ESCUELA SUPERIOR POLITECICA DEL LITORAL

Taller Refactoring

Grupo 1

Integrantes:

- Alcivar Castro José Emanuel

-Richard David Cevallos Apolo

-Ricardo Mitridates Zevallos Mosquera

-Guillermo Isaac Teran Mendoza

-Bryan Jair Segovia Mariscal

Link Repositorio:

- https://github.com/Jose-Alcivar-C/TallerRefactoringG1

Indice

Data class

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.

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.

Técnicas usadas

- Move Fields

- Encapsulate Fields

Código inicial

```java
package modelos;

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

}

```

```java
package modelos;

import java.util.ArrayList;

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<>();
    }

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


}

```

Código refactorizado

```java
package modelos;

import java.util.ArrayList;

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

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

    // Getters and Setters-----------------------------------------------------
    public void anadirParalelos(Paralelo p)
    {
        paralelos.add(p);
    }

    public int getAñosdeTrabajo()
```

Innapropiate Intimacy

La clase “calcularSueldoProfesor” contiene un método que accede directamente a los atributos de otra clase, además de eso, podemos notar que este método no tendría que estar en esta clase, debido a que perfectamente podría estar en la clase “Profesor” y tomar de ahí los atributos, esto nos permite eliminar la clase “calcularSueldoProfesor”, puesto que está de más.

Consecuencias

El código se vuelve más complejo de mantener, incluso se complica el control sobre los datos de cada clase debido a que se accede directamente a ellos.

Técnicas usadas

- Encapsulate Fields

- Move Method

Código inicial

```java
package modelos;

public class calcularSueldoProfesor {

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

Código Refactorizado

```java
package modelos;

import java.util.ArrayList;

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

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

    public double calcularSueldo()
    {
        double sueldo = this.getAñosdeTrabajo()*600 + this.getBonoFijo();
        return sueldo;
    }

    // Getters and Setters-----------------------------------------------------
    public void anadirParalelos(Paralelo p)
```

Speculative Generality

La clase profesor tiene muchos campos que no son utilizados. Por lo tanto, se tiene el code smell "Speculative Generality". Si se refactoriza esta parte se tendrá un código más delgado y un soporte más sencillo.

Consecuencias

El código se vuelve largo y difícil de entender con tantos campos en una sola clase.

Técnicas usadas

Para refactorizar esto simplemente se eliminan estos campos que no se van a usar. En este caso solo se dejaran los campos de código, nombre y apellido para identificar al profesor y los años de trabajo y el bonofijo porque se usan en el método calcularSueldo().

Código inicial

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

    public Profesor(String codigo, String nombre, String apellido, int edad, String direccion, String telefono,int años, double bono) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.apellido = apellido;
        this.edad = edad;
        this.direccion = direccion;
        this.telefono = telefono;
        this.añosdeTrabajo=años;
        this.BonoFijo=bono;
        paralelos= new ArrayList<>();
    }

    public void anadirParalelos(Paralelo p){
        paralelos.add(p);
```

Código Refactorizado

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

    public Profesor(String codigo, String nombre, String apellido, int años, double bono) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.apellido = apellido;
        this.añosdeTrabajo=años;
        this.BonoFijo=bono;
        paralelos= new ArrayList<>();
    }
```

Duplicate Code

La clase estudiante tiene 2 métodos llamados CalcularNotaInicial y CalcularNotaFinal, los cuales tienen código duplicado.

Consecuencias

La duplicidad de código puede llevar a problemas a la hora de modificarlo, debido a que puede dejarse alguna duplicación sin modificar y por lo tanto introduciendo bugs que se pueden evitar.

Técnicas usadas

Para arreglar esto, se usó la técnica de refactorización "Extract method", la cual extrae un fragmento de código en común que se puede agrupar en un solo método para facilitar la modificación de esa funcionalidad.

Código inicial

```
//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se cal
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 calcu

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

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

//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se cal
public double CalcularNotaInicial(Paralelo p, double nexamen,double ndeberes, double nlecciones, double ntalleres){
    double notaInicial= NotaGeneral(p, nexamen, ndeberes, nlecciones, ntalleres);
    return notaInicial;
}

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

public double CalcularNotaFinal(Paralelo p, double nexamen,double ndeberes, double nlecciones, double ntalleres){
    double notaFinal= NotaGeneral(p, nexamen, ndeberes, nlecciones, ntalleres);
    return notaFinal;
}
```

Feature Envy

Los getters y setters de la clase “Ayudante” delegan sus funciones a los de la clase “Estudiante”, en otras palabras, la clase “Ayudante” accede más a atributos de otra clase que a los suyos propios, la clase que delega tiene funciones similares a la clase “Estudiante”, así que podemos reemplazar esto con una herencia.

Consecuencias

El código en este estado se vuelve más difícil de mantener y comprender por quien lo lea, además se puede ver que su función es las misma que la de otra clase.

Técnicas usadas

- Replace Delegation with Inheritance

Código inicial

```java
package modelos;

import java.util.ArrayList;

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
        }
    }
}
```

Código refactorizado

```java
package modelos;

import java.util.ArrayList;

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

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

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

Innapropiate Intimacy

Él método calcularNotaTotal tiene como parámetro un objeto de tipo Paralelo, y este accede a su objeto Materia para posteriormente acceder directamente a los atributos de la clase Materia, lo cual no se debe permitir, además que está violando uno de los pilares de la programación como es el correcto encapsulamiento.

Consecuencias

El acceso sin restricciones en algún momento podría generar consecuencias como que la información pueda ser manipulada.

Técnicas usadas

Una correcta solución, es hacer uso de la técnica de refactorización llamada "Encapsulate Field", la cual hace que los campos sean privados y crea métodos de acceso para el.

Código inicial

```
    //Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. Esta nota
    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;

    }
}
```

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

}
```

Código Refactorizado

```
    public double CalcularNotaTotal(Paralelo p){
        double notaTotal=0;
        for(Paralelo par:paralelos){
            if(p.equals(par)){
                notaTotal=(p.getMateria().getNotaInicial()+p.getMateria().getNotaFinal())/2;

            }
        }
        return notaTotal;

    }
}
```

```
urce  History
package modelos;

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

    public String getCodigo() {
        return codigo;
    }

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

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getFacultad() {
        return facultad;
    }

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

    public double getNotaInicial() {
        return notaInicial;
    }

    public void setNotaInicial(double notaInicial) {
        this.notaInicial = notaInicial;
    }

    public double getNotaFinal() {
        return notaFinal;
    }

    public void setNotaFinal(double notaFinal) {
        this.notaFinal = notaFinal;
    }

    public double getNotaTotal() {
```