ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

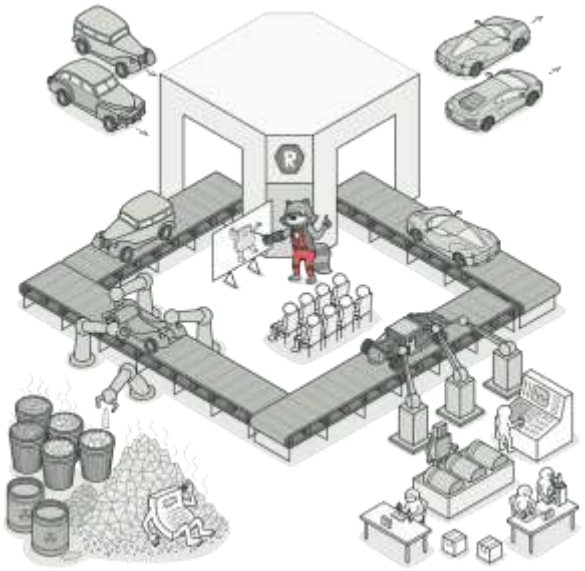
FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

DISEÑO DE SOFTWARE

TALLER 08 REFACTORING - 2021

PARALELO 103

GRUPO 4:



BRAVO SERNA DAVID ALVARO
FRANCO HIDALGO KEYLA FERNANDA
LOPEZ CASTILLO JOHANNA CECILIA

Link del repositorio:
https://github.com/KeylaFrancoH/Taller08-Refactoring.git

Contenido

TALLER DE REFACTORING

SECCIÓN A

1. IDENTIFICACIÓN DE MALOS OLORES

1.1. Duplicate code

1.1.1. Descripción

En la clase Estudiante se encuentran dos métodos que poseen el mismo código y realizan las mismas acciones, esto se nota en los métodos CalcularNotaInicial y CalcularNotaFinal.

1.1.2. Consecuencias

Se puede ocasionar problemas al momento de tener que modificar el código en cada uno de los métodos que lo utilizan, puede causar errores o el mal funcionamiento de este debido a que puede haber olvidos en alguna línea de código, lo cual alteraría el correcto funcionamiento del programa.

1.1.3. Técnica para su eliminación - Extract Method

Extract Method: Esta técnica permite extraer el código para de esta manera poder agruparlo en un solo método separado y facilitar la modificación del método en cada lugar que se utilice.

1.1.1. Captura código al inicio

```
//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;
}
```

1.1.2. Código después de refactorización

```
public double CalcularNota(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;
}
```

1.2. Data Class

1.2.1. Descripción

La clase InformacionAdicionalProfesor posee atributos que pueden ser modificados fuera de la clase, sin embargo, estos atributos pueden pertenecer directamente a la clase Profesor sin necesidad de dividirse en otra.

1.2.2. Consecuencia

Se podría crear código duplicado ya que se podrían crear los atributos faltantes sin conocer que se encuentran en otra clase, también sería necesario crear más métodos adicionales correspondientes porque sus otros atributos se encuentran distribuidos en otras clases.

1.2.3. Técnica para su eliminación – Move fields

Move fields: Se mueven los atributos de InformacionAdicionalProfesor a la clase Profesor y luego se elimina la clase.

1.2.4. Captura código al inicio

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

}
```

1.2.5. Código después de refactorización

```
public class Profesor {
    public int añosdeTrabajo;
    public double BonoFijo;
    public String codigo;
    public String nombre;
    public String apellido;
    public int edad;
    public String direccion;
    public String telefono;
    public ArrayList<Paralelo> paralelos;
```

1.3. Inappropiate Intimacy

1.3.1. Descripción

En la clase estudiante y en la clase CalcularSueldoProfesor se encontraron métodos que usaban los campos internos de otra clase.

1.3.2. Consecuencia

El código se vuelve más difícil de mantener, además, el control de datos de la otra clase es más complicado ya que la información puede ser manipulada. También se vuelve más difícil de comprender.

1.3.3. Técnica para su eliminación – Move method y Move field

Move method – CalcularSueldoProfesor, el método estaba contenido en una clase llamada calcularSueldoProfesor, procedimos a cambiarla esa responsabilidad a la clase profesor, y que ella se encargara de procesarla

Move field – CalcularNotaTotal, se movieron los parámetros que usaba el método calcularNotaTotal a la clase Materia para no estar abusando de los parámetros de otra clase

Move method – CalcularNotaTotal, se movió el método desde la clase estudiante a la clase Materia, para que la clase sea más coherente internamente

1.3.4. Captura código al inicio

CalcularNotaTotal – Clase Estudiante

```
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;

}
```

CalcularSueldoProfesor – Clase CalcularSueldoProfesor

```
public class calcularSueldoProfesor {

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

1.3.5. Código después de refactorización

CalcularNota – Clase Materia

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

CalcularSueldoProfesor – Clase Profesor

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

1.4. Long Parameter List

1.4.1. Descripción

En la clase Profesor, en el método constructor encontramos el code smell Long Parameter List, esto se genera debido a querer delegar menos entre clases utilizando demasiados parámetros para que el método actúe como se precisa, lo identificamos porque tiene más de 4 parámetros.

1.4.2. Consecuencia

La lista de parámetros larga hace que el código sea más difícil de usar y comprender, este code smell puede deberse a métodos demasiados complejos y crea dependencias.

1.4.3. Técnica para su eliminación – Remove Parameter

Remove Parameter: La técnica en este code smell permitió remover el parámetro facultad en el método constructor de la clase Profesor, en los métodos CalcularNotaInicial de la clase Materia también se removieron parámetros

1.4.4. Captura código al inicio

```
public Profesor(String codigo, String nombre, String apellido, String facultad, int edad, String direccion, String telefo
    this.codigo = codigo;
    this.nombre = nombre;
    this.apellido = apellido;
    this.edad = edad;
    this.direccion = direccion;
    this.telefono = telefono;
    paralelos= new ArrayList<>();
}
```

```
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;
}
```

1.4.5. Código después de refactorización

```
public Profesor(String codigo, String nombre, String apellido, 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<>();
}
```

```
public double CalcularNotaTotal(Paralelo p,ArrayList<Paralelo> paralelos){
    double notaTotal=0;
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            notaTotal=(notaInicial+notaFinal)/2;

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

1.5. Feature Envy

1.5.1. Descripción

La clase Ayudante accede a los atributos de la clase Estudiante, ya que sus métodos getters y setters delegan sus funciones a la clase Estudiante, además tienen funciones parecidas por lo cual se puede convertir en una herencia.

1.5.2. Consecuencias

El código se vuelve más complicado de entender por el usuario, por otro lado, el código puede ser reemplazado por la herencia de la otra clase, poseyendo los métodos y atributos de Estudiante.

1.5.3. Técnica para su eliminación - Replace delegatin with inheritance

Replace delegatin with inheritance: Evita que la clase posea gran cantidad de métodos innecesarios, además, evita la necesidad de crearlos para cada nuevo método de clase delegada, en este caso la clase ayudante como debería heredar de la clase estudiante, se eliminó el atributo estudiante y ahora a la clase estudiante le pertenecen los atributos de la clase estudiante

1.5.4. Captura código al inicio

```java
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();
    }
```

1.5.5. Código después de refactorización

Se crea un constructor en la clase Estudiante para que Ayudante pueda heredarlo y usar super.

```
public Estudiante(String matricula, String nombre, String apellido) {
    this.matricula = matricula;
    this.nombre = nombre;
    this.apellido = apellido;
}
```

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

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

1.6. Large class

1.6.1. Descripción

En la clase Estudiante tenemos el code smell Large class, la clase tiene los métodos CalcularNotaInicial, CalcularNotaFinal, CalcularNotaTotal, estos métodos no corresponden a esta clase porque se le está dando más responsabilidades a la clase Estudiante, haciendo crecer el código de esta clase

1.6.2. Consecuencias

El código se vuelve más difícil de entender, leer y desarrollar, creando la tendencia de centralizar funciones en esta clase, aumentando las posibilidades de fallas en el futuro.

1.6.3. Técnica para su eliminación – Extract Class

Extract Class: La técnica que usamos nos permite extraer parte del comportamiento de la clase que puede ser usado de distintas formas.

1.6.4. Captura código al inicio

1.6.5. Código después de refactorización

Se extrajeron los métodos CalcularNotaInicial, CalcularNotaFinal, CalcularNotaTotal, con la finalidad de reducir la clase y cumplir con el principio de Single Responsibility de los principios SOLID

```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;

    //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;
    }

    //Getter y setter del Apellido
    public String getApellido() {
        return apellido;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    //Getter y setter de la Facultad
    public String getFacultad() {
        return facultad;
    }

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

    //Getter y setter de la edad
    public int getEdad() {
        return edad;
    }
```