

ESCUELA SUPERIOR POLITECNICA DEL LITORAL

Diseño de Software

Taller 08 Refactoring

Grupo 5

Integrantes:

- **Luis Adrian Litardo Calderon**
- **Yonkani Manuel Cedeno**
- **Jose Alberto Murillo Escudero**

2021-2022

Contenido

Lazy Class: calcularSueldoProfesor.....	3
Consecuencias:.....	3
Refactoring:.....	3
Captura Inicial:	3
Captura Final:	3
Inappropriate Intimacy: clase Profesor metodo calcularSueldoProfesor().....	4
Consecuencias:.....	4
Refactoring:.....	4
Codigo Inicial:	4
Código Final:.....	4
Duplicate Code: Clase estudiante.....	5
Consecuencias:.....	5
Refactoring:.....	5
Captura Inicial:	5
Captura Final:	5
Speculative Generality: Clase Estudiante y clase Profesor.	6
Consecuencias:.....	6
Refactoring:.....	6
Código Inicial:	6
Código Final:.....	6
Feature Envy : Clase Ayudante y Estudiante.....	7
Consecuencias:.....	7
Refactoring:.....	7
Código Inicial:	7
Código Final:.....	7
Dead Code : Clase Materia.....	8
Consecuencias:.....	8
Refactoring:.....	8
Código Inicial:	8
Código Final:.....	8
Data Class : Clase InformacionAdicionalProfesor.....	9
Consecuencias:.....	9
Refactoring:.....	9
Código Inicial:	9
Código Final:.....	9

Lazy Class: calcularSueldoProfesor

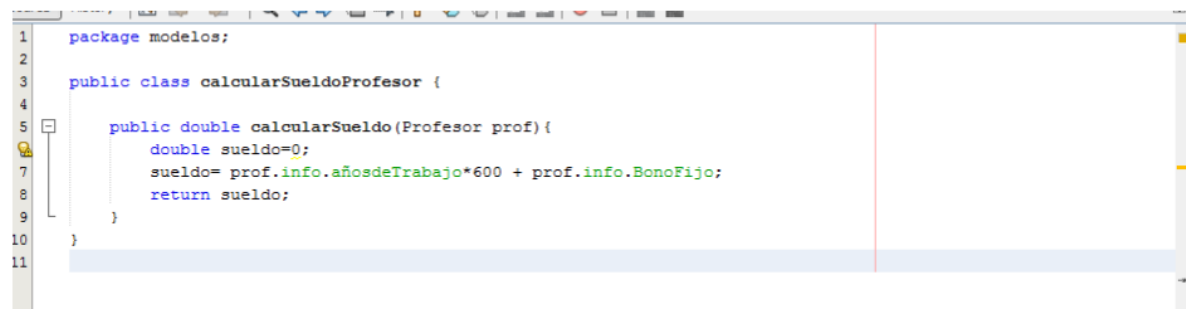
Consecuencias:

La clase calcularSueldoProfesor solo tiene un metodo y este metodo hace llamada a los atributos de otra clase, mantenerla en el codigo solo causaria confusion ademas de aumentar el numero de clases y crear una dificultad al momento de revisar el codigo

Refactoring:

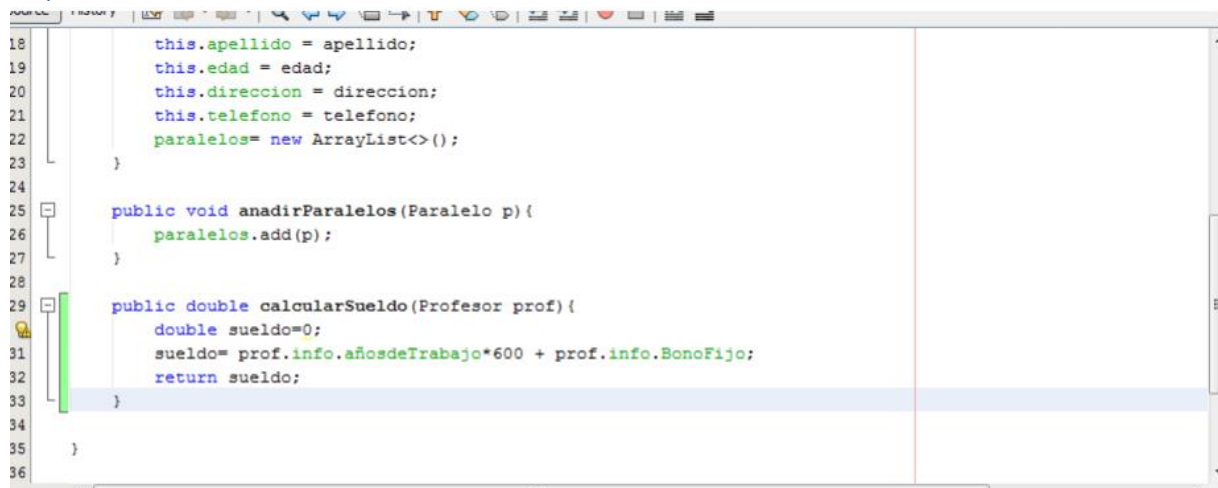
Se usara la tecnica de refactorizacion INLINE CLASS la cual nos permitira mover toda esta clase junto a sus atributos a otra que si sirva, en este caso moveremos el metodo a la clase profesor refactorizando asi el codigo

Captura Inicial:



```
1 package modelos;
2
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 }
11
```

Captura Final:



```
18     this.apellido = apellido;
19     this.edad = edad;
20     this.direccion = direccion;
21     this.telefono = telefono;
22     paralelos= new ArrayList<>();
23 }
24
25 public void anadirParalelos(Paralelo p){
26     paralelos.add(p);
27 }
28
29 public double calcularSueldo(Profesor prof){
30     double sueldo=0;
31     sueldo= prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
32     return sueldo;
33 }
34
35 }
36
```

Inappropriate Intimacy: clase Profesor metodo calcularSueldoProfesor()

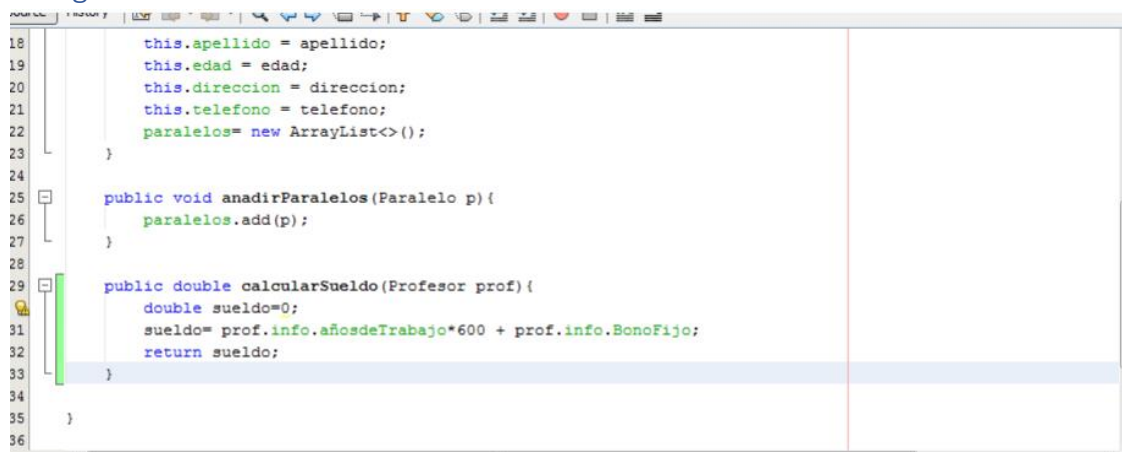
Consecuencias:

Se produce cuando una clase usa los metodos o atributos de otra clase, en este caso en la clase Profesor el metodo calcularSueldoProfesor() hace llamado a los atributos de la clase InformacionAdicionalProfesor

Refactoring:

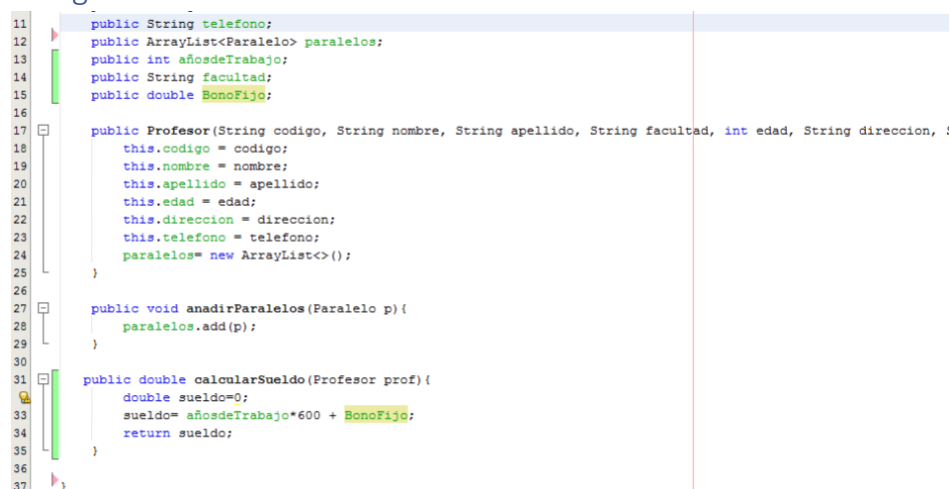
Se usa la tecnica de refactorizacion MOVE FIELD para poder mover los atributos de la clase InformacionAdicionalProfesor a la clase Profesor y que el metodo calcularSueldoProfesor() acceda a los metodos desde Profesor

Codigo Inicial:



```
18         this.apellido = apellido;
19         this.edad = edad;
20         this.direccion = direccion;
21         this.telefono = telefono;
22         paralelos= new ArrayList<>();
23     }
24
25     public void anadirParalelos(Paralelo p){
26         paralelos.add(p);
27     }
28
29     public double calcularSueldo(Profesor prof){
30         double sueldo=0;
31         sueldo= prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
32         return sueldo;
33     }
34
35 }
```

Código Final:



```
11     public String telefono;
12     public ArrayList<Paralelo> paralelos;
13     public int añosdeTrabajo;
14     public String facultad;
15     public double BonoFijo;
16
17     public Profesor(String codigo, String nombre, String apellido, String facultad, int edad, String direccion, !
18         this.codigo = codigo;
19         this.nombre = nombre;
20         this.apellido = apellido;
21         this.edad = edad;
22         this.direccion = direccion;
23         this.telefono = telefono;
24         paralelos= new ArrayList<>();
25     }
26
27     public void anadirParalelos(Paralelo p){
28         paralelos.add(p);
29     }
30
31     public double calcularSueldo(Profesor prof){
32         double sueldo=0;
33         sueldo= añosdeTrabajo*600 + BonoFijo;
34         return sueldo;
35     }
36
37 }
```

Duplicate Code: Clase estudiante

Consecuencias:

Al tener código duplicado en un proyecto bastante grande puede ser difícil de encontrar y corregirlo a parte que el código que las clases se hagan mucho más grandes y poca legibilidad al momento de leer el código haciendo difícil la mantenibilidad y la simpleza.

Refactoring:

La técnica a utilizar es “Extract Method” que va a consistir en mover el código que este duplicado a un nuevo método con un nombre que vaya acorde a su propósito. Se copiará el fragmento del código relevante al nuevo método y se borrará el código viejo.

Captura Inicial:

```
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;
}
```

Captura Final:

```
public double CalcularNota(Paralelo p, double nexamen, double ndeberes, double nlecciones, double ntalleres){
    double notaGeneral=0;
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
            double notaPractico=(ntalleres)*0.20;
            notaGeneral=notaTeorico+notaPractico;
        }
    }
    return notaGeneral;
}
```

Speculative Generality: Clase Estudiante y clase Profesor.

Consecuencias:

Al existir este code smell, el código se vuelve mucho más grande haciendo que sea difícil de entender en otras palabras agregando complejidad y mantener. Además, que se hace uso memoria innecesaria al momento de crear variables que jamás se van a usar.

Refactoring:

Los campos no utilizados van a ser removidos de sus clases correspondientes.

Código Inicial:

```
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 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 Final:

```
public class Estudiante{  
    //Informacion del estudiante  
    public String matricula;  
    public String nombre;  
    public String apellido;  
    public ArrayList<Paralelo> paralelos;  
}
```

```
public class Profesor {  
    public String codigo;  
    public String nombre;  
    public String apellido;  
    public ArrayList<Paralelo> paralelos;  
    public int añosdeTrabajo;  
    public String facultad;  
    public double BonoFijo;  
}
```

Feature Envy : Clase Ayudante y Estudiante

Consecuencias:

Este code smell sucede cuando una clase accede a más objetos de otra clase que a las de la propia clase, en la clase Ayudante tenemos getters y setters que provienen de la clase Estudiante creando así este Smell

Refactoring:

Para poder evitar todo este código podemos reemplazar con herencia de tal forma que Ayudante herede de Estudiante

Código Inicial:

```
0 |      est = e;
1 |    }
2 |    public String getMatricula() {
3 |        return est.getMatricula();
4 |    }
5 |
6 |    public void setMatricula(String matricula) {
7 |        est.setMatricula(matricula);
8 |    }
9 |
10 |    //Getters y setters se delegan en objeto estudiante para no duplicar código
11 |    public String getNombre() {
12 |        return est.getNombre();
13 |    }
14 |
15 |    public String getApellido() {
16 |        return est.getApellido();
17 |    }
18 |
19 |    //Los paralelos se añaden/eliminan directamente del ArrayList de paralelos
```

Código Final:

```
package modelos;

import java.util.ArrayList;

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

    public Ayudante(String nombre, String apellido, String matricula) {
        super(nombre, apellido, matricula);
    }
}
```

Dead Code : Clase Materia

Consecuencias:

Al momento de revisar el código hay clases que no ha sido utilizado provocando que exista confusión al momento de analizar el código además de que consume tiempo de cómputo en código que jamás se va a utilizar.

Refactoring:

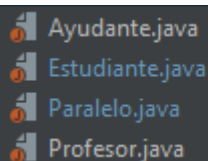
Se aplicará “inline class” que consiste en mover los atributos de una clase a otra hasta que esta clase quede totalmente vacía. Además, se eliminará la clase que ha quedado vacía se procederá a ser eliminada. Se reemplazara las referencias de los atributos de clase donante a la clase recipiente.

Código Inicial:

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

Código Final:

```
public class Estudiante{  
    //Informacion del estudiante  
    public String matricula;  
    public String nombre;  
    public String apellido;  
    public ArrayList<Paralelo> paralelos;  
    public double notaGeneral;  
    public double notaTotal;  
    public String codigoMateria;  
    public String nombreMateria;
```



- Ayudante.java
- Estudiante.java
- Paralelo.java
- Profesor.java

Data Class: Clase InformacionAdicionalProfesor

Consecuencias:

Sería necesario crear más métodos adicionales puesto que los atributos que se usarán se encuentran repartidos entre dos clases distintas, incluso corremos el riesgo de duplicar código en el programa.

Refactoring:

La clase “InformacionAdicionalProfesor” solamente contiene atributos que perfectamente pueden pertenecer directamente a la clase “Profesor”. En esta clase no hay métodos que contengan acciones adicionales, solamente contiene campos que pertenecen a la clase “Profesor”, podemos eliminar la clase “InformacionAdicionalProfesor” y mover esos atributos a la clase “Profesor”. Además, podemos ver que en ambas clases estos atributos están públicos (no están encapsulados), lo cual no debería ser así.

Código Inicial:

```
1 package modelos;
2
3 public class InformacionAdicionalProfesor {
4     public int añosdeTrabajo;
5     public String facultad;
6     public double BonoFijo;
7
8 }
9
```

```
1 package modelos;
2
3 import java.util.ArrayList;
4
5 public class Profesor {
6     public String codigo;
7     public String nombre;
8     public String apellido;
9     public int edad;
10    public String direccion;
11    public String telefono;
12    public InformacionAdicionalProfesor info;
13    public ArrayList<Paralelo> paralelos;
14
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    }
24
25    public void anadirParalelos(Paralelo p) {
26        paralelos.add(p);
27    }
28
29 }
30
31
```

Código Final:

```
1 package modelos;
2
3 import java.util.ArrayList;
4
5 public class Profesor
6 {
7     private int añosdeTrabajo;
8     private String facultad;
9     private double BonoFijo;
10    private String codigo;
11    private String nombre;
12    private String apellido;
13    private int edad;
14    private String direccion;
15    private String telefono;
16    private ArrayList<Paralelo> paralelos;
17
18    public Profesor(int añosdeTrabajo, String codigo, String nombre, String apellido, String facultad, int edad, String direccion,
19                    String telefono)
20    {
21        this.añosdeTrabajo = añosdeTrabajo;
22        this.codigo = codigo;
23        this.nombre = nombre;
24        this.apellido = apellido;
25        this.edad = edad;
26        this.direccion = direccion;
27        this.telefono = telefono;
28        this.paralelos = new ArrayList<>();
29    }
30
31    // Getters and Setters-----
32    public void anadirParalelos(Paralelo p)
33    {
34        paralelos.add(p);
35    }
36
37    public int getAñosdeTrabajo()
```

