Taller Refactoring

Integrantes:

- Kevin Bautista
- Andrés Morales
- Andrés Noboa

Profesor:

- David Jurado

Curso:

- Diseño de Software

Contenido

Temporary Field

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

Este método está usando una variable temporal innecesaria, se tiene que eliminar la variable sueldo y retornar directamente la expresión a la que está igualada sueldo. La técnica usada para la refactorización del problema es Inline Temp

También podemos darnos cuenta de que en ese método hay variables a las que se pueden acceder directamente con los métodos get y set de la clase Profesor e InformacionAdicionalProfesor por lo que se procede a utilizar esos métodos mediante Inline Temp

```
public double calcularSueldo(Profesor prof){
    return (prof.getInfo().getAñosdeTrabajo()*60 + prof.getInfo().getBonoFijo());
}
```

Temporary Field

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

}
```

Este método está utilizando una variable temporal innecesaria, solo se la usa en esa clase y para almacenar un valor, se tiene que eliminar la variable notaTotal y retornar directamente a lo que es igual la variable. La técnica usada para la refactorización en el código es Inline Temp

```
public double CalcularNotaTotal(Paralelo p){
   for(Paralelo par:paralelos){
        if(p.equals(par)){
            return (p.getMateria().notaInicial+p.getMateria().notaFinal)/2;
        }
    }
    return 0;
}
```

Duplicate Code

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

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;
}
```

Los métodos CalcularNotaInicial y CalcularNotaFinal hacen exactamente lo mismo en la misma clase, esto disminuye la calidad del código, tiene datos redundantes.

La solución para este code smell es la de extract method, se crea un solo método con el código necesario, se borra el código de los otros métodos y se llama al nuevo método creado.

Solution Extract Method

```java
private double CalcularNota(Paralelo p, double nexamen,double ndeberes, double nlecciones, double ntalleres){
    for(Paralelo par:paralelos){
        if(p.equals(par)){
            double notaTeorico=(nexamen+ndeberes+nlecciones)*0.80;
            double notaPractico=(ntalleres)*0.20;
            return notaTeorico+notaPractico;
        }
    }
    return 0;
}
//Calcula y devuelve la nota inicial contando examen, deberes, lecciones y talleres. El teorico y el practico se calc
public double CalcularNotaInicial(Paralelo p, double nexamen,double ndeberes, double nlecciones, double ntalleres){
    return CalcularNota(p,nexamen,ndeberes,nlecciones,ntalleres);
}

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

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

Middle Man

La clase calcularSueldoProfesor siempre tiene que llegar a la información de añosdeTrabajo y bonoFijo ubicados en la clase InformacionAdicionalProfesor, pero en el proceso siempre pasa por la clase Profesor como Middle Man.

```
public double calcularSueldo(Profesor prof){
    return (prof.getInfo().getAñosdeTrabajo()*60 + prof.getInfo().getBonoFijo());
}
```

Para refactorizar se solicita que se elimine la clase Middle Man (Profesor), pero en este caso simplemente se pasan los atributos de la clase InformacionAdicionalProfesor a la clase Profesor. De esta manera se corrige la necesidad de que haya que pasar por una clase intermedia para calcularSueldo.

```
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 int añosdeTrabajo;
    public String facultad;
    public double BonoFijo;

public double calcularSueldo(Profesor prof){
    return (prof.getAñosdeTrabajo()*60 + prof.getBonoFijo());
}
```

Feature Envy

El método en calcularSueldo accede más a la información dentro de los objetos de Profesor que a la información de su clase.

```
public double calcularSueldo(Profesor prof){
    return (prof.getAñosdeTrabajo()*60 + prof.getBonoFijo());
}
```

Para realizar la refactorización usamos Move Method, de esta manera el método calcularSueldo se lo moverá a la clase Profesor.

```
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 int añosdeTrabajo;
    public String facultad;
    public double BonoFijo;

    public double calcularSueldo(Profesor prof){
        return (prof.getAñosdeTrabajo()*60 + prof.getBonoFijo());
    }
```

Lazy Class

Una vez terminadas las refactorizaciones, nos quedan muchas clases que simplemente dejan de tener utilidad dentro del código.

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

}
```

```
public class calcularSueldoProfesor {
    /*
    public double calcularSueldo(Profesor prof){
        double sueldo=0;
        sueldo= prof.info.añosdeTrabajo*600 + prof.info.BonoFijo;
        return sueldo;
    }
    */
    public double calcularSueldo(Profesor prof){
        return (prof.getAñosdeTrabajo()*60 + prof.getBonoFijo());
    }
```

Es por esto por lo que se las procede a eliminar con Inline Class, esto es mover los atributos y métodos de la Lazy Class a otra.

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

    public double calcularSueldo(Profesor prof){
        return (prof.getAñosdeTrabajo()*60 + prof.getBonoFijo());
    }
```

Dead Code

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

Este método no está completamente implementado por lo tanto no hace nada. Puede ser debido a que hubo correcciones de código y nadie se preocupó de limpiarlo. Este código está de más, la solución es eliminarlo.