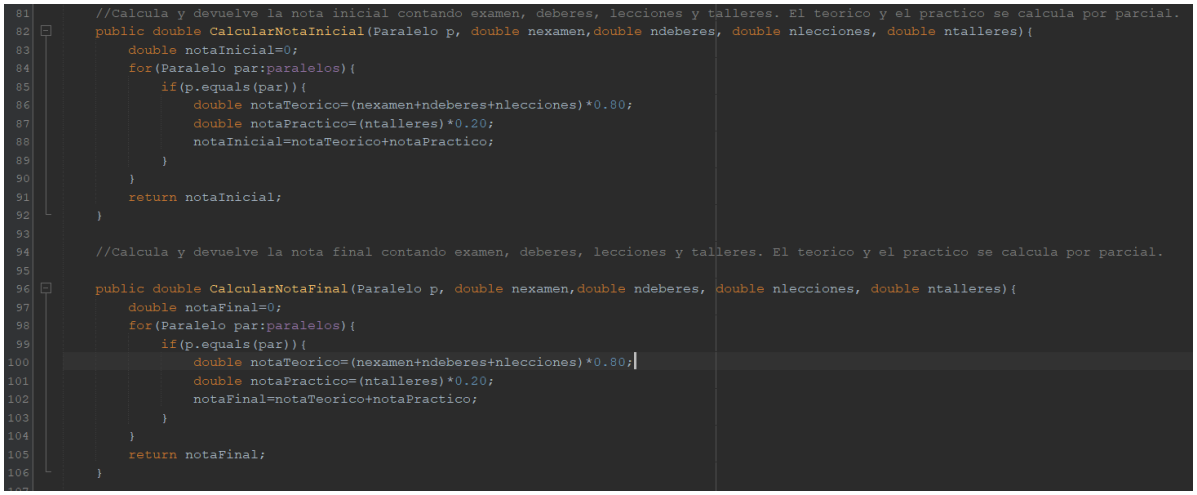
```java
public class Paralelo {
    public int numero;
    public Profesor profesor;
    public ArrayList<Estudiante> estudiantes;
    public Ayudante ayudante;
```

Code Smell: Duplicate Code

Consecuencias. - Código más extenso, se complica la estructura del código volviéndolo más fácil de simplificar y barato de dar soporte.

Smell 1:

Ambos métodos en la clase estudiante hacen exactamente lo mismo teniendo como única excepción a la variable a la que se asigna el resultado del calculo



```
81      //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
82      public double CalcularNotaInicial(Paralelo p, double nexamen,double ndeberes, double nlecciones, double ntalleres){
83          double notaInicial=0;
84          for(Paralelo par:paralelos){
85              if(p.equals(par)){
86                  double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
87                  double notaPractico=(ntalleres)*0.20;
88                  notaInicial=notaTeorico+notaPractico;
89              }
90          }
91          return notaInicial;
92      }
93
94      //Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
95
96      public double CalcularNotaFinal(Paralelo p, double nexamen,double ndeberes, double nlecciones, double ntalleres){
97          double notaFinal=0;
98          for(Paralelo par:paralelos){
99              if(p.equals(par)){
100                 double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
101                 double notaPractico=(ntalleres)*0.20;
102                 notaFinal=notaTeorico+notaPractico;
103             }
104         }
105         return notaFinal;
106     }
```

Técnica de Refactoring. - Parameterize Method

Se combinaron estos métodos en uno más general para realizar el procedimiento de calcular una Nota.

```
public double CalcularNota(Estudiante e, Paralelo p){
    for(Paralelo par:e.getParalelos()){
        if(p.equals(par)){
            double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
            double notaPractico=(ntalleres)*0.20;
            valor=notaTeorico+notaPractico;
        }
    }
    return valor;
}
```

Smell 2:

Las clases estudiante y profesor tienen los mismos atributos.

```java
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;
```

```java
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;
```

Técnica de Refactoring. - Extract Superclass

Se creo una clase padre Persona y se movió todos los atributos compartidos y métodos idénticos a la superclase eliminando así el duplicate code.

```java
public class Profesor extends Persona{
    private String codigo;
    private InformacionAdicionalProfesor info;

    public Profesor(String nombre, String apellido, int edad) {
        super(nombre, apellido, edad);
    }

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

    public String getCodigo() {
        return codigo;
    }

    public void setCodigo(String codigo) {
        this.codigo = codigo;
    }

    public InformacionAdicionalProfesor getInfo() {
        return info;
    }
```

```java
public class Estudiante extends Persona{
    //Informacion del estudiante
    private String matricula;
    private String facultad;

    public Estudiante(String nombre, String apellido, int edad) {
        super(nombre, apellido, edad);
    }

    public String getMatricula() {
        return matricula;
    }

    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }

    public String getFacultad() {
        return facultad;
    }

    public void setFacultad(String facultad) {
        this.facultad = facultad;
    }
```

Code Smell: Data Class

Consecuencias. - Puede provocar la creación de código duplicado, además de dificultar el entendimiento y la organización del código. Las operaciones de datos particulares pueden encontrarse regadas a lo largo del código.

Smell 1:

```
1   package modelos;
2
3   public class InformacionAdicionalProfesor {
4       public int añosdeTrabajo;
5       public String facultad;
6       public double BonoFijo;
7
8   }
```

Técnica de Refactoring. - Extract method

Mediante la resolución de un code smell resuelto anterirormente (Inapropiate Intimacy: Smell 1) tambine se resolvio este codes smell.

```
3   public class InformacionAdicionalProfesor {
4       private int añosdeTrabajo;
5       private String facultad;
6       private double BonoFijo;
7
8       public double calcularSueldo(Profesor prof){
9           return prof.getInfo().getAñosdeTrabajo()*600 + prof.getInfo().getBonoFijo();
10      }
11
```

Code Smell: Dead Code

Consecuencias. - Código innecesario que alarga la clase y dificulta el mantenimiento de esta.

Smell 1:

El método mostrarListado() dice no ser necesario de implementar, pero de todas formas esta creado.

```
36          //Imprime el listado de estudiantes registrados
37          public void mostrarListado(){
38              //No es necesario implementar
39          }
```

Técnica de Refactoring. -

Se elimino el método ya que este solamente esta creado, pero no implementado y no es parte de ninguna clase abstracta ni de una interfaz.

Smell 2:

No existe un atributo facultad, sin embargo, el constructor de profesor lo solicita.

```
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;

    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<>();
    }
```

Técnica de Refactoring. - Remove parameter

Este code smell fue resuelto anterirormente (Long parameter list: Smell 2).

Smell 3:

El método MostrarParalelos() tiene un for en el cual no se realiza ninguna acción.

```
16          //Método para imprimir los paralelos que tiene asignados como ayudante
17          public void MostrarParalelos(){
18              for(Paralelo par:paralelos){
19                  //Muestra la info general de cada paralelo
20              }
21          }
```

Técnica de Refactoring. -

Se elimino el método ya que su única funcionalidad es tener un for que no hace nada.

Code Smell: Divergent Change

Consecuencias. - Hacer un pequeño cambio dentro de una clase hará que se tengan que cambiar muchos métodos al mismo tiempo.

Smell 1:

En la clase estudiante existen varios métodos en lo que se multiplica por un numero especifico que representa el porcentaje de nota de cada curso, como sabemos no todas los cursos tienen el mismo porcentaje por lo tanto si se quiere cambia resto en algún futuro se tendrían que cambiar muchos métodos dentro de esta clase.

```
100                     double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
101                     double notaPractico=(ntalleres)*0.20;
```

Técnica de Refactoring. - Replace Magic Number with Symbolic Constant
Se crearon constantes dentro de la nueva clase Nota para reemplazar los factores multiplicativos que estaban en crudo.

```
public double CalcularNota(Estudiante e, Paralelo p){
    for(Paralelo par:e.getParalelos()){
        if(p.equals(par)){
            double notaTeorico=(nexamen+ndeberes+nlecciones)*constanteTeorica;
            double notaPractico=(ntalleres)*constantePractica;
            valor=notaTeorico+notaPractico;
        }
    }
    return valor;
}
```