

Escuela Superior Politécnica del Litoral



Taller 08

Refactoring

Integrantes:

- Jandry Rodríguez
- Victor Peña

Término

2021 – 1S

Contenido

Large class.....	3
Inappropriate Intimacy.....	4
Feature Envy	5
Dead code	6
Lazy class.....	7
Temporary Field.....	8
Duplicate code	9
Long Parameter list	10
Comments.....	11

Large class

La clase Estudiante tiene 120 líneas de código, es una clase muy larga que provoca un código que al momento de añadir funcionalidades o simplemente analizar su utilidad es muy difícil de leer y mantener:

```
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     }
107
108     //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta nota es solo el promedio de las dos cali
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         return notaTotal;
117     }
118 }
119
120 }
```

Técnica de refactoring aplicada:

Se resolvió con los refactoring de los demás smells

```
25         this.facultad = facultad;
26     }
27
28     public double CalcularNota(Paralelo p, RegistroNotas notas){
29         double nota=0;
30         for(Paralelo par: paralelos){
31             if(p.equals(par)){
32                 double notaTeorico=(notas.getExamen()+notas.getDeberes()+notas.getLecciones())*0.80;
33                 double notaPractico=(notas.getTalleres())*0.20;
34                 nota=notaTeorico+notaPractico;
35             }
36         }
37         return nota;
38     }
39
40     public double CalcularNotaTotal(Paralelo p){
41         double notaTotal=0;
42         for(Paralelo par: paralelos){
43             if(p.equals(par)){
44                 notaTotal=(p.getMateria().notaInicial+p.getMateria().notaFinal)/2;
45             }
46         }
47         return notaTotal;
48     }
49 }
50 }
```

Inappropriate Intimacy

Los atributos de todas las clases son públicos, esto permite la modificación de clases desde otras de manera indebida afectando al funcionamiento del sistema:

```
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;
    //Metodos de la clase
}
```

Técnica de refactoring aplicada:

Encapsulate Field:

```
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;
    //Metodos de la clase
}
```

Feature Envy

La clase Ayudante accede a los atributos de la clase Estudiante como propios, contribuyendo al acoplamiento excesivo entre las clases:

```
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();  
}
```

Técnica de refactoring aplicada:

Replace Delegation with Inheritance:

```
public class Ayudante extends Estudiante{  
    private ArrayList<Paralelo> paralelosAyudantia;  
  
    Ayudante(ArrayList<Paralelo> paralelos){  
        paralelosAyudantia = paralelos;  
    }  
  
    public void anadirParalelos(Paralelo p){  
        paralelosAyudantia.add(p);  
    }  
  
    public void MostrarParalelos(){  
        for(Paralelo par:paralelosAyudantia){  
            //Muestra la info general de cada paralelo  
        }  
    }  
}
```

Dead code

En la clase Paralelo el atributo **ayudante** de tipo Ayudante no se utiliza en toda la clase, dificultando la lectura del código.

```
TalllerRefactoring > src > modelos > Paralelo.java > Paralelo > mostrarListado
1  package modelos;
2
3  import java.util.ArrayList;
4
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 aplicada:

Change bidirectional association with unidirectional:

```
factoring > src > modelos > Paralelo.java > Paralelo
package modelos;
```


```
import java.util.ArrayList;
```

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

    public int getNumero() {
```

Lazy class

La clase InformaciónAdicionalProfesor solo tiene atributos a los que se accede a través de la clase Profesor, esto aumenta el acoplamiento del código y dificulta la reusabilidad:

TalllerRefactoring > src > modelos >  InformacionAdicionalProfesor.java > ...

```
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 aplicada:

Inline Class:

```
5  public class Profesor extends Persona{
6      private String codigo;
7      private ArrayList<Paralelo> paralelos;
8      private int añosdeTrabajo;
9      private String facultad;
10     private double BonoFijo;
```

Temporary Field

En la clase CalcularSueldoProfesor el método CalcularSueldo usa la variable **sueldo** que solo se usa para guardar la información de la fórmula aplicada para calcular el sueldo, teniendo un uso temporal.

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

Técnica de refactoring aplicada:

Inline Temp:

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


Duplicate code

En la clase Estudiante los métodos **CalcularNotaInicial** y **CalcularNotaFinal** tienen la misma funcionalidad, ninguno tiene una utilidad extra sobre el otro:

```
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 aplicada:

Eliminar uno de los métodos:

```
//Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico se calcula por parcial.
public double CalcularNota(Paralelo p, RegistroNotas notas){
    double nota=0;
    for(Paralelo par: paralelos){
        if(p.equals(par)){
            double notaTeorico=(notas.getExamen()+notas.getDeberes()+notas.getLecciones())*0.80;
            double notaPractico=(notas.getTalleres())*0.20;
            nota=notaTeorico+notaPractico;
        }
    }
    return nota;
}
```

Long Parameter list

En la clase Estudiante los métodos **CalcularNotaInicial** y **CalcularNotaFinal** tienen demasiados parámetros, en el futuro al querer usar este método el desarrollador puede olvidar alguno o no saber la utilidad de cada uno de los 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 aplicada:

Introduce Parameter Object:

```
//Calcula y devuelve la nota final contando examen, deberes, lecciones y talleres. El teorico y el practico s
public double CalcularNota(Paralelo p, RegistroNotas notas){
    double nota=0;
    for(Paralelo par: paralelos){
        if(p.equals(par)){
            double notaTeorico=(notas.getExamen()+notas.getDeberes()+notas.getLecciones())*0.80;
            double notaPractico=(notas.getTalleres())*0.20;
            nota=notaTeorico+notaPractico;
        }
    }
    return nota;
}
```

Comments

Los comentarios en el código no son muy útiles puesto que explican cosas intuitivas y solo ocupan espacio en la pantalla que dificulta la lectura del código:

```
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;

    //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;
    }
}
```

Técnica de refactoring aplicada:

Eliminar los comentarios innecesarios:

```
public class Estudiante extends Persona{
    protected String matricula;
    protected String facultad;
    private ArrayList<Paralelo> paralelos;
    private ArrayList<RegistroNotas> NotasParcial;
    private ArrayList<RegistroNotas> NotasFinal;

    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;
    }

    public double CalcularNota(Paralelo p, RegistroNotas notas){
        double nota=0;
        for(Paralelo par:paralelos){
            if(p.equals(par)){
                double notaTecnico=(notas.getExamen()/4+notas.getDeberes()/4+notas.getI)
```